SHORT TERM TASK
SOFTWARE REQUIREMENTS SPECIFICATION and
SOFTWARE USERS MANUAL
CUT ORDER PLANNING

Co-Principal Investigator: Dr. Jane C. Ammons
Co-Principal Investigator: Dr. Charlotte Jacobs-Blecha
Research Investigator: Terri Smith
Research Assistant: Avril Baker
Research Assistant: Bill Warden

Prepared by: Terri Smith

Georgia Institute of Technology
Apparel Manufacturing Technology Center
Computer Science and Information Technology Laboratory
School of Industrial and Systems Engineering

October, 1989 - May, 1991

Research Sponsored by:
U.S. DEFENSE LOGISTICS AGENCY
Contract: DLA900-87-D-0018
Document Control No.: A-8496
# Short Term Task

## Cut Order Planning

### Title and Subtitle

**Cut Order Planning**

### Funding Numbers

DLA900-87-D-0018/0012

### Author(s)

Jane Ammons and Charlotte Jacobs-Blecha

### Performing Organization Name(s) and Address(es)

Georgia Tech Research Institute
215 O'Keefe Building
Atlanta, GA 30332

### Performing Organization Report Number

A-8496

### Sponsorizing/Monitoring Agency Name(s) and Address(es)

Defense Logistics Agency
Cameron Station (DLA-PRM)
Alexandria, VA 22304-6100

### Distribution: Availability Statement

Unclassified/Unlimited

### Abstract

This report details the results of a research project conducted at the Georgia Institute of Technology which investigated methods for improving cut order planning in apparel manufacturing. The project had two complementary objectives. The first objective was to investigate existing solution methodologies for the cut order planning problem. Alternate commercial software packages were examined and their performances comparatively analyzed, using testbed data representative of industrial problems. The results of this research provide important insights into the state-of-the-art in COP solution methods. A mathematical model of the COP problem was developed to facilitate problem specification and to initiate heuristic development. As a result of the complexity analysis, the COP problem was shown to be sufficiently complex that heuristic methods are the only reasonable means of finding solutions in real time. New methods were developed which perform as well as or better than those used in existing commercial packages. These algorithms have been implemented in a prototype software package for easy incorporation into existing commercial software, and will be transferred to industry through a participating s/w vendor.

### Subject Terms

Production Planning, Cut Order Planning Software, Apparel Manufacturing.

### Security Classification

Unclassified

### Number of Pages

322

### Limitation of Abstract

UNLIMITED
4.2.5.6 Findinch.c -------------------------------------- 16
4.2.5.7 Fives.c ---------------------------------------- 17
4.2.5.8 Fours.c ---------------------------------------- 17
4.2.5.9 Getparm.c --------------------------------------- 17
4.2.5.10 Globals.c -------------------------------------- 17
4.2.5.11 Ones.c ----------------------------------------- 17
4.2.5.12 Sixes.c ---------------------------------------- 17
4.2.5.13 Threes.c --------------------------------------- 17
4.2.5.14 Twos.c ----------------------------------------- 18
4.2.5.15 Cherdec.h -------------------------------------- 18
4.2.5.16 Cherlcl.h -------------------------------------- 18
4.2.5.17 Makefile.pc ------------------------------------- 18

4.2.6 Error Messages ------------------------------------- 18

4.3 Improvement Algorithm ---------------------------------- 19
4.3.1 Functional Description --------------------------------- 19
  4.3.1.1 Improvement Algorithm on current solution -------- 19
  4.3.1.2 Improvement Algorithm on an order --------------- 20
4.3.2 Input File ------------------------------------------ 22
4.3.3 Output File ------------------------------------------ 25
4.3.4 Data Structures -------------------------------------- 25
4.3.5 Module Descriptions ---------------------------------- 28
  4.3.5.1 Case_ai.c --------------------------------------- 28
  4.3.5.2 Case_aii.c -------------------------------------- 28
  4.3.5.3 Combply.c --------------------------------------- 28
  4.3.5.4 Combsize.c -------------------------------------- 28
  4.3.5.5 Compswap.c -------------------------------------- 29
  4.3.5.6 Compute.c --------------------------------------- 29
  4.3.5.7 Findinch.c -------------------------------------- 29
  4.3.5.8 Getparm.c --------------------------------------- 29
  4.3.5.9 Globals.c --------------------------------------- 29
  4.3.5.10 Improve.c -------------------------------------- 29
  4.3.5.11 Swapbkwd -------------------------------------- 30
  4.3.5.12 Swapfrwd -------------------------------------- 30
  4.3.5.13 Tranbkwd -------------------------------------- 30
  4.3.5.14 Tranfrwd -------------------------------------- 30
  4.3.5.15 Impdec.h -------------------------------------- 30
4.3.5.16 Implcl.h ---------------------------------------- 30
4.3.5.17 Makefile.pc ------------------------------------- 31
4.3.6 Error Messages ------------------------------------- 31

5.0 References ------------------------------------------- 32

6.0 Appendices
Appendix A Input File
Appendix B Output File
Appendix C Algorithm Detailed Descriptions
Appendix D Savings Algorithm Source Code
Appendix E Cherry Algorithm Source Code
Appendix F Improvement Algorithm Source Code
1.0 Scope

1.1 Identification

This User's Manual provides information, to users and programmers, about three algorithms for the Cut Order Planning problem: Savings, Cherry Picking, and Improvement. This information will enable the user to operate the programs and give the software programmer a better understanding of the software written.

1.2 Purpose

The algorithms were designed to analyze the cut order planning problem. The first algorithm uses a "computed savings" to combine sections in a cut plan. The second algorithm is also a constructive algorithm which combines pieces into one section at a time. The third algorithm tries to improve an existing solution by replacing pieces into other sections. The algorithms presented in this document are used for testing purposes only.

2.0 Equipment Configuration

The software was written on an IBM PC platform with MSDOS environment. All code is written in the Ansi C format. The outputs were generated with a 386/25MHz machine.

3.0 User Requirements

A typical user of the algorithms should be familiar with basic computer skills. The user needs to know how to edit an Ascii file (input file) and print a listing of the output file either on the screen or to a printer.
A software programmer must have knowledge of the C programming language to edit any of the procedures, and should have knowledge of either the Pascal or C languages to read the code. The code contains basic array structures, but no linked list structures. Therefore the user does not need extensive knowledge with the use of pointers but needs to understand arrays and the allocation of memory for such array structures. The software programmer must also be familiar with makefiles and how to compile and link the separate procedures.

4.0 Algorithm Description

4.1 Savings Algorithm

4.1.1 Functional Description

The Savings algorithm uses a "computed savings" to combine sections in a cut plan. The algorithm begins with each unit of fabric ordered in a separate section of initial ply height. Sections in this list are continuously merged into one section based on which merges provide the best savings in inches of fabric. As two sections are merged the new section is placed in a partial section. This partial section is then merged with other sections until the maximum sizes allowed per section is reached. Once this partial section is full, it is saved in a permanent list of sections and can no longer be used to merge with other sections. At this point a new partial section is created by combining two sections and the process begins again. Sections are only merged if the new section created does not exceed the maximum ply height or the maximum sizes allowed per section. When no more mergers are possible the solution is written to an output file and the program terminates.
Before merging sections together, the program calculates a potential savings in fabric for each possible combination of sections. There are basically two ways to merge two sections. If the two sections contain the same size combination, the savings can be computed by placing one section on top of another. The savings for this method is based on the saved cutting cost, since the size combination is only cut once instead of twice. The merger for sections containing the same size combination can also be achieved by changing the size combination to include two of each of the original sizes and leaving the ply height the same. The savings for this method is the decreased cost of fabric required for spreading the merged sections. If the two sections do not contain the same sizes then the savings is computed by changing the size combination and leaving the ply height the same.

Appendix C describes the Savings algorithm in detail and Appendix D contains a printout of the source code for the Savings algorithm.

The Savings algorithm is run simply by typing in "savings" at the DOS prompt in the directory in which the executable program is located. An input file, explained in section 4.1.2, must reside in the current directory in which the user is running the Savings algorithm. All output is written to a file also in that current directory.

4.1.2 Input File

The input file is an Ascii file which must contain the variables needed to run the Savings algorithm. The inputs include (1) an order to be cut, consisting of various sizes required, the quantity desired of each of these sizes, and
the perimeter around the size to cut. (2) The number of units over or under the demand that will be allowed. (3) The parameter K which determines the number of iterations after which the savings list will be updated. (4) The ply height of each of the initial sections. (5) List of L_i's (these are the fabric lengths required for cutting a size combination i - like small and large together - in a particular section. (6) The maximum ply height allowed. (7) The maximum number of sizes allowed per section. (8) The cutting cost per inch of fabric. (9) The unit cost of the fabric.

Each input is placed on a separate line on the input file and must be placed in the following order represented below:

```c
int ou_units : number of units over or under the demand
int max_ply : maximum ply height
int max_sizes : max sizes allowed per section
int init_ply : initial ply height
int k : number of iterations after which the savings list is updated
int q : the ply height in which to use
int cut_cost : cutting cost per inch
int unit_cost : unit cost of size
order_t order : the order
list_t list : list of L_i's (fabric lengths for each size combination)
```
The beginning of the input file would appear as follows:

```
3 : ou_units
3 : max_sizes
4 : max_ply
1 : init_ply
1 : k
1 : cut_cost
1 : unit_cost
5 120 size-30 : order ( #, perimeter, string for size)
5 130 size-32 : order ( #, perimeter, string for size)
5 140 size-34 : order ( #, perimeter, string for size)
1 0 -1 15.31 : list ( 1 size 0, inches)
1 1 -1 15.56 : list ( 1 size 1, inches)
```

A detailed description of the data structures for an order and the list of L_i's is in section 4.1.4.

As each row of the order is read from the input file the values are placed in the ord_var_t structure. For example the first row (5 120 size-30) is placed as follows:

```
number[0] = 5
ch_sizes[0] = "size-30"
perimeter[0] = 120
```

As each row in the list of L_i's is read the variable are placed in the list_t array as follows:

```
List[0].sizes[0] = 1  ( 1 size-30 )
List[0].sizes[1] = 0  ( 0 size-32 )
List[0].inches  = 15.31
```

```
List[1].sizes[0] = 0  ( 0 size-30 )
List[1].sizes[1] = 1  ( 1 size-32 )
List[1].inches  = 15.56
```

Appendix A contains a printout of the entire input file.
4.1.3 Output File

The output file ("OUTPUT") lists the sizes in each section, the inches for that section, the ply height, and the total inches for the section (ply x inches). At the end the total inches are printed along with the number of units over or under the demand. A sample output file is in Appendix B.

4.1.4 Data Structures

MAX_SIZES defines the maximum number of different sizes possible in one order.

MAX_LIST defines the maximum number of possible size combinations as in the input file.

MAX_SAVINGS defines the maximum number of savings that will be kept in a list.

The following structure holds the order which consists of the amount of each size needed, the string value that will be written to the output file associated with that size, and the perimeter of the particular size:

```c
struct order {
    order_t number;
    sizes_t ch_sizes;
    int perimeter[MAX_SIZES];
} ord_var_t;
```

Order.number is an array which holds the amount of each size ordered. It is defined as follows:

```c
typedef int order_t[MAX_SIZES];
```

Order.ch_sizes is an array which holds the string value associated with the size. e.g "Size-32". It is
defined as follows:
typedef char sizes_t[MAX_SIZES][10] :

Order.perimeter is an array which holds the perimeter of each size.

Arrays in the C language begin with zero and not one. Therefore there will be a value in the first position, zero. If the order is 3 size-30 and 5 size-32, the ord_var_t structure will hold the following numbers:

\[
\begin{align*}
\text{number}[0] &= 3 \\
\text{ch\_sizes}[0] &= \text{"size-30"} \\
\text{perimeter}[0] &= 120
\end{align*}
\]
\[
\begin{align*}
\text{number}[1] &= 5 \\
\text{ch\_sizes}[1] &= \text{"size-32"} \\
\text{perimeter}[1] &= 130
\end{align*}
\]

The following structure holds the list of size combinations:

\[
\text{struct list\_t} \{
\quad \text{order\_t sizes;}
\quad \text{float inches:}
\} \text{ list}
\]

List.sizes is an array which holds the amount of each size in the specific size combination.

List.inches holds the amount of fabric in inches need for that specific sizes combination.

An example of what is in the list[] structure:

\[
\begin{align*}
\text{sizes}[0] &= 1 \ (1 \text{ size-30}) \\
\text{sizes}[1] &= 1 \ (1 \text{ size-32}) \\
\text{inches} &= 28.01
\end{align*}
\]

The following structure holds the data necessary for each section.
struct section_t {
    order_t sizes;
    int    ply_height;
    char   merged;
} section

Section.order is an array which holds the amount of each size in the specific size combination for this particular section.

Section.ply_height is the ply height of that section.

Section.merged tells whether the section has already been merged or not.

The following structure contains the data for each savings calculated between two sections:

struct savings_t {
    int    sect1
    int    sect2
    int    ply_height;
    float  savings;
    int    type;
    int    ply1;
    int    ply2;
} save_list;

Save_list.sect1 and save_list.sect2 are the numbers of the two sections being merged. These numbers correspond to their position in the section list.

Save_list.ply_height is the ply height of the new combined section.

Save_list.savings is the savings achieved if these two sections are merged.

Save_list.type is the type of merge that is needed.
   1 : Sizes the same, place on top of another
   2 : Sizes different, rearrange size combination

Save_list.ply1 and save_list.ply2 are the different ply heights of sect1 and sect2.
4.1.5 Module Definitions

4.1.5.1 Case_ai.c
Case_ai() computes the savings of placing one section on top of the other. The two sections must have the same size combination.

4.1.5.2 Case_aii.c
Case_aii() computes the savings of rearranging the size combination of the two sections into one section but keeping the ply height the same.

4.1.5.3 Compute.c
Compute_savings() looks at the two sections to determine which way to compute the savings. If the two sections have the same sizes combination then case_ai.c is called else case_aii.c is called. Ply height of the sections are adjusted and the savings is returned to the main program.

4.1.5.4 Findinch.c
Find_inches() searches through the list of L_i inputs to find the inches for the specific size combination of a section.

4.1.5.5 Getparm.c
Get_parameters() opens the input file and reads in all the inputs into their appropriate variables and structures.

4.1.5.6 Globals.c
Globals defines and initializes all the global variables for the Savings algorithm.
4.1.5.7 Savings.c

This file contains the main program to execute the savings algorithm. The procedure begins by allocating enough memory for all the variables and lists and then calls get_parameters() to read the input file. The main loop in the procedure calculates the savings and merges sections until no more merges are possible by calling the other various procedures. This procedure then writes all the final results to the file "Output". All allocated space is freed and all files are closed.

4.1.5.8 Savdec.h

This file contains all the definitions for various structures and procedures that are used by the Savings algorithm.

4.1.5.9 Savelcl.h

This file defines the global variables for the Savings algorithm as external variables so that the program will compile. The actual definition and initialization is in the globals.c file.

4.1.5.10 Makefile.pc

This file contains the method in which to compile all the procedures and create the executable file Savings.exe. To execute this file type "make makefile.pc".

4.1.6 Error Messages

All error messages provide a short message and the file in which the error occurred.

The following message appears when the program has trouble opening a file in which to write the output.
"CANNOT OPEN OUTPUT FILE savings.c"

The following message appears when the program has trouble allocating memory to any structure needed for the program.

"ALLOCATION ERROR for list savings.c"

The following error appears when the program tries a specific combination of sizes in one section but can not locate the inches for this combination in the input list. The user should check the input file to determine if the value exists or not.

"CANNOT FIND 1 size-30 2 size-32 findinch.c"

4.2 Cherry Picking Algorithm

4.2.1 Functional Description

The Cherry Picking algorithm builds sections by combining certain sizes based on the best utilization of fabric. The algorithm begins by choosing the first (Q1) and second (Q2) most numerous sizes in the order. OU will represent the number of units over or under the demand that is allowed. Any size which has an order quantity greater than (Q2 - OU) is placed into sections such that a minimal amount of fabric is used. All quantities of sizes assigned to a section are reduced appropriately. This process continues until all sizes have been assigned to sections. When this occurs the program terminates and the output is written to a file.
Appendix C describes the algorithm in greater detail and Appendix E contains the source code for the Cherry Picking algorithm.

The Cherry algorithm is run simply by typing in "cherry" at the DOS prompt in the directory in which the executable program is located. An input file, explained in section 4.2.2, must reside in the current directory in which the user is running the Cherry algorithm. All output is written to a file also in that current directory.

4.2.2 Input File

The input file is an Ascii file which must contain the variables needed to run the Savings algorithm. The inputs include (1) an order to be cut, consisting of various sizes required, and the quantity desired of each of these sizes. (2) The number of units over or under the demand that will be allowed. (3) List of $L_i$'s (these are the fabric lengths required for cutting a size combination $i$ - like small and large together - in a particular section. (4) The maximum ply height allowed. (5) The maximum number of sizes allowed per section.

Each input is placed on a separate line on the input file and must be placed in the following order represented below:

```plaintext
int ou_units : number of units over or under the demand
int max_ply : maximum ply height
int max_sizes : max sizes allowed per section
order_t order : the amount of each size needed
list_t list : list of $I$'s (fabric lengths for each sizes combination)
```
The input file would appear as follows:

```
3 : ou_units
3 : max_sizes
4 : max_ply
5 120 size-30 : order ( #, perimeter, string for size)
5 130 size-32 : order ( #, perimeter, string for size)
5 140 size-34 : order ( #, perimeter, string for size)
1 0 -1 15.31 : list ( 1 size 0, inches)
1 1 -1 15.56 : list ( 1 size 1, inches)
-2 : end
```

A detailed description of the data structures for an order and the list of L_1s is in section 4.2.4.

As each row of the order is read from the input file the values are placed in the ord_var_t structure. For example the first row (5 120 size-30) is placed as follows:

```
number[0] = 5
ch_sizes[0] = "size-30"
perimeter[0] = 120
```

As each row in the list of L_1s is read the variable are placed in the list_t array as follows:

```
List[0].sizes[0] = 1   ( 1 size-30 )
List[0].sizes[1] = 0   ( 0 size-32 )
List[0].inches = 15.31

List[1].sizes[0] = 0   ( 0 size-30 )
List[1].sizes[1] = 1   ( 1 size-32 )
List[1].inches = 15.56
```

Appendix A contains a printout of the entire input file.
4.2.3 Output File

The output file ("OUTPUT") lists the sizes in each section, the inches for that section, the ply height, and the total inches for the section (ply x inches). At the end the total inches are printed along with the number of units over or under the demand. A sample output file is in Appendix B.

4.2.4 Data Structures

MAX_SIZES defines the maximum number of different sizes possible in one order.

MAX_LIST defines the maximum number of possible size combinations as in the input file.

The following structure holds the order which consists of the quantity amount of each size needed, the string value that will be written to the output file associated with that size, and the perimeter of the particular size:

```
struct order {
    order_t number;
    sizes_t ch_sizes;
    int   perimeter[MAX_SIZES];
} ord_var_t;
```

Order.number is an array which holds the amount of each size ordered. It is defined as follows:
```
typedef int order_t[MAX_SIZES];
```

Order.ch_sizes is an array which holds the string value associated with the size. e.g "Size-32". It is defined as follows:
```
typedef char sizes_t[MAX_SIZES][10] :
```

Order.perimeter is an array which holds the perimeter of each size.
Arrays in the C language begin with zero and not one. Therefore there will be a value in the first position, zero. If the order is 3 size-30 and 5 size-32, the ord_var_t structure will hold the following numbers:

- number[0] = 3
- number[1] = 5
- ch_sizes[0] = "size-30"
- ch_sizes[1] = "size-32"
- perimeter[0] = 120
- perimeter[1] = 130

The following structure holds the list of LiS:

```c
struct list_t {
    order_t sizes;
    float inches;
} list
```

List.sizes is an array which holds the amount of each size in the specific size combination.

List.inches holds the amount of fabric in inches needed for that specific sizes combination.

An example of what is in the list[] structure:

- sizes[0] = 1 (1 size-30)
- sizes[1] = 1 (1 size-32)
- inches = 28.01

The following structure holds the data necessary for each section:

```c
struct section_t {
    order_t sizes;
    int ply_height;
} section
```

Section.order is an array which holds the amount of each size in the specific size combination for this particular section.

Section.ply_height is the ply height of that section.
4.2.5 Module Descriptions

4.2.5.1 Cherry.c

This file contains the main program which executes the Cherry Picking algorithm. The procedure begins by allocating memory for all the variables and lists for the program and calls get_parameters() to read the input file. The main loop in the program chooses the best combination of sizes in the set S by calling the other various procedures. After all sizes have been used the procedure writes the results to the file "Output", releases all memory, and closes all files.

4.2.5.2 Chkinch.c

Check_inches() determines if the total inches calculated from the current combination is less than the previous best combination. If so this new combination is saved in a temporary section.

4.2.5.3 Clrtemp.c

Clear_temp() initializes the temporary sections.

4.2.5.4 Combine.c

Combine_inches calls find_inches() to calculate the total inches for a specific combination of sizes.

4.2.5.5 Cphold.c

Copy_hold_to_sections() copies all the sections in the temporary holding segment to the final output hold segment of sections.

4.2.5.6 Findinch.c

Find_inches searches through the list of $L_i$ inputs to find the inches for a specific size combination.
4.2.5.7 Fives.c

Fives() is a recursive procedure which groups sizes in combinations of five. Recursive programming means it keeps calling itself until all possible combinations are exhausted.

4.2.5.8 Fours.c

Fours() is a recursive procedure which groups sizes in combinations of four. Recursive programming means it keeps calling itself until all possible combinations are exhausted.

4.2.5.9 Getparm.c

Get_parameters() reads the input file and places all input variables in their appropriate structures.

4.2.5.10 Globals.c

Globals.c defines and initializes all global variables for the Cherry Picking algorithm.

4.2.5.11 Ones.c

Ones() places each size in a section by itself and find the total inches for all these sections combined.

4.2.5.12 Sixes.c

Sixes() is a recursive procedure which groups sizes in combinations of six. Recursive programming means it keeps calling itself until all possible combinations are exhausted.

4.2.5.13 Threes.c

Threes() is a recursive procedure which groups sizes in combinations of three. Recursive programming means it keeps
calling itself until all possible combinations are exhausted.

4.2.5.14 Twos.c

Twos() is a recursive procedure which groups sizes in combinations of two. Recursive programming means it keeps calling itself until all possible combinations are exhausted.

4.2.5.15 Cherdec.h

This file contains all the definitions for various structures and procedures that are used by the Cherry Picking algorithm.

4.2.5.16 Cherlcl.h

This file defines the global variables for the Cherry Picking algorithm as external variables so that the program will compile. The actual definition and initialization is in the globals.c file.

4.2.5.17 Makefile.pc

This file contains the method in which to compile all the procedures and create the executable file Cherry.exe. To execute this file type "make makefile.pc".

4.2.6 Error Messages

All error messages give a short message and the file in which the error occurred.

The following message appears when the program has trouble opening a file in which to write the output.
"CANNOT OPEN OUTPUT FILE   cherry.c"

The following message appears when the program has trouble allocating memory to any structure needed for the program.

"ALLOCATION ERROR for list   cherry.c"

The following error appears when the program tries a specific combination of sizes in one section but can not locate the inches for this combination in the input list. The user should check the input file to determine if the value exists or not.

"CANNOT FIND 1 size-30 2 size-32   findinch.c"

4.3 Improvement Algorithm

4.3.1 Functional Description

4.3.1.1 Improvement Algorithm on current solution

The Improvement algorithm takes a current solution for a cut plan and tries to improve this solution by exchanging sizes in different sections. The algorithm begins by first examining the current solution to see if any sections can be combined to make one section that requires less fabric than the two sections. The new section cannot violate the constraints on the maximum ply height or the maximum sizes allowed per section. Then the algorithm begins with any section and tries to transfer a portion of the section to another section, or swap a portion of the section with the portion of another section without violating the constraint
of the maximum number of sizes allowed per section. When all possible transfers and swaps have been examined for the current section, the best transfer or swap is made and the process begins again for the next section. This process is continued until no more transfers or swaps that improve the solution can be made. At this point the new solution is written to the output file and the program terminates.

Before actually transferring or swapping a portion of a section, the program calculates a possible savings in fabric for each possible combination of sections. There are basically two ways to transfer or swap sizes. If the portion taken from one section and the candidate section contain the same size combination the savings is computed by placing the portion on top of the candidate section. The savings for this method is based on the saved cutting cost since the size combination is only cut once instead of twice. The savings for the merger of a portion of the original section and the entire candidate section containing the same size combination can also be achieved by adding the sizes in the extracted portion to the new section and leaving the ply height the same. The savings for this method is the decreased cost of fabric required for spreading the merged sections. If the portion taken from one section and candidate section do not contain the same size combination then the savings is computed by changing the size combination and leaving the ply height the same.

4.3.1.2 Improvement Algorithm on an order

The Improvement algorithm can also be used to generate a solution from an initial order by either changing the source code slightly or the input file. To accomplish this each unit in the initial order must be placed in a separate section of initial ply height. The improvement algorithm
works as if this were the initial solution and tries to improve upon this solution as explained above.

There are two ways to set up this initial solution for the algorithm. The user can edit the input file to show a initial solution of sections with initial ply heights and one unit of clothing in each section. (See section 4.3.2 Input File). However if the order is large it would be easier to change the source code to omit reading in the first solution from the input file and simply place each size ordered into a section of initial ply height. The two files which would need to be changed are the getparm.c file and the improve.c file. In the getparm.c file the code that reads in the initial solution would have to be deleted. In the improve.c file new source code would have to be added which would take the order and place each unit of fabric into a separate section of initial ply height. This code is represented below:

```c
curr_section = 0;
for (i=0; i< num_of_sizes; i++) {
    for (j=0; j<order[i]; j++) {
        sections[curr_section].ply_height = initial_ply;
        sections[curr_section].sizes[i] = 1;
        ++curr_section;
    }
}
```

The Improvement algorithm will then generate a solution based on the order and write out the solution to the output file.

Appendix C contains a detailed description of the algorithm and Appendix F contains the source code for the Improvement algorithm.
The Improvement algorithm is run simply by typing in "improve" at the DOS prompt in the directory in which the Improvement program is located. An input file, explained in section 4.3.2 needs to be in the current directory in which the user is running the Improvement algorithm. All output is written to a file also in that current directory.

4.3.2 Input File

The input file is an ASCII file which must contain the variables needed to run the Improvement algorithm. The inputs include (1) an order to be cut, consisting of various sizes required, and the quantity desired of each of these sizes (2) The number of units over or under the demand that will be allowed. (3) List of L_i's (these are the fabric lengths required for cutting a size combination i - like small and large together - in a particular section. (4) A solution to the problem to be improved upon. The solution contains the number of sections and the sizes and ply height assigned to each of those sections, the fabric length required for each of the sections, and the deviation of the number of units to be cut from the actual number of units required in the order. (5) The maximum ply height allowed. (6) The maximum number of sizes allowed per section. (7) The cutting cost per inch of fabric. (8) The unit cost of the fabric.

Each input is placed on a separate line on the input file and must be placed in the following order represented below:

\begin{verbatim}
int ou_units : number of units over or under the demand for 1st solution
int max_ply : maximum ply height
\end{verbatim}

22
int max_sizes : max sizes allowed per section
int cut_cost : cutting cost per inch
int unit_cost : unit cost of size
order_t Order : the amount of each size needed
.num_sections : the # of sections in 1st solution
.sections_t section : the sections for the 1st solution
(old ou_units : # of units over or under demand
for new solution
list_t list : list of I’s (fabric lengths
for each sizes combination)

The beginning of the input file would appear as
follows:

3 : ou_units
3 : max_sizes
4 : max_ply
1 : init_ply
1 : k
1 : cut_cost
1 : unit_cost
5 120 size-30 : order ( #, perimeter, string for size)
5 130 size-32 : order ( #, perimeter, string for size)
5 140 size-34 : order ( #, perimeter, string for size)
5 150 size-36 : order ( #, perimeter, string for size)
2 : number of sections
1 0 2 3 -1 2 : sections (sizes, ply height)
1 2 2 3 -1 3 : sections (sizes, ply height)
3 : ou_units for new_section
1 0 -1 15.31 : list ( 1 size 0, inches)
1 1 -1 15.56 : list ( 1 size 1, inches)
.
.
-2 : end

A detailed description of the data structures for an
order, the list of L’s, and a solution is in section 4.3.4.
As each row of the order is read from the input file, the values are placed in the ord_var_t structure. For example, the first row (5 120 size-30) is placed as follows:

number[0] = 5
ch_sizes[0] = "size-30"
perimeter[0] = 120

The input file for the original solution reads as follows:

1 size-30, 2 size-36 with a ply height of 2
1 size-34, 2 size-36 with a ply height of 3

As each row in the input file for the original solution is read, the variables are placed in a section list as follows:

section[0].sizes[0] = 1
section[0].sizes[1] = 0
section[0].sizes[2] = 0
section[0].sizes[3] = 2
section[0].ply_height = 2

section[1].sizes[0] = 0
section[1].sizes[1] = 0
section[1].sizes[2] = 1
section[1].sizes[3] = 2
section[1].ply_height = 3

As each row in the list of L_is is read, the variables are placed in the list_t array as follows:
List[0].sizes[0] = 1  ( 1 size-30 )
List[0].sizes[1] = 0
List[0].inches = 15.31

List[1].sizes[0] = 0
List[1].sizes[1] = 1
List[1].inches = 15.56

Appendix A contains a printout of the entire input file.

4.3.3 Output File

The output file ("OUTPUT") lists both the first solution, given as the input, the second solution generated by the algorithm, the sizes in each section, the inches for that section, the ply height, and the total inches for the section (ply x inches). At the end the total inches are printed along with the number of units over or under the demand. A sample output file is in Appendix B.

4.3.4 Data Structures

MAX_SIZES defines the maximum number of different sizes possible in one order.

MAX_LIST defines the maximum number of possible size combinations as in the input file.

The following structure holds the order which consists of the amount of each size needed, the string value that will be written to the output file associated with that size, and the perimeter of the particular size:

```c
struct order {
    order_t number;
    sizes_t ch_sizes;
```
int perimeter[MAX_SIZES];
} ord_var_t;

Order.number is an array which holds the amount of each size ordered. It is defined as follows:
typedef int order_t[MAX_SIZES].

Order.ch_sizes is an array which holds the string value associated with the size. e.g "Size-32". It is defined as follows:
typedef char sizes_t[MAX_SIZES][10];

Order.perimeter is an array which holds the perimeter of each size.

Arrays in the C language begin with zero and not one. Therefore there will be a value in the first position, zero. If the order is 3 size-30 and 5 size-32, the ord_var_t structure will hold the following numbers:

number[0] = 3
ch_sizes[0] = "size-30"
perimeter[0] = 120

number[1] = 5
ch_sizes[1] = "size-32"
perimeter[1] = 130

The following structure holds the list of L_t's:

struct list_t {
    order_t sizes;
    float inches;
} list

List.sizes is an array which holds the amount of each size in the specific size combination.

List.inches holds the amount of fabric in inches need for that specific sizes combination.

An example of what is in the list[] structure:

sizes[0] = 1  ( 1 size-30 )
sizes[1] = 1  ( 1 size-32)
inches = 28.01

The following structure holds the data necessary for each section.
struct section_t {
    order_t sizes;
    int ply_height;
} section

Section.order is an array which holds the amount of each size in the specific size combination for this particular section.

Section.ply_height is the ply height of that section.

The following structure contains the data for all the savings calculated and saved in a list:

struct savings_t {
    int sect1
    int sect2
    int org_ply_height
    int cand_ply_height
    float savings;
    int type;
    order_t org;
    order_t cand;
    order_t in_sect1;
    order_t in_sect2;
} save_list;

Save_list.sect1 and save_list.sect2 are the numbers of the two sections being merged. These numbers correspond to their position in the section list.

Save_list.org_ply_height is the ply height of the originating section in which the portion was taken.

Save_list.cand_ply_height is the ply height of the candidate section in which the portion will be added or swapped.

Save_list.savings is the savings achieved if these two sections are merged.

Save_list.type is the type of merge that is needed.
1 : Sizes the same, place on top of another
2 : Sizes different, rearrange size combination
Save_list.org is an array which holds sizes of the originating portion to transfer or swap.

Save_list.cand is an array which holds the sizes of the candidate portion to transfer or swap.

Save_list.insect1 is an array which holds all the sizes in the original section before the swap or transfer is made.

Save_list.insect2 is an array which holds all the sizes in the candidate section before the swap or transfer is made.

4.3.5 Module Descriptions

4.3.5.1 Case_ai.c
Case_ai() computes the savings of placing one section or portion of a section on top of another section. The two sections must have the same size combination.

4.3.5.2 Case_aii.c
Case_aii() computes the savings of rearranging the size combination of the two sections into one section but keeping the ply height the same.

4.3.5.3 Combply.c
Combine_ply() combines sections which have the same sizes combination into one section if the new ply height does not violate the maximum ply allowed.

4.3.5.4 Combsize.c
Combine_sizes() combines sections which have the same ply height if the number of sizes in the section does not violate the maximum number of sizes allowed per section.
4.3.5.5 Compswap.c

Compswap() looks at the two sizes and sections to swap to determine which way to compute the savings. If the two sections have the same size combination then caseai.c is called, else caseaii.c is called. Ply height of the sections are adjusted and the savings is returned to the main program.

4.3.5.6 Compute.c

Compute_savings() looks at the size to transfer and the two sections to determine which way to compute the savings. If the two sections have the same sizes combination then caseai.c is called else caseaii.c is called. Ply height of the sections are adjusted and the savings is returned to the main program.

4.3.5.7 Findinch.c

Find_inches() searches through the list of $L_i$ inputs to find the inches of the size combination of a section.

4.3.5.8 Getparm.c

Get_parameters() opens the input file and reads in all the inputs into their appropriate variables and structures.

4.3.5.9 Globals.c

Globals defines and initializes all the global variables for the Improvement algorithm.

4.3.5.10 Improve.c

This file contains the procedure to execute the Improvement algorithm. The procedure begins by allocating all the memory needed for the variables and lists, and then calls get_parameters() to read the input file. The main loop in the procedure continually tries to transfer and swap
sizes between sections until no more transfers or swaps can be made to improve the solution. The procedure then writes all the final data to the file "Output", frees all memory, and closes any open files.

4.3.5.11 Swapbkwd
Swap_backwards() attempts to swap one size from one section with a size from another section working backwards through the list.

4.3.5.12 Swapfrwd
Swap_forwards() attempts to swap one size from one section with a size from another section working forwards through the list.

4.3.5.13 Tranbkwd
Transfer_backwards() attempts to transfer one size from one section to another section working backwards through the list.

4.3.5.14 Tranfrwd
Transfer_forwards() attempts to transfer one size from one section to another section working forwards through the list.

4.3.5.15 Impdec.h
This file contains all the definitions for various structures and procedures that are used by the Improvement algorithm.

4.3.5.16 Implcl.h
This file defines the global variables for the Improvement algorithm as external variables so that the
program will compile. The actual definition and initialization is in the globals.c file.

4.3.5.17 Makefile.pc

This file contains the method in which to compile all the procedures and create the executable file Improve.exe. To execute this file type "make makefile.pc".

4.3.6 Error Messages

All error messages give a short message and the file in which the error occurred.

The following message appears when the program has trouble opening a file in which to write the output.

"CANNOT OPEN OUTPUT FILE improve.c"

The following message appears when the program has trouble allocating memory to any structure needed for the program.

"ALLOCATION ERROR for list improve.c"

The following error appears when the program tries a specific combination of sizes in one section but can not locate the inches for this combination in the input list. The user should check the input file to determine if the value exists or not.

"CANNOT FIND 1 size-30 2 size-32 findinch.c"
5.0 References


6.0 Appendices

Appendix A  Input File

Each algorithm has a unique input file. The Li's in the three input files (one for each algorithm) are all the same. However, each input file begins slightly different depending on the other input variables needed for the algorithm. All other input files are the same except for the initial inputs. The input file in this appendix is used in the Savings algorithm. The beginning of each of the three input files are explained in the sections 4.1.2 Savings Input File, 4.2.2 Cherry Input File, 4.3.2 Improvement Input File.
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>72</td>
<td>120</td>
<td>size-30</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>144</td>
<td>130</td>
<td>size-32</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>360</td>
<td>140</td>
<td>size-34</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>360</td>
<td>150</td>
<td>size-36</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>144</td>
<td>160</td>
<td>size-38</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>72</td>
<td>170</td>
<td>size-40</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>15.31</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>15.56</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>15.82</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>3</td>
<td>-1</td>
<td>16.07</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>16.33</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>5</td>
<td>-1</td>
<td>16.59</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>27.75</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>28.26</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>2</td>
<td>-1</td>
<td>28.76</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>3</td>
<td>-1</td>
<td>29.27</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>4</td>
<td>-1</td>
<td>29.77</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
<td>5</td>
<td>-1</td>
<td>30.28</td>
</tr>
<tr>
<td>28</td>
<td>3</td>
<td>0</td>
<td>-1</td>
<td>40.36</td>
</tr>
<tr>
<td>29</td>
<td>3</td>
<td>1</td>
<td>-1</td>
<td>41.11</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>2</td>
<td>-1</td>
<td>41.87</td>
</tr>
<tr>
<td>31</td>
<td>3</td>
<td>3</td>
<td>-1</td>
<td>42.63</td>
</tr>
<tr>
<td>32</td>
<td>3</td>
<td>4</td>
<td>-1</td>
<td>43.38</td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>5</td>
<td>-1</td>
<td>44.14</td>
</tr>
<tr>
<td>34</td>
<td>4</td>
<td>0</td>
<td>-1</td>
<td>52.93</td>
</tr>
<tr>
<td>35</td>
<td>4</td>
<td>1</td>
<td>-1</td>
<td>53.94</td>
</tr>
<tr>
<td>36</td>
<td>4</td>
<td>2</td>
<td>-1</td>
<td>54.95</td>
</tr>
<tr>
<td>37</td>
<td>4</td>
<td>3</td>
<td>-1</td>
<td>55.96</td>
</tr>
<tr>
<td>38</td>
<td>4</td>
<td>4</td>
<td>-1</td>
<td>56.97</td>
</tr>
<tr>
<td>39</td>
<td>4</td>
<td>5</td>
<td>-1</td>
<td>57.98</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>0</td>
<td>-1</td>
<td>65.50</td>
</tr>
<tr>
<td>41</td>
<td>5</td>
<td>1</td>
<td>-1</td>
<td>66.76</td>
</tr>
<tr>
<td>42</td>
<td>5</td>
<td>2</td>
<td>-1</td>
<td>68.03</td>
</tr>
<tr>
<td>43</td>
<td>5</td>
<td>3</td>
<td>-1</td>
<td>69.28</td>
</tr>
<tr>
<td>44</td>
<td>5</td>
<td>4</td>
<td>-1</td>
<td>70.54</td>
</tr>
<tr>
<td>45</td>
<td>5</td>
<td>5</td>
<td>-1</td>
<td>71.80</td>
</tr>
<tr>
<td>46</td>
<td>6</td>
<td>0</td>
<td>-1</td>
<td>69.19</td>
</tr>
<tr>
<td>47</td>
<td>6</td>
<td>1</td>
<td>-1</td>
<td>70.52</td>
</tr>
<tr>
<td>48</td>
<td>6</td>
<td>2</td>
<td>-1</td>
<td>71.86</td>
</tr>
<tr>
<td>49</td>
<td>6</td>
<td>3</td>
<td>-1</td>
<td>73.19</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>4</td>
<td>-1</td>
<td>74.52</td>
</tr>
<tr>
<td>51</td>
<td>6</td>
<td>5</td>
<td>-1</td>
<td>75.86</td>
</tr>
<tr>
<td>52</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1 -128.01</td>
</tr>
<tr>
<td>53</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2 -128.26</td>
</tr>
<tr>
<td>54</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3 -128.51</td>
</tr>
<tr>
<td>55</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4 -128.76</td>
</tr>
<tr>
<td>56</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5 -129.02</td>
</tr>
<tr>
<td>57</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2 -128.52</td>
</tr>
<tr>
<td>Index</td>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
<td>Column 4</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>58</td>
<td>1 1 1 3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>1 1 1 4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>1 1 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>1 2 1 3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>1 2 1 4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>1 2 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>1 3 1 4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>1 3 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>1 4 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>2 0 1 1</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>2 0 1 2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>2 0 1 3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>2 0 1 4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>2 0 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>2 1 1 0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>2 1 1 2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>2 1 1 3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>2 1 1 4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>2 1 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>2 2 1 0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>2 2 1 1</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>2 2 1 3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>2 2 1 4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>2 2 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>2 3 1 0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>2 3 1 1</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>2 3 1 2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>2 3 1 4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>2 3 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>2 4 1 0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>2 4 1 1</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>2 4 1 2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>2 4 1 3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>2 4 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>2 5 1 0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>2 5 1 1</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>2 5 1 2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>2 5 1 3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>2 5 1 4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>3 0 1 1</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>3 0 1 2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>3 0 1 3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>3 0 1 4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>3 0 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>3 1 1 0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>3 1 1 2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>3 1 1 3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>3 1 1 4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>3 1 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>3 2 1 0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>3 2 1 1</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>3 2 1 3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>3 2 1 4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>3 2 1 5</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>3 3 1 0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>3 3 1 1</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>3 3 1 2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>115</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>116</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>117</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>118</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>119</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>120</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>121</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>122</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>123</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>124</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>125</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>126</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>127</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>129</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>130</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>131</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>132</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>133</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>134</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>135</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>136</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>137</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>138</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>139</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>140</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>141</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>142</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>143</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>144</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>145</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>146</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>147</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>148</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>149</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>150</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>151</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>152</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>153</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>154</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>155</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>156</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>157</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>158</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>159</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>160</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>161</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>162</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>163</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>164</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>165</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>166</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>167</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>168</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>169</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>170</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>171</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>172</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>173</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>174</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>175</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>176</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>177</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>178</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>179</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>180</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>181</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>182</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>183</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>184</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>185</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>186</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>187</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>188</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>189</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>190</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>191</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>192</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>193</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>194</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>195</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>196</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>197</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>198</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>199</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>201</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>202</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>203</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>204</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>205</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>206</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>207</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>208</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>209</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>210</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>211</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>212</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>213</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>214</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>215</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>216</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>217</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>218</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>219</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>220</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>221</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>222</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>223</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>224</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>225</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>226</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>227</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>228</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>286</td>
<td>3 0 1 2 1 4</td>
<td>-1</td>
<td>67.01</td>
<td></td>
</tr>
<tr>
<td>287</td>
<td>3 0 1 2 1 5</td>
<td>-1</td>
<td>67.26</td>
<td></td>
</tr>
<tr>
<td>288</td>
<td>3 0 1 2 2 3</td>
<td>-1</td>
<td>70.97</td>
<td></td>
</tr>
<tr>
<td>289</td>
<td>3 0 1 2 2 4</td>
<td>-1</td>
<td>71.41</td>
<td></td>
</tr>
<tr>
<td>290</td>
<td>3 0 1 2 2 5</td>
<td>-1</td>
<td>71.86</td>
<td></td>
</tr>
<tr>
<td>291</td>
<td>3 0 2 2 1 3</td>
<td>-1</td>
<td>70.75</td>
<td></td>
</tr>
<tr>
<td>292</td>
<td>3 0 2 2 1 4</td>
<td>-1</td>
<td>70.97</td>
<td></td>
</tr>
<tr>
<td>293</td>
<td>3 0 2 2 1 5</td>
<td>-1</td>
<td>71.19</td>
<td></td>
</tr>
<tr>
<td>294</td>
<td>3 0 1 3 1 4</td>
<td>-1</td>
<td>67.26</td>
<td></td>
</tr>
<tr>
<td>295</td>
<td>3 0 1 3 1 5</td>
<td>-1</td>
<td>67.52</td>
<td></td>
</tr>
<tr>
<td>296</td>
<td>3 0 1 4 1 5</td>
<td>-1</td>
<td>69.77</td>
<td></td>
</tr>
<tr>
<td>297</td>
<td>3 0 2 3 1 4</td>
<td>-1</td>
<td>71.41</td>
<td></td>
</tr>
<tr>
<td>298</td>
<td>3 0 2 3 1 5</td>
<td>-1</td>
<td>71.63</td>
<td></td>
</tr>
<tr>
<td>299</td>
<td>3 0 1 3 2 4</td>
<td>-1</td>
<td>71.63</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>3 0 2 4 1 5</td>
<td>-1</td>
<td>72.08</td>
<td></td>
</tr>
<tr>
<td>301</td>
<td>3 0 1 4 2 5</td>
<td>-1</td>
<td>72.3</td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>4 0 1 1 1 2</td>
<td>-1</td>
<td>69.26</td>
<td></td>
</tr>
<tr>
<td>303</td>
<td>4 0 1 1 1 3</td>
<td>-1</td>
<td>70.08</td>
<td></td>
</tr>
<tr>
<td>304</td>
<td>4 0 1 1 1 4</td>
<td>-1</td>
<td>70.3</td>
<td></td>
</tr>
<tr>
<td>305</td>
<td>4 0 1 1 1 5</td>
<td>-1</td>
<td>70.52</td>
<td></td>
</tr>
<tr>
<td>306</td>
<td>4 0 1 2 1 3</td>
<td>-1</td>
<td>70.3</td>
<td></td>
</tr>
<tr>
<td>307</td>
<td>4 0 1 2 1 4</td>
<td>-1</td>
<td>70.52</td>
<td></td>
</tr>
<tr>
<td>308</td>
<td>4 0 1 2 1 5</td>
<td>-1</td>
<td>70.75</td>
<td></td>
</tr>
<tr>
<td>309</td>
<td>4 0 1 3 1 4</td>
<td>-1</td>
<td>70.75</td>
<td></td>
</tr>
<tr>
<td>310</td>
<td>4 0 1 3 1 5</td>
<td>-1</td>
<td>70.97</td>
<td></td>
</tr>
<tr>
<td>311</td>
<td>4 0 1 4 1 5</td>
<td>-1</td>
<td>71.19</td>
<td></td>
</tr>
<tr>
<td>312</td>
<td>1 1 1 2 3 3</td>
<td>-1</td>
<td>68.52</td>
<td></td>
</tr>
<tr>
<td>313</td>
<td>1 1 1 2 3 4</td>
<td>-1</td>
<td>69.28</td>
<td></td>
</tr>
<tr>
<td>314</td>
<td>1 1 1 2 3 5</td>
<td>-1</td>
<td>70.04</td>
<td></td>
</tr>
<tr>
<td>315</td>
<td>1 1 1 2 4 3</td>
<td>-1</td>
<td>72.52</td>
<td></td>
</tr>
<tr>
<td>316</td>
<td>1 1 1 2 4 4</td>
<td>-1</td>
<td>73.41</td>
<td></td>
</tr>
<tr>
<td>317</td>
<td>1 1 1 2 4 5</td>
<td>-1</td>
<td>74.3</td>
<td></td>
</tr>
<tr>
<td>318</td>
<td>1 1 2 2 3 3</td>
<td>-1</td>
<td>72.3</td>
<td></td>
</tr>
<tr>
<td>319</td>
<td>1 1 2 2 3 4</td>
<td>-1</td>
<td>72.97</td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>1 1 3 2 1 3</td>
<td>-1</td>
<td>68.02</td>
<td></td>
</tr>
<tr>
<td>321</td>
<td>1 1 3 2 1 4</td>
<td>-1</td>
<td>68.27</td>
<td></td>
</tr>
<tr>
<td>322</td>
<td>1 1 3 2 1 5</td>
<td>-1</td>
<td>68.52</td>
<td></td>
</tr>
<tr>
<td>323</td>
<td>1 1 3 2 2 3</td>
<td>-1</td>
<td>72.08</td>
<td></td>
</tr>
<tr>
<td>324</td>
<td>1 1 3 2 2 4</td>
<td>-1</td>
<td>72.52</td>
<td></td>
</tr>
<tr>
<td>325</td>
<td>1 1 3 2 2 5</td>
<td>-1</td>
<td>72.97</td>
<td></td>
</tr>
<tr>
<td>326</td>
<td>1 1 4 2 1 3</td>
<td>-1</td>
<td>71.86</td>
<td></td>
</tr>
<tr>
<td>327</td>
<td>1 1 4 2 1 4</td>
<td>-1</td>
<td>72.08</td>
<td></td>
</tr>
<tr>
<td>328</td>
<td>1 1 4 2 1 5</td>
<td>-1</td>
<td>72.3</td>
<td></td>
</tr>
<tr>
<td>329</td>
<td>1 1 3 3 1 4</td>
<td>-1</td>
<td>69.03</td>
<td></td>
</tr>
<tr>
<td>330</td>
<td>1 1 3 3 1 5</td>
<td>-1</td>
<td>69.28</td>
<td></td>
</tr>
<tr>
<td>331</td>
<td>1 1 3 3 3 4</td>
<td>-1</td>
<td>69.53</td>
<td></td>
</tr>
<tr>
<td>332</td>
<td>1 1 3 4 1 5</td>
<td>-1</td>
<td>70.04</td>
<td></td>
</tr>
<tr>
<td>333</td>
<td>1 1 3 3 3 5</td>
<td>-1</td>
<td>70.29</td>
<td></td>
</tr>
<tr>
<td>334</td>
<td>1 1 4 3 4 5</td>
<td>-1</td>
<td>70.54</td>
<td></td>
</tr>
<tr>
<td>335</td>
<td>1 1 4 3 1 4</td>
<td>-1</td>
<td>72.97</td>
<td></td>
</tr>
<tr>
<td>336</td>
<td>1 1 4 3 1 5</td>
<td>-1</td>
<td>73.19</td>
<td></td>
</tr>
<tr>
<td>337</td>
<td>1 1 3 3 2 4</td>
<td>-1</td>
<td>73.19</td>
<td></td>
</tr>
<tr>
<td>338</td>
<td>1 1 2 3 3 4</td>
<td>-1</td>
<td>73.41</td>
<td></td>
</tr>
<tr>
<td>339</td>
<td>1 1 3 3 4 4</td>
<td>-1</td>
<td>73.64</td>
<td></td>
</tr>
<tr>
<td>340</td>
<td>1 1 3 3 2 5</td>
<td>-1</td>
<td>73.64</td>
<td></td>
</tr>
<tr>
<td>341</td>
<td>1 1 4 4 1 5</td>
<td>-1</td>
<td>74.08</td>
<td></td>
</tr>
<tr>
<td>342</td>
<td>1 1 3 4 2 5</td>
<td>-1</td>
<td>74.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>343</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>344</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>345</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>346</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>347</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>348</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>349</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>350</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>351</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>352</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>353</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>354</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>355</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>356</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>357</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>358</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>359</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>360</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>361</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>362</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>363</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>364</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>365</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>366</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>367</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>368</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>369</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>370</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>371</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>372</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>373</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>374</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>375</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>376</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>377</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>378</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>379</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>380</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>381</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>382</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>383</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>384</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>385</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>386</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>387</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>388</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>389</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>390</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>391</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>392</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>393</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>394</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>395</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>396</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>397</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>398</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>399</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>457</td>
<td>4 1 1 3</td>
<td>-1</td>
<td>67.26</td>
<td></td>
</tr>
<tr>
<td>458</td>
<td>4 1 1 4</td>
<td>-1</td>
<td>67.52</td>
<td></td>
</tr>
<tr>
<td>459</td>
<td>4 1 1 5</td>
<td>-1</td>
<td>67.77</td>
<td></td>
</tr>
<tr>
<td>460</td>
<td>1 0 4 2</td>
<td>-1</td>
<td>67.52</td>
<td></td>
</tr>
<tr>
<td>461</td>
<td>1 1 4 2</td>
<td>-1</td>
<td>67.77</td>
<td></td>
</tr>
<tr>
<td>462</td>
<td>4 2 1 3</td>
<td>-1</td>
<td>68.14</td>
<td></td>
</tr>
<tr>
<td>463</td>
<td>4 2 1 4</td>
<td>-1</td>
<td>68.52</td>
<td></td>
</tr>
<tr>
<td>464</td>
<td>4 2 1 5</td>
<td>-1</td>
<td>68.78</td>
<td></td>
</tr>
<tr>
<td>465</td>
<td>1 0 4 3</td>
<td>-1</td>
<td>68.26</td>
<td></td>
</tr>
<tr>
<td>466</td>
<td>1 1 4 3</td>
<td>-1</td>
<td>68.78</td>
<td></td>
</tr>
<tr>
<td>467</td>
<td>1 2 4 3</td>
<td>-1</td>
<td>69.14</td>
<td></td>
</tr>
<tr>
<td>468</td>
<td>4 3 1 4</td>
<td>-1</td>
<td>69.53</td>
<td></td>
</tr>
<tr>
<td>469</td>
<td>4 3 1 5</td>
<td>-1</td>
<td>69.78</td>
<td></td>
</tr>
<tr>
<td>470</td>
<td>1 0 4 4</td>
<td>-1</td>
<td>69.53</td>
<td></td>
</tr>
<tr>
<td>471</td>
<td>1 1 4 4</td>
<td>-1</td>
<td>69.78</td>
<td></td>
</tr>
<tr>
<td>472</td>
<td>1 2 4 4</td>
<td>-1</td>
<td>70.04</td>
<td></td>
</tr>
<tr>
<td>473</td>
<td>1 3 4 4</td>
<td>-1</td>
<td>70.30</td>
<td></td>
</tr>
<tr>
<td>474</td>
<td>4 4 1 5</td>
<td>-1</td>
<td>70.75</td>
<td></td>
</tr>
<tr>
<td>475</td>
<td>1 0 4 5</td>
<td>-1</td>
<td>70.52</td>
<td></td>
</tr>
<tr>
<td>476</td>
<td>1 1 4 5</td>
<td>-1</td>
<td>70.75</td>
<td></td>
</tr>
<tr>
<td>477</td>
<td>1 2 4 5</td>
<td>-1</td>
<td>70.97</td>
<td></td>
</tr>
<tr>
<td>478</td>
<td>1 3 4 5</td>
<td>-1</td>
<td>71.19</td>
<td></td>
</tr>
<tr>
<td>479</td>
<td>1 4 4 5</td>
<td>-1</td>
<td>71.41</td>
<td></td>
</tr>
<tr>
<td>480</td>
<td>2 0 2 1</td>
<td>1 2</td>
<td>-1 66.51</td>
<td></td>
</tr>
<tr>
<td>481</td>
<td>2 0 2 1</td>
<td>1 3</td>
<td>-1 66.76</td>
<td></td>
</tr>
<tr>
<td>482</td>
<td>2 0 2 1</td>
<td>1 4</td>
<td>-1 67.01</td>
<td></td>
</tr>
<tr>
<td>483</td>
<td>2 0 2 1</td>
<td>1 5</td>
<td>-1 67.26</td>
<td></td>
</tr>
<tr>
<td>484</td>
<td>1 0 2 1</td>
<td>2 2</td>
<td>-1 67.01</td>
<td></td>
</tr>
<tr>
<td>485</td>
<td>2 1 2 1</td>
<td>1 3</td>
<td>-1 67.77</td>
<td></td>
</tr>
<tr>
<td>486</td>
<td>2 1 2 1</td>
<td>1 4</td>
<td>-1 68.02</td>
<td></td>
</tr>
<tr>
<td>487</td>
<td>2 1 2 1</td>
<td>1 5</td>
<td>-1 68.27</td>
<td></td>
</tr>
<tr>
<td>488</td>
<td>1 0 2 2</td>
<td>2 3</td>
<td>-1 68.02</td>
<td></td>
</tr>
<tr>
<td>489</td>
<td>1 1 2 2</td>
<td>2 3</td>
<td>-1 68.27</td>
<td></td>
</tr>
<tr>
<td>490</td>
<td>2 2 3 3</td>
<td>1 4</td>
<td>-1 69.03</td>
<td></td>
</tr>
<tr>
<td>491</td>
<td>2 2 3 3</td>
<td>1 5</td>
<td>-1 69.28</td>
<td></td>
</tr>
<tr>
<td>492</td>
<td>1 0 2 3</td>
<td>2 4</td>
<td>-1 69.03</td>
<td></td>
</tr>
<tr>
<td>493</td>
<td>1 1 2 3</td>
<td>2 4</td>
<td>-1 69.28</td>
<td></td>
</tr>
<tr>
<td>494</td>
<td>1 2 2 3</td>
<td>2 4</td>
<td>-1 69.53</td>
<td></td>
</tr>
<tr>
<td>495</td>
<td>2 3 2 4</td>
<td>1 5</td>
<td>-1 70.29</td>
<td></td>
</tr>
<tr>
<td>496</td>
<td>1 0 2 4</td>
<td>2 5</td>
<td>-1 70.04</td>
<td></td>
</tr>
<tr>
<td>497</td>
<td>1 1 2 4</td>
<td>2 5</td>
<td>-1 70.29</td>
<td></td>
</tr>
<tr>
<td>498</td>
<td>1 2 2 4</td>
<td>2 5</td>
<td>-1 70.54</td>
<td></td>
</tr>
<tr>
<td>499</td>
<td>1 3 2 4</td>
<td>2 5</td>
<td>-1 70.79</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>2 0 1 1</td>
<td>2 2</td>
<td>-1 66.76</td>
<td></td>
</tr>
<tr>
<td>501</td>
<td>2 0 2 2</td>
<td>1 3</td>
<td>-1 67.26</td>
<td></td>
</tr>
<tr>
<td>502</td>
<td>2 0 2 2</td>
<td>1 4</td>
<td>-1 67.52</td>
<td></td>
</tr>
<tr>
<td>503</td>
<td>2 0 2 2</td>
<td>1 5</td>
<td>-1 67.77</td>
<td></td>
</tr>
<tr>
<td>504</td>
<td>2 0 1 2</td>
<td>2 3</td>
<td>-1 67.26</td>
<td></td>
</tr>
<tr>
<td>505</td>
<td>2 0 1 2</td>
<td>2 3</td>
<td>-1 67.51</td>
<td></td>
</tr>
<tr>
<td>506</td>
<td>2 0 2 3</td>
<td>1 4</td>
<td>-1 68.02</td>
<td></td>
</tr>
<tr>
<td>507</td>
<td>2 0 2 3</td>
<td>1 5</td>
<td>-1 68.27</td>
<td></td>
</tr>
<tr>
<td>508</td>
<td>2 0 1 1</td>
<td>2 4</td>
<td>-1 67.52</td>
<td></td>
</tr>
<tr>
<td>509</td>
<td>2 0 1 2</td>
<td>2 4</td>
<td>-1 68.02</td>
<td></td>
</tr>
<tr>
<td>510</td>
<td>2 0 1 3</td>
<td>2 4</td>
<td>-1 68.27</td>
<td></td>
</tr>
<tr>
<td>511</td>
<td>2 0 2 4</td>
<td>1 5</td>
<td>-1 68.78</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>2 0 1 2</td>
<td>2 5</td>
<td>-1 68.27</td>
<td></td>
</tr>
<tr>
<td>513</td>
<td>2 0 1 2</td>
<td>2 5</td>
<td>-1 68.52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>514</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>515</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>516</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>517</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>518</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>519</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>520</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>521</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>522</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>523</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>524</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>525</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>526</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>527</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>528</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>529</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>530</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>531</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>532</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>533</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>534</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>535</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>536</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>537</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>538</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>539</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>540</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>541</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>542</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>543</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>544</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>545</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>546</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>547</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>548</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>549</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>550</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>551</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>552</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>553</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>554</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>555</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>556</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>557</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>558</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>559</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>560</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>561</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>562</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>563</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>564</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>565</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>566</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>567</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>568</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>569</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>570</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>628</td>
<td>2 0 1 2 1 4 2 5 -1</td>
<td>72.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>629</td>
<td>2 0 1 3 1 4 2 5 -1</td>
<td>72.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>630</td>
<td>1 0 2 1 2 2 1 3 -1</td>
<td>71.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>631</td>
<td>1 0 2 1 2 2 1 4 -1</td>
<td>71.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>632</td>
<td>1 0 2 1 2 2 1 5 -1</td>
<td>71.63</td>
<td></td>
<td></td>
</tr>
<tr>
<td>633</td>
<td>2 1 2 2 1 3 1 4 -1</td>
<td>72.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>634</td>
<td>2 1 2 2 1 3 1 5 -1</td>
<td>72.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>635</td>
<td>2 1 2 2 1 4 1 5 -1</td>
<td>72.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>636</td>
<td>1 0 2 1 1 2 2 3 -1</td>
<td>71.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>637</td>
<td>1 0 2 1 2 3 1 4 -1</td>
<td>71.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>638</td>
<td>1 0 2 1 2 3 1 5 -1</td>
<td>72.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>639</td>
<td>2 1 1 2 2 3 1 4 -1</td>
<td>72.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>640</td>
<td>2 1 1 2 2 3 1 5 -1</td>
<td>72.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>641</td>
<td>2 1 2 3 1 4 1 5 -1</td>
<td>72.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>642</td>
<td>1 0 2 1 1 2 2 4 -1</td>
<td>71.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>643</td>
<td>1 0 2 1 1 3 2 4 -1</td>
<td>72.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>644</td>
<td>1 0 2 1 2 4 1 5 -1</td>
<td>72.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>645</td>
<td>2 1 1 2 1 3 2 4 -1</td>
<td>72.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>646</td>
<td>2 1 1 2 2 4 1 5 -1</td>
<td>72.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>647</td>
<td>2 1 1 3 2 4 1 5 -1</td>
<td>73.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>648</td>
<td>1 0 2 1 1 2 2 5 -1</td>
<td>72.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>649</td>
<td>1 0 2 1 1 3 2 5 -1</td>
<td>72.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>650</td>
<td>1 0 2 1 1 4 2 5 -1</td>
<td>72.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>651</td>
<td>2 1 1 2 1 3 2 5 -1</td>
<td>72.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>652</td>
<td>2 1 1 2 1 4 2 5 -1</td>
<td>73.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>653</td>
<td>2 1 1 3 1 4 2 5 -1</td>
<td>73.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>654</td>
<td>1 0 1 1 2 2 2 3 -1</td>
<td>71.63</td>
<td></td>
<td></td>
</tr>
<tr>
<td>655</td>
<td>1 0 2 2 2 3 1 4 -1</td>
<td>72.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>656</td>
<td>1 0 2 2 3 3 1 5 -1</td>
<td>72.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>657</td>
<td>1 1 2 2 2 3 1 4 -1</td>
<td>72.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>658</td>
<td>1 1 2 2 3 1 5 -1</td>
<td>72.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>659</td>
<td>2 2 2 3 1 4 1 5 -1</td>
<td>73.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>660</td>
<td>1 0 1 1 2 2 2 4 -1</td>
<td>72.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>661</td>
<td>1 0 2 1 1 3 2 4 -1</td>
<td>72.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>662</td>
<td>1 0 2 2 2 4 1 5 -1</td>
<td>72.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>663</td>
<td>1 1 2 2 3 2 4 -1</td>
<td>72.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>664</td>
<td>1 1 2 2 3 1 5 -1</td>
<td>73.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>665</td>
<td>2 2 1 3 2 4 1 5 -1</td>
<td>73.64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>666</td>
<td>1 0 1 1 2 2 2 5 -1</td>
<td>72.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>667</td>
<td>1 0 2 2 1 3 2 5 -1</td>
<td>72.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>668</td>
<td>1 0 2 2 1 4 2 5 -1</td>
<td>73.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>669</td>
<td>1 1 2 2 1 3 2 5 -1</td>
<td>73.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>670</td>
<td>1 1 2 2 1 4 2 5 -1</td>
<td>73.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>671</td>
<td>2 2 1 3 1 4 2 5 -1</td>
<td>73.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>672</td>
<td>1 0 1 1 2 3 2 4 -1</td>
<td>72.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>673</td>
<td>1 0 1 2 2 3 2 4 -1</td>
<td>72.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>674</td>
<td>1 0 2 3 2 4 1 5 -1</td>
<td>73.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>675</td>
<td>1 1 1 2 2 3 2 4 -1</td>
<td>72.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>676</td>
<td>1 1 2 3 2 4 1 5 -1</td>
<td>73.64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>677</td>
<td>1 2 2 3 2 4 1 5 -1</td>
<td>73.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>678</td>
<td>1 0 1 1 2 3 2 5 -1</td>
<td>72.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>679</td>
<td>1 0 1 2 2 3 2 5 -1</td>
<td>73.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>680</td>
<td>1 0 2 3 1 4 2 5 -1</td>
<td>73.64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>681</td>
<td>1 1 1 2 2 3 2 5 -1</td>
<td>73.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>682</td>
<td>1 1 2 3 1 4 2 5 -1</td>
<td>73.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>683</td>
<td>1 2 2 3 1 4 2 5 -1</td>
<td>74.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>684</td>
<td>1 0 1 1 2 4 2 5 -1</td>
<td>73.41</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>685</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>686</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>687</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>688</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>689</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>690</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>691</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>692</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>693</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>694</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>695</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>696</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>697</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>698</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>699</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>700</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>701</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>702</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>703</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>704</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>705</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>706</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>707</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>708</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>709</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>710</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>711</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>712</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>713</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>714</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>715</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>716</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>717</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>718</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>719</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>720</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>721</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>722</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>723</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>724</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>725</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>726</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>727</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>728</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>729</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>730</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>731</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>732</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>733</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>734</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>735</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>736</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>737</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>738</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>739</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>740</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>741</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
I7

742 1 0 1 1 1 4 2 5 -1 69.28
743 1 0 1 2 1 3 2 5 -1 69.28
744 1 0 1 2 1 4 2 5 -1 69.53
745 1 0 1 3 1 4 2 5 -1 69.78
746 1 1 1 2 1 3 2 5 -1 69.53
747 1 1 1 2 1 4 2 5 -1 69.78
748 1 1 1 3 1 4 2 5 -1 70.04
749 1 2 1 3 1 4 2 5 -1 70.29
750 2 0 1 1 1 2 1 3 1 4 -1 71.41
751 2 0 1 1 1 2 1 3 1 5 -1 71.63
752 2 0 1 1 1 2 1 4 1 5 -1 71.86
753 2 0 1 1 1 3 1 4 1 5 -1 72.08
754 2 0 1 1 1 3 1 4 1 5 -1 72.3
755 1 0 2 1 1 2 1 3 1 4 -1 71.63
756 1 0 2 1 1 2 1 3 1 5 -1 71.86
757 1 0 2 1 1 2 1 4 1 5 -1 72.08
758 1 0 2 1 1 3 1 4 1 5 -1 72.3
759 2 1 1 2 1 3 1 4 1 5 -1 72.75
760 1 0 1 1 2 1 3 1 4 -1 71.86
761 1 0 1 1 2 1 3 1 5 -1 72.08
762 1 0 1 1 2 1 4 1 5 -1 72.3
763 1 0 2 1 1 3 1 4 1 5 -1 72.75
764 1 1 2 1 3 1 4 1 5 -1 72.97
765 1 0 1 1 1 2 2 1 3 1 4 -1 72.08
766 1 0 1 1 1 2 2 1 3 1 5 -1 72.3
767 1 0 1 1 2 3 1 4 1 5 -1 72.75
768 1 0 1 2 2 3 1 4 1 5 -1 72.97
769 1 1 1 2 2 3 1 4 1 5 -1 73.19
770 1 0 1 1 1 2 1 3 2 4 -1 72.3
771 1 0 1 1 1 2 1 2 4 1 5 -1 72.75
772 1 0 1 1 1 3 2 4 1 5 -1 72.97
773 1 0 2 1 1 3 2 4 1 5 -1 73.19
774 1 1 1 2 1 3 2 4 1 5 -1 73.41
775 1 0 1 1 1 2 1 3 2 5 -1 72.75
776 1 0 1 1 1 2 1 4 2 5 -1 72.97
777 1 0 1 1 1 3 1 4 2 5 -1 73.19
778 1 0 1 2 1 3 1 4 2 5 -1 73.41
779 1 1 1 2 1 3 1 4 2 5 -1 73.64
780 3 0 1 1 1 2 1 3 -1 70.52
781 3 0 1 1 1 2 1 4 -1 70.75
782 3 0 1 1 1 2 1 5 -1 70.97
783 3 0 1 1 1 3 1 4 -1 70.97
784 3 0 1 1 1 3 1 5 -1 71.19
785 3 0 1 1 1 4 1 5 -1 71.41
786 3 0 1 2 1 3 1 4 -1 71.19
787 3 0 1 2 1 3 1 5 -1 71.41
788 3 0 1 2 1 4 1 5 -1 71.63
789 3 0 1 3 1 4 1 5 -1 71.86
790 1 0 3 1 1 2 1 3 -1 70.97
791 1 0 3 1 1 2 1 4 -1 71.19
792 1 0 3 1 1 2 1 5 -1 71.41
793 1 0 3 1 1 3 1 4 -1 71.41
794 1 0 3 1 1 3 1 5 -1 71.63
795 1 0 3 1 1 4 1 5 -1 71.86
796 3 1 1 2 1 3 1 4 -1 71.86
797 3 1 1 2 1 3 1 5 -1 72.08
798 3 1 1 2 1 4 1 5 -1 72.3
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>-1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>799</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>800</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>801</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>802</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>803</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>804</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>805</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>806</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>807</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>808</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>809</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>810</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>811</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>812</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>813</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>814</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>815</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>816</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>817</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>818</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>819</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>820</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>821</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>822</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>823</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>824</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>825</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>826</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>827</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>828</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>829</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>830</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>831</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>832</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>833</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>834</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>835</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>836</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>837</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>838</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>839</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>840</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>841</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>842</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>843</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>844</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>845</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>846</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>847</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>848</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>849</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>850</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>851</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>852</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>853</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>854</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>855</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>856</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>857</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>858</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>859</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>860</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>861</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>862</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>863</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>864</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>865</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>866</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>867</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>868</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>869</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>870</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>871</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>872</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>873</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>874</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>875</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>876</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>877</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>878</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>879</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>880</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>881</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>882</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>883</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>884</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>885</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>886</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>887</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>888</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>889</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>890</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>891</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>892</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>893</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>894</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>-1</td>
<td>56.21</td>
</tr>
<tr>
<td>895</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>-1</td>
<td>56.46</td>
</tr>
<tr>
<td>896</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>-1</td>
<td>56.72</td>
</tr>
<tr>
<td>897</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>898</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>899</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>900</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>901</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>902</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>903</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>904</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>905</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>906</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>907</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>908</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>909</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>910</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>911</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>912</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>Row</td>
<td>Columns</td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>---------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>913</td>
<td>1 0 2 4 3 5</td>
<td>-1 74.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>914</td>
<td>2 0 1 1 3 5</td>
<td>-1 72.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>915</td>
<td>1 1 2 2 3 5</td>
<td>-1 73.64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>916</td>
<td>1 1 2 3 3 5</td>
<td>-1 74.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>917</td>
<td>1 1 2 4 3 5</td>
<td>-1 74.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>918</td>
<td>2 0 1 2 3 5</td>
<td>-1 72.97</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>919</td>
<td>2 1 1 2 3 5</td>
<td>-1 73.41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>920</td>
<td>2 0 2 1 2 2</td>
<td>-1 70.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>921</td>
<td>2 0 2 1 2 5</td>
<td>-1 70.97</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>922</td>
<td>2 0 2 1 2 4</td>
<td>-1 71.41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>923</td>
<td>2 0 2 1 2 5</td>
<td>-1 71.86</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>924</td>
<td>2 0 2 2 2 3</td>
<td>-1 71.41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>925</td>
<td>2 0 2 2 2 4</td>
<td>-1 71.86</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>926</td>
<td>2 0 2 2 2 5</td>
<td>-1 72.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>927</td>
<td>2 0 2 3 2 4</td>
<td>-1 72.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>928</td>
<td>2 0 2 3 2 5</td>
<td>-1 72.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>929</td>
<td>2 0 2 4 2 5</td>
<td>-1 73.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>930</td>
<td>2 1 2 2 2 3</td>
<td>-1 71.86</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>931</td>
<td>2 1 2 2 2 4</td>
<td>-1 72.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>932</td>
<td>2 1 2 2 2 5</td>
<td>-1 72.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>933</td>
<td>2 1 2 3 2 4</td>
<td>-1 72.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>934</td>
<td>2 1 2 3 2 5</td>
<td>-1 73.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>935</td>
<td>2 1 2 4 2 5</td>
<td>-1 73.64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>936</td>
<td>2 2 2 3 2 4</td>
<td>-1 73.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>937</td>
<td>2 2 2 3 2 5</td>
<td>-1 73.64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>938</td>
<td>2 2 2 4 2 5</td>
<td>-1 74.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>939</td>
<td>2 3 2 4 2 5</td>
<td>-1 74.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>940</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B Output File

The output file for the Savings and Cherry algorithms contains the inputs to the programs and the sections the algorithm created for the solution. The Improvement algorithm contains the inputs to the program, the beginning solution on which to improve, and the sections generated for the solution.
SAVINGS ALGORITHM

MAX PLY = 47  MAX # OF UNITS PER SECTION = 6
K = 1  INIT PLY = 1  Q = 1

ORDER
6 SIZE size-30
9 SIZE size-32
25 SIZE size-34
2 SIZE size-36
5 SIZE size-38
1 SIZE size-40

******************************************************************************
The # of Final Sections are: 4

SECTION 0 HAS PLY = 1

   AND 1 SIZE size-30
   AND 4 SIZE size-38
   AND 1 SIZE size-40

MARKER LENGTH = 73.86 THE TOTAL LENGTH = 73.86

SECTION 1 HAS PLY = 1

   AND 1 SIZE size-32
   AND 2 SIZE size-34
   AND 2 SIZE size-36
   AND 1 SIZE size-38

MARKER LENGTH = 72.52 THE TOTAL LENGTH = 72.52

SECTION 2 HAS PLY = 5

   AND 1 SIZE size-30
   AND 1 SIZE size-32
   AND 4 SIZE size-34

MARKER LENGTH = 71.19 THE TOTAL LENGTH = 355.95

SECTION 3 HAS PLY = 3

   AND 1 SIZE size-32
   AND 1 SIZE size-34

MARKER LENGTH = 28.52 THE TOTAL LENGTH = 85.56

TOT MARKER = 246.09 TOT LENGTH = 587.89, UNIT OVER/UNDER = 0

TOTAL TIME = 0.008000 SECONDS
MAX PLY = 47  MAX # OF UNITS PER SECTION = 6
UNIT COST = 1 cents  CUT COST = 1 cents
ORDER
6 SIZE size-30
9 SIZE size-32
25 SIZE size-34
2 SIZE size-36
5 SIZE size-38
1 SIZE size-40

FIRST SOLUTION
SECTION 0 HAS PLY = 6  
AND 1 SIZE size-30
AND 1 SIZE size-32
AND 4 SIZE size-34
MARKER LENGTH = 71.19  TOTAL LENGTH = 427.14

SECTION 1 HAS PLY = 2  
AND 1 SIZE size-32
AND 1 SIZE size-36
AND 2 SIZE size-38
MARKER LENGTH = 55.96  TOTAL LENGTH = 111.92

SECTION 2 HAS PLY = 1  
AND 1 SIZE size-32
AND 1 SIZE size-34
AND 1 SIZE size-38
AND 1 SIZE size-40
MARKER LENGTH = 55.90  TOTAL LENGTH = 55.90

TOTAL MARKER = 183.05  TOTAL LENGTH = 594.96

*********************************************************
THE # OF FINAL SECTIONS ARE : 7
SECTION 0 HAS PLY = 1  
AND 1 SIZE size-30
AND 3 SIZE size-34
AND 1 SIZE size-38
AND 1 SIZE size-40
MARKER LENGTH = 72.52  TOTAL LENGTH = 72.52

SECTION 1 HAS PLY = 1  
AND 3 SIZE size-32
AND 2 SIZE size-34
AND 1 SIZE size-36
MARKER LENGTH = 71.03  TOTAL LENGTH = 71.03

SECTION 2 HAS PLY = 1  
AND 4 SIZE size-34
AND 1 SIZE size-36
AND 1 SIZE size-38
MARKER LENGTH = 72.52  TOTAL LENGTH = 72.52

SECTION 3 HAS PLY = 1  
AND 1 SIZE size-32
AND 3 SIZE size-34
AND 2 SIZE size-38
MARKER LENGTH = 72.52  TOTAL LENGTH = 72.52

SECTION 4 HAS PLY = 1  
AND 3 SIZE size-30
AND 2 SIZE size-32
AND 1 SIZE size-38
MARKER LENGTH = 70.52 TOTAL LENGTH = 70.52

SECTION 5 HAS PLY = 1
AND 1 SIZE size-32
AND 5 SIZE size-34
MARKER LENGTH = 71.63 TOTAL LENGTH = 71.63

SECTION 6 HAS PLY = 2
AND 1 SIZE size-30
AND 1 SIZE size-32
AND 4 SIZE size-34
MARKER LENGTH = 71.19 TOTAL LENGTH = 142.38

TOTAL MARKER = 501.93 TOTAL LENGTH = 573.12

UNIT OVER/UNDER = 0
TOTAL TIME = 0.020000
Appendix C  Algorithm Detailed Descriptions
"Savings" Algorithm for COP

INPUT: (1) An order to be cut, consisting of the various sizes required and a quantity desired of each of these sizes. (2) The number of units over or under the demand that will be allowed. (3) The parameter k which determines the number of iterations after which the savings list will be updated. (4) The ply height of each of the initial sections. (5) List of l's (these are the fabric lengths required for cutting a size combination i - like small and large together - in a particular section). (6) Maximum ply height allowed. (7) Maximum number of sizes allowed per section. (8) The cutting cost per inch of fabric. (9) The unit cost of the fabric.

STEPS:

1. Assign each unit in the order to a separate section of the initial ply height.

2. Compute a savings* achieved for combining any pair of sections into a single section. The maximum size of this list can be set to a specific value. It is best to keep it less than or equal to the input K. The savings list is sorted as each value is calculated and placed in the list.

3. Start at the top of the savings list and feasibly** merge sections according the best savings. The first two sections that are merged are placed in a temporary section. Each merge thereafter is made only with this temporary section until the number of sizes per section is reached.

4. Once a the temporary section is full it is saved and cannot be used again.

5. After k mergers in step 3 the savings list should be updated and resorted by performing steps 2 and 3 for all newly created actions, then performing step 3. (note: k will be an input parameter)

6. Continue until no more savings can be achieved (i.e. the savings list has been scanned and the list is exhausted, with no mergers possible).

OUTPUT: (1) The number of sections, the sizes assigned to each of those sections, and the ply height of each section. (2) The total estimated fabric length required to cut the order. (3) The deviation of the number of units to be cut from the actual number of units required in the order.
Savings Computations:

Step 2 of the algorithm requires a computation of savings achieved for combining two sections into one. Described below are the details of this computation, based on whether or not the two sections to be combined contain the same sizes or not.

Case A:

The two sections contain exactly the same size(s). The merger can be accomplished in one of two ways:

(i) Increase ply height by spreading one section on top of the other and making no change to the size combination in the section.

To compute the savings achieved in this situation, the cost savings is essentially based only on the cutting cost. That is, we need a number to reflect the savings of cutting the size combination in this section once instead of twice. (Note the length of fabric required for the section is the same before and after the merger and hence has no effect on the cost savings for the merger).

Let \( e \) represent the number of cutting inches in the pattern for the size combination in the two sections being considered. Then \( e \) is also the number of cutting inches required for the merged section as well. Recall that \( U = \frac{\text{cutting cost}}{\text{inch}} \).

Thus, \( Ue + Ue = \) cost of cutting the two unmerged sections, and \( Ue = \) cost of cutting the merged sections. Hence, \( Ue = \) SAVINGS in cost obtained by merging the two sections. (Illustration attached).

However, the merger for case A could also be accomplished by

(ii) changing the size combination, leaving the ply height the same.

For example, suppose the two sections both contain sizes 32 and 34. The merged section will then contain the size combination 2-32s and 2-34s. Here the savings will be the decreased cost of fabric required for spreading the merged sections. Assume the following notation:

\[
\begin{align*}
  l_{31} &= \text{length of fabric required to cut one layer of the 1st unmerged section}, \\
  l_{32} &= \text{length of fabric required to cut one layer of the 2nd unmerged section}, \\
  l_{33} &= \text{length of fabric required to cut one layer of the 3rd MERGED section}, \\
  p &= \text{ply height of the unmerged and merged sections}
\end{align*}
\]
Recall that $c$ is the unit cost of fabric

Then, the savings can be computed as $cp(l_{11} + l_{12} - l_{13})$. (Illustration attached).

Thus, for case A, the savings is the $\max\{Uc, cp(l_{11} + l_{12} - l_{13})\}$.

If the ply heights of the two section are not equal and the second method of merging the two sections is better the following takes place:

Case B:

The two sections do not contain exactly the same size(s), but are of the same ply height. To maintain consistency, the only possible way to merge two such sections is to merge the size combination, leaving the ply height unchanged. This is precisely the same as case A(ii). Hence the savings computation is

$$cp(l_{11} + l_{12} - l_{13})$$. (Illustration attached).

Case C:

The two sections do not contain the same size and have different ply heights. The only way to merge two such sections is to merge the size combination. This is the same as case B. Hence the savings computation is

$$cp(l_{11} + l_{12} - l_{13})$$. (Illustration attached).

The ply height of the section being merged is chosen so that the minimum number of overages or underages are created.

**Feasibility Checks:**

Step 4 of the algorithm states that section mergers should be done only when feasible. The feasibility of such mergers are based on two conditions:

(1) Will the maximum number of sizes allowed per section be violated? If so, do not merge.

(2) Will the maximum ply height be violated? If so, do not merge.
Cherry Picking Algorithm for COP

INPUT: (1) An order to be cut, consisting of the various sizes required and a specified demand quantity for each of these sizes. (2) The number of units over or under the demand that will be allowed. (3) Maximum ply height allowed. (4) Maximum number of sizes allowed per section. (5) List of li's (these are the fabric lengths required for cutting a size combination i - like small and large together - in a particular section).

STEPS:

1. Let q1 be the largest quantity of any size remaining in the order, and q2 be the second largest, where q2 < q1.
   
   (If there is no such q2, then one of two cases exists. Case 1: Only one size remains in the order, or Case 2: All sizes remaining have the same order quantity. In either case, set q2 = q1.)

   Form set S by selecting all sizes remaining in the order which have a quantity greater than or equal to q2 minus the number of units allowed over the specified demand.

2. The next section created will have ply height = min{q2, max ply height}.
   Combine the sizes in set S in this section in a way so that a minimal amount of fabric will be required, based on the inputs li. For example, if set S contains sizes small and large, it may be necessary to create two sections, one containing size small and the other size large, or only one section may be required which contains both sizes small and large. In the general case, all combinations of the sizes in set S should be considered which do not exceed the maximum number of sizes allowed per section.

3. Reduce the order demand quantities for the sizes in set S by q2.

4. If the order contains a size with positive quantity larger than the number of units allowed under the specified demand, go to step 1.

OUTPUT: (1) The number of sections, the sizes assigned to each of those sections, and the ply height of each section. (2) The total estimated fabric length required to cut the order. (3) The deviation of the number of units to be cut from the actual number of units required in the order.
Improvement Algorithm for COP

INPUT: (1) An order to be cut, consisting of the various sizes required and a quantity desired of each of these sizes. (2) The number of units over or under the demand that will be allowed. (3) A solution to the problem (see below for details) to be improved upon. (4) List of $l_i$'s (these are the fabric lengths required for cutting a size combination $i$ - like small and large together - in a particular section). (5) Maximum ply height allowed. (6) Maximum number of sizes allowed per section. (7) The cutting cost of the fabric per inch. (8) The unit cost of the fabric.

A SOLUTION consists of the following: (1) The number of sections, the sizes assigned to each of those sections, and the ply height of each section. (2) The total estimated fabric length required to cut the order. (3) The deviation of the number of units to be cut from the actual number of units required in the order.

OUTPUT: The output from the improvement algorithm will consist of a solution (as described above).

Step 0: We need to keep track of starting over. If we start over and cannot find any improvements after examining all possible exchanges, then the algorithm will terminate.

Each section contains one or more sizes. A portion of a section will consist of only one size. For example, if a section contains sizes M, M and L, the portions to consider are M, L, and MM.

STEPS:

Step 1. Consider the next portion of one section.

Step 2. Attempt to reassign the portion from its original section to one or more of the remaining sections so that the reassignment satisfies the feasibility checks listed below. If feasible to reassign, compute the savings that would be achieved by making the reassignment.

Step 3 Attempt to swap the portion from its original section with a portion from one of the remaining sections so that the reassignment satisfies the feasibility checks listed below. If feasible compute the savings that would be achieved by making the reassignment.
Step 4 Perform the reassignment based on the best savings computed.

How to perform the merger of the portion with a section and how to compute the associated savings can be described exactly as per the Savings Algorithm:

**Case A:**

The portion and section contain exactly the same size(s). The merger can be accomplished in one of two ways:

(i) Increase ply height by spreading one section on top of the other and making no change to the size combination in the section.

To compute the savings achieved in this situation, the cost savings is essentially based only on the cutting cost. That is, we need a number to reflect the savings of cutting the size combination in this section once instead of twice. (Note the length of fabric required for the section is the same before and after the merger and hence has no effect on the cost savings for the merger).

Let $e_i$ represent the number of cutting inches in the pattern for the size combination in the two sections being considered. Then $e_i$ is also the number of cutting inches required for the merged section as well. Recall that $U = \text{cutting cost/inch}$. Thus, $Ue_i + Ue_i = \text{cost of cutting the two unmerged sections, and}$

$Ue_i = \text{cost of cutting the merged sections. Hence,}$

$Ue_i = \text{SAVINGS in cost obtained by merging the two sections.}$

However, the merger for case A could also be accomplished by

(ii) changing the size combination, leaving the ply height the same.

For example, suppose the two sections both contain sizes 32 and 34. The merged section will then contain the size combination 2-32s and 2-34s. Here the savings will be the decreased cost of fabric required for spreading the merged sections. Assume the following notation:

$l_{11} = \text{length of fabric required to cut one layer of the original section from which the portion will be cut (section A),}$

$l_{12} = \text{length of fabric required to cut one layer of the candidate section into which the portion will be added (section B),}$

$l_{13} = \text{length of fabric required to cut one layer of section A after the reassignment of the portion, and}$
\[ l_{4} = \text{length of fabric required to cut one layer of section B after the reassignment of the portion.} \]

\[ p = \text{ply height of the unmerged and merged sections} \]

Recall that \( c \) is the unit cost of fabric

Then, the savings can be computed as \( cp(l_{11} + l_{12} - l_{13} - l_{14}) \).

Thus, for case A, the savings is the max\{\( Ue_{i} \), \( cp(l_{11} + l_{12} - l_{13} - l_{14}) \}\}.

**Case B:**

The portion and section do not contain exactly the same size(s).

(i) Same ply height.

To maintain consistency, the only possible way to merge two such sections is to merge the size combination, leaving the ply height unchanged.

(ii) Ply heights not the same.

The merger should take place by combining the size combinations, and choosing the ply height so that the minimum number of overages or underages are created and all other feasibility checks are satisfied.

In either case (i) or (ii), we have the same situation as case A(ii). Hence the savings computation is

\[ cp(l_{11} + l_{12} - l_{13} - l_{14}). \]

**Feasibility Checks:**

The feasibility of such mergers are based on two conditions:

(1) Will the maximum number of sizes allowed per section be violated? If so, do not merge.

(2) Will the maximum number of units over and under the demand be violated? If so, do not merge.
Appendix D  Savings Algorithm Source Code
/* - $Header: D:/cops/src/savings/case_ai.c January 1991 - */

#include <stdio.h>
#include <stdlib.h>
#include "savedec.h"
#include "savelcl.h"

float case_ai(sect1, sect2, cut_cost)

{ 
int i;
int e = 0;
float savings;

for (i=0; i < num_of_sizes; i++) {
  e = e + (order.perimeter[i] * sect1.sizes[i]);
  e = e + (order.perimeter[i] * sect2.sizes[i]);
}

savings = (float) cut_cost * e;

return(savings);
}
/* --------------------------------------------------------------*/

FILE NAME : case.aii.c
PROGRAMMER : Terri A. Smith
DATE WRITTEN : January 1991
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: To compute the savings if the units or ply height is not the same in two sections.

#include <stdio.h>
#include <stdlib.h>
#include "savedec.h"
#include "savelcl.h"

float case_aii(sect1, sect2, unit_cost)

    sect1_t sect1;
    sect2_t sect2;
    int unit_cost;

    int i;
    int e = 0;
    float savings;
    float sect1_inch;
    float sect2_inch;
    float merge_inch;
    order_t merged_order;

    sect1_inch = find_inches(sect1.sizes);
    sect2_inch = find_inches(sect2.sizes);

    for (i=0; i<num_of_sizes; i++) {
        merged_order[i] = 0;
        merged_order[i] = merged_order[i] + sect1.sizes[i];
        merged_order[i] = merged_order[i] + sect2.sizes[i];
    }

    merge_inch = find_inches(merged_order);

    savings = unit_cost * sect1.ply_height * (sect1_inch + sect2_inch - merge_inch);

    return(savings);

}
float compute_savings(sect1, sect2, cut_cost, unit_cost, temp_save, max_sizes, max_ply)
{
    int i;
    int e = 0;
    float savings = (float) 0.0;
    float save2 = (float) 0.0;
    char match = 1;
    int num_units = 0;
    int j, k, count;
    char match2;

    temp_save->ply1 = sect1.ply_height;
    temp_save->ply2 = sect2.ply_height;

    for (i = 0; i < num_of_sizes; i++) {
        if (sect1.sizes[i] == sect2.sizes[i])
            match = 0;
        num_units = num_units + sect1.sizes[i];
        num_units = num_units + sect2.sizes[i];
    }

    if (match) { /* sizes in sections are the same */
        if ((sect1.ply_height + sect2.ply_height) <= max_ply) {
            savings = case_all(sect1, sect2, cut_cost);
            temp_save->type = 1;
        }
    }
}
temp_save->ply_height = sect1.ply_height + sect2.ply_height;

else if (num_units <= max_sizes) {
    save2 = case_all(sect1, sect2, unit_cost);
    /* if (save2 > savings) || (temp_save->ply_height > max_ply) */
    temp_save->type= 2;
    savings = save2;
    temp_save->ply_height = sect1.ply_height;
    /* */
}

else if (sect1.ply_height == sect2.ply_height) && (num_units <= max_sizes) {
    savings = case_all(sect1, sect2, unit_cost);
    temp_save->type= 3;
    temp_save->ply_height = sect1.ply_height;
}

else if (num_units <= max_sizes) {
    temp_save->ply_height = sect1.ply_height;
    savings = case_all(sect1, sect2, unit_cost);
    temp_save->type= 4;
}

temp_save->savings = savings;
return(savings);
}
/* -------------------------------------------------------------
-- $Header:: D:/cops/src/savings/findinch.c January 1991
-- ***********************************************************/

FILE NAME : Findinch.c
PROGRAMMER : Terri A. Smith
DATE WRITTEN : January 1991
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE - To find the length (in inches) in the list of Is

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "savedec.h"
#include "savelcl.h"

float find_inches(sizes)

{ order_t sizes;

    int i, j;
    char match = 0;

    i = 0;
    while (((match) && (i < num_list)) ( 
        match = 1;
        for (j=0; j<num_of_sizes; j++) ( 
            if (sizes[i] != list[i].sizes[j]) 
            match = 0;

        )

    )

    if (match)
        return(list[~i].inches);
    else ( 
        printf(" COULDN'T FIND ");
        for (i=0; i<num_of_sizes; i++) ( 
            if (sizes[i] > 0)
                printf("%d %s, sizes[i], order.ch_sizes[i]);

        )

    printf("\n");

    exit(0);

    )

}


/* **************************************************************************
 * $Header:: D:/cops/src/savings/getparm.c December 1990
 * **************************************************************************/

- FILE NAME : Getparm.c
- PROGRAMMER : Terri A. Smith
- DATE WRITTEN : December 1990
- ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE- To read in the parameters from a file

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <savdec.h>
#include "savcli.h"

int get_parameters(out_units, max_ply, max_sizes,
    k, init_ply, q, cut_cost, unit_cost)

int *out_units;
int *max_ply;
int *max_sizes;
int *k;
int *init_ply;
int *q;
int *cut_cost;
int *unit_cost;

{
    int i, j, m, l;
    FILE *fp = NULL;
    int quantity;
    float temp;
    char match;

    if ((fp = fopen("INPUT", "r")) == NULL) {
        printf("Cannot open input file - getparm.c");
        exit(0);
    }

    /* set order and list values to -1 */
    for (i = 0; i < MAX_SIZES; i++) {
        order.number[i] = 0;
        order.ch_sizes[i][0] = 0;
        order.perimeter[i] = 0;
    }

    for (i=0; i<MAX_LIST; i++) {
        list[i].inches = (float) 0.0;
    }

    for (j = 0; j < MAX_SIZES; j++)
        list[i].sizes[j] = 0;
}
fscanf(fp,"d", ou_units);
fscanf(fp,"d", max_ply);
fscanf(fp,"d", max_sizes);
fscanf(fp,"d", init_ply);
fscanf(fp,"d", k);
fscanf(fp,"d", cut_cost);
fscanf(fp,"d", unit_cost);
fscanf(fp,"d", q);

"/ Input Order */
for (i = 0; i < MAX_SIZES; i++) {
    fscanf(fp,"d", &order.number[i]);
    if (order.number[i] == -1) {
        order.number[i] = 0;
        break;
    }
    fscanf(fp,"d", &order.perimeter[i]);
    fscanf(fp,"d", order.ch_sizes[i]);
}
num_of_sizes = i;

"/ Input List */
i=0;
while(1) {
    fscanf(fp,"d", &quantity);
    if (quantity == -2)
        break;
    while (quantity != -1) {
        fscanf(fp,"d", &m);
        if (m >= num_of_sizes) {
            printf("ERROR in reading size variable - getparm.c".);
            exit(0);
        }
        list[i].sizes[m] = quantity;
        fscanf(fp,"d", &quantity);
    }
    fscanf(fp,"d", &list[i].inches);
    ++i;
}
fclose(fp);
return(1);
/* ---------------------------------------------*/
-- $Header: D:/cops/src/savings/globals.h January 1991
-- ---------------------------------------------*/

FILE NAME : Globals.h
PROGRAMMER : Terri A. Smith
DATE WRITTEN : January 1991
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: To declare all global variables

#include <stdio.h>
#include "savedec.h"
#include "savelcl.h"

ord_var_t order;
list_t *list = NULL;
int num_of_sizes;
int num_list;
int total_order = 0;
int curr_tot = 0;
int num_old_sect = 0;
section_t *old_sect = NULL;
INCLUDES = savedec.h
LIBNAME = savelib

OBJS = \
globals.obj \ngetparm.obj \nfindinch.obj \ncase_ai.obj \ncase_aii.obj \ncompute.obj

.c.obj:
 $(CC)
 $(LIB)

globals.obj : globals.c $(INCLUDES)
getparm.obj : getparm.c $(INCLUDES)
findinch.obj : findinch.c $(INCLUDES)
case_ai.obj : case_ai.c $(INCLUDES)
case_aii.obj : case_aii.c $(INCLUDES)
compute.obj : compute.c $(INCLUDES)
savings.obj : savings.c $(INCLUDES)
savings.exe : savings.obj $(OBJS)
cl savings /link savelib.lib

$(B)\savings.exe : savings.exe
 $(CP)

$(I)\savedec.h : savedec.h
 $(CP)
/* ---------------------------* /
--- $Header: D:/cops/src/savings/savedec.h December 1990
--- -------------------------------------------------------------*/

FILE NAME : Savedec.h
PROGRAMMER : Terri A. Smith
DATE WRITTEN : December 1990
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952
PURPOSE: To define all variables and procedures

--- -------------------------------------------------------------*/

#ifndef SAVEDEC_H
#define SAVEDEC_H

#define MAX_LIST 1000
#define MAX_SIZES 25
#define MAX_SAVINGS 2

typedef int order_t[MAX_SIZES];
typedef char sizes_t[MAX_SIZES][10];
typedef struct {
    order_t number;
    sizes_t ch_sizes;
    int perimeter[MAX_SIZES];
} ord_var_t;
typedef struct {
    order_t sizes;
    float inches;
} list_t;
typedef struct {
    order_t sizes;
    int ply_height;
    char merged;
} section_t;
typedef struct {
    int sect1;
    int sect2;
    int ply_height;
    float savings;
    int type;
    int ply1;
    int ply2;
    } savings_t;

int get_parameters(int *units, int *max_ply, int *max_sizes, int *k,
                   int *init_ply, int *q, int *cut_cost, int *unit_cost);
float find_inches(order_t sizes);
58  float case_el(section_t sect1, section_t sect2, int cut_cost);
59  float case_el(section_t sect1, section_t sect2, int unit_cost);
60  float compute_savings(section_t sect1, section_t sect2, int cut_cost,
61                 int unit_cost, savings_t *temp_save, int max_sizes, int max_ply);
62  
63  #endif

64

65

66
Icopulsrclsavingslsavedec.h
December 1990

/*
 
*FILE NAME : Savelcl.h
*PROGRAMMER : Terri A. Smith
*DATE WRITTEN : December 1990
*ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952
* PURPOSE: To define all global variables
*
*/

ifndef SAVELCL_H
#define SAVELCL_H

extern ord_var_t order;
extern list_t *list;
extern int num_list;
extern int num_of_sizes;
extern int total_order;
extern int curr_tot;
extern int num_old_sect;
extern section_t *old_sect;

#endif
/*-----------------------------------------------*/
-- Header: D:/cops/src/savings/savings.c December 1990
--
/*-----------------------------------------------*/

- FILE NAME: Savings.c
- PROGRAMMER: Terri A. Smith
- DATE WRITTEN: December 1990
- ADDRESS: GTRI/CSITL Atlanta GA 30332 (404) 894-8952

- PURPOSE: Main program which controls execution of other procedures

---

#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include "savedec.h"
#include "savelcl.h"

#define cLock() time(NULL)

main(argv, argc)

int argv;
char *argc[];

(

/* Input Variables */
int ou_units;
/* # of units over/under allowed */
int max_ply;
/* max ply height allowed */
int max_sizes;
/* # of sizes allowed / section */
int init_ply;
/* initial ply height */
int k;
/* # of merges allowed */
int cut_cost;
/* cutting cost / inch */
int unit_cost;
/* unit cost */
int q;
/* ply used for initial sections */

/* Output Variables */
float tot_length;
/* the total amt of fabric needed*/
int unit_dev;
/* deviation of units to cut from order */
char unit_string[10];
/* string for over, under */
float inches;
/* used in output */

int i, j, x, y;
/* counters */
int curr_sect = 0;
/* current section */
section_t new_sect;
/* new sections */
section_t *save_sect = NULL; /* new sections */
div_t n;
/* quotient and remainder */
savings_t temp_save; /* temp savings - 1 structure */
savings_t *save_list = NULL; /* savings list */
int num_savings;
/* # of savings in list */
int m, l, r, s; /* counters */

/*-----------------------------------------------*/
```c
char mergers_possible = 1; /* Boolean for loop */
int num_newsect; /* # of total newsect */
int num_savesect; /* # of total newsect */
int num_units; /* # of units in one section */
int unit_count; /* # of units in all sections */
int order_count; /* # of units in order */
int num_mergers = 0; /* # of mergers */
FILE *fp; /* output file pointer */
clock_t start_time, end_time; /* times */
double total_time; /* total time program runs */
float marker; /* inches in marker */
float tot_marker; /* total inches in all markers */
char match2; /* boolean value */
int count; /* counts the sections */
int old_ply; /* ply height of older section */
section_t temp Sect; /* temporary section */
int add Sect; /* # of sections to add */

int nun new Sect; /* # of total new Sect */

int nun save Sect; /* # of total new Sect */

int num_units; /* # of units in one Sect */

int unit_count; /* # of units in all Sects */

int order_count; /* # of units in order */

int num_mergers = 0; /* # of mergers */

FILE *fp; /* output file pointer */

clock_t start_time, end_time; /* times */
double total_time; /* total time program runs */
float marker; /* inches in marker */
float tot_marker; /* total inches in all markers */
char match2; /* boolean value */
int count; /* counts the sections */
int old_ply; /* ply height of older section */
section_t temp Sect; /* temporary section */
int add Sect; /* # of sections to add */

start_time = clock();

if ((fp = fopen("OUTPUT", "w")) == NULL) {
    printf("CANNOT OPEN OUTPUT FILE savings.c\n");
    exit(0);
}

/* Allocation of input list */
if ((list = (list_t *)malloc(MAX_LIST * sizeof(list_t))) == NULL) {
    printf("ALLOCATION ERROR FOR LIST savings.c\n");
    exit(0);
}

/* Get parameters and print initial stuff to output file */
num_list = get parameters(&ou_units, &max_ply, &max_sizes, &k, &init_ply,
&q, &cut_cost, &unit_cost);

fprintf(fp, "SAVINGS ALGORITHM 2\n");
fprintf(fp, "MAX PLY = %d MAX # OF UNITS PER SECTION = %d\n", max_ply, max_sizes);
/* fprintf(fp, "UNIT COST = %d cents\n", unit_cost, cut_cost); */

fprintf(fp, "K = %d INIT PLY = %d 0 = %d\n", k, init_ply, q);

fprintf(fp, "ORDER\n");
for (i=0; i<num_of_sizes; i++) {
    fprintf(fp, "%d SIZE %d\n", order.number[i], order.ch_sizes[i]);
}

/* Allocate space for two sets of sections */
for (i=0; i<num_of_sizes; i++)
    total_order = total_order + order.number[i];
```
if ((old_sect = (section_t *)malloc(total_order * sizeof(section_t))) == NULL) {
    printf("ALLOCATION ERROR FOR OLD SECTION savings.c\n");
    exit(0);
}

if ((save_sect = (section_t *)malloc(total_order * sizeof(section_t))) == NULL) {
    printf("ALLOCATION ERROR FOR SAVE SECTION savings.c\n");
    exit(0);
}

for (i=0; i<num_of_sizes; i++) {
    temp_order[i] = order.number[i];
}

/*
Assign each unit in order to a separate section of initial ply height
*/
for (i = 0; i < total_order; i++) {
    old_sect[i].ply_height = q;
    old_sect[i].merged = 0;
    for (j=0; j< MAX_SIZES; j++)
        old_sect[i].sizes[j] = 0;
    unit_count = 0;
    for (i=0; i<num_of_sizes; i++) {
        n = div(order.number[i], q);
        for (j=0; j<n.quot; j++) {
            old_sect[curr_sect].sizes[i] = 1;
            ++curr_sect;
            unit_count = unit_count + q;
        }
    }
    for (i=0; i<num_of_sizes; i++) {
        n = div(order.number[i], q);
        for (j=0; j<n.rem; j++) {
            unit_dev = total_order - unit_count;
            if ((ou_unites - unit_dev) < 0) {
                old_sect[curr_sect].sizes[i] = 1;
                ++curr_sect;
                unit_count = unit_count + q;
            }
        }
    }
}
num_old_sect = curr_sect;

/*
Allocate space for savings list and initialize
*/
if ((save_list = (savings_t *)malloc(MAX_SAVINGS * sizeof(savings_t))) == NULL) {
    printf("ALLOCATION ERROR FOR SAVINGS LIST savings.c\n");
    exit(0);
for (i=0; i<MAX_SAVINGS; i++) {
    save_list[i].savings = (float) 0.0;
    save_list[i].ply_height = 0;
    save_list[i].type = 0;
}

/*
 * Main loop in the Savings algorithm:
 * - creates a savings list and merges sections one at a time.
 * - a temporary section is created and merged with initial sections
 *   until it is completely filled. It is then saved in the
 *   save section and a new temporary section is started
 * - when all sections are saved to the save section then program is
 *   terminated
 */

num_save_sect = 0;
while (mergers_possible) {
    for (i=0; i<MAX_SAVINGS; i++) {
        save_list[i].savings = (float) 0.0;
        save_list[i].ply_height = 0;
        save_list[i].type = 0;
    }

    printf("NUM OLD SECT = %d\n", num_old_sect);
    if (num_old_sect <= 1) break;

    num_units = 0;

    /*
     * When the max number of units per section is reached, the section
     * is saved in save section
     */
    for (j=0; j<num_of_sizes; j++)
        num_units = old_sect[0].sizes[j] + num_units;

    if (num_units >= max_sizes) {
        memcpy(&save_sect[num_save_sect], &old_sect[0], sizeof(section_t));
        for (i = 0; i<num_old_sect-1; i++)
            memcpy(&old_sect[i], &old_sect[i+1], sizeof(section_t));
        ++num_save_sect;
        --num_old_sect;
    }
    mergers_possible = 0;
    num_savings = 0;
    }/*
Create Savings List

```c
Create Savings List
"/

i = 0;
for (j=i+1; j<num_old_sect; j++) {
    temp_save.sectl = i;
    temp_save.sect2 = j;
    temp_save.ply_height = 0;
    temp_save.savings = (float) 0.0;
    temp_save.type = 0;

    compute_savings(old_sect[i], old_sect[j], cut_cost, unit_cost,
                    &temp_save, max_sizes, max_ply);

    m = 0;
    while((m < num_savings) &&
          (temp_save.savings <= save_list[m].savings))
        ++m;

    if (m != MAX_SAVINGS) {
        for (l = num_savings; l > m; l--) {
            memcpy(&save_list[l], &save_list[l-1], sizeof(savings_t));
        }

        memcpy(&save_list[l], &temp_save, sizeof(savings_t));
        if (num_savings < MAX_SAVINGS-1)
            ++num_savings;
    }
}

/*
   Merge Sections
 */

new_sect.ply_height = q;
new_sect.merged = 0;
for (j=0; j< MAX_SIZES; j++)
    new_sect.sizes[j] = 0;
num_mergers = 0;
m = 0;
for (i=0; i<num_savings; i++) {
    r = save_list[i].sect1;
    s = save_list[i].sect2;
    num_units = 0;

    for (j=0; j<max_sizes; j++) {
        num_units = old_sect[r].sizes[j] + num_units;
        if (save_list[i].type != 1)
            num_units = old_sect[s].sizes[j] + num_units;
    }

    if ((save_list[i].ply_height <= max_ply) &&
        (old_sect[r].merged) &&
        (old_sect[s].merged))}
(num_units <= max_sizes) &&
(save_list[i].type != 0)) {

merger_possible = 1;
old_sector[r].merged = 1;
old_sector[s].merged = 1;

new_sector.ply_height = save_list[i].ply_height;
for (j=0; j<num_of_sizes; j++) {
   new_sector.sizes[j] = new_sector.sizes[j] +
   old_sector[r].sizes[j];
   if (save_list[i].type != 1)
      new_sector.sizes[j] = new_sector.sizes[j] +
      old_sector[s].sizes[j];
}

/*
 * If the savings is achieved by rearranging sizes
 * in one section (not by putting plies on top of
 * each other), then the two ply heights of the sections
 * must be manipulated to keep the order correct.
 * e.g. If one section has ply 3 and the other ply 10
 * one section of ply 3 with both the sizes combinations
 * is made and 7 sections of kept in the list of merging
 * sections
 * */
if (save_list[i].type != 1) {

   for (x=0; x<num_of_sizes; x++)
      hold_sizes[x] = old_sector[x].sizes[x];

   /*
    * Count how many sections in the sections list match
    * the given section to merge
    */
   count = 0;
   for (l=1; l<num_old_sector; l++) {
      match2 = 1;
      for (j=0; j<num_of_sizes; j++) {
         if (old_sector[l].sizes[j] != old_sector[l].sizes[j])
            match2 = 0;
      }

      if (match2)
         ++count;
   }

   /*
    * If the count is greater than the ply height of
    * the temporary section, combine the two sections
    * with the ply height of temporary section and then
    * delete that number (ply height) of sections from
    * the section list
    */
   if ((save_list[i].plyl / q) <= count) {
      count = save_list[i].plyl / q;
      for (l=1; l<num_old_sector; l++) {

match2 = 1;
for (j=0; j<num_of_sizes; j++) {
    if (hold_sizes[j] != old_sect[i].sizes[j])
        match2 = 0;
}

if ((match2) && (count > 0)) {
    for (m=1; m<num_oldsect-1; m++)
        memcpy(&oldsect[m], &oldsect[m+1], sizeof(section_t));
    --num_oldsect;
    --count;
    --l;
}

} /* save_list[i].ply1 <= count */
/
/*
else if the count is less than the ply height
of the temporary section, then the temp section
will have a ply height of count and sections are
added back to the section list based on the the
old_ply (of temp section) minus the the count
*/
else {
    if (count > 0) {
        if (oldsect[0].ply_height > count) {
            old_ply = oldsect[0].ply_height;
            oldsect[0].ply_height = count;
            newsect.ply_height = count;
            for (i=0; i<num_of_sizes; i++) {
                if (oldsect[0].sizes[i] > 0) {
                    addsect = old_ply - count;
                    tempsect.ply_height = q;
                    tempsect.merged = 0;
                    for (j=0; j<num_of_sizes; j++)
                        tempsect.sizes[j] = 0;
                    tempsect.sizes[i] = 1;
                    for (l=0; l<oldsect[0].sizes[l]; l++) {
                        for (j=0; j<addsect; j++)
                            memcpy(&oldsect[num_oldsect+l], &tempsect, sizeof(section_t));
                    }
                } /* if oldsect[0].sizes[i] > 0 */
            } /* for i=0 etc */
        } /* oldsect[0].ply > 0 */
    }

    for (l=1; l<num_oldsect; l++) {
        match2 = 1;
        for (j=0; j<num_of_sizes; j++) {
            if (hold_sizes[j] != oldsect[l].sizes[j])
                match2 = 0;
        }
    }

    if ((match2) && (count > 0)) {
for (m=1; m<num_old_sector-1; m++)
{
    memcpy(&old_sector[m], &old_sector[m+1], sizeof(section_t));
    --num_old_sector;
    --count;
    --I;
}
} /* for l=1 etc */

} /* count > 0 */
} /* else */
} /* if type l= 1 */

++m;
if (++num_mergers >= k)
    break;
}

memcpy(&old_sector[0], &new_sector, sizeof(section_t));

*/

Merges Complete
*/

if (save_list[i].type == 1) {
    for (i=s; i<num_old_sector-1; i++)
        memcpy(&old_sector[i], &old_sector[i+1], sizeof(section_t));
    --num_old_sector;
}

num_savings = 0;

} /* End of While (1) */

if (num_old_sector > 0) {
    for (i=0; i<num_old_sector; i++)
        memcpy(&save_sector[num_save_sector++], &old_sector[i], sizeof(section_t));
}

*/

put final information in output file
*/
end_time = clock();
total_time = ((double) end_time - start_time) / CLK_TCK;

fprintf(fp, "\n\n"%*s\n\n\n");
tot_length = (float) 0.0;
unit_dev = 0;
order_count = 0;
unit_count = 0;
tot_marker = (float) 0.0;

fprintf(fp, "THE # OF FINAL SECTIONS ARE : %d\n", num_save_sector);
for (i=0; i<num_save_sect; i++) {
    fprintf(fp, "SECTION %d HAS PLY = %d\n", i, save_sect[i].ply_height);
    for (j=0; j<num_of_sizes; j++) {
        if (save_sect[i].sizes[j] > 0) {
            fprintf(fp, "AND %d SIZE %s\n", save_sect[i].sizes[j], order.ch_sizes[j]);
            unit_count = unit_count + (save_sect[i].sizes[j] * save_sect[i].ply_height);
        }
        marker = find_inches(save_sect[i].sizes);
        inches = marker * save_sect[i].ply_height;
        fprintf(fp, "MARKER LENGTH = %7.2f THE TOTAL LENGTH = %7.2f\n\n", marker, inches);
        tot_length = tot_length + inches;
        tot_marker = tot_marker + marker;
    }
    for (j=0; j<num_of_sizes; j++)
        order_count = order_count + order.number[j];
    unit_dev = order_count - unit_count;
    if (unit_dev > 0)
        strcpy(unit_string, "UNDER");
    else if (unit_dev == 0)
        strcpy(unit_string, "0");
    else {
        unit_dev = unit_dev - 1;
        strcpy(unit_string, "OVER");
    }
    fprintf(fp, "TOT MARKER = %7.2f TOT LENGTH = %7.2f, UNIT OVER/UNDER = %d %s\n\n",
            marker, tot_length, unit_dev, unit_string);
    fprintf(fp, "TOTAL TIME = %f SECONDS\n", total_time);
    /* Free all space and close output file */
    if (list != NULL) free(list);
    if (save_list != NULL) free(save_list);
    if (old_sect != NULL) free(old_sect);
    fclose(fp);
    return(0);
}
Appendix E  Cherry Algorithm Source Code
/* ----------------------------------------------- */
-- SHeader: D:/cops/src/cherry/cherdec.h December 1990
-- ----------------------------------------------- */

/*-----------------------------------------------*/
- FILE NAME : Cherdec.h
- PROGRAMMER : Terri A. Smith
- DATE WRITTEN : December 1990
- ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952
- PURPOSE: To define all variables and procedures

-- ----------------------------------------------- */
#ifndef CHERDEC
#define CHERDEC_H
#define MAX_LIST 1000
#define MAX_SIZES 25
typedef int order_t[MAX_SIZES];
typedef char sizes_t[MAX_SIZES][10];
typedef struct {
    order_t number;
    sizes_t ch_sizes;
} ord_var_t;
typedef struct {
    order_t sizes;
    float inches;
} list_t;
typedef struct {
    order_t sizes;
    int ply_height;
} section_t;
int get_parameters(int *units, int *max_ply, int *max_sizes);
float find_inches(order_t sizes);
float combine_inches(order_t set_s);
void check_inches(section_t *temp_secs, int *num_temp_secs);
void clear_temp(section_t *temp_secs, int *num_temp_secs);
void copy_hold_to_sections();
void ones(order_t set_s, section_t *temp_secs, int *num_temp_secs);
void twos(order_t set_s, section_t *temp_secs, int *num_temp_secs);
void threes(order_t set_s, section_t *temp_secs, int *num_temp_secs);
58 void fours(order_t set_s, section_t *temp_secs, int *num_temp_secs);
59 void fives(order_t set_s, section_t *temp_secs, int *num_temp_secs);
60 void sixes(order_t set_s, section_t *temp_secs, int *num_temp_secs);
61 #endif
FILE NAME : SaveLcl.h
PROGRAMMER : Terri A. Smith
DATE WRITTEN : December 1990
ADDRESS : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
PURPOSE - To define all global variables

 ifndef CHERCL_H
  define CHERCL_H
endif

extern ord_var_t order;
extern list_t *list;
extern int num_list;
extern int num_of_sizes;
extern order_t temp_order;
extern int num_sections;
extern float total_inches;
extern float prev_inch;
extern section_t *sections;
extern int num_hold_secs;
extern section_t *hold_secs;
extern int ply_height;

#endif
/* $Header: D:\cops\src\cherry\cherry.c December 1990 */

FILE NAME: Cherry.c
PROGRAMMER: Terri A. Smith
DATE WRITTEN: December 1990
ADDRESS: GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: The main program which executes all other procedures

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <time.h>
#include "cherdec.h"
#include "chertcl.h"

#define clock() time(NULL)

main(argc, argv)
int argv;
char *argc[];

{ /* Input Variables */
  int ou_units; /* # of units over/under allowed */
  int max_ply; /* max ply height */
  int max_sizes; /* # of sizes allowed / section */

  /* Output Variables */
  int num_temp_secs; /* # of total sections */
  float tot_length; /* total length of fabric */
  float tot_marker; /* total length of fabric */
  section_t *temp_secs=NULL; /* each section description */
  int unit_dev = 0; /* deviation of # of units from order */
  char unit_string[10]; /* string to print over, under */

  int q1; /* largest quantity in order */
  int q2; /* 2nd largest quantity in order */
  int s_var; /* q2 minus ou_units */
  order_t set_s; /* set S of sizes */
  int max_sections=0; /* max # of sections to allocate */
  char repeat_loop; /* boolean to loop again or not */
  int i, j, k, l, m, n; /* counters */
  float inches; /* used for printing results */
  float marker; /* used for printing results */
  int sets_cnt; /* # of sizes in set S */
  FILE *fp; /* output file pointer */
  int unit_count = 0; /* # of units in all sections */
  int order_count = 0; /* # of units in order */
  int ou_count = 0; /* count to determine if repeat loop */
clock_t start_time, end_time;
double total_time;

start_time = clock();

/*
* Open output file*/
if ((fp = fopen("OUTPUT", "w")) == NULL) {
    printf("CANNOT OPEN OUTPUT FILE cherry.c\n");
    exit(0);}

/* Allocate space for the list of Is*/
if ((list = (list_t *)malloc(MAX_LIST * sizeof(list_t))) == NULL) {
    printf("ALLOCATE ERROR FOR LIST cherry.c\n");
    exit(0);}

num_list = get_parameters(&ou_units, &max_ply, &max_sizes);

fprintf(fp, "CHERRY ALGORITHM\n");
fprintf(fp, "MAX PLY = %d MAX # OF UNITS PER SECTION = %d\n", max_ply, max_sizes);
for (i=0; i<num_of_sizes; i++) {
    fprintf(fp, "%d SIZE %s\n", order.number[i], order.ch_sizes[i]);
    order_count = order_count + order.number[i];}

/* Allocate space for the max number of sections
* for the three list of sections*/
for (i=0; i<MAX_SIZES; i++) {
    max_sections = max Sections + order.number[i];
}

if ((sections = (section_t *)malloc(max_sections * sizeof(section_t))) == NULL) {
    printf("ALLOCATE ERROR FOR SECTIONS cherry.c\n");
    exit(0);}

if ((temp_secs = (section_t *)malloc(max_sections * sizeof(section_t))) == NULL) {
    printf("ALLOCATE ERROR FOR SECTIONS cherry.c\n");
    exit(0);}

if ((hold_secs = (section_t *)malloc(max_sections * sizeof(section_t))) == NULL) {
    printf("ALLOCATE ERROR FOR SECTIONS cherry.c\n");
    exit(0);}

for (i=0; i<max_sections; i++) {
    sections[i].ply_height = 0;
for (j=0; j<MAX_SIZES; j++)
    sections[i].sizes[j] = 0;

num_sections = 0;
/*
 * Main Loop of program
 */
while (1) {
    for (i=0; i<num_sections; i++) {
        temp_secs[i].ply_height = 0;
        for (j=0; j<MAX_SIZES; j++)
            temp_secs[i].sizes[j] = 0;
    }
    repeat_loop = 0;
    /*
     Choose Q1 and Q2
    */
    q1 = 0;
    q2 = 0;
    for (i=0; i<num_of_sizes; i++) {
        if (order.number[i] > order.number[q1])
            q1 = i;
    }
    q2 = 0;
    for (i=0; i<num_of_sizes; i++) {
        if (i == q1)
            if (order.number[i] >= 0) {
                q2 = i;
                break;
            }
    }
    for (i=0; i<num_of_sizes; i++) {
        if (i == q1)
            if (order.number[i] >= order.number[q2])
                q2 = i;
    }
    if (order.number[q2] <= 0)
        q2 = q1;
    /*
     Form set S with all the sizes remaining in the order
     which have a quantity greater than or equal to q2 - the number
     of units allowed over the specified demand
    */
    s_var = order.number[q2] - ou_units;
sets_cnt = 0;
for (i=0; i<MAX_SIZES; i++) {
    if ((order.number[i] >= s_var) && (order.number > 0)) {
        sets[i] = 1;
        ++sets_cnt;
    }
    else
        sets[i] = 0;
}

/*
 * Set ply height of next section to the min(q2, max ply)
 */
ply_height = order.number[q2];
if (max_ply < order.number[q2])
    ply_height = max_ply;

/*
 * Combine all possibilities of sections up to 5 units
 * per section
 */
inches = (float) 9999.0;
for (i=0; i<MAX_SIZES; i++)
    temp_order[i] = 0;
num_temp_secs = 0;
total_inches = (float) 0.0;
ones(set_s, temp_secs, &num_temp_secs);
check_inches(temp_secs, &num_temp_secs);
clear_temp(temp_secs, &num_temp_secs);
if ((sets_cnt > 1) && (max_sizes > 1)) {
    twos(set_s, temp_secs, &num_temp_secs);
    check_inches(temp_secs, &num_temp_secs);
    clear_temp(temp_secs, &num_temp_secs);
}

if ((sets_cnt > 2) && (max_sizes > 2)) {
    threes(set_s, temp_secs, &num_temp_secs);
    check_inches(temp_secs, &num_temp_secs);
    clear_temp(temp_secs, &num_temp_secs);
}

if ((sets_cnt > 3) && (max_sizes > 3)) {
    fours(set_s, temp_secs, &num_temp_secs);
    check_inches(temp_secs, &num_temp_secs);
    clear_temp(temp_secs, &num_temp_secs);
}

if ((sets_cnt > 4) && (max_sizes > 4)) {
    fives(set_s, temp_secs, &num_temp_secs);
    check_inches(temp_secs, &num_temp_secs);
    clear_temp(temp_secs, &num_temp_secs);
if ((sets_cnt > 5) && (max_sizes > 5)) {
    sixes(set_s, temp_secs, &num_temp_secs);
    check_inches(temp_secs, &num_temp_secs);
    clear_temp(temp_secs, &num_temp_secs);
}

copy_hold_to_sections();

  /*
  Reduce the order demand
  */
  for (rm(hold_sections = num_hold_secs); rm < num_sections; rm++) {
    for (n=0; n< num_of_sizes; n++) {
      if (sections[rm].sizes[n] == 1) {
        order.number[n] = order.number[n] - ply_height;
        set_s[n] = 0;
      }
    }
  }

  /* Repeat loop if the order contains a size w/ positive
     quantity greater than the number of units allowed under the
     specified demand, else break out of loop
  */
  ou_count = 0;
  for (i=0; i<num_of_sizes; i++) {
    ou_count = ou_count + order.number[i];
    if (ou_count > ou_units)
      repeat_loop = 1;
  }

  if (repeat_loop)
    break;

  /* END of While (1) */

end_time = clock();
total_time = ((double) end_time - start_time) / CLK_TCK;

  /*
  Print Out Results
  */
  fprintf(fp, "\n
************************************************************\n", run-sections);
  fprintf(fp, "THE NUMBER OF FINAL SECTIONS is\n", num_sections);

  for (i=0; i<num_sections; i++) {
    marker = find_inches(sections[i].sizes);
    inches = marker * sections[i].ply_height;
    total_inches = total_inches + inches;
    tot_marker = tot_marker + marker;
    fprintf(fp, "\nSECTION %d HAS PLY = %.1f\n", i, sections[i].ply_height);
    for (j=0; j<num_of_sizes; j++) {
      if (sections[i].sizes[j] > 0) {
        fprintf(fp, " HAS %.1f SIZE %.1f\n", sections[i].sizes[j], order.ch_sizes[j]);
      }
    }
  }
unit_count = unit_count + (sections[i].sizes[j] * sections[i].ply_height);

fprintf(fp, "MARKER INCHES = %7.2f and TOTAL INCHES %7.2f\n", marker, inches);

fprintf(fp, "\nTOTAL MARKER INCHES = %7.2f TOTAL INCHES = %7.2f\n", tot_marker, total_inches);

unit_dev = order_count - unit_count;
if (unit_dev > 0)
    strcpy(unit_string, "UNDER");
else if (unit_dev == 0)
    strcpy(unit_string, "\0");
else {
    unit_dev = unit_dev * -1;
    strcpy(unit_string, "OVER");
}

fprintf(fp, "UNIT OVER/UNDER = %d %s\n", unit_dev, unit_string);

fprintf(fp, "TOTAL_TIME = %s\n", total_time);

if (list != NULL)
    free(list);
if (sections != NULL)
    free(sections);
if (temp_secs != NULL)
    free(temp_secs);
if (hold_secs != NULL)
    free(hold_secs);

fclose(fp);
return(0);
/* $Header:: D:/cpcs/src/cherry/chkinch.c December 1990 */

/*...................................................................
  - FILE NAME : Chkinch.c
  - PROGRAMMER : Terri A. Smith
  - DATE WRITTEN : December 1990
  - ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952
  - PURPOSE- To determine if the total inches calculated from
    the last grouping of sections is less than any previous
    grouping. If so the sections are saved in the hold
    sections.
  -
  -  
  ------------------------------------------------------------*/

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <memory.h>
#include "cherdec.h"
#include "cherlct.h"

void check_inches(temp_secs, num_temp_secs)
section_t *temp_secs;
int *num_temp_secs;
{
  int m, i, j;
  if ((total_inches < prev_inch) && (total_inches > (float) 0.0)) {
    num_hold_secs = 0;
    for (m=0; m<num_temp_secs; m++) {
      memcpy(&hold_secs[num_hold_secs], &temp_secs[m], sizeof(section_t));
      hold_secs[num_hold_secs].ply_height = ply_height;
      ++num_hold_secs;
    }
    prev_inch = total_inches;
  }
}
void clear_temp(temp_secs, num_temp_secs)
    section_t *temp_secs;
    int *num_temp_secs;

    int i, j;
    float total_inches = 0.0;
    for (i=0; i < *num_temp_secs; i++) {
        for (j=0; j < num_of_sizes; j++) {
            temp_secs[i].sizes[j] = 0;
        }
    }
    *num_temp_secs = 0;
/* --------------------------------------------------------------- *
   * $Header:: D:/cops/src/cherry/combine.c   January 1991           *
   * --------------------------------------------------------------- *

   * FILE NAME    : Combine.c
   * PROGRAMMER   : Terri A. Smith
   * DATE WRITTEN : January 1991
   * ADDRESS      : GTRI/CSITL Atlanta GA 30332  (404) 894-8952
   * PURPOSE      : Finds the length (in inches) of the combine units
                   in one section.
   *
   * --------------------------------------------------------------- *

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherrdec.h"
#include "chercl.h"

float combine_inches(temp_order)

    order_t temp_order;

    float inches;

    inches = find_inches(temp_order);

    return(inches);

)
/* ----------------------------- * /
-- $Header:: D:/cops/src/cherry/cphoid.c December 1990
-- ----------------------------- * /

FILE NAME  : Cphotd.c
PROGRAMMER : Terri A. Smith
DATE WRITTEN: December 1990
ADDRESS    : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE- Copies the temp sections into the hold sections.

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <memory.h>
#include "cherdec.h"
#include "chercli.h"

void copy_hold_to_sections()
{
  int m;

  for (m=0; m<num_hold_secs; m++) {
    memcpy(&sections[num_sections], &hold_secs[m], sizeof(section_t));
    sections[num_sections].ply_height = ply_height;
    ++num_sections;
  }

  prev_inch = (float) 9999.0;
}
/* - ---------------------------------------------------------------*/
/* - FILE NAME : FindInch.c
 - PROGRAMMER : Terri A. Smith
 - DATE WRITTEN : January 1991
 - ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952
 - PURPOSE- Finds the current unit grouping in the list of Is.
 - If it is not found program is exited.
 - MODIFICATION HISTORY-
 - ---------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherrdec.h"
#include "chertcI.h"

float find_inches(sizes)
{
    order_t sizes;
    int i, j;
    char match = 0;
    i = 0;
    while (((match) && (i < num_list)) (match = 1;
        for (j = 0; j < num_of_sizes; j++) {
            if (sizes[j] != list[i].sizes[j])
                match = 0;
        }
        ++i;
    }]
    if (match)
        return(list[--i].inches);
    else {
        printf("\nCOULDNT FIND ");
        for (i = 0; i < num_of_sizes; i++)
            for (sizes[i] > 0)
                printf("%d %s ", sizes[i], order.ch_sizes[i]);
        printf("\n");
        exit(0);
    }
}
*/
/*-------------------------------------------------------------------*/
#define _FILE_NAME_ fives.c
#define _PROGRAMMER_ Terri A. Smith
#define _DATE_WRITTEN_ January 1991
#define _ADDRESS_ GTRI/CSITL Atlanta GA 30332 (404) 894-8952
#define _PURPOSE_ Recursive procedure to group units in fives.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "chercl.h"

void fives(set_s, temp_secs, num_temp_secs)

    order_t set_s;
    section_t *temp_secs;
    int *num_temp_secs;

    float inches;
    int j, k, l, m, n;
    order_t temp_order;
    float hold_inches1;
    float hold_inches2;
    int hold_temp_num;

    hold_temp_num = *num_temp_secs;
    hold_inches2 = total_inches;

    for (i=0; i<num_of_sizes; i++) {
        for (j=i+1; j<num_of_sizes; j++) {
            for (k=j+1; k<num_of_sizes; k++) {
                for (l=k+1; l<num_of_sizes; l++) {
                    for (m=l+1; m<num_of_sizes; m++) {
                        for (n=0; n<num_of_sizes; n++)
                            temp_order[n] = 0;

                        if ((set_s[l] == 1) && (set_s[j] == 1) &&
                            (set_s[k] == 1) && (set_s[l] == 1) &&
                            (set_s[m] == 1)) {
                            temp_order[l] = 1;
                            temp_order[j] = 1;
                            temp_order[k] = 1;
                            temp_order[l] = 1;
                            temp_order[m] = 1;
                            inches = combine_inches(temp_order);
                            if (inches != (float) 0.0) {
                                for (n=0; n<num_of_sizes; n++)
                                    temp_secs[*num_temp_secs].sizes[n] = 0;
total_inches = total_inches + inches;
temp_secs[\*num_temp_secs].sizes[j] = 1;
temp_secs[\*num_temp_secs].sizes[k] = 1;
temp_secs[\*num_temp_secs].sizes[m] = 1;
temp_secs[\*num_temp_secs].sizes[n] = 1;
++\*num_temp_secs;
}
temp_order[1] = 0;
temp_order[2] = 0;
temp_order[3] = 0;
temp_order[4] = 0;
for (n=0; n<\num_of_sizes; n++) {
    if ((n != l) && (n != j) && (n != k) &&
        (n != m) && (set_s[n] == 1)) {
        temp_order[n] = 1;
    }
}
hold_inches1 = total_inches;
ones(temp_order, temp_secs, num_temp_secs);
check_inches(temp_secs, num_temp_secs);
for (n=0; n<\num_of_sizes; n++) {
    if ((n != l) && (n != j) && (n != k) &&
        (n != m) && (set_s[n] == 1)) {
        --\*num_temp_secs;
    }
}
total_inches = hold_inches1;
twos(temp_order, temp_secs, num_temp_secs);
total_inches = hold_inches1;
threes(temp_order, temp_secs, num_temp_secs);
total_inches = hold_inches1;
fours(temp_order, temp_secs, num_temp_secs);
total_inches = hold_inches1;
fives(temp_order, temp_secs, num_temp_secs);
**\*num_temp_secs = hold_temp_num;
total_inches = hold_inches2;
/**
* Recursive procedure to group units in fours
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "cherti.h"

void fours(set_s, temp_secs, num_temp_secs)
{
    order_t set_s;
    section_t *temp_secs;
    int *num_temp_secs;

    float inches;
    int j, i, k, l, m;
    order_t temp_order;
    float hold_inches1;
    float hold_inches2;
    int hold_temp_num;
    hold_temp_num = *num_temp_secs;
    hold_inches2 = total_inches;

    for (i=0; i<num_of_sizes; i++) {
        for (j=i+1; j<num_of_sizes; j++) {
            for (k=j+1; k<num_of_sizes; k++) {
                for (l=k+1; l<num_of_sizes; l++) {
                    temp_order[l] = 0;
                    if (((set_s[i] == 1) && (set_s[j] == 1) &&
                        (set_s[k] == 1) && (set_s[l] == 1))
                        temp_order[i] = 1;
                    temp_order[j] = 1;
                    temp_order[k] = 1;
                    temp_order[l] = 1;
                    inches = combine_inches(temp_order);
                    if (inches != (float) 0.0) {
                        for (m=0; m<num_of_sizes; m++)
                            temp_secs[*num_temp_secs].sizes[m] = 0;
                    total_inches = total_inches + inches;
                    temp_secs[*num_temp_secs].sizes[l] = 1;
58    temp_secs[num_temp_secs].sizes[j] = 1;
59    temp_secs[num_temp_secs].sizes[k] = 1;
60    temp_secs[num_temp_secs].sizes[l] = 1;
61    ++num_temp_secs;
62 }
63    temp_order[1] = 0;
64    temp_order[j] = 0;
65    temp_order[k] = 0;
66    temp_order[1] = 0;
67
68    for (m=0; m<num_of_sizes; m++) {
69        if ((m != i) || (m != j) || (m != k) ||
90        (m != l) || (set_s[m] == 1)) {
91            temp_order[m] = 1;
92        }
93    }
94
95    hold_inches1 = total_inches;
96    ones(temp_order, temp_secs, num_temp_secs);
97    check_inches(temp_secs, num_temp_secs);
98
99    for (m=0; m<num_of_sizes; m++) {
100       if ((m != i) || (m != j) || (m != k) ||
101          (m != l) || (set_s[m] == 1)) {
102            --num_temp_secs;
103        }
104    }
105
106    total_inches = hold_inches1;
107    twos(temp_order, temp_secs, num_temp_secs);
108
109    total_inches = hold_inches1;
110    threes(temp_order, temp_secs, num_temp_secs);
111
112    total_inches = hold_inches1;
113    fours(temp_order, temp_secs, num_temp_secs);
114
115    *num_temp_secs = hold_temp_num;
116    total_inches = hold_inches2;
/* -------------------------------
-- $Header: D:\cops\src\cherry\getperm.c  December 1990
-- -------------------------------*/

FILE NAME : Getparm.c
PROGRAMMER : Terri A. Smith
DATE WRITTEN : December 1990
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: To read in the parameters from a file

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "chercli.h"

int getparameters(ou_units, max_pLy, max_sizes)

int *ou_units;
int *max_pLy;
int *max_sizes;

{ int i, j;
FILE *fp = NULL;
int quantity;
int m;
float temp;

if ((fp = fopen("INPUT", "r")) == NULL) {
  printf("Cannot open input file - getparm.c");
  exit(0);
}

/* set order and list values to -1 */
for (i = 0; i < MAX_SIZES; i++) {
  order.number[i] = 0;
  order.ch_sizes[i][0] = 0;
}

for (i=0; i<MAX_LIST; i++) {
  list[i].inches = (float) 0.0;

  for (j = 0; j < MAX_SIZES; j++)
    list[i].sizes[j] = 0;
}

/* Input Units */
fscanf(fp,"%d", ou_units);
fscanf(fp,"%d", max_pLy);
fscanf(fp,"%d", max_sizes);
/* Input Order */
for (i = 0; i < MAX_SIZES; i++) {
    fscanf(fp, "%hd", &order.number[i]);
    if (order.number[i] == -1) {
        order.number[i] = 0;
        break;
    }
    fscanf(fp, "%s", order.ch_sizes[i]);
}
num_of_sizes = i;

/* Input List */
i = 0;
while (1) {
    fscanf(fp, "%d", &quantity);
    if (quantity == -2)
        break;
    while (quantity != -1) {
        fscanf(fp, "%d", &m);
        if (m >= num_of_sizes) {
            printf("ERROR in reading size variable - getparm.c");
            exit(0);
        }
        list[i].sizes[m] = quantity;
        fscanf(fp, "%d", &quantity);
    }
    fscanf(fp, "%f", &list[i].inches);
    ++i;
}
fclose(fp);
return(1);
/*----------------------------------------------------------*/
-- $Header: D:/cops/src/cherry/globals.h  January 1991
-- ----------------------------------------------------------*/

FILE NAME : Globals.h
PROGRAMMER : Terri A. Smith
DATE WRITTEN : January 1991
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: To declare all global variables

#include <stdio.h>
#include "cherdec.h"
#include "cherccl.h"

ord_var_t order;
list_t *list = NULL;
int num_of_sizes;
int num_list;
order_t temp_order;
section_t *sections = NULL;
int num_sections;
float total_inches = (float) 0.0;
float prev_inch = (float) 9999.0;
int num_hold_secs;
section_t *hold_secs;
int ply_height;
INCLUDES = cherdec.h cherlcl.h
LIBNAME = cherlib

OBJS = \
    globals.obj \
    getparm.obj \
    findinch.obj \
    combine.obj \
    ones.obj \
    chkinch.obj \
    cphold.obj \
    clrtemp.obj \
    twos.obj \
    threes.obj \
    fours.obj \
    fives.obj \
    sixes.obj

c.obj:
    $(CC)
    $(LIB)
    globals.obj : globals.c $(INCLUDES)
    getparm.obj : getparm.c $(INCLUDES)
    findinch.obj : findinch.c $(INCLUDES)
    combine.obj : combine.c $(INCLUDES)
    ones.obj : ones.c $(INCLUDES)
    twos.obj : twos.c $(INCLUDES)
    threes.obj : threes.c $(INCLUDES)
    fours.obj : fours.c $(INCLUDES)
    fives.obj : fives.c $(INCLUDES)
    sixes.obj : sixes.c $(INCLUDES)
    chkinch.obj : chkinch.c $(INCLUDES)
    cphold.obj : cphold.c $(INCLUDES)
    clrtemp.obj : clrtemp.c $(INCLUDES)
    cherry.obj : cherry.c $(INCLUDES)
    cherry.exe : cherry.obj $(OBJS)
    cl cherry /link cherlib.lib
    $(B)\cherry.exe : cherry.exe
$(CP)$

$(I)\backslash$cherdec.h = cherdec.h

$(CP)$
/* --------------------------------------------------------------*/
/* $Header$: D:\cops\src\cherry\ones.c  January 1991*/
/* --------------------------------------------------------------*/
/*---------------------------------------------------------------*/
/* I4 File Name: Ones.c                                         */
/* Program: Terri A. Smith                                     */
/* Date Written: January 1991                                  */
/* Address: GTRI/CSITL Atlanta GA 30332 (404) 894-8952          */
/*---------------------------------------------------------------*/
/* Purpose: Groups units in ones and find inches               */
/*---------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cherdec.h>
#include "chercli.h"

void ones(set_s, temp_secs, num_temp_secs)
{
  float inches;
  int j, i, m;
  j = *num_temp_secs;
  for (i=0; i<num_of_sizes; ++i) {
    if (set_s[i] == 1) {
      for (m=0; m<num_of_sizes; m++)
        temp_secs[i].sizes[m] = 0;
      temp_secs[i].sizes[0] = 1;
      inches = find_inches(temp_secs[i].sizes);
      if (inches != (float) 0.0) {
        total_inches += inches;
        ++j;
      } else
        temp_secs[i].sizes[i] = 0;
    } else
      temp_secs[i].sizes[i] = j;
  }
/* .........................................................
-- Header: D:/cops/src/cherry/sixes.c January 1991
-- .........................................................*/

FILE NAME : sixes.c
- PROGRAMMER : Terri A. Smith
- DATE WRITTEN : January 1991
- ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952
- PURPOSE- Recursive procedure to group units in sixes.
-
-*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherrdec.h"
#include "cherrcl.h"

void sixes(set_s, temp_secs, num_temp_secs)
{
    order t set_s;
    section t *temp_secs;
    int *num_temp_secs;

    float inches;
    int j, i, k, l, m, n, o;
    order t temp_order;
    float hold_inches1;
    float hold_inches2;
    int hold_temp_num;

    hold_temp_num = *num_temp_secs;
    hold_inches2 = total_inches;

    for (i=0; i<num_of_sizes; i++) {
        for (j=i+1; j<num_of_sizes; j++) {
            for (k=j+1; k<num_of_sizes; k++) {
                for (l=k+1; l<num_of_sizes; l++) {
                    for (m=l+1; m<num_of_sizes; m++) {
                        for (n=m+1; n<num_of_sizes; n++) {

                            temp_order[o] = 0;

                            if ((set_s[i] == 1) && (set_s[j] == 1) &&
                                (set_s[k] == 1) && (set_s[l] == 1) &&
                                (set_s[m] == 1) && (set_s[n] == 1)) {
                                temp_order[i] = 1;
                                temp_order[j] = 1;
                                temp_order[k] = 1;
                                temp_order[l] = 1;
                                temp_order[m] = 1;
                                temp_order[n] = 1;
                                inches = combine_inches(temp_order);
                                if (inches != (float) 0.0) {
                                    }}}}}}}}}}
for (o=0; o< num_of_sizes; o++)
    temp_secs["num_temp_secs].sizes[o] = 0;
total_inches = total_inches + inches;
temp_secs["num_temp_secs].sizes[i] = 1;
temp_secs["num_temp_secs].sizes[j] = 1;
temp_secs["num_temp_secs].sizes[k] = 1;
temp_secs["num_temp_secs].sizes[l] = 1;
temp_secs["num_temp_secs].sizes[m] = 1;
temp_secs["num_temp_secs].sizes[n] = 1;
++num_temp_secs;
}
temp_order[1] = 0;
temp_order[2] = 0;
temp_order[k] = 0;
temp_order[l] = 0;
temp_order[m] = 0;
temp_order[n] = 0;

for (o=0; o<num_of_sizes; o++)
    if ((o == i) && (o == j) && (o == k) &&
        (o == l) && (o == m) && (o == n) && (set_s[o] == 1))
        temp_order[o] = 1;

hold_inches1 = total_inches;
ones(temp_order, temp_secs, num_temp_secs);
check_inches(temp_secs, num_temp_secs);

for (o=0; o<num_of_sizes; o++)
    if ((o == i) && (o == j) && (o == k) &&
        (o == l) && (o == m) && (o == n) && (set_s[o] == 1))
        --num_temp_secs;

hold_inches1 = total_inches;
twos(temp_order, temp_secs, num_temp_secs);

hold_inches1 = total_inches;
threes(temp_order, temp_secs, num_temp_secs);

hold_inches1 = total_inches;
fours(temp_order, temp_secs, num_temp_secs);

hold_inches1 = total_inches;
fives(temp_order, temp_secs, num_temp_secs);

hold_inches1 = total_inches;
sixes(temp_order, temp_secs, num_temp_secs);

*num_temp_secs = hold_temp_num;
total_inches = hold_inches2;
/ * Header: D:\cops\src\cherry\threes.c January 1991 * /

/*------------------------------------------------------*/

FILE NAME : Threes.c
PROGRAMMER : Terri A. Smith
DATE WRITTEN : January 1991
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: Recursive procedure to group units in threes

------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "chertel.hu"

void threes(set_s, temp_secs, num_temp_secs)
{
    order_t set_s;
    section_t *temp_secs;
    int *num_temp_secs;
    
    float inches;
    int j, i, k, l;
    order_t temp_order;
    float hold_inches1;
    float hold_inches2;
    int hold_temp_num;
    
    hold_temp_num = *num_temp_secs;
    hold_inches2 = total_inches;
    
    for (i=0; i<num_of_sizes; i++) {
        for (j=i+1; j<num_of_sizes; j++) {
            for (k=j+1; k<num_of_sizes; k++) {
                
                if (((set_s[i] == 1) && (set_s[j] == 1) && (set_s[k] == 1))
                
                temp_order[i] = 0;
                temp_order[j] = 1;
                temp_order[k] = 1;
                
                inches = combine_inches(temp_order);
                if (inches != (float) 0.0) {
                    for (i=0; i<num_of_sizes; i++)
                    temp_secs[*num_temp_secs].sizes[i] = 0;
                    total_inches = total_inches + inches;
                    temp_secs[*num_temp_secs].sizes[i] = 1;
                    temp_secs[*num_temp_secs].sizes[] = 1;
                    temp_secs[*num_temp_secs].sizes[k] = 1;
                    ++*num_temp_secs;
                }
```c
58     temp_order[i] = 0;
59     temp_order[i] = 0;
60     temp_order[k] = 0;
61     for (l=0; l<num_of_sizes; l++) {
62         if (((l == 1) && (l == j) && (l == k) && (set_s[l] == 1)) {
63             temp_order[l] = 1;
64         }
65     }
66     hold_inches1 = total_inches;
67     ones(temp_order, temp_secs, num_temp_secs);
68     check_inches(temp_secs, num_temp_secs);
69     for (l=0; l<num_of_sizes; l++) {
70         if (((l == 1) && (l == j) && (l == k) && (set_s[l] == 1)) {
71             --num_temp_secs;
72         }
73     }
74     total_inches = hold_inches1;
75     twos(temp_order, temp_secs, num_temp_secs);
76     total_inches = hold_inches1;
77     /*
78        for (l=0; l<num_of_sizes; l++) {
79            if (((l == 1) && (l == j) && (l == k) && (set_s[l] == 1)) {
80                --num_temp_secs;
81            }
82        }
83    */
84    total_inches = hold_inches1;
85    threes(temp_order, temp_secs, num_temp_secs);
86    *num_temp_secs = hold_temp_num;
87    total_inches = hold_inches2;
88    
89 }
/* -----------------------------------------------------------------------------
   FILE:    D:/copes/src/cherry/twos.c   January 1991
   PROGRAMMER: Terri A. Smith
   DATE WRITTEN: January 1991
   ADDRESS: GTRI/CSITL Atlanta GA 30332   (404) 894-8952
   PURPOSE: Recursive procedure to group units in twos
   -----------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cherdec.h"
#include "chercl.h"

void twos(set_s, temp seks, num_temp Seks)
{
  order_t set_s;
  section_t *temp seks;
  int *num_temp Seks;
  float inches;
  int i, k, m;
  order_t temp order;
  float hold_inches1;
  float hold_inches2;
  int hold_temp_num;
  hold_temp_num = *num_temp Seks;
  hold_inches2 = total_inches;

  for (i=0; i<num_of_sizes; i++) {
    for (j=i+1; j<num_of_sizes; j++) {
      for (k=0; k<num_of_sizes; k++)
        temp order[k] = 0;

      if ((set_s[i] == 1) && (set_s[j] == 1)) {
        temp order[i] = 1;
        temp order[j] = 1;
        inches = combine_inches(temp order);
        if (inches != (float) 0.0) {
          for (m=0; m<num_of_sizes; m++)
            temp seks[*num_temp seks].sizes[m] = 0;
          total_inches = total_inches + inches;
          temp seks[*num_temp seks].sizes[i] = 1;
          temp seks[*num_temp seks].sizes[j] = 1;
          ++*num_temp seks;
          printf("WITH TOTAL = %.4f, total_inches"); */
        }
        temp order[i] = 0;
      }
    }
  }
}"
temp_order[j] = 0;

for (k=0; k<num_of_sizes; k++) {
    if ((k == i) && (k == j) && (set_s[k] == 1)) {
        temp_order[k] = 1;
    }
}

hold_inches1 = total_inches;
ones(temp_order, temp_secs, num_temp_secs);

check_inches(temp_secs, num_temp_secs);

for (k=0; k<num_of_sizes; k++) {
    if ((k == i) && (k == j) && (set_s[k] == 1)) {
        --num_temp_secs;
    }
}

total_inches = hold_inches1;
twos(temp_order, temp_secs, num_temp_secs);
*num_temp_secs = hold_temp_num;
total_inches = hold_inches2;
      
    )
    )

)
Appendix F  Improvement Algorithm Source Code
/ * ---------*------------------------------------------------------------*/
/* $Header:: D:\cops\src/improv/case_a1.c February 1991 */
/* ---------*------------------------------------------------------------*/

- FILE NAME : case_a1.c
- PROGRAMMER : Terri A. Smith
- DATE WRITTEN : February 1991
- ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952
- PURPOSE- To determine savings if sizes in two sections are
- the same
- *
- #include <stdio.h>
- #include <stdlib.h>
- #include "impdec.h"
- #include "implcl.h"

float case_a1(sect1, portion, cut_cost)

section_t *sect1;
section_t *portion;
int cut_cost;


for (i=0; i<num_of_sizes; i++) {
    e = e + (order.perimeter[i] * sect1->sizes[i]);
    e = e + (order.perimeter[i] * portion->sizes[i]);
}

savings = (float) cut_cost * e;

return(savings);
/* Header: D:/cops/src/improv/case_ali.c February 1991 */

FILE_NAME : case_ali.c
PROGRAMMER : Terri A. Smith
DATE_WRITTEN: February 1991
ADDRESS: GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: To determine savings by lying sizes next to each other instead of on top.

#include <stdio.h>
#include <stdlib.h>
#include "impdec.h"
#include "implcl.h"

float case_ali(i, l, unit_cost)
{
    int i;
    int l;
    int unit_cost;

    float savings = (float) 0.0;
    float sect1_inch;
    float sect2_inch;
    float sect3_inch;
    float sect4_inch;

    sect1_inch = find_inches(in_section[i].sizes);
    sect2_inch = find_inches(in_section[i].sizes);
    sect3_inch = find_inches(sect3.sizes);
    sect4_inch = find_inches(sect4.sizes);

    savings = unit_cost * in_section[i].ply_height * (sect1_inch + sect2_inch - sect3_inch - sect4_inch);

    return(savings);
}
/* -- $Header:: D:/cops/src/improve/combply.c  February 1990 */

FILE NAME: Cobply.c
PROGRAMMER: Terri A. Smith
DATE WRITTEN: April 1990
ADDRESS: GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: This procedure combines sections which have the same sizes
and the ply height of the new section does not exceed the max_ply

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <memory.h>
#include "impdec.h"
#include "imptcl.h"

void combine_ply(max_ply)
{
    int max_ply;

    int i,j,k;
    char match;

    for (i=0; i<num_in_sec; i++) {
        for (j=i+1; j<num_in_sec; j++) {
            match = 1;
            for (k=0; k<num_of_sizes; k++) {
                if (in_section[i].sizes[k] != in_section[j].sizes[k])
                    match = 0;
            }
            if ((match) && ((in_section[i].ply_height + in_section[j].ply_height) <= max_ply)) {
                in_section[i].ply_height = in_section[i].ply_height + in_section[j].ply_height;
                for (i=1; i<num_in_sec-1; i++)
                    memcpy(&in_section[i], &in_section[i+1], sizeof(section_t));
                --j;
                --num_in_sec;
            }
        }
    }
    num_temp_sec = num_in_sec;
    return;
}
/* Header: D:/cops/src/improv/comsize.c February 1990
   */

FILE NAME : Combsize.c
PROGRAMMER : Terri A. Smith
DATE WRITTEN : April 1990
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: This procedure combines all sections in which the
the number of sizes in the new section does not exceed
the max_sizes allowed per section

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <memory.h>
#include "impdec.h"
#include "implc1.h"

void combine_sizes(max_sizes)
    int max_sizes;
{
    int num_units;
    int i, j, t;
    for (i=0; i<num_in_sec; i++) {
        for (j=i+1; j<num_in_sec; j++) {
            num_units = 0;
            for (l=0; l<num_of_sizes; l++)
                num_units = num_units + in_section[i].sizes[l] +
                in_section[j].sizes[l];
            if ((num_units <= max_sizes) &&
                (in_section[i].ply_height == in_section[j].ply_height)) {
                for (l=0; l<num_of_sizes; l++)
                    in_section[i].sizes[l] = in_section[j].sizes[l] +
                    in_section[j].sizes[l];
                for (l=j; l<num_in_sec-1; l++)
                    memcpy(&in_section[l], &in_section[l+1], sizeof(section_t));
                --j;
                --num_in_sec;
            }
        }
    }
    num_temp_sec = num_in_sec;
    return;
}
FILE NAME: compswap.c
PROGRAMMER: Terri A. Smith
DATE WRITTEN: February 1991
ADDRESS: GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: To determine which method to use to compute the savings.

#include <stdio.h>
#include <stdlib.h>
#include "impl.h"

float compute_swap_savings(int i, int cut_cost, int unit_cost, int max_sizes)
{
    float savings;
    int l;
    int cut_cost;
    int unit_cost;
    int max_sizes;

    if (in_section[i].ply_height == in_section[i].ply_height) {
        savings = case_iii(i, l, unit_cost);
        temp_save.type= 3;
        temp_save.cand_ply_height = in_section[i].ply_height;
        temp_save.org_ply_height = in_section[i].ply_height;
    }
    else {
        temp_save.cand_ply_height = in_section[i].ply_height;
        temp_save.org_ply_height = in_section[i].ply_height;
        savings = case_iii(i, l, unit_cost);
        temp_save.type= 4;
    }

    temp_save.savings = savings;
    return(savings);
}
/* ------------------------------
- FILE NAME : compute.c
- PROGRAMMER : Terri A. Smith
- DATE WRITTEN : February 1991
- ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952
- PURPOSE: To determine which method to use to compute
  the savings
- -------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include "mpdec.h"
#include "implct.h"

float compute_savings(i, l, cut_cost, unit_cost, max_sizes)
{
    int i;
    int l;
    int cut_cost;
    int unit_cost;
    int max_sizes;

    int j;
    float savings = (float) 0.0;
    float save2;
    char match = 1;
    int num_units = 0;

    for (j=0; j<num_of_sizes; j++) {
        if (portion.sizes[j] != in_section[l].sizes[j])
            match = 0;

        num_units = num_units + sect4.sizes[j];
    }

    if (match) { /* sizes in sections are the same */
        if (num_units <= max_sizes) {
            save2 = case_all(i, l, unit_cost);

            if (save2 > savings) {
                temp_save.type= 2;
                savings = save2;

                if (in_section[l].ply_height == in_section[l].ply_height)
                    temp_save.cand_ply_height = in_section[l].ply_height;
                else temp_save.cand_ply_height = in_section[l].ply_height;
                temp_save.org_ply_height = in_section[l].ply_height;
            }
        }
    }

    else if ((in_section[l].ply_height == in_section[l].ply_height) && (num_units <= max_sizes)) {
savings = case_all(i, l, unit_cost);
temp_save.type= 3;
temp_save.cand_ply_height = in_section[i].ply_height;
temp_save.org_ply_height = in_section[i].ply_height;

else if (num_units <= max_sizes) {
    if (in_section[i].ply_height != in_section[l].ply_height)
        temp_save.cand_ply_height = in_section[i].ply_height;
    else temp_save.cand_ply_height = in_section[l].ply_height;
    temp_save.org_ply_height = in_section[l].ply_height;
    savings = case_all(i, l, unit_cost);
temp_save.type= 4;
}

temp_save.savings = savings;

return(savings);

}
```c
/* ------------------------------------------------------------------------
 * FILE NAME : Findinch.c
 * PROGRAMMER : Terri A. Smith
 * DATE WRITTEN : February 1991
 * ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952
 * PURPOSE- To determine the number of inches in a section based
 * on the input list Is
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "imdec.h"
#include "implcl.h"

float find_inches(sizes) {
    order_t sizes;

    int i, j;
    char match = 0;
    char empty = 0;

    i = 0;
    while (((match) && (i < num_list)) { 
        empty = 1;
        match = 1;
        for (j=0; j<num_of_sizes; j++) {
            if (sizes[j] != list[i].sizes[j])
                match = 0;
            if (sizes[j] == 0)
                empty = 0;
            }
        ++i;
    }
    if (empty)
        return((float) 0.0);
    if (match)
        return(list[--i].inches);
    else {
        printf(" COULDN'T FIND ");
        for (i=0; i<num_of_sizes; i++) {
            if (sizes[i] > 0)
                printf("%d %s, sizes[i], order.ch_sizes[i]);
            }
        printf("\n");
        exit(0);
    }
```
/* -------------------------------                     */
-- $Header: D:/cops/src/improv/getparm.c February 1990
-- ------------------------------- */

/*-------------------------------*/
- FILE NAME : Getparm.c
- PROGRAMMER : Terri A. Smith
- DATE WRITTEN: February 1990
- ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952
- PURPOSE: To read the input parameters from a file
- ------------------------------- */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include "impdec.h"
#include "implicl.h"

int get_parameters(ou_units, max_ply, max_sizes,
cut_cost, unit_cost, old_ou_units)

int *ou_units;
int *max_ply;
int *max_sizes;
int *cut_cost;
int *unit_cost;
int *old_ou_units;

{ int i, j;

FILE *fp = NULL;
int quantity;
int m;

if ((fp = fopen("INPUT", "r")) == NULL) {
    printf("Cannot open input file - getparm.c");
    exit(0);
}

/* set order and list values to -1 */
for (i = 0; i < MAX_SIZES; i++) {
    order.number[i] = 0;
    order.ch_sizes[i][0] = 0;
    order.perimeter[i] = 0;
}

for (i = 0; i < MAX_LIST; i++) {
    list[i].inches = (float) 0.0;

    for (j = 0; j < MAX_SIZES; j++)
        list[i].sizes[j] = 0;
}
fscanf(fp,"Xd", ou_units);
fscanf(fp,"Xd", max_ply);
fscanf(fp,"Xd", max_sizes);
fscanf(fp,"Xd", cut_cost);
fscanf(fp,"Xd", unit_cost);

/* Input Order */
for (i = 0; i < MAX_SIZES; i++) {
    fscanf(fp,"Xd", &order.number[i]);
    if (order.number[i] == -1) {
        order.number[i] = 0;
        break;
    }
    fscanf(fp,"Xd", &order.perimeter[i]);
    fscanf(fp,"Xd", order.ch_sizes[i]);
}
num_of_sizes = i;

fscanf(fp,"Xd", &num_in_sec);
if ((in_section = (section_t *)malloc(num_in_sec * sizeof(section_t))) == NULL) {
    printf("ALLOCATION ERROR - SECTIONS getparm.c\n");
    exit(0);
}
for (i = 0; i < num_in_sec; i++) {
    in_section[i].ply_height = 0;
    for (m = 0; m < num_of_sizes; m++) {
        in_section[i].sizes[m] = 0;
    }
}

i = 0;
/* Input Sections */
while (i < num_in_sec) {
    fscanf(fp,"Xd", &quantity);
    while (quantity != -1) {
        fscanf(fp,"Xd", &m);
        if (m >= num_of_sizes) {
            printf("ERROR in reading size variable - getparm.c\n");
            exit(0);
        }
        in_section[i].sizes[m] = quantity;
        fscanf(fp,"Xd", &quantity);
    }
    fscanf(fp,"Xd", &in_section[i].ply_height);
    ++i;
}
fscanf(fp,"%d", &quantity);

if (quantity == -2)
    break;

while (quantity != -1) {
    fscanf(fp,"%d", &m);
    if (m >= num_of_sizes) {
        printf("ERROR in reading size variable - getparm.c");
        exit(0);
    }
    list[i].sizes[m] = quantity;
    fscanf(fp,"%d", &quantity);
}

fscanf(fp,"%f", &list[i].inches);

++i;
}
}
fclose(fp);

return(i);
/* ------------------------------*/
-- $Header:: D:/cops/src/improv/globals.h February 1991
-- --------------------------------*/

FILE NAME: Globals.h
PROGRAMMER: Terri A. Smith
DATE WRITTEN: February 1991
ADDRESS: GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: To declare all global variables

#include <stdio.h>
#include "impdec.h"
#include "implcl.h"

ord_var_t order;
list_t *list = NULL;
int num_of_sizes;
int num_list;
section_t *in_section = NULL;
int num_in_sec;
int num_temp_sec;
section_t sect3;
section_t sect4;
section_t portion;
savings_t temp_save;
savings_t save;
/**----------------------------------------------*/
-- $Header:: D:\cops\src\improv\impdec.h February 1990--
-- ____________________________________________/*

/**----------------------------------------------*/
FILE NAME : Impdec.h
PROGRAMMER : Terri A. Smith
DATE WRITTEN : February 1990
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952
PURPOSE- To define all structures and procedures

#ifndef IMPDEC_H
#define IMPDEC_H

#define MAX_LIST 1000
#define MAX_SIZES 25
#define MAX_SAVINGS 400

typedef int order_t[MAX_SIZES];
typedef char sizes_t[MAX_SIZES][10];
typedef struct C {
  order_t number;
  sizes_t ch_sizes;
  int perimeter[MAX_SIZES];
  } ord_var_t;

typedef struct {
  order_t sizes;
  float inches;
  } list_t;

typedef struct {
  order_t sizes;
  int ply_height;
  char merged;
  } section_t;

typedef struct {
  int sect1;
  int sect2;
  int org_ply_height;
  int cand_ply_height;
  float savings;
  int type;
  order_t org;
  order_t cand;
  order_t in_sect1;
  order_t in_sect2;
  } savings_t;
/*----------------------------------------------*/
int get_parameters(int *units, int *max_ply, int *max_sizes,
                    int *cut_cost, int *unit_cost, int* old_units);
float find_inches(order_t sizes);
float case_all(int i, int j, int unit_cost);
float compute_savings(int i, int j, int cut_cost, int unit_cost, int max_sizes);
float compute_swap_savings(int i, int j, int cut_cost, int unit_cost, int max_sizes);
void combine_ply(int max_ply);
void combine_sizes(int max_sizes);
void transfer_forward(int i, int j, int l,
                      int cut_cost, int unit_cost, int max_sizes, int max_ply);
void transfer_backwards(int i, int j, int l,
                         int cut_cost, int unit_cost, int max_sizes, int max_ply);
void swap_forward(int i, int j, int l,
                  int cut_cost, int unit_cost, int max_sizes, int max_ply);
void swap_backwards(int i, int j, int l,
                   int cut_cost, int unit_cost, int max_sizes, int max_ply);
#endif
/*---------------------------------------------*/
/*$Header:: D:\cops\src\improv\impdec.h  February 1990*/
/*---------------------------------------------*/

FILE NAME : Implcl.h
PROGRAMMER : Terri A. Smith
DATE WRITTEN : February 1990
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE- To define all global variables

#define IMPLCL_H

extern ord_var_t order;
extern list_t *list;
extern int num_list;
extern int num_of_sizes;
extern section_t *in_section;
extern int num_in_sec;
extern int num_temp_sec;
extern section_t sect3;
extern section_t sect4;
extern section_t portion;
extern savings_t temp_save;
extern savings_t save;

#endif
```c
/* $Header: D:/cops/src/improv/improve.c February 1990 */

FILE NAME: improve.c
PROGRAMMER: Terri A. Smith
DATE WRITTEN: February 1990
ADDRESS: GTRI/CSITL Atlanta GA 30332 (404) 894-8952

Purpose: The main program which controls flow of execution

#include <stdio.h>
#include <malloc.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "impdec.h"
#include "inploc.h"

#define clocko) time(NULL)

main(argc, argv)
    int argv;
    char *argv[];
{
    /* Input Variables */
    int ou_units;   /* # of units over/under allowed */
    int old_ou_units; /* # of units over/under allowed */
    int max_ply;    /* max ply height allowed */
    int max_sizes;  /* # of sizes allowed / section */
    int init_ply;   /* initial ply height */
    int cut_cost;   /* cutting cost */
    int unit_cost;  /* unit cost */

    /* Output Variables */
    float tot_length; /* the total amt of fabric needed*/
    float tot_marker; /* total fabric between markers */
    int unit_dev;    /* deviation of units to cut from order */
    int unit_count;  /* units in all sections */
    int order_count; /* # of units in order */
    char unit_string[10]; /* OVER or UNDER */

    int i, j, k, l, r, s, m, n; /* counters */
    int total_order = 0; /* total # in order */
    float inches;       /* inches in sections * ply */
    float marker;       /* inches between markers */
    FILE *fp;           /* file pointer for output */
    order_t temp_order; /* temp order */
    clock_t start_time; /* used for timing alg */
    clock_t end_time;   /* used for timing alg */
    double total_time;  /* total execution time */
    char mergers_possible = 1; /* while loop boolean */
```
section_t temp_sec; /* temporary section */

start_time = clock();

if ((fp = fopen("OUTPUT", "w")) == NULL) {
    printf("CANNOT OPEN OUTPUT FILE savings.c\n");
    exit(0);
}

if ((list = (list_t *)malloc(MAX_LIST * sizeof(list_t))) == NULL) {
    printf("ALLOCATION ERROR FOR LIST savings.c\n");
    exit(0);
}

/* Get parameters and print out first solution */

num_list = get_parameters(&ou_units, &max_ply, &max_sizes,
                           &cut_cost, &unit_cost, &old_ou_units);

tot_length = (float) 0.0;
tot_marker = (float) 0.0;
fprintf(fp, "MAX PLY = %d MAX # OF UNITS PER SECTION = %d\n", max_ply, max_sizes);
fprintf(fp, "UNIT COST = %d cents CUT COST = %d cents\n", unit_cost, cut_cost);
fprintf(fp, "ORDER\n");
for (i=0; i<num_of_sizes; i++) {
    fprintf(fp, "%d SIZE %s\n", order.number[i], order.ch_sizes[i]);
}

fprintf(fp, "\nFIRST SOLUTION \n");
for (i=0; i<num_in_sec; i++) {
    fprintf(fp, "SECTION %d HAS PLY = %d\n", i, in_section[i].ply_height);
    for (j=0; j<num_of_sizes; j++) {
        if (in_section[i].sizes[j] > 0) {
            fprintf(fp, " AND %d SIZE %s\n", in_section[i].sizes[j], order.ch_sizes[j]);
        }
    }
    marker = find_inches(in_section[i].sizes);
    inches = marker * in_section[i].ply_height;
    fprintf(fp, "MARKER LENGTH = %7.2f TOTAL LENGTH = %7.2f\n", marker, inches);
    tot_length = tot_length + inches;
    tot_marker = tot_marker + marker;
}

fprintf(fp, "TOTAL MARKER = %7.2f TOTAL LENGTH = %7.2f\n", tot_marker, tot_length);

/* Initialize savings structures */

for (i=0; i<num_of_sizes; i++) {
    save.org[i] = 0;
    save.cand[i] = 0;
    temp_save.org[i] = 0;
    temp_save.cand[i] = 0;
}

/* Initialize savings structures */
for (i=0; i<num_of_sizes; i++) {
    save.org[i] = 0;
    save.cand[i] = 0;
    temp_save.org[i] = 0;
    temp_save.cand[i] = 0;
}
combine any sections with a combination of sizes <= max_sizes
*/

combine_sizes(max_sizes);

/*
Main Loop of program -
The loop begins by trying to place one sizes form one section
into another section. Once all possible transferred are tested,
then the program tries swapping two sizes from two different
sections. The loop begins with the first section. The best
transfer or swap from this section is made and the next section
goes through the same tests etc. Once all sections have been
exhausted then the same is repeated but backwards (starting
with the last section. This whole process is repeated twice.
*/
mergers_possible = 2;
while (mergers_possible > 0) {
  /* combine any sections with same sizes by putting on
   top of each other if it doesn't violate max ply height
   */
  combine_ply(max_ply);

  /*
  Attempt to reassign one portion from original section
  to a new section and calculate savings. Merge only
  the one with the greatest savings
  */
  for (i=0; i<num_in_sec; i++) {
    for (j=0; j<num_of_sizes; j++) {
      save.sect1 = -1;
      save.sect2 = -1;
      save.type = 0;
      save.org_ply_height = 0;
      save.cand_ply_height = 0;
      save.savings = (float) 0.0;
      for (m=0; m<num_of_sizes; m++)
        portion.sizes[m] = 0;
      portion.ply_height = 0;
      for (l=i+1; l<num_in_sec; l++) {
        transfer_forward(i, j, l, cut_cost, unit_cost, max_sizes, max_ply);
        swap_forward(i, j, l, cut_cost, unit_cost, max_sizes, max_ply);
      }
      /*
      Place portion into section. If the two sections have
different ply heights then the smallest ply height is
given to both sections and the section with the larger
ply height is added to the end of the section list with
a ply height equal to larger ply minus the smaller ply
      */
r = save.sect1;
s = save.sect2;

if (save.savings != (float) 0.0) {
    printf("REPLACING PORTION %d %d\n", r, s);
    in_section[r].ply_height = save.org_ply_height;
    in_section[s].ply_height = save.cand_ply_height;

    if (save.org_ply_height < save.cand_ply_height) {
        in_section[s].ply_height = save.org_ply_height;
        temp_sec.ply_height = save.cand_ply_height -
        save.org_ply_height;

        for(m=0; m<num_of_sizes; m++)
            temp_sec.sizes[m] = save.in_sect2[m];

        if ((in_section = realloc(in_section, ((num_temp_sec + 1) 
            * sizeof(section_t)))) == NULL) {
            printf("REALLOCATION ERROR FOR INSECTION improve2.c\n");
            exit(0);
        }

        memcpy(&in_section[num_temp_sec], &temp_sec, sizeof(section_t));
    } else if (save.org_ply_height > save.cand_ply_height) {
        in_section[r].ply_height = save.cand_ply_height;
        temp_sec.ply_height = save.org_ply_height -
        save.cand_ply_height;

        for(m=0; m<num_of_sizes; m++)
            temp_sec.sizes[m] = save.in_sect1[m];

        if ((in_section = realloc(in_section, ((num_temp_sec + 1) 
            * sizeof(section_t)))) == NULL) {
            printf("REALLOCATION ERROR FOR INSECTION improve2.c\n");
            exit(0);
        }

        memcpy(&in_section[num_temp_sec], &temp_sec, sizeof(section_t));
    }

    for(m=0; m<num_of_sizes; m++) {
        in_section[r].sizes[m] = save.org[m];
        in_section[s].sizes[m] = save.cand[m];
    }
}

/* for j */
/* for i */

/* Perform the same sequence of events to transfer and swap sizes but start at end of list and go backwards

Attempt to reassign one portion from original section
to a new section and calculate savings. Merge only
the one with the greatest savings

```
    num_in_sec = num_temp_sec;
    for (i=num_in_sec-1; i>=0; i--) {
        for (j=0; j<num_of_sizes; j++) {
            save sect1 = -1;
            save sect2 = -1;
            save.type = 0;
            save.org_ply_height = 0;
            save.cand_ply_height = 0;
            save.savings = (float) 0.0;
            for (m=0; m<num_of_sizes; m++)
                portion_sizes[m] = 0;
            portion_ply_height = 0;
            for (i=1; i>=0; i--)
                transfer_backwards(i, j, l, cut_cost, unit_cost, max_sizes, max_ply);
            swap_backwards(i, j, l, cut_cost, unit_cost, max_sizes, max_ply);
            r = save sect1;
            s = save sect2;
            if (save.savings != (float) 0.0) {
                printf("REPLACING PORTION %d \n", r, s);
                in_section[r].ply_height = save.org_ply_height;
                in_section[s].ply_height = save.cand_ply_height;
                if (save.org_ply_height < save.cand_ply_height) {
                    in_section[s].ply_height = save.org_ply_height;
                    temp_sec.ply_height = save.cand_ply_height -
                        save.org_ply_height;
                    for(m=0; m<num_of_sizes; m++)
                        temp_sec.sizes[m] = save.in_sect2[m];
                    if (((in_section = realloc(in_section, (num_temp_sec + 1)
                        * sizeof(section_t))) == NULL) {
                        printf("REALLOCATION ERROR FOR INSECTION \n improve2.c\n");
                        exit(0);
                    }
                    memcpy(&in_section[num_temp_sec+1], &temp_sec, sizeof(section_t));
                } else if (save.org_ply_height > save.cand_ply_height) {
                    in_section[r].ply_height = save.cand_ply_height;
                    temp_sec.ply_height = save.org_ply_height -
                        save.cand_ply_height;
                    for(m=0; m<num_of_sizes; m++)
                        temp_sec.sizes[m] = save.in_sect1[m];
                } else {
```
if ((in_section = realloc(in_section, ((num_temp_sec + 1) * sizeof(section_t))) == NULL) {
    printf("REALLOCATION ERROR FOR INSECTION improve2.c\n");
    exit(0);
}
memcpy(&in_section[num_temp_sec++], &temp_sec, sizeof(section_t));
}
for(m=0; m<num_of_sizes; m++) {
    in_section[i].sizes[m] = save.org[m];
    in_section[s].sizes[m] = save.cand[m];
}
/* for j */
/* for i */

num_in_sec = num_temp_sec;
--mergers_possible;
/*/ while */

/* Remove sections that are empty */

for (i=0; i<num_in_sec; i++) {
    order_count = 0;
    for (j=0; j<num_of_sizes; j++) {
        order_count = order_count + in_section[i].sizes[j];
    }
    if (order_count == 0) {
        for (j=i; j<num_in_sec-1; j++) {
            memcpy(&in_section[j], &in_section[j+1], sizeof(section_t));
        }
        num_in_sec = num_in_sec - 1;
    }
}

end_time = clock();
total_time = ((double) end_time - start_time) / CLK_TCK;

fprintf(fp, "\n\n******************************\n");
tot_length = (float) 0.0;
tot_marker = (float) 0.0;
unit_dev = 0;
order_count = 0;
unit_count = 0;

fprintf(fp, "THE # OF FINAL SECTIONS ARE : %d\n", num_in_sec);
for (i=0; i<num_in_sec; i++) {
    fprintf(fp, "SECTION %d HAS PLY %d\n", i, in_section[i].ply_height);
    for (j=0; j<num_of_sizes; j++) {
        if (in_section[i].sizes[j] > 0) {
            fprintf(fp, " AND %d PLY\n", in_section[i].sizes[j], order.ch_sizes[j])
        }
        unit_count = unit_count + (in_section[i].sizes[j] * in_section[i].ply_height);
    }
}
marker = find_inches(in_section[i].sizes);
inches = marker * in_section[i].ply_height;
fprintf(fp, "MARKER LENGTH = %7.2f TOTAL LENGTH = %7.2f inches", marker, inches);
tot_length = tot_length + inches;
tot_marker = tot_marker + marker;
}
for (j=0; j<num_of_sizes; j++)
    order_count = order_count + order.number[j];
unit_dev = order_count - unit_count;
if (unit_dev > 0)
    strcpy(unit_string, "UNDER");
else if (unit_dev == 0)
    strcpy(unit_string, "0");
else {
    unit_dev = unit_dev * -1;
    strcpy(unit_string, "OVER");
}
fprintf(fp, "TOTAL MARKER = %7.2f TOTAL LENGTH = %7.2f inches", tot_marker, tot_length);
fprintf(fp, "MARKER OVER/UNDER = %d %s", unit_dev, unit_string);
fprintf(fp, "TOTAL TIME = %f", total_time);
if (list != NULL)
    free(list);
fclose(fp);
return(0);
INCLUDES = impdec.h implicl.h
LIBNAME = implib

OBJS = \
globals.obj \ngetparm.obj \nfindinch.obj \ncase_all.obj \ncompute.obj \ncompswap.obj \ncombsize.obj \ncomply.obj \ntranfrwd.obj \nswapfrwd.obj \ntrankwd.obj \nswapbkwd.obj

.c.obj:
  $(CC)
  $(LIB)

globals.obj : globals.c $(INCLUDES)
getparm.obj : getparm.c $(INCLUDES)
findinch.obj : findinch.c $(INCLUDES)
case_all.obj : case_all.c $(INCLUDES)
compute.obj : compute.c $(INCLUDES)
compswap.obj : compswap.c $(INCLUDES)
comply.obj : comply.c $(INCLUDES)
combsize.obj : combsize.c $(INCLUDES)
tranfrwd.obj : tranfrwd.c $(INCLUDES)
swapfrwd.obj : swapfrwd.c $(INCLUDES)
trankwd.obj : trankwd.c $(INCLUDES)
swapbkwd.obj : swapbkwd.c $(INCLUDES)

improve.obj : improve.c $(INCLUDES)
improve.exe : improve.obj $(OBJS)
  cl improve /link implib.lib

$(B)\improve.exe : improve.exe
$(CP)
58 \$\{I\}\impdec.h : \impdec.h
59 \$\{CP\}
60
61
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <stdlib.h>
#include <string.h>
#include "impdec.h"
#include "implcl.h"

void swap_backwards(i, j, l, cut_cost, unit_cost, max_sizes, max_ply)
{
    int i;
    int j;
    int l;
    int cut_cost;
    int unit_cost;
    int max_sizes;
    int max_ply;

    int k, m, n; /* counters */
    int num_units; /* num_units in one section */

    for (n=0; n<num_of_sizes; n++) {
        if ((in_section[i].sizes[j] > 0) &&
            (in_section[i].sizes(n) > 0)) {
            for (m=0; m<num_of_sizes; m++) {
                sect3.sizes[m] = in_section[i].sizes[m];
                sect4.sizes[m] = in_section[i].sizes[m];
            }
            sect3.sizes[j] = sect3.sizes[j] - 1;
            sect3.sizes[n] = sect3.sizes[n] + 1;
            sect4.sizes[j] = sect4.sizes[j] + 1;
            sect4.sizes[n] = sect4.sizes[n] - 1;
            temp_save.section1 = i;
            temp_save.section2 = j;
            temp_save.type = 0;
            temp_save.org_ply_height = 0;
        }
    }
temp_save.cand_ply_height = 0;

temp_save.savings = (float) 0.0;

compute_swap_savings(l, l, cut_cost, unit_cost, max_sizes);

num_units = 0;
for (m=0; m<num_of_sizes; m++)
    num_units = num_units + sect4.sizes[m];

if ((temp_save.savings > save.savings) &&
    (num_units <= max_sizes) &&
    (temp_save.type > 0) &&
    (temp_save.cand_ply_height <= max_ply)) {
    memcpy(&save, &temp_save, sizeof(savings_t));

    for (m=0; m<num_of_sizes; m++) {
        if (temp_save.type != 1) {
            save.org[m] = sect3.sizes[m];
            save.cand[m] = sect4.sizes[m];
            save.in_sect1[m] = in_section[i].sizes[m];
            save.in_sect2[m] = in_section[i].sizes[m];
        }
        else
            save.cand[m] = in_section[i].sizes[m];
        /* for m */
    } /* if */
    /* if */
} /* for n */
return;
FILE NAME : Swapfrwd.c
PROGRAMMER : Terri A. Smith
DATE WRITTEN : April 1990
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE - This procedure attempts to swap one size from one
with another size in a different section if feasible. It
works from the start of the section list to the end.

#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <stdlib.h>
#include <string.h>
#include "impdec.h"
#include "implcl.h"

void swap_forward(i, j, l, cut_cost, unit_cost, max_sizes, max_ply)

int i;
int j;
int l;
int cut_cost;
int unit_cost;
int max_sizes;
int max_ply;

{ int k, m, n; /* counters */

int num_units;

for (n=0; n<num_of_sizes; n++)

  if (l = (in_section[i].sizes[i] > 0) &&
      (in_section[l].sizes[l] > 0))
      { for (m=0; m<num_of_sizes; m++)
        sect3.sizes[m] = in_section[i].sizes[m];
        sect4.sizes[m] = in_section[l].sizes[m];
        }

        sect3.sizes[i] = sect3.sizes[i] - 1;
        sect3.sizes[n] = sect3.sizes[n] + 1;
        sect4.sizes[i] = sect4.sizes[i] + 1;
        sect4.sizes[n] = sect4.sizes[n] - 1;
        temp_save.sect1 = 1;
        temp_save.sect2 = 1;
        temp_save.type = 0;
        temp_save.org_ply_height = 0;
temp_save.cand_ply_height = 0;
temp_save.savings = (float) 0.0;
compute_swap_savings(l, l, cut_cost, unit_cost, max_sizes);

num_units = 0;
for (m=0; m<num_of_sizes; m++)
    num_units = num_units + sect4.sizes[m];
if ((temp_save.savings > save.savings) &&
    (num_units <= max_sizes) &&
    (temp_save.type > 0) &&
    (temp_save.cand_ply_height <= max_ply)) {
    memcpy(&save, &temp_save, sizeof(savings_t));
    for (m=0; m<num_of_sizes; m++) {
        if (temp_save.type == 1) {
            save.org[m] = sect3.sizes[m];
            save.cand[m] = sect4.sizes[m];
            save.in_sect1[m] = in_section[i].sizes[m];
            save.in_sect2[m] = in_section[i].sizes[m];
        } else
            save.cand[m] = in_section[i].sizes[m];
    }
}
return;
/*--------------------------------------------*/
/*--------------------------------------------*/
FILE NAME : Tranbkwd.c
PROGRAMMER : Terri A. Smith
DATE WRITTEN : April 1990
ADDRESS : GTRI/CSITL Atlanta GA 30332 (404) 894-8952

PURPOSE: This procedure attempts to transfer one size from one
section into another section if feasible. It works from
the end of the section list to the start.

/*--------------------------------------------*/
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <stdlib.h>
#include <string.h>
#include "imptcl.h"
#include "impdec.h"

void transfer_backwards(i, j, l, cut_cost, unit_cost, max_sizes, max_ply)
{
int i;
int j;
int l;
int cut_cost;
int unit_cost;
int max_sizes;
int max_ply;

int k, m; /* counters */
int num_units; /* num_units in one section */

if (in_section[i].sizes[j] > 0) {
for (m=0; m<num_of_sizes; m++) {
sect3.sizes[m] = in_section[i].sizes[m];
sect4.sizes[m] = in_section[i].sizes[m];
}
sect3.sizes[j] = sect3.sizes[j] - 1;
sect4.sizes[j] = sect4.sizes[j] + 1;
portion.sizes[j] = 1;
portion.ply_height = in_section[i].ply_height;
temp_save.psect1 = i;
temp_save.psect2 = l;
temp_save.ptype = 0;
temp_save.orig_ply_height = 0;
temp_save.cand_ply_height = 0;
temp_save.savings = (float) 0.0;
compute_savings(i, l, cut_cost, unit_cost, max_sizes);
num_units = 0;
for (m=0; m<num_of_sizes; m++)
    num_units = num_units + sect4.sizes[m];

if ((temp_save.savings > save.savings) &&
    (num_units <= max_sizes) &&
    (temp_save.type > 0) &&
    (temp_save.cand.ply_height <= max.ply)) {
    memcpy(&save, &temp_save, sizeof(savings_t));
    for (m=0; m<num_of_sizes; m++) {
        if (temp_save.type == 1) {
            save.orig[m] = sect3.sizes[m];
            save.cand[m] = sect4.sizes[m];
            save.insect1[m] = in_section1.sizes[m];
            save.insect2[m] = in_section2.sizes[m];
        } else
            save.cand[m] = in_section1.sizes[m];
    }
}
return;
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "impdec.h"
#include "implcl.h"

void transfer_forward(i, j, l, cut_cost, unit_cost, max_sizes, max_ply)
{
    int i;
    int j;
    int l;
    int cut_cost;
    int unit_cost;
    int max_sizes;
    int max_ply;

    if (in_section[i].sizes[j] > 0) {
        for (m=0; m<num_of_sizes; m++) {
            sect3.sizes[m] = in_section[i].sizes[m];
            sect4.sizes[m] = in_section[i].sizes[m];
        }

        sect3.sizes[j] = sect3.sizes[j] - 1;
        sect4.sizes[j] = sect4.sizes[j] + 1;
        portion.sizes[j] = 1;
        portion.ply_height = in_section[i].ply_height;
        temp_save.sect1 = i;
        temp_save.sect2 = l;
        temp_save.type = 0;
        temp_save.org_ply_height = 0;
        temp_save.cand_ply_height = 0;
        temp_save.savings = (float) 0.0;
compute_savings(l, l, cut_cost, unit_cost, max_sizes);

num_units = 0;

for (m=0; m<num_of_sizes; m++)
    num_units = num_units + sect4.sizes[m];

if ((temp_save.savings > save.savings) &&
    (num_units <= max_sizes) &&
    (temp_save.type > 0) &&
    (temp_save.card_ply_height <= max_ply)) {
    memcpy(&save, &temp_save, sizeof(savings_t));

    for (m=0; m<num_of_sizes; m++) {
        if (temp_save.type != 1) {
            save.org[m] = sect3.sizes[m];
            save.cand[m] = sect4.sizes[m];
            save.in_sect1[m] = in_section1[i].sizes[m];
            save.in_sect2[m] = in_section2[i].sizes[m];
        }
        else
            save.cand[m] = in_section[i].sizes[m];
    } /* for m */
}
/* if */

return;