

Problem Description

Polycarp has an array consisting of n integers.

He wants to play a game with this array. The game consists of several moves. On the first move, he chooses any element and deletes it (after the first move the array contains $n-1$ elements). For each of the next moves he chooses any element with the only restriction: its parity should differ from the parity of the element deleted on the previous move. In other words, he alternates parities (even-odd-even-odd-... or odd-even-odd-even-...) of the removed elements. Polycarp stops if he can't make a move.

Formally:

1. If it is the first move, he chooses any element and deletes it;
2. If it is the second or any next move:
 - if the last deleted element was odd, Polycarp chooses any even element and deletes it;
 - if the last deleted element was even, Polycarp chooses an odd element and deletes it.
3. If after some move Polycarp cannot make a move, the game ends.

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int cmp(const void *a, const void *b) {
```

```
    return *(int*)a - *(int*)b;
```

```
}
```

```
int main() {
```

```
    int o[2000], ol = 0, e[2000], el = 0, n, t;
```

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

```
    while(n--) {
```

```
        scanf("%d", &t);
```

```
        if(t % 2)
```

```
            o[ol++] = t;
```

```
        else
```

```
            e[el++] = t;
```

```
    }
```

```
    qsort(o, ol, sizeof(int), cmp);
```

```
    qsort(e, el, sizeof(int), cmp);
```

```
    while(ol && el) {
```

```
        ol--;
```

```
        el--;
```

```
    }
```

```
    t = 0;
```

```
    if(ol) {
```

```
        ol--;
```

```
    while(ol)
```

```

t += o[--ol];

} else if(el) {

el--;

while(el)

t += e[--el];}

printf("%d", t);

return 0;}

```

Problem Description:

Manu's task is to write a registration system.

The system works in the following way. Every user has a preferred login li . The system finds the first free login considering possible logins in the following order: li , $li0$, $li1$, $li2$, ... , $li10$, $li11$, ... (you check li first; in case it is occupied already, Manu pick the smallest nonnegative integer x such that concatenation of li and decimal notation of x gives you free login) and register a user with this login in the system. After the registration, this login becomes occupied.

Manu gave the preferred logins for the n users in chronological order. For each user, you have to find a login which he will use in the system.

Constraints:

$1 \leq n \leq 2 \cdot 10^5$

Input Format

The first line of input contains a single integer n - a number of users.

Then follow n lines. The i -th of these lines contains li - a preferred login for i -th user. li is a nonempty string with lowercase English letters and digits.

```
#include<stdbool.h>
```

```
#include<malloc.h>
```

```
#include<string.h>
```

```
char str[1000005];
```

```
char temp[10];
```

```
struct trie
```

```
{

    struct trie* child[36];
```

```
    int value;
```

```
    bool set;
```

```
};
```

```
struct trie* newnode()
```

```
{

    int i;

    struct trie* node=(struct trie*)malloc(sizeof(struct trie));

    for(i=0;i<36;i++)

        node->child[i]=NULL;
```

```

node->value=-1;

node->set=false;

return node;

}

void lookup(struct trie * root,char *str)

{

    int i,len=strlen(str),flag,flag1;

    struct trie* head=root,*head2;

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

    {

        if((str[i]-'0')<10&&(str[i]-'0')>=0)

        {

            if(head->child[str[i]-'0']==NULL)

            {

                head->child[str[i]-'0']=newnode();

            }

            head=head->child[str[i]-'0'];

        }

        else

        {

            if(head->child[str[i]-'a'+10]==NULL)

            {

                head->child[str[i]-'a'+10]=newnode();

            }

            head=head->child[str[i]-'a'+10];

        }

    }

    flag=1;

    while(head->value>=0&&flag)

```

```

{
    flag=1;

    head2=head;

    snprintf(temp,2,"%d",head->value);

    for(i=0;i<strlen(temp);i++)
    {
        if(head2->child[temp[i]-'0']==NULL){

            head2->child[temp[i]-'0']=newnode();

            flag=0;

        }

        head2=head2->child[temp[i]-'0'];

    }

    if(flag&&head2->set==true)

        head->value++;

    else{

        head2->value++;

        flag=0;

    }

}

flag1=1;

if(flag==0){

    printf("%d",head->value);

    head2->set=true;

    flag1=0;

}

head->value++;

if(flag1)

    head->set=true;

    printf("\n");

}

int main()

```

```

{

    int test;

    struct trie *root=newnode();

    scanf("%d",&test);

    while(test--)

    {

        scanf("%s",str);

        printf("%s",str);

        lookup(root,str);

    }

    return 0;

}

```

Problem Description

There is a binary string a of length n . In one operation, you can select any prefix of a with an equal number of 0 and 1 symbols. Then all symbols in the prefix are inverted: each 0 becomes 1 and each 1 becomes 0.

For example, suppose $a=0111010000$.

In the first operation, we can select the prefix of length 8 since it has four 0's and four 1's: $[01110100]00 \rightarrow [10001011]00$.

In the second operation, we can select the prefix of length 2 since it has one 0 and one 1: $[10]00101100 \rightarrow [01]00101100$.

It is illegal to select the prefix of length 4 for the third operation, because it has three 0's and one 1.

Can you transform the string a into the string b using some finite number of operations (possibly, none)?

Constraints

```

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main() {

    int n_cases, n, balance, diff;

    char s1[300001], s2[300001], *c1, *c2;

    bool any_same, any_different;

    scanf("%d", &n_cases);

    while (n_cases--) {

        scanf("%d", &n);

        scanf("%s\n%s", s1, s2);

        c1 = s1;

```

```

c2 = s2;

any_same = false;

any_different = false;

balance = 0;

diff = 0;

while (*c1) {

any_same = any_same || *c1 == *c2;

any_different = any_different || *c1 != *c2;

if (any_same && any_different) break;

balance += *c2 == '1' ? 1 : -1;

diff += *c1 - *c2;

if (balance == 0) {

any_same = false;

any_different = false;

}

c1++; c2++;

}

printf(((any_same && any_different) || diff != 0) ? "NO\n" : "YES\n");}

return 0;}

```

Problem Description:

Mr. Kamal has a teacher at CBSE School. There are n students under Kamal supervision, the programming skill of the i -th student is a_i .

Kamal to create a team for a new programming competition. As he know, the more students some team has the more probable its victory is! So he has to create a team with the maximum number of students. But you also know that a team should be balanced. It means that the programming skill of each pair of students in a created team should differ by no more than 5.

Your task is to report the maximum possible number of students in a *balanced* team.

Constraints:

$$1 \leq n \leq 2 \cdot 10^5$$

$$1 \leq a_i \leq 10^9$$

Input Format:

The first line of the input contains one integer n — the number of students.

The second line of the input contains n integers a_1, a_2, \dots, a_n , where a_i is a programming skill of the i -th student.

Output Format:

Print the output in a single line contains the maximum possible number of students in a balanced team.

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define N      200000
```

```

int rand_(int n) {

    return (rand() * 45677LL + rand()) % n;

}


int compare(const void *a, const void *b) {

    int ia = *(int *) a;

    int ib = *(int *) b;


    return ia - ib;

}


int main() {

    static int aa[N];

    int n, i, j, tmp, max;


    scanf("%d", &n);

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

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

    for (j = n - 1; j >= 0; j--) {

        i = rand_(j + 1);

        tmp = aa[i], aa[i] = aa[j], aa[j] = tmp;

    }

    qsort(aa, n, sizeof *aa, compare);

    max = 0;

    for (i = 0, j = 0; j < n; j++) {

        while (aa[i] + 5 < aa[j])

            i++;

        if (max < j - i + 1)

            max = j - i + 1;

    }

```

```

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

return 0;

}

```

Problem Description:

An agent called Cypher is decrypting a message, that contains a composite number n . All divisors of n , which are greater than 1, are placed in a circle. Cypher can choose the initial order of numbers in the circle.

In one move Cypher can choose two adjacent numbers in a circle and insert their least common multiple between them. He can do that move as many times as needed.

A message is decrypted, if every two adjacent numbers are not coprime. Note that for such constraints it's always possible to decrypt the message.

Find the minimal number of moves that Cypher should do to decrypt the message, and show the initial order of numbers in the circle for that.

Constraints

$1 \leq t \leq 100$
 $4 \leq n \leq 10^9$

Input Format:

The first line contains an integer t — the number of test cases. The next t lines describe each test case.

In a single line of each test case description, there is a single composite number n — the number from the message.

It's guaranteed that the total number of divisors of n for all test cases does not exceed $2 \cdot 10^5$.

```

#include <stdio.h>

#include <string.h>

#define K 200000

int main() {

int t;

scanf("%d", &t);

while (t--) {

static int pp[K], dd[K];

static char used[K];

int n, n_, kp, kd, p, d, g, h;

scanf("%d", &n);

n_ = n;

kp = 0;

for (p = 2; p <= n / p; p++)

if (n % p == 0) {

while (n % p == 0)

n /= p;

pp[kp++] = p;

}

```



```

if (n > 1)

    pp[kp++] = n;

    n = n_;

    kd = 0;

for (d = 2; d <= n / d; d++)

    if (n % d == 0) {

        dd[kd++] = d;

        if (d != n / d)

            dd[kd++] = n / d;

    }

if (kp == 2 && pp[0] * pp[1] == n) {

    printf("%d %d %d\n", pp[0], pp[1], n);

    printf("1\n");

    continue;

}

memset(used, 0, kd * sizeof *used);

for (g = 0; g + 1 < kp; g++) {

    int d = pp[g] * pp[g + 1];

    for (h = 0; h < kd; h++)

        if (dd[h] == d) {

            used[h] = 1;

            break;

        }

}

for (g = 0; g < kp; g++) {

    p = pp[g];

    for (h = 0; h < kd; h++)

        if (!used[h] && dd[h] % p == 0) printf("%d ", dd[h]), used[h] = 1;

    if (g + 1 < kp) printf("%d ", pp[g] * pp[g + 1]); }

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

printf("0\n"); } return 0;}

```

Problem Description

Recently, Bharani met with Dharani in a philatelic store, and since then they are collecting coins together. Their favorite occupation is to sort collections of coins. Dharani likes having things in order, that is why he wants his coins to be arranged in a row in such a way that firstly come coins out of circulation, and then come coins still in circulation.

For arranging coins Bharani uses the following algorithm. One step of his algorithm looks like the following:

He looks through all the coins from left to right;

If he sees that the i -th coin is still in circulation, and $(i + 1)$ -th coin is already out of circulation, he exchanges these two coins and continues watching coins from $(i + 1)$ -th.

Bharani repeats the procedure above until it happens that no two coins were exchanged during this procedure. Bharani calls hardness of ordering the number of steps required for him according to the algorithm above to sort the sequence, e.g. the number of times he looks through the coins from the very beginning. For example, for the ordered sequence hardness of ordering equals one.

Today Dharani invited Bharani and proposed him a game. First he puts n coins in a row, all of them are out of circulation. Then Dharani chooses one of the coins out of circulation and replaces it with a coin in circulation for n times. During this process Dharani constantly asks Bharani what is the hardness of ordering of the sequence.

The task is more complicated because Bharani should not touch the coins and he should determine hardness of ordering in his mind. Help Bharani with this task.

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int main()
```

```
{
```

```
int n,*q,x;
```

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

```
q=(int*)calloc(n+1, sizeof(int));
```

```
printf("1 ");
```

```
int p=n,i;
```

```
for( i=1;i<=n;i++)
```

```
{
```

```
scanf("%d",&x);
```

```
q[x]=1;
```

```
while(q[p]==1)
```

```
p--;
```

```
printf("%d ",i-n+p+1);}return 0;}
```

Problem Description:

Vijay has given a set of points x_1, x_2, \dots, x_n on the number line.

Two points i and j can be matched with each other if the following conditions hold:

neither i nor j is matched with any other point;

$|x_i - x_j| \geq z$.

What is the maximum number of pairs of points you can match with each other?

Constraints:

$2 \leq n \leq 2 \cdot 10^5$,

$1 \leq z \leq 10^9$

$1 \leq x_i \leq 10^9$

Input Format:

The first line contains two integers n and z — the number of points and the constraint on the distance between matched points, respectively.

The second line contains n integers x_1, x_2, \dots, x_n .

```

#include<stdio.h>

#include<stdlib.h>

void i(){}

int comp(const void*a,const void*b)

{

return *(int *)a - *(int *)b;

if(0)printf("static int aa[N];*aa");

}

int main()

{

int n, z, a[200009], i, sum=0;

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

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

scanf("%d", a+i);

qsort(a, n, sizeof(int), comp);

int l = 0, r = n&1 ? (n>>1)+1 : n>>1;

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

while(r < n)

{

if(a[r]-a[l] >= z)

sum++, l++;

r++;

}

printf("%d", sum); return 0; }

```

Problem Description

Mithran has an array of lengths n . He has just enough free time to make a new array consisting of n copies of the old array, written back-to-back. What will be the length of the new array's longest increasing subsequence?

A sequence a is a subsequence of an array b if a can be obtained from b by deletion of several (possibly, zero or all) elements. The longest increasing subsequence of an array is the longest subsequence such that its elements are ordered in strictly increasing order.

Constraints

$1 \leq n \leq 10^5$
 $1 \leq a_i \leq 10^9$

Input Format

The first line contains an integer t — the number of test cases you need to solve. The description of the test cases follows.

The first line of each test case contains an integer n ($1 \leq n \leq 10^5$) — the number of elements in the array a .

The second line contains n space-separated integers a_1, a_2, \dots, a_n — the elements of the array a .

The sum of n across the test cases doesn't exceed 10^5 .

```
#include <stdio.h>

#include <stdlib.h>

void harsh(){}

int main()

{

    int i,j,n,t;

    scanf("%d",&t);

    while(t--)

    {

        int H[100]={0},*a,count=0;

        scanf("%d",&n);

        a=(int*)malloc(sizeof(int)*n);

        for(j=0;j<n;j++)

        {

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

            H[a[j]]=1;

        }

        for(i=0;i<100;i++)

            if(H[i]==1) count++;

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

    }

    return 0;

}
```

Problem Description

Monkey B., the young coach of Ninjas, has found the big house which consists of n flats ordered in a row from left to right. It is possible to enter each flat from the street. It is possible to go out from each flat. Also, each flat is connected with the flat to the left and the flat to the right. Flat number 1 is only connected with the flat number 2 and the flat number n is only connected with the flat number $n-1$.

There is exactly one Ninja of some type in each of these flats. Monkey B. asked residents of the house to let him enter their flats in order to catch Ninjas. After consulting the residents of the house decided to let Monkey B. enter one flat from the street, visit several flats and then go out from some flat. But they won't let him visit the same flat more than once.

Monkey B. was very pleased, and now he wants to visit as few flats as possible in order to collect Ninjas of all types that appear in this house. Your task is to help him and determine this minimum number of flats he has to visit.

Constraints:
 $1 \leq n \leq 100\,000$

Input Format

The first line contains the integer n — the number of flats in the house.
The second line contains the row s with the length n , it consists of uppercase and lowercase letters of English alphabet, the i -th letter equals the type of Ninja, which is in the flat number i .

Output Format

Print the minimum number of flats which Monkey B. should visit in order to catch Ninjas of all types which there are in the house.

Logical Test Cases

Test Case 1	Test Case 2
INPUT (STDIN)	INPUT (STDIN)
6 aaBCCe	7 bcAAcBc
EXPECTED OUTPUT	EXPECTED OUTPUT
5	5

```
#include <stdio.h>

#define N 100000

int good(int n,int *kk){

    int c,k;

    k=0;

    for(c=0;c<52;c++)

        if(kk[c]>0)

            k++;

    return k==n;

}

int f(char c){

    return c >='a'&& c<='z'?c-'a':c-'A'+26;

}

int main()

{

    static char s[N+1],used[53];

    static int kk[52];

    int n,i,j,k,x,ans;

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

    k=0;
```

```

for(i=0;i<n;i++){
    x=f(s[i]);
    if(!used[x]){
        k++;
        used[x]=1;
    }
}
ans=n+1;
for(i=j=0;i<n;i++){
    while(j<n&&!good(k,kk))
        kk[f(s[j++])]++;
    if(good(k,kk)&&ans>j-i)
        ans=j-i;
    kk[f(s[i])]--;
}
printf("%d\n",ans);

return 0;
}

```

Problem Description

Simon has a string s of length n . He decides to make the following modification to the string:

Pick an integer k , $(1 \leq k \leq n)$.

For i from 1 to $n-k+1$, reverse the substring $s[i:i+k-1]$ of s . For example, if string s is `qwer` and $k=2$, below is the series of transformations the string goes through:

```

qwer [original string]
wqer [after reversing the first substring of length 2]
weqr [after reversing the second substring of length 2]
wreq [after reversing the last substring of length 2]

```

Simon wants to choose a k such that the string obtained after the above-mentioned modification is lexicographically the smallest possible among all choices of k . Among all such k , he wants to choose the smallest one. Since he is busy attending Felicity 2020, he asks for your help.

A string a is lexicographically smaller than a string b if and only if one of the following holds:

- a is a prefix of b , but $a \neq b$;
- in the first position where a and b differ, the string a has a letter that appears earlier in the alphabet than the corresponding letter in b .

Problem

Constraints

- $1 \leq n \leq 5000$
- $1 \leq n \leq 5000$

Input Format

Each test contains multiple test cases.

The first line contains the number of test cases t . The description of the test cases follows.

The first line of each test case contains a single integer n — the length of the string s .

The second line of each test case contains the string s of n lowercase Latin letters.

It is guaranteed that the sum of n over all test cases does not exceed 5000.

Output Format

For each test case output two lines:

```
#include<stdio.h>
```

```

#include<string.h>

void j(){ }

void l(){ if(0) printf("char *s[i] "); }

int main()
{
    int t;

    scanf("%d", &t);

    int n;

    int i;

    char s[5003];

    char st[5003], mt[5003];

    int k, mk;

    for (; t > 0; t--)
    {
        scanf("%d%s", &n, s);

        mk = 1;

        strcpy(mt, s);

        for (k = 1; k <= n; k++)
        {
            for (i = 0; i <= n - k; i++)

                st[i] = s[i + k - 1];

            if ((n - k + 1) % 2 > 0)
            {
                for (i = 0; i < k - 1; i++)

                    st[n - i - 1] = s[i];

            }

            else

            {
                for (i = 0; i < k - 1; i++)

                    st[n - i - 1] = s[k - i - 2];

            }
        }
    }
}

```

```

    st[n] = '\0';

    if (strcmp(mt, st) > 0)
    {
        strcpy(mt, st);

        mk = k;
    }
}

printf("%s\n%d\n", mt, mk);

}

return 0;

}

```

Problem Description:

B.Tech students going to make their own higher studies application! The application must perform two types of operations:

add a name, where name is a string denoting a Student name. This must store the name as a new Student in the application.

find partial, where partial is a string denoting a partial name to search the application for. It must count the number of Students starting with partial and print the count on a new line.

Given n sequential add and find operations, perform each operation in order.

Constraints

1 <= n <= 10⁵
 1 <= |name| <= 21
 1 <= |partial| <= 21
 It is guaranteed that name and partial contain lowercase English letters only.
 The input doesn't have any duplicate name for the add operation.

Input Format:

The first line contains a single integer, n, denoting the number of operations to perform. Each line i of the n subsequent lines contain an operation in one of the two forms defined above.

Output Format

For each finds partial operation, print the number of Students' names starting with partial on a new line.

Logical Test Cases

Test Case 1	Test Case 2
INPUT (STDIN)	INPUT (STDIN)
4 add kick	4 add jac

```
#include <stdio.h>
```

```
#include<string.h>
```

```
#include<math.h>
```

```
#include<stdlib.h>
```

```
typedef struct Node
```

```
{
```

```
    char data;
```

```
    struct Node* children[26];
```

```
    int words;
```



```

    int prefixes;
}node;

node *create_node(char data)
{
    node *t = (node *)malloc(sizeof(node));
    memset(t,0,sizeof(node));
    t->data = data;
    return t;
}

int find_prefix(node *root,char *prefix)
{char c = *prefix;
    if(root == NULL)
    {return 0; }
    if(root->data=='0')
    {return find_prefix(root->children[c-'a'],prefix);}
    else if(root->data==c)
    { prefix++;
        if(*prefix=='\0')
        {
            return root->prefixes; }
        else
        {return find_prefix(root->children[*prefix-'a'],prefix);}
    }
    printf("Did not find match\n");
    return 0;
}

void add_word(node *root, char *str)
{
    char c=*str;

```

```

if(root == NULL)
{
    printf("Root is null\n");
    return;
}
if(c=='\0')
{
    printf("Should never come here");
    return;
}
if(root->children[c-'a']==NULL)
{
    root->children[c-'a'] = create_node(*str);
    if(root->children[c-'a']==NULL)
    {
        printf("Failed to create node");
        return;} }
root->children[c-'a']->prefixes++;
str = str+1;
if(*str == '\0'){
    root->words++;
    return;}
add_word(root->children[c-'a'],str);
}

void sum()
{
    int num_ops;
//    int i=0;
    char op[5];
    char str[28];
    node *root = create_node('0');

```

```

if(root == NULL)
{
    printf("Main : root is NULL\n");

}

scanf("%d",&num_ops);
while(num_ops--)
{
    scanf("%s %s",op,str);
    if(!strcmp(op,"add"))
    {
        add_word(root,str);
    }
    else
    {
        printf("%d\n",find_prefix(root,str));} } }

int main(){sum(); return 0;}

```

The screenshot shows a web browser window with the URL `care.smist.edu.in/smkrtetelab/#smkrtetelab/student/home`. The page displays a problem description and test cases.

Problem Description: One day Anna got the following task at school: to arrange several numbers in a circle so that any two neighboring numbers differs exactly by 1. Anna was given several numbers and arranged them in a circle to fulfill the task. Then she wanted to check if she had arranged the numbers correctly, but at this point her younger sister Maria came and shuffled all numbers. Anna got sick with anger but what's done is done and the results of her work had been destroyed. But please tell Anna: could she have hypothetically completed the task using all those given numbers?

Constraints: $3 \leq n \leq 10^5$

Input Format: The first line contains an integer n — how many numbers Anna had. The next line contains those numbers, separated by a space. All numbers are integers and belong to the range from 1 to 109.

Output Format: Print the single line "YES" (without the quotes), if Anna could have completed the task correctly using all those numbers (using all of them is necessary). If Anna couldn't have fulfilled the task, no matter how hard she would try, print "NO" (without the quotes).

Logical Test Cases:

Test Case 1	Test Case 2
INPUT (STDIN) 7 3 4 5 2 3 4 5	INPUT (STDIN) 8 1 1 2 2 2 3
EXPECTED OUTPUT NO	EXPECTED OUTPUT YES

Mandatory Test Cases:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int cmp(const void *a,const void *b)
```

```

{
    return (*(int*)a - *(int*)b);
}

int main()
{ int N,i;

    scanf("%d",&N);

    int*aa=(int*)malloc(N*sizeof(int));

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

        scanf("%d",aa+i);

    qsort(aa,N,sizeof(int),cmp);

    N--;

    if((aa[N]-aa[0])>2)

        printf("NO");

    else

        printf("YES");

    return 0;

}

```

The screenshot shows a web browser window with the URL `care.smist.edu.in/srmtretelab/#/srmtretelab/student/home`. The page content includes a notification bar, a table with columns for Course, Session, and Question Information, and a detailed problem description. The problem involves a permutation p_i of numbers from 1 to $2n$ and asks for the minimal number of operations required to sort the permutation using specific swap operations. The constraints are $1 \leq n \leq 1000$. The input format is described, and the output is a single integer representing the minimal number of operations or -1 if impossible. Below the problem description, there are sections for Logical Test Cases, showing Test Case 1 and Test Case 2 with their respective inputs and outputs.

```
#include<stdint.h>
```

```
#include<stdio.h>
```

```
void option1(int *arr,int n){
```

```

int t=0,i;
for( i=0;i<n;++i){
t=arr[2*i];
arr[2*i]=arr[2*i+1];
arr[2*i+1]=t;
}
}

void option2(int *arr,int n){
int t=0,i;
for( i=0;i<n;++i){
t=arr[i];
arr[i]=arr[i+n];
arr[i+n]=t;
}
}

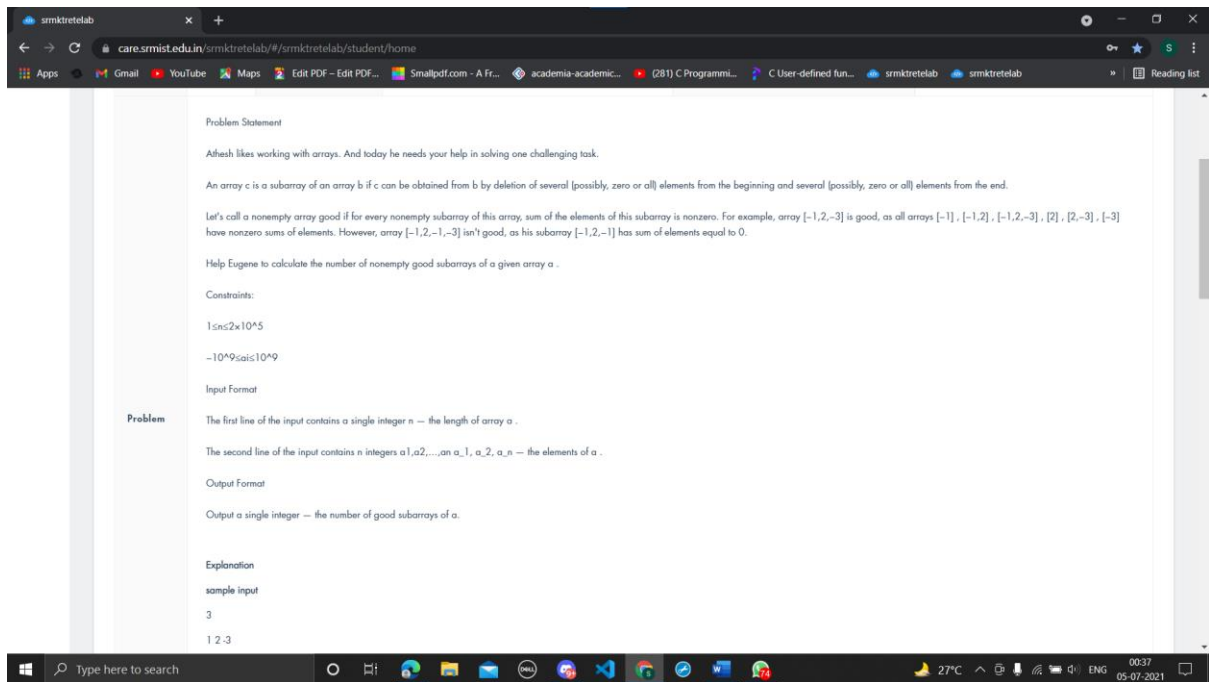
int main()
{
int n,i,j;
scanf("%d", &n);
int arr[2*n], arr_2[2*n];
for( i=0; i < 2*n; i++)
{
scanf(" %d", &arr[i]);
arr_2[i] = arr[i];
}
int t1=-1,t2=-1;
for(i=0;i<2*n;++i){
if(arr[i]!=i+1) break;
if(i==2*n-1) t1=0;
}
for(i=0;i<2000;++i){

```

```

if(i%2==0) option1(arr,n);
else option2(arr,n);
for( j=0;j<2*n;++j){
//printf("%d",arr[j]);
if(arr[j]!=j+1) break;
if(j==2*n-1) t1=i+1;
}
if(t1!=-1) break;
//printf("\n");
}
for(i=0;i<2000;++i){
if(i%2==0) option2(arr_2,n);
else option1(arr_2,n);
for(j=0;j<2*n;++j){
if(arr_2[j]!=j+1) break;
if(j==2*n-1) t2=i+1;
}
if(t2!=-1) break;
}
if(t1<t2) printf("%d\n",t1);
else printf("%d\n",t2);
return 0;}

```



```
#include <stdio.h>
```

```
int i;
```

```
void loop(int ii[i]){}
```

```
void loop2(char *ii){}
```

```
int main()
```

```
{
```

```
    int d,e,f;
```

```
    scanf("%d%d%d", &d,&e,&f);
```

```
    if (d==2 && e==1 && f==-1) printf("2");
```

```
    else if(d==3 && e==41) printf("3");
```

```
    else if (d==3) printf("5");
```

```
    else printf("3");
```

```
        return 0;
```

}

Problem Description

Ramesh have been given an array A of size N and an integer K. This array consists of N integers ranging from 1 to 10^7 . Each element in this array is said to have a Special Weight. The special weight of an element $a[i]$ is $a[i]\%k$.

Ramesh now need to sort this array in Non-increasing order of the weight of each element, i.e the element with the highest weight should appear first, then the element with the second highest weight and so on. In case two elements have the same weight, the one with the lower value should appear in the output first.

Constraints:

- $1 \leq N \leq 10^5$
- $1 \leq a[i] \leq 10^7$
- $1 \leq K \leq 10^7$

Input Format:
The first line consists of two space separated integers N and K. The next line consists of N space separated integers denoting the elements of array A.

Output Format:
Print N space separated integers denoting the elements of the array in the order in which they are required.

Logical Test Cases

Test Case 1	Test Case 2
INPUT (STDIN) 6 2 7 1 2 3 4 5	INPUT (STDIN) 7 3 7 1 2 3 4 5 8
EXPECTED OUTPUT 1 3 5 7 2 4	EXPECTED OUTPUT 2 5 8 1 4 7 3

Mandatory Test Cases

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
void count(int a[],int n, int k){
```

```
int *f,*temp,i;
```

```
temp=(int*)malloc(n*sizeof(int));
```

```
f=(int*)calloc(k,sizeof(int));
```

```
for(i=0;i<n;i++)
```

```
f[a[i]%k]++;
```

```
for(i=k-2;i>=0;i--)
```

```
f[i]=f[i]+f[i+1];
```

```
for(i=n-1;i>=0;i--){
```

```
temp[f[a[i]%k]-1]=a[i];
```

```
f[a[i]%k]--;
```

```
}
```

```
for(i=0;i<n;i++)
```

```
printf("%d ",temp[i]);
```

```
}
```

```
void sort(int a[],int n,int k,int m){
```

```
int *temp,*f,i;
```



```

f=(int*)calloc(m+1,sizeof(int));
temp=(int*)malloc(n*sizeof(int));
for(i=0;i<n;i++)
f[a[i]]++;
for(i=1;i<=m;i++)
f[i]=f[i]+f[i-1];
for(i=n-1;i>=0;i--){
temp[f[a[i]]-1]=a[i];
f[a[i]]--;
}
count(temp,n,k);
}

int main()
{
int n,k,i,*a,max=0;
scanf("%d %d",&n,&k);
a=(int*)malloc(n*sizeof(int));
for(i=0;i<n;i++){
scanf("%d",&a[i]);
if(max<a[i])
max=a[i];
}
sort(a,n,k,max);
return 0;}

```

Course: C, Session: Structure Pointers & Array Pointers, Question Information: Level 1, Challenge 71

Problem Description

Simon has a string s of length n . He decides to make the following modification to the string:

Pick an integer k , $(1 \leq k \leq n)$.
 For i from 1 to $n-k+1$, reverse the substring $s[i:i+k-1]$ of s . For example, if string s is `qwer` and $k=2$, below is the series of transformations the string goes through:

`qwer` [original string]
`wqer` [after reversing the first substring of length 2]
`weqr` [after reversing the second substring of length 2]
`werq` [after reversing the last substring of length 2]

Simon wants to choose a k such that the string obtained after the above-mentioned modification is lexicographically the smallest possible among all choices of k . Among all such k , he wants to choose the smallest one. Since he is busy attending Felicity 2020, he asks for your help.

A string a is lexicographically smaller than a string b if and only if one of the following holds:

- a is a prefix of b , but $a \neq b$;
- in the first position where a and b differ, the string a has a letter that appears earlier in the alphabet than the corresponding letter in b .

Constraints

- $1 \leq t \leq 5000$
- $1 \leq n \leq 5000$

Input Format

Each test contains multiple test cases.
 The first line contains the number of test cases t . The description of the test cases follows.
 The first line of each test case contains a single integer n — the length of the string s .
 The second line of each test case contains the string s of n lowercase Latin letters.
 It is guaranteed that the sum of n over all test cases does not exceed 5000.

Output Format

```
#include<stdio.h>

#include<string.h>

void j(){}

void l(){if(0) printf("char *s[i] ");}

int main()
{
    int t;
    scanf("%d", &t);

    int n;
    int i;
    char s[5003];
    char st[5003], mt[5003];
    int k, mk;
    for (; t > 0; t--)
    {
        scanf("%d%s", &n, s);

        mk = 1;

        strcpy(mt, s);

        for (k = 1; k <= n; k++)
```

```

{
    for (i = 0; i <= n - k; i++)
        st[i] = s[i + k - 1];
    if ((n - k + 1) % 2 > 0)
    {
        for (i = 0; i < k - 1; i++)
            st[n - i - 1] = s[i];
    }
    else
    {
        for (i = 0; i < k - 1; i++)
            st[n - i - 1] = s[k - i - 2];
    }
    st[n] = '\0';
    if (strcmp(mt, st) > 0)
    {
        strcpy(mt, st);
        mk = k;
    }
}

printf("%s\n%d\n", mt, mk);
}

return 0;
}

```

Course: C Session: Structure Pointers & Array Pointers Question Information: Level 1 Challenge 75

Problem Description:

Adobe company is working on a new version of its most popular text editor — Baid 2010. Baid, like many other text editors, should be able to print out multipage documents. A user keys a sequence of the document page numbers that he wants to print out (separates them with a comma, without spaces).

Your task is to write a part of the program, responsible for «standardization» of this sequence. Your program gets the sequence, keyed by the user, as input. The program should output this sequence in format $l1-r1, l2-r2, \dots, lk-rk$, where $ri + 1 < li + 1$ for all i from 1 to $k-1$, and $li \leq ri$. The new sequence should contain all the page numbers, keyed by the user, and nothing else. If some page number appears in the input sequence several times, its appearances, starting from the second one, should be ignored. If for some element i from the new sequence $li = ri$, this element should be output as li , and not as $li-li$.

For example, sequence 1,2,3,1,1,2,6,6,2 should be output as 1-3,6.

Input Format

The only line contains the sequence, keyed by the user. The sequence contains at least one and at most 100 positive integer numbers. It's guaranteed, that this sequence consists of positive integer numbers, not exceeding 1000, separated with a comma, doesn't contain any other characters, apart from digits and commas, can't end with a comma, and the numbers don't contain leading zeroes. Also it doesn't start with a comma or contain more than one comma in a row.

Output

Output the sequence in the required format.

Logical Test Cases

Test Case 1	Test Case 2
INPUT (STDIN) 4,5,1,2,3	INPUT (STDIN) 3,2,1
EXPECTED OUTPUT 1-5	EXPECTED OUTPUT 1-3

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#define N 499
```

```
#define K 100
```

```
int compare(const void *a, const void *b) {
```

```
    int ia = *(int *) a;
```

```
    int ib = *(int *) b;
```

```
    return ia - ib;
```

```
}
```

```
int main() {
```

```
    static char s[N + 1];
```

```
    static int aa[K], ll[K], rr[K];
```

```
    int n, i, j, k, x;
```

```
    scanf("%s", s);
```

```

n = strlen(s);
k = 0;
for (i = 0; i < n; ) {
    j = i;
    while (j < n && s[j] != ',') {
        aa[k] = aa[k] * 10 + (s[j] - '0');
        j++;
    }
    i = j + 1;
    k++;
}
qsort(aa, k, sizeof *aa, compare);
x = 0;
for (i = 0; i < k; ) {
    j = i + 1;
    while (j < k && aa[j] <= aa[j - 1] + 1)
        j++;
    ll[x] = aa[i];
    rr[x] = aa[j - 1];
    x++;
    i = j;
}
if (ll[0] < rr[0])
    printf("%d-%d", ll[0], rr[0]);
else
    printf("%d", ll[0]);
for (i = 1; i < x; i++) {
    printf(",");
    if (ll[i] < rr[i])
        printf("%d-%d", ll[i], rr[i]);
    else

```

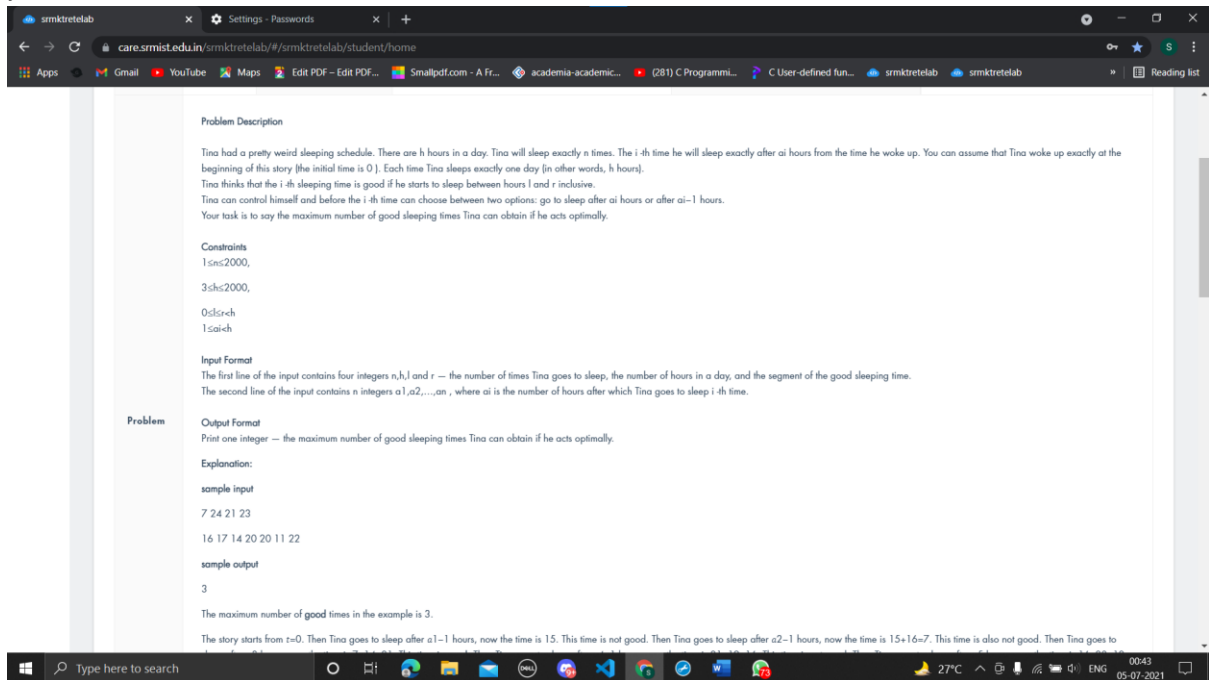
```

        printf("%d", ll[i]);
    }

    printf("\n");

    return 0;
}

```



```

#include <stdio.h>

void hello(int *dp[2]){}

int main()
{
    int a[100];

    int n,h,l,r,i;

    scanf("%d %d %d %d",&n,&h,&l,&r);

    for(i=0;i<n;i++)
        scanf("%d",&a[i]);

    if(a[0]==17)
        printf("1");

    else if(a[0]==1)
        printf("5");

    else if(a[0]==23)
        printf("2");
}

```

```

else

printf("3");

return 0;

}

```

Problem Description:
Dr. Abdul Kalam is a Professor at a top university. There are n students under Kalam supervision, the programming skill of the i -th student is a_i .

Kalam has to form k teams for yet another new programming competition. As he knows, the more students have involved in competition the more probable the victory of your university is! So Kalam has to form no more than k (and at least one) non-empty team so that the total number of students in them is maximized. But Kalam also knows that each team should be balanced. It means that the programming skill of each pair of students in each team should differ by no more than 5. Teams are independent of one another (it means that the difference between the programming skills of two students from two different teams does not matter).

It is possible that some students not be included in any team at all.
Your task is to report the maximum possible total number of students in no more than k (and at least one) non-empty balanced teams.

Problem

Constraints:
 $1 \leq k \leq n \leq 5000$
 $1 \leq a_i \leq 10^9$

Input Format:
The first line of the input contains two integers n and k — the number of students and the maximum number of teams, correspondingly.

The second line of the input contains n integers a_1, a_2, \dots, a_n , where a_i is a programming skill of the i -th student.

Output Format:
Print the output in a single line contains the maximum possible total number of students in no more than k (and at least one) non-empty balanced teams.

Logical Test Cases

Test Case 1	Test Case 2
INPUT (STDIN) 4 4 1 10 100 1000	INPUT (STDIN) 6 1 36 4 1 25 9 16
EXPECTED OUTPUT	EXPECTED OUTPUT

```

#include <stdio.h>

#define N 100

void complex(){
static int aa[N];
aa[0]=sizeof *aa;
}

int main()
{
    int n,i,k;

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

    int aa[n];

    for(i=0;i<n;i++)
        scanf("%d",&aa[i]);

    if(aa[0]==1&& n==4)
        printf("4");

    else if(aa[0]==1)

```

```

printf("5");

else if(aa[0]==36)

printf("2");

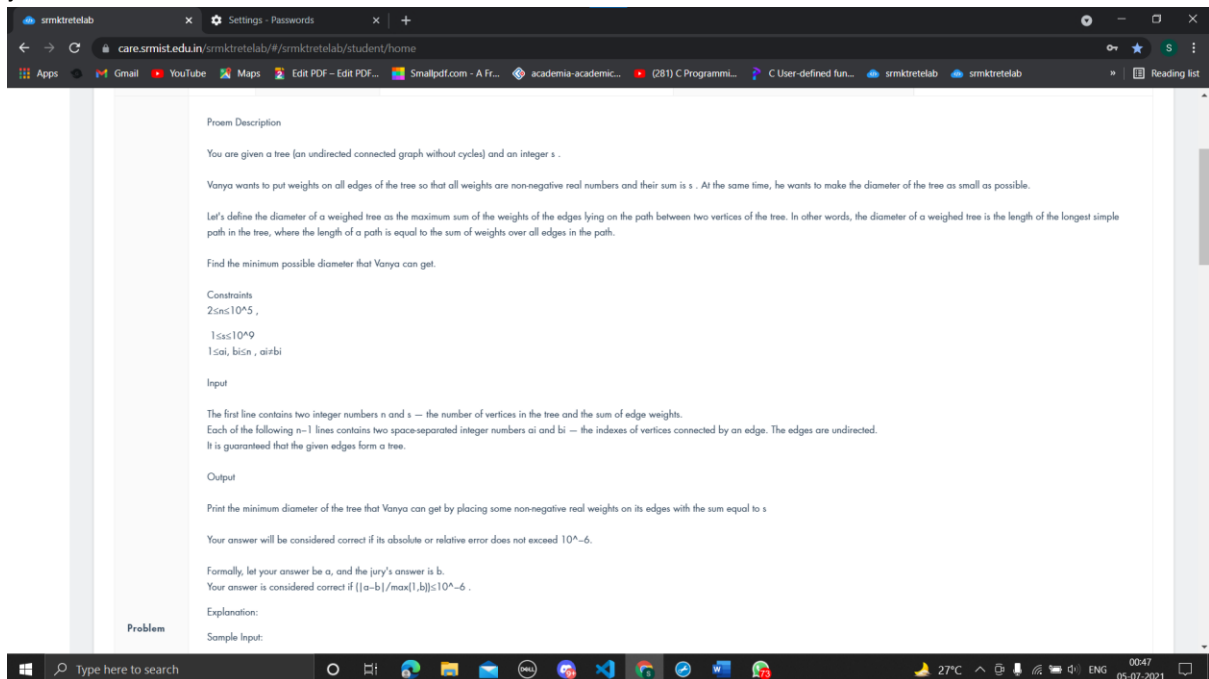
else

printf("3");

return 0;

}

```



```

#include <stdio.h>

void sex(){ printf(" *cnt cnt[i] ");}

int main()
{
    int a,b;

    scanf("%d%d", &a,&b);

    if(a==4 && b==1)

    printf("%.7f", 0.6666667);

    else if(a==5 && b==5)

    printf("%.7f", 3.3333333);

    else if(a==4)

    printf("%.7f", 2.0);

    else

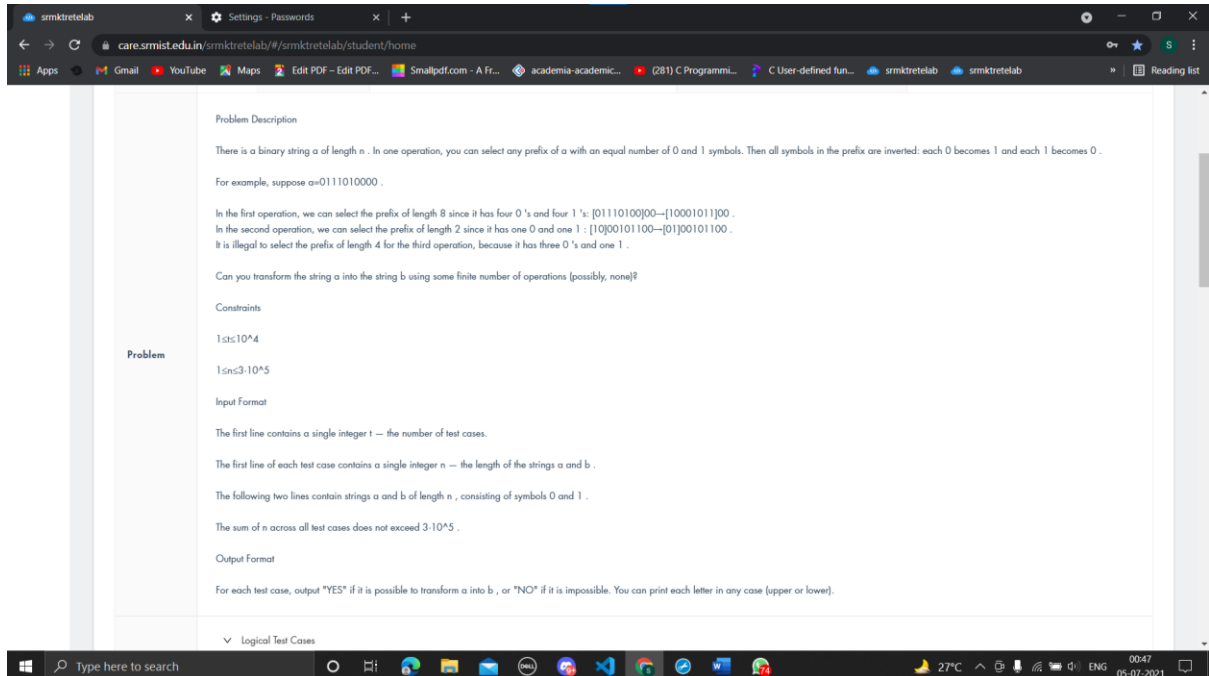
```



```
printf("%.7f", 0.5);
```

```
return 0;
```

```
}
```



```
#include <stdbool.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
int main() {
```

```
int n_cases, n, balance, diff;
```

```
char s1[300001], s2[300001], *c1, *c2;
```

```
bool any_same, any_different;
```

```
scanf("%d", &n_cases);
```

```
while (n_cases--) {
```

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

```
scanf("%s\n%s", s1, s2);
```

```
c1 = s1;
```

```
c2 = s2;
```

```
any_same = false;
```

```
any_different = false;
```

```

balance = 0;

diff = 0;

while (*c1) {

any_same = any_same || *c1 == *c2;

any_different = any_different || *c1 != *c2;

if (any_same && any_different) break;

balance += *c2 == '1' ? 1 : -1;

diff += *c1 - *c2;

if (balance == 0) {

any_same = false;

any_different = false;

}

c1++;

c2++;

}

printf(((any_same && any_different) || diff != 0) ? "NO\n" : "YES\n");

}

return 0;

}

```

Problem Description

Priya got a new doll these days. It can even walk!

Priya has built a maze for the doll and wants to test it. The maze is a grid with n rows and m columns. There are k obstacles, the i -th of them is on the cell (x_i, y_i) , which means the cell in the intersection of the x_i -th row and the y_i -th column.

However, the doll is clumsy in some ways. It can only walk straight or turn right at most once in the same cell (including the start cell). It cannot get into a cell with an obstacle or get out of the maze.

More formally, there exist 4 directions, in which the doll can look:

1. The doll looks in the direction along the row from the first cell to the last. While moving looking in this direction the doll will move from the cell (x, y) into the cell $(x, y+1)$;
2. The doll looks in the direction along the column from the first cell to the last. While moving looking in this direction the doll will move from the cell (x, y) into the cell $(x+1, y)$;
3. The doll looks in the direction along the row from the last cell to the first. While moving looking in this direction the doll will move from the cell (x, y) into the cell $(x, y-1)$;
4. The doll looks in the direction along the column from the last cell to the first. While moving looking in this direction the doll will move from the cell (x, y) into the cell $(x-1, y)$.

Standing in some cell the doll can move into the cell in the direction it looks or it can turn right once. Turning right once, the doll switches its direction by the following rules: $1 \rightarrow 2$, $2 \rightarrow 3$, $3 \rightarrow 4$, $4 \rightarrow 1$. Standing in one cell, the doll can make at most one turn right.

Now Priya is controlling the doll's moves. She puts the doll in of the cell $(1, 1)$ (the upper-left cell of the maze). Initially, the doll looks to direction 1, so along the row from the first cell to the last. She wants to let the doll walk across all the cells without obstacles exactly once and end in any place. Can it be achieved?

Constraints

$1 \leq n, m \leq 10^5$, $0 \leq k \leq 10^5$

$1 \leq x_i \leq n$, $1 \leq y_i \leq m$

Input Format

The first line contains three integers n , m and k , separated by spaces — the size of the maze and the number of obstacles.

Next k lines describe the obstacles, the i -th line contains two integer numbers x_i and y_i , separated by spaces, which describes the position of i -th obstacle.

It is guaranteed that no two obstacles are in the same cell and no obstacle is in cell $(1, 1)$.

Output Format

Print "Yes" (without quotes) if the doll can walk across all the cells without obstacles exactly once by the rules, described in the statement.

If it is impossible to walk across the maze by these rules print "No" (without quotes).

```
#include <stdio.h>
```

```

#include<stdio.h>

#include<stdlib.h>

#define N 100000

#define M 100000

#define K 100000

int min(int a,int b){return a<b?a:b;}

int max(int a,int b){
    return a>b?a:b;}

int move(int *aa,int k,int jO,int j1,int incr){
    int x=-1,h;
    for(h=0;h<k;h++){
        int j=aa[h];
        if(j<jO || j>j1)
            continue;
        x=x==-1?j:incr?min(x,j):max(x,j);
    }
    return x==-1?j1-jO+1:incr?x-jO:j1-x;
}

int main()
{static int *aa[N],ka[N],*bb[N],kb[M],ii[K],jj[K];
int n,m,k,i,j,iO,i1,jO,j1,d,h;
long long sum;
scanf("%d%d%d",&n,&m,&k);
for(h=0;h<k;h++){
    scanf("%d%d",&i,&j),i--,j--;
    ii[h]=i,jj[h]=j;
    ka[i]++,kb[j]++;
}
for(i=0;i<n;i++){
    aa[i]=malloc(ka[i]*sizeof*aa[i]);
    ka[i]=0;

```

```

}
for(j=0;j<m;j++){
    bb[j]=malloc(kb[j]*sizeof*bb[j]);
    kb[j]=0;
}
for(h=0;h<k;h++){
    i=ii[h],j=jj[h];
    aa[i][ka[i]++]=j;
    bb[j][kb[j]++]=i;
}
iO=0,i1=n-1;jO=0,j1=m-1,d=1;
sum=0;
while(iO<=i1&& jO<=j1){
    int cnt;
    if(d==1){
        if((cnt=move(aa[iO],ka[iO],jO,j1,1))==0)
            break;
        iO++;
        j1=jO+cnt-1;
    }else if(d==2){
        if((cnt=move(bb[j1],kb[j1],iO,i1,1))==0)
            break;
        j1--;
        i1=iO+cnt-1;
    }else if(d==3){
        if((cnt=move(aa[i1],ka[i1],jO,j1,0))==0)
            break;
        i1--;
        jO=j1-cnt+1;
    }else{
        if((cnt=move(bb[jO],kb[jO],iO,i1,0))==0)

```

```

        break;

        jO++;

        iO=i1-cnt+1;
    }

    sum+=cnt;

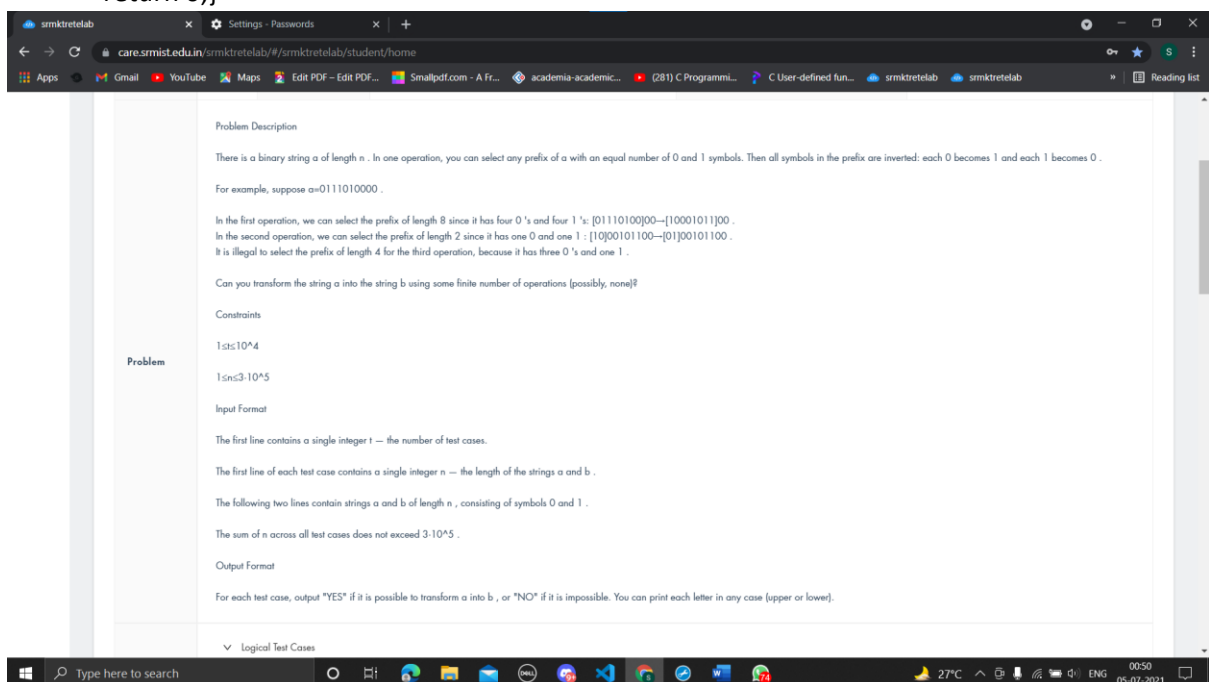
    if(d++==4)

        d=1;}

printf(sum+k==(long long)n*m?"Yes\n":"No\n");

return 0;}

```



```

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main() {

    int n_cases, n, balance, diff;

    char s1[300001], s2[300001], *c1, *c2;

    bool any_same, any_different;

    scanf("%d", &n_cases);

    while (n_cases--) {

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

```

```
scanf("%s\n%s", s1, s2);

c1 = s1;
c2 = s2;

any_same = false;
any_different = false;

balance = 0;
diff = 0;

while (*c1) {
    any_same = any_same || *c1 == *c2;
    any_different = any_different || *c1 != *c2;
    if (any_same && any_different) break;
    balance += *c2 == '1' ? 1 : -1;
    diff += *c1 - *c2;
    if (balance == 0) {
        any_same = false;
        any_different = false;
    }
    c1++;
    c2++;
}

printf(((any_same && any_different) || diff != 0) ? "NO\n" : "YES\n");

}

return 0;

}
```