1999-2000 ACM International Collegiate Programming Contest

Sponsored by IBM

Asia Regional Contest / Shanghai

Shanghai University, Shanghai

Practice Session

November 27, 1999

This problem set should contain three (3) problems on six (6) numbered pages. Please inform a runner immediately if something is missing from your problem set.

1999-2000 ACM International Collegiate Programming Contest
Asia Regional Contest/Shanghai
Practice Session

PROBLEM 1
Card Shuffling
Input file: shuffle.in

Suppose we have 2*n* cards numbered 1, 2, …, *n*, *n*+1, …, 2*n*, 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, …, 2*n*, *n*. That is to put the first *n* cards to positions 2, 4, …, 2*n*. And the rest *n* cards will be put to, in accordance with their original sequence, odd numbered positions 1, 3, ⋯, 2*n*-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 2*n* 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
```

1999-2000 ACM International Collegiate Programming Contest
Asia Regional Contest/Shanghai
Practice Session

PROBLEM 2

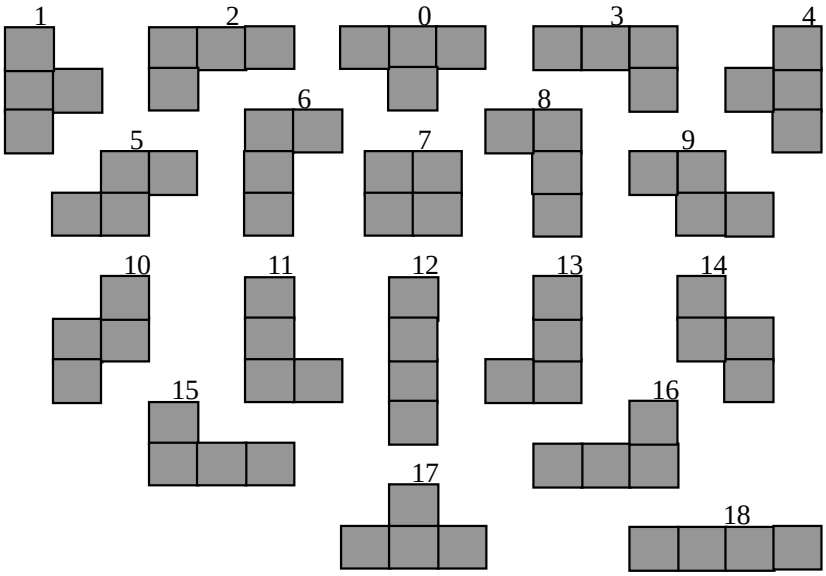
TETRIS

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.

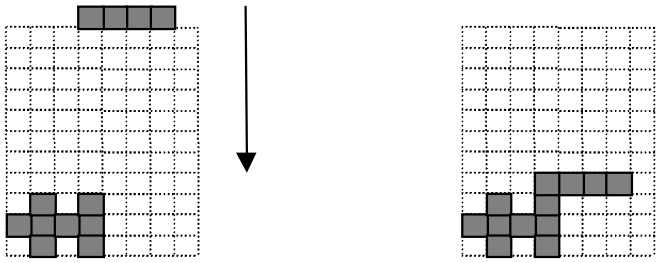
At the very beginning of the e-game age, TETRIS is a very famous game. Today you could also find it on modern game sets. Almost all players have played TETRIS because it's very simple for people in any ages.

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
```

1999-2000 ACM International Collegiate Programming Contest
Asia Regional Contest/Shanghai
Practice Session

PROBLEM 3
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
```