Codeforces Global Round 10

https://codeforces.ml/contest/1392/problems

A. Omkar and Password

A. Omkar and Password

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

Lord Omkar has permitted you to enter the Holy Church of Omkar! To test your worthiness, Omkar gives you a password which you must interpret!

A password is an array a of n positive integers. You apply the following operation to the array: pick any two adjacent numbers that are not equal to each other and replace them with their sum. Formally, choose an index i such that $1 \leq i < n$ and $a_i \neq a_{i+1}$, delete both a_i and a_{i+1} from the array and put $a_i + a_{i+1}$ in their place.

For example, for array $[7, 4, 3, 7]$ you can choose $i = 2$ and the array will become $[7, 4 + 3, 7] = [7, 7, 7]$. Note that in this array you can't apply this operation anymore.

Notice that one operation will decrease the size of the password by 1. What is the shortest possible length of the password after some number (possibly 0) of operations?

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 100$). Description of the test cases follows.

The first line of each test case contains an integer n ($1 \leq n \leq 2 \cdot 10^5$) — the length of the password.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^9$) — the initial contents of your password.

The sum of n over all test cases will not exceed $2 \cdot 10^5$.

Output

For each password, print one integer: the shortest possible length of the password after some number of operations.

Example

input Copy

```
2
4
2 1 3 1
2
420 420
```

output Copy

```
1
2
```

Note

In the first test case, you can do the following to achieve a length of 1:

Pick $i = 2$ to get $[2, 4, 1]$

Pick $i = 1$ to get $[6, 1]$

Pick $i = 1$ to get $[7]$

In the second test case, you can't perform any operations because there is no valid i that satisfies the requirements mentioned above.

- 结论题

- 如果数字全都一样，那么就不能够缩减；如果数字不一样，那就一定可以缩减成1

```
#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)

int ans[105];
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 = true;
                for ( int i = 1 ; i < n ; i++ )
                        if ( a[i] != a[i+1] )
                        {
                                flag = false;
                                break;
                        }
                if ( flag ) ans[cas] = n;
                else ans[cas] = 1;
        }
        for ( int i = 1 ; i <= T ; i++ )
                cout << ans[i ] << endl;
}
```

B. Omkar and Infinity Clock

B. Omkar and Infinity Clock

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

Being stuck at home, Ray became extremely bored. To pass time, he asks Lord Omkar to use his time bending power: Infinity Clock! However, Lord Omkar will only listen to mortals who can solve the following problem:

You are given an array a of n integers. You are also given an integer k. Lord Omkar wants you to do k operations with this array.

Define one operation as the following:

1. Set d to be the maximum value of your array.
2. For every i from 1 to n, replace a_i with $d - a_i$.

The goal is to predict the contents in the array after k operations. Please help Ray determine what the final sequence will look like!

Input

Each test contains multiple test cases. The first line contains the number of cases t ($1 \le t \le 100$). Description of the test cases follows.

The first line of each test case contains two integers n and k ($1 \le n \le 2 \cdot 10^5, 1 \le k \le 10^{18}$) – the length of your array and the number of operations to perform.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$) – the initial contents of your array.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each case, print the final version of array a after k operations described above.

Example

input Copy

```
3
2 1
-199 192
5 19
5 -1 4 2 0
1 2
69
```

output Copy

```
391 0
0 6 1 3 5
0
```

Note

In the first test case the array changes as follows:

- Initially, the array is $[-199, 192]$. $d = 192$.
- After the operation, the array becomes $[d - (-199), d - 192] = [391, 0]$.

- 模拟
- 先按照题目的意思重新构造一次整个数列，用一个数组保存；然后再重新构造一次数列，用另外一个数组保存
- 根据k的奇偶性可以很容易判断最后是哪一个数组

```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 a[max_size];
LL b[max_size];
int main ()
{
        ios::sync_with_stdio(0);
        int T ; cin >> T;
        while (T--)
        {
                LL n , k;
                cin >> n >> k;
                for ( int i = 1 ; i <= n ; i++ ) cin >> a[i];

                LL maxx = - (1ll << 60);
                for ( int i = 1 ; i <= n ; i++ )
                        maxx = max ( maxx , a[i] );
                for ( int i = 1 ; i <= n ; i++ )
                        a[i] = maxx - a[i];
                maxx = - (1ll << 60);
                for ( int i = 1 ; i <= n ; i++ )
                        maxx = max ( maxx , a[i] );
                for ( int i = 1 ; i <= n ; i++ )
                        b[i] = maxx - a[i];
                if ( n == 1 )
                        cout << "0" << endl;
                else if ( k & 1 )
                {
                        for ( int i = 1 ; i <= n ; i++ )
                                cout << a[i] << " ";
                        cout << endl;
                }
                else
                {
                        for ( int i = 1 ; i <= n ; i++ )
                                cout << b[i] << " ";
                        cout << endl;
                }
        }
}
```

C. Omkar and Waterslide

C. Omkar and Waterslide

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

Omkar is building a waterslide in his water park, and he needs your help to ensure that he does it as efficiently as possible.

Omkar currently has n supports arranged in a line, the i-th of which has height a_i. Omkar wants to build his waterslide from the right to the left, so his supports must be nondecreasing in height in order to support the waterslide. In 1 operation, Omkar can do the following: take any **contiguous subsegment** of supports which is **nondecreasing by heights** and add 1 to each of their heights.

Help Omkar find the minimum number of operations he needs to perform to make his supports able to support his waterslide!

An array b is a subsegment of an array c if b can be obtained from c by deletion of several (possibly zero or all) elements from the beginning and several (possibly zero or all) elements from the end.

An array $b_1, b_2, \ldots, b_n$ is called nondecreasing if $b_i \le b_{i+1}$ for every i from 1 to $n-1$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 100$). Description of the test cases follows.

The first line of each test case contains an integer n ($1 \le n \le 2 \cdot 10^5$) — the number of supports Omkar has.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$) — the heights of the supports.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output a single integer — the minimum number of operations Omkar needs to perform to make his supports able to support his waterslide.

Example

input Copy

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

output Copy

```
3
2
0
```

Note

The subarray with which Omkar performs the operation is bolded.

In the first test case:

- First operation:

  $[5, 3, \mathbf{2}, 5] \to [5, 3, \mathbf{3}, 5]$

- Second operation:

  $[5, \mathbf{3}, \mathbf{3}, 5] \to [5, \mathbf{4}, \mathbf{4}, 5]$

- Third operation:

  $[5, \mathbf{4}, \mathbf{4}, 5] \to [5, \mathbf{5}, \mathbf{5}, 5]$

In the third test case, the array is already nondecreasing, so Omkar does 0 operations.

- 思维，结论
- 遍历整个数组，如果当前的数大于等于上一个数，即a[i]>=a[i-1]，我们就不用操作。
  因为这个a[i]一定可以跟着上一个数一起上升，当不能再上升的时候，答案依然是上一个数的结果
- 如果a[i]< a[i-1],我们的ans就增加他们的差值，可以理解为a[i]需要先上升到和a[i-1]同样的高度之后跟着a[i-1]一起上升

```
#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 ans[105];
LL 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];
                LL res = 0;
                for ( int i = 2 ; i <= n ; i++ )
                {
                        if ( a[i] >= a[i-1] ) continue;
                        res += a[i-1] - a[i];
                }
                ans[cas] = res;
        }
        for ( int i = 1 ; i <= T ; i++ )
                cout << ans[i] << endl;
}
```

D. Omkar and Bed Wars

D. Omkar and Bed Wars

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

Omkar is playing his favorite pixelated video game, Bed Wars! In Bed Wars, there are n players arranged in a circle, so that for all j such that $2 \leq j \leq n$, player $j - 1$ is to the left of the player j, and player j is to the right of player $j - 1$. Additionally, player n is to the left of player 1, and player 1 is to the right of player n.

Currently, each player is attacking either the player to their left or the player to their right. This means that each player is currently being attacked by either 0, 1, or 2 other players. A key element of Bed Wars strategy is that if a player is being attacked by exactly 1 other player, then they should logically attack that player in response. If instead a player is being attacked by 0 or 2 other players, then Bed Wars strategy says that the player can logically attack either of the adjacent players.

Unfortunately, it might be that some players in this game are not following Bed Wars strategy correctly. Omkar is aware of whom each player is currently attacking, and he can talk to any amount of the n players in the game to make them instead attack another player — i. e. if they are currently attacking the player to their left, Omkar can convince them to instead attack the player to their right; if they are currently attacking the player to their right, Omkar can convince them to instead attack the player to their left.

Omkar would like all players to be acting logically. Calculate the minimum amount of players that Omkar needs to talk to so that after all players he talked to (if any) have changed which player they are attacking, all players are acting logically according to Bed Wars strategy.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 10^4$). The descriptions of the test cases follows.

The first line of each test case contains one integer n ($3 \leq n \leq 2 \cdot 10^5$) — the amount of players (and therefore beds) in this game of Bed Wars.

The second line of each test case contains a string s of length n. The j-th character of s is equal to `L` if the j-th player is attacking the player to their left, and `R` if the j-th player is attacking the player to their right.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output one integer: the minimum number of players Omkar needs to talk to to make it so that all players are acting logically according to Bed Wars strategy.

It can be proven that it is always possible for Omkar to achieve this under the given constraints.

Example

input Copy

```
5
4
RLRL
6
LRRRRL
8
RLLRRRLL
12
LLLLRRLRRRLL
5
RRRRR
```

output Copy

```
0
1
1
3
2
```

Note

In the first test case, players 1 and 2 are attacking each other, and players 3 and 4 are attacking each other. Each player is being attacked by exactly 1 other player, and each player is attacking the player that is attacking them, so all players are already being logical according to Bed Wars strategy and Omkar does not need to talk to any of them, making the answer 0.

In the second test case, not every player acts logically: for example, player 3 is attacked only by player 2, but doesn't attack him in response. Omkar can talk to player 3 to convert the attack arrangement to `LRLRRL`, in which you can see that all players are being logical according to Bed Wars strategy, making the answer 1.

- 结论
- 如果整个序列全都是一样的，例如:"RRRRRRR...",那么答案就是1+(n-1)/3
- 如果整个序列不一样，那我们可以看做是几个分段的序列，例
  如："LLRRRLLRRRR"我们可以把连续相同的几个序列拿出来并且构建出他们的数组"2 3 2 4"，我们很容易知道不能有3个相同的字母连在一起，也就是说我们每3个字母在一块的时候就一定要有一个变个方向，那么答案就是每段序列/3加起来
- 值得注意的是，首尾有可能是相等的，相等的时候我们不妨把最后一个加到第一个，最后一个变成0

```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)

int ans[10005];

int main ()
{
	ios::sync_with_stdio(0);
	int T ; cin >> T;
	for ( int cas = 1 ; cas <= T ; cas++ )
	{
		int n ; cin >> n;
		string s; cin >> s;
		s = ' ' + s;
		vector <int> v;
		for ( int i = 1 ; i <= n ; i++ )
		{
			int j = i;
			int cnt = 0;
			while ( s[j] == s[i] )
			{
				j++;
				cnt++;
			}
			i = j-1;
			v.push_back(cnt);
		}
		if ( s[1] == s[n] && v.size() > 1 )
		{
			v.back() += v.front();
			v.front() = 0;
		}
		int res = 0;
		if ( v.size() == 1 )
			res = 1 + ( v[0] - 1 ) / 3;
		else
			for ( int i = 0 ; i < v.size() ; i++ )
				res += v[i] / 3;
		ans[cas] = res;
	}
	for ( int i = 1 ; i <= T ; i++ )
		cout << ans[i] << endl;
```

```
}
```

E. Omkar and Duck

E. Omkar and Duck

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

This is an interactive problem.

Omkar has just come across a duck! The duck is walking on a grid with n rows and n columns ($2 \leq n \leq 25$) so that the grid contains a total of n^2 cells. Let's denote by (x, y) the cell in the x-th row from the top and the y-th column from the left. Right now, the duck is at the cell $(1, 1)$ (the cell in the top left corner) and would like to reach the cell (n, n) (the cell in the bottom right corner) by moving either down 1 cell or to the right 1 cell each second.

Since Omkar thinks ducks are fun, he wants to play a game with you based on the movement of the duck. First, for each cell (x, y) in the grid, you will tell Omkar a nonnegative integer $a_{x,y}$ not exceeding 10^{16}, and Omkar will then put $a_{x,y}$ uninteresting problems in the cell (x, y). After that, the duck will start their journey from $(1, 1)$ to (n, n). For each cell (x, y) that the duck crosses during their journey (including the cells $(1, 1)$ and (n, n)), the duck will eat the $a_{x,y}$ uninteresting problems in that cell. Once the duck has completed their journey, Omkar will measure their mass to determine the total number k of uninteresting problems that the duck ate on their journey, and then tell you k.

Your challenge, given k, is to exactly reproduce the duck's path, i. e. to tell Omkar precisely which cells the duck crossed on their journey. To be sure of your mastery of this game, Omkar will have the duck complete q different journeys ($1 \leq q \leq 10^3$). Note that all journeys are independent: at the beginning of each journey, the cell (x, y) will still contain $a_{x,y}$ uninteresting tasks.

Interaction

The interaction will begin with a line containing a single integer n ($2 \leq n \leq 25$), the amount of rows and columns in the grid. Read it.

Your program should then print n lines. The x-th line should contain n integers $a_{x,1}, a_{x,2}, \ldots, a_{x,n}$ satisfying $0 \leq a_{x,y} \leq 10^{16}$, where $a_{x,y}$ is the amount of uninteresting problems Omkar should place in the cell (x, y).

After that, you will first receive a single integer q, the amount of journeys that the duck will take. q queries will follow; each query will consist of a single line containing an integer k, the amount of uninteresting problems that the duck ate on that journey. After each query, given that you have determined that the duck visited the cells $(x_1, y_1), (x_2, y_2), \ldots, (x_{2n-1}, y_{2n-1})$ in that order (it should always be true that $(x_1, y_1) = (1, 1)$ and $(x_{2n-1}, y_{2n-1}) = (n, n)$), you should output $2n - 1$ lines so that the j-th line contains the two integers x_j, y_j.

Bear in mind that if the sum on your path is k, but your path is different from the actual hidden path, then your solution is still wrong!

After printing each line do not forget to output end of line and flush the output. Otherwise, you will get `Idleness limit exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

Hack Format

To hack, first output a line containing n and another line containing q. It must be true that $2 \leq n \leq 25$ and $1 \leq q \leq 1000$. Then, output the q journeys taken by the duck in the same format as described above: for each journey, given that the duck visited the cells $(x_1, y_1), (x_2, y_2), \ldots, (x_{2n-1}, y_{2n-1})$ in that order, you should output $2n - 1$ lines so that the j-th line contains the two integers x_j, y_j. It must be true that $(x_1, y_1) = (1, 1)$ and $(x_{2n-1}, y_{2n-1}) = (n, n)$. Additionally, for each j such that $2 \leq j \leq 2n - 1$, it must be true that $1 \leq x_j, y_j \leq n$ and either $(x_j, y_j) = (x_{j-1} + 1, y_{j-1})$ or $(x_j, y_j) = (x_{j-1}, y_{j-1} + 1)$.

Example

input Copy

```
4



3
23




26
```

- 交互题
- 我们主要是可以把它的路线全都枚举出来，用一个二进制位数表示，然后我们根据这个二进制位数去构造不同的和

```
#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 main ()
{
    ios::sync_with_stdio(0);
    int n;
    cin >> n;
    LL ans[26][26];
    memset ( ans , 0  ,sizeof(ans) );
    for ( int i = 0  ; i < n ; i++ )
    {
        for ( int j = 0 ; j < n ; j++ )
        {
            if ((i-j+n)&2ll) ans[i][j] = (1ll<<(i+j));
            cout << ans[i][j] << " \n"[j==n-1];
        }
    }
    cout << flush;
    int q ; cin >> q;
    for ( int i = 1 ; i <= q ; i++ )
    {
        LL sum ; cin >> sum;
        cout << "1 1" ;
        int row = 0 , col = 0;
        for ( int diag = 0 ; diag < 2*n-2 ; diag++ )
        {
            LL sh = sum&(1ll<<(diag+1));
            if ( row + 1 < n && ans[row+1][col] == sh ) row++;
            else col++;
            cout << " " << row+1 << " " << col+1 ;
        }
        cout << endl << flush;
    }
}
```

F. Omkar and Landslide

F. Omkar and Landslide

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

Omkar is standing at the foot of Celeste mountain. The summit is n meters away from him, and he can see all of the mountains up to the summit, so for all $1 \leq j \leq n$ he knows that the height of the mountain at the point j meters away from himself is h_j meters. It turns out that for all j satisfying $1 \leq j \leq n-1$, $h_j < h_{j+1}$ (meaning that heights are strictly increasing).

Suddenly, a landslide occurs! While the landslide is occurring, the following occurs: every minute, if $h_j + 2 \leq h_{j+1}$, then one square meter of dirt will slide from position $j+1$ to position j, so that h_{j+1} is decreased by 1 and h_j is increased by 1. These changes occur simultaneously, so for example, if $h_j + 2 \leq h_{j+1}$ and $h_{j+1} + 2 \leq h_{j+2}$ for some j, then h_j will be increased by 1, h_{j+2} will be decreased by 1, and h_{j+1} will be both increased and decreased by 1, meaning that in effect h_{j+1} is unchanged during that minute.

The landslide ends when there is no j such that $h_j + 2 \leq h_{j+1}$. Help Omkar figure out what the values of $h_1, \ldots, h_n$ will be after the landslide ends. It can be proven that under the given constraints, the landslide will always end in finitely many minutes.

Note that because of the large amount of input, it is recommended that your code uses fast IO.

Input

The first line contains a single integer n ($1 \leq n \leq 10^6$).

The second line contains n integers $h_1, h_2, \ldots, h_n$ satisfying $0 \leq h_1 < h_2 < \cdots < h_n \leq 10^{12}$ — the heights.

Output

Output n integers, where the j-th integer is the value of h_j after the landslide has stopped.

Example

input Copy

```
4
2 6 7 8
```

output Copy

```
5 5 6 7
```

Note

Initially, the mountain has heights $2, 6, 7, 8$.

In the first minute, we have $2 + 2 \leq 6$, so 2 increases to 3 and 6 decreases to 5, leaving $3, 5, 7, 8$.

In the second minute, we have $3 + 2 \leq 5$ and $5 + 2 \leq 7$, so 3 increases to 4, 5 is unchanged, and 7 decreases to 6, leaving $4, 5, 6, 8$.

In the third minute, we have $6 + 2 \leq 8$, so 6 increases to 7 and 8 decreases to 7, leaving $4, 5, 7, 7$.

In the fourth minute, we have $5 + 2 \leq 7$, so 5 increases to 6 and 7 decreases to 6, leaving $4, 6, 6, 7$.

In the fifth minute, we have $4 + 2 \leq 6$, so 4 increases to 5 and 6 decreases to 5, leaving $5, 5, 6, 7$.

In the sixth minute, nothing else can change so the landslide stops and our answer is $5, 5, 6, 7$.

- 结论题，模拟
- 因为输入的时候是严格递增的，我们假设$h[l,r]$差值都是1，并且$h[r]+2==h[r+1]$，通过模拟我们可以发现，最后的结果就是$h[l]++, h[r+1]--$，并且贡献了一对相同的高度$h[l]==h[l+1]$
- 如果原本就有一对相同的高度$h[l]==h[l+1]$,并且$h[r+1]-h[r]==2$，那就相当于上面的一步操作是从$h[l+1]$开始的，这样一来，我们不仅会拆散原来的那一对相同的高度，还会重新组建一对相同的高度$h[l+1]==h[l+2]$

- 综上所述，如果所有元素没有相同的高度，那么一次执行之后可能会产生一对
  如果一开始就有一对相同的高度，那么一次执行之后也可能会拆散一对
  但是这都不重要，因为由此我们可以推出最后的序列应该是差值为1的等差数列，最多有一对数字高度相同。
- 那我们就可以从$h[1]$开始构造等差数列，不妨一开始的时候使得所有的元素全都-h[1]-i+1也就是把他们太高的那一部分削掉并且累加起来，最后把我们累加的数sum/n分配到每个数当中去，如果分配后还有剩余，我们就从h[1]开始一个一个+1就可以了

```
#include <bits/stdc++.h>
using namespace std;
#define LL long long
#define ULL unsigned long long
#define sigma_size 30
#define max_size (int)(1e6+10)
#define MAX_SIZE (int)(4e6+7)

LL n ;
LL h[max_size] , res[max_size];
LL sum = 0;
int main ()
{
        ios::sync_with_stdio(0);
        cin >> n;
        for ( int i = 1 ; i <= n ; i++ )
        {
                cin >> h[i];
                sum += h[i] - h[1] - i + 1;
        }
        for ( int i = 1 ; i <= n ; i++ )
                res[i] = h[1] + i - 1 + sum/n;
        for ( int i = 1 ; i <= sum%n ; i++ )
                res[i]++;
        for ( int i =1 ; i <= n ; i++ )
                cout << res[i] << " ";
        cout << endl;
}
```

G. Omkar and Pies

G. Omkar and Pies

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

Omkar has a pie tray with k ($2 \leq k \leq 20$) spots. Each spot in the tray contains either a chocolate pie or a pumpkin pie. However, Omkar does not like the way that the pies are currently arranged, and has another ideal arrangement that he would prefer instead.

To assist Omkar, n elves have gathered in a line to swap the pies in Omkar's tray. The j-th elf from the left is able to swap the pies at positions a_j and b_j in the tray.

In order to get as close to his ideal arrangement as possible, Omkar may choose a contiguous subsegment of the elves and then pass his pie tray through the subsegment starting from the left. However, since the elves have gone to so much effort to gather in a line, they request that Omkar's chosen segment contain at least m ($1 \leq m \leq n$) elves.

Formally, Omkar may choose two integers l and r satisfying $1 \leq l \leq r \leq n$ and $r - l + 1 \geq m$ so that first the pies in positions a_l and b_l will be swapped, then the pies in positions a_{l+1} and b_{l+1} will be swapped, etc. until finally the pies in positions a_r and b_r are swapped.

Help Omkar choose a segment of elves such that the amount of positions in Omkar's final arrangement that contain the same type of pie as in his ideal arrangement is the maximum possible. **Note that since Omkar has a big imagination, it might be that the amounts of each type of pie in his original arrangement and in his ideal arrangement do not match**.

Input

The first line contains three integers n, m, and k ($1 \leq m \leq n \leq 10^6$ and $2 \leq k \leq 20$) — the number of elves, the minimum subsegment length, and the number of spots in Omkar's tray respectively.

The second and third lines each contain a string of length k consisting of 0s and 1s that represent initial arrangement of pies and ideal arrangement of pies; the j-th character in each string is equal to 0 if the j-th spot in the arrangement contains a chocolate pie and is equal to 1 if the j-th spot in the arrangement contains a pumpkin pie. It is not guaranteed that the two strings have the same amount of 0s or the same amount of 1s.

n lines follow. The j-th of these lines contains two integers a_j and b_j ($1 \leq a_j, b_j \leq k$, $a_j \neq b_j$) which indicate that the j-th elf from the left can swap the pies at positions a_j and b_j in the tray.

Output

Output two lines.

The first line should contain a single integer s ($0 \leq s \leq k$) equal to the amount of positions that contain the same type of pie in Omkar's final arrangement and in Omkar's ideal arrangement; s should be the maximum possible.

The second line should contain two integers l and r satisfying $1 \leq l \leq r \leq n$ and $r - l + 1 \geq m$, indicating that Omkar should pass his tray through the subsegment $l, l+1, \ldots, r$ to achieve a final arrangement with s positions having the same type of pie as his ideal arrangement.

If there are multiple answers you may output any of them.

Examples

input Copy

```
4 2 5
11000
00011
1 3
3 5
4 2
3 4
```

output Copy

```
5
1 3
```

input Copy

```
4 3 5
11000
00011
1 3
1 5
2 4
1 5
```

output Copy

```
3
1 4
```

Note

In the first test case, the swaps will go like this:

- Swap 1 and 3: `11000` becomes `01100`
- Swap 3 and 5: `01100` becomes `01001`

- 状态压缩dp
- 题目大意：你现在有两个字符串s,t
  你还有n个小精灵，某个小精灵i可以变更s的两个坐标$a[i],b[i]$,使得这两个数交换位置
  你现在需要选定一个长度不小于m的区间[l,r]，使得操作完$(a[l],b[l])(a[l+1],b[l+1])...(a[r],b[r])$ 之后两个串的相似度最大
- 由k的范围可知我们可以把字符串变成一种状态state
- 由于我们交换的时候状态很难从l转移到$l+1$，那么其实我们可以考虑再s执行完l操作之后，我们让t也执行以下l操作，这样我们就可以顺利的完成状态转移了
- 我们用$dp[0][state]$表示s串state状态的时候需要用到的最靠左的l
  用$dp[1][state]$表示t串state状态的时候需要用到的最靠右的l
  如果$dp[1][state]-dp[0][state]>=m$那么$(dp[0][state]+1,dp[1][state])$就是我们要的区间
- 但是我们有可能并不能让这两个串完全相同，这个时候我们就得考虑一下相似度的问题了，设$o1$为s串中1的数目$o2$为t串中1的数目，$same$为两个串共同的1的数目，x为不同元素的数目，y为相似度，我们有：

$$o1+o2-2*same=x=k-y$$

- 要使得y最大，我们就是要让same最大，在本题中我们用state来表示**共同的1**的状态。于此同时，我们枚举state的子集，更新他们的状态。因此我们在枚举state的状态的时候是倒序枚举的

```cpp
#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 o1 , o2;
int x , y;
string s , t;
int p[max_size];
int dp[2][1<<21];

void f ( int& x , int& y )
{
        x = y = 0;
        for ( int i = 0 ; i < k ; i++ )
        {
                if ( s[i] == '1' ) x |= ( 1 << p[i] );
                if ( t[i] == '1' ) y |= ( 1 << p[i] );
        }
}
int main ()
{
        ios::sync_with_stdio(0);
        cin >> n >> m >> k;
        cin >> s >> t;
        for ( int i = 0 ; i < k ; i++ )
        {
                p[i] = i;
                o1 += ( s[i] == '1' );
                o2 += ( t[i] == '1' );
        }
        memset ( dp[0] , 0x3f3f3f3f , sizeof(dp[0]) );
        memset ( dp[1] , 0 , sizeof(dp[1]) );
        f(x,y) ; dp[0][x] = 0 , dp[1][y] = 0;
        for ( int i = 1 ; i <= n ; i++ )
        {
                int a , b ; cin >> a >> b;
                a-- , b--;
                swap ( p[a] , p[b] );
                f(x,y);
                dp[0][x] = min ( dp[0][x] , i );
```

```
                dp[1][y] = max ( dp[1][y] , i );
        }

        int ans = 0;
        int ansl = 1 , ansr = 1;
        for ( int i = (1<<k)-1 ; i >= 0 ; i-- )
        {
                int tmp = k - (o1 + o2 - 2*__builtin_popcount(i));
                if ( tmp > ans && dp[1][i] - dp[0][i] >= m )
                {
                        ans = tmp;
                        ansl = dp[0][i] + 1;
                        ansr = dp[1][i];
                }
                for ( int j = 0 ; j < k ; j++ )
                {
                        if ( i >> j & 1 )
                        {
                                dp[0][i^(1<<j)] = min ( dp[0][i] , dp[0][i^(1<<j)]);
                                dp[1][i^(1<<j)] = max ( dp[1][i] , dp[1][i^(1<<j)]);
                        }
                }
        }
        cout << ans << endl;
        cout << ansl << " " << ansr << endl;
}
```