ACM International Collegiate Programming Contest 2004

Brazil Sub-Regional

October 2rd, 2004

(This problem set contains 7 problems; pages are numbered from 1 to 17)

Hosted by:

Universidade Federal de Minas Gerais, Belo Horizonte, MG
Universidade Regional de Blumenau, Blumenau, SC
Universidade de Brasília, Brasília, DF
Pontifícia Universidade Católica de Campinas, Campinas, SP
Universidade Federal da Paraíba, Campina Grande, PB
Universidade Católica Dom Bosco, Campo Grande, MS
Universidade Estadual do Oeste do Paraná, Cascavel, PR
Universidade de Fortaleza, Fortaleza, CE
Universidade Federal do Amazonas, Manaus, AM
Faculdades COC, Ribeirão Preto, SP
Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ
Fundação Universidade Federal do Rio Grande, Rio Grande, RS
Centro Universitário 9 de Julho, São Paulo, SP
Centro Universitário Triângulo, Uberlândia, MG

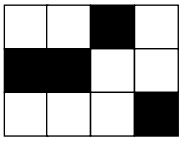
Problem A
Crossword With No Words

Although word square games go back to ancient times – a word square was found in the Roman ruins of Pompeii – it was only in 1913 that the Sunday New York World printed a puzzle called a 'word-cross' invented by Arthur Wynne, a journalist who had the job of devising a weekly puzzle for the comic section of the newspaper. The puzzle was an immediate success, became a weekly feature, and is nowadays probably the most popular and widespread word game in the world.

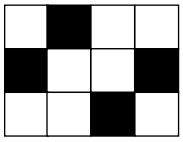
(For those weird people who do not know it, crossword is a puzzle in which a player must fill in words indicated by verbal clues down and across a checkered pattern so as to fit wherever they cross.)

A crossword configuration is the figure formed by empty squares and black squares in a puzzle. Over the first years several types of shapes and figures (diamond, circle, square) were tried before the familiar rectangular shape with a few black squares (used to separate words) was universally adopted. For this problem, we will define that a configuration for a puzzle with N lines and M columns is valid only if

- each column contains exactly one black square; and
- black squares are not in adjacent columns in the same line.



Invalid configuration



Valid configuration

Given a list of lengths of words, all of which must be put in the down (vertical) direction, your task is to find a valid configuration for a puzzle with N lines, M columns and M black squares.

Input

The input contains several test cases. The first line of a test case contains three integers N, M and K, indicating respectively the number of lines in the puzzle ($2 \leq N \leq 2000$) the number of columns in the puzzle ($1 \leq M \leq 2000$) and the number of lengths of words ($1 \leq K \leq 4000$). The second line contains K integers W_k, representing the lengths of words that must be put in the down (vertical) direction ($1 \leq W_k \leq N-1$). The end of input is indicated by $N = M = K = 0$.

The input must be read from standard input.

Output

For each test case in the input your program must produce an answer. The first line of an answer must contain a test case identifier, in the form '#i' where i starts from 1 and is incremented for every test case. Then, if there is a valid configuration for the puzzle, your program must produce M lines of output, describing one such configuration. Each line must contain two integers L and C, separated by a blank space, indicating the position of a black square (L indicates a line number and C indicates a column number, with $1 \leq L \leq N$ and $1 \leq C \leq M$). If more than one valid configuration is possible, print any one of those. If a valid configuration for the puzzle is not possible, your program must output as answer a single line containing the value 0.

The output must be written to standard output.

Sample Input	**Output for the sample input**
5 4 10 2 2 2 2 2 2 2 2 2 2 3 4 6 1 1 1 2 1 2 0 0 0	#1 0 #2 2 1 1 2 2 3 1 4

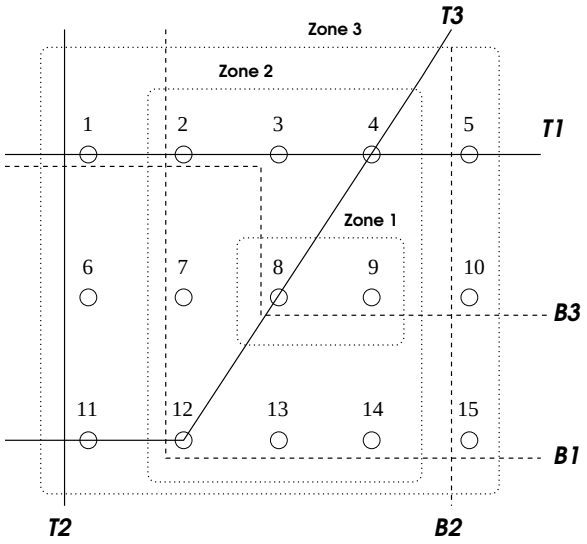
Problem B
GoEasy

The mayor of a city wants to introduce a new transport system to simplify the life of its inhabitants. That will be done via the use of a debit card, which the mayor named "GoEasy". There are two means of transportation in the city: trains and buses. The train system is "zone based" whereas the bus system is "journey based". The fare for a journey is computed as follows:

- There is an initial *two money units* fare for entering the transport system, regardless of the initial means of transportation.
- When travelling by train a customer pays *four money units* for each change of zone.
- When travelling by bus a customer pays *one money unit* each time she/he boards a bus.

A transport system map will provide information about the stations belonging to each zone, and the sequence of stations for each bus and train itinerary. Buses and trains move in both directions in each itinerary, and no train or bus goes through the same station twice during a single trip through an itinerary. It is always possible to go from any station to any other station using trains and/or buses.

The rules for computing fares are strict: if during a train journey a customer enters a given zone twice, she/he is charged twice; similarly, if during a bus journey a customer boards twice the bus for the same itinerary, she/he is charged twice.



Sample Transport Map

In the transport map above a customer can travel from station 2 to station 4 paying just two money units, by using line T1, since they are in the same zone. But if the customer needs to go from station 2 to 5, then the best is to take the bus B3 to station 10 and then take the bus B2 to station 5, paying in total four money units.

Rather than tracking the whole trip of each passenger, the idea of the mayor is that machines will be placed in all stations, and travellers are supposed to swipe their personal GoEasy travel card only when starting AND finishing the whole journey. Since all the machines are interconnected into a network, based on the departure and arrival stations the system can compute the minimum cost possible for the trip, and that amount is charged from the traveller's debit card. All that is missing is a computer system for doing the calculations for the fare to be deducted. So, given the map of the transport system in the city, you must write a program to compute the minimum fare the customer should pay to travel between two given stops/stations.

Input

The input contains several test cases. The first line of a test case contains two integers Z and S, which indicate respectively the number of zones ($1 \leq Z \leq 30$) and the number of train/bus stations in the city ($1 \leq S \leq 100$). Each station has a unique identification number ranging from 1 to S, and each station belongs to exactly one zone. Each of the following Z lines describes the stations belonging to a zone. The description for a zone starts with an integer K indicating the number of stations ($1 \leq K \leq S$) in the zone, followed by K integers representing the stations in the zone. After that comes a line with two integer numbers T and B, representing respectively the number of train itineraries ($1 \leq T \leq 50$) and the number of bus itineraries ($1 \leq B \leq 50$). Next comes T lines describing train itineraries, followed by B lines describing bus itineraries. The description of each itinerary consists of a line containing an integer L indicating the number of stations ($2 \leq L \leq S$) in the itinerary, followed by L integers specifying the sequence of stations in the itinerary. Finally it comes a line with two integers X and Y ($1 \leq X \leq S$, $1 \leq Y \leq S$ and $X \neq Y$), specifying that the customer travelled from station X to station Y. The end of input is indicated by $Z = S = 0$.

The input must be read from standard input.

Output

For each test case your program should output one line, containing an integer representing the amount to be deducted from the traveller's GoEasy card.

The output must be written to standard output.

Sample Input	**Output for the sample input**
3 15 2 8 9 7 2 3 4 7 12 13 14 6 1 5 6 10 11 15 3 3 5 1 2 3 4 5 3 1 6 11 4 4 8 12 11 6 2 7 12 13 14 15 3 5 10 15 6 1 2 3 8 9 10 11 6 3 15 2 8 9 7 2 3 4 7 12 13 14 6 1 5 6 10 11 15 3 3 5 1 2 3 4 5 3 1 6 11 4 4 8 12 11 6 2 7 12 13 14 15 3 5 10 15 6 1 2 3 8 9 10 11 5 0 0	2 4

Problem C
Roman Patrollers

In ancient times, patrollers were used to ensure that all the cities of the Roman Empire were under control. A patroller's job consisted in continuously visiting the cities of the empire, trying to minimise the interval between two visits to each city. The Military Society (MS) wants to simulate the behavior of one such patroller to see how effective they were.

Each cycle of the simulation corresponds to one time unit. The *instantaneous city idleness (ICI)* for a city X after T cycles of the simulation is the number of cycles elapsed since the last visit of the patroller to the city X (i.e. the number of time units the city X remained unvisited). All of the cities have initial instantaneous city idleness equal to zero at the start of the simulation. The *instantaneous empire idleness (IEI)* after each given cycle is the sum of the *instantaneous city idleness* of all cities after that given cycle. Finally, the *empire idleness (EI)* for an N-cycle simulation is the sum of the *instantaneous empire idleness* after each of the N cycles of simulation.

After visiting a city X, the patroller always chooses to visit the *neighbour city* Y with the highest instantaneous city idleness (if more than one city has the highest idleness, the one with the lowest identifier is chosen). Cities X and Y are *neighbour* if there is a road linking the two cities directly, without going through any intermediate city. In the beginning of the simulation, the patroller is located in one of the cities, and is given a map of the Roman Empire containing a description of all the roads in the empire, indicating the length (in kilometers) and which two cities each road connects. A road between cities X and Y can be used both to go from X to Y and from Y to X.

Assuming that a patroller travels one kilometer in one time unit (one simulation cycle) and that the time to visit a city is negligible (equal to zero), MS asks you to determine the empire idleness after an N-cycle simulation.

For clarity, consider the example of an empire which contains 3 cities (1, 2 and 3) and two roads of length 1 km. The first road connects cities 1 and 2, while the second road connects cities 2 and 3. Below you find a trace of a 3-cycle simulation for such a scenario, considering that the patroller starts at city 1.

```
Start of the simulation
Patroller at: 1
  ICI1 = 0, ICI2 = 0, ICI3 = 0
  IEI = 0
  EI = 0

After cycle 1
  Patroller at: 2
  ICI1 = 1, ICI2 = 0, ICI3 = 1
  IEI = 2
  EI = 2

After cycle 2
Patroller at: 1
  ICI1 = 0, ICI2 = 1, ICI3 = 2
  IEI = 3
  EI = 5

After cycle 3
  Patroller at: 2
  ICI1 = 1, ICI2 = 0, ICI3 = 3
  IEI = 4
  EI = 9
```

Therefore, for such a scenario, after 3 simulation cycles the empire idleness is 9.

Input

The input consists of several test cases. The first line of a test case contains four integers C, R, N, and S, indicating respectively the quantity of cities in the empire ($2 \leq C \leq 1000$), the number of roads ($1 \leq R \leq C(C-1)/2$), the number of cycles to be simulated ($1 \leq N \leq 1000$) and the identifier of the starting city of the patroller ($1 \leq S \leq C$). Each city is identified by a distinct integer from 1 to C. Each of the following R lines contains three integers X, Y and D describing a road; X and Y represent cities ($1 \leq X \neq Y \leq C$) and D represents the distance ($1 \leq D \leq 1000$), in kilometers, of the road that connects X and Y directly, without passing through any other city. Each pair of cities X and Y will appear at most once in a road description. You can assume that it is always possible to travel from any city to any other city in the empire using the roads available. The end of input is indicated by $C = R = N = S = 0$.

The input must be read from standard input.

Output

For each test case in the input, your program must produce one line containing the empire idleness after the N-cycle simulation.

The output must be written to standard output.

Sample Input	**Output for the sample input**
2 1 1 1 1 2 2 2 1 2 1 1 2 2 2 1 3 1 1 2 2 2 1 4 1 1 2 2 3 2 3 1 1 2 1 2 3 1 0 0 0 0	2 4 8 10 9

Problem D
Very Special Boxes

Special Box Company (SBC) is a small family-owned and family-run business which produces decorated carton boxes for wrapping gifts. The boxes are hand-made, produced individually from fine materials. When accepting an order from a client, they always produce a few more boxes than needed, to keep a stock of boxes to be sold in the future, if needed. Over the years their stock has been growing, with boxes all over the place, and they decided they needed to organize it a bit more. They have therefore made a list registering the dimensions of every box in their stock.

SBC has just received an order from a client that must be delivered tomorrow, so there is no time to produce new boxes. The client wants a certain number N of boxes *all of the same size*; each box will be used to pack one item of dimensions X, Y and Z. As the carton used in the boxes is very thin, you may assume that a box of size (X, Y, Z) would fit perfectly the item the client wants to wrap. If there are not at least N boxes that fit perfectly, the client wants N boxes that fit the items as tightly as possible. The box size that fits the items as tightly as possible is the one which minimizes the empty space when the item is put inside the box. An item can be rotated in any direction to be accomodated inside a box; therefore, a box of size (X, Y, Z) is as good as a box of size (Y, Z, X), for example.

Can you help SBC finding whether they can fulfill the customer order?

Input

The input consists of several test cases. The first line of a test case contains two integers N and M, indicating respectively the number of boxes the client needs to buy ($1 \leq N \leq 1500$) and the number of boxes in the stock list ($1 \leq M \leq 1500$). The second line contains three integers X, Y and Z, representing the dimensions of the item the client wants to wrap ($0 < X, Y, Z \leq 50$). Each of the next M lines contains three integers A, B and C representing the dimensions of a box in the stock list ($0 < A, B, C \leq 50$). A test case with $N = 0$ indicates the end of the input.

The input must be read from standard input.

Output

For each test case in the input your program must produce one line, containing either:

- the single word 'impossible', in case it is not possible to fulfill the client's order (because there are not at least N boxes of the same size in stock that can contain the item); or
- one integer V, which specifies the volume of empty space left when one of the N items is packed in one of the boxes chosen.

The output must be written to standard output.

Sample Input	**Output for the sample input**
1 1 2 4 3 2 3 4 2 6 3 1 3 7 4 7 10 8 2 2 8 10 6 2 9 7 7 4 6 2 9 1 1 3 3 3 1 1 1 0 0	0 99 impossible

Problem E
Drop Out

Drop Out is the name of a simple card game which is played with a normal deck of 52 cards. Cards are ordered by rank (Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jockey, Queen, King), with Ace being the smallest and King the largest value. Card suits are disregarded. Players (at least two) sit around a table and a shuffled deck is put in the center of the table, card faces down.

At the start of the game, all players are "active". The game proceeds in rounds. In each round, active players are dealt one card from the deck, in clockwise direction regarding their sitting positions. The players who are dealt the smallest card in the round drop out of the game and become inactive. Notice that up to four players may drop out at each round. The game ends when there remains one only active player, which is the winner. If the entire deck is played out before a round finishes, the game is over and all active players at the beginning of this last round are winners.

Given the number of players, their names and a shuffled deck of cards, you must write a program to simulate the game and determine the winner or winners.

Input

The input contains several test cases. Each test case consists of six lines. The first line contains an integer N , indicating the number of players in the game ($2 \leq N \leq 20$). The second line contains a list of player names, separated by spaces. A player name is composed of at most 16 letters from the English alphabet (from 'A' through 'Z' and 'a' through 'z'). Cards are dealt to players in the order given by the list. The next four lines contain the description of the shuffled deck. Card ranks are represented by integers from one to thirteen (1, 11, 12 and 13 denote respectively Ace, Jockey, Queen and King cards). The deck is described in four lines of thirteen integers each, separated by a single space. The deck is listed from top to bottom, so the first card dealt is the first card listed. The end of input is indicated by $N = 0$.

The input must be read from standard input.

Output

For each test case in the input your program must produce one line of output, containing the name of the winner or winners. The list of winners must appear in same order given in the input, and each name must be followed by a space.

The output must be written to standard output.

Sample Input	**Output for the sample input**
4 Sally Claire Mary Beatrice 1 2 3 4 5 6 7 8 8 10 11 12 13 1 2 3 4 5 6 7 9 9 10 11 12 13 1 2 3 4 5 6 7 8 9 10 11 12 13 1 2 3 4 5 6 7 8 9 10 11 12 13 6 Aline Barbie Helen Julia Mary Sally 10 5 9 7 6 10 2 13 1 8 11 12 11 7 11 4 13 4 9 6 8 13 11 2 1 5 9 6 5 3 9 4 1 12 12 13 6 1 10 3 2 7 7 2 4 8 10 5 3 8 3 12 0	Mary Beatrice Helen

Problem F
Hurry Up!

Orienteering, a cross-country race on foot where competitors receive a map and a compass, is a sport very popular in some european countries. Johnny and his friends entered an orienteering competition and intend to win it.

In this competition, each team member wears a different color and starts from a different place. There are some finishing points, each one with a list of colors it "accepts". Every competitor in a team must go from its starting place to one of the finishing places which accept its color. No team member can go to the same finishing place of another member. The team penalty in the game is the sum of the time team members take from their starting to their final places.

To maximize the chances of winning, Johnny and his team members want to determine the most appropriate finishing point for each member of the team, assuming he and his friends advance at possibly different speeds. That is, they want to determine for each team member one different finishing point, so that the total time penalty for the team will be minimized.

You may assume that there will always be an answer (a different finishing point for each team member).

Input

Your program should process several test cases. The first line of a test case contains two integers N and M, representing respectively the number of players in the team and the number of existing finishing points ($1 \leq N \leq M \leq 100$). The next N lines contain each two integers X and Y representing the starting position of a player ($-20000 \leq X, Y \leq 20000$) and a real s, representing the player's speed. Players are identified by the order their starting position appear on the input (the first to appear is number 1, the second is number 2, and so on). These identification numbers are also used to represent the players colors. The following M lines contain each two integers X and Y defining the position of a finishing point ($-20000 \leq X, Y \leq 20000$) and a list of colors Ci accepted by that finishing point ($1 \leq Ci \leq N$); the end of the list is indicated by a value of 0 (zero). The end of input is indicated by $N = M = 0$.

The input must be read from standard input.

Output

For each test case, your program should output a single line, containing a real value representing the minimal time penalty, i.e. the minimal sum of time taken by the players to reach their respective finishing points. Your answers must be rounded to one digit after the decimal point.

The output must be written to standard output.

Sample Input	**Output for the sample input**
1 1 0 0 1.0 1 1 1 0 2 3 100 100 1.0 100 200 1.0 110 100 1 2 0 110 200 1 2 0 200 250 1 0 1 2 0 0 1.0 11111 11111 1 0 11111 -11111 1 0 0 0	1.4 20.0 15713.3

Problem G
Chemistry

International Chemical Products Company (ICPC) is a company known world-wide for its good and affordable products, which include shampoos, cleaning products, bug-killing products, and even some types of vaccines. The ICPC engineers are always researching new ways of reducing their products' manufacturing costs, without lowering their quality. One of their engineers, Mr. Poucher, has a new idea to reduce the cost, which aims at reducing the number of containers necessary to hold the substances during the sequence of chemical reactions to obtain a final substance. These final substances are obtained through a sequence of reactions of the form X + Y $\rightarrow$ Z, where X and Y are either initial substances or intermediate substances that were already generated from previous reactions. These reactions are done inside a *reaction container*, which once emptied can be cleaned up and used again. The process for generating a final substance can be described via a sequence of two simple operations:

- put an available substance in an empty reaction container C;
- perform the reaction X + Y $\rightarrow$ Z either by putting X in the reaction container holding Y, or by putting Y in the reaction container containing X. The order does not affect the end result of the reaction.

What Mr. Poucher noticed was that by choosing smartly the sequence of reaction, ICPC could drastically cut out on the number of reaction containers needed in the company. For example, consider the following sequence of chemical reactions used to obtain final substance P:

```
1) A + B -> T1
2) C + D -> T2
3) E + F -> T3
4) T2 + T3 -> T4
5) T4 + T1 -> P
```

In this example, A, B, C, D, E, and F are the *initial substances* (only appear on the left side of reactions), T1, T2, T3 and T4 are the *intermediate substances* (appear on the left side of at least one reaction, and exactly once on right side of some other reaction) and P is the *final substance* (only appears on the right side of a single reaction, which will be the last one listed). If the sequence of reactions is performed as given then three reaction containers are necessary in order to produce the final substance P:

```
                Containers
Operations      C1   C2   C3
put A in C1:    A    -    -
add B to C1:    T1   -    -
put C in C2:    T1   C    -
add D to C2:    T1   T2   -
```

```
put E in C3:   T1   T2   E
add F to C3:   T1   T2   T3
put T2 in C3:  T1   -    T4
put T4 in C1:  P    -    -
```

Note, however, that if the reactions are performed in the sequence 2, 3, 4, 1, 5, two reaction containers are sufficient:

```
                Containers
Operations       C1   C2
put C in C1:     C
add D to C1:     T2
put E in C2:     T2   E
add F to C2:     T2   T3
put T2 in C2:    -    T4
put A in C1:     A    T4
add B to C1:     T1   T4
put T1 in C2:    -    P
```

You have been hired by ICPC, and your task is to create a computer program to determine the minimum number of reaction containers necessary to perform the sequence of reactions needed to obtain the final substance.

You should assume that:

- The reaction producing the final substance is the last listed, and the reaction producing an intermediate substance will always precede reactions where that intermediate substance is used.
- A sequence of reactions producing the final substance is always possible.
- ICPC has an unlimited supply of initial substances.
- At the beginning of the production process, each initial substance is in a *storage container*, used to hold all ICPC stock of this substance. Such containers cannot be used as *reaction container* to hold intermediate products of reactions.
- All the reaction containers are large enough to hold all the resulting substances.
- The amount of substance generated by a single reaction is just enough to be used as input to a single other reaction. For instance, if an intermediate product Z is necessary as input for two different reaction, this product must be produced twice.
- Every reaction uses exactly two distinct substances and generates also a distinct substance, i.e. all the reaction have the form X + Y $\rightarrow$ Z, where X, Y and Z are all distinct.

Input

The input consists of several test cases. Each test case starts with a line containing a single integer R, indicating the number of reactions to be considered ($1 \leq R \leq 5000$). The following R lines are of the form:

```
S1 + S2 -> S3
```

describing a reaction that consumes S1 and S2 and produces S3 as a result. The names of all substances are case-sensitive alphanumeric strings of size at most 5. A test case with $R = 0$ indicates the end of the input.

The input must be read from standard input.

Output

For each test case in the input your program must produce one line, containing the string '`PRODUCT requires N containers`', where `PRODUCT` is the final substance and `N` is the number of containers needed to produce it.

The output must be written to standard output.

Sample Input	**Output for the sample input**

`1`
`2H + O -> Water`
`5`
`A + B -> T1`
`C + D -> T2`
`E + F -> T3`
`T2 + T3 -> T4`
`T4 + T1 -> P`
`3`
`a + b -> ab`
`ab + c -> abc`
`abc + d -> abcd`
`0`
	`Water requires 1 containers`
	`P requires 2 containers`
	`abcd requires 1 containers`