Codeforces Round #677 (Div. 3)

官方题解

A. Boring Apartments

A. Boring Apartments

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

There is a building consisting of $10\,000$ apartments numbered from 1 to $10\,000$, inclusive.

Call an apartment **boring**, if its number consists of *the same digit*. Examples of boring apartments are $11, 2, 777, 9999$ and so on.

Our character is a troublemaker, and he calls the intercoms of all **boring** apartments, till someone answers the call, in the following order:

- First he calls all apartments consisting of digit 1, in increasing order $(1, 11, 111, 1111)$.
- Next he calls all apartments consisting of digit 2, in increasing order $(2, 22, 222, 2222)$
- And so on.

The resident of the boring apartment x answers the call, and our character **stops** calling anyone further.

Our character wants to know how many digits he pressed in total and your task is to help him to count the total number of keypresses.

For example, if the resident of boring apartment 22 answered, then our character called apartments with numbers $1, 11, 111, 1111, 2, 22$ and the total number of digits he pressed is $1 + 2 + 3 + 4 + 1 + 2 = 13$.

You have to answer t independent test cases.

Input

The first line of the input contains one integer t $(1 \le t \le 36)$ — the number of test cases.

The only line of the test case contains one integer x $(1 \le x \le 9999)$ — the apartment number of the resident who answered the call. It is guaranteed that x consists of the same digit.

Output

For each test case, print the answer: how many digits our character pressed in total.

Example

input Copy

```
4
22
9999
1
777
```

output Copy

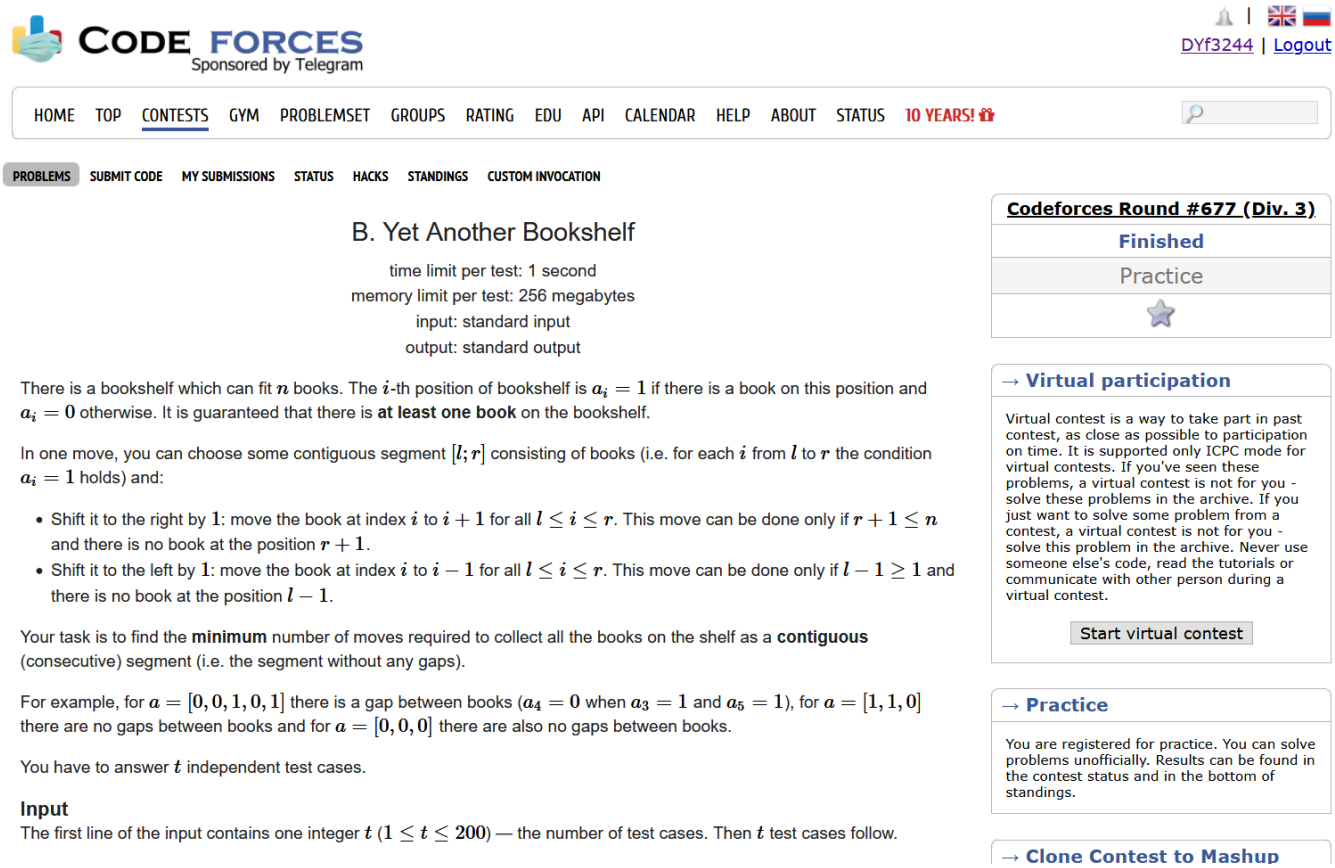
```
13
90
1
66
```


- 模拟
- 先算出这个数的最高位，那么这个数的最高位-1的任何情况就都能满足，同时这个最高位也决定了余数的个数

```cpp
#include <bits/stdc++.h>
using namespace std;
#define LL long long
#define sigma_size 30
#define max_size (int)(1e6+10)
#define MAX (int)(1e5+7)


int ans[40];
int main ()
{
    ios::sync_with_stdio(0);
    int T  ; cin >> T;
    for ( int cas = 1 ; cas <= T ; cas++ )
    {
        int x ; cin >> x;
        int dights = 0 ;
        int max_dights ;
        while ( x )
        {
            if ( x < 10 ) max_dights = x;
            x /= 10;
            dights++;
        }
        int res = (max_dights-1)*10;
        for ( int i = 1 ; i <= dights ; i++ )
            res += i;
        ans[cas] = res;
    }
    for ( int i = 1 ; i <= T ; i++ )
        cout << ans[i] << endl;
}
```

B. Yet Another Bookshelf



CODE FORCES
Sponsored by Telegram

DYf3244 | Logout

HOME TOP CONTESTS GYM PROBLEMSET GROUPS RATING EDU API CALENDAR HELP ABOUT STATUS 10 YEARS!

PROBLEMS SUBMIT CODE MY SUBMISSIONS STATUS HACKS STANDINGS CUSTOM INVOCATION

B. Yet Another Bookshelf

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

There is a bookshelf which can fit n books. The i-th position of bookshelf is $a_i = 1$ if there is a book on this position and $a_i = 0$ otherwise. It is guaranteed that there is **at least one book** on the bookshelf.

In one move, you can choose some contiguous segment $[l; r]$ consisting of books (i.e. for each i from l to r the condition $a_i = 1$ holds) and:

- Shift it to the right by 1: move the book at index i to $i + 1$ for all $l \le i \le r$. This move can be done only if $r + 1 \le n$ and there is no book at the position $r + 1$.
- Shift it to the left by 1: move the book at index i to $i - 1$ for all $l \le i \le r$. This move can be done only if $l - 1 \ge 1$ and there is no book at the position $l - 1$.

Your task is to find the **minimum** number of moves required to collect all the books on the shelf as a **contiguous** (consecutive) segment (i.e. the segment without any gaps).

For example, for $a = [0, 0, 1, 0, 1]$ there is a gap between books ($a_4 = 0$ when $a_3 = 1$ and $a_5 = 1$), for $a = [1, 1, 0]$ there are no gaps between books and for $a = [0, 0, 0]$ there are also no gaps between books.

You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \le t \le 200$) — the number of test cases. Then t test cases follow.

Codeforces Round #677 (Div. 3)
Finished
Practice

→ Virtual participation

Virtual contest is a way to take part in past contest, as close as possible to participation on time. It is supported only ICPC mode for virtual contests. If you've seen these problems, a virtual contest is not for you - solve these problems in the archive. If you just want to solve some problem from a contest, a virtual contest is not for you - solve this problem in the archive. Never use someone else's code, read the tutorials or communicate with other person during a virtual contest.

Start virtual contest

→ Practice

You are registered for practice. You can solve problems unofficially. Results can be found in the contest status and in the bottom of standings.

→ Clone Contest to Mashup

The first line of the test case contains one integer n ($1 \le n \le 50$) — the number of places on a bookshelf. The second line of the test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 1$), where a_i is 1 if there is a book at this position and 0 otherwise. It is guaranteed that there is **at least one book** on the bookshelf.

Output

For each test case, print one integer: the **minimum** number of moves required to collect all the books on the shelf as a contiguous (consecutive) segment (i.e. the segment without gaps).

Example

input Copy

```
5
7
0 0 1 0 1 0 1
3
1 0 0
5
1 1 0 0 1
6
1 0 0 0 0 1
5
1 1 0 1 1
```

output Copy

```
2
0
2
4
1
```

Note

In the first test case of the example, you can shift the segment $[3; 3]$ to the right and the segment $[4; 5]$ to the right. After all moves, the books form the contiguous segment $[5; 7]$. So the answer is 2.

In the second test case of the example, you have nothing to do, all the books on the bookshelf form the contiguous segment already.

In the third test case of the example, you can shift the segment $[5; 5]$ to the left and then the segment $[4; 4]$ to the left again. After all moves, the books form the contiguous segment $[1; 3]$. So the answer is 2.

In the fourth test case of the example, you can shift the segment $[1; 1]$ to the right, the segment $[2; 2]$ to the right, the segment $[6; 6]$ to the left and then the segment $[5; 5]$ to the left. After all moves, the books form the contiguous segment $[3; 4]$. So the answer is 4.

In the fifth test case of the example, you can shift the segment $[1; 2]$ to the right. After all moves, the books form the contiguous segment $[2; 5]$. So the answer is 1.

- 贪心
- 双手一合，所有书归位即可

```cpp
#include <bits/stdc++.h>
using namespace std;
#define LL long long
#define sigma_size 30
#define max_size (int)(3e5+10)
#define MAX (int)(1e5+7)

int ans[205];

int main ()
{
    ios::sync_with_stdio(0);
    int T ; cin >> T;
    for ( int cas = 1 ; cas <= T ; cas++ )
    {
        int n ; cin >> n;
        int a[55];
        for ( int i = 1 ; i <= n ; i++ ) cin >> a[i];
```

```
        int cnt = 0;
        int first = -1 , last = -1;
        for ( int i = 1 ; i <= n ; i++ )
            if ( a[i] )
            {
                cnt++ ;
                last = i ;
            }
        for ( int i = n ; i >= 1 ; i-- ) if ( a[i] ) first = i;
        ans[cas] = last - first - 1 - cnt + 2;
    }
    for ( int i = 1 ; i <= T ; i++ )
        cout << ans[i] << endl;
}
```

C. Dominant Piranha

CODE FORCES
Sponsored by Telegram

DYf3244 | Logout

HOME TOP CONTESTS GYM PROBLEMSET GROUPS RATING EDU API CALENDAR HELP ABOUT STATUS 10 YEARS!

PROBLEMS SUBMIT CODE MY SUBMISSIONS STATUS HACKS STANDINGS CUSTOM INVOCATION

C. Dominant Piranha

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are n piranhas with sizes $a_1, a_2, \ldots, a_n$ in the aquarium. Piranhas are numbered from left to right in order they live in the aquarium.

Scientists of the Berland State University want to find if there is **dominant** piranha in the aquarium. The piranha is called **dominant** if it can eat all the other piranhas in the aquarium (except itself, of course). Other piranhas will do nothing while the **dominant** piranha will eat them.

Because the aquarium is pretty narrow and long, the piranha can eat only one of the adjacent piranhas during one move. Piranha can do as many moves as it needs (or as it can). More precisely:

- The piranha i can eat the piranha $i-1$ if the piranha $i-1$ exists and $a_{i-1} < a_i$.
- The piranha i can eat the piranha $i+1$ if the piranha $i+1$ exists and $a_{i+1} < a_i$.

When the piranha i eats some piranha, its **size increases by one** (a_i becomes $a_i + 1$).

Your task is to find **any dominant** piranha in the aquarium or determine if there are no such piranhas.

Note that you have to find **any** (exactly one) dominant piranha, you don't have to find all of them.

For example, if $a = [5, 3, 4, 4, 5]$, then the third piranha can be **dominant**. Consider the sequence of its moves:

- The piranha eats the second piranha and a becomes $[5, \underline{5}, 4, 5]$ (the underlined piranha is our candidate).
- The piranha eats the third piranha and a becomes $[5, \underline{6}, 5]$.
- The piranha eats the first piranha and a becomes $[\underline{7}, 5]$.
- The piranha eats the second piranha and a becomes $[\underline{8}]$.

You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. Then t test cases follow.

The first line of the test case contains one integer n ($2 \le n \le 3 \cdot 10^5$) — the number of piranhas in the aquarium. The second line of the test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$), where a_i is the size of the i-th piranha.

It is guaranteed that the sum of n does not exceed $3 \cdot 10^5$ ($\sum n \le 3 \cdot 10^5$).

Output

For each test case, print the answer: -1 if there are no dominant piranhas in the aquarium or **index** of **any** dominant piranha otherwise. If there are several answers, you can print any.

Example

input (Copy)

```
6
5
5 3 4 4 5
3
1 1 1
5
4 4 3 4 4
5
```

Codeforces Round #677 (Div. 3)
Finished
Practice

→ Virtual participation

Virtual contest is a way to take part in past contest, as close as possible to participation on time. It is supported only ICPC mode for virtual contests. If you've seen these problems, a virtual contest is not for you - solve these problems in the archive. If you just want to solve some problem from a contest, a virtual contest is not for you - solve this problem in the archive. Never use someone else's code, read the tutorials or communicate with other person during a virtual contest.

Start virtual contest

→ Practice

You are registered for practice. You can solve problems unofficially. Results can be found in the contest status and in the bottom of standings.

→ Clone Contest to Mashup

You can clone this contest to a mashup.

Clone Contest

→ Submit?

Language: GNU G++17 7.3.0

Choose file: 浏览... 未选择文件。

Submit

→ Last submissions

Submission	Time	Verdict
96127484	Oct/20/2020 18:23	Accepted
96120317	Oct/20/2020 18:13	Wrong answer on test 2

→ Problem tags

constructive algorithms greedy *900

No tag edit access

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

output Copy

```
3
-1
4
3
3
1
```

Note

The first test case of the example is described in the problem statement.

In the second test case of the example, there are no dominant piranhas in the aquarium.

In the third test case of the example, the fourth piranha can firstly eat the piranha to the left and the aquarium becomes $[4, 4, 5, 4]$, then it can eat any other piranha in the aquarium.

→ **Contest materials**

- Announcement
- Tutorial

- 模拟
- 只有全部的数字都相等才是不符合条件的。
- 在符合条件的数字当中，我们只需要找到这个数字的最大值，并且这个最大值左右至少能够吃掉一个，就能符合条件

```cpp
#include <bits/stdc++.h>
using namespace std;
#define LL long long
#define sigma_size 30
#define max_size (int)(3e5+10)
#define MAX (int)(1e5+7)

int ans[max_size];
int a[max_size];
int main ()
{
    ios::sync_with_stdio(0);
    int T ; cin >> T;
    for ( int cas = 1 ; cas <= T ; cas++ )
    {
        int n ; cin >> n;
        for ( int i = 1 ;  i <= n ; i++ ) cin >> a[i];
        bool flag = 1;
        int cur = a[1];
        for ( int i = 2 ; i <= n ; i++ )
            if ( a[i] != cur )
            {
                flag = 0;
                break;
            }
        if ( flag ) { ans[cas] = -1 ;}
        else
        {
            int maxx = 0 ;
            int res = -1;
            for ( int i = 1 ; i <= n ; i++ )
                maxx = max ( maxx , a[i] );
            for ( int i = 1 ; i <= n ; i++ )
            {
```

```
                if ( a[i] != maxx ) continue;
                if ( (i > 1 && a[i] > a[i-1]) || ( i < n && a[i] > a[i+1]) )
                {
                    res = i;
                    break;
                }
            }
            ans[cas] = res;
        }
    }
    for ( int i =1  ; i <= T ; i++ )
        cout << ans[i] << endl;
}
```

D. Districts Connection

D. Districts Connection

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are n districts in the town, the i-th district belongs to the a_i-th bandit gang. Initially, no districts are connected to each other.

You are the mayor of the city and want to build $n-1$ two-way roads to connect all districts (two districts can be connected directly or through other connected districts).

If two districts belonging to the same gang are connected **directly** with a road, this gang will revolt.

You don't want this so your task is to build $n-1$ two-way roads in such a way that all districts are reachable from each other (possibly, using intermediate districts) and **each pair** of directly connected districts belong to **different gangs**, or determine that it is impossible to build $n-1$ roads to satisfy all the conditions.

You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \le t \le 500$) — the number of test cases. Then t test cases follow.

The first line of the test case contains one integer n ($2 \le n \le 5000$) — the number of districts. The second line of the test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$), where a_i is the gang the i-th district belongs to.

It is guaranteed that the sum of n does not exceed 5000 ($\sum n \le 5000$).

Output

For each test case, print:

- NO on the only line if it is impossible to connect all districts satisfying the conditions from the problem statement.
- YES on the first line and $n-1$ roads on the next $n-1$ lines. Each road should be presented as a pair of integers x_i and y_i ($1 \le x_i, y_i \le n; x_i \ne y_i$), where x_i and y_i are two districts the i-th road connects.

For each road i, the condition $a[x_i] \ne a[y_i]$ should be satisfied. Also, all districts should be reachable from each other (possibly, using intermediate districts).

Example

input Copy

```
4
5
1 2 2 1 3
3
1 1 1
4
1 1000 101 1000
4
1 2 3 4
```

output Copy

```
YES
1 3
3 5
5 4
1 2
NO
YES
1 2
2 3
3 4
YES
1 2
1 3
1 4
```

- 模拟
- 只有所有的数字都相等的时候才输出NO
- 先对所有的数字按照值进行排序，然后我们让最小的值去连接所有与它不一样的值。然后再让最后一个值去连接这个最小值的其他数字即可

```
#include <bits/stdc++.h>
using namespace std;
#define LL long long
#define sigma_size 30
#define max_size (int)(1e5+10)
#define MAX (int)(1e5+7)

int main ()
{
    ios::sync_with_stdio(0);
    int T ; cin >> T;
    while (T--)
    {
        int n ; cin >> n;
        pair <int,int> a[5005];
        for ( int i = 1 ; i <= n ; i++ ) cin >> a[i].first , a[i].second = i;
        sort ( a+1 , a+1+n );
        if ( a[1].first == a[n].first )
        {
            cout << "NO" << endl;
        }
        else
        {
            cout << "YES" << endl;
            int cur = a[1].first;
            int i = 2;
            while ( i <= n && a[i].first == cur ) i++;
            int j = i;
            while ( j <= n )
            {
                cout << a[1].second << " " << a[j].second << endl;
                j++;
            }
            int k = 2 ;
            while ( k < i )
            {
                cout << a[n].second << " " << a[k].second << endl;
                k++;
            }
        }

    }
}
```

E. Two Round Dances

E. Two Round Dances

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

One day, n people (n is an even number) met on a plaza and made two round dances, each round dance consists of exactly $\frac{n}{2}$ people. Your task is to find the number of ways n people can make two round dances if each round dance consists of exactly $\frac{n}{2}$ people. Each person should belong to exactly one of these two round dances.

Round dance is a dance circle consisting of 1 or more people. Two round dances are indistinguishable (equal) if one can be transformed to another by choosing the first participant. For example, round dances $[1, 3, 4, 2]$, $[4, 2, 1, 3]$ and $[2, 1, 3, 4]$ are indistinguishable.

For example, if $n = 2$ then the number of ways is 1: one round dance consists of the first person and the second one of the second person.

For example, if $n = 4$ then the number of ways is 3. Possible options:

- one round dance — $[1, 2]$, another — $[3, 4]$;
- one round dance — $[2, 4]$, another — $[3, 1]$;
- one round dance — $[4, 1]$, another — $[3, 2]$.

Your task is to find the number of ways n people can make two round dances if each round dance consists of exactly $\frac{n}{2}$ people.

Input

The input contains one integer n ($2 \le n \le 20$), n is an even number.

Output

Print one integer — the number of ways to make two round dances. It is guaranteed that the answer fits in the 64-bit integer data type.

Examples

input	output
2	1
4	3
8	1260
20	12164510040883200

- 公式$\frac{n!}{(n/2)^2 * 2}$

```cpp
#include <bits/stdc++.h>
using namespace std;
#define LL long long
#define sigma_size 30
#define max_size (int)(2e5 + 10)
#define MAX (int)(1e5+7)

LL fac[25];
LL two[25];
void get_fac() {
    fac[0] = fac[1] = 1;
    for ( int i = 2 ; i <= 22 ; i++ )
        fac[i] = fac[i-1] * i;
    two[0] = 1;
    for ( int i = 1 ; i <= 22 ; i++ )
        two[i] = two[i-1] * 2 ;
}

int main ()
{
    get_fac();
    int n ; cin >> n;
    cout << fac[n] / ((n/2)*(n/2)*2) << endl;
}
```

F. Zero Remainder Sum

F. Zero Remainder Sum

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a matrix a of size $n \times m$ consisting of integers.

You can choose **no more than** $\left\lfloor \frac{m}{2} \right\rfloor$ elements in **each row**. Your task is to choose these elements in such a way that their sum is **divisible by** k and this sum is the **maximum**.

In other words, you can choose no more than a half (rounded down) of elements in each row, you have to find the maximum sum of these elements divisible by k.

Note that you can choose zero elements (and the sum of such set is 0).

Input

The first line of the input contains three integers n, m and k ($1 \le n, m, k \le 70$) — the number of rows in the matrix, the number of columns in the matrix and the value of k. The next n lines contain m elements each, where the j-th element of the i-th row is $a_{i,j}$ ($1 \le a_{i,j} \le 70$).

Output

Print one integer — the maximum sum divisible by k you can obtain.

Examples

input Copy

```
3 4 3
1 2 3 4
5 2 2 2
7 1 1 4
```

output Copy

```
24
```

input Copy

```
5 5 4
1 2 4 2 1
3 5 1 2 4
1 5 7 1 2
3 8 7 1 2
8 4 7 1 6
```

output Copy

```
56
```

Note

In the first example, the optimal answer is 2 and 4 in the first row, 5 and 2 in the second row and 7 and 4 in the third row. The total sum is $2 + 4 + 5 + 2 + 7 + 4 = 24$.

- dp

- 我们设dp[i][j][cnt][rem]表示我们访问到了a[i][j]这个数字，并且第i行已经取了cnt个数的情况下，余数为rem的和，因此我们可以有状态转移方程：

$dp[i][j+1][cnt][rem] = max(dp[i][j+1][cnt][rem], dp[i][j][cnt][rem])$不取当前数字的情况 $dp[i][j+1][cnt+1][(rem+a[i][j])$

注意一点的是，如果j已经在边缘了，注意往下一行进行转移

```
#include <bits/stdc++.h>
using namespace std;
#define LL long long
#define ULL unsigned long long
#define sigma_size 30
#define max_size (int)(5e5+10)
#define MAX_SIZE (int)(4e6+7)

int n , m , k;
int a[75][75];

int dp[80][80][80][80];

int main ()
{
    ios::sync_with_stdio(0);
    cin >> n >> m >> k;
    for ( int i = 1 ; i <= n ; i++ )
        for ( int j = 1 ; j <= m ; j++ )
            cin >> a[i][j];
    for ( int i = 0 ; i <= n+5 ; i++ )
        for ( int j = 0 ; j <= m+5 ; j++ )
            for ( int cnt = 0 ; cnt <= m ; cnt++ )
                for ( int rem = 0 ; rem <= k ; rem++ )
                    dp[i][j][cnt][rem] = - ( 1 << 30 );
    dp[1][1][0][0] = 0 ;
    for ( int i = 1 ; i <= n ; i++ )
        for ( int j = 1 ; j <= m ; j++ )
            for ( int cnt = 0 ; cnt <= m/2 ; cnt++ )
                for ( int rem = 0 ; rem < k ; rem++ )
                {
                    if ( dp[i][j][cnt][rem] == - ( 1 << 30) ) continue;
                    if ( j < m )
                    {
                        dp[i][j+1][cnt][rem] = max ( dp[i][j+1][cnt][rem] , dp[i][j][cnt][rem] );
                        if ( cnt < m/2 )
                            dp[i][j+1][cnt+1][(rem+a[i][j])%k] = max ( dp[i][j+1][cnt+1][(rem+a[i]
[j])%k] , dp[i][j][cnt][rem] + a[i][j] );
                    }
                    else
                    {
                        dp[i+1][1][0][rem] = max ( dp[i+1][1][0][rem] , dp[i][j][cnt][rem] );
                        if ( cnt < m/2 )
                            dp[i+1][1][0][(rem+a[i][j])%k] = max (dp[i+1][1][0][(rem+a[i]
[j])%k],dp[i][j][cnt][rem]+a[i][j]);
                    }
                }
    cout << max ( 0 , dp[n+1][1][0][0] ) << endl;
}
```

G. Reducing Delivery Cost

G. Reducing Delivery Cost

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are a mayor of Berlyatov. There are n districts and m two-way roads between them. The i-th road connects districts x_i and y_i. The cost of travelling along this road is w_i. There is some path between each pair of districts, so the city is connected.

There are k delivery routes in Berlyatov. The i-th route is going from the district a_i to the district b_i. There is one courier on each route and the courier will always choose the **cheapest** (minimum by total cost) path from the district a_i to the district b_i to deliver products.

The route can go from the district to itself, some couriers routes can coincide (**and you have to count them independently**).

You can make at most one road to have cost zero (i.e. you choose at most one road and change its cost with 0).

Let $d(x, y)$ be the cheapest cost of travel between districts x and y.

Your task is to find the minimum total courier routes cost you can achieve, if you optimally select the some road and change its cost with 0. In other words, you have to find the minimum possible value of $\sum_{i=1}^{k} d(a_i, b_i)$ after applying the operation described above optimally.

Input

The first line of the input contains three integers n, m and k ($2 \le n \le 1000$; $n - 1 \le m \le min(1000, \frac{n(n-1)}{2})$; $1 \le k \le 1000$) — the number of districts, the number of roads and the number of courier routes.

The next m lines describe roads. The i-th road is given as three integers x_i, y_i and w_i ($1 \le x_i, y_i \le n$; $x_i \ne y_i$; $1 \le w_i \le 1000$), where x_i and y_i are districts the i-th road connects and w_i is its cost. It is guaranteed that there is some path between each pair of districts, so the city is connected. It is also guaranteed that there is at most one road between each pair of districts.

The next k lines describe courier routes. The i-th route is given as two integers a_i and b_i ($1 \le a_i, b_i \le n$) — the districts of the i-th route. The route can go from the district to itself, some couriers routes can coincide (**and you have to count them independently**).

Output

Print one integer — the **minimum** total courier routes cost you can achieve (i.e. the minimum value $\sum_{i=1}^{k} d(a_i, b_i)$, where $d(x, y)$ is the cheapest cost of travel between districts x and y) if you can make some (**at most one**) road cost zero.

Examples

input Copy

```
6 5 2
1 2 5
2 3 7
2 4 4
4 5 2
4 6 8
1 6
5 3
```

output Copy

```
22
```

input Copy

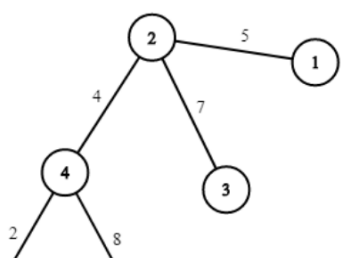
```
5 5 4
1 2 5
2 3 4
1 4 3
4 3 7
3 5 2
1 5
1 3
3 3
1 5
```

output Copy

```
13
```

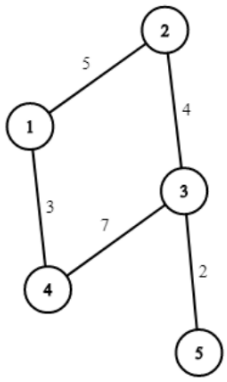
Note

The picture corresponding to the first example:



There, you can choose either the road $(2, 4)$ or the road $(4, 6)$. Both options lead to the total cost 22.

The picture corresponding to the second example:



There, you can choose the road $(3, 4)$. This leads to the total cost 13.

- 枚举，Floyd
- 我们先用Floyd函数求出图上任意两个点之间的最短路径，接着我们尝试去删除每一条边的cost，看看删除之后值的变化情况

```
#include <bits/stdc++.h>
using namespace std;
#define LL long long
#define sigma_size 30
#define max_size (int)(1e6+10)
#define _DEBUG1 freopen("input.txt", "r", stdin);freopen("output.txt", "w", stdout);
#define _DEBUG2 fclose("input.txt");fclose("output.txt");

int n , m , k;
int dp[1005][1005];
struct Edge {
    int u , v , w;
};
vector <Edge> edges;
int a[1005] , b[1005];
int main ()
{
    ios::sync_with_stdio(0);
    cin >> n >> m >> k;
    for ( int i = 1 ; i <= n ; i++ )
        for ( int j = 1 ; j <= n ; j++ )
        {
            if ( i != j )
                dp[i][j] = 1 << 28;
            else dp[i][j] = 0;
        }
    for ( int i = 1 ; i <= m ; i++ )
```

```
    {
        int x , y , w;
        cin >> x >> y >> w;
        edges.push_back({x,y,w});
        dp[x][y] = dp[y][x] = w;
    }
    for ( int i = 1 ; i <= k ; i++ )
        cin >> a[i] >> b[i];
    for ( int k = 1 ; k <= n ; k++ )
    for ( int i = 1 ; i <= n ; i++ )
    for ( int j = 1 ; j <= n ; j++ )
        dp[i][j] = min ( dp[i][j] , dp[i][k] + dp[k][j] );
    int minx = INT_MAX;
    for ( int w = 0 ; w < edges.size() ; w++ )
    {
        int x = edges[w].u , y = edges[w].v;
        int sum = 0 ;
        for ( int i = 1 ; i <= k ; i++ )
            sum += min ( dp[a[i]][b[i]] , min ( dp[a[i]][x] + dp[y][b[i]] , dp[a[i]][y] + dp[x][b[i]]
) ) ;
        minx = min ( minx , sum );
    }
    cout << minx << endl;
}
```