7th Asia-Pacific Informatics Olympiad

Hosted by
National University of Singapore, Singapore

Saturday, 11 May, 2013

Task name	**ROBOTS**	**TOLL**	**TASKSAUTHOR**
Time Limit	1.5s	2.5s	Not Applicable
Heap Size	128MB	128MB	Not Applicable
Stack Size	32MB	32MB	Not Applicable
Points	100	100	100
Number of subtasks	4	5	8
Remark	Submit a program	Submit a program	Submit a data file for each subtask

Language	Compiler version	Compiler options
C	gcc version 4.4.7	-O2 -lm
C++	g++ version 4.4.7	-O2 -lm
Pascal	fpc version 2.4.0	-O2

HAPPY PROGRAMMING!

Task 1: ROBOTS

The engineers at VRI (Voltron Robotics Institute) have built a swarm of n robots. Any two compatible robots that stands on the same grid can merge to form another composite robot.

We label the robots with number 1 to n ($n \leq 9$). Two robots are compatible if they have labels that are consecutive. Originally, each of the n robot has one unique label. A composite robot that is formed after merging two or more robots is assigned two labels, consisting of the minimum and maximum label of the robots that merge into the composite robot.

For example, robot 2 can only merge with robot 3 or robot 1. If robot 2 merges with robot 3, a composite robot 2-3 is formed. If robot 2-3 merges with robot 4-6, a composite robot 2-6 is formed. The robot 1-n is formed when all robots have merged.

The engineers place n robots in a room consisting of $w \times h$ grids, surrounded by walls. Some grids are occluded and cannot be accessed by the robots. Each grid can hold one or more robots, and a robot always occupies exactly one grid. Initially, each robot is placed on a different grid.

The robots are rather primitives. They can move only in a straight line along either the x-axis or y-axis after being pushed by an engineer. After it is pushed in one of the four directions parallel to the x- and y-axis, the robot continues moving, in the direction it is pushed in, until it is blocked either by an occlusion or a wall. After the robot stops moving, it scans for other compatible robots occupying the same grid, and merge with any compatible robot it finds into a larger robot. The merging process continues until no further merging is possible.

To help the robots change direction, the engineers have placed rotating plates in some of the grids. The rotating plates can either rotate in a clockwise or anti-clockwise direction. A robot that moves into a grid with the rotating plate always changes its moving direction by 90 degree in the same direction as the rotating plate. If a robot is being pushed while resting on top of a rotating plate, it rotates by 90 degree before moving off in a straight line, in a direction perpendicular to the direction it is pushed in.

Only one robot can move at one time.

You task is to find the minimum number of pushes such that all n robots are merged together (if possible).

Input

Your program must read from the standard input. The first line of the input file contains three integers, n, w, and h, separated by a space.

The next h lines of the input file describe the room, each line contains w characters. Each of these $w \times h$ characters represent a grid in the room.

A numeric character ('1' to '9') indicates that there is a robot labeled with the corresponding number in the grid. A character 'x' indicates that there is an occlusion in the grid. A character 'A' or 'C' indicates that there is a rotating plate in the grid. 'A' indicates that the plate is rotating anti-clockwise. 'C' indicates that the plate is rotating clockwise. All other grids are represented with a character '.'

Output

Your program must write to the standard output, either a single number indicating the minimum number of pushes needed to merge all n robots, or -1 if merging is not possible.

Subtasks

Your program will be tested on four sets of input instances as follow:

1. (10 points) The instances satisfy $n = 2$, $w \leq 10$, $h \leq 10$, with no rotating plates.
2. (20 points) The instances satisfy $n = 2$, $w \leq 10$, $h \leq 10$.
3. (30 points) The instances satisfy $n \leq 9$, $w \leq 300$, $h \leq 300$.
4. (40 points) The instances satisfy $n \leq 9$, $w \leq 500$, $h \leq 500$.

Sample Input

```
4 10 5
1.........
AA...x4...
..A..x....
2....x....
..C.3.A...
```

Sample Output

```
5
```

Description of the sample input/output

The following 5 steps optimally merge the robot.

1. Push robot 3 rightward. The robot moves right, meets a rotating plate, turns anti-clockwise, and continues its movement up. The robot eventually stops in front of the wall.
2. Push robot 4 upward. The robot moves up, stops in front of the wall, and merges with robot 3 to form robot 3-4.
3. Push robot 2 upward. The robot moves up, meets a rotating plate, turns anti-clockwise, hits a wall and stops.
4. Push robots 2 rightward. The robot rotates anti-clockwise, moves up, stops at the corner, and merges with robot 1 to form robot 1-2.
5. Push robot 3-4 leftward. The robot moves left, stops at the corner, and merges with robot 1-2.

Task 2: `TOLL`

Happyland can be described by a set of N towns (numbered 1 to N) initially connected by M bidirectional roads (numbered 1 to M). Town 1 is the central town. It is guaranteed that one can travel from town 1 to any other town through these roads. The roads are toll roads. A user of the road i has to pay a toll fee of c_i cents to the owner of the road. It is known that all of these c_i's are distinct. Recently, K additional new roads are completed and they are owned by a billionaire Mr Greedy. Mr Greedy can decide the toll fees (not necessarily distinct) of the new roads, and he has to announce the toll fees tomorrow.

Two weeks later, there will be a massive carnival in Happyland! Large number of participants will travel to the central town and parade along the roads. A total of p_j participants will leave from town j and travel toward the central town. They will only travel on a set of selected roads, and the selected roads will be announced a day before the event. By an old tradition, the roads are to be selected by the richest person in Happyland, who is Mr Greedy. Constrained by the same tradition, Mr Greedy must select a set of roads that *minimizes the sum of toll fees in the selected set* and yet at the same time allow anyone to travel from town j to town 1 (hence, the selected roads form a "minimum spanning tree" where the toll fees are the weights of the corresponding edges). If there are multiple such sets of roads, Mr Greedy can select any set as long as the sum is minimum.

Mr Greedy is well-aware that the revenue he received from the K new roads does not solely depends on the toll fees. The revenue from a road is actually the total fee collected from people who travel along the road. More precisely, if p people travel along road i, the revenue from the road i is the product $c_i p$. Note that Mr Greedy can only collect fees from the new roads since he does not own any of the old roads.

Mr Greedy has a sneaky plan. He plans to maximize his revenue during the carnival by manipulating the toll fees and the roads selection. He wants to assign the toll fees to the new roads (which are to be announced tomorrow), and select the roads for the carnival (which are to be announced a day before the carnival), in such a way that maximizes his revenue from the K new roads. Note that Mr Greedy still has to follow the tradition of selecting a set of roads that minimizes the sum of toll fees.

You are a reporter and want to expose his plan. To do so, you have to first write a program to determine how much revenue Mr Greedy can make with his sneaky plan.

Input

Your program must read from the standard input. The first line contains three space-separated integers N, M and K. The next M lines describe the initial M roads. The ith of these lines contains space-separated integers a_i, b_i and c_i, indicating that there is a bidirectional road between towns a_i and b_i with toll fee c_i. The next K lines describe the newly built K additional roads. The ith of these lines contains space-separated integers x_i and y_i, indicating that there is a new road connecting towns x_i and y_i. The last line contains N space-separated integers, the j-th of which is p_j, the number of people from town j traveling to town 1.

The input also satisfies the following constrains.

- $1 \leq N \leq 100000$.
- $1 \leq K \leq 20$.
- $1 \leq M \leq 300000$.
- $1 \leq c_i, p_j \leq 10^6$ for each i and j.
- $c_i \neq c_{i'}$, if $i \neq i'$.
- Between any two towns, there is at most one road (including newly built ones).

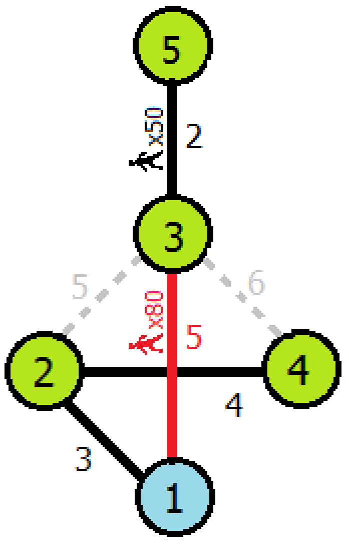
Output

Your program must write to the standard output a single integer, which is the maximum total revenue obtainable.

Sample Input and Output

Input	Output

`5 5 1`
`3 5 2`
`1 2 3`
`2 3 5`
`2 4 4`
`4 3 6`
`1 3`
`10 20 30 40 50`
	`400`



In this sample, Mr Greedy should set the toll fee of the new road (1,3) to be 5 cents. With this toll fee, he can select the roads (3,5), (1,2), (2,4) and (1,3) to minimize sum of toll fees, which is 14 cents. 30 people from town 3 and 50 people from town 5 will pass through the new road to town 1 and hence he can collect an optimal revenue of $(30 + 50) \times 5 = 400$ cents.

If, on the other hand, the toll fee of the new road (1,3) is set to be 10 cents. Now, constrained by the tradition, Mr Greedy must select (3,5), (1,2), (2,4) and (2,3) as this is the only set that minimizes the sum of toll fees. Hence, no revenue will be collected from the new road (1,3) during the carnival.

Subtasks

Your program will be tested on 5 sets of instances as follow:

1. (16 points) $N \le 10, M \le 20$ and $K = 1$.
2. (18 points) $N \le 30, M \le 50$ and $K \le 10$.
3. (22 points) $N \le 1,000, M \le 5,000$ and $K \le 10$.
4. (22 points) $N \le 100,000, M \le 300,000$ and $K \le 15$.
5. (22 points) $N \le 100,000, M \le 300,000$ and $K \le 20$.

Task 3: `TASKSAUTHOR`

There are many programming contests in the world today. Setting a good programming contest task is not easy. One challenge is the setting up of *test data*. A good test data should be able to differentiate a code that meets the goals, from another seemingly correct code that fails in some special cases.

In this task, your role in a contest is reversed! As an experienced programmer, you are helping the Happy Programmer Contest's committee in setting up their test data. The committee has selected two graph problems with a total of 8 different subtasks, and has written a few codes that seemingly solve the graph problems. In designing a subtask, the committee has the intention that some of the codes would get all the points, whereas some would gain zero or some points. You are given all those codes in C, C++ and Pascal versions[1]. For each subtask, your job is to produce a test data X that *differentiates* two given codes, code **A** and code **B**. More specifically, the following two conditions must be met:

1. On input X, code **A** must not lead to Time Limit Exceeded (TLE).
2. On input X, code **B** must lead to TLE.

In addition, the committee prefers smaller test data, with a target of at most T integers in the test data.

The two problems selected by the committee are the Single-Source Shortest Paths (SSSP) problem, and a graph problem we called the Mystery problem. The pseudo-codes of the codes written by the committee are listed in the appendix and the C++ and Pascal implementations can be found in the attached zip file that accompany task 3 in the grading server.

Subtasks

Please refer to Table 1. Each row describes a subtask. Note that subtask 1 to 6 are on the SSSP whereas subtask 7 & 8 are on the Mystery problem. The number of points allocated for the subtasks are listed in column S.

Subtask	Points S	Target T	Problem	Code **A**	Code **B**
1	3	107	SSSP	ModifiedDijkstra	FloydWarshall
2	7	2222	SSSP	FloydWarshall	OptimizedBellmanFord
3	8	105	SSSP	OptimizedBellmanFord	FloydWarshall
4	17	157	SSSP	FloydWarshall	ModifiedDijkstra
5	10	1016	SSSP	ModifiedDijkstra	OptimizedBellmanFord
6	19	143	SSSP	OptimizedBellmanFord	ModifiedDijkstra
7	11	3004	Mystery	Gamble1	RecursiveBacktracking
8	25	3004	Mystery	RecursiveBacktracking	Gamble2

Table 1: The 8 Subtasks.

[1]All codes implement algorithms that are permitted inside the IOI syllabus.

To get any point for a subtask, your test data X must able to differentiate the corresponding code **A** and code **B**. In addition, the number of points you received depends on the number of signed integers in X. Suppose X contains F integers, S points are allocated to the subtask, and T is the targeted size, then the number of points awarded is calculated as follow:

$$\lfloor 0.5 + S \times min\{T/F, 1\} \rfloor$$

where $\lfloor \ \rfloor$ denotes the rounding down operation. Hence, if your test data X contains not more than T integers, the full S points are awarded.

Grading

You must name each of your eight test data as `tasksauthor.outX.1` where `X` is the subtask number. Before submission, you are required to compress your test data files using `gzip`. In Unix systems, the following command produces the compressed file:

```
tar -cvzf tasksauthor.tgz tasksauthor.out*.1
```

In Windows systems, use software like 7-Zip or Winzip to produce this tar-gzipped archive. Submit only the file `tasksauthor.tgz` to the grading server.

The grading server will unpack the compressed file. For each subtask, say subtask `X`, the grader carries out the following steps to determine the number of points to be awarded for subtask `X`:

C1. If test data `tasksauthor.outX.1` does not exist, halts and no point will be awarded.

C2. Checks the format of `tasksauthor.outX.1`.
If the input format is invalid, halts and no point will be awarded.

C3. Runs code **A** with `tasksauthor.outX.1` as the input.
If TLE is triggered, halts and no point will be awarded.

C4. Runs code **B** with `tasksauthor.outX.1` as the input.
If TLE is triggered, halts and awards a number of points calculated using the formula:

$$\lfloor 0.5 + S \times min\{T/F, 1\} \rfloor$$

All the provided codes maintain a counter (the variable `counter`) that keeps track of the number of performed operations. During the execution of a code, when the value of the counter exceeds 1,000,000, then we consider the code has triggered TLE.

Problem Statement 1: Single-Source Shortest Paths (SSSP)

Given a directed weighted graph G and two vertices s and t in G, let $p(s,t)$ to be the shortest path weight from the "source" s to the "destination" t. If t is not reachable from s, then $p(s,t)$ is defined to be 1,000,000,000. In this problem, the input is the graph G and a sequence of Q queries $(s_1,t_1),(s_2,t_2),\ldots,(s_Q,t_Q)$. The output is the corresponding query results $p(s_1,t_1),p(s_2,t_2),\ldots,p(s_Q,t_Q)$.

Input/Output file Format

The input file consists of two blocks. The first block describes the adjacency list of a directed weighted graph G. The second block describes shortest path queries on G.

The first block starts with an integer V in one line, which is the number of vertices in G. The vertices are labelled as $0,1,\ldots,V-1$. Then, V lines follow where each line corresponds to a vertex, starting from vertex 0. Each line starts with n_i that describes how many outgoing-edges the vertex i has. Next, n_i pairs of integers (j,w) follow where each pair corresponds to an outgoing-edge. The first integer j in a pair is the label of the vertex the edge points to, and the second integer w is the edge's weight.

The second block starts with an integer Q in one line. Next, Q lines follows. The k-th line contains two integers s_k and t_k, corresponding to the source and destination vertex respectively.

Any two consecutive integers in one line must be separated by at least one space. Additionally, the input satisfies the following:

1. $0 < V \leq 300$,
2. n_i is a non-negative integer $\forall i \in [0..V-1]$,
3. $0 \leq j < V$,
4. $|w| < 10^6$ where $|w|$ denotes the absolute value of w,
5. $0 \leq \sum_{i=0}^{V-1} n_i \leq 5000$,
6. $0 < Q \leq 10$,
7. $0 \leq s_k < V$, $0 \leq t_k < V$, $\forall k \in [1..Q]$, and
8. the graph G must **not** have any negative weight cycle reachable from any vertex s_k that appeared in the second block.

Recall that the grading server will check for the above constraints in step C2.

The output file format is less relevant in this task. Anyway, the output consists of Q lines, and the k-th line contains the integer $p(s_k,t_k)$. For convenience, the provided codes will print out the value of the variable `counter` at the end of the output.

Sample Input File[2]

```
3
2 1 4 2 1
0
1 1 2
2
0 1
1 0
```

Sample Output File[3]

```
3
1000000000
The value of counter is: 5
```

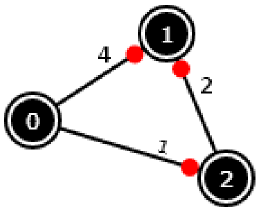


Figure 1: Directed Weighted Graph from the Sample Input File

[2]There are fifteen integers in this input file, therefore $F = 15$.

[3]The value of counter is 5 when the sample input file given above is run on ModifiedDijkstra.cpp/pas.

Problem Statement 2: Mystery

Given an undirected input graph G with V vertices and E edges, label each vertex in G with an integer $\in [0..(X-1)]$ so that no two endpoints of any edge in G has the same label. The value of X must be the lowest possible for graph G.

Input/Output file format

The input file starts with two integers V and E in one line. Then, E lines follows. Each line contains two integers a and b that denotes an *undirected* edge (a, b) exists in G. In addition, the input satisfies the following constraints (to be checked in step C2):

1. $70 < V < 1000$,
2. $1500 < E < 10^6$, and
3. for any edge (a, b), we have $a \neq b$, $0 \leq a < V$, $0 \leq b < V$, and it appears only once in G.

The output file starts with an integer X in one line, the smallest integer so that vertex labelling is feasible. The next line contains V integers that describe the integer label of vertex 0, vertex 1, ..., vertex $V-1$, and the last line is the value of `counter`.

Sample Input File[4]

```
4 5
0 1
0 2
0 3
1 2
2 3
```

Sample Output File[5]

```
3
0 1 2 1
The value of counter is: 18
```

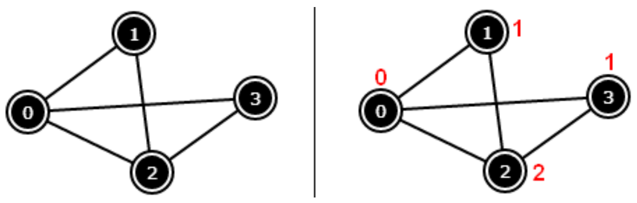


Figure 2: Left: Undirected Graph from the Sample Input File; Right: Its Labels with $X = 3$

[4]There are twelve integers in this input file, therefore $F = 12$. However, this small sample input file is only used for illustration. It is not valid as it's V and E values are too small.

[5]The value of counter is 18 when the sample input file given above is run on RecursiveBacktracking.cpp/pas.

Appendix: Pseudo-codes

Here are the algorithms of the provided codes. The variable `counter` "approximates" the runtime by keeping track of some operations. Our grading server uses the C++ version of these implementation codes.

FloydWarshall.cpp/pas

```
// pre-condition: the graph is stored in an adjacency matrix M
counter = 0
for k = 0 to V-1
   for i = 0 to V-1
      for j = 0 to V-1
         increase counter by 1;
         M[i][j] = min(M[i][j], M[i][k] + M[k][j]);
for each query p(s,t)
   output M[s][t];
```

OptimizedBellmanFord.cpp/pas

```
// pre-condition: the graph is stored in an adjacency list L
counter = 0
for each query p(s,t);
   dist[s] = 0; // s is the source vertex
   loop V-1 times
      change = false;
      for each edge (u,v) in L
         increase counter by 1;
         if dist[u] + weight(u,v) < dist[v]
            dist[v] = dist[u] + weight(u,v);
            change = true;
      if change is false // this is the 'optimized' Bellman Ford
         break from the outermost loop;
   output dist[t];
```

ModifiedDijkstra.cpp/pas

```
// pre-condition: the graph is stored in an adjacency list L
counter = 0;
for each query p(s,t)
   dist[s] = 0;
   pq.push(pair(0, s)); // pq is a priority queue
   while pq is not empty
      increase counter by 1;
      (d, u) = the top element of pq;
      remove the top element from pq;
      if (d == dist[u])
         for each edge (u,v) in L
            if (dist[u] + weight(u,v) ) < dist[v]
               dist[v] = dist[u] + weight(u,v);
               insert pair (dist[v], v) into the pq;
   output dist[t];
```

Gamble1.cpp/pas

```
Sets X = V and labels vertex i in [0..V-1] with i;
Sets counter = 0; // will never get TLE
```

Gamble2.cpp/pas

```
Sets X = V and labels vertex i in [0..V-1] with i;
Sets counter = 1000001; // force this to get TLE
```

RecursiveBacktracking.cpp/pas

```
This algorithm tries X from 2 to V one by one
and stops at the first valid X.

For each X, the backtracking routine label vertex 0 with 0,
then for each vertex u that has been assigned a label,
the backtracking routine tries to assign
the smallest possible label up to label X-1 to its neighbor v,
and backtracks if necessary.

// Please check RecursiveBacktracking.cpp/pas to see
// the exact lines where the iteration counter is increased by 1
```