Team Notebook

THPT CHY

November 10, 2022

Contents

1 1. TroubleShoot

1.1 TroubleShoot

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.
Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it
    .
Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).
Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?
Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

2 2. In Contest

2.1 Batch Gentest

2.1.1 checker

```
ifstream finp("input.inp");
ifstream fout("my.out");
ifstream fans("correct.out");
```

2.1.2 run with checker

```
@echo off
set name=gen
call compile.bat
set name=correct
call compile.bat
set name=my
call compile.bat
set name=checker
call compile.bat

set /A cnt=0
:loop
set /A cnt+=1
echo Running on test %cnt% ...
gen.exe > input.inp
my.exe < input.inp > my.out
correct.exe < input.inp > correct.out
checker.exe
if errorlevel 1 (echo "WA") else goto loop
pause
```

2.1.3 run

```
@echo off
set name=gen
call compile.bat
set name=correct
call compile.bat
set name=my
call compile.bat
set /A cnt=0
:loop
set /A cnt+=1
echo Running on test %cnt% ...
gen.exe > input.inp
my.exe < input.inp > my.out
correct.exe < input.inp > correct.out
fc my.out correct.out > nul
if errorlevel 1 (echo "WA") else goto loop
pause
```

2.2 Pragma

```
#pragma GCC optimize("Ofast")
#pragma GCC optimize ("unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,
     avx,avx2,tune=native")
```

2.3 Random Gen

```
mt19937 RNG(chrono::high_resolution_clock::now().
     time_since_epoch().count());
int UID(int l, int r){
    uniform_int_distribution<int> random_number(l, r);
    return random_number(RNG);
}
```

2.4 Template

```
#include <bits/stdc++.h>
#define up(i,a,b) for (int i = (int)a; i <= (int)b; i++)
#define down(i,a,b) for (int i = (int)a; i >= (int)b; i--)
#define pii pair<int, int>
#define f first
#define s second
#define ep emplace_back
#define all(x) x.begin(), x.end()
using namespace std;

signed main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    #define Task "A"
    if (fopen(Task".inp", "r")){
        freopen(Task".inp", "r", stdin);
        freopen(Task".out", "w", stdout);
    }


}
```

3 3. Mathematics

3.1 CRT

```
struct General_CRT{
    long long addmod(long long a, long long b){
        a %= b;
        if (a < 0) a += b;
        return a;
    }

    struct extype{
        int x,y,d;
    };
    extype exeuclid(int a, int b){
        if (b == 0) return {1, 0, a};
        extype p = exeuclid(b, a%b);
        return {p.y, p.x - a/b*p.y, p.d};
    }

    struct CRTtype{
        bool isSol;
        long long x;
        long long lcm;
    };
    CRTtype CRT(int a, int b, int n, int m){
        extype p = exeuclid(n,m);
        int x = p.x;
        int D = p.d;
        int C = (b - a);
        if (C%D != 0) return {0, -1, -1};
        long long res = addmod(a + 1ll*C/D*x % (m/D) * n, 1ll
             *m/D*n);
        return {1, res, 1ll*m/D*n};
    }

    static const int maxn = 11;
    int a[maxn];
    int n[maxn];
    int E;
    void solve(){
        cin >> E;
        up(i,1,E) cin >> a[i] >> n[i];
        // a % x = n

        int res = a[1];
        int LCM = n[1];
        up(i,2,E){
            int A,B,N,M;
            A = res;
            B = a[i];
            N = LCM;
            M = n[i];
            auto p = CRT(A, B, N, M);
            if (p.isSol == 0){
                cout << "no solution \n";
                return;
            }
            res = p.x;
            LCM = p.lcm;
        }
        cout << res << " " << LCM << "\n";
    }
} //Find the smallest positive integer x such that
// a[1] % x = n[1]
// a[2] % x = n[2]
// ...
```

3.2 Euler Phi

```
struct Euler_Phi{
    static const int maxn = 1e6 + 10;
    int phi[maxn];
    int n;
    void make_phi(int n) {
        for (int i = 1; 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;
            }
        }
    }

    int eulerPhi(int n) {
        if (n == 0) return 0;
        int ans = n;
        for (int x = 2; x*x <= n; x++) {
            if (n % x == 0) {
                ans -= ans / x;
                while (n % x == 0) n /= x;
            }
        }
        if (n > 1) ans -= ans / n;
        return ans;
    }
} my_EP;
```

3.3 Matrix

```
const int maxn = 1e2 + 10;
const int MOD = 1e9 + 7;
int n,m,k;

struct Matrix{
    int c[maxn][maxn];

    void init(){
        memset(c, 0, sizeof(c));
    }

    Matrix operator * (const Matrix& A){
        Matrix res;
        up(i,1,n) up(j,1,n){
            res.c[i][j] = 0;
            up(k,1,n){
                res.c[i][j] = (1ll*c[i][k]*A.c[k][j] + res.c[i
                     ][j]) % MOD;
            }
        }
        return res;
    }

    Matrix Unit(){
        Matrix res;
        up(i,1,n) up(j,1,n){
            if (i == j) res.c[i][j] = 1;
            else res.c[i][j] = 0;
        }
        return res;
    }

    Matrix fpow(Matrix A, int n){
        Matrix res = Unit();
        for (; n; n >>= 1, A = A*A){
            if (n & 1) res = A*res;
        }
        return res;
    }
};

void solve(){
    Matrix A;
    A.init();
    //input for A.c[u][v];

    int k = 2; //number of exponentiation
    A = A.fpow(A, k);

    cout << A.c[1][1];
}
```

3.4 nCk

```
struct nCk{
    static const int maxc = 1e5 + 10;
    static const int MOD = 1e9 + 7;
    int f[maxc << 1]; //factorial
    int inv[maxc << 1]; //inverse
    int fi[maxc << 1]; //inverse factorial
    int n;

    void precompute(int n){
        f[0] = 1;
        up(i,1,n) f[i] = 1ll*f[i-1]*i % MOD;
        inv[1] = 1;
        fi[1] = 1;
        up(i,2,n){
            inv[i] = MOD - 1ll*MOD/i * inv[MOD % i] % MOD;
            fi[i] = 1ll*fi[i-1]*inv[i] % MOD;
        }
    }

    int C(int n, int k){
        if (n == 0 || k == 0) return 1;
        if (k == n) return 1;
        return 1ll*f[n] * fi[k] % MOD * fi[n-k] % MOD;
    }
} my_NCK;
```

3.5 Range Sieve

```
struct Range_Sieve{
    static const int maxn = 1e5 + 10;
    bitset<maxn> mark;
    vector<int> P;

    void range_sieve(long long l, long long r){
        mark.reset();

        for (long long i = 2; i*i <= r; i++){
            for (long long j = max(i*i, (l + i - 1)/i*i); j
                 <= r; j += i){
                mark[j - l] = 1;
            }
        }

        if (l == 1) mark[0] = 1;
        if (l == 0) mark[0] = mark[1] = 1;
        up(i,l,r) if (!mark[i - l]) P.ep(i);
        P.clear();
    }

    void solve(int l, int r){
        range_sieve(l, r);
        for (auto x : P) cout << x << " ";
    }
} RS;
```

4 4. Data Structures

4.1 Fake BIT

```
struct FakeUpdate_FakeGet{
    const int maxn = 1e5 + 10;
    int n;
    vector<int> a,b;
    vector<int> BIT[maxn];
    vector<int> node[maxn];

    void fakeupdate(int x, int yy){
        for (; x; x -= (x & (-x))){
            node[x].ep(yy);
        }
    }
    void fakeget(int x, int yy){
        for (; x <= n+1; x += (x & (-x))){
            node[x].ep(yy);
        }
    }
    void update(int x, int yy, int val){
        for (; x; x -= (x & (-x))){
            for (int y = LB(node[x], yy)+1; y; y -= (y & (-y)
                 )){
                BIT[x][y] = max(BIT[x][y], val);
            }
        }
    }
    int get(int x, int yy){
        int res = 0;
        for (; x <= n+1; x += (x & (-x))){
            for (int y = LB(node[x], yy)+1; y <= (int)node[x
                 ].size(); y += (y & (-y))){
                res = max(res, BIT[x][y]);
            }
        }
        return res;
    }

    void compress(vector<int>& a){
        vector<int> Val = a;
        sort(all(Val));
        purge(Val);
        for (auto& x : a){
            x = LB(Val, x) + 2;
        }
    }

    void input(){
        cin >> n;
        up(i,1,n){
            int x,y;
            cin >> x >> y;
            a.ep(x);
            b.ep(y);
        }
    }

    void init(){
        compress(a);
        compress(b);
        a.insert(a.begin(), -1);
        b.insert(b.begin(), -1);
        up(i,1,n){
            fakeupdate(a[i], b[i]);
            fakeget(a[i]+1, b[i]+1);
        }
        up(i,1,n+2){
            sort(all(node[i]));
            purge(node[i]);
            BIT[i].resize(node[i].size()+5);
        }
    }

    void solve(){
        input();
        init();
        ///do something
    }
} Fake_BIT;
```

4.2 Fenwick Tree

```
struct Fenwick_Tree{
    static const int maxn = 1e5 + 10;
    int F[maxn];
    int n;

    void update(int x, int val){
        for (; x <= n; x += (x & (-x))) F[x] += val;
    }

    int get(int x){
        int res = 0;
        for (; x; x -= (x & (-x))) res += F[x];
        return res;
    }

    int Lower_Prefix(int x){ // Find lowest i in prefix_sum
         such that prefix_sum[i] >= x
        int pos = 0; // Target pos
        int sum = 0; // Prefix sum
        for (int i = log2(n); i >= 0; i--){
            int bi = pos + (1 << i);
            if (bi <= n && sum + F[bi] < x){
                pos = bi; // liftup
                sum += F[bi];
            }
        }
        return pos+1;
    }
} BIT;
```

4.3 Heavy Light Decompose

```
struct Heavy_Light_Decomposition{
    const static int maxn = 1e5 + 10;
    vector<int> a[maxn];
    static int n,R;
//numnode, root of tree
static int W[maxn];
//weight of node
static int Tsize[maxn], heavy[maxn], par[maxn];
//size of tree, heavy node, parent of node

void DFS(int& u){
    Tsize[u] = 1;
    heavy[u] = -1;
    int id_max = -1;
    for (int i = 0; i < (int)a[u].size(); i++){
        int& v = a[u][i];
        if (v == par[u]) continue;
        par[v] = u;
        DFS(v);
        Tsize[u] += Tsize[v];
        if (id_max == -1 || Tsize[v] > Tsize[a[u][id_max
             ]]){
            id_max = i;
            heavy[u] = v;
        }
    }
}

static int in[maxn], out[maxn], node[maxn];
//intime and outime of substree
//node order in HLD
static int HChain[maxn], DChain[maxn];
//head of chain, depth of chain
static int thld;
void HLD(int& u){
    in[u] = ++thld;
    node[thld] = u;
    if (heavy[u] != -1) {
        int v = heavy[u];
        HChain[v] = HChain[u];
        DChain[v] = DChain[u];
        HLD(v);
    }
    for (int& v : a[u]){
        if (v != par[u] && v != heavy[u]){
            HChain[v] = v;
            DChain[v] = DChain[u] + 1;
            HLD(v);
        }
    }
    out[u] = thld;
}

static struct NODE{
    long long sum, lazysum;
    int len;
    void init(){
        sum = lazysum = 0ll;
        len = 1;
    }
} T[maxn << 2];

static struct Segment_Tree{
private:
    void push_up(int nod){}

    void push_down(int nod){}

    void Update(int nod, int l, int r, int u, int v, int
         val){}

    long long Get(int nod, int l, int r, int u, int v){
         return 0; }

public:
    void build(int nod, int l, int r){
        if (l == r){
            T[nod].init();
            T[nod].sum = 1ll*W[node[l]];
            return;
        }
        int mid = (l+r) >> 1;
        build(nod*2, l, mid);
        build(nod*2+1, mid+1, r);
        T[nod].sum = T[nod*2].sum + T[nod*2+1].sum;
        T[nod].len = T[nod*2].len + T[nod*2+1].len;
    }
    void Update(int l, int r, int val){
        Update(1, 1, n, l, r, val);
    }
    long long Get(int l, int r){
        return Get(1, 1, n, l, r);
    }
} Tree;

void real_Update(int x, int val){
    while (x != 0){
        Tree.Update(in[HChain[x]], in[x], val);
        x = par[HChain[x]];
    }
} //update for all parent of node x with val

long long real_Get(int x, int y){
    long long res = 0;
    while (HChain[x] != HChain[y]){
        if (DChain[x] < DChain[y]) swap(x, y);
        res += Tree.Get(in[HChain[x]], in[x]);
        x = par[HChain[x]];
    }
    if (in[x] > in[y]) swap(x, y);
    res += Tree.Get(in[x], in[y]);
    return res;
} //sum of all node from node x to y on tree

void Input(){
    cin >> n >> R;
    up(i,1,n) cin >> W[i];
    up(i,1,n-1){
        int u,v;
        cin >> u >> v;
        a[u].ep(v);
        a[v].ep(u);
    }
}

void Process(){
    DFS(R);
    HChain[R] = R;
    DChain[R] = 1;
    HLD(R);
    Tree.build(1, 1, n);
}

void Answer(){
    int Q;
    cin >> Q;
    char c;
    int x,y,d;
    while (Q--){
        if (c == 'U'){
            cin >> x >> d;
            real_Update(x, d); // update all parent of
                node x
        }
        else {
            cin >> x >> y;
            cout << real_Get(x, y) << "\n"; //sum of all
                node from x to y
```

```
            }
        }
    }
} HLD_TREE;
```

4.4 LCA

```
struct LCA{
    static const int maxn = 100001;
    static const int LIM = 1e9 + 7;
    int n;
    vector<pii> a[maxn];
    int minpar[maxn][18];
    int maxpar[maxn][18];
    int par[maxn][18];
    int h[maxn];

    void in(){
        int u,v,w;
        cin >> n;
        up(i,1,n-1){
            cin >> u >> v >> w;
            a[u].pb({v, w});
            a[v].pb({u, w});
        }
    }

    void DFS(int u, int parent){
        for (pii x : a[u]){
            int v = x.f;
            int w = x.s;
            if (v == parent) continue;
            h[v] = h[u] + 1;
            par[v][0] = u;
            minpar[v][0] = w;
            maxpar[v][0] = w;

            up(i,1,17){
                int p = par[v][i-1];
                par[v][i] = par[p][i-1];
                minpar[v][i] = min(minpar[v][i-1], minpar[p][i
                    -1]);
                maxpar[v][i] = max(maxpar[v][i-1], maxpar[p][i
                    -1]);
            }

            DFS(v, u);
        }
    }
```

```
    pair<int, int> MinMaxLCA(int u, int v){
        int Fmin = LIM;
        int Fmax = -LIM;
        if (h[u] < h[v]) swap(u, v);

        int de = h[u] - h[v];
        down(i, log2(de), 0){
            if (bit(de, i)){
                Fmin = min(Fmin, minpar[u][i]);
                Fmax = max(Fmax, maxpar[u][i]);
                u = par[u][i];
            }
        }

        if (u == v) return make_pair(Fmin, Fmax);

        down(i, 17, 0){
            if (par[u][i] != par[v][i]){
                Fmin = min(Fmin, min(minpar[u][i], minpar[v][i
                    ]));
                Fmax = max(Fmax, max(maxpar[u][i], maxpar[v][i
                    ]));
                u = par[u][i];
                v = par[v][i];
            }
        }

        Fmin = min(Fmin, min(minpar[u][0], minpar[v][0]));
        Fmax = max(Fmax, max(maxpar[u][0], maxpar[v][0]));
        return make_pair(Fmin, Fmax);
    } //Max and Min on path from u to v

    void query(){
        int u,v;
        cin >> u >> v;
        pair<int, int> res = MinMaxLCA(u, v);
    }



     //DP for least lexicography order
    int pmin[maxn][17];
    bool isbetter(int u, int v){
        int PminU = n+1;
        int PminV = n+1;
        down(i, 17, 0){
            if (par[u][i] != par[v][i]){
                PminU = min(PminU, pmin[u][i]);
            PminV = min(PminV, pmin[v][i]);
            u = par[u][i];
            v = par[v][i];
        }
    }
    PminU = min(PminU, pmin[u][0]);
    PminV = min(PminV, pmin[v][0]);
    return (PminU < PminV);
}

int tr[maxn];
void DP(int u){
    pmin[u][0] = u;
    par[u][0] = tr[u];
    up(i, 1, 17){
        int p = par[u][i-1];
        par[u][i] = par[p][i-1];
        pmin[u][i] = min(pmin[u][i-1], pmin[p][i-1]);
    }

    // Usage:
    // if (tr[u] == 0 || isbetter(v, tr[u])) tr[u] = v;
}
} my_LCA;
```

4.5 LIS trace

```
struct LIS{
    const int maxn = 1e5 + 10;
    int F[maxn];
    int B[maxn];
    int A[maxn];
    int n;
    int res = 0;

    void input(){
        cin >> n;
        up(i,1,n) cin >> a[i];
    }

    void DP(){
        up(i,1,n){
            F[i] = lower_bound(B+1, B+res+1, A[i]) - B;
            res = max(res, F[i]);
            B[F[i]] = A[i];
        }
        cout << res;
    }

    void trace(int len){
        vector<int> p;
        down(i,n,1){
            if (F[i] == len){
                p.push_back(A[i]);
                --len;
            }
        }
        reverse(p.begin(), p.end());
        for (auto x : p) cout << x << " ";
    }
} my_LIS;
```

4.6 Ordered Static Set

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#define fbo(x) find_by_order(x)
#define ook(x) order_of_key(x)
#define up(i,a,b) for (int i = (int)a; i <= (int)b; i++)
#define down(i,a,b) for (int i = (int)a; i >= (int)b; i--)
#define int long long
#define all(x) x.begin(), x.end()
using namespace std;
using namespace __gnu_pbds;

tree<
    int,
    null_type,
    less<int>,
    rb_tree_tag,
    tree_order_statistics_node_update
> OMset;

void Add(int x){
    OMset.insert(x);
}

void Remove(int x){
    int pos = OMset.ook(x);
    auto it = OMset.fbo(pos);
    if (*it != x) return;
    OMset.erase(it);
}

string Minimum(){
    return (to_string)(*OMset.begin());
}

string Maximum(){
    return (to_string)(*OMset.rbegin());
}

string Succ(int x){ //least and > x
    auto it = OMset.upper_bound(x);
    if (it == OMset.end()) return "no";
    return (to_string)(*it);
}

string Succ2(int x){ //least and >= x
    auto it = OMset.lower_bound(x);
    if (it == OMset.end()) return "no";
    return (to_string)(*it);
}

string Pred(int x){ //most and < x
    auto it = OMset.lower_bound(x);
    if (it == OMset.begin()) return "no";
    return (to_string)(*--it);
}

string Pred2(int x){ //most and <= x
    auto it = OMset.upper_bound(x);
    if (it == OMset.begin()) return "no";
    return (to_string)(*--it);
}
```

4.7 Sparse Table 2D

```
struct Sparse2D{
    static const int maxn = 500 + 1;
    static const int maxm = 8 + 1;
    int n,m;
    int sp[maxm][maxm][maxn][maxn];
    int logg[maxn];

    void input(){
        cin >> n >> m;
        up(i,1,n) up(j,1,m) cin >> sp[0][0][i][j];
        up(i,2,500) logg[i] = logg[(i >> 1)] + 1;
    }

    void build_sparse(){
        up(u,1,n) up(j,1,8){
            for (int v = 1; v + (1 << j) - 1 <= m; v++){
                sp[0][j][u][v] = min(sp[0][j-1][u][v], sp[0][j
                    -1][u][v + (1 << (j-1))]);
            }
        }

        up(i,1,8) up(j,0,8){
            for (int u = 1; u + (1 << i) - 1 <= n; u++){
                for (int v = 1; v + (1 << j) - 1 <= m; v++){
                    sp[i][j][u][v] = min(sp[i-1][j][u][v], sp[
                        i-1][j][u + (1 << (i-1))][v]);
                }
            }
        }
    }

    int getmin(int u, int v, int p, int q){ //upper left (u,
         v) and lower right (p, q)
        int k = logg[p-u+1];  //do dai hang
        int l = logg[q-v+1];  //do dai cot
        int r = p - (1 << k) + 1;
        int s = q - (1 << l) + 1;
        int m1 = sp[k][l][u][v];
        int m2 = sp[k][l][u][s];
        int m3 = sp[k][l][r][v];
        int m4 = sp[k][l][r][s];

        return min({m1, m2, m3, m4});
    }
} SP2D;
```

4.8 Union Find

```
struct Disjoint_Set_Union{
    static const int maxn = 2e5 + 10;
    int par[maxn];
    int n;

    int root(int x){
        if (par[x] < 0) return x;
        return par[x] = root(par[x]);
    }

    void Union(int u, int v){
        u = root(u);
        v = root(v);
        if (u == v) return;
        if (par[u] > par[v]) swap(u, v);
        par[u] += par[v];
        par[v] = u;
    }
    void init(){
        up(i,0,n) par[i] = -1;
    }

    bool Same(int x, int y){
        return root(x) == root(y);
    }

    bool can_destroy(int l, int r){
        bool ok = 1;
        int cur = l;
        int des = get(r+1);
        //r+1 is the index for the whole sector
        while (cur <= r){
            int u = root(cur);
            if (u != cur) ok = 0;
            par[cur] = des;
            cur = u+1;
        }
        return ok;
    } //destroy [l, r], and return 1 if [l, r] was not
       destroyed before
} DSU;
```

5 5. Graph

5.1 Center of Tree

```
struct Center_of_Tree{
    static const int maxn = 100001;
    int cost[maxn];
    int dx[maxn];
    int dy[maxn];
    vector<pii> a[maxn];
    int n;

    void in(){
        cin >> n;
        int u,v,w;
        up(i, 1, n-1){
            cin >> u >> v >> w;
            a[u].pb({v, w});
            a[v].pb({u, w});
        }
    }

    void DFS(int u, int par, int d[]){
        for (pii x : a[u]){
            int v = x.f;
            int w = x.s;
            if (v == par) continue;
            d[v] = d[u] + w;
            DFS(v, u, d);
        }
    }

    void solve(){
        in();
        DFS(1, -1, cost);
        int X = max_element(cost+1, cost+n+1) - cost;
        DFS(X, -1, dx);
        int Y = max_element(dx+1, dx+n+1) - dx;
        DFS(Y, -1, dy);

        int minn = MOD;
        up(i,1,n) minn = min(minn, max(dx[i], dy[i]));
        cout << minn;
    }

    //Center of Tree is the position such that
    //the Longest path from this node to all other nodes
    //is minimal
} my_center_of_tree;
```

5.2 Dijkstra Dense Graph

```
struct Dijkstra_Dense_Graph{
    static const int maxn = 1001;
    static const int LIM = 1e9 + 7;
    bitset<maxn> marked;
    int n;
    int st, fi;
    int dist[maxn];
    int C[maxn][maxn];

    void Forward(int u){
        marked[u] = true;
        up(v,1,n) dist[v] = min(dist[v], dist[u] + C[u][v]);
    }

    void Dense_Dijkstra(int root){
        memset(dist, 0x3f, (n+1)*sizeof(dist[0]));
        dist[root] = 0;
        up(loop, 1, n){
            int next = -1;
            up(v,1,n) if (!marked[v] && (next == -1 || dist[
                next] > dist[v])) next = v;
            if (next == -1) break;
            Forward(next);
        }
    }

    void solve(){
        cin >> n >> st >> fi;
        up(i,1,n) up(j,1,n) cin >> C[i][j];

        Dense_Dijkstra(st);
        cout << dist[fi];
    }
} GDENSE;
```

5.3 Dijkstra Kth Shortest Path

```
struct Kth_Shortest_Path{
    static const int maxn = 10001;
    vector<pii> a[maxn];
    int n,m;
    int k;
    priority_queue<pii, vector<pii>, greater<pii>> P;
    vector<vector<int>> d;

    void input(){
        int u,v,w;
        cin >> n >> m;
        cin >> k;
        up(i, 1, m){
            cin >> u >> v >> w;
            a[u].pb({v, w});
            a[v].pb({u, w});
        }
        d.resize(n+1, vector<int>());
    }

    void Dijkstra(){
        P.push({0, 1});
        while (!P.empty()){
            int u = P.top().s;
            int curw = P.top().f;
            P.pop();
            if ((d[u].size() != 0 && d[u].back() == curw) ||
                d[u].size() >= k) continue;
            d[u].ep(curw);

            for (pii x : a[u]){
                int v = x.f;
                int w = x.s;
                int val = curw + w;

                if (d[v].size() != 0 && d[v].back() == val ||
                    d[v].size() >= k) continue;
                P.push({val, v});
            }
        }
    }

    void solve(){
        in();
        Dijkstra();
        if (d[n].size() == 0) return cout << -1, 0;
        for (int x : d[n]) cout << x << " ";
        return 0;
    }
} KTH_SHORTEST;
// find the Kth Shortest path in a graph
```

5.4 DP on DAG DFS

```
struct DP_on_DAG{
    static const int maxn = 1e5 + 10;
    vector<pair<int, int>> a[maxn];
    int n,m;
    int f[maxn];

    int DFS(int u){
        if (f[u] != -1) return f[u];
        f[u] = 0;
        for (pii x : a[u]){
            int v = x.f;
            int w = x.s;
            f[u] = max(f[u], DFS(v) + w);
        }
        return f[u];
    }

    void input(){
        cin >> n >> m;
        int u,v,w;
        up(i,1,m){
            cin >> u >> v >> w;
            a[u].push_back({v, w});
        }
    }

    void solve(){
        memset(f, -1, sizeof(f));
        up(i,1,n) f[i] = DFS(i);
        cout << *max_element(f+1, f+n+1);
    }
} DP_DAG;
```

5.5 DP on DAG queue

```
struct DP_on_DAG_queue{
    static const int maxn = 100001;
    vector<pii> a[maxn];
    int n,m;
    queue<int> q;
    int degin[maxn];
    int f[maxn];

    void in(){
        cin >> n >> m;
        int u,v,w;
        up(i, 1, m){
            cin >> u >> v >> w;
            a[u].push_back({v, w});
            degin[v]++;
        }
    }

    void DP(){
        for (int i = 1; i <= n; i++){
            if (degin[i] == 0) q.push(i);
        }

        while (!q.empty()){
            int u = q.front();
            for (pii x : a[u]){
                int v = x.f;
                int w = x.s;
                if (f[v] < f[u] + w){
                    f[v] = f[u] + w;
                }
                if (--degin[v] == 0) q.push(v);
            }
            q.pop();
        }

        cout << *max_element(f+1, f+n+1);
    }
} DP_DAG_queue;
```

5.6 Find Joint and Bridge

```cpp
struct Bridge_Joint{
    static const int maxn = 100002;
    int n, m, par[maxn];
    vector<int> a[maxn];
    int num[maxn], low[maxn];
    vector<pii> Bridge;
    vector<int> Joint;
    int tdfs;

    void DFS(int u){
        int child = bool(par[u] != -1); // bool == 0 khi u la
            goc DFS
        low[u] = num[u] = ++tdfs;
        for (int v : a[u]){
            if (v == par[u]) continue;
            if (!num[v]){
                par[v] = u;
                DFS(v);
                low[u] = min(low[u], low[v]);
                if (low[v] == num[v]){
                    Bridge.push_back({min(u, v), max(u, v)});
                }
                if (low[v] >= num[u]) child++;
            }
            else low[u] = min(low[u], num[v]);
        }
        if (child > 1) Joint.emplace_back(u);
    }

    void input(){
        cin >> n >> m;
        up(i, 1, m){
            int u,v;
            cin >> u >> v;
            a[u].ep(v);
            a[v].ep(u);
        }
    }

    void Find_Bridge_Joint(){
        up(i,1,n){
            if (!par[i]){
                par[i] = -1;
                DFS(i);
            }
        }
    }
} F_BJ;
```

5.7 Find SCC and Compress to DAG

```cpp
struct Find_SCC{
    static const int maxn = 1e5 + 10;
    vector<int> a[maxn];
    vector<int> b[maxn];
    int n,m;
    int low[maxn], idx[maxn], deg[maxn];
    int tdfs, SCC;
    int inSCC[maxn];
    stack<int> st;
    bitset<maxn> exist;

    void Tarjan(int& u){
        low[u] = idx[u] = ++tdfs;
        st.push(u);
        for (int& v : a[u]){
            if (inSCC[v]) continue;
            if (!idx[v]){
                Tarjan(v);
                low[u] = min(low[u], low[v]);
            }
            else low[u] = min(low[u], idx[v]);
        }

        if (low[u] == idx[u]){
            SCC++;
            while (true){
                int v = st.top();
                st.pop();
                inSCC[v] = SCC;
                if (u == v) break;
            }
        }
    }

    void resetdata(){
        up(i,1,n){
            a[i].clear();
            b[i].clear();
        }
        memset(idx, 0, 4*(n+1));
        memset(inSCC, 0, 4*(n+1));
        memset(deg, 0, 4*(n+1));
        exist.reset();
        SCC = tdfs = 0;
    }

    void prepare(){
        cin >> n >> m;
        int u,v;
        up(i,1,m){
            cin >> u >> v;
            a[u].ep(v);
            exist[u] = exist[v] = 1;
        }

        up(i,1,n) if (!idx[i] && exist[i]) Tarjan(i);
    }

    void compress_to_DAG(){
        up(u,1,n) if (exist[u]){
            for (int& v : a[u]){
                int ui = inSCC[u];
                int vi = inSCC[v];
                if (ui != vi) b[ui].ep(vi);
            }
        }
    }

    void solve(){
        prepare();
        compress_to_DAG();
        resetdata();
    }
} FINDSCC;
```

5.8 Floyd

```cpp
struct Floyd_Warshall{
    static const int maxn = 101;
    vector<pii> a[maxn];
    int n,m;
    int tr[maxn][maxn];
    int cost[maxn][maxn];
    int s,f;

    void in(){
        cin >> n >> m >> s >> f;
        memset(cost, 60, sizeof(cost));
        int u,v,w;
        up(i,1,m){
            cin >> u >> v >> w;
            cost[u][v] = w;
            cost[v][u] = w;
        }
    }

    void Floyd(){
        up(i,1,n) up(j, 1, n){
            tr[i][j] = j;
            if (i == j) cost[i][j] = 0;
        }

        up(k,1,n) up(u, 1, n) up(v, 1, n){
            if (cost[u][v] > cost[u][k] + cost[k][v]){
                cost[u][v] = cost[u][k] + cost[k][v];
                tr[u][v] = tr[u][k];
            }
        }
    }

    void Fpath(){
        vector<int> path;
        while (s != f){
            path.ep(s);
            s = tr[s][f];
        }
        path.ep(f);
        cout << path.size() << "\n";
        for (auto x : path) cout << x << " ";
    }
} FLOYD;
```

5.9 Ford Bellman

```
struct Ford_Bellman{
    static const int maxn = 100001;
    vector<pii> a[maxn];
    int n,m,s,f;
    int d[maxn];
    int tr[maxn];
    queue<pii> q;

    void in(){
        cin >> n >> m >> s >> f;
        int u,v,w;
        up(i, 1, m){
            cin >> u >> v >> w;
            a[u].pb({v, w});
            a[v].pb({u, w});
        }
    }

    void FordBell(){
        fill(d, d+n+1, 1000001);
        d[s] = 0;
        q.push({s, d[s]});
        while (!q.empty()){
            pii x = q.front();
            q.pop();
            int u = x.first;
            int curw = x.second;
            for (pii ve : a[u]){
                int v = ve.first;
                int w = ve.second;
                if (d[v] > curw + w){
                    d[v] = curw + w;
                    tr[v] = u;
                    q.push({v, d[v]});
                }
            }
        }
    }

    void out(){
        cout << d[f];
        cout << "\n";
        vector<int> path;
        while (s != f){
            path.ep(f);
            f = tr[f];
        }
        path.ep(s);
        cout << path.size() << "\n";
        reverse(path.begin(), path.end());
        for (auto x : path) cout << x << " ";
    }
} FBELL;
```

5.10 Longest Path on Tree

```
struct Longest_Path_on_Tree{
    static const int maxn = 1e5 + 10;
    int d[maxn];
    vector<pii> a[maxn];
    int n;

    void in(){
        cin >> n;
        int u,v,w;
        up(i, 1, n-1){
            cin >> u >> v >> w;
            a[u].pb({v, w});
            a[v].pb({u, w});
        }
    }

    void DFS(int u, int par){
        for (pii x : a[u]){
            int v = x.f;
            int w = x.s;
            if (v == par) continue;
            d[v] = d[u] + w;
            DFS(v, u);
        }
    }

    void solve(){
        in();
        DFS(1, -1);
        int x = max_element(d+1, d+n+1) - d;
        memset(d, 0, sizeof(d));
        DFS(x, -1);
        cout << *max_element(d+1, d+n+1);
    }
} my_LPOT;
```

5.11 Matching

```
mt19937 RNG(chrono::steady_clock::now().time_since_epoch().
    count());
struct Bipartite_Matching{
    static const int maxn = 100001;
    vector<int> a[maxn];
    int n,m;
    int matX[maxn];
    int matY[maxn];
    int dd[maxn];

    void input(){
        cin >> m >> n;
        int u,v;
        while (cin >> u >> v) a[u].ep(v);
    }

    bool Kuhn(int u, int cur){
        if (dd[u] == cur) return 0;
        dd[u] = cur;
        for (int v : a[u]){
            if (matX[v] == 0 || Kuhn(matX[v], cur)){
                matX[v] = u;
                matY[u] = v;
                return 1;
            }
        }
        return 0;
    }

    int id[maxn];
    void solve(){
        input();
        int res = 0;
        for (int i = 1; i <= m; i++) id[i] = i;
        shuffle(id+1, id+m+1, RNG);
        for (int i = 1; i <= m; i++){
            if (matY[id[i]] == 0){
                res += Kuhn(id[i], id[i]);
            }
        }
        cout << res << "\n";
        for (int i = 1; i <= n; i++) cout << matX[i] << " ";
    }
} MATCHING;
```

5.12 Maxflow - BFS

```
struct EdmondsKarp{
    static const int maxn = 1001;
    int c[maxn][maxn];
    int f[maxn][maxn];
    vector<int> a[maxn];
    int tr[maxn];
    int n,m,s,t;
    int maxflow;
    queue<int> Q;

    void in(){
        cin >> n >> m >> s >> t;
        int u,v,w;
        up(i,1,m){
            cin >> u >> v >> w;
            a[u].push_back(v);
            a[v].push_back(u);
            c[u][v] = w;
        }
    }

    bool argument_path_found(){
        Q = queue<int>{};
        memset(tr, 0, sizeof(tr));

        tr[s] = -1;
        Q.push(s);
        while (!Q.empty()){
            int u = Q.front();
            Q.pop();
            if (u == t) return true;
            for (int v : a[u]){
                if (tr[v] == 0 && f[u][v] < c[u][v]){
                    tr[v] = u;
                    Q.push(v);
                }
            }
        }
        return false;
    }

    void IncFlow(){
        int delta = 1000000007;
        int v = t;
        while (v != s){
            int u = tr[v];
            delta = min(delta, c[u][v] - f[u][v]);
            v = u;
        }

        v = t;
        while (v != s){
            int u = tr[v];
            f[u][v] += delta;
            f[v][u] -= delta;
            v = u;
        }
        maxflow += delta;
    }

    void solve(){
        in();
        while (argument_path_found()){
            IncFlow();
        }
        cout << maxflow;
    }
} BFS_FLOW;
```

5.13 Maxflow - Dinitz

```
const int maxn = 1e3 + 10;
const int maxm = 1e5 + 10;
const int LIM = 1e18 + 10;

struct Edge{
    int u, v;
    int cap;
    int flow = 0;
    Edge(int _u, int _v, int _cap){
        u = _u;
        v = _v;
        cap = _cap;
    }
};

struct Dinitz{
    vector<Edge> E;
    int n,m;
    int s,t;
    vector<int> a[maxn];
    int level[maxn];
    int cur[maxn];
    queue<int> Q;

    Dinitz(int _n, int _s, int _t){
        n = _n;
        s = _s;
        t = _t;
        m = 0;
    }

    void add_edge(int From, int To, int Cap){
        E.push_back(Edge(From, To, Cap));
        E.push_back(Edge(To, From, 0));
        a[From].push_back(m);
        a[To].push_back(m+1);
        m += 2;
    }

    bool BFS(){
        memset(level, -1, sizeof(level[0])*(n+1));
        Q.push(s);
        level[s] = 0;
        while (!Q.empty()){
            int u = Q.front();
            Q.pop();
            for (int id : a[u]){
                int v = E[id].v;
                if (level[v] != -1 || E[id].flow == E[id].cap)
                    continue;
                level[v] = level[u] + 1;
                Q.push(v);
            }
        }
        return (level[t] != -1);
    }
    int DFS(int u, int delta){
        if (u == t) return delta;
        if (delta == 0) return 0;
        for (int& x = cur[u]; x < (int)a[u].size(); x++){
            int id = a[u][x];
            int v = E[id].v;
            if (level[v] != level[u] + 1 || E[id].flow == E[
                id].cap) continue;

            int cur_flow = min(delta, E[id].cap - E[id].flow)
                ;
            int neck = DFS(v, cur_flow);

            if (neck == 0) continue;
            E[id].flow += neck;
            E[id ^ 1].flow -= neck;
            return neck;
        }
        level[u] = LIM;
        return 0;
    }

    int max_flow(){
        int res = 0;
        while (BFS()){
            memset(cur, 0, sizeof(cur[0])*(n+1));
            while (int delta = DFS(s, LIM)){
                res += delta;
            }
        }
        return res;
    }
};

void solve(){
    int n,m,s,t;
    cin >> n >> m >> s >> t;
    Dinitz D(n, s, t);
    int u,v,w;
    while (cin >> u >> v >> w){
        D.add_edge(u, v, w);
    }
    cout << D.max_flow();
}
```

5.14 MST Prim N2

```
struct Prim_ON2{
    static const int maxn = 5050;
    const double LIM = 2e18;

    bool is0[maxn][maxn];
    bool marked[maxn];
    double dist[maxn];
    double sum_MST = 0;
    int connect[maxn];
    int n,m;

    double sqr(int x){
        return x*x;
    }

    double W(int u, int v){return 0;} //weight between u and
         v

    void Forward(int u){
        marked[u] = true;
        up(v,1,n){
            if (v == u) continue;
            double w = W(u, v);
            if (!marked[v] && dist[v] > w) {
                dist[v] = w;
                connect[v] = u;
            }
        }
    }

    void Prim(int root){
        up(i,0,n+1) dist[i] = LIM;
        dist[root] = 0;

        up(loop,1,n){
            int next = -1;
            up(v,1,n){
                if (!marked[v] && (next == -1 || dist[next] >
                    dist[v])) next = v;
            }
            if (next == -1) break;
            sum_MST += dist[next];
            Forward(next);
        }
    }

    void solve(){
        ///input
        Prim(1);
        cout << sum_MST;
    }
} PRIM;
```

5.15 MST Prim NLogM

```
struct Prim_ONlogN{
    static const int maxn = 3e4+10;
    static const int LIM = 1e9+7e8;
    int n,m;
    vector<pii> a[maxn];
    int dist[maxn];
    int connect[maxn];
    long long ans = 0;
    ///distance

    priority_queue<pii, vector<pii>, greater<pii> > Q;

    void Prim(int root){
        memset(dist, 0x3f, (n+1)*sizeof(dist[0]));
        dist[root] = 0;
        Q.push(make_pair(0, root));

        while (!Q.empty()){
            int curw = Q.top().f;
            int u = Q.top().s;
            Q.pop();
            if (curw > dist[u]) continue;

            ans += dist[u];
            dist[u] = -LIM;

            for (pii x : a[u]){
                int w = x.s;
                int v = x.f;
                if (dist[v] > w){
                    dist[v] = w;
                    connect[v] = u;
                    Q.push(make_pair(dist[v], v));
                }
            }
        }
    }

    void solve(){
        cin >> n >> m;
        int u,v,w;
        up(i, 1, m){
            cin >> u >> v >> w;
            a[u].pb({v, w});
            a[v].pb({u, w});
        }
        Prim(1);
        cout << ans << "\n";
    }
} PRIM_NLOGN;
```

6 6. String

6.1 Hash 2D - annotshy

```
// annotshy
/*
  Problem: CIPHER - Dai Ca Di Hoc
*/

#include <bits/stdc++.h>
#define MOD 1000000007
#define ll long long
#define Task "CIPHER"
using namespace std;

const int base = 100003;
const long long MM = 1ll * MOD * MOD;
const int maxn = 1005;

int n,m,A,B;
string a[maxn];
long long H[maxn][maxn];

const int p = 307;
const int q = 37;
long long P[maxn],Q[maxn];

void Prepare_Hash_2D(){
    //Prepare power:
    P[0] = 1;
    Q[0] = 1;
    for(int i = 1;i <= m;++i) P[i] = P[i - 1] * p % MOD;
    for(int i = 1;i <= n;++i) Q[i] = Q[i - 1] * q % MOD;
    //Prepare Hashing (1 1) -> (i j)
    for(int i = 1;i <= n;++i)
        for(int j = 1;j <= m;++j) {
            H[i][j] = (H[i][j - 1] * p % MOD + H[i - 1][j] *
                 q % MOD) % MOD;
            H[i][j] = (H[i][j] - H[i - 1][j - 1] * p * q + MM
                  + a[i][j]) % MOD;
        }
}

long long Get_Hash_2D(int a,int b,int x,int y) {
    long long Hash = (H[x][y] - H[a - 1][y] * Q[x - a + 1] %
         MOD - H[x][b - 1] * P[y - b + 1] + MM) % MOD;
    Hash = (Hash + H[a - 1][b - 1] * Q[x - a + 1] % MOD * P[y
         - b + 1] % MOD) % MOD;
    return Hash;
}

vector<pair<int,int> > ans;
map<long long,int> M;

void Solve(){
    int mx = 0;
    long long Max_Hash;
    for(int i = A;i <= n;++i)
        for(int j = B;j <= m;++j) {
            long long Hash = Get_Hash_2D(i - A + 1,j - B + 1,
                 i,j);
            if(++M[Hash] > mx) {
                mx = M[Hash];
                Max_Hash = Hash;
            }
        }
    for(int i = A;i <= n;++i)
        for(int j = B;j <= m;++j) {
            long long Hash = Get_Hash_2D(i - A + 1,j - B + 1,
                 i,j);
            if(Hash == Max_Hash) ans.pb({i - A + 1,j - B +
                 1});
        }
    int L = ans[0].F,R = ans[0].S;
    cout << A << " " << B << "\n";
    for(int i = L;i <= L + A - 1;++i){
        for(int j = R;j <= R + B - 1;++j){
            cout << a[i][j];
        }
        cout << "\n";
    }
    cout << ans.size() << "\n";
    for(auto x : ans) cout << x.F << " " << x.S << "\n";
}

int main(){
    ios_base::sync_with_stdio(0);
    cout.tie(0); cin.tie(0);
 if(fopen(Task".inp","r")){
  freopen(Task".inp","r",stdin);
  freopen(Task".out","w",stdout);
 }
    cin >> n >> m;
    string nothing; getline(cin,nothing);
    for(int i = 1;i <= n;++i){
        getline(cin,a[i]);
        a[i] = " " + a[i];
    }
    cin >> A >> B;
    Prepare_Hash_2D();
    Solve();
}

// V T AN
```

6.2 Hash

```
struct HashString{
    static const int maxn = 1e5 + 10;
    const int base = 311;
    const int MOD = 1e9 + 7;
    const long long MM = 1ll*MOD*MOD;

    int n;
    long long a[maxn];
    long long hashx[maxn], hashn[maxn];
    long long D[maxn]; // Decryptor

    void create(){
        D[0] = 1;
        up(i,1,n) D[i] = (D[i-1]*base) % MOD;
        up(i,1,n) hashx[i] = (hashx[i-1]*base + a[i]) % MOD;
        down(i,n,1) hashn[i] = (hashn[i+1]*base + a[i]) % MOD
             ;
    }

    long long getx(int u, int v){
        return (hashx[v] - hashx[u-1]*D[v-u+1] + MM) % MOD;
    }

    long long getn(int u, int v){
        return (hashn[u] - hashn[v+1]*D[v-u+1] + MM) % MOD;
    }
} HASH;
```

6.3 KMP Function

```
struct Knutt_Moris_Pratt{
    static const int maxn = 100001;
    int Pi[maxn];

    void make_KMP(string G){
        int n = G.size();
        int k = Pi[1] = 0;
        up(i,2,n){
            while (k && G[k+1] != G[i]) k = Pi[k];
            if (G[k+1] == G[i]) k++;
            Pi[i] = k;
        }
    }
    //Pi[x] computes the length of the longest prefix of s
         that ends at x

    int res[maxn];
    void count_occurence_prefix(string s){
        int n = s.size();
        make_KMP(s);
        fill(res+1, res+n+1, 1);
        down(i,n,1) res[Pi[i]] += res[i];
        up(i,1,n) cout << res[i] << " ";
    }

    void Find_Pattern(string s, string t){
        int n = s.size();
        int m = t.size();

        string g = '@' + t + '@' + s;
        make_KMP(g);

        up(i,1,n) if (Pi[i] == m) cout << i - 2*m << " ";
    }
} KMP;
```

6.4 Manacher

```
struct MANACHER{
    static const int maxn = 1e6 + 10;
    string s;
    int d1[maxn];
    int d2[maxn];
    int pos = -1;
    int maxx = -1;
    int n;

    void Manachers_Odd(){
        for (int l = 1, r = 0, i = 1; i <= n; i++){
            int k = (i > r) ? 1 : min(d1[l+r-i], r - i + 1);
            while (i-k >= 1 && i+k <= n && s[i-k] == s[i+k])
                 k++;

            d1[i] = k--;
            if (maxx < d1[i]*2-1){
                maxx = d1[i]*2-1;
                pos = i - d1[i] + 1;
            }
            if (i + k > r){
                r = i + k;
                l = i - k;
            }
        }
    }

    void Manachers_Even(){
        for (int l = 1, r = 0, i = 2; i <= n; i++){
            int k = (i > r) ? 0 : min(d2[l+r-i], r - i + 1);
            while (i-k-1 >= 1 && i+k <= n && s[i-k-1] == s[i+
                 k]) k++;

            d2[i] = k--;
            if (maxx < d2[i]*2){
                maxx = d2[i]*2;
                pos = i - d2[i];
            }
            if (i + k > r){
                r = i + k;
                l = i - k;
            }
        }
    }

    void solve(){
        cin >> n;
        cin >> s;

        s = '@' + s;
        Manachers_Odd();
        Manachers_Even();
        cout << maxx;
    }
} my_manacher;
```

6.5 String To Int

```
string int_to_string(int x){
    ostringstream ss;
    ss << x;// << x + 1 << x+2;
    return ss.str();
}

int string_to_int(string s){
    istringstream si(s);
    int x;
    si >> x;
    return x;
}
```

6.6 Suffix Array

```
struct Suffix_Array{
    static const int maxn = 4e5 + 10;
    static const int LOG = log2(maxn)+2;
    int n;

    pair<pii, int> L[maxn]; //Lexicography order: of string
         at position have rank i -> start at position L[i].
         second
    int R[LOG][maxn]; //Ranking of string start at position i
         , have length of 2^log -> have rank R[log][i]
    int SA[maxn];

    void build_SA(const string& s){
        up(i,1,n) R[0][i] = s[i];

        for (int POW = 1; (1 << (POW-1)) <= n; POW++){
            up(i,1,n){
                L[i].f.f = R[POW-1][i];
                int k = i + (1 << (POW-1));
                if (k <= n) L[i].f.s = R[POW-1][k];
                else L[i].f.s = -1;
                L[i].s = i;
            }
            sort(L+1, L+n+1);

            int cnt = 0;
            up(i,1,n){
                if (L[i].f == L[i-1].f) R[POW][L[i].s] = R[POW
                    ][L[i-1].s];
                else R[POW][L[i].s] = ++cnt;
            }
        }

        up(i,1,n) SA[i] = L[i].s;
    }

    int LCP[maxn];
```

```cpp
int pos[maxn];
void Kasai(const int SA[], const string& s){
    up(i,1,n) pos[SA[i]] = i;

    int k = 0;
    up(i,1,n){
        int& cur = pos[i];
        int j = SA[cur-1];
        //if suffix pair at position (i, j), have rank (
             cur, cur-1) and LCP equal to k
        //then suffix pair at position (i+1, j'), have
             rank(cur', cur'-1) should have LCP at least k
             -1, and at most...
        //find at at most on this line :)
        while (i+k <= n && j+k <= n && s[i+k] == s[j+k])
             ++k;
        LCP[cur] = k;
        if (k) --k;
    }
}

void make_SA_and_LCP(){
    string s;
    cin >> s;
    n = s.size();
    build_SA(s);
    Kasai(SA, s);
}




//Other applications
int lexi_compare(int i, int j, int l){
    int k = log2(l);
    pair<int, int> a = make_pair(R[k][i], R[k][i+l - (1
         << k)]);
    pair<int, int> b = make_pair(R[k][j], R[k][j+l - (1
         << k)]);
    return (a == b ? 0 : a < b ? -1 : 1);
}

int LCP_2suff(int i, int j){
    int ans = 0;
    for (int k = log2(n); k >= 0; k--) {
        if (R[k][i] == R[k][j]) {
            ans += 1 << k;
            i += 1 << k;
            j += 1 << k;
        }
    }
    return ans;
}

long long Number_of_Distinct_Substrings(){
    long long x = 0ll;
    up(i,1,n) x += 1ll*LCP[i];
    return 1ll*n*(n+1)/2 - x;
}

string Longest_Common_Consecutive_Substring(string X,
     string Y){
    string s = X + '@' + Y + "$";
    int n = s.size();

    int pivot = X.size()+1;
    build_SA(s);
    Kasai(SA, s);

    int maxx = 0;
    int id = 0;
    up(i,1,n){
        bool left = SA[i-1] < pivot;
        bool right = SA[i] >= pivot;
        if (left == right) {
            if (maxx < LCP[i]){
                maxx = LCP[i];
                id = i;
            }
        }
    }

    if (id == 0) return "not found";
    return s.substr(SA[id]-1, maxx);
}




//count pattern inside a string
int check(int pos, string g, const string& s){
    int f = -1;
    int k = g.size();
    int j = SA[pos];
    int suffix_len = n-j+1;

    if (suffix_len >= k) f = 0;
    g.insert(g.begin(), char(254));
    up(i,1,min(k, suffix_len)){
        if (s[j+i-1] < g[i]) return -1;
        if (s[j+i-1] > g[i]) return 1;
    }
    return f;
} //suffix SA[pos]'th compared to g?

int Left_Most(string g, const string& s){
    int l = 0;
    int r = n;
    int flag = 0;
    while (r - l > 1){
        int mid = l + (r-l)/2;
        flag = check(mid, g, s);
        if (flag >= 0) r = mid;
        else l = mid;
    }
    if (check(r, g, s) != 0) return -1;
    return r;
}

int Right_Most(string g, const string& s){
    int l = 1;
    int r = n+1;
    int flag = 0;
    while (r - l > 1){
        int mid = l + (r-l)/2;
        flag = check(mid, g, s);
        if (flag <= 0) l = mid;
        else r = mid;
    }
    if (check(l, g, s) != 0) return -1;
    return l;
}

int Count_Pattern_Inside_a_string(string s){
    n = s.size();
    s = '@' + s;
    string g;
    cin >> g;
    int Left = Left_Most(g, s);
    int Right = Right_Most(g, s);
    int res = -1;
    if (Left == -1 || Right == -1) res = 0;
```

```
        else if (Left != -1 && Right != -1) res = Right -
            Left + 1;
        return res;
    }
} my_suffix;
```

6.7 Trie

```
struct Trie{
    static const int maxn = 1000001;

    struct nod{
        int child[27];
        bool IE;
    };

    struct nod T[maxn];
    int n,m,depth;

    void Add(string s){
        int x = 0; // root;
        for (char c : s){
            int id = c - 'a' + 1;
            if (T[x].child[id] == 0) T[x].child[id] = ++depth
                ;
            x = T[x].child[id];
        }
        T[x].IE = 1;
    }

    bool Find(string s){
        int x = 0;
        for (char c : s){
            int id = c - 'a' + 1;
            if (T[x].child[id] == 0) return 0;
            x = T[x].child[id];
        }
        return T[x].IE;
    }
} TRIE;
```

6.8 Z Function

```
struct Z_Function{
    static const int maxn = 3e5 + 10;
    int n;
    int a[maxn], z[maxn];

    void TinhZ(){
        z[1] = 0;
        int l = 0;
        int r = 0;
        up(i,2,n){
            int x = 0;
            if (i <= r) x = min(z[i-l+1], r-i+1);
            while (i + x <= n && a[x+1] == a[i+x]) x++;

            z[i] = x;
            if (i + x - 1 > r){
                l = i;
                r = i + x - 1;
            }
        }
    }
} ZF;
//z[i] = longest common prefix of a[] till position i
```

7 7. Geometry

7.1 Check Point in ConvexHull in O(logN)

```
const int maxn = 5e3 + 10;
int n,m;
pii a[maxn];

int Sign(long long x){
    if (x < 0) return -1;
    if (x > 0) return 1;
    return 0;
}

pii Vect(pii p, pii q){
    return make_pair(q.x - p.x, q.y - p.y);
}

long long CCW(pii u, pii v){
    return Sign(1ll*u.x*v.y - 1ll*u.y*v.x);
}

bool same_spin(pii root, pii u, pii v){
    return (CCW(root, u)*CCW(root, v) > 0);
}

bool diff_spin(pii root, pii u, pii v){
    return (CCW(root, u)*CCW(root, v) < 0);
}

bool in_convexhull(pii K){
    pii VL = Vect(a[1], a[2]);
    pii VR = Vect(a[1], a[n]);
    pii VT = Vect(a[1], K);
    if (!same_spin(VL, VR, VT)) return 0;
    if (!same_spin(VR, VL, VT)) return 0;

    int L = 2;
    int R = n;
    while (R - L > 1){
        int mid = (L+R) >> 1;
        if (same_spin(Vect(a[1], a[mid]), VT, VL)) R = mid;
        else L = mid;
    }

    pii UL = Vect(a[L], a[1]);
    pii UR = Vect(a[L], a[R]);
    pii UT = Vect(a[L], K);
    if (diff_spin(UL, UR, UT)) return 0;
    if (diff_spin(UR, UL, UT)) return 0;
    if (CCW(UR, UT) == 0) return 0;
    // Quick understand : There is the only case that the
        algorithm will fail after found the triangle contain
         the point in the polygon
    return 1;
}

void solve(){
    cin >> n;
    up(i,1,n) cin >> a[i].x >> a[i].y;
    cin >> m;
    up(i,1,m){
        pii K;
        cin >> K.x >> K.y;
        if (in_convexhull(K)) cout << "YES";
        else cout << "NO";
        cout << "\n";
    }
}
```

7.2 Check Point in Polygon in O(n)

```
const int maxn = 1e5 + 10;
int n,m;
pii P[maxn];
pii K;
```

```
int sign(long long k){
    if (k < 0) return -1;
    if (k > 0) return 1;
    return 0;
}

pii vect(pii p, pii q){
    return make_pair(q.x - p.x, q.y - p.y);
}

long long dot(pii u, pii v){
    long long DOT = u.x*v.x + u.y*v.y;
    return sign(DOT);
}

long long cross(pii u, pii v){
    long long ccw = u.x*v.y - u.y*v.x;
    return sign(ccw);
}

int intersect(pii A, pii C, pii D){
    int ACD = cross(vect(A, C), vect(C, D));
    int CDA = cross(vect(C, D), vect(D, A));

    if (CDA == 0 && dot(vect(A, D), vect(A, C)) <= 0) return
         1;

    if (C.y <= A.y && D.y > A.y && ACD < 0) return 2;
    if (C.y > A.y && D.y <= A.y && ACD > 0) return 2;
    return 0;
}

string Point_in_Polygon(){
    int cnt = 0;
    bool boundary = 0;
    up(i,1,n){
        pii cur = P[i];
        pii next = P[i+1];
        if (intersect(K, cur, next) == 1){
            boundary = 1;
            break;
        }
        if (intersect(K, cur, next) == 2) ++cnt;
    }
    if (boundary) return "BOUNDARY";
    else if (cnt & 1) return "INSIDE";
    return "OUTSIDE";
}

void solve(){
    cin >> n >> m;
    up(i,1,n) cin >> P[i].x >> P[i].y;
    P[n+1] = P[1];
    up(i,1,m){
        cin >> K.x >> K.y;
        cout << Point_in_Polygon() << "\n";
    }
}
```

7.3 Find Convex Hull

```
static const int maxn = 2e5 + 10;
pii a[maxn];
pii R;
int n;

bool least(pii a, pii b){
    return (a.x < b.x || (a.x == b.x && a.y < b.y));
}

void in(){
    cin >> n;
    R.x = R.y = 2e9 + 1;
    up(i,1,n){
        cin >> a[i].x >> a[i].y;
        if (least(a[i], R)) R = a[i];
    }
}

pii Vect(pii a, pii b){
    return (make_pair(b.x - a.x, b.y - a.y));
}

long long CCW(pii a, pii b, pii c){
    pii u = Vect(a, b);
    pii v = Vect(b, c);
    return (1ll*u.x*v.y - 1ll*u.y*v.x);
}

bool counterclockwise(const pii& a, const pii& b){
    long long ccw = CCW(R, a, b);
    if (ccw == 0) return least(a, b);
    return ccw > 0;
}

int top = 0;
pii cv[maxn];
void find_convex(){
//       sort(a+1, a+n+1, counterclockwise);
    top = 1;
    cv[top] = R;

    for (int i = 2; i <= n; i++){
        while (top > 1 && CCW(cv[top-1], cv[top], a[i]) < 0)
            top--;
        cv[++top] = a[i];
    }
    pii endd = cv[top];
    for (int i = 2; i <= n; i++){
        if (a[i] != endd){
            if (CCW(R, a[i], endd) == 0) cv[++top] = a[i];
        }
    }
    cv[top+1] = R;
}

void PRINT(){
    cout << top << "\n";
    up(i,1,top){
        cout << cv[i].x << " " << cv[i].y << "\n";
    }
}
```

7.4 IT LINE

```
struct IT_Duong_Thang{
    static const int maxn = 1e6 + 10;
    static const int MOD = 2e9 + 11;
    int n;
    struct Line{
        int a,b,pos;
    };
    Line a[maxn];
    Line it[maxn << 2];

    void build(int nod, int l, int r){
        it[nod] = {0, MOD, 0};
        if (l == r) return;
        int mid = (l+r) >> 1;
        build(nod*2, l, mid);
        build(nod*2+1, mid+1, r);
    }

    long long get(Line L, int x){
        return 1ll*x*L.a + L.b;
    }

    void update(int nod, int l, int r, Line val){
        int mid = (l + r) >> 1;
        bool aboveLeft = (get(it[nod], l) > get(val, l));
        bool aboveMid = (get(it[nod], mid) > get(val, mid));
        if (aboveMid) swap(val, it[nod]);
        if (l == r) return;
        if (aboveLeft != aboveMid) update(nod*2, l, mid, val)
            ;
        else update(nod*2+1, mid+1, r, val);
    }

    void add(Line L){
        update(1, -maxn, maxn, L);
    }

    Line query(int nod, int l, int r, int x){
        if (l == r) return it[nod];
        int mid = (l + r) >> 1;
        if (x <= mid){
            Line L = query(nod*2, l, mid, x);
            if (get(it[nod], x) < get(L, x)) return it[nod];
            return L;
        }
        Line R = query(nod*2+1, mid+1, r, x);
        if (get(it[nod], x) < get(R, x)) return it[nod];
        return R;
    }

    Line _query(int k){
        return query(1, -maxn, maxn, k);
    }

    void solve(){
        build(1, -maxn, maxn);
        cin >> n;
        up(i,1,n){
            cin >> a[i].a >> a[i].b;
            a[i].pos = i;
            add(a[i]);
        }

        int q;
        cin >> q;
        while (q--){
            int k;
            cin >> k;
            Line res = _query(k);
            cout << get(res, k) << "\n";
        }
    }
} my_IT_Duong_Thang;
```

8 8. Sqrt Decomposition

8.1 Blocking

```
struct Blocker{
    static const int maxn = 1e5 + 10;
    int Bsize;
    multiset<int> BLOCK[maxn];
    int Lblock[maxn], Rblock[maxn];
    int n,q;
    int a[maxn];

    int getblock(const int& pos){
        return (pos + Bsize - 1)/Bsize;
    }

    void divide(){
        int lastblock = getblock(n);
        up(i,1,n){
            int cur = getblock(i);
            BLOCK[cur].insert(a[i]);

            if (cur == lastblock){
                Lblock[i] = (lastblock-1)*Bsize+1;
                Rblock[i] = n;
            }
            else{
                Rblock[i] = min(n, cur*Bsize);
                Lblock[i] = Rblock[i] - Bsize + 1;
            }
        }
    }

    void update(int pos, int val){
        int curblock = getblock(pos);
        auto it = BLOCK[curblock].find(a[pos]);
        BLOCK[curblock].erase(it);
        BLOCK[curblock].insert(val);
        a[pos] = val;
    }

    int query(int l, int r, int k){
        int L = getblock(l);
        int R = getblock(r);

        int minn = 1e9 + 7;
        if (L == R){
            up(i,l,r){
                if (a[i] >= k) minn = min(minn, a[i]);
            }
            return (minn == 1e9 + 7 ? -1 : minn);
        }

        for (int i = l; i <= Rblock[l]; i++){
            if (a[i] >= k) minn = min(minn, a[i]);
        }
        for (int i = Lblock[r]; i <= r; i++){
            if (a[i] >= k) minn = min(minn, a[i]);
        }
        for (int i = L+1; i <= R-1; i++){
            auto it = BLOCK[i].lower_bound(k);
            if (*it >= k) minn = min(minn, *it);
        }
        return (minn == 1e9 + 7 ? -1 : minn);
    }

    void solve(){
        cin >> n >> q;
        up(i,1,n) cin >> a[i];
        Bsize = sqrt(n);
        divide();

        up(i,1,q){
            int type;
            int l,r;
            int pos,val;
            cin >> type;
            if (type == 1){
                cin >> pos >> val;
                update(pos, val);
            }
            else {
                cin >> l >> r >> val;
                cout << query(l, r, val) << "\n";
            }
        }
    }
} my_Blocker; //find the lower_bound(k) in range [1, r]
```

8.2 Mo Algorithm

```
struct Mo_Algorithm{
    static const int maxn = 1e5 + 10;
    static const int maxq = 5e5 + 10;
    static const int BLOCK = 512; //should be (1 << 9)
    int n,q;

    struct Query{
        int l,r,id;
    Query(){}
    Query(int l, int r, int id):
        l(l), r(r), id(id)
    {}

    bool operator < (const Query& O) const {
        if (l/BLOCK != O.l/BLOCK) return l < O.l;
        return (l/BLOCK & 1 ? r < O.r : r > O.r);
    }
} Q[maxq];

struct Counter{
    void add(int x){}

    void sub(int pos){}

    int query(){ return 0; }
} MO;

int ans[maxq];

void solve(){
    cin >> n >> q;

    //do somthing with n here

    up(i,1,q){
        int l,r;
        cin >> l >> r;
        Q[i].l = l;
        Q[i].r = r;
        Q[i].id = i;
    }
    sort(Q+1, Q+q+1);

    int cl = 1;
    int cr = 0;
    up(i,1,q){
        int l = Q[i].l;
        int r = Q[i].r;
        int id = Q[i].id;
        while (cl > l) MO.add(--cl);
        while (cl < l) MO.sub(cl++);
        while (cr > r) MO.sub(cr--);
        while (cr < r) MO.add(++cr);
        ans[id] = MO.query();
    }

    up(i,1,q) cout << ans[i] << "\n";
}
} My_MO;
```

9 9. Others

9.1 DnC

```
const int maxn = 1e5 + 10;
int dp[11][maxn];
int n,k;

int cost(int l, int r){ return 0; }
//This should be done in O(1) or O(log)

void DIVIDE(int before[], int f[], int l, int r, int pl, int
      pr){
    if (l > r) return;

    int mid = (l+r) >> 1;
    int halve = pl;

    for (int i = pl; i <= min(mid-1, pr); i++){
        int val = before[i] + cost(i+1, mid);
        if (val <= f[mid]){
            f[mid] = val;
            halve = i;
        }
    }

    DIVIDE(before, f, l, mid-1, pl, halve);
    DIVIDE(before, f, mid+1, r, halve, pr);
}

void solve(){
    up(i,1,k){
        DIVIDE(dp[i-1], dp[i], i, n, i-1, n-1);
    }
    cout << dp[k][n];
}
```

9.2 DSU rollback

```
const int maxn = 1e5 + 10;
struct Query{
    int u,v,t;
};
struct DSU_save{
    int u,v,su,sv;
};

struct DSU_with_rollback{
    int par[maxn];
    stack<DSU_save> save;
    int comps;

    void init(int n){
        assert(save.empty());
        memset(par, -1, sizeof(par[0])*(n+1));
        comps = n;
    }

    int root(int x){
        while (par[x] >= 0) x = par[x];
        return x;
    }

    bool combinable(int u, int v){
        u = root(u);
        v = root(v);
        if (u == v) return false;

        if (par[u] > par[v]) swap(u, v);
        save.push({u, v, par[u], par[v]});
        --comps;
        par[u] += par[v];
        par[v] = u;
        return true;
    }

    bool connected(int u, int v){
        return (root(u) == root(v));
    }

    void rollback(){
        if (save.empty()) return; //in fact: "vector<DSU_save
            >save" never empty
        DSU_save cur = save.top();
        save.pop();
        ++comps;
        par[cur.u] = cur.su;
        par[cur.v] = cur.sv;
    }
};

struct Segment_Tree{
    int n;
    vector<vector<Query> > T;
int conn[maxn];
DSU_with_rollback dsu;

void init(int _n, int _q){ //be careful: number of n and
     q
    n = _q;
    T.clear();
    T.resize((n << 2) + 5);
    dsu.init(_n);
    memset(conn, -1, sizeof(conn[0])*(n+1));
}

void add_node(int nod, int l, int r, int u, int v, const
     Query& Q){
    if (l > v || r < u) return;
    if (l >= u && r <= v){
        T[nod].push_back(Q);
        return;
    }
    int mid = (l+r) >> 1;
    add_node(nod*2, l, mid, u, v, Q);
    add_node(nod*2+1, mid+1, r, u, v, Q);
}

void DFS(int nod, int l, int r){
    int rollcount = 0;
    for (Query& Q : T[nod]){
        if (Q.t == -1 && dsu.combinable(Q.u, Q.v)) ++
             rollcount;
    }

    if (l == r){ //Reach the Query point
        for (Query& Q : T[nod]) if (Q.t > 0){
            conn[l] = dsu.connected(Q.u, Q.v);
        }
    }
    else {
        int mid = (l+r) >> 1;
        DFS(nod*2, l, mid);
        DFS(nod*2+1, mid+1, r);
    }
    while (rollcount--) dsu.rollback();
}

void add_node(int l, int r, const Query& Q){
    add_node(1, 1, n, l, r, Q);
}

void answer(vector<int>& query_pos){
    for (int& i : query_pos) cout << conn[i];
    }
} Tree;


int n,q;
vector<int> query_pos;
map<pii, stack<int> > M;
void build_tree(){
    string k;
    int type,u,v;
    up(i,1,q){
        cin >> k >> u >> v;
        if (k == "1") type = 1;
        else if (k == "2") type = 2;
        else type = 3;

        if (type == 3) query_pos.push_back(i);
        if (u > v) swap(u,v);
        pii cur = {u, v};

        if (type == 1) M[cur].push(i);
        else if (type == 2){
            if (M[cur].empty()) continue;
            Tree.add_node(M[cur].top(), i, {u, v, -1});
            M[cur].pop();
        }
        else Tree.add_node(i, i, {u, v, i}); // type == "conn
             "
    }

    for (auto& rest : M){
        int u = rest.f.f;
        int v = rest.f.s;
        while (!rest.s.empty()){
            Tree.add_node(rest.s.top(), q, {u, v, -1});
            rest.s.pop();
        }
    }
}

void solve(){
    cin >> n >> q;
    Tree.init(n, q);
    build_tree();
    Tree.DFS(1, 1, q);
    Tree.answer(query_pos);
}
```

9.3 Factorize

```
#define u64 unsigned long long
#define u128 __uint128_t
#define all(x) x.begin(), x.end()
#define ep emplace_back
mt19937_64 RNG(chrono::high_resolution_clock::now().
     time_since_epoch().count());
u64 UID(u64 l, u64 r){
    uniform_int_distribution<mt19937_64::result_type>
         random_number(l, r);
    return random_number(RNG);
}
struct FACTORS{
    const vector<u64> Milbase = {2, 3, 5, 7, 11, 13, 17, 19,
         23, 29, 31, 37};

    u64 fpow(u64 x, u64 n, u64 M){
        u64 res = 1;
        for (; n; n >>= 1, x = (u128)x*x % M){
            if (n & 1) res = (u128)res*x % M;
        }
        return res;
    }

    bool divisible(u64 base, u64 cnt2, u64 d, u64 M){
        u64 extrem = fpow(base, d, M);
        if (extrem == 1 || extrem == M-1) return true;

        up(i,1,cnt2-1){
            extrem = (u128)extrem*extrem % M;
            if (extrem == M-1) return true;
        }
        return false;
    }

    bool MillerRabin(u64 n){
        if (n < 2) return false;

        u64 d = n-1;
        int cnt2 = 0;
        while ((d & 1) == 0){
            d >>= 1;
            ++cnt2;
        }

        for (auto base : Milbase){
            if (n == base) return true;
            if (n % base == 0) return false;
            if (!divisible(base, cnt2, d, n)) return false;
    }
    return true;
}

vector<u64> quickfact(int n){
    vector<u64> res;
    for (int i = 2; i*i <= n; i++){
        while (n % i == 0){
            n /= i;
            res.ep(i);
        }
    }
    if (n > 1) res.ep(n);
    return res;
}

const int C = 2;
u64 f(u64 x, u64 MOD){
    return ((u128)x*x + C) % MOD;
}

u64 ABS(u64 x, u64 y){
    if (x > y) return x-y;
    return y-x;
}

u64 Brent(u64 n){
    u64 x,y,k,res;
    x = y = UID(2, n-1);
    k = 1;
    while (true){
        up(i,1,k){
            x = f(x, n);
            res = __gcd(ABS(x, y), n);
            if (res > 1) return res;
        }
        y = x;
        k <<= 1;
    }
    assert(false);
}

vector<u64> Pollard(u64 x){
    if (MillerRabin(x)) return {x};
    if (x <= 3e4) return quickfact(x);

    vector<u64> res;
    u64 p = Brent(x);
    vector<u64> L = Pollard(p);
    vector<u64> R = Pollard(x/p);
        res.insert(res.end(), all(L));
        res.insert(res.end(), all(R));
        return res;
    } //return prime factors of x in arbitrary order
} FACTS;
```

9.4 IT SEGMENT - example

```
const int maxn = 1e5 + 10;

int n;
int x[maxn], w[maxn], e[maxn];

int realval[maxn];
struct LINE{
    int a,b;
    int id;
    int get(const int& x){
        return a*realval[x] + b;
    }
};
LINE T[maxn << 2]; // IT Doan Thang

bitset<maxn << 2> nothing;
pii query(int nod, int l, int r, int pos){
    pii res;
    res = {T[nod].get(pos), T[nod].id};
    if (l == r) return res;

    pii val;
    int mid = (l+r) >> 1;

    if (pos <= mid) val = query(nod*2, l, mid, pos);
    else val = query(nod*2+1, mid+1, r, pos);
    res = max(res, val);
    return res;
}

void update(int nod, int l, int r, int u, int v, LINE Line){
    if (l > v || r < u) return;

    int mid = (l+r) >> 1;
    if (l >= u && r <= v){
        int id1 = T[nod].id;
        int id2 = Line.id;
        pii TL = {T[nod].get(l), id1};
        pii TR = {T[nod].get(r), id1};
        pii LineL = {Line.get(l), id2};
        pii LineR = {Line.get(r), id2};
        if (nothing[nod]){
            nothing[nod] = 0;
            T[nod] = Line;
            return;
        }

        if (TL >= LineL && TR >= LineR){
            return;
        }
        if (TL <= LineL && TR <= LineR){
            T[nod] = Line;
            return;
        }
    }

    update(nod*2, l, mid, u, v, Line);
    update(nod*2+1, mid+1, r, u, v, Line);
}

vector<int> X; // Vector Roi rac hoa
int SIZE;
void update(int l, int r, const LINE& Line){
    update(1, 1, SIZE, l, r, Line);
}
pii query(int pos){
    return query(1, 1, SIZE, pos);
}
int LB(const int& val){
    return lower_bound(all(X), val) - X.begin();
}
int UB(const int& val){
    return upper_bound(all(X), val) - X.begin() - 1;
}

int f[maxn];
int tr[maxn];
void DP(){
    up(i,1,n){
        int cur = LB(x[i]);
        auto found = query(cur);
        f[i] = found.first + e[i];
        tr[i] = found.second;
        if (w[i] == 0) update(1, SIZE, {0ll, f[i], i});
        // never mess with binary search !!
        // this update mean "insert a line from -oo to oo
           with a = 0 and b = f[i]"
        // 0 = lower_bound(-oo);
        // SIZE = upper_bound(oo) - 1;
        else{
            int l = x[i] - f[i]/w[i];
            int r = x[i] + f[i]/w[i];
            update(LB(l), cur, {w[i], f[i] - w[i]*x[i], i});
            update(cur, UB(r), {-w[i], f[i] + w[i]*x[i], i});
        }
    }
}

void solve(){
    cin >> n;

    up(i,1,n){
        cin >> x[i] >> w[i] >> e[i];
        X.push_back(x[i]);
    }
    sort(all(X));
    purge(X);
    SIZE = X.size();
    nothing.set();

    X.insert(X.begin(), -1e9 - 7);
    X.push_back(1e9 + 7);
    up(i,1,SIZE) realval[i] = X[i];

    DP();

    int last = -1;
    int maxx = 0;
    up(i,1,n){
        if (maxx < f[i]){
            maxx = f[i];
            last = i;
        }
    }
    cout << maxx << "\n";

    vector<int> path;
    while (last != 0){
        path.ep(last);
        last = tr[last];
    }
    reverse(all(path));
    for (auto x : path) cout << x << " ";
}
```

9.5 Kadane1D

```
struct Kadane_Trace1D{
    static const int maxn = 1e6 + 10;
    long long a[maxn];
    int n;

    void KADANE(){
        bool all_negative = 1;
        cin >> n;
        up(i,1,n) {
            cin >> a[i];
            all_negative &= (a[i] < 0);
        }
        if (all_negative){
            cout << 0 << " " << 0 << " " << 0;
            exit(0);
        }

        int l = 1;
        int r = 1;
        long long sum = 0;
        long long res = (long long)(-1e18);

        int pre = 1;
        up(i,1,n){
            sum += a[i];
            if (sum < 0){
                sum = 0;
                pre = i+1;
            }
            else if (sum > res){
                res = sum;
                l = pre;
                r = i;
            }
        }
        cout << res << " " << l << " " << r;
    }
} KAN;
```

9.6 Kadane2D

```
struct Kadane_Trace2D{
    static const int maxn = 501;
    static const long long LIM = 1e18 + 31;
    int a[maxn][maxn];
    long long tr[maxn];
    int n,m;

    void Kadane1D(int i, int k, int& l, int& r, long long&
         lbest){
        long long sum = 0ll;
        int pre = 1;
        up(j,1,m){
            tr[j] += 1ll*a[k][j];
            sum += tr[j];
            if (sum < 0ll){
                sum = 0ll;
                pre = j+1;
            }
            else if (sum > lbest){
                lbest = sum;
                l = pre;
                r = j;
            }
            if (k == n) tr[j] = 0;
        }
    }

    void Kadane2D(){
        cin >> n >> m;
        up(i,1,n) up(j,1,m) cin >> a[i][j];

        long long gbest = -LIM;
        int u,v,x,y;
        u = v = x = y = 1;
        up(i,1,n) up(k,i,n){
            int l = 1;
            int r = 1;
            long long lbest = -LIM;
            Kadane1D(i, k, l, r, lbest);

            if (gbest < lbest){
                gbest = lbest;
                u = i;
                v = k;
                x = l;
                y = r;
            }
        }
        cout << gbest << "\n";
        cout << u << " " << x << " " << v << " " << y << "\n"
             ;
    }
} KAN2D;
```

9.7 Parallel BS

```
struct Parallel_BS{
    static const int maxn = 1e5 + 10;
    int par[maxn], cnt[maxn];
int L[maxn], R[maxn];
int n,m,q;
struct Query{
    int x,y;
    int need;
} Q[maxn];
vector<int> Group[maxn];
pair<int, int> edge[maxn];

int root(int x){
    if (par[x] == 0) return x;
    return par[x] = root(par[x]);
}

void Union(int u, int v){
    u = root(u);
    v = root(v);
    if (u == v) return;
    if (cnt[u] < cnt[v]) swap(u,v);
    par[v] = u;
    cnt[u] += cnt[v];
}

void solve(){
    cin >> n >> m;
    up(i,1,m) cin >> edge[i].u >> edge[i].v;

    cin >> q;
    up(i,1,q){
        cin >> Q[i].x >> Q[i].y >> Q[i].need;
        R[i] = m;
    }

    for (int loop = int(ceil(log2(m))); loop; loop--){
        up(i,1,n) par[i] = 0, cnt[i] = 1;

        up(i,1,q) if (R[i] - L[i] > 1){
            Group[(L[i] + R[i])/2].ep(i);
        }

        up(i,1,m){
            Union(edge[i].u, edge[i].v);
            for (int& t : Group[i]){
                int u = root(Q[t].x);
                int v = root(Q[t].y);
                int sum;
                if (u == v) sum = cnt[u];
                else sum = cnt[u] + cnt[v];

                if (sum >= Q[t].need) R[t] = i;
                else L[t] = i;
            }
            Group[i].clear();
        }
    }

    up(i,1,q) cout << R[i] << "\n";
    }
} PBS;
```

9.8 Persistent Segment Tree

```
const int maxn = 1e5 + 10;
const int LOG = log2(maxn)+2;
int a[maxn];
struct PST_node{
    int l,r,sum;
};
PST_node T[maxn*LOG];
int cnt = 0;
int n,q;


int ROOT[maxn];

void push_up(int nod){
    T[nod].sum = T[T[nod].l].sum + T[T[nod].r].sum;
}

void update(int& nod, int prev_nod, int l, int r, int pos,
     int val){
    nod = ++cnt;
    T[nod] = T[prev_nod];
    if (l == r){
        T[nod].sum += val;
        return;
    }
    int mid = (l+r) >> 1;
    if (pos <= mid) update(T[nod].l, T[prev_nod].l, l, mid,
         pos, val);
    else update(T[nod].r, T[prev_nod].r, mid+1, r, pos, val);
    push_up(nod);
}

int sum(int nod, int l, int r, int u, int v){
    if (u > v) return 0;
    if (l > v || r < u) return 0;
    if (l >= u && r <= v) return T[nod].sum;
    int mid = (l+r) >> 1;
    int L = sum(T[nod].l, l, mid, u, v);
    int R = sum(T[nod].r, mid+1, r, u, v);
    return L+R;
}

int Kth(int& nod, int& prev_nod, int l, int r, int k){
    if (l == r) return l;
    int mid = (l+r) >> 1;
    int CNT = T[T[nod].l].sum - T[T[prev_nod].l].sum;
    if (CNT >= k) return Kth(T[nod].l, T[prev_nod].l, l, mid,
         k);
    return Kth(T[nod].r, T[prev_nod].r, mid+1, r, k - CNT);
}

vector<int> V;
void example(){
    cin >> n >> q;
    up(i,1,n){
        cin >> a[i];
        V.push_back(a[i]);
    }
    sort(all(V));
    V.resize(unique(all(V)) - V.begin());
    up(i,1,n) {
        int x = lower_bound(all(V), a[i]) - V.begin() + 1;
        update(ROOT[i], ROOT[i-1], 1, n, x, 1);
    }

    V.clear();
    V.push_back(-1e9 - 7);
    up(i,1,n) V.push_back(a[i]);
    sort(all(V));
    V.resize(unique(all(V)) - V.begin());

    up(i,1,q){
        int l,r,k;
        cin >> l >> r >> k;
        int x = Kth(ROOT[r], ROOT[l-1], 1, n, k);
        cout << V[x] << "\n";
    }
}
```

9.9 Segment Tree 2D

```
const int maxn = 1e3 + 10;
const int LIM = 1e9 + 7;
int n,m,a,b,c,d;
int P[maxn][maxn];
int S[maxn][maxn];
int sum(int ux, int vx, int uy, int vy){
    return P[vx][vy] - P[ux-1][vy] - P[vx][uy-1] + P[ux-1][uy
        -1];
}

int T[maxn << 2][maxn << 2];
void buildY(int nodx, int lx, int rx, int nody, int l, int r
    ){
    if (l == r){
        if (lx == rx){
            T[nodx][nody] = S[lx][l];
        }
        else T[nodx][nody] = min(T[nodx*2][nody], T[nodx
            *2+1][nody]);
        return;
    }
    int mid = (l+r) >> 1;
    buildY(nodx, lx, rx, nody*2, l, mid);
    buildY(nodx, lx, rx, nody*2+1, mid+1, r);
    T[nodx][nody] = min(T[nodx][nody*2], T[nodx][nody*2+1]);
}

void buildX(int nod, int l, int r){
    if (l != r){
        int mid = (l+r) >> 1;
        buildX(nod*2, l, mid);
        buildX(nod*2+1, mid+1, r);
    }
    buildY(nod, l, r, 1, 1, m);
}

int queryY(int nodx, int nody, int l, int r, int u, int v){
    if (l > v || r < u) return LIM;
    if (l >= u && r <= v) return T[nodx][nody];
    int mid = (l+r) >> 1;
    int L = queryY(nodx, nody*2, l, mid, u, v);
    int R = queryY(nodx, nody*2+1, mid+1, r, u, v);
    return min(L, R);
}

int queryX(int nod, int l, int r, int ux, int vx, int uy,
    int vy){
    if (l > vx || r < ux) return LIM;
    if (l >= ux && r <= vx) return queryY(nod, 1, 1, m, uy,
        vy);
    int mid = (l+r) >> 1;
    int L = queryX(nod*2, l, mid, ux, vx, uy, vy);
    int R = queryX(nod*2+1, mid+1, r, ux, vx, uy, vy);
    return min(L, R);
}

void build2D(){
    buildX(1, 1, n);
}

int query2D(int ux, int vx, int uy, int vy){
    return queryX(1, 1, n, ux, vx, uy, vy);
}
```

9.10 Sum from this node to all other nodes

```
const int maxn = 2e5 + 10;
long long under[maxn];
long long h[maxn];
long long D[maxn];
vector<int> a[maxn];
int n;

void preDFS(int u, int par){
    under[u] = 1;
    for (int v : a[u]){
        if (v == par) continue;
        h[v] = h[u] + 1;
        preDFS(v, u);
        under[u] += under[v];
    }
}

void sumDist(int u, int par, int N){
    for (int v : a[u]){
        if (v == par) continue;
        D[v] = D[u] + 1ll*N - 1ll*2*under[v];
        sumDist(v, u, N);
    }
}

void solve(){
    cin >> n;
    int u,v;
    up(i,1,n-1){
        cin >> u >> v;
        a[u].ep(v);
        a[v].ep(u);
    }
    preDFS(1, -1);
    up(i,1,n) D[1] += 1ll*h[i];
    sumDist(1, -1, n);
    up(i,1,n) cout << D[i] << " ";
}
```

9.11 Sweep Line Closest Pair

```
long long ClosestPair(vector<pair<int, int> > p) {
    int n = p.size();
    sort(p.begin(), p.end());
    set<pair<int, int>> S;

    long long best_dist = 1e18;
    int j = 0;
    for (int i = 0; i < n; i++) {
        int d = ceil(sqrt(best_dist));
        while (p[i].first - p[j].first >= d && j < n) {
            S.erase({p[j].second, p[j].first});
            ++j;
        }

        auto it1 = S.lower_bound({p[i].second - d, p[i].first
            });
        auto it2 = S.upper_bound({p[i].second + d, p[i].first
            });

        for (auto it = it1; it != it2; ++it) {
            int dx = p[i].first - it->second;
            int dy = p[i].second - it->first;
            best_dist = min(best_dist, 1ll*dx*dx + 1ll*dy*dy)
                ;
        }
        S.insert({p[i].second, p[i].first});
    }
    return best_dist;
}
```