Interhouse IT Programming Competition 2020

Hyderabad Public School (Begumpet)

Before You Begin:

- All solutions ***must*** be developed in java and be provided in .java files
- Each file ***must*** contain a comment block at the top including your team name, name of the participants and their class and section
- Solutions will not only be scored on a working and efficient answer but also on the object-oriented style of code and simplicity of the solution
- Documenting your code or the algorithm in the form of comments or a flow diagram in the case of a non-working solution will also earn the solution a partial score. However, in all other cases you are expected to write simple and self-explanatory code.
- Questions with higher difficulty will carry a much higher score.

Question 1 - Pattern

Your first task will be a simple pattern-based output. The pattern will consist of a number diamond in increasing order towards the center.

I/O: Input will consist of a single line i.e., the greatest number in the diamond pattern. Output will be the diamond pattern of the numbers from the smallest on the edges to the greatest in the middle.

Input
```
5
```

Output
```
        1
      1 2 1
    1 2 3 2 1
  1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
  1 2 3 4 3 2 1
    1 2 3 2 1
      1 2 1
        1
```

Input
```
4
```

Output
```
      1
    1 2 1
  1 2 3 2 1
1 2 3 4 3 2 1
  1 2 3 2 1
    1 2 1
      1
```

Question 2 – Search Engines

Search engines are the foundation of our modern databases and computing as a whole. Your task is to implement a simple search engine that can filter out a list of strings and rank the results accordingly.

Task 1: Every search engine implements a system to filter out results based on a generic expression known as a query. A sample query would be for example:

mic would produce the results **mic**hael, **mic**helle, **mic**ah, a**mic**able, cera**mic**.

Whereas mich__l* would result in **michael** and **michelle**

Notice the underscores and asterisks around the words, these are known as wildcards which can be placed before, after or in between words. For this task, the wildcard rules are as follows:

- * asterisk: Any number of characters 0-N
- _ underscore: Exactly one character
- ? question mark: One or zero characters

For example: child* would result in child, children and childish. Whereas child___ would only result in children and childish

Task 2: For optimum results, search engines always rank the results from best possible match at the top to the least possible match at the bottom and this is done using word groups. A word group is the first set of characters in a row before being separated. For example: in the query mich__l*, 'mic' would be the first word group and 'l' would be the second. In general, search engine ranking algorithms are extremely complicated but for the sake of this task you need to only rank the results from 1 to 3. The rules of ranking by word group are as follows, now let's take the example query *mic* and its results:

Rank 1. If the first word group appears exactly at the start of the phrase – **mic**hael, **mic**helle
Rank 2. If the first word group appears anywhere in the phrase - a**mic**able
Rank 3. If the first word group appears exactly at the end of the phrase – cera**mic**

I/O: The first line will consist of a single number N for the number of subsequent inputs. The next N-1 lines will contain either words or sentences i.e., search data, and the last line will be the search query.

Input 1

```
5
tiny
intimidate
latin
inception
*in*
```

Output 1

```
intimidate
inception
tiny
latin
```

Input 2

```
5
tiny
intimidate
latin
inception
??_in*
```

Output 2

```
tiny
latin
```

Question 3 – Load Balancing

Modern enterprise systems like Facebook and Google are built to take in billions of web requests per minute. However no single computer or server can handle that level of load and would instantly shut down. The solution for this is to run multiple servers (sometimes thousands) that each take a share of their load and do the processing simultaneously. The component responsible for distributing work to each of these servers is known as a load balancer, its job is not to do any of the work directly but to simply distribute the load based on the algorithm it sees fit. Your task is to write a simple load balancing algorithm to distribute a set of tasks to a bunch of servers, *in a given minute*.

Now in reality not every server has the same capability, instead an organization can sometimes decide to buy new high-performing servers and have them work alongside old low-performing ones. Each server has a finite set of resources (processor, ram & disk space) and calculating the capability of a server is a very complex algorithm but for the sake of this task we will give it a simple combined score called **RU/m** (Resource Units per Minute). Now let's say a server has a capability of 200 RU/m and it has to process two types of tasks **X** and **Y** each taking up 20 and 40 RUs, so this means that ideally the server can handle **6X** and **2Y** tasks in a minute for maximum utilization. In reality, these tasks can be for example: clicking the like button or editing your profile picture, while the 'like' is a very simple task and does not need much RUs whereas the latter is very graphics intensive and will require way more RUs.

The load balancer splits the tasks using what's called a **Round Robin** method. This simply means distributing one task to each server turn by turn and then repeating the process for the leftover tasks, exactly like how cards are distributed to players in a poker game, one by one per player.

I/O: The first line will contain the number of lines of input N. the next line will be a space separated list of numbers, each representing the RU/m of that server. The next N-2 lines will be task definitions (a single character) and their RUs, this will be in the form of **A 30** or **B 25**. The last line will be a string of these characters in the order of their arrival from left to right.

The output for M servers will contain M+1 lines. Each line consisting of a string of the above characters, representing the tasks the server has been given for that minute. If a task could not be given to a server due to lack of RU/m then simply move on to the next server. The last line will be the left-over tasks from the original queue.

Input	**Output**
5	AACCBB
200 100 100 200	ABC
A 50	BCA
B 30	CACAA
C 20	A
AABCABCACACCABCABA	

Explanation:

The load balancer distributes these tasks in 7 iterations to servers M1 (200 RU/m), M2 (100 RU/m), M3 (100 RU/m) and M4 (200 RU/m), in that given minute.

Iteration 1 – M1 gets A, M2 gets A, M3 gets B and M4 gets C. Their each remaining RU/m is 150, 50, 70 and 180. *Leftover Queue: ABCACACCABCABA*

Iteration 2 – M1 gets A, M2 gets B, M3 gets C and M4 gets A. Their each remaining RU/m is 100, 20, 50 and 130. *Leftover Queue: CACCABCABA*

Iteration 3 – M1 gets C, M2 cannot get A due to insufficient RU/m therefore M3 gets A and M4 gets the next C. Their each remaining RU/m is 80, 20, 0 and 110. *Leftover Queue: CABCABA*

Iteration 4 – M1 gets C, neither M2 nor M3 can get the next A due to insufficient RU/m and therefore M4 gets A. Their each remaining RU/m is 60, 20, 0 and 60. *Leftover Queue: BCABA*

Iteration 5 – M1 gets B, M2 gets C, M3 has no RU/m left and M4 gets A. Their each remaining RU/m is 30, 0, 0 and 10. *Leftover Queue: BA*

Iteration 6 – M1 gets B, M2 and M3 have no RU/m left, and M4 has insufficient RU/m. Their each remaining RU/m is 0, 0, 0 and 10. *Leftover Queue: A*

Iteration 7 – No server has enough RU/m to take A and the algorithm stops. Their each remaining RU/m is 0, 0, 0 and 10. *Leftover Queue: A*

Question 4 – IP Ranges

Just like how your ISP assigns a unique IP address to your internet connection, your router also assigns a unique local IP address to each device that is using the WiFi or is connected to the router. Now to check if your phone and computer are connected to the same network, the router does something called **IP Scanning,** which simply means checking every IP address within a given range and seeing if it responds to any kind of ping. Your task is to write an algorithm to check if an IP address exists within a range.

An IP address is 4 numbers each between 0 and 255 separated by a period. So all the way from 0.0.0.0, followed by 0.0.0.1 up until 255.255.255.255. Once an IP number reaches 256, it’s value is converted to 0 and the number left to it is incremented by 1.

I/O: The first line will contain the number of lines of input N. The next two lines will contain the starting and ending range (including the ranges). Followed by the next N-2 lines of IP addresses.

The output will be of N-2 lines and will simply print out a word ‘true’ or ‘false’ to indicate if the IP falls within the range. You are also expected to print out error messages in the case of a malformed IP in the input, for example: 192.256.29.14.

Input

```
6
10.20.12.0
10.20.13.255
192.168.0.10
10.20.12.34
10.20.13.128
10.20.11.255
```

Output

```
false
true
true
false
```