QUESTIONNAIRE DE LA FOLLE COURSE INFORMATIQUE THE MAD PROGRAMMING RACE PROBLEMS

UNIVERSITÉ DE SHERBROOKE
Département de génie électrique et de génie informatique
Facultéde génie

ÉCOLE POLYTECHNIQUE MONTRÉAL
Département de génie électrique et de génie informatique

UNIVERSITÉ BISHOP'S UNIVERSITY
Computer Science Department

Novembre 1999

A few words on the Mad Programming Race

This already the 5th edition of the "La Folle Course Informatique" or "The Mad Programming Race". By way of a short story, the first competition was held in Sherbrooke and organized by the Department of Electrical and Software engineering. The 2nd, 3rd and 4th edition involved the participation of the École Polytechnique de Montréal which held the competition on its campus while it was run simultaneously in Sherbrooke. This year will be associated with " Le prix AbacUS " and a bilingual questionnaire thanks to the collaboration of Bishop's University. This also explains why our name is now bilingual.

We want to thank Jean-Marie Dirand (professor at the Université de Sherbrooke) for having read the questionnaire and making many pertinent remarks. Others who took part in the drafting of the questionnaire and in the writing of the programs must be also thanked. Here is the list in alphabetical order: Alex Boisvert, Paule Bolduc, Jean-Denis Boyer, Paul-André Chassé, Nicolas Courtemanche, Guillaume Coté, Rusty Deschênes, Jean-Yves Hervé, Nelly Khouzam, Roch Lefebvre, Benoît Parent, Nicolas Pelletier, Ginette Roy.

Martine Bellaïche, École Polytechnique de Montréal.
Gaétan Haché, Bishops University.
Ruben Gonzalez-Rubio, Université de Sherbrooke.

Introduction

Read this preamble with the questionnaire of the fifth edition of the *Mad Programming Race*. It contains a summary of the rules and various instructions related to the handing-over of the programs.

The Mad Programming Race

The Mad Programming Race is a programming competition. The participating teams must write programs according to given specifications. Each program that successfully runs earns points for the team and the winning team is the one that accumulates the most points.

For each problem, you must conceive a C, C++ or Java program [1] that respects the specifications. This program will be tested with one or more testing files, a number of points will be granted to you according to the number of files correctly treated by your program. The number of tests and points corresponding is indicated at the end of each problem. The same sets of testing files be used to evaluate the programs of all the teams.

New rules for bonus points For a particular problem, bonus points will be given to teams whose programs will successfully pass all the testing files. The amount of the bonus will be determined according to the difficulty of the problem and the time of submission.

The following table indicates how the bonus points will be granted:

Problem number	Bonus at t_0	Bonus at the end of the race
1, 2	30 %	0 %
3, 4, 5, 6, 7, 8	40 %	0 %
9, 10	50 %	0 %

where t_0 is the starting time of the competition and t_f is the end of the competition.

Hence, bonus points decreases in a linear way, as time advances. For example, if after four hours of competition a team hands-over the program for question 10 and that it passes all the tests, it has a bonus of 25 %. If a team hands-over the program for question 10 after six hours of competition and it passes all the tests, it earns a bonus of 12,5 %.

[1] In Sherbrooke we chose these three languages, however, the corrector may allow other compiled languages to be used. If your local organization chooses to use a different language, it is necessary then to consult the particular procedures for the handing-over of programs. In the questionnaire, we refer only to the C, C++ and Java languages.

The bonus is a percentage calculated according to the following formula:

$$B = p \times \frac{t_r}{t_t}$$

where, B is the percentage bonus to be granted, p is the starting percentage at t_0, t_r is time remaining when the program is delivered and t_t is the total time of the race. The granularity of time is that of the marker and is estimated at two seconds.

In a sentence, the quicker the program is handed-over, the larger is the bonus, of course provided that the program passes all the tests.

In eight hours, it is not very probable but not impossible that a team has sufficient time to program all the problems presented. You will have to show judgement by choosing the problems you try to solve.

We tried to present the problems in a uniform way and without ambiguity. Each statement comprises a description and specification of the problem to be solved, as well as examples and files that could be needed for the problem.

At the end of the race, the classification of the teams will be established according to the total of the points accumulated for each handed-over program. In the possibility where two teams would obtain an equal number of points, the first to reach that number of points will be the winning team.

Handing-over programs

You must hand-over only one source file for each problem. This file will have the extension "`.C`" if it must be compiled in C, "`.CPP`" if it is in C++. The first part of the name will be made up of the number of the problem followed by the number of your team according to the format "`P##_EQ##`". For example, problem 3 of team 9, coded in C++, would bear the name "`P03_EQ09.CPP`" and "`P03_EQ15.CPP`" for the same problem of team 15. In Java, and for problem 3 team 9, the source file "`p03_eq09.java`" must contains a class named "`p03_eq09`" which must contains the `main`. Be careful to write the class name in lower cases since Java is case sensitive: this is very important for the correction. Since you must submit only one file (exactly one for each problem), if you want to define more than one class they must be non-public or inner-class. The name of the file comprises exactly 8 characters before the point. **Files which do not conform to this format will not be considered for correction**.

Each handed-over file will be compiled in order to produce an object program. This program will be run several times with the testing files. The output files will be analyzed in order to check whether they are in conformity with the specifications, and the points will be granted accordingly. A program must compile without any error (warnings will be tolerated). A program which does not compile will not thus be given any point. With the correction, the outputs on the console (like `printf` or others) are also tolerated, but the execution time increases quickly. It is thus advised to avoid them in order not to exceed allocated time. Note that your source code will not be examined so you have total freedom on the programming style you will use.

Note that the correction is automated and is carried out during the race in real time. Unless there is a technical problem with the correction system, you will be able to consult your results on a monitor and this a few moments after you handed-over your program.

The inputs and outputs are always done via ASCII text files. Those will be named according to the number of the problem with the format "`P##.ENT`" for input files and "`P##.SOR`" for output files. The `##` indicates the number of the problem, it varies between `01` and `10`. Their contents and how to use them is clearly defined in the statement of each problem. Moreover, one example is presented for each one of these files.

Take for granted that the input files which will be used to test your programs will follow rigorously the format which is indicated in each problem. The examples of files shown in the text will be provided to you.

It is crucial that the files produced by your programs respect the specifications rigorously since they will be automatically corrected.

When you hand-over your program for its evaluation, you have to copy it in a " deposit box " which will be indicated to you at the time of the race as well as the exact procedure. You will be able to deposit only once your solution to each problem. Once a program is handed-over, no modification will be accepted. If you deposit your program again, the new copy will be ignored.

The rules

- Only one computer will be assigned to each team. A team can use only the computer which is assigned to it. In the event of a breakdown, the team has to wait until an organizer assigns a new computer.
- A study room near the computer room will be available to the teams.
- You have the right to bring and to use any relevant documentation, as long as it is printed or hand-written. Any material support (other then documents) is prohibited during the race, including portable diskettes and computers. **A team bringing such unauthorized material will be automatically disqualified.**
- The local area network will be cut from the external world and therefore Internet will not be usable.
- In order to avoid annoying accidents, any food or drink will be prohibited in the computer rooms.
- We count on the honesty and the good faith of the participants.

Final Remarks

The Mad Programming Race is organized by a team of voluntary members, which is renewed with each edition. We endeavor to write specifications as clearly as possible.

We wrote programs to the specifications of the problems, we wrote the sets of testing files and program correctors by devoting much effort and time. Even during the race, we check in order to make sure that all is fine and that everything occurs in an equitable way. However, we do not claim

perfection! For this reason, we ask you to call upon your "computer-sportsmanship" spirit in order to accept the "official" classifications given at the end of the race. Indeed, it is practically impossible to change the distribution of the prices if changes in the classification occurred. We think that the greatest reward associated with this competition is satisfaction to have made an effort to write programs and hopefully to have learned something. However, we are open to remarks which could improve the future competitions or which inform us with an error.

The "Grand Prix AbacUS" will be allotted after a few days to make sure that all the possible checks were made.

Conventions used in the questionnaire

To indicate the beginning and the end of a file we use the symbols ▷ and ◁ respectively. Of course these symbols do not form part of the file.

For example, the following file contains only one line with the chain `hello`.

▷

```
hello
```

◁

Contents

The Specifications

Specification 1

The detective

Paul-André Chassé

`File to be produced containing your source code:  P01_EQ##.*`

Most of all children dreamt one day of becoming a detective like Poirot, Sherlock Holmes or Colombo. The game *Clue* of *PARKER BROTHER* makes it possible for anyone to test their logical deduction capacity. The rules of this game are so simple that it is often believed that it is by chance that one wins the game. That is certainly not always the case, for it often happens that one of the players seems to have a particular gift to find the solution after only a few guesses. If you are such a player, this problem is for you.

The rules

The game is played between 3 and 6 players.

The goal of the game is to solve the enigma of the assassination of the owner of a castle whose body was found in one of the rooms of his castle. To win the game, it is necessary to answer three questions:

1. Who is the culprit?
2. With which weapon was the crime committed?
3. In which room did the crime take place?

The original game comprises the following accessories:

- a small playing board illustrating the nine rooms of the castle,

- a die,
- six pawns of colors representing the six suspects,
- six miniature weapons and
- a deck of 21 cards which contains:
  - one card for each of the six suspects,
  - one card for each of the six weapons,
  - one card for each room of the castle.

The cards are only used to represent the suspects, the weapons and the rooms of the castle and do not have any numerical value.

The preparations for the game are as followed: If it is not already done, one of the players, the dealer, groups the cards in three smaller decks, one deck for each of the following:

- suspect cards,
- weapon cards,
- room cards.

Each of the three decks is shuffled and placed face down on the table. Then, without looking at it, the dealer takes a card on each of the three decks and puts the three of them in an envelope which is put in the center of the playing board. This envelope contains the solution which the players will try to discover, i.e. the culprit, the weapon and the room. The remaining cards of the three decks are mixed together then distributed to the players, one by one, starting at the dealer's left and continuing clockwise until all the cards are given.

The pawns are put on their starting box on the playing board and each player chooses one of the pawns by taking the one which is closest to him. Usually the player who has the red pawn begins the game.

The game proceeds as follows:

The players play in turns, the first one to play is the one with the red pawn. At each one of his turns, a player tries to bring his pawn in one of the rooms by moving it the number of boxes corresponding to the number (1 to 6) he obtained by throwing the die. If the number obtained by the die is sufficiently high to make it possible to enter one of the rooms, the player can choose to enter this room and make a guess. The goal of the guess is to determine by elimination which cards are in the envelope. The guess must comprise exactly one of the suspects, exactly one of the weapons and exactly the room in which the pawn of the player is. The suspect and the weapon are the choice of the player. The player makes a verbal announce of his guess to the other players.

When a guess is made, the player immediately to the left of the player that made the guess must check if he has one of the cards named in the guess. If so, he must show it only to the player who made the guess. That proves that the guess was wrong since the shown card is not in the envelope. The player who made the guess then notes the card in his detective notebook and the game continues.

If the player to the left of the one that makes a guess cannot prove that the assumption is false, i.e. he does not have any the cards named in the guess, then the following player will check if he has one of the cards named in the guess and so on. Note that even if a player has more than one card named in the guess, he shows only one of them, one of his choice, to the player who made the guess.

If none of the other players can prove that the guess is false, the player who made the guess can either finish his turn or carry a charge. A player carries a charge when he believes that he knows which cards are in the envelope. A player carries a charge by naming the three cards he believes to be in the envelope. Then he looks at the cards in the envelope making sure that nobody else than him can see them. If the cards in the envelope are those that he named, he then shows them to everyone and wins the game. If he was mistaken, he puts the cards back in the envelope and the game continues with the other players. The player who makes a false charge loses the game but his cards are still available to the other players as if he was still playing.

When a player believes he has guessed the solution, he must await his turn to play to carrying charge. Note that it is not necessary to be in a room nor to make a guess as a preliminary to carry a charge. It is also allowed to make a guess and on the same turn carry a charge, even if the guess is wrong.

In order to obtain information more quickly or simply to mislead his adversaries, a player who makes a guess can name one or more cards he has in his hand.

Problem

We put in a file the key elements of the game outcome *Clue* as seen by the winning player. We omitted just a small detail, the final charge.

From the information contained in the input file, you must write a program which will find the three cards of the final charge.

For copyrights reasons, we changed the name of the characters and the intrigue of the play. The rules of the game remain however the same. The goal of our version of the game consists in finding:

1. Who damaged the Nissan Maxima of the vice-chancellor of the University of Sherbrooke[1]?
2. Which kind of vehicle the culprit was driving?
3. In which parking lot the offense took place?

In our game the possible suspects are:

Suspect	Code
The senior of the FLSH	1
A bus driver	2
A former vice-rector	3
The Minister of education	4
A poor student	5
A rich student	6

The possible vehicles are:

[1] This is a game, the characters and the components are fictitious and without any connection with reality.

Vehicle	Code
a minibus from the CMTS	7
A blue limousine	8
A 1976 CCM Targa bicycle	9
A red Jeep	10
A red Honda Civic CRX	11
A blue Hyundai	12

The possible parking lots are:

Parking lot	Code
Green parking lot of the Central House	13
Parking lot of the Sporting Center	14
Parking lot of the FLSH	15
Parking lot of the Faculty of Administration	16
Parking lot of J.S. Bourque building	17
Parking lot of the Carrefour de l'Estrie	18
Parking lot of Cinema 9	19
Parking lot of the St-Hubert Restaurant	20
Parking lot on Wellington south	21

In order to simplify the inputs/outputs of your program, we will use the numeric codes indicated in the second column of the preceding tables in order to identify the suspects, the vehicles and the parking lots.

The players will be also represented by whole numbers. The player having the red pawn, which for this problem will be also the dealer, is number 1. The player on his left is number 2 and so on.

Here in detail the outcome of the game corresponding to the input file given in the example of the next section:

1. The game was played by three players and your program will have all the information available to player #3 who won the game.
2. After the cards of the culprit, his vehicle and parking lot were put in the envelope, the 18 remaining cards were distributed.
3. Player #3 received the 6 following cards:
   (a) The senior of the FLSH
   (b) a poor student
   (c) green parking lot of the central house
   (d) parking lot of the sporting center
   (e) parking lot of the Carrefour de l'Estrie
   (f) Parking lot of the St-Hubert restaurant
4. Player #1 throws the die and succeeds in entering the parking lot of the St-Hubert restaurant. He declares:

> I suspect the minister of education of having damaged the Nissan Maxima of the vice-chancellor while driving the blue Hyundai in the parking lot of the St-Hubert restaurant.

5. Player #2 looks at his cards and since he has at least one of the cards related to the guess made by player #1, player #2 chooses one of these cards and shows it to player #1. Player #1's guess being false, he chooses to pass his turn.

6. Player #2 throws the die and succeeds in entering the parking lot of the J.S. Bourque building. He declares:

> I suspect the bus driver of having damaged the Nissan Maxima of the vice-chancellor while driving a red Jeep in the parking lot of the J.S. Bourque building.

7. Player #3 looks at his cards and says: I do not have anything to prove that this guess is false. Player #1 looks at his cards and says: I do not have anything to prove that this guess is false.

8. Although no other player could prove that the guess was false, player #2 chooses to pass his turn without carrying a charge.

9. Player #3 throws the die and succeeds in entering the parking lot of the St-Hubert restaurant. He declares:

> I suspect the rich student of having damaged the Nissan Maxima of the vice-chancellor while driving a red Jeep in the parking lot of the St-Hubert restaurant.

10. Player #1 looks at his cards and says: I do not have anything to prove that this guess is false. Player #2 looks at his cards and says: I do not have anything to prove that this guess is false.

11. Although no other player could prove that the guess was false player #3 chooses to pass his turn without carrying a charge.

12. Player #1 throws the die and succeeds in entering the parking lot of Cinema 9. He declares:

> I suspect the senior of the FLSH of having damaged the Nissan Maxima of the vice-chancellor while driving a blue Hyundai in the parking lot of Cinema 9.

13. Player #2 looks at his cards and says: I do not have anything to prove that this guess is false. Player #3 looks at his cards and since he has at least one of the cards he shows one to player #1. His guess being false, player #1 chooses to pass his turn.

14. Player #2 throws the die but does not obtain a sufficient result to enter another parking lot. His turn finishes without him being able to make a guess.

15. Player #3 throws the die and succeeds in entering the parking lot of Cinema 9. He declares:

> I suspect the minister of education of having damaged the Nissan Maxima of the vice-chancellor while driving a 1976 CCM Targa bicycle in the parking lot of Cinema 9.

16. Player #1 looks at his cards and says: I do not have anything to prove that this guess is false. Player #2 looks at his cards and since he has at least one of the cards he shows one to player #3. The card shown by player #2 is the one of the Minister of education.

17. Although his guess is wrong, player #3 has accumulated sufficient information to carry a charge. He says then:

18. I accuse the rich student to have damaged the Nissan Maxima of the vice-chancellor while driving a red Jeep in the parking lot of Cinema 9.

19. Player #3 opens the envelope and finds out that his charge is good and he wins the game.

The Input File

The input file is named "`P01.ENT`". It contains between 4 and 40 lines.

The first line of this file contains a whole number j corresponding to the number of players for this game ($3 \leq j \leq 6$).

The second line contains a whole number g corresponding to the number of the player that won the game.

The third line contains a succession of whole numbers separated by one or more spaces. These numbers are the numeric codes (given in the previous section) corresponding to the cards that the player number g received at the beginning of the game.

Each following line until the one before last contains 6 whole numbers separated by one or more spaces. These numbers give information related to a guess made by one of the players during the game:

- the first number indicates the number of the player who made the guess.
- the three following numbers indicate the numeric code of the suspect, the vehicle and the parking lot that were indicated in the guess.
- the fifth number is the number of the player who contradicted the guess. A zero "0" indicates that no player was able to contradict the guess.
- If the player who made the guess is the winning player then the sixth number corresponds to the numeric code of the card that was shown to prove that the guess was wrong. If the player who made the guess is not the player that won the game, the sixth number is zero "0".

Hint: You can always assume the following: When a player, different than player g (the winner), makes a guess which is not contradicted by any other players then you may assume that this player made a guess having in his hand at least one of the cards related to his guess.

The last line of the input file contains 6 zeros separated by spaces (0 0 0 0 0 0) to indicate to your program that it has all the necessary information to discover the three cards in the envelope.

Here for example, the input file corresponding to game in the previous section:

▷

```
3
3
1 5 13 14 18 20
1 4 12 20 2 0
2 2 10 17 0 0
3 6 10 20 0 0
1 1 12 19 3 0
3 4 9 19 2 4
0 0 0 0 0 0
```

◁

Output File

You must write the file "`P01.SOR`" which will contain a line with three integers separated by spaces. These three numbers correspond to the final charge. The first number corresponds to the numeric code of the culprit, the second to the numeric code of the vehicle and the third to the numeric code of the parking lot.

Here, for example, the output file corresponding to the input file given previously.

▷

```
6 10 19
```

◁

Marking

The problem will be corrected using four different input testing files.

A test will be regarded as successful if the program manages to find the solution and correctly write it in the output file.

Nombre de fichiers réussis	Points accordés
Successful test file	Points
1	80
2	120
3	160
4	200

Maximum time of execution of your program : 20 seconds.

Specification 2

The hidden words

Ruben Gonzalez-Rubio et Jean-Denis Boyer

`File to be produced containing your source code: P02_EQ##.*`

This problem scans for words in a square grid of letters. Its difficulty is to know how to read in any direction.

Problem

Consider a square grid containing the capital letters (ASCII characters `[A.. Z]`). We want to count all the occurrences of a word (a character string containing capital letters).

A word m is in a text T if there is a sub-chain of T equal to the chain m. Following are the "reading rules" to find a word in the text.

The word has to hold on one line, column or diagonal[1]. Following are the ways that a word can be read in the grid:

- on a line (horizontal)
  - From left to right;
  - from right to left;
- on a column (vertical)
  - From top to bottom;
  - From bottom to top;

[1] A word cannot start on a line, column or diagonal to end on another line, column or diagonal

- on a diagonal:
  - From left to right;
    - From top to bottom
    - From bottom to top;
  - From right to left;
    - from top to bottom;
    - from bottom to top;

The input file

The input file you must read is "`P02.ENT`". It is divided in three parts.

The first part is the first line of the file containing the word to seek. Its size lies between 1 to 64 characters. The second part is an integer n which is size of the grid (a grid of size n is a $n \times n$ grid, i.e. n lines and n columns). The value of n range between 1 and 256. The third part is the grid itself: at this point in the input file until the end of the file, each line correspond to a line in the grid. Such a line in the file contains n ASCII capital letters seperated by one or more blank spaces.

Each part of the input file are separated by a blank line.

▷

```
ABACUS

7

A B A C U S A
B B X Z B B S
A B A B A U W
C Q C C C G R
U U U A U M J
S S B B Y S D
L A R R G R B
```

◁

Output file

You must write in the file "`P02.SOR`". This output file must contain the number of occurrences of the word in the grid.

▷

```
5
```

◁

In order to clarify the problem, here another example of input file and its corresponding output file (answer):

▷

```
ABA

7

A B A C U S A
B B X Z B B S
A B A B A U W
C Q C C C G R
U U U A U M J
S S B B Y S D
L A R R G R B
```

◁

Output file:

▷

```
16
```

◁

Marking

The problem will be corrected using four different input testing files.

A test will be regarded as successful if the program manages to find the solution and correctly write it in the output file.

Nombre de fichiers réussis	Points accordés
Successful test file	Points
1	80
2	120
3	160
4	200

Maximum time of execution of your program : 20 seconds.

Specification 3

The Lottery

Paul-André Chassé

`File to be produced containing your source code:  P03_EQ##.*`

In certain lotteries, the player must choose n numbers from a set of integers from 1 to m inclusively where $m \geq n$. The first prize is won if at least one of the players chose exactly the same n numbers as those obtained from a draw which is generally carried out with a mechanical device that randomly chooses n balls numbered from 1 to m. The n numbers corresponding to the n balls drawn are then sorted in ascending order to easily recognize the winning combination.

For example, in lottery 6/49, the winning combination is made up of 6 numbers ranging between 1 to 49. If, during a drawing, the first number to came out is 23, the second is 42, the third is 17, the fourth is 2, the fifth 18 and finally the sixth 33, then the winning combination will be $\{2, 17, 18, 23, 33, 42\}$.

The Problem

In the remainder of this text the following symbols are used:

A series of numbers between { and } is used to represent a combination. For example $\{1, 2, 3, 4, 5, 6\}$ and $\{2, 5, 7, 8, 10, 11\}$.

The symbol $\{a_i\}_{i=1}^{n}$ represents a combination of n numbers where, for $i = 1, 2, \ldots n$, each a_i is an integer between 1 and m. Hence $\{a_i\}_{i=1}^{6}$ is equivalent to $\{a_1, a_2, a_3, a_4, a_5, a_6\}$.

The symbol C_m^n represents the number of combinations of n elements among m elements. One calculates this number by using the formula $C_m^n = \frac{m!}{n!(m-n)!}$. One easily sees that the number C_m^n is exactly the number of all possible combinations in a lottery of the type we described.

We want to assign to each combination an integer index ranging between 1 and C_m^n and such that each combination corresponds to only one index and each index corresponds only to one combination. One can suppose that we will build a table with two columns containing C_m^n lines, each line being indexed from 1 to C_m^n and containing one combination.

The first column will be reserved for the indices which go from 1 to C_m^n. On the first line, the index is 1, on the second line the index is 2 and so on.

The second column will contain all the C_m^n possible combinations laid out while following the following rules:

1. The numbers forming a combination must be in ascending order, i.e. from smallest to largest when going from left to right.

2. The combination to be registered on the first line is $a_i = i$ for $i = 1, 2, \ldots, n$. For $n = 6$ the first combination is $\{1, 2, 3, 4, 5, 6\}$.

3. For the subsequent lines of the second column, the combination to be registered is built by incrementing the combination registered on the preceding line according to the algorithm 1

Algorithm 1: Next combination
Input: n, m, Combination[$index$]
Output: Combination[$index + 1$]
NEXTCOMBINATION(n, m, $Combination_{index}$)

```
(1)   {a_i}_{i=1}^n ← Combination_index
(2)   if (a_1 == (m − n + 1))
(3)      return ''No other combination''
(4)   i ← n
(5)   while (a_i == (m − n + i))
(6)      do
(7)      i ← i − 1
(8)      od
(9)   a_i ← a_i + 1
(10)  while (i < n)
(11)     do
(12)     a_{i+1} ← a_i + 1
(13)     i ← i + 1
(14)     od
(15)  Combination_{index+1} ← {a_i}_{i=1}^n
(16)  return Combination_{index+1}
```

Here are examples from lotto 6/49.

Example 1.

The combination registered in line 1 is $\{1, 2, 3, 4, 5, 6\}$. The last value of this combination is 6 as 6 is not equal 49, the combination to be registered in line 2 is $\{1, 2, 3, 4, 5, 7\}$.

Example 2.

The combination registered in line 44 is $\{1, 2, 3, 4, 5, 49\}$ since the 6th element is equal to 49 and that 5 is not equal to 48, the combination to be registered in line 45 is $\{1, 2, 3, 4, 6, 7\}$.

Example 3.

The combination registered in line 1711512 is $\{1, 37, 46, 47, 48, 49\}$. Since the 6th, 5th, 4th and 3rd elements are equal to their respective limit and that 37 is not equal to 45, the combination to be registered with the line 1711513 is $\{1, 38, 39, 40, 41, 42\}$, and for the line 1711514 the combination is $\{1, 38, 39, 40, 41, 43\}$.

Example 4.

The combination registered in line 1712304 is $\{1, 45, 46, 47, 48, 49\}$. Since the 6th, 5th, 4th, 3rd and 2nd elements are equal to their respective limit, the combination with the line 1712305 is $\{2, 3, 4, 5, 6, 7\}$.

4. The combination registered on the last line is $\{a_i\}_{i=1}^{n}$ such as $a_i = (m-n+i)$; for example, with the 6/49 lottery ($n = 6$ and $m = 49$), the last combination in the list is $\{44, 45, 46, 47, 48, 49\}$.

Here is a complete example.

For a lottery of the type 3/7 the number of combinations would be 35. By using the criteria of classification given by the algorithm, the 35 combinations would be ordered as in table 3.1

Your work consists in writing a program that:

- will instantaneously find the index corresponding to a given 6/49 combination.
- moreover, your program will also have to do the opposite work, i.e. for any index read in entry, it must find the corresponding combination instantaneously.

Last detail, for this problem, there is no complementary number (as in the actual 6/49 lottery).

Input File

The file that your program must read is "`P03.ENT`". This file will contain at most 50 000 lines.

The first line of this file contains a code which indicates in which direction, for all this file, your program must carry its work.

The number `1` indicates that each following line contains a combination for which you must calculate the index.

The number `2` indicates that each following line contains an index for which you must find the corresponding combination.

The format of all the remaining lines depends on the code of the first line

If the code is `1`, then a line consists of 6 integers separated by a space.

If the code is `2`, then a line consists of a single integer.

▷

Index	Combination
1	1, 2, 3
2	1, 2, 4
3	1, 2, 5
4	1, 2, 6
5	1, 2, 7
6	1, 3, 4
7	1, 3, 5
8	1, 3, 6
9	1, 3, 7
10	1, 4, 5
11	1, 4, 6
12	1, 4, 7
13	1, 5, 6
14	1, 5, 7
15	1, 6, 7
16	2, 3, 4
17	2, 3, 5
18	2, 3, 6
19	2, 3, 7
20	2, 4, 5
21	2, 4, 6
22	2, 4, 7
23	2, 5, 6
24	2, 5, 7
25	2, 6, 7
26	3, 4, 5
27	3, 4, 6
28	3, 4, 7
29	3, 5, 6
30	3, 5, 7
31	3, 6, 7
32	4, 5, 6
33	4, 5, 7
34	4, 6, 7
35	5, 6, 7

Table 3.1: Index and combinations

```
 1
 1  2  3  4  5  6
 1  2  3  4  5 49
 1 37 46 47 48 49
 1 45 46 47 48 49
 2  3  4  5  6  7
 2  4 20 32 39 41
 2  9 10 15 48 49
 2 16 26 42 43 47
 3  5 19 21 24 41
 3 10 16 36 41 43
 3 21 22 29 42 43
 4  8 10 14 15 48
 4 14 24 28 32 41
 5  7 11 37 38 43
 6  7 17 32 39 40
 7  9 11 15 28 36
 8 12 15 17 29 44
 9 19 32 34 35 45
11 17 21 29 32 41
44 45 46 47 48 49
```

◁

Output file

You must write the file "P03.SOR". This file will have the same format whatever the format of the input file. It will contain the same number of lines as the input file less one line. On each line, you will write index of the combination followed by a space and then the 6 numbers of the corresponding combination separated by spaces.

▷

```
       1  1  2  3  4  5  6
      44  1  2  3  4  5 49
 1711512  1 37 46 47 48 49
 1712304  1 45 46 47 48 49
 1712305  2  3  4  5  6  7
 2000000  2  4 20 32 39 41
 2500000  2  9 10 15 48 49
 3000000  2 16 26 42 43 47
 3500000  3  5 19 21 24 41
 4000000  3 10 16 36 41 43
 4500000  3 21 22 29 42 43
 5000000  4  8 10 14 15 48
 5500000  4 14 24 28 32 41
```

```
  6000000  5  7 11 37 38 43
  7000000  6  7 17 32 39 40
  8000000  7  9 11 15 28 36
  9000000  8 12 15 17 29 44
 10000000  9 19 32 34 35 45
 11000000 11 17 21 29 32 41
 13983816 44 45 46 47 48 49
```

◁

Points

The problem will be corrected using four different input testing files.

A test will be regarded as successful if the program manages to find the solution and correctly write it in the output file.

Nombre de fichiers réussis	Points accordés
Successful test file	Points
1	120
2	180
3	240
4	300

Maximum time of execution of your program : 20 seconds.

Specification 4

Latin Square

Guillaume Coté

The great mathematician Leonhard Euler (1707-1783) introduced latin squares as a "nouvelle espèce de carrés magiques" (a new kind of magic squares). A latin square of order n is an n by n array of symbols in which every symbol occurs exactly once in each row and only once in each column.

The problem

In the following we consider a latin square of order 9 with one more restriction: in each of the 9 small 3×3 squares it is also required that the numbers 1 to 9 appears once and only once.

For this problem you will be given an incompleted latin square.

	6		4		9		5	
4		3	7			6		
		8		1		3		9
1		7		5		2		
	8			4	3		6	
	5		9				8	1
7					8	5		4
6	2			7				8
		5	2		4		7	

It is the job of your program to find a number into each box so that each row across, each column down, and each small 3×3 square within the larger square (there are 9 of these) will contain each number 1 through 9.

2	6	1	4	3	9	8	5	7
4	9	3	7	8	5	6	1	2
5	7	8	6	1	2	3	4	9
1	4	7	8	5	6	2	9	3
9	8	2	1	4	3	7	6	5
3	5	6	9	2	7	4	8	1
7	1	9	3	6	8	5	2	4
6	2	4	5	7	1	9	3	8
8	3	5	2	9	4	1	7	6

The input file

The file that you must read is "`P04.ENT`". The empty boxes to fill are indicates by a `_`. The numbers (from 1 to 9) and the empty boxes are sperated by a blank space. An input file will contains exactly nine (9) lines, each of them having 9 symbols.

▷

```
_ 6 _ 4 _ 9 _ 5 _
4 _ 3 7 _ _ 6 _ _
_ _ 8 _ 1 _ 3 _ 9
1 _ 7 _ 5 _ 2 _ _
_ 8 _ _ 4 3 _ 6 _
_ 5 _ 9 _ _ _ 8 1
7 _ _ _ _ 8 5 _ 4
6 2 _ _ 7 _ _ _ 8
_ _ 5 2 _ 4 _ 7 _
```

◁

The output file

You must write in the file "`P04.SOR`". It is a table of integers separated by blanks, 9 integers for each line and obviously with 9 lines.

Following is the output file corresponding to the input file above.

▷

```
2 6 1 4 3 9 8 5 7
4 9 3 7 8 5 6 1 2
5 7 8 6 1 2 3 4 9
1 4 7 8 5 6 2 9 3
9 8 2 1 4 3 7 6 5
3 5 6 9 2 7 4 8 1
7 1 9 3 6 8 5 2 4
```

```
6 2 4 5 7 1 9 3 8
8 3 5 2 9 4 1 7 6
```

◁

Marking

The problem will be corrected using four different input testing files.

A test will be regarded as successful if the program returns the proper result.

Nombre de fichiers réussis Successful test file	Points accordés Points
1	120
2	180
3	240
4	300

Maximum time of execution of your program : 20 seconds.

Specification 5

Links

Martine Bellaïche, Jean-Yves Hervé

`File to be produced containing your source code:  P05_EQ##.*`

In the operating system Unix or Linux, there is an important command which makes it possible to create symbolic links to a file. These links make it possible to save memory space on a hard disk. For simplification, the name of a file represents a certain region of the memory space on the disk where the data contained in the file are saved. Starting from this file, one can create a link which points to the same region of memory on the hard disk. One can then starting from this link establish another one; and starting from this other link one can create another one, and so on in order to obtain a list of links starting from the source file.

The Problem

Example of only one list of links.

The pair `fortran cobol` identifies a link between the source element `fortran` and the destination element `cobol`, i.e. starting from the element `cobol`, one finds the element `fortran`. One enumerates in order the elements of a list of pairs so to identify the links between these elements:

```
fortran cobol
cobol modula
modula eiffel
eiffel c
```

One thus manages to identify the list `c eiffel modula cobol fortran`. It is necessary to enumerate the pairs in order i.e. the pair `cobol modula` must be provided before the pair `modula eiffel`.

Example of two lists whose links are merged. In each one of these lists, the pairs are given in order. The program must find the two lists.

```
italy brazil
fortran cobol
brazil mexico
mexico france
cobol modula
france united-states
modula eiffel
eiffel c
united-states canada
```

One obtains the list `c eiffel modula cobol fortran` and `canada united-states france mexico brazil italy`.

To this problem, one adds the following additional considerations (each one related to a particular situation):

- One or more lists of links are being partially constructed and that a new pair is added. For one (or more) of these lists, suppose that the source element of this pair is already in the list, other than the first element, and the destination element is a new one. In this case, for each of such lists, a new list is added.

  For example:

  ```
  fortran cobol
  cobol modula
  modula eiffel
  eiffel c
  cobol java
  ```

  One then obtains the list `c eiffel modula cobol fortran` and the list `java cobol fortran`.

- One started to form two distinct lists of links whose links are merged. One wants to add in these lists a pair whose source element is the first element of one of the list and the destination element is an element of the other list. The destination element can only belong to one list. In that case second list must be rupture.

  Example

  ```
  italy brazil
  fortran cobol
  brazil mexico
  mexico france
  cobol  modula
  france united-states
  ```

```
modula eiffel
eiffel c
united-states canada
canada modula
```

Notice that before the insertion of the last pair `canada modula`, one has the list `canada united-states france mexico brazil italy` and `c eiffel modula cobol fortran`. At the time of the addition of the pair `canada modula`, the two lists are modified and one obtains the list `c eiffel modula canada united-states france mexico brazil italy` and the list `cobol fortran`.

- Suppose that one or more lists of links are partially constructed and that a new pair is added. For one (or more) of these lists, suppose that the source element of this pair is already in the list, other than the first element. Suppose also that the destination element belongs to exactly one of the lists. Then in this case, for each of such lists, a new list is added and the initial list that contains the destination element is ruptured.

```
italy brazil
fortran cobol
brazil mexico
mexico france
cobol modula
france united-states
modula eiffel
eiffel c
united-states canada
modula france
```

Before adding the last pair, we have the list `canada united-states france mexico brazil italy` and the list `c eiffel modula cobol fortran`. After inserting the last pair, we get the following: the list `mexico brazil italy`, the list `c eiffel modula cobol fortran` and a new list `canada united-states france modula cobol fortran`.

Input file

The file that you must read is "`P05.ENT`". On the first line, there is the number of pairs representing a link, and on each following line, one finds the pair: "source element" and "destination element" which will form the one or several lists of links. The maximum number of pairs is 100. There is at least one pair. The following input file is in french (remember that the name of the links are only strings).

▷

```
14
italie bresil
manitoba ontario
```

```
fortran cobol
bresil mexique
ontario quebec
mexique france
quebec alberta
cobol modula
france cobol
france etats-unis
modula eiffel
etats-unis canada
etats-unis ontario
fortran c
```

◁

Output file

You must write in the file "`P05.SOR`" which will contain on each line a list of links.

▷

```
eiffel modula cobol france mexique bresil italie
manitoba
c fortran
canada etats-unis france mexique bresil italie
alberta quebec ontario etats-unis france mexique bresil italie
```

◁

Marking

The problem will be corrected using four different input testing files.

A test will be regarded as successful if the program manages to find the solution and correctly write it in the output file.

Nombre de fichiers réussis	Points accordés
Successful test file	Points
1	120
2	180
3	240
4	300

Maximum time of execution of your program : 20 seconds.

Specification 6

Queue Simulation

Martine Bellaïche et Jean-Yves Hervé

`File to be produced containing your source code:  P06_EQ##.*`

Simulation plays an important role in several fields: transport, communication, networking, flight simulators, games, etc. The principal goal of a simulation is to obtain statistics, in order to measure the performance of an existing system or to predict the performance of a proposed system. Before building a new system, it is preferable to simulate its behavior in order to measure its effectiveness. Generally simulation will be done on computers, rather than using a prototype. The principal advantages are reduced cost and time of evaluation.

The problem

For our problem, we are interested in the simulation of a queueing system. The queueing system comprises clients, a queue and a server. The clients arrive at random in the system, await their turn in the queue if necessary and are served by the server.

To simulate the behavior of a system, it is necessary to know the inter-arrival time, the service time as well as the duration of the simulation. The inter-arrival time is the time between two consecutive arrivals of clients. The time of service is the time taken by the server to achieve its work. These two times are represented by random numbers. The duration of simulation is the total time of simulation. In the following example, the time is indicated only when a client arrives or when a client is served. Moreover, the contents of the queue are modified with each event.

Example of a simulation:

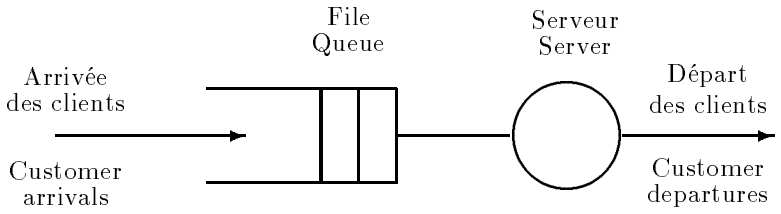

Figure 6.1: A queue

Random inter-arrival times: $6, 6, 9, 6, 7, 10, 7, 12, 14, 8$.

Random service times: $13, 15, 16, 13, 14, 17, 19, 21, 15, 20$.

Duration of simulation: 85

Notation: (C_n, TI,TS): C client n , inter-arrival time TI, service time TS.

Time	Event	queue
0	system opening	
6	client arrival (C_1, 6, 13) service starting time C_1	
12	client arrival (C_2 ,6,15)	(C_2)
19	service ending time C_1 service starting time C_2	
21	client arrival (C_3, 9, 16)	(C_3)
27	client arrival (C_4, 6, 13)	(C_3)(C_4)
34	service ending time C_2 service starting time C_3 client arrival (C_5,7,14)	(C_4) (C_5)
44	client arrival (C_6,10, 17)	(C_4)(C_5) (C_6)
50	service ending time C_3 service starting time C_4	(C_5) (C_6)
51	client arrival (C_7 ,7, 19)	(C_5) (C_6) (C_7)
63	service ending time C_4 service starting time C_5 client arrival (C_8,12,21)	(C_6) (C_7) (C_8)
77	service ending time C_5 service starting time C_6 client arrival (C_9, 14, 15)	(C_7) (C_8) (C_9)
85	End of the simulation	(C_7) (C_8) (C_9)
94	service ending time C_6	

Your work consists of writing a program that will simulate of the queuing system and calculates its statistics.

During simulation, it is possible that the server is not serving any clients. This is possible when:

1. At a given time, there is no more client in the queue and none has arrived yet.
2. All the client have arrived and have been served before the end of the simulation.

Input file

The name of the input file is "`P06.ENT`".

On the first lines, there is two (2) integers separated by one or more blank spaces. The first integer indicates the number of lines to be treated and the second one the duration of the simulation. Each following line correspond to a client. On such a line there is 2 integers separated by one or more blank spaces. The first integer indicates the inter-arrival time. The second integer indicates the service time for this client. The maximum number of clients 500 and their is at least one client.

Here

▷

```
10 150
6 1
6 1
9 16
6 13
7 14
10 17
7 19
12 2
14 5
8 4
```

◁

Output file

You must output your results in the file "`P06.SOR`" must contain on each line the following statistics:

The maximum length of the queue.

The total number of entries carried out in the queue (one refuses a clients who arrives at the same time as the end of simulation).

The total number clients who did not wait in the queue.

The sum of the waiting time in the queue of the clients that have been served.

The number of clients who wait in the queue at the end of simulation.

The number of services carried out by the server.

The total service time carried out by the server (at the end of the simulation, the server must finish its service).

Here the output file corresponding the input file above:

```
▷

3
10
3
160
0
10
92

◁
```

Marking

The problem will be corrected using four different input testing files.

A test will be regarded as successful if the program returns the proper statistics of the simulation.

Nombre de fichiers réussis	Points accordés
Successful test file	Points
1	120
2	180
3	240
4	300

Maximum time of execution of your program : 20 seconds.

Specification 7

Bridges on the Internet

Alex Boisvert

`File to be produced containing your source code:  P07_EQ##.*`

Internet is a complex communication digital network to which millions of computers are connected. The quantity of information conveyed simultaneously over the network is phenomenal. Nevertheless, the transmission capacity on Internet is limited and, for this reason, the information transmitted by a computer must be conveyed judiciously to the receiver if one wishes to maximize the quantity of exchanged information and to avoid congestion in the network.

The Internet is composed of sub-networks. For example, a computer is connected to a local area network, which in turn is connected with another network and so on. The various networks are *inter-connected* by devices which distribute the information so that it goes quickly to its destination.

To allow the simultaneous use of the network by several computers, the messages which circulate on the network are generally divided into small *packets* of information. Each packet of information contains the identity of the transmitter, that of the receiver and a certain quantity of information that in theory, only the receiver can interpret. Individually, each packet can be conveyed in an independent way.

Among the various devices which direct information through the network, the basic functional unit is called a **bridge**[1]. In a simplistic way, the bridges decide which path a packet will follow

[1] Not to be confused with a *router*. Routers are more complex and "intelligents" than the bridges.

between the transmitting computer and the receiver.

The Problem

You must write a program which simulates an *inter-connected* network in which computers exchange packets of information and in which the bridges switch the packets of information between the computers.

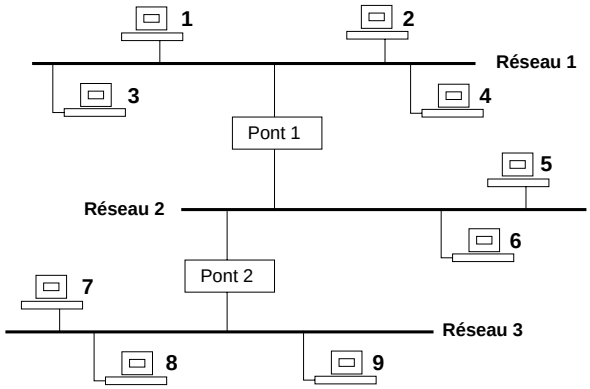

Figure 7.1: Interconnected network composed of 3 networks, 9 computers and 2 bridges. (Réseau = Network, Pont = Bridge).

From now until the end of this problem, we will assume that a bridge links two different networks. We will also assume that when a packet of information is transmitted on a local area network, all the devices connected with this network receive the packet simultaneously. The propagation time of the packets and the time required by the bridges to process a packet are assume to be constant.

Now let us describe the operation of a bridge. A bridge knows only the two networks directly connected to it. It maintains a table of origin of each packets it receives. For each received packet, the bridge records in a table the identity of the transmitter and from which of the two networks the packet was received. By doing this, if later this same bridge receives a packet of information intended for this transmitter, it knows where to send the packet.

The decision to allow a packet to go on the other network to which the bridge is connected depends on the identity of the transmitter and the receiver. If the identity of the receiver (say computer A) is already in the bridge's table which means that the bridge has processed before a packet sent by computer A, then the bridge knows on which side is computer A. If the receiver (computer A) is on the same side from which comes the actual packet to convey, then the bridge disregards the packet. If not, the bridge re-transmits the packet to the other network.

When the bridge re-transmits a packet , it re-transmits it in its original form, without adding information to it.

Notes:

- For this problem, only computers transmit packets . A bridge does not transmit a packet by itself. It merely transfers packets coming from computers.
- A computer is connected to only one network.
- It is possible to have more than one bridge between two networks for reasons of robustness at the event of breakdown of a bridge or a network link.
- An inter-connected network may have loops. This means that it is possible to have different paths between two computers and through different local area networks. This would be the case if a bridge was put between Network 1 and Network 3 in Figure 7.1.
- If a bridge receives simultaneously the same packet from both the networks to which it is connected, then both copies of the packet are disregarded. After that any packet sent by the same transmitter will also be disregard by this bridge.

See Figure 7.1 for an example of the structure of a network.

The input

The input is made up of two principal elements: a specification of the structure of the global network and a list of transmitted packets. The structure of the network is subdivided into two parts: a list of computers connected with networks and a list of bridges connecting the networks. These two lists are preceded by the number of entries which compose them.

For the list of computers, one finds a single computer identification number followed by the number of the network to which it is connected. For the list of the bridges, there is a bridge identification number followed by the numbers of the two networks to which it is connected. There are always at least a computer and at least a bridge in these lists.

The list of packets transmitted is preceded by the number of packets transmitted during the simulation. Each transmission includes the identity of the transmitting computer followed by the receiver's identity. There is always at least a packet to transmit.

It will be assumed that a packet is necessarily conveyed to its destination before a new packet is transmitted on the network.

The input file contains a maximum of 25 bridges, a maximum of 25 local area networks, a maximum of 100 computers and a maximum of 100 packets.

▷

```
9
1 1
2 1
3 1
4 1
5 2
6 2
7 3
```

```
8 3
9 3
2
1 1 2
2 2 3
6
1 4
4 1
9 4
4 9
5 9
9 5
```

◁

The output file

You must create and write in the file "`P07.SOR`" which will contain the list of all the bridges which convey the transmitted packet for each packet sent. For a particular packet, the list of the bridges must appear on the same line. The order of appearance of the lines in the output file must correspond to the order of appearance of the packets in the input file. However, the order of enumeration of the bridges on the same line is free for you to choose and does not represent necessarily the chronological order of routing of a given packet.

If, for an transmitted packet, no bridge re-transmits its, you must leave an empty line in your output file.

Moreover, a bridge appears only once on a line even if it receives the same packet more than one time.

▷

```
1 2

2 1
1 2
2
2
```

◁

Example

This example use the network layout of Figure 7.1:

1. Computer 1 transmits a packet to Computer 4. In that case this is how the packet is processed:

   (a) Computer 4 and Bridge 1 receive the packet (so are Computers 2 and 3, but it is not important here).

   (b) Brigde 1 puts in its table that Computer 1 is on the same side of Network 1.

   (c) Bridge 1 transmits the packet on Network 2 since it doesn't have any information on Computer 4.

   (d) Bridge 2 receives the packet.

   (e) Bridge 2 puts in its table that Computer 1 is on the same side of Network 2.

   (f) Bridge 2 transmits the packet on Network 3 since it doesn't have Computer 4 in its table.

2. Computer 4 transmits a packet to Computer 1. This is how the packet is processed:

   (a) Computer 4 and Bridge 1 receive the packet.

   (b) Bridge 1 puts in its table that Computer 4 is on the same side of Network 1.

   (c) Bridge 1 does not transmit the packet since, from its table, its knows that Computer 1 is not the other side.

3. Computer 9 transmits a packet to Computer 4.

   (a) Bridge 2 receives the packet.

   (b) Bridge 2 puts in its table that Computer 9 is on the same side of Network 3.

   (c) Bridge 2 transmits the packet since it does not have any information on Computer 4.

   (d) Bridge 1 receives the packet.

   (e) Bridge 1 puts in its table that Computer 9 is on the same side of Network 2.

   (f) Bridge 1 transmits the packet since it knows that Computer 4 is on the same side as Network 1.

   (g) Computer 4 receives the packet.

4. Computer 4 transmits a packet to Computer 9.

   (a) Bridge 1 receives the packet.

   (b) Bridge 1 notice that it already has Computer 4 in its table so it does not update its table (see 2b).

   (c) Bridge 1 transmits the packet since it knows that Computer 9 is on the other side.

   (d) Bridge 2 receives the packet.

   (e) Bridge 2 puts in its table that Computer 4 is on the same side of Network 2.

   (f) Bridge 2 transmits the packet since it knows that Computer 9 is on the same side of Network 3.

Marking

The problem will be corrected using four different input testing files.

A test will be regarded as successful if the program manages to find the solution and correctly write it in the output file.

Nombre de fichiers réussis	Points accordés
Successful test file	Points
1	160
2	240
3	320
4	400

Maximum time of execution of your program : 20 seconds.

Specification 8

Honey, I blow my fuse!

Benoît Parent et Ruben Gonzalez-Rubio

`File to be produced containing your source code: P08_EQ##.*`

Everyone knows that components of an electrical circuit can be damaged if a short-circuit occurs. The fuse was invented to protect expensive components or even the user of an electrical device.

Figure 8.1 shows an industrial electric circuit. For simplification, we put only one alternative current source AC, 3 fuses (F_1, F_2, F_3) and loads (C_1, C_2). The electrical current on the branches of the circuit is a function of the tension (voltage) of the source and the loads. If the loads are constant the current remains also constant, but if the load changes, more current may be needed. For example, when using an electrical saw, more current (more amps) is needed to cut harder wood than less harder wood. Common sense says that if the consumed current exceeds the current threshold of the fuse then it melts and the power is shutoff from the circuit. In other words, the fuse is blown!

Let us look in detail at an electrical installation (see Figure 8.1)

Case 1 Let $F_1 = F_2 = 100A$ and $F_3 = 200A$. The loads C_1 consumes $35A$ and C_2 consumes $40A$. The current which passes by F_3 is the sum of the currents of the loads $35 + 40 = 75A$. Here in this case everything is fine and it can be so for a very long time. If the load C_1 consumes more current (more amps A) say $250A$, then the fuse F_1 blows.

Case 2 Now, suppose that we have the following values: $F_1 = F_2 = 100A$, $F_3 = 120A$ and the loads C_1 consumes $35A$ and C_2 consumes $40A$. The current which passes by F_3 is the sum of the currents of the loads $35 + 40 = 75A$. Still, everything is fine, but if the load C_1 consumes more current (more amps A), let us say $250A$, then the fuses F_1 and F_3 blow, stopping the current in all the installation, which in general is the last thing you want.

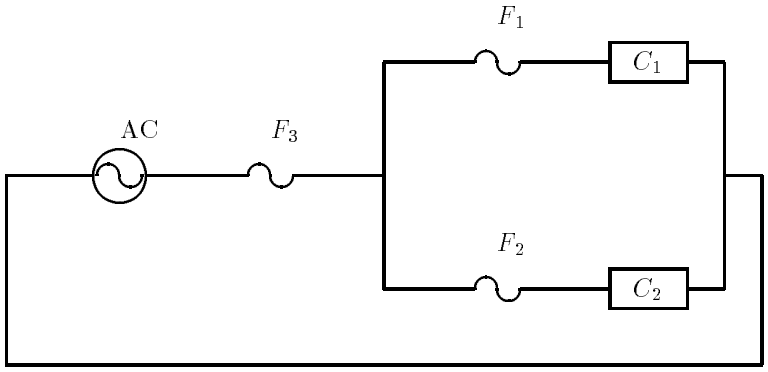

Figure 8.1: Electrical circuit with fuses

You wonder why? Here the reasons: a fast fuse[1] is characterized by a rated maximum current I_n. This rated maximum current indicates the maximum current the fuse can hold during an infinite time. The characteristic curve of the rupture time of a fuse is related to I_n and i the current going through the fuse. Figure 8.2 shows the curve of a $100A$ fuse. For any other fuse the shape of the curve is similar, it is just "shifted" according to the value I_n.

The curve is divided into two parts given by the equations:

$$t_1 = \frac{0.5}{\frac{i}{I_n} - 1} + 0.5 \tag{8.1}$$

and

$$t_2 = \frac{2}{\frac{i}{I_n}} \tag{8.2}$$

The two curves intersect each other when $i = 2I_n$ thus $t = 1(\text{sec})$. One observes that the more the current through the fuse, the less longer it lasts. Equation 8.1 is to be used when the current going through the fuse is lower than $2I_n$. If $i > 2I_n$ then Equation 8.2 is used.

The characteristic curves are defined by the rated current I_n. For this problem, a $200A$ fuse will blow in one second if a current of $400A$ is going through it. A $150A$ fuse blows in one second if a current of $300A$ is going through it, etc...

Let us look at Case 1 and calculate times of rupture when a current of $250A$ is applied on C_1.

Fuse	I_n	i	Equation	t
F_1	$100A$	250	8.2	$0.8sec$
F_2	$100A$	40	8.1	∞
F_3	$200A$	290	8.1	$1.61sec$

[1] We consider in this problem only fast fuses. The other type of fuses are the delayed one.

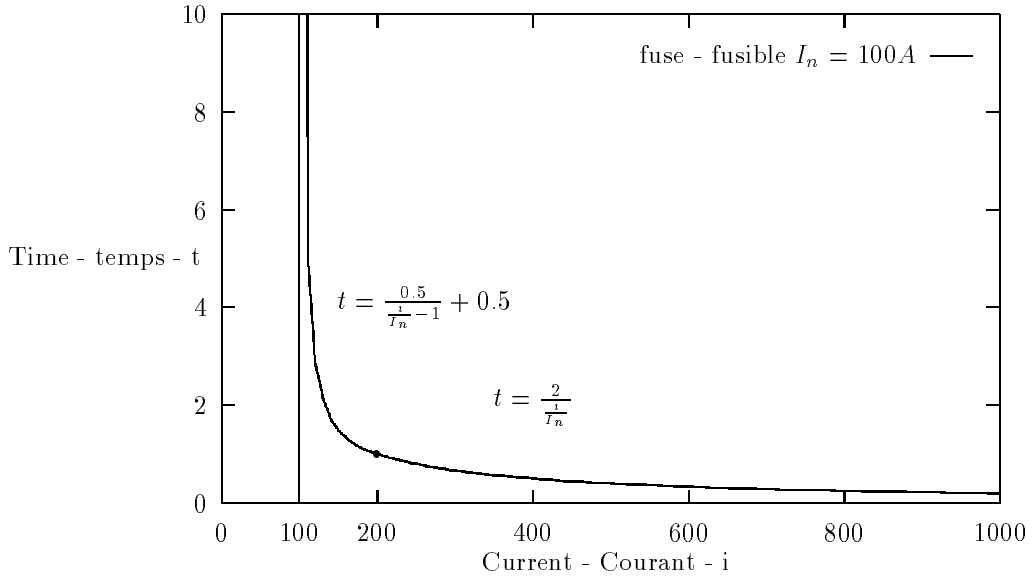

Figure 8.2: Electrical circuit with fuses

If the current remains on longer than $0.8sec$ the fuse F_1 blows but F_3 remains intact, because the relation $\frac{t_{f_3}}{t_{f_1}} \geq 2$ is true[2]. The time of rupture calculated for the fuse F_1 is represented by t_{F_1}, and that of the fuse F_3 is represented by t_{F_3}.

One sees that t_{F_1} and t_{F_3} are very close to each other. If the current remains longer than $0.8sec$, we determine that the fuses F_1 and F_3 blow because the relation $\frac{t_{f_3}}{t_{f_1}} \geq 2$ is false; therefore the relation $\frac{t_{f_3}}{t_{f_1}} < 2$ is true.

In this problem, the model we use is very close to reality even if we made some simplifications. Some details have been left out as the aging of a fuse.

Problem

We ask you to write a simulator of an electric installations to know which fuses will blow in the case of an overload.

The diagram of an electric installation with fuses always take a tree like shape as in figure 8.3. The loads are the leaves of the tree. All the points g (ground) are connected together.

The principal fuse F_{101} is at the entry of the installation (at the root of the tree) and then several branches develop. On each one, one finds a fuse with either several other branches, or a load. It is supposed that a load is protected directly by only one fuse.

[2]This rule is given in more detail later in the text. We use the value 2 to simplify the problem.

It will be guaranteed that a fuse in a lower level will have a rated current equal to or higher than those which are in a higher level in the tree.

A fuse is characterized by a triplet

$$F = \{N, F_f, I_n\}$$

where N is an integer used as an identifier, F_f is the identifying integer of the fuse of the immediate lowest level[3] and I_n the maximum current the fuse can hold, i.e. its rated current.

A load is characterized by a triplet

$$C = \{N, F_f, i\}$$

where N is an integer used as an identifier, F_f is the identifying integer of the fuse at the immediate lower level and i indicates the current normally going through the load.

The overload is indicated by a quadruple:

$$SC = \{N, F_f, i, d\}$$

where N is an integer used as an identifier, F_f is the identifying integer of the fuse at the immediate lowest level, i is the current required by the load, and thus the current passing through the fuse, at the time of the overload and d the duration of the overload. This duration represents the time of the overload if the circuit supported it. If the duration is higher than the rupture time when current i is going through, then the fuse blows thus cutting off the current.

We point out that the current going through a fuse of lower level must be equal to the sum of the currents going through the fuses of immediate higher level.

The rules to determine which fuses will blow:

1. For one or more fuses to blow, the duration of the overload must be higher or equal to the time of rupture indicated by the characteristic curve of the fuse(s). An overload of $200A$ during $0.5sec$ does not make a fuse to blow with $I_n = 100A$. The same overload applied during $1.0sec$ blows the fuse. Again the same overload with a theoretical duration of $2sec$ blows the fuse in one second.

2. Two adjacent fuses will blow if the relation $\frac{t_{F_n}}{t_{F_{n+1}}} < 2$ holds and if the duration of the overload is higher than $t_{F_{n+1}}$. Note that this rule can be applied several times to a branch, in fact on all the adjacent fuses of the branch for which the relation holds. If on a branch a fuse does not blow, all the fuses of lower level will not blow. To be more precise, consider an installation including four fuses F_1, F_2, F_3 and F_4 in series. The fuse F_1 is the root and F_4 is connected to the load. The following table shows which fuses blow. The duration of the overload is of $2sec$. The current of the overload is $200A$.

Values	Fuses that blow
$F_1 = 100$, $F_2 = 100$, $F_3 = 100$ et $F_4 = 100$	F_1, F_2, F_3 et F_4
$F_1 = 1\ 000$, $F_2 = 500$, $F_3 = 120$ and $F_4 = 100$	F_3 and F_4
$F_1 = 1\ 000$, $F_2 = 120$, $F_3 = 100$ and $F_4 = 80$	F_2, F_3 and F_4

[3] The root is a particular case, for it does not have a lower level. Therefore 0 is used as an identifier.

Input file

The file that you must read is "`P08.ENT`". The input is divided into two parts.

The first part that holds on one line indicates the potential overload quadruple.

The second part contains the description of the installation, a fuse or a load by line, therefore a triplet. The fuses and the loads of the installation are given without any particular order. It is up to you to sort whatever needed to be sorted.

The different parts of the input file are separated by an empty line. The separator in the triplets and the quadruples is of one or more spaces.

It is guaranteed that the installation is realizable, i.e. it forms a tree of fuses and loads, such as figure 8.3.

There can be between 1 to 10 000 fusible. There can be between 1 to 9 999 loads, the identifiers of fuses go from 1 to 16 000 and those of loads from 16 001 to 32 000. Identifier 0 is used to indicate a fuse connected with the root of the tree (the source AC).

All the numerical values are integer except duration time d which will be of type `double`.

Here the file representing the tree of the figure 8.3. Observe that the lines characterizing the fuses and the loads appear in the file in no particular order and that the identifiers are unique .

▷

```
16082 2 120 3.5

101 0 1000
109 101 140
107 101 500
45 109 120
32 109 100
33 109 200
34 107 100
35 107 100
41 45 100
2 45 80
7 33 100
15 33 100
21000 41 50
16082 2 20
22092 32 20
16094 7 25
16014 15 25
16012 34 40
22000 35 48
```

◁

Output file

In the output file we must have either the identifying integer of the blown fuse(s) if any, or either the identifying number of the load if no fuses are blown.

You must write in the file "`P08.SOR`" which will contain on a line one or more integers, representing the fuses that blow or possibly the identifier of the load if no fuse blows. The integers must be separated by one or more spaces.

▷

```
2 45 109
```

◁

Here the fuses 109, 45 and 2 are blown. The order in which the identifier appear on a line is not important.

In order to clarify the problem, here another example:

The input file:

▷

```
16082 2 120 3.5

101 0 1000
109 101 300
107 101 500
45 109 200
32 109 100
33 109 200
34 107 100
35 107 100
41 45 100
2 45 80
7 33 100
15 33 100
21000 41 50
16082 2 20
22092 32 20
16094 7 25
16014 15 25
16012 34 40
22000 35 48
```

◁

According to this previous input file, and up to the ordering of the integers on each lines, this is the output file that your program should produce:

▷

```
2
```

◁

Marking

The problem will be corrected using four different input testing files.

A test will be regarded as successful if the program manages to find the solution and correctly write it in the output file.

Nombre de fichiers réussis	Points accordés
Successful test file	Points
1	160
2	240
3	320
4	400

Maximum time of execution of your program : 20 seconds.

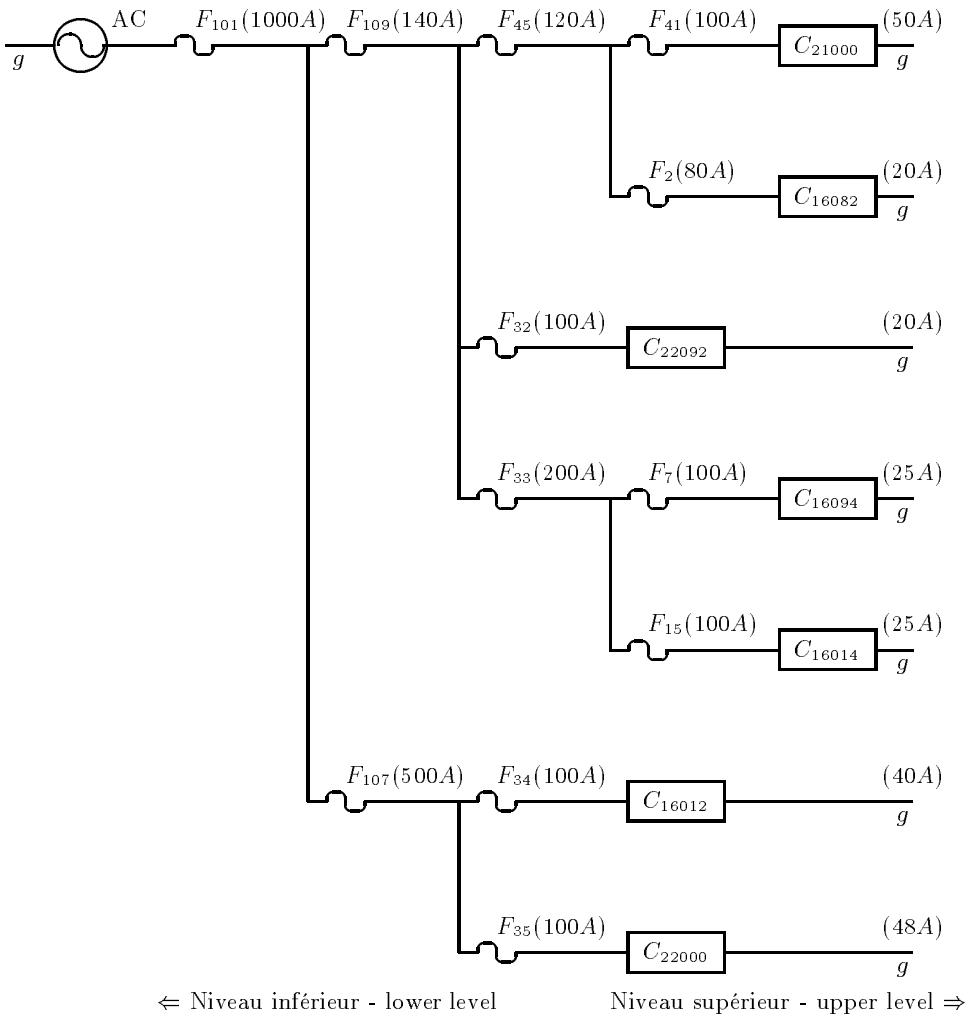

Figure 8.3: Tree of fuses and loads

Specification 9

The Powerline

Gaétan Haché, Nicolas Courtemanche et Nicolas Pelletier

`File to be produced containing your source code: P09_EQ##.*`

Following the ice-storm, several powerlines were out of service. Since then, Hydro-Québec wants to add redundancy in its network in order to make it more reliable. They need you to help them to choose the site of the pylons of new powerlines.

Problem

We want to build a powerline through a very mountainous area. Your problem consists in writing a program which will determine the site of the pylons of this powerline according to the following informations and rules:

1. the meter is the only measuring unit that will be used in this problem.

2. Seen from the air, the layout of the powerline is a straight line. One will agree as of now that the problem arises and is solved in two dimensions.

3. the relief of the ground between the first and last pylons is the initial data of the problem. The ground is divided by sections of flat ground so that each section has a constant slope over all its length. The relief will be described by the length (horizontal) and the elevation (vertical) of each section. For example, a section of 400 meters long with a 300 meters elevation as a slope of 3/4.

4. the pylons are 30 meters high.

5. the wire must be at least 20 meters away from the ground. To simplify the problem one measures this distance vertically (and not along a perpendicular to the ground which of course would be more realistic).

6. To calculate the distance from the ground to wire, it will be supposed that a wire between two pylons follows a parabolic trajectory of equation

$$y = ax^2 + bx + c$$

The coefficient a is fixed for throughout the problem and is set to

$$a = 0.00016.$$

Your program will have to determine the coefficients b and c according to the position of the tops of two adjacents pylons (see Remark 1 further in the text).

7. No maximum distance is imposed between two pylons. This way, it is possible not to have any pylon between two pylons crossing a ravine of two kilometers, as long as the ravine is sufficiently deep.

8. The rules to determine the site of a pylon between two others are the following ones:

   (a) For each section of ground between two adjacent pylons, let us say A and B, you locate the place where the distance between the ground and the wire is minimal. The calculation of this distance must be done between the ground and the theoretical trajectory of the wire. In other words:

      i. If the wire does not touch the ground, you must determine the place where the wire is closer to the ground.

      ii. If the wire touches the ground, then you imagine that the wire passes under ground (so that its parabolic trajectory is not modified) and you determine the place where the wire would be farthest away from surface.

   (b) The rule to follow to determine (if necessary) the site of a pylon is the following one:

      i. If for each section of ground the wire is at more than 20 meters of the ground, no pylon is added between A and B.

      ii. If not, you locate the site of a new pylon, say C, at the place where the distance between the wire and the ground is minimal. This minimum may not be unique when the ground is composed of several sections of different slopes. If the minimum is reached at more than one place, you choose the section of ground having the smallest slope in absolute value [1] and locate the site of the pylon C exactly at the place where the wire is closer to the ground (while measuring vertically).

   (c) Each time you determine a new site for a pylon, let us say C, between two pylons A and B, you must determine the new trajectory of the wire between A and C and also between C and B. Then, by using the same directives that was just applied to pylons A and B, you must determine, if necessary, the site of a pylon between A and C then between C and B.

[1] You can suppose that the slopes in absolute value of the sections of grounds are all distinct from each other.

(d) You stop determining new sites of pylons when the wire is at more than 20 meters of the ground in any point located between two pylons.

REMARK 1 *Recall that in the equation $y = ax^2 + bx + c$, only the coefficient a modifies the curvature of the parabola and that b and c locate the parabola in the plane. In other words, let (x_0, y_0) and (x_1, y_1) be the co-ordinates of two points corresponding to the tops of two adjacent pylons. Then for the fixed coefficient a, there are coefficients b and c, uniquely determined, such that the parabola of equation $y = ax^2 + bx + c$ passes through the points (x_0, y_0) and (x_1, y_1). In particular one has*

$$b = \frac{y_1 - y_0}{x_1 - x_0} - a(x_0 + x_1).$$

We leave it to you to find c.

The input file

The file that you must read is "`P09.ENT`".

As we already explained, the relief is described section by section, each one being described by its length and its elevation. The first line of the input file will contain the number of sections of grounds, each following line corresponding to a section of ground. These lines contain two numbers which are respectively the length and the elevation one of the section of ground. For example, a file containing the following line:

▷

```
3
300 100
100 -200
200 0
```

◁

represents three sections of ground as shown in Figure 9.1.

The first and last pylons are respectively located at the beginning and the end of the ground. Thus in the preceding example, 600 meters separate them.

Important Remark. You can suppose the following things:

1. No more than 1 000 sections of ground will be described in an input file.
2. No more than 1 000 pylons will be necessary for the construction of the powerline.

The output file

You must write in the file in "`P09.SOR`". The first line must contain the number of pylon sites your program has determined (excluding the site for the first and last pylons). Each other line of the file

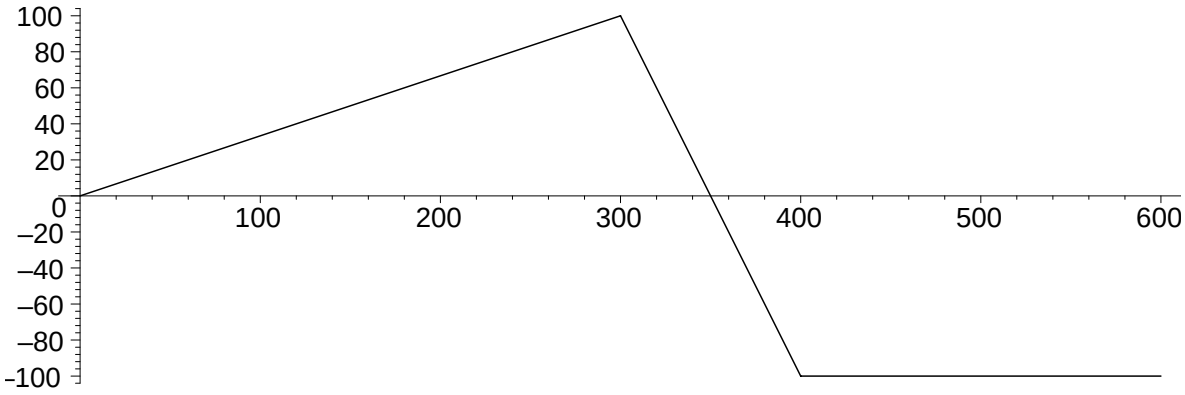

Figure 9.1: Ground profile

indicates a pylon site by giving the distance that separates it from the preceding one. For example, the output file

```
▷


1
300

◁
```

indicates that a site was determined. The powerline is thus made up of 3 pylons such that 300 meters separate the first from the second, 300 separate the second from the third (which is also the last one).

Important remark: All calculations must be done with variables of type **double**. It is only at the end, when you have determined all the pylon sites, that you must round-up your results so that one finds only intgers in the output file.

For the marking, an error of ± 1 meter will be tolerated.

Marking

The problem will be corrected using four different input testing files.

A test will be regarded as successful if the program manages to find the solution and correctly write it in the output file.

Nombre de fichiers réussis Successful test file	Points accordés Points
1	240
2	360
3	480
4	600

Maximum time of execution of your program : 20 seconds.

Specification 10

The Cat and the Mouse

Rusty Deschênes

`File to be produced containing your source code:  P10_EQ##.*`

An hungry cat must catch its dinner (a mouse). Help the cat to reach the mouse before it dies of hunger. The mouse being alive, it will move while the cat runs after it. The mouse is however not at all concerned with the cat; it will thus go wherever it feels like to go, no matter where is the cat.

The problem

The cat and the mouse are in a labyrinth made up of square boxes.

Here the characteristics of the labyrinth:

- There will be a maximum of 1024 boxes.
- A number n is associated with each box ($1 \leq N \leq 1024$).
- Two boxes cannot have the same number.
- Each box can have up to 4 passages (North, South, East, West). In certain cases, passages will be prohibited by a wall.
- Each box has at least one exit.
- The labyrinth has no particular layout. Figure 10.1 is an example of a labyrinth. Each box carries its number in its top-left corner, the walls are indicated by solid lines and the passages between boxes by dotted lines. Observe that the shape of the labyrinth needs not to be square or rectangular.

Here are the rules to follow:

- The cat must take the shortest way to reach the mouse.
- There is always at least one possible way for the cat to catch the mouse.
- If there are several ways, there will be of them one which will be shorter than all the others.
- The cat and the mouse move in turn, of only one box at the same time.
- the cat moves first.
- Valid displacements for the cat are: towards North, South, East or West, provided that such a displacement is possible (no wall in the way).
- Valid displacements for the mouse are: towards North, South, East, or West, provided that such a displacement is possible.
- The cat catch the mouse when both occupy the same box.
- With each displacement of the cat, the number of boxes between the cat and the mouse must decrease.

The input file

The file that you must read is "`P10.ENT` ". The input is divided into three parts.

The first part is a line containing three integers indicating respectively the positions of the cat, the position of the the mouse and the number of boxes of the labyrinth.

The second part contains the information about the layout of the labyrinth. The number of lines in this second part of the input file is equal to the number of boxes in the labyrinth. More precisely there is, in the second part, one line for each boxe. Such a line contains 5 integers: the number of the box, then the number of the adjacent box in the north direction, then the number of the adjacent box in the east direction, then the number of the adjacent box in the south direction, and finally the number of the adjacent box in the west direction. A zero (0) indicates that there is no exit in the corresponding direction.

The third part is the list of displacements of the mouse. On each line one will find an integer N, where $1 \leq N \leq 4$ to be interpreted the following way: North = 1, East = 2, South = 3, West = 4.

Each part is separated by an empty line, the data are separated by one or more spaces.

The number of displacements of the mouse contained in the file will be higher or equal to the number of displacements needed by the cat to catch the mouse.

▷

```
1 8 11

1 0 0 4 0
2 0 3 5 0
```

```
3 10 0 6 2
4 1 5 0 0
5 2 6 8 4
6 3 20 9 5
7 0 8 0 0
8 5 9 0 7
9 6 0 0 8
10 0 0 3 0
20 0 0 0 6

2
1
◁
```

Figure 10.1 shows the layout of the labyrinth according to the input file given as example.

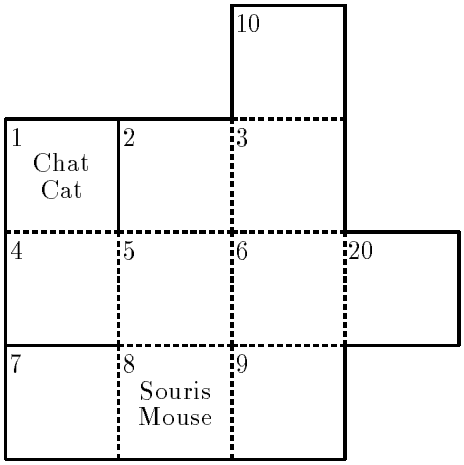

Figure 10.1: The labyrinth, the cat and the mouse with their starting position

The ouput file

You must write the file "`P10.SOR` " which will contain the list of boxes by which the cat will pass. The starting box and the box where the cat catches the mouse must be included. This list of boxes by which the cat passes must be the shortest one.

```
▷
1
4
5
6
```

◁

The cat starts at box 1 and the mouse starts at box 8. The cat being the first to move, it will move towards box 4. The mouse will carry out the first displacement contained in the file: 2 (East), which will bring it to box 9. The cat will move then to box 5. The mouse will carry out its second displacement: 1 (North) to find itself in box 6. Finally, the cat will move to box 6 where it catchs the mouse.

Marking

The problem will be corrected using four different input testing files.

A test will be regarded as successful if the program manages to find the solution and correctly write it in the output file.

Nombre de fichiers réussis	Points accordés
Successful test file	Points
1	240
2	360
3	480
4	600

Maximum time of execution of your program : 20 seconds.