4.2 The First Selected Problem

with Best Solutions

Day 1: "Islands in the Sea"

The SEA is represented by an N times N grid. The task is to reconstruct a MAP of islands only from some CODED INFORMATION about the horizontal and vertical distribution of the islands. To illustrate this code, consider the following map:

```
*   * *       1 2
  * * *   *   3 1
*   *   *     1 1 1
  * * * * *   5
* *   *   *   2 1 1
      *       1

1 1 4 2 2 1
1 2   3   2
1
```

The numbers on the right of each row represent the order and size of the groups of islands in that rows. For example, "1 2" in the first row means that this row contains a group of one island followed by a group of two islands; with sea of arbitrary length to the left and right of each island group. Similarly, the sequence "1 1 1" below the first column means that this column contains three groups with one island each, etc.

PROBLEM STATEMENT

Implement a program which repeats the following steps until a given input file containing several information blocks has been read completely:

1. Read the next information block from an ASCII input file (for the data structure of that file see also the examples below) and display it on the screen. Each information block consists of the size of the square grid, followed by the row constraints and the column constraints. Each constraint for a single row or column appears on a single line as a sequence of numbers separated by spaces and terminated by 0.
2. Reconstruct the map (or all of the maps, if more then one solution is possible, see Example-4) and display it/them on the screen.
3. Write the map(s) to the end of an ASCII output file. Each blank must be represented by a pair of spaces. Each island should be represented by a '*' followed by a space. Different maps satisfying the same constraints should be separated by a blank line. If there is no map satisfying the constraints, indicate it by a line saying "no map". The

solutions to the different information blocks must be separated by a line saying "next problem".

TECHNICAL CONSTRAINTS

Constraint-1: N must be not less than 1 and not larger than 8.

Constraint-2: Put your solution program into an ASCII text file named

"C:\IOI\DAY-1\413-PROG.xxx".

Extension .xxx is:

```
- .BAS  for BASIC      programs,
- .C    for C          programs,
- .LCN  for LOGO       programs,
- .PAS  for Pascal     programs.
```

Constraint-3: The name of the ASCII input file for reading the coded information from must be

"C:\IOI\DAY-1\413-SEAS.IN".

Constraint-4: The name of the ASCII output file for writing the map(s) to must be
"C:\IOI\DAY-1\413-SEAS.OU".

EXAMPLES

Example-1 (the problem above):

```
6        6 is the size of the grid.
1 2 0    <-- Start of the first line constraint
3 1 0
1 1 1 0
5 0
2 1 1 0
1 0
1 1 1 0  <-- Start of the first colm constraint
1 2 0
4 0
2 3 0
2 0
1 2 0
```

Example-2. Solution:

```
4        columns:  1  2  3    4
0        row 1:
1 0      row 2:       *
2 0      row 3:    *  *
0        row 4:
0
1 0
2 0
0
```

Example-3.

```
2        Note that there is no map
```

```
0        satisfying the constraints.
0
2 0
2 0
```

Example-4.

Note that there are two different maps

```
1 0      satisfying the constraints.
1 0
1 0
1 0
```

SAMPLE FILES

We provided these correct example files for your convenience:

"C:\IOI\DAY-1\413-SEAS.IN" and
"C:\IOI\DAY-1\413-SEAS.OU".

WARNING: Successful execution of your program with these examples does not necessarily guarantee that your program is correct !!!

CREDITS

```
                                                 points
Read an information block from
the input file and display it                    05
Process all information blocks one by one
until the input file is read completely                  00
Reconstruct one map for each information
block (if it has a solution) and display it      35
Write the solution map to the output file        05
Reconstruct all possible maps (if there
are several solutions) and display them                  20
Write all solution maps correctly
separated to the output file                     10
Identify information blocks having no solution           05
Technical constraints completely obeyed                  10
-------------------------------------------------------------------
                                        maximal 100
```

One of the Best Solutions of Day 1

The problem can be recognized as simular to the well konwn two person game "Destroy Ships". The objects asked for are very similar but the information gaining is different.

The problem is solved using recursion and backtracking. The procedure find_maps assigns an island mark or an empty mark to the actual field and calls itself recursively in order to assign to the next field. If all fields could be assigned successfully according to the given constraints, the map will be shown and the recursion will be backtracked by restoring the old values. By

this exhausting recursion and backtracking process it is garanted that all solutions will be found.

Description of the program:

The algorithm is described in a semi formal manner:

```
read the data;
solve the problem.

read the data:
  for all rows
    read data and check its correctness.

solve the problem:
  initialize pointers to the the field;
  start with the first field;
  search map.

search map:
  if map complete
     draw map;
  if empty mark will fit to the current field
     assign empty mark;
     next field;
     search map;
     backtrack restoring the old value;
  if island mark will fit   to the current field
     assign island mark;
     next field;
     search map;
     backtrack restoring the old value.
```

Protokoll of a run:

```
 4
 2 0
 2 0
 2 0
 2 0
 2 0
 2 0
 2 0
 2 0

   1 2 3 4
 1     * *
 2     * *
 3 * *
 4 * *

   1 2 3 4
 1 * *
 2 * *
 3     * *
 4     * *
```

Program in Pascal

```
PROGRAM IOI92_Day_1_Islands_in_the_sea;
```

```
{ Author: Matej Ondrusek, CZ }
 USES crt;

 TYPE line=ARRAY[0..5] OF integer;
        { line of input file }
 VAR inp,out:text;     { input,output file }
     n:integer;{ number of rows/collums }
     row,col:ARRAY[1..8] OF line;
        { rows, columns information }
     rp,cp,rs,cs:ARRAY[1..8] OF integer;
        { r/c pointer, r/c start }
     map:ARRAY[1..8,1..8] OF integer;  { map of sea }
     nomap:boolean;    { no map can be found }
        { at least one map found }

 { Read one block of input file }

 PROCEDURE readinput;
  VAR i,j:integer;

  { If there is some mistake in input file ... }

  PROCEDURE eiif;
  BEGIN
   clrscr;
   writeln(#7,'Error in input file !');
   halt
  END;

  { Read one line of informations from input line }

  PROCEDURE readline(VAR a:line; VAR s:integer);
   VAR j:integer;
  BEGIN
   a[0]:=-1;
   s:=n+2;
   j:=1;
   IF eoln(inp) THEN eiif;
   read(inp,a[j]);
   WHILE a[j]>0 DO
    BEGIN
     IF j=5 THEN eiif;
     write(a[j],' ');
     s:=s-1-a[j];
     j:=j+1;
     IF eoln(inp) THEN eiif;
     read(inp,a[j]);
    END;
   writeln('0');
   readln(inp);
   IF s=n+2 THEN s:=n+1;
   IF a[j]<0 THEN eiif;
   IF s<1 THEN nomap:=true;
  END;

 BEGIN
  IF eoln(inp) THEN eiif;
  readln(inp,n); writeln(n);
  IF (n>8) OR (n<1) THEN
   BEGIN
    writeln('N is out of range !');
```

```
   halt;
  END;
 FOR i:=1 TO n DO readline(row[i],rs[i]);
 FOR i:=1 TO n DO readline(col[i],cs[i]);
 readln(inp);
 IF ioresult<>0 THEN eiif;
 {$i+}
 writeln; writeln('Press any key to continue.'); writeln;
 WHILE readkey=#0 DO;
END;

{ Write map to the screen and to the output file }

PROCEDURE write_map;
 VAR i,j:integer;
BEGIN
 IF found THEN writeln(out);
 found:=true;
 write('  ');
 FOR i:=1 TO n DO write(i,' '); writeln;
 FOR i:=1 TO n DO
  BEGIN
   write(i,' ');
   FOR j:=1 TO n DO
    IF map[i,j]=1
    THEN BEGIN write('* '); write(out,'* '); END
    ELSE BEGIN write('  '); write(out,'  '); END;
   writeln; writeln(out);
  END;
 writeln; writeln('Press any key to continue.');
 writeln;
 WHILE readkey=#0 DO;
END;

{ Recurrent procedure for reconstructing maps.
  It puts space or island on the r/c position
  (if possible) and calls itself for next position }

PROCEDURE find_maps(r,c:integer);
 VAR rc,cc:boolean;
BEGIN
 IF c=n+1 THEN
  BEGIN
   r:=r+1; c:=1;
  END;
 IF r=n+1 THEN
  BEGIN
   write_map;
   exit;
  END;

 { Try to put space on this position }

 IF ((row[r,rp[r]]=0) OR (row[r,rp[r]]=-1) AND
    (rs[r]>c)) AND
    ((col[c,cp[c]]=0) OR (col[c,cp[c]]=-1) AND
    (cs[c]>r)) THEN
  BEGIN
   map[r,c]:=0; rc:=false; cc:=false;
   IF row[r,rp[r]]=0 THEN
      BEGIN inc(rs[r]); rc:=true; row[r,rp[r]]:=-1
```

```
      END;
    IF col[c,cp[c]]=0 THEN
       BEGIN inc(cs[c]); cc:=true; col[c,cp[c]]:=-1
       END;
    find_maps(r,c+1);
    IF rc
    THEN BEGIN row[r,rp[r]]:=0; dec(rs[r]) END;
    IF cc
    THEN BEGIN col[c,cp[c]]:=0; dec(cs[c]) END;
   END;

  { Try to put island on this position }

  IF ((row[r,rp[r]]>0) OR (row[r,rp[r]]=-1) AND
     (rs[r]<=n)) AND
     ((col[c,cp[c]]>0) OR (col[c,cp[c]]=-1) AND
     (cs[c]<=n)) THEN
   BEGIN
    map[r,c]:=1; rc:=false; cc:=false;
    inc(rs[r]); inc(cs[c]);
    IF row[r,rp[r]]=-1
    THEN BEGIN rc:=true; inc(rp[r]) END;
    IF col[c,cp[c]]=-1
    THEN BEGIN cc:=true; inc(cp[c]) END;
    dec(row[r,rp[r]]); dec(col[c,cp[c]]);
    find_maps(r,c+1);
    inc(row[r,rp[r]]); inc(col[c,cp[c]]);
    IF rc THEN dec(rp[r]);
    IF cc THEN dec(cp[c]);
    dec(rs[r]); dec(cs[c]);
   END;

 END;

 { This is the main one-problem solving procedure }

 PROCEDURE solveproblem;
  VAR i:integer;

 BEGIN
  writeln(out,'next problem');
  FOR i:=1 TO n DO
   BEGIN
    rp[i]:=0; cp[i]:=0;
   END;
  found:=false;
  IF not nomap THEN find_maps(1,1);
  IF not found THEN
   BEGIN
    writeln('No map!'); writeln(out,'no map');
    writeln; writeln('Press any key to continue.');
    WHILE readkey=#0 DO;
   END;
 END;

BEGIN
 assign(inp,'c:\ioi\day-1\413-seas.in'); reset(inp);
 assign(out,'c:\ioi\day-1\413-seas.ou');
 rewrite(out);

 { This is the main repetition
```

```
   - UNTIL the input file is read completely }

 WHILE not eof(inp) DO
  BEGIN
   nomap:=false;
   clrscr;
   readinput;
   solveproblem;
  END;

 close(inp);
 close(out);

END.
```

Program in C

```
/************************************************

  Shawn Smith   7/15/92   Day 1  "Islands in the Sea"
************************************************/

#include<stdio.h>

#define SEASIN "413-SEAS.IN"
#define SEASOU "413-SEAS.OU"
#define bputc(ch) (putchar(ch),putc(ch,fileout))
#define bprintf(str)
 (printf(str),fprintf(fileout,str))
#define PAUSE()
 ( printf("hit any key for the next problem\n"),  \
                 getch() == 'q' ? exit(0) : 0)
#define placespace(xpos,ypos,xislpos,spaceleft) \
        (spaceleft == 0 || left[xpos][ypos] > 0 ? 0 :    \
            searchrow(xpos+1,ypos,xislpos,spaceleft-1))

int xisls[5][9],numxisls[9],yisls[5][8],
    numyisls[8],gridsize;
int curyisl[8][9],left[8][9],spaces[9];
int successcount;
FILE *filein,*fileout;

main()
{
  if((filein = fopen(SEASIN,"r")) == NULL)
    printf("error opening %s\n",SEASIN),exit(1);
  if((fileout = fopen(SEASOU,"w")) == NULL)
    printf("error opening %s\n",SEASOU),exit(1);

  while(readblock() == 0) {
    collectrowstats();
    initfirstrow();
    successcount = 0;
    searchrow(0,0,0,spaces[0]);
    if(successcount == 0)
      bprintf("no map\n");
    PAUSE();
  }

  fclose(fileout);
```

```
  fclose(filein);
}

readblock()
{
  int i,j,count;

  if(fscanf(filein,"%d",&gridsize) == EOF)
    return -1;
  bprintf("next problem\n");

  for(i=0; i < gridsize; i++) {
    fscanf(filein,"%d",&xisls[0][i]);
    for(count=0; xisls[count][i] > 0; count++)
      fscanf(filein,"%d",&xisls[count+1][i]);
    numxisls[i] = count;
  }
  numxisls[gridsize] = 0;
  for(i=0; i < gridsize; i++) {
    fscanf(filein,"%d",&yisls[0][i]);
    for(count=0; yisls[count][i] > 0; count++)
      fscanf(filein,"%d",&yisls[count+1][i]);
    numyisls[i] = count;
  }

  printf("grid size: %d\n",gridsize);
  for(i=0; i < gridsize; i++) {
    printf("row #%d:",i+1);
    for(j=0; j < numxisls[i]; j++)
      printf(" %d",xisls[j][i]);
    printf(" 0\n");
  }
  for(i=0; i < gridsize; i++) {
    printf("col #%d:",i+1);
    for(j=0; j < numyisls[i]; j++)
      printf(" %d",yisls[j][i]);
    printf(" 0\n");
  }
  return 0;
}

collectrowstats()
{
  int i,j,sum;

  for(i=0; i <= gridsize; i++) {
    for(sum=j=0; j < numxisls[i]; j++)
      sum += xisls[j][i];
    spaces[i] = gridsize-sum;
  }
}

initfirstrow()
{
  int i;

  for(i=0; i < gridsize; i++) {
    left[i][0] = -1;
    curyisl[i][0] = 0;
  }
}
```

```
searchrow(int xpos,int ypos,int xislpos,
          int spaceleft)
{

  if(xpos == gridsize && spaceleft > 0) return;
  else if(xpos == gridsize && ypos < gridsize) {
    propagaterow(ypos);
    searchrow(0,ypos+1,0,spaces[ypos+1]);
  }
  else if(xpos == gridsize) success();
  else if(xislpos == numxisls[ypos] ||
       left[xpos][ypos] == 0)
    placespace(xpos,ypos,xislpos,spaceleft);
  else if(left[xpos][ypos] > 0)
    placeisland(xpos,ypos,xislpos,spaceleft);
  else {
    placeisland(xpos,ypos,xislpos,spaceleft);
    placespace(xpos,ypos,xislpos,spaceleft);
  }
}

placeisland(int xpos,int ypos,int xislpos,
            int spaceleft)
{
  int i,changed = 0;

  for(i=0; i < xisls[xislpos][ypos]; i++,xpos++)
    if(xpos == gridsize || left[xpos][ypos] == 0 ||
       curyisl[xpos][ypos] >= numyisls[xpos])
      goto restore;
    else if(left[xpos][ypos] == -1) {
      left[xpos][ypos] =
        yisls[curyisl[xpos][ypos]][xpos];
      changed |= 1 << i;
    }
  if(xpos < gridsize)
    placespace(xpos,ypos,xislpos+1,spaceleft);
  else
    searchrow(xpos,ypos,xislpos+1,spaceleft);

restore:
  for(i--,xpos--; i >= 0; i--,xpos--)
    if((changed >> i) & 1)
      left[xpos][ypos] = -1;
}

/*
placespace(int xpos,int ypos,int xislpos,
           int spaceleft)
{
  if(spaceleft > 0 && left[xpos][ypos] <= 0)
    searchrow(xpos+1,ypos,xislpos,spaceleft-1);
}
*/

propagaterow(int ypos)
{
  int i;

  for(i=0; i < gridsize; i++) {
```

```
    left[i][ypos+1] = left[i][ypos] > 0 ?
    left[i][ypos]-1 : -1;
    curyisl[i][ypos+1] = curyisl[i][ypos] +
    (left[i][ypos] == 1 ? 1 : 0);
  }
}

success()
{
  int i,j;

  for(i=0; i < gridsize; i++)
    if(curyisl[i][gridsize] != numyisls[i])
      return;

  if(successcount > 0)
    bputc('\n');

  for(i=0; i < gridsize; i++) {
    for(j=0; j < gridsize; j++) {
      bputc(left[j][i] > 0 ? '*' : ' ');
      bputc(' ');
    }
    bputc('\n');
  }
  successcount++;
}
```

4.3 The Second Selected Problem

with Best Solutions

Day 2: "Climbing a Mountain"

A mountain climbers club has P members, numbered from 1 to P. Every member climbs at the same speed and there is no difference in speed between climbing up and down. Climber number i consumes C(i) units of SUPPLIES per day and can carry at most S(i) such units. All C(i) and S(i) are integer numbers.

Assume that a climber with a sufficient amount of supplies would need N days to reach the top of the mountain. The mountain may be too high, so that a single climber cannot carry all the necessary supplies. Therefore a GROUP of climbers starts at the same place and at the same time. A climber who descends prematurely before reaching the top gives his unneeded supplies to other climbers. The climbers do not rest during the expedition.

The PROBLEM is to plan a schedule for the climbing club. At least one climber must reach the top of the mountain and all climbers of the selected group return to the starting point.

PROBLEM STATEMENT

Implement a program which does the following:

1. Read from the keyboard the integer number N of days needed to arrive at the top, the number P of climbers in the club, and (for all i from 1 to P) the numbers S(i) and C(i). You may assume that the inputs are integers. Reject inputs that make no sense.
2. Try to find a schedule for climbing the mountain. Determine a possible group a(1), ..., a(k) of climbers who should participate in the party and (for all j from 1 to k) the number M(j) of supplies which climber a(j) carries at the start. Note that there may not exist a schedule for all combinations of N and the S(i) and C(i).
3. Output the following information on the screen:
    - a) the number k of climbers actually participating in the party,
    - b) the total amount of supplies needed,
    - c) the climber numbers a(1), .., a(k),
    - d) for all a(j), j between 1 and k, the initial amount M(j) of supplies to carry for climber a(j),
    - e) the day D(j) when climber a(j) starts descending.
4. A schedule is OPTIMAL if
    - a) the number of participating climbers is minimal and
    - b) among all groups satisfying condition a) the total of consumed supplies is minimal. Try to find a nearly optimal schedule.

TECHNICAL CONSTRAINTS

Constraint-1: Put your solution program into an ASCII text file named

"C:\IOI\DAY-2\422-PROG.xxx".

Extension .xxx is:

```
- .BAS  for BASIC      programs,
- .C    for C          programs,
- .LCN  for LOGO       programs,
- .PAS  for Pascal     programs.
```

Constraint-2: Programs must reject inputs where N is less than 1 or greater than 100. P must be not less than 1 and not greater than 20.

EXAMPLE(S)

The following could be a dialogue with your program:

```
Days to arrive to top:  4
Number of club members: 5
Maximal supply for climber 1 : 7
Daily consumption for climber 1 : 1
Maximal supply for climber 2 : 8
Daily consumption for climber 2 : 2
Maximal supply for climber 3 : 12
```

```
Daily consumption for climber 3 : 2
Maximal supply for climber 4 : 15
Daily consumption for climber 4 : 3
Maximal supply for climber 5 : 7
Daily consumption for climber 5 : 1

2 climbers needed,
total amount of supplies is 10.
Climber(s) 1, 5 will go.
Climber 1 carries 7 and descends after 4 day(s)
Climber 5 carries 3 and descends after 1 day(s)

Plan another party (Y/N) Y

Days to arrive to top:  2
Number of club members: 1
Maximal supply for climber 1 : 3
Daily consumption for climber 1 : 1
Climbing party impossible.
Plan another party (Y/N) N

Good bye
```

SAMPLE FILES

For your convenience, some files containing test data and correct sample output have been prepared; please look into the directory "C:\IOI\DAY-2". WARNING: Successful execution of your program with these examples does not necessarily guarantee that your program is correct !!!

CREDITS

	points
User dialogue as illustrated above	10
Find a solution for the special case where all C(i)=1 and and all S(i) are equal	20
Find a solution for general case	20
Find a nearly optimal solution for general case	30
Detect unsolvable situations	10
Technical constraints obeyed	10
	maximal 100

One of the Best Solutions of Day 2

The problem remembers to the task to prepare a timeschedule for a school or similar coordination problems.

The problem will be solved using a backtracking technique.

During input of data the climbers are classified by daily consumption, so that two climbers of the same daily consumption belong to the same group. The number of such groups is kept in the variable "g". Each group is sorted by the carrying facility of the group members. By this way a tree structure is produced.

In order to solve the problem each group sends the first climber. In a recursive search using the procedure findnextclimber it is computed how far the selected climber will come and how much additional food units he needs. If the goal is not yet reached the next climber of the team is selected by calling the procedure findnextclimber indirect recursively via the procedure useclimbergroup.

Description of the program:

The algorithm is described in a semi formal manner:

```
read the data;
solve the problem.

read the data:
  read data;
  classify the climbers;
  sort by daily consumption within the classes.

solve the problem:
  initialize;
  for i from 1 to g use climber group (i);
  output the best result.

  use climber group:
    find next climber.

    find next climber:
      compute the number of additional food units needed;
      if the problem is totaly solved
         then actualize the best solution;
         else use climber group.
```

Protocoll of the run:

```
Days to arrive to top: 4
Number of club members: 5
Maximal supply for climber 1: 7
Daily consumption for climber 1: 1
Maximal supply for climber 2: 8
Daily consumption for climber 2: 2
Maximal supply for climber 3: 12
Daily consumption for climber 3: 2
Maximal supply for climber 4: 15
Daily consumption for climber 4: 3
Maximal supply for climber 5: 7
Daily consumption for climber 5: 1

2 climbers needed,
total amount of supplies is 10.
Climber(s) 1, 5 will go.
Climber 5 carries 7 and descends after 4 day(s)
Climber 1 carries 3 and descends after 1 day(s)

Plan another party (Y/N) Y

Days to arrive to top: 2
Number of club members: 1
Maximal supply for climber 1: 3
```

```
       Daily consumption for climber 1: 1
       Climbing party impossible.

       Plan another party (Y/N) N

       Good bye
```

Program in Pascal

```
PROGRAM IOI92_2_climbing_party;
{ Author: Matej Ondrusek, CZ }

 USES crt;

 CONST maxp=20;{ max. number of members }
       maxn=100;       { max. days }

 VAR  z:char;
      correct:boolean;
      n,p:integer;     { days, members }
      s,c:array[0..maxp] of integer;
        { max supplies, consumation }
      cg,cp:array[1..maxp] of integer;
        { cons.groups, cg pointers }
      party,bparty:array[1..maxp] of integer;
        { actual/best return's days }
      ds:array[0..maxn] of integer;
        { left supplies on days }
      dp:integer;      { days pointer }
      supplies,bsupplies:integer;
        { used/best party use supplies}
      climbers,bclimbers:integer;
        { actual/best number of clim. }
      msupplies:integer;       { minus supplies }
      g:integer;       { number of groups }
      i:integer;

 { ********************************************** }
 { ****** This procedure is the input dialog ****** }
 { ********************************************** }

 PROCEDURE readdata;
  VAR i,j,l:integer;
 BEGIN
  {$i-}
  REPEAT
   write('Days to arrive to top: '); readln(n);
   IF ioresult<>0 THEN exit;
   IF (n<1) OR (n>100) THEN
    BEGIN
     writeln(' Oops! n, is out of range (1..100) !');
     writeln(' Maybe, next time it''ll be OK ...');
     writeln;
    END;
  UNTIL (n>=1) AND (n<=100);

  REPEAT
   write('Number of club members: '); readln(p);
   IF ioresult<>0 THEN exit;
   IF (p<1) THEN
```

```
  BEGIN
   writeln(' Your club doesnt have many members.');
   writeln;
  END;
 IF (p>20) THEN
  BEGIN
   writeln('  So many climbers !');
   writeln('  It''s nice, but not allowed.');
   writeln;
  END;
UNTIL (p>=1) AND (p<=20);

g:=0;
FOR i:=1 TO p DO
 BEGIN
  REPEAT
   write('Maximal supply FOR climber ',i,' : ');
   readln(s[i]);
   IF ioresult<>0 THEN exit;
   IF (s[i]<0) THEN
    BEGIN
     writeln('  I think, it cannot be negative.');
     writeln;
    END;
   IF (s[i]=0) THEN
    BEGIN
     writeln('  It''s strange, but OK.');
     writeln('  Maybe, it''s only a child ...');
    END;
  UNTIL s[i]>=0;

  REPEAT
   write('Daily consumption for climber ',i,' : ');
   readln(c[i]);
   IF ioresult<>0 THEN exit;
   IF (c[i]<1) THEN
    BEGIN
     writeln(' climber should eat at least 1.');
     writeln;
    END;
  UNTIL c[i]>=1;

  j:=1; WHILE (j<=g)AND(c[cp[j]]<>c[i]) DO j:=j+1;

  IF j>g THEN
   BEGIN
    g:=g+1;
    cp[j]:=i;
    cg[i]:=0;
   END
  ELSE IF s[i]>=s[cp[j]] THEN
   BEGIN cg[i]:=cp[j]; cp[j]:=i END
  ELSE BEGIN
   l:=cp[j];
   WHILE s[cg[l]]>s[i] DO l:=cg[l];
   cg[i]:=cg[l];
   cg[l]:=i;
  END;
 END;
{$i+}
correct:=true;
```

```
END;

{ ********************************************** }
{ ***** This is only forward declaration ... ***** }
{ ********************************************** }

PROCEDURE useclimbergroup(g:integer); FORWARD;

{ ********************************************** }
{ * This procedure finds next day to be computed * }
{ * and then it tries to use all climbers        * }
{ * on the top of groups.                        * }
{ ********************************************** }

PROCEDURE findnextclimber;
 VAR i,dps:integer;
BEGIN
 dps:=dp;
 WHILE (dp>0) AND (ds[dp]<=0) DO
  BEGIN
   ds[dp-1]:=ds[dp-1]+ds[dp];
   dp:=dp-1;
  END;
 IF dp=0 THEN
  BEGIN
   IF (climbers<bclimbers) OR (climbers=bclimbers)
      AND (supplies<bsupplies)
    THEN BEGIN
     bclimbers:=climbers;
     bsupplies:=supplies;
     msupplies:=ds[0];
     bparty:=party;
    END;
   END
  ELSE
   FOR i:=1 TO g DO
    IF cp[i]>0 THEN useclimbergroup(i);
  WHILE dp<dps DO
   BEGIN
    ds[dp]:=ds[dp]-ds[dp+1];
    dp:=dp+1;
   END;
END;

{ ********************************************** }
{ * This procedure puts the best climber         * }
{ * from group g into expedition.                * }
{ ********************************************** }

PROCEDURE useclimbergroup(g:integer);
 VAR cn:integer;
BEGIN
 cn:=cp[g];
 IF (dp+1)*c[cn]<=s[cn] THEN
  BEGIN
   cp[g]:=cg[cn];
   party[cn]:=dp;
   climbers:=climbers+1;
   supplies:=supplies+2*dp*c[cn];
   ds[dp]:=ds[dp]-s[cn]+(dp+1)*c[cn];
   FOR i:=1 TO dp-1 DO ds[i]:=ds[i]+c[cn];
```

```
    findnextclimber;

    FOR i:=1 TO dp-1 DO ds[i]:=ds[i]-c[cn];
    ds[dp]:=ds[dp]+s[cn]-(dp+1)*c[cn];
    supplies:=supplies-2*dp*c[cn];
    climbers:=climbers-1;
    party[cn]:=0;
    cp[g]:=cn;
   END;
 END;

 { ********************************************** }
 { * This is the main one-problem solving procedure }
 { ********************************************** }

 PROCEDURE solveproblem;
  VAR i,j:integer;
 BEGIN

  bclimbers:=maxp+1;
  climbers:=0; supplies:=0;
  FOR i:=0 TO n DO ds[i]:=0;
  dp:=n;
  FOR i:=1 TO p DO party[i]:=0;

  FOR i:=1 TO g DO useclimbergroup(i);

  writeln;
  IF bclimbers=maxp+1 THEN
   BEGIN
    writeln('Climbing party impossible.');
    exit;
   END;
  j:=1; WHILE bparty[j]=0 DO j:=j+1;
  FOR i:=j+1 TO p DO
   IF (bparty[i]>0) AND (bparty[i]<bparty[j])
    THEN j:=i;
  write  (bclimbers);
  write  (' climbers needed,');
  write  (' total amount of supplies is ');
  writeln(bsupplies,'.');
  write('climber(s)');
  FOR i:=1 TO p DO
   IF bparty[i]>0 THEN write(' ',i,',');
  writeln(#8,' will go.');
  FOR i:=1 TO p DO
   IF (i<>j) AND (bparty[i]>0) THEN writeln
    ('Climber ',i,' carries ',s[i],
     ' and descends after ',bparty[i],' days');
  write  ('Climber ',j,' carries ',s[j]+msupplies);
  writeln(' and descends after ',
          bparty[j],' day(s)');
 END;

BEGIN
 s[0]:=0; c[0]:=0;
 clrscr;
 REPEAT
  correct:=false;
  REPEAT
```

```
   readdata;
   IF not correct THEN
    BEGIN
     writeln('  Only integers, please ?!?');
     writeln;
    END;
  UNTIL correct;
  solveproblem;
  REPEAT
   writeln; write('Plan another party (Y/N) ');
   readln(z);
  UNTIL z in ['y','n','Y','N'];
 UNTIL (z='n') OR (z='N');
 writeln;
 writeln('Good bye');
 writeln;
END.
```

Program in C

```
/*****************************************************
  Shawn Smith   7/17/92   IOI "Climbing a Mountain"
*****************************************************/

#include<stdio.h>
#include<search.h>
#include<math.h>

#define divup(a,b)
 ((int)ceil((double)(a)/(double)(b)))

struct HIKE {
  int number,carries,descends,C,S;
  /* I swapped C and S by accident */
} hike[20],best[20];

int timeabort;
int bestP,bestC,curP,curC,
    index[20],unused[20],maxindex;
int Ndays,Pclimbers;
FILE *filein;

main()
{
  int yn;

  do {
    readinfo();
    findunique();
    resetbest();
    planperson(Ndays,0,0,0,0);
    printbest();

    printf("\nPlan another party (Y/N)? ");
    while(kbhit()) getch();
    yn = getch();
  } while(yn == 'y' || yn == 'Y');
}

readinfo()
```

```
{
  int i;

  printf("\n\nDays to arrive to top: ");
  scanf("%d",&Ndays);
  if(Ndays < 1 || Ndays > 100)
    printf("Error: Invalid number of days."),exit(0);

  printf("Number of club members: ");
  scanf("%d",&Pclimbers);
  if(Pclimbers < 1 || Pclimbers > 20)
    printf("Error: Invalid number of climbers."),
    exit(0);

  for(i=0; i < Pclimbers; i++) {
    hike[i].number = i+1;

    printf("Maximal supply for climber %2d:    ",
     hike[i].number);
    scanf("%d",&hike[i].C);
    if(hike[i].C < 1)
      printf("Error: Supply must be greater 0."),
       exit(0);

    printf("Daily consumption for climber %2d: ",
     hike[i].number);
    scanf("%d",&hike[i].S);
    if(hike[i].S < 1)
      printf("Error: Consumption must greater 0."),
       exit(0);
  }
  printf("This may run for a while.
   If time is running out, hit any key.\n");
  timeabort = 0;
}

int sortorder(const void *A,const void *B)
{
  struct HIKE *a,*b;

  a = (struct HIKE *)A;
  b = (struct HIKE *)B;
  if(a->S == b->S)
    if(b->C == a->C)
      return a->number -b->number;
    else
      return b->C -a->C;
  else
    return a->S -b->S;
}

findunique()
{
  int i;

  qsort(hike,Pclimbers,sizeof(struct HIKE),
   sortorder);
  maxindex=1;
  index[0]=0;
  unused[0]=1;
  for(i=1; i < Pclimbers; i++)
```

```
    if(hike[i].S == hike[i-1].S ||
      hike[i].C <= hike[i-1].C)
      unused[maxindex-1]++;
    else {
      index[maxindex] = i;
      unused[maxindex] = 1;
      maxindex++;
    }
}

resetbest()
{
  int i;

  for(i=0; i < Pclimbers; i++) {
    best[i].number = i+1;
    best[i].carries = best[i].descends = 0;
  }
  bestP=Pclimbers+1;
}

planperson(int days,int rate,int provide,int curP,
  int curC)
{
  int i,j,needs,makeup;

  if(curP >= bestP) return;
                          /* trim the search tree */

  for(j=0; j < maxindex; j++)
    if(unused[j] > 0) {
      i = index[j];
      if(hike[i].C-hike[i].S*days >= 0) {
        hike[i].descends = days;
        needs = hike[i].S*days*2+provide;
        makeup = needs-hike[i].C;

        if(makeup > 0) {
          hike[i].carries = hike[i].C;
          index[j]++;
          unused[j]--;
          planperson(divup(makeup,hike[i].S+rate),
     hike[i].S+rate,makeup, curP+1,curC+hike[i].C);
          index[j]--;
          unused[j]++;
        }
        else {
          hike[i].carries = needs;
          checkbest(curP+1,curC+needs);
        }
        hike[i].descends = 0;
        hike[i].carries = 0;
      }
    }
}

checkbest(int curP,int curC)
{
  if(curP < bestP || (curP == bestP && curC < bestC))
  {
    bestP = curP;
```

```
    bestC = curC;
    memcpy(best,hike,Pclimbers*sizeof(struct HIKE));
  }
  printf(".");
  while(kbhit())
    bestP = 0,getch();
}

int bestorder(const void *A,const void *B)
{
  struct HIKE *a,*b;

  a = (struct HIKE *)A;
  b = (struct HIKE *)B;
  if(a->descends)
    if(b->descends)    return a->number -b->number;
    else               return -1;
  else if(b->descends) return 1;
  else                 return 0;
}

printbest()
{
  int i,first;

  qsort(best,Pclimbers,sizeof(struct HIKE),bestorder)
  ;
  for(bestP=i=0; i < Pclimbers; i++)
    if(best[i].descends)
      bestP++;

  if(bestP == 0) {
    printf("\nClimbing party impossible.\n");
    return;
  }

  printf("\n%d climber%s needed,
    total amount of supplies is %d.\n",
    bestP,bestP > 1 ? "s" : "",bestC);
  printf("Climber%s",bestP > 1 ? "s" : "");
  for(i=0,first=1; i < bestP; i++)
    printf("%s %d",first ? first=0,"" : ",",
      best[i].number);
  printf(" will go.\n");
  for(i=0; i < bestP; i++)
    printf("Climber %d carries %d and descends
      after %d day(s).\n", best[i].number,
      best[i].carries,best[i].descends);
}
```