DAY 15:

SESSION 1:

Photograph Height

ID:11157 Solved By 531 Users

There are **N** students in a high school class. For simplicity, the students are named **A**, **B**, **C**, . . . (up to the N^{th} letter of the alphabet). No two students have exactly the same height.

The Principal of the school wants to organize a class photograph where the students are arranged in ascending order of heights. The class teacher has been asked to list the students in ascending order of heights to send to the photographer.

The teacher does not know the heights of the various students. However, there are a set of **K** photographs, which between them have all the students in that class. From each of the photographs, the teacher obtained a list of students in ascending height order and needs to combine them into one list of students in ascending height order.

It is possible that the photographs do not fully determine the ordering in the list. In this case, as it is vacation time, the teacher needs to write to some of the students to send her their exact height, so that she can create the complete list.

The objective is to write a program that can list the students to whom she must write to determine their heights. If it is not necessary to write to any student, the output should be **N/A**

Boundary Condition(s):
2 <= N <= 26
1 <= K <= 10

Input Format:
The first line contains N and K separated by a comma.
The next K lines, each contains a comma separated list of the students in ascending order from the corresponding photograph.

Output Format:
The first line contains the list of students whose height must be known to create the ordered list separated by space or N/A.

Example Input/Output 1:
Input:
8,3
D,C,E,F,G,H
C,A,E
D,C,B,E

Output:
A B

Explanation:
Here N is **8** and K is **3.** The students are **A**, **B**, **C**, **D**, **E**, **F**, **G** and **H** (the first N letters of the alphabet).

From photographs 1, 2 and 3, we can determine that A, B, C and D are shorter than E, F, G, H, and that **E**, **F**, **G**, **H** are the last four in the ordered list.

From the first photograph, D is shorter than C, and from the second photograph, C is shorter than A. From the third photograph, C is shorter than B. The possible orders of the first four could be **D**, **C**, **A**, **B** or **D**, **C**, **B**, **A.** No information is present in any of the photographs which says which order is correct. Hence the teacher must write to both B and A to determine their heights so as to establish the correct order. The output must be sorted in alphabetic order, and hence the output is **A B**.

Example Input/Output 2:

Input:
8,3
D,C,E,F,G,H
C,A,B,E
D,B

Output:
N/A

Explanation:
Here N is **8** and K is **3**, and so there are 8 students and 3 photographs. As the students are named the first N letters of the alphabet, the students are **A**, **B**, **C**, **D**, **E**, **F**, **G** and **H**.

The first photograph shows the ordering as D, C, E, F, G, H, that is D is shorter than C who is shorter than E and so on. The second photograph shows that C is shorter than A, who is shorter than B who is shorter than E. The third photograph shows that D is shorter than B.

From photograph 1, E, F, G and H are all taller than C, and are in that order. From photograph 2, A and B are shorter than E and in that order. Hence A, B, C and D are all shorter than E, F, G and H. Hence the last four in the final ordered list must be **E**, **F**, **G**, **H** in that order.

From the first photograph, D is shorter than C, and from the second photograph, C is shorter than A who is shorter than B. Hence the order of the first four is **D**, **C**, **A**, **B**.

Hence the full list is **D**, **C**, **A**, **B**, **E**, **F**, **G**, **H**. The full order can be determined without writing to any student to get their height. Hence the output is **N/A**.

Max Execution Time Limit: 500 millisecs

Code:

```
import java.util.*;
public class photographHeight {

    public static void main(String[] args) {
        //Your Code Here
        Scanner in = new Scanner(System.in);
        String firstLine[] = in.nextLine().split(",");
        int N=Integer.parseInt(firstLine[0].trim());
        int K=Integer.parseInt(firstLine[1].trim());
        String names="ABCDEFGHIJKLMNOPQRSTUVWXYZ".substring(0,N); //getting the
range of alpabets for caculation
        List<String>photos = new ArrayList<>();
        for(int ctr=1;ctr<=K;ctr++){
            photos.add(in.nextLine().trim().replaceAll(",",""));
        }
        boolean missing=false;
        for(Character name:names.toCharArray()){
            int relCount=0;
            Queue<Character> queue = new ArrayDeque<>();
            List<Character> visited = new ArrayList<>();
            queue.add(name);
            visited.add(name);

            while(!queue.isEmpty()){ //loop to calculate for all successors
                Character student = queue.poll();
                for(String seq: photos){
                    if(seq.contains(student.toString())){
```

```
                        String succ=seq.substring(seq.indexOf(student)); //succ -
successor
                        for(Character succStud:succ.toCharArray()){
                            if(!visited.contains(succStud)){
                                queue.add(succStud);
                                visited.add(succStud);
                                relCount++;
                            }
                        }
                    }
                }
            }

            queue.add(name);
            while(!queue.isEmpty()){ //loop to calculate for all predessors
                Character student = queue.poll();
                for(String seq: photos){
                    if(seq.contains(student.toString())){
                        String
pred=seq.substring(0,seq.indexOf(student));//substring alone chamges here
                        for(Character predStud:pred.toCharArray()){
                            if(!visited.contains(predStud)){
                                queue.add(predStud);
                                visited.add(predStud);
                                relCount++;
                            }
                        }
                    }
                }
            }

            if(relCount!=N-1){
                System.out.print(name+" ");
                missing=true;
            }
        }
        if(!missing){
            System.out.print("N/A");
        }
    }
}
```

Session 2:

Minimum Jumps to Reach End

ID:11158 Solved By 621 Users

There are **N** stones arranged in a row. Each stone has a positive value, which indicates the maximum number of stones a person can cross in one jump from it. A person always starts from the first stone and he wants to reach the final or last stone. The program must accept N integers representing the N stones as the input. The program must print the minimum number of jumps that the person can reach to the last stone as the output.

Boundary Condition(s):
1 <= N <= 10^5
1 <= Each integer value <= 10^3

Input Format:
The first line contains N.
The second line contains N integers separated by a space.

Output Format:
The first line contains the minimum number of jumps that the person can reach to the last stone.

Example Input/Output 1:
Input:
5
2 3 1 1 4

Output:
2

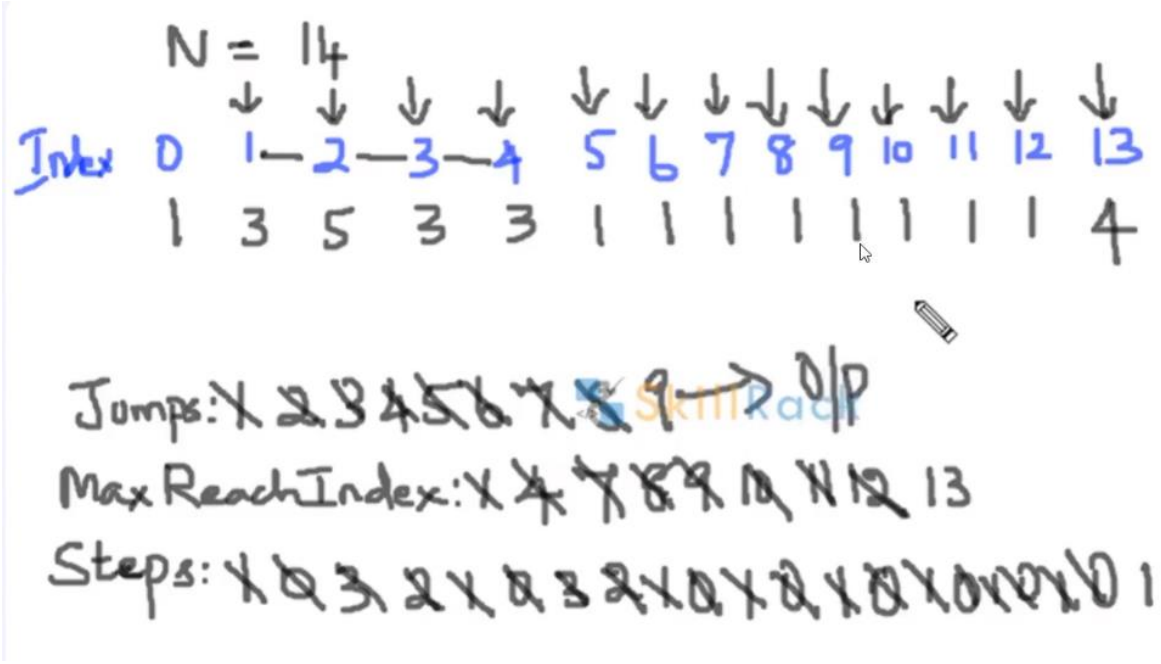
Explanation:
Here the minimum number of jumps that the person can reach to the last stone is **2**.
In the 1st jump, he can jump from 2 to 3 (i.e., from the 1st stone to the 2nd stone).
In the 2nd jump, he can jump from 3 to 4 (i.e., from the 2nd stone to the 5th stone).

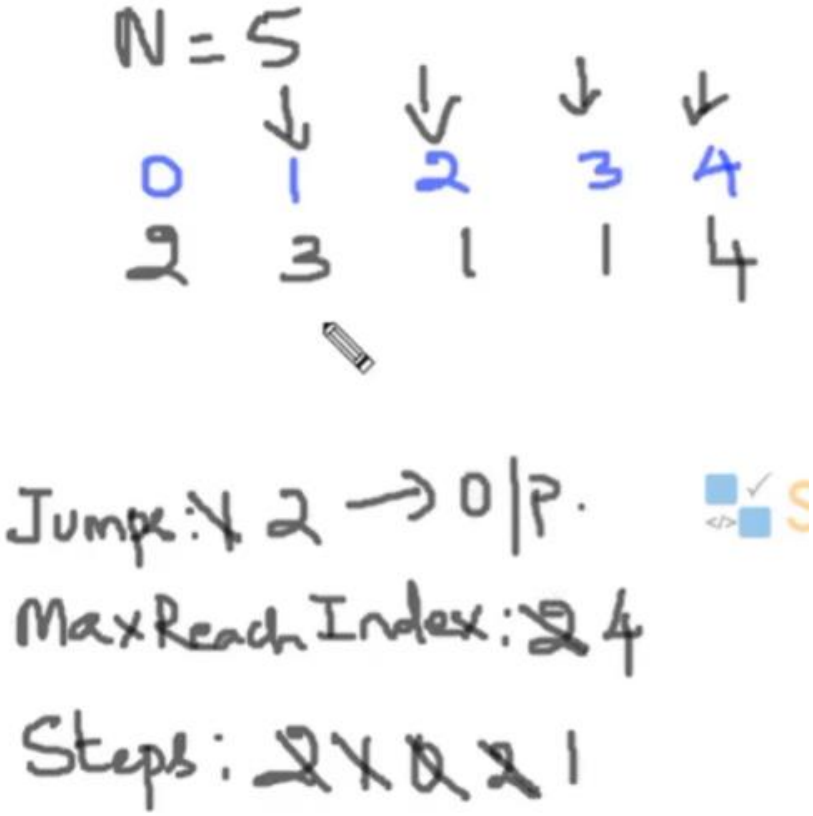
Example Input/Output 2:
Input:
14
1 3 5 3 3 1 1 1 1 1 1 1 1 4

Output:
9

Max Execution Time Limit: 500 millisecs

Logic:


N = 14
Index 0 1 2 3 4 5 6 7 8 9 10 11 12 13
1 3 5 3 3 1 1 1 1 1 1 1 1 4
Jumps: X 2 3 4 5 6 X X 9 → O/P
MaxReachIndex: X 4 X 8 9 10 X 12 13
Steps: X 1 3 2 X 1 3 2 X 1 X 1 X 1 X 1 X 1 0


N = 5
0 1 2 3 4
2 3 1 1 4
Jump: X 2 → O/P.
MaxReachIndex: 2 4
Steps: 2 X 1 2 1

Code:

```java
import java.util.*;

public class minimumJumpsToReachEnd {

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int N = s.nextInt();
        if (N == 1) { //if there is only one stone no jump is required
            System.out.println(0);
            return;
        }
        int arr[] = new int[N];
        for (int index = 0; index < N; index++) {
            arr[index] = s.nextInt();
        }
        int jumps = 1;
        int maxReachIndex = arr[0];
        int steps = arr[0];

        for (int index = 1; index < N; index++) { //refer logic photo
            if (index == N - 1) {
                break;
            }
            if (index + arr[index] > maxReachIndex) {
                maxReachIndex = index + arr[index];
            }
            steps--;
            if (steps == 0) {
                jumps++;
                steps = maxReachIndex - index;
            }
        }
        System.out.println(jumps);
    }
}
```

Session 3:

Browsing Center Computers

ID:11159 Solved By 586 Users

In a browsing center, the owner accepts the next day's browsing slot booking from **N** customers via internet. Each browsing slot contains the **start time** and the **end time** in 24-hr format. The browsing center owner must have at least **M** computers so that no one is waiting. The browsing solt of N customers are passed as the input. The program must print the minimum number of computers **M** so that no one is waiting in the center.

Boundary Condition(s):
1 <= N <= 10^5

Input Format:
The first line contains N.
The next N lines, each containing the start time and the end time of a browsing slot.

Output Format:
The first line contains M.

Example Input/Output 1:
Input:
6
9:00 11:00
9:30 10:30
9:30 12:00
9:45 13:00
11:00 15:00
10:00 14:00

Output:
5

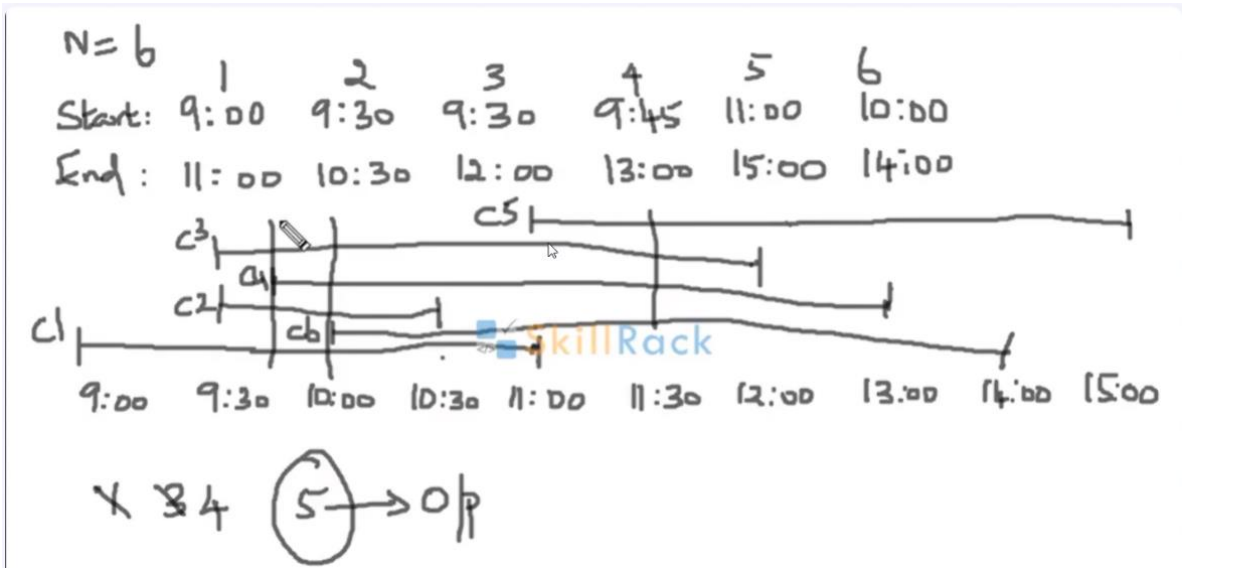
Explanation:
The browsing center owner must have at least **5** computers so that no one is waiting.
At **10:00**, there must be at least **5** computers.

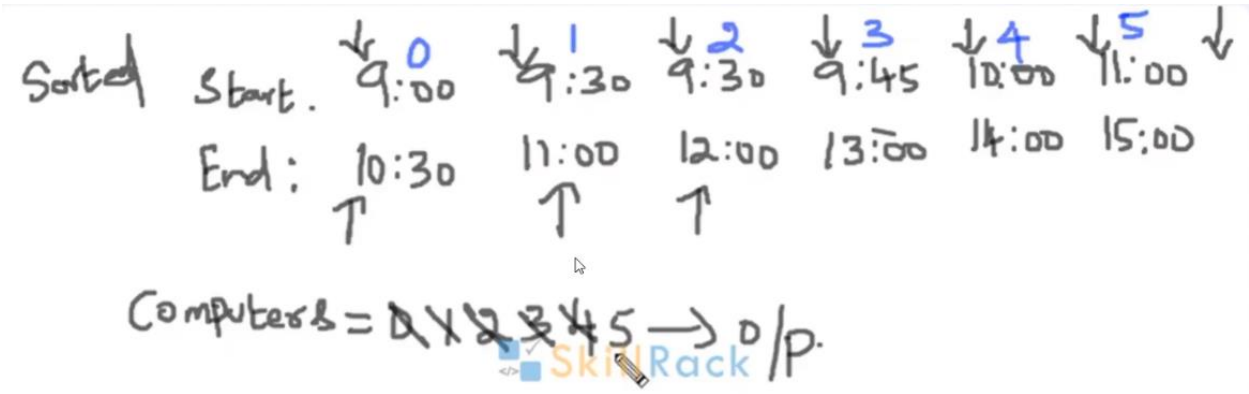
Example Input/Output 2:
Input:
6
9:00 15:00
10:00 10:30
11:00 13:00
13:00 14:00
14:30 15:00
14:30 16:00

Output:
3

Max Execution Time Limit: 100 millisecs

Logic:





Code:

```
import java.util.*;

public class browserCenterComputer {

    public static void main(String[] args) {
        // Your Code Here
        Scanner in = new Scanner(System.in);
        int N = in.nextInt();
        in.nextLine(); //for bypassing
        List<Integer> start = new ArrayList<>();
        List<Integer> end = new ArrayList<>();
        for (int counter = 0; counter < N; counter++) {
            String slot[] = in.nextLine().split("\\s+"); //splitting using spaces
            start.add(toMins(slot[0]));
            end.add(toMins(slot[1]));
```

```
        }
        int computerneeded = 0, startIndex = 0, endIndex = 0;
        Collections.sort(start);
        Collections.sort(end);
        while (startIndex < start.size() && endIndex < end.size()) {
            int currReq = Math.abs(startIndex - endIndex) + 1;
            if (currReq > computerneeded) {
                computerneeded = currReq;
            }
            if (start.get(startIndex) < end.get(endIndex)) {
                startIndex++;
            } else {
                endIndex++;
            }
            while (startIndex < start.size() && endIndex < end.size() &&
start.get(startIndex) >= end.get(endIndex)) {
                endIndex++;
            }
        }
        System.out.println(computerneeded);
    }

    private static int toMins(String time) { //coverting the given time to mins
//General Info for any sum - always covert the time to minutes or seconds for
easy calculation
        String hourMins[] = time.split(":");
        int hours = Integer.parseInt(hourMins[0]);
        int minutes = Integer.parseInt(hourMins[1]);
        return (hours * 60) + minutes;
    }
}
```