Contents

.Vscode

Additional Problems I

Advertisement (gready, The nearest smaller element to the left)

Time limit: 1.00 s **Memory limit:** 512 MB

A fence consists of n vertical boards. The width of each board is 1 and their heights may vary. You want to attach a rectangular advertisement to the fence. What is the maximum area of such an advertisement?

Input

The first input line contains an integer n: the width of the fence.

After this, there are n integers $k_1, k_2, \ldots, k_n$: the height of each board.

Output

Print one integer: the maximum area of an advertisement.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq k_i \leq 10^9$

Example

Input:

```
8
4 1 5 3 3 2 4 1
```

Output:

```
10
```

Solution

```
const int maxN = 2e5+5;
int N, l[maxN], r[maxN];
ll best, a[maxN];
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++)
        scanf("%lld", &a[i]);
    for(int i = 1; i <= N; i++){
        l[i] = i-1;
        while(a[l[i]] >= a[i])
            l[i] = l[l[i]];
    }
    for(int i = N; i >= 1; i--){
        r[i] = i+1;
        while(a[r[i]] >= a[i])
            r[i] = r[r[i]];
    }
    for(int i = 1; i <= N; i++)
        best = max(best, (r[i]-l[i]-1)*a[i]);
    printf("%lld\n", best);
}
```

Bit Inversions ((segtree, dp) lognest common substr after each update)

Time limit: 1.00 s **Memory limit:** 512 MB

There is a bit string consisting of n bits. Then, there are some changes that invert one given bit. Your task is to report, after each change, the length of the longest substring whose each bit is the same.

Input

The first input line has a bit string consisting of n bits. The bits are numbered $1, 2, \ldots, n$.

The next line contains an integer m: the number of changes.

The last line contains m integers $x_1, x_2, \ldots, x_m$ describing the changes.

Output

After each change, print the length of the longest substring whose each bit is the same.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq m \leq 2 \cdot 10^5$
- $1 \leq x_i \leq n$

Example

Input:

```
001011
3
3 2 5
```

Output:

```
4 2 3
```

Explanation: The bit string first becomes 000011, then 010011, and finally 010001.

Solution

```
const int maxN = 2e5+5;
const int SIZE = 4*maxN;
int N, M, mp[maxN], lo[SIZE], hi[SIZE], dp[3][SIZE];
char S[maxN];
int len(int i){
    return hi[i]-lo[i]+1;
}
int combine(int a, int b){
    return ((a<0)^(b<0)) ? a : a+b;
}
bool allsame(int i){
    return abs(dp[2][i]) == len(i);
}
void pull(int i){
    dp[0][i] = (allsame(2*i) ? combine(dp[0][2*i], dp[0][2*i+1]) : dp[0][2*i]);
    dp[1][i] = (allsame(2*i+1) ? combine(dp[1][2*i+1], dp[1][2*i]) : dp[1][2*i+1]);
    dp[2][i] = max(abs(combine(dp[1][2*i], dp[0][2*i+1])), max(dp[2][2*i], dp[2][2*i+1]));
}
void init(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r){
        mp[l] = i;
        dp[0][i] = dp[1][i] = (S[l-1] == '0' ? -1 : 1);
        dp[2][i] = abs(dp[0][i]);
        return;
    }
    int m = (l+r)/2;
    init(2*i, l, m);
    init(2*i+1, m+1, r);
    pull(i);
}
void update(int idx){
    int i = mp[idx];
    dp[0][i] *= -1;
    dp[1][i] = dp[0][i];
    i >>= 1;
    while(i){
        pull(i);
```

```
        i >>= 1;
    }
}
int query(){
    return dp[2][1];
}
int main(){
    scanf("%s %d", S, &M);
    N = (int) strlen(S);
    init(1, 1, N);
    for(int i = 0, x; i < M; i++){
        scanf("%d", &x);
        update(x);
        printf("%d%c", query(), (" \n")[i==M-1]);
    }
}
```

Bubble Sort Rounds I (number of rounds to sort with bubble sort)

Time limit: 1.00 s **Memory limit:** 512 MB

Bubble sort is a sorting algorithm that consists of a number of rounds. On each round the algorithm scans the array from left to right and swaps any adjacent elements that are in the wrong order. Given an array of n integers, calculate the number of bubble sort rounds needed to sort the array.

Input

The first line has an integer n: the array size.

The next line has n integers $x_1, x_2, \ldots, x_n$: the array contents.

Output

Print one integer: the number of rounds.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

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

Output:

```
3
```

Explanation: Bubble sort needs three rounds to sort this array. The array contents after each round are $[2, 3, 1, 4, 4]$, $[2, 1, 3, 4, 4]$, and $[1, 2, 3, 4, 4]$.

Solution

```
typedef pair<int,int> pii;
const int maxN = 2e5+1;
int N, x[maxN];
vector<pii> sorted;
map<int,int> inv;
```

```
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d", &x[i]);
        sorted.push_back({x[i], i});
    }
    sort(sorted.begin(), sorted.end(), greater<pii>());
    int worst_dist = 0;
    for(int i = 0; i < N; i++){
        int targ = N-i-1;
        int start = sorted[i].second;
        worst_dist = max(worst_dist, start - targ);
    }
    printf("%d\n", worst_dist);
}
```

Cyclic Array (divide into min num subarrays with at most k sum of each)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a cyclic array consisting of n values. Each element has two neighbors; the elements at positions n and 1 are also considered neighbors. Your task is to divide the array into subarrays so that the sum of each subarray is at most k. What is the minimum number of subarrays?

Input

The first input line contains integers n and k.

The next line has n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

There is always at least one division (i.e., no value in the array is larger than k).

Output

Print one integer: the minimum number of subarrays.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq x_i \leq 10^9$
- $1 \leq k \leq 10^{18}$

Example

Input:

```
8 5
2 2 2 1 3 1 2 1
```

Output:

```
3
```

Explanation: We can create three subarrays: $[2, 2, 1]$, $[3, 1]$, and $[2, 1, 2]$ (remember that the array is cyclic).

Solution

```
const int maxN = 2e5+1;
int N, last[maxN];
ll K, x[2*maxN];
int main(){
    scanf("%d %lld", &N, &K);
```

```
    for(int i = 0; i < N; i++){
        scanf("%lld", &x[i]);
        x[i+N] = x[i];
    }
    int r = 0;
    ll sum = 0;
    for(int l = 0; l < N; l++){
        while(r < l+N && sum + x[r] <= K){
            sum += x[r];
            r++;
        }
        last[l] = r % N;
        sum -= x[l];
    }
    int opt = 0;
    for(int i = 0; i < N; i++)
        opt = last[opt];
    int cnt = 1;
    sum = x[opt];
    for(int i = opt+1; i < N+opt; i++){
        if(sum + x[i] <= K) sum += x[i];
        else {
            sum = x[i];
            cnt++;
        }
    }
    printf("%d\n", cnt);
}
```

List of Sums (sum of each element pair)

Time limit: 1.00 s **Memory limit:** 512 MB

List A consists of n positive integers, and list B contains the sum of each element pair of list A. For example, if $A = [1, 2, 3]$, then $B = [3, 4, 5]$, and if $A = [1, 3, 3, 3]$, then $B = [4, 4, 4, 6, 6, 6]$. Given list B, your task is to reconstruct list A.

Input

The first input line has an integer n: the size of list A.

The next line has $\frac{n(n-1)}{2}$ integers: the contents of list B.

You can assume that there is a list A that corresponds to the input, and each value in A is between $1 \ldots k$.

Output

Print n integers: the contents of list A.

You can print the values in any order. If there are more than one solution, you can print any of them.

Constraints

- $3 \le n \le 100$
- $1 \le k \le 10^9$

Example

Input:

```
4
4 4 4 6 6 6
```

Output:

```
1 3 3 3
```

Explanation: In this case list A can be either $[1, 3, 3, 3]$ or $[2, 2, 2, 4]$ and both solutions are accepted.

Solution

```
const int maxN = 101, maxM = 4951;
int N, M;
multiset<ll> Bvals;
ll A[maxN], B[maxM];
bool check(ll a0){
    A[0] = a0;
    Bvals.clear();
    for(int i = 0; i < M; i++)
        Bvals.insert(B[i]);
    for(int i = 1; i < N; i++){
        for(int j = 0; j < i-1; j++){
            if(Bvals.find(A[j]+A[i-1]) == Bvals.end())
                return false;
            Bvals.erase(Bvals.lower_bound(A[j]+A[i-1]));
        }
        A[i] = *Bvals.begin() - A[0];
    }
    // Confirm that the last element, A[N-1], works. We desire
    // the sum with the other A[i] to produce the remaining Bvals
    for(int j = 0; j < N-1; j++){
        if(Bvals.find(A[j]+A[N-1]) == Bvals.end())
            return false;
        Bvals.erase(Bvals.lower_bound(A[j]+A[N-1]));
    }
    return true;
}
int main(){
    scanf("%d", &N);
    M = N*(N-1)/2;
    for(int i = 0; i < M; i++)
        scanf("%lld", &B[i]);
    sort(B, B+M);
    for(int i = 1; i < M; i++){
        ll a0 = (B[0] + B[1] - B[i])/2;
        if(1 <= a0 && a0 <= B[0]/2 && check(a0)){
            for(int j = 0; j < N; j++)
                printf("%lld%c", A[j], (" \n")[j==N-1]);
            return 0;
        }
    }
    printf("IMPOSSIBLE\n");
}
```

Maximum Building I (the largest rectangular area of empty cells in a grid)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a map of a forest where some squares are empty and some squares have trees. What is the maximum area of a rectangular building that can be placed in the forest so that no trees must be cut down?

Input

The first input line contains integers n and m: the size of the forest.

After this, the forest is described. Each square is empty (.) or has trees (*).

Input

Print the maximum area of a rectangular building.

Constraints

- $1 \leq n, m \leq 1000$

Example

Input:

```
4 7
...*.*.
.*.....
.......
......*
```

Output:

```
12
```

Solution

```
const int maxN = 1002;
int N, M, best, h[maxN], l[maxN], r[maxN];
char S[maxN][maxN];
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0; i < N; i++)
        scanf(" %s", S[i]);
    h[0] = h[M+1] = -1;
    for(int i = 1; i <= N; i++){
        for(int j = 1; j <= M; j++)
            h[j] = (S[i-1][j-1] == '*' ? 0 : h[j]+1);
        int smaller;
        for(int j = 1; j <= M; j++){
            smaller = j-1;
            while(h[smaller] >= h[j])
                smaller = l[smaller];
            l[j] = smaller;
        }
        for(int j = M; j >= 1; j--){
            smaller = j+1;
            while(h[smaller] >= h[j])
                smaller = r[smaller];
            r[j] = smaller;
        }
        for(int j = 1; j <= M; j++){
            int base = r[j]-l[j]-1;
            best = max(best, base * h[j]);
```

```
        }
    }
    printf("%d\n", best);
}
```

Multiplication Table ((binary search) there are at least $(n*n+1)/2$ elements less than or equal to x)

Time limit: 1.00 s **Memory limit:** 512 MB

Find the middle element when the numbers in an $n \times n$ multiplication table are sorted in increasing order. It is assumed that n is odd. For example, the 3×3 multiplication table is as follows:

1	2	3
2	4	6
3	6	9

The numbers in increasing order are $[1, 2, 2, 3, 3, 4, 6, 6, 9]$, so the answer is 3.

Input

The only input line has an integer n.

Output

Print one integer: the answer to the task.

Constraints

- $1 \leq n < 10^6$

Example

Input:

```
3
```

Output:

```
3
```

Solution

```
ll N;
ll f(ll x){
    ll cnt = 0;
    for(ll i = 1; i <= N; i++)
        cnt += min(N, x/i);
    return cnt;
}
int main(){
    scanf("%lld", &N);
    ll lo = 0, hi = N*N;
    while(hi-lo > 1){
        ll mid = lo + (hi-lo)/2;
        if(f(mid) < (N*N+1)/2)  lo = mid;
        else                    hi = mid;
    }
    printf("%lld\n", lo+1);
}
```

Pyramid Array (increasing then decreasing using bit tree)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array consisting of n distinct integers. On each move, you can swap any two adjacent values. You want to transform the array into a pyramid array. This means that the final array has to be first increasing and then decreasing. It is also allowed that the final array is only increasing or decreasing. What is the minimum number of moves needed?

Input

The first input line has an integer n: the size of the array.

The next line has n distinct integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print one integer: the minimum number of moves.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

```
4
2 1 5 3
```

Output:

```
1
```

Explanation: You may swap the first two values which creates a pyramid array $[1, 2, 5, 3]$.

Solution

```
typedef pair<int,int> pii;
const int maxN = 2e5+1;
int N, ds[maxN];
pii X[maxN];
void update(int idx){
    for(int i = idx; i <= N; i += -i&i) ds[i]++;
}
int query(int idx){
    int cnt = 0;
    for(int i = idx; i; i -= -i&i)
        cnt += ds[i];
    return cnt;
}
int main(){
    scanf("%d", &N);
    for(int i = 0, x; i < N; i++){
        scanf("%d", &x);
        X[i] = {x, i+1};
    }
    sort(X, X+N, [](pii a, pii b){
        return a.first > b.first;
    });
```

```
    long long ans = 0;
    for(int k = 0; k < N; k++){
        int idx = X[k].second;
        int l = query(idx);
        ans += min(l, k-l);
        update(idx);
    }
    printf("%lld\n", ans);
}
```

Shortest Subsequence(shortest string not a subsequence)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a DNA sequence consisting of characters A, C, G, and T. Your task is to find the shortest DNA sequence that is not a subsequence of the original sequence.

Input

The only input line contains a DNA sequence with n characters.

Output

Print the shortest DNA sequence that is not a subsequence of the original sequence. If there are several solutions, you may print any of them.

Constraints

- $1 \le n \le 10^6$

Example

Input:

```
ACGTACGT
```

Output:

```
AAA
```

Solution

```
const int maxN = 1e6+5;
int N, cnt;
char S[maxN], ch[4] = {'A', 'C', 'G', 'T'};
bool has[4];
vector<int> segs;
map<char,int> mp;
void init(){
    mp['A'] = 0;
    mp['C'] = 1;
    mp['G'] = 2;
    mp['T'] = 3;
}
int main(){
    init();
    scanf(" %s", S);
    N = (int) strlen(S);
    for(int i = 0; i < N; i++){
        int c = mp[S[i]];
```

```
        if(!has[c]){
            has[c] = true;
            cnt++;
            if(cnt == 4){
                fill(has, has+4, false);
                segs.push_back(i);
                cnt = 0;
            }
        }
    }
    for(int i : segs)
        printf("%c", S[i]);
    for(int i = 0; i < 4; i++){
        if(!has[i]){
            printf("%c\n", ch[i]);
            return 0;
        }
    }
}
```

Sorting Methods (steps to sort a permutation using four different sorting methods. (dsu ,BIT))

Time limit: 1.00 s **Memory limit:** 512 MB

Here are some possible methods using which we can sort the elements of an array in increasing order: - At each step, choose two adjacent elements and swap them. - At each step, choose any two elements and swap them. - At each step, choose any element and move it to another position. - At each step, choose any element and move it to the front of the array. Given a permutation of numbers $1, 2, \ldots, n$, calculate the minimum number of steps to sort the array using the above methods.

Input

The first input line contains an integer n.

The second line contains n integers describing the permutation.

Output

Print four numbers: the minimum number of steps using each method.

Constraints

- $1 \le n \le 2 \cdot 10^5$

Example

Input:

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

Output:

```
20 6 5 6
```

Solution

```
const int maxN = 2e5+5;
int N, x[maxN], ds[maxN];
```

```
ll bit[maxN], ans[4];
set<int> S;
set<int>::iterator it;
int find(int u){
    if(ds[u] < 0)   return u;
    ds[u] = find(ds[u]);
    return ds[u];
}
bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v)  return false;
    if(ds[u] < ds[v])   swap(u, v);
    ds[v] += ds[u];
    ds[u] = v;
    return true;
}
void update(int idx, int val){
    for(int i = idx; i < maxN; i += -i&i)
        bit[i] += val;
}
ll query(int idx){
    int sum = 0;
    for(int i = idx; i > 0; i -= -i&i)
        sum += bit[i];
    return sum;
}
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++)
        scanf("%d", &x[i]);
    int K = N;
    ans[3] = N;
    fill(ds+1, ds+N+1, -1);
    for(int i = 1; i <= N; i++){
        ans[0] += (i-query(x[i])-1);
        update(x[i], 1);
        if(!merge(i, x[i]))
            ans[1] += (-ds[find(i)]-1);
        S.insert(x[i]);
        it = S.lower_bound(x[i]);
        if(++it != S.end())
            S.erase(it);
        if(x[N-i+1] == K){
            K--;
            ans[3]--;
        }
    }
    ans[2] = N - (int) S.size();
    for(int i = 0; i < 4; i++)
        printf("%lld%c", ans[i], (" \n")[i==3]);
}
```

Special Substrings(counting substrings where each character appears the same number of times (dp,freq arr))

Time limit: 1.00 s **Memory limit:** 512 MB

A substring is called special if every character that appears in the string appears the same number of times in the substring. Your task is to count the number of special substrings in a given string.

Input

The only input line has a string of length n. Every character is between a...z.

Output

Print one integer: the number of special substrings.

Constraints

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

Example

Input:

```
abccabab
```

Output:

```
5
```

Explanation: The special substrings are abc, cab, abccab, bccaba and ccabab.

Solution

```
const int maxN = 2e5+1;
int N, M;
char S[maxN];
bool exists[26];
map<char,int> ch;
map<vector<int>,ll> dp;
bool containsEach(vector<int> V){
    for(int i = 0; i < (int) V.size(); i++)
        if(V[i] <= 0)
            return false;
    return true;
}
int main(){
    scanf(" %s", S);
    N = (int) strlen(S);
    for(int i = 0; i < N; i++){
        int c = (int) (S[i] - 'a');
        if(!exists[c]){
            ch[S[i]] = M++;
            exists[c] = true;
        }
    }
    ll ans = 0;
    vector<int> freq(M);
    dp[freq]++;
    for(int i = 0; i < N; i++){
        int c = ch[S[i]];
        freq[c]++;
        if(containsEach(freq))
            for(int j = 0; j < M; j++)
                freq[j]--;
```

```
        ans += dp[freq];
        dp[freq]++;
    }
    printf("%lld\n", ans);
}
```

Stack Weights (segment tree which stack if heavier filled with reltive sized coins)

Time limit: 1.00 s **Memory limit:** 512 MB

You have n coins, each of which has a distinct weight. There are two stacks which are initially empty. On each step you move one coin to a stack. You never remove a coin from a stack. After each move, your task is to determine which stack is heavier (if we can be sure that either stack is heavier).

Input

The first input line has an integer n: the number of coins. The coins are numbered $1, 2, \ldots, n$. You know that coin i is always heavier than coin $i-1$, but you don't know their exact weights.

After this, there are n lines that describe the moves. Each line has two integers c and s: move coin c to stack s ($1 =$ left, $2 =$ right).

Output

After each move, print $<$ if the right stack is heavier, $>$ if the left stack is heavier, and ? if we can't know which stack is heavier.

Constraints

- $1 \le n \le 2 \cdot 10^5$

Example

Input:

```
3
2 1
3 2
1 1
```

Output:

```
>
<
?
```

Explanation: After the last move, if the coins are $[2, 3, 4]$, the left stack is heavier, but if the coins are $[1, 2, 5]$, the right stack is heavier.

Solution

```
const int maxN = 2e5+1, SIZE = 4*maxN;
const int INF = 0x3f3f3f3f;
int N, lo[SIZE], hi[SIZE], delta[SIZE], mx[SIZE], mn[SIZE];
void push(int i){
    if(delta[i] != 0){
        delta[2*i] += delta[i];
        delta[2*i+1] += delta[i];
        delta[i] = 0;
    }
}
```

```
void pull(int i){
    mn[i] = min(mn[2*i]+delta[2*i], mn[2*i+1]+delta[2*i+1]);
    mx[i] = max(mx[2*i]+delta[2*i], mx[2*i+1]+delta[2*i+1]);
}
void init(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r)  return;
    int m = l+(r-l)/2;
    init(2*i, l, m);
    init(2*i+1, m+1, r);
    pull(i);
}
void increment(int i, int l, int r, int v){
    if(l > hi[i] || r < lo[i])  return;
    if(l <= lo[i] && hi[i] <= r){
        delta[i] += v; return;
    }
    push(i);
    increment(2*i, l, r, v);
    increment(2*i+1, l, r, v);
    pull(i);
}
int minimum(int i, int l, int r){
    if(l > hi[i] || r < lo[i])      return INF;
    if(l <= lo[i] && hi[i] <= r)    return mn[i]+delta[i];
    push(i);
    int lmin = minimum(2*i, l, r);
    int rmin = minimum(2*i+1, l, r);
    pull(i);
    return min(lmin, rmin);
}
int maximum(int i, int l, int r){
    if(l > hi[i] || r < lo[i])      return -INF;
    if(l <= lo[i] && hi[i] <= r)    return mx[i]+delta[i];
    push(i);
    int lmax = maximum(2*i, l, r);
    int rmax = maximum(2*i+1, l, r);
    pull(i);
    return max(lmax, rmax);
}
void place_coin(int c, int s){
    int v = (s == 1 ? -1 : 1);
    increment(1, 1, c, v);
}
char query_stacks(){
    int totmin = minimum(1, 1, N);
    int totmax = maximum(1, 1, N);
    if(totmin >= 0 && totmax <= 0)  return '?';
    else if(totmin >= 0)            return '<';
    else if(totmax <= 0)            return '>';
    else                            return '?';
}
int main(){
    scanf("%d", &N);
    init(1, 1, N);
    for(int i = 0, c, s; i < N; i++){
```

```
        scanf("%d %d", &c, &s);
        place_coin(c, s);
        printf("%c\n", query_stacks());
    }
}
```

Swap Game (number of swaps to sort a grid (bfs))

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a 3×3 grid containing the numbers $1, 2, \ldots, 9$. Your task is to perform a sequence of moves so that the grid will look like this:

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

On each move, you can swap the numbers in any two adjacent squares (horizontally or vertically). What is the minimum number of moves required?

Input

The input has three lines, and each of them has three integers.

Output

Print one integer: the minimum number of moves.

Example

Input:

```
2 1 3
7 5 9
8 4 6
```

Output:

```
4
```

Solution

```
const int N = 9;
const int x[] = {0, 1, 2, 3, 4, 5, 1, 2, 4, 5, 7, 8};
const int y[] = {3, 4, 5, 6, 7, 8, 0, 1, 3, 4, 6, 7};
unordered_map<string,int> dist;
queue<string> Q;
string perm;
void init(){
    perm = "123456789";
    dist[perm] = 1;
    Q.push(perm);
    while(!Q.empty()){
        perm = Q.front(); Q.pop();
        int d = dist[perm];
        for(int i = 0; i < 12; i++){
            swap(perm[x[i]], perm[y[i]]);
            if(!dist[perm]){
                dist[perm] = d+1;
                Q.push(perm);
            }
```

```
            swap(perm[x[i]], perm[y[i]]);
        }
    }
}
int main(){
    init();
    for(int i = 0; i < N; i++)
        scanf(" %c", &perm[i]);
    printf("%d\n", dist[perm]-1);
}
```

Writing Numbers ((binary search) how many times the digit '1' is used to write all numbers up to a given number.)

Time limit: 1.00 s **Memory limit:** 512 MB

You would like to write a list of positive integers $1, 2, 3, \ldots$ using your computer. However, you can press each key 0–9 at most n times during the process. What is the last number you can write?

Input

The only input line contains the value of n.

Output

Print the last number you can write.

Constraints

- $1 \leq n \leq 10^{18}$

Example

Input:

```
5
```

Output:

```
12
```

Explanation: You can write the numbers $1, 2, \ldots, 12$. This requires that you press key 1 five times, so you cannot write the number 13.

Solution

```
ll p[19];
void init(){
    p[0] = 1;
    for(int i = 1; i <= 18; i++)
        p[i] = p[i-1] * 10;
}
ll check(ll N){
    ll cnt = 0, l = N, r = 0;
    for(int i = 0; l; i++){
        int d = l % 10;
        l /= 10;
        cnt += (d == 1 ? l*p[i]+r+1 : (l+(d!=0))*p[i]);
        r += p[i] * d;
    }
```

```
    return cnt;
}
int main(){
    ll K;
    scanf("%lld", &K);
    ll lo = 1, hi = 1e18;
    init();
    while(hi-lo > 1){
        ll mid = lo+(hi-lo)/2;
        if(check(mid) <= K) lo = mid;
        else                hi = mid;
    }
    printf("%lld\n", lo);
}
```

Additional Problems Ii

Bit Substrings ((fft) Count the number of non-empty substrings of a binary string that contain exactly k ones, for every k from 0 to n.)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a bit string of length n. Your task is to calculate for each k between $0 \ldots n$ the number of non-empty substrings that contain exactly k ones. For example, if the string is 101, there are: - 1 substring that contains 0 ones: 0 - 4 substrings that contain 1 one: 01, 1, 1, 10 - 1 substring that contains 2 ones: 101 - 0 substrings that contain 3 ones

Input

The only input line contains a binary string of length n.

Output

Print $n + 1$ values as specified above.

Constraints

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

Example

Input:

```
101
```

Output:

```
1 4 1 0
```

Solution

```
typedef double ld;
typedef complex<ld> cd;
const int maxN = 2e5+5;
const ld PI = acos(-1);
ll ans[maxN];
char S[maxN];
void fft(vector<cd> &a, bool inv){
    int N = (int) a.size();
    for(int i = 1, j = 0; i < N; i++){
```

```
        int bit = N>>1;
        for(; j&bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if(i < j)
            swap(a[i], a[j]);
    }
    for(int len = 2; len <= N; len <<= 1){
        ld theta = 2*PI / len * (inv ? -1 : 1);
        cd wlen(cos(theta), sin(theta));
        for(int i = 0; i < N; i += len){
            cd w(1);
            for(int j = 0; j < len / 2; j++){
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if(inv)
        for(cd &z : a)
            z /= N;
}
int next_two_pow(int x){
    return 1<<(32 - __builtin_clz(x));
}
void solve(int l, int r){
    if(l+1 == r){
        ans[(int) (S[l] - '0')]++;
        return;
    }
    int m = (l+r)/2;
    int sz = next_two_pow(r-l);
    vector<cd> A(sz), B(sz);
    for(int i = m-1, cnt = 0; i >= l; i--){
        if(S[i] == '1') cnt++;
        A[cnt] += 1;
    }
    for(int i = m, cnt = 0; i < r; i++){
        if(S[i] == '1') cnt++;
        B[cnt] += 1;
    }
    fft(A, false);
    fft(B, false);
    for(int i = 0; i < sz; i++)
        A[i] *= B[i];
    fft(A, true);
    for(int i = 0; i < sz && i < maxN; i++)
        ans[i] += llround(A[i].real());
    solve(l, m);
    solve(m, r);
}
int main(){
    scanf(" %s", S);
    int N = (int) strlen(S);
```

```
    solve(0, N);
    for(int i = 0; i <= N; i++)
        printf("%lld%c", ans[i], (" \n")[i==N]);
}
```

Book Shop II (bounded knapsack , expanding items with quantity into log(k) groups)

Time limit: 1.00 s **Memory limit:** 512 MB

You are in a book shop which sells n different books. You know the price, the number of pages and the number of copies of each book. You have decided that the total price of your purchases will be at most x. What is the maximum number of pages you can buy? You can buy several copies of the same book.

Input

The first input line contains two integers n and x: the number of book and the maximum total price.

The next line contains n integers $h_1, h_2, \ldots, h_n$: the price of each book.

The next line contains n integers $s_1, s_2, \ldots, s_n$: the number of pages of each book.

The last line contains n integers $k_1, k_2, \ldots, k_n$: the number of copies of each book.

Output

Print one integer: the maximum number of pages.

Constraints

- $1 \leq n \leq 100$
- $1 \leq x \leq 10^5$
- $1 \leq h_i, s_i, k_i \leq 1000$

Example

Input:

```
3 10
2 6 3
8 5 4
3 5 2
```

Output:

```
28
```

Explanation: You can buy three copies of book 1 and one copy of book 3. The price is $3 \cdot 2 + 3 = 9$ and the number of pages is $3 \cdot 8 + 4 = 28$.

Solution

```
const int maxN = 100, maxK = 1000, maxX = 1e5+1;
int N, X, h[maxN], s[maxN], k[maxN];
ll dp[maxX];
int main(){
    scanf("%d %d", &N, &X);
    for(int i = 0; i < N; i++)  scanf("%d", &h[i]);
    for(int i = 0; i < N; i++)  scanf("%d", &s[i]);
    for(int i = 0; i < N; i++)  scanf("%d", &k[i]);
    fill(dp+1, dp+X+1, -1);
    for(int i = 0; i < N; i++){
        for(int b = 1; k[i] > 0; b++){
```

```
            int amnt = min(b, k[i]);
            k[i] -= b;
            int price = amnt * h[i];
            int pages = amnt * s[i];
            for(int j = X; j >= price; j--)
                if(dp[j-price] != -1)
                    dp[j] = max(dp[j], dp[j-price] + pages);
        }
    }
    for(int i = 1; i <= X; i++)
        dp[i] = max(dp[i], dp[i-1]);
    printf("%lld\n", dp[X]);
}
```

Coding Company (counting dp ,split into teams that max - min per team is at most k)

Time limit: 1.00 s **Memory limit:** 512 MB

Your company has n coders, and each of them has a skill level between 0 and 100. Your task is to divide the coders into teams that work together. Based on your experience, you know that teams work well when the skill levels of the coders are about the same. For this reason, the penalty for creating a team is the skill level difference between the best and the worst coder. In how many ways can you divide the coders into teams such that the sum of the penalties is at most x?

Input

The first input line has two integers n and x: the number of coders and the maximum allowed penalty sum.

The next line has n integers $t_1, t_2, \ldots, t_n$: the skill level of each coder.

Output

Print one integer: the number of valid divisions modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 100$
- $0 \leq x \leq 5000$
- $0 \leq t_i \leq 100$

Example

Input:

```
3 2
2 5 3
```

Output:

```
3
```

Solution

```
const int maxN = 101, maxX = 5e3+1;
const ll MOD = 1e9+7;
int N, X, t[maxN];
ll dp[maxN][maxN][maxX];
int main(){
    scanf("%d %d", &N, &X);
```

```
    for(int i = 1; i <= N; i++)
        scanf("%d", &t[i]);
    sort(t+1, t+N+1);
    t[0] = t[1];
    dp[0][0][0] = 1;
    for(int i = 1; i <= N; i++){
        for(int j = N; j >= 0; j--){
            for(int k = X; k >= 0; k--){
                ll cnt = dp[i-1][j][k];
                int newk = k + j * (t[i]-t[i-1]);
                if(newk > X)    continue;
                dp[i][j][newk] = (dp[i][j][newk] + (j+1) * cnt) % MOD;
                if(j != N)  dp[i][j+1][newk] = (dp[i][j+1][newk] + cnt) % MOD;
                if(j != 0)  dp[i][j-1][newk] = (dp[i][j-1][newk] + j * cnt) % MOD;
            }
        }
    }
    ll tot = 0;
    for(int i = 0; i <= X; i++)
        tot = (tot + dp[N][0][i]) % MOD;
    printf("%lld\n", tot);
}
```

Coin Arrangement (greedy distiribute so that each cell has 1 coin)

Time limit: 1.00 s **Memory limit:** 512 MB

There is a $2 \times n$ grid whose each cell contains some number of coins. The total number of coins is $2n$. Your task is to arrange the coins so that each cell contains exactly one coin. On each move you can choose any coin and move it one step left, right, up or down. What is the minimum number of moves if you act optimally?

Input

The first input line has an integer n: the width of the grid.

After this, there are two lines that describe the grid. Each line has n integers: the number of coins in each cell.

Output

Print one integer: the minimum number of moves.

Constraints

- $1 \le n \le 10^5$

Example

Input:

```
4
0 1 0 1
2 0 1 3
```

Output:

```
5
```

Solution

```
const int maxN = 1e5;
int N, a[2][maxN];
```

```
int main(){
    scanf("%d", &N);
    for(int r = 0; r < 2; r++){
        for(int i = 0; i < N; i++){
            scanf("%d", &a[r][i]);
            a[r][i]--;
        }
    }
    ll ans = 0;
    int top = 0, bot = 0;
    for(int i = 0; i < N; i++){
        top += a[0][i];
        bot += a[1][i];
        if((top < 0 && bot > 0) || (top > 0 && bot < 0)){
            if(abs(top) < abs(bot)){
                ans += abs(top);
                bot += top;
                top = 0;
            } else {
                ans += abs(bot);
                top += bot;
                bot = 0;
            }
        }
        ans += abs(top + bot);
    }
    printf("%lld\n", ans);
}
```

Coin Grid(dfs remove all coins ,each move removes all coins from a selected row or column.)

Time limit: 1.00 s **Memory limit:** 512 MB

There is an $n \times n$ grid whose each square is empty or has a coin. On each move, you can remove all coins in a row or column. What is the minimum number of moves after which the grid is empty?

Input

The first input line has an integer n: the size of the grid. The rows and columns are numbered $1, 2, \ldots, n$.

After this, there are n lines describing the grid. Each line has n characters: each character is either . (empty) or o (coin).

Output

First print an integer k: the minimum number of moves. After this, print k lines describing the moves.

On each line, first print 1 (row) or 2 (column), and then the number of a row or column. You can print any valid solution.

Constraints

- $1 \leq n \leq 100$

Example

Input:

```
3
..o
o.o
...
```

Output:

```
2
1 2
2 3
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 105;
char S[maxN];
bool used[maxN], inZ[2][maxN];
int N, cnt, mt[maxN];
vector<int> G[maxN];
bool kuhns(int u){
    if(used[u]) return false;
    used[u] = true;
    for(int v : G[u]){
        if(!mt[v] || kuhns(mt[v])){
            mt[v] = u;
            return true;
        }
    }
    return false;
}
void dfs(int t, int u){
    inZ[t][u] = true;
    if(t == 0 && !inZ[1][mt[u]]){
        dfs(1, mt[u]);
    } else if(t == 1){
        for(int v : G[u])
            if(!inZ[0][v])
                dfs(0, v);
    }
}
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++){
        scanf(" %s", S);
        for(int j = 1; j <= N; j++)
            if(S[j-1] == 'o')
                G[j].push_back(i);
    }
    for(int i = 1; i <= N; i++){
        kuhns(i);
        fill(used+1, used+N+1, false);
    }
    cnt = 0;
    for(int i = 1; i <= N; i++){
        if(mt[i]){
            cnt++;
            used[mt[i]] = true;
        }
```

```
    }
    for(int i = 1; i <= N; i++)
        if(!used[i])
            dfs(1, i);
    printf("%d\n", cnt);
    for(int i = 1; i <= N; i++)
        if(inZ[0][i])
            printf("1 %d\n", i);
    for(int i = 1; i <= N; i++)
        if(!inZ[1][i])
            printf("2 %d\n", i);
}
```

Food Division (distribute values between neighbor based on need array)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n children around a round table. For each child, you know the amount of food they currently have and the amount of food they want. The total amount of food in the table is correct. At each step, a child can give one unit of food to his or her neighbour. What is the minimum number of steps needed?

Input

The first input line contains an integer n: the number of children.

The next line has n integers $a_1, a_2, \ldots, a_n$: the current amount of food for each child.

The last line has n integers $b_1, b_2, \ldots, b_n$: the required amount of food for each child.

Output

Print one integer: the minimum number of steps.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $0 \le a_i, b_i \le 10^6$

Example

Input:

```
3
3 5 0
2 4 2
```

Output:

```
2
```

Explanation: Child 1 gives one unit of food to child 3, and child 2 gives one unit of food to child 3.

Solution

```
const int maxN = 2e5+1;
int N, A[maxN];
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++)  scanf("%d", &A[i]);
    for(int i = 1, b; i <= N; i++){
        scanf("%d", &b);
        A[i] += (A[i-1] - b);
```

```
    }
    sort(A+1, A+N+1);
    ll ans = 0;
    for(int i = 1; i <= N; i++)
        ans += abs(A[i] - A[(N+1)/2]);
    printf("%lld\n", ans);
}
```

Grid Puzzle I (max flow , choose n cell so that each have rows , col have n filled squeras)

Time limit: 1.00 s **Memory limit:** 512 MB

There is an $n \times n$ grid, and your task is to choose from each row and column some number of squares. How can you do that?

Input

The first input line has an integer n: the size of the grid. The rows and columns are numbered $1, 2, \ldots, n$.

The next line has n integers $a_1, a_2, \ldots, a_n$: You must choose exactly a_i squares from the ith row.

The last line has n integers $b_1, b_2, \ldots, b_n$: You must choose exactly b_j squares from the jth column.

Output

Print n lines describing which squares you choose (X means that you choose a square, . means that you don't choose it). You can print any valid solution.

If it is not possible to satisfy the conditions print only -1.

Constraints

- $1 \leq n \leq 50$
- $0 \leq a_i \leq n$
- $0 \leq b_j \leq n$

Example

Input:

```
5
0 1 3 2 0
1 2 2 0 1
```

Output:

```
.....
..X..
.XX.X
XX...
.....
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 105, maxM = 2605;
const int INF = 0x3f3f3f3f;
int N, rowTot, colTot, edgeID, p[maxN], cap[maxN][maxN];
bool vis[maxM];
vector<int> path, F[maxN];
```

```
vector<pii> G[maxN];
int bfs(int s = 0, int t = 2*N+1){
    fill(p, p+2*N+2, -1);
    p[s] = -2;
    queue<pii> Q;
    Q.push({s, INF});
    while(!Q.empty()){
        int u = Q.front().first;
        int f = Q.front().second;
        Q.pop();
        for(int v : F[u]){
            if(p[v] == -1 && cap[u][v]){
                p[v] = u;
                int aug = min(f, cap[u][v]);
                if(v == t)  return aug;
                Q.push({v, aug});
            }
        }
    }
    return 0;
}
void dfs(int u = 0){
    path.push_back(u);
    if(u == N)  return;
    for(pii e : G[u]){
        int v = e.first;
        int id = e.second;
        if(cap[u][v] == 0 && !vis[id]){
            vis[id] = true;
            dfs(v);
            return;
        }
    }
}
int maxflow(int s = 0, int t = 2*N+1){
    int flow = 0, aug = 0;
    while(aug = bfs()){
        flow += aug;
        int u = t;
        while(u != s){
            int v = p[u];
            cap[v][u] -= aug;
            cap[u][v] += aug;
            u = v;
        }
    }
    return flow;
}
int main(){
    scanf("%d", &N);
    for(int i = 1, c; i <= N; i++){
        scanf("%d", &c);
        G[0].push_back({i, ++edgeID});
        F[0].push_back(i);
        F[i].push_back(0);
        cap[0][i] += c;
```

```
        rowTot += c;
    }
    for(int i = N+1, c; i <= 2*N; i++){
        scanf("%d", &c);
        G[i].push_back({2*N+1, ++edgeID});
        F[i].push_back(2*N+1);
        F[2*N+1].push_back(i);
        cap[i][2*N+1] += c;
        colTot += c;
    }
    for(int i = 1; i <= N; i++){
        for(int j = N+1; j <= 2*N; j++){
            G[i].push_back({j, ++edgeID});
            F[i].push_back(j);
            F[j].push_back(i);
            cap[i][j]++;
        }
    }
    int K = maxflow();
    if(rowTot != colTot || K != rowTot){
        printf("-1\n");
        return 0;
    }
    for(int i = 1; i <= N; i++){
        for(int j = 1; j <= N; j++)
            printf("%c", (cap[i][j+N] ? '.' : 'X'));
        printf("\n");
    }
}
```

Grid Puzzle II (min flow, bellman ford)

Time limit: 1.00 s **Memory limit:** 512 MB

There is an $n \times n$ grid whose each square has some number of coins in it. You know for each row and column how many squares you must choose from that row or column. You get all coins from every square you choose. What is the maximum number of coins you can collect and how could you choose the squares so that the given conditions are satisfied?

Input

The first input line has an integer n: the size of the grid. The rows and columns are numbered $1, 2, \ldots, n$.

The next line has n integers $a_1, a_2, \ldots, a_n$: You must choose exactly a_i squares from the ith row.

The next line has n integers $b_1, b_2, \ldots, b_n$: You must choose exactly b_j squares from the jth column.

Finally, there are n lines describing the grid. You can assume that the sums of $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ are equal.

Output

First print an integer k: the maximum number of coins you can collect. After this print n lines describing which squares you choose (X means that you choose a square, . means that you don't choose it).

If it is not possible to satisfy the conditions print only -1.

Constraints

- $1 \leq n \leq 50$

- $0 \leq a_i \leq n$
- $0 \leq b_j \leq n$
- $0 \leq c_{ij} \leq 1000$

Example

Input:

```
5
0 1 3 2 0
1 2 2 0 1
2 5 1 5 1
0 2 5 1 2
3 8 9 3 5
1 4 3 7 3
0 3 6 2 8
```

Output:

```
32
.....
..X..
.XX.X
XX...
.....
```

Solution

```
const int maxN = 105, maxM = 2605;
const int INF = 0x3f3f3f3f;
int N, K, rowtot, coltot, edgecnt, p[maxN], d[maxN];
bool inq[maxN], vis[maxM];
vector<int> path, G[maxN];
struct Edge {
    int u, v, r, c;
} edges[maxM], redges[maxM];
void read_input(){
    scanf("%d", &N);
    for(int v = 1, cap; v <= N; v++){
        scanf("%d", &cap);
        rowtot += cap;
        edgecnt++;
        G[0].push_back(edgecnt);
        G[v].push_back(-edgecnt);
        edges[edgecnt] = {0, v, cap, 0};
        redges[edgecnt] = {v, 0, 0, 0};
    }
    for(int u = N+1, cap; u <= 2*N; u++){
        scanf("%d", &cap);
        coltot += cap;
        edgecnt++;
        G[u].push_back(edgecnt);
        G[2*N+1].push_back(-edgecnt);
        edges[edgecnt] = {u, 2*N+1, cap, 0};
        redges[edgecnt] = {2*N+1, u, 0, 0};
    }
    for(int u = 1; u <= N; u++){
        for(int v = N+1, cost; v <= 2*N; v++){
            scanf("%d", &cost);
```

```
            edgecnt++;
            G[u].push_back(edgecnt);
            G[v].push_back(-edgecnt);
            edges[edgecnt] = {u, v, 1, -cost};
            redges[edgecnt] = {v, u, 0, cost};
        }
    }
}
void bellman_ford(int s = 0){
    fill(inq, inq+2*N+2, false);
    fill(d, d+2*N+2, INF);
    fill(p, p+2*N+2, 0);
    queue<int> Q;
    Q.push(s);
    d[s] = 0;
    inq[s] = true;
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        inq[u] = false;
        for(int i : G[u]){
            Edge e = (i < 0 ? redges[-i] : edges[i]);
            if(e.r > 0 && d[e.v] > d[u] + e.c){
                d[e.v] = d[u] + e.c;
                p[e.v] = i;
                if(!inq[e.v]){
                    inq[e.v] = true;
                    Q.push(e.v);
                }
            }
        }
    }
}
int minimum_cost_flow(int s = 0, int t = 2*N+1){
    int flow = 0, cost = 0;
    while(flow < K){
        bellman_ford(s);
        if(d[t] == INF) break;
        int aug = K-flow;
        int u = t;
        while(u != s){
            Edge e = (p[u] < 0 ? redges[-p[u]] : edges[p[u]]);
            aug = min(aug, e.r);
            u = e.u;
        }
        flow += aug;
        cost += aug * d[t];
        u = t;
        while(u != s){
            if(p[u] < 0){
                redges[-p[u]].r -= aug;
                edges[-p[u]].r += aug;
            } else {
                redges[p[u]].r += aug;
                edges[p[u]].r -= aug;
            }
            u = (p[u] < 0 ? redges[-p[u]].u : edges[p[u]].u);
```

```
        }
    }
    return (flow < K ? INF : cost);
}
int main(){
    read_input();
    K = rowtot;
    int max_reward = -minimum_cost_flow();
    if(rowtot != coltot || max_reward == -INF){
        printf("-1\n");
        return 0;
    }
    printf("%d\n", max_reward);
    for(Edge e : edges){
        if(e.u != 0 && e.v != 2*N+1){
            printf("%c", (e.r == 0 ? 'X' : '.'));
            if(e.v == 2*N)  printf("\n");
        }
    }
}
```

Increasing Array II (gready , make array increasing)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers. You want to modify the array so that it is increasing, i.e., every element is at least as large as the previous element. On each move, you can increase or decrease the value of any element by one. What is the minimum number of moves required?

Input

The first input line contains an integer n: the size of the array.

Then, the second line contains n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print the minimum number of moves.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

```
5
3 8 5 6 5
```

Output:

```
4
```

Solution

```
int N, x;
priority_queue<int> Q;
int main(){
    scanf("%d", &N);
```

```
    ll ans = 0;
    for(int i = 0; i < N; i++){
        scanf("%d", &x);
        Q.push(x);
        ans += Q.top() - x;
        Q.pop();
        Q.push(x);
    }
    printf("%lld\n", ans);
}
```

Mex Grid Queries (grundy xor)

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a two-dimensional grid whose rows and columns are 1-indexed. Each square contains the smallest nonnegative integer that does not appear to the left on the same row or above on the same column. Your task is to calculate the value at square (y, x).

Input

The only input line contains two integers y and x.

Output

Print one integer: the value at square (y, x).

Constraints

- $1 \le y, x \le 10^9$

Example

Input:

```
3 5
```

Output:

```
6
```

Solution

```
int x, y;
int main(){
    scanf("%d %d", &x, &y);
    printf("%d\n", (x-1)^(y-1));
}
```

Programmers and Artists (gready ,pri queue)

Time limit: 1.00 s **Memory limit:** 512 MB

A company wants to hire a programmers and b artists. There are a total of n applicants, and each applicant can become either a programmer or an artist. You know each applicant's programming and artistic skills. Your task is to select the new employees so that the sum of their skills is maximum.

Input

The first input line has three integers a, b and n: the required number of programmers and artists, and the total number of applicants.

After this, there are n lines that describe the applicants. Each line has two integers x and y: the applicant's programming and artistic skills.

Output

Print one integer: the maximum sum of skills.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $0 \le a, b \le n$
- $a + b \le n$
- $1 \le x, y \le 10^9$

Example

Input:

```
2 1 4
3 7
9 8
1 5
4 2
```

Output:

```
20
```

Explanation: An optimal solution is to hire two programmers with skills 9 and 4 and one artist with skill 7. The sum of the skills is $9 + 4 + 7 = 20$.

Solution

```
typedef pair<int,int> pii;
const int maxN = 2e5+5;
int N, A, B;
pii cand[maxN];
multiset<int> initial_artist_set(){
    vector<int> all;
    for(int i = A; i < N; i++)
        all.push_back(cand[i].second);
    sort(all.begin(), all.end());
    reverse(all.begin(), all.end());
    multiset<int> best;
    for(int i = 0; i < B; i++)
        best.insert(all[i]);
    return best;
}
int main(){
    scanf("%d %d %d", &A, &B, &N);
    for(int i = 0, x, y; i < N; i++){
        scanf("%d %d", &x, &y);
        cand[i] = {x, y};
    }
    sort(cand, cand+N, [](const pii a, const pii b){
        return a.first == b.first ? a.second < b.second : a.first > b.first;
```

```
    });
    ll pref = 0;
    priority_queue<int> prefix_deltas;
    for(int i = 0; i < A; i++){
        pref += cand[i].first;
        prefix_deltas.push(cand[i].second - cand[i].first);
    }
    multiset<int> remaining_art = initial_artist_set();
    ll suf = accumulate(remaining_art.begin(), remaining_art.end(), 0LL);
    ll ans = pref + suf;
    for(int i = A; i < A+B; i++){
        const int x = cand[i].first;
        const int y = cand[i].second;
        pref += x;
        prefix_deltas.push(y - x);
        pref += prefix_deltas.top();
        prefix_deltas.pop();
        auto worst_artist = remaining_art.lower_bound(y);
        int art_value = *worst_artist;
        remaining_art.erase(worst_artist);
        suf -= art_value;
        ans = max(ans, pref + suf);
    }
    printf("%lld\n", ans);
}
```

Reversal Sorting (treap , sort array by reversing subarray)

Time limit: 1.00 s **Memory limit:** 512 MB

You have an array that contains a permutation of integers $1, 2, \ldots, n$. Your task is to sort the array in increasing order by reversing subarrays. You can construct any solution that has at most n reversals.

Input

The first input line has an integer n: the size of the array. The array elements are numbered $1, 2, \ldots, n$.

The next line has n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

First print an integer k: the number of reversals.

After that, print k lines that describe the reversals. Each line has two integers a and b: you reverse a subarray from position a to position b.

Constraints

- $1 \le n \le 2 \cdot 10^5$

Example

Input:

```
4
2 3 1 4
```

Output:

```
2
1 3
```

```
2 3
```

Solution

```
const int maxN = 2e5+5;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> dist(1, (int) 2e9);
struct Node {
    bool rev;
    int value, prior, sz;
    Node *p, *l, *r;
    Node(){}
    Node(int val, int pri){
        prior = pri;
        value = val;
        sz = 1;
        p = l = r = nullptr;
    }
};
int N, X[maxN];
Node *root, *ndptr[maxN];
int sz(Node *t){    return t ? t->sz : 0;       }
int val(Node *t){   return t ? t->value : -1;   }
void flip(Node *t){
    if(!t)  return;
    t->rev ^= true;
}
void pull(Node *t){
    if(!t)  return;
    if(t->l)    t->l->p = t;
    if(t->r)    t->r->p = t;
    t->sz = sz(t->l) + sz(t->r) + 1;
}
void push(Node *t){
    if(!t)  return;
    if(t->rev){
        swap(t->l, t->r);
        flip(t->l); flip(t->r);
        t->rev = false;
    }
}
Node* merge(Node *x, Node *y){
    if(!x || !y)    return x ? x : y;
    push(x); push(y);
    if(x->prior < y->prior){
        x->r = merge(x->r, y);
        pull(x);
        return x;
    } else {
        y->l = merge(x, y->l);
        pull(y);
        return y;
    }
}
pair<Node*,Node*> split(Node *x, int k){
    if(!x)  return {nullptr, nullptr};
    pair<Node*,Node*> y = {nullptr, nullptr};
```

```
    push(x);
    if(k <= sz(x->l)){
        y = split(x->l, k);
        x->l = y.second;
        pull(x);
        y.second = x;
    } else {
        y = split(x->r, k-sz(x->l)-1);
        x->r = y.first;
        pull(x);
        y.first = x;
    }
    return y;
}
void heapify(Node *t){
    if(!t)  return;
    Node *mx = t;
    if(t->l && t->l->prior > mx->prior) mx = t->l;
    if(t->r && t->r->prior > mx->prior) mx = t->r;
    if(mx != t){
        swap(t->prior, mx->prior);
        heapify(mx);
    }
}
Node* build(int x, int k){
    if(k == 0)  return nullptr;
    int mid = k/2;
    Node *t = new Node(X[x+mid], dist(rng));
    ndptr[X[x+mid]] = t;
    t->l = build(x, mid);
    t->r = build(x+mid+1, k-mid-1);
    heapify(t);
    pull(t);
    return t;
}
void reverse(int x, int k){
    pair<Node*,Node*> y, z;
    y = split(root, x-1);
    z = split(y.second, k);
    flip(z.first);
    y.second = merge(z.first, z.second);
    root = merge(y.first, y.second);
    root->p = nullptr;
}
int orderOf(int v){
    Node* t = ndptr[v];
    vector<Node*> walk;
    while(t){
        walk.push_back(t);
        t = t->p;
    }
    reverse(walk.begin(), walk.end());
    for(Node* nd : walk){
        push(nd);
        pull(nd);
    }
```

```
    t = ndptr[v];
    int idx = sz(t->l);
    while(t){
        if(t->p && t == t->p->r)
            idx += sz(t->p->l) + 1;
        t = t->p;
    }
    return idx;
}
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)
        scanf("%d", &X[i]);
    root = build(0, N);
    printf("%d\n", N);
    for(int l = 1; l <= N; l++){
        int r = orderOf(l)+1;
        reverse(l, r-l+1);
        printf("%d %d\n", l, r);
    }
}
```

School Excursion (dsu , subset dp find all possible groups that can be assignd to two sets)

Time limit: 1.00 s **Memory limit:** 512 MB

A group of n children are coming to Helsinki. There are two possible attractions: a child can visit either Korkeasaari (zoo) or Linnanmäki (amusement park). There are m pairs of children who want to visit the same attraction. Your task is to find all possible alternatives for the number of children that will visit Korkeasaari. The children's wishes have to be taken into account.

Input

The first input line has two integers n and m: the number of children and their wishes. The children are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the children's wishes. Each line has two integers a and b: children a and b want to visit the same attraction.

Output

Print a bit string of length n where a one-bit at index i indicates that it is possible that exactly i children visit Korkeasaari (the bit string is to be considered one-indexed).

Constraints

- $1 \le n \le 10^5$
- $0 \le m \le 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
10011
```

Explanation: The number of children visiting Korkeasaari can be 1, 4 or 5.

Solution

```
const int maxN = 1e5+1;
int N, M, a, b, ds[maxN];
multiset<int> sizes;
bitset<maxN> dp;
int find(int u){
    if(ds[u] < 0)   return u;
    ds[u] = find(ds[u]);
    return ds[u];
}
bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v)  return false;
    if(ds[u] < ds[v])   swap(u, v);
    ds[v] += ds[u];
    ds[u] = v;
    return true;
}
int main(){
    scanf("%d %d", &N, &M);
    fill(ds+1, ds+N+1, -1);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        merge(a, b);
    }
    for(int i = 1; i <= N; i++)
        if(find(i) == i)
            sizes.insert(-ds[i]);
    dp[0] = 1;
    for(int sz : sizes)
        dp |= (dp<<sz);
    for(int i = 1; i <= N; i++)
        printf("%d", dp[i] ? 1 : 0);
    printf("\n");
}
```

Stick Divisions (gready , minimize cost of cutting stick cost is the lenght of the stick)

Time limit: 1.00 s **Memory limit:** 512 MB

You have a stick of length x and you want to divide it into n sticks, with given lengths, whose total length is x. On each move you can take any stick and divide it into two sticks. The cost of such an operation is the length of the original stick. What is the minimum cost needed to create the sticks?

Input

The first input line has two integers x and n: the length of the stick and the number of sticks in the division.

The second line has n integers $d_1, d_2, \ldots, d_n$: the length of each stick in the division.

Output

Print one integer: the minimum cost of the division.

Constraints

- $1 \le x \le 10^9$
- $1 \le n \le 2 \cdot 10^5$
- $\sum d_i = x$

Example

Input:

```
8 3
2 3 3
```

Output:

```
13
```

Explanation: You first divide the stick of length 8 into sticks of length 3 and 5 (cost 8). After this, you divide the stick of length 5 into sticks of length 2 and 3 (cost 5). The total cost is $8 + 5 = 13$.

Solution

```
int N, x, d;
ll sum;
priority_queue<int, vector<int>, greater<int>> sticks;
int main(){
    scanf("%d %d", &x, &N);
    for(int i = 0; i < N; i++){
        scanf("%d", &d);
        sticks.push(d);
    }
    while(sticks.size() > 1){
        int a = sticks.top(); sticks.pop();
        int b = sticks.top(); sticks.pop();
        sticks.push(a+b);
        sum += a + b;
    }
    printf("%lld\n", sum);
}
```

Swap Round Sorting (dfs gready ,sort a permutation can do multiple non overlapping swap in the same round)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array containing a permutation of numbers $1, 2, \ldots, n$, and your task is to sort the array using swap rounds. On each swap round, you can choose any number of distinct pairs of elements and swap each pair. Your task is to find the minimum number of rounds and show how you can choose the pairs in each round.

Input

The first input line has an integer n: the size of the array.

The second line has n integers $x_1, x_2, \ldots, x_n$: the initial permutation.

Output

First print an integer k: the minimum number of rounds.

Then, for each round, print the number of swaps and the indices of each swap. You can print any valid solution.

Constraints

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

Example

Input:

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

Output:

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

Explanation: The initial array is $[5, 2, 1, 3, 4]$. After round 1, the array becomes $[1, 2, 5, 4, 3]$. After round 2, the array becomes $[1, 2, 3, 4, 5]$.

Solution

```
typedef pair<int,int> pii;
const int maxN = 2e5+1;
bool vis[maxN];
int N, a[maxN];
vector<int> cycle;
vector<pii> ans;
bool zero_rounds(){
    for(int i = 1; i <= N; i++)
        if(a[i] != i)
            return false;
    printf("0\n");
    return true;
}
bool one_round(bool second = false){
    for(int i = 1; i <= N; i++){
        if(a[i] != i){
            if(i != a[a[i]])    { ans.clear(); return false; }
            else if(i < a[i])   ans.push_back({i, a[i]});
        }
    }
    if(!second) printf("1\n");
    printf("%d\n", (int) ans.size());
    for(pii p : ans){
        printf("%d %d\n", p.first, p.second);
        swap(a[p.first], a[p.second]);
    }
    return true;
}
void dfs(int u){
    vis[u] = true;
    cycle.push_back(u);
    if(!vis[a[u]])  dfs(a[u]);
}
int main(){
    scanf("%d", &N);
```

```
    for(int i = 1; i <= N; i++)
        scanf("%d", &a[i]);
    if(!zero_rounds() && !one_round()){
        printf("2\n");
        for(int i = 1; i <= N; i++){
            if(!vis[i]){
                dfs(i);
                int K = (int) cycle.size();
                for(int u = 0; u < K/2; u++){
                    ans.push_back({cycle[u], cycle[K-u-1]});
                    swap(a[cycle[u]], a[cycle[K-u-1]]);
                }
                cycle.clear();
            }
        }
        printf("%d\n", (int) ans.size());
        for(pii p : ans)
            printf("%d %d\n", p.first, p.second);
        ans.clear();
        one_round(true);
    }
}
```

Advanced Graph Problems Ramez

Acyclic Graph Edges (converting a graph into asyclic one)

Time limit: 1.00 s **Memory limit:** 512 MB

Given an undirected graph, your task is to choose a direction for each edge so that the resulting directed graph is acyclic.

Input

The first input line has two integers n and m: the number of nodes and edges. The nodes are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the edges. Each line has two distinct integers a and b: there is an edge between nodes a and b.

Output

Print m lines describing the directions of the edges. Each line has two integers a and b: there is an edge from node a to node b. You can print any valid solution.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq m \leq 2 \cdot 10^5$
- $1 \leq a, b \leq n$

Example

Input:

```
3 3
1 2
2 3
3 1
```

Output:

```
1 2
3 2
3 1
```

Solution

```
int N, M, a, b;
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        printf("%d %d\n", min(a, b), max(a, b));
    }
}
```

Course Schedule II (Topo sort with the lexographically smallest ordering)

Time limit: 1.00 s **Memory limit:** 512 MB

You want to complete n courses that have requirements of the form "course a has to be completed before course b". You want to complete course 1 as soon as possible. If there are several ways to do this, you want then to complete course 2 as soon as possible, and so on. Your task is to determine the order in which you complete the courses.

Input

The first input line has two integers n and m: the number of courses and requirements. The courses are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the requirements. Each line has two integers a and b: course a has to be completed before course b.

You can assume that there is at least one valid schedule.

Output

Print one line having n integers: the order in which you complete the courses.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
2 1 3 4
```

Solution

```
const int maxN = 2e5;
int N, M, a, b, idx, in[maxN], ans[maxN];
vector<int> G[maxN];
priority_queue<int> Q;
```

```
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        G[b].push_back(a);
        in[a]++;
    }
    for(int i = 1; i <= N; i++)
        if(in[i] == 0)
            Q.push(i);
    idx = N;
    while(!Q.empty()){
        int u = Q.top(); Q.pop();
        ans[idx--] = u;
        for(int v : G[u]){
            in[v]--;
            if(in[v] == 0)
                Q.push(v);
        }
    }
    for(int i = 1; i <= N; i++)
        printf("%d%c", ans[i], (" \n")[i==N]);
}
```

Creating Offices (what's the maximum number of nodes selected such that the distance between any two is at least d)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities and $n-1$ roads between them. There is a unique route between any two cities, and their distance is the number of roads on that route. A company wants to have offices in some cities, but the distance between any two offices has to be at least d. What is the maximum number of offices they can have?

Input

The first input line has two integers n and d: the number of cities and the minimum distance. The cities are numbered $1, 2, \ldots, n$.

After this, there are $n-1$ lines describing the roads. Each line has two integers a and b: there is a road between cities a and b.

Output

First print an integer k: the maximum number of offices. After that, print the cities which will have offices. You can print any valid solution.

Constraints

- $1 \le n, d \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

```
3 5
```

Output:

```
2
1 4
```

Solution

```
const int maxN = 2e5+1, logN = 20;
const int INF = 0x3f3f3f3f;
int N, D, ct[maxN], sz[maxN], best[maxN];
int timer, p[maxN][logN], d[maxN], in[maxN], out[maxN];
bool vis[maxN];
vector<int> G[maxN];
void dfs(int u = 1, int par = 1){
    in[u] = ++timer;
    d[u] = d[par]+1;
    p[u][0] = par;
    for(int i = 1; i < logN; i++)
        p[u][i] = p[p[u][i-1]][i-1];
    for(int v : G[u])
        if(v != par)
            dfs(v, u);
    out[u] = ++timer;
}
bool ancestor(int u, int v){
    return in[u] <= in[v] && out[u] >= out[v];
}
int lca(int u, int v){
    if(ancestor(u, v))  return u;
    if(ancestor(v, u))  return v;
    for(int i = logN-1; i >= 0; i--)
        if(!ancestor(p[u][i], v))
            u = p[u][i];
    return p[u][0];
}
int dist(int u, int v){
    return d[u] + d[v] - 2*d[lca(u, v)];
}
int find_size(int u, int p = -1){
    if(vis[u])  return 0;
    sz[u] = 1;
    for(int v : G[u])
        if(v != p)
            sz[u] += find_size(v, u);
    return sz[u];
}
int find_centroid(int u, int p, int n){
    for(int v : G[u])
        if(v != p)
            if(!vis[v] && sz[v] > n/2)
                return find_centroid(v, u, n);
    return u;
}
void build_centroid_tree(int u = 1, int p = -1){
    find_size(u);
    int c = find_centroid(u, -1, sz[u]);
```

```
    vis[c] = true;
    ct[c] = p;
    for(int v : G[c])
        if(!vis[v])
            build_centroid_tree(v, c);
}
void update(int u){
    best[u] = 0;
    int v = u;
    while(ct[v] != -1){
        v = ct[v];
        best[v] = min(best[v], dist(u, v));
    }
}
int query(int u){
    int ans = best[u];
    int v = u;
    while(ct[v] != -1){
        v = ct[v];
        ans = min(ans, best[v] + dist(u, v));
    }
    return ans;
}
int main(){
    scanf("%d %d", &N, &D);
    for(int i = 0, a, b; i < N-1; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
    }
    dfs();
    build_centroid_tree();
    fill(best+1, best+N+1, INF);
    vector<int> order(N);
    for(int i = 0; i < N; i++)
        order[i] = i+1;
    sort(order.begin(), order.end(), [](int a, int b){
        return d[a] == d[b] ? a < b : d[a] > d[b];
    });
    vector<int> ans;
    for(int u : order){
        int dist_to_office = query(u);
        if(dist_to_office >= D){
            ans.push_back(u);
            update(u);
        }
    }
    sort(ans.begin(), ans.end());
    int K = (int) ans.size();
    printf("%d\n", K);
    for(int i = 0; i < K; i++)
        printf("%d%c", ans[i], (" \n")[i==K-1]);
}
```

Critical Cities (find nodes that appear on every route between any two nodes)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities and m flight connections between them. A city is called a critical city if it appears on every route from a city to another city. Your task is to find all critical cities from Syrjälä to Lehmälä.

Input

The first input line has two integers n and m: the number of cities and flights. The cities are numbered $1, 2, \ldots, n$. City 1 is Syrjälä, and city n is Lehmälä.

Then, there are m lines describing the connections. Each line has two integers a and b: there is a flight from city a to city b. All flights are one-way.

You may assume that there is a route from Syrjälä to Lehmälä.

Output

First print an integer k: the number of critical cities. After this, print k integers: the critical cities in increasing order.

Constraints

- $2 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
3
1 2 5
```

Solution

```
const int maxN = 1e5+1;
int N, M, p[maxN], sdom[maxN], idom[maxN], label[maxN];
int timer, tin[maxN], inv[maxN], dsu[maxN];
vector<int> ans, G[maxN], GR[maxN], DT[maxN], bucket[maxN];
void dfs(int u = 1){
    tin[u] = ++timer;
    inv[timer] = u;
    label[timer] = sdom[timer] = dsu[timer] = timer;
    for(int v : G[u]){
        if(!tin[v]){
            dfs(v);
            p[tin[v]] = tin[u];
        }
        GR[tin[v]].push_back(tin[u]);
    }
}
int find(int u, bool x = false){
    if(u == dsu[u]) return x ? -1 : u;
    int v = find(dsu[u], true);
```

```
    if(v < 0)   return u;
    if(sdom[label[dsu[u]]] < sdom[label[u]])
        label[u] = label[dsu[u]];
    dsu[u] = v;
    return x ? v : label[u];
}
void build_dominator_tree(){
    dfs();
    for(int u = N; u > 0; u--){
        for(int v : GR[u])
            sdom[u] = min(sdom[u], sdom[find(v)]);
        if(u > 1)   bucket[sdom[u]].push_back(u);
        for(int v : bucket[u])
            idom[v] = (sdom[find(v)] == sdom[v] ? sdom[v] : find(v));
        if(u > 1)   dsu[u] = p[u];
    }
    for(int u = 2; u <= N; u++){
        if(idom[u] != sdom[u])
            idom[u] = idom[idom[u]];
        DT[inv[u]].push_back(inv[idom[u]]);
        DT[inv[idom[u]]].push_back(inv[u]);
    }
}
bool dfs_dominator_tree(int u = 1, int p = -1){
    bool good = (u == N);
    for(int v : DT[u])
        if(v != p)
            good |= dfs_dominator_tree(v, u);
    if(good)    ans.push_back(u);
    return good;
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
    }
    build_dominator_tree();
    dfs_dominator_tree();
    sort(ans.begin(), ans.end());
    int K = (int) ans.size();
    printf("%d\n", K);
    for(int i = 0; i < K; i++)
        printf("%d%c", ans[i], (" \n")[i==K-1]);
}
```

Even Outdegree Edges (choose a direction for each edge so that each node has an even outdegree.)

Time limit: 1.00 s **Memory limit:** 512 MB

Given an undirected graph, your task is to choose a direction for each edge so that in the resulting directed graph each node has an even outdegree. The outdegree of a node is the number of edges coming out of that node.

Input

The first input line has two integers n and m: the number of nodes and edges. The nodes are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the edges. Each line has two integers a and b: there is an edge between nodes a and b.

You may assume that the graph is simple, i.e., there is at most one edge between any two nodes and every edge connects two distinct nodes.

Output

Print m lines describing the directions of the edges. Each line has two integers a and b: there is an edge from node a to node b. You can print any valid solution.

If there are no solutions, only print IMPOSSIBLE.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

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

Solution

```
typedef pair<int,int> pii;
const int maxN = 1e5+5, maxM = 2e5+5;
int N, M, timer, in[maxN];
bool even[maxN];
vector<pii> G[maxN];
pii ans[maxM];
void dfs(int u, int p = -1){
    in[u] = ++timer;
    for(pii e : G[u]){
        int v = e.first, id = e.second;
        if(v != p){
            if(!in[v]){
                dfs(v, u);
                if(even[v]){
                    ans[id] = {u, v};
                    even[u] ^= true;
                } else {
                    ans[id] = {v, u};
                    even[v] ^= true;
                }
            } else if(in[v] < in[u]){
                even[u] ^= true;
```

```
                ans[id] = {u, v};
            }
        }
    }
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back({b, i});
        G[b].push_back({a, i});
    }
    fill(even+1, even+N+1, true);
    for(int i = 1; i <= N; i++)
        if(!in[i])
            dfs(i);
    bool good = true;
    for(int i = 1; i <= N; i++)
        good &= even[i];
    if(good){
        for(int i = 0; i < M; i++)
            printf("%d %d\n", ans[i].first, ans[i].second);
    } else printf("IMPOSSIBLE\n");
}
```

Flight Route Requests (minimum number of directed edges to connect nodes with connection requirements)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities with airports but no flight connections. You are given m requests which routes should be possible to travel. Your task is to determine the minimum number of one-way flight connections which makes it possible to fulfil all requests.

Input

The first input line has two integers n and m: the number of cities and requests. The cities are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the requests. Each line has two integers a and b: there has to be a route from city a to city b. Each request is unique.

Output

Print one integer: the minimum number of flight connections.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

```
3 1
3 4
```

Output:

```
4
```

Explanation: You can create the connections $1 \to 2$, $2 \to 3$, $2 \to 4$ and $3 \to 1$. Then you can also fly from city 3 to city 4 using the route $3 \to 1 \to 2 \to 4$.

Solution

```
const int maxN = 1e5+1;
int N, M, ds[maxN], vis[maxN];
vector<int> G[maxN], CC[maxN];
int find(int u){
    if(ds[u] < 0)   return u;
    ds[u] = find(ds[u]);
    return ds[u];
}
bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v)  return false;
    if(ds[u] < ds[v])   swap(u, v);
    ds[v] += ds[u];
    ds[u] = v;
    return true;
}
bool dfs(int u){
    vis[u] = -1;
    bool hascycle = false;
    for(int v : G[u]){
        if(vis[v] == -1)        return true;
        else if(vis[v] == 0)    hascycle |= dfs(v);
    }
    vis[u] = 1;
    return hascycle;
}
int main(){
    scanf("%d %d", &N, &M);
    fill(ds+1, ds+N+1, -1);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        merge(a, b);
    }
    int K = 0;
    unordered_map<int,int> getID;
    for(int u = 1; u <= N; u++){
        int rep = find(u);
        if(!getID[rep])
            getID[rep] = ++K;
        CC[getID[rep]].push_back(u);
    }
    int ans = 0;
    for(int k = 1; k <= K; k++){
        int sz = (int) CC[k].size();
        bool hascycle = false;
```

```
        for(int u : CC[k])
            if(!hascycle && vis[u] == 0)
                hascycle |= dfs(u);
        ans += (hascycle ? sz : sz-1);
    }
    printf("%d\n", ans);
}
```

Graph Girth (find the length of the shortest cycle)

Time limit: 1.00 s **Memory limit:** 512 MB

Given an undirected graph, your task is to determine its girth, i.e., the length of its shortest cycle.

Input

The first input line has two integers n and m: the number of nodes and edges. The nodes are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the edges. Each line has two integers a and b: there is an edge between nodes a and b.

You may assume that there is at most one edge between each two nodes.

Output

Print one integer: the girth of the graph. If there are no cycles, print -1.

Constraints

- $1 \le n \le 2500$
- $1 \le m \le 5000$

Example

Input:

```
5 6
1 2
1 3
2 4
2 5
3 4
4 5
```

Output:

```
3
```

Solution

```
const int maxN = 2501;
bool vis[maxN];
int N, M, best, p[maxN], dist[maxN];
vector<int> G[maxN];
void reset(){
    fill(vis+1, vis+N+1, false);
    fill(dist+1, dist+N+1, 0);
    fill(p+1, p+N+1, 0);
}
void bfs(int start){
    queue<int> Q;
```

```
    Q.push(start);
    vis[start] = true;
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        for(int v : G[u]){
            if(!vis[v]){
                dist[v] = dist[u]+1;
                vis[v] = true;
                p[v] = u;
                Q.push(v);
            } else if(v != u[p]){
                best = min(best, dist[u]+dist[v]+1);
            }
        }
    }
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
    }
    best = N+1;
    for(int start = 1; start <= N; start++){
        reset();
        bfs(start);
    }
    printf("%d\n", best == N+1 ? -1 : best);
}
```

Network Breakdown (counting components after each edge deletion, offline queries)

Time limit: 1.00 s **Memory limit:** 512 MB

Syrjälä's network has n computers and m connections between them. The network consists of components of computers that can send messages to each other. Nobody in Syrjälä understands how the network works. For this reason, if a connection breaks down, nobody will repair it. In this situation a component may be divided into two components. Your task is to calculate the number of components after each connection breakdown.

Input

The first input line has three integers n, m and k: the number of computers, connections and breakdowns. The computers are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the connections. Each line has two integers a and b: there is a connection between computers a and b. Each connection is between two different computers, and there is at most one connection between two computers.

Finally, there are k lines describing the breakdowns. Each line has two integers a and b: the connection between computers a and b breaks down.

Output

After each breakdown, print the number of components.

Constraints

- $1 \le n \le 10^5$

- $1 \leq m \leq 2 \cdot 10^5$
- $1 \leq k \leq m$
- $1 \leq a, b \leq n$

Example

Input:

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

Output:

```
2 2 3
```

Solution

```
const int maxN = 1e5+1;
const int maxM = 2e5;
typedef pair<int,int> pii;
int N, M, K, a, b, cnt, ds[maxN], ans[maxM];
pii edges[maxM], queries[maxM];
set<pii> S;
int find(int u){
    if(ds[u] < 0)   return u;
    ds[u] = find(ds[u]);
    return ds[u];
}
bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v)  return false;
    if(ds[u] < ds[v])   swap(u, v);
    ds[v] += ds[u];
    ds[u] = v;
    return true;
}
int main(){
    scanf("%d %d %d", &N, &M, &K);
    fill(ds+1, ds+N+1, -1);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        if(b > a)   swap(a, b);
        edges[i] = {a, b};
    }
    for(int i = 0; i < K; i++){
        scanf("%d %d", &a, &b);
        if(b > a)   swap(a, b);
        queries[i] = {a, b};
        S.insert({a, b});
    }
    cnt = N;
    for(int i = 0; i < M; i++)
```

```
        if(S.find(edges[i]) == S.end())
            if(merge(edges[i].first, edges[i].second))
                cnt--;
    for(int i = K-1; i >= 0; i--){
        ans[i] = cnt;
        if(merge(queries[i].first, queries[i].second))
            cnt--;
    }
    for(int i = 0; i < K; i++)
        printf("%d%c", ans[i], (" \n")[i==K-1]);
}
```

Network Renovation (minimum number of edges to add so deletion of any edge doesn't disconnet the graph)

Time limit: 1.00 s **Memory limit:** 512 MB

Syrjälä's network consists of n computers and $n-1$ connections between them. It is possible to send data between any two computers. However, if any connection breaks down, it will no longer be possible to send data between some computers. Your task is to add the minimum number of new connections in such a way that you can still send data between any two computers even if any single connection breaks down.

Input

The first input line has an integer n: the number of computers. The computers are numbered $1, 2, \ldots, n$.

After this, there are $n-1$ lines describing the connections. Each line has two integers a and b: there is a connection between computers a and b.

Output

First print an integer k: the minimum number of new connections. After this, print k lines describing the connections. You can print any valid solution.

Constraints

- $3 \le n \le 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
2
2 4
4 5
```

Solution

```
const int maxN = 1e5+1;
int N, K;
vector<int> leaves, G[maxN];
```

```
void dfs(int u = 1, int p = -1){
    if((int) G[u].size() == 1)  leaves.push_back(u);
    for(int v : G[u])
        if(v != p)
            dfs(v, u);
}
int main(){
    scanf("%d", &N);
    for(int i = 0, a, b; i < N-1; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
    }
    dfs();
    int K = (int) leaves.size();
    printf("%d\n", (K+1)/2);
    for(int i = 0; i < (K+1)/2; i++)
        printf("%d %d\n", leaves[i], leaves[i+K/2]);
}
```

Strongly Connected Edges (redirecting edges so that the resulting graph is strongly connected)

Time limit: 1.00 s **Memory limit:** 512 MB

Given an undirected graph, your task is to choose a direction for each edge so that the resulting directed graph is strongly connected.

Input

The first input line has two integers n and m: the number of nodes and edges. The nodes are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the edges. Each line has two integers a and b: there is an edge between nodes a and b.

You may assume that the graph is simple, i.e., there are at most one edge between two nodes and every edge connects two distinct nodes.

Output

Print m lines describing the directions of the edges. Each line has two integers a and b: there is an edge from node a to node b. You can print any valid solution.

If there are no solutions, only print IMPOSSIBLE.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

```
3 3
1 2
1 3
2 3
```

Output:

```
1 2
2 3
3 1
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 1e5+1;
const int maxM = 2e5+1;
int N, M, timer, tin[maxN], low[maxN];
bool bridge_exists, vis[maxN], used[maxM];
vector<pii> G[maxN];
pii edges[maxM];
void dfs(int u = 1, int p = 0){
    vis[u] = true;
    tin[u] = low[u] = ++timer;
    for(pii P : G[u]){
        int v = P.first;
        int id = P.second;
        if(used[id])    continue;
        used[id] = true;
        edges[id] = {u, v};
        if(vis[v]){
            low[u] = min(low[u], tin[v]);
        } else {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] > tin[u])
                bridge_exists = true;
        }
    }
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back({b, i});
        G[b].push_back({a, i});
    }
    dfs();
    if(timer == N && !bridge_exists){
        for(int i  = 0; i < M; i++)
            printf("%d %d\n", edges[i].first, edges[i].second);
    } else printf("IMPOSSIBLE\n");
}
```

Transfer Speeds Sum (sum of all mininmum edges in all possible paths between any two nodes)

Time limit: 1.00 s **Memory limit:** 512 MB

A computer network has n computers and $n-1$ connections between two computers. Information can be exchanged between every pair of computers using the connections. Each connection has a certain transfer speed. Let $d(a,b)$ denote the transfer speed between computers a and b, which is the speed of the slowest connection on the route between a and b. Your task is to compute the sum of transfer speeds between all pairs of computers.

Input

The first line contains the integer n: the number of computers. The computers are numbered $1, 2, \ldots, n$.

After this, there are $n-1$ lines, which describe the connections. Each line has three integers a, b and x: there is a connection between computers a and b with transfer speed x.

Output

Print one integer: the sum of transfer speeds.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq x \leq 10^6$

Example

Input:

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

Output:

```
12
```

Explanation: The following figure corresponds to the sample input:

Here $d(1,2) = 5$, $d(1,3) = 1$, $d(1,4) = 2$, $d(2,3) = 1$, $d(2,4) = 2$, and $d(3,4) = 1$, so the sum of transfer speeds is 12.

Solution

```
typedef array<int,3> edge;
const int maxN = 2e5+1;
int N, ds[maxN];
ll ans;
vector<edge> edges;
int find(int u){
    if(ds[u] < 0)   return u;
    ds[u] = find(ds[u]);
    return ds[u];
}
bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v)  return false;
    if(ds[u] < ds[v])   swap(u, v);
    ds[v] += ds[u];
    ds[u] = v;
    return true;
}
int main(){
    scanf("%d", &N);
    for(int i = 0, a, b, x; i < N-1; i++){
        scanf("%d %d %d", &a, &b, &x);
        edges.push_back({a, b, x});
    }
    sort(edges.begin(), edges.end(), [](const edge& e1, const edge& e2){
        return e1[2] > e2[2];
```

```
    });
    fill(ds, ds+N, -1);
    for(const edge& e: edges){
        int a = find(e[0]), b = find(e[1]);
        int sza = -ds[a], szb = -ds[b];
        ll x = e[2];
        ans += x * sza * szb;
        merge(a, b);
    }
    printf("%lld\n", ans);
}
```

Tree Isomorphism I (check if two rooted trees are the same with different node naming, tree hashing for every subtree)

Time limit: 1.00 s **Memory limit:** 512 MB

Given two rooted trees, your task is to find out if they are isomorphic, i.e., it is possible to draw them so that they look the same.

Input

The first input line has an integer t: the number of tests. Then, there are t tests described as follows:

The first line has an integer n: the number of nodes in both trees. The nodes are numbered $1, 2, \ldots, n$, and node 1 is the root.

Then, there are $n-1$ lines describing the edges of the first tree, and finally $n-1$ lines describing the edges of the second tree.

Output

For each test, print "YES", if the trees are isomorphic, and "NO" otherwise.

Constraints

- $1 \le t \le 1000$
- $2 \le n \le 10^5$
- the sum of all values of n is at most 10^5

Example

Input:

```
2
3
1 2
2 3
1 2
1 3
3
1 2
2 3
1 3
3 2
```

Output:

```
NO
YES
```

Solution

```
const int maxN = 1e5+1;
int N, treeID, name[2][maxN];
vector<int> G[2][maxN];
map<vector<int>,int> mp;
void reset(){
    for(int t = 0; t < 2; t++)
        for(int i = 1; i <= N; i++)
            G[t][i].clear();
}
void dfs(int t, int u, int p){
    vector<int> childNames;
    for(int v : G[t][u]){
        if(v != p){
            dfs(t, v, u);
            childNames.push_back(name[t][v]);
        }
    }
    sort(childNames.begin(), childNames.end());
    if(!mp[childNames]) mp[childNames] = ++treeID;
    name[t][u] = mp[childNames];
}
void solve_case(){
    scanf("%d", &N);
    reset();
    for(int t = 0; t < 2; t++){
        for(int i = 0, a, b; i < N-1; i++){
            scanf("%d %d", &a, &b);
            G[t][a].push_back(b);
            G[t][b].push_back(a);
        }
        dfs(t, 1, -1);
    }
    printf("%s\n", name[0][1] == name[1][1] ? "YES" : "NO");
}
int main(){
    int T;
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        solve_case();
    }
}
```

Tree Isomorphism II (find the centroied for two unrooted trees and check if they are the same with different node naming, tree hashing for every subtree)

Time limit: 1.00 s **Memory limit:** 512 MB

Given two (not rooted) trees, your task is to find out if they are isomorphic, i.e., it is possible to draw them so that they look the same.

Input

The first input line has an integer t: the number of tests. Then, there are t tests described as follows:

The first line has an integer n: the number of nodes in both trees. The nodes are numbered $1, 2, \ldots, n$.

Then, there are $n-1$ lines describing the edges of the first tree, and finally $n-1$ lines describing the edges of the second tree.

Output

For each test, print "YES", if the trees are isomorphic, and "NO" otherwise.

Constraints

- $1 \leq t \leq 1000$
- $2 \leq n \leq 10^5$
- the sum of all values of n is at most 10^5

Example

Input:

```
2
3
1 2
2 3
1 2
1 3
3
1 2
2 3
1 3
3 2
```

Output:

```
YES
YES
```

Solution

```
const int maxN = 1e5+1;
int N, treeID, sz[2][maxN], name[2][maxN];
vector<int> centroids[2], G[2][maxN];
map<vector<int>,int> mp;
void reset(){
    mp.clear();
    treeID = 0;
    for(int t = 0; t < 2; t++){
        centroids[t].clear();
        for(int i = 1; i <= N; i++){
            sz[t][i] = name[t][i] = 0;
            G[t][i].clear();
        }
    }
}
void dfs1(int t, int u, int p){
    sz[t][u] = 1;
    bool is_centroid = true;
    for(int v : G[t][u]){
        if(v != p){
            dfs1(t, v, u);
            sz[t][u] += sz[t][v];
            if(sz[t][v] > N/2)  is_centroid = false;
```

```
        }
    }
    if(N-sz[t][u] > N/2)    is_centroid = false;
    if(is_centroid)         centroids[t].push_back(u);
}
void dfs2(int t, int u, int p){
    vector<int> childNames;
    for(int v : G[t][u]){
        if(v != p){
            dfs2(t, v, u);
            childNames.push_back(name[t][v]);
        }
    }
    sort(childNames.begin(), childNames.end());
    if(!mp[childNames]) mp[childNames] = ++treeID;
    name[t][u] = mp[childNames];
}
void solve_case(){
    scanf("%d", &N);
    reset();
    for(int t = 0; t < 2; t++){
        for(int i = 0, a, b; i < N-1; i++){
            scanf("%d %d", &a, &b);
            G[t][a].push_back(b);
            G[t][b].push_back(a);
        }
        dfs1(t, 1, -1);
    }
    for(int root1 : centroids[0]){
        for(int root2 : centroids[1]){
            dfs2(0, root1, -1);
            dfs2(1, root2, -1);
            if(name[0][root1] == name[1][root2]){
                printf("YES\n");
                return;
            }
        }
    }
    printf("NO\n");
}
int main(){
    int T;
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        solve_case();
    }
}
```

Tree Traversals (Constructing the postorder from pre and inorder)

Time limit: 1.00 s **Memory limit:** 512 MB

There are three common ways to traverse the nodes of a binary tree: - Preorder: First process the root, then the left subtree, and finally the right subtree. - Inorder: First process the left subtree, then the root, and finally the right subtree. - Postorder: First process the left subtree, then the right subtree, and finally the root. There is a binary tree of n nodes with distinct labels. You are given the preorder and inorder traversals of the tree, and your task is to determine its postorder traversal.

Input

The first input line has an integer n: the number of nodes. The nodes are numbered $1, 2, \ldots, n$.

After this, there are two lines describing the preorder and inorder traversals of the tree. Both lines consist of n integers.

You can assume that the input corresponds to a binary tree.

Output

Print the postorder traversal of the tree.

Constraints

- $1 \le n \le 10^5$

Example

Input:

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

Output:

```
3 1 4 2 5
```

Solution

```
const int maxN = 1e5+5;
int N, timer, A[maxN], B[maxN], C[maxN], idx[maxN];
int L[maxN], R[maxN];
int solve(int l1 = 0, int r1 = N-1, int l2 = 0, int r2 = N-1){
    if(l1 > r1 || l2 > r2)  return 0;
    int root = A[l1];
    int m2 = idx[root];
    int len = m2-l2;
    L[root] = solve(l1+1, l1+len, l2, m2-1);
    R[root] = solve(l1+len+1, r1, m2+1, r2);
    return root;
}
void dfs(int u){
    if(L[u])    dfs(L[u]);
    if(R[u])    dfs(R[u]);
    C[timer++] = u;
}
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)  scanf("%d", &A[i]);
    for(int i = 0; i < N; i++){
        scanf("%d", &B[i]);
        idx[B[i]] = i;
    }
    int root = solve();
    dfs(root);
    for(int i = 0; i < N; i++)
        printf("%d%c", C[i], (" \n")[i==N-1]);
}
```

Visiting Cities (Dijkstra, find the cities in the minimum route from 1 to n)

Time limit: 1.00 s **Memory limit:** 512 MB

You want to travel from Syrjälä to Lehmälä by plane using a minimum-price route. Which cities will you certainly visit?

Input

The first input line contains two integers n and m: the number of cities and the number of flights. The cities are numbered $1, 2, \ldots, n$. City 1 is Syrjälä, and city n is Lehmälä.

After this, there are m lines describing the flights. Each line has three integers a, b, and c: there is a flight from city a to city b with price c. All flights are one-way flights.

You may assume that there is a route from Syrjälä to Lehmälä.

Output

First print an integer k: the number of cities that are certainly in the route. After this, print the k cities sorted in increasing order.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$
- $1 \le c \le 10^9$

Example

Input:

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

Output:

```
4
1 3 4 5
```

Solution

```
typedef pair<int,ll> edge;
typedef pair<ll,int> node;
const int maxN = 1e5+1;
const ll INF = 0x3f3f3f3f3f3f3f3f;
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<ll> distrib((ll) 1e9, (ll) 2e9);
const ll MOD1 = distrib(rng);
const ll MOD2 = distrib(rng);
int N, M;
ll dist[2][maxN], ways[2][2][maxN];
vector<edge> G[2][maxN];
vector<int> ans;
void init(){
```

```
    for(int t = 0; t < 2; t++)
        for(int i = 1; i <= N; i++)
            dist[t][i] = INF;
}
void dijkstra(int type, int source){
    priority_queue<node, vector<node>, greater<node>> Q;
    dist[type][source] = 0;
    ways[type][0][source] = 1;
    ways[type][1][source] = 1;
    Q.push({0, source});
    while(!Q.empty()){
        int u = Q.top().second;
        ll d = Q.top().first;
        Q.pop();
        if(d > dist[type][u])   continue;
        for(edge e : G[type][u]){
            int v = e.first;
            ll w = e.second;
            if(dist[type][v] > d+w){
                ways[type][0][v] = ways[type][0][u];
                ways[type][1][v] = ways[type][1][u];
                dist[type][v] = d+w;
                Q.push({d+w, v});
            } else if(dist[type][v] == d+w){
                ways[type][0][v] += ways[type][0][u];
                ways[type][0][v] %= MOD1;
                ways[type][1][v] += ways[type][1][u];
                ways[type][1][v] %= MOD2;
            }
        }
    }
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b, c; i < M; i++){
        scanf("%d %d %d", &a, &b, &c);
        G[0][a].push_back({b, c});
        G[1][b].push_back({a, c});
    }
    init();
    dijkstra(0, 1);
    dijkstra(1, N);
    ll tot1 = ways[0][0][N], tot2 = ways[0][1][N];
    for(int u = 1; u <= N; u++){
        ll ways1 = ways[0][0][u] * ways[1][0][u] % MOD1;
        ll ways2 = ways[0][1][u] * ways[1][1][u] % MOD2;
        if(ways1 == tot1 && ways2 == tot2 && dist[0][u] + dist[1][u] == dist[0][N])
            ans.push_back(u);
    }
    int K = (int) ans.size();
    printf("%d\n", K);
    for(int i = 0; i < K; i++)
        printf("%d%c", ans[i], (" \n")[i==K-1]);
}
```

Advanced Techniques

Apples and Bananas (fft count pair sum)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n apples and m bananas, and each of them has an integer weight between $1 \dots k$. Your task is to calculate, for each weight w between $2 \dots 2k$, the number of ways we can choose an apple and a banana whose combined weight is w.

Input

The first input line contains three integers k, n and m: the number k, the number of apples and the number of bananas.

The next line contains n integers $a_1, a_2, \dots, a_n$: weight of each apple.

The last line contains m integers $b_1, b_2, \dots, b_m$: weight of each banana.

Output

For each integer w between $2 \dots 2k$ print the number of ways to choose an apple and a banana whose combined weight is w.

Constraints

- $1 \leq k, n, m \leq 2 \cdot 10^5$
- $1 \leq a_i \leq k$
- $1 \leq b_i \leq k$

Example

Input:

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

Output:

```
0 0 1 2 1 2 4 2 0
```

Explanation: For example for $w = 8$ there are 4 different ways: we can pick an apple of weight 5 in two different ways and a banana of weight 3 in two different ways.

Solution

```
typedef double ld;
typedef complex<ld> cd;
const int SIZE = 1<<19;
const ld PI = acos(-1);
int N, M, K;
vector<cd> A(SIZE), B(SIZE);
void fft(vector<cd> &a, bool inv){
    int N = (int) a.size();
    for(int i = 1, j = 0; i < N; i++){
        int bit = N>>1;
        for(; j&bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if(i < j)
            swap(a[i], a[j]);
```

```
    }
    for(int len = 2; len <= N; len <<= 1){
        ld theta = 2*PI / len * (inv ? -1 : 1);
        cd wlen(cos(theta), sin(theta));
        for(int i = 0; i < N; i += len){
            cd w(1);
            for(int j = 0; j < len / 2; j++){
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if(inv)
        for(cd &z : a)
            z /= N;
}
int main(){
    scanf("%d %d %d", &K, &N, &M);
    for(int i = 0, x; i < N; i++){
        scanf("%d", &x);
        A[x] += 1;
    }
    for(int i = 0, x; i < M; i++){
        scanf("%d", &x);
        B[x] += 1;
    }
    fft(A, false);
    fft(B, false);
    for(int i = 0; i < SIZE; i++)
        A[i] *= B[i];
    fft(A, true);
    for(int i = 2; i <= 2*K; i++)
        printf("%lld%c", llround(A[i].real()), (" \n")[i==2*K]);
}
```

Corner Subgrid Count ((bitset)Count all rectangular subgrids of size at least 2×2 whose four corners are black cells.)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an $n \times n$ grid whose each square is either black or white. A subgrid is called beautiful if its height and width is at least two and all of its corners are black. How many beautiful subgrids are there within the given grid?

Input

The first input line has an integer n: the size of the grid.

Then there are n lines describing the grid: 1 means that a square is black and 0 means it is white.

Output

Print the number of beautiful subgrids.

Constraints

- $1 \le n \le 3000$

Example

Input:

```
5
00010
11111
00110
11001
00010
```

Output:

```
4
```

Solution

```
##pragma GCC target("popcnt")
const int maxN = 3005;
int N;
ll ans;
bitset<maxN> B[maxN];
int f(int X){
    return X*(X-1);
}
int main(){
    cin.sync_with_stdio(0);
    cin.tie(0);
    cin >> N;
    for(int i = 0; i < N; i++)
        cin >> B[i];
    for(int i = 0; i < N; i++)
        for(int j = i+1; j < N; j++)
            ans += f((B[i]&B[j]).count());
    cout << (ans/2);
}
```

Cut and Paste (treap cut and paste to the end)

Time limit: 1.00 s **Memory limit:** 512 MB

Given a string, your task is to process operations where you cut a substring and paste it to the end of the string. What is the final string after all the operations?

Input

The first input line has two integers n and m: the length of the string and the number of operations. The characters of the string are numbered $1, 2, \ldots, n$.

The next line has a string of length n that consists of characters A–Z.

Finally, there are m lines that describe the operations. Each line has two integers a and b: you cut a substring from position a to position b.

Output

Print the final string after all the operations.

Constraints

- $1 \le n, m \le 2 \cdot 10^5$

- $1 \leq a \leq b \leq n$

Example

Input:

```
7 2
AYBABTU
3 5
3 5
```

Output:

```
AYABTUB
```

Solution

```
const int maxN = 2e5+5;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> dist(1, (int) 2e9);
struct Node {
    char ch;
    int prior, sz;
    Node *l, *r;
    Node(){}
    Node(char c, int p){
        prior = p;
        ch = c;
        sz = 1;
        l = r = nullptr;
    }
};
int N, M, a, b;
char S[maxN];
Node *root;
int sz(Node *t){ return t ? t->sz : 0; }
void pull(Node *t){
    if(!t)  return;
    t->sz = sz(t->l) + sz(t->r) + 1;
}
void push(Node *t){
    if(!t)  return;
}
Node* merge(Node *x, Node *y){
    if(!x || !y)    return x ? x : y;
    push(x); push(y);
    if(x->prior < y->prior){
        x->r = merge(x->r, y);
        pull(x);
        return x;
    } else {
        y->l = merge(x, y->l);
        pull(y);
        return y;
    }
}
pair<Node*,Node*> split(Node *x, int k){
    if(!x)  return {nullptr, nullptr};
    pair<Node*,Node*> y = {nullptr, nullptr};
```

```
    push(x);
    if(k <= sz(x->l)){
        y = split(x->l, k);
        x->l = y.second;
        pull(x);
        y.second = x;
    } else {
        y = split(x->r, k-sz(x->l)-1);
        x->r = y.first;
        pull(x);
        y.first = x;
    }
    return y;
}
void heapify(Node *t){
    if(!t)  return;
    Node *mx = t;
    if(t->l && t->l->prior > mx->prior) mx = t->l;
    if(t->r && t->r->prior > mx->prior) mx = t->r;
    if(mx != t){
        swap(t->prior, mx->prior);
        heapify(mx);
    }
}
Node* build(int x, int k){
    if(k == 0)  return nullptr;
    int mid = k/2;
    Node *t = new Node(S[x+mid], dist(rng));
    t->l = build(x, mid);
    t->r = build(x+mid+1, k-mid-1);
    heapify(t);
    pull(t);
    return t;
}
void cut(int x, int k){
    pair<Node*,Node*> y, z;
    y = split(root, x-1);
    z = split(y.second, k);
    y.second = merge(z.second, z.first);
    root = merge(y.first, y.second);
}
void print(Node *t){
    if(!t)  return;
    push(t);
    print(t->l);
    printf("%c", t->ch);
    print(t->r);
}
int main(){
    scanf("%d %d %s", &N, &M, S);
    root = build(0, N);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        int len = (b-a+1);
        cut(a, len);
    }
```

```
    print(root);
}
```

Distinct Routes II(min cost flow bellman ford)

Time limit: 1.00 s **Memory limit:** 512 MB

A game consists of n rooms and m teleporters. At the beginning of each day, you start in room 1 and you have to reach room n. You can use each teleporter at most once during the game. You want to play the game for exactly k days. Every time you use any teleporter you have to pay one coin. What is the minimum number of coins you have to pay during k days if you play optimally?

Input

The first input line has three integers n, m and k: the number of rooms, the number of teleporters and the number of days you play the game. The rooms are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the teleporters. Each line has two integers a and b: there is a teleporter from room a to room b.

There are no two teleporters whose starting and ending room are the same.

Output

First print one integer: the minimum number of coins you have to pay if you play optimally. Then, print k route descriptions according to the example. You can print any valid solution.

If it is not possible to play the game for k days, print only -1.

Constraints

- $2 \le n \le 500$
- $1 \le m \le 1000$
- $1 \le k \le n - 1$
- $1 \le a, b \le n$

Example

Input:

```
8 10 2
1 2
1 3
2 5
2 4
3 5
3 6
4 8
5 8
6 7
7 8
```

Output:

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

Solution

```
const int maxN = 501, maxM = 1001;
const int INF = 0x3f3f3f3f;
int N, M, K, p[maxN], d[maxN];
bool inq[maxN], vis[maxM];
vector<int> path, G[maxN];
struct Edge {
    int u, v, r, c;
} edges[maxM], redges[maxM];
void bellman_ford(){
    fill(inq+1, inq+N+1, false);
    fill(d+1, d+N+1, INF);
    fill(p+1, p+N+1, 0);
    queue<int> Q;
    Q.push(1);
    d[1] = 0;
    inq[1] = true;
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        inq[u] = false;
        for(int i : G[u]){
            Edge e = (i < 0 ? redges[-i] : edges[i]);
            if(e.r > 0 && d[e.v] > d[u] + e.c){
                d[e.v] = d[u] + e.c;
                p[e.v] = i;
                if(!inq[e.v]){
                    inq[e.v] = true;
                    Q.push(e.v);
                }
            }
        }
    }
}
int minimum_cost_flow(){
    int flow = 0, cost = 0;
    while(flow < K){
        bellman_ford();
        if(d[N] == INF) break;
        int aug = K-flow;
        int u = N;
        while(u != 1){
            Edge e = (p[u] < 0 ? redges[-p[u]] : edges[p[u]]);
            aug = min(aug, e.r);
            u = e.u;
        }
        flow += aug;
        cost += aug * d[N];
        u = N;
        while(u != 1){
            if(p[u] < 0){
                redges[-p[u]].r -= aug;
                edges[-p[u]].r += aug;
            } else {
                redges[p[u]].r += aug;
                edges[p[u]].r -= aug;
            }
```

```
            u = (p[u] < 0 ? redges[-p[u]].u : edges[p[u]].u);
        }
    }
    return (flow < K ? -1 : cost);
}
void dfs(int u = 1){
    path.push_back(u);
    if(u == N)  return;
    for(int i : G[u]){
        if(i > 0 && edges[i].r == 0 && !vis[i]){
            vis[i] = true;
            dfs(edges[i].v);
            return;
        }
    }
}
int main(){
    scanf("%d %d %d", &N, &M, &K);
    for(int i = 1, u, v; i <= M; i++){
        scanf("%d %d", &u, &v);
        G[u].push_back(i);
        G[v].push_back(-i);
        edges[i] = {u, v, 1, 1};
        redges[i] = {v, u, 0, -1};
    }
    int minCoins = minimum_cost_flow();
    if(minCoins == -1){
        printf("-1\n");
        return 0;
    }
    printf("%d\n", minCoins);
    for(int i = 0; i < K; i++){
        path.clear();
        dfs();
        int sz = (int) path.size();
        printf("%d\n", sz);
        for(int j = 0; j < sz; j++)
            printf("%d%c", path[j], (" \n")[j==sz-1]);
    }
}
```

Dynamic Connectivity (link cut tree)

Time limit: 1.00 s **Memory limit:** 512 MB

Consider an undirected graph that consists of n nodes and m edges. There are two types of events that can happen: - A new edge is created between nodes a and b. - An existing edge between nodes a and b is removed. Your task is to report the number of components after every event.

Input

The first input line has three integers n, m and k: the number of nodes, edges and events.

After this there are m lines describing the edges. Each line has two integers a and b: there is an edge between nodes a and b. There is at most one edge between any pair of nodes.

Then there are k lines describing the events. Each line has the form "t a b" where t is 1 (create a new edge) or 2 (remove an edge). A new edge is always created between two nodes that do not already have an edge between them,

and only existing edges can get removed.

Output

Print $k+1$ integers: first the number of components before the first event, and after this the new number of components after each event.

Constraints

- $2 \leq n \leq 10^5$
- $1 \leq m, k \leq 10^5$
- $1 \leq a, b \leq n$

Example

Input:

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

Output:

```
2 2 2 1
```

Solution

```
typedef pair<int,int> pii;
typedef array<int,3> triple;
const int maxN = 1e5+1;
const int SIZE = 3e5+1;
const int INF = 0x3f3f3f3f;
int N, M, K, root, components, del_time[SIZE];
bool used[SIZE];
pii edges[SIZE];
triple ops[maxN];
map<pii,deque<int>> edge_id_map;
struct Node {
    bool rev = false;
    Node *c[2] = {nullptr, nullptr}, *p = nullptr;
    int mn, id;
    Node(int i){
        mn = id = i;
    }
    void push(){
        if(rev){
            swap(c[0], c[1]);
            if(c[0])    c[0]->rev ^= true;
            if(c[1])    c[1]->rev ^= true;
            rev = false;
        }
    }
    void pull(){
        mn = id;
        for(int i = 0; i < 2; i++)
```

```
            if(c[i] && del_time[c[i]->mn] < del_time[mn])
                mn = c[i]->mn;
    }
} *LCT[SIZE];
bool notRoot(Node *t){
    return t->p && (t->p->c[0] == t || t->p->c[1] == t);
}
void rotate(Node *t){
    Node *p = t->p;
    bool b = (p->c[0] == t);
    if((t->p = p->p) && notRoot(p)) t->p->c[(t->p->c[1] == p)] = t;
    if((p->c[!b]=t->c[b]))          p->c[!b]->p = p;
    t->c[b] = p;
    p->p = t;
    p->pull();
}
void splay(Node *t){
    while(notRoot(t)){
        Node *p = t->p;
        p->push();
        t->push();
        rotate(t);
    }
    t->push();
    t->pull();
}
Node* access(Node *t){
    Node *last = nullptr;
    for(Node *u = t; u; u = u->p){
        splay(u);
        u->c[1] = last;
        last = u;
    }
    splay(t);
    return last;
}
void evert(Node *t){
    access(t);
    t->rev = true;
}
void link(Node *u, Node *v){
    evert(u);
    u->p = v;
}
void cut(Node *u, Node *v){
    evert(u);
    access(v);
    if(v->c[0]) v->c[0]->p = 0;
    v->c[0] = 0;
    v->pull();
}
Node* path(Node *u, Node *v){
    evert(u);
    access(v);
    return v;
}
```

```
bool connected(Node *u, Node *v){
    path(u, v);
    while(v->c[0])
        v = v->c[0];
    return u == v;
}
void create_edge(int u, int v){
    int id = edge_id_map[{u, v}].front();
    if(!connected(LCT[u], LCT[v])){
        components--;
        link(LCT[id], LCT[u]);
        link(LCT[id], LCT[v]);
        used[id] = true;
    } else {
        int mn = path(LCT[u], LCT[v])->mn;
        if(del_time[mn] < del_time[id]){
            int cu = edges[mn].first;
            int cv = edges[mn].second;
            cut(LCT[mn], LCT[cu]);
            cut(LCT[mn], LCT[cv]);
            used[mn] = false;
            link(LCT[id], LCT[u]);
            link(LCT[id], LCT[v]);
            used[id] = true;
        }
    }
}
void destroy_edge(int u, int v){
    int id = edge_id_map[{u, v}].front();
    edge_id_map[{u, v}].pop_front();
    if(!used[id])   return;
    cut(LCT[id], LCT[u]);
    cut(LCT[id], LCT[v]);
    used[id] = false;
    components++;
}
int main(){
    scanf("%d %d %d", &N, &M, &K);
    root = 1;
    for(int i = 1; i <= N; i++){
        LCT[i] = new Node(i);
        del_time[i] = INF;
    }
    for(int i = 1, a, b; i <= M; i++){
        scanf("%d %d", &a, &b);
        if(a > b)   swap(a, b);
        edges[N+i] = {a, b};
        del_time[N+i] = INF;
        LCT[N+i] = new Node(N+i);
        edge_id_map[{a, b}].push_back(N+i);
    }
    for(int i = 1, t, a, b; i <= K; i++){
        scanf("%d %d %d", &t, &a, &b);
        if(a > b)   swap(a, b);
        ops[i] = {t, a, b};
        edges[N+M+i] = {a, b};
```

```
        if(t == 2){
            int id = edge_id_map[{a, b}].back();
            del_time[id] = i;
        } else {
            del_time[N+M+i] = INF;
            LCT[N+M+i] = new Node(N+M+i);
            edge_id_map[{a, b}].push_back(N+M+i);
        }
    }
    components = N;
    for(int i = N+1; i <= N+M; i++){
        int u = edges[i].first;
        int v = edges[i].second;
        create_edge(u, v);
    }
    printf("%d ", components);
    for(int i = 1; i <= K; i++){
        int t = ops[i][0];
        int u = ops[i][1];
        int v = ops[i][2];
        if(t == 1)  create_edge(u, v);
        else        destroy_edge(u, v);
        printf("%d%c", components, (" \n")[i==K]);
    }
}
```

Eulerian Subgraphs (2^cycle)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an undirected graph that has n nodes and m edges. We consider subgraphs that have all nodes of the original graph and some of its edges. A subgraph is called Eulerian if each node has even degree. Your task is to count the number of Eulerian subgraphs modulo $10^9 + 7$.

Input

The first input line has two integers n and m: the number of nodes and edges. The nodes are numbered $1, 2, \ldots, n$.

After this, there are m lines that describe the edges. Each line has two integers a and b: there is an edge between nodes a and b. There is at most one edge between two nodes, and each edge connects two distinct nodes.

Output

Print the number of Eulerian subgraphs modulo $10^9 + 7$.

Constraints

- $1 \le n \le 10^5$
- $0 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

2

Explanation: You can either keep or remove all edges, so there are two possible Eulerian subgraphs.

Solution

```
const int maxN = 1e5+1;
const int MOD = 1e9+7;
int N, M, ans, ds[maxN];
int find(int u){
    if(ds[u] < 0)   return u;
    ds[u] = find(ds[u]);
    return ds[u];
}
bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v)  return false;
    if(ds[u] < ds[v])   swap(u, v);
    ds[v] += ds[u];
    ds[u] = v;
    return true;
}
int main(){
    scanf("%d %d", &N, &M);
    fill(ds+1, ds+N+1, -1);
    ans = 1;
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        if(!merge(a, b))
            ans = (2 * ans) % MOD;
    }
    printf("%d\n", ans);
}
```

Hamming Distance (minimum difference between two strings out of n binary strings, can be done with trie)

Time limit: 1.00 s **Memory limit:** 512 MB

The Hamming distance between two strings a and b of equal length is the number of positions where the strings differ. You are given n bit strings, each of length k and your task is to calculate the minimum Hamming distance between two strings.

Input

The first input line has two integers n and k: the number of bit strings and their length.

Then there are n lines each consisting of one bit string of length k.

Output

Print the minimum Hamming distance between two strings.

Constraints

- $2 \le n \le 2 \cdot 10^4$
- $1 \le k \le 30$

Example

Input:

```
5 6
110111
001000
100001
101000
101110
```

Output:

```
1
```

Explanation: The strings 101000 and 001000 differ only at the first position.

Solution

```
const int maxN = 2e4;
int N, ans, b[maxN];
int scanBinary(){
    char c;
    int res = 0;
    while((c = getchar()) != '\n'){
        res <<= 1;
        res += (c-'0')&1;
    }
    return res;
}
int main(){
    scanf("%d %d ", &N, &ans);
    for(int i = 0; i < N; i++)
        b[i] = scanBinary();
    for(int i = 0; i < N; i++)
        for(int j = i+1; j < N; j++)
            ans = min(ans, __builtin_popcount(b[i]^b[j]));
    printf("%d\n", ans);
}
```

Houses and Schools (divide and conquer dp Minimize the total walking distance of all children. by placing k schools)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n houses on a street, numbered $1, 2, \ldots, n$. The distance of houses a and b is $|a - b|$. You know the number of children in each house. Your task is to establish k schools in such a way that each school is in some house. Then, each child goes to the nearest school. What is the minimum total walking distance of the children if you act optimally?

Input

The first input line has two integers n and k: the number of houses and the number of schools. The houses are numbered $1, 2 \ldots, n$.

After this, there are n integers $c_1, c_2, \ldots, c_n$: the number of children in each house.

Output

Print the minimum total distance.

Constraints

- $1 \leq k \leq n \leq 3000$
- $1 \leq c_i \leq 10^9$

Example

Input:

```
6 2
2 7 1 4 6 4
```

Output:

```
11
```

Explanation: Houses 2 and 5 will have schools.

Solution

```
const int maxN = 3001;
const ll INF = 0x3f3f3f3f3f3f3f3f;
int N, K;
ll best, p[3][maxN], dp[maxN][maxN];
ll travel(int d, int a, int b){
    return (p[d][b] - p[d][a-1]) - (p[0][b] - p[0][a-1]) * (d == 1 ? a : N-b+1);
}
ll C(int a, int b){
    int m = (a+b)/2;
    return travel(1, a, m) + travel(2, m+1, b);
}
void solve(int k, int a = 1, int b = N, int optl = 1, int optr = N){
    if(a > b)   return;
    int m = (a+b)/2;
    int opt = -1;
    dp[k][m] = INF;
    for(int i = optl; i <= m; i++){
        if(dp[k-1][i] + C(i, m) < dp[k][m]){
            dp[k][m] = dp[k-1][i] + C(i, m);
            opt = i;
        }
    }
    solve(k, a, m-1, optl, opt);
    solve(k, m+1, b, opt, optr);
}
int main(){
    scanf("%d %d", &N, &K);
    for(int i = 1; i <= N; i++){
        ll x;
        scanf("%lld", &x);
        p[0][i] = p[0][i-1] + x;
        p[1][i] = p[1][i-1] + i * x;
        p[2][i] = p[2][i-1] + (N-i+1) * x;
    }
    for(int i = 1; i <= N; i++)
        dp[1][i] = travel(2, 1, i);
    for(int k = 2; k <= K; k++)
        solve(k);
    best = INF;
    for(int i = 1; i <= N; i++)
```

```
        best = min(best, dp[K][i] + travel(1, i, N));
    printf("%lld\n", best);
}
```

Knuth Division (range dp + prefix sum Minimize total cost to split an array into single-element subarrays)

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n numbers, your task is to divide it into n subarrays, each of which has a single element. On each move, you may choose any subarray and split it into two subarrays. The cost of such a move is the sum of values in the chosen subarray. What is the minimum total cost if you act optimally?

Input

The first input line has an integer n: the array size. The array elements are numbered $1, 2, \ldots, n$.

The second line has n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print one integer: the minimum total cost.

Constraints

- $1 \le n \le 5000$
- $1 \le x_i \le 10^9$

Example

Input:

```
5
2 7 3 2 5
```

Output:

```
43
```

Solution

```
const ll INF = 0x3f3f3f3f3f3f3f3f;
const int maxN = 5005;
int N, split[maxN][maxN];
ll pre[maxN], dp[maxN][maxN];
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++){
        scanf("%lld", &pre[i]);
        pre[i] += pre[i-1];
    }
    for(int len = 0; len <= N; len++){
        for(int l = 0; l+len <= N; l++){
            int r = l+len;
            if(len < 2){
                dp[l][r] = 0;
                split[l][r] = l;
                continue;
            }
            dp[l][r] = INF;
```

```
            for(int m = split[l][r-1]; m <= split[l+1][r]; m++){
                ll possible = dp[l][m] + dp[m][r] + pre[r] - pre[l];
                if(possible < dp[l][r]){
                    dp[l][r] = possible;
                    split[l][r] = m;
                }
            }
        }
    }
    printf("%lld\n", dp[0][N]);
}
```

Meet in the Middle (Count subsets with sum x)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n numbers. In how many ways can you choose a subset of the numbers with sum x?

Input

The first input line has two numbers n and x: the array size and the required sum.

The second line has n integers $t_1, t_2, \ldots, t_n$: the numbers in the array.

Output

Print the number of ways you can create the sum x.

Constraints

- $1 \leq n \leq 40$
- $1 \leq x \leq 10^9$
- $1 \leq t_i \leq 10^9$

Example

Input:

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

Output:

```
3
```

Solution

```
const int maxN = 40;
int N, t[maxN];
ll x, sum, cnt;
unordered_map<ll,int> freq;
int main(){
    scanf("%d %lld", &N, &x);
    for(int i = 0; i < N; i++)
        scanf("%d", &t[i]);
    sort(t, t+N);
    if(N == 1){
        printf("%d\n", x == t[0]);
        return 0;
    }
    freq.reserve(1<<(N/2-1));
```

```
    for(int i = 0; i < (1<<(N/2-1)); i++){
        sum = 0;
        for(int j = 0; j < N/2-1; j++)
            if(i&(1<<j))
                sum += t[j];
        freq[sum]++;
    }
    for(int i = 0; i < (1<<((N+1)/2+1)); i++){
        sum = 0;
        for(int j = 0; j < (N+1)/2+1; j++)
            if(i&(1<<j))
                sum += t[N/2-1+j];
        if(freq.find(x-sum) != freq.end())
            cnt += freq[x-sum];
    }
    printf("%lld\n", cnt);
}
```

Monster Game I (dp convex hull trick + segtree)

Time limit: 1.00 s **Memory limit:** 512 MB

You are playing a game that consists of n levels. Each level has a monster. On levels $1, 2, \ldots, n-1$, you can either kill or escape the monster. However, on level n you must kill the final monster to win the game. Killing a monster takes sf time where s is the monster's strength and f is your skill factor (lower skill factor is better). After killing a monster, you get a new skill factor. What is the minimum total time in which you can win the game?

Input

The first input line has two integers n and x: the number of levels and your initial skill factor.

The second line has n integers $s_1, s_2, \ldots, s_n$: each monster's strength.

The third line has n integers $f_1, f_2, \ldots, f_n$: your new skill factor after killing a monster.

Output

Print one integer: the minimum total time to win the game.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x \le 10^6$
- $1 \le s_1 \le s_2 \le \cdots \le s_n \le 10^6$
- $x \ge f_1 \ge f_2 \ge \cdots \ge f_n \ge 1$

Example

Input:

```
5 100
20 30 30 50 90
90 60 20 20 10
```

Output:

```
4800
```

Explanation: The best way to play is to kill the third and fifth monster.

Solution

```
const int maxN = 2e5+1;
const int SIZE = 4e6;
const ll INF = 0x3f3f3f3f3f3f3f3f;
struct Line {
    ll m, b;
    ll operator()(const ll x) const {
        return m * x + b;
    }
} seg[SIZE];
int N, lo[SIZE], hi[SIZE];
ll X, s[maxN], f[maxN];
void build(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    seg[i] = {0, INF};
    if(l == r)  return;
    int m = (l+r)/2;
    build(2*i, l, m);
    build(2*i+1, m+1, r);
}
void insert(int i, Line L){
    int l = lo[i], r = hi[i];
    if(l == r){
        if(L(l) < seg[i](l))
            seg[i] = L;
        return;
    }
    int m = (l+r)/2;
    if(seg[i].m < L.m)  swap(seg[i], L);
    if(seg[i](m) > L(m)){
        swap(seg[i], L);
        insert(2*i, L);
    } else insert(2*i+1, L);
}
ll query(int i, ll x){
    int l = lo[i], r = hi[i];
    if(l == r)  return seg[i](x);
    int m = (l+r)/2;
    if(x < m)   return min(seg[i](x), query(2*i, x));
    else        return min(seg[i](x), query(2*i+1, x));
}
int main(){
    scanf("%d %lld", &N, &X);
    for(int i = 0; i < N; i++)  scanf("%lld", &s[i]);
    for(int i = 0; i < N; i++)  scanf("%lld", &f[i]);
    build(1, 1, 1e6);
    insert(1, {X, 0});
    for(int i = 0; i < N-1; i++){
        ll best = query(1, s[i]);
        insert(1, {f[i], best});
    }
    printf("%lld\n", query(1, s[N-1]));
}
```

Monster Game II (Li Chao Segment Tree)

Time limit: 1.00 s **Memory limit:** 512 MB

You are playing a game that consists of n levels. Each level has a monster. On levels $1, 2, \ldots, n-1$, you can either kill or escape the monster. However, on level n you must kill the final monster to win the game. Killing a monster takes sf time where s is the monster's strength and f is your skill factor. After killing a monster, you get a new skill factor (lower skill factor is better). What is the minimum total time in which you can win the game?

Input

The first input line has two integers n and x: the number of levels and your initial skill factor.

The second line has n integers $s_1, s_2, \ldots, s_n$: each monster's strength.

The third line has n integers $f_1, f_2, \ldots, f_n$: your new skill factor after killing a monster.

Output

Print one integer: the minimum total time to win the game.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x \le 10^6$
- $1 \le s_i, f_i \le 10^6$

Example

Input:

```
5 100
50 20 30 90 30
60 20 20 10 90
```

Output:

```
2600
```

Explanation: The best way to play is to kill the second and fifth monster.

Solution

```
const int maxN = 2e5+1;
const int SIZE = 4e6;
const ll INF = 0x3f3f3f3f3f3f3f3f;
struct Line {
    ll m, b;
    ll operator()(const ll x) const {
        return m * x + b;
    }
} seg[SIZE];
int N, lo[SIZE], hi[SIZE];
ll X, s[maxN], f[maxN];
void build(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    seg[i] = {0, INF};
    if(l == r)  return;
    int m = (l+r)/2;
    build(2*i, l, m);
    build(2*i+1, m+1, r);
}
```

```
void insert(int i, Line L){
    int l = lo[i], r = hi[i];
    if(l == r){
        if(L(l) < seg[i](l))
            seg[i] = L;
        return;
    }
    int m = (l+r)/2;
    if(seg[i].m < L.m)  swap(seg[i], L);
    if(seg[i](m) > L(m)){
        swap(seg[i], L);
        insert(2*i, L);
    } else insert(2*i+1, L);
}
ll query(int i, ll x){
    int l = lo[i], r = hi[i];
    if(l == r)  return seg[i](x);
    int m = (l+r)/2;
    if(x < m)   return min(seg[i](x), query(2*i, x));
    else        return min(seg[i](x), query(2*i+1, x));
}
int main(){
    scanf("%d %lld", &N, &X);
    for(int i = 0; i < N; i++)  scanf("%lld", &s[i]);
    for(int i = 0; i < N; i++)  scanf("%lld", &f[i]);
    build(1, 1, 1e6);
    insert(1, {X, 0});
    for(int i = 0; i < N-1; i++){
        ll best = query(1, s[i]);
        insert(1, {f[i], best});
    }
    printf("%lld\n", query(1, s[N-1]));
}
```

Necessary Cities (find articulation points , find all necessary cities)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities and m roads between them. There is a route between any two cities. A city is called necessary if there is no route between some other two cities after removing that city (and adjacent roads). Your task is to find all necessary cities.

Input

The first input line has two integers n and m: the number of cities and roads. The cities are numbered $1, 2, \ldots, n$.

After this, there are m lines that describe the roads. Each line has two integers a and b: there is a road between cities a and b. There is at most one road between two cities, and every road connects two distinct cities.

Output

First print an integer k: the number of necessary cities. After that, print a list of k cities. You may print the cities in any order.

Constraints

- $2 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
2
4 5
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 1e5+1;
bool vis[maxN], is_artic[maxN];
int N, M, timer, tin[maxN], low[maxN];
vector<int> ans, G[maxN];
void dfs(int u = 1, int p = -1){
    vis[u] = true;
    tin[u] = low[u] = ++timer;
    int children = 0;
    for(int v : G[u]){
        if(v != p){
            if(vis[v])  low[u] = min(low[u], tin[v]);
            else {
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if(low[v] >= tin[u] && p != -1){
                    if(!is_artic[u]){
                        ans.push_back(u);
                        is_artic[u] = true;
                    }
                }
                children++;
            }
        }
    }
    if(p == -1 && children > 1){
        if(!is_artic[u]){
            ans.push_back(u);
            is_artic[u] = true;
        }
    }
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
    }
    dfs();
    int K = (int) ans.size();
```

```
    printf("%d\n", K);
    for(int i = 0; i < K; i++)
        printf("%d%c", ans[i], (" \n")[i==K-1]);
}
```

Necessary Roads (Find all necessary roads in graph where removing such edge increases the number of components)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities and m roads between them. There is a route between any two cities. A road is called necessary if there is no route between some two cities after removing that road. Your task is to find all necessary roads.

Input

The first input line has two integers n and m: the number of cities and roads. The cities are numbered $1, 2, \ldots, n$.

After this, there are m lines that describe the roads. Each line has two integers a and b: there is a road between cities a and b. There is at most one road between two cities, and every road connects two distinct cities.

Output

First print an integer k: the number of necessary roads. After that, print k lines that describe the roads. You may print the roads in any order.

Constraints

- $2 \leq n \leq 10^5$
- $1 \leq m \leq 2 \cdot 10^5$
- $1 \leq a, b \leq n$

Example

Input:

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

Output:

```
2
3 5
4 5
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 1e5+1;
int N, M, timer, tin[maxN], low[maxN];
bool vis[maxN];
vector<int> G[maxN];
vector<pii> bridges;
void dfs(int u = 1, int p = 0){
    vis[u] = true;
    tin[u] = low[u] = ++timer;
    for(int v : G[u]){
```

```
        if(v != p){
            if(vis[v])  low[u] = min(low[u], tin[v]);
            else {
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if(low[v] > tin[u])
                    bridges.push_back({u, v});
            }
        }
    }
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
    }
    dfs();
    printf("%d\n", (int) bridges.size());
    for(pii P : bridges)
        printf("%d %d\n", P.first, P.second);
}
```

New Roads Queries (lct , for each query determine the earliest day two cities become connected.)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities in Byteland but no roads between them. However, each day, a new road will be built. There will be a total of m roads. Your task is to process q queries of the form: "after how many days can we travel from city a to city b for the first time?"

Input

The first input line has three integers n, m and q: the number of cities, roads and queries. The cities are numbered $1, 2, \ldots, n$.

After this, there are m lines that describe the roads in the order they are built. Each line has two integers a and b: there will be a road between cities a and b.

Finally, there are q lines that describe the queries. Each line has two integers a and b: we want to travel from city a to city b.

Output

For each query, print the number of days, or -1 if it is never possible.

Constraints

- $1 \le n, m, q \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

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

Output:

```
2
-1
4
```

Solution

```
const int maxN = 4e5+1;
int N, M, Q, x, y, root, idcounter;
struct Node {
    bool rev = false;
    Node *c[2] = {nullptr, nullptr}, *p = nullptr;
    int val, mx, id = ++idcounter;
    Node(int v){
        val = mx = v;
    }
    void push(){
        if(rev){
            swap(c[0], c[1]);
            if(c[0])    c[0]->rev ^= true;
            if(c[1])    c[1]->rev ^= true;
            rev = false;
        }
    }
    void pull(){
        mx = val;
        for(int i = 0; i < 2; i++)
            if(c[i])
                mx = max(mx, c[i]->mx);
    }
} *LCT[maxN];
bool notRoot(Node *t){
    return t->p && (t->p->c[0] == t || t->p->c[1] == t);
}
void rotate(Node *t){
    Node *p = t->p;
    bool b = (p->c[0] == t);
    if((t->p = p->p) && notRoot(p)) t->p->c[(t->p->c[1] == p)] = t;
    if((p->c[!b]=t->c[b]))          p->c[!b]->p = p;
    t->c[b] = p;
    p->p = t;
    p->pull();
}
void splay(Node *t){
    while(notRoot(t)){
        Node *p = t->p;
        p->push();
        t->push();
        rotate(t);
    }
    t->push();
```

```
    t->pull();
}
Node* access(Node *t){
    Node *last = nullptr;
    for(Node *u = t; u; u = u->p){
        splay(u);
        u->c[1] = last;
        last = u;
    }
    splay(t);
    return last;
}
void evert(Node *t){
    access(t);
    t->rev = true;
}
void link(Node *u, Node *v){
    evert(u);
    u->p = v;
}
void cut(Node *u, Node *v){
    evert(u);
    access(v);
    if(v->c[0]) v->c[0]->p = 0;
    v->c[0] = 0;
    v->pull();
}
Node* path(Node *u, Node *v){
    evert(u);
    access(v);
    return v;
}
Node* LCA(Node *u, Node *v){
    evert(LCT[root]);
    access(u);
    return access(v);
}
bool connected(Node *u, Node *v){
    path(u, v);
    while(v->c[0])
        v = v->c[0];
    return u == v;
}
int main(){
    scanf("%d %d %d", &N, &M, &Q);
    for(int i = 1; i <= N; i++)
        LCT[i] = new Node(0);
    for(int i = 1; i <= M; i++){
        scanf("%d %d", &x, &y);
        LCT[N+i] = new Node(i);
        if(!connected(LCT[x], LCT[y])){
            link(LCT[x], LCT[N+i]);
            link(LCT[y], LCT[N+i]);
        }
    }
    root = 1;
```

```
    for(int q = 0; q < Q; q++){
        scanf("%d %d", &x, &y);
        if(connected(LCT[x], LCT[y])){
            Node *p = path(LCT[x], LCT[y]);
            printf("%d\n", p->mx);
        } else printf("-1\n");
    }
}
```

One Bit Positions (fft k distance)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a binary string of length n. Your task is to calculate, for every k between $1 \ldots n-1$, the number of ways we can choose two positions i and j such that $i - j = k$ and there is a one-bit at both positions.

Input

The only input line has a string that consists only of characters 0 and 1.

Output

For every distance k between $1 \ldots n-1$ print the number of ways we can choose two such positions.

Constraints

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

Example

Input:

```
1001011010
```

Output:

```
1 2 3 0 2 1 0 1 0
```

Solution

```
typedef double ld;
typedef complex<ld> cd;
const int maxN = 2e5+5;
const int SIZE = 1<<19;
const ld PI = acos(-1);
char S[maxN];
vector<cd> A(SIZE), B(SIZE);
void fft(vector<cd> &a, bool inv){
    int N = (int) a.size();
    for(int i = 1, j = 0; i < N; i++){
        int bit = N>>1;
        for(; j&bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if(i < j)
            swap(a[i], a[j]);
    }
    for(int len = 2; len <= N; len <<= 1){
        ld theta = 2*PI / len * (inv ? -1 : 1);
        cd wlen(cos(theta), sin(theta));
```

```
        for(int i = 0; i < N; i += len){
            cd w(1);
            for(int j = 0; j < len / 2; j++){
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if(inv)
        for(cd &z : a)
            z /= N;
}
int main(){
    scanf(" %s", S);
    int N = (int) strlen(S);
    for(int i = 0; i < N; i++){
        if(S[i] == '1'){
            A[i] += 1;
            B[N-i-1] += 1;
        }
    }
    fft(A, false);
    fft(B, false);
    for(int i = 0; i < SIZE; i++)
        A[i] *= B[i];
    fft(A, true);
    for(int i = N; i < 2*N-1; i++)
        printf("%lld%c", llround(A[i].real()), (" \n")[i==2*N-2]);
}
```

Parcel Delivery (min cost max flow, min total cost to send k thing from city 1 to city n using directed roads with capacity and cost)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities and m routes through which parcels can be carried from one city to another city. For each route, you know the maximum number of parcels and the cost of a single parcel. You want to send k parcels from Syrjälä to Lehmälä. What is the cheapest way to do that?

Input

The first input line has three integers n, m and k: the number of cities, routes and parcels. The cities are numbered $1, 2, \ldots, n$. City 1 is Syrjälä and city n is Lehmälä.

After this, there are m lines that describe the routes. Each line has four integers a, b, r and c: there is a route from city a to city b, at most r parcels can be carried through the route, and the cost of each parcel is c.

Output

Print one integer: the minimum total cost or -1 if there are no solutions.

Constraints

- $2 \le n \le 500$
- $1 \le m \le 1000$
- $1 \le k \le 100$

- $1 \leq a, b \leq n$
- $1 \leq r, c \leq 1000$

Example

Input:

```
4 5 3
1 2 5 100
1 3 10 50
1 4 7 500
2 4 8 350
3 4 2 100
```

Output:

```
750
```

Explanation: One parcel is delivered through route $1 \to 2 \to 4$ (cost $1 \cdot 450 = 450$) and two parcels are delivered through route $1 \to 3 \to 4$ (cost $2 \cdot 150 = 300$).

Solution

```
const int maxN = 501, maxM = 1001;
const ll INF = 0x3f3f3f3f3f3f3f3f;
int N, M, K, p[maxN];
ll d[maxN], cap[maxN][maxN], cost[maxN][maxN];
bool inq[maxN];
vector<int> G[maxN];
struct Edge {
    int u, v;
    ll r, c;
} edges[maxM], redges[maxM];
void bellman_ford(){
    fill(inq+1, inq+N+1, false);
    fill(d+1, d+N+1, INF);
    fill(p+1, p+N+1, 0);
    queue<int> Q;
    Q.push(1);
    d[1] = 0;
    inq[1] = true;
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        inq[u] = false;
        for(int i : G[u]){
            Edge e = (i < 0 ? redges[-i] : edges[i]);
            if(e.r > 0 && d[e.v] > d[u] + e.c){
                d[e.v] = d[u] + e.c;
                p[e.v] = i;
                if(!inq[e.v]){
                    inq[e.v] = true;
                    Q.push(e.v);
                }
            }
        }
    }
}
ll minimum_cost_flow(){
    ll flow = 0, cost = 0;
```

```
    while(flow < K){
        bellman_ford();
        if(d[N] == INF) break;
        ll aug = K-flow;
        int u = N;
        while(u != 1){
            Edge e = (p[u] < 0 ? redges[-p[u]] : edges[p[u]]);
            aug = min(aug, e.r);
            u = e.u;
        }
        flow += aug;
        cost += aug * d[N];
        u = N;
        while(u != 1){
            if(p[u] < 0){
                redges[-p[u]].r -= aug;
                edges[-p[u]].r += aug;
            } else {
                redges[p[u]].r += aug;
                edges[p[u]].r -= aug;
            }
            u = (p[u] < 0 ? redges[-p[u]].u : edges[p[u]].u);
        }
    }
    return (flow < K ? -1 : cost);
}
int main(){
    scanf("%d %d %d", &N, &M, &K);
    for(int i = 1, u, v, r, c; i <= M; i++){
        scanf("%d %d %d %d", &u, &v, &r, &c);
        G[u].push_back(i);
        G[v].push_back(-i);
        edges[i] = {u, v, r, c};
        redges[i] = {v, u, 0, -c};
    }
    printf("%lld\n", minimum_cost_flow());
}
```

Reachability Queries (Strongly Connected Components)

Time limit: 1.00 s **Memory limit:** 512 MB

A directed graph consists of n nodes and m edges. The edges are numbered $1, 2, \ldots, n$. Your task is to answer q queries of the form "can you reach node b from node a?"

Input

The first input line has three integers n, m and q: the number of nodes, edges and queries.

Then there are m lines describing the edges. Each line has two distinct integers a and b: there is an edge from node a to node b.

Finally there are q lines describing the queries. Each line consists of two integers a and b: "can you reach node b from node a?"

Output

Print the answer for each query: either "YES" or "NO".

Constraints

- $1 \leq n \leq 5 \cdot 10^4$
- $1 \leq m, q \leq 10^5$

Example

Input:

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

Output:

```
YES
NO
YES
```

Solution

```
const int maxN = 5e4+5;
bool vis[maxN];
int N, M, Q, ds[maxN];
vector<int> order, component;
vector<int> G[maxN], H[maxN], SCC[maxN];
bitset<maxN> reachable[maxN];
int find(int u){
    if(ds[u] < 0)   return u;
    ds[u] = find(ds[u]);
    return ds[u];
}
bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v)  return false;
    if(ds[u] < ds[v]){
        ds[u] += ds[v];
        ds[v] = u;
    } else {
        ds[v] += ds[u];
        ds[u] = v;
    }
    return true;
}
void dfs1(int u){
    vis[u] = true;
    for(int v : G[u])
        if(!vis[v])
            dfs1(v);
    order.push_back(u);
}
void dfs2(int u){
    vis[u] = true;
    component.push_back(u);
```

```
    for(int v : H[u])
        if(!vis[v])
            dfs2(v);
}
void dfs3(int u){
    vis[u] = true;
    for(int v : SCC[u]){
        if(!vis[v])
            dfs3(v);
        reachable[u] |= reachable[v];
    }
}
int main(){
    scanf("%d %d %d", &N, &M, &Q);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        H[b].push_back(a);
    }
    fill(vis+1, vis+N+1, false);
    for(int i = 1; i <= N; i++)
        if(!vis[i])
            dfs1(i);
    fill(ds+1, ds+N+1, -1);
    fill(vis+1, vis+N+1, false);
    for(int i = 1; i <= N; i++){
        int u = order[N-i];
        if(!vis[u]){
            dfs2(u);
            for(int j = 0; j+1 < (int) component.size(); j++)
                merge(component[j], component[j+1]);
            component.clear();
        }
    }
    for(int i = 1; i <= N; i++){
        int rep = find(i);
        reachable[rep].set(i);
        for(int v : G[i])
            SCC[rep].push_back(find(v));
    }
    fill(vis+1, vis+N+1, false);
    for(int i = 1; i <= N; i++)
        if(!vis[i])
            dfs3(i);
    for(int i = 0, a, b; i < Q; i++){
        scanf("%d %d", &a, &b);
        a = find(a); b = find(b);
        printf(reachable[a][b] ? "YES\n" : "NO\n");
    }
}
```

Reachable Nodes (how many nodes are reachable from each node in a DAG using reverse topological order)

Time limit: 1.00 s **Memory limit:** 512 MB

A directed acyclic graph consists of n nodes and m edges. The nodes are numbered $1, 2, \ldots, n$. Calculate for each

node the number of nodes you can reach from that node (including the node itself).

Input

The first input line has two integers n and m: the number of nodes and edges.

Then there are m lines describing the edges. Each line has two distinct integers a and b: there is an edge from node a to node b.

Output

Print n integers: for each node the number of reachable nodes.

Constraints

- $1 \le n \le 5 \cdot 10^4$
- $1 \le m \le 10^5$

Example

Input:

```
5 6
1 2
1 3
1 4
2 3
3 5
4 5
```

Output:

```
5 3 2 2 1
```

Solution

```
const int maxN = 5e4+1;
int N, M, a, b, in[maxN];
bitset<maxN> ans[maxN];
vector<int> G[maxN];
queue<int> Q;
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        G[b].push_back(a);
        in[a]++;
    }
    for(int i = 1; i <= N; i++){
        if(in[i] == 0){
            ans[i].set(i);
            Q.push(i);
        }
    }
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        for(int v : G[u]){
            ans[v] |= ans[u];
            in[v]--;
            if(in[v] == 0){
```

```
                ans[v].set(v);
                Q.push(v);
            }
        }
    }
    for(int i = 1; i <= N; i++)
        printf("%d%c", (int) ans[i].count(), (" \n")[i==N]);
}
```

Reversals and Sums (lazy treap , subarray reverse + range sum)

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, you have to process following operations: - reverse a subarray - calculate the sum of values in a subarray

Input

The first input line has two integers n and m: the size of the array and the number of operations. The array elements are numbered $1, 2, \ldots, n$.

The next line as n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Finally, there are m lines that describe the operations. Each line has three integers t, a and b. If $t = 1$, you should reverse a subarray from a to b. If $t = 2$, you should calculate the sum of values from a to b.

Output

Print the answer to each operation where $t = 2$.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq m \leq 10^5$
- $0 \leq x_i \leq 10^9$
- $1 \leq a \leq b \leq n$

Example

Input:

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

Output:

```
8
9
```

Solution

```
const int maxN = 2e5+1;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> dist(1, (int) 2e9);
struct Node {
    bool rev;
    int prior, sz;
    ll val, sum;
```

```
    Node *l, *r;
    Node(){}
    Node(ll v, int p){
        prior = p;
        sum = val = v;
        sz = 1;
        l = r = nullptr;
    }
};
int N, M, type, a, b;
ll val[maxN];
Node *root;
int sz(Node *t){ return t ? t->sz : 0; }
ll sum(Node *t){ return t ? t->sum : 0; }
void flip(Node *t){
    if(!t)  return;
    t->rev ^= true;
}
void pull(Node *t){
    if(!t)  return;
    t->sz = sz(t->l) + sz(t->r) + 1;
    t->sum = sum(t->l) + sum(t->r) + t->val;
}
void push(Node *t){
    if(!t)  return;
    if(t->rev){
        swap(t->l, t->r);
        flip(t->l); flip(t->r);
        t->rev = false;
    }
}
Node* merge(Node *x, Node *y){
    if(!x || !y)    return x ? x : y;
    push(x); push(y);
    if(x->prior < y->prior){
        x->r = merge(x->r, y);
        pull(x);
        return x;
    } else {
        y->l = merge(x, y->l);
        pull(y);
        return y;
    }
}
pair<Node*,Node*> split(Node *x, int k){
    if(!x)  return {nullptr, nullptr};
    pair<Node*,Node*> y = {nullptr, nullptr};
    push(x);
    if(k <= sz(x->l)){
        y = split(x->l, k);
        x->l = y.second;
        pull(x);
        y.second = x;
    } else {
        y = split(x->r, k-sz(x->l)-1);
        x->r = y.first;
```

```
        pull(x);
        y.first = x;
    }
    return y;
}
void heapify(Node *t){
    if(!t)  return;
    Node *mx = t;
    if(t->l && t->l->prior > mx->prior) mx = t->l;
    if(t->r && t->r->prior > mx->prior) mx = t->r;
    if(mx != t){
        swap(t->prior, mx->prior);
        heapify(mx);
    }
}
Node* build(int x, int k){
    if(k == 0)  return nullptr;
    int mid = k/2;
    Node *t = new Node(val[x+mid], dist(rng));
    t->l = build(x, mid);
    t->r = build(x+mid+1, k-mid-1);
    heapify(t);
    pull(t);
    return t;
}
void reverse(int x, int k){
    pair<Node*,Node*> y, z;
    y = split(root, x-1);
    z = split(y.second, k);
    flip(z.first);
    y.second = merge(z.first, z.second);
    root = merge(y.first, y.second);
}
ll getSum(int x, int k){
    pair<Node*,Node*> y, z;
    y = split(root, x-1);
    z = split(y.second, k);
    ll ans = sum(z.first);
    y.second = merge(z.first, z.second);
    root = merge(y.first, y.second);
    return ans;
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0; i < N; i++)
        scanf("%lld", &val[i]);
    root = build(0, N);
    for(int i = 0; i < M; i++){
        scanf("%d %d %d", &type, &a, &b);
        int len = (b-a+1);
        if(type == 1)
            reverse(a, len);
        else if(type == 2)
            printf("%lld\n", getSum(a, len));
    }
}
```

Signal Processing (fft bedan)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given two integer sequences: a signal and a mask. Your task is to process the signal by moving the mask through the signal from left to right. At each mask position calculate the sum of products of aligned signal and mask values in the part where the signal and the mask overlap.

Input

The first input line consists of two integers n and m: the length of the signal and the length of the mask.

The next line consists of n integers $a_1, a_2, \ldots, a_n$ defining the signal.

The last line consists of m integers $b_1, b_2, \ldots, b_m$ defining the mask.

Output

Print $n + m - 1$ integers: the sum of products of aligned values at each mask position from left to right.

Constraints

- $1 \le n, m \le 2 \cdot 10^5$
- $1 \le a_i, b_i \le 100$

Example

Input:

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

Output:

```
3 11 13 10 16 9 4
```

Explanation: For example, at the second mask position the sum of aligned products is $2 \cdot 1 + 3 \cdot 3 = 11$.

Solution

```
typedef double ld;
typedef complex<ld> cd;
const int maxN = 2e5+5;
const int SIZE = 1<<19;
const ld PI = acos(-1);
int N, M;
vector<cd> A(SIZE), B(SIZE);
void fft(vector<cd> &a, bool inv){
    int n = (int) a.size();
    for(int i = 1, j = 0; i < n; i++){
        int bit = n>>1;
        for(; j&bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if(i < j)
            swap(a[i], a[j]);
    }
    for(int len = 2; len <= n; len <<= 1){
        ld theta = 2*PI / len * (inv ? -1 : 1);
        cd wlen(cos(theta), sin(theta));
        for(int i = 0; i < n; i += len){
```

```
            cd w(1);
            for(int j = 0; j < len / 2; j++){
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if(inv)
        for(cd &z : a)
            z /= n;
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a; i < N; i++){
        scanf("%d", &a);
        A[i] = a;
    }
    for(int i = 0, b; i < M; i++){
        scanf("%d", &b);
        B[M-i-1] = b;
    }
    fft(A, false);
    fft(B, false);
    for(int i = 0; i < SIZE; i++)
        A[i] *= B[i];
    fft(A, true);
    for(int i = 0; i < N+M-1; i++)
        printf("%lld%c", llround(A[i].real()), (" \n")[i==N+M-2]);
}
```

Subarray Squares (dp convex hull trick, Minimize the sum of squared subarray sums by partitioning the array into k parts)

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n elements, your task is to divide into k subarrays. The cost of each subarray is the square of the sum of the values in the subarray. What is the minimum total cost if you act optimally?

Input

The first input line has two integers n and k: the array elements and the number of subarrays. The array elements are numbered $1, 2, \ldots, n$.

The second line has n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print one integer: the minimum total cost.

Constraints

- $1 \le k \le n \le 3000$
- $1 \le x_i \le 10^5$

Example

Input:

```
8 3
2 3 1 2 2 3 4 1
```

Output:

```
110
```

Explanation: An optimal solution is $[2,3,1]$, $[2,2,3]$, $[4,1]$, whose cost is $(2+3+1)^2+(2+2+3)^2+(4+1)^2=110$.

Solution

```
typedef pair<ll,ll> pll;
const int maxN = 3005;
struct Line {
    ll m, b, c;
    ll operator()(ll x){
        return m * x + b;
    }
};
struct CHT {
    Line dq[2*maxN];
    int fptr, bptr;
    void clear(){
        dq[0] = {0, 0, 0};
        fptr = 0; bptr = 1;
    }
    bool pop_back(Line& L, Line& L1, Line& L2){
        ll v1 = (L.b - L2.b) * (L2.m - L1.m);
        ll v2 = (L2.m - L.m) * (L1.b - L2.b);
        return (v1 == v2 ? L.c > L1.c : v1 < v2);
    }
    bool pop_front(Line& L1, Line& L2, ll x){
        ll v1 = L1(x);
        ll v2 = L2(x);
        return (v1 == v2 ? L1.c < L2.c : v1 > v2);
    }
    void insert(Line L){
        while(bptr-fptr >= 2 && pop_back(L, dq[bptr-1], dq[bptr-2]))    bptr--;
        dq[bptr++] = L;
    }
    pll query(ll x){
        while(bptr-fptr >= 2 && pop_front(dq[fptr], dq[fptr+1], x))     fptr++;
        return {dq[fptr](x), dq[fptr].c};
    }
};
CHT cht;
int N, K, cnt[maxN];
ll pre[maxN], dp[maxN];
int main(){
    scanf("%d %d", &N, &K);
    for(int i = 1; i <= N; i++){
        scanf("%lld", &pre[i]);
        pre[i] += pre[i-1];
        dp[i] = pre[i]*pre[i];
    }
    for(int k = 1; k <= K-1; k++){
        cht.clear();
        for(int i = 1; i <= k; i++)
```

```
            cht.insert({-2*pre[i], dp[i]+pre[i]*pre[i], cnt[i]});
        for(int i = k+1; i <= N; i++){
            pll P = cht.query(pre[i]);
            cht.insert({-2*pre[i], dp[i]+pre[i]*pre[i], cnt[i]});
            dp[i] = pre[i]*pre[i] + P.first;
            cnt[i] = P.second + 1;
        }
    }
    printf("%lld\n", dp[N]);
}
```

Substring Reversals (treap, reverse substring)

Time limit: 1.00 s **Memory limit:** 512 MB

Given a string, your task is to process operations where you reverse a substring of the string. What is the final string after all the operations?

Input

The first input line has two integers n and m: the length of the string and the number of operations. The characters of the string are numbered $1, 2, \ldots, n$.

The next line has a string of length n that consists of characters A–Z.

Finally, there are m lines that describe the operations. Each line has two integers a and b: you reverse a substring from position a to position b.

Output

Print the final string after all the operations.

Constraints

- $1 \le n, m \le 2 \cdot 10^5$
- $1 \le a \le b \le n$

Example

Input:

```
7 2
AYBABTU
3 4
4 7
```

Output:

```
AYAUTBB
```

Solution

```
const int maxN = 2e5+5;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> dist(1, (int) 2e9);
struct Node {
    bool rev;
    char ch;
    int prior, sz;
    Node *l, *r;
    Node(){}
```

```
    Node(char c, int p){
        prior = p;
        ch = c;
        sz = 1;
        l = r = nullptr;
    }
};
int N, M, a, b;
char S[maxN];
Node *root;
int sz(Node *t){ return t ? t->sz : 0; }
void flip(Node *t){
    if(!t)  return;
    t->rev ^= true;
}
void pull(Node *t){
    if(!t)  return;
    t->sz = sz(t->l) + sz(t->r) + 1;
}
void push(Node *t){
    if(!t)  return;
    if(t->rev){
        swap(t->l, t->r);
        flip(t->l); flip(t->r);
        t->rev = false;
    }
}
Node* merge(Node *x, Node *y){
    if(!x || !y)    return x ? x : y;
    push(x); push(y);
    if(x->prior < y->prior){
        x->r = merge(x->r, y);
        pull(x);
        return x;
    } else {
        y->l = merge(x, y->l);
        pull(y);
        return y;
    }
}
pair<Node*,Node*> split(Node *x, int k){
    if(!x)  return {nullptr, nullptr};
    pair<Node*,Node*> y = {nullptr, nullptr};
    push(x);
    if(k <= sz(x->l)){
        y = split(x->l, k);
        x->l = y.second;
        pull(x);
        y.second = x;
    } else {
        y = split(x->r, k-sz(x->l)-1);
        x->r = y.first;
        pull(x);
        y.first = x;
    }
    return y;
```

```
}
void heapify(Node *t){
    if(!t)  return;
    Node *mx = t;
    if(t->l && t->l->prior > mx->prior) mx = t->l;
    if(t->r && t->r->prior > mx->prior) mx = t->r;
    if(mx != t){
        swap(t->prior, mx->prior);
        heapify(mx);
    }
}
Node* build(int x, int k){
    if(k == 0)  return nullptr;
    int mid = k/2;
    Node *t = new Node(S[x+mid], dist(rng));
    t->l = build(x, mid);
    t->r = build(x+mid+1, k-mid-1);
    heapify(t);
    pull(t);
    return t;
}
void reverse(int x, int k){
    pair<Node*,Node*> y, z;
    y = split(root, x-1);
    z = split(y.second, k);
    flip(z.first);
    y.second = merge(z.first, z.second);
    root = merge(y.first, y.second);
}
void print(Node *t){
    if(!t)  return;
    push(t);
    print(t->l);
    printf("%c", t->ch);
    print(t->r);
}
int main(){
    scanf("%d %d %s", &N, &M, S);
    root = build(0, N);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        int len = (b-a+1);
        reverse(a, len);
    }
    print(root);
}
```

Task Assignment (Minimum Cost Maximum Flow in a bipartite graph)

Time limit: 1.00 s **Memory limit:** 512 MB

A company has n employees and there are n tasks that need to be done. We know for each employee the cost of carrying out each task. Every employee should be assigned to exactly one task. What is the minimum total cost if we assign the tasks optimally and how could they be assigned?

Input

The first input line has one integer n: the number of employees and the number of tasks that need to be done.

After this, there are n lines each consisting of n integers. The ith line consists of integers $c_{i1}, c_{i2}, \ldots, c_{in}$: the cost of each task when it is assigned to the ith employee.

Output

First print the minimum total cost.

Then print n lines each consisting of two integers a and b: you assign the bth task to the ath employee.

If there are multiple solutions you can print any of them.

Constraints

- $1 \leq n \leq 200$
- $1 \leq c_{ij} \leq 1000$

Example

Input:

```
4
17 8 16 9
7 15 12 19
6 9 10 11
14 7 13 10
```

Output:

```
33
1 4
2 1
3 3
4 2
```

Explanation: The minimum total cost is 33. We can reach this by assigning employee 1 task 4, employee 2 task 1, employee 3 task 3 and employee 4 task 2. This will cost $9 + 7 + 10 + 7 = 33$.

Solution

```
const int maxN = 402, maxM = 40401;
const ll INF = 0x3f3f3f3f3f3f3f3f;
int N, p[maxN];
ll d[maxN], cap[maxN][maxN], cost[maxN][maxN];
bool inq[maxN], vis[maxM];
vector<int> path, G[maxN];
struct Edge {
    int u, v;
    ll r, c;
} edges[maxM], redges[maxM];
void bellman_ford(int start = 0){
    fill(inq, inq+maxN, false);
    fill(d, d+maxN, INF);
    fill(p, p+maxN, 0);
    queue<int> Q;
    Q.push(start);
    d[start] = 0;
    inq[start] = true;
    while(!Q.empty()){
```

```
        int u = Q.front(); Q.pop();
        inq[u] = false;
        for(int i : G[u]){
            Edge e = (i < 0 ? redges[-i] : edges[i]);
            if(e.r > 0 && d[e.v] > d[u] + e.c){
                d[e.v] = d[u] + e.c;
                p[e.v] = i;
                if(!inq[e.v]){
                    inq[e.v] = true;
                    Q.push(e.v);
                }
            }
        }
    }
}
ll minimum_cost_flow(){
    ll flow = 0, cost = 0;
    while(flow < N){
        bellman_ford();
        if(d[2*N+1] == INF) break;
        ll aug = N-flow;
        int u = 2*N+1;
        while(u != 0){
            Edge e = (p[u] < 0 ? redges[-p[u]] : edges[p[u]]);
            aug = min(aug, e.r);
            u = e.u;
        }
        flow += aug;
        cost += aug * d[2*N+1];
        u = 2*N+1;
        while(u != 0){
            if(p[u] < 0){
                redges[-p[u]].r -= aug;
                edges[-p[u]].r += aug;
            } else {
                redges[p[u]].r += aug;
                edges[p[u]].r -= aug;
            }
            u = (p[u] < 0 ? redges[-p[u]].u : edges[p[u]].u);
        }
    }
    return (flow < N ? -1 : cost);
}
void dfs(int u = 0){
    if(u == 2*N+1)  return;
    if(u != 0)      path.push_back(u);
    for(int i : G[u]){
        if(i > 0 && edges[i].r == 0 && !vis[i]){
            vis[i] = true;
            dfs(edges[i].v);
            return;
        }
    }
}
int main(){
    scanf("%d", &N);
```

```
    int edgeID = 1;
    for(int u = 1; u <= N; u++){
        for(int v = N+1, c; v <= 2*N; v++){
            scanf("%d", &c);
            G[u].push_back(edgeID);
            G[v].push_back(-edgeID);
            edges[edgeID] = {u, v, 1, c};
            redges[edgeID] = {v, u, 0, -c};
            edgeID++;
        }
    }
    for(int v = 1; v <= N; v++){
        G[0].push_back(edgeID);
        G[v].push_back(-edgeID);
        edges[edgeID] = {0, v, 1, 0};
        redges[edgeID] = {v, 0, 0, 0};
        edgeID++;
    }
    for(int u = N+1; u <= 2*N; u++){
        G[u].push_back(edgeID);
        G[2*N+1].push_back(-edgeID);
        edges[edgeID] = {u, 2*N+1, 1, 0};
        redges[edgeID] = {2*N+1, u, 0, 0};
        edgeID++;
    }
    printf("%lld\n", minimum_cost_flow());
    for(int i = 0; i < N; i++){
        path.clear();
        dfs();
        printf("%d %d\n", path[0], path[1]-N);
    }
}
```

Bitwise Operations

Maximum Xor Subarray

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to find the maximum xor sum of a subarray.

Input

The first line has an integer n: the size of the array.

The next line has n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print one integer: the maximum xor sum in a subarray.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $0 \le x_i \le 10^9$

Example

Input:

```
4
5 1 5 9
```

Output:

```
13
```

Solution

```
struct Node {
    Node *c[2];
};
int N, xum, best;
Node *root;
void update(int x){
    Node *cur = root;
    for(int i = 30; i >= 0; i--){
        if(x&(1<<i)){
            if(!cur->c[1])  cur->c[1] = new Node();
            cur = cur->c[1];
        } else {
            if(!cur->c[0])  cur->c[0] = new Node();
            cur = cur->c[0];
        }
    }
}
int query(int x){
    int res = 0;
    Node *cur = root;
    for(int i = 30; i >= 0; i--){
        if(x&(1<<i)){
            if(cur->c[0]){
                res += (1<<i);
                cur = cur->c[0];
            } else cur = cur->c[1];
        } else {
            if(cur->c[1]){
                res += (1<<i);
                cur = cur->c[1];
            } else cur = cur->c[0];
        }
    }
    return res;
}
int main(){
    scanf("%d", &N);
    root = new Node();
    update(0);
    for(int i = 0, x; i < N; i++){
        scanf("%d", &x);
        xum ^= x;
        update(xum);
        best = max(best, query(xum));
    }
    printf("%d\n", best);
}
```

Xor Pyramid Peak

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a xor pyramid where each number is the xor of lower-left and lower-right numbers. Here is an example pyramid:

Given the bottom row of the pyramid, your task is to find the topmost number.

Input

The first line has an integer n: the size of the pyramid.

The next line has n integers $a_1, a_2, \ldots, a_n$: the bottom row of the pyramid.

Output

Print one integer: the topmost number.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le a_i \le 10^9$

Example

Input:

```
8
2 10 5 12 9 5 1 5
```

Output:

```
9
```

Solution

```
const int maxN = 2e5+5;
int N, D, a, xum;
int two_divisibility(int x){
    int res = 0;
    while(x){
        x >>= 1;
        res += x;
    }
    return res;
}
int main(){
    scanf("%d", &N);
    D = two_divisibility(N-1);
    for(int i = 0; i < N; i++){
        scanf("%d", &a);
        int d1 = two_divisibility(i);
        int d2 = two_divisibility(N-i-1);
        if(D - d1 - d2 == 0)
            xum ^= a;
    }
    printf("%d\n", xum);
}
```

Construction Problems

Chess Tournament

Time limit: 1.00 s **Memory limit:** 512 MB

There will be a chess tournament of n players. Each player has announced the number of games they want to play. Each pair of players can play at most one game. Your task is to determine which games will be played so that everybody will be happy.

Input

The first input line has an integer n: the number of players. The players are numbered $1, 2, \ldots, n$.

The next line has n integers $x_1, x_2, \ldots, x_n$: for each player, the number of games they want to play.

Output

First print an integer k: the number of games. Then, print k lines describing the games. You can print any valid solution.

If there are no solutions, print "IMPOSSIBLE".

Constraints

- $1 \le n \le 10^5$
- $\sum_{i=1}^{n} x_i \le 2 \cdot 10^5$

Example

Input:

```
5
1 3 2 0 2
```

Output:

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

Solution

```
##include <ext/pb_ds/assoc_container.hpp>
##include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef pair<int,int> pii;
int N, K;
vector<pii> added, ans;
tree<pii, null_type, less<pii>, rb_tree_tag,
tree_order_statistics_node_update> T;
int main(){
    scanf("%d", &N);
    for(int i = 1, x; i <= N; i++){
        scanf("%d", &x);
        T.insert({x, i});
    }
    for(int i = N-1; i >= 0; i--){
        pii P = *T.find_by_order(i);
```

```
            T.erase(T.find_by_order(i));
            int degu = P.first;
            int u = P.second;
            added.clear();
            for(int j = 1; j <= degu; j++){
                if(i-j < 0 || (*T.find_by_order(i-j)).first == 0){
                    printf("IMPOSSIBLE\n");
                    return 0;
                }
                pii Q = *T.find_by_order(i-j);
                T.erase(T.find_by_order(i-j));
                int degv = Q.first;
                int v = Q.second;
                ans.push_back({min(u, v), max(u, v)});
                added.push_back({degv-1, v});
            }
            for(pii Q : added)
                T.insert({Q.first, Q.second});
        }
        sort(ans.begin(), ans.end());
        K = (int) ans.size();
        printf("%d\n", K);
        for(int i = 0; i < K; i++)
            printf("%d %d\n", ans[i].first, ans[i].second);
}
```

Inverse Inversions

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to create a permutation of numbers $1, 2, \ldots, n$ that has exactly k inversions. An inversion is a pair (a, b) where $a < b$ and $p_a > p_b$ where p_i denotes the number at position i in the permutation.

Input

The only input line has two integers n and k.

Output

Print a line that contains the permutation. You can print any valid solution.

Constraints

- $1 \le n \le 10^6$
- $0 \le k \le \frac{n(n-1)}{2}$

Example

Input:

```
5 4
```

Output:

```
1 5 2 4 3
```

Solution

```
const int maxN = 1e6+1;
int N, idx, ans[maxN];
```

```
ll K;
deque<int> DQ;
int main(){
    scanf("%d %lld", &N, &K);
    for(int i = 1; i <= N; i++)
        DQ.push_back(i);
    for(int k = N-1; k >= 0; k--){
        if(k <= K){
            ans[++idx] = DQ.back();
            DQ.pop_back();
            K -= k;
        } else {
            ans[++idx] = DQ.front();
            DQ.pop_front();
        }
    }
    for(int i = 1; i <= idx; i++)
        printf("%d%c", ans[i], (" \n")[i==idx]);
}
```

Monotone Subsequences

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to create a permutation of numbers $1, 2, \ldots, n$ whose longest monotone subsequence has exactly k elements. A monotone subsequence is either increasing or decreasing. For example, some monotone subsequences in $[2, 1, 4, 5, 3]$ are $[2, 4, 5]$ and $[4, 3]$.

Input

The first input line has an integer t: the number of tests.

After this, there are t lines. Each line has two integers n and k.

Output

For each test, print a line that contains the permutation. You can print any valid solution. If there are no solutions, print IMPOSSIBLE.

Constraints

- $1 \le t \le 1000$
- $1 \le k \le n \le 100$

Example

Input:

```
3
5 3
5 2
7 7
```

Output:

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

Solution

```
int N, K;
void solve(){
    scanf("%d %d", &N, &K);
    if(K*K < N){
        printf("IMPOSSIBLE\n");
        return;
    }
    vector<int> ans;
    int l = 1, r = K;
    while(true){
        for(int i = r; i >= l; i--)
            ans.push_back(i);
        if(r == N)  break;
        l = r+1;
        r = min(r+K, N);
    }
    for(int i = 0; i < N; i++)
        printf("%d%c", ans[i], (" \n")[i==N-1]);
}
int main(){
    int T;
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        solve();
    }
}
```

Counting Problems

Counting Bishops (ncr Stirling)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to count the number of ways k bishops can be placed on an $n \times n$ chessboard so that no two bishops attack each other. Two bishops attack each other if they are on the same diagonal.

Input

The only input line has two integers n and k: the board size and the number of bishops.

Output

Print one integer: the number of ways modulo $10^9 + 7$.

Constraints

- $1 \le n \le 500$
- $1 \le k \le n^2$

Example

Input:

```
5 4
```

Output:

```
2728
```

Solution

```
const int maxN = 501;
const ll MOD = 1e9+7;
int N, K;
ll fac[maxN], inv[maxN];
ll T[maxN], B[2*maxN], W[2*maxN];
ll fastpow(ll x, ll b){
    ll res = 1;
    while(b){
        if(b&1) res = (res * x) % MOD;
        x = (x * x) % MOD;
        b >>= 1;
    }
    return res;
}
ll choose(int n, int k){
    if(k < 0 || n < k)  return 0;
    return fac[n] * inv[k] % MOD * inv[n-k] % MOD;
}
ll stirling2(int n, int k){
    ll sum = 0;
    for(int i = 0; i <= k; i++){
        ll a = (i&1 ? -1 : 1);
        ll b = choose(k, i);
        ll c = fastpow(k-i, n);
        ll term = a * b % MOD * c % MOD;
        sum = (sum + term + MOD) % MOD;
    }
    return sum * inv[k] % MOD;
}
void compute_T(int n){
    for(int k = 0; k <= n; k++)
        T[k] = stirling2(n+1, n+1-k);
}
void init_choose(){
    fac[0] = inv[0] = 1;
    for(int i = 1; i < maxN; i++){
        fac[i] = (fac[i-1] * i) % MOD;
        inv[i] = fastpow(fac[i], MOD-2);
    }
}
ll calculate(int n, int k){
    if(n == 1)      return 1;
    if(k >= 2*n)    return 0;
    memset(B, 0, sizeof(B));
    memset(W, 0, sizeof(W));
    int m = n/2, odd = n&1;
    for(int k = 0; k <= m+odd; k++){
        compute_T(2*m-1-k+odd);
        ll a = choose(m+odd, k);
        ll b = choose(m, k);
        for(int i = 0; i+k < n; i++){
            B[i+k] += (a * T[i]) % MOD;
            B[i+k] %= MOD;
            W[i+k] += (b * T[i]) % MOD;
            W[i+k] %= MOD;
```

```
        }
    }
    ll sum = 0;
    for(int b = 0; b <= min(N-1, K); b++){
        int w = K-b;
        sum += (B[b] * W[w]) % MOD;
        sum %= MOD;
    }
    return sum;
}
int main(){
    init_choose();
    scanf("%d %d", &N, &K);
    printf("%lld\n", calculate(N, K));
}
```

Counting Permutations (counting dp number of perms where where no two adjacent elements differ by 1)

Time limit: 1.00 s **Memory limit:** 512 MB

A permutation of integers $1, 2, \ldots, n$ is called beautiful if there are no adjacent elements whose difference is 1. Given n, your task is to count the number of beautiful permutations.

Input

The only input line contains an integer n.

Output

Print the number of beautiful permutations of $1, 2, \ldots, n$ modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 1000$

Example

Input:

```
5
```

Output:

```
14
```

Solution

```
const int maxN = 1000;
const ll MOD = 1e9+7;
int N;
ll dp[maxN+1];
void init(){
    dp[0] = dp[1] = 1;
    dp[2] = dp[3] = 0;
    for(int i = 4; i <= maxN; i++){
        ll a = (i+1) * dp[i-1] % MOD;
        ll b = (i-2) * dp[i-2] % MOD;
        ll c = (i-5) * dp[i-3] % MOD;
        ll d = (i-3) * dp[i-4] % MOD;
```

```
        dp[i] = (a-b-c+d);
        while(dp[i] < 0)
            dp[i] += MOD;
        dp[i] %= MOD;
    }
}
int main(){
    init();
    scanf("%d", &N);
    printf("%lld\n", dp[N]);
}
```

Counting Sequences (count the number of sequences of length n with values from 1 to k, such that each number from 1 to k appears at least once.)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to count the number of sequences of length n where each element is an integer between $1 \dots k$ and each integer between $1 \dots k$ appears at least once in the sequence. For example, when $n = 6$ and $k = 4$, some valid sequences are $[1, 3, 1, 4, 3, 2]$ and $[2, 2, 1, 3, 4, 2]$.

Input

The only input line has two integers n and k.

Output

Print one integer: the number of sequences modulo $10^9 + 7$.

Constraints

- $1 \le k \le n \le 10^6$

Example

Input:

```
6 4
```

Output:

```
1560
```

Solution

```
const int maxN = 1e6+1;
const ll MOD = 1e9+7;
int N, K;
ll fac[maxN], inv[maxN];
ll fastpow(ll x, ll b){
    ll res = 1;
    while(b){
        if(b&1) res = (res * x) % MOD;
        x = (x * x) % MOD;
        b >>= 1;
    }
    return res;
}
void init_choose(){
    fac[0] = inv[0] = 1;
```

```
    for(int i = 1; i < maxN; i++){
        fac[i] = (fac[i-1] * i) % MOD;
        inv[i] = fastpow(fac[i], MOD-2);
    }
}
ll choose(int n, int k){
    if(k < 0 || k > n)  return 0;
    return fac[n] * inv[k] % MOD * inv[n-k] % MOD;
}
ll T(int n, int k){
    ll sum = 0;
    for(int j = 0; j <= k; j++){
        ll a = (j&1 ? -1 : 1);
        ll b = choose(k, j);
        ll c = fastpow(k-j, n);
        ll term = (a * b % MOD * c % MOD + MOD) % MOD;
        sum = (sum + term) % MOD;
    }
    return sum;
}
int main(){
    init_choose();
    scanf("%d %d", &N, &K);
    printf("%lld\n", T(N, K));
}
```

Empty String (Count the number of ways to fully remove a string by recursively deleting adjacent equal character (DP + combinatorics))

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a string consisting of n characters between a and z. On each turn, you may remove any two adjacent characters that are equal. Your goal is to construct an empty string by removing all the characters. In how many ways can you do this?

Input

The only input line has a string of length n.

Output

Print one integer: the number of ways modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 500$

Example

Input:

```
aabccb
```

Output:

```
3
```

Solution

```
const int maxN = 505;
const ll MOD = 1e9+7;
int N;
char S[maxN];
ll fac[maxN], inv[maxN], dp[maxN][maxN];
ll fastpow(ll x, ll b){
    ll res = 1;
    while(b){
        if(b&1) res = (res * x) % MOD;
        x = (x * x) % MOD;
        b >>= 1;
    }
    return res;
}
ll choose(int n, int k){
    if(k < 0 || n < k)  return 0;
    return fac[n] * inv[k] % MOD * inv[n-k] % MOD;
}
void init(){
    fac[0] = inv[0] = 1;
    for(int i = 1; i < maxN; i++){
        fac[i] = (fac[i-1] * i) % MOD;
        inv[i] = fastpow(fac[i], MOD-2);
    }
    for(int i = 0; i < maxN; i++)
        for(int j = i; j < maxN; j++)
            dp[i][j] = -1;
}
ll solve(int l, int r){
    if((r-l+1)&1)       return 0;
    if(l > r)           return 1;
    if(dp[l][r] != -1)  return dp[l][r];
    ll cnt = 0;
    for(int m = l+1; m <= r; m++){
        if(S[l] == S[m]){
            ll subcases = solve(l+1, m-1) * solve(m+1, r) % MOD;
            ll aftercombine = subcases * choose((r-l+1)/2, (m-l+1)/2) % MOD;
            cnt = (cnt + aftercombine) % MOD;
        }
    }
    return dp[l][r] = cnt;
}
int main(){
    scanf(" %s", S);
    N = (int) strlen(S);
    init();
    printf("%lld\n", N&1 ? 0 : solve(0, N-1));
}
```

Functional Graph Distribution (stirling numbers Count the number of functional graphs with n nodes and k components)

Time limit: 1.00 s **Memory limit:** 512 MB

A functional graph is a directed graph where each node has outdegree 1. For example, here is a functional graph

that has 9 nodes and 2 components:

Given n, your task is to calculate for each $k = 1 \ldots n$ the number of functional graphs that have n nodes and k components.

Input

The only input line has an integer n: the number of nodes.

Output

Print n lines: for each $k = 1 \ldots n$ the number of graphs modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 5000$

Example

Input:

```
3
```

Output:

```
17
9
1
```

Solution

```
const int maxN = 5001;
const ll MOD = 1e9+7;
int N;
ll pown[maxN], fac[maxN], inv[maxN], S[maxN][maxN];
ll inverse(ll x){
    ll res = 1;
    ll b = MOD-2;
    while(b){
        if(b&1) res = (res * x) % MOD;
        x = (x * x) % MOD;
        b >>= 1;
    }
    return res;
}
void init_powers(){
    pown[0] = 1;
    for(int i = 1; i < maxN; i++)
        pown[i] = (pown[i-1] * N) % MOD;
}
void init_choose(){
    fac[0] = inv[0] = 1;
    for(int i = 1; i < maxN; i++){
        fac[i] = (fac[i-1] * i) % MOD;
        inv[i] = inverse(fac[i]);
    }
}
void init_stirling(){
    S[1][1] = 1;
    for(int n = 2; n < maxN; n++)
```

```
        for(int k = 1; k <= n; k++)
            S[n][k] = (S[n-1][k-1] - (n-1) * S[n-1][k]) % MOD;
}
ll choose(int n, int k){
    if(k < 0 || k > n)  return 0;
    return fac[n] * inv[k] % MOD * inv[n-k] % MOD;
}
ll stirling1(int n, int k){
    return abs(S[n][k]);
}
ll T(int n, int k){
    ll sum = 0;
    for(int j = 0; j <= n-1; j++){
        ll a = choose(n-1, j);
        ll b = pown[n-1-j];
        ll c = stirling1(j+1, k);
        sum += a * b % MOD * c % MOD;
        sum %= MOD;
    }
    return sum;
}
int main(){
    scanf("%d", &N);
    init_powers();
    init_choose();
    init_stirling();
    for(int k = 1; k <= N; k++)
        printf("%lld\n", T(N, k));
}
```

Grid Completion (Count the number of ways to complete n × n grid so that every row and col contains exactly one A and one B,)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to create an $n \times n$ grid whose each row and column has exactly one A and B. Some of the characters have already been placed. In how many ways can you complete the grid?

Input

The first input line has an integer n: the size of the grid.

After this, there are n lines that describe the grid. Each line has n characters: . means an empty square, and A and B show the characters already placed.

You can assume that every row and column has at most one A and B.

Output

Print one integer: the number of ways modulo $10^9 + 7$.

Constraints

- $2 \leq n \leq 500$

Example

Input:

```
5
.....
..AB.
.....
B....
...A.
```

Output:

```
16
```

Solution

```
const ll MOD = 1e9+7;
const int maxN = 505;
int N, p[maxN], q[maxN], C[6];
bool inp[maxN], inq[maxN];
char S[maxN];
ll fact[maxN], inv[maxN];
ll inverse(ll x){
    ll res = 1;
    ll b = MOD-2;
    while(b){
        if(b&1) res = (res * x) % MOD;
        x = (x * x) % MOD;
        b >>= 1;
    }
    return res;
}
ll choose(int x, int y){
    return (fact[x] * inv[y] % MOD) * inv[x-y] % MOD;
}
void init(){
    fact[0] = inv[0] = 1;
    for(int i = 1; i <= N; i++){
        fact[i] = (fact[i-1] * i) % MOD;
        inv[i] = (inv[i-1] * inverse(i)) % MOD;
    }
}
ll f(int i, int j, int k){
    ll res = (choose(C[0], i)
            * choose(C[1], j) % MOD
            * choose(C[2], k) % MOD
            * choose(C[3], i) % MOD);
    res = (res * fact[i] % MOD
               * fact[C[4]-i-j] % MOD
               * fact[C[5]-i-k] % MOD);
    if((i+j+k) % 2 == 1)
        res = (MOD - res);
    return res;
}
int main(){
    scanf("%d", &N);
    init();
    for(int i = 0; i < N; i++){
        scanf(" %s", S);
        p[i] = q[i] = -1;
        for(int j = 0; j < N; j++){
```

```
            if(S[j] == 'A') { p[i] = j; inp[j] = true; }
            if(S[j] == 'B') { q[i] = j; inq[j] = true; }
        }
    }
    for(int i = 0; i < N; i++){
        if(p[i] == -1 && q[i] == -1)                C[0]++;
        if(p[i] == -1 && q[i] != -1 && !inp[q[i]])  C[1]++;
        if(p[i] != -1 && q[i] == -1 && !inq[p[i]])  C[2]++;
    }
    for(int i = 0; i < N; i++){
        if(!inp[i] && !inq[i])  C[3]++;
        if(!inp[i])             C[4]++;
        if(!inq[i])             C[5]++;
    }
    ll ans = 0;
    for(int i = 0; i <= min(C[0], C[3]); i++)
        for(int j = 0; j <= C[1]; j++)
            for(int k = 0; k <= C[2]; k++)
                ans = (ans + f(i, j, k)) % MOD;
    printf("%lld\n", ans);
}
```

Grid Paths II (Count the number of right-down paths from (1,1) to (n,n) avoiding m traps)

Time limit: 1.00 s **Memory limit:** 512 MB

Consider an $n \times n$ grid whose top-left square is $(1, 1)$ and bottom-right square is (n, n). Your task is to move from the top-left square to the bottom-right square. On each step you may move one square right or down. In addition, there are m traps in the grid. You cannot move to a square with a trap. What is the total number of possible paths?

Input

The first input line contains two integers n and m: the size of the grid and the number of traps.

After this, there are m lines describing the traps. Each such line contains two integers y and x: the location of a trap.

You can assume that there are no traps in the top-left and bottom-right square.

Output

Print the number of paths modulo $10^9 + 7$.

Constraints

- $1 \le n \le 10^6$
- $1 \le m \le 1000$
- $1 \le y, x \le n$

Example

Input:

```
3 1
2 2
```

Output:

```
2
```

Solution

```
const int maxN = 2e6+1;
const int maxM = 1002;
const ll MOD = 1e9+7;
int N, M, x, y;
ll fact[maxN], inv[maxN], dp[maxM];
struct point {int x, y;} p[maxM];
ll inverse(ll a){
    ll res = 1;
    ll b = MOD-2;
    while(b > 0){
        if(b&1)
            res = (res * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return res;
}
void init(){
    fact[0] = inv[0] = 1;
    for(int i = 1; i < maxN; i++){
        fact[i] = i * fact[i-1] % MOD;
        inv[i] = inverse(fact[i]);
    }
}
ll choose(int n, int k){
    return fact[n] * inv[n-k] % MOD * inv[k] % MOD;
}
int main(){
    init();
    scanf("%d %d", &N, &M);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &x, &y);
        p[i] = {x, y};
    }
    p[M] = {N, N};
    sort(p, p+M+1, [](point a, point b){
        return a.x == b.x ? a.y < b.y : a.x < b.x;
    });
    for(int i = 0; i <= M; i++){
        dp[i] = choose(p[i].x+p[i].y-2, p[i].x-1);
        ll subtract = 0;
        for(int j = 0; j < i; j++){
            if(p[i].x >= p[j].x && p[i].y >= p[j].y){
                int dx = p[i].x-p[j].x, dy = p[i].y-p[j].y;
                subtract += dp[j] * choose(dx+dy, dx);
                subtract %= MOD;
            }
        }
        dp[i] = (dp[i] - subtract + MOD) % MOD;
    }
    printf("%lld\n", dp[M]);
}
```

Permutation Inversions (Count the number of permutations of size n with exactly k inversions using dp on permutation inversion counts)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to count the number of permutations of $1, 2, \ldots, n$ that have exactly k inversions (i.e., pairs of elements in the wrong order). For example, when $n = 4$ and $k = 3$, there are 6 such permutations: - $[1, 4, 3, 2]$ - $[2, 3, 4, 1]$ - $[2, 4, 1, 3]$ - $[3, 1, 4, 2]$ - $[3, 2, 1, 4]$ - $[4, 1, 2, 3]$

Input

The only input line has two integers n and k.

Output

Print the answer modulo $10^9 + 7$.

Constraints

- $1 \le n \le 500$
- $0 \le k \le \frac{n(n-1)}{2}$

Example

Input:

```
4 3
```

Output:

```
6
```

Solution

```
const int maxN = 501;
const int maxK = maxN*(maxN-1)/2;
const ll MOD = 1e9+7;
int N, K;
ll dp[maxN][maxK];
void init(){
    for(int i = 1; i < maxN; i++){
        int r = i*(i-1)/2;
        dp[i][0] = dp[i][r] = 1;
        for(int j = 1; j <= r/2; j++){
            dp[i][j] = (dp[i-1][j] + dp[i][j-1]) % MOD;
            if(j-i >= 0)
                dp[i][j] = (dp[i][j] - dp[i-1][j-i] + MOD) % MOD;
        }
        for(int j = r/2+1; j < r; j++)
            dp[i][j] = dp[i][r-j];
    }
}
int main(){
    init();
    scanf("%d %d", &N, &K);
    printf("%lld\n", dp[N][K]);
}
```

Dynamic Programming Mosayed

Array Description [DP on Sequences] (Counts ways to fill an array prefix ending at index i with value v, based on valid transitions from i-1.)

Time limit: 1.00 s **Memory limit:** 512 MB

You know that an array has n integers between 1 and m, and the absolute difference between two adjacent values is at most 1. Given a description of the array where some values may be unknown, your task is to count the number of arrays that match the description.

Input

The first input line has two integers n and m: the array size and the upper bound for each value.

The next line has n integers $x_1, x_2, \ldots, x_n$: the contents of the array. Value 0 denotes an unknown value.

Output

Print one integer: the number of arrays modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq m \leq 100$
- $0 \leq x_i \leq m$

Example

Input:

```
3 5
2 0 2
```

Output:

```
3
```

Explanation: The arrays $[2, 1, 2]$, $[2, 2, 2]$ and $[2, 3, 2]$ match the description.

Solution

```
const int maxN = 1e5;
const int maxM = 100;
const ll MOD = 1e9+7;
int N, M, x[maxN+1];
ll ans, dp[maxN+1][maxM+1];
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 1; i <= N; i++)
        scanf("%d", &x[i]);
    if(x[1])
        dp[1][x[1]] = 1;
    else
        for(int i = 1; i <= M; i++)
            dp[1][i] = 1;
    for(int i = 2; i <= N; i++){
        for(int j = 1; j <= M; j++){
            dp[i][j] = dp[i-1][j];
            if(j != 1)  dp[i][j] += dp[i-1][j-1];
            if(j != M)  dp[i][j] += dp[i-1][j+1];
```

```
            dp[i][j] %= MOD;
        }
        if(x[i])
            for(int j = 0; j <= M; j++)
                if(j != x[i])
                    dp[i][j] = 0;
    }
    for(int i= 1; i <= M; i++)
        ans = (ans + dp[N][i]) % MOD;
    printf("%lld\n", ans);
}
```

Book Shop [0/1 Knapsack] (Maximizes pages for a given budget by deciding for each book whether to include it or not.)

Time limit: 1.00 s **Memory limit:** 512 MB

You are in a book shop which sells n different books. You know the price and number of pages of each book. You have decided that the total price of your purchases will be at most x. What is the maximum number of pages you can buy? You can buy each book at most once.

Input

The first input line contains two integers n and x: the number of books and the maximum total price.

The next line contains n integers $h_1, h_2, \ldots, h_n$: the price of each book.

The last line contains n integers $s_1, s_2, \ldots, s_n$: the number of pages of each book.

Output

Print one integer: the maximum number of pages.

Constraints

- $1 \leq n \leq 1000$
- $1 \leq x \leq 10^5$
- $1 \leq h_i, s_i \leq 1000$

Example

Input:

```
4 10
4 8 5 3
5 12 8 1
```

Output:

```
13
```

Explanation: You can buy books 1 and 3. Their price is $4 + 5 = 9$ and the number of pages is $5 + 8 = 13$.

Solution

```
const int maxN = 1000;
const int maxX = 1e5;
int N, X, h[maxN], s[maxN], dp[maxX+1];
int main(){
    scanf("%d %d", &N, &X);
    for(int i = 0; i < N; i++)  scanf("%d", &h[i]);
```

```
    for(int i = 0; i < N; i++)  scanf("%d", &s[i]);
    fill(dp+1, dp+X+1, -1);
    for(int i = 0; i < N; i++)
        for(int j = X-h[i]; j >= 0; j--)
            if(dp[j] != -1)
                dp[j+h[i]] = max(dp[j+h[i]], dp[j]+s[i]);
    for(int i = 1; i <= X; i++)
        dp[i] = max(dp[i], dp[i-1]);
    printf("%d\n", dp[X]);
}
```

Coin Combinations I [Unbounded Knapsack] (Counts permutations of coins that form a sum by iterating through sums and then coins.)

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a money system consisting of n coins. Each coin has a positive integer value. Your task is to calculate the number of distinct ways you can produce a money sum x using the available coins. For example, if the coins are $\{2, 3, 5\}$ and the desired sum is 9, there are 8 ways: - $2+2+5$ - $2+5+2$ - $5+2+2$ - $3+3+3$ - $2+2+2+3$ - $2+2+3+2$ - $2+3+2+2$ - $3+2+2+2$

Input

The first input line has two integers n and x: the number of coins and the desired sum of money.

The second line has n distinct integers $c_1, c_2, \ldots, c_n$: the value of each coin.

Output

Print one integer: the number of ways modulo $10^9 + 7$.

Constraints

- $1 \le n \le 100$
- $1 \le x \le 10^6$
- $1 \le c_i \le 10^6$

Example

Input:

```
3 9
2 3 5
```

Output:

```
8
```

Solution

```
const int maxN = 100;
const int maxX = 1e6;
const ll MOD = 1e9+7;
int N, X, c[maxN];
ll dp[maxX+1];
int main(){
    scanf("%d %d", &N, &X);
    for(int i = 0; i < N; i++)
        scanf("%d", &c[i]);
    dp[0] = 1;
```

```
    for(int i = 0; i < X; i++)
        if(dp[i] != 0)
            for(int j = 0; j < N; j++)
                if(i+c[j] <= X)
                    dp[i+c[j]] = (dp[i+c[j]] + dp[i]) % MOD;
    printf("%lld\n", dp[X]);
}
```

Coin Combinations II [Unbounded Knapsack] (Counts unique combinations of coins that form a sum by iterating through coins and then sums.)

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a money system consisting of n coins. Each coin has a positive integer value. Your task is to calculate the number of distinct ordered ways you can produce a money sum x using the available coins. For example, if the coins are $\{2, 3, 5\}$ and the desired sum is 9, there are 3 ways: - $2 + 2 + 5$ - $3 + 3 + 3$ - $2 + 2 + 2 + 3$

Input

The first input line has two integers n and x: the number of coins and the desired sum of money.

The second line has n distinct integers $c_1, c_2, \ldots, c_n$: the value of each coin.

Output

Print one integer: the number of ways modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 100$
- $1 \leq x \leq 10^6$
- $1 \leq c_i \leq 10^6$

Example

Input:

```
3 9
2 3 5
```

Output:

```
3
```

Solution

```
const int maxX = 1e6;
const ll MOD = 1e9+7;
int N, X, c;
ll dp[maxX+1];
int main(){
    scanf("%d %d", &N, &X);
    dp[0] = 1;
    for(int i = 0; i < N; i++){
        scanf("%d", &c);
        for(int j = 0; j <= X-c; j++)
            dp[j+c] = (dp[j+c] + dp[j]) % MOD;
    }
    printf("%lld\n", dp[X]);
}
```

Counting Numbers [Digit DP] (Counts numbers up to N with a certain property by building them digit by digit while tracking constraints.)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to count the number of integers between a and b where no two adjacent digits are the same.

Input

The only input line has two integers a and b.

Output

Print one integer: the answer to the problem.

Constraints

- $0 \le a \le b \le 10^{18}$

Example

Input:

```
123 321
```

Output:

```
171
```

Solution

```
const int maxN = 20;
bool tight[maxN];
ll dp[10][maxN];
ll solve(ll x){
    if(x <= 10) return x;
    vector<int> D;
    memset(dp, 0, sizeof(dp));
    memset(tight, 0, sizeof(tight));
    for(int i = 0; x; i++){
        D.push_back(x%10);
        x /= 10;
    }
    reverse(D.begin(), D.end());
    int N = (int) D.size();
    tight[0] = true;
    for(int i = 1; i < N; i++)
        tight[i] = tight[i-1] && (D[i] != D[i-1]);
    for(int d = 1; d < D[0]; d++)
        dp[d][0] = 1;
    for(int i = 1; i < N; i++){
        // Prefix [0..i-1] is same as X
        if(tight[i-1])
            for(int d = 0; d < D[i]; d++)
                if(d != D[i-1])
                    dp[d][i]++;
        // Prefix [0..i-1] is all leading zeros
        for(int d = 1; d <= 9; d++)
            dp[d][i]++;
        //  All other cases
```

```
        for(int d1 = 0; d1 <= 9; d1++)
            for(int d2 = 0; d2 <= 9; d2++)
                if(d1 != d2)
                    dp[d2][i] += dp[d1][i-1];
    }
    ll cnt = tight[N-1];
    for(int d = 0; d <= 9; d++)
        cnt += dp[d][N-1];
    return cnt;
}
int main(){
    ll a, b;
    scanf("%lld %lld", &a, &b);
    printf("%lld\n", solve(b) - solve(a-1));
}
```

Counting Tilings [Profile DP] (Counts ways to tile a grid column by column, where the state (mask) represents the boundary's shape with the next column.)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to count the number of ways you can fill an $n \times m$ grid using 1×2 and 2×1 tiles.

Input

The only input line has two integers n and m.

Output

Print one integer: the number of ways modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 10$
- $1 \leq m \leq 1000$

Example

Input:

```
4 7
```

Output:

```
781
```

Solution

```
const int maxN = 11;
const int maxM = 1001;
const ll MOD = 1e9+7;
int N, M;
ll dp[maxN][maxM][1<<maxN];
int main(){
    scanf("%d %d", &N, &M);
    dp[N][0][0] = 1;
    for(int i = 1; i <= M; i++){
        for(int j = 0; j < (1<<N); j++)
            dp[0][i][j<<1] = dp[N][i-1][j];
        for(int j = 1; j <= N; j++){
```

```
            int x = 1<<(j-1);
            int y = 1<<j;
            for(int mask = 0; mask < (1<<(N+1)); mask++){
                dp[j-1][i][mask] %= MOD;
                if((mask&x) && (mask&y))    continue;
                if(mask&x)                  dp[j][i][mask^x] += dp[j-1][i][mask];
                else if(mask&y)             dp[j][i][mask^y] += dp[j-1][i][mask];
                else {
                    dp[j][i][mask^x] += dp[j-1][i][mask];
                    dp[j][i][mask^y] += dp[j-1][i][mask];
                }
            }
        }
    }
    printf("%lld\n", dp[N][M][0] % MOD);
}
```

Counting Towers [DP with Complex States] (Counts tower configurations of height i based on whether the previous layer i-1 ended in a single block or two separate blocks.)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to build a tower whose width is 2 and height is n. You have an unlimited supply of blocks whose width and height are integers. For example, here are some possible solutions for $n = 6$:

Given n, how many different towers can you build? Mirrored and rotated towers are counted separately if they look different.

Input

The first input line contains an integer t: the number of tests.

After this, there are t lines, and each line contains an integer n: the height of the tower.

Output

For each test, print the number of towers modulo $10^9 + 7$.

Constraints

- $1 \le t \le 100$
- $1 \le n \le 10^6$

Example

Input:

```
3
2
6
1337
```

Output:

```
8
2864
640403945
```

Solution

```
const int maxN = 1e6;
const ll MOD = 1e9+7;
int T, N;
ll dp[maxN+1];
int main(){
    dp[1] = 2;
    dp[2] = 8;
    for(int i = 3; i <= maxN; i++)
        dp[i] = ((6*dp[i-1] - 7*dp[i-2]) % MOD + MOD) % MOD;
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        scanf("%d", &N);
        printf("%lld\n", dp[N]);
    }
}
```

Dice Combinations [1D DP] (Counts ways to form a sum i by summing results from smaller sums like i-1, i-2, etc.)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to count the number of ways to construct sum n by throwing a dice one or more times. Each throw produces an outcome between 1 and 6. For example, if $n = 3$, there are 4 ways: - $1 + 1 + 1$ - $1 + 2$ - $2 + 1$ - 3

Input

The only input line has an integer n.

Output

Print the number of ways modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 10^6$

Example

Input:

```
3
```

Output:

```
4
```

Solution

```
const int maxN = 1e6;
const int MOD = 1e9+7;
int N, dp[maxN+1];
int main(){
    scanf("%d", &N);
    dp[0] = 1;
    for(int i = 1; i <= N; i++)
        for(int j = 1; j <= 6 && i-j >= 0; j++)
            dp[i] = (dp[i] + dp[i-j]) % MOD;
    printf("%d\n", dp[N]);
}
```

Edit Distance [DP on Strings] (Finds the minimum operations to convert one string to another by considering insertion, deletion, or substitution at each character.)

Time limit: 1.00 s **Memory limit:** 512 MB

The edit distance between two strings is the minimum number of operations required to transform one string into the other. The allowed operations are: - Add one character to the string. - Remove one character from the string. - Replace one character in the string. For example, the edit distance between LOVE and MOVIE is 2, because you can first replace L with M, and then add I. Your task is to calculate the edit distance between two strings.

Input

The first input line has a string that contains n characters between A–Z.

The second input line has a string that contains m characters between A–Z.

Output

Print one integer: the edit distance between the strings.

Constraints

- $1 \leq n, m \leq 5000$

Example

Input:

```
LOVE
MOVIE
```

Output:

```
2
```

Solution

```
const int maxN = 5e3+5;
int N, M, dp[maxN][maxN];
char a[maxN], b[maxN];
int main(){
    scanf("%s %s", a, b);
    N = (int) strlen(a);
    M = (int) strlen(b);
    memset(dp, 0x3f, sizeof(dp));
    for(int i = 0; i <= N; i++){
        for(int j = 0; j <= M; j++){
            if(i == 0)      dp[i][j] = j;
            else if(j == 0) dp[i][j] = i;
            else if(a[i-1] == b[j-1])   dp[i][j] = dp[i-1][j-1];
            else dp[i][j] = 1 + min({dp[i-1][j], dp[i][j-1], dp[i-1][j-1]});
        }
    }
    printf("%d\n", dp[N][M]);
}
```

Elevator Rides [DP with Bitmasking] (Finds the minimum rides for a subset of people (mask) by trying to add each person to a previously calculated optimal state.)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n people who want to get to the top of a building which has only one elevator. You know the weight of each person and the maximum allowed weight in the elevator. What is the minimum number of elevator rides?

Input

The first input line has two integers n and x: the number of people and the maximum allowed weight in the elevator.

The second line has n integers $w_1, w_2, \ldots, w_n$: the weight of each person.

Output

Print one integer: the minimum number of rides.

Constraints

- $1 \leq n \leq 20$
- $1 \leq x \leq 10^9$
- $1 \leq w_i \leq x$

Example

Input:

```
4 10
4 8 6 1
```

Output:

```
2
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 20;
const int INF = maxN+1;
int N;
pii dp[1<<maxN];
ll X, w[maxN];
int main(){
    scanf("%d %lld", &N, &X);
    for(int i = 0; i < N; i++)
        scanf("%lld", &w[i]);
    dp[0] = {1, 0};
    for(int mask = 1; mask < (1<<N); mask++){
        dp[mask] = {INF, 0};
        for(int i = 0; i < N; i++){
            if(mask&(1<<i)){
                pii can = dp[mask^(1<<i)];
                if(can.second + w[i] <= X){
                    can.second += w[i];
                } else {
                    can.first++;
                    can.second = w[i];
                }
                dp[mask] = min(dp[mask], can);
            }
        }
    }
    printf("%d\n", dp[(1<<N)-1].first);
}
```

Grid Paths I [Grid DP] (Counts paths to cell (i,j) by adding the number of paths from the cell above and the cell to the left.)

Time limit: 1.00 s **Memory limit:** 512 MB

Consider an $n \times n$ grid whose squares may have traps. It is not allowed to move to a square with a trap. Your task is to calculate the number of paths from the upper-left square to the lower-right square. You can only move right or down.

Input

The first input line has an integer n: the size of the grid.

After this, there are n lines that describe the grid. Each line has n characters: . denotes an empty cell, and * denotes a trap.

Output

Print the number of paths modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 1000$

Example

Input:

```
4
....
.*..
...*
*...
```

Output:

```
3
```

Solution

```
const int maxN = 1000;
const ll MOD = 1e9+7;
int N;
char c[maxN+1][maxN+1];
ll dp[maxN+1][maxN+1];
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++)
        for(int j = 1; j <= N; j++)
            scanf(" %c", &c[i][j]);
    dp[1][1] = (c[1][1] == '.');
    for(int i = 1; i <= N; i++){
        for(int j = 1; j <= N; j++){
            if(c[i][j] == '.'){
                if(c[i-1][j] == '.')    dp[i][j] += dp[i-1][j];
                if(c[i][j-1] == '.')    dp[i][j] += dp[i][j-1];
                dp[i][j] %= MOD;
            }
        }
    }
```

```
    printf("%lld\n", dp[N][N]);
}
```

Increasing Subsequence [LIS DP] (Finds the length of the longest increasing subsequence ending at an index i by checking all previous elements.)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array containing n integers. Your task is to determine the longest increasing subsequence in the array, i.e., the longest subsequence where every element is larger than the previous one. A subsequence is a sequence that can be derived from the array by deleting some elements without changing the order of the remaining elements.

Input

The first line contains an integer n: the size of the array.

After this there are n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print the length of the longest increasing subsequence.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq x_i \leq 10^9$

Example

Input:

```
8
7 3 5 3 6 2 9 8
```

Output:

```
4
```

Solution

```
int N, a;
multiset<int> S;
multiset<int>::iterator it;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d", &a);
        S.insert(a);
        it = S.lower_bound(a);
        it++;
        if(it != S.end())
            S.erase(it);
    }
    printf("%d\n", (int) S.size());
}
```

Increasing Subsequence II

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to calculate the number of increasing subsequences it contains. If two subsequences have the same values but in different positions in the array, they are counted separately.

Input

The first input line has an integer n: the size of the array.

The second line has n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print one integer: the number of increasing subsequences modulo $10^9 + 7$.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

```
3
2 1 3
```

Output:

```
5
```

Explanation: The increasing subsequences are [2], [1], [3], [2, 3] and [1, 3].

Solution

```
const int maxN = 2e5+5;
const ll MOD = 1e9+7;
int N;
ll ans, ds[maxN];
struct Operation { int x, idx; } ops[maxN];
void update(int idx, ll val){
    for(int i = idx; i < maxN; i += -i&i)
        ds[i] = (ds[i] + val) % MOD;
}
ll query(int idx){
    ll sum = 0;
    for(int i = idx; i > 0; i -= -i&i)
        sum = (sum + ds[i]) % MOD;
    return sum;
}
int main(){
    scanf("%d", &N);
    for(int i = 0, x; i < N; i++){
        scanf("%d", &x);
        ops[i] = {x, i+1};
    }
    sort(ops, ops+N, [](Operation A, Operation B){
        return A.x == B.x ? B.idx < A.idx : A.x < B.x;
    });
    for(int i = 0; i < N; i++){
        int idx = ops[i].idx;
        ll amnt = query(idx)+1;
        ans = (ans + amnt) % MOD;
        update(idx, amnt);
    }
```

```
    printf("%lld\n", ans);
}
```

Longest Common Subsequence

Time limit: 1.00 s **Memory limit:** 512 MB

Given two arrays of integers, find their longest common subsequence. A subsequence is a sequence of array elements from left to right that can contain gaps. A common subsequence is a subsequence that appears in both arrays.

Input

The first line has two integers n and m: the sizes of the arrays.

The second line has n integers $a_1, a_2, \ldots, a_n$: the contents of the first array.

The third line has m integers $b_1, b_2, \ldots, b_m$: the contents of the second array.

Output

First print the length of the longest common subsequence.

After that, print an example of such a sequence. If there are several solutions, you can print any of them.

Constraints

- $1 \le n, m \le 1000$
- $1 \le a_i, b_i \le 10^9$

Example

Input:

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

Output:

```
3
1 2 4
```

Solution

```
const int maxN = 1e3+1;
int N, M, a[maxN], b[maxN];
int len[maxN][maxN];
char ptr[maxN][maxN];
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 1; i <= N; i++) scanf("%d", &a[i]);
    for(int i = 1; i <= M; i++) scanf("%d", &b[i]);
    for(int i = 1; i <= N; i++){
        for(int j = 1; j <= M; j++){
            if(len[i][j] <= len[i-1][j]){
                len[i][j] = len[i-1][j];
                ptr[i][j] = 'L';
            }
            if(len[i][j] <= len[i][j-1]){
                len[i][j] = len[i][j-1];
                ptr[i][j] = 'U';
            }
```

```
            if(a[i] == b[j] && len[i][j] <= len[i-1][j-1] + 1){
                len[i][j] = len[i-1][j-1] + 1;
                ptr[i][j] = 'D';
            }
        }
    }
    vector<int> ans;
    int i = N, j = M;
    while(i != 0 && j != 0){
        char dir = ptr[i][j];
        if(dir == 'L')      i--;
        else if(dir == 'U') j--;
        else if(dir == 'D'){
            ans.push_back(a[i]);
            i--; j--;
        }
    }
    reverse(ans.begin(), ans.end());
    int l = len[N][M];
    printf("%d\n", l);
    for(int i = 0; i < l; i++)
        printf("%d%c", ans[i], (" \n")[i==l-1]);
}
```

Minimizing Coins [Unbounded Knapsack] (Finds minimum coins for a sum i by trying each coin c and using the pre-calculated result for sum i-c.)

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a money system consisting of n coins. Each coin has a positive integer value. Your task is to produce a sum of money x using the available coins in such a way that the number of coins is minimal. For example, if the coins are $\{1, 5, 7\}$ and the desired sum is 11, an optimal solution is $5 + 5 + 1$ which requires 3 coins.

Input

The first input line has two integers n and x: the number of coins and the desired sum of money.

The second line has n distinct integers $c_1, c_2, \ldots, c_n$: the value of each coin.

Output

Print one integer: the minimum number of coins. If it is not possible to produce the desired sum, print -1.

Constraints

- $1 \le n \le 100$
- $1 \le x \le 10^6$
- $1 \le c_i \le 10^6$

Example

Input:

```
3 11
1 5 7
```

Output:

```
3
```

Solution

```
const int maxX = 1e6;
const int INF = 0x3f3f3f3f;
int N, X, c, dp[maxX+1];
int main(){
    scanf("%d %d", &N, &X);
    fill(dp+1, dp+X+1, INF);
    for(int i = 0; i < N; i++){
        scanf("%d", &c);
        for(int j = 0; j <= X-c; j++)
            if(dp[j] != INF)
                dp[j+c] = min(dp[j+c], dp[j]+1);
    }
    printf("%d\n", dp[X] == INF ? -1 : dp[X]);
}
```

Money Sums [Subset Sum] (Determines all possible sums that can be formed by iteratively adding each coin's value to previously achievable sums.)

Time limit: 1.00 s **Memory limit:** 512 MB

You have n coins with certain values. Your task is to find all money sums you can create using these coins.

Input

The first input line has an integer n: the number of coins.

The next line has n integers $x_1, x_2, \ldots, x_n$: the values of the coins.

Output

First print an integer k: the number of distinct money sums. After this, print all possible sums in increasing order.

Constraints

- $1 \leq n \leq 100$
- $1 \leq x_i \leq 1000$

Example

Input:

```
4
4 2 5 2
```

Output:

```
9
2 4 5 6 7 8 9 11 13
```

Solution

```
const int maxX = 1e5;
int N, x, cnt, largest;
bool dp[maxX+1];
int main(){
    scanf("%d", &N);
    dp[0] = true;
    for(int i = 0; i < N; i++){
        scanf("%d", &x);
```

```
        for(int j = maxX-x; j >= 0; j--)
            if(dp[j])
                dp[j+x] = true;
    }
    for(int i = 1; i <= maxX; i++){
        if(dp[i]){
            largest = i;
            cnt++;
        }
    }
    printf("%d\n", cnt++);
    for(int i = 1; i <= maxX; i++)
        if(dp[i])
            printf("%d%c", i, (" \n")[i==largest]);
}
```

Mountain Range

Time limit: 1.00 s **Memory limit:** 512 MB

There are n mountains in a row, each with a specific height. You begin your hang gliding route from some mountain. You can glide from mountain a to mountain b if mountain a is taller than mountain b and all mountains between a and b. What is the maximum number of mountains you can visit on your route?

Input

The first line has an integer n: the number of mountains.

The next line has n integers $h_1, h_2, \ldots, h_n$: the heights of the mountains.

Output:

Print one integer: the maximum number of mountains.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq h_i \leq 10^9$

Example

Input:

```
10
20 15 17 35 25 40 12 19 13 12
```

Output:

```
5
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 2e5+1;
const int INF = 0x3f3f3f3f;
int N, dp[maxN];
vector<int> tallers;
vector<pii> mountains;
set<int,greater<int>> S_l;
set<int> S_r;
int main(){
```

```
    scanf("%d", &N);
    for(int i = 1, h; i <= N; i++){
        scanf("%d", &h);
        mountains.push_back({h, i});
    }
    sort(mountains.begin(), mountains.end(), greater<pii>());
    int last_height = INF;
    for(pii m : mountains){
        int h = m.first, i = m.second;
        if(last_height != h){
            for(int t : tallers){
                S_l.insert(t);
                S_r.insert(t);
            }
            tallers.clear();
        }
        auto l_ptr = S_l.lower_bound(i);
        auto r_ptr = S_r.lower_bound(i);
        int l = (l_ptr == S_l.end() ? 0 : *l_ptr);
        int r = (r_ptr == S_r.end() ? 0 : *r_ptr);
        dp[i] = max(dp[l], dp[r]) + 1;
        tallers.push_back(i);
        last_height = h;
    }
    int best = 0;
    for(int i = 1; i <= N; i++)
        best = max(best, dp[i]);
    printf("%d\n", best);
}
```

Projects [DP with Sorting] (Maximizes profit by sorting projects by end time and deciding for each whether to take it or skip it based on the last non-overlapping project.)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n projects you can attend. For each project, you know its starting and ending days and the amount of money you would get as reward. You can only attend one project during a day. What is the maximum amount of money you can earn?

Input

The first input line contains an integer n: the number of projects.

After this, there are n lines. Each such line has three integers a_i, b_i, and p_i: the starting day, the ending day, and the reward.

Output

Print one integer: the maximum amount of money you can earn.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le a_i \le b_i \le 10^9$
- $1 \le p_i \le 10^9$

Example

Input:

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

Output:

```
7
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 2e5+1;
int N, a[maxN], b[maxN];
ll p[maxN], dp[2*maxN];
struct project {int time, id, type;} times[2*maxN];
map<int,int> mp;
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++){
        scanf("%d %d %lld", &a[i], &b[i], &p[i]);
        times[2*i] = {a[i], i, 0};
        times[2*i+1] = {b[i], i, 1};
    }
    sort(times+2, times+2*N+2, [](project A, project B){
        return A.time == B.time ? A.id < B.id : A.time < B.time;
    });
    for(int i = 2; i <= 2*N+1; i++)
        if(!mp[times[i].time])
            mp[times[i].time] = i;
    for(int i = 2; i <= 2*N+1; i++){
        if(times[i].type == 0)  dp[i] = dp[i-1];
        else dp[i] = max(dp[i-1], dp[mp[a[times[i].id]]-1] + p[times[i].id]);
    }
    printf("%lld\n", dp[2*N+1]);
}
```

Rectangle Cutting [Grid DP] (Finds minimum cuts for a h x w rectangle by testing all possible horizontal/vertical cuts and using results for smaller rectangles.)

Time limit: 1.00 s **Memory limit:** 512 MB

Given an $a \times b$ rectangle, your task is to cut it into squares. On each move you can select a rectangle and cut it into two rectangles in such a way that all side lengths remain integers. What is the minimum possible number of moves?

Input

The only input line has two integers a and b.

Output

Print one integer: the minimum number of moves.

Constraints

- $1 \leq a, b \leq 500$

Example

Input:

```
3 5
```

Output:

```
3
```

Solution

```
const int INF = 0x3f3f3f3f;
int A, B, dp[501][501];
int solve(int a, int b){
    if(a > b)   swap(a, b);
    if(dp[a][b] != INF)     return dp[a][b];
    if(a == b)              return dp[a][b] = 0;
    if(a == 1 || b == 1)    return dp[a][b] = (a == 1 ? b-1 : a-1);
    for(int i = 1; i < a; i++)
        dp[a][b] = min(dp[a][b], solve(i, b) + solve(a-i, b) + 1);
    for(int i = 1; i < b; i++)
        dp[a][b] = min(dp[a][b], solve(a, i) + solve(a, b-i) + 1);
    return dp[a][b];
}
int main(){
    memset(dp, 0x3f, sizeof(dp));
    scanf("%d %d", &A, &B);
    printf("%d\n", solve(A, B));
}
```

Removal Game [Interval DP] (Finds the max score difference in a range [l,r] by assuming optimal play when choosing either the leftmost or rightmost element.)

Time limit: 1.00 s **Memory limit:** 512 MB

There is a list of n numbers and two players who move alternately. On each move, a player removes either the first or last number from the list, and their score increases by that number. Both players try to maximize their scores. What is the maximum possible score for the first player when both players play optimally?

Input

The first input line contains an integer n: the size of the list.

The next line has n integers $x_1, x_2, \ldots, x_n$: the contents of the list.

Output

Print the maximum possible score for the first player.

Constraints

- $1 \leq n \leq 5000$
- $-10^9 \leq x_i \leq 10^9$

Example

Input:

```
4
4 5 1 3
```

Output:

```
8
```

Solution

```
const int maxN = 5000;
int N, x[maxN+1];
ll p[maxN+1], dp[maxN+1][maxN+1];
bool found[maxN+1][maxN+1];
ll sum(int l, int r){
    return p[r] - p[l-1];
}
ll solve(int l, int r){
    if(found[l][r]) return dp[l][r];
    if(l == r)      return x[l];
    found[l][r]=true;
    return dp[l][r] = max(x[l]+sum(l+1, r)-solve(l+1, r), x[r]+sum(l,r-1)-solve(l,r-1));
}
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++){
        scanf("%d", &x[i]);
        p[i] = p[i-1] + x[i];
    }
    printf("%lld\n", solve(1, N));
}
```

Removing Digits [Greedy DP] (Finds minimum steps from a number n to 0 by greedily subtracting one of its digits at each step.)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an integer n. On each step, you may subtract one of the digits from the number. How many steps are required to make the number equal to 0?

Input

The only input line has an integer n.

Output

Print one integer: the minimum number of steps.

Constraints

- $1 \le n \le 10^6$

Example

Input:

```
27
```

Output:

5

Explanation: An optimal solution is $27 \to 20 \to 18 \to 10 \to 9 \to 0$.

Solution

```
const int maxN = 1e6;
const int INF = 0x3f3f3f3f;
int N, dp[maxN+1];
int main(){
    scanf("%d", &N);
    fill(dp+1, dp+N+1, INF);
    for(int i = 1; i <= N; i++){
        int d = i;
        while(d > 0){
            if(d%10 != 0)
                dp[i] = min(dp[i], dp[i-(d%10)]+1);
            d /= 10;
        }
    }
    printf("%d\n", dp[N]);
}
```

Two Sets II [Subset Sum / Partition] (Counts ways to partition numbers 1..N into two equal sum sets by finding the number of ways to form half the total sum.)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to count the number of ways numbers $1, 2, \ldots, n$ can be divided into two sets of equal sum. For example, if $n = 7$, there are four solutions: - $\{1, 3, 4, 6\}$ and $\{2, 5, 7\}$ - $\{1, 2, 5, 6\}$ and $\{3, 4, 7\}$ - $\{1, 2, 4, 7\}$ and $\{3, 5, 6\}$ - $\{1, 6, 7\}$ and $\{2, 3, 4, 5\}$

Input

The only input line contains an integer n.

Output

Print the answer modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 500$

Example

Input:

7

Output:

4

Solution

```
/**
 * A058377
 * Retrivied from https://oeis.org/A058377
 */
int N, mp[501];
```

```
void init(){
    mp[3] = 1;
    mp[4] = 1;
    mp[7] = 4;
    mp[8] = 7;
    mp[11] = 35;
    mp[12] = 62;
    mp[15] = 361;
    mp[16] = 657;
    mp[19] = 4110;
    mp[20] = 7636;
    mp[23] = 49910;
    mp[24] = 93846;
    mp[27] = 632602;
    mp[28] = 1199892;
    mp[31] = 8273610;
    mp[32] = 15796439;
    mp[35] = 110826888;
    mp[36] = 212681976;
    mp[39] = 512776583;
    mp[40] = 915017346;
    mp[43] = 965991877;
    mp[44] = 536015750;
    mp[47] = 245739109;
    mp[48] = 497111739;
    mp[51] = 319303648;
    mp[52] = 661531964;
    mp[55] = 919134122;
    mp[56] = 526597333;
    mp[59] = 857860749;
    mp[60] = 323065127;
    mp[63] = 142897738;
    mp[64] = 454622296;
    mp[67] = 832964922;
    mp[68] = 112945499;
    mp[71] = 21130483;
    mp[72] = 134912852;
    mp[75] = 58392205;
    mp[76] = 594756797;
    mp[79] = 116285762;
    mp[80] = 46020942;
    mp[83] = 199311883;
    mp[84] = 637852081;
    mp[87] = 681073029;
    mp[88] = 687587655;
    mp[91] = 403293621;
    mp[92] = 859161650;
    mp[95] = 135570834;
    mp[96] = 16951659;
    mp[99] = 90174435;
    mp[100] = 906443459;
    mp[103] = 445036292;
    mp[104] = 696064542;
    mp[107] = 589938798;
    mp[108] = 17765238;
    mp[111] = 671699270;
```

```
mp[112] = 979144036;
mp[115] = 717567569;
mp[116] = 962408760;
mp[119] = 24874238;
mp[120] = 651719820;
mp[123] = 882521441;
mp[124] = 815306501;
mp[127] = 177673311;
mp[128] = 381523124;
mp[131] = 647107433;
mp[132] = 822743471;
mp[135] = 951125976;
mp[136] = 913762232;
mp[139] = 860772858;
mp[140] = 718799291;
mp[143] = 546548485;
mp[144] = 520899315;
mp[147] = 348826222;
mp[148] = 979465686;
mp[151] = 551269897;
mp[152] = 226367872;
mp[155] = 224237396;
mp[156] = 275455845;
mp[159] = 597702194;
mp[160] = 580526114;
mp[163] = 837164670;
mp[164] = 579854574;
mp[167] = 643888367;
mp[168] = 547744591;
mp[171] = 362172782;
mp[172] = 203753851;
mp[175] = 472078730;
mp[176] = 414960148;
mp[179] = 751350256;
mp[180] = 597996235;
mp[183] = 302890488;
mp[184] = 463389357;
mp[187] = 28369705;
mp[188] = 625260957;
mp[191] = 439010166;
mp[192] = 625343710;
mp[195] = 345211145;
mp[196] = 783212645;
mp[199] = 171026155;
mp[200] = 746149676;
mp[203] = 574908810;
mp[204] = 400819234;
mp[207] = 797285006;
mp[208] = 671068618;
mp[211] = 857904807;
mp[212] = 118539037;
mp[215] = 258515519;
mp[216] = 103887197;
mp[219] = 692754470;
mp[220] = 307691579;
mp[223] = 543477917;
```

```
mp[224] = 494845326;
mp[227] = 461141061;
mp[228] = 681627336;
mp[231] = 194431224;
mp[232] = 798222254;
mp[235] = 34177070;
mp[236] = 324550451;
mp[239] = 395144714;
mp[240] = 314224734;
mp[243] = 256354567;
mp[244] = 216295565;
mp[247] = 556521816;
mp[248] = 594547313;
mp[251] = 701665484;
mp[252] = 287171616;
mp[255] = 305999810;
mp[256] = 817725356;
mp[259] = 456522567;
mp[260] = 42456953;
mp[263] = 53352478;
mp[264] = 896195082;
mp[267] = 374247344;
mp[268] = 900048655;
mp[271] = 613110673;
mp[272] = 91338349;
mp[275] = 904876664;
mp[276] = 300880501;
mp[279] = 604541603;
mp[280] = 441166519;
mp[283] = 73667549;
mp[284] = 41483999;
mp[287] = 51276243;
mp[288] = 149197976;
mp[291] = 539103967;
mp[292] = 982253554;
mp[295] = 969982399;
mp[296] = 98482383;
mp[299] = 605461327;
mp[300] = 65785519;
mp[303] = 548373331;
mp[304] = 421491751;
mp[307] = 480765781;
mp[308] = 925355425;
mp[311] = 850819946;
mp[312] = 434384766;
mp[315] = 747079619;
mp[316] = 140715817;
mp[319] = 154291092;
mp[320] = 436737393;
mp[323] = 694334366;
mp[324] = 278298264;
mp[327] = 100011200;
mp[328] = 186925353;
mp[331] = 951803656;
mp[332] = 826521841;
mp[335] = 467206470;
```

```
mp[336] = 625245512;
mp[339] = 37014692;
mp[340] = 370302058;
mp[343] = 942594593;
mp[344] = 625802329;
mp[347] = 696810018;
mp[348] = 839447903;
mp[351] = 973813010;
mp[352] = 340829958;
mp[355] = 333578000;
mp[356] = 162910708;
mp[359] = 898163184;
mp[360] = 938735258;
mp[363] = 969420912;
mp[364] = 767331949;
mp[367] = 49040853;
mp[368] = 864361228;
mp[371] = 666086921;
mp[372] = 681324453;
mp[375] = 406330883;
mp[376] = 715350645;
mp[379] = 276425302;
mp[380] = 691275326;
mp[383] = 936153559;
mp[384] = 806887794;
mp[387] = 672700998;
mp[388] = 324056520;
mp[391] = 624244157;
mp[392] = 887151949;
mp[395] = 111928807;
mp[396] = 555201478;
mp[399] = 892266330;
mp[400] = 406976742;
mp[403] = 179624853;
mp[404] = 766709833;
mp[407] = 615024703;
mp[408] = 25276943;
mp[411] = 675398735;
mp[412] = 373735428;
mp[415] = 283235362;
mp[416] = 613728485;
mp[419] = 581407804;
mp[420] = 135760574;
mp[423] = 421029356;
mp[424] = 925084280;
mp[427] = 409496848;
mp[428] = 345330916;
mp[431] = 754684998;
mp[432] = 388429454;
mp[435] = 997193850;
mp[436] = 722803385;
mp[439] = 747396848;
mp[440] = 879532546;
mp[443] = 363180870;
mp[444] = 301109892;
mp[447] = 337814331;
```

```
    mp[448] = 439172004;
    mp[451] = 142119927;
    mp[452] = 761228466;
    mp[455] = 649629227;
    mp[456] = 868002592;
    mp[459] = 927500726;
    mp[460] = 71856333;
    mp[463] = 621649641;
    mp[464] = 86464550;
    mp[467] = 159973467;
    mp[468] = 281964303;
    mp[471] = 220801847;
    mp[472] = 172221992;
    mp[475] = 631635476;
    mp[476] = 30971150;
    mp[479] = 664722592;
    mp[480] = 692804591;
    mp[483] = 620001363;
    mp[484] = 22371363;
    mp[487] = 541326371;
    mp[488] = 8514587;
    mp[491] = 996643776;
    mp[492] = 106479414;
    mp[495] = 920757401;
    mp[496] = 236457589;
    mp[499] = 608650075;
    mp[500] = 71857061;
}
int main(){
    init();
    scanf("%d", &N);
    printf("%d\n", mp[N]);
}
```

Geometry

All Manhattan Distances

Time limit: 1.00 s **Memory limit:** 512 MB

Given a set of points, calculate the sum of all Manhattan distances between two point pairs.

Input

The first line has an integer n: the number of points.

The following n lines describe the points. Each line has two integers x and y. You can assume that each point is distinct.

Output

Print the sum of all Manhattan distances.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $-10^9 \le x, y \le 10^9$

Example

Input:

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

Output:

```
36
```

Solution

```
const int maxN = 2e5+1;
int N, x[maxN], y[maxN];
__int128 xcur, ycur, ans;
ostream& operator<<(ostream& o, const __int128& x) {
    if(x == numeric_limits<__int128>::min())
        return o << "-170141183460469231731687303715884105728";
    else if(x < 0)  return o << "-" << -x;
    else if(x < 10) return o << (char)(x + '0');
    else            return o << x / 10 << (char)(x % 10 + '0');
}
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)
        scanf("%d %d", &x[i], &y[i]);
    sort(x, x+N);
    sort(y, y+N);
    xcur = ycur = ans = 0;
    for(int i = 1; i < N; i++){
        xcur += (__int128) (x[i] - x[i-1]) * i;
        ycur += (__int128) (y[i] - y[i-1]) * i;
        ans += xcur + ycur;
    }
    cout << ans << endl;
}
```

Area of Rectangles (union area of n axis-aligned rectangles using a sweep line algorithm with a segment tree to track vertical coverage)

Time limit: 1.00 s **Memory limit:** 512 MB

Given n rectangles, your task is to determine the total area of their union.

Input

The first line has an integer n: the number of rectangles.

After that, there are n lines describing the rectangles. Each line has four integers x_1, y_1, x_2 and y_2: a rectangle begins at point (x_1, y_1) and ends at point (x_2, y_2).

Output

Print the total area covered by the rectangles.

Constraints

- $1 \leq n \leq 10^5$
- $-10^6 \leq x_1 < x_2 \leq 10^6$
- $-10^6 \leq y_1 < y_2 \leq 10^6$

Example

Input:

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

Output:

```
24
```

Solution

```
typedef array<int,4> Operation;
const int maxN = 1e5;
const int SZ = 9e6;
int N, lo[SZ], hi[SZ];
ll area, delta[SZ], score[SZ];
Operation op[2*maxN];
int len(int i){
    return hi[i]-lo[i]+1;
}
void pull(int i){
    if(lo[i] == hi[i])  score[i] = (delta[i] > 0 ? 1 : 0);
    else                score[i] = (delta[i] > 0 ? len(i) : score[2*i] + score[2*i+1]);
}
void build(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r)  return;
    int m = l+(r-l)/2;
    build(2*i, l, m);
    build(2*i+1, m+1, r);
}
void increment(int i, int l, int r, ll val){
    if(l > hi[i] || r < lo[i])  return;
    if(l <= lo[i] && hi[i] <= r){
        delta[i] += val;
        pull(i);
        return;
    }
    increment(2*i, l, r, val);
    increment(2*i+1, l, r, val);
    pull(i);
}
ll query(){
    return score[1];
}
int main(){
    scanf("%d", &N);
    for(int i = 0, a, b, c, d; i < N; i++){
        scanf("%d %d %d %d", &a, &b, &c, &d);
```

```
        op[2*i] = {1, b, a+1, c};
        op[2*i+1] = {-1, d, a+1, c};
    }
    sort(op, op+2*N, [](Operation A, Operation B){
       return (A[1] == B[1] ? A[0] < B[0] : A[1] < B[1]);
    });
    build(1, -1e6-5, 1e6+5);
    int lst = -1e6;
    for(int i = 0; i < 2*N; i++){
        int t = op[i][0], y = op[i][1], x1 = op[i][2], x2 = op[i][3];
        area += (y-lst) * query();
        increment(1, x1, x2, t);
        lst = y;
    }
    printf("%lld\n", area);
}
```

Convex Hull

Time limit: 1.00 s **Memory limit:** 512 MB

Given a set of n points in the two-dimensional plane, your task is to determine the convex hull of the points.

Input

The first input line has an integer n: the number of points.

After this, there are n lines that describe the points. Each line has two integers x and y: the coordinates of a point.

You may assume that each point is distinct, and the area of the hull is positive.

Output

First print an integer k: the number of points in the convex hull.

After this, print k lines that describe the points. You can print the points in any order. Print all points that lie on the convex hull.

Constraints

- $3 \le n \le 2 \cdot 10^5$
- $-10^9 \le x, y \le 10^9$

Example

Input:

```
6
2 1
2 5
3 3
4 3
4 4
6 3
```

Output:

```
4
2 1
2 5
4 4
6 3
```

Solution

```
const int maxN = 2e5+5;
struct Point {
    int x, y;
    void read(){ scanf("%d %d", &x, &y); }
    Point operator +(const Point& b) const { return Point{x+b.x, y+b.y}; }
    Point operator -(const Point& b) const { return Point{x-b.x, y-b.y}; }
    ll operator *(const Point& b) const { return (ll) x * b.y - (ll) y * b.x; }
    bool operator <(const Point& b) const { return x == b.x ? y < b.y : x < b.x; }
    void operator +=(const Point& b) { x += b.x; y += b.y; }
    void operator -=(const Point &b) { x -= b.x; y -= b.y; }
    void operator *=(const int k) { x *= k; y *= k; }
    ll cross(const Point& b, const Point& c) const {
        return (b - *this) * (c - *this);
    }
};
int N, S;
Point P[maxN];
vector<Point> hull;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)
        P[i].read();
    sort(P, P+N);
    for(int t = 0; t < 2; t++){
        for(int i = 0; i < N; i++){
            while((int) hull.size()-S >= 2){
                Point P1 = hull[hull.size()-2];
                Point P2 = hull[hull.size()-1];
                if(P1.cross(P2, P[i]) <= 0) break;
                hull.pop_back();
            }
            hull.push_back(P[i]);
        }
        hull.pop_back();
        S = hull.size();
        reverse(P, P+N);
    }
    printf("%d\n", S);
    for(Point h : hull)
        printf("%d %d\n", h.x, h.y);
}
```

Intersection Points

Time limit: 1.00 s **Memory limit:** 512 MB

Given n horizontal and vertical line segments, your task is to calculate the number of their intersection points. You can assume that no parallel line segments intersect, and no endpoint of a line segment is an intersection point.

Input

The first line has an integer n: the number of line segments.

Then there are n lines describing the line segments. Each line has four integers: x_1, y_1, x_2 and y_2: a line segment begins at point (x_1, y_1) and ends at point (x_2, y_2).

Output

Print the number of intersection points.

Constraints

- $1 \leq n \leq 10^5$
- $-10^6 \leq x_1 \leq x_2 \leq 10^6$
- $-10^6 \leq y_1 \leq y_2 \leq 10^6$
- $(x_1, y_1) \neq (x_2, y_2)$

Example

Input:

```
3
2 3 7 3
3 1 3 5
6 2 6 6
```

Output:

```
2
```

Solution

```
const int maxN = 1e5+1;
const int maxX = 1e6+5;
const int SIZE = 2*maxX;
struct Operation {
    int type, y, a, b;
};
int N, ds[SIZE];
vector<Operation> ops;
void update(int idx, int val){
    for(int i = idx; i < SIZE; i += -i&i)
        ds[i] += val;
}
int query(int idx){
    int sum = 0;
    for(int i = idx; i > 0; i -= -i&i)
        sum += ds[i];
    return sum;
}
int main(){
    scanf("%d", &N);
    for(int i = 0, a, b, c, d; i < N; i++){
        scanf("%d %d %d %d", &a, &b, &c, &d);
        if(a == c){
            // Vertical
            ops.push_back({2, b, a+maxX, -1});
            ops.push_back({3, d, a+maxX, -1});
        } else {
            // Horizontal
            ops.push_back({1, b, a+maxX, c+maxX});
        }
    }
    sort(ops.begin(), ops.end(), [](Operation A, Operation B){
        return A.y < B.y;
```

```
    });
    ll ans = 0;
    for(Operation O : ops){
        if(O.type == 1)         ans += query(O.b) - query(O.a-1);
        else if(O.type == 2)    update(O.a, 1);
        else if(O.type == 3)    update(O.a, -1);
    }
    printf("%lld\n", ans);
}
```

Line Segment Intersection

Time limit: 1.00 s **Memory limit:** 512 MB

There are two line segments: the first goes through the points (x_1, y_1) and (x_2, y_2), and the second goes through the points (x_3, y_3) and (x_4, y_4). Your task is to determine if the line segments intersect, i.e., they have at least one common point.

Input

The first input line has an integer t: the number of tests.

After this, there are t lines that describe the tests. Each line has eight integers x_1, y_1, x_2, y_2, x_3, y_3, x_4 and y_4.

Output

For each test, print "YES" if the line segments intersect and "NO" otherwise.

Constraints

- $1 \le t \le 10^5$
- $-10^9 \le x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4 \le 10^9$
- $(x_1, y_1) \neq (x_2, y_2)$
- $(x_3, y_3) \neq (x_4, y_4)$

Example

Input:

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

Output:

```
NO
YES
YES
YES
YES
```

Solution

```
struct Point {
    int x, y;
    void read(){ scanf("%d %d", &x, &y); }
    Point operator +(const Point& b) const { return Point{x+b.x, y+b.y}; }
    Point operator -(const Point& b) const { return Point{x-b.x, y-b.y}; }
```

```
    ll operator *(const Point& b) const { return (ll) x * b.y - (ll) y * b.x; }
    void operator +=(const Point& b) { x += b.x; y += b.y; }
    void operator -=(const Point &b) { x -= b.x; y -= b.y; }
    void operator *=(const int k) { x *= k; y *= k; }
    ll cross(const Point& b, const Point& c) const {
        return (b - *this) * (c - *this);
    }
};
int T;
Point P[4];
void lineintersect(Point P1, Point P2, Point P3, Point P4){
    // Parallel case
    if((P2-P1) * (P4-P3) == 0){
        // Collinear case, check bounding boxes
        if(P1.cross(P2, P3) == 0){
            for(int i = 0; i < 2; i++){
                if(max(P1.x, P2.x) < min(P3.x, P4.x) || max(P1.y, P2.y) < min(P3.y, P4.y)){
                    printf("NO\n");
                    return;
                }
                swap(P1, P3);
                swap(P2, P4);
            }
            printf("YES\n");
            return;
        }
        // Non-collinear parallel lines never intersect
        printf("NO\n");
        return;
    }
    // Non-parallel case
    for(int i = 0; i < 2; i++){
        ll s1 = P1.cross(P2, P3);
        ll s2 = P1.cross(P2, P4);
        if((s1 < 0 && s2 < 0) || (s1 > 0 && s2 > 0)){
            printf("NO\n");
            return;
        }
        swap(P1, P3);
        swap(P2, P4);
    }
    printf("YES\n");
}
int main(){
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        for(int i = 0; i < 4; i++)
            P[i].read();
        lineintersect(P[0], P[1], P[2], P[3]);
    }
}
```

Maximum Manhattan Distances

Time limit: 1.00 s **Memory limit:** 512 MB

A set is initially empty and n points are added to it. Calculate the maximum Manhattan distance of two points

after each addition.

Input

The first line has an integer n: the number of points.

The following n lines describe the points. Each line has two integers x and y. You can assume that each point is distinct.

Output

After each addition, print the maximum distance.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $-10^9 \le x, y \le 10^9$

Example

Input:

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

Output:

```
0
3
4
4
7
```

Solution

```
const ll INF = 0x3f3f3f3f3f3f3f3f;
int N;
ll l1, l2, l3, l4;
int main(){
    scanf("%d", &N);
    l1 = l4 = -INF;
    l2 = l3 = INF;
    for(int i = 0; i < N; i++){
        ll x, y;
        scanf("%lld %lld", &x, &y);
        l1 = max(l1, x+y);
        l2 = min(l2, x+y);
        l3 = min(l3, x-y);
        l4 = max(l4, x-y);
        printf("%lld\n", max(l1-l2, l4-l3));
    }
}
```

Minimum Euclidean Distance

Time limit: 1.00 s **Memory limit:** 512 MB

Given a set of points in the two-dimensional plane, your task is to find the minimum Euclidean distance between two distinct points. The Euclidean distance of points (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Input

The first input line has an integer n: the number of points.

After this, there are n lines that describe the points. Each line has two integers x and y. You may assume that each point is distinct.

Output

Print one integer: d^2 where d is the minimum Euclidean distance (this ensures that the result is an integer).

Constraints

- $2 \le n \le 2 \cdot 10^5$
- $-10^9 \le x, y \le 10^9$

Example

Input:

```
4
2 1
4 4
1 2
6 3
```

Output:

```
2
```

Solution

```
typedef pair<ll,ll> pll;
const int maxN = 2e5+1;
int N;
ll best;
bool isLeft[maxN];
pll P[maxN];
ll square(ll x){
    return x*x;
}
ll dist(pll A, pll B){
    ll dx = A.first-B.first;
    ll dy = A.second-B.second;
    return square(dx) + square(dy);
}
void solve(vector<int> PX, vector<int> PY){
    int len = PX.size();
    if(len == 1)  return;
    vector<int> LX, LY, RX, RY;
    for(int i = 0; i < len; i++){
        if(i < len/2){
            LX.push_back(PX[i]);
            isLeft[PX[i]] = true;
        } else {
            RX.push_back(PX[i]);
            isLeft[PX[i]] = false;
```

```
        }
    }
    for(int id : PY){
        if(isLeft[id])  LY.push_back(id);
        else            RY.push_back(id);
    }
    solve(LX, LY);
    solve(RX, RY);
    ll midX = P[LX.back()].first;
    vector<int> stripe;
    for(int id : PY)
        if(square(P[id].first-midX) < best)
            stripe.push_back(id);
    for(int i = 0; i < (int) stripe.size(); i++)
        for(int j = i+1; j < (int) stripe.size() && square(P[stripe[i]].second-P[stripe[j]].second) < best
            best = min(best, dist(P[stripe[i]], P[stripe[j]]));
}
int main(){
    scanf("%d", &N);
    vector<int> sortedX, sortedY;
    for(int i = 0, x, y; i < N; i++){
        scanf("%d %d", &x, &y);
        P[i] = {x, y};
        sortedX.push_back(i);
        sortedY.push_back(i);
    }
    sort(sortedX.begin(), sortedX.end(), [](int A, int B){
        return P[A].first < P[B].first;
    });
    sort(sortedY.begin(), sortedY.end(), [](int A, int B){
        return P[A].second < P[B].second;
    });
    best = LLONG_MAX;
    solve(sortedX, sortedY);
    printf("%lld\n", best);
}
```

Point Location Test

Time limit: 1.00 s **Memory limit:** 512 MB

There is a line that goes through the points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$. There is also a point $p_3 = (x_3, y_3)$. Your task is to determine whether p_3 is located on the left or right side of the line or if it touches the line when we are looking from p_1 to p_2.

Input

The first input line has an integer t: the number of tests.

After this, there are t lines that describe the tests. Each line has six integers: x_1, y_1, x_2, y_2, x_3 and y_3.

Output

For each test, print "LEFT", "RIGHT" or "TOUCH".

Constraints

- $1 \le t \le 10^5$
- $-10^9 \le x_1, y_1, x_2, y_2, x_3, y_3 \le 10^9$

- $x_1 \neq x_2$ or $y_1 \neq y_2$

Example

Input:

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

Output:

```
LEFT
RIGHT
TOUCH
```

Solution

```
struct Point {
    int x, y;
    void read(){ scanf("%d %d", &x, &y); }
    Point operator +(const Point& b) const { return Point{x+b.x, y+b.y}; }
    Point operator -(const Point& b) const { return Point{x-b.x, y-b.y}; }
    ll operator *(const Point& b) const { return (ll) x * b.y - (ll) y * b.x; }
    void operator +=(const Point& b) { x += b.x; y += b.y; }
    void operator -=(const Point &b) { x -= b.x; y -= b.y; }
    void operator *=(const int k) { x *= k; y *= k; }
    ll cross(const Point& b, const Point& c) const {
        return (b - *this) * (c - *this);
    }
};
int T;
Point P[3];
int main(){
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        for(int i = 0; i < 3; i++)
            P[i].read();
        ll cross = P[0].cross(P[1], P[2]);
        if(cross < 0)       printf("RIGHT\n");
        else if(cross > 0)  printf("LEFT\n");
        else                printf("TOUCH\n");
    }
}
```

Point in Polygon

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a polygon of n vertices and a list of m points. Your task is to determine for each point if it is inside, outside or on the boundary of the polygon. The polygon consists of n vertices $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. The vertices (x_i, y_i) and (x_{i+1}, y_{i+1}) are adjacent for $i = 1, 2, \ldots, n-1$, and the vertices (x_1, y_1) and (x_n, y_n) are also adjacent.

Input

The first input line has two integers n and m: the number of vertices in the polygon and the number of points.

After this, there are n lines that describe the polygon. The ith such line has two integers x_i and y_i.

You may assume that the polygon is simple, i.e., it does not intersect itself.

Finally, there are m lines that describe the points. Each line has two integers x and y.

Output

For each point, print "INSIDE", "OUTSIDE" or "BOUNDARY".

Constraints

- $3 \leq n, m \leq 1000$
- $1 \leq m \leq 1000$
- $-10^9 \leq x_i, y_i \leq 10^9$
- $-10^9 \leq x, y \leq 10^9$

Example

Input:

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

Output:

```
INSIDE
OUTSIDE
BOUNDARY
```

Solution

```
const int maxN = 1001;
struct Point {
    int x, y;
    void read(){ scanf("%d %d", &x, &y); }
    Point operator +(const Point& b) const { return Point{x+b.x, y+b.y}; }
    Point operator -(const Point& b) const { return Point{x-b.x, y-b.y}; }
    ll operator *(const Point& b) const { return (ll) x * b.y - (ll) y * b.x; }
    void operator +=(const Point& b) { x += b.x; y += b.y; }
    void operator -=(const Point& b) { x -= b.x; y -= b.y; }
    void operator *=(const int k) { x *= k; y *= k; }
    ll cross(const Point& b, const Point& c) const {
        return (b - *this) * (c - *this);
    }
};
int N, M;
Point P[maxN];
bool pointlineintersect(Point P1, Point P2, Point P3){
    if(P2.cross(P1, P3) != 0)   return false;
    return (min(P2.x, P3.x) <= P1.x && P1.x <= max(P2.x, P3.x))
        && (min(P2.y, P3.y) <= P1.y && P1.y <= max(P2.y, P3.y));
}
void pointinpolygon(){
    int cnt = 0;
    bool boundary = false;
```

```
    for(int i = 1; i <= N; i++){
        int j = (i == N ? 1 : i+1);
        if(pointlineintersect(P[0], P[i], P[j]))
            boundary = true;
        if(P[i].x <= P[0].x && P[0].x < P[j].x && P[0].cross(P[i], P[j]) < 0)       cnt++;
        else if(P[j].x <= P[0].x && P[0].x < P[i].x && P[0].cross(P[j], P[i]) < 0)  cnt++;
    }
    if(boundary)    printf("BOUNDARY\n");
    else if(cnt&1)  printf("INSIDE\n");
    else            printf("OUTSIDE\n");
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 1; i <= N; i++)
        P[i].read();
    for(int i = 0; i < M; i++){
        P[0].read();
        pointinpolygon();
    }
}
```

Polygon Area

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to calculate the area of a given polygon. The polygon consists of n vertices $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. The vertices (x_i, y_i) and (x_{i+1}, y_{i+1}) are adjacent for $i = 1, 2, \ldots, n-1$, and the vertices (x_1, y_1) and (x_n, y_n) are also adjacent.

Input

The first input line has an integer n: the number of vertices.

After this, there are n lines that describe the vertices. The ith such line has two integers x_i and y_i.

You may assume that the polygon is simple, i.e., it does not intersect itself.

Output

Print one integer: $2a$ where the area of the polygon is a (this ensures that the result is an integer).

Constraints

- $3 \le n \le 1000$
- $-10^9 \le x_i, y_i \le 10^9$

Example

Input:

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

Output:

```
16
```

Solution

```
const int maxN = 1001;
int N;
ll x[maxN], y[maxN], ans;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)
        scanf("%lld %lld", &x[i], &y[i]);
    x[N] = x[0]; y[N] = y[0];
    for(int i = 0; i < N; i++){
        ans += x[i] * y[i+1];
        ans -= y[i] * x[i+1];
    }
    printf("%lld\n", abs(ans));
}
```

Polygon Lattice Points

Time limit: 1.00 s **Memory limit:** 512 MB

Given a polygon, your task is to calculate the number of lattice points inside the polygon and on its boundary. A lattice point is a point whose coordinates are integers. The polygon consists of n vertices $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. The vertices (x_i, y_i) and (x_{i+1}, y_{i+1}) are adjacent for $i = 1, 2, \ldots, n-1$, and the vertices (x_1, y_1) and (x_n, y_n) are also adjacent.

Input

The first input line has an integer n: the number of vertices.

After this, there are n lines that describe the vertices. The ith such line has two integers x_i and y_i.

You may assume that the polygon is simple, i.e., it does not intersect itself.

Output

Print two integers: the number of lattice points inside the polygon and on its boundary.

Constraints

- $3 \le n \le 10^5$
- $-10^9 \le x_i, y_i \le 10^9$

Example

Input:

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

Output:

```
6 8
```

Solution

```
const int maxN = 1e5+5;
int N;
ll area, boundary, inside, x[maxN], y[maxN];
```

```
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)
        scanf("%lld %lld", &x[i], &y[i]);
    x[N] = x[0]; y[N] = y[0];
    for(int i = 0; i < N; i++){
        area += x[i] * y[i+1];
        area -= y[i] * x[i+1];
    }
    area = abs(area);
    for(int i = 0; i < N; i++){
        if(x[i+1] == x[i])      boundary += abs(y[i+1]-y[i]);
        else if(y[i+1] == y[i]) boundary += abs(x[i+1]-x[i]);
        else boundary += __gcd(abs(x[i+1]-x[i]), abs(y[i+1]-y[i]));
    }
    printf("%lld %lld\n", (area+2-boundary)/2, boundary);
}
```

Graph Algorithms Ramez

Building Roads (DSU, finding minimum spanning tree)

Time limit: 1.00 s **Memory limit:** 512 MB

Byteland has n cities, and m roads between them. The goal is to construct new roads so that there is a route between any two cities. Your task is to find out the minimum number of roads required, and also determine which roads should be built.

Input

The first input line has two integers n and m: the number of cities and roads. The cities are numbered $1, 2, \ldots, n$.

After that, there are m lines describing the roads. Each line has two integers a and b: there is a road between those cities.

A road always connects two different cities, and there is at most one road between any two cities.

Output

First print an integer k: the number of required roads.

Then, print k lines that describe the new roads. You can print any valid solution.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
1
2 3
```

Solution

```
typedef pair<int,int> pii;
int N, M, a, b, ds[100001];
vector<pii> ans;
int find(int u){
    if(ds[u] < 0)   return u;
    ds[u] = find(ds[u]);
    return ds[u];
}
bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v)  return false;
    if(ds[u] < ds[v])   swap(u, v);
    ds[v] += ds[u];
    ds[u] = v;
    return true;
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 1; i <= N; i++) ds[i] = -1;
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        merge(a, b);
    }
    for(int i = 1; i < N; i++)
        if(merge(i, i+1))
            ans.push_back({i, i+1});
    printf("%d\n", (int) ans.size());
    for(pii P : ans)
        printf("%d %d\n", P.first, P.second);
}
```

Building Teams (DFS, creating two separate teams)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n pupils in Uolevi's class, and m friendships between them. Your task is to divide the pupils into two teams in such a way that no two pupils in a team are friends. You can freely choose the sizes of the teams.

Input

The first input line has two integers n and m: the number of pupils and friendships. The pupils are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the friendships. Each line has two integers a and b: pupils a and b are friends.

Every friendship is between two different pupils. You can assume that there is at most one friendship between any two pupils.

Output

Print an example of how to build the teams. For each pupil, print "1" or "2" depending on to which team the pupil will be assigned. You can print any valid team.

If there are no solutions, print "IMPOSSIBLE".

Constraints

- $1 \leq n \leq 10^5$

- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
1 2 2 1 2
```

Solution

```
const int maxN = 1e5+1;
int N, M, a, b;
bool possible, vis[maxN], team[maxN];
vector<int> G[maxN];
void dfs(int u, int p = 0){
    for(int v : G[u]){
        if(v != p){
            if(!vis[v]){
                team[v] = !team[u];
                vis[v] = true;
                dfs(v, u);
            } else
                if(team[v] == team[u])
                    possible = false;
        }
    }
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
    }
    possible = true;
    for(int i = 1; i <= N; i++){
        if(!vis[i]){
            vis[i] = true;
            dfs(i);
        }
    }
    if(!possible)
        printf("IMPOSSIBLE\n");
    else
        for(int i = 1; i <= N; i++)
            printf("%d%c", (team[i] ? 1 : 2), (" \n")[i==N]);
}
```

Coin Collector (DFS, maximum path between any two nodes with weighted nodes)

Time limit: 1.00 s **Memory limit:** 512 MB

A game has n rooms and m tunnels between them. Each room has a certain number of coins. What is the maximum number of coins you can collect while moving through the tunnels when you can freely choose your starting and ending room?

Input

The first input line has two integers n and m: the number of rooms and tunnels. The rooms are numbered $1, 2, \ldots, n$.

Then, there are n integers $k_1, k_2, \ldots, k_n$: the number of coins in each room.

Finally, there are m lines describing the tunnels. Each line has two integers a and b: there is a tunnel from room a to room b. Each tunnel is a one-way tunnel.

Output

Print one integer: the maximum number of coins you can collect.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le k_i \le 10^9$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
16
```

Solution

```
const int maxN = 1e5+1;
bool vis[maxN];
int N, M, K, rt[maxN];
ll k[maxN], ck[maxN], dp[maxN];
vector<int> ord, comp, G[maxN], GR[maxN], SCC[maxN];
void dfs1(int u){
    vis[u] = true;
    for(int v : G[u])
        if(!vis[v])
            dfs1(v);
    ord.push_back(u);
}
void dfs2(int u){
    vis[u] = true;
    comp.push_back(u);
    for(int v : GR[u])
        if(!vis[v])
            dfs2(v);
}
void dfs3(int u){
```

```
    vis[u] = true;
    dp[u] = ck[u];
    for(int v : SCC[u]){
        if(!vis[v])
            dfs3(v);
        dp[u] = max(dp[u], dp[v]+ck[u]);
    }
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 1; i <= N; i++)
        scanf("%d", &k[i]);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        GR[b].push_back(a);
    }
    for(int i = 1; i <= N; i++)
        if(!vis[i])
            dfs1(i);
    fill(vis+1, vis+N+1, false);
    reverse(ord.begin(), ord.end());
    for(int u : ord){
        if(!vis[u]){
            dfs2(u);
            K++;
            for(int v : comp){
                ck[K] += k[v];
                rt[v] = K;
            }
            comp.clear();
        }
    }
    for(int u = 1; u <= N; u++)
        for(int v : G[u])
            if(rt[v] != rt[u])
                SCC[rt[u]].push_back(rt[v]);
    fill(vis+1, vis+K+1, false);
    for(int i = 1; i <= K; i++)
        if(!vis[i])
            dfs3(i);
    ll best = 0;
    for(int i = 1; i <= K; i++)
        best = max(best, dp[i]);
    printf("%lld\n", best);
}
```

Counting Rooms (DFS, Counting componenets)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a map of a building, and your task is to count the number of its rooms. The size of the map is $n \times m$ squares, and each square is either floor or wall. You can walk left, right, up, and down through the floor squares.

Input

The first input line has two integers n and m: the height and width of the map.

Then there are n lines of m characters describing the map. Each character is either . (floor) or # (wall).

Output

Print one integer: the number of rooms.

Constraints

- $1 \leq n, m \leq 1000$

Example

Input:

```
5 8
######
##..#...#
#####.#.#
##..#...#
######
```

Output:

```
3
```

Solution

```
char c;
int N, M, cnt;
int h[] = {1, -1, 0, 0}, v[] = {0, 0, 1, -1};
bool vis[1000][1000];
void dfs(int x, int y){
    vis[x][y] = true;
    for(int i = 0; i < 4; i++){
        int dx = x+h[i], dy = y+v[i];
        if(0 <= dx && dx < N && 0 <= dy && dy < M && !vis[dx][dy])
            dfs(dx, dy);
    }
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0; i < N; i++){
        for(int j = 0; j < M; j++){
            scanf(" %c", &c);
            vis[i][j] = (c == '#');
        }
    }
    for(int i = 0; i < N; i++){
        for(int j = 0; j < M; j++){
            if(!vis[i][j]){
                dfs(i, j);
                cnt++;
            }
        }
    }
    printf("%d\n", cnt);
}
```

Course Schedule (Topo Sort, Scheduling courses with dependencies)

Time limit: 1.00 s **Memory limit:** 512 MB

You have to complete n courses. There are m requirements of the form "course a has to be completed before course b". Your task is to find an order in which you can complete the courses.

Input

The first input line has two integers n and m: the number of courses and requirements. The courses are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the requirements. Each line has two integers a and b: course a has to be completed before course b.

Output

Print an order in which you can complete the courses. You can print any valid order that includes all the courses.

If there are no solutions, print "IMPOSSIBLE".

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq m \leq 2 \cdot 10^5$
- $1 \leq a, b \leq n$

Example

Input:

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

Output:

```
3 4 1 5 2
```

Solution

```cpp
//A topological sort takes a directed acyclic graph (DAG) and produces
//a linear ordering of its vertices such that for every directed edge
// u → v, u comes before v in that order.
// Returns a vector of nodes in a valid order; if a cycle exists, the size will be < n.
// useful for: Scheduling with Dependencies, Course Prerequisites,
vi topologicalSort(int n, vector<vi>& adj, vi& inDeg) {
    queue<int> q;
    for (int i = 1; i <= n; i++) {
        if (inDeg[i] == 0)
            q.push(i);
    }

    vi order;

    while (!q.empty()) {
        int parent = q.front(); q.pop();
        order.push_back(parent);
        for (int child : adj[parent]) {
            if (--inDeg[child] == 0)
```

```
                q.push(child);
            }
        }

        return order;
}

void Ramez() {
        int n, m; cin >> n >> m;
        vector<vi> adj(n + 1);
        vi inDeg(n + 1);

        for (int i = 0; i < m; i++) {
            int a, b;
            cin >> a >> b;
            adj[a].push_back(b);
            inDeg[b]++;
        }

        vi order = topologicalSort(n, adj, inDeg);

        if (order.size() < n) {
            cout << "IMPOSSIBLE\n";
            return;
        }

        for (int c : order)
            cout << c << ' ';
        cout << '\n';
}
}
```

Cycle Finding (Bellman-Ford, Negative Cycles Detection)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a directed graph, and your task is to find out if it contains a negative cycle, and also give an example of such a cycle.

Input

The first input line has two integers n and m: the number of nodes and edges. The nodes are numbered $1, 2, \ldots, n$.

After this, the input has m lines describing the edges. Each line has three integers a, b, and c: there is an edge from node a to node b whose length is c.

Output

If the graph contains a negative cycle, print first "YES", and then the nodes in the cycle in their correct order. If there are several negative cycles, you can print any of them. If there are no negative cycles, print "NO".

Constraints

- $1 \le n \le 2500$
- $1 \le m \le 5000$
- $1 \le a, b \le n$
- $-10^9 \le c \le 10^9$

Example

Input:

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

Output:

```
YES
1 2 4 1
```

Solution

```
struct edge {
    int u, v, w;
};

// Bellman-Ford Algorithm (Negative Cycles Detection)
void Ramez() {
    int n, m; cin >> n >> m;
    vector<edge> edges;
    for (int i = 0; i < m; i++) {
        int u, v, w; cin >> u >> v >> w;
        edges.push_back({ u, v, w });
    }

    vi dis(n + 1, 0), parent(n + 1, -1);


    // Relaxation (n - 1) times
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            auto [u, v, w] = edges[j];
            if (dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                parent[v] = u;
            }
        }
    }

    // if any edge can still be relaxed, that means there's a cycle
    int x = -1;
    for (int i = 0; i < m; i++){
        auto [u, v, w] = edges[i];
        if (dis[v] > dis[u] + w) {
            dis[v] = dis[u] + w;
            parent[v] = u;
            x = v;
        }
    }

    if (x == -1) {
        cout << "NO\n";
        return;
```

```
    }


    // x is a vertex that was relaxed in the nth iteration.
    // To ensure we are inside the cycle, move n steps from x
    int y = x;
    for (int i = 0; i < n; i++) {
        y = parent[y];
    }

    // Reconstruct the cycle
    vector<int> cycle;
    int curr = y;
    do {
        cycle.push_back(curr);
        curr = parent[curr];
    } while (curr != y);
    cycle.push_back(y);
    reverse(all(cycle));

    cout << "YES\n" << cycle;
}
```

De Bruijn Sequence (bit string that contains all possible substrings of length n)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to construct a minimum-length bit string that contains all possible substrings of length n. For example, when $n = 2$, the string 00110 is a valid solution, because its substrings of length 2 are 00, 01, 10 and 11.

Input

The only input line has an integer n.

Output

Print a minimum-length bit string that contains all substrings of length n. You can print any valid solution.

Constraints

- $1 \le n \le 15$

Example

Input:

```
2
```

Output:

```
00110
```

Solution

```
const int maxN = 15;
const int SIZE = (1<<maxN)+maxN;
char ans[SIZE];
int N, ansptr;
vector<int> G[SIZE];
stack<int> S;
```

```
void init(){
    for(int u = 0; u < 1<<(N-1); u++){
        int v = (u<<1)%(1<<(N-1));
        G[u].push_back(v);
        G[u].push_back(v|1);
    }
}
void solve(){
    S.push(0);
    while(!S.empty()){
        int u = S.top();
        if(!G[u].empty()){
            int v = G[u].back();
            G[u].pop_back();
            S.push(v);
        } else {
            ans[ansptr++] = (char) (u&1) + '0';
            S.pop();
        }
    }
    for(int i = 0; i < N-2; i++)
        ans[ansptr++] = '0';
}
int main(){
    scanf("%d", &N);
    if(N == 1){
        printf("01");
        return 0;
    }
    init();
    solve();
    for(int i = 0; i < ansptr; i++)
        printf("%c", ans[i]);
}
```

Distinct Routes (maximum number of distinct routes between 1 and n)

Time limit: 1.00 s **Memory limit:** 512 MB

A game consists of n rooms and m teleporters. At the beginning of each day, you start in room 1 and you have to reach room n. You can use each teleporter at most once during the game. How many days can you play if you choose your routes optimally?

Input

The first input line has two integers n and m: the number of rooms and teleporters. The rooms are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the teleporters. Each line has two integers a and b: there is a teleporter from room a to room b.

There are no two teleporters whose starting and ending room are the same.

Output

First print an integer k: the maximum number of days you can play the game. Then, print k route descriptions according to the example. You can print any valid solution.

Constraints

- $2 \leq n \leq 500$
- $1 \leq m \leq 1000$
- $1 \leq a, b \leq n$

Example

Input:

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

Output:

```
2
3
1 2 6
4
1 3 4 6
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 501, maxM = 1001;
const int INF = 0x3f3f3f3f;
int N, M, p[maxN], cap[maxN][maxN];
bool vis[maxM];
vector<int> path, F[maxN];
vector<pii> G[maxN];
int bfs(int s = 1, int t = N){
    fill(p+1, p+N+1, -1);
    p[s] = -2;
    queue<pii> Q;
    Q.push({s, INF});
    while(!Q.empty()){
        int u = Q.front().first;
        int f = Q.front().second;
        Q.pop();
        for(int v : F[u]){
            if(p[v] == -1 && cap[u][v]){
                p[v] = u;
                int aug = min(f, cap[u][v]);
                if(v == t)  return aug;
                Q.push({v, aug});
            }
        }
    }
    return 0;
}
void dfs(int u = 1){
    path.push_back(u);
    if(u == N)  return;
```

```
    for(pii e : G[u]){
        int v = e.first;
        int id = e.second;
        if(cap[u][v] == 0 && !vis[id]){
            vis[id] = true;
            dfs(v);
            return;
        }
    }
}
int maxflow(int s = 1, int t = N){
    int flow = 0, aug = 0;
    while(aug = bfs()){
        flow += aug;
        int u = t;
        while(u != s){
            int v = p[u];
            cap[v][u] -= aug;
            cap[u][v] += aug;
            u = v;
        }
    }
    return flow;
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back({b, i});
        F[a].push_back(b);
        F[b].push_back(a);
        cap[a][b]++;
    }
    int K = maxflow();
    printf("%d\n", K);
    for(int k = 0; k < K; k++){
        path.clear();
        dfs();
        int sz = (int) path.size();
        printf("%d\n", sz);
        for(int i = 0; i < sz; i++)
            printf("%d%c", path[i], (" \n")[i==sz-1]);
    }
}
```

Download Speed (Maximum routes sum from 1 to n)

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a network consisting of n computers and m connections. Each connection specifies how fast a computer can send data to another computer. Kotivalo wants to download some data from a server. What is the maximum speed he can do this, using the connections in the network?

Input

The first input line has two integers n and m: the number of computers and connections. The computers are numbered $1, 2, \ldots, n$. Computer 1 is the server and computer n is Kotivalo's computer.

After this, there are m lines describing the connections. Each line has three integers a, b and c: computer a can send data to computer b at speed c.

Output

Print one integer: the maximum speed Kotivalo can download data.

Constraints

- $1 \leq n \leq 500$
- $1 \leq m \leq 1000$
- $1 \leq a, b \leq n$
- $1 \leq c \leq 10^9$

Example

Input:

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

Output:

```
6
```

Solution

```
typedef pair<int,ll> pil;
const int maxN = 501;
const ll INF = 0x3f3f3f3f3f3f3f3f;
int N, M, p[maxN];
ll cap[maxN][maxN];
vector<int> G[maxN];
ll bfs(int s = 1, int t = N){
    fill(p+1, p+N+1, -1);
    p[s] = -2;
    queue<pil> Q;
    Q.push({s, INF});
    while(!Q.empty()){
        int u = Q.front().first;
        ll f = Q.front().second;
        Q.pop();
        for(int v : G[u]){
            if(p[v] == -1 && cap[u][v]){
                p[v] = u;
                ll aug = min(f, cap[u][v]);
                if(v == t)  return aug;
                Q.push({v, aug});
            }
        }
    }
    return 0;
}
ll maxflow(int s = 1, int t = N){
    ll flow = 0, aug = 0;
```

```
    while(aug = bfs()){
        flow += aug;
        int u = t;
        while(u != s){
            int v = p[u];
            cap[v][u] -= aug;
            cap[u][v] += aug;
            u = v;
        }
    }
    return flow;
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        ll c;
        scanf("%d %d %lld", &a, &b, &c);
        G[a].push_back(b);
        G[b].push_back(a);
        cap[a][b] += c;
    }
    printf("%lld\n", maxflow());
}
```

Flight Discount (Dijkstra, minimum route with one coupon for one edge)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to find a minimum-price flight route from Syrjälä to Metsälä. You have one discount coupon, using which you can halve the price of any single flight during the route. However, you can only use the coupon once. When you use the discount coupon for a flight whose price is x, its price becomes $\lfloor x/2 \rfloor$ (it is rounded down to an integer).

Input

The first input line has two integers n and m: the number of cities and flight connections. The cities are numbered $1, 2, \ldots, n$. City 1 is Syrjälä, and city n is Metsälä.

After this there are m lines describing the flights. Each line has three integers a, b, and c: a flight begins at city a, ends at city b, and its price is c. Each flight is unidirectional.

You can assume that it is always possible to get from Syrjälä to Metsälä.

Output

Print one integer: the price of the cheapest route from Syrjälä to Metsälä.

Constraints

- $2 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$
- $1 \le c \le 10^9$

Example

Input:

```
3 4
1 2 3
```

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

Output:

```
2
```

Solution

```
void Ramez() {
    int n, m; cin >> n >> m;
    vector<vector<pii>> adj(n + 1);
    for (int i = 0; i < m; i++) {
        int a, b, c; cin >> a >> b >> c;
        adj[a].push_back({ b, c });
    }

    vector<vi> dis(n+1, vi(2, LLONG_MAX));
    dis[1][0] = 0;

    // Priority queue: (current cost, current node, coupon used state)
    priority_queue<tuple<int,int,int>, vector<tuple<int,int,int>>, greater<>> pq;
    pq.push({0, 1, 0});

    while (!pq.empty()) {
        auto [parentCost, parent, used] = pq.top(); pq.pop();
        if (parentCost > dis[parent][used]) continue;

        for (auto [child, childCost] : adj[parent]) {
            // Option 1: Do not use coupon.
            if (dis[child][used] > parentCost + childCost) {
                dis[child][used] = parentCost + childCost;
                pq.push({dis[child][used], child, used});
            }

            // Option 2: Use coupon (only if not used before).
            if (!used && dis[child][1] > parentCost + (childCost / 2)) {
                dis[child][1] = parentCost + (childCost / 2);
                pq.push({dis[child][1], child, 1});
            }
        }
    }

    cout << min(dis[n][0], dis[n][1]) << " ";
}
```

Flight Routes (k-dijkstra, finding the first k shortest routes)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to find the k shortest flight routes from Syrjälä to Metsälä. A route can visit the same city several times. Note that there can be several routes with the same price and each of them should be considered (see the example).

Input

The first input line has three integers n, m, and k: the number of cities, the number of flights, and the parameter k. The cities are numbered $1, 2, \ldots, n$. City 1 is Syrjälä, and city n is Metsälä.

After this, the input has m lines describing the flights. Each line has three integers a, b, and c: a flight begins at city a, ends at city b, and its price is c. All flights are one-way flights.

You may assume that there are at least k distinct routes from Syrjälä to Metsälä.

Output

Print k integers: the prices of the k cheapest routes sorted according to their prices.

Constraints

- $2 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$
- $1 \le c \le 10^9$
- $1 \le k \le 10$

Example

Input:

```
4 6 3
1 2 1
1 3 3
2 3 2
2 4 6
3 2 8
3 4 1
```

Output:

```
4 4 7
```

Explanation: The cheapest routes are $1 \to 3 \to 4$ (price 4), $1 \to 2 \to 3 \to 4$ (price 4) and $1 \to 2 \to 4$ (price 7).

Solution

```cpp
// K-Dijkstra
void Ramez() {
    int n, m, k; cin >> n >> m >> k;
    vector<vector<pii>> adj(n + 1);

    for (int i = 0; i < m; i++) {
        int a, b, c; cin >> a >> b >> c;
        adj[a].push_back({ b, c });
    }

    vector<vi> dis(n + 1);
    priority_queue<pii, vector<pii>, greater<pii>> pq; // {cost, node}
    pq.push({ 0, 1 });

    while (!pq.empty()) {
        auto [parentCost, parent] = pq.top(); pq.pop();
        if (dis[parent].size() >= k) continue;
        dis[parent].push_back(parentCost);

        for (auto [child, childCost] : adj[parent]) {
            pq.push({ parentCost + childCost, child });
        }
    }
```

```
    cout << dis[n] << "\n";
}
```

Flight Routes Check (Checking if a directed graph is strongly connected componenet (SCC))

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities and m flight connections. Your task is to check if you can travel from any city to any other city using the available flights.

Input

The first input line has two integers n and m: the number of cities and flights. The cities are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the flights. Each line has two integers a and b: there is a flight from city a to city b. All flights are one-way flights.

Output

Print "YES" if all routes are possible, and "NO" otherwise. In the latter case also print two cities a and b such that you cannot travel from city a to city b. If there are several possible solutions, you can print any of them.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
NO
4 2
```

Solution

```
const int maxN = 2e5+1;
int N, M, a, b;
bool vis1[maxN], vis2[maxN];
vector<int> G1[maxN], G2[maxN];
void dfs1(int u = 1, int p = 0){
    vis1[u] = true;
    for(int v : G1[u])
        if(v != p && !vis1[v])
            dfs1(v, u);
}
void dfs2(int u = 1, int p = 0){
    vis2[u] = true;
```

```
    for(int v : G2[u])
        if(v != p && !vis2[v])
            dfs2(v, u);
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        G1[a].push_back(b);
        G2[b].push_back(a);
    }
    dfs1();
    dfs2();
    for(int i = 1; i <= N; i++){
        if(!vis1[i] || !vis2[i]){
            printf("NO\n");
            if(!vis1[i])    printf("1 %d\n", i);
            else            printf("%d 1\n", i);
            return 0;
        }
    }
    printf("YES\n");
}
```

Game Routes (DP on DAG, number of ways to reach i from 1)

Time limit: 1.00 s **Memory limit:** 512 MB

A game has n levels, connected by m teleporters, and your task is to get from level 1 to level n. The game has been designed so that there are no directed cycles in the underlying graph. In how many ways can you complete the game?

Input

The first input line has two integers n and m: the number of levels and teleporters. The levels are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the teleporters. Each line has two integers a and b: there is a teleporter from level a to level b.

Output

Print one integer: the number of ways you can complete the game. Since the result may be large, print it modulo $10^9 + 7$.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

```
3 4
1 4
```

Output:

```
3
```

Solution

```
void Ramez() {
    int n, m; cin >> n >> m;
    vector<vi> adj(n + 1);
    for (int i = 0; i < m; i++){
        int u, v; cin >> u >> v;
        adj[u].push_back(v);
    }

    vi ways(n + 1, -1);

    function<int(int)> dfs = [&](int u) -> int {
        if(u == n) return 1;
        if(ways[u] != -1) return ways[u];

        int ans = 0;
        for (int v : adj[u]){
            ans = (ans + dfs(v))%MOD;
        }

        return ways[u] = ans;
    };

    cout << dfs(1);
}
```

Giant Pizza (Kosaraju's, finding a strongly connected component in a graph (SCC))

Time limit: 1.00 s **Memory limit:** 512 MB

Uolevi's family is going to order a large pizza and eat it together. A total of n family members will join the order, and there are m possible toppings. The pizza may have any number of toppings. Each family member gives two wishes concerning the toppings of the pizza. The wishes are of the form "topping x is good/bad". Your task is to choose the toppings so that at least one wish from everybody becomes true (a good topping is included in the pizza or a bad topping is not included).

Input

The first input line has two integers n and m: the number of family members and toppings. The toppings are numbered $1, 2, \ldots, m$.

After this, there are n lines describing the wishes. Each line has two wishes of the form "+ x" (topping x is good) or "- x" (topping x is bad).

Output

Print a line with m symbols: for each topping "+" if it is included and "-" if it is not included. You can print any valid solution.

If there are no valid solutions, print "IMPOSSIBLE".

Constraints

- $1 \leq n, m \leq 10^5$
- $1 \leq x \leq m$

Example

Input:

```
3 5
+ 1 + 2
- 1 + 3
+ 4 - 2
```

Output:

```
- + + + -
```

Solution

```
const int maxN = 2e5+1;
bool vis[maxN];
char ans[maxN];
int N, M, K, in[maxN], rt[maxN];
vector<int> ord, comp, G[maxN], GR[maxN], C[maxN], SCC[maxN];
int flip(int x){
    return (x&1 ? x+1 : x-1);
}
void add_edge(char c1, int a, char c2, int b){
    a = 2*a - (c1 == '-');
    b = 2*b - (c2 == '-');
    G[flip(a)].push_back(b);
    G[flip(b)].push_back(a);
    GR[a].push_back(flip(b));
    GR[b].push_back(flip(a));
}
void dfs1(int u){
    vis[u] = true;
    for(int v : G[u])
        if(!vis[v])
            dfs1(v);
    ord.push_back(u);
}
void dfs2(int u){
    vis[u] = true;
    comp.push_back(u);
    for(int v : GR[u])
        if(!vis[v])
            dfs2(v);
}
int main(){
    scanf("%d %d", &M, &N);
    for(int i = 0, a, b; i < M; i++){
        char c1, c2;
        scanf(" %c %d %c %d", &c1, &a, &c2, &b);
        add_edge(c1, a, c2, b);
    }
    for(int i = 1; i <= 2*N; i++)
        if(!vis[i])
```

```
        dfs1(i);
fill(vis+1, vis+2*N+1, false);
reverse(ord.begin(), ord.end());
for(int u : ord){
    if(!vis[u]){
        dfs2(u);
        K++;
        for(int v : comp){
            rt[v] = K;
            C[K].push_back(v);
        }
        comp.clear();
    }
}
// Impossible iff x and not(x) belong to same SCC
for(int i = 1; i <= N; i++){
    if(rt[2*i] == rt[2*i-1]){
        printf("IMPOSSIBLE\n");
        return 0;
    }
}
for(int u = 1; u <= 2*N; u++){
    for(int v : G[u]){
        if(rt[u] != rt[v]){
            SCC[rt[u]].push_back(rt[v]);
            in[rt[v]]++;
        }
    }
}
queue<int> Q;
ord.clear();
for(int u = 1; u <= K; u++){
    if(in[u] == 0){
        ord.push_back(u);
        Q.push(u);
    }
}
while(!Q.empty()){
    int u = Q.front(); Q.pop();
    for(int v : SCC[u]){
        in[v]--;
        if(in[v] == 0){
            ord.push_back(v);
            Q.push(v);
        }
    }
}
fill(vis+1, vis+N+1, false);
reverse(ord.begin(), ord.end());
for(int k : ord){
    for(int u : C[k]){
        int i = (u+1)/2;
        if(!vis[i]){
            ans[i] = (u&1 ? '-' : '+');
            vis[i] = true;
        }
```

```
        }
    }
    for(int i = 1; i <= N; i++)
        printf("%c%c", ans[i], (" \n")[i==N]);
}
```

Hamiltonian Flights (Counting Hamiltonian Paths, path that visits every vertex once)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities and m flight connections between them. You want to travel from Syrjälä to Lehmälä so that you visit each city exactly once. How many possible routes are there?

Input

The first input line has two integers n and m: the number of cities and flights. The cities are numbered $1, 2, \ldots, n$. City 1 is Syrjälä, and city n is Lehmälä.

Then, there are m lines describing the flights. Each line has two integers a and b: there is a flight from city a to city b. All flights are one-way flights.

Output

Print one integer: the number of routes modulo $10^9 + 7$.

Constraints

- $2 \le n \le 20$
- $1 \le m \le n^2$
- $1 \le a, b \le n$

Example

Input:

```
4 6
1 2
1 3
2 3
3 2
2 4
3 4
```

Output:

```
2
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 20;
const int SIZE = 1<<maxN;
const ll MOD = 1e9+7;
int N, M;
bool inqueue[maxN][SIZE];
// dp[mask][u] = "number of ways to start at 1, visit exactly the cities in mask, and end at city u."
// We only allow masks that always include city 1, and we'll build up masks by gradually adding one unvisi
ll dp[maxN][SIZE];
vector<int> G[maxN];
queue<pii> Q;
int main(){
```

```
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a-1].push_back(b-1);
    }
    dp[0][1] = 1;
    Q.push({0, 1});
    inqueue[0][1] = true;
    while(!Q.empty()){
        int u = Q.front().first;
        int mask = Q.front().second;
        Q.pop();
        if(u != N-1){
            for(int v : G[u]){
                int newMask = mask|(1<<v);
                if((mask&(1<<v)) == 0){
                    dp[v][newMask] += dp[u][mask];
                    dp[v][newMask] %= MOD;
                    if(!inqueue[v][newMask]){
                        Q.push({v, newMask});
                        inqueue[v][newMask] = true;
                    }
                }
            }
        }
    }
    printf("%lld\n", dp[N-1][(1<<N)-1]);
}
```

High Score (Bellman ford, Detecting positive cycles)

Time limit: 1.00 s **Memory limit:** 512 MB

You play a game consisting of n rooms and m tunnels. Your initial score is 0, and each tunnel increases your score by x where x may be both positive or negative. You may go through a tunnel several times. Your task is to walk from room 1 to room n. What is the maximum score you can get?

Input

The first input line has two integers n and m: the number of rooms and tunnels. The rooms are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the tunnels. Each line has three integers a, b and x: the tunnel starts at room a, ends at room b, and it increases your score by x. All tunnels are one-way tunnels.

You can assume that it is possible to get from room 1 to room n.

Output

Print one integer: the maximum score you can get. However, if you can get an arbitrarily large score, print -1.

Constraints

- $1 \le n \le 2500$
- $1 \le m \le 5000$
- $1 \le a, b \le n$
- $-10^9 \le x \le 10^9$

Example

Input:

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

Output:

```
5
```

Solution

```
struct edge {
    int u, v, w;
};

// Bellman-Ford Algorithm (Maximum Score)
void Ramez() {
    int n, m; cin >> n >> m;

    vector<edge> edges;

    for (int i = 0; i < m; i++){
        int u, v, w; cin >> u >> v >> w;
        edges.push_back({u, v, w});
    }

    vi score(n + 1, LLONG_MIN);
    score[1] = 0;

    // Relaxation (n - 1) times
    for (int i = 1; i <= n - 1; i++){
        for (int j = 0; j < m; j++){
            auto [u, v, w] = edges[j];
            if(score[u] != LLONG_MIN){
                score[v] = max(score[v], score[u] + w);
            }
        }
    }


    /*
    After the initial relaxation steps, we check if any edge can still be relaxed.
    If it can, that means there's a cycle (specifically a "positive cycle" for maximizing the score)
    that can improve the score.
    */
    vector<bool> hasPositiveCycle(n + 1, false);
    for (int i = 0; i < m; i++){
        auto [u, v, w] = edges[i];
        if(score[u] != LLONG_MIN && score[v] < score[u] + w){
            hasPositiveCycle[v] = true;
        }
    }


    /*
    However, simply detecting an edge that can be relaxed doesn't tell us which vertices
```

```
    might be affected downstream by this cycle.
    The propagation loop iterates over all edges several times (in this case, n times)
    to "spread" the effect of the positive cycle
    */
    for (int i = 1; i <= n; i++) {
        for (int j = 0; j < m; j++) {
            auto [u, v, w] = edges[j];
            if(hasPositiveCycle[u]) hasPositiveCycle[v] = true;
        }
    }

    if(hasPositiveCycle[n]) cout << -1 << "\n";
    else cout << score[n] << "\n";
}
}
```

Investigation (dijkstra, minimum cost, number of such routes, max/min length of such routes)

Time limit: 1.00 s **Memory limit:** 512 MB

You are going to travel from Syrjälä to Lehmälä by plane. You would like to find answers to the following questions: - what is the minimum price of such a route? - how many minimum-price routes are there? (modulo $10^9 + 7$) - what is the minimum number of flights in a minimum-price route? - what is the maximum number of flights in a minimum-price route?

Input

The first input line contains two integers n and m: the number of cities and the number of flights. The cities are numbered $1, 2, \ldots, n$. City 1 is Syrjälä, and city n is Lehmälä.

After this, there are m lines describing the flights. Each line has three integers a, b, and c: there is a flight from city a to city b with price c. All flights are one-way flights.

You may assume that there is a route from Syrjälä to Lehmälä.

Output

Print four integers according to the problem statement.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$
- $1 \le c \le 10^9$

Example

Input:

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

Output:

```
5 2 1 2
```

Solution

```
void Ramez() {
    int n, m; cin >> n >> m;
    vector<vector<pii>> adj(n + 1);

    for (int i = 0; i < m; i++) {
        int a, b, c; cin >> a >> b >> c;
        adj[a].push_back({ b, c });
    }

    vi dis(n+1, LLONG_MAX), ways(n+1, 0), minF(n+1, LLONG_MAX), maxF(n+1, 0);
    dis[1] = 0; ways[1] = 1; minF[1] = maxF[1] = 0;

    priority_queue<pii, vector<pii>, greater<pii>> pq; // {cost, node}
    pq.push({ 0, 1 });


    while (!pq.empty()) {
        auto [uCost, u] = pq.top(); pq.pop();

        if (uCost > dis[u]) continue;

        for (auto [v, vCost] : adj[u]) {
            if(uCost + vCost < dis[v]){
                dis[v] = uCost + vCost;
                ways[v] = ways[u];
                minF[v] = minF[u] + 1;
                maxF[v] = maxF[u] + 1;
                pq.push({ uCost + vCost, v });
            } else if (uCost + vCost == dis[v]){
                ways[v] = (ways[v] + ways[u]) % MOD;
                minF[v] = min(minF[v], minF[u] + 1);
                maxF[v] = max(maxF[v], maxF[u] + 1);
            }
        }
    }


    cout << dis[n] << " " << ways[n] << " " << minF[n] << " " << maxF[n];
}
```

Labyrinth (BFS, finding path from A to B)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a map of a labyrinth, and your task is to find a path from start to end. You can walk left, right, up and down.

Input

The first input line has two integers n and m: the height and width of the map.

Then there are n lines of m characters describing the labyrinth. Each character is . (floor), # (wall), A (start), or B (end). There is exactly one A and one B in the input.

Output

First print "YES", if there is a path, and "NO" otherwise.

If there is a path, print the length of the shortest such path and its description as a string consisting of characters L (left), R (right), U (up), and D (down). You can print any valid solution.

Constraints

- $1 \leq n, m \leq 1000$

Example

Input:

```
5 8
######
##.A#...#
##.##.#B#
##......#
######
```

Output:

```
YES
9
LDDRRRRRU
```

Solution

```
typedef pair<int,int> pii;
##define x first
##define y second
const int h[] = {1, -1, 0, 0}, v[] = {0, 0, 1, -1};
bool vis[1000][1000];
char c, par[1000][1000], ans[1000000];
int N, M, sx, sy, ex, ey, dist[1000][1000];
queue<pii> Q;
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0; i < N; i++){
        for(int j = 0; j < M; j++){
            scanf(" %c", &c);
            if(c == '#')    vis[i][j] = true;
            else if(c == 'A'){
                sx = i; sy = j;
            } else if(c == 'B'){
                ex = i; ey = j;
            }
        }
    }
    vis[sx][sy] = true;
    Q.push({sx, sy});
    while(!Q.empty()){
        pii P = Q.front(); Q.pop();
        for(int i = 0; i < 4; i++){
            int dx = P.x+h[i];
            int dy = P.y+v[i];
            if(0 <= dx && dx < N && 0 <= dy && dy < M && !vis[dx][dy]){
                if(i == 0)      par[dx][dy] = 'D';
                else if(i == 1) par[dx][dy] = 'U';
                else if(i == 2) par[dx][dy] = 'R';
                else if(i == 3) par[dx][dy] = 'L';
```

```
                dist[dx][dy] = dist[P.x][P.y]+1;
                vis[dx][dy] = true;
                Q.push({dx, dy});
            }
        }
    }
    if(!vis[ex][ey]){
        printf("NO\n");
        return 0;
    }
    printf("YES\n%d\n", dist[ex][ey]);
    pii P = {ex, ey};
    for(int i = dist[ex][ey]; i > 0; i--){
        ans[i] = par[P.x][P.y];
        if(ans[i] == 'D')       P = {P.x-1, P.y};
        else if(ans[i] == 'U')  P = {P.x+1, P.y};
        else if(ans[i] == 'R')  P = {P.x, P.y-1};
        else if(ans[i] == 'L')  P = {P.x, P.y+1};
    }
    for(int i = 1; i <= dist[ex][ey]; i++)
        printf("%c", ans[i]);
    printf("\n");
}
```

Longest Flight Route (DP on DAG, maximum number of cities visited on a path from city 1 to i)

Time limit: 1.00 s **Memory limit:** 512 MB

Uolevi has won a contest, and the prize is a free flight trip that can consist of one or more flights through cities. Of course, Uolevi wants to choose a trip that has as many cities as possible. Uolevi wants to fly from Syrjälä to Lehmälä so that he visits the maximum number of cities. You are given the list of possible flights, and you know that there are no directed cycles in the flight network.

Input

The first input line has two integers n and m: the number of cities and flights. The cities are numbered $1, 2, \ldots, n$. City 1 is Syrjälä, and city n is Lehmälä.

After this, there are m lines describing the flights. Each line has two integers a and b: there is a flight from city a to city b. Each flight is a one-way flight.

Output

First print the maximum number of cities on the route. After this, print the cities in the order they will be visited. You can print any valid solution.

If there are no solutions, print "IMPOSSIBLE".

Constraints

- $2 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
4
1 3 4 5
```

Solution

```
vi topologicalSort(int n, vector<vi>& adj, vi& inDeg) {
    queue<int> q;
    for (int i = 1; i <= n; i++) {
        if (inDeg[i] == 0)
            q.push(i);
    }

    vi order;

    while (!q.empty()) {
        int parent = q.front(); q.pop();
        order.push_back(parent);
        for (int child : adj[parent]) {
            if (--inDeg[child] == 0)
                q.push(child);
        }
    }

    return order;
}

void Ramez() {
    int n, m; cin >> n >> m;
    vector<vi> adj(n + 1);
    vi inDeg(n + 1);

    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        inDeg[b]++;
    }

    vi order = topologicalSort(n, adj, inDeg);

    // dp[i] = the maximum number of cities you can visit on any path from city 1 ending at city i
    vi dp(n + 1, -1), parent(n + 1, -1);
    dp[1] = 1;

    for (int u : order) {
        if (dp[u] < 0)    // not reachable from 1
            continue;

        for (int v : adj[u]) {
```

```
            if (dp[u] + 1 > dp[v]) {
                dp[v] = dp[u] + 1;
                parent[v] = u;
            }
        }
    }

    if (dp[n] < 0) {
        cout << "IMPOSSIBLE\n";
        return;
    }

    // reconstruct
    vector<int> path;
    for (int cur = n; cur != -1; cur = parent[cur])
        path.push_back(cur);

    reverse(all(path));
    cout << path.size() << '\n';
    cout << path << "\n";
}
```

Mail Delivery (Euler tour, can you visit all edges exactly once?)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to deliver mail to the inhabitants of a city. For this reason, you want to find a route whose starting and ending point are the post office, and that goes through every street exactly once.

Input

The first input line has two integers n and m: the number of crossings and streets. The crossings are numbered $1, 2, \ldots, n$, and the post office is located at crossing 1.

After that, there are m lines describing the streets. Each line has two integers a and b: there is a street between crossings a and b. All streets are two-way streets.

Every street is between two different crossings, and there is at most one street between two crossings.

Output

Print all the crossings on the route in the order you will visit them. You can print any valid solution.

If there are no solutions, print "IMPOSSIBLE".

Constraints

$2 \le n \le 10^5$ $1 \le m \le 2 \cdot 10^5$ $1 \le a, b \le n$

Example

Input:

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

```
3 6
4 5
```

Output:

```
1 2 6 3 2 4 5 3 1
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 1e5+1;
const int maxM = 2e5+1;
int N, M, deg[maxN];
bool tour_exists, used[maxM];
vector<pii> G[maxN];
vector<int> tour;
stack<int> S;
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back({b, i});
        G[b].push_back({a, i});
        deg[a]++; deg[b]++;
    }
    tour_exists = true;
    for(int i = 1; i <= N; i++)
        if(deg[i]&1)
            tour_exists = false;
    if(!tour_exists){
        printf("IMPOSSIBLE\n");
        return 0;
    }
    S.push(1);
    while(!S.empty()){
        int u = S.top();
        if(deg[u]){
            while(!G[u].empty()){
                int v = G[u].back().first;
                int id = G[u].back().second;
                G[u].pop_back();
                if(!used[id]){
                    deg[u]--; deg[v]--;
                    used[id] = true;
                    S.push(v);
                    break;
                }
            }
        } else {
            tour.push_back(u);
            S.pop();
        }
    }
    // Should be M+1 nodes in the Euler tour
    // If not, it means the graph was not connected
    if((int) tour.size() != M+1)
        printf("IMPOSSIBLE\n");
    else
```

```
        for(int i = 0; i <= M; i++)
            printf("%d%c", tour[i], (" \n")[i==M]);
}
```

Message Route (BFS, minimum nodes in a path from 1 to n)

Time limit: 1.00 s **Memory limit:** 512 MB

Syrjälä's network has n computers and m connections. Your task is to find out if Uolevi can send a message to Maija, and if it is possible, what is the minimum number of computers on such a route.

Input

The first input line has two integers n and m: the number of computers and connections. The computers are numbered $1, 2, \ldots, n$. Uolevi's computer is 1 and Maija's computer is n.

Then, there are m lines describing the connections. Each line has two integers a and b: there is a connection between those computers.

Every connection is between two different computers, and there is at most one connection between any two computers.

Output

If it is possible to send a message, first print k: the minimum number of computers on a valid route. After this, print an example of such a route. You can print any valid solution.

If there are no routes, print "IMPOSSIBLE".

Constraints

- $2 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
3
1 4 5
```

Solution

```
void solve() {
    int n, m; cin >> n >> m;
    vector<vector<int>> graph(n + 1);
    for (int i = 0; i < m; i++) {
        int u, v; cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }

    vector<bool> visited(n + 1);
```

```
    visited[1] = true;
    vector<int> parents(n + 1);
    queue<int> q;
    q.push(1);
    while (!q.empty()) {
        int s = q.front(); q.pop();
        for (auto u : graph[s]) {
            if (visited[u]) continue;
            visited[u] = true;
            q.push(u);
            parents[u] = s;
        }
    }

    if(parents[n] == 0) {
        cout << "IMPOSSIBLE" << "\n";
        return;
    }

    stack<int> path;
    path.push(n);
    while (path.top() != 1){
        path.push(parents[path.top()]);
    }

    cout << path.size() << "\n";
    while(!path.empty()) {
        cout << path.top() << " ";
        path.pop();
    }
}
```

Monsters (Simulation for monsters and one player)

Time limit: 1.00 s **Memory limit:** 512 MB

You and some monsters are in a labyrinth. When taking a step to some direction in the labyrinth, each monster may simultaneously take one as well. Your goal is to reach one of the boundary squares without ever sharing a square with a monster. Your task is to find out if your goal is possible, and if it is, print a path that you can follow. Your plan has to work in any situation; even if the monsters know your path beforehand.

Input

The first input line has two integers n and m: the height and width of the map.

After this there are n lines of m characters describing the map. Each character is . (floor), # (wall), A (start), or M (monster). There is exactly one A in the input.

Output

First print "YES" if your goal is possible, and "NO" otherwise.

If your goal is possible, also print an example of a valid path (the length of the path and its description using characters D, U, L, and R). You can print any path, as long as its length is at most $n \cdot m$ steps.

Constraints

- $1 \le n, m \le 1000$

Example

Input:

```
5 8
######
##M..A..#
##.#.M#.#
##M#..#..
##.######
```

Output:

```
YES
5
RRDDR
```

Solution

```
const int dx[] = { -1, 1, 0, 0 };
const int dy[] = { 0, 0, -1, 1 };
char dir[] = { 'U', 'D', 'L', 'R' };


void Ramez() {
    int n, m; cin >> n >> m;
    vector<string> grid(n); cin >> grid;

    queue<pair<int, int>> q;
    int x, y; // start

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (grid[i][j] == 'M')
                q.push({ i,j });
            else if (grid[i][j] == 'A')
                x = i, y = j;


    q.push({ x, y });

    vector<vi> next(1005, vi(1005, 0));
    next[x][y] = -1;

    while (!q.empty()) {
        auto it = q.front();
        x = it.first; y = it.second;
        q.pop();

        // valid
        if (grid[x][y] == 'A' && (x == 0 || x == n - 1 || y == 0 || y == m - 1)) {
            cout << "YES" << endl;
            string ans;
            int d = next[x][y];
            while (d != -1) {
                ans += dir[d];
                x -= dx[d];
                y -= dy[d];
                d = next[x][y];
```

```
            }

            reverse(ans.begin(), ans.end());
            cout << ans.size() << endl;
            cout << ans;
            return;
        }

        for (int i = 0; i < 4; i++) {
            int newX = x + dx[i], newY = y + dy[i];
            if (newX < 0 || newX >= n || newY < 0 || newY >= m || grid[newX][newY] != '.') continue;

            grid[newX][newY] = grid[x][y];
            if (grid[newX][newY] == 'A') next[newX][newY] = i;
            q.push({ newX, newY });
        }
    }

    cout << "NO";
}
```

Planets Cycles (compute, for every node, the length of its "path-to-cycle" plus the cycle size.)

Time limit: 1.00 s **Memory limit:** 512 MB

You are playing a game consisting of n planets. Each planet has a teleporter to another planet (or the planet itself). You start on a planet and then travel through teleporters until you reach a planet that you have already visited before. Your task is to calculate for each planet the number of teleportations there would be if you started on that planet.

Input

The first input line has an integer n: the number of planets. The planets are numbered $1, 2, \ldots, n$.

The second line has n integers $t_1, t_2, \ldots, t_n$: for each planet, the destination of the teleporter. It is possible that $t_i = i$.

Output

Print n integers according to the problem statement.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le t_i \le n$

Example

Input:

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

Output:

```
3 3 1 3 4
```

Solution

```
const int maxN = 2e5+1;
bool vis[maxN];
int N, t[maxN], in[maxN], ans[maxN];
vector<int> G[maxN];
queue<int> Q;
void dfs1(int u){
    for(int v : G[u]){
        if(!vis[v]){
            ans[v] = ans[u]+1;
            vis[v] = true;
            dfs1(v);
        }
    }
}
void dfs2(int u, int d = 1){
    vis[u] = true;
    int v = t[u];
    if(vis[v]){
        ans[u] = d;
    } else {
        dfs2(v, d+1);
        ans[u] = ans[v];
    }
    dfs1(u);
}
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++){
        scanf("%d", &t[i]);
        G[t[i]].push_back(i);
        in[t[i]]++;
    }
    for(int i = 1; i <= N; i++)
        if(in[i] == 0)
            Q.push(i);
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        int v = t[u];
        in[v]--;
        if(in[v] == 0)
            Q.push(v);
    }
    for(int i = 1; i <= N; i++)
        if(in[i] && !vis[i])
            dfs2(i);
    for(int i = 1; i <= N; i++)
        printf("%d%c", ans[i], (" \n")[i==N]);
}
```

Planets Queries I (finding the k-th ancestor)

Time limit: 1.00 s **Memory limit:** 512 MB

You are playing a game consisting of n planets. Each planet has a teleporter to another planet (or the planet itself). Your task is to process q queries of the form: when you begin on planet x and travel through k teleporters, which planet will you reach?

Input

The first input line has two integers n and q: the number of planets and queries. The planets are numbered $1, 2, \ldots, n$.

The second line has n integers $t_1, t_2, \ldots, t_n$: for each planet, the destination of the teleporter. It is possible that $t_i = i$.

Finally, there are q lines describing the queries. Each line has two integers x and k: you start on planet x and travel through k teleporters.

Output

Print the answer to each query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le t_i \le n$
- $1 \le x \le n$
- $0 \le k \le 10^9$

Example

Input:

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

Output:

```
1
2
4
```

Solution

```
class Tree {
private:
    int MAXPOW;
    vector<vi> up; // up[n][k] = 2^k-th ancestor of node n

public:
    Tree(int n, const vi &parents) {
        MAXPOW = ceil(log2(1e9 + 5));
        up.assign(n+1, vi(MAXPOW + 1, -1));

        for (int v = 1; v <= n; ++v) {
            up[v][0] = parents[v - 1];
        }

        for (int k = 1; k <= MAXPOW; ++k) {
            for (int v = 1; v <= n; ++v) {
                int m = up[v][k-1];
                if (m != -1) {
                    up[v][k] = up[m][k-1];
                }
            }
```

```
        }
    }

    int kth_parent(int n, int k) const {
        for (int bit = 0; bit <= MAXPOW && n != -1; ++bit) {
            if (k & (1 << bit)) {
                n = up[n][bit];
            }
        }
        return n;
    }
};


void Ramez() {
    int n, q; cin >> n >> q;
    vi parents(n); cin >> parents;

    Tree t(n, parents);

    while (q--){
        int node, k; cin >> node >> k;
        cout << t.kth_parent(node, k) << "\n";
    }
}
```

Planets Queries II (queries about the length of a path with cycles)

Time limit: 1.00 s **Memory limit:** 512 MB

You are playing a game consisting of n planets. Each planet has a teleporter to another planet (or the planet itself). You have to process q queries of the form: You are now on planet a and want to reach planet b. What is the minimum number of teleportations?

Input

The first input line contains two integers n and q: the number of planets and queries. The planets are numbered $1, 2, \ldots, n$.

The second line contains n integers $t_1, t_2, \ldots, t_n$: for each planet, the destination of the teleporter.

Finally, there are q lines describing the queries. Each line has two integers a and b: you are now on planet a and want to reach planet b.

Output

For each query, print the minimum number of teleportations. If it is not possible to reach the destination, print -1.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

```
1 3
1 4
```

Output:

```
1
2
-1
```

Solution

```
const int maxN = 2e5+1, logN = 20;
bool vis[maxN];
int N, Q, ds[maxN], d[maxN], comp[maxN], cyc[maxN], p[logN][maxN];
vector<int> cycleSeeds, G[maxN];
int find(int u){
    if(ds[u] < 0)   return u;
    ds[u] = find(ds[u]);
    return ds[u];
}
bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v)  return false;
    if(ds[u] < ds[v])   swap(u, v);
    ds[v] += ds[u];
    ds[u] = v;
    return true;
}
void init_parents(){
    for(int i = 1; i < logN; i++)
        for(int j = 1; j <= N; j++)
            p[i][j] = p[i-1][p[i-1][j]];
}
int travel(int u, int k){
    int v = u;
    for(int i = logN-1; i >= 0; i--)
        if(k&(1<<i))
            v = p[i][v];
    return v;
}
void dfs(int u, int compID){
    vis[u] = true;
    comp[u] = compID;
    for(int v : G[u]){
        if(!vis[v]){
            d[v] = d[u]+1;
            dfs(v, compID);
        } else cyc[u] = d[u]+1;
        cyc[u] = max(cyc[u], cyc[v]);
    }
}
int query(int a, int b){
    if(a == b)  return 0;
    // Start and end are on different components
    if(comp[a] != comp[b])  return -1;
    // Start is on cycle
    if(cyc[a]){
        if(!cyc[b]) return -1;
```

```
        return (d[a]-d[b]+cyc[a]) % cyc[a];
    }
    // Start and end are on tree (must check if same tree)
    if(!cyc[b]){
        if(d[a] <= d[b])    return -1;
        int dist = d[a]-d[b];
        return (travel(a, dist) == b) ? dist : -1;
    }
    // Start is on tree and end is on cycle
    int root = a;
    for(int i = logN-1; i >= 0; i--){
        int par = p[i][root];
        if(!cyc[par])
            root = par;
    }
    root = p[0][root];
    return (d[a]-d[root]) + query(root, b);
}
int main(){
    scanf("%d %d", &N, &Q);
    fill(ds+1, ds+N+1, -1);
    for(int i = 1, x; i <= N; i++){
        scanf("%d", &x);
        p[0][i] = x;
        G[x].push_back(i);
        if(!merge(x, i))
            cycleSeeds.push_back(x);
    }
    init_parents();
    int compID = 1;
    for(int seed : cycleSeeds)
        dfs(seed, compID++);
    for(int i = 0, a, b; i < Q; i++){
        scanf("%d %d", &a, &b);
        printf("%d\n", query(a, b));
    }
}
```

Planets and Kingdoms (Kosaraju's, Dividing graph into strongly connected componenets (SCC))

Time limit: 1.00 s **Memory limit:** 512 MB

A game has n planets, connected by m teleporters. Two planets a and b belong to the same kingdom exactly when there is a route both from a to b and from b to a. Your task is to determine for each planet its kingdom.

Input

The first input line has two integers n and m: the number of planets and teleporters. The planets are numbered $1, 2, \ldots, n$.

After this, there are m lines describing the teleporters. Each line has two integers a and b: you can travel from planet a to planet b through a teleporter.

Output

First print an integer k: the number of kingdoms. After this, print for each planet a kingdom label between 1 and k. You can print any valid solution.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq m \leq 2 \cdot 10^5$
- $1 \leq a, b \leq n$

Example

Input:

```
5 6
1 2
2 3
3 1
3 4
4 5
5 4
```

Output:

```
2
1 1 1 2 2
```

Solution

```
const int maxN = 1e5+1;
bool vis[maxN];
int N, M, rt[maxN];
vector<int> ord, comp, G[maxN], GR[maxN];
void dfs1(int u){
    vis[u] = true;
    for(int v : G[u])
        if(!vis[v])
            dfs1(v);
    ord.push_back(u);
}
void dfs2(int u){
    vis[u] = true;
    comp.push_back(u);
    for(int v : GR[u])
        if(!vis[v])
            dfs2(v);
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        GR[b].push_back(a);
    }
    for(int i = 1; i <= N; i++)
        if(!vis[i])
            dfs1(i);
    int K = 0;
    fill(vis+1, vis+N+1, false);
    reverse(ord.begin(), ord.end());
    for(int u : ord){
        if(!vis[u]){
            dfs2(u);
```

```
            K++;
            for(int v : comp)
                rt[v] = K;
            comp.clear();
        }
    }
    printf("%d\n", K);
    for(int i = 1; i <= N; i++)
        printf("%d%c", rt[i], (" \n")[i==N]);
}
```

Police Chase (Minimum number of edges removed to separte 1 from n)

Time limit: 1.00 s **Memory limit:** 512 MB

Kaaleppi has just robbed a bank and is now heading to the harbor. However, the police wants to stop him by closing some streets of the city. What is the minimum number of streets that should be closed so that there is no route between the bank and the harbor?

Input

The first input line has two integers n and m: the number of crossings and streets. The crossings are numbered $1, 2, \ldots, n$. The bank is located at crossing 1, and the harbor is located at crossing n.

After this, there are m lines that describing the streets. Each line has two integers a and b: there is a street between crossings a and b. All streets are two-way streets, and there is at most one street between two crossings.

Output

First print an integer k: the minimum number of streets that should be closed. After this, print k lines describing the streets. You can print any valid solution.

Constraints

- $2 \le n \le 500$
- $1 \le m \le 1000$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
2
3 4
1 4
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 501;
const int INF = 0x3f3f3f3f;
int N, M, p[maxN], cap[maxN][maxN];
```

```
bool vis[maxN];
vector<int> G[maxN];
int bfs(int s = 1, int t = N){
    fill(p+1, p+N+1, -1);
    p[s] = -2;
    queue<pii> Q;
    Q.push({s, INF});
    while(!Q.empty()){
        int u = Q.front().first;
        int f = Q.front().second;
        Q.pop();
        for(int v : G[u]){
            if(p[v] == -1 && cap[u][v]){
                p[v] = u;
                int aug = min(f, cap[u][v]);
                if(v == t)  return aug;
                Q.push({v, aug});
            }
        }
    }
    return 0;
}
void dfs(int u = 1){
    vis[u] = true;
    for(int v : G[u])
        if(!vis[v] && cap[u][v])
            dfs(v);
}
int maxflow(int s = 1, int t = N){
    int flow = 0, aug = 0;
    while(aug = bfs()){
        flow += aug;
        int u = t;
        while(u != s){
            int v = p[u];
            cap[v][u] -= aug;
            cap[u][v] += aug;
            u = v;
        }
    }
    return flow;
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
        cap[a][b]++;
        cap[b][a]++;
    }
    printf("%d\n", maxflow());
    dfs();
    for(int u = 1; u <= N; u++){
        if(!vis[u]) continue;
        for(int v : G[u])
```

```
            if(!vis[v])
                printf("%d %d\n", u, v);
    }
}
```

Road Construction (DSU, queries on number of components and largest one)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities and initially no roads between them. However, every day a new road will be constructed, and there will be a total of m roads. A component is a group of cities where there is a route between any two cities using the roads. After each day, your task is to find the number of components and the size of the largest component.

Input

The first input line has two integers n and m: the number of cities and roads. The cities are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the new roads. Each line has two integers a and b: a new road is constructed between cities a and b.

You may assume that every road will be constructed between two different cities.

Output

Print m lines: the required information after each day.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
4 2
3 3
2 3
```

Solution

```
struct DSU {
    vector<int> parent, rank, size;
    int c, maxSize;

    DSU(int n) : parent(n + 1), rank(n + 1, 0), size(n + 1, 1), c(n), maxSize(1) {
        for (int i = 1; i <= n; ++i) parent[i] = i;
    }

    int find(int i) {
        return (parent[i] == i ? i : (parent[i] = find(parent[i])));
    }
```

```
    bool same(int i, int j) {
        return find(i) == find(j);
    }

    int get_size(int i) {
        return size[find(i)];
    }

    // number of connected components
    int count() {
        return c;
    }

    int getMaxSize(){
        return maxSize;
    }

    int merge(int i, int j) {
        if ((i = find(i)) == (j = find(j))) return -1;
        else --c;

        if (rank[i] > rank[j]) swap(i, j);
        parent[i] = j;
        size[j] += size[i];
        if (rank[i] == rank[j]) rank[j]++;

        if (size[j] > maxSize) {
            maxSize = size[j];
        }

        return j;
    }
};

void Ramez() {
    int n, m; cin >> n >> m;

    DSU dsu(n);

    while (m--) {
        int u, v; cin >> u >> v;
        dsu.merge(u, v);
        cout << dsu.count() << " " << dsu.getMaxSize() << "\n";
    }
}
```

Road Reparation (KruskalMST, finding minimum spanning tree)

Time limit: 1.00 s **Memory limit:** 128 MB

There are n cities and m roads between them. Unfortunately, the condition of the roads is so poor that they cannot be used. Your task is to repair some of the roads so that there will be a decent route between any two cities. For each road, you know its reparation cost, and you should find a solution where the total cost is as small as possible.

Input

The first input line has two integers n and m: the number of cities and roads. The cities are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the roads. Each line has three integers a, b and c: there is a road between cities a and b, and its reparation cost is c. All roads are two-way roads.

Every road is between two different cities, and there is at most one road between two cities.

Output

Print one integer: the minimum total reparation cost. However, if there are no solutions, print "IMPOSSIBLE".

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq m \leq 2 \cdot 10^5$
- $1 \leq a, b \leq n$
- $1 \leq c \leq 10^9$

Example

Input:

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

Output:

```
14
```

Solution

```
struct DSU {
    vector<int> parent, size;
    int count; // of component

    DSU(int n) : parent(n + 1), size(n + 1, 1), count(n) { iota(all(parent), 0); }

    int find(int i) { return (parent[i] == i ? i : (parent[i] = find(parent[i]))); }

    bool same(int i, int j) { return find(i) == find(j); }

    int getSize(int i) { return size[find(i)]; }

    int merge(int i, int j) {
        if ((i = find(i)) == (j = find(j))) return -1;
        else --count;
        if (size[i] > size[j]) swap(i, j);
        parent[i] = j;
        size[j] += size[i];
        return j;
    }
};
```

```cpp
class Edge {
public:
    int u, v, w;
    Edge() : u(0), v(0), w(0) {}
    Edge(int u, int v, int w) : u(u), v(v), w(w) {}
    bool operator<(Edge const& other) const { return w < other.w; }
};

inline istream& operator>>(istream& is, Edge& edge) {
    return (is >> edge.u >> edge.v >> edge.w);
}

// it returens the minimum cost of forming a minimum spanning tree
// if it can't be formed it returns -1
int kruskalMST(vector<Edge>& edges, int n) {
    int cost = 0;
    DSU dsu(n + 1);
    sort(all(edges));
    for (const auto& [u, v, w] : edges) {
        if (!dsu.same(u, v)) {
            cost += w;
            dsu.merge(u, v);
        }
    }
    int root = dsu.find(1);
    int sz = dsu.getSize(root);
    if (sz == n) return cost;
    else return -1;
}


void Ramez() {
    int n, m; cin >> n >> m;
    vector<Edge> edges(m); cin >> edges;
    int answer = kruskalMST(edges, n);
    if(answer == -1) cout << "IMPOSSIBLE\n";
    else cout << answer << "\n";
}
```

Round Trip (finding and printing a cycle undirected graph)

Time limit: 1.00 s **Memory limit:** 512 MB

Byteland has n cities and m roads between them. Your task is to design a round trip that begins in a city, goes through two or more other cities, and finally returns to the starting city. Every intermediate city on the route has to be distinct.

Input

The first input line has two integers n and m: the number of cities and roads. The cities are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the roads. Each line has two integers a and b: there is a road between those cities.

Every road is between two different cities, and there is at most one road between any two cities.

Output

First print an integer k: the number of cities on the route. Then print k cities in the order they will be visited. You can print any valid solution.

If there are no solutions, print "IMPOSSIBLE".

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq m \leq 2 \cdot 10^5$
- $1 \leq a, b \leq n$

Example

Input:

```
5 6
1 3
1 2
5 3
1 5
2 4
4 5
```

Output:

```
4
3 5 1 3
```

Solution

```
const int maxN = 1e5+1;
int N, M, a, b, p[maxN], ds[maxN];
vector<int> ans, G[maxN];
void dfs(int u){
    for(int v : G[u]){
        if(v != p[u]){
            p[v] = u;
            dfs(v);
        }
    }
}
int find(int u){
    if(ds[u] < 0)   return u;
    ds[u] = find(ds[u]);
    return ds[u];
}
bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v)  return false;
    if(ds[u] < ds[v])   swap(u, v);
    ds[v] += ds[u];
    ds[u] = v;
    return true;
}
int main(){
    scanf("%d %d", &N, &M);
    fill(ds+1, ds+N+1, -1);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        if(!merge(a, b)){
            dfs(a);
            int u = b;
```

```
            while(u != 0){
                ans.push_back(u);
                u = p[u];
            }
            int K = ans.size();
            printf("%d\n", K+1);
            for(int j = 0; j < K; j++)
                printf("%d ", ans[j]);
            printf("%d\n", b);
            return 0;
        } else {
            G[a].push_back(b);
            G[b].push_back(a);
        }
    }
    printf("IMPOSSIBLE\n");
}
```

Round Trip II (find and print a cycle in a directed graph)

Time limit: 1.00 s **Memory limit:** 512 MB

Byteland has n cities and m flight connections. Your task is to design a round trip that begins in a city, goes through one or more other cities, and finally returns to the starting city. Every intermediate city on the route has to be distinct.

Input

The first input line has two integers n and m: the number of cities and flights. The cities are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the flights. Each line has two integers a and b: there is a flight connection from city a to city b. All connections are one-way flights from a city to another city.

Output

First print an integer k: the number of cities on the route. Then print k cities in the order they will be visited. You can print any valid solution.

If there are no solutions, print "IMPOSSIBLE".

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
4
2 1 3 2
```

Solution

```
// BFS
void Ramez() {
    int n, m; cin >> n >> m;
    vector<vi> adj(n + 1);
    for (int i = 0; i < m; i++) {
        int a, b; cin >> a >> b;
        adj[a].push_back(b);
    }

    // Color: 0 = unvisited, 1 = in-stack, 2 = done
    vi color(n + 1, 0), parents(n + 1, -1);
    int start = -1, end = -1;;
    bool found = false;

    function<bool(int)> dfs = [&](int u) -> bool {
        color[u] = 1;

        for (int v : adj[u]) {
            if (color[v] == 0) {
                parents[v] = u;
                if (dfs(v)) return true;
            }

            if (color[v] == 1) {
                start = v;
                end = u;
                found = true;
                return true;
            }
        }

        color[u] = 2;
        return false;
    };

    for (int i = 1; i <= n && !found; i++) {
        if (color[i] == 0) dfs(i);
    }

    if (!found) {
        cout << "IMPOSSIBLE\n";
        return;
    }

    vi cycle;

    cycle.push_back(start);
    for (int x = end; x != start; x = parents[x]) cycle.push_back(x);
    cycle.push_back(start);
    reverse(all(cycle));

    cout << cycle.size() << "\n" << cycle;
}
```

School Dance (Kuhn's, finds a maximum matching in an unweighted bipartite graph)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n boys and m girls in a school. Next week a school dance will be organized. A dance pair consists of a boy and a girl, and there are k potential pairs. Your task is to find out the maximum number of dance pairs and show how this number can be achieved.

Input

The first input line has three integers n, m and k: the number of boys, girls, and potential pairs. The boys are numbered $1, 2, \ldots, n$, and the girls are numbered $1, 2, \ldots, m$.

After this, there are k lines describing the potential pairs. Each line has two integers a and b: boy a and girl b are willing to dance together.

Output

First print one integer r: the maximum number of dance pairs. After this, print r lines describing the pairs. You can print any valid solution.

Constraints

- $1 \leq n, m \leq 500$
- $1 \leq k \leq 1000$
- $1 \leq a \leq n$
- $1 \leq b \leq m$

Example

Input:

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

Output:

```
2
1 2
3 1
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 505;
bool used[maxN];
int N, M, K, cnt, mt[maxN];
vector<int> G[maxN];
vector<pii> pairs;
bool kuhns(int u){
    if(used[u]) return false;
    used[u] = true;
    for(int v : G[u]){
        if(!mt[v] || kuhns(mt[v])){
            mt[v] = u;
            return true;
        }
    }
```

```
    return false;
}
int main(){
    scanf("%d %d %d", &N, &M, &K);
    for(int i = 0, a, b; i < K; i++){
        scanf("%d %d", &a, &b);
        G[b].push_back(a);
    }
    for(int i = 1; i <= M; i++){
        fill(used+1, used+N+1, false);
        kuhns(i);
    }
    cnt = 0;
    for(int i = 1; i <= N; i++){
        if(mt[i]){
            pairs.push_back({i, mt[i]});
            cnt++;
        }
    }
    printf("%d\n", cnt);
    for(pii P : pairs)
        printf("%d %d\n", P.first, P.second);
}
```

Shortest Routes I (Dijkstra, shortest route from node 1 to every other node)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities and m flight connections between them. Your task is to determine the length of the shortest route from Syrjälä to every city.

Input

The first input line has two integers n and m: the number of cities and flight connections. The cities are numbered $1, 2, \ldots, n$, and city 1 is Syrjälä.

After that, there are m lines describing the flight connections. Each line has three integers a, b and c: a flight begins at city a, ends at city b, and its length is c. Each flight is a one-way flight.

You can assume that it is possible to travel from Syrjälä to all other cities.

Output

Print n integers: the shortest route lengths from Syrjälä to cities $1, 2, \ldots, n$.

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$
- $1 \le c \le 10^9$

Example

Input:

```
3 4
1 2 6
1 3 2
3 2 3
```

```
1 3 4
```

Output:

```
0 5 2
```

Solution

```
// Dijkstra
void Ramez() {
    int n, m; cin >> n >> m;
    vector<vector<pii>> adj(n + 1);
    for (int i = 0; i < m; i++){
        int a, b, c; cin >> a >> b >> c;
        adj[a].push_back({b, c});
    }

    vi vis(n + 1), dis(n + 1);
    priority_queue<pii, vector<pii>, greater<pii>> pq; // {cost, node}
    pq.push({0, 1});

    while(!pq.empty()){
        auto [parentCost, parent] = pq.top(); pq.pop();
        if(vis[parent]) continue;
        vis[parent] = 1; dis[parent] = parentCost;

        for(auto [child, childCost] : adj[parent]){
            if(!vis[child]){
                pq.push({parentCost + childCost, child});
            }
        }
    }

    for (int i = 1; i <= n; i++){
        cout << dis[i] << " ";
    }
}
}
```

Shortest Routes II (Floyed warshall, shortest route from any node x to any node y)

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities and m roads between them. Your task is to process q queries where you have to determine the length of the shortest route between two given cities.

Input

The first input line has three integers n, m and q: the number of cities, roads, and queries.

Then, there are m lines describing the roads. Each line has three integers a, b and c: there is a road between cities a and b whose length is c. All roads are two-way roads.

Finally, there are q lines describing the queries. Each line has two integers a and b: determine the length of the shortest route between cities a and b.

Output

Print the length of the shortest route for each query. If there is no route, print -1 instead.

Constraints

- $1 \leq n \leq 500$
- $1 \leq m \leq n^2$
- $1 \leq q \leq 10^5$
- $1 \leq a, b \leq n$
- $1 \leq c \leq 10^9$

Example

Input:

```
4 3 5
1 2 5
1 3 9
2 3 3
1 2
2 1
1 3
1 4
3 2
```

Output:

```
5
5
8
-1
3
```

Solution

```cpp
// Floyed Warshall Algorithm
void Ramez() {
    int n, m, q; cin >> n >> m >> q;

    vector<vi> dis(n + 1, vi(n + 1, LLONG_MAX));

    for (int i = 1; i <= n; i++){
        dis[i][i] = 0;
    }


    for (int i = 0; i < m; i++){
        int a, b, c; cin >> a >> b >> c;
        dis[a][b] = min(dis[a][b], c);
        dis[b][a] = min(dis[b][a], c);
    }

    for (int k = 1; k <= n; k++){
        for (int i = 1; i <= n; i++){
            for (int j = 1; j <= n; j++){
                if (dis[i][k] < LLONG_MAX && dis[k][j] < LLONG_MAX)
                    dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
            }
        }
    }

    while(q--){
```

```
        int a, b; cin >> a >> b;
        if(dis[a][b] == LLONG_MAX) cout << "-1\n";
        else cout << dis[a][b] << "\n";
    }
}
```

Teleporters Path (can you move from node 1 to n passing through all edges once?)

Time limit: 1.00 s **Memory limit:** 512 MB

A game has n levels and m teleportes between them. You win the game if you move from level 1 to level n using every teleporter exactly once. Can you win the game, and what is a possible way to do it?

Input

The first input line has two integers n and m: the number of levels and teleporters. The levels are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the teleporters. Each line has two integers a and b: there is a teleporter from level a to level b.

You can assume that each pair (a, b) in the input is distinct.

Output

Print $m + 1$ integers: the sequence in which you visit the levels during the game. You can print any valid solution.

If there are no solutions, print "IMPOSSIBLE".

Constraints

- $2 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

```
5 6
1 2
1 3
2 4
2 5
3 1
4 2
```

Output:

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

Solution

```
const int maxN = 1e5+1;
bool vis[maxN];
int N, M, in[maxN];
vector<int> tour, G[maxN];
stack<int> S;
void dfs(int u = 1, int p = -1){
    vis[u] = true;
    for(int v : G[u])
        if(v != p && !vis[v])
```

```
            dfs(v, u);
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        in[b]++;
    }
    dfs();
    bool exists = vis[N];
    exists &= (in[1]+1 == (int) G[1].size());
    exists &= (in[N]-1 == (int) G[N].size());
    for(int i = 2; i < N; i++){
        exists &= (in[i] == (int) G[i].size());
        if(!vis[i]) exists &= (in[i] == 0 && G[i].size() == 0);
    }
    if(!exists){
        printf("IMPOSSIBLE\n");
        return 0;
    }
    S.push(1);
    while(!S.empty()){
        int u = S.top();
        if(!G[u].empty()){
            int v = G[u].back();
            G[u].pop_back();
            S.push(v);
        } else {
            tour.push_back(u);
            S.pop();
        }
    }
    reverse(tour.begin(), tour.end());
    for(int i = 0; i < M+1; i++)
        printf("%d%c", tour[i], (" \n")[i==M]);
}
```

Interactive Problems

Hidden Integer (binary search)

Time limit: 1.00 s **Memory limit:** 512 MB

There is a hidden integer x. Your task is to find the value of x. To do this, you can ask questions: you can choose an integer y and you will be told if $y < x$.

Interaction

This is an interactive problem. Your code will interact with the grader using standard input and output. You can start asking questions right away.

On your turn, you can print one of the following:

- "? y", where $1 \le y \le 10^9$: ask if $y < x$. The grader will return YES if $y < x$ and NO otherwise.
- "! x": report that the hidden integer is x. Your program must terminate after this.

Each line should be followed by a line break. You must make sure the output gets flushed after printing each line.

Constraints

- $1 \leq x \leq 10^9$
- you can ask at most 30 questions of type ?

Example

```
? 3
YES
? 6
YES
? 7
NO
! 7
```

Solution

```
const int maxX = 1e9;
int main(){
    string result;
    int l = 0, r = maxX + 1;
    while(l + 1 < r){
        int m = l + (r-l) / 2;
        cout << "? " << m << endl;
        cin >> result;
        if(result == "YES") l = m;
        else                r = m;
    }
    cout << "! " << l+1 << endl;
}
```

Hidden Permutation

Time limit: 1.00 s **Memory limit:** 512 MB

There is a hidden permutation $a_1, a_2, \ldots, a_n$ of integers $1, 2, \ldots, n$. Your task is to find this permutation. To do this, you can ask questions: you can choose two indices i and j and you will be told if $a_i < a_j$.

Interaction

This is an interactive problem. Your code will interact with the grader using standard input and output. You should start by reading a single integer n: the length of the permutation.

On your turn, you can print one of the following:

- "? i j", where $1 \leq i, j \leq n$: ask if $a_i < a_j$. The grader will return YES if $a_i < a_j$ and NO otherwise.
- "! a_1 $a_2 \ldots a_n$": report that the hidden permutation is $a_1, a_2, \ldots, a_n$. Your program must terminate after this.

Each line should be followed by a line break. You must make sure the output gets flushed after printing each line.

Constraints

- $1 \leq n \leq 1000$
- you can ask at most 10^4 questions of type ?

Example

```
3
? 3 2
NO
? 3 1
```

```
YES
! 3 1 2
```

Explanation: The hidden permutation is $[3, 1, 2]$. The first question asks if $a_3 < a_2$ which is false, so the answer is NO. The second question asks if $a_3 < a_1$ which is true, so the answer is YES.

Solution

```
const int maxN = 1001;
int N, ans[maxN];
string result;
vector<int> inv;
int binsearch(int i, int l, int r){
    if(l == r)  return l-1;
    int m = l + (r-l) / 2;
    cout << "? " << i << " " << inv[m-1] << endl;
    cin >> result;
    if(result == "YES") return binsearch(i, l, m);
    else                return binsearch(i, m+1, r);
}
int main(){
    cin >> N;
    if(N == 1){
        cout << "! 1" << endl;
        return 0;
    }
    inv.push_back(1);
    for(int i = 2; i <= N; i++){
        int pos = binsearch(i, 1, i);
        inv.emplace(inv.begin() + pos, i);
    }
    for(int i = 1; i <= N; i++)
        ans[inv[i-1]] = i;
    cout << "! ";
    for(int i = 1; i < N; i++)
        cout << ans[i] << " ";
    cout << ans[N] << endl;
}
```

Permuted Binary Strings

Time limit: 1.00 s **Memory limit:** 512 MB

There is a hidden permutation $a_1, a_2, \ldots, a_n$ of integers $1, 2, \ldots, n$. Your task is to find this permutation. To do this, you can ask questions: you can choose a binary string $b_1 b_2 \ldots b_n$ and you will receive the binary string $b_{a_1} b_{a_2} \ldots b_{a_n}$.

Interaction

This is an interactive problem. Your code will interact with the grader using standard input and output. You should start by reading a single integer n: the length of the permutation.

On your turn, you can print one of the following:

- "? $b_1 b_2 \ldots b_n$", where $b_i \in \{0, 1\}$: The grader will return the binary string $b_{a_1} b_{a_2} \ldots b_{a_n}$.
- "! a_1 $a_2 \ldots a_n$": report that the hidden permutation is $a_1, a_2, \ldots, a_n$. Your program must terminate after this.

Each line should be followed by a line break. You must make sure the output gets flushed after printing each line.

Constraints

- $1 \le n \le 1000$
- you can ask at most 10 questions of type ?

Example

```
3
? 100
100
? 010
001
? 001
010
! 1 3 2
```

Explanation: The hidden permutation is $[1, 3, 2]$. In the first question $b_1 b_2 b_3 = 100$ and the grader returns $b_{a_1} b_{a_2} b_{a_3} = b_1 b_3 b_2 = 100$. In the second question $b_1 b_2 b_3 = 010$ and the grader returns $b_1 b_3 b_2 = 001$.

Solution

```
const int maxSize = 1024;
int N, ans[maxSize];
string response;
int main(){
    cin >> N;
    for(int i = 1; (1<<(i-1)) < N; i++){
        cout << "? ";
        for(int j = 0; j < N; j++)
            cout << (bool) (j & (1<<(i-1)));
        cout << endl;
        cin >> response;
        for(int j = 0; j < N; j++)
            if(response[j] == '1')
                ans[j] += (1<<(i-1));
    }
    cout << "! ";
    for(int i = 0; i < N; i++)
        cout << ans[i] + 1 << (i == N-1 ? "" : " ");
    cout << endl;
}
```

Introductory Problems

Apple Division

Time limit: 1.00 s **Memory limit:** 512 MB

There are n apples with known weights. Your task is to divide the apples into two groups so that the difference between the weights of the groups is minimal.

Input

The first input line has an integer n: the number of apples.

The next line has n integers $p_1, p_2, \ldots, p_n$: the weight of each apple.

Output

Print one integer: the minimum difference between the weights of the groups.

Constraints

- $1 \le n \le 20$
- $1 \le p_i \le 10^9$

Example

Input:

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

Output:

```
1
```

Explanation: Group 1 has weights 2, 3 and 4 (total weight 9), and group 2 has weights 1 and 7 (total weight 8).

Solution

```
const ll INF = 0x3f3f3f3f3f3f3f3f;
int N, p[20];
ll a, b, best;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)
        scanf("%d", &p[i]);
    best = INF;
    for(int i = 0; i < (1<<N); i++){
        a = 0; b = 0;
        for(int j = 0; j < N; j++){
            if(i&(1<<j))    a += p[j];
            else            b += p[j];
        }
        best = min(best, abs(a-b));
    }
    printf("%lld\n", best);
}
```

Bit Strings

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to calculate the number of bit strings of length n. For example, if $n = 3$, the correct answer is 8, because the possible bit strings are 000, 001, 010, 011, 100, 101, 110, and 111.

Input

The only input line has an integer n.

Output

Print the result modulo $10^9 + 7$.

Constraints

- $1 \le n \le 10^6$

Example

Input:

```
3
```

Output:

```
8
```

Solution

```
const ll MOD = 1e9+7;
int N;
ll fastpow2(int x){
    ll res = 1;
    ll a = 2;
    while(x > 0){
        if(x&1)
            res = (res * a) % MOD;
        a = (a * a) % MOD;
        x >>= 1;
    }
    return res;
}
int main(){
    scanf("%d", &N);
    printf("%lld\n", fastpow2(N));
}
```

Chessboard and Queens

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to place eight queens on a chessboard so that no two queens are attacking each other. As an additional challenge, each square is either free or reserved, and you can only place queens on the free squares. However, the reserved squares do not prevent queens from attacking each other. How many possible ways are there to place the queens?

Input

The input has eight lines, and each of them has eight characters. Each square is either free (.) or reserved (*).

Output

Print one integer: the number of ways you can place the queens.

Example

Input:

```
........
........
..*.....
........
........
.....**.
...*....
........
```

Output:

65

Solution

```
const int N = 8;
char c;
int ans;
bool b[N][N], input[N][N];
bool place(int x, int y){
    bool yes = true;
    for(int i = 0; i < N; i++)
        if(b[x][i] || b[i][y])
            yes = false;
    for(int i = 0; x-i >= 0 && y-i >= 0; i++)
        if(b[x-i][y-i])
            yes = false;
    for(int i = 0; x-i >= 0 && y+i < N; i++)
        if(b[x-i][y+i])
            yes = false;
    return yes;
}
bool check(){
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            if(b[i][j] && input[i][j])
                return false;
    return true;
}
void dfs(int i){
    if(i == N){
        if(check())
            ans++;
        return;
    }
    for(int j = 0; j < N; j++){
        if(place(i, j)){
            b[i][j] = true;
            dfs(i+1);
            b[i][j] = false;
        }
    }
}
int main(){
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            scanf(" %c", &c);
            input[i][j] = (c == '*');
        }
    }
    dfs(0);
    printf("%d\n", ans);
}
```

Coin Piles

Time limit: 1.00 s **Memory limit:** 512 MB

You have two coin piles containing a and b coins. On each move, you can either remove one coin from the left pile and two coins from the right pile, or two coins from the left pile and one coin from the right pile. Your task is to efficiently find out if you can empty both the piles.

Input

The first input line has an integer t: the number of tests.

After this, there are t lines, each of which has two integers a and b: the numbers of coins in the piles.

Output

For each test, print "YES" if you can empty the piles and "NO" otherwise.

Constraints

- $1 \le t \le 10^5$
- $0 \le a, b \le 10^9$

Example

Input:

```
3
2 1
2 2
3 3
```

Output:

```
YES
NO
YES
```

Solution

```
int T, a, b;
int main(){
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        scanf("%d %d", &a, &b);
        printf(((a+b)%3 != 0 || a > 2*b || b > 2*a) ? "NO\n" : "YES\n");
    }
}
```

Creating Strings

Time limit: 1.00 s **Memory limit:** 512 MB

Given a string, your task is to generate all different strings that can be created using its characters.

Input

The only input line has a string of length n. Each character is between a–z.

Output

First print an integer k: the number of strings. Then print k lines: the strings in alphabetical order.

Constraints

- $1 \le n \le 8$

Example

Input:

```
aabac
```

Output:

```
20
aaabc
aaacb
aabac
aabca
aacab
aacba
abaac
abaca
abcaa
acaab
acaba
acbaa
baaac
baaca
bacaa
bcaaa
caaab
caaba
cabaa
cbaaa
```

Solution

```
int N;
char S[9];
set<string> perms;
int main(){
    scanf("%s", S);
    N = (int) strlen(S);
    sort(S, S+N);
    perms.insert(S);
    while(next_permutation(S, S+N))
        perms.insert(string(S, S+N));
    cout << perms.size() << '\n';
    for(string perm : perms)
        cout << perm << '\n';
}
```

Digit Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Consider an infinite string that consists of all positive integers in increasing order: 12345678910111213141516171819202122232425...
Your task is to process q queries of the form: what is the digit at position k in the string?

Input

The first input line has an integer q: the number of queries.

After this, there are q lines that describe the queries. Each line has an integer k: a 1-indexed position in the string.

Output

For each query, print the corresponding digit.

Constraints

- $1 \leq q \leq 1000$
- $1 \leq k \leq 10^{18}$

Example

Input:

```
3
7
19
12
```

Output:

```
7
4
1
```

Solution

```
int Q;
ll K;
ll pow10(int x){
    ll res = 1;
    for(int i = 0; i < x; i++)
        res *= 10;
    return res;
}
int solve(ll N){
    if(N < 9)   return (int) N+1;
    int len = 1;
    while(9 * pow10(len-1) * len < N){
        N -= 9 * pow10(len-1) * len;
        len++;
    }
    string S = to_string(pow10(len-1) + N/len);
    return (int) (S[N%len] - '0');
}
int main(){
    scanf("%d", &Q);
    for(int q = 0; q < Q; q++){
        scanf("%lld", &K);
        printf("%d\n", solve(K-1));
    }
}
```

Gray Code

Time limit: 1.00 s **Memory limit:** 512 MB

A Gray code is a list of all 2^n bit strings of length n, where any two successive strings differ in exactly one bit (i.e., their Hamming distance is one). Your task is to create a Gray code for a given length n.

Input

The only input line has an integer n.

Output

Print 2^n lines that describe the Gray code. You can print any valid solution.

Constraints

- $1 \le n \le 16$

Example

Input:

```
2
```

Output:

```
00
01
11
10
```

Solution

```
const int maxN = 16;
int N;
bool b[maxN+1];
void print(){
    for(int i = N; i > 0; i--)
        printf("%d", b[i]);
    printf("\n");
}
int main(){
    scanf("%d", &N);
    print();
    for(int i = 1; i < (1<<N); i++){
        int LSB = __builtin_ffs(i);
        b[LSB] ^= 1;
        print();
    }
}
```

Grid Coloring I

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an $n \times m$ grid where each cell contains one character A, B, C or D. For each cell, you must change the character to A, B, C or D. The new character must be different from the old one. Your task is to change the characters in every cell such that no two adjacent cells have the same character.

Input

The first line has two integers n and m: the number of rows and columns.

The next n lines each have m characters: the description of the grid.

Output

Print n lines each with m characters: the description of the final grid.

You may print any valid solution.

If no solution exists, just print IMPOSSIBLE.

Constraints

- $1 \leq n, m \leq 500$

Example

Input:

```
3 4
AAAA
BBBB
CCDD
```

Output:

```
CDCD
DCDC
ABAB
```

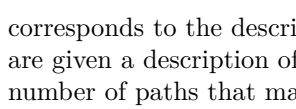
Solution

```
int N, M;
const char cs[2][2] = {{'A', 'B'}, {'C', 'D'}};
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0; i < N; i++){
        for(int j = 0; j < M; j++){
            char c;
            scanf(" %c", &c);
            bool odd = (i+j)&1;
            bool small = c < 'C';
            printf("%c", cs[small][odd]);
        }
        printf("\n");
    }
}
```

Grid Path Description

Time limit: 1.00 s **Memory limit:** 512 MB

There are 88418 paths in a 7×7 grid from the upper-left square to the lower-left square. Each path corresponds to a 48-character description consisting of characters D (down), U (up), L (left) and R (right). For example, the path



corresponds to the description DRURRRRRDDDLUULDDDLDRRURDDLLLLLURULURRUULDLLDDDD. You are given a description of a path which may also contain characters ? (any direction). Your task is to calculate the number of paths that match the description.

Input

The only input line has a 48-character string of characters ?, D, U, L and R.

Output

Print one integer: the total number of paths.

Example

Input:

```
??????R??????U???????????????????????????LD????D?
```

Output:

```
201
```

Solution

```
const int N = 7;
int ans;
char c[N*N+1];
bool vis[N+1][N+1];
bool inbounds(int x, int y){
    return 1 <= x && x <= N && 1 <= y && y <= N;
}
void dfs(int x, int y, int i){
    if(i == N*N-1 || (x == N && y == 1)){
        ans += (i == N*N-1 && (x == N && y == 1));
        return;
    }
    if((!inbounds(x+1, y) || vis[x+1][y]) && (!inbounds(x-1, y) || vis[x-1][y]))
        if(inbounds(x, y-1) && !vis[x][y-1] && inbounds(x, y+1) && !vis[x][y+1])
            return;
    if((!inbounds(x, y+1) || vis[x][y+1]) && (!inbounds(x, y-1) || vis[x][y-1]))
        if(inbounds(x+1, y) && !vis[x+1][y] && inbounds(x-1, y) && !vis[x-1][y])
            return;
    vis[x][y] = true;
    if(c[i] == 'D' || c[i] == '?')
        if(inbounds(x+1, y) && !vis[x+1][y])
            dfs(x+1, y, i+1);
    if(c[i] == 'U' || c[i] == '?')
        if(inbounds(x-1, y) && !vis[x-1][y])
            dfs(x-1, y, i+1);
    if(c[i] == 'R' || c[i] == '?')
        if(inbounds(x, y+1) && !vis[x][y+1])
            dfs(x, y+1, i+1);
    if(c[i] == 'L' || c[i] == '?')
        if(inbounds(x, y-1) && !vis[x][y-1])
            dfs(x, y-1, i+1);
    vis[x][y] = false;
}
int main(){
    scanf(" %s", c);
    dfs(1, 1, 0);
    printf("%d\n", ans);
}
```

Increasing Array

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers. You want to modify the array so that it is increasing, i.e., every element is at least as large as the previous element. On each move, you may increase the value of any element by one. What is the minimum number of moves required?

Input

The first input line contains an integer n: the size of the array.

Then, the second line contains n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print the minimum number of moves.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

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

Output:

```
5
```

Solution

```
int N;
ll x, mx, sum;
int main(){
    scanf("%d", &N);
    scanf("%lld", &mx);
    for(int i = 1; i < N; i++){
        scanf("%lld", &x);
        mx = max(mx, x);
        sum += (mx - x);
    }
    printf("%lld\n", sum);
}
```

Knight Moves Grid

Time limit: 1.00 s **Memory limit:** 512 MB

There is a knight on an $n \times n$ chessboard. For each square, print the minimum number of moves the knight needs to do to reach the top-left corner.

Input

The only line has an integer n.

Output

Print the number of moves for each square.

Constraints

- $4 \leq n \leq 1000$

Example

Input:

```
8
```

Output:

```
0 3 2 3 2 3 4 5
3 4 1 2 3 4 3 4
2 1 4 3 2 3 4 5
3 2 3 2 3 4 3 4
2 3 2 3 4 3 4 5
3 4 3 4 3 4 5 4
4 3 4 3 4 5 4 5
5 4 5 4 5 4 5 6
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 1e3+5;
const int M = 8;
const int h[M] = {2, 2, 1, -1, -2, -2, -1, 1};
const int v[M] = {-1, 1, 2, 2, 1, -1, -2, -2};
int N, dist[maxN][maxN];
bool vis[maxN][maxN];
queue<pii> Q;
bool in_bounds(int x, int y){
    return 0 <= x && x < N && 0 <= y && y < N;
}
int main(){
    scanf("%d", &N);
    Q.push({0, 0});
    vis[0][0] = true;
    while(!Q.empty()){
        pii P = Q.front();
        Q.pop();
        int x = P.first, y = P.second;
        for(int i = 0; i < M; i++){
            int dx = x + h[i], dy = y + v[i];
            if(in_bounds(dx, dy) && !vis[dx][dy]){
                dist[dx][dy] = dist[x][y] + 1;
                vis[dx][dy] = true;
                Q.push({dx, dy});
            }
        }
    }
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            printf("%d%c", dist[i][j], (" \n")[j==N-1]);
}
```

Mex Grid Construction

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to construct an $n \times n$ grid where each square has the smallest nonnegative integer that does not appear to the left on the same row or above on the same column.

Input

The only line has an integer n.

Output

Print the grid according to the example.

Constraints

- $1 \le n \le 100$

Example

Input:

```
5
```

Output:

```
0 1 2 3 4
1 0 3 2 5
2 3 0 1 6
3 2 1 0 7
4 5 6 7 0
```

Solution

```
int N;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            printf("%d%c", i ^ j, (" \n")[j==N-1]);
}
```

Missing Number

Time limit: 1.00 s **Memory limit:** 512 MB

You are given all numbers between $1, 2, \ldots, n$ except one. Your task is to find the missing number.

Input

The first input line contains an integer n.

The second line contains $n - 1$ numbers. Each number is distinct and between 1 and n (inclusive).

Output

Print the missing number.

Constraints

- $2 \le n \le 2 \cdot 10^5$

Example

Input:

```
5
2 3 1 5
```

Output:

```
4
```

Solution

```
int N, x, xum;
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++)
        xum ^= i;
    for(int i = 0; i < N-1; i++){
        scanf("%d", &x);
        xum ^= x;
    }
    printf("%d\n", xum);
}
```

Number Spiral

Time limit: 1.00 s **Memory limit:** 512 MB

A number spiral is an infinite grid whose upper-left square has number 1. Here are the first five layers of the spiral:

Your task is to find out the number in row y and column x.

Input

The first input line contains an integer t: the number of tests.

After this, there are t lines, each containing integers y and x.

Output

For each test, print the number in row y and column x.

Constraints

- $1 \le t \le 10^5$
- $1 \le y, x \le 10^9$

Example

Input:

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

Output:

```
8
1
15
```

Solution

```
int T;
ll X, Y;
ll solve(ll x, ll y){
    ll l = max(x, y)-1;
    if(l&1){
        if(x < y)   return l*l + x;
        else        return l*l+2*l-y+2;
    } else {
        if(x < y)   return l*l+2*l-x+2;
        else        return l*l + y;
    }
}
int main(){
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        scanf("%lld %lld", &X, &Y);
        printf("%lld\n", solve(X, Y));
    }
}
```

Palindrome Reorder

Time limit: 1.00 s **Memory limit:** 512 MB

Given a string, your task is to reorder its letters in such a way that it becomes a palindrome (i.e., it reads the same forwards and backwards).

Input

The only input line has a string of length n consisting of characters A–Z.

Output

Print a palindrome consisting of the characters of the original string. You may print any valid solution. If there are no solutions, print "NO SOLUTION".

Constraints

- $1 \le n \le 10^6$

Example

Input:

```
AAAACACBA
```

Output:

```
AACABACAA
```

Solution

```
char S[1000001];
int N, odd, freq[26];
int main(){
    scanf("%s", S);
    N = (int) strlen(S);
    for(int i = 0; i < N; i++)
        freq[(int) (S[i]-'A')]++;
```

```
    odd = -1;
    for(int i = 0; i < 26; i++){
        if(freq[i]&1){
            if(odd != -1){
                printf("NO SOLUTION\n");
                return 0;
            } else {
                odd = i;
            }
        }
    }
    for(int i = 0; i < 26; i++)
        for(int j = 0; j < freq[i]/2; j++)
            printf("%c", (char) (i+'A'));
    if(odd != -1)
        printf("%c", (char) (odd+'A'));
    for(int i = 25; i >= 0; i--)
        for(int j = 0; j < freq[i]/2; j++)
            printf("%c", (char) (i+'A'));
}
```

Permutations

Time limit: 1.00 s **Memory limit:** 512 MB

A permutation of integers $1, 2, \ldots, n$ is called beautiful if there are no adjacent elements whose difference is 1. Given n, construct a beautiful permutation if such a permutation exists.

Input

The only input line contains an integer n.

Output

Print a beautiful permutation of integers $1, 2, \ldots, n$. If there are several solutions, you may print any of them. If there are no solutions, print "NO SOLUTION".

Constraints

- $1 \leq n \leq 10^6$

Example 1

Input:

```
5
```

Output:

```
4 2 5 3 1
```

Example 2

Input:

```
3
```

Output:

```
NO SOLUTION
```

Solution

```
int N;
int main(){
    scanf("%d", &N);
    if(N == 1)      printf("1 ");
    else if(N <= 3) printf("NO SOLUTION\n");
    else {
        for(int i = 2; i <= N; i += 2)
            printf("%d ", i);
        for(int i = 1; i <= N; i += 2)
            printf("%d ", i);
    }
}
```

Raab Game I

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a two player game where each player has n cards numbered $1, 2, \ldots, n$. On each turn both players place one of their cards on the table. The player who placed the higher card gets one point. If the cards are equal, neither player gets a point. The game continues until all cards have been played. You are given the number of cards n and the players' scores at the end of the game, a and b. Your task is to give an example of how the game could have played out.

Input

The first line contains one integer t: the number of tests.

Then there are t lines, each with three integers n, a and b.

Output

For each test case print YES if there is a game with the given outcome and NO otherwise.

If the answer is YES, print an example of one possible game. Print two lines representing the order in which the players place their cards. You can give any valid example.

Constraints

- $1 \le t \le 1000$
- $1 \le n \le 100$
- $0 \le a, b \le n$

Example

Input:

```
5
4 1 2
2 0 1
3 0 0
2 1 1
4 4 1
```

Output:

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

```
YES
1 2 3
1 2 3
YES
1 2
2 1
NO
```

Solution

```
int T;
int main(){
    scanf("%d", &T);
    for(int t = 0, n, a, b; t < T; t++){
        scanf("%d %d %d\n", &n, &a, &b);
        // Degenerate case: all ties
        if(a == 0 && b == 0){
            printf("YES\n");
            for(int i = 0; i < 2; i++)
                for(int j = 1; j <= n; j++)
                    printf("%d%c", j, (" \n")[j==n]);
            continue;
        }
        // Otherwise, each player must win at least once
        // and the total wins cannnot exceed n
        if(a + b > n || a == 0 || b == 0){
            printf("NO\n");
            continue;
        }
        // General case: trade wins then handle ties
        printf("YES\n");
        int ties = n - a - b;
        vector<int> xs(n+1);
        // Start with both players playing 1, 2, 3, ...
        // Cyclic shift a player by k to the right
        // in order to give them k wins
        // E.g. n = 5, shift b by k = 2
        // as = 1 2 3 4 5 (3 wins)
        // bs = 4 5 1 2 3 (2 wins)
        for(int i = 1; i <= a+b; i++){
            int pos = (i + b - 1) % (a + b) + 1;
            xs[pos] = i;
        }
        for(int i = a+b+1; i <= n; i++)
            xs[i] = i;
        // Print out solution
        for(int i = 1; i <= n; i++)
            printf("%d%c", i, (" \n")[i==n]);
        for(int i = 1; i <= n; i++)
            printf("%d%c", xs[i], (" \n")[i==n]);
    }
}
```

Repetitions

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a DNA sequence: a string consisting of characters A, C, G, and T. Your task is to find the longest repetition in the sequence. This is a maximum-length substring containing only one type of character.

Input

The only input line contains a string of n characters.

Output

Print one integer: the length of the longest repetition.

Constraints

- $1 \le n \le 10^6$

Example

Input:

```
ATTCGGGA
```

Output:

```
3
```

Solution

```
const int maxN = 1e6+5;
char S[maxN];
int N, cur, best;
int main(){
    scanf(" %s", S);
    N = (int) strlen(S);
    cur = best = 1;
    for(int i = 1; i < N; i++){
        if(S[i] == S[i-1])  cur++;
        else                cur = 1;
        best = max(best, cur);
    }
    printf("%d\n", best);
}
```

String Reorder

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to reorder the characters of a string so that no two adjacent characters are the same. What is the lexicographically minimal such string?

Input

The only line has a string of length n consisting of characters A–Z.

Output

Print the lexicographically minimal reordered string where no two adjacent characters are the same. If it is not possible to create such a string, print -1.

Constraints

- $1 \le n \le 10^6$

Example

Input:

```
HATTIVATTI
```

Output:

```
AHATITITVT
```

Solution

```
const int maxN = 1e6+5;
int freq[26];
char S[maxN], ans[maxN];
bool possible(int current){
    int mode = 0, total = 0;
    for(int c = 0; c < 26; c++){
        if(freq[mode] < freq[c])
            mode = c;
        total += freq[c];
    }
    return (freq[mode] <= (total+1)/2) && (freq[current] <= total/2);
}
int main(){
    scanf(" %s", S);
    const int N = (int) strlen(S);
    for(int i = 0; i < N; i++)
        freq[(int) (S[i] - 'A')]++;
    int lastchar = -1;
    for(int i = 0; i < N; i++){
        bool filled = false;
        for(int c = 0; c < 26 && !filled; c++){
            if(freq[c] > 0 && c != lastchar){
                freq[c]--;
                if(possible(c)){
                    ans[i] = (char) (c + 'A');
                    lastchar = c;
                    filled = true;
                } else freq[c]++;
            }
        }
        if(!filled){
            printf("-1\n");
            return 0;
        }
    }
    for(int i = 0; i < N; i++)
        printf("%c", ans[i]);
    printf("\n");
}
```

Tower of Hanoi

Time limit: 1.00 s **Memory limit:** 512 MB

The Tower of Hanoi game consists of three stacks (left, middle and right) and n round disks of different sizes. Initially, the left stack has all the disks, in increasing order of size from top to bottom. The goal is to move all the disks to the right stack using the middle stack. On each move you can move the uppermost disk from a stack to

another stack. In addition, it is not allowed to place a larger disk on a smaller disk. Your task is to find a solution that minimizes the number of moves.

Input

The only input line has an integer n: the number of disks.

Output

First print an integer k: the minimum number of moves.

After this, print k lines that describe the moves. Each line has two integers a and b: you move a disk from stack a to stack b.

Constraints

- $1 \le n \le 16$

Example

Input:

```
2
```

Output:

```
3
1 2
1 3
2 3
```

Solution

```
int N;
void move(int from, int to, int depth){
    if(depth == 1){
        printf("%d %d\n", from, to);
        return;
    }
    int other = 6 - from - to;
    move(from, other, depth-1);
    printf("%d %d\n", from, to);
    move(other, to, depth-1);
}
int main(){
    scanf("%d", &N);
    printf("%d\n", (1<<N)-1);
    move(1, 3, N);
}
```

Trailing Zeros

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to calculate the number of trailing zeros in the factorial $n!$. For example, $20! = 2432902008176640000$ and it has 4 trailing zeros.

Input

The only input line has an integer n.

Output

Print the number of trailing zeros in $n!$.

Constraints

- $1 \leq n \leq 10^9$

Example

Input:

```
20
```

Output:

```
4
```

Solution

```
int N, ans;
int main(){
    scanf("%d", &N);
    while(N > 0){
        N /= 5;
        ans += N;
    }
    printf("%d\n", ans);
}
```

Two Knights

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to count for $k = 1, 2, \ldots, n$ the number of ways two knights can be placed on a $k \times k$ chessboard so that they do not attack each other.

Input

The only input line contains an integer n.

Output

Print n integers: the results.

Constraints

- $1 \leq n \leq 10000$

Example

Input:

```
8
```

Output:

```
0
6
28
96
252
550
```

```
1056
1848
```

Solution

```
int N;
int main(){
    scanf("%d", &N);
    for(int k = 1; k <= N; k++){
        ll cnt = 1LL + (k-1)*(k-2)/2;
        cnt = cnt * (k-1) * (k+4);
        printf("%lld\n", cnt);
    }
}
```

Two Sets

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to divide the numbers $1, 2, \ldots, n$ into two sets of equal sum.

Input

The only input line contains an integer n.

Output

Print "YES", if the division is possible, and "NO" otherwise.

After this, if the division is possible, print an example of how to create the sets. First, print the number of elements in the first set followed by the elements themselves in a separate line, and then, print the second set in a similar way.

Constraints

- $1 \leq n \leq 10^6$

Example 1

Input:

```
7
```

Output:

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

Example 2

Input:

```
6
```

Output:

```
NO
```

Solution

```
int N;
int main(){
    scanf("%d", &N);
    if(N%4 == 1 || N%4 == 2)    printf("NO\n");
    else if(N%4 == 3){
        printf("YES\n");
        printf("%d\n", N/2);
        for(int i = 2; i <= N/2; i += 2)
            printf("%d %d ", i, N-i);
        printf("%d\n%d\n", N, N/2+1);
        for(int i = 1; i <= N/2; i += 2)
            printf("%d %d ", i, N-i);
    } else {
        printf("YES\n");
        printf("%d\n", N/2);
        for(int i = 2; i <= N/2; i += 2)
            printf("%d %d ", i, N-i+1);
        printf("\n%d\n", N/2);
        for(int i = 1; i <= N/2; i += 2)
            printf("%d %d ", i, N-i+1);
    }
}
```

Weird Algorithm

Time limit: 1.00 s **Memory limit:** 512 MB

Consider an algorithm that takes as input a positive integer n. If n is even, the algorithm divides it by two, and if n is odd, the algorithm multiplies it by three and adds one. The algorithm repeats this, until n is one. For example, the sequence for $n = 3$ is as follows:

$$3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

Your task is to simulate the execution of the algorithm for a given value of n.

Input

The only input line contains an integer n.

Output

Print a line that contains all values of n during the algorithm.

Constraints

- $1 \leq n \leq 10^6$

Example

Input:

```
3
```

Output:

```
3 10 5 16 8 4 2 1
```

Solution

```
long long N;
int main(){
    scanf("%lld", &N);
    while(N > 1){
        printf("%lld ", N);
        if(N&1) N = 3*N+1;
        else    N >>= 1;
    }
    printf("1\n");
}
```

Mathematics Ezz

Another Game

Time limit: 1.00 s **Memory limit:** 512 MB

There are n heaps of coins and two players who move alternately. On each move, a player selects some of the nonempty heaps and removes one coin from each heap. The player who removes the last coin wins the game. Your task is to find out who wins if both players play optimally.

Input

The first input line contains an integer t: the number of tests. After this, t test cases are described:

The first line contains an integer n: the number of heaps.

The next line has n integers $x_1, x_2, \ldots, x_n$: the number of coins in each heap.

Output

For each test case, print "first" if the first player wins the game and "second" if the second player wins the game.

Constraints

- $1 \le t \le 2 \cdot 10^5$
- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$
- the sum of all n is at most $2 \cdot 10^5$

Example

Input:

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

Output:

```
first
second
first
```

Solution

```
int T, N, x;
bool even;
int main(){
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        scanf("%d", &N);
        even = true;
        for(int i = 0; i < N; i++){
            scanf("%d", &x);
            even &= !(x&1);
        }
        printf("%s\n", even ? "second" : "first");
    }
}
```

Binomial Coefficients

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to calculate n binomial coefficients modulo $10^9 + 7$. A binomial coefficient $\binom{a}{b}$ can be calculated using the formula $\frac{a!}{b!(a-b)!}$. We assume that a and b are integers and $0 \leq b \leq a$.

Input

The first input line contains an integer n: the number of calculations.

After this, there are n lines, each of which contains two integers a and b.

Output

Print each binomial coefficient modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 10^5$
- $0 \leq b \leq a \leq 10^6$

Example

Input:

```
3
5 3
8 1
9 5
```

Output:

```
10
8
126
```

Solution

```
const int maxA = 1e6;
const ll MOD = 1e9+7;
int N, a, b;
ll fact[maxA+1], inv[maxA+1];
ll inverse(ll x){
```

```
    ll res = 1;
    ll expo = MOD-2;
    while(expo > 0){
        if(expo&1)
            res = (res * x) % MOD;
        x = (x * x) % MOD;
        expo >>= 1;
    }
    return res;
}
void init(){
    fact[0] = inv[0] = 1;
    for(int i = 1; i <= maxA; i++){
        fact[i] = i * fact[i-1] % MOD;
        inv[i] = inverse(fact[i]);
    }
}
int main(){
    init();
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d %d", &a, &b);
        printf("%lld\n", fact[a] * inv[b] % MOD * inv[a-b] % MOD);
    }
}
```

Bracket Sequences I

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to calculate the number of valid bracket sequences of length n. For example, when $n = 6$, there are 5 sequences: - ()()() - ()(()) - (())() - ((())) - (()())

Input

The only input line has an integer n.

Output

Print the number of sequences modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 10^6$

Example

Input:

```
6
```

Output:

```
5
```

Solution

```
const ll MOD = 1e9+7;
ll N, numerator, denominator;
ll fastpow(ll a, ll b){
```

```
    ll res = 1;
    while(b > 0){
        if(b&1)
            res = (res * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return res;
}
ll inverse(ll a){
    return fastpow(a, MOD-2);
}
int main(){
    scanf("%lld", &N);
    if(N&1) printf("0\n");
    else {
        numerator = 1;
        for(int i = 1; i <= N; i++)
            numerator = (numerator * i) % MOD;
        denominator = 1;
        for(int i = 1; i <= N/2; i++)
            denominator = (denominator * i) % MOD;
        denominator = (denominator * denominator) % MOD;
        denominator = (denominator * (N/2+1)) % MOD;
        printf("%lld\n", (numerator*inverse(denominator))%MOD);
    }
}
```

Bracket Sequences II

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to calculate the number of valid bracket sequences of length n when a prefix of the sequence is given.

Input

The first input line has an integer n.

The second line has a string of k characters: the prefix of the sequence.

Output

Print the number of sequences modulo $10^9 + 7$.

Constraints

- $1 \le k \le n \le 10^6$

Example

Input:

```
6
(()
```

Output:

```
2
```

Explanation: There are two possible sequences: (())() and (()()).

Solution

```
const int maxN = 1e6+5;
const ll MOD = 1e9+7;
char S[maxN];
int N, M, K;
ll tot, bad, fact[maxN], inv[maxN];
ll inverse(ll x){
    ll res = 1;
    ll b = MOD-2;
    while(b){
        if(b&1) res = (res * x) % MOD;
        x = (x * x) % MOD;
        b >>= 1;
    }
    return res;
}
ll choose(int x, int y){
    return (fact[x] * inv[y] % MOD) * inv[x-y] % MOD;
}
void init(){
    fact[0] = inv[0] = 1;
    for(int i = 1; i <= N; i++){
        fact[i] = (fact[i-1] * i) % MOD;
        inv[i] = (inv[i-1] * inverse(i)) % MOD;
    }
}
int main(){
    scanf("%d %s", &N, S);
    M = (int) strlen(S);
    init();
    int open = 0, closed = 0;
    for(int i = 0; i < M; i++){
        if(S[i] == '(')         open++;
        else if(S[i] == ')')    closed++;
        if(closed > open){
            printf("0\n");
            return 0;
        }
    }
    if(N&1 || open > N/2){
        printf("0\n");
        return 0;
    }
    tot = choose(N-open-closed, N/2-open);
    bad = choose(N-open-closed, N/2-open-1);
    printf("%lld\n", ((tot-bad)%MOD+MOD)%MOD);
}
```

Candy Lottery

Time limit: 1.00 s **Memory limit:** 512 MB

There are n children, and each of them independently gets a random integer number of candies between 1 and k. What is the expected maximum number of candies a child gets?

Input

The only input line contains two integers n and k.

Output

Print the expected number rounded to six decimal places (rounding half to even).

Constraints

- $1 \leq n \leq 100$
- $1 \leq k \leq 100$

Example

Input:

```
2 3
```

Output:

```
2.444444
```

Solution

```
int N, K;
double ans, a, b;
int main(){
    scanf("%d %d", &N, &K);
    for(int i = 1; i <= K; i++){
        a = b = 1.0;
        for(int j = 1; j <= N; j++){
            a *= (double) i / K;
            b *= (double) (i-1) / K;
        }
        ans += (a-b) * i;
    }
    printf("%.6f\n", ans);
}
```

Christmas Party

Time limit: 1.00 s **Memory limit:** 512 MB

There are n children at a Christmas party, and each of them has brought a gift. The idea is that everybody will get a gift brought by someone else. In how many ways can the gifts be distributed?

Input

The only input line has an integer n: the number of children.

Output

Print the number of ways modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 10^6$

Example

Input:

```
4
```

Output:

```
9
```

Solution

```
const int maxN = 1e6;
const ll MOD = 1e9+7;
int N;
ll dp[maxN+1];
void init(){
    dp[2] = 1;
    for(int i = 3; i <= maxN; i++)
        dp[i] = (i-1) * (dp[i-1] + dp[i-2]) % MOD;
}
int main(){
    init();
    scanf("%d", &N);
    printf("%lld\n", dp[N]);
}
```

Common Divisors

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n positive integers. Your task is to find two integers such that their greatest common divisor is as large as possible.

Input

The first input line has an integer n: the size of the array.

The second line has n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print the maximum greatest common divisor.

Constraints

- $2 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^6$

Example

Input:

```
5
3 14 15 7 9
```

Output:

```
7
```

Solution

```
const int maxX = 1e6;
int N, a, cnt, d[maxX+1];
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d", &a);
        d[a]++;
    }
    for(int i = maxX; i >= 1; i--){
        cnt = 0;
        for(int j = i; j <= maxX; j += i)
            cnt += d[j];
        if(cnt >= 2){
            printf("%d\n", i);
            return 0;
        }
    }
}
```

Counting Coprime Pairs

Time limit: 1.00 s **Memory limit:** 512 MB

Given a list of n positive integers, your task is to count the number of pairs of integers that are coprime (i.e., their greatest common divisor is one).

Input

The first input line has an integer n: the number of elements.

The next line has n integers $x_1, x_2, \ldots, x_n$: the contents of the list.

Output

Print one integer: the answer for the task.

Constraints

- $1 \le n \le 10^5$
- $1 \le x_i \le 10^6$

Example

Input:

```
8
5 4 20 1 16 17 5 15
```

Output:

```
19
```

Solution

```
const int maxX = 1e6+1;
ll ans;
int N, dp[maxX];
bool b[maxX];
vector<int> primes;
```

```
void init(){
    fill(b+2, b+maxX, true);
    for(int i = 2; i*i < maxX; i++)
        if(b[i])
            for(int j = i*i; j < maxX; j += i)
                b[j] = false;
    for(int i = 2; i < maxX; i++)
        if(b[i])
            primes.push_back(i);
}
void compute(int x){
    vector<int> pf;
    for(int p : primes){
        if(x == 1)  break;
        else if(b[x]){
            pf.push_back(x);
            break;
        }
        if(x % p)   continue;
        pf.push_back(p);
        while(x % p == 0)
            x /= p;
    }
    int K = (int) pf.size();
    for(int mask = 0; mask < (1<<K); mask++){
        int mu = 1;
        for(int i = 0; i < K; i++)
            if(mask&(1<<i))
                mu *= pf[i];
        int k = __builtin_popcount(mask);
        ans += (k&1 ? -dp[mu] : dp[mu]);
        dp[mu]++;
    }
}
int main(){
    init();
    scanf("%d", &N);
    for(int i = 0, x; i < N; i++){
        scanf("%d", &x);
        compute(x);
    }
    printf("%lld\n", ans);
}
```

Counting Divisors

Time limit: 1.00 s **Memory limit:** 512 MB

Given n integers, your task is to report for each integer the number of its divisors. For example, if $x = 18$, the correct answer is 6 because its divisors are $1, 2, 3, 6, 9, 18$.

Input

The first input line has an integer n: the number of integers.

After this, there are n lines, each containing an integer x.

Output

For each integer, print the number of its divisors.

Constraints

- $1 \le n \le 10^5$
- $1 \le x \le 10^6$

Example

Input:

```
3
16
17
18
```

Output:

```
5
2
6
```

Solution

```
const int maxX = 1e6;
int N, x, d[maxX+1];
void init(){
    for(int i = 1; i <= maxX; i++)
        for(int j = i; j <= maxX; j += i)
            d[j]++;
}
int main(){
    scanf("%d", &N);
    init();
    for(int i = 0; i < N; i++){
        scanf("%d", &x);
        printf("%d\n", d[x]);
    }
}
```

Counting Grids

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to count the number of different $n \times n$ grids whose each square is black or white. Two grids are considered to be different if it is not possible to rotate one of them so that they look the same.

Input

The only input line has an integer n: the size of the grid.

Output

Print one integer: the number of grids modulo $10^9 + 7$.

Constraints

- $1 \le n \le 10^9$

Example

Input:

```
4
```

Output:

```
16456
```

Solution

```
const ll MOD = 1e9+7;
ll N, A, B, C;
ll fastpow(ll a, ll b){
    ll res = 1;
    while(b > 0){
        if(b&1)
            res = (res * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return res;
}
int main(){
    scanf("%lld", &N);
    A = N * N;
    B = (A+3*(N&1))/4;
    C = (A+(N&1))/2;
    printf("%lld\n", ((fastpow(2, A)+2*fastpow(2, B)+fastpow(2, C))*fastpow(4,MOD-2))%MOD);
}
```

Counting Necklaces

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to count the number of different necklaces that consist of n pearls and each pearl has m possible colors. Two necklaces are considered to be different if it is not possible to rotate one of them so that they look the same.

Input

The only input line has two numbers n and m: the number of pearls and colors.

Output

Print one integer: the number of different necklaces modulo $10^9 + 7$.

Constraints

- $1 \le n, m \le 10^6$

Example

Input:

```
4 3
```

Output:

```
24
```

Solution

```
const ll MOD = 1e9+7;
int N, M;
ll ans;
ll fastpow(ll a, ll b){
    ll res = 1;
    while(b > 0){
        if(b&1)
            res = (res * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return res;
}
int main(){
    scanf("%d %d", &N, &M);
    for(int k = 0; k < N; k++)
        ans = (ans + fastpow(M, __gcd(k, N))) % MOD;
    ans = (ans * fastpow(N, MOD-2)) % MOD;
    printf("%lld\n", ans);
}
```

Creating Strings II

Time limit: 1.00 s **Memory limit:** 512 MB

Given a string, your task is to calculate the number of different strings that can be created using its characters.

Input

The only input line has a string of length n. Each character is between a–z.

Output

Print the number of different strings modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 10^6$

Example

Input:

```
aabac
```

Output:

```
20
```

Solution

```
const int maxN = 1e6+5;
const ll MOD = 1e9+7;
int N, freq[26];
char S[maxN];
ll fact[maxN], inv[maxN];
ll inverse(ll x){
    ll res = 1;
    ll expo = MOD-2;
```

```
    while(expo > 0){
        if(expo&1)
            res = (res * x) % MOD;
        x = (x * x) % MOD;
        expo >>= 1;
    }
    return res;
}
void init(){
    fact[0] = inv[0] = 1;
    for(int i = 1; i < maxN; i++){
        fact[i] = i * fact[i-1] % MOD;
        inv[i] = inverse(fact[i]);
    }
}
int main(){
    scanf("%s", S);
    N = (int) strlen(S);
    init();
    for(int i = 0; i < N; i++)
        freq[(int) (S[i]-'a')]++;
    ll ans = fact[N];
    for(int i = 0; i < 26; i++)
        ans = ans * inv[freq[i]] % MOD;
    printf("%lld\n", ans);
}
```

Dice Probability

Time limit: 1.00 s **Memory limit:** 512 MB

You throw a dice n times, and every throw produces an outcome between 1 and 6. What is the probability that the sum of outcomes is between a and b?

Input

The only input line contains three integers n, a and b.

Output

Print the probability rounded to six decimal places (rounding half to even).

Constraints

- $1 \le n \le 100$
- $1 \le a \le b \le 6n$

Example

Input:

```
2 9 10
```

Output:

```
0.194444
```

Solution

```
const int maxN = 100;
int N, a, b;
double sum, dp[maxN+1][6*maxN+1];
int main(){
    scanf("%d %d %d", &N, &a, &b);
    dp[0][0] = 1;
    for(int i = 1; i <= N; i++)
        for(int j = 1; j <= 6*maxN; j++)
            for(int k = 1; k <= 6; k++)
                if(j-k >= 0)
                    dp[i][j] += dp[i-1][j-k]/6;
    for(int i = a; i <= b; i++)
        sum += dp[N][i];
    printf("%.6f\n", sum);
}
```

Distributing Apples

Time limit: 1.00 s **Memory limit:** 512 MB

There are n children and m apples that will be distributed to them. Your task is to count the number of ways this can be done. For example, if $n = 3$ and $m = 2$, there are 6 ways: $[0, 0, 2]$, $[0, 1, 1]$, $[0, 2, 0]$, $[1, 0, 1]$, $[1, 1, 0]$ and $[2, 0, 0]$.

Input

The only input line has two integers n and m.

Output

Print the number of ways modulo $10^9 + 7$.

Constraints

- $1 \le n, m \le 10^6$

Example

Input:

```
3 2
```

Output:

```
6
```

Solution

```
const int maxN = 2e6;
const ll MOD = 1e9+7;
int N, M;
ll fact[maxN], inv[maxN];
ll inverse(ll x){
    ll res = 1;
    ll expo = MOD-2;
    while(expo > 0){
        if(expo&1)
            res = (res * x) % MOD;
        x = (x * x) % MOD;
```

```
        expo >>= 1;
    }
    return res;
}
void init(){
    fact[0] = inv[0] = 1;
    for(int i = 1; i < maxN; i++){
        fact[i] = i * fact[i-1] % MOD;
        inv[i] = inverse(fact[i]);
    }
}
ll choose(int n, int k){
    return fact[n] * inv[k] % MOD * inv[n-k] % MOD;
}
int main(){
    init();
    scanf("%d %d", &N, &M);
    printf("%lld\n", choose(N+M-1, M));
}
```

Divisor Analysis

Time limit: 1.00 s **Memory limit:** 512 MB

Given an integer, your task is to find the number, sum and product of its divisors. As an example, let us consider the number 12: - the number of divisors is 6 (they are 1, 2, 3, 4, 6, 12) - the sum of divisors is $1+2+3+4+6+12 = 28$ - the product of divisors is $1 \cdot 2 \cdot 3 \cdot 4 \cdot 6 \cdot 12 = 1728$ Since the input number may be large, it is given as a prime factorization.

Input

The first line has an integer n: the number of parts in the prime factorization.

After this, there are n lines that describe the factorization. Each line has two numbers x and k where x is a prime and k is its power.

Output

Print three integers modulo $10^9 + 7$: the number, sum and product of the divisors.

Constraints

- $1 \le n \le 10^5$
- $2 \le x \le 10^6$
- each x is a distinct prime
- $1 \le k \le 10^9$

Example

Input:

```
2
2 2
3 1
```

Output:

```
6 28 1728
```

Solution

```
const int maxN = 1e5;
const ll MOD = 1e9+7;
int N;
ll x[maxN], k[maxN];
ll tau, sigma, pi, mu;
ll fastpow(ll a, ll b){
    ll res = 1;
    while(b > 0){
        if(b&1)
            res = (res * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return res;
}
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)
        scanf("%lld %lld", &x[i], &k[i]);
    tau = 1;
    for(int i = 0; i < N; i++)
        tau = (tau * (k[i]+1)) % MOD;
    sigma = 1;
    for(int i = 0; i < N; i++){
        ll numerator = (fastpow(x[i], k[i]+1)-1+MOD) % MOD;
        ll denominator = fastpow(x[i]-1, MOD-2);
        ll geoSum = numerator * denominator % MOD;
        sigma = (sigma * geoSum) % MOD;
    }
    pi = 1;
    mu = 1;
    for(int i = 0; i < N; i++){
        ll p = fastpow(x[i], k[i]*(k[i]+1)/2);
        mu = fastpow(mu, k[i]+1) * fastpow(p, pi) % MOD;
        pi = (pi * (k[i]+1)) % (MOD-1);
    }
    printf("%lld %lld %lld\n", tau, sigma, mu);
}
```

Exponentiation

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to efficiently calculate values a^b modulo $10^9 + 7$. Note that in this task we assume that $0^0 = 1$.

Input

The first input line contains an integer n: the number of calculations.

After this, there are n lines, each containing two integers a and b.

Output

Print each value a^b modulo $10^9 + 7$.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $0 \le a, b \le 10^9$

Example

Input:

```
3
3 4
2 8
123 123
```

Output:

```
81
256
921450052
```

Solution

```
const ll MOD = 1e9+7;
int N;
ll A, B;
ll fastpow(ll a, ll b){
    ll res = 1;
    while(b > 0){
        if(b&1)
            res = (res * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return res;
}
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%lld %lld", &A, &B);
        printf("%lld\n", fastpow(A, B));
    }
}
```

Exponentiation II

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to efficiently calculate values a^{b^c} modulo $10^9 + 7$. Note that in this task we assume that $0^0 = 1$.

Input

The first input line has an integer n: the number of calculations.

After this, there are n lines, each containing three integers a, b and c.

Output

Print each value a^{b^c} modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 10^5$
- $0 \leq a, b, c \leq 10^9$

Example

Input:

```
3
3 7 1
15 2 2
3 4 5
```

Output:

```
2187
50625
763327764
```

Solution

```
const ll MOD = 1e9+7;
int N;
ll A, B, C;
ll fastpow(ll a, ll b, ll mod){
    ll res = 1;
    while(b > 0){
        if(b&1)
            res = (res * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%lld %lld %lld", &A, &B, &C);
        printf("%lld\n", fastpow(A, fastpow(B, C, MOD-1), MOD));
    }
}
```

Fibonacci Numbers

Time limit: 1.00 s **Memory limit:** 512 MB

The Fibonacci numbers can be defined as follows: - $F_0 = 0$ - $F_1 = 1$ - $F_n = F_{n-2} + F_{n-1}$ Your task is to calculate the value of F_n for a given n.

Input

The only input line has an integer n.

Output

Print the value of F_n modulo $10^9 + 7$.

Constraints

- $0 \leq n \leq 10^{18}$

Example

Input:

```
10
```

Output:

```
55
```

Solution

```
const ll MOD = 1e9+7;
ll N, x[2][2], y[2][2];
void mult(ll A[2][2], ll B[2][2]){
    ll C[2][2];
    memset(C, 0, sizeof(C));
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 2; j++){
            for(int k = 0; k < 2; k++){
                C[i][j] += A[i][k] * B[k][j];
                C[i][j] %= MOD;
            }
        }
    }
    for(int i = 0; i < 2; i++)
        for(int j = 0; j < 2; j++)
            A[i][j] = C[i][j];
}
int main(){
    x[0][1] = x[1][0] = x[1][1] = y[0][0] = y[1][1] = 1;
    scanf("%lld", &N);
    while(N){
        if(N&1)
            mult(y, x);
        mult(x, x);
        N >>= 1;
    }
    printf("%lld\n", y[0][1]);
}
```

Graph Paths I

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a directed graph that has n nodes and m edges. Your task is to count the number of paths from node 1 to node n with exactly k edges.

Input

The first input line contains three integers n, m and k: the number of nodes and edges, and the length of the path. The nodes are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the edges. Each line contains two integers a and b: there is an edge from node a to node b.

Output

Print the number of paths modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 100$
- $1 \leq m \leq n(n-1)$
- $1 \leq k \leq 10^9$
- $1 \leq a, b \leq n$

Example

Input:

```
3 4 8
1 2
2 3
3 1
3 2
```

Output:

```
2
```

Explanation: The paths are $1 \to 2 \to 3 \to 1 \to 2 \to 3 \to 1 \to 2 \to 3$ and $1 \to 2 \to 3 \to 2 \to 3 \to 2 \to 3 \to 2 \to 3$.

Solution

```
const int maxN = 100;
const ll MOD = 1e9+7;
int N, M, K, a, b;
ll X[maxN][maxN], Y[maxN][maxN];
void mult(ll A[maxN][maxN], ll B[maxN][maxN]){
    ll C[N][N];
    memset(C, 0, sizeof(C));
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            for(int k = 0; k < N; k++){
                C[i][j] += A[i][k] * B[k][j];
                C[i][j] %= MOD;
            }
        }
    }
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            A[i][j] = C[i][j];
}
int main(){
    scanf("%d %d %d", &N, &M, &K);
    for(int i = 0; i < M; i++){
        scanf("%d %d", &a, &b);
        X[a-1][b-1]++;
    }
    for(int i = 0; i < N; i++)
        Y[i][i] = 1;
    while(K){
        if(K&1)
            mult(Y, X);
        mult(X, X);
        K >>= 1;
    }
    printf("%lld\n", Y[0][N-1]);
}
```

Graph Paths II

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a directed weighted graph having n nodes and m edges. Your task is to calculate the minimum path length from node 1 to node n with exactly k edges.

Input

The first input line contains three integers n, m and k: the number of nodes and edges, and the length of the path. The nodes are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the edges. Each line contains three integers a, b and c: there is an edge from node a to node b with weight c.

Output

Print the minimum path length. If there are no such paths, print -1.

Constraints

- $1 \le n \le 100$
- $1 \le m \le n(n-1)$
- $1 \le k \le 10^9$
- $1 \le a, b \le n$
- $1 \le c \le 10^9$

Example

Input:

```
3 4 8
1 2 5
2 3 4
3 1 1
3 2 2
```

Output:

```
27
```

Solution

```
typedef unsigned long long ull;
const int maxN = 100;
const ull INF = 1e19;
int N, M, K, a, b;
ull c, X[maxN][maxN], Y[maxN][maxN];
void combine(ull A[maxN][maxN], ull B[maxN][maxN]){
    ull C[maxN][maxN];
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            C[i][j] = INF;
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            for(int k = 0; k < N; k++)
                if(A[i][k] != INF && B[k][j] != INF)
                    C[i][j] = min(C[i][j], A[i][k] + B[k][j]);
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            A[i][j] = C[i][j];
```

```
}
int main(){
    scanf("%d %d %d", &N, &M, &K);
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++)
            X[i][j] = Y[i][j] = INF;
        Y[i][i] = 0;
    }
    for(int i = 0; i < M; i++){
        scanf("%d %d %llu", &a, &b, &c);
        X[a-1][b-1] = min(X[a-1][b-1], c);
    }
    while(K){
        if(K&1)
            combine(Y, X);
        combine(X, X);
        K >>= 1;
    }
    if(Y[0][N-1] == INF)    printf("-1\n");
    else                    printf("%llu\n", Y[0][N-1]);
}
```

Grundy's Game

Time limit: 1.00 s **Memory limit:** 512 MB

There is a heap of n coins and two players who move alternately. On each move, a player chooses a heap and divides into two nonempty heaps that have a different number of coins. The player who makes the last move wins the game. Your task is to find out who wins if both players play optimally.

Input

The first input line contains an integer t: the number of tests.

After this, there are t lines that describe the tests. Each line has an integer n: the number of coins in the initial heap.

Output

For each test case, print "first" if the first player wins the game and "second" if the second player wins the game.

Constraints

- $1 \le t \le 10^5$
- $1 \le n \le 10^6$

Example

Input:

```
3
6
7
8
```

Output:

```
first
second
first
```

Solution

```
const int maxN = 1e6+1;
/**
 * A036685
 * Retrivied from https://oeis.org/A036685
 */
int T, lose[42] = { 0, 1, 2, 4, 7, 10, 20, 23, 26, 50, 53, 270, 273, 276, 282, 285, 288, 316, 334, 337, 34
bool b[maxN];
void init(){
    for(int x : lose)
        b[x] = true;
}
int main(){
    init();
    scanf("%d", &T);
    for(int t = 0, N; t < T; t++){
        scanf("%d", &N);
        printf("%s\n", b[N] ? "second" : "first");
    }
}
```

Inversion Probability

Time limit: 1.00 s **Memory limit:** 512 MB

An array has n integers $x_1, x_2, \ldots, x_n$, and each of them has been randomly chosen between 1 and r_i. An inversion is a pair (a, b) where $a < b$ and $x_a > x_b$. What is the expected number of inversions in the array?

Input

The first input line contains an integer n: the size of the array.

The second line contains n integers $r_1, r_2, \ldots, r_n$: the range of possible values for each array position.

Output

Print the expected number of inversions rounded to six decimal places (rounding half to even).

Constraints

- $1 \le n \le 100$
- $1 \le r_i \le 100$

Example

Input:

```
3
5 2 7
```

Output:

```
1.057143
```

Solution

```
const int maxN = 101;
int N, r[maxN];
long double ans;
int f(int x){
```

```
    return x * (x-1) / 2;
}
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d", &r[i]);
        for(int j = 0; j < i; j++){
            int tot = (r[j] <= r[i]) ? f(r[j]) : (f(r[i]) + (r[j]-r[i]) * r[i]);
            ans += (long double) tot / (r[i] * r[j]);
        }
    }
    printf("%.6Lf\n", ans);
}
```

Josephus Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a game where there are n children (numbered $1, 2, \ldots, n$) in a circle. During the game, every second child is removed from the circle, until there are no children left. Your task is to process q queries of the form: "when there are n children, who is the kth child that will be removed?"

Input

The first input line has an integer q: the number of queries.

After this, there are q lines that describe the queries. Each line has two integers n and k: the number of children and the position of the child.

Output

Print q integers: the answer for each query.

Constraints

- $1 \le q \le 10^5$
- $1 \le k \le n \le 10^9$

Example

Input:

```
4
7 1
7 3
2 2
1337 1313
```

Output:

```
2
6
1
1107
```

Solution

```
typedef pair<int,int> pii;
int Q;
map<pii,int> f;
int solve(int N, int K){
```

```
    if(f[{N, K}])   return f[{N, K}];
    if(N == 1)      return f[{N, K}] = 1;
    if(2*K <= N)    return f[{N, K}] = 2*K;
    int newN = (N/2)+(N&1);
    int newK = K-N/2;
    if(N&1){
        int ans = solve(newN, newK);
        return f[{N, K}] = ((2*ans-3)+(N+1)) % (N+1);
    } else {
        return f[{N, K}] = 2*solve(newN, newK)-1;
    }
}
int main(){
    scanf("%d", &Q);
    for(int q = 0, N, K; q < Q; q++){
        scanf("%d %d", &N, &K);
        printf("%d\n", solve(N, K));
    }
}
```

Moving Robots

Time limit: 1.00 s **Memory limit:** 512 MB

Each square of an 8×8 chessboard has a robot. Each robot independently moves k steps, and there can be many robots on the same square. On each turn, a robot moves one step left, right, up or down, but not outside the board. It randomly chooses a direction among those where it can move. Your task is to calculate the expected number of empty squares after k turns.

Input

The only input line has an integer k.

Output

Print the expected number of empty squares rounded to six decimal places (rounding half to even).

Constraints

- $1 \leq k \leq 100$

Example

Input:

```
10
```

Output:

```
23.120740
```

Solution

```
const int maxK = 101;
const int N = 8, SZ = N*N;
int K;
double expected, ans[SZ], dp[maxK][SZ];
int main(){
    scanf("%d", &K);
    for(int i = 0; i < SZ; i++)
```

```
        ans[i] = 1.0;
    for(int start = 0; start < SZ; start++){
        memset(dp, 0, sizeof(dp));
        dp[0][start] = 1.0;
        for(int k = 0; k < K; k++){
            for(int u = 0; u < SZ; u++){
                vector<int> V;
                if(N <= u)          V.push_back(u-N);
                if(u < N*(N-1))     V.push_back(u+N);
                if(u % N != 0)      V.push_back(u-1);
                if(u % N != N-1)    V.push_back(u+1);
                for(int v : V)
                    dp[k+1][v] += (dp[k][u] / V.size());
            }
        }
        for(int u = 0; u < SZ; u++)
            ans[u] *= (1 - dp[K][u]);
    }
    for(int i = 0; i < SZ; i++)
        expected += ans[i];
    printf("%.6f\n", expected);
}
```

Next Prime

Time limit: 1.00 s **Memory limit:** 512 MB

Given a positive integer n, find the next prime number after it.

Input

The first line has an integer t: the number of tests.

After that, each line has a positive integer n.

Output

For each test, print the next prime after n.

Constraints

- $1 \leq t \leq 20$
- $1 \leq n \leq 10^{12}$

Example

Input:

```
5
1
2
3
42
1337
```

Output:

```
2
3
5
```

```
43
1361
```

Solution

```
const int sqrtN = 1e6+5;
bool prime[sqrtN];
vector<ll> p;
void init_primes(){
    for(int i = 2; i < sqrtN; i++)  prime[i] = true;
    for(int i = 2; i * i <= sqrtN; i++)
        if(prime[i])
            for(int j = 2*i; j < sqrtN; j += i)
                prime[j] = false;
    for(int i = 2; i < sqrtN; i++)
        if(prime[i])
            p.push_back(i);
}
bool is_prime(ll n){
    for(int i = 0; i < (int) p.size() && p[i] < n; i++)
        if(n % p[i] == 0)
            return false;
    return true;
}
int main(){
    init_primes();
    int T;
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        ll n;
        scanf("%lld", &n);
        n++;
        while(!is_prime(n))
            n++;
        printf("%lld\n", n);
    }
}
```

Nim Game I

Time limit: 1.00 s **Memory limit:** 512 MB

There are n heaps of sticks and two players who move alternately. On each move, a player chooses a non-empty heap and removes any number of sticks. The player who removes the last stick wins the game. Your task is to find out who wins if both players play optimally.

Input

The first input line contains an integer t: the number of tests. After this, t test cases are described:

The first line contains an integer n: the number of heaps.

The next line has n integers $x_1, x_2, \ldots, x_n$: the number of sticks in each heap.

Output

For each test case, print "first" if the first player wins the game and "second" if the second player wins the game.

Constraints

- $1 \le t \le 2 \cdot 10^5$
- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$
- the sum of all n is at most $2 \cdot 10^5$

Example

Input:

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

Output:

```
first
first
second
```

Solution

```
int T, N, x, xum;
int main(){
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        scanf("%d", &N);
        xum = 0;
        for(int i = 0; i < N; i++){
            scanf("%d", &x);
            xum ^= x;
        }
        printf("%s\n", (xum ? "first" : "second"));
    }
}
```

Nim Game II

Time limit: 1.00 s **Memory limit:** 512 MB

There are n heaps of sticks and two players who move alternately. On each move, a player chooses a non-empty heap and removes 1, 2, or 3 sticks. The player who removes the last stick wins the game. Your task is to find out who wins if both players play optimally.

Input

The first input line contains an integer t: the number of tests. After this, t test cases are described:

The first line contains an integer n: the number of heaps.

The next line has n integers $x_1, x_2, \ldots, x_n$: the number of sticks in each heap.

Output

For each test case, print "first" if the first player wins the game and "second" if the second player wins the game.

Constraints

- $1 \le t \le 2 \cdot 10^5$
- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$
- the sum of all n is at most $2 \cdot 10^5$

Example

Input:

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

Output:

```
first
first
second
```

Solution

```
int T, N, x, xum;
int main(){
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        scanf("%d", &N);
        xum = 0;
        for(int i = 0; i < N; i++){
            scanf("%d", &x);
            xum ^= (x%4);
        }
        printf(xum ? "first\n" : "second\n");
    }
}
```

Permutation Order

Time limit: 1.00 s **Memory limit:** 512 MB

Let $p(n, k)$ denote the kth permutation (in lexicographical order) of $1 \dots n$. For example, $p(4, 1) = [1, 2, 3, 4]$ and $p(4, 2) = [1, 2, 4, 3]$. Your task is to process two types of tests: - Given n and k, find $p(n, k)$ - Given n and $p(n, k)$, find k

Input

The first line has an integer t: the number of tests.

Each test is either "1 n k" or "2 n $p(n, k)$".

Output

For each test, print the answer according to the example.

Constraints

- $1 \leq t \leq 1000$
- $1 \leq n \leq 20$
- $1 \leq k \leq n!$

Example

Input:

```
6
1 4 1
1 4 2
2 4 1 2 3 4
2 4 1 2 4 3
1 5 42
2 5 2 4 5 3 1
```

Output:

```
1 2 3 4
1 2 4 3
1
2
2 4 5 3 1
42
```

Solution

```
const int maxN = 21;
ll fac[maxN];
void init(){
    fac[0] = 1;
    for(int i = 1; i < maxN; i++)
        fac[i] = fac[i-1] * i;
}
vector<int> query_one(const vector<int>& a, ll k){
    if(a.empty())   return {};
    int n = (int) a.size();
    int first_digit = (int) (k / fac[n-1]);
    ll new_k = k - (first_digit * fac[n-1]);
    vector<int> new_a;
    for(int i = 0; i < n; i++)
        if(i != first_digit)
            new_a.push_back(a[i]);
    vector<int> p = query_one(new_a, new_k);
    p.push_back(a[first_digit]);
    return p;
}
ll query_two(int n, const vector<int>& p){
    ll k = 1;
    for(int i = 0; i < n; i++){
        ll place_value = fac[n-i-1];
        int num_smaller = 0;
        for(int j = 0; j < i; j++)
            if(p[j] < p[i])
                num_smaller++;
        int digit_in_place = p[i] - num_smaller - 1;
        k += place_value * digit_in_place;
```

```
    }
    return k;
}
int main(){
    init();
    int T;
    scanf("%d", &T);
    for(int t = 0, type, n; t < T; t++){
        scanf("%d %d", &type, &n);
        if(type == 1){
            ll k;
            scanf("%lld", &k);
            vector<int> a;
            for(int i = 1; i <= n; i++)
                a.push_back(i);
            vector<int> p = query_one(a, k-1);
            reverse(p.begin(), p.end());
            for(int i = 0; i < n; i++)
                printf("%d%c", p[i], (" \n")[i==n-1]);
        } else {
            vector<int> p(n);
            for(int i = 0; i < n; i++)
                scanf("%d", &p[i]);
            printf("%lld\n", query_two(n, p));
        }
    }
}
```

Permutation Rounds

Time limit: 1.00 s **Memory limit:** 512 MB

There is a sorted array $[1, 2, \ldots, n]$ and a permutation $p_1, p_2, \ldots, p_n$. On each round, all elements move according to the permutation: the element at position i moves to position p_i. After how many rounds is the array sorted again for the first time?

Input

The first line has an integer n.

The next line contains n integers $p_1, p_2, \ldots, p_n$.

Output

Print the number of rounds modulo $10^9 + 7$.

Constraints

- $1 \le n \le 2 \cdot 10^5$

Example

Input:

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

Output:

```
4
```

Explanation: The array changes as follows after the rounds:

- Round 1: $[6, 3, 2, 5, 1, 4, 8, 7]$
- Round 2: $[4, 2, 3, 1, 6, 5, 7, 8]$
- Round 3: $[5, 3, 2, 6, 4, 1, 8, 7]$
- Round 4: $[1, 2, 3, 4, 5, 6, 7, 8]$

Solution

```
const int maxN = 2e5+1;
const ll MOD = (ll) 1e9+7;
int N, perm[maxN];
bool vis[maxN], is_prime[maxN];
vector<int> primes;
map<int,int> ans;
void init_primes(){
    for(int i = 2; i < maxN; i++)   is_prime[i] = true;
    for(int i = 2; i < maxN; i++){
        if(is_prime[i]){
            primes.push_back(i);
            for(int j = 2 * i; j < maxN; j += i)
                is_prime[j] = false;
        }
    }
}
int dfs(int u){
    vis[u] = true;
    if(vis[perm[u]])    return 1;
    return dfs(perm[u]) + 1;
}
void prime_factor(int x){
    for(int i = 0; i < (int) primes.size() && primes[i] <= x; i++){
        int p = primes[i];
        int num_divisions = 0;
        while(x % p == 0){
            x /= p;
            num_divisions++;
        }
        if(num_divisions > 0)
            ans[p] = max(ans[p], num_divisions);
    }
}
int main(){
    init_primes();
    scanf("%d", &N);
    for(int i = 1; i <= N; i++)
        scanf("%d", &perm[i]);
    for(int u = 1; u <= N; u++){
        if(!vis[u]){
            int len = dfs(u);
            prime_factor(len);
        }
    }
    ll prod = 1;
    for(auto const& [prime, power] : ans)
        for(int i = 0; i < power; i++)
            prod = (prod * prime) % MOD;
```

```
    printf("%lld\n", prod);
}
```

Prime Multiples

Time limit: 1.00 s **Memory limit:** 512 MB

You are given k distinct prime numbers $a_1, a_2, \ldots, a_k$ and an integer n. Your task is to calculate how many of the first n positive integers are divisible by at least one of the given prime numbers.

Input

The first input line has two integers n and k.

The second line has k prime numbers $a_1, a_2, \ldots, a_k$.

Output

Print one integer: the number integers within the interval $1, 2, \ldots, n$ that are divisible by at least one of the prime numbers.

Constraints

- $1 \leq n \leq 10^{18}$
- $1 \leq k \leq 20$
- $2 \leq a_i \leq n$

Example

Input:

```
20 2
2 5
```

Output:

```
12
```

Explanation: the 12 numbers are $2, 4, 5, 6, 8, 10, 12, 14, 15, 16, 18, 20$.

Solution

```
const double EPS = 0.001;
const int maxK = 20;
int K;
ll N, cnt, a[maxK];
int main(){
    scanf("%lld %d", &N, &K);
    for(int i = 0; i < K; i++)
        scanf("%lld", &a[i]);
    cnt = N;
    double RHS = log(N) + EPS;
    for(int mask = 0; mask < (1<<K); mask++){
        bool odd = (__builtin_popcount(mask)&1);
        ll prod = 1;
        double LHS = 0.0;
        for(int i = 0; i < K; i++){
            if(mask&(1<<i)){
                LHS += log(a[i]);
                prod *= a[i];
            }
```

```
        }
        if(LHS < RHS)
            cnt += (odd ? 1 : -1) * (N/prod);
    }
    printf("%lld\n", cnt);
}
```

Stair Game

Time limit: 1.00 s **Memory limit:** 512 MB

There is a staircase consisting of n stairs, numbered $1, 2, \ldots, n$. Initially, each stair has some number of balls. There are two players who move alternately. On each move, a player chooses a stair k where $k \neq 1$ and it has at least one ball. Then, the player moves any number of balls from stair k to stair $k-1$. The player who moves last wins the game. Your task is to find out who wins the game when both players play optimally. Note that if there are no possible moves at all, the second player wins.

Input

The first input line has an integer t: the number of tests. After this, t test cases are described:

The first line contains an integer n: the number of stairs.

The next line has n integers $p_1, p_2, \ldots, p_n$: the initial number of balls on each stair.

Output

For each test, print "first" if the first player wins the game and "second" if the second player wins the game.

Constraints

- $1 \leq t \leq 2 \cdot 10^5$
- $1 \leq n \leq 2 \cdot 10^5$
- $0 \leq p_i \leq 10^9$
- the sum of all n is at most $2 \cdot 10^5$

Example

Input:

```
3
3
0 2 1
4
1 1 1 1
2
5 3
```

Output:

```
first
second
first
```

Solution

```
int T, N, p, xum;
int main(){
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        scanf("%d", &N);
```

```
        xum = 0;
        for(int i = 0; i < N; i++){
            scanf("%d", &p);
            if(i%2)
                xum ^= p;
        }
        printf(xum ? "first\n" : "second\n");
    }
}
```

Stick Game

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a game where two players remove sticks from a heap. The players move alternately, and the player who removes the last stick wins the game. A set $P = \{p_1, p_2, \ldots, p_k\}$ determines the allowed moves. For example, if $P = \{1, 3, 4\}$, a player may remove 1, 3 or 4 sticks. Your task is find out for each number of sticks $1, 2, \ldots, n$ if the first player has a winning or losing position.

Input

The first input line has two integers n and k: the number of sticks and moves.

The next line has k integers $p_1, p_2, \ldots, p_k$ that describe the allowed moves. All integers are distinct, and one of them is 1.

Output

Print a string containing n characters: W means a winning position, and L means a losing position.

Constraints

- $1 \le n \le 10^6$
- $1 \le k \le 100$
- $1 \le p_i \le n$

Example

Input:

```
10 3
1 3 4
```

Output:

```
WLWWWWLWLW
```

Solution

```
const int maxN = 1e6;
const int maxK = 100;
int N, K, p[maxK];
bool dp[maxN+1];
int main(){
    scanf("%d %d", &N, &K);
    for(int i = 0; i < K; i++)
        scanf("%d", &p[i]);
    for(int i = 1; i <= N; i++){
        for(int j = 0; j < K; j++)
            if(i-p[j] >= 0 && !dp[i-p[j]])
                dp[i] = true;
```

```
        printf("%c", ("LW")[dp[i]]);
    }
}
```

Sum of Divisors

Time limit: 1.00 s **Memory limit:** 512 MB

Let $\sigma(n)$ denote the sum of divisors of an integer n. For example, $\sigma(12) = 1 + 2 + 3 + 4 + 6 + 12 = 28$. Your task is to calculate the sum $\sum_{i=1}^{n} \sigma(i)$ modulo $10^9 + 7$.

Input

The only input line has an integer n.

Output

Print $\sum_{i=1}^{n} \sigma(i)$ modulo $10^9 + 7$.

Constraints

- $1 \le n \le 10^{12}$

Example

Input:

```
5
```

Output:

```
21
```

Solution

```
const ll MOD = 1e9+7;
ll N, sum;
ll sigma(ll x){
    return ((x%MOD)*((x+1)%MOD)/2)%MOD;
}
int main(){
    scanf("%lld", &N);
    for(ll l = 1; l <= N; l++){
        ll r = N/(N/l);
        sum += (N/l)%MOD*(sigma(r)-sigma(l-1)+MOD)%MOD;
        sum = (sum+MOD)%MOD;
        l = r;
    }
    printf("%lld\n", sum);
}
```

Sum of Four Squares

Time limit: 1.00 s **Memory limit:** 512 MB

A well known result in number theory is that every non-negative integer can be represented as the sum of four squares of non-negative integers. You are given a non-negative integer n. Your task is to find four non-negative integers a, b, c and d such that $n = a^2 + b^2 + c^2 + d^2$.

Input

The first line has an integer t: the number of test cases.

Each of the next t lines has an integer n.

Output

For each test case, print four non-negative integers a, b, c and d that satisfy $n = a^2 + b^2 + c^2 + d^2$.

Constraints

- $1 \leq t \leq 1000$
- $0 \leq n \leq 10^7$
- the sum of all n is at most 10^7

Example

Input:

```
3
5
30
322266
```

Output:

```
2 1 0 0
1 2 3 4
314 159 265 358
```

Solution

```
typedef pair<int,int> pii;
typedef array<int,4> solution;
const int maxN = 1e7+1;
set<int> squares;
map<int,solution> small_solutions;
bool prime[maxN];
void init_squares(){
    for(int i = 0; i * i <= maxN; i++)
        squares.insert(i * i);
}
void init_primes(){
    for(int i = 2; i < maxN; i++)   prime[i] = true;
    for(int i = 2; i * i < maxN; i++)
        if(prime[i])
            for(int j = 2*i; j < maxN; j += i)
                prime[j] = false;
}
void precompute_small(){
    for(int a = 0; a <= 100; a++)
        for(int b = 0; b <= 100; b++)
            for(int c = 0; c <= 20; c++)
                for(int d = 0; d <= 5; d++)
                    small_solutions[a*a + b*b + c*c + d*d] = {a, b, c, d};
}
// Returns a and b such that
// a and b have opposite parity
// and p := n - a^2 - b^2 is prime
```

```
pii find_ab(int n){
    int sqrtn = (int) sqrt(n);
    while(true){
        int a = (rand() % sqrtn) + 1;
        int b = (rand() % sqrtn) + 1;
        if((a+b) % 2 == 0)  a++;
        int p = n - a*a - b*b;
        if(p >= 2 && prime[p])
            return {a, b};
    }
}
// If prime p is congruent to 1 (mod 4),
// then there exists integers (c, d)
// such that c^2 + d^2 = p.
// Naively find such c and d.
pii find_cd(int p){
    for(int c = 0; c*c <= p; c++){
        int d_squared = p - c*c;
        if(squares.find(d_squared) != squares.end())
            return {c, (int) sqrt(d_squared)};
    }
    return {-1, -1};
}
solution solve(int n){
    if(small_solutions.find(n) != small_solutions.end()){
        return small_solutions[n];
    } else if(n == 0){
        return {0, 0, 0, 0};
    } else if(n % 4 == 0){
        solution sol = solve(n / 4);
        for(int i = 0; i < 4; i++)  sol[i] *= 2;
        return sol;
    } else if(n % 4 == 1 || n % 4 == 3){
        solution sol = solve(2 * n);
        sort(sol.begin(), sol.end(), [](int a, int b){
            return (a % 2 > b % 2);
        });
        return {
            (sol[0] + sol[1]) / 2,
            (sol[0] - sol[1]) / 2,
            (sol[2] + sol[3]) / 2,
            (sol[2] - sol[3]) / 2,
        };
    } else {
        pii ab = find_ab(n);
        int a = ab.first, b = ab.second;
        int p = n - a*a - b*b;
        pii cd = find_cd(p);
        int c = cd.first, d = cd.second;
        return {a, b, c, d};
    }
}
int main(){
    init_squares();
    init_primes();
    precompute_small();
```

```
    int T;
    scanf("%d", &T);
    for(int t = 0, n; t < T; t++){
        scanf("%d", &n);
        solution sol = solve(n);
        for(int i = 0; i < 4; i++)  sol[i] = abs(sol[i]);
        printf("%d %d %d %d\n", sol[0], sol[1], sol[2], sol[3]);
    }
}
```

Throwing Dice

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to calculate the number of ways to get a sum n by throwing dice. Each throw yields an integer between $1 \ldots 6$. For example, if $n = 10$, some possible ways are $3+3+4$, $1+4+1+4$ and $1+1+6+1+1$.

Input

The only input line contains an integer n.

Output

Print the number of ways modulo $10^9 + 7$.

Constraints

- $1 \le n \le 10^{18}$

Example

Input:

```
8
```

Output:

```
125
```

Solution

```
const ll MOD = 1e9+7;
ll N, x[6][6], y[6][6];
void init(){
    for(int i = 0; i < 6; i++)
        x[0][i] = 1;
    for(int i = 0; i < 5; i++)
        x[i+1][i] = 1;
    for(int i = 0; i < 6; i++)
        y[i][i] = 1;
}
void mult(ll A[6][6], ll B[6][6]){
    ll C[6][6];
    memset(C, 0, sizeof(C));
    for(int i = 0; i < 6; i++){
        for(int j = 0; j < 6; j++){
            for(int k = 0; k < 6; k++){
                C[i][j] += A[i][k] * B[k][j];
                C[i][j] %= MOD;
            }
```

```
        }
    }
    for(int i = 0; i < 6; i++)
        for(int j = 0; j < 6; j++)
            A[i][j] = C[i][j];
}
int main(){
    init();
    scanf("%lld", &N);
    while(N){
        if(N&1)
            mult(y, x);
        mult(x, x);
        N >>= 1;
    }
    printf("%lld\n", y[0][0]);
}
```

Triangle Number Sums

Time limit: 1.00 s **Memory limit:** 512 MB

A triangle number is a positive integer of the form $1 + 2 + \cdots + k$. The first triangle numbers are 1, 3, 6, 10 and 15. Every positive integer can be represented as a sum of triangle numbers. For example, $42 = 21 + 21$ and $1337 = 1326 + 10 + 1$. Given a positive integer n, determine the smallest number of triangle numbers that sum to n.

Input

The first line has an integer t: the number of tests.

After that, each line has a positive integer n.

Output

For each test, print the smallest number of triangle numbers.

Constraints

- $1 \leq t \leq 100$
- $1 \leq n \leq 10^{12}$

Example

Input:

```
5
1
2
3
42
1337
```

Output:

```
1
2
1
2
3
```

Solution

```
const ll maxN = (ll) 1e12;
unordered_set<ll> S;
vector<ll> nums;
bool is_two_sum(ll n){
    int l = 0, r = (int) nums.size() - 1;
    while(l < (int) nums.size() && r >= 0){
        ll cur = nums[l] + nums[r];
        if(cur == n)        return true;
        else if(cur < n)    l++;
        else                r--;
    }
    return false;
}
int answer(ll n){
    if(S.find(n) != S.end())    return 1;
    else if(is_two_sum(n))      return 2;
    else                        return 3;
}
int main(){
    S.reserve((int) 1e6);
    for(ll i = 1, num = 1; num <= maxN; i++, num += i){
        nums.push_back(num);
        S.insert(num);
    }
    int T;
    scanf("%d", &T);
    for(int t = 0; t < T; t++){
        ll n;
        scanf("%lld", &n);
        printf("%d\n", answer(n));
    }
}
```

Range Queries

Distinct Values Queries

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers and q queries of the form: how many distinct values are there in a range $[a, b]$?

Input

The first input line has two integers n and q: the array size and number of queries.

The next line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line has two integers a and b.

Output

For each query, print the number of distinct values in the range.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

- $1 \leq a \leq b \leq n$

Example

Input:

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

Output:

```
2
3
3
```

Solution

```
typedef array<int,3> triple;
const int maxN = 2e5+1;
int N, Q, a, b, x[maxN], ptr[maxN], ds[maxN], ans[maxN];
triple queries[maxN];
map<int,int> mp;
set<int> S;
void update(int idx, int val){
    for(int i = idx; i <= N; i += -i&i)
        ds[i] += val;
}
int query(int idx){
    int sum = 0;
    for(int i = idx; i > 0; i -= -i&i)
        sum += ds[i];
    return sum;
}
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1; i <= N; i++)
        scanf("%d", &x[i]);
    for(int i = N; i > 0; i--){
        if(mp[x[i]])
            ptr[i] = mp[x[i]];
        mp[x[i]] = i;
    }
    for(int i = 1; i <= N; i++){
        if(S.count(x[i]) == 0)
            update(i, 1);
        S.insert(x[i]);
    }
    for(int i = 0; i < Q; i++){
        scanf("%d %d", &a, &b);
        queries[i] = {a, b, i};
    }
    sort(queries, queries+Q);
    int l = 1;
    for(int q = 0; q < Q; q++){
        while(l < queries[q][0]){
            if(ptr[l])
```

```
                update(ptr[l], 1);
            l++;
        }
        ans[queries[q][2]] = query(queries[q][1]) - query(queries[q][0]-1);
    }
    for(int q = 0; q < Q; q++)
        printf("%d\n", ans[q]);
}
```

Dynamic Range Minimum Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to process q queries of the following types: - update the value at position k to u - what is the minimum value in range $[a, b]$?

Input

The first input line has two integers n and q: the number of values and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line has three integers: either "1 k u" or "2 a b".

Output

Print the result of each query of type 2.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le x_i, u \le 10^9$
- $1 \le k \le n$
- $1 \le a \le b \le n$

Example

Input:

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

Output:

```
2
1
3
```

Solution

```
const int maxN = 2e5, SIZE = 4*maxN;
const int INF = 0x3f3f3f3f;
int N, Q, t, a, b, lo[SIZE], hi[SIZE], mn[SIZE];
void push(int i){
}
void pull(int i){
    mn[i] = min(mn[2*i], mn[2*i+1]);
```

```
}
void init(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r){
        scanf("%d", &mn[i]);
        return;
    }
    int m = l+(r-l)/2;
    init(2*i, l, m);
    init(2*i+1, m+1, r);
    pull(i);
}
void assign(int i, int l, int r, int v){
    if(l > hi[i] || r < lo[i])  return;
    if(l <= lo[i] && hi[i] <= r){
        mn[i] = v; return;
    }
    push(i);
    assign(2*i, l, r, v);
    assign(2*i+1, l, r, v);
    pull(i);
}
int minimum(int i, int l, int r){
    if(l > hi[i] || r < lo[i])      return INF;
    if(l <= lo[i] && hi[i] <= r)    return mn[i];
    push(i);
    int lmin = minimum(2*i, l, r);
    int rmin = minimum(2*i+1, l, r);
    pull(i);
    return min(lmin, rmin);
}
int main(){
    scanf("%d %d", &N, &Q);
    init(1, 1, N);
    for(int q = 0; q < Q; q++){
        scanf("%d %d %d", &t, &a, &b);
        if(t == 1)      assign(1, a, a, b);
        else if(t == 2) printf("%d\n", minimum(1, a, b));
    }
}
```

Dynamic Range Sum Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to process q queries of the following types: - update the value at position k to u - what is the sum of values in range $[a, b]$?

Input

The first input line has two integers n and q: the number of values and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line has three integers: either "1 k u" or "2 a b".

Output

Print the result of each query of type 2.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le x_i, u \le 10^9$
- $1 \le k \le n$
- $1 \le a \le b \le n$

Example

Input:

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

Output:

```
14
2
11
```

Solution

```
const int maxN = 2e5;
int N, Q, t, a, b;
ll x[maxN+1], ds[maxN+1];
void update(int idx, ll val){
    ll diff = val - x[idx];
    for(int i = idx; i <= N; i += -i&i)
        ds[i] += diff;
    x[idx] = val;
}
ll query(int idx){
    ll sum = 0;
    for(int i = idx; i > 0; i -= -i&i)
        sum += ds[i];
    return sum;
}
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1; i <= N; i++){
        scanf("%lld", &x[0]);
        update(i, x[0]);
    }
    for(int q = 0; q < Q; q++){
        scanf("%d %d %d", &t, &a, &b);
        if(t == 1)  update(a, b);
        else        printf("%lld\n", query(b)-query(a-1));
    }
}
```

Forest Queries

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an $n \times n$ grid representing the map of a forest. Each square is either empty or contains a tree. The upper-left square has coordinates $(1, 1)$, and the lower-right square has coordinates (n, n). Your task is to process q

queries of the form: how many trees are inside a given rectangle in the forest?

Input

The first input line has two integers n and q: the size of the forest and the number of queries.

Then, there are n lines describing the forest. Each line has n characters: . is an empty square and * is a tree.

Finally, there are q lines describing the queries. Each line has four integers y_1, x_1, y_2, x_2 corresponding to the corners of a rectangle.

Output

Print the number of trees inside each rectangle.

Constraints

- $1 \leq n \leq 1000$
- $1 \leq q \leq 2 \cdot 10^5$
- $1 \leq y_1 \leq y_2 \leq n$
- $1 \leq x_1 \leq x_2 \leq n$

Example

Input:

```
4 3
.*..
*.**
**..
****
2 2 3 4
3 1 3 1
1 1 2 2
```

Output:

```
3
1
2
```

Solution

```
const int maxN = 1001;
char c;
int N, Q, a[maxN][maxN];
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1; i <= N; i++){
        for(int j = 1; j <= N; j++){
            scanf(" %c", &c);
            a[i][j] = a[i-1][j]+a[i][j-1]-a[i-1][j-1];
            if(c == '*')    a[i][j]++;
        }
    }
    for(int q = 0; q < Q; q++){
        int x1, y1, x2, y2;
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
        printf("%d\n", a[x2][y2]-a[x2][y1-1]-a[x1-1][y2]+a[x1-1][y1-1]);
    }
}
```

Forest Queries II

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an $n \times n$ grid representing the map of a forest. Each square is either empty or has a tree. Your task is to process q queries of the following types: - Change the state (empty/tree) of a square. - How many trees are inside a rectangle in the forest?

Input

The first input line has two integers n and q: the size of the forest and the number of queries.

Then, there are n lines describing the forest. Each line has n characters: . is an empty square and * is a tree.

Finally, there are q lines describing the queries. The format of each line is either "1 y x" or "2 y_1 x_1 y_2 x_2".

Output

Print the answer to each query of the second type.

Constraints

- $1 \le n \le 1000$
- $1 \le q \le 2 \cdot 10^5$
- $1 \le y, x \le n$
- $1 \le y_1 \le y_2 \le n$
- $1 \le x_1 \le x_2 \le n$

Example

Input:

```
4 3
.*..
*.**
**..
****
2 2 2 3 4
1 3 3
2 2 2 3 4
```

Output:

```
3
4
```

Solution

```
const int maxN = 1000;
int N, Q;
char c[maxN+1][maxN+1];
ll ds[maxN+1][maxN+1];
void update(int x, int y, ll val){
    for(int i = x; i <= N; i += -i&i)
        for(int j = y; j <= N; j += -j&j)
            ds[i][j] += val;
}
ll query(int x, int y){
    ll sum = 0;
    for(int i = x; i > 0; i -= -i&i)
        for(int j = y; j > 0; j -= -j&j)
```

```
            sum += ds[i][j];
    return sum;
}
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1; i <= N; i++){
        for(int j = 1; j <= N; j++){
            scanf(" %c", &c[i][j]);
            if(c[i][j] == '*')
                update(i, j, 1);
        }
    }
    for(int q = 0, t, x1, y1, x2, y2; q < Q; q++){
        scanf("%d %d %d", &t, &x1, &y1);
        if(t == 1){
            if(c[x1][y1] == '*'){
                update(x1, y1, -1);
                c[x1][y1] = '.';
            } else if(c[x1][y1] == '.'){
                update(x1, y1, 1);
                c[x1][y1] = '*';
            }
        } else if(t == 2){
            scanf("%d %d", &x2, &y2);
            printf("%lld\n", query(x2, y2)-query(x2,y1-1)-query(x1-1,y2)+query(x1-1,y1-1));
        }
    }
}
```

Hotel Queries

Time limit: 1.00 s **Memory limit:** 512 MB

There are n hotels on a street. For each hotel you know the number of free rooms. Your task is to assign hotel rooms for groups of tourists. All members of a group want to stay in the same hotel. The groups will come to you one after another, and you know for each group the number of rooms it requires. You always assign a group to the first hotel having enough rooms. After this, the number of free rooms in the hotel decreases.

Input

The first input line contains two integers n and m: the number of hotels and the number of groups. The hotels are numbered $1, 2, \ldots, n$.

The next line contains n integers $h_1, h_2, \ldots, h_n$: the number of free rooms in each hotel.

The last line contains m integers $r_1, r_2, \ldots, r_m$: the number of rooms each group requires.

Output

Print the assigned hotel for each group. If a group cannot be assigned a hotel, print 0 instead.

Constraints

- $1 \le n, m \le 2 \cdot 10^5$
- $1 \le h_i \le 10^9$
- $1 \le r_i \le 10^9$

Example

Input:

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

Output:

```
3 5 0 1 1
```

Solution

```
const int maxN = 2e5+1;
int N, M, h[maxN], k, ans;
int lo[4*maxN], hi[4*maxN], mx[4*maxN], mp[maxN];
void pull(int i){
    mx[i] = (h[mx[2*i]] >= h[mx[2*i+1]] ? mx[2*i] : mx[2*i+1]);
}
void init(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r){
        mp[l] = i;
        mx[i] = l;
        return;
    }
    int m = l+(r-l)/2;
    init(2*i, l, m);
    init(2*i+1, m+1, r);
    pull(i);
}
void update(int i, int v){
    h[i] -= v;
    i = mp[i];
    i >>= 1;
    while(i > 0){
        pull(i);
        i >>= 1;
    }
}
int find(int i, int v){
    if(lo[i] == hi[i])  return lo[i];
    return (h[mx[2*i]] >= v ? find(2*i, v) : find(2*i+1, v));
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 1; i <= N; i++)
        scanf("%d", &h[i]);
    init(1, 1, N);
    for(int i = 0; i < M; i++){
        scanf("%d", &k);
        ans = h[mx[1]] < k ? 0 : find(1, k);
        if(ans) update(ans, k);
        printf("%d%c", ans, (" \n")[i==M-1]);
    }
}
```

Increasing Array Queries

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array that consists of n integers. The array elements are indexed $1, 2, \ldots, n$. You can modify the array using the following operation: choose an array element and increase its value by one. Your task is to process q queries of the form: when we consider a subarray from position a to position b, what is the minimum number of operations after which the subarray is increasing? An array is increasing if each element is greater than or equal with the previous element.

Input

The first input line has two integers n and q: the size of the array and the number of queries.

The next line has n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Finally, there are q lines that describe the queries. Each line has two integers a and b: the starting and ending position of a subarray.

Output

For each query, print the minimum number of operations.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$
- $1 \le a \le b \le n$

Example

Input:

```
5 3
2 10 4 2 5
3 5
2 2
1 4
```

Output:

```
2
0
14
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 2e5+1;
const int SIZE = 4*maxN;
int N, Q, lo[SIZE], hi[SIZE];
ll a[maxN], ass[SIZE], mx[SIZE], sm[SIZE], ans[maxN];
vector<pii> queries[maxN];
int len(int i){
    return hi[i]-lo[i]+1;
}
void assign(int i, ll val){
    ass[i] = mx[i] = val;
    sm[i] = val * len(i);
}
void push(int i){
    if(ass[i]){
        assign(2*i, ass[i]);
        assign(2*i+1, ass[i]);
        ass[i] = 0;
```

```
    }
}
void pull(int i){
    sm[i] = sm[2*i] + sm[2*i+1];
    mx[i] = max(mx[2*i], mx[2*i+1]);
}
void init(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r){
        sm[i] = a[l];
        return;
    }
    int m = (l+r)/2;
    init(2*i, l, m);
    init(2*i+1, m+1, r);
    pull(i);
}
void update(int i, int l, int r, ll val){
    if(l > hi[i] || r < lo[i])  return;
    if(l <= lo[i] && hi[i] <= r){
        assign(i, val);
        return;
    }
    push(i);
    update(2*i, l, r, val);
    update(2*i+1, l, r, val);
    pull(i);
}
int orderOf(int i, int l, int val){
    if(lo[i] == hi[i])                  return lo[i];
    push(i);
    int idx = -1;
    if(hi[2*i] <= l || mx[2*i] < val)   idx = orderOf(2*i+1, l, val);
    else                                idx = orderOf(2*i, l, val);
    pull(i);
    return idx;
}
ll sum(int i, int l, int r){
    if(l > hi[i] || r < lo[i])      return 0;
    if(l <= lo[i] && hi[i] <= r)    return sm[i];
    push(i);
    ll left = sum(2*i, l, r);
    ll right = sum(2*i+1, l, r);
    pull(i);
    return left+right;
}
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1; i <= N; i++)
        scanf("%lld", &a[i]);
    init(1, 1, N);
    for(int q = 0, l, r; q < Q; q++){
        scanf("%d %d", &l, &r);
        queries[l].push_back({r, q});
    }
    for(int i = 2; i <= N; i++) a[i] += a[i-1];
```

```
    for(int l = N; l >= 1; l--){
        int val = a[l]-a[l-1];
        int modifyR = (mx[1] < val ? N+1 : orderOf(1, l, val));
        update(1, l, modifyR-1, val);
        for(pii q : queries[l]){
            int r = q.first;
            int id = q.second;
            ans[id] = sum(1, l, r) - (a[r]-a[l-1]);
        }
    }
    for(int i = 0; i < Q; i++)
        printf("%lld\n", ans[i]);
}
```

List Removals

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a list consisting of n integers. Your task is to remove elements from the list at given positions, and report the removed elements.

Input

The first input line has an integer n: the initial size of the list. During the process, the elements are numbered $1, 2, \ldots, k$ where k is the current size of the list.

The second line has n integers $x_1, x_2, \ldots, x_n$: the contents of the list.

The last line has n integers $p_1, p_2, \ldots, p_n$: the positions of the elements to be removed.

Output

Print the elements in the order they are removed.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$
- $1 \le p_i \le n - i + 1$

Example

Input:

```
5
2 6 1 4 2
3 1 3 1 1
```

Output:

```
1 2 2 6 4
```

Explanation: The contents of the list are $[2, 6, 1, 4, 2]$, $[2, 6, 4, 2]$, $[6, 4, 2]$, $[6, 4]$, $[4]$ and $[]$.

Solution

```
##include <ext/pb_ds/assoc_container.hpp>
##include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
const int maxN = 2e5;
int N, p, x[maxN+1];
tree<int, null_type, less<int>, rb_tree_tag,
```

```
tree_order_statistics_node_update> T;
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++){
        scanf("%d", &x[i]);
        T.insert(i);
    }
    for(int i = 0; i < N; i++){
        scanf("%d", &p);
        printf("%d%c", x[*T.find_by_order(p-1)], (" \n")[i==N-1]);
        T.erase(T.find_by_order(p-1));
    }
}
```

Missing Coin Sum Queries

Time limit: 1.00 s **Memory limit:** 512 MB

You have n coins with positive integer values. The coins are numbered $1, 2, \ldots, n$. Your task is to process q queries of the form: "if you can use coins $a \ldots b$, what is the smallest sum you cannot produce?"

Input

The first input line has two integers n and q: the number of coins and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the value of each coin.

Finally, there are q lines that describe the queries. Each line has two values a and b: you can use coins $a \ldots b$.

Output

Print the answer for each query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$
- $1 \le a \le b \le n$

Example

Input:

```
5 3
2 9 1 2 7
2 4
4 4
1 5
```

Output:

```
4
1
6
```

Explanation: First you can use coins $[9, 1, 2]$, then coins $[2]$ and finally coins $[2, 9, 1, 2, 7]$.

Solution

```
/* Fast Input and Output Template */
char _i[1<<24], _o[1<<10], __[20], _, _c; int _i0, _o0, _n;
##define readio()    { fread(_i, 1, 1<<24, stdin); }
```

```
##define writeio()   { fwrite(_o, 1, _o0, stdout); }
##define scanu(x)    { for (x = _i[_i0++] & 15; 47 < (_ = _i[_i0++]); x = x * 10 + (_ & 15)); }
##define putnumu(x)  { _ = 0; do __[_++] = x % 10 | '0'; while (x /= 10); while (_--) _o[_o0++] = __[_]; }
##define putnl()     { _o[_o0++] = '\n'; }
/* End of Template */
const int maxN = 2e5+2;
const int logC = 32, logN = 18;
const int INF = 0x3f3f3f3f;
int N, Q, mn[logC][logN][maxN];
ll pre[logC][maxN];
int layer(int x){
    return 31 - __builtin_clz(x);
}
ll sum(int l, int a, int b){
    return pre[l][b] - pre[l][a-1];
}
int minimum(int l, int a, int b){
    a--; b--;
    int len = b-a+1;
    int k = (int) floor(log2(len));
    return min(mn[l][k][a], mn[l][k][b-(1<<k)+1]);
}
void init(){
    scanu(N); scanu(Q);
    for(int i = 1, x; i <= N; i++){
        scanu(x);
        int l = layer(x);
        pre[l][i] = x;
        for(int j = 0; j < logC; j++)
            mn[j][0][i-1] = (j == l ? x : INF);
    }
    for(int l = 0; l < logC; l++){
        for(int j = 1; j <= N; j++)
            pre[l][j] += pre[l][j-1];
        for(int i = 1; i < logN; i++)
            for(int j = 0; j <= N-(1<<i); j++)
                mn[l][i][j] = min(mn[l][i-1][j], mn[l][i-1][j+(1<<(i-1))]);
    }
}
ll query(int a, int b){
    ll ans = 1;
    for(int l = 0; l < logC && ans >= (1<<l); l++)
        if(minimum(l, a, b) <= ans)
            ans += sum(l, a, b);
    return ans;
}
int main(){
    readio();
    init();
    for(int q = 0, a, b; q < Q; q++){
        scanu(a); scanu(b);
        ll ans = query(a, b);
        putnumu(ans);
        putnl();
    }
    writeio();
```

```
}
```

Movie Festival Queries

Time limit: 1.00 s **Memory limit:** 512 MB

In a movie festival, n movies will be shown. You know the starting and ending time of each movie. Your task is to process q queries of the form: if you arrive and leave the festival at specific times, what is the maximum number of movies you can watch? You can watch two movies if the first movie ends before or exactly when the second movie starts. You can start the first movie exactly when you arrive and leave exactly when the last movie ends.

Input

The first input line has two integers n and q: the number of movies and queries.

After this, there are n lines describing the movies. Each line has two integers a and b: the starting and ending time of a movie.

Finally, there are q lines describing the queries. Each line has two integers a and b: your arrival and leaving time.

Output

Print the maximum number of movies for each query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le a < b \le 10^6$

Example

Input:

```
4 3
2 5
6 10
4 7
9 10
5 9
2 10
7 10
```

Output:

```
0
2
1
```

Solution

```
const int logN = 17, maxT = 1e6;
int N, Q, dp[maxT+1][logN+1];
vector<int> movies[maxT+1];
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 0, a, b; i < N; i++){
        scanf("%d %d", &a, &b);
        movies[a].push_back(b);
    }
    priority_queue<int, vector<int>, greater<int>> ends;
    for(int t = maxT; t >= 0; t--){
```

```
        for(int b : movies[t])
            ends.push(b);
        dp[t][0] = ends.empty() ? maxT+1 : ends.top();
    }
    for(int k = 1; k <= logN; k++)
        for(int t = 0; t <= maxT; t++)
            dp[t][k] = dp[t][k-1] > maxT ? maxT+1 : dp[dp[t][k-1]][k-1];
    for(int q = 0, a, b; q < Q; q++){
        scanf("%d %d", &a, &b);
        int ans = 0;
        for(int k = logN; k >= 0; k--){
            if(dp[a][k] <= b){
                a = dp[a][k];
                ans += (1<<k);
            }
        }
        printf("%d\n", ans);
    }
}
```

Pizzeria Queries

Time limit: 1.00 s **Memory limit:** 512 MB

There are n buildings on a street, numbered $1, 2, \ldots, n$. Each building has a pizzeria and an apartment. The pizza price in building k is p_k. If you order a pizza from building a to building b, its price (with delivery) is $p_a + |a - b|$. Your task is to process two types of queries: - The pizza price p_k in building k becomes x. - You are in building k and want to order a pizza. What is the minimum price?

Input

The first input line has two integers n and q: the number of buildings and queries.

The second line has n integers $p_1, p_2, \ldots, p_n$: the initial pizza price in each building.

Finally, there are q lines that describe the queries. Each line is either "1 k x" or "2 k".

Output

Print the answer for each query of type 2.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le p_i, x \le 10^9$
- $1 \le k \le n$

Example

Input:

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

Output:

```
5
4
```

Solution

```
const int maxN = 2e5+1, SIZE = 4*maxN;
const int INF = 0x3f3f3f3f;
int N, Q, p[maxN], mp[maxN], lo[SIZE], hi[SIZE], mn[2][SIZE];
void pull(int i){
    mn[0][i] = min(mn[0][2*i], mn[0][2*i+1]);
    mn[1][i] = min(mn[1][2*i], mn[1][2*i+1]);
}
void init(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r){
        scanf("%d", &p[l]);
        mn[0][i] = l+p[l];
        mn[1][i] = (N-l+1)+p[l];
        mp[l] = i;
        return;
    }
    int m = (l+r)/2;
    init(2*i, l, m);
    init(2*i+1, m+1, r);
    pull(i);
}
void update(int idx, int val){
    int i = mp[idx];
    mn[0][i] = idx+val;
    mn[1][i] = (N-idx+1)+val;
    i >>= 1;
    while(i){
        pull(i);
        i >>= 1;
    }
}
int query(int i, int t, int l, int r){
    if(l > hi[i] || r < lo[i])      return INF;
    if(l <= lo[i] && hi[i] <= r)    return mn[t][i];
    int left = query(2*i, t, l, r);
    int right = query(2*i+1, t, l, r);
    return min(left, right);
}
int main(){
    scanf("%d %d", &N, &Q);
    init(1, 1, N);
    for(int q = 0, t, k, x; q < Q; q++){
        scanf("%d %d", &t, &k);
        if(t == 1){
            scanf("%d", &x);
            update(k, x);
        } else if(t == 2){
            int left = query(1, 1, 1, k);
            int right = query(1, 0, k, N);
            printf("%d\n", min(left-(N-k+1), right-k));
        }
    }
}
```

Polynomial Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to maintain an array of n values and efficiently process the following types of queries: - Increase the first value in range $[a, b]$ by 1, the second value by 2, the third value by 3, and so on. - Calculate the sum of values in range $[a, b]$.

Input

The first input line has two integers n and q: the size of the array and the number of queries.

The next line has n values $t_1, t_2, \ldots, t_n$: the initial contents of the array.

Finally, there are q lines describing the queries. The format of each line is either "1 a b" or "2 a b".

Output

Print the answer to each sum query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le t_i \le 10^6$
- $1 \le a \le b \le n$

Example

Input:

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

Output:

```
17
32
```

Solution

```
const int SIZE = 8e5;
int N, Q, t, a, b, lo[SIZE], hi[SIZE];
ll d1[SIZE], d2[SIZE], s1[SIZE], s2[SIZE];
int len(int i){
    return hi[i]-lo[i]+1;
}
ll sigma(int i){
    return ((ll) hi[i] * (hi[i]+1))/2 - ((ll) lo[i] * (lo[i]-1))/2;
}
void push(int i){
    if(d1[i]){
        d1[2*i] += d1[i];
        d1[2*i+1] += d1[i];
        d1[i] = 0;
    }
    if(d2[i]){
        d2[2*i] += d2[i];
        d2[2*i+1] += d2[i];
```

```
        d2[i] = 0;
    }
}
void pull(int i){
    ll l1 = s1[2*i] + (d1[2*i] * len(2*i));
    ll r1 = s1[2*i+1] + (d1[2*i+1] * len(2*i+1));
    s1[i] = l1 + r1;
    ll l2 = s2[2*i] + (d2[2*i] * sigma(2*i));
    ll r2 = s2[2*i+1] + (d2[2*i+1] * sigma(2*i+1));
    s2[i] = l2 + r2;
}
void build(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r){
        scanf("%lld", &s1[i]);
        return;
    }
    int m = l+(r-l)/2;
    build(2*i, l, m);
    build(2*i+1, m+1, r);
    pull(i);
}
void increment(int type, int i, int l, int r, ll v){
    if(l > hi[i] || r < lo[i])  return;
    if(l <= lo[i] && hi[i] <= r){
        if(type == 1)   d1[i] += v;
        if(type == 2)   d2[i] += v;
        return;
    }
    push(i);
    increment(type, 2*i, l, r, v);
    increment(type, 2*i+1, l, r, v);
    pull(i);
}
ll sum(int type, int i, int l, int r){
    if(l > hi[i] || r < lo[i])  return 0;
    if(l <= lo[i] && hi[i] <= r){
        if(type == 1)   return s1[i] + (d1[i] * len(i));
        if(type == 2)   return s2[i] + (d2[i] * sigma(i));
        return 0;
    }
    push(i);
    ll lsum = sum(type, 2*i, l, r);
    ll rsum = sum(type, 2*i+1, l, r);
    pull(i);
    return lsum + rsum;
}
void update(int l, int r){
    increment(1, 1, l, r, -l+1);
    increment(2, 1, l, r, 1);
}
ll query(int l, int r){
    return sum(1, 1, l, r) + sum(2, 1, l, r);
}
int main(){
    scanf("%d %d", &N, &Q);
```

```
    build(1, 1, N);
    for(int q = 0; q < Q; q++){
        scanf("%d %d %d", &t, &a, &b);
        if(t == 1)      update(a, b);
        else if(t == 2) printf("%lld\n", query(a, b));
    }
}
```

Prefix Sum Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to process q queries of the following types: - update the value at position k to u - what is the maximum prefix sum in range $[a, b]$?

Input

The first input line has two integers n and q: the number of values and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line has three integers: either "1 k u" or "2 a b".

Output

Print the result of each query of type 2.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $-10^9 \le x_i, u \le 10^9$
- $1 \le k \le n$
- $1 \le a \le b \le n$

Example

Input:

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

Output:

```
5
2
0
```

Solution

```
typedef pair<ll,ll> pll;
const int maxN = 2e5+1;
const int SIZE = 4*maxN;
int N, Q, lo[SIZE], hi[SIZE], mp[maxN];
ll sum[SIZE], pre[SIZE];
void pull(int i){
    pre[i] = max(pre[2*i], sum[2*i]+pre[2*i+1]);
    sum[i] = sum[2*i] + sum[2*i+1];
```

```
}
void init(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r){
        scanf("%lld", &sum[i]);
        pre[i] = max(0LL, sum[i]);
        mp[l] = i;
        return;
    }
    int m = (l+r)/2;
    init(2*i, l, m);
    init(2*i+1, m+1, r);
    pull(i);
}
void update(int idx, int val){
    int i = mp[idx];
    sum[i] = val;
    pre[i] = max(0LL, sum[i]);
    i >>= 1;
    while(i){
        pull(i);
        i >>= 1;
    }
}
pll query(int i, int l, int r){
    if(l > hi[i] || r < lo[i])      return {0, 0};
    if(l <= lo[i] && hi[i] <= r)    return {pre[i], sum[i]};
    pll left = query(2*i, l, r);
    pll right = query(2*i+1, l, r);
    return {max(left.first, left.second+right.first), left.second+right.second};
}
int main(){
    scanf("%d %d", &N, &Q);
    init(1, 1, N);
    for(int q = 0, t, a, b; q < Q; q++){
        scanf("%d %d %d", &t, &a, &b);
        if(t == 1)
            update(a, b);
        else if(t == 2)
            printf("%lld\n", query(1, a, b).first);
    }
}
```

Range Interval Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array x of n integers, your task is to process q queries of the form: how many integers i satisfy $a \leq i \leq b$ and $c \leq x_i \leq d$?

Input

The first line has two integers n and q: the number of values and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line has four integers a, b, c and d: how many integers i satisfy $a \leq i \leq b$ and $c \leq x_i \leq d$?

Output

Print the result of each query.

Constraints

- $1 \leq n, q \leq 2 \cdot 10^5$
- $1 \leq x_i \leq 10^9$
- $1 \leq a \leq b \leq n$
- $1 \leq c \leq d \leq 10^9$

Example

Input:

```
8 4
3 2 4 5 1 1 5 3
2 4 2 4
5 6 2 9
1 8 1 5
3 3 4 4
```

Output:

```
2
0
8
1
```

Solution

```
typedef array<int,5> operation;
const int maxN = 2e5+5;
int N, Q, ds[maxN], ans[maxN];
vector<operation> ops;
void update(int idx, int val){
    for(int i = idx; i <= N; i += -i&i)
        ds[i] += val;
}
int query(int idx){
    int sum = 0;
    for(int i = idx; i > 0; i -= -i&i)
        sum += ds[i];
    return sum;
}
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1, x; i <= N; i++){
        scanf("%d", &x);
        ops.push_back({x, 0, i, i, -1});
    }
    for(int q = 1, a, b, c, d; q <= Q; q++){
        scanf("%d %d %d %d", &a, &b, &c, &d);
        ops.push_back({c, -1, a, b, q});
        ops.push_back({d, 1, a, b, q});
    }
    sort(ops.begin(), ops.end());
    for(operation op : ops){
        int type = op[1];
```

```
        if(type == 0)   update(op[2], 1);
        else {
            int a = op[2], b = op[3], qid = op[4];
            int cnt = query(b) - query(a-1);
            ans[qid] += type * cnt;
        }
    }
    for(int q = 1; q <= Q; q++)
        printf("%d\n", ans[q]);
}
```

Range Queries and Copies

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to maintain a list of arrays which initially has a single array. You have to process the following types of queries: - Set the value a in array k to x. - Calculate the sum of values in range $[a, b]$ in array k. - Create a copy of array k and add it to the end of the list.

Input

The first input line has two integers n and q: the array size and the number of queries.

The next line has n integers $t_1, t_2, \ldots, t_n$: the initial contents of the array.

Finally, there are q lines describing the queries. The format of each line is one of the following: "1 k a x", "2 k a b" or "3 k".

Output

Print the answer to each sum query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le t_i, x \le 10^9$
- $1 \le a \le b \le n$

Example

Input:

```
5 6
2 3 1 2 5
3 1
2 1 1 5
2 2 1 5
1 2 2 5
2 1 1 5
2 2 1 5
```

Output:

```
13
13
13
15
```

Solution

```
const int maxN = 2e5+1;
struct Node {
    Node *l, *r;
    int lo, hi;
    ll sum;
    Node(int low, int high, int val) : l(nullptr), r(nullptr), lo(low), hi(high), sum(val) {}
    Node(Node *left, Node *right) : l(left), r(right), sum(0) {
        if(left){
            sum += left->sum;
            lo = left->lo;
        }
        if(right){
            sum += right->sum;
            hi = right->hi;
        }
    }
};
int N, Q, type, k, a, b, x, cnt, t[maxN];
Node* seg[maxN];
Node* build(int l, int r){
    if(l == r)  return new Node(l, r, t[l]);
    int m = (l+r)/2;
    return new Node(build(l, m), build(m+1, r));
}
Node* update(Node* node, int idx, int val){
    if(node->lo == node->hi)    return new Node(idx, idx, val);
    int m = (node->lo+node->hi)/2;
    if(idx <= m)    return new Node(update(node->l, idx, val), node->r);
    else            return new Node(node->l, update(node->r, idx, val));
}
ll query(Node* node, int l, int r){
    if(l > node->hi || r < node->lo)    return 0;
    if(l <= node->lo && node->hi <= r)  return node->sum;
    return query(node->l, l, r) + query(node->r, l, r);
}
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1; i <= N; i++)
        scanf("%d", &t[i]);
    cnt = 1;
    seg[cnt++] = build(1, N);
    for(int q = 0; q < Q; q++){
        scanf("%d %d", &type, &k);
        if(type == 1){
            scanf("%d %d", &a, &x);
            seg[k] = update(seg[k], a, x);
        } else if(type == 2){
            scanf("%d %d", &a, &b);
            printf("%lld\n", query(seg[k], a, b));
        } else if(type == 3){
            seg[cnt++] = new Node(seg[k]->l, seg[k]->r);
        }
    }
}
```

Range Update Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to process q queries of the following types: - increase each value in range $[a, b]$ by u - what is the value at position k?

Input

The first input line has two integers n and q: the number of values and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line has three integers: either "1 a b u" or "2 k".

Output

Print the result of each query of type 2.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le x_i, u \le 10^9$
- $1 \le k \le n$
- $1 \le a \le b \le n$

Example

Input:

```
8 3
3 2 4 5 1 1 5 3
2 4
1 2 5 1
2 4
```

Output:

```
5
6
```

Solution

```
const int maxN = 2e5+1;
int N, Q, t, a, b, k;
ll x, ds[maxN];
void update(int idx, ll val){
    for(int i = idx; i <= N; i += -i&i)
        ds[i] += val;
}
ll query(int idx){
    ll sum = 0;
    for(int i = idx; i > 0; i -= -i&i)
        sum += ds[i];
    return sum;
}
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1; i <= N; i++){
        scanf("%lld", &x);
        update(i, x);
```

```
        update(i+1, -x);
    }
    for(int q = 0; q < Q; q++){
        scanf("%d", &t);
        if(t == 1){
            scanf("%d %d %lld", &a, &b, &x);
            update(b+1, -x);
            update(a, x);
        } else if(t == 2){
            scanf("%d", &k);
            printf("%lld\n", query(k));
        }
    }
}
```

Range Updates and Sums

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to maintain an array of n values and efficiently process the following types of queries: - Increase each value in range $[a, b]$ by x. - Set each value in range $[a, b]$ to x. - Calculate the sum of values in range $[a, b]$.

Input

The first input line has two integers n and q: the array size and the number of queries.

The next line has n values $t_1, t_2, \ldots, t_n$: the initial contents of the array.

Finally, there are q lines describing the queries. The format of each line is one of the following: "1 a b x", "2 a b x", or "3 a b".

Output

Print the answer to each sum query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le t_i, x \le 10^6$
- $1 \le a \le b \le n$

Example

Input:

```
6 5
2 3 1 1 5 3
3 3 5
1 2 4 2
3 3 5
2 2 4 5
3 3 5
```

Output:

```
7
11
15
```

Solution

```
const int maxN = 2e5;
int N, Q, t, a, b, lo[4*maxN], hi[4*maxN];
ll x, del[4*maxN], ass[4*maxN], sum[4*maxN];
int len(int i){
    return hi[i]-lo[i]+1;
}
void increment(int i, ll v){
    del[i] += v;
    sum[i] += len(i) * v;
}
void assign(int i, ll v){
    ass[i] = v;
    del[i] = 0;
    sum[i] = len(i) * v;
}
void push(int i){
    if(ass[i]){
        assign(2*i, ass[i]);
        assign(2*i+1, ass[i]);
        ass[i] = 0;
    }
    if(del[i]){
        increment(2*i, del[i]);
        increment(2*i+1, del[i]);
        del[i] = 0;
    }
}
void pull(int i){
    sum[i] = sum[2*i] + sum[2*i+1];
}
void build(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r){
        scanf("%lld", &sum[i]);
        return;
    }
    int m = l+(r-l)/2;
    build(2*i, l, m);
    build(2*i+1, m+1, r);
    pull(i);
}
void increment(int i, int l, int r, ll v){
    if(l > hi[i] || r < lo[i])  return;
    if(l <= lo[i] && hi[i] <= r){
        increment(i, v); return;
    }
    push(i);
    increment(2*i, l, r, v);
    increment(2*i+1, l, r, v);
    pull(i);
}
void assign(int i, int l, int r, ll v){
    if(l > hi[i] || r < lo[i])  return;
    if(l <= lo[i] && hi[i] <= r){
        assign(i, v); return;
```

```
    }
    push(i);
    assign(2*i, l, r, v);
    assign(2*i+1, l, r, v);
    pull(i);
}
ll query(int i, int l, int r){
    if(l > hi[i] || r < lo[i])  return 0;
    if(l <= lo[i] && hi[i] <= r){
        return sum[i];
    }
    push(i);
    ll lsum = query(2*i, l, r);
    ll rsum = query(2*i+1, l, r);
    pull(i);
    return lsum + rsum;
}
int main(){
    scanf("%d %d", &N, &Q);
    build(1, 1, N);
    for(int q = 0; q < Q; q++){
        scanf("%d %d %d", &t, &a, &b);
        if(t == 1){
            scanf("%lld", &x);
            increment(1, a, b, x);
        } else if(t == 2){
            scanf("%lld", &x);
            assign(1, a, b, x);
        } else if(t == 3){
            printf("%lld\n", query(1, a, b));
        }
    }
}
```

Range Xor Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to process q queries of the form: what is the xor sum of values in range $[a, b]$?

Input

The first input line has two integers n and q: the number of values and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line has two integers a and b: what is the xor sum of values in range $[a, b]$?

Output

Print the result of each query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$
- $1 \le a \le b \le n$

Example

Input:

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

Output:

```
3
0
6
4
```

Solution

```
const int maxN = 2e5;
int N, Q, a, b;
ll x[maxN+1];
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1; i <= N; i++){
        scanf("%lld", &x[i]);
        x[i] ^= x[i-1];
    }
    for(int q = 0; q < Q; q++){
        scanf("%d %d", &a, &b);
        printf("%lld\n", x[b]^x[a-1]);
    }
}
```

Salary Queries

Time limit: 1.00 s **Memory limit:** 512 MB

A company has n employees with certain salaries. Your task is to keep track of the salaries and process queries.

Input

The first input line contains two integers n and q: the number of employees and queries. The employees are numbered $1, 2, \ldots, n$.

The next line has n integers $p_1, p_2, \ldots, p_n$: each employee's salary.

After this, there are q lines describing the queries. Each line has one of the following forms:

- ! k x: change the salary of employee k to x
- ? a b: count the number of employees whose salary is between $a \ldots b$

Output

Print the answer to each ? query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le p_i \le 10^9$
- $1 \le k \le n$

- $1 \le x \le 10^9$
- $1 \le a \le b \le 10^9$

Example

Input:

```
5 3
3 7 2 2 5
? 2 3
! 3 6
? 2 3
```

Output:

```
3
2
```

Solution

```
###include <ext/pb_ds/assoc_container.hpp>
###include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef pair<int,int> pii;
const int maxN = 2e5;
char c;
int N, Q, a, b, p[maxN+1];
tree<pii, null_type, less<pii>, rb_tree_tag,
tree_order_statistics_node_update> T;
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1; i <= N; i++){
        scanf("%d", &p[i]);
        T.insert({p[i], i});
    }
    for(int i = 0; i < Q; i++){
        scanf(" %c %d %d", &c, &a, &b);
        if(c == '!'){
            T.erase({p[a], a});
            T.insert({b, a});
            p[a] = b;
        } else if(c == '?'){
            printf("%ld\n", T.order_of_key({b+1, 0}) - T.order_of_key({a, 0}));
        }
    }
}
```

Static Range Minimum Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to process q queries of the form: what is the minimum value in range $[a, b]$?

Input

The first input line has two integers n and q: the number of values and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line has two integers a and b: what is the minimum value in range $[a, b]$?

Output

Print the result of each query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$
- $1 \le a \le b \le n$

Example

Input:

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

Output:

```
2
1
1
4
```

Solution

```
const int maxN = 2e5+1;
const int logN = 19;
int N, Q, a, b, l, k, x[maxN][logN];
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 0; i < N; i++)
        scanf("%d", &x[i][0]);
    for(int i = 1; i < logN; i++)
        for(int j = 0; j <= N-(1<<i); j++)
            x[j][i] = min(x[j][i-1], x[j+(1<<(i-1))][i-1]);
    for(int q = 0; q < Q; q++){
        scanf("%d %d", &a, &b);
        a--; b--;
        l = b-a+1;
        k = log2(l);
        printf("%d\n", min(x[a][k], x[b-(1<<k)+1][k]));
    }
}
```

Static Range Sum Queries

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to process q queries of the form: what is the sum of values in range $[a, b]$?

Input

The first input line has two integers n and q: the number of values and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line has two integers a and b: what is the sum of values in range $[a, b]$?

Output

Print the result of each query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$
- $1 \le a \le b \le n$

Example

Input:

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

Output:

```
11
2
24
4
```

Solution

```
const int maxN = 2e5;
int N, Q, a, b;
ll x[maxN+1];
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1; i <= N; i++){
        scanf("%lld", &x[i]);
        x[i] += x[i-1];
    }
    for(int q = 0; q < Q; q++){
        scanf("%d %d", &a, &b);
        printf("%lld\n", x[b]-x[a-1]);
    }
}
```

Subarray Sum Queries

Time limit: 1.00 s **Memory limit:** 512 MB

There is an array consisting of n integers. Some values of the array will be updated, and after each update, your task is to report the maximum subarray sum in the array.

Input

The first input line contains integers n and m: the size of the array and the number of updates. The array is indexed $1, 2, \ldots, n$.

The next line has n integers: $x_1, x_2, \ldots, x_n$: the initial contents of the array.

Then there are m lines describing the changes. Each line has two integers k and x: the value at position k becomes x.

Output

After each update, print the maximum subarray sum. Empty subarrays (with sum 0) are allowed.

Constraints

- $1 \le n, m \le 2 \cdot 10^5$
- $-10^9 \le x_i \le 10^9$
- $1 \le k \le n$
- $-10^9 \le x \le 10^9$

Example

Input:

```
5 3
1 2 -3 5 -1
2 6
3 1
2 -2
```

Output:

```
9
13
6
```

Solution

```
const int SIZE = 8e5;
int N, Q, k, lo[SIZE], hi[SIZE], mp[SIZE];
ll x, sum[SIZE], dp[SIZE], dpl[SIZE], dpr[SIZE];
void pull(int i){
    dpl[i] = max(dpl[2*i], sum[2*i] + dpl[2*i+1]);
    dpr[i] = max(dpr[2*i+1], sum[2*i+1] + dpr[2*i]);
    dp[i] = max(dpr[2*i] + dpl[2*i+1], max(dp[2*i], dp[2*i+1]));
    sum[i] = sum[2*i] + sum[2*i+1];
}
void build(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r){
        scanf("%lld", &sum[i]);
        dpl[i] = dpr[i] = dp[i] = sum[i];
        mp[l] = i;
        return;
    }
    int m = l+(r-l)/2;
    build(2*i, l, m);
    build(2*i+1, m+1, r);
    pull(i);
}
void update(int a, ll b){
    a = mp[a];
    dpl[a] = dpr[a] = dp[a] = sum[a] = b;
    a >>= 1;
    while(a > 0){
```

```
        pull(a);
        a >>= 1;
    }
}
int main(){
    scanf("%d %d", &N, &Q);
    build(1, 1, N);
    for(int q = 0; q < Q; q++){
        scanf("%d %lld", &k, &x);
        update(k, x);
        printf("%lld\n", max(0LL, dp[1]));
    }
}
```

Sliding Window Problems

Sliding Window Cost

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers. Your task is to calculate for each window of k elements, from left to right, the minimum total cost of making all elements equal. You can increase or decrease each element with cost x where x is the difference between the new and the original value. The total cost is the sum of such costs.

Input

The first line contains two integers n and k: the number of elements and the size of the window.

Then there are n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Output $n - k + 1$ values: the costs.

Constraints

- $1 \le k \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

```
8 3
2 4 3 5 8 1 2 1
```

Output:

```
2 2 5 7 7 1
```

Solution

```
const int maxN = 2e5+1;
int N, K, x[maxN];
ll losum, hisum;
multiset<int> lo, hi;
ll cost(){
    ll median = *lo.rbegin();
    ll locost = (median * lo.size()) - losum;
    ll hicost = hisum - (median * hi.size());
```

```
    return locost + hicost;
}
void lowToHigh(){
    ll val = *lo.rbegin();
    lo.erase(lo.lower_bound(val));
    hi.insert(val);
    losum -= val;
    hisum += val;
}
void highToLow(){
    ll val = *hi.begin();
    lo.insert(val);
    hi.erase(hi.lower_bound(val));
    losum += val;
    hisum -= val;
}
void adjust(){
    int tot = (int) (lo.size() + hi.size());
    if(lo.size() < (tot+1)/2)       highToLow();
    else if(lo.size() > (tot+1)/2)  lowToHigh();
}
void erase(ll val){
    ll median = *lo.rbegin();
    if(val > median){
        hi.erase(hi.lower_bound(val));
        hisum -= val;
    } else {
        lo.erase(lo.lower_bound(val));
        losum -= val;
    }
    adjust();
}
void insert(ll val){
    if(lo.size() == 0){
        lo.insert(val);
        losum += val;
        return;
    }
    ll median = *lo.rbegin();
    if(val > median){
        hi.insert(val);
        hisum += val;
    } else {
        lo.insert(val);
        losum += val;
    }
    adjust();
}
int main(){
    scanf("%d %d", &N, &K);
    for(int i = 0; i < K; i++){
        scanf("%d", &x[i]);
        insert(x[i]);
    }
    printf("%lld%c", cost(), (" \n")[K==N]);
    for(int i = K; i < N; i++){
```

```
        scanf("%d", &x[i]);
        insert(x[i]);
        erase(x[i-K]);
        printf("%lld%c", cost(), (" \n")[i==N-1]);
    }
}
```

Sliding Window Distinct Values

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers. Your task is to calculate the number of distinct values in each window of k elements, from left to right.

Input

The first line contains two integers n and k: the number of elements and the size of the window.

Then there are n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print $n - k + 1$ values: the numbers of distinct values.

Constraints

- $1 \le k \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

```
8 3
1 2 3 2 5 2 2 2
```

Output:

```
3 2 3 2 2 1
```

Solution

```
const int maxN = 2e5;
int N, K, x[maxN];
map<int,int> freq;
int main(){
    scanf("%d %d", &N, &K);
    for(int i = 0; i < N; i++)
        scanf("%d", &x[i]);
    int distinct = 0;
    for(int i = 0; i < K; i++){
        if(freq[x[i]] == 0)
            distinct++;
        freq[x[i]]++;
    }
    printf("%d%c", distinct, (" \n")[N==K]);
    for(int i = K; i < N; i++){
        if(freq[x[i-K]] == 1)
            distinct--;
        freq[x[i-K]]--;
        if(freq[x[i]] == 0)
```

```
            distinct++;
        freq[x[i]]++;
        printf("%d%c", distinct, (" \n")[i==N-1]);
    }
}
```

Sliding Window Median

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers. Your task is to calculate the median of each window of k elements, from left to right. The median is the middle element when the elements are sorted. If the number of elements is even, there are two possible medians and we assume that the median is the smaller of them.

Input

The first line contains two integers n and k: the number of elements and the size of the window.

Then there are n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print $n - k + 1$ values: the medians.

Constraints

- $1 \le k \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

```
8 3
2 4 3 5 8 1 2 1
```

Output:

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

Solution

```
##include <ext/pb_ds/assoc_container.hpp>
##include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef pair<int,int> pii;
const int maxN = 2e5;
int N, K, x[maxN];
tree<pii, null_type, less<pii>, rb_tree_tag,
tree_order_statistics_node_update> T;
int main(){
    scanf("%d %d", &N, &K);
    for(int i = 0; i < K; i++){
        scanf("%d", &x[i]);
        T.insert({x[i], i});
    }
    printf("%d%c", (*T.find_by_order((K-1)/2)).first, (" \n")[N==K]);
    for(int i = K; i < N; i++){
        scanf("%d", &x[i]);
        T.insert({x[i], i});
        T.erase({x[i-K], i-K});
```

```
        printf("%d%c", (*T.find_by_order((K-1)/2)).first, (" \n")[i==N-1]);
    }
}
```

Sliding Window Minimum

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers. Your task is to calculate the minimum of each window of k elements, from left to right. In this problem the input data is large and it is created using a generator.

Input

The first line contains two integers n and k: the number of elements and the size of the window.

The next line contains four integers x, a, b and c: the input generator parameters. The input is generated as follows:

- $x_1 = x$
- $x_i = (ax_{i-1} + b) \bmod c$ for $i = 2, 3, \ldots, n$

Output

Print the xor of all window minimums.

Constraints

- $1 \le k \le n \le 10^7$
- $0 \le x, a, b \le 10^9$
- $1 \le c \le 10^9$

Example

Input:

```
8 5
3 7 1 11
```

Output:

```
3
```

Explanation: The input array is $[3, 0, 1, 8, 2, 4, 7, 6]$. The windows are $[3, 0, 1, 8, 2]$, $[0, 1, 8, 2, 4]$, $[1, 8, 2, 4, 7]$ and $[8, 2, 4, 7, 6]$, and their minimums are 0, 0, 1 and 2. Thus, the answer is $0 \oplus 0 \oplus 1 \oplus 2 = 3$.

Solution

```
typedef pair<ll,int> pii;
struct Window {
    deque<pii> dq;
    int l, r;
    Window(){
        l = r = 0;
    }
    void push_back(ll x){
        while(!dq.empty() && dq.back().first >= x)   dq.pop_back();
        dq.push_back({x, r});
        r++;
    }
    void pop_front(){
        if(dq.front().second == l)  dq.pop_front();
        l++;
    }
```

```
    ll get_min(){
        return dq.empty() ? -1 : dq.front().first;
    }
};
int N, K;
ll x, a, b, c;
int main(){
    scanf("%d %d", &N, &K);
    scanf("%lld %lld %lld %lld", &x, &a, &b, &c);
    Window window;
    for(int i = 0; i < K; i++){
        window.push_back(x);
        x = (a * x + b) % c;
    }
    ll xum = window.get_min();
    for(int i = K; i < N; i++){
        window.push_back(x);
        window.pop_front();
        xum ^= window.get_min();
        x = (a * x + b) % c;
    }
    printf("%lld\n", xum);
}
```

Sliding Window Sum

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers. Your task is to calculate the sum of each window of k elements, from left to right. In this problem the input data is large and it is created using a generator.

Input

The first line contains two integers n and k: the number of elements and the size of the window.

The next line contains four integers x, a, b and c: the input generator parameters. The input is generated as follows:

- $x_1 = x$
- $x_i = (ax_{i-1} + b) \bmod c$ for $i = 2, 3, \ldots, n$

Output

Print the xor of all window sums.

Constraints

- $1 \le k \le n \le 10^7$
- $0 \le x, a, b \le 10^9$
- $1 \le c \le 10^9$

Example

Input:

```
8 5
3 7 1 11
```

Output:

```
12
```

Explanation: The input array is $[3, 0, 1, 8, 2, 4, 7, 6]$. The windows are $[3, 0, 1, 8, 2]$, $[0, 1, 8, 2, 4]$, $[1, 8, 2, 4, 7]$ and $[8, 2, 4, 7, 6]$, and their sums are 14, 15, 22 and 27. Thus, the answer is $14 \oplus 15 \oplus 22 \oplus 27 = 12$.

Solution

```
const int maxN = 1e7+1;
int N, K;
ll a, b, c, pre[maxN];
int main(){
    scanf("%d %d", &N, &K);
    scanf("%lld %lld %lld %lld", &pre[1], &a, &b, &c);
    for(int i = 2; i <= N; i++)
        pre[i] = (a * pre[i-1] + b) % c;
    for(int i = 2; i <= N; i++)
        pre[i] += pre[i-1];
    ll xum = 0;
    for(int i = K; i <= N; i++)\
        xum ^= (pre[i] - pre[i-K]);
    printf("%lld\n", xum);
}
```

Sliding Window Xor

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers. Your task is to calculate the bitwise xor of each window of k elements, from left to right. In this problem the input data is large and it is created using a generator.

Input

The first line contains two integers n and k: the number of elements and the size of the window.

The next line contains four integers x, a, b and c: the input generator parameters. The input is generated as follows:

- $x_1 = x$
- $x_i = (ax_{i-1} + b) \bmod c$ for $i = 2, 3, \ldots, n$

Output

Print the xor of all window xors.

Constraints

- $1 \le k \le n \le 10^7$
- $0 \le x, a, b \le 10^9$
- $1 \le c \le 10^9$

Example

Input:

```
8 5
3 7 1 11
```

Output:

```
0
```

Explanation: The input array is $[3, 0, 1, 8, 2, 4, 7, 6]$. The windows are $[3, 0, 1, 8, 2]$, $[0, 1, 8, 2, 4]$, $[1, 8, 2, 4, 7]$ and $[8, 2, 4, 7, 6]$, and their xors are 8, 15, 8 and 15. Thus, the answer is $8 \oplus 15 \oplus 8 \oplus 15 = 0$.

Solution

```
const int maxN = 1e7+1;
int N, K;
ll x, a, b, c;
int main(){
    scanf("%d %d", &N, &K);
    scanf("%lld %lld %lld %lld", &x, &a, &b, &c);
    ll xum = 0;
    for(int i = 1; i <= N; i++){
        int l = max(1, i-K+1);
        int r = min(i, N-K+1);
        if((r - l) % 2 == 0)
            xum ^= x;
        x = (a * x + b) % c;
    }
    printf("%lld\n", xum);
}
```

Sorting And Searching

Apartments

Time limit: 1.00 s **Memory limit:** 512 MB

There are n applicants and m free apartments. Your task is to distribute the apartments so that as many applicants as possible will get an apartment. Each applicant has a desired apartment size, and they will accept any apartment whose size is close enough to the desired size.

Input

The first input line has three integers n, m, and k: the number of applicants, the number of apartments, and the maximum allowed difference.

The next line contains n integers $a_1, a_2, \ldots, a_n$: the desired apartment size of each applicant. If the desired size of an applicant is x, they will accept any apartment whose size is between $x - k$ and $x + k$.

The last line contains m integers $b_1, b_2, \ldots, b_m$: the size of each apartment.

Output

Print one integer: the number of applicants who will get an apartment.

Constraints

- $1 \le n, m \le 2 \cdot 10^5$
- $0 \le k \le 10^9$
- $1 \le a_i, b_i \le 10^9$

Example

Input:

```
4 3 5
60 45 80 60
30 60 75
```

Output:

```
2
```

Solution

```
const int maxN = 2e5+1;
int N, M, K, a[maxN], b[maxN];
int main(){
    scanf("%d %d %d", &N, &M, &K);
    for(int i = 0; i < N; i++)  scanf("%d", &a[i]);
    for(int i = 0; i < M; i++)  scanf("%d", &b[i]);
    sort(a, a+N);
    sort(b, b+M);
    int cnt = 0;
    int aptr = 0, bptr = 0;
    while(aptr < N){
        while(bptr < M && b[bptr] < a[aptr]-K)  bptr++;
        if(bptr < M && a[aptr]-K <= b[bptr] && b[bptr] <= a[aptr]+K){
            cnt++;
            aptr++;
            bptr++;
        } else aptr++;
    }
    printf("%d\n", cnt);
}
```

Array Division

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array containing n positive integers. Your task is to divide the array into k subarrays so that the maximum sum in a subarray is as small as possible.

Input

The first input line contains two integers n and k: the size of the array and the number of subarrays in the division.

The next line contains n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print one integer: the maximum sum in a subarray in the optimal division.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le k \le n$
- $1 \le x_i \le 10^9$

Example

Input:

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

Output:

```
8
```

Explanation: An optimal division is $[2, 4], [7], [3, 5]$ where the sums of the subarrays are $6, 7, 8$. The largest sum is the last sum 8.

Solution

```
const int maxN = 2e5;
int N, K, a[maxN];
ll lo = 0, hi = 1e18;
bool check(ll X){
    int k = 1;
    ll sum = 0;
    for(int i = 0; i < N; i++){
        sum += a[i];
        if(sum > X){
            k++;
            sum = a[i];
        }
    }
    return k <= K;
}
int main(){
    scanf("%d %d", &N, &K);
    for(int i = 0; i < N; i++){
        scanf("%d", &a[i]);
        lo = max(lo, (ll) a[i]);
    }
    while(lo <= hi){
        ll mid = lo + (hi-lo)/2;
        if(check(mid))  hi = mid-1;
        else            lo = mid+1;
    }
    printf("%lld\n", lo);
}
```

Collecting Numbers

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array that contains each number between $1 \ldots n$ exactly once. Your task is to collect the numbers from 1 to n in increasing order. On each round, you go through the array from left to right and collect as many numbers as possible. What will be the total number of rounds?

Input

The first line has an integer n: the array size.

The next line has n integers $x_1, x_2, \ldots, x_n$: the numbers in the array.

Output

Print one integer: the number of rounds.

Constraints

- $1 \le n \le 2 \cdot 10^5$

Example

Input:

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

Output:

```
3
```

Solution

```
const int maxN = 2e5+1;
int N, x, cnt, pos[maxN];
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++){
        scanf("%d", &x);
        pos[x] = i;
    }
    cnt = 1;
    for(int i = 2; i <= N; i++)
        if(pos[i-1] > pos[i])
            cnt++;
    printf("%d\n", cnt);
}
```

Collecting Numbers II

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array that contains each number between $1 \ldots n$ exactly once. Your task is to collect the numbers from 1 to n in increasing order. On each round, you go through the array from left to right and collect as many numbers as possible. Given m operations that swap two numbers in the array, your task is to report the number of rounds after each operation.

Input

The first line has two integers n and m: the array size and the number of operations.

The next line has n integers $x_1, x_2, \ldots, x_n$: the numbers in the array.

Finally, there are m lines that describe the operations. Each line has two integers a and b: the numbers at positions a and b are swapped.

Output

Print m integers: the number of rounds after each swap.

Constraints

- $1 \le n, m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
2
3
4
```

Solution

```
const int maxN = 2e5+5;
int N, M, cnt, x[maxN], pos[maxN];
void update(int a, int b){
    if(pos[x[a]-1] <= pos[x[a]] && b < pos[x[a]-1]) cnt++;
    if(pos[x[a]-1] > pos[x[a]] && b >= pos[x[a]-1]) cnt--;
    if(pos[x[a]+1] >= pos[x[a]] && b > pos[x[a]+1]) cnt++;
    if(pos[x[a]+1] < pos[x[a]] && b <= pos[x[a]+1]) cnt--;
    pos[x[a]] = b;
    if(pos[x[b]-1] <= pos[x[b]] && a < pos[x[b]-1]) cnt++;
    if(pos[x[b]-1] > pos[x[b]] && a >= pos[x[b]-1]) cnt--;
    if(pos[x[b]+1] >= pos[x[b]] && a > pos[x[b]+1]) cnt++;
    if(pos[x[b]+1] < pos[x[b]] && a <= pos[x[b]+1]) cnt--;
    pos[x[b]] = a;
    swap(x[a], x[b]);
}
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 1; i <= N; i++){
        scanf("%d", &x[i]);
        pos[x[i]] = i;
    }
    pos[N+1] = N+1;
    cnt = 1;
    for(int i = 1, ptr = 0; i <= N; i++){
        if(ptr > pos[i])
            cnt++;
        ptr = pos[i];
    }
    for(int i = 0, a, b; i < M; i++){
        scanf("%d %d", &a, &b);
        update(a, b);
        printf("%d\n", cnt);
    }
}
```

Concert Tickets

Time limit: 1.00 s **Memory limit:** 512 MB

There are n concert tickets available, each with a certain price. Then, m customers arrive, one after another. Each customer announces the maximum price they are willing to pay for a ticket, and after this, they will get a ticket with the nearest possible price such that it does not exceed the maximum price.

Input

The first input line contains integers n and m: the number of tickets and the number of customers.

The next line contains n integers $h_1, h_2, \ldots, h_n$: the price of each ticket.

The last line contains m integers $t_1, t_2, \ldots, t_m$: the maximum price for each customer in the order they arrive.

Output

Print, for each customer, the price that they will pay for their ticket. After this, the ticket cannot be purchased again.

If a customer cannot get any ticket, print -1.

Constraints

- $1 \le n, m \le 2 \cdot 10^5$
- $1 \le h_i, t_i \le 10^9$

Example

Input:

```
5 3
5 3 7 8 5
4 8 3
```

Output:

```
3
8
-1
```

Solution

```
int N, M, h, t;
multiset<int> prices;
int main(){
    scanf("%d %d", &N, &M);
    for(int i = 0; i < N; i++){
        scanf("%d", &h);
        prices.insert(-h);
    }
    for(int i = 0; i < M; i++){
        scanf("%d", &t);
        if(prices.lower_bound(-t) == prices.end())
            printf("-1\n");
        else {
            printf("%d\n", -(*prices.lower_bound(-t)));
            prices.erase(prices.lower_bound(-t));
        }
    }
}
```

Distinct Numbers

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a list of n integers, and your task is to calculate the number of distinct values in the list.

Input

The first input line has an integer n: the number of values.

The second line has n integers $x_1, x_2, \ldots, x_n$.

Output

Print one integers: the number of distinct values.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

```
5
2 3 2 2 3
```

Output:

```
2
```

Solution

```
int N, x;
set<int> S;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d", &x);
        S.insert(x);
    }
    printf("%d\n", (int) S.size());
}
```

Distinct Values Subarrays

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, count the number of subarrays where each element is dictinct.

Input

The first line has an integer n: the array size.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array contents.

Output

Print the number of subarrays with distinct elements.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

```
4
1 2 1 3
```

Output:

```
8
```

Explanation: The subarrays are [1] (two times), [2], [3], [1, 2], [1, 3], [2, 1] and [2, 1, 3].

Solution

```
const int maxN = 2e5+1;
int N, r, x[maxN];
ll ans;
set<int> S;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d", &x[i]);
    }
    for(int l = 0; l < N; l++){
        while(r < N && S.find(x[r]) == S.end()){
            S.insert(x[r]);
            r++;
        }
        ans += r - l;
        S.erase(x[l]);
    }
    printf("%lld\n", ans);
}
```

Distinct Values Subarrays II

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to calculate the number of subarrays that have at most k distinct values.

Input

The first input line has two integers n and k.

The next line has n integers $x_1, x_2, \ldots, x_n$: the contents of the array.

Output

Print one integer: the number of subarrays.

Constraints

- $1 \le k \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

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

Output:

```
10
```

Solution

```
const int maxN = 2e5+5;
int N, K, x[maxN];
map<int,int> freq;
ll ans;
int main(){
```

```
    scanf("%d %d", &N, &K);
    for(int i = 1; i <= N; i++)
        scanf("%d", &x[i]);
    int unique = 0;
    int r = 0;
    for(int l = 1; l <= N; l++){
        while(r < N && (freq[x[r+1]] >= 1 || unique < K)){
            r++;
            freq[x[r]]++;
            if(freq[x[r]] == 1)
                unique++;
        }
        ans += (r-l+1);
        freq[x[l]]--;
        if(!freq[x[l]])
            unique--;
    }
    printf("%lld\n", ans);
}
```

Distinct Values Subsequences

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, count the number of subsequences where each element is dictinct. A subsequence is a sequence of array elements from left to right that may have gaps.

Input

The first line has an integer n: the array size.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array contents.

Output

Print the number of subsequences with distinct elements. The answer can be large, so print it modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq x_i \leq 10^9$

Example

Input:

```
4
1 2 1 3
```

Output:

```
11
```

Explanation: The subsequences are [1] (two times), [2], [3], [1, 2], [1, 3] (two times), [2, 1], [2, 3], [1, 2, 3] and [2, 1, 3].

Solution

```
const int maxN = 2e5;
const ll MOD = 1e9+7;
int N, x[maxN];
map<int,int> freq;
```

```
ll mod_inv(ll a){
    ll res = 1;
    ll b = MOD-2;
    while(b > 0){
        if(b&1)
            res = (res * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return res;
}
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)
        scanf("%d", &x[i]);
    ll ans = 0;
    ll prod = 1;
    for(int i = 0; i < N; i++){
        ll seqs = (prod * mod_inv(freq[x[i]]+1)) % MOD;
        ans = (ans + seqs) % MOD;
        freq[x[i]]++;
        prod = (prod * (freq[x[i]]+1) ) % MOD;
        prod = (prod * mod_inv(freq[x[i]])) % MOD;
    }
    printf("%lld\n", ans);
}
```

Factory Machines

Time limit: 1.00 s **Memory limit:** 512 MB

A factory has n machines which can be used to make products. Your goal is to make a total of t products. For each machine, you know the number of seconds it needs to make a single product. The machines can work simultaneously, and you can freely decide their schedule. What is the shortest time needed to make t products?

Input

The first input line has two integers n and t: the number of machines and products.

The next line has n integers $k_1, k_2, \ldots, k_n$: the time needed to make a product using each machine.

Output

Print one integer: the minimum time needed to make t products.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le t \le 10^9$
- $1 \le k_i \le 10^9$

Example

Input:

```
3 7
3 2 5
```

Output:

```
8
```

Explanation: Machine 1 makes two products, machine 2 makes four products and machine 3 makes one product.

Solution

```
const int maxN = 2e5;
int N;
ll T, cnt, k[maxN];
bool check(ll t){
    cnt = 0;
    for(int i = 0; i < N; i++){
        cnt += t/k[i];
        if(cnt >= T)
            return true;
    }
    return false;
}
int main(){
    scanf("%d %lld", &N, &T);
    for(int i = 0; i < N; i++)
        scanf("%lld", &k[i]);
    ll lo = 0, hi = 1e18;
    while(lo <= hi){
        ll mid = lo + (hi-lo)/2;
        if(check(mid))  hi = mid-1;
        else            lo = mid+1;
    }
    printf("%lld\n", lo);
}
```

Ferris Wheel

Time limit: 1.00 s **Memory limit:** 512 MB

There are n children who want to go to a Ferris wheel, and your task is to find a gondola for each child. Each gondola may have one or two children in it, and in addition, the total weight in a gondola may not exceed x. You know the weight of every child. What is the minimum number of gondolas needed for the children?

Input

The first input line contains two integers n and x: the number of children and the maximum allowed weight.

The next line contains n integers $p_1, p_2, \ldots, p_n$: the weight of each child.

Output

Print one integer: the minimum number of gondolas.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x \le 10^9$
- $1 \le p_i \le x$

Example

Input:

```
4 10
7 2 3 9
```

Output:

```
3
```

Solution

```
const int maxN = 2e5;
int N, X, l, ans, p[maxN];
int main(){
    scanf("%d %d", &N, &X);
    for(int i = 0; i < N; i++)
        scanf("%d", &p[i]);
    sort(p, p+N);
    l = 0;
    for(int r = N-1; r >= l; r--){
        if(p[l] + p[r] <= X)
            l++;
        ans++;
    }
    printf("%d\n", ans);
}
```

Josephus Problem I

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a game where there are n children (numbered $1, 2, \ldots, n$) in a circle. During the game, every other child is removed from the circle until there are no children left. In which order will the children be removed?

Input

The only input line has an integer n.

Output

Print n integers: the removal order.

Constraints

- $1 \le n \le 2 \cdot 10^5$

Example

Input:

```
7
```

Output:

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

Solution

```
###include <ext/pb_ds/assoc_container.hpp>
###include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
int N, idx;
tree<int, null_type, less<int>, rb_tree_tag,
```

```
tree_order_statistics_node_update> T;
int main(){
    scanf("%d", &N);
    for(int i = 1; i <= N; i++)
        T.insert(i);
    idx = 1;
    while(T.size()){
        idx %= T.size();
        int x = *T.find_by_order(idx);
        T.erase(x);
        printf("%d%c", x, (" \n")[T.size() == 0]);
        idx++;
    }
}
```

Josephus Problem II

Time limit: 1.00 s **Memory limit:** 512 MB

Consider a game where there are n children (numbered $1, 2, \ldots, n$) in a circle. During the game, repeatedly k children are skipped and one child is removed from the circle. In which order will the children be removed?

Input

The only input line has two integers n and k.

Output

Print n integers: the removal order.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $0 \le k \le 10^9$

Example

Input:

```
7 2
```

Output:

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

Solution

```
##include <ext/pb_ds/assoc_container.hpp>
##include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
int N, K, idx;
tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> T;
int main(){
    scanf("%d %d", &N, &K);
    for(int i = 1; i <= N; i++)
        T.insert(i);
    idx = K;
    while(T.size()){
        idx %= T.size();
```

```
        int x = *T.find_by_order(idx);
        T.erase(x);
        printf("%d%c", x, (" \n")[T.size() == 0]);
        idx += K;
    }
}
```

Maximum Subarray Sum

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to find the maximum sum of values in a contiguous, nonempty subarray.

Input

The first input line has an integer n: the size of the array.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Output

Print one integer: the maximum subarray sum.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $-10^9 \le x_i \le 10^9$

Example

Input:

```
8
-1 3 -2 5 3 -5 2 2
```

Output:

```
9
```

Solution

```
const int maxN = 2e5;
const ll INF = 0x3f3f3f3f3f3f3f3f;
int N;
ll maxSum, curSum, x[maxN];
int main(){
    scanf("%d", &N);
    maxSum = -INF;
    for(int i = 0; i < N; i++){
        scanf("%lld", &x[i]);
        maxSum = max(maxSum, x[i]);
    }
    for(int i = 0; i < N; i++){
        curSum += x[i];
        maxSum = max(maxSum, curSum);
        if(curSum < 0)  curSum = 0;
    }
    printf("%lld\n", maxSum);
}
```

Maximum Subarray Sum II

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to find the maximum sum of values in a contiguous subarray with length between a and b.

Input

The first input line has three integers n, a and b: the size of the array and the minimum and maximum subarray length.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Output

Print one integer: the maximum subarray sum.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le a \le b \le n$
- $-10^9 \le x_i \le 10^9$

Example

Input:

```
8 1 2
-1 3 -2 5 3 -5 2 2
```

Output:

```
8
```

Solution

```
const int maxN = 2e5+1;
int N, A, B;
ll pre[maxN];
multiset<ll> S;
int main(){
    scanf("%d %d %d", &N, &A, &B);
    for(int i = 1; i <= N; i++){
        scanf("%lld", &pre[i]);
        pre[i] += pre[i-1];
    }
    for(int i = A; i <= B; i++)
        S.insert(pre[i]);
    ll best = *S.rbegin();
    for(int i = 1; i+A <= N; i++){
        S.erase(pre[i+A-1]);
        S.insert(pre[min(i+B, N)]);
        best = max(best, *S.rbegin()-pre[i]);
    }
    printf("%lld\n", best);
}
```

Missing Coin Sum

Time limit: 1.00 s **Memory limit:** 512 MB

You have n coins with positive integer values. What is the smallest sum you cannot create using a subset of the coins?

Input

The first line has an integer n: the number of coins.

The second line has n integers $x_1, x_2, \ldots, x_n$: the value of each coin.

Output

Print one integer: the smallest coin sum.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

```
5
2 9 1 2 7
```

Output:

```
6
```

Solution

```
const int maxN = 2e5;
int N, x[maxN];
ll res;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)
        scanf("%d", &x[i]);
    sort(x, x+N);
    res = 1;
    for(int i = 0; i < N && x[i] <= res; i++)
        res += x[i];
    printf("%lld\n", res);
}
```

Movie Festival

Time limit: 1.00 s **Memory limit:** 512 MB

In a movie festival n movies will be shown. You know the starting and ending time of each movie. What is the maximum number of movies you can watch entirely?

Input

The first input line has an integer n: the number of movies.

After this, there are n lines that describe the movies. Each line has two integers a and b: the starting and ending times of a movie.

Output

Print one integer: the maximum number of movies.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq a < b \leq 10^9$

Example

Input:

```
3
3 5
4 9
5 8
```

Output:

```
2
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 2e5+1;
int N, a[maxN], b[maxN], dp[2*maxN];
struct Movie {int t, id, type;} movies[2*maxN];
map<int,int> mp;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d %d", &a[i], &b[i]);
        movies[2*i] = {a[i], i, 0};
        movies[2*i+1] = {b[i], i, 1};
    }
    sort(movies, movies+2*N, [](Movie A, Movie B){
        return A.t == B.t ? A.id < B.id : A.t < B.t;
    });
    for(int i = 0; i < 2*N; i++)
        mp[movies[i].t] = i;
    for(int i = 0; i < 2*N; i++){
        if(movies[i].type == 0) dp[i] = dp[i-1];
        else dp[i] = max(dp[i-1], dp[mp[a[movies[i].id]]]+1);
    }
    printf("%d\n", dp[2*N-1]);
}
```

Movie Festival II

Time limit: 1.00 s **Memory limit:** 512 MB

In a movie festival, n movies will be shown. Syrjälä's movie club consists of k members, who will be all attending the festival. You know the starting and ending time of each movie. What is the maximum total number of movies the club members can watch entirely if they act optimally?

Input

The first input line has two integers n and k: the number of movies and club members.

After this, there are n lines that describe the movies. Each line has two integers a and b: the starting and ending time of a movie.

Output

Print one integer: the maximum total number of movies.

Constraints

- $1 \le k \le n \le 2 \cdot 10^5$
- $1 \le a < b \le 10^9$

Example

Input:

```
5 2
1 5
8 10
3 6
2 5
6 9
```

Output:

```
4
```

Solution

```
const int maxN = 2e5, SIZE = 5e6;
const int INF = 0x3f3f3f3f;
int N, K, ans, A[maxN], B[maxN], lo[SIZE], hi[SIZE], d[SIZE], mx[SIZE];
struct Movie { int t, id, type; } movies[2*maxN];
map<int,int> mp;
void push(int i){
    if(d[i]){
        d[2*i] += d[i];
        d[2*i+1] += d[i];
        d[i] = 0;
    }
}
void pull(int i){
    mx[i] = max(mx[2*i]+d[2*i], mx[2*i+1]+d[2*i+1]);
}
void init(int i, int l, int r){
    lo[i] = l; hi[i] = r;
    if(l == r)  return;
    int m = l+(r-l)/2;
    init(2*i, l, m);
    init(2*i+1, m+1, r);
    pull(i);
}
void increment(int i, int l, int r){
    if(l > hi[i] || r < lo[i])  return;
    if(l <= lo[i] && hi[i] <= r){
        d[i]++; return;
    }
    push(i);
    increment(2*i, l, r);
```

```
    increment(2*i+1, l, r);
    pull(i);
}
int maximum(int i, int l, int r){
    if(l > hi[i] || r < lo[i])      return -INF;
    if(l <= lo[i] && hi[i] <= r)    return mx[i]+d[i];
    push(i);
    int lmax = maximum(2*i, l, r);
    int rmax = maximum(2*i+1, l, r);
    pull(i);
    return max(lmax, rmax);
}
int main(){
    scanf("%d %d", &N, &K);
    for(int i = 0; i < N; i++){
        scanf("%d %d", &A[i], &B[i]);
        movies[2*i] = {A[i], i, 0};
        movies[2*i+1] = {B[i], i, 1};
    }
    sort(movies, movies+2*N, [](Movie a, Movie b){
        return a.t == b.t ? A[a.id] > A[b.id] : a.t < b.t;
    });
    for(int i = 0; i < 2*N; i++)
        mp[movies[i].t] = 2*i+1;
    init(1, 1, 4*N);
    for(int i = 0; i < 2*N; i++){
        if(movies[i].type == 1){
            int id = movies[i].id;
            int a = mp[A[id]];
            int b = mp[B[id]];
            if(maximum(1, a, b) < K){
                increment(1, a, b-1);
                ans++;
            }
        }
    }
    printf("%d\n", ans);
}
```

Nearest Smaller Values

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to find for each array position the nearest position to its left having a smaller value.

Input

The first input line has an integer n: the size of the array.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Output

Print n integers: for each array position the nearest position with a smaller value. If there is no such position, print 0.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x_i \le 10^9$

Example

Input:

```
8
2 5 1 4 8 3 2 5
```

Output:

```
0 1 0 3 4 3 3 7
```

Solution

```
const int maxN = 2e5+1;
int N, x[maxN], ds[maxN];
int main(){
    scanf("%d ", &N);
    for(int i = 1; i <= N; i++){
        scanf("%d", &x[i]);
        int k = i-1;
        while(x[k] >= x[i])
            k = ds[k];
        ds[i] = k;
        printf("%d%c", ds[i], (" \n")[i==N]);
    }
}
```

Nested Ranges Check

Time limit: 1.00 s **Memory limit:** 512 MB

Given n ranges, your task is to determine for each range if it contains some other range and if some other range contains it. Range $[a, b]$ contains range $[c, d]$ if $a \le c$ and $d \le b$.

Input

The first input line has an integer n: the number of ranges.

After this, there are n lines that describe the ranges. Each line has two integers x and y: the range is $[x, y]$.

You may assume that no range appears more than once in the input.

Output

First print a line that describes for each range (in the input order) if it contains some other range (1) or not (0).

Then print a line that describes for each range (in the input order) if some other range contains it (1) or not (0).

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x < y \le 10^9$

Example

Input:

```
4
1 6
2 4
4 8
3 6
```

Output:

```
1 0 0 0
0 1 0 1
```

Solution

```
typedef array<int,3> triple;
const int maxN = 2e5+1;
const int SIZE = 2*maxN;
int N, ans[2][maxN], ds[SIZE];
triple intervals[maxN];
set<int> unique_vals;
map<int,int> mp;
void update(int idx, int val){
    for(int i = idx; i < SIZE; i += -i&i)   ds[i] += val;
}
int query(int idx){
    int cnt = 0;
    for(int i = idx; i > 0; i -= -i&i)
        cnt += ds[i];
    return cnt;
}
int main(){
    scanf("%d", &N);
    for(int i = 0, x, y; i < N; i++){
        scanf("%d %d", &x, &y);
        intervals[i] = {x, y, i};
        unique_vals.insert(x);
        unique_vals.insert(y);
    }
    sort(intervals, intervals+N, [](triple A, triple B){
        return (A[0] == B[0] ? A[1] > B[1] : A[0] < B[0]);
    });
    int val_id = 1;
    for(int x : unique_vals)
        mp[x] = val_id++;
    for(int i = N-1; i >= 0; i--){
        int y = mp[intervals[i][1]];
        int id = intervals[i][2];
        ans[0][id] = query(y);
        update(y, 1);
    }
    fill(ds, ds+SIZE, 0);
    for(int i = 0; i < N; i++){
        int y = mp[intervals[i][1]];
        int id = intervals[i][2];
        ans[1][id] = i-query(y-1);
        update(y, 1);
    }
    for(int i = 0; i < 2; i++)
        for(int j = 0; j < N; j++)
```

```
            printf("%d%c", (ans[i][j] ? 1 : 0), (" \n")[j==N-1]);
}
```

Nested Ranges Count

Time limit: 1.00 s **Memory limit:** 512 MB

Given n ranges, your task is to count for each range how many other ranges it contains and how many other ranges contain it. Range $[a, b]$ contains range $[c, d]$ if $a \leq c$ and $d \leq b$.

Input

The first input line has an integer n: the number of ranges.

After this, there are n lines that describe the ranges. Each line has two integers x and y: the range is $[x, y]$.

You may assume that no range appears more than once in the input.

Output

First print a line that describes for each range (in the input order) how many other ranges it contains.

Then print a line that describes for each range (in the input order) how many other ranges contain it.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq x < y \leq 10^9$

Example

Input:

```
4
1 6
2 4
4 8
3 6
```

Output:

```
2 0 0 0
0 1 0 1
```

Solution

```
typedef array<int,3> triple;
const int maxN = 2e5+1;
const int SIZE = 2*maxN;
int N, ans[2][maxN], ds[SIZE];
triple intervals[maxN];
set<int> unique_vals;
map<int,int> mp;
void update(int idx, int val){
    for(int i = idx; i < SIZE; i += -i&i)   ds[i] += val;
}
int query(int idx){
    int cnt = 0;
    for(int i = idx; i > 0; i -= -i&i)
        cnt += ds[i];
    return cnt;
}
```

```
int main(){
    scanf("%d", &N);
    for(int i = 0, x, y; i < N; i++){
        scanf("%d %d", &x, &y);
        intervals[i] = {x, y, i};
        unique_vals.insert(x);
        unique_vals.insert(y);
    }
    sort(intervals, intervals+N, [](triple A, triple B){
        return (A[0] == B[0] ? A[1] > B[1] : A[0] < B[0]);
    });
    int val_id = 1;
    for(int x : unique_vals)
        mp[x] = val_id++;
    for(int i = N-1; i >= 0; i--){
        int y = mp[intervals[i][1]];
        int id = intervals[i][2];
        ans[0][id] = query(y);
        update(y, 1);
    }
    fill(ds, ds+SIZE, 0);
    for(int i = 0; i < N; i++){
        int y = mp[intervals[i][1]];
        int id = intervals[i][2];
        ans[1][id] = i-query(y-1);
        update(y, 1);
    }
    for(int i = 0; i < 2; i++)
        for(int j = 0; j < N; j++)
            printf("%d%c", ans[i][j], (" \n")[j==N-1]);
}
```

Playlist

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a playlist of a radio station since its establishment. The playlist has a total of n songs. What is the longest sequence of successive songs where each song is unique?

Input

The first input line contains an integer n: the number of songs.

The next line has n integers $k_1, k_2, \ldots, k_n$: the id number of each song.

Output

Print the length of the longest sequence of unique songs.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le k_i \le 10^9$

Example

Input:

```
8
1 2 1 3 2 7 4 2
```

Output:

```
5
```

Solution

```
int N, k, l, ans;
map<int,int> mp;
int main(){
    scanf("%d", &N);
    l = 1;
    for(int r = 1; r <= N; r++){
        scanf("%d", &k);
        if(mp[k]){
            ans = max(ans, r-l);
            l = max(l, mp[k]+1);
            mp[k] = r;
        } else {
            ans = max(ans, r-l+1);
            mp[k] = r;
        }
    }
    ans = max(N-l+1, ans);
    printf("%d\n", ans);
}
```

Reading Books

Time limit: 1.00 s **Memory limit:** 512 MB

There are n books, and Kotivalo and Justiina are going to read them all. For each book, you know the time it takes to read it. They both read each book from beginning to end, and they cannot read a book at the same time. What is the minimum total time required?

Input

The first input line has an integer n: the number of books.

The second line has n integers $t_1, t_2, \ldots, t_n$: the time required to read each book.

Output

Print one integer: the minimum total time.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le t_i \le 10^9$

Example

Input:

```
3
2 8 3
```

Output:

```
16
```

Solution

```
int N;
ll t, mx, sum;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%lld", &t);
        mx = max(mx, t);
        sum += t;
    }
    printf("%lld\n", max(sum, 2*mx));
}
```

Restaurant Customers

Time limit: 1.00 s **Memory limit:** 512 MB

You are given the arrival and leaving times of n customers in a restaurant. What was the maximum number of customers in the restaurant at any time?

Input

The first input line has an integer n: the number of customers.

After this, there are n lines that describe the customers. Each line has two integers a and b: the arrival and leaving times of a customer.

You may assume that all arrival and leaving times are distinct.

Output

Print one integer: the maximum number of customers.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le a < b \le 10^9$

Example

Input:

```
3
5 8
2 4
3 9
```

Output:

```
2
```

Solution

```
const int maxN = 4e5;
int N, a, b, cur, ans;
struct event {int time, type;} events[maxN];
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d %d", &a, &b);
        events[2*i] = {a, 1};
```

```
        events[2*i+1] = {b, -1};
    }
    sort(events, events+2*N, [](event A, event B){
        return A.time == B.time ? A.type < B.type : A.time < B.time;
    });
    for(int i = 0; i < 2*N; i++){
        cur += events[i].type;
        ans = max(ans, cur);
    }
    printf("%d\n", ans);
}
```

Room Allocation

Time limit: 1.00 s **Memory limit:** 512 MB

There is a large hotel, and n customers will arrive soon. Each customer wants to have a single room. You know each customer's arrival and departure day. Two customers can stay in the same room if the departure day of the first customer is earlier than the arrival day of the second customer. What is the minimum number of rooms that are needed to accommodate all customers? And how can the rooms be allocated?

Input

The first input line contains an integer n: the number of customers.

Then there are n lines, each of which describes one customer. Each line has two integers a and b: the arrival and departure day.

Output

Print first an integer k: the minimum number of rooms required.

After that, print a line that contains the room number of each customer in the same order as in the input. The rooms are numbered $1, 2, \ldots, k$. You can print any valid solution.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le a \le b \le 10^9$

Example

Input:

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

Output:

```
2
1 2 1
```

Solution

```
const int maxN = 2e5;
set<int> S;
int N, a, b, rooms, ans[maxN+1];
struct query {int type, time, id;} queries[2*maxN];
int main(){
```

```
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        S.insert(i+1);
        scanf("%d %d", &a, &b);
        queries[2*i] = {0, a, i+1};
        queries[2*i+1] = {1, b, i+1};
    }
    sort(queries, queries+2*N, [](query A, query B){
        if(A.id == B.id)    return A.type < B.type;
        return A.time == B.time ? A.type < B.type : A.time < B.time;
    });
    for(int i = 0; i < 2*N; i++){
        if(queries[i].type == 0){
            ans[queries[i].id] = *S.begin();
            S.erase(S.begin());
        } else if(queries[i].type == 1){
            S.insert(ans[queries[i].id]);
        }
    }
    for(int i = 1; i <= N; i++)
        rooms = max(rooms, ans[i]);
    printf("%d\n", rooms);
    for(int i = 1; i <= N; i++)
        printf("%d%c", ans[i], (" \n")[i==N]);
}
```

Stick Lengths

Time limit: 1.00 s **Memory limit:** 512 MB

There are n sticks with some lengths. Your task is to modify the sticks so that each stick has the same length. You can either lengthen and shorten each stick. Both operations cost x where x is the difference between the new and original length. What is the minimum total cost?

Input

The first input line contains an integer n: the number of sticks.

Then there are n integers: $p_1, p_2, \ldots, p_n$: the lengths of the sticks.

Output

Print one integer: the minimum total cost.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le p_i \le 10^9$

Example

Input:

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

Output:

```
5
```

Solution

```
const int maxN = 2e5;
int N, p[maxN];
ll median, sum;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++)
        scanf("%d", &p[i]);
    sort(p, p+N);
    median = p[(N-1)/2];
    for(int i = 0; i < N; i++)
        sum += abs(p[i]-median);
    printf("%lld\n", sum);
}
```

Subarray Divisibility

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to count the number of subarrays where the sum of values is divisible by n.

Input

The first input line has an integer n: the size of the array.

The next line has n integers $a_1, a_2, \ldots, a_n$: the contents of the array.

Output

Print one integer: the required number of subarrays.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $-10^9 \le a_i \le 10^9$

Example

Input:

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

Output:

```
1
```

Solution

```
const int maxN = 2e5;
int N, pre, a, freq[maxN];
ll cnt;
int mod(int x){
    return (x % N + N) % N;
}
int main(){
    scanf("%d", &N);
    freq[0] = 1;
    for(int i = 0; i < N; i++){
        scanf("%d", &a);
```

```
        pre = mod(pre+a);
        cnt += freq[pre];
        freq[pre]++;
    }
    printf("%lld\n", cnt);
}
```

Subarray Sums I

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n positive integers, your task is to count the number of subarrays having sum x.

Input

The first input line has two integers n and x: the size of the array and the target sum x.

The next line has n integers $a_1, a_2, \ldots, a_n$: the contents of the array.

Output

Print one integer: the required number of subarrays.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x, a_i \le 10^9$

Example

Input:

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

Output:

```
3
```

Solution

```
const int maxN = 2e5;
int N, x, l, sum, cnt, a[maxN+1];
int main(){
    scanf("%d %d", &N, &x);
    for(int r = 1; r <= N; r++){
        scanf("%d", &a[r]);
        sum += a[r];
        while(sum > x)
            sum -= a[l++];
        if(sum == x)    cnt++;
    }
    printf("%d\n", cnt);
}
```

Subarray Sums II

Time limit: 1.00 s **Memory limit:** 512 MB

Given an array of n integers, your task is to count the number of subarrays having sum x.

Input

The first input line has two integers n and x: the size of the array and the target sum x.

The next line has n integers $a_1, a_2, \ldots, a_n$: the contents of the array.

Output

Print one integer: the required number of subarrays.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $-10^9 \le x, a_i \le 10^9$

Example

Input:

```
5 7
2 -1 3 5 -2
```

Output:

```
2
```

Solution

```
int N;
ll X, a, cnt, pre;
map<ll,int> freq;
int main(){
    scanf("%d %lld", &N, &X);
    freq[0] = 1;
    for(int i = 0; i < N; i++){
        scanf("%lld", &a);
        pre += a;
        cnt += freq[pre-X];
        freq[pre]++;
    }
    printf("%lld\n", cnt);
}
```

Sum of Four Values

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers, and your task is to find four values (at distinct positions) whose sum is x.

Input

The first input line has two integers n and x: the array size and the target sum.

The second line has n integers $a_1, a_2, \ldots, a_n$: the array values.

Output

Print four integers: the positions of the values. If there are several solutions, you may print any of them. If there are no solutions, print IMPOSSIBLE.

Constraints

- $1 \le n \le 1000$
- $1 \le x, a_i \le 10^9$

Example

Input:

```
8 15
3 2 5 8 1 3 2 3
```

Output:

```
2 4 6 7
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 1001;
int N;
ll X, a[maxN];
unordered_map<ll,vector<pii>> mp;
int main(){
    scanf("%d %lld", &N, &X);
    mp.reserve(maxN*maxN);
    for(int i = 1; i <= N; i++){
        scanf("%lld", &a[i]);
        for(int j = 1; j < i; j++){
            ll psum = a[i] + a[j];
            if(psum >= X)   continue;
            if(mp.find(X-psum) != mp.end()){
                for(pii P : mp[X-psum]){
                    if(P.first != j && P.second != j && P.first != i && P.second != i){
                        printf("%d %d %d %d\n", P.first, P.second, j, i);
                        return 0;
                    }
                }
            } else mp[psum].push_back({j, i});
        }
    }
    printf("IMPOSSIBLE\n");
}
```

Sum of Three Values

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers, and your task is to find three values (at distinct positions) whose sum is x.

Input

The first input line has two integers n and x: the array size and the target sum.

The second line has n integers $a_1, a_2, \ldots, a_n$: the array values.

Output

Print three integers: the positions of the values. If there are several solutions, you may print any of them. If there are no solutions, print IMPOSSIBLE.

Constraints

- $1 \le n \le 5000$
- $1 \le x, a_i \le 10^9$

Example

Input:

```
4 8
2 7 5 1
```

Output:

```
1 3 4
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 5000;
int N, X, v, r;
pii a[maxN];
int main(){
    scanf("%d %d", &N, &X);
    for(int i = 0; i < N; i++){
        scanf("%d", &v);
        a[i] = {v, i+1};
    }
    sort(a, a+N);
    for(int i = 0; i < N; i++){
        v = X - a[i].first;
        r = N-1;
        for(int l = i+1; l < r; l++){
            while(l+1 < r && a[l].first + a[r].first > v)   r--;
            if(a[l].first + a[r].first == v){
                printf("%d %d %d\n", a[i].second, a[l].second, a[r].second);
                return 0;
            }
        }
    }
    printf("IMPOSSIBLE\n");
}
```

Sum of Two Values

Time limit: 1.00 s **Memory limit:** 512 MB

You are given an array of n integers, and your task is to find two values (at distinct positions) whose sum is x.

Input

The first input line has two integers n and x: the array size and the target sum.

The second line has n integers $a_1, a_2, \ldots, a_n$: the array values.

Output

Print two integers: the positions of the values. If there are several solutions, you may print any of them. If there are no solutions, print IMPOSSIBLE.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le x, a_i \le 10^9$

Example

Input:

```
4 8
2 7 5 1
```

Output:

```
2 4
```

Solution

```
typedef pair<int,int> pii;
const int maxN = 2e5+1;
int N, x, a;
pii p[maxN];
int find(int val){
    int l = 1, r = N;
    while(l <= r){
        int m = l+(r-l)/2;
        if(p[m].first == val)       return p[m].second;
        else if(p[m].first < val)   l = m+1;
        else                        r = m-1;
    }
    return 0;
}
int main(){
    scanf("%d %d", &N, &x);
    for(int i = 1; i <= N; i++){
        scanf("%d", &a);
        p[i] = {a, i};
    }
    sort(p+1, p+N+1);
    for(int i = 1; i <= N; i++){
        int other = find(x-p[i].first);
        if(other != 0 && other != p[i].second){
            printf("%d %d\n", p[i].second, other);
            return 0;
        }
    }
    printf("IMPOSSIBLE\n");
}
```

Tasks and Deadlines

Time limit: 1.00 s **Memory limit:** 512 MB

You have to process n tasks. Each task has a duration and a deadline, and you will process the tasks in some order one after another. Your reward for a task is $d - f$ where d is its deadline and f is your finishing time. (The starting time is 0, and you have to process all tasks even if a task would yield negative reward.) What is your maximum reward if you act optimally?

Input

The first input line has an integer n: the number of tasks.

After this, there are n lines that describe the tasks. Each line has two integers a and d: the duration and deadline of the task.

Output

Print one integer: the maximum reward.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le a, d \le 10^6$

Example

Input:

```
3
6 10
8 15
5 12
```

Output:

```
2
```

Solution

```
const int maxN = 2e5;
int N, a, d;
long long timer, reward;
struct task {int a, d;} tasks[maxN];
int main (){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d %d", &a, &d);
        tasks[i] = {a, d};
    }
    sort(tasks, tasks+N, [](task x, task y){
        return x.a == y.a ? x.d < y.d : x.a < y.a;
    });
    for(int i = 0; i < N; i++){
        timer += tasks[i].a;
        reward += (tasks[i].d - timer);
    }
    printf("%lld\n", reward);
}
```

Towers

Time limit: 1.00 s **Memory limit:** 512 MB

You are given n cubes in a certain order, and your task is to build towers using them. Whenever two cubes are one on top of the other, the upper cube must be smaller than the lower cube. You must process the cubes in the given order. You can always either place the cube on top of an existing tower, or begin a new tower. What is the minimum possible number of towers?

Input

The first input line contains an integer n: the number of cubes.

The next line contains n integers $k_1, k_2, \ldots, k_n$: the sizes of the cubes.

Output

Print one integer: the minimum number of towers.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le k_i \le 10^9$

Example

Input:

```
5
3 8 2 1 5
```

Output:

```
2
```

Solution

```
int N, k;
multiset<int> S;
multiset<int>::iterator it;
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N; i++){
        scanf("%d", &k);
        it = S.upper_bound(k);
        if(it != S.end())
            S.erase(it);
        S.insert(k);
    }
    printf("%d\n", (int) S.size());
}
```

Traffic Lights

Time limit: 1.00 s **Memory limit:** 512 MB

There is a street of length x whose positions are numbered $0, 1, \ldots, x$. Initially there are no traffic lights, but n sets of traffic lights are added to the street one after another. Your task is to calculate the length of the longest passage without traffic lights after each addition.

Input

The first input line contains two integers x and n: the length of the street and the number of sets of traffic lights.

Then, the next line contains n integers $p_1, p_2, \ldots, p_n$: the position of each set of traffic lights. Each position is distinct.

Output

Print the length of the longest passage without traffic lights after each addition.

Constraints

- $1 \leq x \leq 10^9$
- $1 \leq n \leq 2 \cdot 10^5$
- $0 < p_i < x$

Example

Input:

```
8 3
3 6 2
```

Output:

```
5 3 3
```

Solution

```
const int maxN = 2e5;
set<int> S;
set<int>::iterator f, c, b;
int N, X, a, best, ans[maxN];
struct light {int pos, id;} x[maxN+2];
int main(){
    scanf("%d %d", &X, &N);
    x[0] = {0, 0}; x[N+1] = {X, 1};
    S.insert(0); S.insert(X);
    for(int i = 0; i < N; i++){
        scanf("%d", &a);
        x[i+1] = {a, i+2};
        S.insert(a);
    }
    sort(x, x+maxN+2, [](light A, light B){
        return A.pos < B.pos;
    });
    for(int i = 1; i <= maxN+1; i++)
        best = max(best, x[i].pos - x[i-1].pos);
    sort(x, x+maxN+2, [](light A, light B){
        return A.id > B.id;
    });
    for(int i = 0; i < N; i++){
        ans[N-i-1] = best;
        c = S.find(x[i].pos);
        f = next(c, 1);
        b = next(c, -1);
        best = max(best, *f - *b);
        S.erase(c);
    }
    for(int i = 0; i < N; i++)
        printf("%d%c", ans[i], (" \n")[i==N]);
}
```

String Algorithms Ramez

Counting Patterns (count for each pattern the number of positions where it appears in the string)

Time limit: 1.00 s **Memory limit:** 512 MB

Given a string and patterns, count for each pattern the number of positions where it appears in the string.

Input

The first input line has a string of length n.

The next input line has an integer k: the number of patterns. Finally, there are k lines that describe the patterns.

The string and the patterns consist of characters a–z.

Output

For each pattern, print the number of positions.

Constraints

- $1 \le n \le 10^5$
- $1 \le k \le 5 \cdot 10^5$
- the total length of the patterns is at most $5 \cdot 10^5$

Example

Input:

```
aybabtu
3
bab
abc
a
```

Output:

```
1
0
2
```

Solution

```
const int maxN = 1e5+5;
struct Node {
    int len, link, cnt;
    map<char,int> nxt;
} node[2*maxN];
char S[maxN], T[5*maxN];
int N, M, K, sz, last;
void extend(char c){
    int cur = sz++;
    node[cur].cnt = 1;
    node[cur].len = node[last].len + 1;
    int p = last;
    while(p != -1 && !node[p].nxt.count(c)){
        node[p].nxt[c] = cur;
        p = node[p].link;
    }
    if(p == -1){
        node[cur].link = 0;
    } else {
        int q = node[p].nxt[c];
        if(node[p].len + 1 == node[q].len){
            node[cur].link = q;
        } else {
```

```
            int clone = sz++;
            node[clone].len = node[p].len + 1;
            node[clone].nxt = node[q].nxt;
            node[clone].link = node[q].link;
            while(p != -1 && node[p].nxt[c] == q){
                node[p].nxt[c] = clone;
                p = node[p].link;
            }
            node[q].link = node[cur].link = clone;
        }
    }
    last = cur;
}
void init(){
    node[0].len = 0;
    node[0].link = -1;
    sz = 1;
    last = 0;
}
void update_cnts(){
    vector<int> states_by_len[sz];
    for(int i = 0; i < sz; i++)
        states_by_len[node[i].len].push_back(i);
    for(int i = sz-1; i >= 0; i--)
        for(int u : states_by_len[i])
            if(node[u].link != -1)
                node[node[u].link].cnt += node[u].cnt;
}
int query(){
    int u = 0;
    for(int i = 0; i < M; i++){
        char c = T[i];
        if(!node[u].nxt.count(c))   return 0;
        else                        u = node[u].nxt[c];
    }
    return node[u].cnt;
}
int main(){
    scanf(" %s %d", S, &K);
    N = (int) strlen(S);
    init();
    for(int i = 0; i < N; i++)
        extend(S[i]);
    update_cnts();
    for(int i =  0; i < K; i++){
        scanf(" %s", T);
        M = (int) strlen(T);
        printf("%d\n", query());
    }
}
```

Distinct Subsequences (how many differnt strings you can get by removing any number of chars)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a string. You can remove any number of characters from it, but you cannot change the order of the

remaining characters. How many different strings can you generate?

Input

The first input line contains a string of size n. Each character is one of a–z.

Output

Print one integer: the number of strings modulo $10^9 + 7$.

Constraints

- $1 \le n \le 5 \cdot 10^5$

Example

Input:

```
aybabtu
```

Output:

```
103
```

Solution

```
const int maxN = 5e5+5;
const ll MOD = 1e9+7;
int N;
ll tot, dp[26];
char S[maxN];
int main(){
    scanf(" %s", S);
    N = (int) strlen(S);
    for(int i = 0; i < N; i++){
        int c = (int) (S[i] - 'a');
        dp[c]++;
        for(int j = 0; j < 26; j++)
            if(j != c)
                dp[c] = (dp[c] + dp[j]) % MOD;
    }
    for(int i = 0; i < 26; i++)
        tot = (tot + dp[i]) % MOD;
    printf("%lld\n", tot);
}
```

Distinct Substrings (count number of distinct substrings)

Time limit: 1.00 s **Memory limit:** 512 MB

Count the number of distinct substrings that appear in a string.

Input

The only input line has a string of length n that consists of characters a–z.

Output

Print one integer: the number of substrings.

Constraints

- $1 \leq n \leq 10^5$

Example

Input:

abaa

Output:

8

Explanation: the substrings are a, b, aa, ab, ba, aba, baa and abaa.

Solution

```
const int maxN = 1e5+5;
struct Node {
    ll dp;
    int len, link;
    map<char,int> nxt;
} node[2*maxN];
char S[maxN];
int N, sz, last;
void init(){
    node[0].len = 0;
    node[0].link = -1;
    sz = 1;
    last = 0;
}
void extend(char c){
    int cur = sz++;
    node[cur].len = node[last].len + 1;
    int p = last;
    while(p != -1 && !node[p].nxt.count(c)){
        node[p].nxt[c] = cur;
        p = node[p].link;
    }
    if(p == -1){
        node[cur].link = 0;
    } else {
        int q = node[p].nxt[c];
        if(node[p].len + 1 == node[q].len){
            node[cur].link = q;
        } else {
            int clone = sz++;
            node[clone].len = node[p].len + 1;
            node[clone].nxt = node[q].nxt;
            node[clone].link = node[q].link;
            while(p != -1 && node[p].nxt[c] == q){
                node[p].nxt[c] = clone;
                p = node[p].link;
            }
            node[q].link = node[cur].link = clone;
        }
    }
    last = cur;
}
```

```
void calc(int u = 0){
    node[u].dp = 1;
    for(const auto& [c, v] : node[u].nxt){
        if(!node[v].dp) calc(v);
        node[u].dp += node[v].dp;
    }
}
int main(){
    scanf(" %s", S);
    N = (int) strlen(S);
    init();
    for(int i = 0; i < N; i++)
        extend(S[i]);
    calc();
    printf("%lld\n", node[0].dp-1);
}
```

Finding Borders (find number of prefixes = suffexies)

Time limit: 1.00 s **Memory limit:** 512 MB

A border of a string is a prefix that is also a suffix of the string but not the whole string. For example, the borders of abcababcab are ab and abcab. Your task is to find all border lengths of a given string.

Input

The only input line has a string of length n consisting of characters a–z.

Output

Print all border lengths of the string in increasing order.

Constraints

- $1 \le n \le 10^6$

Example

Input:

```
abcababcab
```

Output:

```
2 5
```

Solution

```
const ll MOD = 1e9+7;
const ll p1 = 31;
const ll p2 = 37;
const int maxN = 1e6+5;
int N;
ll pow1[maxN], pow2[maxN], ph1, ph2, sh1, sh2;
char S[maxN];
int main(){
    scanf("%s", S);
    N = (int) strlen(S);
    pow1[0] = pow2[0] = 1;
    for(int i = 1; i < N; i++){
```

```
        pow1[i] = (pow1[i-1] * p1) % MOD;
        pow2[i] = (pow2[i-1] * p2) % MOD;
    }
    for(int i = 0; i < N-1; i++){
        int l = (S[i] - 'a' + 1);
        int r = (S[N-i-1] - 'a' + 1);
        ph1 = (ph1 + l * pow1[i]) % MOD;
        ph2 = (ph2 + l * pow2[i]) % MOD;
        sh1 = (sh1 * p1 + r) % MOD;
        sh2 = (sh2 * p2 + r) % MOD;
        if(ph1 == sh1 && ph2 == sh2)
            printf("%d ", i+1);
    }
}
```

Finding Patterns (check if patterns appear in string)

Time limit: 1.00 s **Memory limit:** 512 MB

Given a string and patterns, check for each pattern if it appears in the string.

Input

The first input line has a string of length n.

The next input line has an integer k: the number of patterns. Finally, there are k lines that describe the patterns.

The string and the patterns consist of characters a–z.

Output

For each pattern, print "YES" if it appears in the string and "NO" otherwise.

Constraints

- $1 \le n \le 10^5$
- $1 \le k \le 5 \cdot 10^5$
- the total length of the patterns is at most $5 \cdot 10^5$

Example

Input:

```
aybabtu
3
bab
abc
ayba
```

Output:

```
YES
NO
YES
```

Solution

```
const int maxN = 1e5+5;
struct Node {
    int len, link, cnt, firstpos;
    map<char,int> nxt;
```

```
} node[2*maxN];
char S[maxN], T[5*maxN];
int N, M, K, sz, last;
void extend(char c){
    int cur = sz++;
    node[cur].len = node[last].len + 1;
    int p = last;
    while(p != -1 && !node[p].nxt.count(c)){
        node[p].nxt[c] = cur;
        p = node[p].link;
    }
    if(p == -1){
        node[cur].link = 0;
    } else {
        int q = node[p].nxt[c];
        if(node[p].len + 1 == node[q].len){
            node[cur].link = q;
        } else {
            int clone = sz++;
            node[clone].len = node[p].len + 1;
            node[clone].nxt = node[q].nxt;
            node[clone].link = node[q].link;
            while(p != -1 && node[p].nxt[c] == q){
                node[p].nxt[c] = clone;
                p = node[p].link;
            }
            node[q].link = node[cur].link = clone;
        }
    }
    last = cur;
}
void init(){
    node[0].len = 0;
    node[0].link = -1;
    sz = 1;
    last = 0;
}
bool query(){
    int u = 0;
    for(int i = 0; i < M; i++){
        char c = T[i];
        if(!node[u].nxt.count(c))   return false;
        else                        u = node[u].nxt[c];
    }
    return true;
}
int main(){
    scanf(" %s %d", S, &K);
    N = (int) strlen(S);
    init();
    for(int i = 0; i < N; i++)
        extend(S[i]);
    for(int i =  0; i < K; i++){
        scanf(" %s", T);
        M = (int) strlen(T);
        printf("%s\n", query() ? "YES" : "NO");
```

```
    }
}
```

Finding Periods (find all prefixes that can generate the whole string by repeat)

Time limit: 1.00 s **Memory limit:** 512 MB

A period of a string is a prefix that can be used to generate the whole string by repeating the prefix. The last repetition may be partial. For example, the periods of abcabca are abc, abcabc and abcabca. Your task is to find all period lengths of a string.

Input

The only input line has a string of length n consisting of characters a–z.

Output

Print all period lengths in increasing order.

Constraints

- $1 \le n \le 10^6$

Example

Input:

```
abcabca
```

Output:

```
3 6 7
```

Solution

```
const ll MOD = 1e9+7;
const ll p1 = 31, p2 = 37;
const int maxN = 1e6+5;
int N;
char S[maxN];
ll pow1[maxN], pow2[maxN], h1[maxN], h2[maxN];
void init(){
    pow1[0] = pow2[0] = 1;
    h1[0] = h2[0] = (int) (S[0] - 'a' + 1);
    for(int i = 1; i < maxN; i++){
        int c = (int) (S[i] - 'a' + 1);
        pow1[i] = (pow1[i-1] * p1) % MOD;
        pow2[i] = (pow2[i-1] * p2) % MOD;
        h1[i] = (h1[i-1] * p1 + c) % MOD;
        h2[i] = (h2[i-1] * p2 + c) % MOD;
    }
}
int main(){
    scanf(" %s", S);
    N = (int) strlen(S);
    init();
    for(int len = 1; len <= N; len++){
        ll hash1 = h1[len-1], hash2 = h2[len-1];
        for(int i = 0; i < (N/len)-1; i++){
            hash1 = (hash1 * pow1[len] + h1[len-1]) % MOD;
```

```
            hash2 = (hash2 * pow2[len] + h2[len-1]) % MOD;
        }
        hash1 = (hash1 * pow1[N%len] + h1[N%len-1]) % MOD;
        hash2 = (hash2 * pow2[N%len] + h2[N%len-1]) % MOD;
        if(hash1 == h1[N-1] && hash2 == h2[N-1])    printf("%d ", len);
    }
}
```

Longest Palindrome (Determine the longest palindromic substring of the string)

Time limit: 1.00 s **Memory limit:** 512 MB

Given a string, your task is to determine the longest palindromic substring of the string. For example, the longest palindrome in aybabtu is bab.

Input

The only input line contains a string of length n. Each character is one of a–z.

Output

Print the longest palindrome in the string. If there are several solutions, you may print any of them.

Constraints

- $1 \le n \le 10^6$

Example

Input:

```
aybabtu
```

Output:

```
bab
```

Solution

```
const int maxN = 1e6+5;
char S[maxN];
int N, bestl, bestr, d1[maxN], d2[maxN];
int main(){
    scanf(" %s", S);
    N = (int) strlen(S);
    for(int i = 0, l = 0, r = -1; i < N; i++){
        int k = (i > r ? 1 : min(d1[l+r-i], r-i+1));
        while(0 <= i-k && i+k < N && S[i-k] == S[i+k])  k++;
        if(2*k-1 > bestr-bestl+1){
            bestl = i-k+1;
            bestr = i+k-1;
        }
        d1[i] = k--;
        if(i+k > r){
            l = i-k;
            r = i+k;
        }
    }
    for(int i = 0, l = 0, r = -1; i < N; i++){
        int k = (i > r ? 0 : min(d2[l+r-i+1], r-i+1));
```

```
        while(0 <= i-k-1 && i+k < N && S[i-k-1] == S[i+k])  k++;
        if(2*k > bestr-bestl+1){
            bestl = i-k;
            bestr = i+k-1;
        }
        d2[i] = k--;
        if(i+k > r){
            l = i-k-1;
            r = i+k;
        }
    }
    for(int i = bestl; i <= bestr; i++)
        printf("%c", S[i]);
}
```

Minimal Rotation (Determine the lexicographically minimal rotation of a string.)

Time limit: 1.00 s **Memory limit:** 512 MB

A rotation of a string can be generated by moving characters one after another from beginning to end. For example, the rotations of acab are acab, caba, abac, and baca. Your task is to determine the lexicographically minimal rotation of a string.

Input

The only input line contains a string of length n. Each character is one of a–z.

Output

Print the lexicographically minimal rotation.

Constraints

- $1 \le n \le 10^6$

Example

Input:

```
acab
```

Output:

```
abac
```

Solution

```
const int maxN = 2e6+5;
int N, F[maxN];
char S[maxN];
int main(){
    scanf("%s", S);
    N = (int) strlen(S);
    for(int i = 0; i < N; i++)
        S[i+N] = S[i];
    N = (int) strlen(S);
    fill(F, F+N, -1);
    int k = 0;
    for(int i = 1; i < N; i++){
        char c = S[i];
```

```
        int f = F[i-k-1];
        while(f != -1 && c != S[k+f+1]){
            if(c < S[k+f+1])
                k = i-f-1;
            f = F[f];
        }
        if(c != S[k+f+1]){
            if(c < S[k])
                k = i;
            F[i-k] = -1;
        } else F[i-k] = f+1;
    }
    for(int i = 0; i < N/2; i++)
        printf("%c", S[i+k]);
}
```

Palindrome Queries (update character and check if a substring is palindrome)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a string that consists of n characters between a–z. The positions of the string are indexed $1, 2, \ldots, n$. Your task is to process m operations of the following types: - Change the character at position k to x - Check if the substring from position a to position b is a palindrome

Input

The first input line has two integers n and m: the length of the string and the number of operations.

The next line has a string that consists of n characters.

Finally, there are m lines that describe the operations. Each line is of the form "1 k x" or "2 a b".

Output

For each operation 2, print YES if the substring is a palindrome and NO otherwise.

Constraints

- $1 \le n, m \le 2 \cdot 10^5$
- $1 \le k \le n$
- $1 \le a \le b \le n$

Example

Input:

```
7 5
aybabtu
2 3 5
1 3 x
2 3 5
1 5 x
2 3 5
```

Output:

```
YES
NO
YES
```

Solution

```
const ll MOD = 1e9+7;
const ll p[2] = {31, 37};
const int maxN = 2e5+5;
char S[maxN];
int N, K;
ll pows[2][maxN], invs[2][maxN], F[2], B[2], ds[2][2][maxN];
ll inverse(ll x){
    ll res = 1;
    ll b = MOD-2;
    while(b){
        if(b&1) res = (res * x) % MOD;
        x = (x * x) % MOD;
        b >>= 1;
    }
    return res;
}
void update(int pt, int t, int idx, ll val){
    val %= MOD; val += MOD; val %= MOD;
    for(int i = idx+1; i < maxN; i += -i&i)
        ds[pt][t][i] = (ds[pt][t][i] + val) % MOD;
}
ll query(int pt, int t, int idx){
    if(idx < 0) return 0;
    ll sum = 0;
    for(int i = idx+1; i > 0; i -= -i&i)
        sum = (sum + ds[pt][t][i]) % MOD;
    return sum;
}
void init(){
    for(int t = 0; t < 2; t++){
        pows[t][0] = invs[t][0] = 1;
        for(int i = 1; i < maxN; i++){
            pows[t][i] = (pows[t][i-1] * p[t]) % MOD;
            invs[t][i] = inverse(pows[t][i]);
        }
        update(t, 0, 0, (int) (S[0]-'a'+1) * pows[t][N-1]);
        update(t, 1, 0, (int) (S[0]-'a'+1));
        for(int i = 1; i < N; i++){
            int c = (int) (S[i]-'a'+1);
            update(t, 0, i, c * pows[t][N-i-1]);
            update(t, 1, i, c * pows[t][i]);
        }
    }
}
void modify(int a, char c){
    int newchar = (int) (c - 'a' + 1);
    int oldchar = (int) (S[a] - 'a' + 1);
    for(int t = 0; t < 2; t++){
        update(t, 0, a, -oldchar * pows[t][N-a-1]);
        update(t, 0, a, newchar * pows[t][N-a-1]);
        update(t, 1, a, -oldchar * pows[t][a]);
        update(t, 1, a, newchar * pows[t][a]);
    }
    S[a] = c;
}
```

```
bool palindrome(int a, int b){
    for(int t = 0; t < 2; t++){
        F[t] = (query(t, 0, b) - query(t, 0, a-1) + MOD) % MOD;
        F[t] = (F[t] * invs[t][N-b-1]) % MOD;
        B[t] = (query(t, 1, b) - query(t, 1, a-1) + MOD) % MOD;
        B[t] = (B[t] * invs[t][a]) % MOD;
    }
    return F[0] == B[0] && F[1] == B[1];
}
int main(){
    scanf("%d %d %s", &N, &K, S);
    init();
    for(int i = 0, t, a, b; i < K; i++){
        char c;
        scanf("%d", &t);
        if(t == 1){
            scanf("%d %c", &a, &c);
            modify(a-1, c);
        } else if(t == 2){
            scanf("%d %d", &a, &b);
            printf("%s\n", palindrome(a-1, b-1) ? "YES" : "NO");
        }
    }
}
```

Pattern Positions (find first index of pattern in string)

Time limit: 1.00 s **Memory limit:** 512 MB

Given a string and patterns, find for each pattern the first position (1-indexed) where it appears in the string.

Input

The first input line has a string of length n.

The next input line has an integer k: the number of patterns. Finally, there are k lines that describe the patterns.

The string and the patterns consist of characters a–z.

Output

Print the first position for each pattern (or -1 if it does not appear at all).

Constraints

- $1 \le n \le 10^5$
- $1 \le k \le 5 \cdot 10^5$
- the total length of the patterns is at most $5 \cdot 10^5$

Example

Input:

```
aybabtu
3
bab
abc
a
```

Output:

```
3
-1
1
```

Solution

```
const int maxN = 1e5+5;
struct Node {
    int len, link, cnt, firstpos;
    map<char,int> nxt;
} node[2*maxN];
char S[maxN], T[5*maxN];
int N, M, K, sz, last;
void extend(char c){
    int cur = sz++;
    node[cur].cnt = 1;
    node[cur].firstpos = node[last].len;
    node[cur].len = node[last].len + 1;
    int p = last;
    while(p != -1 && !node[p].nxt.count(c)){
        node[p].nxt[c] = cur;
        p = node[p].link;
    }
    if(p == -1){
        node[cur].link = 0;
    } else {
        int q = node[p].nxt[c];
        if(node[p].len + 1 == node[q].len){
            node[cur].link = q;
        } else {
            int clone = sz++;
            node[clone].len = node[p].len + 1;
            node[clone].nxt = node[q].nxt;
            node[clone].link = node[q].link;
            node[clone].firstpos = node[q].firstpos;
            while(p != -1 && node[p].nxt[c] == q){
                node[p].nxt[c] = clone;
                p = node[p].link;
            }
            node[q].link = node[cur].link = clone;
        }
    }
    last = cur;
}
void init(){
    node[0].len = 0;
    node[0].link = -1;
    sz = 1;
    last = 0;
}
void update_cnts(){
    vector<int> states_by_len[sz];
    for(int i = 0; i < sz; i++)
        states_by_len[node[i].len].push_back(i);
    for(int i = sz-1; i >= 0; i--)
        for(int u : states_by_len[i])
            if(node[u].link != -1)
```

```
                node[node[u].link].cnt += node[u].cnt;
}
int query_cnt(){
    int u = 0;
    for(int i = 0; i < M; i++){
        char c = T[i];
        if(!node[u].nxt.count(c))   return 0;
        else                        u = node[u].nxt[c];
    }
    return node[u].cnt;
}
int query_idx(){
    int u = 0;
    for(int i = 0; i < M; i++){
        char c = T[i];
        if(!node[u].nxt.count(c))   return -1;
        else                        u = node[u].nxt[c];
    }
    return node[u].firstpos-M+2;
}
int main(){
    scanf(" %s %d", S, &K);
    N = (int) strlen(S);
    init();
    for(int i = 0; i < N; i++)
        extend(S[i]);
    update_cnts();
    for(int i =  0; i < K; i++){
        scanf(" %s", T);
        M = (int) strlen(T);
        printf("%d\n", query_idx());
    }
}
```

Repeating Substring (find the longest repeating substring in a given string.)

Time limit: 1.00 s **Memory limit:** 512 MB

A repeating substring is a substring that occurs in two (or more) locations in the string. Your task is to find the longest repeating substring in a given string.

Input

The only input line has a string of length n that consists of characters a–z.

Output

Print the longest repeating substring. If there are several possibilities, you can print any of them. If there is no repeating substring, print -1.

Constraints

- $1 \le n \le 10^5$

Example

Input:

```
cababababc
```

Output:

abab

Solution

```
const int maxN = 1e5+5;
struct Node {
    int len, link, cnt, firstpos;
    map<char,int> nxt;
} node[2*maxN];
char S[maxN];
bool vis[2*maxN];
int N, sz, last, bestidx, bestlen;
void extend(char c){
    int cur = sz++;
    node[cur].cnt = 1;
    node[cur].firstpos = node[last].len;
    node[cur].len = node[last].len + 1;
    int p = last;
    while(p != -1 && !node[p].nxt.count(c)){
        node[p].nxt[c] = cur;
        p = node[p].link;
    }
    if(p == -1){
        node[cur].link = 0;
    } else {
        int q = node[p].nxt[c];
        if(node[p].len + 1 == node[q].len){
            node[cur].link = q;
        } else {
            int clone = sz++;
            node[clone].len = node[p].len + 1;
            node[clone].nxt = node[q].nxt;
            node[clone].link = node[q].link;
            node[clone].firstpos = node[q].firstpos;
            while(p != -1 && node[p].nxt[c] == q){
                node[p].nxt[c] = clone;
                p = node[p].link;
            }
            node[q].link = node[cur].link = clone;
        }
    }
    last = cur;
}
void init(){
    node[0].len = 0;
    node[0].link = -1;
    sz = 1;
    last = 0;
}
void update_cnts(){
    vector<int> states_by_len[sz];
    for(int i = 0; i < sz; i++)
        states_by_len[node[i].len].push_back(i);
    for(int i = sz-1; i >= 0; i--)
        for(int u : states_by_len[i])
```

```
            if(node[u].link != -1)
                node[node[u].link].cnt += node[u].cnt;
}
void dfs(int u = 0){
    vis[u] = true;
    if(node[u].len > bestlen && node[u].cnt > 1 && u != 0){
        bestidx = node[u].firstpos - node[u].len + 1;
        bestlen = node[u].len;
    }
    for(const auto& [c, v] : node[u].nxt)
        if(!vis[v])
            dfs(v);
}
int main(){
    scanf(" %s", S);
    N = (int) strlen(S);
    init();
    for(int i = 0; i < N; i++)
        extend(S[i]);
    update_cnts();
    bestlen = -1;
    dfs();
    if(bestlen == -1)   printf("-1\n");
    else {
        for(int i = 0; i < bestlen; i++)
            printf("%c", S[bestidx+i]);
    }
}
```

Required Substring (number of strings of length n having a given pattern of length m as their substring.)

Time limit: 1.00 s **Memory limit:** 512 MB

Your task is to calculate the number of strings of length n having a given pattern of length m as their substring. All strings consist of characters A–Z.

Input

The first input line has an integer n: the length of the final string.

The second line has a pattern of length m.

Output

Print the number of strings modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 1000$
- $1 \leq m \leq 100$

Example

Input:

```
6
ABCDB
```

Output:

52

Explanation: The final string will be of the form ABCDBx or xABCDB where x is any character between A–Z.

Solution

```
const int maxN = 1005, maxM = 505;
const ll MOD = 1e9+7;
int N, M, best[26][maxM];
char S[maxN];
ll ans, dp[maxM][maxN];
ll pow26(ll b){
    ll a = 26;
    ll res = 1;
    while(b){
        if(b&1) res = (res * a) % MOD;
        a = (a * a) % MOD;
        b >>= 1;
    }
    return res;
}
bool good(vector<char> pre, int k){
    int SZ = (int) pre.size();
    for(int i = 0; i < SZ-k; i++)
        if(pre[i+k] != S[i])
            return false;
    return true;
}
int main(){
    scanf("%d %s", &N, S);
    M = (int) strlen(S);
    if(M > N){
        printf("0\n");
        return 0;
    }
    for(int r = 0; r < M; r++){
        for(int c = 0; c < 26; c++){
            vector<char> pre;
            for(int i = 0; i < r; i++)
                pre.push_back(S[i]);
            pre.push_back((char) (c+'A'));
            for(int k = 0; k < r+1; k++){
                if(good(pre, k)){
                    best[c][r] = r-k+1;
                    break;
                }
            }
        }
    }
    dp[0][0] = 1;
    for(int i = 1; i <= N; i++)
        for(int j = 0; j < M; j++)
            for(int c = 0; c < 26; c++)
                dp[best[c][j]][i] = (dp[best[c][j]][i] + dp[j][i-1]) % MOD;
    ans = pow26(N);
    for(int i = 0; i < M; i++)
        ans = (ans - dp[i][N] + MOD) % MOD;
```

```
    printf("%lld\n", ans);
}
```

String Functions (the maximum length of a substring that begins at position i and is a prefix of the string, and optionally whose length is at most $i-1$)

Time limit: 1.00 s **Memory limit:** 512 MB

We consider a string of n characters, indexed $1, 2, \ldots, n$. Your task is to calculate all values of the following functions: - $z(i)$ denotes the maximum length of a substring that begins at position i and is a prefix of the string. In addition, $z(1) = 0$. - $\pi(i)$ denotes the maximum length of a substring that ends at position i, is a prefix of the string, and whose length is at most $i-1$. Note that the function z is used in the Z-algorithm, and the function π is used in the KMP algorithm.

Input

The only input line has a string of length n. Each character is between a–z.

Output

Print two lines: first the values of the z function, and then the values of the π function.

Constraints

- $1 \le n \le 10^6$

Example

Input:

```
abaabca
```

Output:

```
0 0 1 2 0 0 1
0 0 1 1 2 0 1
```

Solution

```
const int maxN = 1e6+5;
char S[maxN];
int N, z[maxN], pi[maxN];
int main(){
    scanf(" %s", S);
    N = (int) strlen(S);
    for(int i = 1, l = 0, r = 0; i < N; i++){
        if(i <= r)  z[i] = min(r-i+1, z[i-l]);
        while(i+z[i] < N && S[z[i]] == S[i+z[i]])   z[i]++;
        if(i+z[i]-1 > r)    l = i, r = i+z[i]-1;
    }
    for(int i = 0; i < N; i++)
        printf("%d%c", z[i], (" \n")[i==N-1]);
    for(int i = 1; i < N; i++){
        int j = pi[i-1];
        while(j > 0 && S[i] != S[j])    j = pi[j-1];
        if(S[i] == S[j])    j++;
        pi[i] = j;
    }
    for(int i = 0; i < N; i++)
```

```
        printf("%d%c", pi[i], (" \n")[i==N-1]);
}
```

String Matching (count the number of positions where the pattern occurs in the string.)

Time limit: 1.00 s **Memory limit:** 512 MB

Given a string and a pattern, your task is to count the number of positions where the pattern occurs in the string.

Input

The first input line has a string of length n, and the second input line has a pattern of length m. Both of them consist of characters a–z.

Output

Print one integer: the number of occurrences.

Constraints

- $1 \le n, m \le 10^6$

Example

Input:

```
saippuakauppias
pp
```

Output:

```
2
```

Solution

```
const int maxN = 1e6+5, numP = 3;
const ll MOD = 1e9+7;
const ll prime[numP] = {31, 37, 43};
int N, M, cnt;
ll sh[numP], th[numP], pw[numP][maxN];
char S[maxN], T[maxN];
bool hashes_equal(){
    for(int p = 0; p < numP; p++)
        if(sh[p] != th[p])
            return false;
    return true;
}
int main(){
    scanf("%s %s", S, T);
    N = (int) strlen(S); M = (int) strlen(T);
    if(M > N){
        printf("0\n");
        return 0;
    }
    for(int p = 0; p < numP; p++){
        pw[p][0] = 1;
        for(int i = 1; i < maxN; i++)
            pw[p][i] = (pw[p][i-1] * prime[p]) % MOD;
    }
```

```
    for(int i = 0; i < M; i++){
        for(int p = 0; p < numP; p++){
            sh[p] = (sh[p] + (S[i] - 'a' + 1) * pw[p][M-i-1]) % MOD;
            th[p] = (th[p] + (T[i] - 'a' + 1) * pw[p][M-i-1]) % MOD;
        }
    }
    if(hashes_equal())  cnt++;
    for(int i = M; i < N; i++){
        for(int p = 0; p < numP; p++){
            sh[p] = (sh[p] * prime[p] - (S[i-M] - 'a' + 1) * pw[p][M]) % MOD;
            sh[p] = (sh[p] + (S[i] - 'a' + 1) + MOD) % MOD;
        }
        if(hashes_equal())  cnt++;
    }
    printf("%d\n", cnt);
}
```

String Transform (generating the string from last character of all the possible rotations of a string)

Time limit: 1.00 s **Memory limit:** 512 MB

Consider the following string transformation: - append the character # to the string (we assume that # is lexicographically smaller than all other characters of the string) - generate all rotations of the string - sort the rotations in increasing order - based on this order, construct a new string that contains the last character of each rotation For example, the string babc becomes babc#. Then, the sorted list of rotations is #babc, abc#b, babc#, bc#ba, and c#bab. This yields a string cb#ab.

Input

The only input line contains the transformed string of length $n + 1$. Each character of the original string is one of a–z.

Output

Print the original string of length n.

Constraints

- $1 \leq n \leq 10^6$

Example

Input:

```
cb#ab
```

Output:

```
babc
```

Solution

```
const int maxN = 1e6+5;
int N, nxt[maxN];
char S[maxN];
int main(){
    scanf(" %s", S);
    N = (int) strlen(S);
    for(int i = 0; i < N; i++)  nxt[i] = i;
```

```
    stable_sort(nxt, nxt+N, [](int a, int b){
        return S[a] < S[b];
    });
    int ptr = nxt[0];
    while(ptr != 0){
        ptr = nxt[ptr];
        printf("%c", S[ptr]);
    }
}
```

Substring Distribution (for every integer from 1 to n print the number of distinct substring of that length)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a string of length n. For every integer between $1 \ldots n$ you need to print the number of distinct substrings of that length.

Input

The only input line has a string of length n that consists of characters a–z.

Output

For each integer between $1 \ldots n$ print the number of distinct substrings of that length.

Constraints

- $1 \leq n \leq 10^5$

Example

Input:

```
abab
```

Output:

```
2 2 2 1
```

Solution

```
const int maxN = 1e5+5;
struct Node {
    int len, link;
    map<char,int> nxt;
} node[2*maxN];
char S[maxN];
bool vis[2*maxN];
int N, sz, last, dist[2*maxN];
ll ans[maxN];
void extend(char c){
    int cur = sz++;
    node[cur].len = node[last].len + 1;
    int p = last;
    while(p != -1 && !node[p].nxt.count(c)){
        node[p].nxt[c] = cur;
        p = node[p].link;
    }
    if(p == -1){
```

```
        node[cur].link = 0;
    } else {
        int q = node[p].nxt[c];
        if(node[p].len + 1 == node[q].len){
            node[cur].link = q;
        } else {
            int clone = sz++;
            node[clone].len = node[p].len + 1;
            node[clone].nxt = node[q].nxt;
            node[clone].link = node[q].link;
            while(p != -1 && node[p].nxt[c] == q){
                node[p].nxt[c] = clone;
                p = node[p].link;
            }
            node[q].link = node[cur].link = clone;
        }
    }
    last = cur;
}
void init(){
    node[0].len = 0;
    node[0].link = -1;
    sz = 1;
    last = 0;
}
void bfs(int s = 0){
    queue<int> Q;
    vis[s] = true;
    dist[s] = 0;
    Q.push(s);
    while(!Q.empty()){
        int u = Q.front();
        Q.pop();
        ans[dist[u]]++;
        ans[node[u].len+1]--;
        for(const auto& [c, v] : node[u].nxt){
            if(!vis[v]){
                dist[v] = dist[u]+1;
                vis[v] = true;
                Q.push(v);
            }
        }
    }
}
int main(){
    scanf(" %s", S);
    N = (int) strlen(S);
    init();
    for(int i = 0; i < N; i++)
        extend(S[i]);
    bfs();
    for(int i = 1; i <= N; i++){
        ans[i] += ans[i-1];
        printf("%lld%c", ans[i], (" \n")[i==N]);
    }
}
```

Substring Order I (kth smallest distinct substring)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a string of length n. If all of its distinct substrings are ordered lexicographically, what is the k - th smallest of them?

Input

The first input line has a string of length n that consists of characters a–z.

The second input line has an integer k.

Output

Print the kth smallest distinct substring in lexicographical order.

Constraints

- $1 \le n \le 10^5$
- $1 \le k \le \frac{n(n+1)}{2}$
- It is guaranteed that k does not exceed the number of distinct substrings.

Example

Input:

```
babaacbaab
10
```

Output:

```
aba
```

Explanation: The 10 smallest distinct substrings in order are a, aa, aab, aac, aacb, aacba, aacbaa, aacbaab, ab, and aba.

Solution

```
const int maxN = 1e5+5;
struct Node {
    ll dp;
    int len, link;
    map<char,int> nxt;
} node[2*maxN];
vector<char> ans;
char S[maxN];
int N, sz, last;
ll K;
void init(){
    node[0].len = 0;
    node[0].link = -1;
    sz = 1;
    last = 0;
}
void extend(char c){
    int cur = sz++;
    node[cur].len = node[last].len + 1;
    int p = last;
    while(p != -1 && !node[p].nxt.count(c)){
        node[p].nxt[c] = cur;
```

```
        p = node[p].link;
    }
    if(p == -1){
        node[cur].link = 0;
    } else {
        int q = node[p].nxt[c];
        if(node[p].len + 1 == node[q].len){
            node[cur].link = q;
        } else {
            int clone = sz++;
            node[clone].len = node[p].len + 1;
            node[clone].nxt = node[q].nxt;
            node[clone].link = node[q].link;
            while(p != -1 && node[p].nxt[c] == q){
                node[p].nxt[c] = clone;
                p = node[p].link;
            }
            node[q].link = node[cur].link = clone;
        }
    }
    last = cur;
}
void calc(int u = 0){
    node[u].dp = 1;
    for(const auto& [c, v] : node[u].nxt){
        if(!node[v].dp) calc(v);
        node[u].dp += node[v].dp;
    }
}
void dfs(int u, ll k){
    if(k < 0)   return;
    for(const auto& [c, v] : node[u].nxt){
        if(node[v].dp <= k) k -= node[v].dp;
        else {
            ans.push_back(c);
            dfs(v, k-1);
            return;
        }
    }
}
int main(){
    scanf(" %s %lld", S, &K);
    N = (int) strlen(S);
    init();
    for(int i = 0; i < N; i++)
        extend(S[i]);
    calc();
    dfs(0, K-1);
    int M = (int) ans.size();
    for(int i = 0; i < M; i++)
        printf("%c", ans[i]);
    printf("\n");
}
```

Substring Order II (kth smallest not distinct substring)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a string of length n. If all of its substrings (not necessarily distinct) are ordered lexicographically, what is the kth smallest of them?

Input

The first input line has a string of length n that consists of characters a–z.

The second input line has an integer k.

Output

Print the kth smallest substring in lexicographical order.

Constraints

- $1 \le n \le 10^5$
- $1 \le k \le \frac{n(n+1)}{2}$

Example

Input:

```
baabaa
10
```

Output:

```
ab
```

Explanation: The 10 smallest substrings in order are a, a, a, a, aa, aa, aab, aaba, aabaa, and ab.

Solution

```
const int maxN = 1e5+5;
struct Node {
    ll dp;
    int len, cnt, link;
    map<char,int> nxt;
} node[2*maxN];
vector<char> ans;
char S[maxN];
int N, sz, last;
ll K;
void init(){
    node[0].len = 0;
    node[0].link = -1;
    sz = 1;
    last = 0;
}
void extend(char c){
    int cur = sz++;
    node[cur].cnt = 1;
    node[cur].len = node[last].len + 1;
    int p = last;
    while(p != -1 && !node[p].nxt.count(c)){
        node[p].nxt[c] = cur;
        p = node[p].link;
    }
    if(p == -1){
        node[cur].link = 0;
    } else {
```

```
        int q = node[p].nxt[c];
        if(node[p].len + 1 == node[q].len){
            node[cur].link = q;
        } else {
            int clone = sz++;
            node[clone].len = node[p].len + 1;
            node[clone].nxt = node[q].nxt;
            node[clone].link = node[q].link;
            while(p != -1 && node[p].nxt[c] == q){
                node[p].nxt[c] = clone;
                p = node[p].link;
            }
            node[q].link = node[cur].link = clone;
        }
    }
    last = cur;
}
void update_cnts(){
    vector<int> states_by_len[sz];
    for(int i = 0; i < sz; i++)
        states_by_len[node[i].len].push_back(i);
    for(int i = sz-1; i >= 0; i--)
        for(int u : states_by_len[i])
            if(node[u].link != -1)
                node[node[u].link].cnt += node[u].cnt;
}
void calc(int u = 0){
    node[u].dp = node[u].cnt;
    for(const auto& [c, v] : node[u].nxt){
        if(!node[v].dp) calc(v);
        node[u].dp += node[v].dp;
    }
}
void dfs(int u, ll k){
    if(k < 0)   return;
    for(const auto& [c, v] : node[u].nxt){
        if(node[v].dp <= k) k -= node[v].dp;
        else {
            ans.push_back(c);
            dfs(v, k-node[v].cnt);
            return;
        }
    }
}
int main(){
    scanf(" %s %lld", S, &K);
    N = (int) strlen(S);
    init();
    for(int i = 0; i < N; i++)
        extend(S[i]);
    update_cnts();
    calc();
    dfs(0, K-1);
    int M = (int) ans.size();
    for(int i = 0; i < M; i++)
        printf("%c", ans[i]);
```

```
    printf("\n");
}
```

Word Combinations (How many ways can you create a string from a collection of words)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a string of length n and a dictionary containing k words. In how many ways can you create the string using the words?

Input

The first input line has a string containing n characters between a–z.

The second line has an integer k: the number of words in the dictionary.

Finally there are k lines describing the words. Each word is unique and consists of characters a–z.

Output

Print the number of ways modulo $10^9 + 7$.

Constraints

- $1 \le n \le 5000$
- $1 \le k \le 10^5$
- the total length of the words is at most 10^6

Example

Input:

```
ababc
4
ab
abab
c
cb
```

Output:

```
2
```

Explanation: The possible ways are ab+ab+c and abab+c.

Solution

```
const int maxN = 5005, K = 26;
const ll MOD = 1e9+7;
struct Node {
    int next[K];
    bool leaf = false;
    Node(){ fill(next, next+K, -1); }
};
int N, M;
ll dp[maxN];
char S[maxN], word[maxN];
vector<Node> trie(1);
void add_word(){
    scanf(" %s", word);
```

```
    int v = 0;
    M = (int) strlen(word);
    for(int i = 0; i < M; i++){
        int c = (int) (word[i] - 'a');
        if(trie[v].next[c] == -1){
            trie[v].next[c] = trie.size();
            trie.emplace_back();
        }
        v = trie[v].next[c];
    }
    trie[v].leaf = true;
}
int main(){
    scanf(" %s %d", S, &N);
    for(int i = 0; i < N; i++)
        add_word();
    M = (int) strlen(S);
    dp[M] = 1;
    for(int i = M-1; i >= 0; i--){
        int v = 0;
        for(int j = i; j < M; j++){
            int c = (int) (S[j] - 'a');
            if(trie[v].next[c] == -1)   break;
            v = trie[v].next[c];
            if(trie[v].leaf)
                dp[i] = (dp[i] + dp[j+1]) % MOD;
        }
    }
    printf("%lld\n", dp[0]);
}
```

Tree Algorithms Ramez

Company Queries I (find the k-th ancestor)

Time limit: 1.00 s **Memory limit:** 512 MB

A company has n employees, who form a tree hierarchy where each employee has a boss, except for the general director. Your task is to process q queries of the form: who is employee x's boss k levels higher up in the hierarchy?

Input

The first input line has two integers n and q: the number of employees and queries. The employees are numbered $1, 2, \ldots, n$, and employee 1 is the general director.

The next line has $n-1$ integers $e_2, e_3, \ldots, e_n$: for each employee $2, 3, \ldots, n$ their boss.

Finally, there are q lines describing the queries. Each line has two integers x and k: who is employee x's boss k levels higher up?

Output

Print the answer for each query. If such a boss does not exist, print -1.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le e_i \le i - 1$
- $1 \le x \le n$

- $1 \leq k \leq n$

Example

Input:

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

Output:

```
3
1
-1
```

Solution

```
class Tree {
private:
    int MAXPOW;
    vector<vi> up; // up[n][k] = 2^k-th ancestor of node n

public:
    Tree(int n, const vi &parents) {
        MAXPOW = ceil(log2(n));
        up.assign(n+1, vi(MAXPOW + 1, -1));

        for (int v = 2; v <= n; ++v) {
            up[v][0] = parents[v - 2];
        }

        for (int k = 1; k <= MAXPOW; ++k) {
            for (int v = 1; v <= n; ++v) {
                int m = up[v][k-1];
                if (m != -1) {
                    up[v][k] = up[m][k-1];
                }
            }
        }
    }

    int kth_parent(int n, int k) const {
        for (int bit = 0; bit <= MAXPOW && n != -1; ++bit) {
            if (k & (1 << bit)) {
                n = up[n][bit];
            }
        }
        return n;
    }
};


void Ramez() {
    int n, q; cin >> n >> q;
    vi parents(n - 1); cin >> parents;
```

```
    Tree t(n, parents);

    while (q--){
        int node, k; cin >> node >> k;
        cout << t.kth_parent(node, k) << "\n";
    }
}
```

Company Queries II (Find the least common ancestor)

Time limit: 1.00 s **Memory limit:** 512 MB

A company has n employees, who form a tree hierarchy where each employee has a boss, except for the general director. Your task is to process q queries of the form: who is the lowest common boss of employees a and b in the hierarchy?

Input

The first input line has two integers n and q: the number of employees and queries. The employees are numbered $1, 2, \ldots, n$, and employee 1 is the general director.

The next line has $n-1$ integers $e_2, e_3, \ldots, e_n$: for each employee $2, 3, \ldots, n$ their boss.

Finally, there are q lines describing the queries. Each line has two integers a and b: who is the lowest common boss of employees a and b?

Output

Print the answer for each query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le e_i \le i - 1$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
3
1
1
```

Solution

```
struct Tree {
    int n, LOG;
    vi depth;
    vector<vi> up;

    Tree(const vector<vi>& adj, int root = 0) {
        n = adj.size();
```

```
        LOG = ceil(log2(n));
        depth.assign(n, 0);
        up.assign(LOG + 1, vi(n, -1));

        dfs(adj, root, root);

        for (int k = 1; k <= LOG; ++k) {
            for (int v = 0; v < n; ++v) {
                int p = up[k - 1][v];
                up[k][v] = (p < 0 ? -1 : up[k - 1][p]);
            }
        }
    }

    // To get the parent and depth of each node
    void dfs(const vector<vi>& adj, int v, int parent) {
        up[0][v] = parent;
        for (int u : adj[v]) {
            if (u == parent) continue;
            depth[u] = depth[v] + 1;
            dfs(adj, u, v);
        }
    }

    int kth_ancestor(int v, int dist) const {
        for (int k = 0; dist && v >= 0; ++k) {
            if (dist & 1) v = up[k][v];
            dist >>= 1;
        }
        return v;
    }


    int LCA(int a, int b) const {
        if (depth[a] < depth[b]) swap(a, b);
        a = kth_ancestor(a, depth[a] - depth[b]);
        if (a == b) return a;
        for (int k = LOG; k >= 0; --k) {
            if (up[k][a] != up[k][b]) {
                a = up[k][a];
                b = up[k][b];
            }
        }
        return up[0][a];
    }
};

void Ramez() {
    int n, q; cin >> n >> q;

    vector<vi> adj(n);
    for (int i = 1, boss; i < n; ++i) {
        cin >> boss;
        --boss;
        adj[boss].push_back(i);
        adj[i].push_back(boss);
```

```
    }

    Tree t(adj);
    while (q--) {
        int u, v;
        cin >> u >> v;
        cout << t.LCA(u - 1, v - 1) + 1 << '\n';
    }
}
```

Counting Paths (Partial sum on tree)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a tree consisting of n nodes, and m paths in the tree. Your task is to calculate for each node the number of paths containing that node.

Input

The first input line contains integers n and m: the number of nodes and paths. The nodes are numbered $1, 2, \ldots, n$.

Then there are $n - 1$ lines describing the edges. Each line contains two integers a and b: there is an edge between nodes a and b.

Finally, there are m lines describing the paths. Each line contains two integers a and b: there is a path between nodes a and b.

Output

Print n integers: for each node $1, 2, \ldots, n$, the number of paths containing that node.

Constraints

- $1 \le n, m \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
3 1 3 1 1
```

Solution

```
struct Tree {
    int n, LOG;
    vi depth;
    vector<vi> up;

    Tree(const vector<vi>& adj, int root = 0) {
```

```
        n = adj.size();
        LOG = ceil(log2(n));
        depth.assign(n, 0);
        up.assign(LOG + 1, vi(n, -1));

        dfs(adj, root, root);
        for (int k = 1; k <= LOG; ++k) {
            for (int v = 0; v < n; ++v) {
                int p = up[k - 1][v];
                up[k][v] = (p < 0 ? -1 : up[k - 1][p]);
            }
        }
    }

    void dfs(const vector<vi>& adj, int v, int parent) {
        up[0][v] = parent;
        for (int u : adj[v]) {
            if (u == parent) continue;
            depth[u] = depth[v] + 1;
            dfs(adj, u, v);
        }
    }

    int kth_ancestor(int v, int dist) const {
        for (int k = 0; dist && v >= 0; ++k) {
            if (dist & 1) v = up[k][v];
            dist >>= 1;
        }
        return v;
    }

    int LCA(int a, int b) const {
        if (depth[a] < depth[b]) swap(a, b);
        a = kth_ancestor(a, depth[a] - depth[b]);
        if (a == b) return a;
        for (int k = LOG; k >= 0; --k) {
            if (up[k][a] != up[k][b]) {
                a = up[k][a];
                b = up[k][b];
            }
        }
        return up[0][a];
    }
};

void Ramez() {
    int n, m;
    cin >> n >> m;
    vector<vi> adj(n);
    for (int i = 0; i < n - 1; i++) {
        int u, v;
        cin >> u >> v;
        --u; --v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
```

```
    Tree tr(adj, 0);
    vi sub(n, 0);

    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        --a; --b;
        int l = tr.LCA(a, b);
        sub[a]++;
        sub[b]++;
        sub[l]--;
        if (l != 0) sub[tr.up[0][l]]--;
    }

    // accumulate differences bottom-up
    function<void(int,int)> dfs2 = [&](int u, int p) {
        for (int v : adj[u]) {
            if (v == p) continue;
            dfs2(v, u);
            sub[u] += sub[v];
        }
    };
    dfs2(0, -1);

    for (int i = 0; i < n; i++) {
        cout << sub[i] << (i + 1 == n ? '\n' : ' ');
    }
}
```

Distance Queries (Queries about distance between any two nodes in a tree)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a tree consisting of n nodes. Your task is to process q queries of the form: what is the distance between nodes a and b?

Input

The first input line contains two integers n and q: the number of nodes and queries. The nodes are numbered $1, 2, \ldots, n$.

Then there are $n - 1$ lines describing the edges. Each line contains two integers a and b: there is an edge between nodes a and b.

Finally, there are q lines describing the queries. Each line contains two integer a and b: what is the distance between nodes a and b?

Output

Print q integers: the answer to each query.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
1
3
2
```

Solution

```
struct Tree {
    int n, LOG;
    vi depth;
    vector<vi> up;

    Tree(const vector<vi>& adj, int root = 0) {
        n = adj.size();
        LOG = ceil(log2(n));
        depth.assign(n, 0);
        up.assign(LOG + 1, vi(n, -1));

        dfs(adj, root, root);
        for (int k = 1; k <= LOG; ++k) {
            for (int v = 0; v < n; ++v) {
                int p = up[k - 1][v];
                up[k][v] = (p < 0 ? -1 : up[k - 1][p]);
            }
        }
    }

    void dfs(const vector<vi>& adj, int v, int parent) {
        up[0][v] = parent;
        for (int u : adj[v]) {
            if (u == parent) continue;
            depth[u] = depth[v] + 1;
            dfs(adj, u, v);
        }
    }

    int kth_ancestor(int v, int dist) const {
        for (int k = 0; dist && v >= 0; ++k) {
            if (dist & 1) v = up[k][v];
            dist >>= 1;
        }
        return v;
    }

    int LCA(int a, int b) const {
        if (depth[a] < depth[b]) swap(a, b);
        a = kth_ancestor(a, depth[a] - depth[b]);
        if (a == b) return a;
```

```
        for (int k = LOG; k >= 0; --k) {
            if (up[k][a] != up[k][b]) {
                a = up[k][a];
                b = up[k][b];
            }
        }
        return up[0][a];
    }
};

void Ramez() {
    int n, q;
    cin >> n >> q;
    vector<vi> adj(n);
    for (int i = 0; i < n - 1; i++) {
        int u, v;
        cin >> u >> v;
        --u; --v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    Tree tr(adj, 0);
    // answer queries: dist(a,b) = depth[a] + depth[b] - 2*depth[LCA]
    while (q--) {
        int a, b;
        cin >> a >> b;
        --a; --b;
        int l = tr.LCA(a, b);
        int dist = tr.depth[a] + tr.depth[b] - 2 * tr.depth[l];
        cout << dist << '\n';
    }
}
```

Distinct Colors (Distinct numbers in each node's subtree, array segment = tree flatenning)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a rooted tree consisting of n nodes. The nodes are numbered $1, 2, \ldots, n$, and node 1 is the root. Each node has a color. Your task is to determine for each node the number of distinct colors in the subtree of the node.

Input

The first input line contains an integer n: the number of nodes. The nodes are numbered $1, 2, \ldots, n$.

The next line consists of n integers $c_1, c_2, \ldots, c_n$: the color of each node.

Then there are $n - 1$ lines describing the edges. Each line contains two integers a and b: there is an edge between nodes a and b.

Output

Print n integers: for each node $1, 2, \ldots, n$, the number of distinct colors.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le a, b \le n$

- $1 \leq c_i \leq 10^9$

Example

Input:

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

Output:

```
3 1 2 1 1
```

Solution

```cpp
struct FenwickPURQ {
    int n;
    vi f;

    FenwickPURQ(int size) : n(size), f(n + 1, 0) {}

    void add(int idx, int val) {
        for (; idx <= n; idx += idx & -idx) f[idx] += val;
    }

    int prefix(int idx) {
        int res = 0;
        for (; idx > 0; idx -= idx & -idx) res += f[idx];
        return res;
    }

    int rangeQuery(int l, int r) {
        return prefix(r) - prefix(l - 1);
    }
};

int n;
vector<vi> G;
vi color;
vi in_t, out_t;
int timer;

void dfs(int u, int p) {
    in_t[u] = ++timer;
    for (int v : G[u]) {
        if (v == p) continue;
        dfs(v, u);
    }
    out_t[u] = ++timer;
}

void Ramez() {
    cin >> n;
    color.assign(n + 1, 0);
```

```
for (int i = 1; i <= n; i++) cin >> color[i];
G.assign(n + 1, vi());
for (int i = 0; i < n - 1; i++) {
    int a, b;
    cin >> a >> b;
    G[a].push_back(b);
    G[b].push_back(a);
}

in_t.assign(n + 1, 0);
out_t.assign(n + 1, 0);
timer = 0;
dfs(1, 0);

int N2 = 2 * n;
vi x(N2 + 1);
for (int i = 1; i <= n; i++) {
    x[in_t[i]] = color[i];
    x[out_t[i]] = color[i];
}

vi ptr(N2 + 1, 0);
unordered_map<int, int> last;
last.reserve(N2 * 2);
for (int i = N2; i >= 1; i--) {
    auto it = last.find(x[i]);
    if (it != last.end()) ptr[i] = it->second;
    last[x[i]] = i;
}

FenwickPURQ BIT(N2);
unordered_set<int> seen;
seen.reserve(N2 * 2);
for (int i = 1; i <= N2; i++) {
    if (seen.insert(x[i]).second) {
        BIT.add(i, 1);
    }
}

struct Query { int L, R, idx; };
vector<Query> queries;
queries.reserve(n);
for (int i = 1; i <= n; i++) {
    queries.push_back({in_t[i], out_t[i], i});
}
sort(all(queries), [](auto &a, auto &b){ return a.L < b.L; });

vi ans(n + 1);
int l = 1;
for (auto &q : queries) {
    while (l < q.L) {
        if (ptr[l] != 0) BIT.add(ptr[l], 1);
        l++;
    }
    ans[q.idx] = BIT.rangeQuery(q.L, q.R);
}
```

```
    for (int i = 1; i <= n; i++) {
        cout << ans[i] << (i == n ? '\n' : ' ');
    }
}
```

Finding a Centroid (node such that when it is becomes the root, each subtree has at most $\lfloor n/2 \rfloor$ nodes.)

Time limit: 1.00 s **Memory limit:** 512 MB

Given a tree of n nodes, your task is to find a centroid, i.e., a node such that when it is appointed the root of the tree, each subtree has at most $\lfloor n/2 \rfloor$ nodes.

Input

The first input line contains an integer n: the number of nodes. The nodes are numbered $1, 2, \ldots, n$.

Then there are $n-1$ lines describing the edges. Each line contains two integers a and b: there is an edge between nodes a and b.

Output

Print one integer: a centroid node. If there are several possibilities, you can choose any of them.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
3
```

Solution

```
const int maxN = 2e5+5;
int N, a, b, p[maxN], sz[maxN];
vector<int> G[maxN];
void dfs(int u = 1){
    sz[u] = 1;
    for(int v : G[u]){
        if(v != p[u]){
            p[v] = u;
            dfs(v);
            sz[u] += sz[v];
        }
    }
}
int main(){
```

```
    scanf("%d", &N);
    for(int i = 0; i < N-1; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
    }
    dfs();
    for(int i = 1; i <= N; i++){
        bool centroid = true;
        if(p[i] != 0 && N-sz[i] > N/2)
            centroid = false;
        for(int v : G[i])
            if(v != p[i] && sz[v] > N/2)
                centroid = false;
        if(centroid){
            printf("%d\n", i);
            return 0;
        }
    }
}
```

Path Queries (Sum of nodes on the path from root node to node s, updates supported)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a rooted tree consisting of n nodes. The nodes are numbered $1, 2, \ldots, n$, and node 1 is the root. Each node has a value. Your task is to process following types of queries: - change the value of node s to x - calculate the sum of values on the path from the root to node s

Input

The first input line contains two integers n and q: the number of nodes and queries. The nodes are numbered $1, 2, \ldots, n$.

The next line has n integers $v_1, v_2, \ldots, v_n$: the value of each node.

Then there are $n-1$ lines describing the edges. Each line contains two integers a and b: there is an edge between nodes a and b.

Finally, there are q lines describing the queries. Each query is either of the form "1 s x" or "2 s".

Output

Print the answer to each query of type 2.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le a, b, s \le n$
- $1 \le v_i, x \le 10^9$

Example

Input:

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

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

Output:

```
11
8
```

Solution

```
struct FenwickPUPQ {
    int n;
    vector<int> f;
    FenwickPUPQ(int _n) : n(_n), f(n + 1, 0) {}

    void update(int idx, int val) {
        for (; idx <= n; idx += idx & -idx)
            f[idx] += val;
    }

    int query(int idx) {
        int res = 0;
        for (; idx > 0; idx -= idx & -idx)
            res += f[idx];
        return res;
    }

    int rangeQuery(int l, int r) {
        return query(r) - query(l - 1);
    }
};

void Ramez() {
    int n, q; cin >> n >> q;
    vi a(n + 1); for (int i = 1; i <= n; i++) cin >> a[i];

    vector<vi> adj(n + 1);
    for (int i = 0; i < n - 1; i++) {
        int u, v; cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    int MAX = 3 * n;
    vi in(MAX), out(MAX);
    int timer = 1;

    // tree flattening
    function<void(int, int)> dfs = [&](int u, int p) {
        in[u] = timer++;
        for (int v : adj[u]) {
            if(v == p) continue;
            dfs(v, u);
        }
        out[u] = timer++;
    };
```

```
dfs(1, 0);

FenwickPUPQ tree(MAX);

for (int i = 1; i <= n; i++) {
    tree.update(in[i], a[i]);
    tree.update(out[i], -a[i]);
}


while (q--) {
    int ops; cin >> ops;
    if (ops == 1) {
        int s, x; cin >> s >> x;
        // a[s] = x ===> a[s] += x - a[s]
        tree.update(in[s], x - a[s]);
        // -a[s] = -x ===> -a[s] += -x + a[s]
        tree.update(out[s], a[s] - x);
        a[s] = x;
    }
    else {
        int s; cin >> s;
        cout << tree.query(in[s]) << "\n";
    }
}
```

Path Queries II (Find maximum node on the path between node a and b, updates supported)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a tree consisting of n nodes. The nodes are numbered $1, 2, \ldots, n$. Each node has a value. Your task is to process following types of queries: - change the value of node s to x - find the maximum value on the path between nodes a and b.

Input

The first input line contains two integers n and q: the number of nodes and queries. The nodes are numbered $1, 2, \ldots, n$.

The next line has n integers $v_1, v_2, \ldots, v_n$: the value of each node.

Then there are $n - 1$ lines describing the edges. Each line contains two integers a and b: there is an edge between nodes a and b.

Finally, there are q lines describing the queries. Each query is either of the form "1 s x" or "2 a b".

Output

Print the answer to each query of type 2.

Constraints

- $1 \le n, q \le 2 \cdot 10^5$
- $1 \le a, b, s \le n$
- $1 \le v_i, x \le 10^9$

Example

Input:

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

Output:

```
4 3
```

Solution

```cpp
/**
 * Link Cut Tree
 */
const int maxN = 2e5+1;
int N, Q, k, x, y, root, idcounter;
struct Lazy {
    int a = 1, b = 0;
    bool empty(){
        return (a == 1 && b == 0);
    }
};
struct Node {
    Lazy tag;
    bool rev = false;
    Node *c[2] = {nullptr, nullptr}, *p = nullptr;
    int sz, val, sum, mn, mx, id = ++idcounter;
    Node(int v){
        sz = 1;
        val = sum = mn = mx = v;
    }
    void apply(Lazy other){
        mn = mn*other.a + other.b;
        mx = mx*other.a + other.b;
        val = val*other.a + other.b;
        sum = sum*other.a + sz*other.b;
        tag = {tag.a*other.a, tag.b*other.a+other.b};
    }
    void push(){
        if(rev){
            swap(c[0], c[1]);
            if(c[0])    c[0]->rev ^= true;
            if(c[1])    c[1]->rev ^= true;
            rev = false;
        }
        if(!tag.empty()){
            if(c[0])    c[0]->apply(tag);
            if(c[1])    c[1]->apply(tag);
            tag = Lazy();
        }
```

```
    }
    void pull(){
        sum = mn = mx = val;
        sz = 1;
        for(int i = 0; i < 2; i++){
            if(c[i]){
                mn = min(mn, c[i]->mn);
                mx = max(mx, c[i]->mx);
                sum += c[i]->sum;
                sz += c[i]->sz;
            }
        }
    }
} *LCT[maxN];
bool notRoot(Node *t){
    return t->p && (t->p->c[0] == t || t->p->c[1] == t);
}
void rotate(Node *t){
    Node *p = t->p;
    bool b = (p->c[0] == t);
    if((t->p = p->p) && notRoot(p)) t->p->c[(t->p->c[1] == p)] = t;
    if((p->c[!b]=t->c[b]))          p->c[!b]->p = p;
    t->c[b] = p;
    p->p = t;
    p->pull();
}
void splay(Node *t){
    while(notRoot(t)){
        Node *p = t->p;
        p->push();
        t->push();
        rotate(t);
    }
    t->push();
    t->pull();
}
Node* access(Node *t){
    Node *last = nullptr;
    for(Node *u = t; u; u = u->p){
        splay(u);
        u->c[1] = last;
        last = u;
    }
    splay(t);
    return last;
}
void evert(Node *t){
    access(t);
    t->rev = true;
}
void link(Node *u, Node *v){
    evert(u);
    u->p = v;
}
void cut(Node *u, Node *v){
    evert(u);
```

```
    access(v);
    if(v->c[0]) v->c[0]->p = 0;
    v->c[0] = 0;
    v->pull();
}
Node* path(Node *u, Node *v){
    evert(u);
    access(v);
    return v;
}
Node* LCA(Node *u, Node *v){
    evert(LCT[root]);
    access(u);
    return access(v);
}
bool connected(Node *u, Node *v){
    path(u, v);
    while(v->c[0])
        v = v->c[0];
    return u == v;
}
int main(){
    scanf("%d %d", &N, &Q);
    for(int i = 1; i <= N; i++){
        scanf("%d", &x);
        LCT[i] = new Node(x);
    }
    for(int i = 0; i < N-1; i++){
        scanf("%d %d", &x, &y);
        link(LCT[x], LCT[y]);
    }
    root = 1;
    for(int i = 0; i < Q; i++){
        scanf("%d", &k);
        if(k == 1){
            scanf("%d %d", &x, &y);
            Node *p = path(LCT[x], LCT[x]);
            p->apply({0, y});
        } else if(k == 2){
            scanf("%d %d", &x, &y);
            Node *p = path(LCT[x], LCT[y]);
            printf("%d ", p->mx);
        }
    }
}
```

Subordinates (finding the size of each sub-tree)

Time limit: 1.00 s **Memory limit:** 512 MB

Given the structure of a company, your task is to calculate for each employee the number of their subordinates.

Input

The first input line has an integer n: the number of employees. The employees are numbered $1, 2, \ldots, n$, and employee 1 is the general director of the company.

After this, there are $n-1$ integers: for each employee $2, 3, \ldots, n$ their direct boss in the company.

Output

Print n integers: for each employee $1, 2, \ldots, n$ the number of their subordinates.

Constraints

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

Example

Input:

```
5
1 1 2 3
```

Output:

```
4 1 1 0 0
```

Solution

```
const int maxN = 2e5;
int sz[maxN+1];
vi G[maxN+1];

void dfs(int u){
    sz[u] = 1;
    for(int v : G[u]){
        dfs(v);
        sz[u] += sz[v];
    }
}

void Ramez() {
    int n; cin >> n;
    for (int v = 2; v <= n; v++){
        int u; cin >> u;
        G[u].push_back(v);
    }

    dfs(1);

    for (int i = 1; i <= n; i++){
        cout << sz[i] - 1 << " ";
    }
}
```

Subtree Queries (Sum of subtree nodes, updates supported)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a rooted tree consisting of n nodes. The nodes are numbered $1, 2, \ldots, n$, and node 1 is the root. Each node has a value. Your task is to process following types of queries: - change the value of node s to x - calculate the sum of values in the subtree of node s

Input

The first input line contains two integers n and q: the number of nodes and queries. The nodes are numbered $1, 2, \ldots, n$.

The next line has n integers $v_1, v_2, \ldots, v_n$: the value of each node.

Then there are $n-1$ lines describing the edges. Each line contans two integers a and b: there is an edge between nodes a and b.

Finally, there are q lines describing the queries. Each query is either of the form "1 s x" or "2 s".

Output

Print the answer to each query of type 2.

Constraints

- $1 \leq n, q \leq 2 \cdot 10^5$
- $1 \leq a, b, s \leq n$
- $1 \leq v_i, x \leq 10^9$

Example

Input:

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

Output:

```
8
10
```

Solution

```
struct FenwickPURQ {
    int n;
    vi f;

    FenwickPURQ(int size) : n(size), f(n + 1, 0) {}

    void add(int idx, int val) {
        for (; idx <= n; idx += idx & -idx)
            f[idx] += val;
    }

    int prefix(int idx) const {
        int res = 0;
        for (; idx > 0; idx -= idx & -idx)
            res += f[idx];
        return res;
    }

    int rangeQuery(int l, int r) const {
        return prefix(r) - prefix(l - 1);
    }
};
```

```
int n, q;
vector<vi> adj;
vi tin, tout, flat_val, timer;
vi init_val;
FenwickPURQ *fenw;

void dfs(int u, int p) {
    tin[u] = ++timer[0];
    flat_val[timer[0]] = init_val[u];
    for (int v : adj[u]) if (v != p) {
        dfs(v, u);
    }
    tout[u] = timer[0];
}

void Ramez() {
    cin >> n >> q;
    init_val.assign(n + 1, 0);
    for (int i = 1; i <= n; i++) cin >> init_val[i];
    adj.assign(n + 1, vi());
    for (int i = 0; i < n - 1; i++) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    tin.assign(n + 1, 0);
    tout.assign(n + 1, 0);
    flat_val.assign(n + 1, 0);
    timer.assign(1, 0);

    // Flatten tree
    dfs(1, 0);

    // Build Fenwick tree
    fenw = new FenwickPURQ(n);
    for (int i = 1; i <= n; i++) {
        fenw->add(i, flat_val[i]);
    }

    // Process queries
    while (q--) {
        int type;
        cin >> type;
        if (type == 1) {
            int s, x;
            cin >> s >> x;
            // compute delta
            int idx = tin[s];
            int current = fenw->rangeQuery(idx, idx);
            int delta = x - current;
            fenw->add(idx, delta);
        } else if (type == 2) {
            int s;
            cin >> s;
            int res = fenw->rangeQuery(tin[s], tout[s]);
```

```
            cout << res << '\n';
        }
    }
}
```

Tree Diameter (the longest distance between any two nodes)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a tree consisting of n nodes. The diameter of a tree is the maximum distance between two nodes. Your task is to determine the diameter of the tree.

Input

The first input line contains an integer n: the number of nodes. The nodes are numbered $1, 2, \ldots, n$.

Then there are $n - 1$ lines describing the edges. Each line contains two integers a and b: there is an edge between nodes a and b.

Output

Print one integer: the diameter of the tree.

Constraints

- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq a, b \leq n$

Example

Input:

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

Output:

```
3
```

Explanation: The diameter corresponds to the path $2 \to 1 \to 3 \to 5$.

Solution

```
int32_t main() {
    FastIO();
    int n;
    cin >> n;
    vector<vi> adj(n + 1);
    for (int i = 0; i < n - 1; i++) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    auto bfs = [&](int src) {
        vector<int> dist(n + 1, -1);
        queue<int> q;
        q.push(src);
```

```
        dist[src] = 0;
        int far = src;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int v : adj[u]) {
                if (dist[v] == -1) {
                    dist[v] = dist[u] + 1;
                    q.push(v);
                    if (dist[v] > dist[far]) far = v;
                }
            }
        }
        return make_pair(far, dist[far]);
    };
    // First BFS from node 1 (or any node)
    int start = 1;
    auto res1 = bfs(start);
    // Second BFS from the farthest node found
    auto res2 = bfs(res1.first);
    // The second distance is the diameter
    cout << res2.second;
    return 0;
}
```

Tree Distances I (maximum distance from every node, maximum distance up or down)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a tree consisting of n nodes. Your task is to determine for each node the maximum distance to another node.

Input

The first input line contains an integer n: the number of nodes. The nodes are numbered $1, 2, \ldots, n$.

Then there are $n-1$ lines describing the edges. Each line contains two integers a and b: there is an edge between nodes a and b.

Output

Print n integers: for each node $1, 2, \ldots, n$, the maximum distance to another node.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
2 3 2 3 3
```

Solution

```
const int maxN = 2e5+1;
int N, a, b, down[maxN], up[maxN], best[maxN];
vector<int> G[maxN];
void dfs1(int u = 1, int p = 0){
    for(int v : G[u]){
        if(v != p){
            dfs1(v, u);
            down[u] = max(down[u], down[v]+1);
        }
    }
}
void dfs2(int u = 1, int p = 0){
    int first = 0, second = 0;
    for(int v : G[u]){
        if(v != p){
            if(down[v] >= down[first]){
                second = first;
                first = v;
            } else if(down[v] > down[second]){
                second = v;
            }
        }
    }
    for(int v : G[u]){
        if(v != p){
            up[v] = max(up[v], up[u]+1);
            if(v == first){
                if(second != 0){
                    up[v] = max(up[v], down[second]+2);
                }
            } else if(first != 0){
                up[v] = max(up[v], down[first]+2);
            }
            dfs2(v, u);
        }
    }
    best[u] = max(up[u], down[u]);
}
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N-1; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
    }
    dfs1();
    dfs2();
    for(int i = 1; i <= N; i++)
        printf("%d%c", best[i], (" \n")[i==N]);
}
```

Tree Distances II (Sum of all distances from any node to any other node)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a tree consisting of n nodes. Your task is to determine for each node the sum of the distances from

the node to all other nodes.

Input

The first input line contains an integer n: the number of nodes. The nodes are numbered $1, 2, \ldots, n$.

Then there are $n-1$ lines describing the edges. Each line contains two integers a and b: there is an edge between nodes a and b.

Output

Print n integers: for each node $1, 2, \ldots, n$, the sum of the distances.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
6 9 5 8 8
```

Solution

```
const int maxN = 2e5+1;
int N, a, b, sz[maxN];
ll down[maxN], up[maxN];
vector<int> G[maxN];
void dfs1(int u = 1, int p = 0){
    sz[u] = 1;
    for(int v : G[u]){
        if(v != p){
            dfs1(v, u);
            sz[u] += sz[v];
            down[u] += down[v] + sz[v];
        }
    }
}
void dfs2(int u = 1, int p = 0){
    if(p != 0)
        up[u] = (up[p]+down[p]) + N - (2*sz[u]+down[u]);
    for(int v : G[u])
        if(v != p)
            dfs2(v, u);
}
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N-1; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
```

```
        G[b].push_back(a);
    }
    dfs1();
    dfs2();
    for(int i = 1; i <= N; i++)
        printf("%lld%c", down[i]+up[i], (" \n")[i==N]);
}
```

Tree Matching (making pairs out of the tree)

Time limit: 1.00 s **Memory limit:** 512 MB

You are given a tree consisting of n nodes. A matching is a set of edges where each node is an endpoint of at most one edge. What is the maximum number of edges in a matching?

Input

The first input line contains an integer n: the number of nodes. The nodes are numbered $1, 2, \ldots, n$.

Then there are $n-1$ lines describing the edges. Each line contains two integers a and b: there is an edge between nodes a and b.

Output

Print one integer: the maximum number of pairs.

Constraints

- $1 \le n \le 2 \cdot 10^5$
- $1 \le a, b \le n$

Example

Input:

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

Output:

```
2
```

Explanation: One possible matching is $(1, 2)$ and $(3, 4)$.

Solution

```
const int maxN = 2e5+1;
int N, a, b, cnt;
bool used[maxN];
vector<int> G[maxN];
void dfs(int u = 1, int par = 0){
    for(int v : G[u])
        if(v != par)
            dfs(v, u);
    if(!used[par] && !used[u] && par != 0){
        used[par] = used[u] = true;
        cnt++;
    }
```

```
}
int main(){
    scanf("%d", &N);
    for(int i = 0; i < N-1; i++){
        scanf("%d %d", &a, &b);
        G[a].push_back(b);
        G[b].push_back(a);
    }
    dfs();
    printf("%d\n", cnt);
}
```