DAY11:

SESSION 1:

Problem 1:

Selling Wine Bottles

ID:11129 Solved By 734 Users

There are **N** wine bottles packed and arranged in a row from left to right. The wine bottles can be sold only one per year with a condition that only the leftmost or the rightmost wine bottle can be sold. The price of a wine bottle **P(i)** where 1 <= i <= N (which has not been sold yet) increases by it's initial price P(i) every year. Find the maximum revenue that can be obtained by selling the wine bottles based on the above conditions.

Boundary Condition(s):
1 <= N <= 1000
1 <= P(i) <= 100

Input Format:
The first line contains N.
The second line contains N integers representing the price of the wine bottles separated by a space.

Output Format:
The first line contains the maximum revenue that can be obtained by selling the wine bottles based on the above conditions.

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

Output:
29

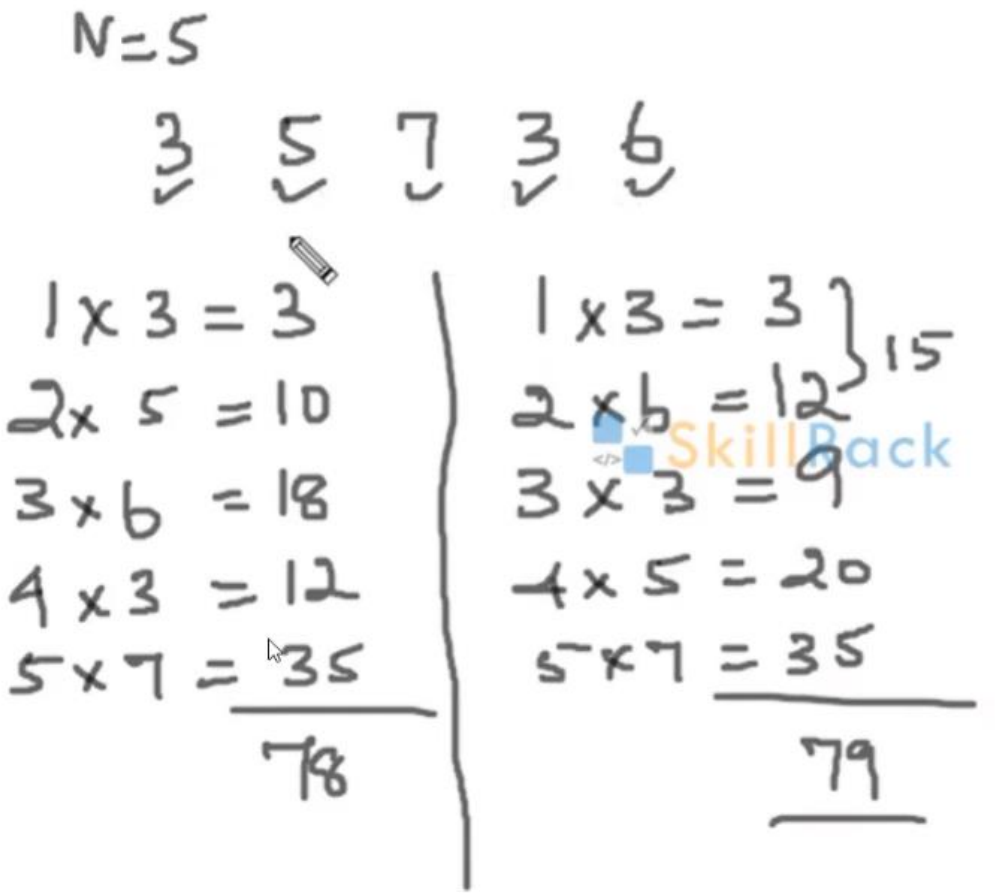
Explanation:
Max revenue = **1***1 + **3***2 + **2***3 + **4***4 = 29

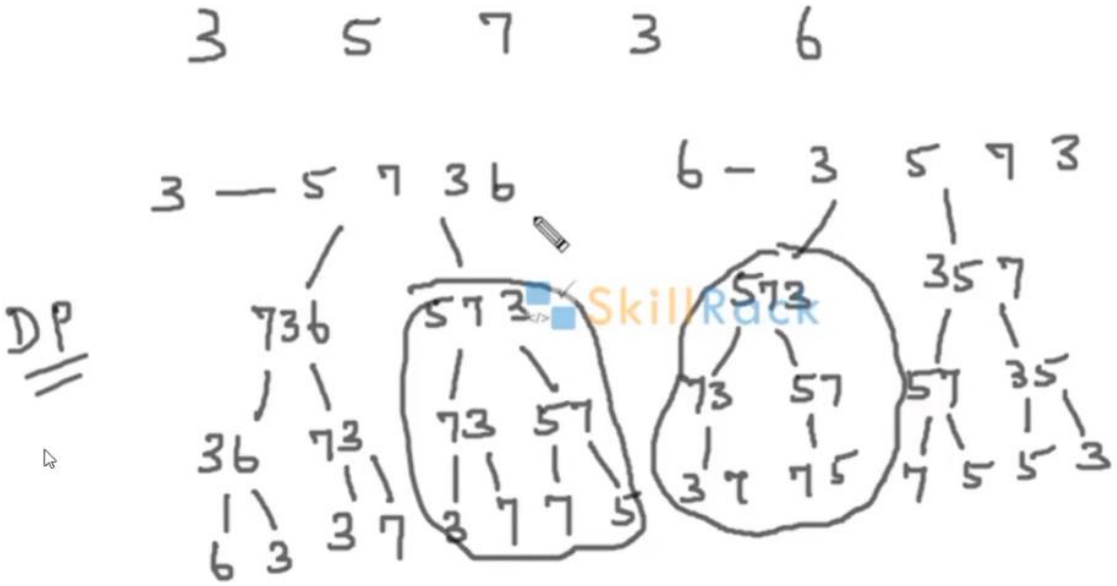
Example Input/Output 2:
Input:
5
3 5 7 3 6

Output:
79

Max Execution Time Limit: 500 millisecs

Logic:


N=5
3 5 7 3 6
1x3=3
2x5=10
3x6=18
4x3=12
5x7=35
78
1x3=3
2x6=12
3x3=9
4x5=20
5x7=35
79
15


3 5 7 3 6
DP

Code:

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

    public static void main(String[] args) {
        //Your Code Here
        Scanner in = new Scanner(System.in);
        int N=in.nextInt();
        int prices[]=new int[N];
        for(int index=0;index<N;index++){
            prices[index]=in.nextInt();
        }
        int max[][] =new int[N][N]; //memosation array
        System.out.println(maxRevenue(prices,max,0,N-1,1));
    }

    private static int maxRevenue(int[] prices,int[][] max,int left,int right,int
year){
        if(max[left][right]!=0){
            return max[left][right];
        }
        if(left==right){  //base condition for recursion
            return prices[left]*year; //left can also be coded as right since
when one element is present in the array left==right
        }

        int
leftRevenue=prices[left]*year+maxRevenue(prices,max,left+1,right,year+1);
//left+1 since already calculated left element

        int rightRevenue=prices[right]*year+maxRevenue(prices,max,left,right-
1,year+1); //right-1 since already calculated right element

        max[left][right]=Math.max(leftRevenue,rightRevenue);
        return max[left][right];
    }
}
```

Session 2:

Problem 1:

Social Media Leader

ID:11130 Solved By 760 Users

There are **N** people in a Facebook group. The group follows the following three rules.
1) The group leader does not follow anyone.
2) Everyone in the group follows the group leader.
3) No more than one group leader is allowed.
The N people are numbered from 1 to N. The program must accept the integer **N** and **R** relationships. Each relationship contains two integers representing who follows whom. The program must print an integer denoting the group leader. If the group leader can not be found, the program must print **-1** as the output.

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

Input Format:
The first line contains N and R separated by a space.
The next R lines, each containing 2 integers separated by a space.

Output Format:
The first line contains an integer denoting the group leader or -1.

Example Input/Output 1:
Input:
5 7
1 3
2 3
3 4
5 3
1 4
5 4
2 4

Output:
4

Explanation:
Here the group leader is **4** as everyone follows 4 in the group and the 4 has not followed anyone.

Example Input/Output 2:
Input:
5 9
1 2
2 4
3 4
1 5
1 4
3 2
5 4
2 5
4 2

Output:
-1

Max Execution Time Limit: 100 millisecs

Code:

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

    public static void main(String[] args) {
        //Your Code Here
        Scanner in = new Scanner(System.in);
        int N=in.nextInt();
        int R=in.nextInt();
        int arr[] = new int[N+1];
```

```
        int follower,personFollowed;
        for(int ctr=0;ctr<R;ctr++){
            follower=in.nextInt();
            personFollowed=in.nextInt();
            arr[follower]--;        //decrementing the follower count by 1
            arr[personFollowed]++;  //increasind the person followed count by 1
        }
        // int leader=-1; alternate
        for(int index=1;index<=N;index++){
            if(arr[index] == N-1){  //checking if the personFollowed count is
equal to 4 when N is 5
                System.out.println(index);
                // leader=index;
                return;
            }
        }
        System.out.println("-1");
    }
}
```

SESSION 3:

Problem1:

Intelligent Chef

ID:11131 Solved By 617 Users

There are **N** persons in a hotel. Each person has their own preferences for food. The hotel chef wants to prepare the food items as minimum as possible.The program must accept the food preferences of each person as the input. The program must print the minimum number of food items that must be prepared to serve everyone in the hotel.
Note: Each person eats only one food item but has many options.

Boundary Condition(s):
1 <= N <= 10^4
1 <= Length of the name of each food item <= 50

Input Format:
The first line contains N.
The next N lines, each containing the string value(s) representing the preferences of food items of a person.

Output Format:
The first line contains the minimum number of food items that must be prepared to serve everyone in the hotel.

Example Input/Output 1:
Input:
5
Dosa Poori
Idli Poori
Idli Poori
Idli Dosa
Poori

Output:
2

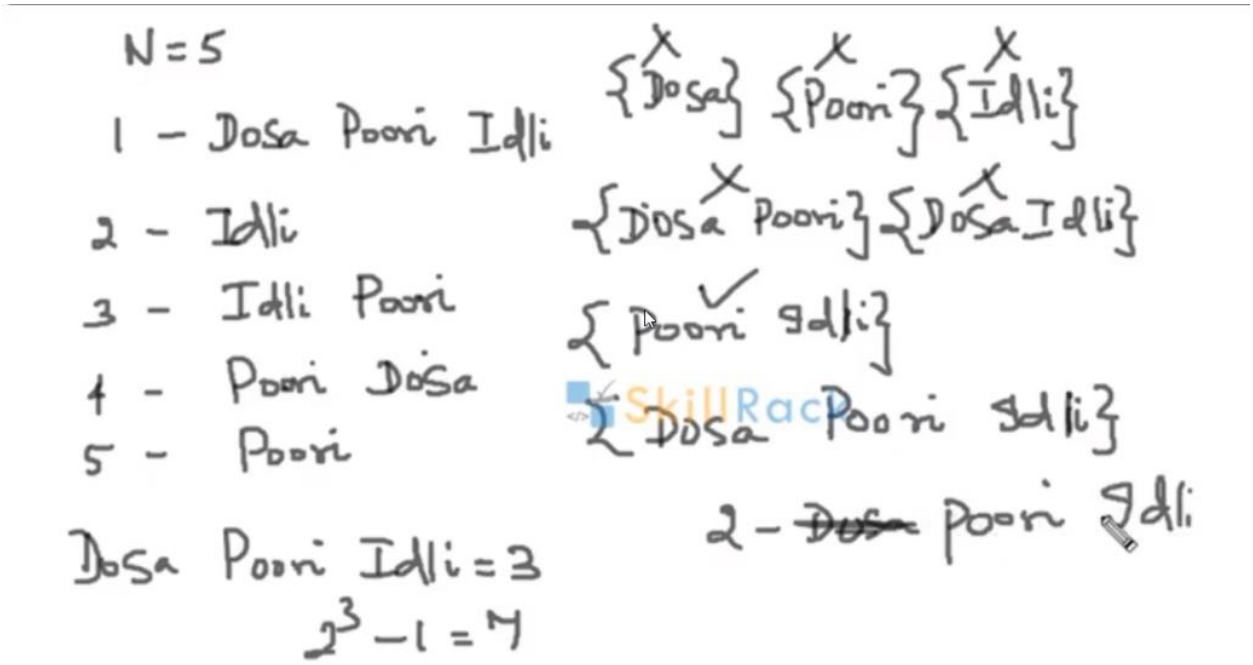
Explanation:
Here N = 5 representing the 5 persons in the hotel.
At least **2** food items (**Idli** & **Poori**) must be prepared to serve everyone in the hotel.

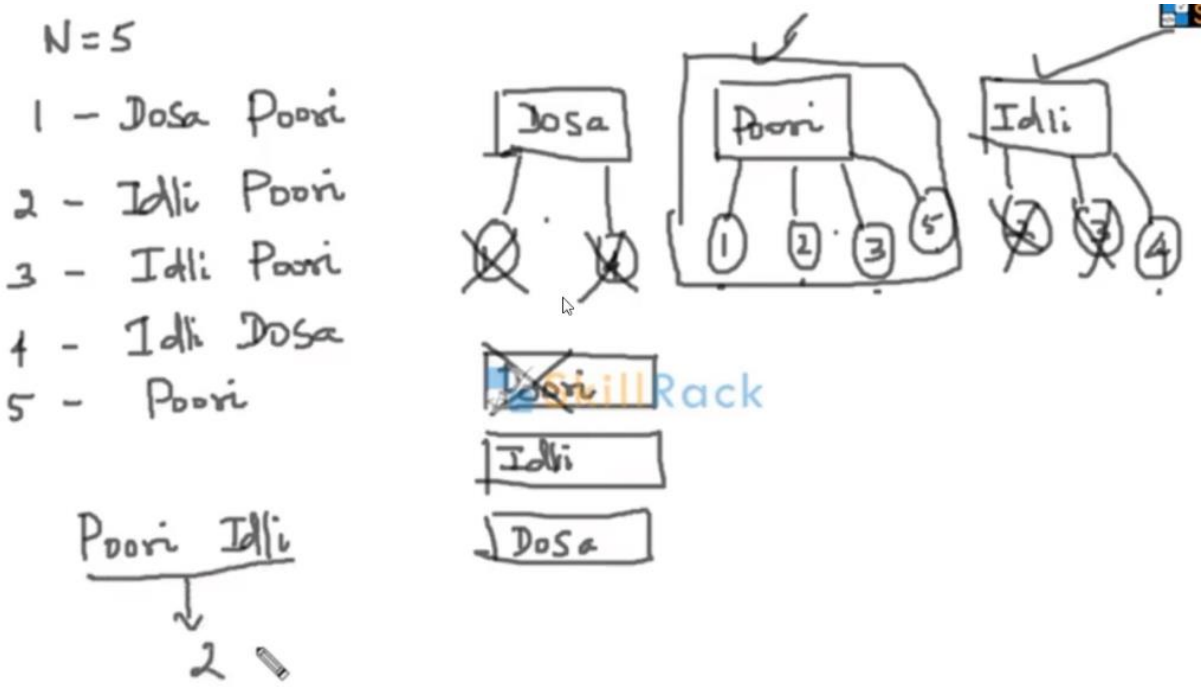
Example Input/Output 2:
Input:
10
Chapati Idli Pongal Poori Dosa
Poori Dosa Chapati
Idli Dosa Poori
Pongal
Dosa Pongal Poori Chapati
Idli Pongal Poori
Idli Pongal Chapati Dosa
Dosa Chapati
Chapati Idli Pongal Poori Dosa
Chapati

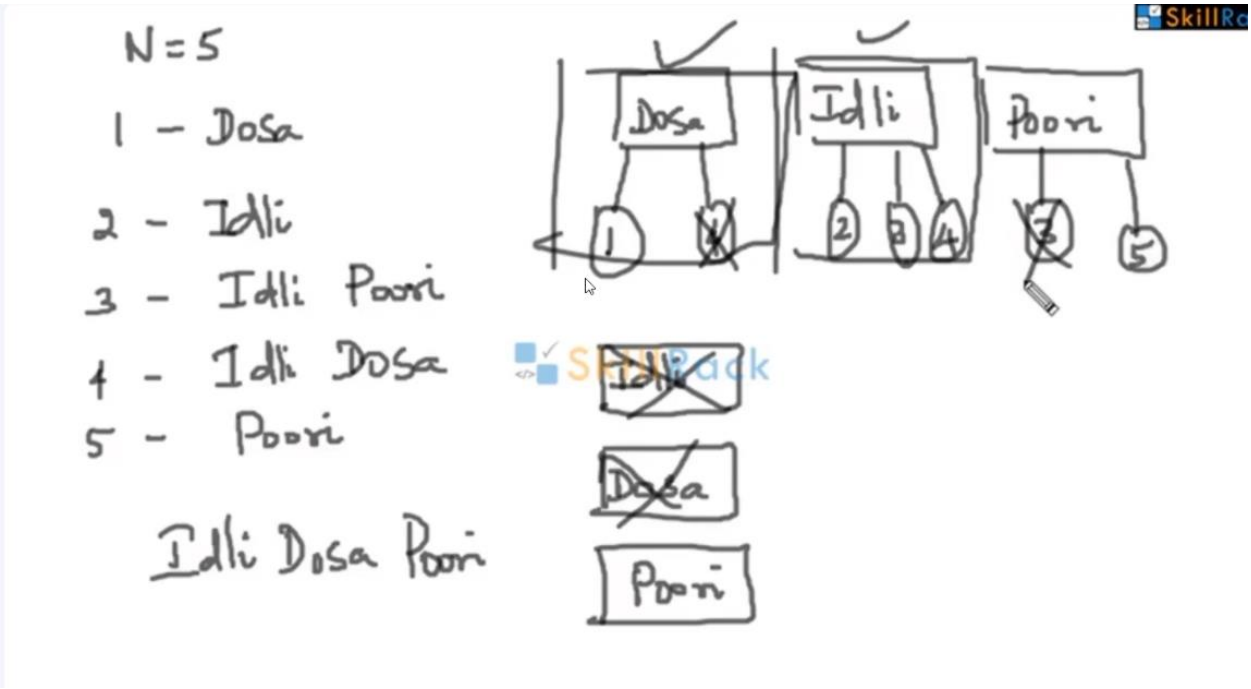
Output:
3

Max Execution Time Limit: 2000 millisecs

Logic:


N=5
1 - Dosa Poori Idli
2 - Idli
3 - Idli Poori
4 - Poori Dosa
5 - Poori
Dosa Poori Idli = 3
$2^3 - 1 = 7$
{Dosa} {Poori} {Idli}
{Dosa Poori} {Dosa Idli}
{Poori Idli}
{Dosa Poori Idli}
2 - Dosa poori Idli


N=5
1 - Dosa Poori
2 - Idli Poori
3 - Idli Poori
4 - Idli Dosa
5 - Poori
Poori Idli
2
Dosa
Poori
Idli
1
2
3
5
1
3
4
Poori
Idli
Dosa



Code:

```
import java.util.*;
class FoodItem implements Comparable<FoodItem>{
    String name;
    List<Integer> customers;

    @Override
    public int compareTo(FoodItem other){
        return this.customers.size() - other.customers.size();
    }
}
public class IntelligentChef {

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int N = s.nextInt();
        s.nextLine(); //moving to the nextline as the next input is given in the
next line
        Map<String,FoodItem> foodItemMap = new HashMap<>(); //fooditem is a class
above
        List<Integer> remainingCustomers = new ArrayList<>();
        for(int customer = 1; customer <= N; customer++){
            remainingCustomers.add(customer);
            String preferences[] = s.nextLine().split("\\s+");//splitting using
spaces
```

```
            for(String item:preferences){
                if(!foodItemMap.containsKey(item)){
                    FoodItem fi = new FoodItem();
                    fi.name = item;
                    fi.customers = new ArrayList<>();
                    foodItemMap.put(item, fi);
                }
                foodItemMap.get(item).customers.add(customer);
            }
        }
        int count = 0;
        while(!remainingCustomers.isEmpty()){
            List<FoodItem> items = new ArrayList<>(foodItemMap.values());
            Collections.sort(items, Collections.reverseOrder()); //sorting in
descending order as the food item having the highest no of customers preference
will be on the top
            FoodItem mostPreferred = items.get(0);
            count++;
            foodItemMap.remove(mostPreferred.name);
            remainingCustomers.removeAll(mostPreferred.customers);
            for(String foodItem:foodItemMap.keySet()){
                foodItemMap.get(foodItem).customers.removeAll(mostPreferred.custo
mers);
            }
        }
        System.out.println(count);
    }
}
```