Catlin Gabel 2018 Fall Programming Contest

Intermediate 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. Triforce (130)
2. Angles (140)
3. Interception (150)
4. Apples (160)
5. BPM Matching (170)
6. Homework (180)
7. Desktop (190)
8. Hedge Maze (200)
9. Find the Treasure (210)
10. Logic Puzzle (220)
11. Performance (230)
12. Solitaire (240)
13. Sandstorm (250)
14. Ristet (260)
15. Super Bowling (270)

1. Triforce (130)
Input File: None.

While many believe that Link defeated Ganon in the Legend of Zelda in an epic battle, they actually staged the fight and agreed to play a round of poker for the Triforce, which Link lost. Link thus needs a Triforce like the one below! Please output the pattern exactly as shown below, as only one Triforce will appease Hyrule.

Input:

None.

Output:

Please output the pattern exactly as shown below.

Sample Output:

```
   /\
  /__\
 /\  /\
/__\/__\
```

2. Angles (140)
Input File: angles.txt

The sum of the angle measures in a triangle is 180 degrees. Given the angle measures of two angles of a triangle, determine the angle measure of the third angle.

Input:

An integer *n*, the total number of triangles.
For the next *n* lines:
2 space-separated integers *A* and *B*, which are angle measures in degrees ($0 < A, B < 180$).

Output:

For each triangle, the angle measure of the third angle. There should be *n* values outputted, one value per line.

Sample Input:

```
2
30 60
90 45
```

Sample Output:

```
90
45
```

3. Interception (150)
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
```

4. Apples (160)
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 ($1 \le b < 10$).
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 ($0 \le a < 100$).

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
```

5. BPM Matching (170)
Input File: bpm.txt

You are about to listen to your favorite music on your earphones during a school pep rally when suddenly, the school band starts to perform their own piece at the exact same time that you start your song! Normally, you can tune out your surroundings, but since the BPM of the band music is slightly different than that of your earphones' music, you find it hard to concentrate on either tune. You do notice some interesting rhythmic patterns occurring every time the beats of the two songs align. You wonder how often this alignment happens.

Input:

An integer *n*, the total number of cases.
For the next *n* lines:
Two space separated integers, *a* and *b*, the BPMs of the two songs ($16 \le a, b \le 900$).

Output:

For each case, the number of seconds (rounded to the nearest integer) between consecutive synchronized beats. There should be *n* values outputted, one value per line.

Sample Input:

```
2
60 90
120 114
```

Sample Output:

```
2
10
```

6. Homework (180)
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
```

7. Desktop (190)
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 ($1 \le h \le 10$), the width of the design ($1 \le w \le 10$), and the number of times you want the design repeated ($1 \le n \le 10$).
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
```

8. Hedge Maze (200)
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 ≤ *s* ≤ 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
```

9. Find the Treasure (210)

Input File: treasure.txt

When you landed on the desert island, you expected the stereotypical scenery of a coconut tree with indestructible coconuts. What you did not expect was a paved road leading to a cluster of crumbling obelisks. Each obelisk contains an epitaph that instructs to go to another obelisk. You follow the signs and arrive at a particularly monolithic structure (eerily resembling the Washington Monument!) marked with an X. You notice a handle. You pull the handle, anticipating treasure. Instead, you find that the small compartment contains a note, which to your chagrin reads:

YOU GOT PRANKED! GO BACK TO THE BEGINNING AND DIG UP THE BOUNTY.

Tragically, in your quest for treasure, you forgot where you began! All you remember is that you visited every obelisk once. Given a map of the site, can you determine which obelisk contains the treasure?

Input:

An integer, *n*, the number of test cases.
For each test case:
An integer, *b*, the number of structures in the city ($1 < b < 100$).
For the next *b* lines:
Two space separated strings, the name of the structure (a 1-10 character alphanumeric string) and the name of the structure to go to next. If the structure is the monolith, “X” will be the name of the structure to go to next.

Output:

For each test case, the name of the building that contains the treasure, beginning from which, following the trail of signs, you visit all other buildings once, ending at the monolith. There should be *n* names outputted, one name per line.

Example Input:

```
1
5
One Two
Three Four
Five X
Two Five
Four One
```

Example Output:

```
Three
```

10. Logic Puzzle (220)

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
```

11. Performance (230)
Input File: performance.txt

You are about to perform several piano pieces on stage! However, you did not sleep much last night thanks to a large amount of homework. You suspect that if there are too many notes in too little time, you will make a mistake, declaring it to the entire audience. You would like to know the duration of the quickest group of *n* notes.

Input:

An integer *p*, the total number of pieces.
For each piece:
Two space separated integers, *t* and *n*: the total number of notes in the piece ($5 \le t \le 200$) and the size of the group ($2 \le n \le t$).
A line with *t* space-separated non-negative integers; the *i*th integer is the number of milliseconds the *i*th note occurs after the start of the song. All notes will occur within 1000000 milliseconds after the start of the song.

Output:

For each piece, the shortest amount of time for the quickest group of *n* notes, in milliseconds. There should be *p* values outputted, one value per line.

Example Input:

```
2
6 2
40 1040 1540 2540 4540 6040
10 4
1000 1500 2000 4000 5000 6000 6333 6667 7000 10000
```

Example Output:

```
500
1000
```

12. Solitaire (240)

Input File: solitaire.txt

Since all your friends are playing catch with watermelons, you decide to play some solitaire. You have a lot of experience playing solitaire, so to test your luck you decide to deal all the cards into several face up piles and attempt to complete the game without moving any cards.
A deck of cards consists of 52 cards, divided into four suits, which are Spades (represented by “S”), Clubs (represented by “C”), Hearts (represented by “H”), and Diamonds (represented by “D”). Each suit consists of 13 ranks, which from lowest to highest are Ace (represented by “A”), 2, 3, 4, 5, 6, 7, 8, 9, 10 (represented by “T”), Jack (represented by “J”), Queen (represented by “Q”), and King (represented by “K”).
To complete a game, you need to collect all the cards. A card can be collected if it is the lowest ranked uncollected card of a suit and is the top card of a pile. The next card will then become the top card of the pile.

Input:

An integer, *n*, the number of games.
For each game:
An integer, *p*, the number of piles of cards.
p sets of 2 lines. In each set:
On the first line, a positive integer, *c*, the number of cards in the pile.
On the second line, *c* space separated names of cards (2 character alphanumeric strings in the format of rank then suit; for example 3C represents the Three of Clubs), from bottom to top.

Output:

For each game, print “True” if all the cards can be collected, and “False” if not. There should be *n* values outputted, one value per line.

Example Input:

```
2
1
52
KD QD JD TD 9D 8D 7D 6D 5D 4D 3D 2D AD KH QH JH TH 9H 8H 7H 6H 5H 4H 3H 2H AH KC QC JC TC 9C 8C 7C 6C 5C 4C 3C 2C AC KS QS JS TS 9S 8S 7S 6S 5S 4S 3S 2S AS
4
13
AS 2S 3S 4S 5S 6S 7S 8S 9S TS JS QS KS
13
AC 2C 3C 4C 5C 6C 7C 8C 9C TC JC QC KC
13
AH 2H 3H 4H 5H 6H 7H 8H 9H TH JH QH KH
13
AD 2D 3D 4D 5D 6D 7D 8D 9D TD JD QD KD
```

Example Output:

```
True
False
```

13. Sandstorm (250)
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 $\boldsymbol{s}$ identical stakes in total for securing the tent. Each stake except for the one rooted stake has an endurance value $\boldsymbol{e}$, which is the number of seconds it takes to plant the stake. If there are $\boldsymbol{n}$ stakes in the ground when the stake is fully planted (including the stake just planted and the rooted stake), a stake with endurance value $\boldsymbol{e}$ 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!
```

14. Ristet (260)
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 ≤ *x* ≤ 1000) and the maximum side length of a square (1 ≤ *s* ≤ *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
```

15. Super Bowling (270)
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
```