French-Australian Regional Informatics Olympiad

Friday 9th March, 2012

Duration: 4 hours

3 questions

All questions should be attempted

Contest Rules

The contest rules are set by the FARIO problems committee. This committee alone is responsible for the interpretation of the rules and of the contest questions, and is fully responsible for clarifying or altering the rules or contest questions in unforeseen circumstances.

Contest Environment

- The contest will last for four hours, and will be held at each student's school from 9am until 1pm local time on Friday 9th March, 2012.
- Each student may use one and only one computer, along with any compilers, interpreters, debuggers and associated development environments.
- Students may use calculators, printers and any written material (including printouts of code). Students may also use language documentation in electronic form, such as help files.
- Students may **not** use any source code in electronic form, and may not communicate with other contestants.
- Students may **not** use the Internet for any purpose other than submitting solutions to the contest site or mailing queries to the problems committee, as described below.

Code Restrictions

- Solutions will only be accepted in C or C++.
- Programs must be single-threaded and single-process. Any malicious code will lead to immediate disqualification.
- The source code for each solution must not exceed 40 000 bytes in size.

Submitting Solutions

- The contest submission site will be linked to from `http://www.fario.org/` . The log-in details are the same as those used for the AIOC training site and the December School of Excellence.
- **Students must submit their own solutions as they go.** The submission site will only accept submissions during the contest period (4 hours from first log-in).
- Once the student's 4 hours have expired, **further submissions will not be accepted**. Do not leave all your submissions until the final few minutes!
- When a solution is submitted, it will be queued for judging and evaluation. The results of any sample cases (Public testcases) will be available, while the full results of a submission can be viewed with the use of a token. Students will have a limited number of tokens per question to use, and this limit will be displayed on the submission page.

If you have any queries regarding either the conduct of the contest or the specific contest problems, please e-mail your query to *fario@orac.amt.edu.au*. Your query will be answered as soon as possible. For urgent issues (such as losing your Internet connection), please contact Mr Jarrah Lacko on 0408 817 082 or Mr Christopher Chen on 0438 498 318.

Good luck!

Problem 1
Master Chef

Input File: *standard input*
Output File: *standard output*

Time and Memory Limits: 0.5 seconds, 64 MB

Roderick was once the world's greatest light-switch switcher. With astonishing dexterity and lightning speed, Rod's unique capability of hitting many adjacent light-switches simultaneously gave him an edge unparalleled in the light-switching business. However in these tough economic times, Rod has had to turn to alternate pursuits. Yesterday he was called up and offered a place on the popular competitive cooking TV show *Master Chef*, but he needs your help to determine whether he has what it takes to truly master the kitchen.

Roderick will be given N ingredients lined up in front of him. His strategy for speedy cooking involves selecting, at random, a set of ingredients that form one contiguous subsequence of the line-up (that is, all ingredients between the a^{th} and b^{th} ingredient inclusive for some random selection of a and b such that $1 \leq a \leq b \leq N$). In one swift swoop using one of his exceptionally large hands, Roderick will collect all the selected ingredients and pour them into the pot for mixing. Each ingredient has a tastiness value T (which may be negative - some ingredients are truly foul), and the tastiness of the final meal will be the *average* tastiness of all selected ingredients. For example, in the following line-up of ingredients, if Roderick happens to select the ones with tastiness 3, -1 and 5, the tastiness of the meal would be $(3 + -1 + 5)/3 = 2\frac{1}{3}$.

6	3	-1	5

Since Roderick has no idea in advance which subsequence of ingredients he will happen to pick up on the day of the competition, he wants you to write a program that will calculate the expected tastiness of the meal he will create (ie. average tastiness of all possible meals he may create). In the example above with 4 ingredients, there are 10 possible selections of ingredients that Roderick could make meals from. The meal tastiness values would be: 6, 3, -1, 5, 4.5, 1, 2, $2.\overline{6}$ [1], $2.\overline{3}$ and 3.25. The average tastiness of all these 10 possible meals is 2.875. As Roderick isn't great with numbers, he wants you to give the final expected tastiness **rounded to the next integer towards zero** (that is, truncated to zero decimal places). In this example, the answer would hence be 2.

Input

- The first line of input will contain one integer N, representing the number of ingredients.
- The following line will contain N integers, representing the tastiness of each ingredient from left to right.

Output

Output should consist of a single integer: the (rounded) average tastiness of all possible meals, where a meal's tastiness is the average tastiness of all ingredients in a subsequence of ingredients. Rounding is towards zero.

[1] $2.\overline{6}$ means 2.6666666..., or 2.6 recurring

Sample Input 1

```
4
6 3 -1 5
```

Sample Input 2

```
8
-3 10 19 9 7 10 0 5
```

Sample Input 3

```
3
-10 -4 1
```

Sample Output 1

```
2
```

Sample Output 2

```
8
```

Sample Output 3

```
-4
```

Explanation

The first sample case corresponds to the example discussed in the problem. In the second sample case, the average tastiness of the 36 possible meals is approximately 8.12, which is truncated to 8. In the third sample case, the average is -4.30556, which is rounded towards zero to become -4.

Constraints & Subtasks

In all subtasks, the tastiness of each ingredient will be an integer between -1000 and 1000 inclusive.

- Subtask 1 (20 points): $1 \leq N \leq 200$
- Subtask 2 (20 points): $1 \leq N \leq 2,000$
- Subtask 3 (60 points): $1 \leq N \leq 200,000$

You will score the full allocated marks for a subtask if your program returns the correct answer for every test case within that subtask. If your program fails any test case in a subtask, your score will be 0 for that subtask.

Problem 2
Happy Feet

Input File: *standard input*
Output File: *standard output*

Time and Memory Limits: 0.5 seconds, 64 MB

Forget space flights, undersea submarine voyages and desert treks. Today's frontier of tourism is watching the famous dancing penguins of Antarctica. As the founder of a new company selling hiking tours in the southern-most continent, your killer whale of a business plan is to find the perfect hiking tour of the penguin colonies that will be most enjoyable for the wealthy tourists.

After lengthy discussions with Antarctican penguin experts, you have developed a map of all the human-walkable trails in the region. Each trail leads from one intersection to another intersection and can be walked in either direction. Note that a trail may lead from an intersection back to itself, and there may be multiple trails between two intersections. The experts have given you accurate figures of how many penguins will be seen dancing on each trail. You would like to find a path with the highest total number of penguins seen along the way. This path can start at any intersection and finish at any intersection (including the starting intersection). It may even involve visiting the same intersection multiple times. The only condition is that **each trail in the path must have strictly more penguins than any previous trail**. Tourists exhibit a strange psychological complex requiring their expectations to be continually exceeded. Failure to do so would mean a catastrophic post-trip influx of negative travel reviews.

Given the trail map, write a program to find the largest total number of penguins that can be seen on such a path.

Input

- The first line of input will contain two integers N and M, representing the number of intersections and number of trails respectively.
- The following M lines of input will each contain three integers a_i, b_i, and p_i representing a trail. The trail connects intersections a_i and b_i and has p_i penguins that can be seen dancing on it. Intersections are labelled $1..N$, that is, $1 \leq a_i, b_i \leq N$.

Output

Output should consist of a single integer: the maximum possible number of penguins that can be seen on a path satisfying the described conditions.

Sample Input

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

Sample Output

```
26
```

Explanation

The optimal path in the sample case above takes the following trails, beginning at intersection 3 and ending at intersection 5.

From	To	Penguins seen
3	1	1
1	4	3
4	3	4
3	4	5
4	5	6
5	5	7

This path has a total of $1 + 3 + 4 + 5 + 6 + 7 = 26$ penguins. No valid path exists with more penguins, hence 26 is the correct answer.

Subtasks & Constraints

In all subtasks, $1 \leq N \leq 1,000$ and each trail will have between 1 and $1,000,000$ penguins (inclusive).

- Subtask 1 (30 points): $1 \leq M \leq 1,000$.
- Subtask 2 (40 points): $1 \leq M \leq 1,000,000$ and it is guaranteed that no two trails will have the same number of penguins on them (that is, all trail penguin counts will be unique).
- Subtask 3 (30 points): $1 \leq M \leq 1,000,000$.

You will score the full allocated marks for a subtask if your program returns the correct answer for every test case within that subtask. If your program fails any test case in a subtask, your score will be 0 for that subtask.

Problem 3
Blackout

Input File: *standard input*
Output File: *standard output*

Time and Memory Limits: 1 second, 128 MB

This time they had planned for every possible failure and made every system redundant: two independent electric fences, two control centers, two power lines connected to two separate power plants, and two GPS tracking devices attached to each beast. Nothing could possibly go wrong with this amount of security and redundancy.

What they never predicted is now happening: a national strike in every non-renewable energy power plant in the country, following the announcement of a new target of producing 100% of the country's energy from renewable sources within 5 years. The ensuing blackout is there to last, and the backup generators can only power the electric fences for a few more hours, before every Tyrannosaurus Rex of the brand new park is free to explore the neighboring residential areas.

You have been asked to participate in implementing the emergency plan. Paleontologists have recently discovered that T-Rexes won't eat salty food, and conjectured that salt can be used as a T-Rex repellent. Further, the brand new park is conveniently arranged as a $S \times S$ grid, broken up into unit squares. The plan is to use helicopters to drop sea water on certain squares of land so that the T-Rexes will avoid them and stay in the park. Your job is to write a program that will determine where to pour sea water each hour after the blackout in order to minimise the number of escaped T-Rexes.

There are two types of squares in the park; empty squares that T-Rexes can enter, and squares with obstacles, which they can not. The GPS trackers will give you the last position of each T-Rex in this grid before the power goes down. While you won't be able to track their movement, you do know that every hour each T-Rex can move to any of the 4 squares adjacent to its previous position (provided the square is not an obstacle, nor has had sea water dropped onto it). Each hour, you will be able to drop sea water on W squares, permanently preventing any T-Rex from entering these squares.

Input

- The first line of input will contain three integers S, W and T. S represents the number of lines and rows of the grid, W represents the number of squares you can drop sea-water on every hour, and T represents the number of T-Rexes in the park.
- The following S lines of input will each contain S space-separated integers, 0 representing an empty square, and 1 representing an obsacle.
- The following T lines of input will each contain two integers, C_i and R_i ($0 \leq C_i, R_i < S$), representing the column and row of the last known position of a T-Rex before the blackout. Rows are numbered from top to bottom, columns are numbered from left to right (so position "0 0" would be the top-left square).

Output

- The first line of output should consist of a single integer D, the total number of times water will be dropped on a cell. As you can not drop water on to the same cell multiple times (this would be useless), $D \leq S^2$.
- Each of the D following lines should contain two integers, DC_i and DR_i ($0 \leq DC_i, DR_i < S$), representing the column and the row of a square to drop sea water on. The locations are given in the order the water should be dropped, which means the first W lines correspond to the squares targeted during the first hour of the blackout, the next W lines to the squares targeted during the second hour, and so on.

Sample Input

```
5 2 1
0 1 0 0 0
0 0 0 0 0
0 0 0 1 0
0 0 0 0 0
0 1 1 1 0
2 3
```

Sample Output

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

Explanation

The park is a 5×5 grid, with only one T-Rex. You can drop water on 2 squares every hour. The grid is represented below, where T is the initial location of the T-Rex.

```
0 1 0 0 0
0 0 0 0 0
0 0 0 1 0
0 0 T 0 0
0 1 1 1 0
```

One way to prevent the T-Rex from escaping the park is to drop water on two squares in the first hour, on the left $(1, 3)$ and above $(2, 2)$ the T-Rex's initial location. This means the T-Rex can either stay where it is, or move to the right. At the end of the first hour, the situation is the following, where W marks the squares where water has been dropped, and T marks the two squares where the T-Rex may now be located.

```
0 1 0 0 0
0 0 0 0 0
0 0 W 1 0
0 W T T 0
0 1 1 1 0
```

Dropping water on square $(4, 3)$ will prevent the T-Rex from ever escaping the 2-square area that it is now confined to.

Constraints & Subtasks

- Subtask 1 (20 points): $S = 5$, $W = 1$, $T = 1$.
- Subtask 2 (20 points): $S = 10$, $W = 2$, $T = 1$.
- Subtask 3 (20 points): $S = 15$, $W = 3$, $T = 2$.
- Subtask 4 (20 points): $S = 20$, $W = 4$, $T = 5$.
- Subtask 5 (20 points): $S = 30$, $W = 5$, $T = 10$.

For each test case of each subtask, you will score 100% if the output of your program enables the capture of the most T-Rexes among all judges' solutions. Otherwise, your score for that test will be :

$$\frac{100 \times [\text{Number of T-Rexes your output captured}]}{\text{Maximum number of T-Rexes captured by judges' solutions}}$$

If your program gives an invalid output, your score will be 0 for that test case.

Your total score for a subtask will be the **sum** of your score for each of its test cases.