Notebook - Maratona de Programação

Lenhadoras de Segtree

Contents

1 Misc

1.1 Builtin Overflow

```
// Returns true if the operation results in overflow.

bool __builtin_add_overflow (type1 a, type2 b, type3 *res)
bool __builtin_sadd_overflow (int a, int b, int *res)
bool __builtin_saddl_overflow (long int a, long int b, long int *res)
bool __builtin_saddll_overflow (long long int a, long long int b, long long int *res)
bool __builtin_uadd_overflow (unsigned int a, unsigned int b, unsigned int *res)
bool __builtin_uaddl_overflow (unsigned long int a, unsigned long int b, unsigned long int *res)
bool __builtin_uaddll_overflow (unsigned long long int a, unsigned long long int b, unsigned long long int *res)

bool __builtin_sub_overflow (type1 a, type2 b, type3 *res)
bool __builtin_ssub_overflow (int a, int b, int *res)
bool __builtin_ssubl_overflow (long int a, long int b, long int *res)
bool __builtin_ssubll_overflow (long long int a, long long int b, long long int *res)
bool __builtin_usub_overflow (unsigned int a, unsigned int b, unsigned int *res)
bool __builtin_usubl_overflow (unsigned long int a, unsigned long int b, unsigned long int *res)
bool __builtin_usubll_overflow (unsigned long long int a, unsigned long long int b, unsigned long long int *res)

bool __builtin_mul_overflow (type1 a, type2 b, type3 *res)
bool __builtin_smul_overflow (int a, int b, int *res)
bool __builtin_smull_overflow (long int a, long int b, long int *res)
bool __builtin_smulll_overflow (long long int a, long long int b, long long int *res)
bool __builtin_umul_overflow (unsigned int a, unsigned int b, unsigned int *res)
bool __builtin_umull_overflow (unsigned long int a, unsigned long int b, unsigned long int *res)
bool __builtin_umulll_overflow (unsigned long long int a, unsigned long long int b, unsigned long long int *res)

bool __builtin_add_overflow_p (type1 a, type2 b, type3 c)
bool __builtin_sub_overflow_p (type1 a, type2 b, type3 c)
bool __builtin_mul_overflow_p (type1 a, type2 b, type3 c)
```

1.2 Split

```
vector<string> split(string txt, char key = ' '){
    vector<string> ans;

    string palTemp = "";
    for(int i = 0; i < txt.size(); i++){

        if(txt[i] == key){
            if(palTemp.size() > 0){
                ans.push_back(palTemp);
                palTemp = "";
            }
        } else{
            palTemp += txt[i];
        }

    }

    if(palTemp.size() > 0)
        ans.push_back(palTemp);

    return ans;
}
```

1.3 Lower Upper

```
 cout << (char)tolower('A') << '\n';
```

1.4 Eraseunique

```
sort(v.begin(), v.end());
v.erase(unique(v.begin(), v.end()), v.end());
```

1.5 Int128

```
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
```

2 Data Structures

2.1 Ordered Set

```
// Description:
// insert(k) - add element k to the ordered set
// erase(k) - remove element k from the ordered set
// erase(it) - remove element it points to from the ordered set
// order_of_key(k) - returns number of elements strictly smaller than k
// find_by_order(n) - return an iterator pointing to the k-th element in the ordered set (counting from zero).

// Problem:
// https://cses.fi/problemset/task/2169/

// Complexity:
// O(log n) for all operations

// How to use:
// ordered_set<int> os;
// cout << os.order_of_key(1) << '\n;
// cout << os.find_by_order(1) << '\n;

// Notes
// The ordered set only contains different elements
```

```cpp
// By using less_equal<T> instead of less<T> on using ordered_set declaration
// The ordered_set becomes an ordered_multiset
// So the set can contain elements that are equal

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T,null_type,less<T>, rb_tree_tag,tree_order_statistics_node_update>;

void Erase(ordered_set<int>& a, int x){
    int r = a.order_of_key(x);
    auto it = a.find_by_order(r);
    a.erase(it);
}
```

2.2 Sparse Table

```cpp
// Description:
// Data structure to query for minimum and maximum

// Problem:
// https://cses.fi/problemset/task/1647/

// Complexity:
// Build O(n log n)
// Query O(1)

#include <bits/stdc++.h>

using namespace std;

const int MAX = 2e5+17;
const int INF = 1e9+17;

struct SparseTable {
  int n;
  vector<int> arr;
  vector<vector<int>> st;
  vector<int> log_2;

  SparseTable(vector<int>& arr, int& n) : arr(arr), n(n) {
    build();
  }

  void build() {
    log_2.resize(MAX + 1);

    log_2[1] = 0;
    for (int i = 2; i <= MAX; i++) {
      log_2[i] = log_2[i/2] + 1;
    }

    int K = log_2[n + 1];

    st.resize(MAX, vector<int>(K + 1));

    for (int i = 0; i < MAX; i++) {
      for (int j = 0; j < K + 1; j++) {
        st[i][j] = INF;
      }
    }

    for (int i = 0; i < n; i++) {
      st[i][0] = arr[i];
    }

    for (int j = 1; j <= K; j++) {
      for (int i = 0; i + (1 << j) < MAX; i++) {
        st[i][j] = min(st[i][j-1], st[i + (1 << (j - 1))][j - 1]);
      }
    }
  }

  int query(int l, int r) {
    int j = log_2[r - l + 1];
    return min(st[l][j], st[r - (1 << j) + 1][j]);
  }
};
```

2.3 Dsu

```cpp
#include <bits/stdc++.h>

using namespace std;

const int MAX = 1e6+17;

struct DSU {
    int n;
    vector<int> link, sizes;

    DSU(int n) {
        this->n = n;
        link.assign(n+1, 0);
        sizes.assign(n+1, 1);

        for (int i = 0; i <= n; i++)
            link[i] = i;
    }

    int find(int x) {
        while (x != link[x])
            x = link[x];

        return x;
    }

    bool same(int a, int b) {
        return find(a) == find(b);
    }

    void unite(int a, int b) {
        a = find(a);
        b = find(b);

        if (a == b) return;

        if (sizes[a] < sizes[b])
            swap(a, b);

        sizes[a] += sizes[b];
        link[b] = a;
    }

    int size(int x) {
        return sizes[x];
    }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int cities, roads; cin >> cities >> roads;
    vector<int> final_roads;
    int ans = 0;
    DSU dsu = DSU(cities);
    for (int i = 0, a, b; i < roads; i++) {
        cin >> a >> b;
        dsu.unite(a, b);
    }

    for (int i = 2; i <= cities; i++) {
        if (!dsu.same(1, i)) {
            ans++;
            final_roads.push_back(i);
            dsu.unite(1,i);
        }
    }

    cout << ans << '\n';
    for (auto e : final_roads) {
        cout << "1 " << e << '\n';
    }

}
```

2.4 Mergesort Tree Ordered Set

```
// Description:
// In each node, the tree keeps a sorted list of elements in that range.
// It can be used to find how many elements are greater than x in a given range.
// It can also be used to find the position of an element if the list was sorted.
// query(i, j, k) - how many elements greater than k are in the range (i, j)
// update(i, val) - changes the value of the element on index i to val

// Problem:
// https://www.beecrowd.com.br/judge/pt/problems/view/3097

// Complexity:
// O(n log ^ 2 ^ 2 n) for build
// O(log ^ 2 n) for query

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T,null_type,less_equal<T>, rb_tree_tag,tree_order_statistics_node_update>;

struct MergeSortTree {
  vector<ordered_set<int>> tree;
  vector<int> v;
  int n;

  MergeSortTree(int n, vector<int>& v) : n(n), v(v) {
    int sz = 1;
    while (sz < n) sz *= 2;

    tree.resize(2 * sz);

    build(0, 0, n - 1, v);
  }

  void Erase(ordered_set<int>& a, int x){
    int r = a.order_of_key(x);
    auto it = a.find_by_order(r);
    a.erase(it);
  }

  ordered_set<int> merge(ordered_set<int>& a, ordered_set<int>& b) {
    ordered_set<int> res;

    for (auto e : a) res.insert(e);
    for (auto e : b) res.insert(e);

    return res;
  }

  void build(int pos, int ini, int fim, vector<int>& v) {
    if (ini == fim) {
      if (ini < (int)v.size()) {
        tree[pos].insert(v[ini]);
      }
      return;
    }

    int mid = ini + (fim - ini) / 2;

    build(2 * pos + 1, ini, mid, v);
    build(2 * pos + 2, mid + 1, fim, v);

    tree[pos] = merge(tree[2 * pos + 1], tree[2 * pos + 2]);
  }

  // how many elements greater than val in vector v
  int search(ordered_set<int>& v, int val) {
    return (int)v.size() - v.order_of_key(val + 1);
  }

  // how many elements greater than val in the range (p, q)
  int query(int pos, int ini, int fim, int p, int q, int val) {
    if (fim < p || ini > q) {
      return 0;
    }

    if (ini >= p && fim <= q) {
      return search(tree[pos], val);
    }

    int mid = ini + (fim - ini) / 2;
    return query(2 * pos + 1, ini, mid, p, q, val) + query(2 * pos + 2, mid + 1, fim, p, q, val);
  }

  void update(int pos, int ini, int fim, int id, int val) {
    if (ini == id && fim == id) {
      if (!tree[pos].empty()) Erase(tree[pos], v[id]);
      tree[pos].insert(val);
      return;
    }

    if (fim < id || ini > id) {
      return;
    }

    int mid = ini + (fim - ini) / 2;
    update(2 * pos + 1, ini, mid, id, val);
    update(2 * pos + 2, mid + 1, fim, id, val);

    if (!tree[pos].empty()) Erase(tree[pos], v[id]);
    tree[pos].insert(val);
  }

  int query(int p, int q, int val) {
    return query(0, 0, n - 1, p, q, val);
  }

  void update(int id, int val) {
    update(0, 0, n - 1, id, val);
    v[id] = val;
  }
};
```

2.5 Two Sets

```
// Description
// THe values are divided in two multisets so that one of them contain all values that are
// smaller than the median and the other one contains all values that are greater or equal to the median.

// Problem:
// https://atcoder.jp/contests/abc306/tasks/abc306_e
// Problem I - Maratona Feminina de çãProgramao da Unicamp 2023
// https://codeforces.com/group/WYIydkiPyE/contest/450037/attachments

// Complexity:
// Add and remove elements - O(log n)
// Return sum of biggest or smallest set or return the median - O(1)

using ll = long long;

struct TwoSets {
  multiset<int> small;
  multiset<int> big;
  ll sums = 0;
  ll sumb = 0;
  int n = 0;

  int size_small() {
    return small.size();
  }

  int size_big() {
    return big.size();
  }

  void balance() {
    while (size_small() > n / 2) {
      int v = *small.rbegin();
      small.erase(prev(small.end()));
      big.insert(v);
      sums -= v;
      sumb += v;
    }
    while (size_big() > n - n / 2) {
      int v = *big.begin();
      big.erase(big.begin());
      small.insert(v);
      sumb -= v;
      sums += v;
    }
  }

  void add(int x) {
    n++;
    small.insert(x);
    sums += x;
    while (!small.empty() && *small.rbegin() > *big.begin()) {
      int v = *small.rbegin();
      small.erase(prev(small.end()));
      big.insert(v);
      sums -= v;
      sumb += v;
    }
    balance();
  }

  bool rem(int x) {
    n--;
    auto it1 = small.find(x);
    auto it2 = big.find(x);
    bool flag = false;
    if (it1 != small.end()) {
      sums -= *it1;
      small.erase(it1);
      flag = true;
    } else if (it2 != big.end()) {
      sumb -= *it2;
      big.erase(it2);
      flag = true;
    }
    balance();
    return flag;
  }

  ll sum_small() {
    return sums;
  }

  ll sum_big() {
    return sumb;
  }

  int median() {
    return *big.begin();
  }
};
```

2.6 Sparse Table2d

```
// Description
// Minimum queries in a 2D grid

// Problem:
// https://codeforces.com/group/YgJmumGtHD/contest/103794/problem/D

// Complexity:
// Build O(N * M * log(N) * log(M))
// Query O(1)
// Memory COmplexity: O(N * M * log(N) * log(M))

const int MAX = 410;

struct SparseTable2D {
  vector<vector<int>> matrix;
  vector<vector<vector<vector<int>>>> table;
  int n, m;

  SparseTable2D(vector<vector<int>>& matrix, int n, int m) : matrix(matrix), n(n), m(m) {
    table.resize(MAX, vector<vector<vector<int>>>(MAX, vector<vector<int>>(log2(MAX) + 1, vector<int>(log2(MAX) + 1))));
    build();
  }

  int f(int a, int b) {
    return max(a, b);
  }

  void build() {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        table[i][j][0][0] = matrix[i][j];
      }
    }

    for (int k = 1; k <= (int)(log2(n)); k++) {
      for (int i = 0; i + (1 << k) - 1 < n; i++) {
        for (int j = 0; j + (1 << k) - 1 < m; j++) {
          table[i][j][k][0] = f(
          table[i][j][k - 1][0],
          table[i + (1 << (k - 1))][j][k - 1][0]);
        }
      }
    }

    for (int k = 1; k <= (int)(log2(m)); k++) {
      for (int i = 0; i < n; i++) {
        for (int j = 0; j + (1 << k) - 1 < m; j++) {
          table[i][j][0][k] = f(
          table[i][j][0][k - 1],
          table[i][j + (1 << (k - 1))][0][k - 1]);
        }
      }
    }

    for (int k = 1; k <= (int)(log2(n)); k++) {
      for (int l = 1; l <= (int)(log2(m)); l++) {
        for (int i = 0; i + (1 << k) - 1 < n; i++) {
          for (int j = 0; j + (1 << l) - 1 < m; j++) {
            table[i][j][k][l] = f(
              f(
                table[i][j][k - 1][l - 1],
                table[i + (1 << (k - 1))][j][k - 1][l - 1]
              ),
              f(
                table[i][j + (1 << (l - 1))][k - 1][l - 1],
                table[i + (1 << (k - 1))][j + (1 << (l - 1))][k - 1][l - 1])
              );
          }
        }
      }
    }
  }

  int query(int x1, int y1, int x2, int y2) {
    int k = log2(x2 - x1 + 1);
    int l = log2(y2 - y1 + 1);

    return f(
      f(
        table[x1][y1][k][l],
        table[x2 - (1 << k) + 1][y1][k][l]
      ),
      f(
        table[x1][y2 - (1 << l) + 1][k][l],
        table[x2 - (1 << k) + 1][y2 - (1 << l) + 1][k][l]
      )
    );
  }
};
```

2.7 Priority Queue

```
// Description:
// Keeps the largest (by default) element at the top of the queue

// Problem:
// https://cses.fi/problemset/task/1164/

// Complexity:
// O(log n) for push and pop
// O (1) for looking at the element at the top

// How to use:
// prioriy_queue<int> pq;
// pq.push(1);
// pq.top();
// pq.pop()

// Notes
// To use the priority queue keeping the smallest element at the top

priority_queue<int, vector<int>, greater<int>> pq;
```

2.8 Psum2d

```
// Description:
// Queries the sum of a rectangle that goes from grid[from_row][from_col] to grid[to_row][to_col]

// Problem:
// https://cses.fi/problemset/task/1652/

// Complexity:
// O(n) build
// O(1) query

for (int i = 1; i <= n; i++) {
  for (int j = 1; j <= n; j++) {
    psum[i][j] = grid[i][j] + psum[i - 1][j] + psum[i][j - 1] - psum[i - 1][j - 1];
  }
}

while (q--) {
  int from_row, to_row, from_col, to_col;
  cin >> from_row >> from_col >> to_row >> to_col;
  cout << psum[to_row][to_col] - psum[from_row - 1][to_col] -
  psum[to_row][from_col - 1] + psum[from_row - 1][from_col - 1] << '\n';
}
```

2.9 Mergesort Tree Vector

```
// Description:
// In each node, the tree keeps a sorted list of elements in that range.
// It can be used to find how many elements are greater than x in a given range.
// It can also be used to find the position of an element if the list was sorted.
// query(i, j, k) - how many elements greater than k are in the range (i, j)

// Problem:
// https://www.spoj.com/problems/KQUERY

// Complexity:
// O(n log n) for build
// O(log ^ 2 n) for query

struct MergeSortTree {
  vector<vector<int>> tree;
  int n;

  MergeSortTree(int n, vector<int>& v) : n(n) {
    int sz = 1;
    while (sz < n) sz *= 2;

    tree.assign(2 * sz, vector<int>());
    build(0, 0, n - 1, v);
  }

  vector<int> merge(vector<int>& a, vector<int>& b) {
    vector<int> res((int)a.size() + (int)b.size());
    int it = 0, jt = 0, curr = 0;

    while (it < (int)a.size() && jt < (int)b.size()) {
      if (a[it] <= b[jt]) {
        res[curr++] = a[it++];
      } else {
        res[curr++] = b[jt++];
      }
    }

    while (it < (int)a.size()) {
      res[curr++] = a[it++];
    }

    while (jt < (int)b.size()) {
      res[curr++] = b[jt++];
    }

    return res;
  }

  void build(int pos, int ini, int fim, vector<int>& v) {
    if (ini == fim) {
      if (ini < (int)v.size()) {
        tree[pos].pb(v[ini]);
      }
      return;
    }

    int mid = ini + (fim - ini) / 2;

    build(2 * pos + 1, ini, mid, v);
    build(2 * pos + 2, mid + 1, fim, v);

    tree[pos] = merge(tree[2 * pos + 1], tree[2 * pos + 2]);
  }

  // how many elements greater than val in vector v
  int search(vector<int>& v, int val) {
    auto it = upper_bound(v.begin(), v.end(), val);
    if (it == v.end()) return 0;
    return (int)v.size() - (it - v.begin());
  }

  // how many elements greater than val in the range (p, q)
  int query(int pos, int ini, int fim, int p, int q, int val) {
    if (fim < p || ini > q) {
      return 0;
    }

    if (ini >= p && fim <= q) {
      return search(tree[pos], val);
    }

    int mid = ini + (fim - ini) / 2;
    return query(2 * pos + 1, ini, mid, p, q, val) + query(2 * pos + 2, mid + 1, fim, p, q, val);
  }

  int query(int p, int q, int val) {
    return query(0, 0, n - 1, p, q, val);
  }
};
```

2.10 Segment With Maximum Sum

```
// Description:
// Query - get sum of segment that is maximum among all segments
// E.g
// Array: 5 -4 4 3 -5
// Maximum segment sum: 8 because 5 + (-4) + 4 + 3 = 8
// Update - update element at position id to a value val

// Problem:
// https://codeforces.com/edu/course/2/lesson/4/2/practice/contest/273278/problem/A

// Complexity:
// O(log n) for both query and update

// How to use:
// Segtree seg = Segtree(n);
// seg.build(v);

// Notes
// The maximum segment sum can be a negative number
// In that case, taking zero elements is the best choice
// So we need to take the maximum between 0 and the query
// max(0LL, seg.query(0, n).max_seg)

using ll = long long;

typedef ll ftype_node;

struct Node {
    ftype_node max_seg;
    ftype_node pref;
    ftype_node suf;
    ftype_node sum;

    Node(ftype_node max_seg, ftype_node pref, ftype_node suf, ftype_node sum) : max_seg(max_seg), pref(pref), suf(suf), sum(sum) {};
};

typedef Node ftype;

struct Segtree {
    vector<ftype> seg;
    int n;
    const ftype NEUTRAL = Node(0, 0, 0, 0);

    Segtree(int n) {
        int sz = 1;
        // potencia de dois mais proxima
        while (sz < n) sz *= 2;
        this->n = sz;

        // numero de nos da seg
        seg.assign(2*sz, NEUTRAL);
    }

    ftype f(ftype a, ftype b) {
        ftype_node max_seg = max({a.max_seg, b.max_seg, a.suf + b.pref});
        ftype_node pref = max(a.pref, a.sum + b.pref);
        ftype_node suf = max(b.suf, b.sum + a.suf);
        ftype_node sum = a.sum + b.sum;

        return Node(max_seg, pref, suf, sum);
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int id, int val) {
        if (ini > id || fim < id) {
            return;
        }

        if (ini == id && fim == id) {
            seg[pos] = Node(val, val, val, val);

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, id, val);
        update(d, m + 1, fim, id, val);

        seg[pos] = f(seg[e], seg[d]);
    }

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            // se a çãposio existir no array original
            // seg tamanho potencia de dois
            if (ini < (int)v.size()) {
                seg[pos] = Node(v[ini], v[ini], v[ini], v[ini]);
            }
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        build(e, ini, m, v);
        build(d, m + 1, fim, v);

        seg[pos] = f(seg[e], seg[d]);
    }

    ftype query(int p, int q) {
        return query(0, 0, n - 1, p, q);
    }

    void update(int id, int val) {
        update(0, 0, n - 1, id, val);
    }

    void build(vector<int> &v) {
        build(0, 0, n - 1, v);
    }

    void debug() {
        for (auto e : seg) {
            cout << e.max_seg << ' ' << e.pref << ' ' << e.suf << ' ' << e.sum << '\n';
        }
        cout << '\n';
    }
};
```

2.11 Minimum And Amount

```
// Description:
// Query - get minimum element in a range (l, r) inclusive
// and also the number of times it appears in that range
// Update - update element at position id to a value val

// Problem:
// https://codeforces.com/edu/course/2/lesson/4/1/practice/contest/273169/problem/C

// Complexity:
// O(log n) for both query and update

// How to use:
// Segtree seg = Segtree(n);
// seg.build(v);

#define pii pair<int, int>
#define mp make_pair
#define ff first
#define ss second

const int INF = 1e9+17;

typedef pii ftype;

struct Segtree {
    vector<ftype> seg;
    int n;
    const ftype NEUTRAL = mp(INF, 0);

    Segtree(int n) {
        int sz = 1;
        while (sz < n) sz *= 2;
        this->n = sz;

        seg.assign(2*sz, NEUTRAL);
    }

    ftype f(ftype a, ftype b) {
        if (a.ff < b.ff) return a;
        if (b.ff < a.ff) return b;

        return mp(a.ff, a.ss + b.ss);
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

```

```cpp
        return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int id, int val) {
        if (ini > id || fim < id) {
            return;
        }

        if (ini == id && fim == id) {
            seg[pos] = mp(val, 1);

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, id, val);
        update(d, m + 1, fim, id, val);

        seg[pos] = f(seg[e], seg[d]);
    }

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            if (ini < (int)v.size()) {
                seg[pos] = mp(v[ini], 1);
            }
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        build(e, ini, m, v);
        build(d, m + 1, fim, v);

        seg[pos] = f(seg[e], seg[d]);
    }

    ftype query(int p, int q) {
        return query(0, 0, n - 1, p, q);
    }

    void update(int id, int val) {
        update(0, 0, n - 1, id, val);
    }

    void build(vector<int> &v) {
        build(0, 0, n - 1, v);
    }

    void debug() {
        for (auto e : seg) {
            cout << e.ff << ' ' << e.ss << '\n';
        }
        cout << '\n';
    }
};
```

2.12 Dynamic Implicit Sparse

```cpp
// Description:
// Indexed at one

// When the indexes of the nodes are too big to be stored in an array
// and the queries need to be answered online so we can't sort the nodes and compress them
// we create nodes only when they are needed so there'll be (Q*log(MAX)) nodes
// where Q is the number of queries and MAX is the maximum index a node can assume

// Query - get sum of elements from range (l, r) inclusive
// Update - update element at position id to a value val

// Problem:
// https://cses.fi/problemset/task/1648

// Complexity:
// O(log n) for both query and update

// How to use:
// MAX is the maximum index a node can assume

// Segtree seg = Segtree(MAX);

typedef long long ftype;

const int MAX = 1e9+17;

struct Segtree {
    vector<ftype> seg, d, e;
    const ftype NEUTRAL = 0;
    int n;

    Segtree(int n) {
        this->n = n;
        create();
        create();
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int id, int val) {
        if (ini > id || fim < id) {
            return;
        }

        if (ini == fim) {
            seg[pos] = val;

            return;
        }

        int m = (ini + fim) >> 1;

        if (id <= m) {
            if (e[pos] == 0) e[pos] = create();
            update(e[pos], ini, m, id, val);
        } else {
            if (d[pos] == 0) d[pos] = create();
            update(d[pos], m + 1, fim, id, val);
        }

        seg[pos] = f(seg[e[pos]], seg[d[pos]]);
    }

    ftype query(int p, int q) {
        return query(1, 1, n, p, q);
    }

    void update(int id, int val) {
        update(1, 1, n, id, val);
    }
};
```

2.13 Range Query Point Update

```
// Description:
// Indexed at zero
// Query - get sum of elements from range (l, r) inclusive
// Update - update element at position id to a value val

// Problem:
// https://codeforces.com/edu/course/2/lesson/4/1/practice/contest/273169/problem/B

// Complexity:
// O(log n) for both query and update

// How to use:
// Segtree seg = Segtree(n);
// seg.build(v);

// Notes
// Change neutral element and f function to perform a different operation

// If you want to change the operations to point query and range update
// Use the same segtree, but perform the following operations
// Query - seg.query(0, id);
// Update - seg.update(l, v); seg.update(r + 1, -v);

typedef long long ftype;

struct Segtree {
    vector<ftype> seg;
    int n;
    const ftype NEUTRAL = 0;

    Segtree(int n) {
        int sz = 1;
        while (sz < n) sz *= 2;
        this->n = sz;

        seg.assign(2*sz, NEUTRAL);
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int id, int val) {
        if (ini > id || fim < id) {
            return;
        }

        if (ini == id && fim == id) {
            seg[pos] = val;

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, id, val);
        update(d, m + 1, fim, id, val);

        seg[pos] = f(seg[e], seg[d]);
    }

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            if (ini < (int)v.size()) {
                seg[pos] = v[ini];
            }
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        build(e, ini, m, v);
        build(d, m + 1, fim, v);

        seg[pos] = f(seg[e], seg[d]);
    }

    ftype query(int p, int q) {
        return query(0, 0, n - 1, p, q);
    }

    void update(int id, int val) {
        update(0, 0, n - 1, id, val);
    }

    void build(vector<int> &v) {
        build(0, 0, n - 1, v);
    }

    void debug() {
        for (auto e : seg) {
            cout << e << ' ';
        }
        cout << '\n';
    }
};
```

2.14 Segtree2d

```
// Description:
// Indexed at zero
// Given a N x M grid, where i represents the row and j the column, perform the following operations
// update(i, j) - update the value of grid[i][j]
// query(i1, j1, i2, j2) - return the sum of values inside the rectangle
// defined by grid[i1][j1] and grid[i2][j2] inclusive

// Problem:
// https://cses.fi/problemset/task/1739/

// Complexity:
// Time complexity:
// O(log N * log M) for both query and update
// O(N * M) for build
// Memory complexity:
// 4 * M * N

// How to use:
// Segtree2D seg = Segtree2D(n, m);
// vector<vector<int>> v(n, vector<int>(m));
// seg.build(v);

struct Segtree2D {
    const int MAXN = 1025;
    const int NEUTRAL = 0;
    int N, M;

    vector<vector<int>> seg;

    Segtree2D(int N, int M) {
        this->N = N;
        this->M = M;
        seg.assign(4*MAXN, vector<int>(4*MAXN, NEUTRAL));
    }

    int f(int a, int b) {
      return max(a, b);
    }

    void buildY(int noX, int lX, int rX, int noY, int lY, int rY, vector<vector<int>> &v){
        if(lY == rY){
            if(lX == rX){
                seg[noX][noY] = v[rX][rY];
            }else{
                seg[noX][noY] = f(seg[2*noX+1][noY], seg[2*noX+2][noY]);
            }
        }else{
            int m = (lY+rY)/2;

            buildY(noX, lX, rX, 2*noY+1, lY, m, v);
            buildY(noX, lX, rX, 2*noY+2, m+1, rY, v);

            seg[noX][noY] = f(seg[noX][2*noY+1], seg[noX][2*noY+2]);
        }
    }

    void buildX(int noX, int lX, int rX, vector<vector<int>> &v){
        if(lX != rX){
            int m = (lX+rX)/2;

            buildX(2*noX+1, lX, m, v);
            buildX(2*noX+2, m+1, rX, v);
        }

        buildY(noX, lX, rX, 0, 0, M - 1, v);
    }

    void updateY(int noX, int lX, int rX, int noY, int lY, int rY, int y){
        if(lY == rY){
            if(lX == rX){
                seg[noX][noY] = !seg[noX][noY];
            }else{
                seg[noX][noY] = seg[2*noX+1][noY] + seg[2*noX+2][noY];
            }
        }else{
            int m = (lY+rY)/2;

            if(y <= m){
                updateY(noX, lX, rX, 2*noY+1,lY, m, y);
            }else if(m < y){
                updateY(noX, lX, rX, 2*noY+2, m+1, rY, y);
            }

            seg[noX][noY] = seg[noX][2*noY+1] + seg[noX][2*noY+2];
        }
    }

    void updateX(int noX, int lX, int rX, int x, int y){
        int m = (lX+rX)/2;

        if(lX != rX){
            if(x <= m){
                updateX(2*noX+1, lX, m, x, y);
            }else if(m < x){
                updateX(2*noX+2, m+1, rX, x, y);
            }
        }

        updateY(noX, lX, rX, 0, 0, M - 1, y);
    }

    int queryY(int noX, int noY, int lY, int rY, int aY, int bY){
        if(aY <= lY && rY <= bY) return seg[noX][noY];

        int m = (lY+rY)/2;

        if(bY <= m) return queryY(noX, 2*noY+1, lY, m, aY, bY);
        if(m < aY) return queryY(noX, 2*noY+2, m+1, rY, aY, bY);

        return f(queryY(noX, 2*noY+1, lY, m, aY, bY), queryY(noX, 2*noY+2, m+1, rY, aY, bY));
    }

    int queryX(int noX, int lX, int rX, int aX, int bX, int aY, int bY){
        if(aX <= lX && rX <= bX) return queryY(noX, 0, 0, M - 1, aY, bY);

        int m = (lX+rX)/2;

        if(bX <= m) return queryX(2*noX+1, lX, m, aX, bX, aY, bY);
        if(m < aX) return queryX(2*noX+2, m+1, rX, aX, bX, aY, bY);
        return f(queryX(2*noX+1, lX, m, aX, bX, aY, bY), queryX(2*noX+2, m+1, rX, aX, bX, aY, bY));
    }

    void build(vector<vector<int>> &v) {
        buildX(0, 0, N - 1, v);
    }

    int query(int aX, int aY, int bX, int bY) {
        return queryX(0, 0, N - 1, aX, bX, aY, bY);
    }

    void update(int x, int y) {
        updateX(0, 0, N - 1, x, y);
    }
};
```

2.15 Lazy Addition To Segment

```
// Description:
// Query - get sum of elements from range (l, r) inclusive
// Update - add a value val to elementos from range (l, r) inclusive

// Problem:
// https://codeforces.com/edu/course/2/lesson/5/1/practice/contest/279634/problem/A

// Complexity:
// O(log n) for both query and update

// How to use:
// Segtree seg = Segtree(n);
// seg.build(v);

// Notes
// Change neutral element and f function to perform a different operation

const long long INF = 1e18+10;

typedef long long ftype;

struct Segtree {
    vector<ftype> seg;
    vector<ftype> lazy;
    int n;
    const ftype NEUTRAL = 0;
    const ftype NEUTRAL_LAZY = -1; // change to -INF if there are negative numbers

    Segtree(int n) {
        int sz = 1;
        while (sz < n) sz *= 2;
        this->n = sz;

        seg.assign(2*sz, NEUTRAL);
        lazy.assign(2*sz, NEUTRAL_LAZY);
    }

    ftype apply_lazy(ftype a, ftype b, int len) {
        if (b == NEUTRAL_LAZY) return a;
        if (a == NEUTRAL_LAZY) return b * len;
        else return a + b * len;
    }

    void propagate(int pos, int ini, int fim) {
        if (ini == fim) {
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
        lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);

        seg[e] = apply_lazy(seg[e], lazy[pos], m - ini + 1);
        seg[d] = apply_lazy(seg[d], lazy[pos], fim - m);

        lazy[pos] = NEUTRAL_LAZY;
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        propagate(pos, ini, fim);

        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int p, int q, int val) {
        propagate(pos, ini, fim);

        if (ini > q || fim < p) {
            return;
        }

        if (ini >= p && fim <= q) {
            lazy[pos] = apply_lazy(lazy[pos], val, 1);
            seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, p, q, val);
        update(d, m + 1, fim, p, q, val);

        seg[pos] = f(seg[e], seg[d]);
    }

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            if (ini < (int)v.size()) {
                seg[pos] = v[ini];
            }
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        build(e, ini, m, v);
        build(d, m + 1, fim, v);

        seg[pos] = f(seg[e], seg[d]);
    }

    ftype query(int p, int q) {
        return query(0, 0, n - 1, p, q);
    }

    void update(int p, int q, int val) {
        update(0, 0, n - 1, p, q, val);
    }

    void build(vector<int> &v) {
        build(0, 0, n - 1, v);
    }

    void debug() {
        for (auto e : seg) {
            cout << e << ' ';
        }
        cout << '\n';
        for (auto e : lazy) {
            cout << e << ' ';
        }
        cout << '\n';
        cout << '\n';
    }
};
```

2.16 Lazy Assignment To Segment

```
const long long INF = 1e18+10;

typedef long long ftype;

struct Segtree {
    vector<ftype> seg;
    vector<ftype> lazy;
    int n;
    const ftype NEUTRAL = 0;
    const ftype NEUTRAL_LAZY = -1; // Change to -INF if there are negative numbers

    Segtree(int n) {
        int sz = 1;
        // potencia de dois mais proxima
        while (sz < n) sz *= 2;
        this->n = sz;

        // numero de nos da seg
        seg.assign(2*sz, NEUTRAL);
        lazy.assign(2*sz, NEUTRAL_LAZY);
    }

    ftype apply_lazy(ftype a, ftype b, int len) {
        if (b == NEUTRAL_LAZY) return a;
        if (a == NEUTRAL_LAZY) return b * len;
        else return b * len;
    }

    void propagate(int pos, int ini, int fim) {
        if (ini == fim) {
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        lazy[e] = apply_lazy(lazy[e], lazy[pos], 1);
        lazy[d] = apply_lazy(lazy[d], lazy[pos], 1);

        seg[e] = apply_lazy(seg[e], lazy[pos], m - ini + 1);
        seg[d] = apply_lazy(seg[d], lazy[pos], fim - m);

        lazy[pos] = NEUTRAL_LAZY;
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        propagate(pos, ini, fim);

        if (ini >= p && fim <= q) {
            return seg[pos];
        }

        if (q < ini || p > fim) {
            return NEUTRAL;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        return f(query(e, ini, m, p, q), query(d, m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int p, int q, int val) {
        propagate(pos, ini, fim);

        if (ini > q || fim < p) {
            return;
        }

        if (ini >= p && fim <= q) {
            lazy[pos] = apply_lazy(lazy[pos], val, 1);
            seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);

            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        update(e, ini, m, p, q, val);
        update(d, m + 1, fim, p, q, val);

        seg[pos] = f(seg[e], seg[d]);
    }

    void build(int pos, int ini, int fim, vector<int> &v) {
        if (ini == fim) {
            // se a çãposio existir no array original
            // seg tamanho potencia de dois
            if (ini < (int)v.size()) {
                seg[pos] = v[ini];
            }
            return;
        }

        int e = 2*pos + 1;
        int d = 2*pos + 2;
        int m = ini + (fim - ini) / 2;

        build(e, ini, m, v);
        build(d, m + 1, fim, v);

        seg[pos] = f(seg[e], seg[d]);
    }

    ftype query(int p, int q) {
        return query(0, 0, n - 1, p, q);
    }

    void update(int p, int q, int val) {
        update(0, 0, n - 1, p, q, val);
    }

    void build(vector<int> &v) {
        build(0, 0, n - 1, v);
    }

    void debug() {
        for (auto e : seg) {
            cout << e << ' ';
        }
        cout << '\n';
        for (auto e : lazy) {
            cout << e << ' ';
        }
        cout << '\n';
        cout << '\n';
    }
};
```

2.17 Lazy Dynamic Implicit Sparse

```
// Description:
// Indexed at one

// When the indexes of the nodes are too big to be stored in an array
// and the queries need to be answered online so we can't sort the nodes and compress them
// we create nodes only when they are needed so there'll be (Q*log(MAX)) nodes
// where Q is the number of queries and MAX is the maximum index a node can assume

// Query - get sum of elements from range (l, r) inclusive
// Update - update element at position id to a value val

// Problem:
// https://oj.uz/problem/view/IZhO12_apple

// Complexity:
// O(log n) for both query and update

// How to use:
// MAX is the maximum index a node can assume
// Create a default null node
// Create a node to be the root of the segtree

// Segtree seg = Segtree(MAX);

const int MAX = 1e9+10;
const long long INF = 1e18+10;

typedef long long ftype;

struct Segtree {
    vector<ftype> seg, d, e, lazy;
    const ftype NEUTRAL = 0;
    const ftype NEUTRAL_LAZY = -1; // change to -INF if the elements can be negative
    int n;

    Segtree(int n) {
        this->n = n;
        create();
        create();
    }

    ftype apply_lazy(ftype a, ftype b, int len) {
        if (b == NEUTRAL_LAZY) return a;
        else return b * len; // change to a + b * len to add to an element instead of updating it
    }

    void propagate(int pos, int ini, int fim) {
        if (seg[pos] == 0) return;

        if (ini == fim) {
            return;
        }

        int m = (ini + fim) >> 1;

        if (e[pos] == 0) e[pos] = create();
        if (d[pos] == 0) d[pos] = create();

        lazy[e[pos]] = apply_lazy(lazy[e[pos]], lazy[pos], 1);
        lazy[d[pos]] = apply_lazy(lazy[d[pos]], lazy[pos], 1);

        seg[e[pos]] = apply_lazy(seg[e[pos]], lazy[pos], m - ini + 1);
        seg[d[pos]] = apply_lazy(seg[d[pos]], lazy[pos], fim - m);

        lazy[pos] = NEUTRAL_LAZY;
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        lazy.push_back(-1);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        propagate(pos, ini, fim);
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
    }

    void update(int pos, int ini, int fim, int p, int q, int val) {
        propagate(pos, ini, fim);
        if (ini > q || fim < p) {
            return;
        }

        if (ini >= p && fim <= q) {
            lazy[pos] = apply_lazy(lazy[pos], val, 1);
            seg[pos] = apply_lazy(seg[pos], val, fim - ini + 1);

            return;
        }

        int m = (ini + fim) >> 1;

        if (e[pos] == 0) e[pos] = create();
        update(e[pos], ini, m, p, q, val);

        if (d[pos] == 0) d[pos] = create();
        update(d[pos], m + 1, fim, p, q, val);

        seg[pos] = f(seg[e[pos]], seg[d[pos]]);
    }

    ftype query(int p, int q) {
        return query(1, 1, n, p, q);
    }

    void update(int p, int q, int val) {
        update(1, 1, n, p, q, val);
    }
};
```

2.18 Persistent

```
// Description:
// Persistent segtree allows for you to save the different versions of the segtree between each update
// Indexed at one
// Query - get sum of elements from range (l, r) inclusive
// Update - update element at position id to a value val

// Problem:
// https://cses.fi/problemset/task/1737/

// Complexity:
// O(log n) for both query and update

// How to use:
// vector<int> raiz(MAX); // vector to store the roots of each version
// Segtree seg = Segtree(INF);
// raiz[0] = seg.create(); // null node
// curr = 1; // keep track of the last version

// raiz[k] = seg.update(raiz[k], idx, val); // updating version k
// seg.query(raiz[k], l, r) // querying version k
// raiz[++curr] = raiz[k]; // create a new version based on version k

const int MAX = 2e5+17;
const int INF = 1e9+17;

typedef long long ftype;

struct Segtree {
    vector<ftype> seg, d, e;
    const ftype NEUTRAL = 0;
    int n;

    Segtree(int n) {
        this->n = n;
    }

    ftype f(ftype a, ftype b) {
        return a + b;
    }

    ftype create() {
        seg.push_back(0);
        e.push_back(0);
        d.push_back(0);
        return seg.size() - 1;
    }

    ftype query(int pos, int ini, int fim, int p, int q) {
        if (q < ini || p > fim) return NEUTRAL;
        if (pos == 0) return 0;
        if (p <= ini && fim <= q) return seg[pos];
        int m = (ini + fim) >> 1;
        return f(query(e[pos], ini, m, p, q), query(d[pos], m + 1, fim, p, q));
    }

    int update(int pos, int ini, int fim, int id, int val) {
        int novo = create();

        seg[novo] = seg[pos];
        e[novo] = e[pos];
        d[novo] = d[pos];

        if (ini == fim) {
            seg[novo] = val;
            return novo;
        }

        int m = (ini + fim) >> 1;

        if (id <= m) e[novo] = update(e[novo], ini, m, id, val);
        else d[novo] = update(d[novo], m + 1, fim, id, val);

        seg[novo] = f(seg[e[novo]], seg[d[novo]]);

        return novo;
    }

    ftype query(int pos, int p, int q) {
        return query(pos, 1, n, p, q);
    }

    int update(int pos, int id, int val) {
        return update(pos, 1, n, id, val);
    }
};
```

3 Strings

3.1 Lcs

```
// Description:
// Finds the longest common subsquence between two string

// Problem:
// https://codeforces.com/gym/103134/problem/B

// Complexity:
```

```
// O(mn) where m and n are the length of the strings

string lcsAlgo(string s1, string s2, int m, int n) {
  int LCS_table[m + 1][n + 1];

  for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
      if (i == 0 || j == 0)
        LCS_table[i][j] = 0;
      else if (s1[i - 1] == s2[j - 1])
        LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
      else
        LCS_table[i][j] = max(LCS_table[i - 1][j], LCS_table[i][j - 1]);
    }
  }

  int index = LCS_table[m][n];
  char lcsAlgo[index + 1];
  lcsAlgo[index] = '\0';

  int i = m, j = n;
  while (i > 0 && j > 0) {
    if (s1[i - 1] == s2[j - 1]) {
      lcsAlgo[index - 1] = s1[i - 1];
      i--;
      j--;
      index--;
    }

    else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
      i--;
    else
      j--;
  }

  return lcsAlgo;
}
```

3.2 Z-function

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    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;
    }
    return z;
}
```

3.3 Hash2

```
// Hashed String {{{
class HashedString {
  static const int M = (1LL << 61) - 1;
  static const int B;
  static vector<int> pow;

  int N;
  vector<int> p_hash;

  __int128 mul(int a, int b) { return (__int128)a * b; }
  int mod_mul(int a, int b) { return mul(a, b) % M; }

  public:
  explicit HashedString(string const& s) {
    while (size(pow) < size(s) + 1) pow.push_back(mod_mul(pow.back(), B));

    p_hash.resize(size(s) + 1);
    p_hash[0] = 0;
    for (int i = 0; i < size(s); i++)
      p_hash[i + 1] = (mul(p_hash[i], B) + s[i]) % M;
  }

  int get_hash(int l, int r) {
    int raw_val = p_hash[r + 1] - mod_mul(p_hash[l], pow[r - l + 1]);
    return (raw_val + M) % M;
  }

  int prefix(int len) { return get_hash(0, len-1); }
  int suffix(int len) { return get_hash(N-len, N-1); }
  int whole() { return get_hash(0, N-1); }
  int substr(int l, int len) {
    int r = l+len-1;
    r = min(r, N-1);
    return get_hash(l, r);
  }
};
vector<int> HashedString::pow{1};
mt19937 rng((uint32_t)chrono::steady_clock::now().time_since_epoch().count());
const int HashedString::B = uniform_int_distribution<int>(0, M - 1)(rng);
//}}}
```

3.4 Trie

```
const int K = 26;

struct Vertex {
    int next[K];
    bool output = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].output = true;
}

int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
```

```cpp
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
```

3.5 Generate All Permutations

```cpp
vector<string> generate_permutations(string s) {
    int n = s.size();
    vector<string> ans;

    sort(s.begin(), s.end());

    do {
        ans.push_back(s);
    } while (next_permutation(s.begin(), s.end()));

    return ans;
}
```

3.6 Kmp

```cpp
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    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;
    }
    return pi;
}
```

3.7 Hash

```cpp
// Description:
// Turns a string into a integer.
// If the hash is different then the strings are different.
// If the hash is the same the strings may be different.

// Problem:
// https://codeforces.com/gym/104518/problem/I

// Complexity:
// O(n) to calculate the hash
// O(1) to query

// Notes:
// Primes 1000000007, 1000041323, 100663319, 201326611, 1000015553, 1000028537

struct Hash {
    const ll P = 31;
    int n; string s;
    vector<ll> h, hi, p;
    Hash() {}
    Hash(string s): s(s), n(s.size()), h(n), hi(n), p(n) {
        for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1) % MOD;
        for (int i=0;i<n;i++)
            h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
        for (int i=n-1;i>=0;i--)
            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P) % MOD;
    }
    int query(int l, int r) {
        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
    int query_inv(int l, int r) {
        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l+1] % MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
};
```

3.8 Generate All Sequences Length K

```cpp
// gera todas as ípossveis êsequncias usando as letras em set (de comprimento n) e que tenham tamanho k
// sequence = ""
vector<string> generate_sequences(char set[], string sequence, int n, int k) {
    if (k == 0){
        return { sequence };
    }

    vector<string> ans;
    for (int i = 0; i < n; i++) {
        auto aux = generate_sequences(set, sequence + set[i], n, k - 1);
        ans.insert(ans.end(), aux.begin(), aux.end());
        // for (auto e : aux) ans.push_back(e);
    }

    return ans;
}
```

3.9 Suffix Array

```cpp
// Description:
// Suffix array is an array with the indexes of the starting letter of every
// suffix in an array sorted in lexicographical order.

// Problem:
// https://codeforces.com/edu/course/2/lesson/2/1/practice/contest/269100/problem/A

// Complexity:
// O(n log n) with radix sort
// O(n log ^ 2 n) with regular sort

// Notes:
// Relevant Problems
// Substring search: Queries to know whether a given substring is present in a string
// Binary search for the first suffix that is greater or equal
// O(log n |p|) where |p| is the total size of the substrings queried
```

```
//
// Substring size: Queries to know how many times a given substring appears in a string
// Binary search both for first and last that is greater or equal
//
// Number of different substrings:
// A given suffix gives sz new substrings being sz the size of the suffix
// We can subtract the lcp (longest common prefix) to remove substrings
// that were already counted.
//
// Longest common substring between two strings:
// We can calculate the suffix array and lcp array of the two strings
// concantened with a character greater than $ and smaller than A (like ’&’)
// The answer will be the lcp between two consecutive suffixes that belong to different strings
// (index at suffix array <= size of the first array)

void radix_sort(vector<pair<pair<int, int>, int>>& a) {
  int n = a.size();
  vector<pair<pair<int, int>, int>> ans(n);

  vector<int> count(n);

  for (int i = 0; i < n; i++) {
    count[a[i].first.second]++;
  }

  vector<int> p(n);

  p[0] = 0;
  for (int i = 1; i < n; i++) {
    p[i] = p[i - 1] + count[i - 1];
  }

  for (int i = 0; i < n; i++) {
    ans[p[a[i].first.second]++] = a[i];
  }

  a = ans;

  count.assign(n, 0);

  for (int i = 0; i < n; i++) {
    count[a[i].first.first]++;
  }

  p.assign(n, 0);

  p[0] = 0;
  for (int i = 1; i < n; i++) {
    p[i] = p[i - 1] + count[i - 1];
  }

  for (int i = 0; i < n; i++) {
    ans[p[a[i].first.first]++] = a[i];
  }

  a = ans;
}

vector<int> p, c;

vector<int> suffix_array(string s) {
  int n = s.size();
  vector<pair<char, int>> a(n);
  p.assign(n, 0);
  c.assign(n, 0);

  for (int i = 0; i < n; i++) {
    a[i] = mp(s[i], i);
  }

  sort(a.begin(), a.end());

  for (int i = 0; i < n; i++) {
    p[i] = a[i].second;
  }

  c[p[0]] = 0;
  for (int i = 1; i < n; i++) {
    if (a[i].first == a[i - 1].first) c[p[i]] = c[p[i - 1]];
    else c[p[i]] = c[p[i - 1]] + 1;
  }

  int k = 0;
  while ((1 << k) < n) {
    vector<pair<pair<int, int>, int >> a(n);
    for (int i = 0; i < n; i++) {
      a[i] = mp(mp(c[i], c[(i + (1 << k)) % n]), i);
    }

    radix_sort(a);

    for (int i = 0; i < n; i++) {
      p[i] = a[i].second;
    }

    c[p[0]] = 0;
    for (int i = 1; i < n; i++) {
      if (a[i].first == a[i - 1].first) c[p[i]] = c[p[i - 1]];
      else c[p[i]] = c[p[i - 1]] + 1;
    }

    k++;
  }

  /* for (int i = 0; i < n; i++) {
    for (int j = p[i]; j < n; j++) {
      cout << s[j];
    }
    cout << ’\n’;
  } */

  return p;
}

// the first suffix will alway be $ the (n - 1)th character in the string
vector<int> lcp_array(string s) {
  int n = s.size();
  vector<int> ans(n);
  // minimum lcp
  int k = 0;
  for (int i = 0; i < n - 1; i++) {
    // indice in the suffix array p of suffix starting in i
    int pi = c[i];
    // start index of the previous suffix in suffix array
    int j = p[pi - 1];
    while (s[i + k] == s[j + k]) k++;
    ans[pi] = k;
    k = max(k - 1, 0);
  }

  return ans;
}
```

4 Algorithms

4.1 Lcs

```
// Longest Common Subsequence
//
// Computa a LCS entre dois arrays usando
// o algoritmo de Hirschberg para recuperar
//
// O(n*m), O(n+m) de memoria

int lcs_s[MAX], lcs_t[MAX];
int dp[2][MAX];

// dp[0][j] = max lcs(s[li...ri], t[lj, lj+j])
void dp_top(int li, int ri, int lj, int rj) {
  memset(dp[0], 0, (rj-lj+1)*sizeof(dp[0][0]));
  for (int i = li; i <= ri; i++) {
    for (int j = rj; j >= lj; j--)
      dp[0][j - lj] = max(dp[0][j - lj],
      (lcs_s[i] == lcs_t[j]) + (j > lj ? dp[0][j-1 - lj] : 0));
    for (int j = lj+1; j <= rj; j++)
      dp[0][j - lj] = max(dp[0][j - lj], dp[0][j-1 - lj]);
  }
}

// dp[1][j] = max lcs(s[li...ri], t[lj+j, rj])
void dp_bottom(int li, int ri, int lj, int rj) {
  memset(dp[1], 0, (rj-lj+1)*sizeof(dp[1][0]));
  for (int i = ri; i >= li; i--) {
    for (int j = lj; j <= rj; j++)
      dp[1][j - lj] = max(dp[1][j - lj],
      (lcs_s[i] == lcs_t[j]) +  (j < rj ? dp[1][j+1 - lj] : 0));
    for (int j = rj-1; j >= lj; j--)
      dp[1][j - lj] = max(dp[1][j - lj], dp[1][j+1 - lj]);
  }
}

void solve(vector<int>& ans, int li, int ri, int lj, int rj) {
  if (li == ri){
    for (int j = lj; j <= rj; j++)
      if (lcs_s[li] == lcs_t[j]){
        ans.push_back(lcs_t[j]);
        break;
      }
    return;
  }
  if (lj == rj){
    for (int i = li; i <= ri; i++){
      if (lcs_s[i] == lcs_t[lj]){
        ans.push_back(lcs_s[i]);
        break;
      }
    }
    return;
  }
  int mi = (li+ri)/2;
  dp_top(li, mi, lj, rj), dp_bottom(mi+1, ri, lj, rj);

  int j_ = 0, mx = -1;

  for (int j = lj-1; j <= rj; j++) {
    int val = 0;
    if (j >= lj) val += dp[0][j - lj];
    if (j < rj) val += dp[1][j+1 - lj];

    if (val >= mx) mx = val, j_ = j;
  }
  if (mx == -1) return;
  solve(ans, li, mi, lj, j_), solve(ans, mi+1, ri, j_+1, rj);
}

vector<int> lcs(const vector<int>& s, const vector<int>& t) {
  for (int i = 0; i < s.size(); i++) lcs_s[i] = s[i];
  for (int i = 0; i < t.size(); i++) lcs_t[i] = t[i];
  vector<int> ans;
  solve(ans, 0, s.size()-1, 0, t.size()-1);
  return ans;
}
```

4.2 Biggest K

```
// Description: Gets sum of k biggest or k smallest elements in an array

// Problem: https://atcoder.jp/contests/abc306/tasks/abc306_e

// Complexity: O(log n)

struct SetSum {
    ll s = 0;
    multiset<ll> mt;
    void add(ll x){
        mt.insert(x);
        s += x;
    }
    int pop(ll x){
        auto f = mt.find(x);
        if(f == mt.end()) return 0;
        mt.erase(f);
        s -= x;
        return 1;
    }
};

struct BigK {
    int k;
    SetSum gt, mt;
    BigK(int _k){
        k = _k;
    }
    void balancear(){
        while((int)gt.mt.size() < k && (int)mt.mt.size()){
            auto p = (prev(mt.mt.end()));
            gt.add(*p);
            mt.pop(*p);
        }
        while((int)mt.mt.size() && (int)gt.mt.size() &&
        *(gt.mt.begin()) < *(prev(mt.mt.end())) ){
            ll u = *(gt.mt.begin());
            ll v = *(prev(mt.mt.end()));
            gt.pop(u); mt.pop(v);
            gt.add(v); mt.add(u);
        }
    }
    void add(ll x){
        mt.add(x);
        balancear();
    }
    void rem(ll x){
        //x = -x;
        if(mt.pop(x) == 0)
            gt.pop(x);
        balancear();
    }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int n, k, q; cin >> n >> k >> q;

    BigK big = BigK(k);

    int arr[n] = {};

    while (q--) {
        int pos, num; cin >> pos >> num;
        pos--;
        big.rem(arr[pos]);
        arr[pos] = num;
        big.add(arr[pos]);

        cout << big.gt.s << '\n';
    }

    return 0;
}
```

4.3 Lis

```
// Returns the size of the sequence
int lis(vector<int> const& a) {
    int n = a.size();
    vector<int> d(n, 1);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i])
                d[i] = max(d[i], d[j] + 1);
        }
    }

    int ans = d[0];
    for (int i = 1; i < n; i++) {
        ans = max(ans, d[i]);
    }
    return ans;
}

// Returns the sequence
template<typename T> vector<T> lis(vector<T>& v) {
  int n = v.size(), m = -1;
  vector<T> d(n+1, INF);
  vector<int> l(n);
  d[0] = -INF;

  for (int i = 0; i < n; i++) {
    // Para non-decreasing use upper_bound()
    int t = lower_bound(d.begin(), d.end(), v[i]) - d.begin();
    d[t] = v[i], l[i] = t, m = max(m, t);
  }

  int p = n;
  vector<T> ret;
  while (p--) if (l[p] == m) {
    ret.push_back(v[p]);
    m--;
  }
  reverse(ret.begin(),ret.end());

  return ret;
}
```

4.4 Binary Search First True

```
int first_true(int lo, int hi, function<bool(int)> f) {
  hi++;
  while (lo < hi) {
    int mid = lo + (hi - lo) / 2;
    if (f(mid)) {
      hi = mid;
    } else {
      lo = mid + 1;
    }
  }
  return lo;
}
```

4.5 Subsets

```
void subsets(vector<int>& nums){
  int n = nums.size();
  int powSize = 1 << n;

  for(int counter = 0; counter < powSize; counter++){
    for(int j = 0; j < n; j++){
      if((counter & (1LL << j)) != 0) {
        cout << nums[j] << ' ';
      }
    }
    cout << '\n';
  }
}
```

4.6 Ternary Search

```
double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);      //evaluates the function at m1
        double f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                    //return the maximum of f(x) in [l, r]
}
```

4.7 Binary Search Last True

```
int last_true(int lo, int hi, function<bool(int)> f) {
  lo--;
  while (lo < hi) {
    int mid = lo + (hi - lo + 1) / 2;
    if (f(mid)) {
      lo = mid;
    } else {
      hi = mid - 1;
    }
  }
  return lo;
}
```

4.8 Delta-encoding

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    int n, q;
    cin >> n >> q;
    int [n];
    int delta[n+2];

    while(q--){
        int l, r, x;
        cin >> l >> r >> x;
        delta[l] += x;
        delta[r+1] -= x;
    }

    int curr = 0;
    for(int i=0; i < n; i++){
        curr += delta[i];
        v[i] = curr;
    }

    for(int i=0; i< n; i++){
        cout << v[i] << ' ';
    }
    cout << '\n';

    return 0;
}
```

4.9 Binary Search Real

```
int cnt = 100;
ld l = 1e-7, r = 1e6+10;
while (cnt--) {
    ld mid = (l + r) / 2;

    if (f(mid)) {
        l = mid;
    } else {
        r = mid;
    }
}
```

5 Template

5.1 Template

```
#include <bits/stdc++.h>
using namespace std;

#define int long long
#define optimize std::ios::sync_with_stdio(false); cin.tie(NULL);
#define vi vector<int>
#define ll long long
#define pb push_back
#define mp make_pair
#define ff first
#define ss second
#define pii pair<int, int>
#define MOD 1000000007
#define sqr(x) ((x) * (x))
#define all(x) (x).begin(), (x).end()
#define FOR(i, j, n) for (int i = j; i < n; i++)
#define qle(i, n) (i == n ? "\n" : " ")
#define endl "\n"
const int oo = 1e9;
const int MAX = 1e6;

int32_t main(){ optimize;

    return 0;
}
```

5.2 Template Clean

```
// Notes:
// Compile and execute
// g++ teste.cpp -o teste -std=c++17
// ./teste < teste.txt

// Print with precision
// cout << fixed << setprecision(12) << value << endl;

// File as input and output
// freopen("input.txt", "r", stdin);
// freopen("output.txt", "w", stdout);

#include <bits/stdc++.h>
using namespace std;

#define pb push_back
#define mp make_pair
#define mt make_tuple
#define ff first
#define ss second
#define ld long double
#define ll long long
#define int long long
#define pii pair<int, int>
#define tii tuple<int, int, int>

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);



    return 0;
}
```

6 Math

6.1 Crt

```
ll crt(const vector<pair<ll, ll>> &vet){
    ll ans = 0, lcm = 1;
    ll a, b, g, x, y;
    for(const auto &p : vet) {
        tie(a, b) = p;
        tie(g, x, y) = gcd(lcm, b);
        if((a - ans) % g != 0) return -1; // no solution
        ans = ans + x * ((a - ans) / g) % (b / g) * lcm;
        lcm = lcm * (b / g);
        ans = (ans % lcm + lcm) % lcm;
    }
    return ans;
}
```

6.2 Check If Bit Is On

```
// msb de 0 é undefined
#define msb(n) (32 - __builtin_clz(n))
// #define msb(n) (64 - __builtin_clzll(n) )
// popcount
#define popcount(x) __builtin_popcountll((unsigned ll)x)
// turn bit off

bool bit_on(int n, int bit) {
    if(1 & (n >> bit)) return true;
    else return false;
}
```

6.3 To Decimal

```
const string digits { "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" };

long long to_decimal(const string& rep, long long base) {
  long long n = 0;

  for (auto c : rep) {
    // if the number can't be represented in this base
    if (c > digits[base - 1]) return -1;
    n *= base;
    n += digits.find(c);
  }

  return n;
}
```

6.4 Multiplicative Inverse

```
ll extend_euclid(ll a, ll b, ll &x, ll &y) {
    if (a == 0)
    {
        x = 0; y = 1;
        return b;
    }
    ll x1, y1;
    ll d = extend_euclid(b%a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}

// gcd(a, m) = 1 para existir solucao
// ax + my = 1, ou a*x = 1 (mod m)
ll inv_gcd(ll a, ll m) { // com gcd
  ll x, y;
  extend_euclid(a, m, x, y);
  return (((x % m) +m) %m);
}

ll inv(ll a, ll phim) { // com phi(m), se m for primo entao phi(m) = p-1
  ll e = phim-1;
  return fexp(a, e, MOD);
}
```

6.5 Fft

```
/ FFT {{{
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> &A, bool invert) {
  int N = size(A);

  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) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    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 (invert) {
    for (auto &x : A)
      x /= N;
  }
}

vector<int> multiply(vector<int> const& A, vector<int> const& B) {
  vector<cd> fa(begin(A), end(A)), fb(begin(B), end(B));
  int N = 1;
  while (N < size(A) + size(B))
    N <<= 1;
  fa.resize(N);
  fb.resize(N);

  fft(fa, false);
  fft(fb, false);
  for (int i = 0; i < N; i++)
    fa[i] *= fb[i];
  fft(fa, true);

  vector<int> result(N);
  for (int i = 0; i < N; i++)
    result[i] = round(fa[i].real());
  return result;
}
// }}}
```

6.6 Function Root

```
const ld EPS1 = 1e-9; // iteration precision error
const ld EPS2 = 1e-4; // output precision error

ld f(ld x) {
  // exp(-x) == e^(-x)
  return p * exp(-x) + q * sin(x) + r * cos(x) + s * tan(x) + t * x * x + u;
}

ld root(ld a, ld b) {
  while (b - a >= EPS1) {
    ld c = (a + b) / 2.0;
    ld y = f(c);

    if (y < 0) b = c;
    else a = c;
  }

  return (a + b) / 2;
}

int main() {
  ld ans = root(0, 1);
  if (abs(f(ans)) <= EPS2) cout << fixed << setprecision(4) << ans << '\n';
  else  cout << "No solution\n";

  return 0;
}
```

6.7 Set Operations

```
// Complexity;
// O(n * m) being n and m the sizes of the two sets
// 2*(count1+count2)-1 (where countX is the distance between firstX and lastX):

vector<int> res;
set_union(s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(res, res.begin()));
set_intersection(s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(res, res.begin()));
// present in the first set, but not in the second
set_difference(s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(res, res.begin()));
// present in one of the sets, but not in the other
set_symmetric_difference(s1.begin(), s1.end(), s2.begin(), s2.end(), inserter(res, res.begin()));
```

6.8 Subsets

```
void subsets(vector<int>& nums){
  int n = nums.size();
  int powSize = 1 << n;

  for(int counter = 0; counter < powSize; counter++) {
    for(int j = 0; j < n; j++) {
      if((counter & (1LL << j)) != 0) {
        cout << nums[j] << ' ';
      }
      cout << '\n';
    }
  }
}
```

6.9 Pascalsrule Stifel

```
// Description:
// Calculates a binomial n chooses k based on the value of a previous binomial.

// Complexity:
// O(n * k)

vector<vector<int>> comb(MAX + 1, vector<int>(MAX + 1, 0));

for (int n = 0; n <= MAX; n++) {
  comb[n][0] = 1;
}

for (int n = 1; n <= MAX; n++) {
  for (int k = 1; k <= n; k++) {
    comb[n][k] = comb[n - 1][k - 1] + comb[n - 1][k];
  }
}
```

6.10 Sieve Of Eratosthenes

```
vector<bool> is_prime(MAX, true);
vector<int> primes;

void sieve() {
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i < MAX; i++) {
        if (is_prime[i]) {
            primes.push_back(i);

            for (int j = i + i; j < MAX; j += i)
                is_prime[j] = false;
        }
    }
}
```

6.11 Linear Diophantine Equation

```
// int a, b, c, x1, x2, y1, y2; cin >> a >> b >> c >> x1 >> x2 >> y1 >> y2;
// int ans = -1;
// if (a == 0 && b == 0) {
//     if (c != 0) ans = 0;
//     else ans = (x2 - x1 + 1) * (y2 - y1 + 1);
// }
// else if (a == 0) {
//     if (c % b == 0 && y1 <= c / b && y2 >= c / b) ans = (x2 - x1 + 1);
//     else ans = 0;
// }
// else if (b == 0) {
//     if (c % a == 0 && x1 <= c / a && x2 >= c / a) ans = (y2 - y1 + 1);
//     else ans = 0;
// }

// Careful when a or b are negative or zero

// if (ans == -1) ans =  find_all_solutions(a, b, c, x1, x2, y1, y2);
// cout << ans << '\n';

// Problems:
// https://www.spoj.com/problems/CEQU/
// http://codeforces.com/problemsets/acmsguru/problem/99999/106

// consider trivial case a or b is 0
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

// x and y are one solution and g is the gcd, all passed as reference
// minx <= x <= maxx miny <= y <= maxy
bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

void shift_solution(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

// return number of solutions in the interval
int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
```

```
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

6.12 Mobius

```
vector<int> m(MAXN, 0), lp(MAXN, 0);
m[1] = 1;
for (int i = 2; i < MAXN; ++i) {
    if (!lp[i]) for (int j = i; j < MAXN; j += i)
        if (!lp[j]) lp[j] = i;
    m[i] = [&](int x) {
        int cnt = 0;
        while (x > 1) {
            int k = 0, d = lp[x];
            while (x % d == 0) {
                x /= d;
                ++k;
                if (k > 1) return 0;
            }
            ++cnt;
        }
        if (cnt & 1) return -1;
        return 1;
    }(i);
}
```

6.13 Representation Arbitrary Base

```
const string digits { "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" };

string representation(int n, int b) {
  string rep;

  do {
    rep.push_back(digits[n % b]);
    n /= b;
  } while (n);

  reverse(rep.begin(), rep.end());

  return rep;
}
```

6.14 Horner Algorithm

```
// Description:
// Evaluates y = f(x)

// Problem:
// https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=439

// Complexity:
// O(n)

using polynomial = std::vector<int>;

polynomial p {6, -5, 2}; // p(x) = x^2 - 5x + 6;

int degree(const polynomial& p) {
  return p.size() - 1;
}

int evaluate(const polynomial& p, int x) {
  int y = 0, N = degree(p);

  for (int i = N; i >= 0; --i) {
    y *= x;
    y += p[i];
  }

  return y;
}
```

6.15 Prime Factors

```
vector<pair<long long, int>> fatora(long long n) {
  vector<pair<long long, int>> ans;
  for(long long p = 2; p*p <= n; p++) {
    if(n % p == 0) {
      int expoente = 0;
      while(n % p == 0) {
        n /= p;
        expoente++;
      }
      ans.emplace_back(p, expoente);
    }
  }
  if(n > 1) ans.emplace_back(n, 1);
  return ans;
}
```

6.16 Binary To Decimal

```
int binary_to_decimal(long long n) {
  int dec = 0, i = 0, rem;

  while (n!=0) {
    rem = n % 10;
    n /= 10;
    dec += rem * pow(2, i);
    ++i;
  }

  return dec;
}

long long decimal_to_binary(int n) {
  long long bin = 0;
  int rem, i = 1;

  while (n!=0) {
    rem = n % 2;
    n /= 2;
    bin += rem * i;
    i *= 10;
  }

  return bin;
}
```

6.17 Matrix Exponentiation

```
// Description:
// Calculate the nth term of a linear recursion

// Example Fibonacci:
// Given a linear recurrence, for example fibonacci
// F(n) = n, x <= 1
// F(n) = F(n - 1) + F(n - 2), x > 1

// The recurrence has two terms, so we can build a matrix 2 x 1 so that
// n + 1 = transition * n

// (2 x 1) = (2 x 2) * (2 x 1)
// F(n)       = a b * F(n - 1)
// F(n - 1)     c d   F(n - 2)

// Another Example:
// Given a grid 3 x n, you want to color it using 3 distinct colors so that
// no adjacent place has the same color. In how many different ways can you do that?
// There are 6 ways for the first column to be colored using 3 distinct colors
// ans 6 ways using 2 equal colors and 1 distinct one

// Adding another column, there are:
// 3 ways to go from 2 equal to 2 equal
// 2 ways to go from 2 equal to 3 distinct
// 2 ways to go from 3 distinct to 2 equal
// 2 ways to go from 3 distinct to 3 distinct

// So we star with matrix 6 6 and multiply it by the transition 3 2 and get 18 12
//                           6 6
               2 2          12 12
// the we can exponentiate this matrix to find the nth column

// Problem:
// https://cses.fi/problemset/task/1722/

// Complexity:
// O(log n)

// How to use:
// vector<vector<ll>> v = {{1, 1}, {1, 0}};
// Matriz transition = Matriz(v);
// cout << fexp(transition, n)[0][1] << '\n';

using ll = long long;

const int MOD = 1e9+7;

struct Matriz{
    vector<vector<ll>> mat;
    int rows, columns;

    vector<ll> operator[](int i){
        return mat[i];
    }

    Matriz(vector<vector<ll>>& matriz){
        mat = matriz;
        rows = mat.size();
        columns = mat[0].size();
    }

    Matriz(int row, int column, bool identity=false){
        rows = row;  columns = column;
        mat.assign(rows, vector<ll>(columns, 0));
        if(identity) {
            for(int i = 0; i < min(rows,columns); i++) {
                mat[i][i] = 1;
            }
        }
    }

    Matriz operator * (Matriz a) {
        assert(columns == a.rows);
        vector<vector<ll>> resp(rows, vector<ll>(a.columns, 0));

        for(int i = 0; i < rows; i++){
            for(int j = 0; j < a.columns; j++){
                for(int k = 0; k < a.rows; k++){
                    resp[i][j] = (resp[i][j] + (mat[i][k] * 1LL * a[k][j]) % MOD) % MOD;
                }
            }
        }
        return Matriz(resp);
    }

    Matriz operator + (Matriz a) {
        assert(rows == a.rows && columns == a.columns);
        vector<vector<ll>> resp(rows, vector<ll>(columns,0));
        for(int i = 0; i < rows; i++){
            for(int j = 0; j < columns; j++){
                resp[i][j] = (resp[i][j] + mat[i][j] + a[i][j]) % MOD;
            }
        }
        return Matriz(resp);
    }
};

Matriz fexp(Matriz base, ll exponent){
    Matriz result = Matriz(base.rows, base.rows, 1);
    while(exponent > 0){
        if(exponent & 1LL) result = result * base;
        base = base * base;
        exponent  = exponent >> 1;
    }
    return result;
}
```

6.18 Fast Exponentiation

```
ll fexp(ll b, ll e, ll mod) {
    ll res = 1;
    b %= mod;
    while(e){
        if(e & 1LL)
            res = (res * b) % mod;
        e = e >> 1LL;
        b = (b * b) % mod;
    }
    return res;
}
```

6.19 Divisors

```
vector<long long> all_divisors(long long n) {
  vector<long long> ans;
  for(long long a = 1; a*a <= n; a++){
    if(n % a == 0) {
      long long b = n / a;
      ans.push_back(a);
      if(a != b) ans.push_back(b);
    }
  }
  sort(ans.begin(), ans.end());
  return ans;
}
```

6.20 Ntt

```
// Aritmetica Modular
//
// O mod tem q ser primo

template<int p> struct mod_int {
  ll expo(ll b, ll e) {
    ll ret = 1;
    while (e) {
      if (e % 2) ret = ret * b % p;
      e /= 2, b = b * b % p;
    }
    return ret;
  }
  ll inv(ll b) { return expo(b, p-2); }

  using m = mod_int;
  int v;
  mod_int() : v(0) {}
  mod_int(ll v_) {
    if (v_ >= p or v_ <= -p) v_ %= p;
    if (v_ < 0) v_ += p;
    v = v_;
  }
  m& operator +=(const m& a) {
    v += a.v;
    if (v >= p) v -= p;
    return *this;
  }
  m& operator -=(const m& a) {
    v -= a.v;
    if (v < 0) v += p;
    return *this;
  }
  m& operator *=(const m& a) {
    v = v * ll(a.v) % p;
    return *this;
  }
  m& operator /=(const m& a) {
    v = v * inv(a.v) % p;
    return *this;
  }
  m operator -(){ return m(-v); }
  m& operator ^=(ll e) {
    if (e < 0) {
      v = inv(v);
      e = -e;
    }
    v = expo(v, e);
    // possivel otimizacao:
    // cuidado com 0^0
    // v = expo(v, e%(p-1));
    return *this;
  }
  bool operator ==(const m& a) { return v == a.v; }
  bool operator !=(const m& a) { return v != a.v; }

  friend istream& operator >>(istream& in, m& a) {
    ll val; in >> val;
    a = m(val);
    return in;
  }
  friend ostream& operator <<(ostream& out, m a) {
    return out << a.v;
  }
  friend m operator +(m a, m b) { return a += b; }
  friend m operator -(m a, m b) { return a -= b; }
  friend m operator *(m a, m b) { return a *= b; }
  friend m operator /(m a, m b) { return a /= b; }
  friend m operator ^(m a, ll e) { return a ^= e; }
};

typedef mod_int<(int)1e9+7> mint;

// NTT
//
// Precisa do mint (primitivas de aritmetica modular)
//
// O(n log (n))

const int MOD = 998244353;
typedef mod_int<MOD> mint;

void ntt(vector<mint>& a, bool rev) {
  int n = a.size(); auto b = a;
  assert(!(n&(n-1)));
  mint g = 1;
  while ((g^(MOD / 2)) == 1) g += 1;
  if (rev) g = 1 / g;

  for (int step = n / 2; step; step /= 2) {
    mint w = g^(MOD / (n / step)), wn = 1;
    for (int i = 0; i < n/2; i += step) {
      for (int j = 0; j < step; j++) {
        auto u = a[2 * i + j], v = wn * a[2 * i + j + step];
        b[i+j] = u + v; b[i + n/2 + j] = u - v;
      }
      wn = wn * w;
    }
    swap(a, b);
  }
  if (rev) {
    auto n1 = mint(1) / n;
    for (auto& x : a) x *= n1;
  }
}

vector<mint> convolution(const vector<mint>& a, const vector<mint>& b) {
  vector<mint> l(a.begin(), a.end()), r(b.begin(), b.end());
  int N = l.size()+r.size()-1, n = 1;
  while (n <= N) n *= 2;
  l.resize(n);
  r.resize(n);
  ntt(l, false);
  ntt(r, false);
  for (int i = 0; i < n; i++) l[i] *= r[i];
  ntt(l, true);
  l.resize(N);
  return l;
}
```

6.21 Phi

```
// Description:
// Euler's totient function.
// phi(n) is the amount of numbers in the range (1, n) that are coprime with n

// Complexity:
// phi(n) - sqrt(n)
// phi of all numbers from 1 to n - O (n log log n)

// Properties:
// phi(p ^ k) = p ^ k - p ^ (k - 1)
// phi(p) = p - 1
// phi(ab) = phi(a) * phi(b) * d / phi(d) being d = gcd(a, b)

int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}

void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

6.22 Division Trick

```
for(int l = 1, r; l <= n; l = r + 1) {
    r = n / (n / l);
    // n / i has the same value for l <= i <= r
    // from l to r, n / i has the same value which is n / l
}

/* 100/1 = 100
100/2 = 50
100/3 = 33
100/4 = 25
100/5 = 20
100/6 = 16
100/7 = 14
100/8 = 12
100/9 = 11
100/10 = 10
100/11 = 9
100/12 = 8
100/13 = 7
100/14 = 7
100/15 = 6
100/16 = 6
100/17 = 5
100/18 = 5
100/19 = 5
100/20 = 5
100/21 = 4
100/22 = 4
100/23 = 4
100/24 = 4
100/25 = 4
100/26 = 3
100/27 = 3
100/28 = 3
100/29 = 3
100/30 = 3
100/31 = 3
100/32 = 3
100/33 = 3
100/34 = 2
100/35 = 2
100/36 = 2
100/37 = 2
100/38 = 2
100/39 = 2
100/40 = 2
100/41 = 2
100/42 = 2
100/43 = 2
100/44 = 2
100/45 = 2
100/46 = 2
100/47 = 2
100/48 = 2
100/49 = 2
100/50 = 2
100/51 = 1
100/52 = 1
100/53 = 1
100/54 = 1
100/55 = 1
100/56 = 1
100/57 = 1
100/58 = 1
100/59 = 1
100/60 = 1
100/61 = 1
100/62 = 1
100/63 = 1
100/64 = 1
100/65 = 1
100/66 = 1
100/67 = 1
100/68 = 1
100/69 = 1
100/70 = 1
100/71 = 1
100/72 = 1
100/73 = 1
100/74 = 1
100/75 = 1
100/76 = 1
100/77 = 1
100/78 = 1
100/79 = 1
100/80 = 1
100/81 = 1
100/82 = 1
100/83 = 1
100/84 = 1
100/85 = 1
100/86 = 1
100/87 = 1
100/88 = 1
100/89 = 1
100/90 = 1
100/91 = 1
100/92 = 1
```

```
100/93 = 1
100/94 = 1
100/95 = 1
100/96 = 1
100/97 = 1
100/98 = 1
100/99 = 1 */
```

6.23 Ceil

```
long long division_ceil(long long a, long long b) {
    return 1 + ((a - 1) / b); // if a != 0
}
```

7 DP

7.1 Knapsack

```
int val[MAXN], peso[MAXN], dp[MAXN][MAXS];

int knapsack(int n, int m){ // n Objetos | Peso max
    for(int i=0;i<=n;i++){
       for(int j=0;j<=m;j++){
            if(i==0 or j==0)
                dp[i][j] = 0;
            else if(peso[i-1]<=j)
                dp[i][j] = max(val[i-1]+dp[i-1][j-peso[i-1]], dp[i-1][j]);
            else
                dp[i][j] = dp[i-1][j];
       }
    }
    return dp[n][m];
}
```

7.2 Knapsack With Index

```
void knapsack(int W, int wt[], int val[], int n) {
    int i, w;
    int K[n + 1][W + 1];

    for (i = 0; i <= n; i++) {
        for (w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1] +
                    K[i - 1][w - wt[i - 1]], K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }

    int res = K[n][W];
    cout<< res << endl;

    w = W;
    for (i = n; i > 0 && res > 0; i--) {
        if (res == K[i - 1][w])
            continue;
        else {
            cout<<" "<<wt[i - 1] ;
            res = res - val[i - 1];
            w = w - wt[i - 1];
        }
    }
}

int main()
{
    int val[] = { 60, 100, 120 };
    int wt[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);

    knapsack(W, wt, val, n);

    return 0;
}
```

7.3 Substr Palindrome

```
// êvoc deve informar se a substring de S formada pelos elementos entre os indices i e j
// é um palindromo ou ãno.

char s[MAX];
int calculado[MAX][MAX]; // inciado com false, ou 0
int tabela[MAX][MAX];

int is_palin(int i, int j){
  if(calculado[i][j]){
    return tabela[i][j];
  }
  if(i == j) return true;
  if(i + 1 == j) return s[i] == s[j];

  int ans = false;
  if(s[i] == s[j]){
    if(is_palin(i+1, j-1)){
      ans = true;
    }
  }
  calculado[i][j] = true;
  tabela[i][j] = ans;
  return ans;
}
```

7.4 Edit Distance

```
// Description:
// Minimum number of operations required to transform a string into another
// Operations allowed: add character, remove character, replace character

// Parameters:
// str1 - string to be transformed into str2
// str2 - string that str1 will be transformed into
// m - size of str1
// n - size of str2

// Problem:
// https://cses.fi/problemset/task/1639

// Complexity:
// O(m x n)

// How to use:
// memset(dp, -1, sizeof(dp));
// string a, b;
// edit_distance(a, b, (int)a.size(), (int)b.size());

// Notes:
// Size of dp matriz is m x n

int dp[MAX][MAX];

int edit_distance(string &str1, string &str2, int m, int n) {
    if (m == 0) return n;
    if (n == 0) return m;
```

```cpp

    if (dp[m][n] != -1) return dp[m][n];

    if (str1[m - 1] == str2[n - 1]) return dp[m][n] = edit_distance(str1, str2, m - 1, n - 1);
    return dp[m][n] = 1 + min({edit_distance(str1, str2, m, n - 1), edit_distance(str1, str2, m - 1, n), edit_distance(str1, str2, m - 1, n - 1)});
}
```

7.5 Coins

```cpp
int tb[1005];
int n;
vector<int> moedas;

int dp(int i){
  if(i >= n)
    return 0;
  if(tb[i] != -1)
    return tb[i];

  tb[i] = max(dp(i+1), dp(i+2) + moedas[i]);
  return tb[i];
}

int main(){
  memset(tb,-1,sizeof(tb));
}
```

7.6 Digits

```cpp
// achar a quantidade de numeros menores que R que possuem no maximo 3 digitos nao nulos
// a ideia eh utilizar da ordem lexicografica para checar isso pois se temos por exemplo
// o numero 8500, a gente sabe que se pegarmos o numero 7... qualquer digito depois do 7
// sera necessariamente menor q 8500

string r;
int tab[20][2][5];

// i - digito de R
// menor - ja pegou um numero menor que um digito de R
// qt - quantidade de digitos nao nulos
int dp(int i, bool menor, int qt){
    if(qt > 3) return 0;
    if(i >= r.size()) return 1;
    if(tab[i][menor][qt] != -1) return tab[i][menor][qt];

    int dr = r[i]-'0';
    int res = 0;

    for(int d = 0; d <= 9; d++) {
        int dnn = qt + (d > 0);
        if(menor == true) {
            res += dp(i+1, true, dnn);
        }
        else if(d < dr) {
            res += dp(i+1, true, dnn);
        }
        else if(d == dr) {
            res += dp(i+1, false, dnn);
        }
    }

    return tab[i][menor][qt] = res;
}
```

7.7 Minimum Coin Change

```cpp
int n;
vector<int> valores;

int tabela[1005];

int dp(int k){
  if(k == 0){
    return 0;
  }
  if(tabela[k] != -1)
    return tabela[k];
  int melhor = 1e9;
  for(int i = 0; i < n; i++){
    if(valores[i] <= k)
      melhor = min(melhor,1 + dp(k - valores[i]));
  }
  return tabela[k] = melhor;
}
```

7.8 Kadane

```cpp
// Description:
// Finds the maximum (or minimum) sum of some subarray of a given array

// Problem:
// https://leetcode.com/problems/maximum-subarray/description/

// Complexity:
// O(n)

// Notes
// To solve the minimum subarray problem, start the variable ans with INF and change the max operations to min operations
// To not count the empty subarray as a subrray, start the variable ans with -INF
// To get the biggest possible subarray with that sum, change if (curr > ans) to if (curr >= ans)
// If the empty subarray is the answer, start and end will be equal to -1

int ans = 0, curr = 0;
int startidx = 0, start = -1, end = -1;

for (int i = 0; i < n; i++) {
  // MAXIMUM SUBARRAY PROBLEM
  curr = max(curr + v[i], v[i]);
  ans = max(ans, curr);

  /*
  RECOVER INDEXES MAXIMUM SUBARRAY PROBLEM
  if (curr + v[i] < v[i]) {
    startidx = i;
    curr = v[i];
  }
  else curr += v[i];

  if (curr > ans) {
    ans = curr;
    start = startidx;
    end = i;
  }
  */

  // MINIMUM SUBARRAY PROBLEM
  // curr = min(curr + v[i], v[i]);
  // ans = min(ans, curr);

  /*
  // MINIMUM SUBARRAY PROBLEM
  if (curr + v[i] > v[i]) {
    startidx = i;
    curr = v[i];
  }
  else curr += v[i];

  if (curr < ans) {
    ans = curr;
    start = startidx;
    end = i;
  }
  */
}

// cout << ans << ' ' << start << ' ' << end << '\n';
```

7.9 Divide And Conquer

```
// Description:
// Divide the array in k intervals

// Problem:
// https://cses.fi/problemset/task/2086/

// Recurrence
// dp(i, j) = min(dp(i - 1, k - 1), C(k, j))

int dp[MAX][MAX];

vector<int> v, psum;
int k, n;

int C(int start, int end) {
  if (start == 0) return psum[end] * psum[end];

  return (psum[end] - psum[start - 1]) * (psum[end] - psum[start - 1]);
}

void solve(int sz, int l, int r, int optl, int optr) {
  if (l > r) return;

  int mid = l + (r - l) / 2;

  pii best = {INF, l};

  for (int j = optl; j < min(mid, optr + 1); j++) {
    best = min(best, make_pair(C(j + 1, mid) + dp[sz - 1][j], j));
  }

  dp[sz][mid] = best.first;

  int opt = best.second;
  solve(sz, l, mid - 1, optl, opt);
  solve(sz, mid + 1, r, opt, optr);
}

int32_t main() {
  ios::sync_with_stdio(false);
  cin.tie(NULL);

  cin >> n >> k;
  v.resize(n);
  psum.assign(n, 0);

  for (int idx = 0; idx < n; idx++) {
    for (int sz = 0; sz <= n; sz++) {
      dp[sz][idx] = INF;
    }
  }

  for (int i = 0; i < n; i++) {
    cin >> v[i];
    psum[i] = v[i];
    if (i != 0) psum[i] += psum[i - 1];
  }

  for (int i = 0; i < n; i++) {
    dp[1][i] = C(0, i);
  }

  for (int sz = 2; sz <= n; sz++) {
    solve(sz, 1, n, 0, n - 1);

    /* for (int idx = 0; idx < n; idx++) {
      for (int i = 0; i < idx; i++) {
        dp[sz][idx] = min(dp[sz][idx], dp[sz - 1][i] + C(i + 1, idx));
      }
    } */

  }

  cout << dp[k][n - 1] << '\n';

  return 0;
}
```

7.10 Knuth

```
// Description
// Optimization of range dp

// Problem
// https://www.spoj.com/problems/BRKSTRNG/

// Recurrence
// dp(i, j) = min(dp(i, k) + d[(k + 1, j) + C(i, j)]) for (int k = i; k < j; k++)

int dp[MAX][MAX], opt[MAX][MAX];
vector<int> v;
int n, m;

int solve(int l, int r) {
  if (abs(l - r) <= 1) return 0;

  int& memo = dp[l][r];
  if (memo != -1) return memo;

  memo = INF;
  for (int i = l + 1; i < r; i++) {
    memo = min(memo, solve(l, i) + solve(i, r) + v[r] - v[l]);
  }

  return memo;
}

int32_t main() {
  ios::sync_with_stdio(false);
  cin.tie(NULL);

  while (cin >> n >> m) {
    v.resize(m + 2, 0);
    v[m + 1] = n;
    for (int i = 1; i <= m; i++) {
      cin >> v[i];
    }

    for (int i = 0; i <= m + 1; i++) {
      for (int j = 0; j <= m + 1; j++) {
        opt[i][i] = i;
        dp[i][i] = 0;
        dp[i][j] = INF;
        if (abs(i - j) <= 1) dp[i][j] = 0;
        if (abs(i - j) <= 1) opt[i][j] = i;
      }
    }

    // memset(dp, -1, sizeof(dp));
    // cout << solve(0, m + 1) << '\n';

    for (int l = m; l >= 0; l--) {
      for (int r = l + 1; r <= m + 1; r++) {
        if (abs(l - r) <= 1) continue;
        /* for (int i = l + 1; i < r; i++) {
          dp[l][r] = min(dp[l][r], dp[l][i] + dp[i][r] + v[r] - v[l]);
        } */

        ll ans = INF;
          for (int k=opt[l][r-1]; k<=min(r-1, opt[l+1][r]); k++) {
            ll best = dp[l][k] + dp[k][r];
            if (ans > best) {
                ans = best;
                opt[l][r] = k;
            }
          }
          dp[l][r] = ans + v[r] - v[l];
      }
    }

    cout << dp[0][m + 1] << ’\n’;
  }

  return 0;
}
```

7.11 Cht

```
// Description:
// Write in terms of a line y = ax + b

// Problem:
// https://atcoder.jp/contests/dp/tasks/dp_z/

int n, c;
vector<int> h;
int dp[MAX];

const ll is_query = -INF;
struct Line{
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const{
        if(rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if(!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct Cht : public multiset<Line>{ // maintain max m*x+b
    bool bad(iterator y){
        auto z = next(y);
        if(y == begin()){
            if(z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if(z == end()) return y->m == x->m && y->b <= x->b;
        return (ld)(x->b - y->b)*(z->m - y->m) >= (ld)(y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b){ // min -> insert (-m,-b) -> -eval()
        m *= -1; b *= -1;
        auto y = insert({ m, b });
        y->succ = [=]{ return next(y) == end() ? 0 : &*next(y); };
        if(bad(y)){ erase(y); return; }
        while(next(y) != end() && bad(next(y))) erase(next(y));
        while(y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x){
        auto l = *lower_bound((Line) { x, is_query });
        return -(l.m * x + l.b);
    }
};

int cost(int a, int b) {
  return (h[b] - h[a]) * (h[b] - h[a]) + c;
}

int32_t main() {
  ios::sync_with_stdio(false);
  cin.tie(NULL);

  cin >> n >> c;
  h.resize(n);

  for (int i = 0; i < n; i++) {
    cin >> h[i];
    dp[i] = INF;
  }

  dp[0] = 0;
  Cht cht = Cht();
  cht.insert_line(-2 * h[0], h[0] * h[0] + dp[0]);

  for (int i = 1; i < n; i++) {
    dp[i] = h[i] * h[i] + c + cht.eval(h[i]);

    cht.insert_line(-2 * h[i], h[i] * h[i] + dp[i]);

    // for (int j = 0; j < idx; j++) {
    //   dp[idx] = min(dp[idx], dp[j] + cost(j, idx));
    // }
  }

  cout << dp[n - 1] << ’\n’;

  return 0;
}
```

8 Graphs

8.1 Dijkstra

```
const int MAX = 2e5+7;
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj(MAX);

void dijkstra(int s, vector<int> & d, vector<int> & p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);

    d[s] = 0;
    set<pair<int, int>> q;
    q.insert({0, s});
    while (!q.empty()) {
        int v = q.begin()->second;
        q.erase(q.begin());

        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                q.erase({d[to], to});
                d[to] = d[v] + len;
                p[to] = v;
                q.insert({d[to], to});
            }
        }
    }
}

vector<int> restore_path(int s, int t) {
    vector<int> path;

    for (int v = t; v != s; v = p[v])
        path.push_back(v);
    path.push_back(s);

    reverse(path.begin(), path.end());
    return path;
}

int adj[MAX][MAX];
int dist[MAX];
int minDistance(int dist[], bool sptSet[], int V) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

void dijkstra(int src, int V) {

    bool sptSet[V];
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet, V);

        sptSet[u] = true;


        for (int v = 0; v < V; v++)
            if (!sptSet[v] && adj[u][v]
                && dist[u] != INT_MAX
                && dist[u] + adj[u][v] < dist[v])
                dist[v] = dist[u] + adj[u][v];
    }
}
```

8.2 Bipartite

```
const int NONE = 0, BLUE = 1, RED = 2;
vector<vector<int>> graph(100005);
vector<bool> visited(100005);
int color[100005];

bool bfs(int s = 1){

    queue<int> q;
    q.push(s);
    color[s] = BLUE;

    while (not q.empty()){
        auto u = q.front(); q.pop();

        for (auto v : graph[u]){
            if (color[v] == NONE){
                color[v] = 3 - color[u];
                q.push(v);
            }
            else if (color[v] == color[u]){
                return false;
            }
        }
    }

    return true;
}

bool is_bipartite(int n){

    for (int i = 1; i<=n; i++)
        if (color[i] == NONE and not bfs(i))
            return false;

    return true;
}
```

8.3 Eulerian Undirected

```
// Description:
// Hierholzer's Algorithm
// An Eulerian path is a path that passes through every edge exactly once.
// An Eulerian circuit is an Eulerian path that starts and ends on the same node.

// An Eulerian path exists in an undirected graph if the degree of every node is even (not counting self-edges)
// except for possibly exactly two nodes that have and odd degree (start and end nodes).
// An Eulerian circuit exists in an undirected graph if the degree of every node is even.

// The graph has to be conected (except for isolated nodes which are allowed because there
// are no edges connected to them).

// Problem:
// https://cses.fi/problemset/task/1691

// Complexity:
// O(E * log(E)) where E is the number of edges

// How to use
// Check whether the path exists before trying to find it
// Find the root - any node that has at least 1 outgoing edge
// (if the problem requires that you start from a node v, the root will be the node v)
// Count the degree;
//
// for (int i = 0; i < m; i++) {
//  int a, b; cin >> a >> b;
//  adj[a].pb(b); adj[b].pb(a);
//  root = a;
//  degree[a]++; degree[b]++;
// }

// Notes
// If you want to find a path start and ending nodes v and u
// if ((is_eulerian(n, root, start, end) != 1) || (start != v) || (end != u)) cout << "IMPOSSIBLE\n"

// It can be speed up to work on O(E) on average by using unordered_set instead of set

// It works when there are self loops, but not when there are multiple edges
// It the graph has multiple edges, add more notes to simulate the edges
// e.g
// 1 2
// 1 2
// 1 2
// becomes
// 3 4
// 4 1
// 1 2

vector<bool> visited;
vector<int> degree;
vector<vector<int>> adj;

void dfs(int v) {
  visited[v] = true;
  for (auto u : adj[v]) {
    if (!visited[u]) dfs(u);
  }
}

int is_eulerian(int n, int root, int& start, int& end) {
  start = -1, end = -1;
  if (n == 1) return 2; // only one node
  visited.assign(n + 1, false);
  dfs(root);

  for (int i = 1; i <= n; i++) {
    if (!visited[i] && degree[i] > 0) return 0;
  }

  for (int i = 1; i <= n; i++) {
    if (start == -1 && degree[i] % 2 == 1) start = i;
    else if (end == -1 && degree[i] % 2 == 1) end = i;
    else if (degree[i] % 2 == 1) return 0;
  }

  if (start == -1 && end == -1) {start = root; end = root; return 2;} // has eulerian circuit and path
  if (start != -1 && end != -1) return 1; // has eulerian path
  return 0; // no eulerian path nor circuit
}

vector<int> path;
vector<set<int>> mark;

void dfs_path(int v) {
  visited[v] = true;

  while (degree[v] != 0) {
    degree[v]--;
    int u = adj[v][degree[v]];
    if (mark[v].find(u) != mark[v].end()) continue;
    mark[v].insert(u);
    mark[u].insert(v);
    int next_edge = adj[v][degree[v]];
    dfs_path(next_edge);
  }
  path.pb(v);
}

void find_path(int n, int start) {
  path.clear();
  mark.resize(n + 1);
  visited.assign(n + 1, false);
  dfs_path(start);
}
```

8.4 Kruskall

```
struct DSU {
    int n;
    vector<int> link, sizes;

    DSU(int n) {
        this->n = n;
        link.assign(n+1, 0);
        sizes.assign(n+1, 1);

        for (int i = 0; i <= n; i++)
            link[i] = i;
    }

    int find(int x) {
        while (x != link[x])
            x = link[x];

        return x;
    }

    bool same(int a, int b) {
        return find(a) == find(b);
    }

    void unite(int a, int b) {
        a = find(a);
        b = find(b);

        if (a == b) return;

        if (sizes[a] < sizes[b])
            swap(a, b);

        sizes[a] += sizes[b];
        link[b] = a;
    }
};

struct Edge {
    int u, v;
    long long weight;

    Edge() {}

    Edge(int u, int v, long long weight) : u(u), v(v), weight(weight) {}

    bool operator<(const Edge& other) const {
        return weight < other.weight;
    }

    bool operator>(const Edge& other) const {
        return weight > other.weight;
    }
};

vector<Edge> kruskal(vector<Edge> edges, int n) {
    vector<Edge> result; // arestas da MST
    long long cost = 0;
    sort(edges.begin(), edges.end());

    DSU dsu(n);

    for (auto e : edges) {
        if (!dsu.same(e.u, e.v)) {
            cost += e.weight;
            result.push_back(e);
            dsu.unite(e.u, e.v);
        }
    }

    return result;
}
```

8.5 Negative Cycle

```
// Description
// Detects any cycle in which the sum of edge weights is negative.
// Alternatively, we can detect whether there is a negative cycle
// starting from a specific vertex.

// Problem:
// https://cses.fi/problemset/task/1197

// Complexity:
// O(n * m)

// Notes
// In order to consider only the negative cycles located on the path from a to b,
// Reverse the graph, run a dfs from node b and mark the visited nodes
// Consider only the edges that connect to visited nodes when running bellman-ford
// on the normal graph

struct Edge {
  int a, b, cost;
  Edge(int a, int b, int cost) : a(a), b(b), cost(cost) {}
};

int n, m;
vector<Edge> edges;
const int INF = 1e9+10;

void negative_cycle() {
  // uncomment to find negative cycle starting from a vertex v
  // vector<int> d(n + 1, INF);
  // d[v] = 0;
  vector<int> d(n + 1, 0);
  vector<int> p(n + 1, -1);
  int x;
  // uncomment to find all negative cycles
  // // set<int> s;
  for (int i = 1; i <= n; ++i) {
    x = -1;
    for (Edge e : edges) {
      // if (d[e.a] >= INF) continue;
      if (d[e.b] > d[e.a] + e.cost) {
        // d[e.b] = max(-INF, d[e.a] + e.cost);
        d[e.b] = d[e.a] + e.cost;
        p[e.b] = e.a;
        x = e.b;
        // // s.insert(e.b);
      }
    }
  }

  if (x == -1)
    cout << "NO\n";
  else {
    // // int y = all nodes in set s
    int y = x;
    for (int i = 1; i <= n; ++i) {
      y = p[y];
    }

    vector<int> path;
    for (int cur = y;; cur = p[cur]) {
      path.push_back(cur);
      if (cur == y && path.size() > 1) break;
    }
    reverse(path.begin(), path.end());

    cout << "YES\n";
    for (int u : path)
        cout << u << ' ';
    cout << '\n';
  }
}
```

8.6 Floyd Warshall

```
#include <bits/stdc++.h>

using namespace std;
using ll = long long;

const int MAX  = 507;
const long long INF = 0x3f3f3f3f3f3f3f3fLL;

ll dist[MAX][MAX];
int n;

void floyd_warshall() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) dist[i][j] = 0;
            else if (!dist[i][j]) dist[i][j] = INF;
        }
    }

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                // trata o caso no qual o grafo tem arestas com peso negativo
                if (dist[i][k] < INF && dist[k][j] < INF){
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                }
            }
        }
    }
}
```

8.7 Centroid Find

```
// Description:
// Indexed at zero
// Find a centroid, that is a node such that when it is appointed the root of the tree,
// each subtree has at most floor(n/2) nodes.

// Problem:
// https://cses.fi/problemset/task/2079/

// Complexity:
```

```cpp
// O(n)

// How to use:
// get_subtree_size(0);
// cout << get_centroid(0) + 1 << endl;

int n;
vector<int> adj[MAX];
int subtree_size[MAX];

int get_subtree_size(int node, int par = -1) {
  int &res = subtree_size[node];
  res = 1;
  for (int i : adj[node]) {
    if (i == par) continue;
    res += get_subtree_size(i, node);
  }
  return res;
}

int get_centroid(int node, int par = -1) {
  for (int i : adj[node]) {
    if (i == par) continue;

    if (subtree_size[i] * 2 > n) { return get_centroid(i, node); }
  }
  return node;
}

int main() {
  cin >> 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);
  }

  get_subtree_size(0);
  cout << get_centroid(0) + 1 << endl;
}
```

8.8 Eulerian Directed

```cpp
// Description:
// Hierholzer's Algorithm
// An Eulerian path is a path that passes through every edge exactly once.
// An Eulerian circuit is an Eulerian path that starts and ends on the same node.

// An Eulerian path exists in an directed graph if the indegree and outdegree is equal
// for every node (not counting self-edges)
// except for possibly exactly one node that have outdegree - indegree = 1
// and one node that has indegree - outdegreee = 1 (start and end nodes).
// An Eulerian circuit exists in an directed graph if the indegree and outdegree is equal for every node.

// The graph has to be conected (except for isolated nodes which are allowed because there
// are no edges connected to them).

// Problem:
// https://cses.fi/problemset/task/1693

// Complexity:
// O(E) where E is the number of edges

// How to use
// Check whether the path exists before trying to find it
// Find the root - any node that has at least 1 outgoing edge
// (if the problem requires that you start from a node v, the root will be the node v)
// Count the degree;
//
// for (int i = 0; i < m; i++) {
//  int a, b; cin >> a >> b;
//  adj[a].pb(b);
//  root = a;
//  outdegree[a]++; indegree[b]++;
// }

// Notes
// It works when there are self loops, but not when there are multiple edges

vector<bool> visited;
vector<int> outdegree, indegree;
vector<vector<int>> adj, undir;

void dfs(int v) {
  visited[v] = true;
  for (auto u : undir[v]) {
    if (!visited[u]) dfs(u);
  }
}

int is_eulerian(int n, int root, int &start, int& end) {
  start = -1, end = -1;
  if (n == 1) return 2; // only one node
  visited.assign(n + 1, false);
  dfs(root);

  for (int i = 1; i <= n; i++) {
    if (!visited[i] && (i == n || i == 1 || outdegree[i] + indegree[i] > 0)) return 0;
  }

  // start => node with indegree - outdegree = 1
  // end => node with outdegree - indegree = 1
  for (int i = 1; i <= n; i++) {
    if (start == -1 && indegree[i] - outdegree[i] == 1) start = i;
    else if (end == -1 && outdegree[i] - indegree[i] == 1) end = i;
    else if (indegree[i] != outdegree[i]) return 0;
  }

  if (start == -1 && end == -1) {start = root; end = root; return 2;} // has eulerian circuit and path
  if (start != -1 && end != -1) {swap(start, end); return 1;} // has eulerian path
  return 0; // no eulerian path nor circuit
}

vector<int> path;

void dfs_path(int v) {
  visited[v] = true;

  while (outdegree[v] != 0) {
    int u = adj[v][--outdegree[v]];
    int next_edge = adj[v][outdegree[v]];
    dfs_path(next_edge);
  }
  path.pb(v);
}

```

```cpp
void find_path(int n, int start) {
  path.clear();
  visited.assign(n + 1, false);
  dfs_path(start);
  reverse(path.begin(), path.end());
}
```

8.9 Lca

```cpp
// Description:
// Find the lowest common ancestor between two nodes in a tree

// Problem:
// https://cses.fi/problemset/task/1135

// Complexity:
// O(log n)

// How to use:
// preprocess();
// lca(a, b);

// Notes
// To calculate the distance between two nodes use the following formula
// level_peso[a] + level_peso[b] - 2*level_peso[lca(a, b)]

// If you just need to know if a node is the ancestor of another node or not

vector<vector<int>> adj;
vector<int> tin, tout;

void dfs(int v, int p, int& idx) {
  tin[v] = idx++;

  for (auto u : adj[v]) {
    if (u == p) continue;
    dfs(u, v, idx);
  }

  tout[v] = idx++;
}

bool is_ancestor(int a, int b) {
  return (tin[a] >= tin[b] && tout[b] <= tout[a])
  || (tin[b] >= tin[a] && tout[a] <= tout[b]);
}

// LCA

const int MAX = 2e5+10;
const int BITS = 30;

vector<pii> adj[MAX];
vector<bool> visited(MAX);

int up[MAX][BITS + 1];
int level[MAX];
int level_peso[MAX];

void find_level() {
  queue<pii> q;

  q.push(mp(1, 0));
  visited[1] = true;

  while (!q.empty()) {
    auto [v, depth] = q.front();
    q.pop();
    level[v] = depth;

    for (auto [u,d] : adj[v]) {
      if (!visited[u]) {
        visited[u] = true;
        up[u][0] = v;
        q.push(mp(u, depth + 1));
      }
    }
  }
}

void find_level_peso() {
  queue<pii> q;

  q.push(mp(1, 0));
  visited[1] = true;

  while (!q.empty()) {
    auto [v, depth] = q.front();
    q.pop();
    level_peso[v] = depth;

    for (auto [u,d] : adj[v]) {
      if (!visited[u]) {
        visited[u] = true;
        up[u][0] = v;
        q.push(mp(u, depth + d));
      }
    }
  }
}

int lca(int a, int b) {
   // get the nodes to the same level
    int mn = min(level[a], level[b]);

    for (int j = 0; j <= BITS; j++) {
      if (a != -1 && ((level[a] - mn) & (1 << j))) a = up[a][j];
      if (b != -1 && ((level[b] - mn) & (1 << j))) b = up[b][j];
    }

    // special case
    if (a == b) return a;

    // binary search
    for (int j = BITS; j >= 0; j--) {
      if (up[a][j] != up[b][j]) {
        a = up[a][j];
        b = up[b][j];
      }
    }
    return up[a][0];
}

void preprocess() {
  visited = vector<bool>(MAX, false);
  find_level();
  visited = vector<bool>(MAX, false);
  find_level_peso();

  for (int j = 1; j <= BITS; j++) {
    for (int i = 1; i <= n; i++) {
      if (up[i][j - 1] != -1) up[i][j] = up[up[i][j - 1]][j - 1];
    }
  }
}
```

8.10 Ford Fulkerson Edmonds Karp

```cpp
// Description:
```

```cpp
// Obtains the maximum possible flow rate given a network. A network is a graph with a single source vertex and a single sink vertex in which each edge has a capacity

// Complexity:
// O(V * E^2) where V is the number of vertex and E is the number of edges

int n;
vector<vector<int>> capacity;
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

8.11 Kuhn

```cpp
// Description
// Matching algorithm for unweighted bipartite graph ::

// Problem:
// https://codeforces.com/gym/104252/problem/H

// Complexity:
// O(V * E) in which V is the number of vertexes and E is the number of edges

// Notes:
// Indexed at zero

int n, k;
// adjacency list
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;

bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}

int main() {
    // ... reading the graph g ...

    mt.assign(k, -1);
    vector<bool> used1(n, false);
    for (int v = 0; v < n; ++v) {
        for (int to : g[v]) {
            if (mt[to] == -1) {
                mt[to] = v;
                used1[v] = true;
                break;
            }
        }
    }
    for (int v = 0; v < n; ++v) {
        if (used1[v])
            continue;
        used.assign(n, false);
        try_kuhn(v);
    }

    for (int i = 0; i < k; ++i)
        if (mt[i] != -1)
            printf("%d %d\n", mt[i] + 1, i + 1);
}
```

8.12 Cycle Path Recovery

```cpp
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

8.13 Hld Vertex

```
// Description:
// Make queries and updates between two vertexes on a tree
// Query path - query path (a, b) inclusive
// Update path - update path (a, b) inclusive
// Query subtree - query subtree of a
// Update subtree - update subtree of a
// Update - update vertex or edge
// Lca - get lowest common ancestor of a and b
// Search - perform a binary search to find the last node with a certain property
// on the path from a to the root

// Problem:
// https://codeforces.com/gym/101908/problem/L

// Complexity:
// O(log ^2 n) for both query and update

// How to use:
// HLD hld = HLD(n + 1, adj)

// Notes
// Change the root of the tree on the constructor if it's different from 1
// Use together with Segtree

typedef long long ftype;

struct HLD {
  vector<int> parent;
  vector<int> pos;
  vector<int> head;
  vector<int> subtree_size;
  vector<int> level;
  vector<int> heavy_child;
  vector<ftype> subtree_weight;
  vector<ftype> path_weight;
  vector<vector<int>> adj;
  vector<int> at;
  Segtree seg = Segtree(0);
  int cpos;
  int n;
  int root;
  vector<vector<int>> up;

  HLD() {}

  HLD(int n, vector<vector<int>>& adj, int root = 1) : adj(adj), n(n), root(root) {
    seg = Segtree(n);
    cpos = 0;
    at.resize(n);
    parent.resize(n);
    pos.resize(n);
    head.resize(n);
    subtree_size.assign(n, 1);
    level.assign(n, 0);
    heavy_child.assign(n, -1);
    parent[root] = -1;
    dfs(root, -1);
    decompose(root, -1);
  }

  void dfs(int v, int p) {
    parent[v] = p;
    if (p != -1) level[v] = level[p] + 1;
    for (auto u : adj[v]) {
      if (u != p) {
        dfs(u, v);
        subtree_size[v] += subtree_size[u];
        if (heavy_child[v] == -1 || subtree_size[u] > subtree_size[heavy_child[v]]) heavy_child[v] = u;
      }
    }
  }

  void decompose(int v, int chead) {
    // start a new path
    if (chead == -1) chead = v;

    // consecutive ids in the hld path
    at[cpos] = v;
    pos[v] = cpos++;
    head[v] = chead;

    // if not a leaf
    if (heavy_child[v] != -1) decompose(heavy_child[v], chead);

    // light child
    for (auto u : adj[v]){
      // start new path
      if (u != parent[v] && u != heavy_child[v]) decompose(u, -1);
    }
  }

  ftype query_path(int a, int b) {
    if(pos[a] < pos[b]) swap(a, b);

    if(head[a] == head[b]) return seg.query(pos[b], pos[a]);
    return seg.f(seg.query(pos[head[a]], pos[a]), query_path(parent[head[a]], b));
  }

  // iterative
  /*ftype query_path(int a, int b) {
    ftype ans = 0;

    while (head[a] != head[b]) {
      if (level[head[a]] > level[head[b]]) swap(a, b);
      ans = seg.merge(ans, seg.query(pos[head[b]], pos[b]));
      b = parent[head[b]];
    }

    if (level[a] > level[b]) swap(a, b);
    ans = seg.merge(ans, seg.query(pos[a], pos[b]));
    return ans;
  }*/

  ftype query_subtree(int a) {
    return seg.query(pos[a], pos[a] + subtree_size[a] - 1);
  }

  void update_path(int a, int b, int x) {
    if(pos[a] < pos[b]) swap(a, b);

    if(head[a] == head[b]) return (void)seg.update(pos[b], pos[a], x);
    seg.update(pos[head[a]], pos[a], x); update_path(parent[head[a]], b, x);
  }

  void update_subtree(int a, int val) {
    seg.update(pos[a], pos[a] + subtree_size[a] - 1, val);
  }

  void update(int a, int val) {
    seg.update(pos[a], pos[a], val);
  }

  //edge
  void update(int a, int b, int val) {
    if (level[a] > level[b]) swap(a, b);
    update(b, val);
  }

  int lca(int a, int b) {
    if(pos[a] < pos[b]) swap(a, b);
    return head[a] == head[b] ? b : lca(parent[head[a]], b);
  }

  void search(int a) {
    a = parent[a];
    if (a == -1) return;
    if (seg.query(pos[head[a]], pos[head[a]]+subtree_size[head[a]]-1) + pos[a]-pos[head[a]]+1 == subtree_size[head[a]]) {
      seg.update(pos[head[a]], pos[a], 1);
      return search(parent[head[a]]);
    }
    int l = pos[head[a]], r = pos[a]+1;
    while (l < r) {
      int m = (l+r)/2;
      if (seg.query(m, m+subtree_size[at[m]]-1) + pos[a]-m+1 == subtree_size[at[m]]) {
        r = m;
      }
      else l = m+1;
    }
    seg.update(l, pos[a], 1);
  }

  /* k-th ancestor of x
  int x, k; cin >> x >> k;

  for (int b = 0; b <= BITS; b++) {
    if (x != -1 && (k & (1 << b))) {
      x = up[x][b];
    }
  }

    cout << x << '\n';
    */
    void preprocess() {
      up.assign(n + 1, vector<int>(31, -1));

      for (int i = 1; i < n; i++) {
        up[i][0] = parent[i];
      }

      for (int i = 1; i < n; i++) {
        for (int j = 1; j <= 30; j++) {
          if (up[i][j - 1] != -1) up[i][j] = up[up[i][j - 1]][j - 1];
        }
      }
    }

    int getKth(int p , int q , int k){
      int a = lca(p,q) , d ;

      if( a == p ){
          d = level[q] - level[p] + 1 ;
          swap(p,q);
          k = d - k + 1 ;
      }
      else if( a == q ) ;
      else {
          if( k > level[p] - level[a] + 1 ) {
              d = level[p] + level[q] - 2 * level[a] + 1 ;
              k = d - k + 1 ;
              swap(p,q);
          }
          else ;
      }
      int lg ; for( lg = 1 ; (1 << lg) <= level[p] ; ++lg ); lg--;
      k--;
      for( int i = lg ; i >= 0 ; i-- ){
          if( (1 << i) <= k ){
              p = up[p][i];
              k -= ( 1 << i );
          }
      }
      return p;
  }
  };
```

8.14 Small To Large

```
// Problem:
// https://codeforces.com/contest/600/problem/E

  void process_colors(int curr, int parent) {

  for (int n : adj[curr]) {
    if (n != parent) {
      process_colors(n, curr);

            if (colors[curr].size() < colors[n].size()) {
                sum_num[curr] = sum_num[n];
                vmax[curr] = vmax[n];
        swap(colors[curr], colors[n]);
      }

      for (auto [item,vzs] : colors[n]) {
                if(colors[curr][item]+vzs > vmax[curr]){
                    vmax[curr] = colors[curr][item] + vzs;
                    sum_num[curr] = item;
                }
```

```
                 else if(colors[curr][item]+vzs == vmax[curr]){
                     sum_num[curr] += item;
                 }

                 colors[curr][item] += vzs;
      }
    }
  }

}


int32_t main() {

  int n; cin >> n;

  for (int i = 1; i <= n; i++) {
    int a; cin >> a;
    colors[i][a] = 1;
        vmax[i] = 1;
        sum_num[i] = a;
  }

  for (int i = 1; i < n; i++) {
    int a, b; cin >> a >> b;

    adj[a].push_back(b);
    adj[b].push_back(a);
  }

  process_colors(1, 0);

  for (int i = 1; i <= n; i++) {
    cout << sum_num[i] << (i < n ? " " : "\n");
  }

    return 0;

}

```

8.15 Centroid Decomposition

```
int n;
vector<set<int>> adj;
vector<char> ans;

vector<bool> removed;

vector<int> subtree_size;

int dfs(int u, int p = 0) {
  subtree_size[u] = 1;

  for(int v : adj[u]) {
    if(v != p && !removed[v]) {
      subtree_size[u] += dfs(v, u);
        }
    }

  return subtree_size[u];
}

int get_centroid(int u, int sz, int p = 0) {
  for(int v : adj[u]) {
    if(v != p && !removed[v]) {
      if(subtree_size[v]*2 > sz) {
        return get_centroid(v, sz, u);
            }
        }
    }

  return u;
}

char get_next(char c) {
    if (c != 'Z') return c + 1;
    return '$';
}

bool flag = true;

void solve(int node, char c) {
  int center = get_centroid(node, dfs(node));
    ans[center] = c;
    removed[center] = true;

    for (auto u : adj[center]) {
        if (!removed[u]) {
            char next = get_next(c);
            if (next == '$') {
                flag = false;
                return;
            }
            solve(u, next);
        }
    }
}

int32_t main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    cin >> n;
    adj.resize(n + 1);
    ans.resize(n + 1);
    removed.resize(n + 1);
    subtree_size.resize(n + 1);

    for (int i = 1; i <= n - 1; i++) {
        int u, v; cin >> u >> v;
        adj[u].insert(v);
        adj[v].insert(u);
    }

    solve(1, 'A');

    if (!flag) cout << "Impossible!\n";
    else {
        for (int i = 1; i <= n; i++) {
            cout << ans[i] << ' ';
        }
        cout << '\n';
    }

    return 0;
}
```

8.16 Min Cost Max Flow

```
// Dinitz Min Cost {{{
const int INF = 0x3f3f3f3f3f3f3f3f;

struct Dinitz {
  struct Edge {
    int v, u, cap, flow=0, cost;
    Edge(int v, int u, int cap, int cost) : v(v), u(u), cap(cap), cost(cost) {}
  };

  int n, s, t;
  Dinitz(int n, int s, int t) : n(n), s(s), t(t) {
    adj.resize(n);
  }

  vector<Edge> edges;
  vector<vector<int>> adj;
  void add_edge(int v, int u, int cap, int cost) {
    edges.emplace_back(v, u, cap, cost);
    adj[v].push_back(size(edges)-1);
    edges.emplace_back(u, v, 0, -cost);
    adj[u].push_back(size(edges)-1);
  }

  vector<int> dist;
  bool spfa() {
    dist.assign(n, INF);

    queue<int> Q;
    vector<bool> inqueue(n, false);

    dist[s] = 0;
    Q.push(s);
    inqueue[s] = true;

    vector<int> cnt(n);

    while (!Q.empty()) {
      int v = Q.front(); Q.pop();
      inqueue[v] = false;

      for (auto eid : adj[v]) {
        auto const& e = edges[eid];
        if (e.cap - e.flow <= 0) continue;
        if (dist[e.u] > dist[e.v] + e.cost) {
          dist[e.u] = dist[e.v] + e.cost;
          if (!inqueue[e.u]) {
            Q.push(e.u);
            inqueue[e.u] = true;
          }
        }
      }
    }

    return dist[t] != INF;
  }

  int cost = 0;
  vector<int> ptr;
  int dfs(int v, int f) {
    if (v == t || f == 0) return f;
    for (auto &cid = ptr[v]; cid < size(adj[v]);) {
      auto eid = adj[v][cid];
      auto &e = edges[eid];
      cid++;
      if (e.cap - e.flow <= 0) continue;
      if (dist[e.v] + e.cost != dist[e.u]) continue;
      int newf = dfs(e.u, min(f, e.cap-e.flow));
      if (newf == 0) continue;
      e.flow += newf;
      edges[eid^1].flow -= newf;
      cost += e.cost * newf;
      return newf;
    }
    return 0;
  }

  int total_flow = 0;
  int flow() {
    while (spfa()) {
      ptr.assign(n, 0);
      while (int newf = dfs(s, INF))
        total_flow += newf;
    }
    return total_flow;
  }
};
//}}}
```

8.17 Hungarian

```
// Description:
// A matching algorithm for weighted bipartite graphs that returns
// a perfect match

// Problem:
// https://codeforces.com/gym/103640/problem/H

// Complexity:
// O(V ^ 3) in which V is the number of vertexs

// Notes:
// Indexed at 1

// n is the number of items on the right side and m the number of items
// on the left side of the graph

// Returns minimum assignment cost and which items were matched

pair<int, vector<pii>> hungarian(int n, int m, vector<vector<int>> A) {
  vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
  for (int i=1; i<=n; ++i) {
    p[0] = i;
    int j0 = 0;
    vector<int> minv (m+1, INF);
    vector<char> used (m+1, false);
    do {
      used[j0] = true;
      int i0 = p[j0],  delta = INF,  j1;
      for (int j=1; j<=m; ++j)
        if (!used[j]) {
          int cur = A[i0][j]-u[i0]-v[j];
          if (cur < minv[j])
            minv[j] = cur,  way[j] = j0;
          if (minv[j] < delta)
            delta = minv[j],  j1 = j;
        }
      for (int j=0; j<=m; ++j)
        if (used[j])
          u[p[j]] += delta,  v[j] -= delta;
        else
          minv[j] -= delta;
      j0 = j1;
    } while (p[j0] != 0);
    do {
      int j1 = way[j0];
      p[j0] = p[j1];
      j0 = j1;
    } while (j0);
  }

  vector<pair<int, int>> result;
  for (int i = 1; i <= m; ++i){
    result.push_back(make_pair(p[i], i));
  }

  int C = -v[0];

  return mp(C, result);
}
```

8.18 2sat

```
// Description:
// Solves expression of the type (a v b) ^ (c v d) ^ (e v f)

// Problem:
```

```cpp
// https://cses.fi/problemset/task/1684

// Complexity:
// O(n + m) where n is the number of variables and m is the number of clauses

#include <bits/stdc++.h>
#define pb push_back
#define mp make_pair
#define pii pair<int, int>
#define ff first
#define ss second

using namespace std;

struct SAT {
    int nodes;
    int curr = 0;
    int component = 0;
    vector<vector<int>> adj;
    vector<vector<int>> rev;
    vector<vector<int>> condensed;
    vector<pii> departure;
    vector<bool> visited;
    vector<int> scc;
    vector<int> order;

    // 1 to nodes
    // nodes + 1 to 2 * nodes
    SAT(int nodes) : nodes(nodes) {
        adj.resize(2 * nodes + 1);
        rev.resize(2 * nodes + 1);
        visited.resize(2 * nodes + 1);
        scc.resize(2 * nodes + 1);
    }

    void add_imp(int a, int b) {
        adj[a].pb(b);
        rev[b].pb(a);
    }

    int get_not(int a) {
        if (a > nodes) return a - nodes;
        return a + nodes;
    }

    void add_or(int a, int b) {
        add_imp(get_not(a), b);
        add_imp(get_not(b), a);
    }

    void add_nor(int a, int b) {
        add_or(get_not(a), get_not(b));
    }

    void add_and(int a, int b) {
        add_or(get_not(a), b);
        add_or(a, get_not(b));
        add_or(a, b);
    }

    void add_nand(int a, int b) {
        add_or(get_not(a), b);
        add_or(a, get_not(b));
        add_or(get_not(a), get_not(b));
    }

    void add_xor(int a, int b) {
        add_or(a, b);
        add_or(get_not(a), get_not(b));
    }

    void add_xnor(int a, int b) {
        add_or(get_not(a), b);
        add_or(a, get_not(b));
    }

    void departure_time(int v) {
        visited[v] = true;

        for (auto u : adj[v]) {
            if (!visited[u]) departure_time(u);
        }

        departure.pb(mp(++curr, v));
    }

    void find_component(int v, int component) {
        scc[v] = component;
        visited[v] = true;

        for (auto u : rev[v]) {
            if (!visited[u]) find_component(u, component);
        }
    }

    void topological_order(int v) {
        visited[v] = true;

        for (auto u : condensed[v]) {
            if (!visited[u]) topological_order(u);
        }

        order.pb(v);
    }

    bool is_possible() {
        component = 0;
        for (int i = 1; i <= 2 * nodes; i++) {
            if (!visited[i]) departure_time(i);
        }

        sort(departure.begin(), departure.end(), greater<pii>());

        visited.assign(2 * nodes + 1, false);

        for (auto [_, node] : departure) {
            if (!visited[node]) find_component(node, ++component);
        }

        for (int i = 1; i <= nodes; i++) {
            if (scc[i] == scc[i + nodes]) return false;
        }

        return true;
    }

    int find_value(int e, vector<int> &ans) {
        if (e > nodes && ans[e - nodes] != 2) return !ans[e - nodes];
        if (e <= nodes && ans[e + nodes] != 2) return !ans[e + nodes];
        return 0;
    }

    vector<int> find_ans() {
        condensed.resize(component + 1);

        for (int i = 1; i <= 2 * nodes; i++) {
            for (auto u : adj[i]) {
                if (scc[i] != scc[u]) condensed[scc[i]].pb(scc[u]);
            }
        }

        visited.assign(component + 1, false);

        for (int i = 1; i <= component; i++) {
            if (!visited[i]) topological_order(i);
        }

        reverse(order.begin(), order.end());

        // 0 - false
        // 1 - true
        // 2 - no value yet
        vector<int> ans(2 * nodes + 1, 2);

        vector<vector<int>> belong(component + 1);

        for (int i = 1; i <= 2 * nodes; i++) {
            belong[scc[i]].pb(i);
        }

        for (auto p : order) {
            for (auto e : belong[p]) {
                ans[e] = find_value(e, ans);
            }
        }

        return ans;
    }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m; cin >> n >> m;

    SAT sat = SAT(m);

    for (int i = 0; i < n; i++) {
        char op1, op2; int a, b; cin >> op1 >> a >> op2 >> b;
        if (op1 == ’+’ && op2 == ’+’) sat.add_or(a, b);
        if (op1 == ’-’ && op2 == ’-’) sat.add_or(sat.get_not(a), sat.get_not(b));
        if (op1 == ’+’ && op2 == ’-’) sat.add_or(a, sat.get_not(b));
        if (op1 == ’-’ && op2 == ’+’) sat.add_or(sat.get_not(a), b);
    }

    if (!sat.is_possible()) cout << "IMPOSSIBLE\n";
    else {
        vector<int> ans = sat.find_ans();
        for (int i = 1; i <= m; i++) {
            cout << (ans[i] == 1 ? ’+’ : ’-’) << ’ ’;
        }
        cout << ’\n’;
    }

    return 0;
}
```

8.19 Tarjan Bridge

```
// Description:
// Find a bridge in a connected unidirected graph
// A bridge is an edge so that if you remove that edge the graph is no longer connected

// Problem:
// https://cses.fi/problemset/task/2177/

// Complexity:
// O(V + E) where V is the number of vertices and E is the number of edges

int n;
vector<vector<int>> adj;

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v]) {
                IS_BRIDGE(v, to);
            }
        }
    }
}

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i, -1);
    }
}
```

8.20 Find Cycle

```
bitset<MAX> visited;
vector<int> path;
vector<int> adj[MAX];

bool dfs(int u, int p){

    if (visited[u]) return false;

    path.pb(u);
    visited[u] = true;

    for (auto v : adj[u]){
        if (visited[v] and u != v and p != v){
            path.pb(v); return true;
        }

        if (dfs(v, u)) return true;
    }

    path.pop_back();
    return false;
}

bool has_cycle(int N){

    visited.reset();

    for (int u = 1; u <= N; ++u){
        path.clear();
        if (not visited[u] and dfs(u,-1))
```

```
            return true;
    }

    return false;
}
```

8.21 Prim

```
int n;
vector<vector<int>> adj; // adjacency matrix of graph
const int INF = 1000000000; // weight INF means there is no edge

struct Edge {
    int w = INF, to = -1;
};

void prim() {
    int total_weight = 0;
    vector<bool> selected(n, false);
    vector<Edge> min_e(n);
    min_e[0].w = 0;

    for (int i=0; i<n; ++i) {
        int v = -1;
        for (int j = 0; j < n; ++j) {
            if (!selected[j] && (v == -1 || min_e[j].w < min_e[v].w))
                v = j;
        }

        if (min_e[v].w == INF) {
            cout << "No MST!" << endl;
            exit(0);
        }

        selected[v] = true;
        total_weight += min_e[v].w;
        if (min_e[v].to != -1)
            cout << v << " " << min_e[v].to << endl;

        for (int to = 0; to < n; ++to) {
            if (adj[v][to] < min_e[to].w)
                min_e[to] = {adj[v][to], v};
        }
    }

    cout << total_weight << endl;
}
```

8.22 Blossom

```
// Description:
// Matching algorithm for general graphs (non-bipartite)

// Problem:
// https://acm.timus.ru/problem.aspx?space=1&num=1099

// Complexity:
// O(n^3)

// vector<pii> Blossom(vector<vector<int>>& graph) {
vector<int> Blossom(vector<vector<int>>& graph) {
  int n = graph.size(), timer = -1;
  vector<int> mate(n, -1), label(n), parent(n),
              orig(n), aux(n, -1), q;
  auto lca = [&](int x, int y) {
    for (timer++; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
    }
  };
  auto blossom = [&](int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = mate[v];
      if (label[w] == 1) label[w] = 0, q.push_back(w);
      orig[v] = orig[w] = a; v = parent[w];
    }
  };
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv; mate[pv] = v; v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1; parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.push_back(mate[x]);
        } else if (label[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  // Time halves if you start with (any) maximal matching.
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;

  /*
  vector<bool> used(n, false);
  vector<pii> ans;
  for (int i = 0; i < n; i++) {
    if (matching[i] == -1 || used[i]) continue;
    used[i] = true;
    used[matching[i]] = true;
    ans.emplace_back(i, matching[i]);
  }
  return ans;
  */
}
```

8.23 Dinic

```
// Description:
// Obtains the maximum possible flow rate given a network. A network is a graph with a single source vertex and a single sink vertex in which each edge has a capacity

// Problem:
// https://codeforces.com/gym/103708/problem/J

// Complexity:
```

```cpp
// O(V^2 * E) where V is the number of vertex and E is the number of edges

// Unit network
// A unit network is a network in which for any vertex except source and sink either incoming or outgoing edge is unique and has unit capacity (matching problem).
// Complexity on unit networks: O(E * sqrt(V))

// Unity capacity networks
// A more generic settings when all edges have unit capacities, but the number of incoming and outgoing edges is unbounded
// Complexity on unity capacity networks: O(E * sqrt(E))

// How to use:
// Dinic dinic = Dinic(num_vertex, source, sink);
// dinic.add_edge(vertex1, vertex2, capacity);
// cout << dinic.max_flow() << '\n';

#include <bits/stdc++.h>

#define pb push_back
#define mp make_pair
#define pii pair<int, int>
#define ff first
#define ss second
#define ll long long

using namespace std;

const ll INF = 1e18+10;

struct Edge {
    int from;
    int to;
    ll capacity;
    ll flow;
    Edge* residual;

    Edge() {}

    Edge(int from, int to, ll capacity) : from(from), to(to), capacity(capacity) {
        flow = 0;
    }

    ll get_capacity() {
        return capacity - flow;
    }

    ll get_flow() {
        return flow;
    }

    void augment(ll bottleneck) {
        flow += bottleneck;
        residual->flow -= bottleneck;
    }

    void reverse(ll bottleneck) {
        flow -= bottleneck;
        residual->flow += bottleneck;
    }

    bool operator<(const Edge& e) const {
        return true;
    }
};

struct Dinic {
    int source;
    int sink;
    int nodes;
    ll flow;
    vector<vector<Edge*>> adj;
    vector<int> level;
    vector<int> next;
    vector<int> reach;
    vector<bool> visited;
    vector<vector<int>> path;

    Dinic(int source, int sink, int nodes) : source(source), sink(sink), nodes(nodes) {
        adj.resize(nodes + 1);
    }

    void add_edge(int from, int to, ll capacity) {
        Edge* e1 = new Edge(from, to, capacity);
        Edge* e2 = new Edge(to, from, 0);
        // Edge* e2 = new Edge(to, from, capacity);
        e1->residual = e2;
        e2->residual = e1;
        adj[from].pb(e1);
        adj[to].pb(e2);
    }

    bool bfs() {
        level.assign(nodes + 1, -1);
        queue<int> q;
        q.push(source);
        level[source] = 0;

        while (!q.empty()) {
            int node = q.front();
            q.pop();

            for (auto e : adj[node]) {
                if (level[e->to] == -1 && e->get_capacity() > 0) {
                    level[e->to] = level[e->from] + 1;
                    q.push(e->to);
                }
            }
        }

        return level[sink] != -1;
    }

    ll dfs(int v, ll flow) {
        if (v == sink)
            return flow;

        int sz = adj[v].size();
        for (int i = next[v]; i < sz; i++) {
            Edge* e = adj[v][i];
            if (level[e->to] == level[e->from] + 1 && e->get_capacity() > 0) {
                ll bottleneck = dfs(e->to, min(flow, e->get_capacity()));
                if (bottleneck > 0) {
                    e->augment(bottleneck);
                    return bottleneck;
                }
            }

            next[v] = i + 1;
        }

        return 0;
    }

    ll max_flow() {
        flow = 0;
        while(bfs()) {
            next.assign(nodes + 1, 0);
            ll sent = -1;
            while (sent != 0) {
                sent = dfs(source, INF);
                flow += sent;
            }
        }
        return flow;
    }

    void reachable(int v) {
        visited[v] = true;

        for (auto e : adj[v]) {
            if (!visited[e->to] && e->get_capacity() > 0) {
                reach.pb(e->to);
                visited[e->to] = true;
                reachable(e->to);
            }
        }
    }

    void print_min_cut() {
        reach.clear();
        visited.assign(nodes + 1, false);
        reach.pb(source);
        reachable(source);

        for (auto v : reach) {
            for (auto e : adj[v]) {
                if (!visited[e->to] && e->get_capacity() == 0) {
                    cout << e->from << ’ ’ << e->to << ’\n’;
                }
            }
        }
    }

    ll build_path(int v, int id, ll flow) {
        visited[v] = true;
        if (v == sink) {
            return flow;
        }

        for (auto e : adj[v]) {
            if (!visited[e->to] && e->get_flow() > 0) {
                visited[e->to] = true;
                ll bottleneck = build_path(e->to, id, min(flow, e->get_flow()));
                if (bottleneck > 0) {
                    path[id].pb(e->to);
                    e->reverse(bottleneck);
                    return bottleneck;
                }
            }
        }

        return 0;
    }

    void print_flow_path() {
        path.clear();
        ll sent = -1;
        int id = -1;
        while (sent != 0) {
            visited.assign(nodes + 1, false);
            path.pb(vector<int>{});
            sent = build_path(source, ++id, INF);
                path[id].pb(source);
            }
            path.pop_back();

            for (int i = 0; i < id; i++) {
                cout << path[i].size() << ’\n’;
                reverse(path[i].begin(), path[i].end());
                for (auto e : path[i]) {
                    cout << e << ’ ’;
                }
                cout << ’\n’;
            }
        }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m; cin >> n >> m;

    Dinic dinic = Dinic(1, n, n);

    for (int i = 1; i <= m; i++) {
        int v, u; cin >> v >> u;
        dinic.add_edge(v, u, 1);
    }

    cout << dinic.max_flow() << ’\n’;
    // dinic.print_min_cut();
    // dinic.print_flow_path();

    return 0;
}
```

8.24 Bellman Ford

```
// Description:
// Finds the shortest path from a vertex v to any other vertex

// Problem:
// https://cses.fi/problemset/task/1673

// Complexity:
// O(n * m)

struct Edge {
  int a, b, cost;
  Edge(int a, int b, int cost) : a(a), b(b), cost(cost) {}
};

int n, m;
vector<Edge> edges;
const int INF = 1e9+10;

void bellman_ford(int v, int t) {
  vector<int> d(n + 1, INF);
  d[v] = 0;
  vector<int> p(n + 1, -1);

  for (;;) {
    bool any = false;
    for (Edge e : edges) {
      if (d[e.a] >= INF) continue;
      if (d[e.b] > d[e.a] + e.cost) {
        d[e.b] = d[e.a] + e.cost;
        p[e.b] = e.a;
        any = true;
      }
    }
    if (!any) break;
  }

  if (d[t] == INF)
    cout << "No path from " << v << " to " << t << ".";
  else {
    vector<int> path;
    for (int cur = t; cur != -1; cur = p[cur]) {
      path.push_back(cur);
    }
    reverse(path.begin(), path.end());

    cout << "Path from " << v << " to " << t << ": ";
    for (int u : path) {
      cout << u << ' ';
    }
  }
}
```

8.25 Hld Edge

```
// Description:
// Make queries and updates between two vertexes on a tree

// Problem:
// https://www.spoj.com/problems/QTREE/

// Complexity:
// O(log ^2 n) for both query and update

// How to use:
// HLD hld = HLD(n + 1, adj)

// Notes
// Change the root of the tree on the constructor if it's different from 1
// Use together with Segtree

struct HLD {
  vector<int> parent;
  vector<int> pos;
  vector<int> head;
  vector<int> subtree_size;
  vector<int> level;
  vector<int> heavy_child;
  vector<ftype> subtree_weight;
  vector<ftype> path_weight;
  vector<vector<int>> adj;
  vector<int> at;
  Segtree seg = Segtree(0);
  int cpos;
  int n;
  int root;

  HLD() {}

  HLD(int n, vector<vector<int>>& adj, int root = 1) : adj(adj), n(n), root(root) {
    seg = Segtree(n);
    cpos = 0;
    at.assign(n, 0);
    parent.assign(n, 0);
    pos.assign(n, 0);
    head.assign(n, 0);
    subtree_size.assign(n, 1);
    level.assign(n, 0);
    heavy_child.assign(n, -1);
    parent[root] = -1;
    dfs(root, -1);
    decompose(root, -1);
  }

  void dfs(int v, int p) {
    parent[v] = p;
    if (p != -1) level[v] = level[p] + 1;
    for (auto u : adj[v]) {
      if (u != p) {
        dfs(u, v);
        subtree_size[v] += subtree_size[u];
        if (heavy_child[v] == -1 || subtree_size[u] > subtree_size[heavy_child[v]]) heavy_child[v] = u;
      }
    }
  }

  void decompose(int v, int chead) {
    // start a new path
    if (chead == -1) chead = v;

    // consecutive ids in the hld path
    at[cpos] = v;
    pos[v] = cpos++;
    head[v] = chead;

    // if not a leaf
    if (heavy_child[v] != -1) decompose(heavy_child[v], chead);

    // light child
    for (auto u : adj[v]){
      // start new path
      if (u != parent[v] && u != heavy_child[v]) decompose(u, -1);
    }
  }

  ll query_path(int a, int b) {
    if (a == b) return 0;
    if(pos[a] < pos[b]) swap(a, b);

    if(head[a] == head[b]) return seg.query(pos[b] + 1, pos[a]);
    return seg.f(seg.query(pos[head[a]], pos[a]), query_path(parent[head[a]], b));
  }

  ftype query_subtree(int a) {
    if (subtree_size[a] == 1) return 0;
    return seg.query(pos[a] + 1, pos[a] + subtree_size[a] - 1);
  }

  void update_path(int a, int b, int x) {
    if (a == b) return;
    if(pos[a] < pos[b]) swap(a, b);

    if(head[a] == head[b]) return (void)seg.update(pos[b] + 1, pos[a], x);
    seg.update(pos[head[a]], pos[a], x); update_path(parent[head[a]], b, x);
  }

  void update_subtree(int a, int val) {
    if (subtree_size[a] == 1) return;
    seg.update(pos[a] + 1, pos[a] + subtree_size[a] - 1, val);
  }

  // vertex
  void update(int a, int val) {
    seg.update(pos[a], pos[a], val);
  }

  //edge
  void update(int a, int b, int val) {
    if (parent[a] == b) swap(a, b);
    update(b, val);
  }

  int lca(int a, int b) {
    if(pos[a] < pos[b]) swap(a, b);
    return head[a] == head[b] ? b : lca(parent[head[a]], b);
  }
};
```

8.26 Tree Diameter

```
#include<bits/stdc++.h>

using namespace std;

const int MAX = 3e5+17;

vector<int> adj[MAX];
bool visited[MAX];

int max_depth = 0, max_node = 1;

void dfs (int v, int depth) {
    visited[v] = true;

    if (depth > max_depth) {
        max_depth = depth;
        max_node = v;
    }

    for (auto u : adj[v]) {
        if (!visited[u]) dfs(u, depth + 1);
    }
}

int tree_diameter() {
    dfs(1, 0);
    max_depth = 0;
    for (int i = 0; i < MAX; i++) visited[i] = false;
    dfs(max_node, 0);
    return max_depth;
}
```

9 Geometry

9.1 Shoelace Boundary

```
// Description
// Shoelace formula finds the area of a polygon
// Boundary points return the number of integer points on the edges of a polygon
// not counting the vertexes

// Problem
// https://codeforces.com/gym/101873/problem/G

// Complexity
// O(n)

// before dividing by two
int shoelace(vector<point> & points) {
    int n = points.size();
    vector<point> v(n + 2);

    for (int i = 1; i <= n; i++) {
        v[i] = points[i - 1];
    }
    v[n + 1] = points[0];

    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += (v[i].x * v[i + 1].y - v[i + 1].x * v[i].y);
    }

    sum = abs(sum);
    return sum;
}

int boundary_points(vector<point> & points) {
    int n = points.size();
    vector<point> v(n + 2);

    for (int i = 1; i <= n; i++) {
        v[i] = points[i - 1];
    }
    v[n + 1] = points[0];

    int ans = 0;
    for (int i = 1; i <= n; i++) {
        if (v[i].x == v[i + 1].x) ans += abs(v[i].y - v[i + 1].y) - 1;
        else if (v[i].y == v[i + 1].y) ans += abs(v[i].x - v[i + 1].x) - 1;
        else ans += gcd(abs(v[i].x - v[i + 1].x), abs(v[i].y - v[i + 1].y)) - 1;
    }
    return points.size() + ans;
}
```

9.2 Mindistpair

```
ll MinDistPair(vp &vet){
    int n = vet.size();
    sort(vet.begin(), vet.end());
    set<point> s;

    ll best_dist = LLINF;
    int j=0;
    for(int i=0;i<n;i++){
        ll d = ceil(sqrt(best_dist));
        while(j<n and vet[i].x-vet[j].x >= d){
            s.erase(point(vet[j].y, vet[j].x));
            j++;
        }

        auto it1 = s.lower_bound({vet[i].y - d, vet[i].x});
        auto it2 = s.upper_bound({vet[i].y + d, vet[i].x});

        for(auto it=it1; it!=it2; it++){
            ll dx = vet[i].x - it->y;
            ll dy = vet[i].y - it->x;
            if(best_dist > dx*dx + dy*dy){
                best_dist = dx*dx + dy*dy;
                // vet[i] e inv(it)
            }
        }

        s.insert(point(vet[i].y, vet[i].x));
    }
    return best_dist;
}
```

9.3 2d

```
#define vp vector<point>
#define ld long double
const ld EPS = 1e-6;
const ld PI = acos(-1);

// typedef ll cod;
// bool eq(cod a, cod b){ return (a==b); }
typedef ld cod;
bool eq(cod a, cod b){ return abs(a - b) <= EPS; }

struct point{
    cod x, y;
    int id;
    point(cod x=0, cod y=0): x(x), y(y){}

    point operator+(const point &o) const{ return {x+o.x, y+o.y}; }
    point operator-(const point &o) const{ return {x-o.x, y-o.y}; }
    point operator*(cod t) const{ return {x*t, y*t}; }
    point operator/(cod t) const{ return {x/t, y/t}; }
    cod operator*(const point &o) const{ return x * o.x + y * o.y; }
    cod operator^(const point &o) const{ return x * o.y - y * o.x; }
    bool operator<(const point &o) const{
        return (eq(x, o.x) ? y < o.y : x < o.x);
    }
    bool operator==(const point &o) const{
        return eq(x, o.x) and eq(y, o.y);
    }
  friend ostream& operator<<(ostream& os, point p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};

int ccw(point a, point b, point e){ // -1=dir; 0=collinear; 1=esq;
    cod tmp = (b-a) ^ (e-a); // vector from a to b
    return (tmp > EPS) - (tmp < -EPS);
}

ld norm(point a){ // Modulo
    return sqrt(a * a);
}
cod norm2(point a){
    return a * a;
}
bool nulo(point a){
    return (eq(a.x, 0) and eq(a.y, 0));
}
point rotccw(point p, ld a){
    // a = PI*a/180; // graus
    return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+p.x*sin(a)));
}
point rot90cw(point a) { return point(a.y, -a.x); };
point rot90ccw(point a) { return point(-a.y, a.x); };

ld proj(point a, point b){ // a sobre b
    return a*b/norm(b);
}
ld angle(point a, point b){ // em radianos
    ld ang = a*b / norm(a) / norm(b);
    return acos(max(min(ang, (ld)1), (ld)-1));
}
ld angle_vec(point v){
    // return 180/PI*atan2(v.x, v.y); // graus
    return atan2(v.x, v.y);
}
ld order_angle(point a, point b){ // from a to b ccw (a in front of b)
    ld aux = angle(a,b)*180/PI;
    return ((a^b)<=0 ? aux:360-aux);
}
bool angle_less(point a1, point b1, point a2, point b2){ // ang(a1,b1) <= ang(a2,b2)
    point p1((a1*b1), abs((a1^b1)));
    point p2((a2*b2), abs((a2^b2)));
    return (p1^p2) <= 0;
}

ld area(vp &p){ // (points sorted)
    ld ret = 0;
    for(int i=2;i<(int)p.size();i++)
        ret += (p[i]-p[0])^(p[i-1]-p[0]);
    return abs(ret/2);
}
ld areaT(point &a, point &b, point &c){
    return abs((b-a)^(c-a))/2.0;
}

point center(vp &A){
    point c = point();
    int len = A.size();
    for(int i=0;i<len;i++)
        c=c+A[i];
    return c/len;
}

point forca_mod(point p, ld m){
    ld cm = norm(p);
    if(cm<EPS) return point();
    return point(p.x*m/cm,p.y*m/cm);
}

ld param(point a, point b, point v){
    // v = t*(b-a) + a // return t;
    // assert(line(a, b).inside_seg(v));
    return ((v-a) * (b-a)) / ((b-a) * (b-a));
}

bool simetric(vp &a){ //ordered
    int n = a.size();
    point c = center(a);
    if(n&1) return false;
    for(int i=0;i<n/2;i++)
        if(ccw(a[i], a[i+n/2], c) != 0)
            return false;
    return true;
}

point mirror(point m1, point m2, point p){
    // mirror point p around segment m1m2
    point seg = m2-m1;
    ld t0 = ((p-m1)*seg) / (seg*seg);
    point ort = m1 + seg*t0;
    point pm = ort-(p-ort);
    return pm;
}


///////////
//  Line  //
///////////

struct line{
    point p1, p2;
    cod a, b, c; // ax+by+c = 0;
    // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
    line(point p1=0, point p2=0): p1(p1), p2(p2){
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = p1 ^ p2;
    }
    line(cod a=0, cod b=0, cod c=0): a(a), b(b), c(c){
        // Gera os pontos p1 p2 dados os coeficientes
        // isso aqui eh um lixo mas quebra um galho kkkkkk
        if(b==0){
            p1 = point(1, -c/a);
            p2 = point(0, -c/a);
        }else{
            p1 = point(1, (-c-a*1)/b);
            p2 = point(0, -c/b);
        }
    }

    cod eval(point p){
        return a*p.x+b*p.y+c;
    }
    bool inside(point p){
        return eq(eval(p), 0);
    }
    point normal(){
        return point(a, b);
    }

    bool inside_seg(point p){
        return (
            ((p1-p) ^ (p2-p)) == 0 and
            ((p1-p) * (p2-p)) <= 0
        );
    }

};

// be careful with precision error
vp inter_line(line l1, line l2){
    ld det = l1.a*l2.b - l1.b*l2.a;
    if(det==0) return {};
    ld x = (l1.b*l2.c - l1.c*l2.b)/det;
    ld y = (l1.c*l2.a - l1.a*l2.c)/det;
    return {point(x, y)};
}

// segments not collinear
vp inter_seg(line l1, line l2){
    vp ans = inter_line(l1, l2);
    if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.inside_seg(ans[0]))
        return {};
    return ans;
}
bool seg_has_inter(line l1, line l2){
    // if collinear
    if (l1.inside_seg(l2.p1) || l1.inside_seg(l2.p2) || l2.inside_seg(l1.p1) || l2.inside_seg(l1.p2)) return true;

    return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.p2, l2.p2) < 0 and
           ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.p2, l1.p2) < 0;
}

ld dist_seg(point p, point a, point b){ // point - seg
    if((p-a)*(b-a) < EPS) return norm(p-a);
    if((p-b)*(a-b) < EPS) return norm(p-b);
    return abs((p-a)^(b-a)) / norm(b-a);
}

ld dist_line(point p, line l){ // point - line
    return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
}

line bisector(point a, point b){
    point d = (b-a)*2;
    return line(d.x, d.y, a*a - b*b);
}

line perpendicular(line l, point p){ // passes through p
    return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
}


////////////
// Circle //
////////////

struct circle{
    point c; cod r;
    circle() : c(0, 0), r(0){}
    circle(const point o) : c(o), r(0){}
    circle(const point a, const point b){
        c = (a+b)/2;
        r = norm(a-c);
    }
    circle(const point a, const point b, const point cc){
        assert(ccw(a, b, cc) != 0);
        c = inter_line(bisector(a, b), bisector(b, cc))[0];
        r = norm(a-c);
    }
    bool inside(const point &a) const{
        return norm(a - c) <= r + EPS;
    }
};

pair<point, point> tangent_points(circle cr, point p) {
    ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
    point p1 = rotccw(cr.c-p, -theta);
    point p2 = rotccw(cr.c-p, theta);
    assert(d1 >= cr.r);
    p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
    p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
    return {p1, p2};
}


circle incircle(point p1, point p2, point p3){
    ld m1 = norm(p2-p3);
    ld m2 = norm(p1-p3);
    ld m3 = norm(p1-p2);
    point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
    ld s = 0.5*(m1+m2+m3);
    ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
    return circle(c, r);
}

circle circumcircle(point a, point b, point c) {
    circle ans;
    point u = point((b-a).y, -(b-a).x);
    point v = point((c-a).y, -(c-a).x);
    point n = (c-b)*0.5;
    ld t = (u^n)/(v^u);
    ans.c = ((a+c)*0.5) + (v*t);
    ans.r = norm(ans.c-a);
    return ans;
}

vp inter_circle_line(circle C, line L){
    point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.p1)*(ab) / (ab*ab));
    ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s / (ab*ab);
    if (h2 < -EPS) return {};
    if (eq(h2, 0)) return {p};
    point h = (ab/norm(ab)) * sqrt(h2);
    return {p - h, p + h};
}

vp inter_circle(circle C1, circle C2){
    if(C1.c == C2.c) { assert(C1.r != C2.r); return {}; }
    point vec = C2.c - C1.c;
    ld d2 = vec*vec, sum = C1.r+C2.r, dif = C1.r-C2.r;
    ld p = (d2 + C1.r*C1.r - C2.r*C2.r)/(d2*2), h2 = C1.r*C1.r - p*p*d2;
    if (sum*sum < d2 or dif*dif > d2) return {};
    point mid = C1.c + vec*p, per = point(-vec.y, vec.x) * sqrt(max((ld)0, h2) / d2);
    if(eq(per.x, 0) and eq(per.y, 0)) return {mid};
    return {mid + per, mid - per};
}

// minimum circle cover O(n) amortizado
circle min_circle_cover(vp v){
    random_shuffle(v.begin(), v.end());
    circle ans;
    int n = v.size();
    for(int i=0;i<n;i++) if(!ans.inside(v[i])){
        ans = circle(v[i]);
        for(int j=0;j<i;j++) if(!ans.inside(v[j])){
            ans = circle(v[i], v[j]);
            for(int k=0;k<j;k++) if(!ans.inside(v[k])){
                ans = circle(v[i], v[j], v[k]);
            }
        }
    }
    return ans;
}
```

9.4 Inside Polygon

```
// Description
// Checks if a given point is inside, outside or on the boundary of a polygon

// Problem
// https://cses.fi/problemset/task/2192/

// Complexity
// O(n)

int inside(vp &p, point pp){
    // 1 - inside / 0 - boundary / -1 - outside
    int n = p.size();
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(line({p[i], p[j]}).inside_seg(pp))
            return 0; // boundary
    }
    int inter = 0;
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p[i], p[j], pp)==1)
            inter++; // up
        else if(p[j].x <= pp.x and pp.x < p[i].x and ccw(p[i], p[j], pp)==-1)
            inter++; // down
    }

    if(inter%2==0) return -1; // outside
    else return 1; // inside
}
```

9.5 Convexhull

```
// Graham Scan
struct pt {
    double x, y;
    bool operator == (pt const& t) const {
        return x == t.x && y == t.y;
    }
};

int orientation(pt a, pt b, pt c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) == 0; }

void convex_hull(vector<pt>& a, bool include_collinear = false) {
    pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size()-1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }

    vector<pt> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i], include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }

    if (include_collinear == false && st.size() == 2 && st[0] == st[1])
        st.pop_back();

    a = st;
}


// Monotone Chain
struct pt {
    double x, y;
};

int orientation(pt a, pt b, pt c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool ccw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o > 0 || (include_collinear && o == 0);
}

void convex_hull(vector<pt>& a, bool include_collinear = false) {
    if (a.size() == 1)
        return;

    sort(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.x, a.y) < make_pair(b.x, b.y);
    });
    pt p1 = a[0], p2 = a.back();
    vector<pt> up, down;
    up.push_back(p1);
    down.push_back(p1);
    for (int i = 1; i < (int)a.size(); i++) {
        if (i == a.size() - 1 || cw(p1, a[i], p2, include_collinear)) {
            while (up.size() >= 2 && !cw(up[up.size()-2], up[up.size()-1], a[i], include_collinear))
                up.pop_back();
            up.push_back(a[i]);
        }
        if (i == a.size() - 1 || ccw(p1, a[i], p2, include_collinear)) {
            while (down.size() >= 2 && !ccw(down[down.size()-2], down[down.size()-1], a[i], include_collinear))
                down.pop_back();
            down.push_back(a[i]);
        }
    }

    if (include_collinear && up.size() == a.size()) {
        reverse(a.begin(), a.end());
        return;
    }
    a.clear();
    for (int i = 0; i < (int)up.size(); i++)
        a.push_back(up[i]);
    for (int i = down.size() - 2; i > 0; i--)
        a.push_back(down[i]);
}
```

9.6 Delaunay

```
typedef long long ll;

bool ge(const ll& a, const ll& b) { return a >= b; }
bool le(const ll& a, const ll& b) { return a <= b; }
bool eq(const ll& a, const ll& b) { return a == b; }
bool gt(const ll& a, const ll& b) { return a > b; }
bool lt(const ll& a, const ll& b) { return a < b; }
int sgn(const ll& a) { return a >= 0 ? a ? 1 : 0 : -1; }

struct pt {
    ll x, y;
    pt() { }
    pt(ll _x, ll _y) : x(_x), y(_y) { }
    pt operator-(const pt& p) const {
        return pt(x - p.x, y - p.y);
    }
    ll cross(const pt& p) const {
        return x * p.y - y * p.x;
    }
    ll cross(const pt& a, const pt& b) const {
        return (a - *this).cross(b - *this);
    }
    ll dot(const pt& p) const {
        return x * p.x + y * p.y;
    }
    ll dot(const pt& a, const pt& b) const {
        return (a - *this).dot(b - *this);
    }
    ll sqrLength() const {
        return this->dot(*this);
    }
    bool operator==(const pt& p) const {
        return eq(x, p.x) && eq(y, p.y);
    }
};

const pt inf_pt = pt(1e18, 1e18);

struct QuadEdge {
    pt origin;
    QuadEdge* rot = nullptr;
    QuadEdge* onext = nullptr;
    bool used = false;
    QuadEdge* rev() const {
        return rot->rot;
    }
    QuadEdge* lnext() const {
        return rot->rev()->onext->rot;
    }
    QuadEdge* oprev() const {
        return rot->onext->rot;
    }
    pt dest() const {
        return rev()->origin;
    }
};

QuadEdge* make_edge(pt from, pt to) {
    QuadEdge* e1 = new QuadEdge;
    QuadEdge* e2 = new QuadEdge;
    QuadEdge* e3 = new QuadEdge;
    QuadEdge* e4 = new QuadEdge;
    e1->origin = from;
    e2->origin = to;
    e3->origin = e4->origin = inf_pt;
    e1->rot = e3;
    e2->rot = e4;
    e3->rot = e2;
    e4->rot = e1;
    e1->onext = e1;
    e2->onext = e2;
    e3->onext = e4;
    e4->onext = e3;
    return e1;
}

void splice(QuadEdge* a, QuadEdge* b) {
    swap(a->onext->rot->onext, b->onext->rot->onext);
    swap(a->onext, b->onext);
}

void delete_edge(QuadEdge* e) {
    splice(e, e->oprev());
    splice(e->rev(), e->rev()->oprev());
    delete e->rev()->rot;
    delete e->rev();
    delete e->rot;
    delete e;
}
```

```

QuadEdge* connect(QuadEdge* a, QuadEdge* b) {
    QuadEdge* e = make_edge(a->dest(), b->origin);
    splice(e, a->lnext());
    splice(e->rev(), b);
    return e;
}

bool left_of(pt p, QuadEdge* e) {
    return gt(p.cross(e->origin, e->dest()), 0);
}

bool right_of(pt p, QuadEdge* e) {
    return lt(p.cross(e->origin, e->dest()), 0);
}

template <class T>
T det3(T a1, T a2, T a3, T b1, T b2, T b3, T c1, T c2, T c3) {
    return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) +
           a3 * (b1 * c2 - c1 * b2);
}

bool in_circle(pt a, pt b, pt c, pt d) {
// If there is __int128, calculate directly.
// Otherwise, calculate angles.
#if defined(__LP64__) || defined(_WIN64)
    __int128 det = -det3<__int128>(b.x, b.y, b.sqrLength(), c.x, c.y,
                                   c.sqrLength(), d.x, d.y, d.sqrLength());
    det += det3<__int128>(a.x, a.y, a.sqrLength(), c.x, c.y, c.sqrLength(), d.x,
                          d.y, d.sqrLength());
    det -= det3<__int128>(a.x, a.y, a.sqrLength(), b.x, b.y, b.sqrLength(), d.x,
                          d.y, d.sqrLength());
    det += det3<__int128>(a.x, a.y, a.sqrLength(), b.x, b.y, b.sqrLength(), c.x,
                          c.y, c.sqrLength());
    return det > 0;
#else
    auto ang = [](pt l, pt mid, pt r) {
        ll x = mid.dot(l, r);
        ll y = mid.cross(l, r);
        long double res = atan2((long double)x, (long double)y);
        return res;
    };
    long double kek = ang(a, b, c) + ang(c, d, a) - ang(b, c, d) - ang(d, a, b);
    if (kek > 1e-8)
        return true;
    else
        return false;
#endif
}

pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<pt>& p) {
    if (r - l + 1 == 2) {
        QuadEdge* res = make_edge(p[l], p[r]);
        return make_pair(res, res->rev());
    }
    if (r - l + 1 == 3) {
        QuadEdge *a = make_edge(p[l], p[l + 1]), *b = make_edge(p[l + 1], p[r]);
        splice(a->rev(), b);
        int sg = sgn(p[l].cross(p[l + 1], p[r]));
        if (sg == 0)
            return make_pair(a, b->rev());
        QuadEdge* c = connect(b, a);
        if (sg == 1)
            return make_pair(a, b->rev());
        else
            return make_pair(c->rev(), c);
    }
    int mid = (l + r) / 2;
    QuadEdge *ldo, *ldi, *rdo, *rdi;
    tie(ldo, ldi) = build_tr(l, mid, p);
    tie(rdi, rdo) = build_tr(mid + 1, r, p);
    while (true) {
        if (left_of(rdi->origin, ldi)) {
            ldi = ldi->lnext();
            continue;
        }
        if (right_of(ldi->origin, rdi)) {
            rdi = rdi->rev()->onext;
            continue;
        }
        break;
    }
    QuadEdge* basel = connect(rdi->rev(), ldi);
    auto valid = [&basel](QuadEdge* e) { return right_of(e->dest(), basel); };
    if (ldi->origin == ldo->origin)
        ldo = basel->rev();
    if (rdi->origin == rdo->origin)
        rdo = basel;
    while (true) {
        QuadEdge* lcand = basel->rev()->onext;
        if (valid(lcand)) {
            while (in_circle(basel->dest(), basel->origin, lcand->dest(),
                             lcand->onext->dest())) {
                QuadEdge* t = lcand->onext;
                delete_edge(lcand);
                lcand = t;
            }
        }
        QuadEdge* rcand = basel->oprev();
        if (valid(rcand)) {
            while (in_circle(basel->dest(), basel->origin, rcand->dest(),
                             rcand->oprev()->dest())) {
                QuadEdge* t = rcand->oprev();
                delete_edge(rcand);
                rcand = t;
            }
        }
        if (!valid(lcand) && !valid(rcand))
            break;
        if (!valid(lcand) ||
            (valid(rcand) && in_circle(lcand->dest(), lcand->origin,
                                       rcand->origin, rcand->dest())))
            basel = connect(rcand, basel->rev());
        else
            basel = connect(basel->rev(), lcand->rev());
    }
    return make_pair(ldo, rdo);
}

vector<tuple<pt, pt, pt>> delaunay(vector<pt> p) {
    sort(p.begin(), p.end(), [](const pt& a, const pt& b) {
        return lt(a.x, b.x) || (eq(a.x, b.x) && lt(a.y, b.y));
    });
    auto res = build_tr(0, (int)p.size() - 1, p);
    QuadEdge* e = res.first;
    vector<QuadEdge*> edges = {e};
    while (lt(e->onext->dest().cross(e->dest(), e->origin), 0))
        e = e->onext;
    auto add = [&p, &e, &edges]() {
        QuadEdge* curr = e;
        do {
            curr->used = true;
            p.push_back(curr->origin);
            edges.push_back(curr->rev());
            curr = curr->lnext();
        } while (curr != e);
    };
    add();
    p.clear();
    int kek = 0;
    while (kek < (int)edges.size()) {
        if (!(e = edges[kek++])->used)
            add();
    }
    vector<tuple<pt, pt, pt>> ans;
    for (int i = 0; i < (int)p.size(); i += 3) {
        ans.push_back(make_tuple(p[i], p[i + 1], p[i + 2]));
    }
    return ans;
}
```

9.7 Closest Pair Points

```
// Description
// Find the squared distance between the closest two points among n points
// Also finds which pair of points is closest (could be more than one)

// Problem
// https://cses.fi/problemset/task/2194/

// Complexity
// O(n log n)

ll closest_pair_points(vp &vet){
    pair<point, point> ans;
    int n = vet.size();
    sort(vet.begin(), vet.end());
    set<point> s;

    ll best_dist = LLONG_MAX;
    int j=0;
    for(int i=0;i<n;i++){
        ll d = ceil(sqrt(best_dist));
        while(j<n and vet[i].x-vet[j].x >= d){
            s.erase(point(vet[j].y, vet[j].x));
            j++;
        }

        auto it1 = s.lower_bound({vet[i].y - d, vet[i].x});
        auto it2 = s.upper_bound({vet[i].y + d, vet[i].x});

        for(auto it=it1; it!=it2; it++){
            ll dx = vet[i].x - it->y;
            ll dy = vet[i].y - it->x;

            if(best_dist > dx*dx + dy*dy){
                best_dist = dx*dx + dy*dy;
                // closest pair points
                ans = mp(vet[i], point(it->y, it->x));
            }
        }

        s.insert(point(vet[i].y, vet[i].x));
    }

    // best distance squared
    return best_dist;
}
```