1998-99 ACM International Collegiate Programming Contest

Sponsored by IBM

Asia Regional Contest/Shanghai

Shanghai University, Shanghai

November 28, 1998

This problem set should contain eight (8) problems on seventeen (17) numbered pages. Please inform a runner immediately if something is missing from your problem set.

1998-99 ACM International Collegiate Programming Contest
Asia Regional Contest/Shanghai

PROBLEM A
Card Shuffling
Input file: shuffle.in

Suppose we have $2n$ cards numbered 1, 2, …, n, $n+1$, …, $2n$, and this is also its original sequence. One card shuffle is the process of making the original sequence become $n+1$, 1, $n+2$, 2, …, $2n$, n. That is to put the first n cards to positions 2, 4, …, $2n$. And the rest n cards will be put to, in accordance with their original sequence, odd numbered positions 1, 3, ? , $2n-1$. It has been proved that for any natural number n, the original sequence can be regained after a certain number of card shuffles. Nevertheless we don't know exactly how many times we need to shuffle the cards to return to the beginning sequence for a specific number n, and we ask for your help.

Input
A sequence of natural numbers each representing the number n mentioned above (that is $2n$ cards). This sequence is terminated by the number 0. Notice each n will be small enough to guarantee not too long total shuffling time. ($n < 10000$)

Output
For each number in the input file, report the total number of shuffling required to regain the original sequence.

Sample Input
```
10
20
0
```

Output for the Sample Input
```
The number of shuffling for n  = 10 is: 6
The number of shuffling for n = 20 is: 20
```

1998-99 ACM International Collegiate Programming Contest
Asia Regional Contest/Shanghai

PROBLEM B

Puzzle

Input file: puzzle.in

Do you remember a child game -- puzzle -- a 4*4 square with numbers from 1 to 15 and an empty place? This problem is ask you to write a program to solve this problem.
We have a baby. It likes to play this game. He couldn't find any way to solve the puzzle if someone made the puzzle square muss, then cry loudly. This is so bad for a programmer who wants to work quietly.
A puzzle is a square with 15 blocks marked numbers from 1 to 15 and an empty place for player to move it from one position to the other. If blocks in their very positions are as the graph shown below, we call the puzzle is sequenced (also called puzzle problem solved).

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	▨

At the beginning time, a player is given a non-sequenced square. He (or she) tries to find a way to move blocks then make the puzzle sequenced. Players can move one numbered block below, upper, right, or left to the empty place in each step. If a player moves a numbered block left to the empty place to the empty position, we also say he (or she) moved the empty place left one step.
For example, we have a square at beginning time as below:

1	2	3	4
5	6	7	8
9	10	15	11
13	▨	14	12

Now, if we moved the empty place in below ways, we would have the square sequenced.
Right, Up, Right, Down
You are asked write a program to solve this problem

Input

The input file has two parts. The first part contains an integer *n* in a single line representing the number of puzzle cases in this file. The second part contains 4**n* lines representing all data of puzzle cases. Data of one puzzle case include four lines. The first line represents the first row in original square, and so on. There are four integers in lines of the second part. Each integer represents the block number in its such position, 0 for the empty place. You should assume that it always has a way to solve the puzzle problem case proposed in our input file.

Output

You should output the way to make a square sequenced. Integers 0, 1, 2, and 3 are representing acts to move the numbered block below, left, upper, and right to the empty place. Integer -1 represents the end of your answer for one puzzle problem case. Any more output will result in a message ‘Wrong Answer’ from the judge.

Assumption

An answer would get ‘Correct Answer’ from the judge if it could make the scrambled square sequenced, even it contains a cycle moving sequence such like 0, 1, 2, 3.

Sample Input

```
1
1 2 3 4
5 6 7 8
9 10 15 11
13 0 14 12
```

Output for the Sample Input

```
3 2 3 0 -1
```

1998-99 ACM International Collegiate Programming Contest
Asia Regional Contest/Shanghai

PROBLEM C
Container
Input file: containe.in

Recently, Innovcompu Company has produced a new kind of computer named ACM-Control. One ACM-Control actually consists of one host and two terminals. The weight of the host is A (in kg), and the weight of each terminal is B (in kg). One container can accommodate some hosts and some terminals, but the total weight of these hosts and terminals should not be more than the loading capacity of this container. In this case, we suppose the volume of the container is large enough. The manager of Transportation Company wants to use N available containers to accommodate as many ACM-Controls as possible, which will minimize the transportation cost for each ACM-Control.

You are required to write a program to get an optimal plan to accommodate ACM-Controls with the given containers. A plane is optimal if it can accommodate the maximum number of ACM-Controls.

Input

Contains 3*k+1 lines, with k test cases. The end of file is a star symbol (*).

Line 1	A B	Two integers, represented the weight of each host and each terminal in kg, case 1
Line 2	N	The number of containers(0<N<200), case 1
Line 3	P_1 P_2 ? P_N	N integers, represented the loading capacity of N containers in kg, case 1 (0 < Pi < 1000, 1 < Pi/A < 50, 1 < Pi/B < 50)
?		
Line 3*k-2	A B	Two integers, represented the weight of each host and each terminal in kg, case k
Line 3*k-1	N	The number of containers(0<N<200), case k
Line 3*k	P_1 P_2 ? P_N	N integers, represented the loading capacity of N containers in kg, case k (0 < Pi < 1000, 1 < Pi/A < 50, 1 < Pi/B < 50)
Line 3*k+1	*	The sign of file end

Output

Contains k lines.

Line 1	The maximum number of ACM-Controls accommodated by the containers of case 1
Line 2	The maximum number of ACM-Controls accommodated by the containers of case 2
?	
Line k	The maximum number of ACM-Controls accommodated by the containers of case k

Sample Input

```
5 3
2
10 10
4 4
3
12 13 11
*
```

Output for the Sample Input

```
1
2
```

PROBLEM D
The Dream Team
Input file: team.in

1998 years after Mesiah's birth, a joke happens on the earth. That is, the Crazil football team lost in a queer world final match. Being a programmer, you decide to write a program to find out why such thing happens, you are going to calculate the weariness of Crazilians or find out the probability of some other reasons.
Oh, that's too complex, fortunately you don't need to write that program now. Here is the leading reason, Mario Argalo, the coach of Crazil team, not selected a perfect team. It is a hard work to select 22 men out of so many great players, it is even harder since old Argalo hasn't the ability-data of players onhand. Though somebody can say the team is not a real team, it seems more like 22 individual person, but this is not the leading reason. Argalo can achieve his fifth world champion if he select 22 top Crazilian guys and giving them two months for training, but he even chose someone with much honor but low ability. Anyway, we can select Juninho, Muller, and Savio in our sequad.
OK, now Muxemburg is the new coach. In last 5 years, he marked every player with an ability value (b,m,f), b represents the ability that the person play as a backward, while m for middlefield, f for forward. Thus, Muxemburg's first task is to select a perfect team base on the ability values. The squad contain 22 players, one team member act as a backward, middlefield or forward, but can't have two positions. Muxemburg calculate the value of whole team like this way, for every person, his contribution to the whole team value is his ability value in his position, that is, in (b,m,f), a backward contributes b, middlefield m, and forwards f. The goal is to make a maximum-value team.
There are so many good player in Crazil, so Muxemburg asked you to write a program helping him. The program must read the numbers of players in each position, and the ability value for every candidate, then make the great team in Brazilian history.

Input
Input file is a single data set of this problem. three non-negative numbers (B,M,F) in first line, the number of players Muxemburg want on each position, B for backward, M for middlefield, F for forward, with B+M+F=20.(remember that two position reserved for goalkeeper) The second line is a single positive integer N>20, the number of candidates. Then next N lines each give (name, b,m,f), the name and ability value of a certain person. b, m or f is an integer within 0 and 100. The name is a string no longer than 20 charactors, the valid charactors in that string are English letters and digits. Although there are many players with the same name in Crazil, like Carlos or Silva, in the input file there will never be such two persons. There may be at most 5000 persons in the input file.
Data items in the same line is seperated by one or more blanks.

Output

Output should be 27 lines. First line reads "Backward:", followed by B lines each give a backward's name, a blank line, next line reads "Middlefield", followed by M lines for M middlefields, also a balnk line, then "Forward" and corresponding F lines, another blank line, last line reads "Team Value = v", where v is replaced by the maximum team value. See example.

Sample Input

```
6 7 7
24
Carlos 94 80 90
Ardair 85 55 73
ZeCarlos 83 45 75
Silva1 75 80 80
Cafu 85 85 87
Lois 73 75 62
Concalv 80 75 40
Juninho 70 90 85
Flavio 73 90 80
DeJaminha 60 95 80
Pangza 0 100 0
Rivaldo 70 85 70
Denelson 60 100 85
Dunga 75 87 74
Edmundo 50 70 95
Muller 73 83 93
Savio 70 84 95
Leonardo 65 89 79
Rai 50 85 85
Etin 70 80 100
Ronaldo 40 80 100
Dodo 50 70 87
Romario 45 85 98
Carioca 70 70 94
```

Output for the Sample Input

```
Backward:
Carlos
Cafu
Ardair
Concalv
ZeCarlos
Silva1

Middlefield:
Juninho
Flavio
DeJaminha
Pangza
Denelson
Dunga
Leonardo

Forward:
Ronaldo
Edmundo
Etin
Muller
Savio
Romario
Carioca

Team Value = 1828
```

PROBLEM E

Data Mining

Input file: mining.in

Data mining is a process of discovering hidden information from databases. One goal is to mine functional dependencies. Let R(A_1, …, A_n) be a relation scheme, and let X be a subset of { A_1, …, A_n }. We say X->A_i, read "X functionally determines Ai" or "Ai functionally depends on X" if whatever relation *r* is the current value for R, it is not possible that *r* has two tuples that agree in the components for all attributes in the set X yet disagree in values for A_i. Notice functional dependency is a semantic term. Yet in this problem you are to report all the functional dependencies based on the current value of a relation. To complicate the problem, there are certain kinds of functional dependencies you should not report. The first one is trivial dependencies. A functional dependency X->A_i is trivial if A_i is an attribute in X, say A_1A_2->A_1. The second kind of functional dependencies you should avoid is the redundant ones. A functional dependency X->A_i is redundant if a subset of X has already functionally determined Ai. For example, if A_1->A_2, then certainly A_1A_3->A_2. Now, to make the description clearer, let see an example. Suppose we have a simple relation with attributes A and B which has two tuples:

```
A B
0 0
0 1
```

We don't have A->B because the two tuples agree on value A yet disagree on B. However we do have B->A because we can't find any two tuples which have the same value on B but are different on A.

Input

The first line contains two integers *r* and *c* which are the number of rows and columns of the following relation. The second line has *c* letters separated by blanks. What follows is a matrix of integers having *r* rows and *c* columns, and each column is named by the corresponding letter in the second line. This pattern may occur several times until two zeroes are in the place of number of rows and columns.

Output

You should report on each input relation all the functional dependencies beginning with a line indicating its sequence in the input relations. The sequence of the functional dependencies in each relation is not important.

Sample Input

```
3 3
A B C
0 0 0
0 1 0
1 1 1
0 0
```

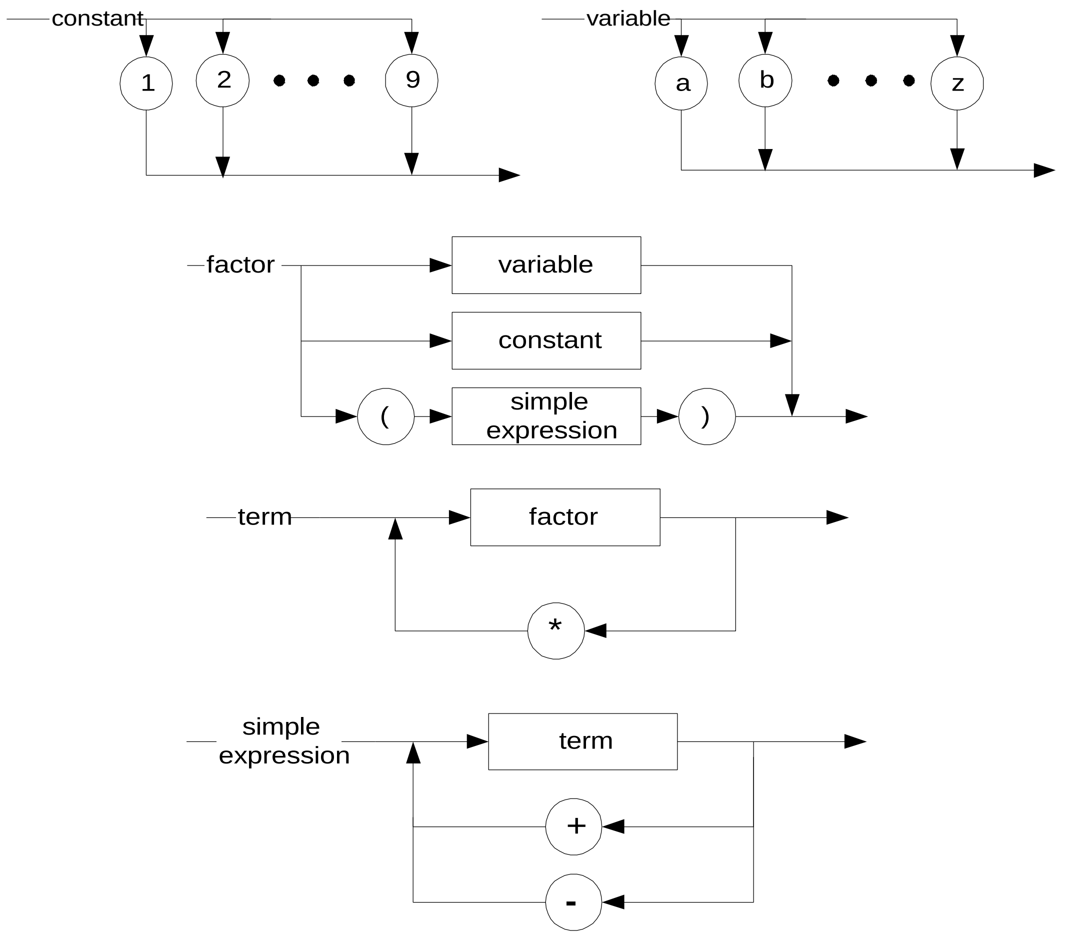
Output for the Sample Input

```
Functional dependencies for relation 1:
A->C
C->A
```

PROBLEM F
Simple Expression
Input file: simple.in

Mathematicians often need to determine if two algebraic expressions are equal, so they want a software which can do it. ACM has devoted to this hard project. They first begin the research of ‘Simple Expression Equation’.

Simple Expression is a predigestion of the normal algebraic expression, which its declaration is such as below:



If one Simple Expression can be transformed to another through equation transformation, then we say the two Simple Expressions are equal. For example, `(3*a+6*c)*a-c*a*4` and `a*(2*c+3*a)` are equal, but `a*a*(1+c)` is not equal to `a*a+c`.

You are required to write a program to determine if two Simple Expressions are equal.

Input

Contains 2*k+1 lines, with k test cases. Each Simple Expression contains no more than 255 characters.

Line 1	Integer k , represented the number of test cases in the input files
Line 2	The first Simple Expression of case 1
Line 3	The second Simple Expression of case 1
?	
Line 2*k	The first Simple Expression of case k
Line 2*k+1	The second Simple Expression of case k

Output

Contains k lines. Each line is the judgement of the k^{th} pair of Simple Expressions. If the two Simple Expressions are equal, the judgement is "equal", otherwise " not equal"

Line 1	Judgement
?	
Line k	Judgement

Sample Input

```
3
(3*a+6*c)*a-c*a*4
a*(2*c+3*a)
a*a*(1+c)
a*a+c
1
((1))
```

Output for the Sample Input

```
equal
not equal
equal
```

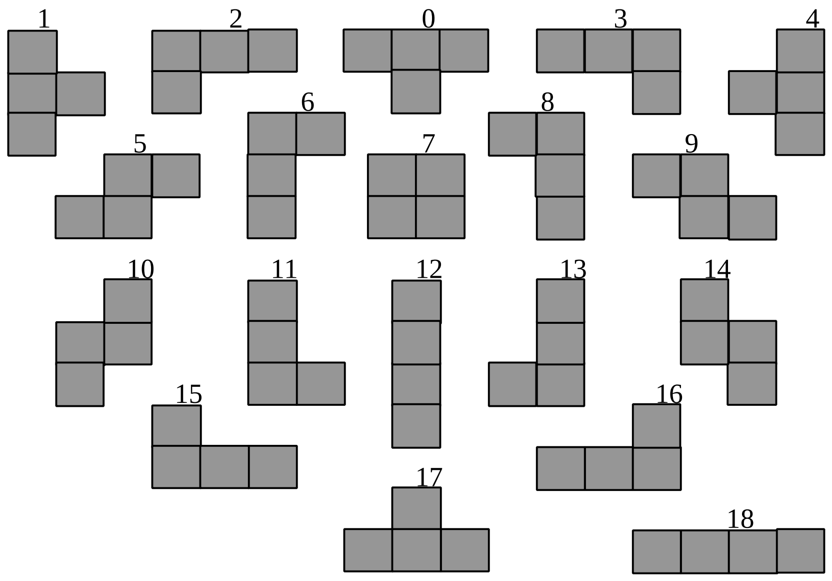
1998-99 ACM International Collegiate Programming Contest
Asia Regional Contest/Shanghai

PROBLEM G
TETRIS
Input file: tetris.in

We are in an age of electronic games. There are many electronic games the youth can play on PC, Saturn ™, PC Engine ™, Family Computer ™, Super Family Computer ™, Play Station ™, Nintendo 64 ™, Mega Drive ™, and even Arcades on street.

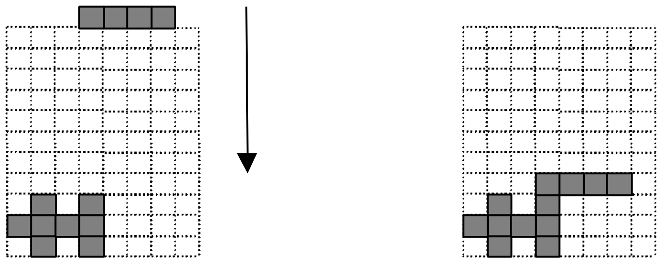
At the very beginning of the e-game age, TETRIS is a very famous game. Today you could also find it on modern game sets. Almost all players have played TETRIS because it's very simple for people in any ages.

A TETRIS game is based on an vertical playing region containing *m*n* basic squares. A player has to decide that how to place stone blocks each occupies 4 basic squares. There are seven basic types of 4-squares stone blocks, and every basic type of 4-squares stone block can rotate 90, 180, 270, and 360 degree, so there are total 19 configurations of stone blocks. The graph below shows all 19 configurations:

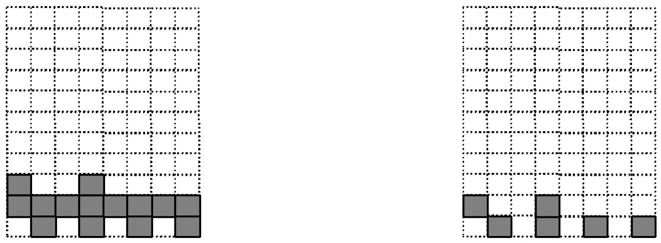


In this graph, we number configurations from zero to eighteen.

In the TETRIS game, stone blocks fall down from the top of playing region (outside of the region) one by one. Every stone block must be regarded as a rigid whole. A stone block will fall down vertically until it touchs the ground of the region or it piles up other stone blocks which fell down before. For example, a stone block in configuration 18 falls down in playing region (like left graph shows) will result in the status that the right graph shows.



Another important rule of this game is that if all squares in a horizontal line were occupied by stone blocks, the horizontal line would be ‘wiped’. This means that all stone blocks in this line will be deleted and all stone blocks above to this line will fall down a basic square height. Graphs below show the playing region before and after a wipe.



One stone block’s falling down and all wipes after that is called one turn. If any part of a stone block is out of the playing region after a turn, then ‘Game Over’. The player has no way but suspire for his carelessness.
The game has a scoring system to indicate the level of players. If a player was success in falling down a stone block without causing ‘Game Over’, he would get 50 score points. He also could get bonus points if there are any wipes happened in this turn. There are 200 bonus points for one wipe, 400 bonus points for two wipes, 800 bonus points for three wipes, and 1600 bonus points for four wipes.
Your task is to write a program to determine if a player is ‘Game Over’. If not, you must output the playing region in the end. In any case, you must compute the player’s score and his turn before ‘Game Over’ or end. Any turns after ‘Game Over’ or causing ‘Game Over’ would not get any score. There is no stone square in the playing region at beginning.

Input

The input file consists of several test cases. Each case will begin with two integers *m* ($4 < m < 26$) and *n* ($3 < n < 41$) which indicates height and width of the playing region. And *m* and *n* equals zero means the end of the input file. Following *m* and *n*, there are several turns of the case. Each turn include two integers *c* and *p*. The first integer *c* represents the number of stone blocks falling down in this turn and the second integer *p* represents the column in which the left most of the stone block. While *c* equals zero and *p* equals zero indicates the end of this case. You can assume that there will always be valid data in any test cases.

Output

For every test case, you should first output the test case no as '`Game No: #X`' in one line, then the score the player had got as '`Score:` *score*' in one line. If the player lost in this game, you should output 'Game Over' in the next line, otherwise output the playing region from top to bottom by lines with no spaces between them. A blank line should be printed between two test cases.

Sample Input

```
12 8
12 1 12 2
12 4 12 5 12 7 12 8
12 3 12 6
7 5
0 3
0 0
8 8
11 3 13 3
11 3
18 2 12 1
0 0
0 0
```

Output for the Sample Input

```
Game No: #1
Score: 2100
Turns: 10
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00111000
00011100
00001100

Game No: #2
Score: 100
Turns: 3
Game Over
```

PROBLEM H

Scoring of the Contest

Input file: scoring.in

In the ACM/ICPC regional contest, solutions to problems submitted for judging are called runs. Each run is judged as accepted or rejected, and the team is notified of the results. Teams that solve the median number of problems or higher will be ranked according to the most problems solved; teams solving less than the median number of problems will be acknowledged but not ranked. For the purposes of awards, or in determining qualifier(s) for the World Finals, teams who solve the same number of problems are ranked by least total time used and penalized. The total time used and penalized is the sum of the time consumed for each problem solved. The time consumed for a solved problem is the time elapsed from the beginning of the contest to the submittal of the accepted run plus 20 minutes of penalty for each rejected run. There is no time consumed for a problem that is not solved.
Your program will read a list of judgement for runs, and print out the scores for the 1st , 2nd and 3rd teams.

Input

Input file may contain several test data sets.
Every test data consists of two parts. The first part contains a positive integer, which tells number of teams. The following lines are judgement for runs. One line for one run that includes time of submission, team number, problem number and judgement. And one zero in a single line for end of the case.
The end line of input file contains only one zero for number of teams.

Output

The score for the top three teams which contains team number, number of problem solved, time used and rank.

Assumption
There may be several teams in the same place. For example, if several teams are 3rd, output them all; if two teams are in 2nd place, then there will be no 3rd team, etc.

Sample Input

```
3
12 1 2 yes
14 3 2 no
25 3 1 yes
29 1 1 no
38 3 2 yes
39 2 1 no
45 1 1 no
0
0
```

Output for the Sample Input

```
Case 1:
Team No.   Problem solved   Time used     Rank
3          2                83            1
1          1                12            2
2          0                0             3
```