Problem A
Program Modules

Input File: A.DAT
Program Source File: A.PAS or A.C or A.CPP

The relations $\rightarrow$ and $\Rightarrow$ are defined over the set of functions of a program.

- **A** $\rightarrow$ **B** means that the function **A** directly calls the function **B**;
- **A** $\Rightarrow$ **B** means that the function **A** calls the function **B** (directly or indirectly).

In addition, it is known that:

- if **A** $\rightarrow$ **B** then **A** $\Rightarrow$ **B**
- if **A** $\Rightarrow$ **B** and **B** $\Rightarrow$ **C** then **A** $\Rightarrow$ **C**

A computer aided software engineering system is able to group the functions of a program into modules according to the following rules:

1. If **A** $\Rightarrow$ **B** and **B** $\Rightarrow$ **A** then the functions **A** and **B** are grouped in the same module.
2. Any two functions which do not satisfy (1) must be in different modules.

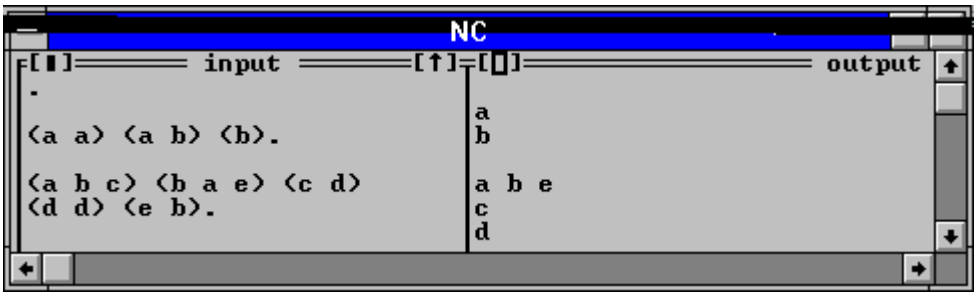

Figure 1. Input and output samples of the program

Write a program that is able to modularize a program according to the rules (1) and (2) given above. The program reads sets of data from a text file. Each set of data stands for a different program to be modularized and has the format:

(Fun$_1$ Fun$_{11}$ ... Fun$_{1n1}$) (Fun$_2$ Fun$_{21}$ ... Fun$_{2n2}$) ... (Fun$_m$ Fun$_{m1}$ ... Fun$_{m\ nm}$) .

where **Fun$_i$** and **Fun$_{ij}$**, **i = 1,m**, **j = 1,n$_i$**, **m,n$_i$** $\geq$ **0**, are letters (case sensitive) designating names of functions. The meaning of the term **(Fun$_i$ Fun$_{i1}$...Fun$_{ini}$)**, **n$_i$** $\geq$ **0**, is: **Fun$_i$**$\rightarrow$**Fun$_{ij}$**, **j = 1,n$_i$**. The data set is ended by a dot. Spaces, tabs and line breaks are used freely in the input. Input data are correct, i.e. each data set contains exactly one term **(Fun Fun$_1$...Fun$_n$)**, for each **Fun** of the program encoded by the data set, and **Fun$_i$** $\neq$**Fun$_j$** for **i** $\neq$ **j**, **i**,**j** **=1,n**.

The result of the program is on standard output. For each data set the program lists the function names of each computed module, one module per line. The names of the functions of a module are listed in ascending lexicographic order and are separated by single spaces. The modules are listed ascendingly according to the lexicographic order of their first component (function name). The output for the data set is followed by an empty line.

The example from figure 1 shows an input file, which contains three data sets, and the corresponding output. The first data set stands for an empty set of functions and the corresponding output is an empty line. For the second data set there are two singleton modules **a** and **b**, whereas for the third data set there are three modules **a b e**, **c** and **d**.

Eastern European Regional Programming Contest
Bucharest, Romania
25 October 1997

Problem B
Network Connections

Input File: B.DAT
Program Source File: B.PAS or B.C or B.CPP

Bob, who is a network administrator, supervises a network of computers. He is keeping a log connections between the computers in the network. Each connection is bi-directional. Two computers are interconnected if they are directly connected or if they are interconnected with the same computer. Occasionally, Bob has to decide, quickly, whether two given computers are connected, directly or indirectly, according to the log information. Write a program which based on information input from a text file counts the number of successful and the number of unsuccessful answers to the questions of the kind : *is* $computer_i$ *interconnected with* $computer_j$ *?*

input	output
10 c 1 5 c 2 7 q 7 1 c 3 9 q 9 6 c 2 5 q 7 5	1 , 2

Figure 1

The program reads data from a text file, as follows:

1. The number of computers in the network (a strictly positive integer);
2. A list of pairs of the form:
(a) ***c computer$_i$ computer$_j$*** , where $computer_i$ and $computer_j$ are integers from 1 to *no_of_computers*. A pair of this form shows that ***computer$_i$* and *computer$_j$* get interconnected.**

(b) ***q computer$_i$ computer$_j$*** , where $computer_i$ and $computer_j$ are integers from 1 to *no_of_computers*. A pair of this form stands for the question: ***is computer$_i$ interconnected with computer$_j$?***

Each pair is on a separate line. Pairs can appear in any order, regardless of their type. The log is updated after each pair of type (a) and each pair of type (b) is processed according to the current network configuration.

For example, the input file illustrated in figure 1 corresponds to a network of 10 computers and 7 pairs. There are N1 successfully answered questions and N2 unsuccessfully answered questions. The program prints these two numbers to the standard output on the same line, in the order: successful answers, unsuccessful answers, as shown in figure 1.

Problem C
Straightest Paths

Input File: C.DAT
Program Source File: C.PAS or C.C or C.CPP

A digitized map is represented on a two dimensional grid as a set of points enclosed within a contour. The closed contour of the map (which can be of any rectangular shape) and the obstacles on the map (inside the contour) are marked using the character with the decimal code 219. The free points on the map are designated by spaces. There are exactly two start-stop points on the map which are marked using the character with the decimal code 1. The start-stop points cannot appear outside the map or on the contour of the map. There are no other points on the map except free points (spaces), obstacles (decimal code 219) and start-stop points (decimal code 1).

A path between two map points runs vertically and/or horizontally through the free points on the map (inside the map contour). The *length* of the path is the number of points the path runs through, including the start-stop points. The *turns* of the path are the direction changes while traversing the path.

Write a program that, for each map read from a text file, finds all possible shortest paths with the minimum number of turns between the start-stop points on the map and marks the points of these paths using the character with the decimal code 176. All paths are marked on the same map. The start-stop points preserve their initial markings. For example, the first map in figure 1 has 6 shortest paths of 32 points long and with 3 turns each.

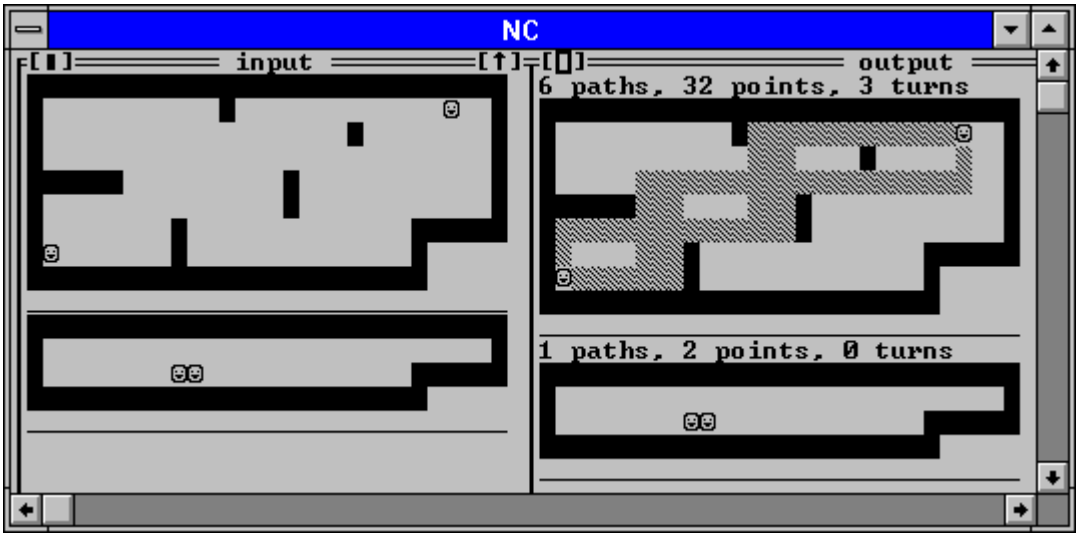

Figure 1. An input and output sample of the program

Each map is terminated by a line full of underscores _. There are at most 40 lines and at most 80 characters in a line for each map, including the termination line. The lines can be of different length. Input data are correct.

The standard output file contains the maps with the marked paths. Each map is output in the same format it is read from the input file, including the termination line, and is preceeded by a line which specifies the number of shortest paths, their length and turns as shown in figure 1. Figure 1 is an example of the program input and output for two maps.

Problem D
Sandorf's Cipher

Input File: D.DAT
Program Source File: D.PAS or D.C or D.CPP

Count Mathias Sandorf, a hero of Jules Verne, encrypted the secret messages of the Trieste conspirators by using a *cipher key* - a 6×6 hard paper square with some holes in it. In figure 1 the holes are marked white. One edge of the cipher key has a pinch-mark. Empty messages are coded as empty strings. For encrypting a non empty message follow the Sandorf's algorithm below.

1) Extend the message, by appending trailing **#** characters, until its length equals a multiple of 36, and then reverse the extended message. By convention, original messages do not end with **#** but, however, they can contain **#**. For example, the message:

```
Real Programmers use Fortran. Quiche Eaters use Pascal.
```

is 55 characters long and has to be extended by appending 17 trailing **#**. Right after step 1, the message is:

```
##################.lacsaP esu sretaE ehciuQ .nartroF
esu sremmargorP laeR
```

2) Position the cipher key on a sheet of paper, the pinch-mark facing upwards.

3) Copy the next 9 characters from the message (initially the first 9 characters) onto the paper through the holes of the cipher key, one character per hole, from left to right and top to bottom.

4) Turn the cipher key 90 degrees clockwise and repeat from step 3 until the key gets to its initial position, with the pinch-mark up. The result is a 6×6 matrix of characters, called a *coded group*.

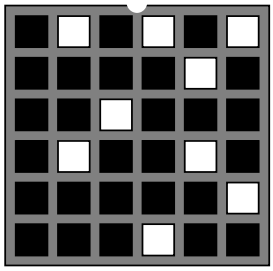

Figure 1. Sandorf's cipher key

5) Repeat from step 2 until all characters of the message are copied into coded groups, then append the rows of the coded groups from the first row of the first generated coded group to the last row of the last generated coded group, following the order of rows in each group and the order of groups. For the given message there are two coded groups, shown below,

```
u#l# #          geuhoc
as#r##           raPir
ec##st           sutrl
##aa##          rQae o
EP# ##          emFR .
 #e#s.          meanrs
```

which, when appended, yield the final encrypted message:

```
u#l# #as#r##ec##st##aa##EP# ## #e#s.geuhoc raPir sutrlrQae oemFR .meanrs
```

Write a program that decrypts messages encrypted by using the Sandorf's algorithm. The encrypted messages are read from a text file, one message per line ending with a line-break. Lines can be of different length. All characters of a line (except the line-break) are part of the encrypted message which cannot exceed 108 characters. The decrypted messages are printed on the standard output file. Each output line contains a message in its original form, starting from the beginning of the line. Decoding the above encrypted message yields the result:

```
Real Programmers use Fortran. Quiche Eaters use Pascal.
```


More examples of program input and output are illustrated in figure 2. Notice that there are two empty coded messages which correspond to empty original messages.

input	output
irdetn ihtoteao pesms soiaCet snaoi #e#r# edt#aasdrtltn ca reyor feneiv o#kginasksoaemlt f odatrnip as ene w#lerhiomfte t rSp se ntt.is id,raal o#r#i#etbanagvareioml nbfooheo lny e# #s#holp#lpt#iiok# w#so ts.tiis h H### ##d#l##a####n##o)##Dg#(##i#n#o#	Competition is a disease that sooner or later infects every trade and profession and makes it take a long step forward. Still, it remains the obligation of every honorable man to oppose it with his skills. (Donald Honig)

Figure 2. An input and output sample of the program

Southeastern European Regional Programming Contest
Bucharest, Romania
October 25, 1997

Problem E
Critical Links

Input File: E.DAT
Program Source File: E.PAS or E.C or E.CPP

In a computer network a link **L**, which interconnects two servers, is considered critical if there are at least two servers **A** and **B** such that all network interconnection paths between **A** and **B** pass through **L**. Removing a critical link generates two disjoint sub-networks such that any two servers of a sub-network are interconnected. For example, the network shown in figure 1 has three critical links that are marked bold: **0 - 1**, **3 - 4** and **6 - 7**.

It is known that: (1) the connection links are bi-directional; (2) a server is not directly connected to itself; (3) two servers are interconnected if they are directly connected or if they are interconnected with the same server; (4) the network can have stand-alone sub-networks.

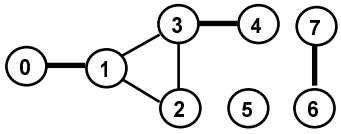

Figure 1. Critical links

Write a program that finds all critical links of a given computer network. The program reads sets of data from a text file. Each data set specifies the structure of a network and has the format:

no_of_servers
server$_0$ (no_of_direct_connections) connected_server ... connected_server
.....
server$_{no_of_servers-1}$ (no_of_direct_connections) connected_server ... connected_server

The first line contains a positive integer **no_of_servers** (possibly **0**) which is the number of network servers. The next **no_of_servers** lines, one for each server in the network, are randomly ordered and show the way servers are connected. The line corresponding to **server$_k$**, **0** ≤ **k** ≤ **no_of_servers-1**, specifies the number of direct connections of **server$_k$** and the servers which are directly connected to **server$_k$**. Servers are represented by integers from **0** to **no_of_servers-1**. Input data are correct. The first data set from figure 2 corresponds to the network in figure 1, while the second data set specifies an empty network.

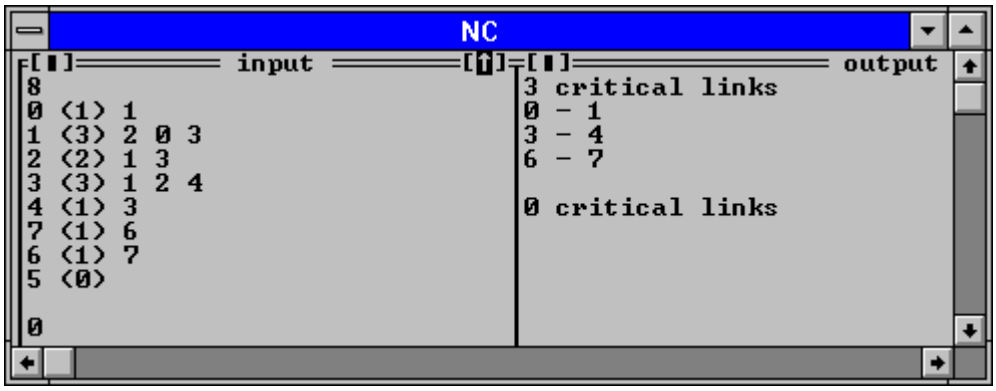

Figure 2. An example of program input and output

The result of the program is on standard output. For each data set the program prints the number of critical links and the critical links, one link per line, starting from the beginning of the line, as shown in figure 2. The links are listed in ascending order according to their first element. The output for the data set is followed by an empty line.

Problem F
Two Way Traffic

Input File: F.DAT
Program Source File: F.PAS or F.C or F.CPP

A motorway is passing through a tunnel, **d** meters long, as illustrated in figure 1. The rate of the traffic directed eastwards is of one vehicle every **t1** seconds, and the rate of the traffic directed westwards is of one vehicle every **t2** seconds. The vehicles going east have a constant speed of **v1** m/s, while the vehicles going west have a constant speed of **v2** m/s.

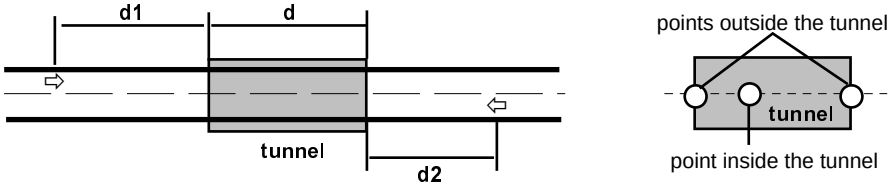

Figure 1 The motorway

Write a program that computes the number of vehicle rendezvous that occur inside the tunnel during the closed time interval **[ti,tf]** that is measured in seconds. It is known that:

- A rendezvous point is considered inside the tunnel if it is in between the entrances of the tunnel. Rendezvous at the tunnel entrances are outside the tunnel, as shown in figure 1.

- At time **0** there is a vehicle advancing eastwards that is **d1** meters away from the entrance of the tunnel, and there is another vehicle advancing westwards that is **d2** meters away from the entrance of the tunnel. These vehicles are part of the already existing traffic on the motorway.

- Input and output data are integers. The magnitude range of the problem parameters is as follows: **0** ≤ **d,d1,d2** ≤ **100000**, **0** < **v1,v2** ≤ **100**, **0** < **t1,t2** ≤ **10000**, **-10000** ≤ **ti,tf** ≤ **10000**.

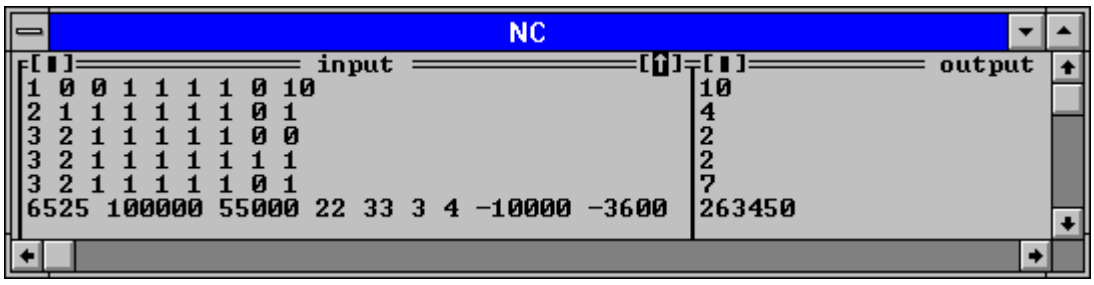

Figure 2. A sample of program input and output

The program reads sets of data from a text file. Each data set is on a separate line and has the format: **d d1 d2 v1 v2 t1 t2 ti tf**. The input data are correct. For each data set the program writes to the standard output the number of rendezvous. Each result is on a separate line. An example of program input/output for six data sets is shown in figure 2.

Problem G
Tile Puzzle

Input File: G.DAT
Program Source File: G.PAS or G.C or G.CPP

In a puzzle a rectangular surface of width **w** and height **h** is split into a collection of rectangular tiles. The tiles with the same dimensions form a similarity group. Solving the puzzle is to rebuild the surface by using the tiles from the collection. There can be many solutions. Two solutions are considered different if the tiles from at least one similarity group follow different placement patterns in the two solutions. For example, in the puzzle from figure 1 there are two groups of two similar tiles each. The tiles can be combined in eleven different ways to solve the puzzle.

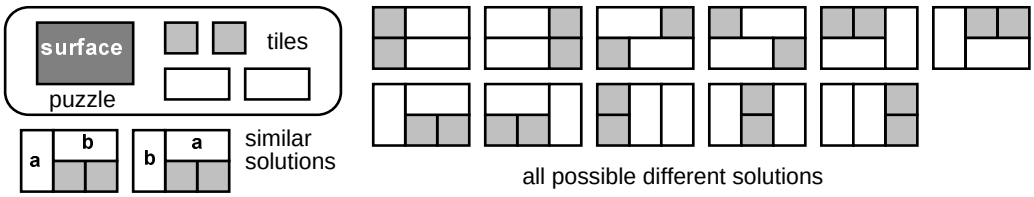

Figure 1. Combining tiles to rebuild a surface

Write a program that for a given puzzle, as described above, computes the total number of different solutions of the puzzle. The puzzle surface and the tiles have integer dimensions.

The program reads sets of data from a text file. Each data set, that describes a puzzle, has the format **w h n** m_1 w_1 h_1 ...m_n w_n h_n, where $0 < w,h \le 100$ are the dimensions of the surface to be rebuild, $0 < n \le 10$ is the number of groups of similar tiles, m_k is the number of similar tiles in the **k**-th group and w_k h_k are the width and the height of the tiles in the **k**-th group. Input data are correct, i.e. the tiles can be always combined to rebuild the surface and all tiles must be used in the process.

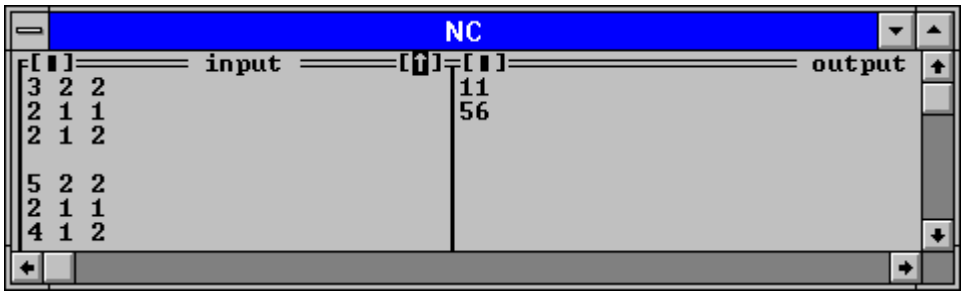

Figure 2. An input and output sample of the program

For each set of data, the program writes the number of tile combinations on a separate line. Figure 2 illustrates a sample of the program input/output. The first data set corresponds to the puzzle shown in figure 1.

Problem H
Safari Holiday

Input File: H.DAT
Program Source File: H.PAS or H.C or H.CPP

A safari holiday is organized for **n** people according to the following rules:

- During the safari, the participants are going out every day, in groups of exactly **k** persons, to enjoy wild-life watching in deep savanna.
- The number of persons in a group is the same every day and cannot exceed a maximum value **kmax** agreed by the safari organizers.
- For any given day, during the safari, each participant is member of a group in that day.
- Each day every person is in a totally new company, i.e. the remaining **k-1** persons of his/her group are different from those in the previous days.
- At the end of the safari everybody must have been in the company of everybody, i.e. for any two participants **A** and **B** there is a day, during the safari, such that **A** and **B** are in the same group that day.

For example, for **4** safari participants and for **kmax = 3** there are **2** groups of **2** persons each every day and the safari schedule can be as follows:

```
          group 1  group 2
day 1:     1,2      3,4
day 2:     1,3      2,4
day 3:     1,4      2,3
```

Write a program that, for a given pair **n**,**kmax** (the number of safari participants and the maximum size of the groups), computes the largest possible size of the safari groups (the largest possible value of **k** such that **k** ≤ **kmax**) and the corresponding number of days of the safari holiday.

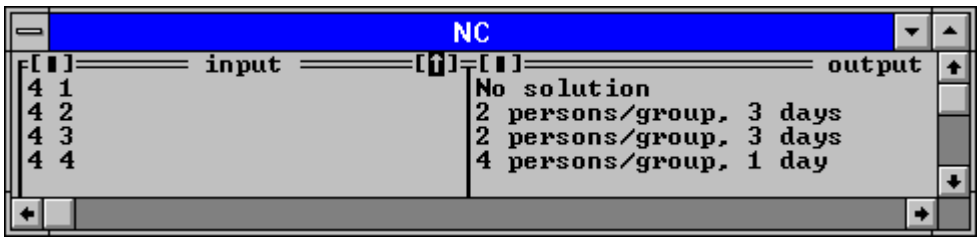

Figure 1. An example of program input and output

The program reads sets of data from a text file. Each data set is on a separate line and has the format: **n kmax**, where **n** and **kmax** are strictly positive integers, that can have large values. For each data set the program prints to the standard output the corresponding size of the safari groups and the number of days of the safari holiday, as illustrated in figure 1. If the problem has no solution the program prints the message **No solution**. Each result is on a separate line. An example of program input and output for four data sets is shown in figure 1.