# Spring 2012 Programming Competition

Run by UPE at
Rensselaer Polytechnic Institute

March 31, 2012
12:15 - 2:15

Sponsored By




# 0. Introduction

Welcome to this semester's programming competition held by UPE. This semester we are lucky enough to be sponsored by Palantir, Microsoft, TripAdvisor, and Vistaprint. To find more details and to submit your solutions please check `progcomp.upe.cs.rpi.edu`.

Please remember our simple rules:

1. All submitted code must be your own.
2. You are not allowed communicate with/ask questions of any other person.
3. You are allowed to use the internet.

Any attempt to tamper with competition server is considered dishonesty with respect to the contest. Please remember you are always subject to RPI's COMEC policy.

## 0.1 Questions and Submissions

In this document you will find 4 problems in no particular order. The input and output should be handled as you see fit, but we recommend using file input and output as you will be downloading an input file from the server and uploading an output file to the server. You will have two minutes to run your code and submit the output file and program file.

## 0.2 Input Format

This semester the first line of the input will be a single number `n` which is the number of test cases you will find in that text file. There will then be `n` inputs as defined in the "single input" subsection of each problem.

This means that you should implement the following pseudocode where `RUN SINGLE SOLUTION` reads in the necessary inputs from `F` and then returns the expected output.

```
F = file(input.txt)
Fout = file(output.txt)
n= F.readint()
for i = 1 to n
    print Fout, RUN_SINGLE_SOLUTION(F)
```

Please understand that we will be diffing your output file with one generated by our solution, as such please do not have any extra newlines (you can edit your output by hand if necessary)

# 1. Laser

You are aiming a 1.21 jiggawatt laser at a perfectly spherical balloon. The aiming mechanism is pretty simplistic: you just set the angle from horizontal. While this is fairly easy, it's pretty tough to know before firing the laser if you are going to hit the target, and since Doc needs his generator back soon, you don't have much time for trial and error.

The coordinate system used in this problem is `(north,up)`.

The basic setup is a laser beam will be generated from the point `(0,l)` it's direction will be determined by `r1` the angle from horizontal. The target is centered `x` meters due north of the laser and `h` meters above the ground plane `(x,h)` and is a perfect sphere of radius `b`. See figure 1.1 for a pictorial explanation.

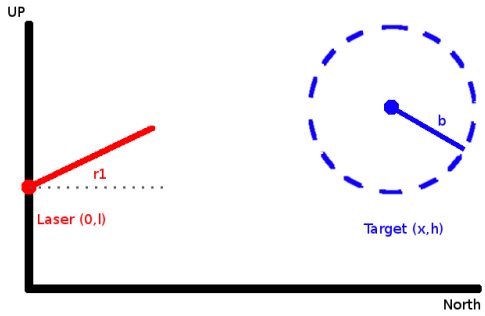


Figure 1.1: Laser problem schematic

## 1.1 Single Input

`l r1`
`x h b`

## 1.2 Limits

`l,x,h,b` are integers
`r1` is floating point

`number of test cases = 10000`
$0 \leq$ `l,h,x,b` $\leq 100$
`x` $>$ `b`
$0 \leq$ `r1` $< \pi/2$

## 1.3 Single Output

{`hit,miss`}

## 1.4 Sample Input

Notice that grazing the target as in input 2 here counts as a hit.

```
4
0 0
2 0 1
1 0
2 0 1
0 .785
2 0 1
2 0
18 0 1
```

## 1.5 Sample Output

```
hit
hit
miss
miss
```

# 2. 1-d battleships

You are playing a simplified game of battleships in 1 dimension, and want to know the probability that a certain shot will hit a single randomly placed ship on your opponents board. Given a board of size `s` where `h` is a hit, `m` is a miss, and `*` is a position which you have no direct information about. Given the size of the hidden ship `n`, return the probability that a shot at `x` (0 indexed) will hit the ship.

You can always assume that the board information is consistent. For instance, for any size ship you wouldn't see `hmh`, and for a ship greater than 2 you wouldn't see `mh*m`.

## 2.1 Single Input

```
s n x
s characters in {h,m,*}
```

## 2.2 Limits

```
number of test cases = 10000
s ≤ 10000
1 ≤ n ≤ 100
x ≤ s
```

## 2.3 Single Output

```
p
```

`p` is the probability rounded to the nearest percent that a shot at position `x` is a hit.

## 2.4 Sample Input

```
4
5 2 2
*h***
9 4 2
****h****
9 4 2
****h**m*
5 2 2
**h**
```

## 2.5 Sample Output

```
50
50
67
100
```

# 3. Campus Walk

Greg is a lazy RPI senior. In his opinion, walking up stairs and hills is for underclassmen (unless of course there isn't a flatter route or the flatter route is too inconvenient). Greg really wants to know when there is a better route for him available.

Given a list of distances and terrain types between two places (non-directional), find the distance of the shortest paths (or determine that there isn't a path return -1) subject to three conditions.

- Using all three types of terrain (stairs, flat, and hilly)
- Using only flat and hilly terrain
- Using only flat terrain

## 3.1 Single Input

```
FROM TO
n
(next n rows)
Place1 Place2 d {flat, hill, stairs}
```

## 3.2 Limits

```
number of test cases = 100
1 ≤ n ≤ 10000
1 ≤ d ≤ 10000
```

## 3.3 Single Output

```
d1 d2 d3
```

- `d1` is distance using stairs, hills, and flat
- `d2` is distance using hills and flat
- `d3` is distance using just flat

Return -1 if the travel is impossible

## 3.4 Sample Input

```
3
Commons JOHNSON
5
Commons Armory 25 hill
Armory Bridge 10 flat
Bridge Sage 15 stairs
```

```
Bridge DCC 25 flat
DCC JOHNSON 40 flat
JOHNSON VCC
9
JOHNSON JOHNSONBRIDGE 10 flat
JOHNSONBRIDGE GREEN 30 flat
GREEN LIBRARY 50 hill
GREEN VCC 49 hill
LIBRARY VCC 5 flat
JOHNSON JOHNSONFIRSTFLOOR 1 flat
JOHNSONFIRSTFLOOR JROWL 50 flat
JROWL MRC 50 flat
MRC LIBRARY 40 flat
BLITMAN UNION
13
BLITMAN SHUTTLE 5 flat
SHUTTLE UNION 10000 flat
BLITMAN BOTAPPROACH 45 flat
BOTAPPROACH TOPAPPROACH 100 stairs
TOPAPPROACH CARNEGIE 200 hill
TOPAPPROACH CARNEGIE 100 stairs
CARNEGIE FIELD 300 hill
FIELD QUAD 100 stairs
QUAD FIFTEENTH 100 stairs
FIFTEENTH UNION 25 stairs
CARNEGIE AMOSEATON 50 stairs
AMOSEATON BRIDGE 500 flat
BRIDGE UNION 100 hill
```

## 3.5 Sample output

```
100 100 -1
89 89 146
770 10005 10005
```

# 4. Railroads

It has just been made clear by some unknown terrorist organization that they are planning on bombing `k` connections between cities. You are a Russian rail engineer, and the prime minister is on the phone and wants to know which pairs of cities could be disconnected by this evil plot.

Given a list of city to city connections, a value `k` and a list of city tuples. Return `z`, the number of city tuples which could be disconnected by destroying `k` rail connections.

## 4.1 Single Input

```
n,m,k are integers
cities are strings
n m k
(next n lines city to city connections)
city1 city2
(next m lines city tuples)
city1 city2
```

## 4.2 Limits

```
number of test cases = 100
n,m,k are integers
```
$0 <$ `n,m` $< 10$
$0 <$ `k` $< 5$

## 4.3 Single Output

```
z
```

## 4.4 Sample Input

```
4
6 2 1
Stalingrad Leningrad
Leningrad Petropavlovsky
Petropavlovsky Svetlograd
Pillau Protva
Protva Stalluponen
Stalluponen Yuryuzansky
Svetlograd Stalingrad
Yuryuzansky Stalingrad
6 2 2
A B
B C
```

```
C D
D A
A E
E F
F A
A B
4 2 1
A B
B C
C D
D A
A B
B C
9 1 3
A B
A C
A D
B C
C D
C F
D E
D F
E F
A F
```

## 4.5 Sample Output

```
1
1
0
0
```

### 4.5.1 Explanation

For the first input we can go from Svetlograd - Petropavlovsky - Leningrad - Stalingrad But we cannot go from Yuryuzansy to Stalingrad.

The second input I didn't feel like writing Russian names anymore. For F-A we can only go F-E-A, for A-B we can go A-B or A-D-C-B 2 distinct paths.

For the first two we get an output of 1.

The third example is a circle obviously there are two connections between any two cities.

The fourth example is more difficult, I recommend drawing it out. The paths are A-B-C-F A-C-D-F and A-D-E-F