iNautix Test – 11

1) Hackland Election

There are n citizens voting in this year's HackLand election. Each voter writes the name of their chosen candidate on a ballot and places it in a ballot box. The candidate with the highest number of votes wins the election; if two or more candidates have the same number of votes, then the tied candidates' names are ordered alphabetically and the last name wins.

Complete the electionWinner function in your editor. It has 1 parameter: an array of strings, votes, describing the votes in the ballot box. This function must review these votes and return a string representing the name of the winning candidate.

Input Format

The locked stub code in your editor reads the following input from stdin and passes it to your function:

The first line contains an integer, n, denoting the size of the votes array.

Each line i of the n subsequent lines (where 0 ≤ i < n) of strings contains a citizen's vote in the form of a candidate's name.

Constraints

• 1 ≤ n ≤ 104

Output Format

Your function must return a string denoting the name of the winner. This is printed to stdout by the locked stub code in your editor.

Sample Input 1

10

Alex

Michael

Harry

Dave

Michael

Victor

Harry

Alex

Mary

Mary

Sample Output 1

Michael

Explanation 1

votes = {"Alex", "Michael", "Harry", "Dave", "Michael", "Victor", "Harry", "Alex", "Mary", "Mary"}

Alex, Harry, Michael, and Mary are all tied for the highest number of votes. Because Michael is alphabetically last, we return his name as the winner.

Sample Input 2

10

Victor

Veronica

Ryan

Dave

Maria

Maria

Farah

Farah

Ryan

Veronica

Sample Output 2

Veronica

Explanation 2

votes = {"Victor", "Veronica", "Ryan", "Dave", "Maria", "Maria", "Farah", "Farah", "Ryan", "Veronica"}

Veronica, Ryan, Maria, and Farah are all tied for the highest number of votes. Because Veronica is alphabetically last, we return her name as the winner.

Solution:

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

char\* fun(char \*\*);

int main()

{

int num,i;

char \*\*a;

scanf("%d",&num);

// allocating memory dynamically for 2D array

a=(char \*\*) malloc (num \* sizeof(char \*));

for(i=0;i<num;i++)

{

a[i]=(char \*) malloc (100 \* sizeof(char));

scanf("%s",a[i]);

}

// ans string to catch the winner

char \*ans;

ans=fun(a);

printf("ans is %s",ans);

return 0;

}

char\* fun(char \*\*a)

{

char \*winner= (char \*) malloc (100 \* sizeof(char));

char str1[100],str2[100];

int i,num=0,j;

// since the number of votes are not passed i'm calculating explicitly

for(i=0;a[i]!=0;i++)

{

num++;

}

// array to store number of votes for each candidate

int \*vote\_array = (int \*) malloc (num\* sizeof(int));

//initializing the array to 1

for(i=0;i<num;i++)

{

vote\_array[i]=1;

}

for(i=0;i<num;i++)

{

for(j=i+1;j<num;j++)

{

if(a[i]!='\0' && a[j]!='\0')

{

//comparing first candidate name with all other names, if matches increment the vote for that candidat and make the repeating name= null

if(strcmp(a[i],a[j])==0)

{

vote\_array[i]++;

a[j]='\0';

}

}

}

}

// sorting and finding maximum number of votes out of all candi's

int max=vote\_array[0];

for(i=1;i<num;i++)

{

if(vote\_array[i]>max)

{

max=vote\_array[i];

}

}

// if many got same max votes store them in separate array finalists

char \*\*finalists = (char \*\*) malloc ( num \* sizeof(char \*));

int k=0;

for(i=0;i<num;i++)

{

if(vote\_array[i]==max)

{

finalists[k]=(char \*) malloc (100 \* sizeof(char));

strcpy(finalists[k],a[i]);

k++;

}

}

char temp[100];

// again sorting the finalists to get the winner (lexicographically largest)

for(i=0;i<k;i++)

{

for(j=i+1;j<k;j++)

{

if(strcmp(finalists[j],finalists[i])>0)

{

strcpy(temp,finalists[i]);

strcpy(finalists[i],finalists[j]);

strcpy(finalists[j],temp);

}

}

}

strcpy(winner,finalists[0]);

return winner;

}

2) Sherlock and GCD

Sherlock is stuck. He has an array A={a1, a2, ..., aN}. He wants to know if there exists a subset, B = {ai1, ai2, … , aik} where 1 ≤ i1 < i2 < … < ik ≤ N, of this array which has the following properties

• B is non-empty.

• All elements of B are relatively prime, i.e. there exists no integer x (x > 1) that evenly divides all elements of B.

Note that x may or may not be an element of A.

Constraints

1 ≤ T ≤ 10

1 ≤ N ≤ 100

1 ≤ ai ≤ 105

1 ≤ i ≤ N

Input Format

The first line contains a single integer T, the number of test cases.

Each test case consists of two lines: a single integer N on the first line representing the size of the array.In the next line there are N space separated integers a1, a2, ..., aN representing the elements of the array A.Na1,a

Output Format

Print YES if there exists any such subset, and or NO, if not.1≤i≤

5 ∀1≤i≤N

Sample input #00

2

3

1 2 3

2

2 4

Sample output #00

YES

NO

Explanation #00

In first test case, {1},{1,2}, {1,3}, {2,3} and {1,2,3} are the possible subsets where no integer greater than 1 divides the elements.

In second test case, no non-empty subset exists which satisfies the given condition.

Solution:

#include <stdio.h>

int gcd(int x,int y) {

if(y==0)

return x;

return gcd(y,x%y);

}

int main() {

int test;

scanf("%d",&test);

while(test--)

{

int range,i,num,g=0;

scanf("%d",&range);

for(i=0;i<range;i++)

{

scanf("%d",&num);

g = gcd(g,num);

}

if(g==1)

printf("YES\n");

else

printf("NO\n");

}

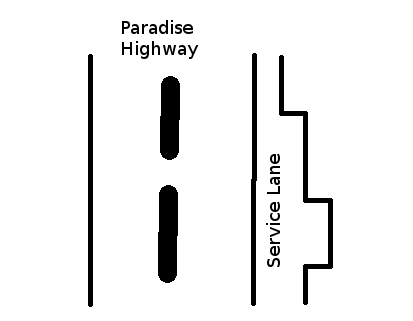
return 0;

}

1. Service Lane

Hobbes is driving his favorite vehicle on Paradise Highway. He notices that the "Check Engine" light of his vehicle is on, and he wants to service it immediately to avoid any risks. Luckily, a service lane runs parallel to the highway. The length of the highway (and the service lane) is ***N*** units. The service lane consists of ***N*** segments of unit length, and each segment can have a different width denoted by *1*, *2* or *3*. An array ***width****[]* of length ***N*** gives the widths of the different lanes, where ***width****[****k****]* represents the width of the ***k****th* segment of the service lane.

When Hobbes sees the "Check Engine" light, he is arriving at segment ***i***, and, the next exit that he needs to take to find a repair station is ***j***.  *Hobbes therefore must pass through all segments* ***i*** *to* ***j****, inclusive of both****i*** *and* ***j****.*



Hobbes has three types of vehicles - bike, car and truck, with widths *1*, *2* and *3* respectively. Given the entry and exit point of Hobbes' vehicle in the service lane, output the type of largest vehicle which can pass through the service lane (including the entry & exit segment). Assume that there is never more than *1000* segments on the path from *i* to *j* (both inclusive).

**Note**

1. If *width[k]* is 1, only the bike can pass through the *kth* segment.
2. If *width[k]* is 2, then both the bike and car can pass through the *kth* segment.
3. If *width[k]* is 3, then any of the bike, car or truck can pass through the ***k****th* segment.

**Input Format**  
The first line of input contains two integers - *N* and *T*, where *N* is the length of the freeway, and *T* is the number of test cases.

The next line has *N* space separated integers which represents the *width* array.

*T* test cases follow. Each test case contains two integers - *i* and *j*, where *i* is the index of segment through which Hobbes enters the service lane and *j* is the index of the lane segment where he exits.

**Output Format**  
For each test case, print (on a separate line) the number (*1*, *2* or *3*) that represents the largest vehicle type that can pass through the service lane.

**Note**  
*Hobbes has to pass through all segments from index i to index j (both inclusive).*

**Constraints**

* *1 ≤ T ≤1000*
* *2 ≤ N ≤ 105*
* *0 ≤ i < j < N*
* *j ≤****i*** *+ 999*
* *1 ≤ width[k] ≤ 3, where 0 <= k < N*

**Sample Input #00**

8 5

2 3 1 2 3 2 3 3

0 3

4 6

6 7

3 5

0 7

**Sample Output #00**

1

2

3

2

1

**Explanation #00**  
Below is the representation of lane.

**|HIGHWAY|Lane| -> Width**

**0: | |--| 2**

**1: | |---| 3**

**2: | |-| 1**

**3: | |--| 2**

**4: | |---| 3**

**5: | |--| 2**

**6: | |---| 3**

**7: | |---| 3**

1. (0, 3): Because *width[2]* = *1*, only the bike, represented as *1* can pass through this segment.
2. (4, 6): Here the largest vehicle that can pass through is the car, because  of the the 5th segment's width.
3. (6, 7): In this example, the vehicle enters at the 6th segment and exits at the 7th segment. Both segments allow even the truck to pass through, so *3* is the answer.
4. (3, 5): *width[3] = width[5] = 2*. While 4th segment allow the truck, the *3rd* and *5th* allow up to 2, the car width.
5. (0, 7): The bike is the only vehicle which can pass through the 2nd segment, which limits the capacity of the whole lane to 1.

Solution :

#include <stdio.h>

int main() {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/

long long int num,test;

long long int i,j,a[100000],m,n,min;

scanf("%d%d",&num,&test);

for(i=0;i<num;i++)

{

scanf("%d",&a[i]);

}

while(test--){

scanf("%d%d",&m,&n);

min = a[m];

for(i=m;i<=n;i++){ //finding the minimum of available vehicles (m to n)

if(a[i]<min){

min = a[i];

}

}

printf("%d\n",min);

}

return 0;

}