Eleventh Annual
University of Central Florida

High School Programming Tournament: Online Edition

Problems – Division 2

Problem Name	**Filename**
HSPT and the Chocolate Factory	chocolate
Jenny's Number	jenny
Mario's Maze	maze
North by Northwest	north
Forgotten Password	password
Radio Stations	radio
Snack Sort	snack
Healing Sniper	sniper
Sue in Bourbon	suburban

Call your program file:
filename.c, *filename*.cpp, *filename*.java, or *filename*.py

For example, if you are solving North by Northwest,
Call your program file:
north.c, north.cpp, north.java, or north.py
Call your Java class: north

HSPT and the Chocolate Factory

Filename: `chocolate`

The members of the HSPT judges work very hard to make the contest a fun and memorable contest. As such, Glenn, chief judge, wishes to give all the members one chocolate for each problem on which they worked.

Each judge is given a unique ID numbered from 1 to n. This year there's an interesting property of the judges, though: if person i and person j both worked on the same problem, it is guaranteed that all people k such that $i \le k \le j$ also worked on the same problem. This allows Glenn to order the people from left-to-right in an increasing order according to their ID numbers and thus problems to be defined by a range L to R of all the people who contributed to this problem.

Glenn has a machine that, given two numbers L and R, gives **exactly one chocolate** to each person starting from L and advancing rightward going up to R. Currently, for each problem Glenn enters the corresponding L and R values for the people who worked on the problem. However, this can be time-consuming, and being a problem-solver himself, Glenn would like to know the minimum number of times he has to enter values into the machine with an optimal selection of intervals that results in each person receiving the same number of chocolates at the end.

For example, if the final array of chocolates should be [1, 2, 2, 3, 3, 2, 2, 1], then the chocolates could be distributed in three intervals: [1, 8], [2, 7], and [4, 5]. Therefore, Glenn can distribute the chocolates with only three operations of the machine.

The Problem:

Given n, the number of HSPT team members, m, the number of problems, and the judges interval, determine the minimum number of times Glenn has to operate the chocolate-giving machine.

The Input:

The first line of input will contain a single integer, c, representing the number of contests to process. The first line of each contest contains two integers, n and m ($1 \le n \le 10^5$; $1 \le m \le 10^5$), representing the highest judge ID and the number of problems, respectively. The following m lines contain two integers, L and R ($1 \le L \le R \le n$), which represents the interval of judges that worked on this problem.

The Output:

For each contest, output a single integer, x, on a newline, where x is the minimum number of times Glenn has to operate the machine and still results in each person receiving the correct number of chocolates at the end.

Sample Input:

```
2
8 6
1 3
6 8
5 7
4 5
4 5
2 4
5 3
1 3
2 4
3 5
```

Sample Output:

```
3
3
```

Jenny's Number

Filename: `jenny`

Brice is head over heels for his crush, Jenny, a promising and beautiful Math major at UCF. Everyone knows that Jenny's phone number is 867-5309. Brice knows that Jenny, being infatuated with mathematical quirks, will only enter somebody's phone number into her phone if they follow these three criteria:

- The number is a prime
- The next odd number is a prime
- The digits in the number are all distinct

We'll call numbers that pass these criteria "Jenny Numbers." As you might have figured out, 867-5309 is a Jenny number.

Brice was about to get a new phone anyway, so he figures he should make the most of it and choose a number such that he has a chance that Jenny would put his number into her phone.

The Problem:

Given the number of digits d in a phone number, print the ith jenny number with d digits. Leading zeroes may not count towards a number's digit count.

The Input:

The first line of the input will contain a single integer, t, representing the number of tests. The first line of each test will contain two integers, d and i ($0 < d < 100$; $0 < i < 10^5$), representing the number of digits in a phone number and the ith one to print, respectively.

The Output:

For each test, output the ith Jenny number with d digits, or if one does not exist, instead output "`Brice doesn't stand a chance!`"

Sample Input:

```
2
2 3
1 9
```

Sample Output:

```
41
Brice doesn't stand a chance!
```

Mario’s Maze

Filename: `maze`

Mario has just received news that Princess Peach is trapped in a far-away castle and needs to be rescued. The castle can be represented as a rectangular grid, where each room is a single story within the building at a floor between 1 and 9, inclusive. For whatever reason, there are no rooms above other rooms. In addition to rooms, there are elevators, giant fans, and poles. The elevators can transfer someone between any 2 rooms that are both adjacent (north, south, east, or west) to the elevator, regardless of what story they are on. The fans are positioned on the ground and blow upward, so when someone enters the space containing a fan, they are directed upward, and thus may easily transport to a strictly higher room that is directly north, south, east, or west of the fan. Finally, the poles can be used just like the fans, but to travel strictly downward to an adjacent room, by sliding down like a fireman.

When Mario arrives at the castle, he may enter from any side along the border, as long as he enters a room on the first story. He can’t enter a room with an elevator, fan, or pole from outside the castle. Once inside, he may travel north, south, east, or west to adjacent rooms as many times as required to reach the princess, but he can only travel to an adjacent room if it is on the same story or a level up or down (Mario can only jump so far). Of course, Mario can also use the elevators, fans, and poles to get around.

The princess is located on the 9th story. Once he reaches the princess, he will use a magic mushroom to teleport them back home. Unfortunately, he only has one magic mushroom. Otherwise, he would just use it to teleport to the princess as well.

There’s one more thing. Some of the rooms may contain poisonous gas. It may be wise for Mario to avoid those rooms or any room that is adjacent to them (north, south, east, or west).

The Problem:

Determine if it is possible to rescue the princess while avoiding the poison gas rooms and rooms adjacent to poisonous gas, or if Mario will have to find another way.

The Input:

The first line will contain an integer, *t*, the number of test cases. For every test case, there will first be a line containing 2 integers, *n* and *m* ($3 \le n \le 100$; $3 \le m \le 100$), representing the dimensions of the castle, respectively. Each of the next *n* lines each contains *m* characters, representing the rooms of the castle. A number of 1 to 9 indicates the height of the room (1 is the first story). E means there’s an elevator in that space, F indicates a giant fan, P stands for pole, G means there’s poisonous gas, and * is the cell where the princess is being held. It is guaranteed that no two adjacent spaces will both contain one of the 4 special spaces (elevators, fans, poles, or poisonous gas). Also, the princess will not be located along the border of the castle or in a room adjacent to poisonous gas.

The Output:

For each test case, output a single line: "`You've got this!`" if Mario can reach the princess or "`Find another way`" if it's impossible to reach her.

Sample Input:

```
2
9 9
955443322
96999999P
979956677
989769999
9*9699977
999799786
9G8F4561E
999392232
999991999
3 3
234
5*6
789
```

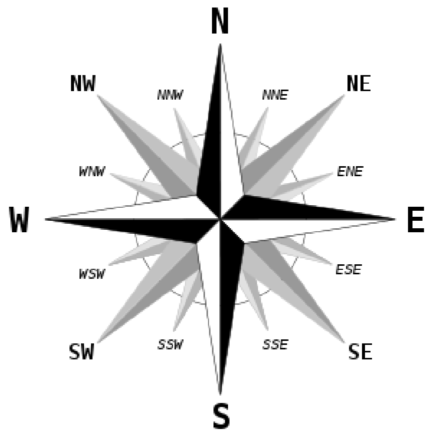
Sample Output:

```
You've got this!
Find another way
```

North by Northwest

Filename: `north`

Billy always hears the weathermen on TV refer to cardinal directions in the most complicated way possible. They say the direction of the wind is "East-northeast" when they really just mean "Almost east", or they say "South-southwest" when they could easily say "South-ish". If only there was a simple numerical system that Billy could understand…



The Problem:

Convert each cardinal direction to a number representing it in degrees, where north is 0°, east is 90°, south is 180°, and west is 270°.

The Input:

The first line contains an integer, *d*, the number of directions to convert. The next *d* lines each contain a cardinal direction represented by 1, 2, or 3 characters. It is guaranteed that each direction will be valid (meaning one of the directions in the picture above).

The Output:

For each direction, output a single line in the following format: "`d is n degrees`" where *d* is the direction given in the input and *n* is the decimal representation of the number of degrees ($0 \leq n < 360$). Round the decimal and output to one decimal place.

Sample Input:

```
2
ENE
S
```

Sample Output:

```
ENE is 67.5 degrees
S is 180.0 degrees
```

Forgotten Password

Filename: `password`

Amon has been trying to log onto CodeForces to brush up on his coding skills for an upcoming interview. Unfortunately, it's been several years since Amon has solved a single problem, so he's having trouble remembering what his password was. All he remembers about his password is that it is composed only of the digits 0-9 and every substring of the password that doesn't begin with '0' is divisible by 3, his favorite number. Here, a substring is a contiguous sequence of digits within a string.

For example, we'll try the password 403065. Several of its substrings are divisible by 3, including 30 and 6. But this couldn't be Amon's password because it contains substrings that aren't divisible by 3, such as 40 and 65.

Amon doesn't remember anything else about his password, but he has a few guesses for its length.

The Problem:

Help Amon figure out how many possible passwords exist for each given length where all substrings are divisible by 3, so he can decide if it's worth his time to try and guess his password.

The Input:

The first line will contain an integer, *t*, denoting the number of lengths Amon wants to try. Each of the next *t* lines will contain a single integer, *n* ($1 \leq n \leq 15$), representing the length of the password.

The Output:

For each length, output a single line containing the number of different passwords that exist for that length.

Sample Input:

```
2
1
3
```

Sample Output:

```
4
64
```

Radio Stations

Filename: `radio`

Pablo likes to listen to the radio while driving, and he has several favorite stations that he switches between by pressing the preset radio station buttons in his car. Since he will need extra space to store his maple syrup and Tim Hortons donuts for the return trip, Pablo is getting a rental car for his upcoming road trip to Canada. Unfortunately, since this will be his first time in that car, the preset radio stations will likely be uninteresting to him.

Pablo's on a tight schedule and knows he won't have time to figure out how to change the preset stations. Instead, he'll use the stations that are already set as well as the seek up/down buttons, which change the station to the next valid higher/lower station, wrapping around if it's already set to the highest/lowest station.

The Problem:

Please help Pablo out by telling him the optimal number of button presses needed to switch from one station to the other.

The Input:

The first line will contain an integer, d, denoting the number of days the trip will last. For each day, there will be several lines. The first of these lines will contain 3 integers, n, p, and s ($1 \le p \le n$; $2 \le n \le 100$; $1 \le s \le 100$), representing the number of radio stations available that day, the number of preset stations in the rental car, and the number of switches Pablo will perform, respectively. The second line contains n numbers, each representing the frequency of an available station ($88 < f < 108$). These numbers will be given in sorted order and will all contain one (odd) decimal place. The third line contains p numbers, the preset radio stations. These will also be given in sorted order and each one will be one of the n stations above. The next s lines will each contain 2 numbers. These numbers are station frequencies from above and will be different from each other.

The Output:

For each day, first output a single line in following format: "`Day #`*`i`*`:`" where i is the day number, starting with 1. For each of the next s lines for that day, output the minimum number of button presses required to switch from the first station to the second.

(Sample Input and Sample Output follow on next page)

Sample Input:

```
2
2 1 1
88.5 107.9
88.5
88.5 107.9
11 3 3
89.9 90.1 90.5 92.3 93.7 94.9 95.3 96.7 97.5 98.3 101.1
92.3 94.9 97.5
89.9 101.1
101.1 92.3
94.9 90.1
```

Sample Output:

```
Day #1:
1

Day #2:
1
1
3
```

Snack Sort

Filename: `snack`

Om Nom just loves his candy! So much so that he doesn't want to eat any other kind of snacks. Therefore, he wants you to help him sort the snacks in each of his cabinets. Luckily, in his world, candies have the most number of vowels in their names (and note that "y" is considered a vowel and is worth double!). So he just wants you to sort the snacks by number of vowels in their names (again, "y" counts as two). If multiple snacks have the same number of vowels, then sort those subset of snacks in alphabetical order by the entire name. We'll call this Snack Sort!

The Problem:

Given a list of candies in a cabinet, sort them using Snack Sort.

The Input:

Input will begin with a single, positive integer, *c*, on a line by itself, representing the number of cabinets that Om Nom has. Following this there will be *c* cabinet descriptions. Each cabinet starts with a line containing a single positive integer, *s* ($1 \le s \le 200$), representing the number of snacks in the cabinet. Then, on each of the following *s* lines, there will be a single string of lower-case letters containing between 1 and 80 characters.

The Output:

For each cabinet, output the header "`Cabinet #`*`i`*`:`" on a line by itself where *i* is the number of the cabinet in the file (starting with 1). After this, output the sorted snacks, one snack per line. Leave a blank line after the output for each cabinet.

Sample Input:

```
2
3
pretzels
gum
chips
1
licorice
```

Sample Output:

```
Cabinet #1:
pretzels
chips
gum

Cabinet #2:
licorice
```

Healing Sniper

Filename: `sniper`

There are n soldiers on a battlefield arranged in a row. Some may be allied soldiers, and some may be enemy soldiers. Ana is equipped with a healing sniper; it fires up to k specially-designed piercing healing bullets. Each bullet i will increase the health of the first b_i soldiers from the left by 1, regardless of whether or not they are allied.

The Problem:

The strength of the allied soldiers is the sum of their health. Likewise, the strength of the enemy soldiers is the sum of their health. Ana must make the battle as favorable as possible for her allies. She must fire some of her bullets such that the difference between the allied strength and the enemy strength is as high as possible.

The Input:

The first line of the input file may begin with a single positive integer, t, representing the number of missions. For each mission, three lines follow. The first contains two integers, n and k ($1 \leq n \leq 10^5$; $1 \leq k \leq 10^5$), representing the number of soldiers, and the number of bullets, respectively. The next line contains n integers, h_i ($-2{,}147{,}483{,}648 \leq h_i \leq 2{,}147{,}483{,}647$; $h \neq 0$), representing the current health of each soldier. If the corresponding health value is positive, then that soldier is an ally with health h_i; if the health value is negative, then that soldier is an enemy with health $|h_i|$. The last line contains k integers, b_i ($1 \leq b_i \leq n$), representing the description of each bullet.

The Output:

For each mission, output a single line with a single integer, the maximum difference between allied and enemy strength.

Sample Input:

```
2
6 5
3 -2 -4 1 5 -1
3 5 4 1 3
1 2
-1
1 1
```

Sample Output:

```
4
-1
```

Sue in Bourbon

Filename: `suburban`

Sue is in charge of building a new suburban community in Bourbon County, Kentucky, and has to perform inspections around the town to make sure everything is being built properly. The suburbs are based on a grid, where the grid lines represent streets and each destination only lies on the intersection of two streets. Every intersection is at a lattice point (a point where both coordinates are integers). Sue has to drive along the streets, so depending on where the points are she may not be able to simply drive in a straight line.

The Problem:

Given the Euclidean distance between Sue's intersection and a destination intersection, what is the shortest Manhattan distance that could correspond to some destination intersection?

The Input:

The first line of the input file will begin with a single positive integer, *t*, representing the number of distances. For each distance, there will be a single line containing a single positive integer, *x* ($1 \le x \le 10^9$), such that $\sqrt{\boldsymbol{x}}$ is the Euclidean distance.

The Output:

For each distance, output a single integer, the shortest possible Manhattan distance. If there is no such possible Manhattan distance, print -1 instead.

Sample Input:

```
4
5
25
3
2
```

Sample Output:

```
3
5
-1
2
```