Irish Collegiate Programming Competition 2016

Problem Set

University College Cork ACM Student Chapter

April 2, 2016

Instructions

Rules

- All mobile phones, laptops, and other electronic devices must be powered off and stowed away for the duration of the contest.
- The only networked resource that teams are permitted to access is the submission system.
- If a team discovers an ambiguity or error in a problem statement, they should tell an organiser. If the organisers agree that an ambiguity or error exists, a clarification will be issued to all teams.
- No multi-threading is allowed, and no sub-processes.
- No file input/output is allowed. All input to your program will be provided via standard input (stdin) and all output should be printed to standard output (stdout). Examples of how to do this in each of the languages is provided in the resources section of the submission site.

Submission Instructions

- The submission system URL is: `http://4c245.ucc.ie:8080/`
- Your password will be provided by the organisers. Please notify an organiser if you are unable to log in.
- Submissions should consist of a single source file, **not a compiled executable**.
- To submit, click the "Submit a Solution" link, complete the submission form, upload your source file, and click "Save".
- Your submission should now be listed in your submission queue.
- Java solutions should be a single source file and should not include the package header. The testing script will also be renaming your file to `Pn.java` (where n is the problem number) and as such the main class for problem 1 should be defined as: `public class P1 {...}` and will be called with: `java P1 < input-file`

- C and C++ submissions will be compiled with the flags `-lm -O2`.
- C# submissions will be run using mono since the test environment is running on Linux.
- Haskell submissions will be compiled with the flag `-O2`.

Testing and Scoring

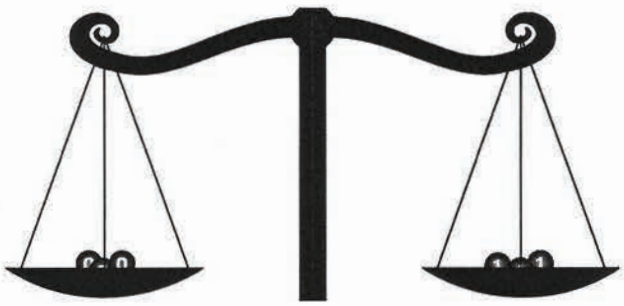
- Solutions should be submitted in the form of source code, which will be compiled on the test environment. A solution will be accepted if it compiles on the test environment and produces the correct output for the sample inputs given in the question.
- The output from your program should be terminated by a new line and should not contain any additional whitespace at the beginning or end of lines.
- A solution will be tested against 10 separate test cases which are designed to test the limits and corner cases of the inputs. One point is given for each correct output.
- Programs are limited to one second of CPU time and 256MB of RAM.
- If a solution is submitted while an earlier solution to the same problem is still in the queue, the earlier submission will not be tested. The earlier submission will be marked with the message: "Old, not going to be tested".
- In the event of a tie, the winning team will be the one whose last scoring submission was submitted earliest.

This page is intentionally left blank.

1 Bonnie's Balanced Binary Bucket Bonanza

Bonnie operates a dark, underground gaming ring where players play a peculiar game called *Binary Bingo*. The game starts with players picking a random integer from Bonnie's hat. Then, the following is repeated until the number reaches zero:

1. If the number is even, a ball is added to the left-hand side of a balance.
2. If the number is odd, a ball is added to the right-hand side.
3. The number is then divided by 2, ignoring any decimal remainder.



Bonnie recently heard about the dot-com boom and would like to take *Binary Bingo* game online to the masses. She has asked you to build a digital version of her game and would like to test your program against some historical records of games she has run.

Input The first line contains an integer T, $1 \leq T \leq 1000$, corresponding to the number of integers in this test case. Each of the following T lines contain a single integer X, $0 \leq X \leq 2^{31} - 1$.

Output The output should consist of T lines, where each line contains either a -1, 0, or 1. The output for each line should be:

-1 if there is more balls on the left-hand side,

0 if there is an equal number of balls on either side, and

1 if there are more balls on the right-hand side.

Sample Input 1	Output 1	Sample Input 2	Output 2
5		5	
2	0	34	-1
3	1	35	0
5	1	36	-1
8	-1	55	1
10	0	56	0

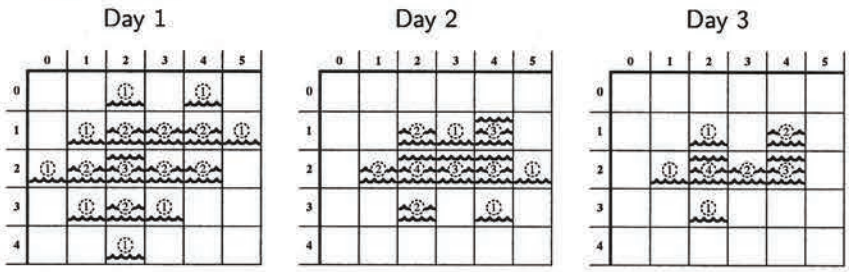
2 Predicting Precipitation, Simulating Storms, and Forecasting Flooding

The storm season has started, and the Association of Competent Meteorologists is eager to provide flood warnings to the anxious citizenry. With this in mind they have started crunching data. With the help of the newest meteorological techniques, and historical weather big data, they are able to predict the centre and intensity of storms.

The considerate meteorologists have divided the country into equally sized square cells, and have made predictions for each one. The occurrence of towns and storm centres are isolated to a cell, and may overlap within the cell.

The flood level, L, of a cell denotes the cumulative amount of rain that storms in the vicinity have contributed to the cell. The impact of a storm on a cell is dependant on two factors: the current strength of the storm, and the cell's distance from the centre of the storm. The storm is strongest at its centre, and the strength decreases by 1 for each cell (orthogonally) away from the centre. Additionally, at the start of each day, the flood level of each cell, and the strength of each storm is reduced by 1, until they reach 0. Both decreases happen before any new storms start.

Once the meteorologists have forecasted information about imminent storms, your task is to calculate the flood levels for each town, at the end of each day in the forecast period.



The flooding at the end of each day for the instance specified in sample input 2.

Input The first line contains three integers D, N, and S, corresponding to the number of days in the forecast, the number of towns on the map, and the number of storms on the map respectively, with $1 \leq D \leq 25$, $1 \leq N, S \leq 250$.

Each of the following N lines contain two integers x_i, and y_i, $0 \leq x_i, y_i \leq 1000$, representing the coordinates of town i.

The next S lines specify the storms present on the map. Each line contains four integers x_j, y_j, f_j, and b_j, respectively representing the (x_j, y_j) coordinates of the storm j, the strength of the storm, and the day the storm begins on. In each case, $1 \leq f_j \leq 500$, $1 \leq b_j \leq D$, and $0 \leq x_i, y_i \leq 1000$.

Note that no town will have the same coordinates as another town. Similarly, no two storms will have the same coordinates, even if they begin on different days.

Output The output should consist of D lines, with each line representing the forecast at the end of a single day in the prediction period. Each line should

contain N integers, where the i_{th} integer is L_i, the flood level of the i_th town, with $0 \leq L_i \leq 1000000$.

Sample Input 1

```
1 2 1
2 2
2 3
2 2 3 1
```

Sample Input 2

```
3 2 3
3 1
3 2
2 2 3 1
4 1 2 1
4 2 2 2
```

Sample Output 1

```
3 2
```

Sample Output 2

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

3 Zoe's Zany Zeros

The maths professor paused for a moment, stroking his chin. "Time to stretch those brain cells of yours!", he said, out of the blue. "As you know, the *factorial* of a non-negative integer n is denoted by $n!$ and is defined thus..." He walked to the blackboard, armed with a piece of chalk, and wrote the following definition in the most expert handwriting:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

"I'm going to write a non-negative integer on the board", he announced, spinning around to face the class, his face twisted in a malicious smirk. "I want you to count the number of trailing zeros in the factorial of this integer. In other words, you must count how many consecutive zeros appear at the end of the decimal representation of the integer's factorial." He paced up and down the chalkboard, relishing in the moment. "For example", he continued, "if I give you 6, the answer is 1, because $6!$ evaluates to 720, which has one trailing zero; if I give you 11, the answer is 2, because $11!$ evaluates to 39916800, which has two trailing zeros."

After writing a large integer on the chalkboard, the professor sat on the corner of his desk, arms folded defiantly, waiting for someone to deliver the answer. The students were silent, daunted by the tedious multiplications ahead of them; but Zoe, who was quietly sitting at the back, had an idea...

Input The first line contains an integer T, such that $2 \leq T \leq 10000$, the number of integers to process. Each of the following T lines consist of a single integer n, such that $0 \leq n \leq 2^{31} - 1$.

Output The output of your program should consist of T lines, where each line specifies the number of trailing zeros in the factorial of the integer on the corresponding input line.

Sample Input 1

```
2
6
11
```

Sample Output 1

```
1
2
```

Sample Input 2

```
3
2
5
15
```

Sample Output 2

```
0
1
3
```

4 Sorting Steven's Stack of Sooty Socks

Spring has arrived, and Steven has used up all his pairs of socks. Steven is now forced to perform his annual sock cleaning ritual. After several loads of washings, he is left with a pile of unmatched socks, each varying in colour and level of disrepair.

The $2 \times N$ socks on the pile are each of a shade of grey, ranging from 0 (completely black) to M (completely white), and have a number of holes, ranging from 0 to M. Now he must pair the freshly laundered socks in such a way that the most similar socks are reunited to form up N pairs.

To tackle this painstaking task, Steven has a system; he scans the stack of all unpaired socks, finds the two most similar socks, pairs them and puts them in a drawer. He repeats this until all socks have been stowed away in the sock drawer.

Let s and t be two socks, with s_c and t_c denoting their respective colour, and s_h, t_h denoting their number of holes. The difference between any two socks is computed using the following function:

$$\text{diff}(s,t) = \sqrt{(s_c - t_c)^2 + (s_h - t_h)^2}$$

Naturally, the smaller the diff(s,t) value, the more similar two socks are. Steven pairs socks in order of this difference measure and removes them from the pile. After extensive inspection of the stack, Steven can guarantee that his method never leads to a situation in which finding the next two socks to pair is ambiguous. Your task is to help Steven by implementing his matching methodology and tell him which socks should be paired.

Input The first line of input contains two integers N and M, corresponding to the number of pairs to be formed from the stack of socks, and the upper bound on colour/disrepair-level respectively, where $1 \leq N \leq 350$ and $1 \leq M \leq 50000$. Each of the following $2 \times N$ lines contain two integers, the sock's colour $0 \leq c_i \leq M$ and the number of holes $0 \leq h_i \leq M$ for sock i.

Output The output should consist of N lines, each containing four integers identifying the the pair of socks by their respective colour and number of holes. The pairs should be printed in the order dictated by Steven's methodology. Each pair (s,t) should be lexicographically ordered on colour first, then number of holes.

Sample Input 1	Output 1	Sample Input 2	Output 2
6 10	5 0 7 0	10 100	22 51 22 60
7 0	7 6 8 8	65 2	82 0 91 8
3 2	3 2 10 1	28 22	66 55 75 69
8 8		75 69	43 3 65 2
10 1		91 8	2 20 28 22
5 0		43 3	
7 6		22 51	
		2 20	
		66 55	
		22 60	
		82 0	

5 Plotting Potential Pacific Port Patrols Perfectly

The newly established Principality of Sealand is building a sea defensive system. In order to ensure the principality is properly defended, Sealand's navy should perform a set of patrols along routes linking several ports.

Leaders of the principality have identified some paramount patrol routes between pairs of ports which the navy are required to serve. Additionally, they specifies a list of the possible connections between patrols, a connection from the end of one patrol to the start of another. Note that all connections are directed, are guaranteed to not have any cycles in the route network, and will have at most one direct connection between pairs of ports. Ships may only travel along the required or optional routes, and in the direction specified.

A patrol (a, b) is completed if a ship sails from port a to port b. Every location should be visited by exactly one ship. Ships may start at the start location of any mandatory patrol, and may end their service at the end location of a patrol.

Sealand being a rather small nation, with limited funds, will allow the navy to use up to k ships to cover the required patrols. Your task is to find a set of patrol routes that should be undertaken.

The leaders have divided the problem into distinct regions for which a schedule is needed. Your task is to find a patrol schedule for each region.

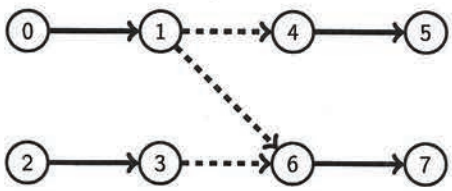


Figure 2: The possible defensive patrol routes for the second region of Sample Input 1. Solid lines represent the required patrol routes, dashed lines represent the possible connections to the start of the next patrol.

Figure 2 shows the Sample Input 1. One solution for this sample, with $k = 2$, consists of the first ship starting at port 0, it covers the mandatory patrol to port 1, continues to port 4, and finally covers the mandatory patrol from 4 to 5. The second ship starts at port 2, covers the mandatory patrol from 2 to 3, takes the allowed path from 3 to 6, and finally covers the mandatory patrol from 6 to 7. Note that the first ship could also have traversed the route from 1 to 6 but doing so would have meant that the mandatory patrol from 4 to 5 could not have been covered and the second ship would have to have stopped at 3.

Input The first line of input contains an integer R, $1 \leq R \leq 10$, representing the number of regions for which the patrols are to be plotted. The remainder of the input file contains R distinct sections, where each section is given in the following format. Regions are independent, with identifiers starting from 0 in each one. If there is a surplus of unused ships in a region, they cannot be used in another region.

The first line for each region contains four integers n, r, o, and k which respectively describe the number of ports, number of required routes, number of optional routes, and the number of the available ships. $2 \leq n \leq 500$, $r = n/2$, $0 \leq o \leq 1000$, $2 \leq k \leq 500$. The next r lines contain two space-separated integers

(a, b), with $0 \leq a, b \leq n-1$, representing a required route going from port a to port b. These routes must be covered by a naval ship. Similarly, the next o lines contain two space-separated integers, (b', a'), specifying the set of optional routes that may be used but are not required. It is guaranteed that the optional route (b', a') will be a valid link between the end of a required route and the start of another required route.

Output For each region, the first line should contain the number of ships used in your solution, k', where $k' \leq k$. Each of the next k' lines, should specify the ordered sequence of ports visited by that ship.

There may be multiple valid solutions for a single region, you may output any of them and its correctness will be verified. All values should be space-separated and the solution should be immediately followed by a new line.

Sample Input 1

```
2
4 2 0 2
0 1
2 3
8 4 3 2
0 1
2 3
4 5
6 7
1 4
1 6
3 6
```

Sample Output 1

```
2
0 1
2 3
2
0 1 4 5
2 3 6 7
```

Sample Input 2

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

Sample Output 2

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

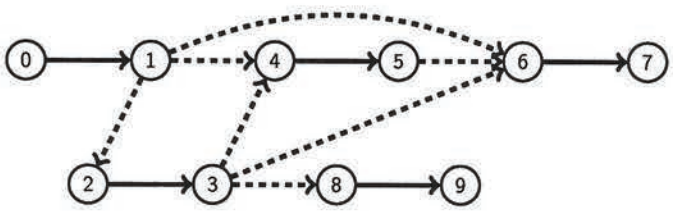


Figure 3: The possible defensive patrol routes for the first region of Sample Input 2. Solid lines represent the required patrol routes, dashed lines represent the possible connections to the start of the next patrol.