Team notebook

Ho Chi Minh City University of Science

October 6, 2024

Contents

1 Data-Structures

1.1 AUGIT

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef pair<int,int> Interval;
const Interval NO_INTERVAL_FOUND = {1,0};
template <class Node_CItr, class Node_Itr, class
    Cmp_Fn, class _Alloc> //Augmented IT
struct interval_node_update_policy{
    typedef int metadata_type;
    bool doOverlap(Interval i1, Node_CItr i2){
        return (i1.first <= (*i2)->second &&
             (*i2)->first <= i1.second);
    }
    Interval overlapSearch(Interval i){
        for(Node_CItr it = node_begin(); it !=
             node_end();){
            if(doOverlap(i,it)){
                return {(*it)->first,
                     (*it)->second};
            }
            if(it.get_l_child() != node_end() &&
                 it.get_l_child().get_metadata() >=
                 i.first){
                it = it.get_l_child();
            }
            else{
                it = it.get_r_child();
            }
        }
        return NO_INTERVAL_FOUND;
    }
    void operator()(Node_Itr it, Node_CItr
        end_it){
        int max_high = (*it)->second;
        if(it.get_l_child() != end_it){
            max_high =
                max(max_high,it.get_l_child().get_metadata());
        }
        if(it.get_r_child() != end_it){
            max_high = max(max_high,
                it.get_r_child().get_metadata());
        }
        const_cast<int&>(it.get_metadata()) =
            max_high;
    }
    virtual Node_CItr node_begin() const = 0;
    virtual Node_CItr node_end() const = 0;
    virtual ~interval_node_update_policy() {}
};
typedef
    tree<Interval,null_type,less<Interval>,rb_tree_tag,interval_node_update_policy>
    IntervalTree;
//All operations include .erase(),
    .overlapSearch(), .insert()
//Initialize an IT by: "IntervalTree IT;"
int main(){
}
```

1.2 DSU

```
#include<bits/stdc++.h>
using namespace std;
struct DSU{
    vector<int> parent;
    vector<int> size;
    vector<int> rank;
    DSU(int maxn){
        parent.assign(maxn,-1);
        size.assign(maxn,-1);
        rank.assign(maxn,-1);
    }
    void makeset(int v){
        parent[v] = v;
        size[v] = 1;
        rank[v] = 0;
    }
    int findset(int v){ //Path Compression
        if(v == parent[v]){
            return v;
        }
        return findset(parent[v]);
    }
    void unionsizerank(int a, int b){ //Ultra
        Combination
        a = findset(a);
        b = findset(b);
        if(a != b){
            if(rank[a] < rank[b]){
                swap(a,b);
            }
            else if(size[a] < size[b]){
                swap(a,b);
            }
        }
        parent[b] = a;
        if(rank[a] == rank[b]){
            rank[a]++;
        }
        size[a] += size[b];
    }
    void unionsize(int a, int b){ //Path
        Compression + Size Heuristics
        a = findset(a);
        b = findset(b);
        if(a != b){
            if(size[a] < size[b]){
                swap(a,b);
            }
        }
        parent[b] = a;
        size[a] += size[b];
    }
    void unionrank(int a, int b){ //Path
        Compression + Rank Heuristics
        a = findset(a);
        b = findset(b);
        if(a != b){
            if(rank[a] < rank[b]){
                swap(a,b);
            }
        }
        parent[b] = a;
        if(rank[b] == rank[a]){
            rank[a]++;
        }
    }
    void pathcompression(int a, int b){ //No
        Heuristics + Path Compression
        a = findset(a);
        b = findset(b);
        if(a != b){
            parent[b] = a;
        }
    }
};
int main(){
}
```

1.3 SegmentTree

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

struct SegmentTree
{
    int n; const int INF = 1e9+7;
    vector<int> t;
    vector<int> lazy;

    SegmentTree(int n)
    {
        this->n = n;
        t.assign(4 * (n + 1), 0);
        lazy.assign(4 * (n + 1), 0);
}

void build(int v, int tl, int tr, vector<int>
    &a)
{
    if (tl == tr)
    {
        t[v] = a[tl];
    }
    else
    {
        int tm = (tl + tr) / 2;
        build(v * 2, tl, tm, a);
        build(v * 2 + 1, tm + 1, tr, a);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }
}

int sum(int v, int tl, int tr, int l, int r)
{
    if (l > r)
        return 0;
    if (l == tl && r == tr)
    {
        return t[v];
    }
    int tm = (tl + tr) / 2;
    return sum(v * 2, tl, tm, l, min(r, tm)) +
        sum(v * 2 + 1, tm + 1, tr, max(l, tm +
        1), r);
}

void update(int v, int tl, int tr, int pos,
    int new_val) //Single update
{
    if (tl == tr)
    {
        t[v] = new_val;
    }
    else
    {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v * 2, tl, tm, pos, new_val);
        else
            update(v * 2 + 1, tm + 1, tr, pos,
                new_val);
        t[v] = t[v * 2] + t[v * 2 + 1];
    }
}

void push(int v)
{
    t[v * 2] += lazy[v];
    lazy[v * 2] += lazy[v];
    t[v * 2 + 1] += lazy[v];
    lazy[v * 2 + 1] += lazy[v];
    lazy[v] = 0;
}

void update(int v, int tl, int tr, int l, int
    r, int addend) //Range update
{
    if (l > r)
        return;
    if (l == tl && tr == r)
    {
        t[v] += addend;
        lazy[v] += addend;
    }
    else
    {
        push(v);
        int tm = (tl + tr) / 2;
        update(v * 2, tl, tm, l, min(r, tm),
            addend);
        update(v * 2 + 1, tm + 1, tr, max(l,
            tm + 1), r, addend);
        t[v] = max(t[v * 2], t[v * 2 + 1]);
    }
}

int query(int v, int tl, int tr, int l, int
    r) //Query range with Lazy
{
    if (l > r)
        return -INF;
    if (l == tl && tr == r)
            return t[v];
        push(v);
        int tm = (tl + tr) / 2;
        return max(query(v * 2, tl, tm, l, min(r,
            tm)),
                   query(v * 2 + 1, tm + 1, tr,
                       max(l, tm + 1), r));
    }
};

int main()
{
}
```

1.4 SegmentTree2D

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

struct SegmentTree2D
{
    int n, m;
    vector<vector<int>> t;
    vector<vector<int>> a;

    SegmentTree2D(int n, int m)
    {
        this->n = n;
        this->m = m;
        t.assign(4 * n, vector<int>(4 * m));
    }

    SegmentTree2D(vector<vector<int>> &a)
    {
        this->a = a;
    }

    void build_y(int vx, int lx, int rx, int vy,
        int ly, int ry)
    {
        if (ly == ry)
        {
```

```
        if (lx == rx)
            t[vx][vy] = a[lx][ly];
        else
            t[vx][vy] = t[vx * 2][vy] + t[vx *
                2 + 1][vy];
    }
    else
    {
        int my = (ly + ry) / 2;
        build_y(vx, lx, rx, vy * 2, ly, my);
        build_y(vx, lx, rx, vy * 2 + 1, my +
            1, ry);
        t[vx][vy] = t[vx][vy * 2] + t[vx][vy *
            2 + 1];
    }
}

void build_x(int vx, int lx, int rx)
{
    if (lx != rx)
    {
        int mx = (lx + rx) / 2;
        build_x(vx * 2, lx, mx);
        build_x(vx * 2 + 1, mx + 1, rx);
    }
    build_y(vx, lx, rx, 1, 0, m - 1);
}

int sum_y(int vx, int vy, int tly, int try_,
    int ly, int ry)
{
    if (ly > ry)
        return 0;
    if (ly == tly && try_ == ry)
        return t[vx][vy];
    int tmy = (tly + try_) / 2;
    return sum_y(vx, vy * 2, tly, tmy, ly,
        min(ry, tmy)) + sum_y(vx, vy * 2 + 1,
        tmy + 1, try_, max(ly, tmy + 1), ry);
}

int sum_x(int vx, int tlx, int trx, int lx,
    int rx, int ly, int ry)
{
        if (lx > rx)
            return 0;
        if (lx == tlx && trx == rx)
            return sum_y(vx, 1, 0, m - 1, ly, ry);
        int tmx = (tlx + trx) / 2;
        return sum_x(vx * 2, tlx, tmx, lx, min(rx,
            tmx), ly, ry) + sum_x(vx * 2 + 1, tmx
            + 1, trx, max(lx, tmx + 1), rx, ly,
            ry);
    }
};

int main()
{

}
```

1.5 stringHash

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

long long compute_hash(string const& s) {
    const int p = 31;
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) *
            p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
```

1.6 Treap

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
using namespace std;
struct item{
    int key, prior;
    item *l, *r;
    item () { }
    item (int key) : key(key), prior(rand()),
        l(NULL), r(NULL) { }
    item (int key, int prior) : key(key),
        prior(prior), l(NULL), r(NULL) { }
};
typedef item* pitem;
void split (pitem t, int key, pitem & l, pitem &
    r) {
    if (!t)
        l = r = NULL;
    else if (t->key <= key)
        split (t->r, key, t->r, r), l = t;
    else
        split (t->l, key, l, t->l), r = t;
}
void insert (pitem & t, pitem it) {
    if (!t)
        t = it;
    else if (it->prior > t->prior)
        split (t, it->key, it->l, it->r), t = it;
    else
        insert (t->key <= it->key ? t->r : t->l,
            it);
}
void merge (pitem & t, pitem l, pitem r) {
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r), t = l;
    else
        merge (r->l, l, r->l), t = r;
}
void erase (pitem & t, int key) {
    if (t->key == key) {
        pitem th = t;
        merge (t, t->l, t->r);
        delete th;
    }
    else
        erase (key < t->key ? t->l : t->r, key);
}
pitem unite (pitem l, pitem r) {
    if (!l || !r) return l ? l : r;
    if (l->prior < r->prior) swap (l, r);
    pitem lt, rt;
    split (r, l->key, lt, rt);
    l->l = unite (l->l, lt);
    l->r = unite (l->r, rt);
    return l;
}
pitem search(pitem t, int key){
    if(t == NULL || t->key == key){
        return t;
    }
    if(t->key < key){
        return search(t->l,key);
    }
    return search(t->r,key);
}
pitem newItem(int key){
    pitem temp = new item(key);
    temp->l = NULL, temp->r = NULL;
    return temp;
}
int main(){
}
```

2 DP

2.1 CHT

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

//convex hull trick
// Decreasing Insertion, Query Min
struct CHT {
    vector<long long> a, b;
    bool cross(int i, int j, int k) {
        return (a[j] - a[i]) * (b[k] - b[i]) >=
            (a[k] - a[i]) * (b[j] - b[i]);
    }

    void add(long long A, long long B) {
        a.push_back(A);
        b.push_back(B);
        while (a.size() > 2 && cross(a.size() - 3,
             a.size() - 2, a.size() - 1)) {
            a.erase(a.end() - 2);
            b.erase(b.end() - 2);
        }
    }

    long long query(long long x) {
        int l = 0, r = a.size() - 1;
        while (l < r) {
            int mid = l + (r - l) / 2;
            long long f1 = a[mid] * x + b[mid];
            long long f2 = a[mid + 1] * x + b[mid
                + 1];
            if (f1 > f2)
                l = mid + 1;
            else
                r = mid;
        }
        return a[l] * x + b[l];
    }
};
```

2.2 DNC

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

//divide and conquer
const long long INF = 1e18;
const int MAXN = 3e3 + 5;
const int MAXM = 3e3 + 5;
int n;
long long f[MAXN][MAXM];
long long c[MAXN][MAXM];
long long sum[MAXN];

long long Cost(int i, int j) {
    if (i > j)
        return 0;
    long long ans = sum[j] - sum[i - 1];
    return ans * ans;
}

void divide(int i, int L, int R, int optL, int
     optR) {
    if (L > R)
        return;
    int mid = (L + R) / 2, cut = optL;
    f[i][mid] = INF;
    for (int k = optL; k <= min(mid, optR); k++) {
        long long cur = f[i - 1][k] + Cost(k + 1,
             mid);
        if (f[i][mid] > cur) {
            f[i][mid] = cur;
            cut = k;
        }
    }
    divide(i, L, mid - 1, optL, cut);
    divide(i, mid + 1, R, cut, optR);
}

void solve() {
    long long n, k;
    cin >> n >> k;
    for (int i = 1; i <= n; i++)
        cin >> sum[i];
    for (int i = 1; i <= n; i++)
        sum[i] += sum[i - 1];
    for (int i = 1; i <= n; i++)
        f[1][i] = Cost(1, i);
    for (int i = 2; i <= k; i++) {
        divide(i, 1, n, 1, n);
    }
    cout << f[k][n];
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    solve();
    return 0;
}
```

2.3 Knuth

```
/*
* Complexity: O(N^2)
* f[i][j] = min(f[i][k] + f[k][j] + c[i][j], i <
    k < j)
* a[i][j] = min(k | i < k < j && f[i][j] =
    f[i][k] + f[k][j]
+ c[i][j])
* Sufficient condition: a[i][j - 1] <= a[i][j] <=
    a[i + 1][j
] or
* c[x][z] + c[y][t] <= c[x][t] + c[y][z]
    (quadrangle
inequality) and c[y][z] <= c[x][t]
    (monotonicity), x <=
y <= z <= t
*/
#include <bits/stdc++.h>
using namespace std;

const int INF = (int)1e9;
const int MAXN = 2e3 + 5;
int n;
int f[MAXN][MAXN];
int c[MAXN][MAXN];
int a[MAXN][MAXN];
long long v[MAXN];

void knuth() {
    for (int i = 1; i <= n; i++) {
        f[i][i] = 0;
        a[i][i] = i;
    }
    for (int len = 1; len <= n - 1; len++)
        for (int i = 1; i <= n - len; i++) {
            int j = i + len;
            f[i][j] = INF;
            for (int k = a[i][j - 1]; k <= a[i +
                1][j]; k++) {
                if (f[i][j] > f[i][k - 1] + f[k][j]
                    + c[i][j]) {
                    f[i][j] = f[i][k - 1] + f[k][j]
                        + c[i][j];
                    a[i][j] = k;
                }
            }
        }
    cout << f[1][n] << '\n';
}
```

3 Geometry

3.1 Angle

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

typedef long long ll;
/**
 * Description: A class for ordering angles (as
     represented
by int points and
* a number of rotations around the origin).
    Useful for
rotational sweeping.
* Sometimes also represents points or vectors.
* Usage:
* vector<Angle> v = {w[0], w[0].t360() ...}; //
    sorted
* int j = 0; rep(i,0,n) { while (v[j] <
    v[i].t180()) ++j; } * // sweeps j such that
    (j-i) represents the number of
     positively oriented triangles with vertices
         at 0 and i
 * Status: Used, works well
 */

#pragma once
struct Angle{
    int x, y;
    int t;
    Angle (int x, int y, int t=0) : x(x), y(y),
        t(t) {}
    Angle operator-(Angle b) const { return
        {x-b.x, y-b.y, t};}
    int half() const {
        assert(x || y);
        return y < 0 || (y == 0 && x < 0);
    }
    Angle t90() const { return {-y, x, t +
        (half() && x >= 0)};}
    Angle t180() const { return {-x, -y, t +
        half()}; }
    Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
 // add a.dist2() and b.dist2() to also compare
     distances
    return make_tuple(a.t, a.half(), a.y *
        (ll)b.x) < make_tuple(b.t, b.half(), a.x
        * (ll)b.y);
}
// Given two points, this calculates the smallest
    angle between them, i.e., the angle that
    covers the defined line
// segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle
    b){
    if(b<a) swap(a,b);
    return (b < a.t180() ?
        make_pair(a, b) : make_pair(b, a.t360())
    );
};
Angle operator+(Angle a, Angle b) { // point a +
    vector b
    Angle r(a.x + b.x, a.y + b.y, a.t);
    if (a.t180() < r) r.t--;
    return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b -
    angle a
    int tu = b.t - a.t; a.t = b.t;
    return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x,
        tu - (b < a)
    };
}
```

3.2 CircleIntersection

```
/**
 * Description: Computes the pair of points at
     which two
 * circles intersect. Returns false in case of no
     intersection.
 * Status: stress-tested
 */

#pragma once
#include "bits/stdc++.h"
#include <Point.h>

typedef Point<double> P;

bool circleInter(P a, P b, double r1, double r2,
    pair<P, P>* out) {
    if (a == b) {
        assert(r1 != r2);
        return false;
    }

    P vec = b - a;
    double d2 = vec.dist2();
    double sum = r1 + r2;
    double dif = r1 - r2;
    double p = (d2 + r1*r1 - r2*r2) / (d2 * 2);
    double h2 = r1*r1 - p*p*d2;

    if (sum*sum < d2 || dif*dif > d2)
        return false;

    P mid = a + vec * p;
    P per = vec.perp() * sqrt(fmax(0, h2) / d2);

    *out = {mid + per, mid - per};
    return true;
}
```

3.3 CircleLine

```
/*
* Description: Finds the intersection between a
    circle and
     a line.
 * Returns a vector of either 0, 1, or 2
     intersection points
     .
 * P is intended to be Point<double>.
 */
#pragma once
#include <bits/stdc++.h>
#include <Point.h>
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
 P ab = b - a, p = a + ab * (c-a).dot(ab) /
     ab.dist2();
 double s = a.cross(b, c), h2 = r*r - s*s /
     ab.dist2();
 if (h2 < 0) return {};
 if (h2 == 0) return {p};
 P h = ab.unit() * sqrt(h2);
 return {p - h, p + h};
}
```

3.4 CirclePolygonIntersection

```
/**
 * Description: Returns the area of the
     intersection of a circle with a
 * ccw polygon.
 * Time: O(n)
 * Status: Tested on GNYR 2019 Gerrymandering,
     stress-tested
 */
#pragma once
#include <bits/stdc++.h>

typedef Point<double> P;

#define arg(p, q) atan2(p.cross(q), p.dot(q))

double circlePoly(P c, double r, vector<P> ps) {
    auto tri = [&](P p, P q) {
        auto r2 = r * r / 2;
        P d = q - p;
        auto a = d.dot(p) / d.dist2();
        auto b = (p.dist2() - r * r) / d.dist2();
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a - sqrt(det));
        auto t = min(1., -a + sqrt(det));
        if (t < 0 || 1 <= s) return arg(p, q) * r2;
        P u = p + d * s;
        P v = p + d * t;
        return arg(p, u) * r2 + u.cross(v) / 2 +
            arg(v, q) * r2;
    };

    auto sum = 0.0;
    rep(i, 0, sz(ps)) {
        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)]
            - c);
    }

    return sum;
}
```

3.5 CircleTangents

```
/**
 * Description: Finds the external tangents of
     two circles,
     or internal if r2 is negated.
 * Can return 0, 1, or 2 tangents -- 0 if one
     circle
     contains the other (or overlaps it, in the
         internal
     case, or if the circles are the same);
 * 1 if the circles are tangent to each other (in
     which case
       .first = .second and the tangent line is
           perpendicular
       to the line between the centers).
 * .first and .second give the tangency points at
     circle 1
     and 2 respectively.
 * To find the tangents of a circle with a point
     set r2 to
     0.
 * Status: tested
*/
#pragma once
#include "Point.h"
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P
    c2, double r2
){
P d = c2 - c1;
 double dr = r1 - r2, d2 = d.dist2(), h2 = d2 -
     dr * dr;
 if (d2 == 0 || h2 < 0) return {};
 vector<pair<P, P>> out;
 for (double sign : {-1, 1}) {
  P v = (d * dr + d.perp() * sqrt(h2) * sign) /
      d2;
  out.push_back({c1 + v * r1, c2 + v * r2});
 }
 if (h2 == 0) out.pop_back();
return out; }
```

3.6 circumcircle

```
/**
 * Description:\\
 * The circumcirle of a triangle is the circle
     intersecting
     all three vertices. ccRadius returns the
         radius of the
     circle going through points A, B and C and
         ccCenter
     returns the center of the same circle.
 * Status: tested
*/
#pragma once
#include "Point.h"
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P&
    C) {
 return (B-A).dist()*(C-B).dist()*(A-C).dist()/
   abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
 P b = C-A, c = B-A;
 return A +
     (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

3.7 ClosestPair

```
/**
 * Source: https://codeforces.com/blog/entry/58747
 * Description: Finds the closest pair of points.
 */
#pragma once
#include "Point.h"
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
 assert(sz(v) > 1);
 set<P> S;
 sort(all(v), [](P a, P b) { return a.y < b.y; });
 pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
 int j = 0;
 for (P p : v) {
  P d{1 + (ll)sqrt(ret.first), 0};
  while (v[j].y <= p.y - d.x) S.erase(v[j++]);
  auto lo = S.lower_bound(p - d), hi =
      S.upper_bound(p + d);
  for (; lo != hi; ++lo)
   ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
  S.insert(p);
}
 return ret.second;
}
```

3.8 ConvexHull

```
/**
Returns a vector of the points of the convex hull
    in counter
    -clockwise order.
Points on the edge of the hull between two other
    points are
    not considered part of the hull.
*/
#pragma once
#include "Point.h"
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
 if (sz(pts) <= 1) return pts;
 sort(all(pts));
 vector<P> h(sz(pts)+1);
 int s = 0, t = 0;
 for (int it = 2; it--; s = --t,
     reverse(all(pts)))
  for (P p : pts) {
   while (t >= s + 2 && h[t-2].cross(h[t-1], p)
       <= 0) t--;
h[t++] = p; }
 return {h.begin(), h.begin() + t - (t == 2 &&
     h[0] == h[1])
     };
}
```

3.9 DelaunayTriangulation

```
/**
 * Author: Mattias de Zalenski
 * Date: Unknown
 * Source: Geometry in C
 * Description: Computes the Delaunay
    triangulation of a set
 * of points.
 * Each circumcircle contains none of the input
    points.
 * If any three points are collinear or any four
    are on the same circle,
 * behavior is undefined.
 * Time: O(n^2)
 * Status: stress-tested
 */

#pragma once
#include "Point.h"
#include "3dHull.h"

template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
    if (sz(ps) == 3) {
        int d = (ps[0].cross(ps[1], ps[2]) < 0);
        trifun(0, 1 + d, 2 - d);
    }

    vector<P3> p3;
    for (P p : ps)
        p3.emplace_back(p.x, p.y, p.dist2());

    if (sz(ps) > 3) {
        for (auto t : hull3d(p3)) {
            if ((p3[t.b] - p3[t.a])
                    .cross(p3[t.c] - p3[t.a])
                    .dot(P3(0, 0, 1)) < 0) {
                trifun(t.a, t.c, t.b);
            }
        }
    }
}
```

3.10 FastDelaunay

```
/**
 * Author: Philippe Legault
 * Date: 2016
 * License: MIT
 * Source: https://github.com/Bathlamos/delaunay-
    triangulation/
 * Description: Fast Delaunay triangulation.
 * Each circumcircle contains none of the input
    points.
 * There must be no duplicate points.
 * If all points are on a line, no triangles will
    be
    returned.
 * Should work for doubles as well, though there
    may be
    precision issues in   circ  .
 * Returns triangles in order \{t[0][0], t[0][1],
    t[0][2], t
    [1][0], \dots\}, all counter-clockwise.
 * Time: O(n \log n)
 * Status: stress-tested
 */
#pragma once
#include "Point.h"
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords
    are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any
    other point
struct Quad {
 Q rot, o; P p = arb; bool mark;
 P& F() { return r()->p; }
 Q& r() { return rot->rot; }
 Q prev() { return rot->o->rot; }
 Q next() { return r()->prev(); }
} *H;
bool circ(P p, P a, P b, P c) { // is p in the
    circumcircle?
 lll p2 = p.dist2(), A = a.dist2()-p2,
     B = b.dist2()-p2, C = c.dist2()-p2;
 return p.cross(a,b)*C + p.cross(b,c)*A +
    p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
 Q r = H ? H : new Quad{new Quad{new Quad{new
     Quad{0}}}};
 H = r->o; r->r()->r() = r;
 rep(i,0,4) r=r->rot, r->p = arb, r->o = i & 1 ?
     r : r->r();
 r->p = orig; r->F() = dest;
return r; }
void splice(Q a, Q b) {
 swap(a->o->rot->o, b->o->rot->o); swap(a->o,
     b->o);
}
Q connect(Q a, Q b) {
 Q q = makeEdge(a->F(), b->p);
 splice(q, a->next());
 splice(q->r(), b);
 return q;
}
pair<Q,Q> rec(const vector<P>& s) {
 if (sz(s) <= 3) {
  Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1],
      s.back());
  if (sz(s) == 2) return { a, a->r() };
  splice(a->r(), b);
  auto side = s[0].cross(s[1], s[2]);
  Q c = side ? connect(b, a) : 0;
  return {side < 0 ? c->r() : a, side < 0 ? c :
      b->r() };
 }
#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
 Q A, B, ra, rb;
 int half = sz(s) / 2;
 tie(ra, A) = rec({all(s) - half});
 tie(B, rb) = rec({sz(s) - half + all(s)});
 while ((B->p.cross(H(A)) < 0 && (A = A->next()))
     ||
      (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
 Q base = connect(B->r(), A);
 if (A->p == ra->p) ra = base->r();
 if (B->p == rb->p) rb = base;
#define DEL(e, init, dir) Q e = init->dir; if
    (valid(e)) \
  while (circ(e->dir->F(), H(base), e->F())) { \
   Q t = e->dir; \
   splice(e, e->prev()); \
   splice(e->r(), e->r()->prev()); \
   e->o = H; H = e; e = t; \
  }
  for(;;){
    DEL(LC, base->r(), o) DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC),
         H(LC))))
     base = connect(RC, base->r());
    else
     base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}
vector<P> triangulate(vector<P> pts) {
 sort(all(pts)); assert(unique(all(pts)) ==
      pts.end());
 if (sz(pts) < 2) return {};
 Q e = rec(pts).first;
 vector<Q> q = {e};
 int qi = 0;
 while (e->o->F().cross(e->F(), e->p) < 0) e =
      e->o;

#define ADD { Q c = e; do { c->mark = 1;
     pts.push_back(c->p);\
 q.push_back(c->r()); c = c->next(); } while (c
     != e); }
 ADD; pts.clear();
 while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
 return pts;
}
```

3.11 HullDiameter

```
/**
 * Author: Oleksandr Bacherikov, chilli
 * Date: 2019-05-05
 * License: Boost Software License
 * Source: https://codeforces.com/blog/entry/48868
 * Description: Returns the two points with max
     distance on
      a convex hull (ccw,
 * no duplicate/collinear points).
 * Status: stress-tested, tested on
     kattis:roberthood
 */
#pragma once
#include "Point.h"
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
 int n = sz(S), j = n < 2 ? 0 : 1;
 pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
 rep(i,0,j)
  for (;; j = (j + 1) % n) {
   res = max(res, {(S[i] - S[j]).dist2(), {S[i],
        S[j]}});
   if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] -
        S[i]) >= 0)
   break; }
 return res.second;
}
```

3.12 InsidePolygon

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-04-26
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Returns true if p lies within the
     polygon.
      If strict is true,
 * it returns false for points on the boundary.
     The
      algorithm uses
 * products in intermediate steps so watch out
     for overflow.
 * Time: O(n)
 * Usage:
 * vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
 * bool in = inPolygon(v, P{3, 3}, false);
 * Status: stress-tested and tested on
     kattis:pointinpolygon
 */
#pragma once
#include "Point.h"
#include "OnSegment.h"
#include "SegmentDistance.h"
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict =
     true) {
 int cnt = 0, n = sz(p);
 rep(i,0,n) {
  P q = p[(i + 1) % n];
  if (onSegment(p[i], q, a)) return !strict;
  //or: if (segDist(p[i], q, a) <= eps) return
       !strict;
  cnt ^= ((a.y<p[i].y) - (a.y<q.y)) *
       a.cross(p[i], q) > 0;
 }
 return cnt; }
```

3.13 linearTransformation

```
/**
 Apply the linear transformation (translation,
     rotation and
      scaling) which takes line p0-p1 to line
          q0-q1 to point
      r.
*/
#pragma once
#include "Point.h"
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
  const P& q0, const P& q1, const P& r) {
 P dp = p1-p0, dq = q1-q0, num(dp.cross(dq),
      dp.dot(dq));
 return q0 + P((r-p0).cross(num),
      (r-p0).dot(num))/dp.dist2
();
}
```

3.14 lineDistance

```
/**
 * Returns the signed distance between point p
     and the line
     containing points a and b. Positive value on
         left side
     and negative on right as seen from a towards
         b. a==b
     gives nan. P is supposed to be Point<T> or
         Point3D<T>
     where T is e.g. double or long long. It uses
         products
     in intermediate steps so watch out for
         overflow if
     using int or long long. Using Point3D will
         always give
     a non-negative distance. For Point3D, call
         .dist on the
      result of the cross product.
 * Status: tested
*/
#pragma once
#include "Point.h"
template<class P>
double lineDist(const P& a, const P& b, const P&
    p) {
 return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

3.15 lineIntersection

```
/**
If a unique intersection point of the lines going
    through s1
     ,e1 and s2,e2 exists \{1, point\} is returned.
If no intersection point exists \{0, (0,0)\} is
    returned and
     if infinitely many exists \{-1, (0,0)\} is
         returned.
The wrong position will be returned if P is
    Point<ll> and
     the intersection point does not have integer
     coordinates.
Products of three coordinates are used in
    intermediate steps
      so watch out for overflow if using int or ll.
Usage:
 auto res = lineInter(s1,e1,s2,e2);
*
*
*  if (res.first == 1)
  cout << "intersection point at " << res.second
      << endl;
  Status: stress-tested, and tested through
      half-plane tests
**/


#pragma once
#include "Point.h"
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
 auto d = (e1 - s1).cross(e2 - s2);
 if (d == 0) // if parallel
  return {-(s1.cross(e1, s2) == 0), P(0, 0)};
 auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
 return {1, (s1 * p + e1 * q) / d};
}
```

3.16 LineProjectionReflection

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-10-29
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Projects point p onto line ab.
     Set refl=true
      to get reflection
 * of point p across line ab insted. The wrong
     point will be
      returned if P is
 * an integer point and the desired point doesnt
     have
      integer coordinates.
 * Products of three coordinates are used in
     intermediate
      steps so watch out
 * for overflow.
 * Status: stress-tested
*/
#pragma once
#include "Point.h"
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
P v = b - a;
 return p -
     v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

3.17 ManhattanMST

```
/**
 * Author: chilli, Takanori MAEHARA
 * Date: 2019-11-02
 * License: CC0
 * Source:
     https://github.com/spaghetti-source/algorithm/
      blob/master/geometry/rectilinear_mst.cc
 * Description: Given N points, returns up to 4*N
     edges,
      which are guaranteed
 * to contain a minimum spanning tree for the
     graph with
      edge weights w(p, q) =
 * |p.x - q.x| + |p.y - q.y|. Edges are in the
     form (
      distance, src, dst). Use a
 * standard MST algorithm on the result to find
     the final MST.
* Time: O(N \log N)
* Status: Stress-tested
*/
```

```
#pragma once
#include "Point.h"
typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
 vi id(sz(ps));
 iota(all(id), 0);
 vector<array<int, 3>> edges;
 rep(k,0,4) {
  sort(all(id), [&](int i, int j) {
      return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
  map<int, int> sweep;
  for (int i : id) {
   for (auto it = sweep.lower_bound(-ps[i].y);
           it != sweep.end(); sweep.erase(it++)) {
    int j = it->second;
    P d = ps[i] - ps[j];
    if (d.y > d.x) break;
    edges.push_back({d.y + d.x, i, j});
}
   sweep[-ps[i].y] = i;
  }
  for (P& p : ps) if (k & 1) p.x = -p.x; else
      swap(p.x, p.y)
      ;
}
 return edges;
}
```

3.18 MinimumEnclosingCircle

```
/**
 * Author: Andrew He, chilli
 * Date: 2019-05-07
 * License: CC0
 * Source: folklore
 * Description: Computes the minimum circle that
     encloses a
     set of points.
 * Time: expected O(n)
 * Status: stress-tested
 */
#pragma once
#include "circumcircle.h"
pair<P, double> mec(vector<P> ps) {
 shuffle(all(ps), mt19937(time(0)));
 P o = ps[0];
 double r = 0, EPS = 1 + 1e-8;
 rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r *
     EPS) {
  o = ps[i], r = 0;
  rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
   o = (ps[i] + ps[j]) / 2;
   r = (o - ps[i]).dist();
   rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
    o = ccCenter(ps[i], ps[j], ps[k]);
    r = (o - ps[i]).dist();
   }
} }
 return {o, r};
}
```

3.19 OnSegment

```
/**
 * Author: Victor Lecomte, chilli
 * Date: 2019-04-26
 * License: CC0
 * Source: https://vlecomte.github.io/cp-geo.pdf
 * Description: Returns true iff p lies on the
     line segment
     from s to e.
 * Use \texttt{(segDist(s,e,p)<=epsilon)} instead
     when using
      Point<double>.
 * Status:
*/
#pragma once
#include "Point.h"
template<class P> bool onSegment(P s, P e, P p) {
 return p.cross(s, e) == 0 && (s - p).dot(e - p)
     <= 0;
}
```

4 Graph-Theory

4.1 2SAT

```
// task : n people, each people have 2 request :
    + x or - x
// Ask : Is there a way build array m elements
    that for each people,
// at least one of two request is satisfied. If
    yes, print it.
#include <bits/stdc++.h>

using namespace std;

const int N = (int)1e5 + 9, N2 = N << 1;
int n, m;
vector<int> g[N2];     // [1,n]:   +    ;
    [1+n,n+n]:   -
int st[N2], top; // stack
int scc_cnt, scc_id[N2];
int tme, in[N2], low[N2]; // scc
bool was_tarjan[N2];    // check

void tarjan(int u) {
    st[++top] = u;
    in[u] = low[u] = ++tme;
    was_tarjan[u] = true;
    for (int v : g[u]) {
        if (!was_tarjan[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else {
            low[u] = min(low[u], in[v]);
        }
    }
    if (low[u] == in[u]) {
        ++scc_cnt;
        while (st[top] != u) {
            scc_id[st[top]] = scc_cnt;
            in[st[top]] = low[st[top]] = N;
            --top;
        }
        scc_id[st[top--]] = scc_cnt;
        in[u] = low[u] = N;
    }
}

vector<int> g2[N2];
void compress_scc_to_dag() {
    for (int u = 1; u <= m; ++u)
        for (int v : g[u])
            if (scc_id[v] != scc_id[u])
                g2[scc_id[u]].emplace_back(scc_id[v]);
}

bool was[N2];
int topo[N2];
void dfs(int u) {
    // toposort
    was[u] = true;
    for (int v : g2[u])
        if (!was[v])
            dfs(v);
    topo[u] = top--;
}
void toposort_g2(){
    top = scc_cnt;
    fill(was + 1, was + scc_cnt + 1, false);
    for (int i = 1; i <= scc_cnt; ++i)
        if (!was[i])
            dfs(i);
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cin >> m >> n;
    char uu, vv;
    for (int u, v; m-- > 0;)
    {
        cin >> uu >> u >> vv >> v;
        g[u + (uu == '+' ? n : 0)].emplace_back(v
            + (vv == '+' ? 0 : n));
        g[v + (vv == '+' ? n : 0)].emplace_back(u
            + (uu == '+' ? 0 : n));
    }
    m = n << 1;
    for (int i = 1; i <= m; ++i)
        if (!was_tarjan[i])
            tarjan(i);
    for (int i = 1; i <= n; ++i)
        if (scc_id[i] == scc_id[i +
                              n])
        {
            cout << "IMPOSSIBLE";
            return 0;
        }
    compress_scc_to_dag();
    toposort_g2();
    for (int i = 1; i <= n; ++i)
    {
        cout << (topo[scc_id[i]] > topo[scc_id[i +
            n]] ? '+' : '-') << " ";
    }
}
```

4.2 BlossomDarkMagic

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

// Cap ghep co trong so lon nhat - Thay Hoang
#define MCM MaxCostMatching
namespace MaxCostMatching {
#define dist(e) (lab[e.u] + lab[e.v] -
    g[e.u][e.v].w * 2)
    const int maxn = 1e3 + 5;
    const int oo = (int)1e9;
    struct Edge {
        int u, v, w;
    } g[maxn][maxn];
    int n, m, n_x;
    int lab[maxn], match[maxn], slack[maxn],
        st[maxn], pa[maxn];
    int flower_from[maxn][maxn], s[maxn],
        vis[maxn];
    vector<int> flower[maxn];
    deque<int> q;

    void init(int _n) {
        n = _n;
        for (int u = 1; u <= n; u++) {
            for (int v = 1; v <= n; v++) {
                g[u][v] = Edge{u, v, 0};
            }
        }
    }

    void add(int u, int v, int w) {
        g[u][v].w = max(g[u][v].w, w);
        g[v][u].w = max(g[v][u].w, w);
    }

    void update_slack(int u, int x) {
        if (!slack[x] || dist(g[u][x]) <
            dist(g[slack[x]][x]))
            slack[x] = u;
    }

    void set_slack(int x) {
        slack[x] = 0;
        for (int u = 1; u <= n; u++) {
            if (g[u][x].w > 0 && st[u] != x &&
                s[st[u]] == 0)
                update_slack(u, x);
        }
    }

    void q_push(int x) {
        if (x <= n)
            return q.push_back(x);
        for (int i = 0; i < flower[x].size(); i++)
            q_push(flower[x][i]);
    }

    void set_st(int x, int b) {
        st[x] = b;
        if (x <= n)
            return;
        for (int i = 0; i < flower[x].size(); i++)
            set_st(flower[x][i], b);
    }
```

```
int get_pr(int b, int xr) {
    int pr = find(flower[b].begin(),
         flower[b].end(), xr) -
         flower[b].begin();
    if (pr % 2 == 1) {
        reverse(flower[b].begin() + 1,
             flower[b].end());
        return (int)flower[b].size() - pr;
    }
    else {
        return pr;
    }
}

void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n)
        return;
    Edge e = g[u][v];
    int xr = flower_from[u][e.u], pr =
         get_pr(u, xr);
    for (int i = 0; i < pr; i++)
        set_match(flower[u][i],
                  flower[u][i ^ 1]);
    set_match(xr, v);
    rotate(flower[u].begin(),
         flower[u].begin() + pr,
           flower[u].end());
}

void augment(int u, int v) {
    int xnv = st[match[u]];
    set_match(u, v);
    if (!xnv)
        return;
    set_match(xnv, st[pa[xnv]]);
    augment(st[pa[xnv]], xnv);
}

int get_lca(int u, int v){
    static int t = 0;
    for (t++; u || v; swap(u, v)) {
        if (u == 0)
            continue;
        if (vis[u] == t)
            return u;
        vis[u] = t;
        u = st[match[u]];
        if (u)
            u = st[pa[u]];
    }
    return 0;
}

void add_blossom(int u, int lca, int v) {
    int b = n + 1;
    while (b <= n_x && st[b])
        b++;
    if (b > n_x)
        n_x++;
    lab[b] = 0, s[b] = 0;
    match[b] = match[lca];
    flower[b].clear();
    flower[b].push_back(lca);
    for (int x = u, y; x != lca; x =
         st[pa[y]]) {
        flower[b].push_back(x),
            flower[b].push_back(y =
            st[match[x]]), q_push(y);
    }
    reverse(flower[b].begin() + 1,
         flower[b].end());
    for (int x = v, y; x != lca; x =
         st[pa[y]]) {
        flower[b].push_back(x),
            flower[b].push_back(y =
            st[match[x]]), q_push(y);
    }
    set_st(b, b);
    for (int x = 1; x <= n_x; x++)
        g[b][x].w = g[x][b].w = 0;
    for (int x = 1; x <= n; x++)
        flower_from[b][x] = 0;
    for (int i = 0; i < flower[b].size(); i++)
         {
        int xs = flower[b][i];
        for (int x = 1; x <= n_x; x++) {
            if (g[b][x].w == 0 ||
                 dist(g[xs][x]) <
                 dist(g[b][x])) {
                g[b][x] = g[xs][x], g[x][b] =
                    g[x][xs];
            }
        }
        for (int x = 1; x <= n; x++) {
            if (flower_from[xs][x])
                flower_from[b][x] = xs;
        }
    }
    set_slack(b);
}

void expand_blossom(int b) {
    for (int i = 0; i < flower[b].size(); i++)
         {
        set_st(flower[b][i], flower[b][i]);
    }
    int xr = flower_from[b][g[b][pa[b]].u], pr
         = get_pr(b,xr);
    for (int i = 0; i < pr; i += 2) {
        int xs = flower[b][i], xns =
            flower[b][i + 1];
        pa[xs] = g[xns][xs].u;
        s[xs] = 1, s[xns] = 0;
        slack[xs] = 0, set_slack(xns);
        q_push(xns);
    }
    s[xr] = 1, pa[xr] = pa[b];
    for (int i = pr + 1; i < flower[b].size();
         i++) {
        int xs = flower[b][i];
        s[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
}

int on_found_Edge(const Edge &e) {
    int u = st[e.u], v = st[e.v];
    if (s[v] == -1) {
        pa[v] = e.u, s[v] = 1;
        int nu = st[match[v]];
        slack[v] = slack[nu] = 0;
        s[nu] = 0, q_push(nu);
    }
    else if (s[v] == 0) {
        int lca = get_lca(u, v);
        if (!lca)
            return augment(u, v), augment(v,
                u), 1;
        else
            add_blossom(u, lca, v);
    }
    return 0;
}

int matching() {
    fill(s, s + n_x + 1, -1), fill(slack,
        slack + n_x + 1, 0);
    q.clear();
    for (int x = 1; x <= n_x; x++) {
        if (st[x] == x && !match[x])
            pa[x] = 0, s[x] = 0, q_push(x);
    }
    if (q.empty())
        return 0;
    while (1) {
        while (q.size()) {
            int u = q.front();
            q.pop_front();
            if (s[st[u]] == 1)
                continue;
            for (int v = 1; v <= n; v++) {
                if (g[u][v].w > 0 && st[u] !=
                    st[v]) {
                    if (dist(g[u][v]) == 0) {
                        if
                            (on_found_Edge(g[u][v]))
                            return 1;
                    }
                    else
                        update_slack(u, st[v]);
                }
            }
        }
        int d = oo;
```

```
        for (int b = n + 1; b <= n_x; b++) {
            if (st[b] == b && s[b] == 1)
                d = min(d, lab[b] / 2);
        }
        for (int x = 1; x <= n_x; x++) {
            if (st[x] == x && slack[x]) {
                if (s[x] == -1)
                    d = min(d,
                        dist(g[slack[x]][x]));
                else if (s[x] == 0)
                    d = min(d,
                        dist(g[slack[x]][x]) /
                        2);
            }
        }
        for (int u = 1; u <= n; u++) {
            if (s[st[u]] == 0) {
                if (lab[u] <= d)
                    return 0;
                lab[u] -= d;
            }
            else if (s[st[u]] == 1)
                lab[u] += d;
        }
        for (int b = n + 1; b <= n_x; b++) {
            if (st[b] == b) {
                if (s[st[b]] == 0)
                    lab[b] += d * 2;
                else if (s[st[b]] == 1)
                    lab[b] -= d * 2;
            }
        }
        q.clear();
        for (int x = 1; x <= n_x; x++) {
            if (st[x] == x && slack[x] &&
                 st[slack[x]] != x &&
                 dist(g[slack[x]][x]) == 0) {
                if
                     (on_found_Edge(g[slack[x]][x]))
                    return 1;
            }
        }
        for (int b = n + 1; b <= n_x; b++) {
```

```
                if (st[b] == b && s[b] == 1 &&
                     lab[b] == 0)
                    expand_blossom(b);
            }
        }
        return 0;
    }

    int maxcost() {
        fill(match, match + n + 1, 0);
        n_x = n;
        int tot_weight = 0;
        int n_matches = 0;
        for (int u = 0; u <= n; u++)
            st[u] = u, flower[u].clear();
        int w_max = 0;
        for (int u = 1; u <= n; u++) {
            for (int v = 1; v <= n; v++) {
                flower_from[u][v] = (u == v ? u :
                     0);
                w_max = max(w_max, g[u][v].w);
            }
        }
        for (int u = 1; u <= n; u++)
            lab[u] = w_max;
        while (matching())
            n_matches++;
        for (int u = 1; u <= n; u++) {
            if (match[u] && match[u] < u) {
                tot_weight += g[u][match[u]].w;
            }
        }
        return tot_weight;
    }
}

int main()
{
    MCM::init(4);
    MCM::add(1, 2, 5);
    MCM::add(2, 3, 10);
    MCM::add(3, 4, 2);
    cout << MCM::maxcost() << "\n";
    return 0;
```

```
}
```

4.3 Dinic

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

//maxium flow
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v),
        u(u), cap(
cap)
 {}

};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;
    Dinic(int n, int s, int t) : n(n), s(s), t(t)
        {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }
    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }
    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow
                    < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }
    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int &cid = ptr[v]; cid <
             (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] ||
                 edges[id].cap - edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed,
                 edges[id].cap - edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }
    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s,
                 flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

4.4 RandomKuhn

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

//rng
unsigned seed =
     std::chrono::system_clock::now().time_since_epoch().
//blossom ko trong so
const int N = 1e4 + 5;
const int M = 1e5 + 5;
array<int, 3> edge[M];
vector<int> a[N];
bool f[N], ml[N];
int mr[N];
bool dfs(int u) {
    f[u] = true;
    for(auto i : a[u]) {
        if (f[mr[i]])
            continue;
        if (!mr[i] || dfs(mr[i]))
        {
            mr[i] = u;
            return true;
        }
    }
    return false;
}
int max_matching(int n)
{
    int cnt = 0;
    for(int i=1;i<=n;++i) shuffle(a[i].begin(),
         a[i].end(), default_random_engine(seed));
    for(int i=1;i<=n;++i)
    {
        for(auto j : a[i])
        {
            if (!mr[j])
            {
                mr[j] = i;
                ml[i] = true;
                ++cnt;
                break;
            }
        }
    }
    for (bool run = 1; run;)
    {
        for(int i=1;i<=n;++i) f[i] = false;
        run = 0;
        for(int i=1;i<=n;++i)
        {
            if (ml[i])
                continue;
            if (dfs(i))
                ml[i] = true, ++cnt, run = 1;
        }
    }
    return cnt;
}
```

4.5 SPFA

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

//min cost maximum flow
struct Edge {
    int from, to, capacity, cost;
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>
     &d, vector<int> &p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u]
                 + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, vector<Edge> edges, int
     K, int s, int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }

    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```

5 Mathematics

5.1 ExEuclidCRT

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

using namespace std;

int inv(int a, int m) {
        int m0 = m, t, q;
        int x0 = 0, x1 = 1;
        if (m == 1)
                return 0;

        // Apply extended Euclid Algorithm DARK
            MAGIC
        while (a > 1) {
                q = a / m;
                t = m;
                m = a % m, a = t;
                t = x0;
                x0 = x1 - q * x0;
                x1 = t;
        }

        if (x1 < 0)
                x1 += m0;

        return x1;
}

// k is size of num[] and rem[]. Returns the
    smallest
// number x such that:
// x % num[0] = rem[0],
// ..................
// x % num[k-2] = rem[k-1]
int findMinX(vector<int> num, vector<int> rem,
    int k) {
        int prod = 1;
        for (int i = 0; i < k; i++)
                prod *= num[i];

        int result = 0;

        for (int i = 0; i < k; i++) {
                int pp = prod / num[i];
                result += rem[i] * inv(pp, num[i])
                    * pp;
        }

        return result % prod;
}

// Driver method
int main(void)
{
    vector<int> num,rem;
    //input num && rem;
        int k = sizeof(num) / sizeof(num[0]);
        cout << "x is " << findMinX(num, rem, k);
        return 0;
}
```

5.2 FFT

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

using namespace std;
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> &a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j)
            swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <<= 1) {
        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 (cd &x : a)
            x /= n;
    }
}

vector<long long> multiply(vector<long long>
    const &a, vector<long long> const &b) {
    vector<cd> fa(a.begin(), a.end()),
        fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        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<long long> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

5.3 FWHT

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

//black magic
int fpow(int n, long long k, int p = (int)1e9 +
    7) {
    int r = 1;
    for (; k; k >>= 1) {
        if (k & 1)
            r = (long long)r * n % p;
        n = (long long)n * n % p;
    }
    return r;
}
```

```
/*
 * matrix:
 * +1 +1
 * +1 -1
 */
void XORFFT(int a[], int n, int p, int invert) {
    for (int i = 1; i < n; i <<= 1) {
        for (int j = 0; j < n; j += i << 1) {
            for (int k = 0; k < i; k++) {
                int u = a[j + k], v = a[i + j + k];
                a[j + k] = u + v;
                if (a[j + k] >= p)
                    a[j + k] -= p;
                a[i + j + k] = u - v;
                if (a[i + j + k] < 0)
                    a[i + j + k] += p;
            }
        }
    }
    if (invert) {
        long long inv = fpow(n, p - 2, p);
        for (int i = 0; i < n; i++)
            a[i] = a[i] * inv % p;
    }
}
/*
 * Matrix:
 * +1 +1
 * +1 +0
 */
void ORFFT(int a[], int n, int p, int invert) {
    for (int i = 1; i < n; i <<= 1) {
        for (int j = 0; j < n; j += i << 1) {
            for (int k = 0; k < i; k++) {
                int u = a[j + k], v = a[i + j + k];
                if (!invert) {
                    a[j + k] = u + v;
                    a[i + j + k] = u;
                    if (a[j + k] >= p)
                        a[j + k] -= p;
                }
                else {
                    a[j + k] = v;
                    a[i + j + k] = u - v;
                    if (a[i + j + k] < 0)
                        a[i + j + k] += p;
                }
            }
        }
    }
}
/*
 * matrix:
 * +0 +1
 * +1 +1
 */
void ANDFFT(int a[], int n, int p, int invert) {
    for (int i = 1; i < n; i <<= 1) {
        for (int j = 0; j < n; j += i << 1) {
            for (int k = 0; k < i; k++) {
                int u = a[j + k], v = a[i + j + k];
                if (!invert) {
                    a[j + k] = v;
                    a[i + j + k] = u + v;
                    if (a[i + j + k] >= p)
                        a[i + j + k] -= p;
                }
                else {
                    a[j + k] = v - u;
                    if (a[j + k] < 0)
                        a[j + k] += p;
                    a[i + j + k] = u;
                }
            }
        }
    }
}

const int maxn = 1e5 + 5;
int n, p;
int a[maxn];
int b[maxn];
int c[maxn];

void testXOR() {
    fill_n(a, maxn, 0);
    fill_n(b, maxn, 0);
    fill_n(c, maxn, 0);
    for (int i = 0; i < n; i++)
        a[i] = (long long)rand() * rand() % 100000;
    for (int i = 0; i < n; i++)
        b[i] = (long long)rand() * rand() % 100000;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            c[i ^ j] = (c[i ^ j] + (long long)a[i]
                * b[j]) % p;
        }
    }
    XORFFT(a, n, p, 0);
    XORFFT(b, n, p, 0);
    for (int i = 0; i < n; i++)
        a[i] = (long long)a[i] * b[i] % p;
    XORFFT(a, n, p, 1);
    for (int i = 0; i < n; i++) {
        cerr << a[i] << " " << c[i] << "\n";
        assert(a[i] == c[i]);
    }
}

void testOR() {
    fill_n(a, maxn, 0);
    fill_n(b, maxn, 0);
    fill_n(c, maxn, 0);
    for (int i = 0; i < n; i++)
        a[i] = (long long)rand() * rand() % 100000;
    for (int i = 0; i < n; i++)
        b[i] = (long long)rand() * rand() % 100000;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            c[i | j] = (c[i | j] + (long long)a[i]
                * b[j]) % p;
        }
    }
    ORFFT(a, n, p, 0);
    ORFFT(b, n, p, 0);
    for (int i = 0; i < n; i++)
        a[i] = (long long)a[i] * b[i] % p;
    ORFFT(a, n, p, 1);
    for (int i = 0; i < n; i++) {
        cerr << a[i] << " " << c[i] << "\n";
        assert(a[i] == c[i]);
    }
}

void testAND() {
    fill_n(a, maxn, 0);
    fill_n(b, maxn, 0);
    fill_n(c, maxn, 0);
    for (int i = 0; i < n; i++)
        a[i] = (long long)rand() * rand() % 100000;
    for (int i = 0; i < n; i++)
        b[i] = (long long)rand() * rand() % 100000;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            c[i & j] = (c[i & j] + (long long)a[i]
                * b[j]) % p;
        }
    }
    ANDFFT(a, n, p, 0);
    ANDFFT(b, n, p, 0);
    for (int i = 0; i < n; i++)
        a[i] = (long long)a[i] * b[i] % p;
    ANDFFT(a, n, p, 1);
    for (int i = 0; i < n; i++) {
        cerr << a[i] << " " << c[i] << "\n";
        assert(a[i] == c[i]);
    }
}
```

5.4 GaussianElimination

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

using namespace std;

const double EPS = 1e-9;
const int INF = 2; // it doesnt actually have to
    be infinity or a big number

int gauss(vector<vector<double>> a,
    vector<double> &ans) {
    int n = (int)a.size();
    int m = (int)a[0].size() - 1;
    vector<int> where(m, -1);
    for (int col = 0, row = 0; col < m && row <
        n; ++col) {
        int sel = row;
        for (int i = row; i < n; ++i)
            if (abs(a[i][col]) > abs(a[sel][col]))
                sel = i;

        if (abs(a[sel][col]) < EPS)continue;
        for (int i = col; i <= m; ++i)
            swap(a[sel][i], a[row][i]);
        where[col] = row;
        for (int i = 0; i < n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j = col; j <= m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign(m, 0);
    for (int i = 0; i < m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] /
                a[where[i]][i];
    for (int i = 0; i < n; ++i) {
        double sum = 0;
        for (int j = 0; j < m; ++j)
            sum += ans[j] * a[i][j];
        if (abs(sum - a[i][m]) > EPS)
            return 0;
    }
    for (int i = 0; i < m; ++i)
        if (where[i] == -1) return INF;
    return 1;
}
```

5.5 nCrAnymod

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

const int N = 1e6 + 9;
using ll = long long;

int power(long long n, long long k, const int
    mod) {
    int ans = 1 % mod;
    n %= mod;
    if (n < 0)
        n += mod;
    while (k) {
        if (k & 1)
            ans = (long long)ans * n % mod;
        n = (long long)n * n % mod;
        k >>= 1;
    }
    return ans;
}

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

ll inverse(ll a, ll m) {
    ll x, y;
    ll g = extended_euclid(a, m, x, y);
    if (g != 1) return -1;
    return (x % m + m) % m;
}

// returns n! % mod without taking all the
    multiple factors of p into account that
    appear in the factorial
// mod = multiple of p
// O(mod) * log(n)
int factmod(ll n, int p, const int mod) {
    vector<int> f(mod + 1);
    f[0] = 1 % mod;
    for (int i = 1; i <= mod; i++) {
        if (i % p)
            f[i] = 1LL * f[i - 1] * i % mod;
        else f[i] = f[i - 1];
    }
    int ans = 1 % mod;
    while (n > 1) {
        ans = 1LL * ans * f[n % mod] % mod;
        ans = 1LL * ans * power(f[mod], n / mod,
            mod) % mod;
        n /= p;
    }
    return ans;
}

ll multiplicity(ll n, int p) {
    ll ans = 0;
    while (n) {
        n /= p;
        ans += n;
    }
    return ans;
}

// C(n, r) modulo p^k
// O(p^k log n)
int ncr(ll n, ll r, int p, int k) {
    if (n < r or r < 0) return 0;
    int mod = 1;
    for (int i = 0; i < k; i++) {
        mod *= p;
    }
    ll t = multiplicity(n, p) - multiplicity(r,
        p) - multiplicity(n - r, p);
    if (t >= k) return 0;
    int ans = 1LL * factmod(n, p, mod) *
        inverse(factmod(r, p, mod), mod) % mod *
        inverse(factmod(n - r, p, mod), mod) %
        mod;
    ans = 1LL * ans * power(p, t, mod) % mod;
    return ans;
}
```

```

// finds x such that x % m1 = a1, x % m2 = a2. m1
    and m2 may not be coprime
// here, x is unique modulo m = lcm(m1, m2).
    returns (x, m). on failure, m = -1.
pair<ll, ll> CRT(ll a1, ll m1, ll a2, ll m2) {
    ll p, q;
    ll g = extended_euclid(m1, m2, p, q);
    if (a1 % g != a2 % g) return make_pair(0, -1);
    ll m = m1 / g * m2;
    p = (p % m + m) % m;
    q = (q % m + m) % m;
    return make_pair((p * a2 % m * (m1 / g) % m +
        q * a1 % m * (m2 / g) % m) % m, m);
}

int spf[N];
vector<int> primes;
void sieve() {
    for (int i = 2; i < N; i++) {
        if (spf[i] == 0) spf[i] = i,
            primes.push_back(i);
        int sz = primes.size();
        for (int j = 0; j < sz && i * primes[j] <
            N && primes[j] <= spf[i]; j++) {
            spf[i * primes[j]] = primes[j];
        }
    }
}

// O(m log(n) log(m))
int ncr(ll n, ll r, int m) {
    if (n < r or r < 0) return 0;
    pair<ll, ll> ans({0, 1});
    while (m > 1) {
        int p = spf[m], k = 0, cur = 1;
        while (m % p == 0) {
            m /= p;
            cur *= p;
            ++k;
        }
        ans = CRT(ans.first, ans.second, ncr(n, r,
            p, k), cur);
    }
```

```
    return ans.first;
}

int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    sieve();
    int t;
    cin >> t;
    while (t--) {
        ll n, k;
        cin >> n >> k;
        int m;
        cin >> m;
        ll r = (n + k - 1) / k;
        cout << r << " " << ncr((k - n % k) % k +
            r - 1, r - 1, m) << "\n";
    }
    return 0;
}
```

6 Miscellaneous

6.1 StressTest

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

const string NAME = "testing";
const int NTEST = 100;
mt19937
    rd(chrono::steady_clock::now().time_since_epoch().cou

long long Rand(long long l, long long h) {
    return l + rd() * 1LL * rd() % (h - l + 1);
}

int main() {
    srand(time(NULL));
    for (int iTest = 1; iTest <= NTEST; iTest++) {
        ofstream inp((NAME + ".inp").c_str());
        inp << Rand(1, 5) << ' ' << Rand(1, 5);
        inp.close();
        system((NAME + ".exe").c_str());
        system((NAME + "_trau.exe").c_str());
        if (system(("fc " + NAME + ".out " + NAME
            + ".ans").c_str()) != 0)
        {
            cout << "Test " << iTest << ":
                WRONG!\n";
            return 0;
        }
        cout << "Test " << iTest << ": CORRECT!\n";
    }
    return 0;
}
```

7 String-Algorithms

7.1 AhoCorasick

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

using namespace std;

//quickly search multiple patterns in a text
const int K = 26;
struct Vertex{
    int next[K];
    bool leaf = 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].leaf = true;
}

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

7.2 KMP

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

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;
}
```

7.3 RabinKarp

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

/*
Given two strings - s and t, determine if s
    appears in t
if it does, enumerate all its occurrences in
    O(|s| + |t|) time
*/

vector<int> rabin_karp(string const& s, string
    const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();

    vector<long long> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
        p_pow[i] = (p_pow[i-1] * p) % m;

    vector<long long> h(T + 1, 0);
    for (int i = 0; i < T; i++)
        h[i+1] = (h[i] + (t[i] - 'a' + 1) *
            p_pow[i]) % m;
    long long h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - ’a’ + 1) * p_pow[i])
           % m;

    vector<int> occurrences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (h[i+S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            occurrences.push_back(i);
    }
    return occurrences;
}
```

7.4 SuffixArray

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

using namespace std;
#define x first
#define y second

string s;
vector<int> p(400007), c(400007), lcp(400007);
int n, k;

void build(int n)
{
    vector<pair<int, int>> a(n);
    for (int i = 0; i < n; ++i)
        a[i] = {s[i], i};
    sort(a.begin(), a.end());
    for (int i = 0; i < n; ++i)
        p[i] = a[i].y;
    c[p[0]] = 0;
    for (int i = 1; i < n; ++i)
    {
        c[p[i]] = c[p[i - 1]];
        if (a[i].x != a[i - 1].x)
            c[p[i]] += 1;
    }
    k = 0;
    while ((1 << k) < n)
    {
        vector<pair<pair<int, int>, int>> a(n);
        for (int i = 0; i < n; ++i)
        {
            a[i] = {{c[i], c[(i + (1 << k)) % n]},
                i};
        }
        // Radix sort
        vector<int> cnt(n);
        for (auto i : a)
        {
            cnt[i.x.y]++;
        }
        vector<pair<pair<int, int>, int>> b(n);
        vector<int> pos(n);
        pos[0] = 0;
        for (int i = 1; i < n; ++i)
            pos[i] = pos[i - 1] + cnt[i - 1];
        for (auto i : a)
        {
            b[pos[i.x.y]] = i;
            pos[i.x.y]++;
        }
        a = b;
        ////////////////////////////////////
        vector<int> cnt2(n);
        for (auto i : a)
        {
            cnt2[i.x.x]++;
        }
        vector<pair<pair<int, int>, int>> f(n);
        vector<int> pos2(n);
        pos2[0] = 0;
        for (int i = 1; i < n; ++i)
            pos2[i] = pos2[i - 1] + cnt2
                                      [i - 1];
        for (auto i : a)
        {
            f[pos2[i.x.x]] = i;
            pos2[i.x.x]++;
        }
        a = f;
        //////////////
        for (int i = 0; i < n; ++i)
            p[i] = a[i].y;
        c[p[0]] = 0;
        for (int i = 1; i < n; ++i)
        {
            c[p[i]] = c[p[i - 1]];
            if (a[i].x != a[i - 1].x)
                c[p[i]]++;
        }
        k++;
    }
}
void buildlcp(int n)
{
    k = 0;
    for (int i = 0; i < n - 1; ++i)
    {
        k = max(0, k - 1);
        lcp[c[i]] = k;
        int s1 = i, s2 = p[c[i] - 1];
        for (int j = k; j <= n - i + 1; ++j)
        {
            if (s[s1 + j] == s[s2 + j])
            {
                k++;
                lcp[c[i]] = k;
            }
            else
                break;
        }
    }
}
```

7.5 Zfunction

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

vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) {
            z[i] = min(r - i, z[i - 1]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if(i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}
```