Catlin Gabel 2018 Fall Programming Contest

Advanced Division Problem Set

Rules:

1. Each question is worth a different number of points; the maximum point value of each question is in parenthesis next to its name. Teams receive points based off of the number of test cases solved correctly in their best submission. Additional attempts on a question will *not* affect points but will affect time score, which is used for tiebreakers. You do not need to solve the problems in order.

2. The internet may not be accessed during the contest coding phase. Only one computer per team can be out during this time, which means that phones need to be put away.

3. The teams with the highest points at the end of the contest will receive awards. Ties will be broken in favor of the team that has the lower time score.

Problems:

1. Fancy Frame (130)
2. Interception (140)
3. Apples (150)
4. Homework (160)
5. Desktop (170)
6. Hedge Maze (180)
7. Logic Puzzle (190)
8. Sandstorm (200)
9. Ristet (210)
10. Super Bowling (220)
11. Parkour run (230)
12. Websites (240)
13. Virus (250)
14. Mushrooms (260)
15. Basketbob (270)

1. Fancy Frame (130)
Input File: None.

Your friend is shopping for a new frame to display a picture of robots. However, as there are many frames to choose from, your friend needs some help picking a frame. Please output the pattern shown below.

Input:

None.

Output:

Please output the pattern exactly as shown below.

Sample Output:

```
\^/   \^/
<#>-o-<#>
/v\ | /v\
 | \|/ |
 o--0--o
 | /|\ |
\^/ | \^/
<#>-o-<#>
/v\   /v\
```

2. Interception (140)
Input File: interception.txt

A hangry raptor hurls a pair of scissors at your airplane! Dismayed, you quickly shove the raptor aside and hurl a rock at the scissors hoping to intercept it before it slices your airplane.

Given the velocity of the scissors and the distance between the projectiles at initial position and the airplane, determine the minimum velocity needed to intercept the pair of scissors with the rock. You launch the rock 1 second after the scissors are thrown at the same place it is thrown, and all projectiles travel along the same line. The scissors and rock are special objects impervious to the effects of gravity and air resistance.

Input:

An integer, n, the total number of cases.
For the next n lines:
Two space separated integers, v and d, specifying velocity of scissors ($0 < v < 100$) and distance from airplane ($100 \leq d < 1000$), respectively.

Output:

For each case, the smallest integer rock launch velocity such that the rock will intercept the scissors before it reaches the airplane. There should be n values outputted, one value per line. Note that the bounds on the parameters ensure that the rock can always intercept the scissors.

Sample Input:

```
2
50 150
100 500
```

Sample Output:

```
76
126
```

3. Apples (150)
Input File: apples.txt

In need of community service hours, you volunteer at your school's cafeteria. It recently received a large shipment of apples of various brands. Your task is to sort the apples by brand into baskets. Each basket can only hold 10 apples, so you must allow several baskets per brand. You are wondering how many baskets you need to hold all the apples.

Input:

An integer, *n*, the total number of cases.
For the next *n* sets of lines:
An integer *b*, the number of brands in a shipment ($1 \leq b < 10$).
For the next *b* lines:
A space separated string and integer, *name* and *a*, the name of the brand (a 1-10 character alphanumeric string) and the number of apples received from that brand ($0 \leq a < 100$).

Output:

For each case, the total number of baskets needed to hold all the apples such that each basket contains apples of only one brand, and no more than 10 apples. There should be *n* values outputted, one value per line.

Sample Input:

```
2
1
Delicious 42
3
Good 19
Bad 0
Ugly 1
```

Sample Output:

```
5
3
```

4. Homework (160)
Input File: homework.txt

An endless stream of homework stands between you and a relaxing night of sleep. You have to complete several assignments of various durations. Between assignments, you take short breaks, each of which is ten percent of the total amount of time you have been working since you began your homework (time spent on breaks does not count toward time working). You are done with your homework once you have completed the last assignment. You notice that the time it takes for you to finish your homework depends on the order in which you complete your assignments. Now, you would like to figure out the difference between the maximum and minimum amounts of time in which you can finish all your work given a certain set of assignments.

Input:

An integer, *n*, the number of sets of assignments.
For the next *n* sets of input:
An integer *a*, the total number of assignments ($1 < a \leq 10$).
A line with *a* space-separated non-negative integers, stating the amount of time an assignment takes. Each assignment takes no more than 1000 minutes to complete.

Output:

For each set of assignments, the difference, in minutes, between the maximum and minimum amounts of time to finish all your work, rounded to the nearest tenth of a minute. There should be *n* values outputted, one value per line.

Sample Input:

```
2
2
1 100
5
10 50 30 20 40
```

Sample Output:

```
9.9
20.0
```

5. Desktop (170)
Input File: desktop.txt

You've made a new desktop design and wish to have it tessellate across your screen, but you can't find the "Tile" option on your computer. Given a design and the number of times you want it to repeat in each direction, print out the resulting pattern.

Input:

An integer, *d*, the total number of designs.
For each design:
Three space separated integers, *h*, *w*, and *n*: the height of the design ($1 \leq h \leq 10$), the width of the design ($1 \leq w \leq 10$), and the number of times you want the design repeated ($1 \leq n \leq 10$).
For the next *h* lines:
A string of *w* characters.

Output:

For each design, print the design repeated *n* times vertically and horizontally, followed by a new line (except for the last design). There should be *d* designs outputted.

Sample Input:

```
1
3 4 2
XXXX
XOOX
XXXX
```

Sample Output:

```
XXXXXXXX
XOOXXOOX
XXXXXXXX
XXXXXXXX
XOOXXOOX
XXXXXXXX
```

6. Hedge Maze (180)
Input File: hedgemaze.txt

Meow the Cat (aka Chairman Meow) is playing with a ball of red yarn. Unfortunately, she bats the yarn a little too hard, sending it flying over a conveniently placed hedge maze!

She is currently on the ground on the North side of the maze and would like to reach the other side of the maze, but she gets lost easily. Luckily, since she is a cat, she can jump onto the hedges in a straight line path from the North side to the South side of the maze. However, since she is a cat, she is lazy and does not like jumping onto and down from hedges. Please help Meow determine the minimum number times she must jump on a hedge to get to the other side. Note that if Meow is currently on a hedge, she does not need to jump a square with a hedge that is one square to the South of her location.

Input:

An integer, *n*, the number of mazes.
For each maze:
An integer, *s*, the size of the maze ($1 \leq s \leq 10$).
For the next *s* lines:
A string of length *s* consisting of spaces, which represent empty space, and the character *X*, which represents a hedge.

Output:

For each maze, the minimum number of jumps Meow must take to get from the top to the bottom of the maze if she travels in a vertical path. There should be *n* values outputted, one value per line.

Example Input:

```
1
6
X X X
  X XX
XX   X
 X X X
   X X
XXXX X
```

Example Output:

```
1
```

7. Logic Puzzle (190)

Input File: logic.txt

Your school is attempting to showcase the detriments of student hypercompetitiveness by running a (completely ethical) experiment on a group of 10 students. First, each student takes a 100 question test; their score on the test is the number of correctly answered questions. Instead of telling students their score, they grab a few pairs of students and tell them who scored higher (this was supposed to impress on them that comparing scores causes emotional damage). After this process, the experimenters offer the students a prize for collaboratively comparing their scores.

The students are named Eno, Owt, Reeth, Rouf, Vife, Xis, Neves, Thige, Enin, and Ent. The students know the following information:

Xis scored higher than Eno.
Ent scored higher than Vife.
Neves scored higher than Rouf.
Thige scored higher than Owt.
Vife scored higher than Reeth.
Enin scored higher than Owt.
Eno scored higher than Neves.
Thige scored higher than Rouf.
Eno scored higher than Vife.
Ent scored the same as Owt.
Neves scored the same as Enin.

Input:

An integer, *n*, the number of questions.
For the next *n* lines:
Two space separated strings, *a* and *b*, which represents the question “Does student *a* or student *b* have a higher score?” It is guaranteed that *a* and *b* are names of different students.

Output:

For each question, print the name of the student with the higher score. If they have the same score, “=” should be printed. If the answer cannot be determined, “?” should be printed. A total of *n* answers should be printed, one answer per line.

Sample Input:

```
2
Owt Thige
Ent Reeth
```

Sample Output:

```
Thige
Ent
```

8. Sandstorm (200)
Input File: sandstorm.txt

For no apparent reason you decide to go camping in the desert with a group of friends. You each set up a tent and secure it with several stakes. Exhausted, you go to sleep.
A howling sandstorm awakens you. All the stakes except for one have been uprooted! You frantically scramble to secure the rest of the stakes before your tent blows away.
You have ***s*** identical stakes in total for securing the tent. Each stake except for the one rooted stake has an endurance value ***e***, which is the number of seconds it takes to plant the stake. If there are ***n*** stakes in the ground when the stake is fully planted (including the stake just planted and the rooted stake), a stake with endurance value ***e*** can withstand

$(\boldsymbol{e}^3 / (4\boldsymbol{s} - 4\boldsymbol{n}) + 0.00001)$

seconds of the storm before blowing away. The rooted stake will not blow away.
What is the minimum number of seconds needed to secure all the stakes?

Input:

An integer, *t*, the number of tents.
For each tent:
Two space separated integers, *s* and *e*, the number of stakes ($1 < s < 10000$) and the endurance value of each stake ($1 < e < 10000$).

Output:

For each tent, the minimum number of seconds needed to secure all the stakes. If it is not possible to do so, “Good luck!” should be printed. There should be *n* lines of output.

Sample Input:

```
2
2 5
8 6
```

Sample Output:

```
5
Good luck!
```

9. Ristet (210)
Input File: ristet.txt

Mathus and Jonathan are playing a game called *Ristet*. In *Ristet*, players attempt to form a complete horizontal or vertical line on a square grid of unit cells. Players take turns shading a square with side length less than or equal to a predetermined number anywhere onto the grid, as long as at least one cell becomes shaded: the shaded square can contain cells that are already shaded. It is Jonathan's turn and he would like to know the smallest size square needed to win in one move.

Input:

An integer, *n*, the number of positions.
For each position:
Two space separated integers, *x* and *s*: the width and height of the grid ($2 \leq x \leq 1000$) and the maximum side length of a square ($1 \leq s \leq x$).
For the next *s* lines:
A string of length *s* consisting of spaces and the character *X*. A space represents unshaded squares and an *X* represents a shaded square. It is guaranteed that a player has not already won.

Output:

For each position, print the side length of the smallest square needed to win in one move. If no such square exists, "Impossible" should be printed. There should be *n* lines of output.

Example Input:

```
2
4 4
  XX
 X X
X X
XX
4 1
X  X
 X X
 XX
X X
```

Example Output:

```
2
Impossible
```

10. Super Bowling (220)

Input File: bowling.txt

You are playing Super Bowling in Physical Education class today! Nothing is more fun than trying to roll explosive bowling balls towards pins when there are no lanes, no bumpers, and some bowling ball paths intersect...

Everyone starts at a different location and rolls one bowling ball. If two bowling balls collide (which happens when their centers are less than one meter apart), they explode! The nervous TA, standing far away from the action, would like to know how many bowling balls remain unexploded after certain periods of time.

Input:

Two space separated integers, n and q: the number of bowling balls ($2 \le n < 50$) and the number of times the TA will inquire about the number of remaining bowling balls ($1 \le q < 100$).

For the next n lines:

Four space separated integers, x, y, vx, and vy: how much to the East in meters the bowling ball is from the TA ($10 \le x \le 100$), how much to the North in meters the bowling ball is from the TA ($10 \le y \le 100$), the Eastward velocity in meters per second of the bowling ball ($0 \le vx \le 10$), and the Northward velocity in meters per second of the bowling ball ($0 \le vy \le 10$). It is guaranteed that no two bowling balls will start at the same location and a collision will involve exactly two bowling balls.

For the next q lines:

An integer, t, the time in seconds after the bowling balls start rolling the TA will ask for the number of remaining bowling balls ($0 \le t < 100$).

Output:

For each time, the number of unexploded bowling balls. There should be q numbers outputted, one number per line.

Example Input:

```
4 2
16 16 1 1
22 10 0 2
25 10 0 1
16 19 1 1
3
6
```

Example Output:

```
4
2
```

11. Parkour run (230)
Input File: parkour.txt

Having escaped with a bag of not-so-liquid-resistant artifacts, you decide that it's time to leave the city in your brand-new-definitely-not-stolen helicopter. Unfortunately, a nearby cherry coke geyser erupts, sending sugary waves towards you!
You want to get from your starting location to the helicopter, which is located on the top of a building. There are several nearby towers of crates that you can jump on. It takes you 1 second to move between adjacent crate towers, and you can only move between adjacent crate towers with a height difference of at most one crate. The geyser will flood a level every few seconds, destroying any towers less than the fluid level of the geyser. Will you reach the helicopter before the geyser destroys all the crates?

Input:

An integer, *n*, the number of test cases.
For each test case:
Two space separated integers, *s* and *g*: the length and width of the square of crate towers ($1 < s \leq 20$) and the time it takes for the geyser to raise the water level by one crate ($1 \leq g < 20$).
For the next *s* lines:
s space separated integers, which are the heights of the crate towers (positive integers less than 1000). The line index is the x-coordinate and the character index is the y-coordinate of the corresponding crate tower.

Output:

For each test case, the least amount of time, in seconds, to reach the helicopter (bottom right square) from your starting point (top left square). If it is not possible to reach the helicopter, "Get a jetpack." should be outputted. There should be *n* lines of output.

Example Input:

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

Example Output:

```
14
```

12. Websites (240)

Input File: websites.txt

You are clicking random links on your friends' websites when you experience deja vu: you've reached the same page you started on!
You wonder how many pages on each of your friends' sites can be reached by starting on that page and following a trail of links. No page will contain a link to itself.

Input:

An integer, *n*, the number of data sets.
For each data set:
An integer, *w*, the number of different pages on a friend's website ($0 < w < 200$).
w sets of 2 lines. In each set:
On the first line, the name of the page (a 1-10 character alphanumeric string) and the number of links ($0 \leq l < w$) on that page, separated by a space.
On the second line, *l* space separated names of other pages the current page contains links to.

Output:

For each data set, the number of pages that can be reached by starting on that page and clicking on links. There should be *n* values outputted, one value per line.

Example Input:

```
2
3
HomePage 1
Page1
Page1 0

GoBack 2
Page1 HomePage
3
Then 1
Next
Next 1
Then
First 2
Then Next
```

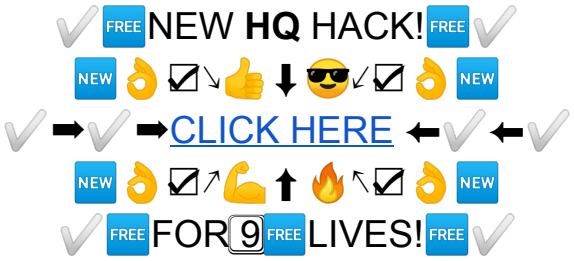
Example Output:

```
0
2
```

13. Virus (250)

Input File: virus.txt

You are online shopping for Kiwi slicers when you see the following pop up ad:



Ignoring all the tech advice you've received over the years, you click on the link. A virus immediately downloads itself onto your computer and infects your network…*

Each of the *c* computers (numbered 1, 2, 3, …, *c*) in the network can withstand the virus for various lengths of time before becoming infected and transmitting the virus to all its neighbors. This process continues until all the computers in the network are infected by the virus.

You are wondering the longest possible time the virus could take to infect the entire network if it could start from any computer. It is guaranteed that the virus will infect the entire network in a finite amount of time.

Input:

An integer, *n*, the number of data sets.
For each data set:
An integer, *c*, the number of computers in the network ($1 < c < 10000$).
A line with *c* space separated integers. The *i*th integer, c_i, is the number of seconds computer *i* can withstand the virus before sending it to all its neighbors ($0 \leq c_i < 1000$).
For the next *c* - 1 lines:
Two space separated integers, *a* and *b*, denoting that computers *a* and *b* are neighbors.

Output:

For each data set, the longest possible time, in seconds, that the virus could take to infect the entire network if it could start from any computer. There should be *n* values outputted, one value per line.

Example Input:

```
1
5
5 8 2 1 9
2 4
1 3
4 5
3 4
```

Example Output:

```
18
```

14. Mushrooms (260)
Input file: mushrooms.txt

You are in a mythical forest and want to gather giant mushrooms. After 7 hours of trudging, you finally find a giant mushroom surrounded by a grove of ***b*** pineapple bushes. As every experienced mushroom gatherer knows, one must collect at least ***b*** pineapples for self defense, otherwise the flying pineapple monster will transmogrify you into a pineapple (yikes!)! Since it is getting dark, you only have enough time for one revolution around the bushes. Each bush contains various numbers of pineapples. To remove all ***p*** pineapples from a bush, you need to throw p^2 apples at the bush. You have to carry all the pineapples collected; every time you finish passing a bush (including the first and last ones), regardless of whether you collected any pineapples at that bush, you eat one apple per pineapple collected.

Since your apple reserves are running low, you wonder what is the smallest number of apples you need to harvest the giant mushroom.

Input:

An integer, *n*, the number of data sets.
For each data set:
An integer, *b*, the number of pineapple bushes ($0 < b < 1000$).
b space separated non-negative integers; the *i*th integer is the number of pineapples on the *i*th bush you visit. Each bush contains no more than 1000 pineapples, and it is guaranteed that there are a total of at least *b* pineapples between all the bushes.

Output:

For each data set, the smallest number of apples you need to obtain the required amount of pineapples. There should be *n* values outputted, one value per line.

Example Input:

```
1
3
2 1 1
```

Example Output:

```
12
```

15. Basketbob (270)
Input File: basketbob.txt

Builder Bob would like to construct several new square arenas for people to play a game he calls *Basketbob*. In the game, two players each take turns trying to *bob* a cube into a square *basket*.

There are several baskets in the arena, guaranteed to not intersect. While he has spectacularly precise measurements for the dimensions of the arena and the baskets, he unfortunately spilled bacon flavored root beer on his measurements for the dimension of the cube. He only remembers that the side length of the cube is at most the length of the smallest basket and at least one unit, and there exists at least one position on the arena such that the edge of the cube is at least k units from all baskets and the walls.
Please help Bob by determining the largest possible side length of the cube.

Input:

An integer, n, the number of data sets.
For each data set:
Three space separated integers, l, b, and k: the side length of the arena ($5 \leq l < 100000$), the number of baskets in the arena ($1 \leq b < 100$), and the minimum distance of the cube from baskets and walls ($0 \leq k < 100000$).
For the next b lines:
Three space separated integers, x, y, and s: the distance from the West side of the basket to the West wall of the arena ($0 \leq x < l$), the distance from the North side of the basket to the North wall of the arena ($0 \leq y < l$), and the side length of the basket ($1 \leq s < l$).

Output:

For each data set, the largest possible side length of the cube. There should be n values outputted, one value per line.

Example Input:

```
1
7 2 1
0 0 3
3 4 3
```

Example Output:

```
2
```