University of Engineering and Technology - TurboDB (22-23) Notebook

Mục lục

1 Flow and Matching

1.1 Maximum Flow (Dinic)

```cpp
// In case we need to find Maximum flow of network with both minimum capacity and maximum capacity,
     let s* and t* be virtual source and virtual sink.

/*
Then, each edge (u->v) with lower cap l and upper cap r will be changed in to 3 edge:

- u->v whit capacity r-l
- u->t* with capacity l
- s*->v with capacity l
*/
// We need add one other edge t->s with capacity Inf
// Maximum Flow on original graph is the Maximum Flow on new graph: s*->t*

struct Edge
{
    int u, v;
    ll c;
    Edge() {}
    Edge(int u, int v, ll c)
    {
        this->u = u;
        this->v = v;
        this->c = c;
    }
};
struct Dinic
{
    const ll Inf = 1e17;
    vector<vector<int>> adj;
    vector<vector<int>::iterator> cur;
    vector<Edge> s;
    vector<int> h;
    int sink, t;
    int n;
    Dinic(int n)
    {
        this->n = n;
        adj.resize(n + 1);
        h.resize(n + 1);
        cur.resize(n + 1);
        s.reserve(n);
    }
    void AddEdge(int u, int v, ll c)
    {
        s.emplace_back(u, v, c);
        adj[u].push_back(s.size() - 1);
        s.emplace_back(v, u, 0);
        adj[v].push_back(s.size() - 1);
    }
    bool BFS()
    {
        fill(h.begin(), h.end(), n + 2);
        queue<int> pq;
        h[t] = 0;
        pq.emplace(t);
        while (pq.size())
        {
            int c = pq.front();
            pq.pop();
            for (auto i : adj[c])
                if (h[s[i ^ 1].u] == n + 2 && s[i ^ 1].c != 0)
                {
                    h[s[i ^ 1].u] = h[c] + 1;
                    if (s[i ^ 1].u == sink)
                        return true;
                    pq.emplace(s[i ^ 1].u);
                }
        }
        return false;
    }
    ll DFS(int v, ll flowin)
    {
        if (v == t)
            return flowin;
        ll flowout = 0;
        for (; cur[v] != adj[v].end(); ++cur[v])
        {
            int i = *cur[v];
            if (h[s[i].v] + 1 != h[v] || s[i].c == 0)
                continue;
            ll q = DFS(s[i].v, min(flowin, s[i].c));
            flowout += q;
            if (flowin != Inf)
                flowin -= q;
            s[i].c -= q;
            s[i ^ 1].c += q;
            if (flowin == 0)
                break;
        }
        return flowout;
    }
    void BlockFlow(ll &flow)
    {
        for (int i = 1; i <= n; ++i)
            cur[i] = adj[i].begin();
        flow += DFS(sink, Inf);
    }
    ll MaxFlow(int s, int t)
    {
        this->sink = s;
        this->t = t;
```

```
        ll flow = 0;
        while (BFS())
            BlockFlow(flow);
        return flow;
    }
};
```

1.2 Maximum Matching (HopCroft - Karp)

```
// Trace to find vertex cover and independence set
/*
        Y* = Set of vertices y such that exist an argument path from y to a vertex x which isn't
              matched
        X* = Set of matched vertices x that x isn't matched with a vertex in Y*

        (X* v Y*) is vertex cover
*/

struct HopCroft_Karp
{
    const int NoMatch = -1;
    vector<int> h, S, match;
    vector<vector<int>> adj;
    int nx, ny;
    bool found;
    HopCroft_Karp(int nx = 0, int ny = 0)
    {
        this->nx = nx;
        this->ny = ny;
        S.reserve(nx);
        h.resize(ny + 5);
        adj.resize(nx + 5);
        match.resize(ny + 5, NoMatch);
    }
    void Clear()
    {
        for (int i = 1; i <= nx; ++i)
            adj[i].clear();
        S.clear();
        fill(match.begin(), match.end(), NoMatch);
    }
    void AddEdge(int x, int y)
    {
        adj[x].emplace_back(y);
    }
    bool BFS()
    {
        fill(h.begin(), h.end(), 0);
        queue<int> q;
        for (auto x : S)
            for (auto i : adj[x])
                if (h[i] == 0)
                {
                    q.emplace(i);
                    h[i] = 1;
                }
        while (q.size())
        {
            int x, ypop = q.front();
            q.pop();
            if ((x = match[ypop]) == NoMatch)
                return true;
            for (auto i : adj[x])
                if (h[i] == 0)
                {
                    h[i] = h[ypop] + 1;
                    q.emplace(i);
                }
        }
        return false;
    }
    void dfs(int v, int lv)
    {
        for (auto i : adj[v])
            if (h[i] == lv + 1)
            {
                if (match[i] == NoMatch)
                    found = 1;
                else
                    dfs(match[i], lv + 1);
                if (found)
                {
                    match[i] = v;
                    return;
                }
            }
    }
    int MaxMatch()
    {
        int ans(0);
        for (int i = 1; i <= nx; ++i)
            S.emplace_back(i);
        while (BFS())
        {
            for (int i = S.size() - 1; ~i; --i)
            {
                found = 0;
                dfs(S[i], 0);
                if (found)
                {
                    ++ans;
                    S[i] = S.back();
                    S.pop_back();
                }
            }
        }
        return ans;
    }

};
```

1.3 Maximum Matching (O(n^2))

```
// start from 1
// O(n^2)
struct Maximum_matching
{
    int nx, ny, t;
    vector<int> Visited, match;
    vector<vector<int>> a;

    Maximum_matching(int nx = 0, int ny = 0)
    {
        Assign(nx, ny);
    }

    void Assign(int nx, int ny)
    {
        this->nx = nx;
        this->ny = ny;
        t = 0;
        Visited.assign(nx + 5, 0);
        match.assign(ny + 5, 0);
        a.resize(nx + 5, {});
    }

    void AddEdge(int x, int y)
    {
        a[x].emplace_back(y);
    }

    bool visit(int u)
    {
        if (Visited[u] != t)
            Visited[u] = t;
        else
            return false;

        for (int i = 0; i < a[u].size(); i++)
        {
            int v = a[u][i];
            if (!match[v] || visit(match[v]))
            {
                match[v] = u;
                return true;
            }
        }
        return false;
    }

    int MaxMatch()
    {
        int ans(0);

        for (int i = 1; i <= nx; i++)
        {
            t++;
            ans += visit(i);
        }

        return ans;
    }
};
```

1.4 Min Cost Flow

```
struct Edge
{
    int u, v;
    ll c, w;
    Edge(const int &u, const int &v, const ll &c, const ll &w) : u(u), v(v), c(c), w(w) {}
};
struct MaxFlowMinCost
{
    const ll Inf = 1e17;
    int n, source, sink;
    vector<ll> d;
    vector<int> par;
    vector<bool> inqueue;
    vector<Edge> s;
    vector<vector<int>> adj;
    MaxFlowMinCost(int n)
    {
        this->n = n;
        s.reserve(n * 2);
        d.resize(n + 5);
        inqueue.resize(n + 5);
        par.resize(n + 5);
        adj.resize(n + 5);
    }
    void AddEdge(int u, int v, ll c, ll w)
    {
        s.emplace_back(u, v, c, w);
        adj[u].emplace_back(s.size() - 1);
        s.emplace_back(v, u, 0, -w);
        adj[v].emplace_back(s.size() - 1);
    }
    bool SPFA()
    {
        fill(d.begin(), d.end(), Inf);
        fill(par.begin(), par.end(), s.size());
        fill(inqueue.begin(), inqueue.end(), 0);
        d[sink] = 0;
        queue<int> q;
        q.emplace(sink);
        inqueue[sink] = 1;
        while (q.size())
        {
            int c = q.front();
            inqueue[c] = 0;
            q.pop();
            for (auto i : adj[c])
                if (s[i ^ 1].c > 0 && d[s[i].v] > d[c] + s[i ^ 1].w)
                {
                    par[s[i].v] = i ^ 1;
                    d[s[i].v] = d[c] + s[i ^ 1].w;
                    if (!inqueue[s[i].v])
                    {
                        q.emplace(s[i].v);
                        inqueue[s[i].v] = 1;
                    }
                }
        }
        return (d[source] < Inf);
    }
    pair<ll, ll> MaxFlow(int so, int t, ll k)
    {
        source = so;
        sink = t;
        ll Flow(0), cost(0);
        while (k && SPFA())
        {
            ll q(Inf);
            int v = source;
            while (v != sink)
            {
                q = min(q, s[par[v]].c);
                v = s[par[v]].v;
            }

            q = min(q, k);

            cost += d[source] * q;
            Flow += q;
            k -= q;

            v = source;
            while (v != sink)
            {
                s[par[v]].c -= q;
                s[par[v] ^ 1].c += q;
                v = s[par[v]].v;
            }
        }
        return {Flow, cost};
    }
};
```

2 Geometry

2.1 Pick's Theorem

```
Given a certain lattice polygon (all its vertices have integer coordinates in some 2D grid) with non-
      zero area.

We denote its area by $S$, the number of points with integer coordinates lying strictly inside the
      polygon by $I$ and the number of points lying on polygon sides by $B$.

Then, the Pick's formula states:

\begin{center}
$S=I+ \frac{B}{2} - 1$
\end{center}
```

2.2 Smallest Circle - Emo Welzl (Contain all points)

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

typedef pair<ld, ld> point;
typedef pair<point, ld> circle;
#define X first
#define Y second

// O(n)
// Remember to change size of set points
// All point must be save in array a[] below

namespace emowelzl
{
    const int N = 100005; // Size of set points
    int n;
    point a[N];

    point operator+(point a, point b)
    {
        return point(a.X + b.X, a.Y + b.Y);
    }
    point operator-(point a, point b) { return point(a.X - b.X, a.Y - b.Y); }
    point operator/(point a, ld x) { return point(a.X / x, a.Y / x); }
    ld abs(point a) { return sqrt(a.X * a.X + a.Y * a.Y); }

    point center_from(ld bx, ld by, ld cx, ld cy)
    {
        ld B = bx * bx + by * by, C = cx * cx + cy * cy, D = bx * cy - by * cx;
        return point((cy * B - by * C) / (2 * D), (bx * C - cx * B) / (2 * D));
    }

    circle circle_from(point A, point B, point C)
    {
        point I = center_from(B.X - A.X, B.Y - A.Y, C.X - A.X, C.Y - A.Y);
        return circle(I + A, abs(I));
    }

    circle f(int n, vector<point> T)
    {
        if (T.size() == 3 || n == 0)
        {
            if (T.size() == 0)
                return circle(point(0, 0), -1);
            if (T.size() == 1)
                return circle(T[0], 0);
            if (T.size() == 2)
                return circle((T[0] + T[1]) / 2, abs(T[0] - T[1]) / 2);
            return circle_from(T[0], T[1], T[2]);
        }
        random_shuffle(a + 1, a + n + 1);
        circle Result = f(0, T);
        for (int i = 1; i <= n; i++)
            if (abs(Result.X - a[i]) > Result.Y + 1e-9)
            {
                T.push_back(a[i]);
```

```
                Result = f(i - 1, T);
                T.pop_back();
            }
        return Result;
    }
};

int main()
{
    cin >> emowelzl::n;

    for (int i = 1; i <= emowelzl::n; ++i)
        cin >> emowelzl::a[i].X >> emowelzl::a[i].Y;

    cout << emowelzl::f(emowelzl::n, {}).Y * 2;
}
```

2.3 Closest pair of points in set

```
// Find pair of points that have closest distance

#include <iostream>
#include <cstdio>
#include <algorithm>
#include <iomanip>
#include <cmath>
#include <vector>

using namespace std;

using ll = long long;
using ld = long double;

const int N = 5e4 + 2;
const ll Inf = 1e17;
#define sq(x) ((x) * (x))

struct Point
{
    ll x, y;
    int id;

    Point(const ll &x = 0, const ll &y = 0) : x(x), y(y) {}

    Point operator-(const Point &a) const
    {
        return Point(x - a.x, y - a.y);
    }
    ll len()
    {
        return x * x + y * y;
    }
};

namespace ClosestPoint
{
    int n, xa, ya;
    ll ans;
    Point a[N];

    ll Bruteforce(int l, int r)
    {
        for (; l < r; ++l)
            for (int i = l + 1; i <= r; ++i)
                if (ans > (a[l] - a[i]).len())
                {
                    ans = (a[l] - a[i]).len();
                    xa = a[l].id;
                    ya = a[i].id;
                }
        return ans;
    }

    void Brute(vector<int> &s)
    {
        sort(s.begin(), s.end(), [&](const int &x, const int &y)
             { return a[x].y < a[y].y; });
        for (int i = 0; i < s.size(); ++i)
            for (int j = i + 1; j < s.size() && sq(abs(a[s[i]].y - a[s[j]].y)) <= ans; ++j)
                if (ans > (a[s[i]] - a[s[j]]).len())
                {
                    ans = (a[s[i]] - a[s[j]]).len();
                    xa = a[s[i]].id;
                    ya = a[s[j]].id;
                }
    }

    void DAC(int l, int r)
    {
        if (r - l <= 3)
        {
            Bruteforce(l, r);
            return;
        }
        int mid = (l + r) / 2;

        DAC(l, mid);
        DAC(mid + 1, r);

        vector<int> s;
        for (int i = l; i <= r; ++i)
            if (sq(a[i].x - a[mid].x) <= ans)
                s.push_back(i);

        Brute(s);
    }

    void calc()
    {
        sort(a + 1, a + n + 1, [&](const Point &a, const Point &b)
             { return a.x < b.x || (a.x == b.x && a.y < b.y); });
        ans = Inf;

        DAC(1, n);

        if (xa > ya)
            swap(xa, ya);

        cout << xa << " " << ya << "\n";
        cout << fixed << setprecision(6) << sqrt((ld)ans);
    }
};

Point a[N];
int n;

void Read()
{
    cin >> n;
    for (int i = 1; i <= n; ++i)
    {
        cin >> a[i].x >> a[i].y;
        a[i].id = i;
    }
}

void Solve()
{
    ClosestPoint::n = n;
    for (int i = 1; i <= n; ++i)
        ClosestPoint::a[i] = a[i];

    ClosestPoint::calc();
}

int32_t main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    Read();
    Solve();
}
```

2.4 Manhattan MST

```
// Idea is to reduce number of edges which are candidates to be in the MST
// Then apply Kruskal algorithm to find MST

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

constexpr int N = 2e5 + 2;
constexpr ll Inf = 1e17;

namespace manhattanMST
{
    /// disjoint set union
    struct dsu
    {
        int par[N];
        dsu()
        {
            memset(par, -1, sizeof par);
```

```
    }

    int findpar(int v)
    {
        return par[v] < 0 ? v : par[v] = findpar(par[v]);
    }

    bool Merge(int u, int v)
    {
        u = findpar(u);
        v = findpar(v);
        if (u == v)
            return false;

        if (par[u] < par[v])
            swap(u, v);

        par[v] += par[u];
        par[u] = v;

        return true;
    }
};

// Fenwick Tree Min

struct FenwickTreeMin
{
    pair<ll, int> a[N];
    int n;

    FenwickTreeMin(int n = 0)
    {
        Assign(n);
    }

    void Assign(int n)
    {
        this->n = n;
        fill(a, a + n + 1, make_pair(Inf, -1));
    }

    void Update(int p, pair<ll, int> v)
    {
        for (; p <= n; p += p & -p)
            a[p] = min(a[p], v);
    }

    pair<ll, int> Get(int p)
    {
        pair<ll, int> ans({Inf, -1});

        for (; p; p -= p & -p)
            ans = min(ans, a[p]);
        return ans;
    }
};

struct Edge
{
    int u, v;
    ll w;
    Edge(const int &u = 0, const int &v = 0, const ll &w = 0) : u(u), v(v), w(w) {}
    bool operator<(const Edge &a) const
    {
        return w < a.w;
    }
};

int n;
ll x[N], y[N];
vector<Edge> edges;

ll dist(int i, int j)
{
    return abs(x[i] - x[j]) + abs(y[i] - y[j]);
}

#define Find(x, v) (lower_bound(x.begin(), x.end(), v) - x.begin() + 1)
void createEdge(int a1, int a2, int b1, int b2, int c1, int c2)
{
    vector<array<ll, 4>> v;
    vector<ll> s;

    for (int i = 1; i <= n; i++)
    {
        v.push_back({a1 * x[i] + a2 * y[i],
                     b1 * x[i] + b2 * y[i],
                     c1 * x[i] + c2 * y[i],
                     i});
        s.emplace_back(b1 * x[i] + b2 * y[i]);
    }

    sort(s.begin(), s.end());
    s.resize(unique(s.begin(), s.end()) - s.begin());
    sort(v.begin(), v.end());

    FenwickTreeMin f(n);

    for (auto [num1, num2, cost, idx] : v)
    {
        num2 = Find(s, num2);

        int res = f.Get(num2).second;
        if (res != -1)
            edges.emplace_back(res, idx, dist(res, idx));

        f.Update(num2, make_pair(cost, idx));
    }
}

void calc()
{
    edges.clear();

    createEdge(1, -1, -1, 0, 1, 1);  // R1
    createEdge(-1, 1, 0, -1, 1, 1);  // R2
    createEdge(-1, -1, 0, 1, 1, -1); // R3
    createEdge(1, 1, -1, 0, 1, -1);  // R4
    createEdge(-1, 1, 1, 0, -1, -1); // R5
    createEdge(1, -1, 0, 1, -1, -1); // R6
    createEdge(1, 1, 0, -1, -1, 1);  // R7
    createEdge(-1, -1, 1, 0, -1, 1); // R8

    dsu f;

    sort(edges.begin(), edges.end());
    vector<pair<int, int>> res;
    ll ans(0);

    for (auto i : edges)
        if (f.Merge(i.u, i.v))
        {
            ans += i.w;
            res.emplace_back(i.u, i.v);
        }

    cout << ans << "\n";

    for (auto i : res)
        cout << i.first << " " << i.second << "\n";
}
};

int32_t main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    cin >> manhattanMST::n;

    for (int i = 1; i <= manhattanMST::n; ++i)
        cin >> manhattanMST::x[i] >> manhattanMST::y[i];

    manhattanMST::calc();
}
```

3 Numerical algorithms

3.1 SQRT For Loop

```
// Calculate n/1 + n/2 + ... + n/n

#define Cal(a, b) ((b) - (a) + 1)

ll calc(ll n)
{
    ll ans = 0;

    for (ll i = 1; i <= n / i; ++i)
        ans += Cal(n / (i + 1) + 1, n / i) * i;

    for (ll i = 1; i < n / i; ++i)
        ans += n / i;

    return ans;
}
```

3.2 Rabin Miller - Prime Checker

```cpp
// There is another version of Rabin Miller using random in the implementation of Pollard Rho

ll mul(ll a, ll b, ll mod)
{
    a %= mod;
    b %= mod;
    ll q = (ld)a * b / mod;
    ll r = a * b - q * mod;
    return (r % mod + mod) % mod;
}
ll pow(ll a, ll n, ll m)
{
    ll result = 1;
    a %= m;
    while (n > 0)
    {
        if (n & 1)
            result = mul(result, a, m);
        n >>= 1;
        a = mul(a, a, m);
    }
    return result;
}
pair<ll, ll> factor(ll n)
{
    ll s = 0;
    while ((n & 1) == 0)
    {
        s++;
        n >>= 1;
    }
    return {s, n};
}
bool test(ll s, ll d, ll n, ll witness)
{
    if (n == witness)
        return true;
    ll p = pow(witness, d, n);
    if (p == 1)
        return true;
    for (; s > 0; s--)
    {
        if (p == n - 1)
            return true;
        p = mul(p, p, n);
    }
    return false;
}
bool miller(ll n)
{
    if (n < 2)
        return false;
    if ((n & 1) == 0)
        return n == 2;
    ll s, d;
    tie(s, d) = factor(n - 1);
    return test(s, d, n, 2) && test(s, d, n, 3) && test(s, d, n, 5) &&
           test(s, d, n, 7) && test(s, d, n, 11) && test(s, d, n, 13) &&
           test(s, d, n, 17) && test(s, d, n, 19) && test(s, d, n, 23);
}
```

3.3 Chinese Remain Theorem

```cpp
using ll = long long;
using ld = long double;
namespace CRT
{
    // b must contain distinct element
    // x % b_i = a_i
    // x % m == (a1 * m2 * m3 * ... * m_k) * [(m2 * m3 * ... * mk) ^ -1 mod m_1] + (a2 * m1 * m3 * ...
        * m_k) / [(m1 * m3 * ... * mk) ^ -1 mod m2] + ...
    // Call CRT(a, b, phi) [Default phi is empty]
    // return {r, m} that x % m == r

    // In case of overflow, use this function
    ll Mul(ll a, ll b, const ll &mod)
    {
        ll q = (ld)a * b / mod;
        ll r = a * b - q * mod;

        return (r % mod + mod) % mod;
    }

    ll Pow(ll a, ll b, const ll &mod)
    {
        ll ans(1);
        for (; b; b >>= 1)
        {
            if (b & 1)
                ans = Mul(ans, a, mod);
            a = Mul(a, a, mod);
        }

        return ans;
    }

    ll calPhi(ll n)
    {
        ll ans = 1;

        for (ll i = 2; i * i <= n; ++i)
            if (n % i == 0)
            {
                while (n % i == 0)
                {
                    n /= i;
                    ans *= i;
                }

                ans = ans / i * (i - 1);
            }

        if (n != 1)
            ans *= n - 1;

        return ans;
    }

    pair<ll, ll> solve(const vector<ll> &a, const vector<ll> &b, vector<ll> phi = {})
    {
        assert(a.size() == b.size()); // Assume a and b have the same size
        ll m = 1;

        {
            m = 1;
            for (auto i : b)
                m *= i;
        }

        if (phi.empty())
        {
            for (auto i : b)
                phi.emplace_back(calPhi(i));
        }

        ll r = 0;

        for (int i = 0; i < (int)b.size(); ++i)
            r = (r + Mul(Mul(a[i], m / b[i], m), Pow(m / b[i], phi[i] - 1, m), m)) % m;

        return make_pair(r, m);
    }
};
```

3.4 Pollard Rho - Factorialize

```cpp
// You can change code Rabin-Miller (preposition)
struct PollardRho
{
    ll n;
    map<ll, int> ans;
    PollardRho(ll n) : n(n) {}
    ll random(ll u)
    {
        return abs(rand()) % u;
    }
    ll mul(ll a, ll b, ll mod)
    {
        a %= mod;
        b %= mod;
        ll q = (ld)a * b / mod;
        ll r = a * b - q * mod;
        return (r % mod + mod) % mod;
    }

    ll pow(ll a, ll b, ll m)
    {
        ll ans = 1;
        a %= m;
        for (; b; b >>= 1)
```

```
    {
        if (b & 1)
            ans = mul(ans, a, m);
        a = mul(a, a, m);
    }
    return ans;
}

pair<ll, ll> factor(ll n)
{
    ll s = 0;
    while ((n & 1) == 0)
    {
        s++;
        n >>= 1;
    }
    return {s, n};
}
// Rabin - Miller
bool miller(ll n)
{
    if (n < 2)
        return 0;
    if (n == 2)
        return 1;
    ll s = 0, m = n - 1;
    while (m % 2 == 0)
    {
        s++;
        m >>= 1;
    }
    // 1 - 0.9 ^ 40
    for (int it = 1; it <= 40; it++)
    {
        ll u = random(n - 2) + 2;
        ll f = pow(u, m, n);
        if (f == 1 || f == n - 1)
            continue;
        for (int i = 1; i < s; i++)
        {
            f = mul(f, f, n);
            if (f == 1)
                return 0;
            if (f == n - 1)
                break;
        }
        if (f != n - 1)
            return 0;
    }
    return 1;
}

ll f(ll x, ll n)
{
    return (mul(x, x, n) + 1) % n;
}
// Find a factor
ll findfactor(ll n)
{
    ll x = random(n - 1) + 2;
    ll y = x;
    ll p = 1;
    while (p == 1)
    {
        x = f(x, n);
        y = f(f(y, n), n);
        p = __gcd(abs(x - y), n);
    }
    return p;
}
// prime factorization
void pollard_rho(ll n)
{
    if (n <= 1000000)
    {
        for (int i = 2; i * i <= n; i++)
        {
            while (n % i == 0)
            {
                ans[i]++;
                n /= i;
            }
        }
        if (n > 1)
            ans[n]++;
        return;
    }

    if (miller(n))
    {
        ans[n]++;
        return;
        }
        ll p = 0;

        while (p == 0 || p == n)
            p = findfactor(n);

        pollard_rho(n / p);
        pollard_rho(p);
    }
};
```

3.5 FFT

```
using cd = complex<double>;
const double PI = acos(-1);
// invert == true means Interpolation
// invert == false means dft
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;
    }
}
```

3.6 FFT (Mod 998244353)

```
constexpr int N = 1e5 + 5; // keep N double of n+m

// Call init() before call mul()

constexpr ll mod = 998244353;

ll Pow(ll a, ll b, ll mod)
{
    ll ans(1);
    for (; b; b >>= 1)
    {
        if (b & 1)
            ans = ans * a % mod;
        a = a * a % mod;
    }
    return ans;
}

namespace ntt
{
    const int N = ::N;
    const long long mod = ::mod, rt = 3;
    ll G[55], iG[55], itwo[55];
    void add(int &a, int b)
    {
        a += b;
```

```
        if (a >= mod)
            a -= mod;
    }
    void init()
    {
        int now = (mod - 1) / 2, len = 1, irt = Pow(rt, mod - 2, mod);
        while (now % 2 == 0)
        {
            G[len] = Pow(rt, now, mod);
            iG[len] = Pow(irt, now, mod);
            itwo[len] = Pow(1 << len, mod - 2, mod);
            now >>= 1;
            len++;
        }
    }
    void dft(ll *x, int n, int fg = 1) // fg=1 for dft, fg=-1 for inverse dft
    {
        for (int i = (n >> 1), j = 1, k; j < n; ++j)
        {
            if (i < j)
                swap(x[i], x[j]);
            for (k = (n >> 1); k & i; i ^= k, k >>= 1)
                ;
            i ^= k;
        }
        for (int m = 2, now = 1; m <= n; m <<= 1, now++)
        {
            ll r = fg > 0 ? G[now] : iG[now];
            for (int i = 0, j; i < n; i += m)
            {
                ll tr = 1, u, v;
                for (j = i; j < i + (m >> 1); ++j)
                {
                    u = x[j];
                    v = x[j + (m >> 1)] * tr % mod;
                    x[j] = (u + v) % mod;
                    x[j + (m >> 1)] = (u + mod - v) %
                                      mod;
                    tr = tr * r % mod;
                }
            }
        }
    }

    // Take two sequence a, b;
    // return answer in sequence a

    void mul(ll *a, ll *b, int n, int m)
    {
        // a: 0,1,2,...,n-1; b: 0,1,2,...,m-1

        int nn = n + m - 1;
        if (n == 0 || m == 0)
        {
            memset(a, 0, nn * sizeof(a[0]));
            return;
        }
        int L, len;
        for (L = 1, len = 0; L < nn; ++len, L <<= 1)
            ;
        if (n < L)
            memset(a + n, 0, (L - n) * sizeof(a[0]));
        if (m < L)
            memset(b + m, 0, (L - m) * sizeof(b[0]));
        dft(a, L, 1); // dft(a)
        dft(b, L, 1); // dft(b)
        // Merge
        for (int i = 0; i < L; ++i)
            a[i] = a[i] * b[i] % mod;
        // Interpolation
        dft(a, L, -1);
        for (int i = 0; i < L; ++i)
            a[i] = a[i] * itwo[len] % mod;
    }
};
```

3.7 Count Primes

```
// To initialize, call init_count_primes() first.
// Function count_primes(n) will compute the number of
// prime numbers lower than or equal to n.
//
// Time complexity: Around O(n ^ 0.75)

constexpr int N = 1e5 + 5; // keep N larger than max(sqrt(n) + 2)
bool prime[N];
int prec[N];
vector<int> P;

ll rec(ll n, int k)
{
    if (n <= 1 || k < 0)
        return 0;
    if (n <= P[k])
        return n - 1;

    if (n < N && ll(P[k]) * P[k] > n)
        return n - 1 - prec[n] + prec[P[k]];

    const int LIM = 250;
    static int memo[LIM * LIM][LIM];
    bool ok = n < LIM * LIM;
    if (ok && memo[n][k])
        return memo[n][k];

    ll ret = n / P[k] - rec(n / P[k], k - 1) + rec(n, k - 1);

    if (ok)
        memo[n][k] = ret;
    return ret;
}
void init_count_primes()
{
    prime[2] = true;
    for (int i = 3; i < N; i += 2)
        prime[i] = true;
    for (int i = 3, j; i * i < N; i += 2)
        if (prime[i])
            for (j = i * i; j < N; j += i + i)
                prime[j] = false;

    for (int i = 1; i < N; ++i)
        if (prime[i])
            P.push_back(i);

    for (int i = 1; i < N; ++i)
        prec[i] = prec[i - 1] + prime[i];
}

ll count_primes(ll n)
{
    if (n < N)
        return prec[n];
    int k = prec[(int)sqrt(n) + 1];
    return n - 1 - rec(n, k) + prec[P[k]];
}
```

3.8 Interpolation (Mod a prime)

```
// You can change mod into other prime number
// update k to the degree of polynomial
// Just work when we know a[1] = P(1), a[2] = P(2),..., a[k] = P(k) [The degree of P(x) is k-1]
// update() then build() then cal()

/*
 * Complexity: O(Nlog(mod), N)
 */

constexpr ll mod = 1e9 + 7; // Change mod here
constexpr ll N = 1e5 + 5;   // Change size here

struct Interpolation
{
    ll a[N], fac[N], ifac[N], prf[N], suf[N];
    int k;

    ll Pow(ll a, ll b)
    {
        ll ans(1);
        for (; b; b >>= 1)
        {
            if (b & 1)
                ans = ans * a % mod;
            a = a * a % mod;
        }

        return ans;
    }

    void upd(int u, ll v)
    {
        a[u] = v;
    }

    void build()
```

```
    {
        fac[0] = ifac[0] = 1;
        for (int i = 1; i < N; i++)
        {
            fac[i] = (long long)fac[i - 1] * i % mod;
            ifac[i] = Pow(fac[i], mod - 2);
        }
    }

    // Calculate P(x)
    ll calc(int x)
    {
        prf[0] = suf[k + 1] = 1;

        for (int i = 1; i <= k; i++)
            prf[i] = prf[i - 1] * (x - i + mod) % mod;

        for (int i = k; i >= 1; i--)
            suf[i] = suf[i + 1] * (x - i + mod) % mod;

        ll res = 0;

        for (int i = 1; i <= k; i++)
        {
            if (!((k - i) & 1))
                res = (res + prf[i - 1] * suf[i + 1] % mod * ifac[i - 1] % mod * ifac[k - i] % mod * a
                      [i]) % mod;
            else
                res = (res - prf[i - 1] * suf[i + 1] % mod * ifac[i - 1] % mod * ifac[k - i] % mod * a
                      [i] % mod + mod) % mod;
        }

        return res;
    }
};
```

3.9 Bignum

```
/// M is the number of digits in the answer
/// In case that we don't use multiplication, let BASE be 1e17 or 1e18
/// a = Bignum("5")
/// The operator / is only for integer, the result is integer too

using cd = complex<long double>;
const long double PI = acos(-1);
const int M = 2000;
const ll BASE = 1e8;
const int gd = log10(BASE);
const int maxn = M / gd + 1;
struct Bignum
{
    int n;
    ll a[maxn];
    Bignum(ll x = 0)
    {
        memset(a, 0, sizeof a);
        n = 0;
        do
        {
            a[n++] = x % BASE;
            x /= BASE;
        } while (x);
    }
    Bignum(const string &s)
    {
        Convert(s);
    }
    ll stoll(const string &s)
    {
        ll ans(0);
        for (auto i : s)
            ans = ans * 10 + i - '0';
        return ans;
    }
    void Convert(const string &s)
    {
        memset(a, 0, sizeof a);
        n = 0;
        for (int i = s.size() - 1; ~i; --i)
        {
            int j = max(0, i - gd + 1);
            a[n++] = stoll(s.substr(j, i - j + 1));
            i = j;
        }
        fix();
    }
    void fix()
    {
        ++n;
        for (int i = 0; i < n - 1; ++i)
        {
            a[i + 1] += a[i] / BASE;
            a[i] %= BASE;
            if (a[i] < 0)
            {
                a[i] += BASE;
                --a[i + 1];
            }
        }
        while (n > 1 && a[n - 1] == 0)
            --n;
    }
    Bignum &operator+=(const Bignum &x)
    {
        n = max(n, x.n);
        for (int i = 0; i < n; ++i)
            a[i] += x.a[i];
        fix();
        return *this;
    }
    Bignum &operator-=(const Bignum &x)
    {
        for (int i = 0; i < x.n; ++i)
            a[i] -= x.a[i];
        fix();
        return *this;
    }
    Bignum &operator*=(const Bignum &x)
    {
        vector<ll> c(x.n + n, 0);
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < x.n; ++j)
                c[i + j] += a[i] * x.a[j];
        n += x.n;
        for (int i = 0; i < n; ++i)
            a[i] = c[i];
        fix();
        return *this;
    }
    Bignum &operator/=(const ll &x)
    {
        ll r = 0ll;
        for (int i = n - 1; i > -1; --i)
        {
            r = r * BASE + a[i];
            a[i] = r / x;
            r %= x;
        }
        fix();
        return *this;
    }
    Bignum operator+(const Bignum &s)
    {
        Bignum c;
        copy(a, a + n, c.a);
        c.n = n;
        c += s;
        return c;
    }
    Bignum operator-(const Bignum &s)
    {
        Bignum c;
        copy(a, a + n, c.a);
        c.n = n;
        c -= s;
        return c;
    }
    Bignum operator*(const Bignum &s)
    {
        Bignum c;
        copy(a, a + n, c.a);
        c.n = n;
        c *= s;
        return c;
    }
    Bignum operator/(const ll &x)
    {
        Bignum c;
        copy(a, a + n, c.a);
        c.n = n;
        c /= x;
        return c;
    }
    ll operator%(const ll &x)
    {
        ll ans(0);
        for (int i = n - 1; ~i; --i)
            ans = (ans * BASE + a[i]) % x;
        return ans;
    }
```

```
    }
    int com(const Bignum &s) const
    {
        if (n < s.n)
            return 1;
        if (n > s.n)
            return 2;
        for (int i = n - 1; i > -1; --i)
            if (a[i] > s.a[i])
                return 2;
            else if (a[i] < s.a[i])
                return 1;
        return 3;
    }
    bool operator<(const Bignum &s) const
    {
        return com(s) == 1;
    }
    bool operator>(const Bignum &s) const
    {
        return com(s) == 2;
    }
    bool operator==(const Bignum &s) const
    {
        return com(s) == 3;
    }
    bool operator<=(const Bignum &s) const
    {
        return com(s) != 2;
    }
    bool operator>=(const Bignum &s) const
    {
        return com(s) != 1;
    }
    void read()
    {
        string s;
        cin >> s;
        Convert(s);
    }
    void print()
    {
        int i = n;
        while (i > 0 && a[i] == 0)
            --i;
        cout << a[i];
        for (--i; ~i; --i)
            cout << setw(gd) << setfill('0')
                 << a[i];
    }
};
```

3.10 Bignum with FFT multiplication

```
// Replace function *= in Bignum implementation with below code:
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;
    }
}
Bignum &operator*=(const Bignum &x)
{
    int m = 1;
    while (m < n + x.n)
        m <<= 1;
    vector<cd> fa(m), fb(m);
    for (int i = 0; i < m; ++i)
    {
        fa[i] = a[i];
        fb[i] = x.a[i];
    }
    fft(fa, false); /// dft
    fft(fb, false); /// dft
    for (int i = 0; i < m; i++)
        fa[i] *= fb[i];
    fft(fa, true); /// Interpolation
    n = m;
    for (int i = 0; i < n; ++i)
        a[i] = round(fa[i].real());
    fix();
    return *this;
}
```

3.11 Tonelli Shanks (Find square root modulo prime)

```
/*
Takes as input an odd prime p and n < p and returns r such that r * r = n [mod p].
There's exist r if and only if n ^ [(p-1) / 2] = 1 (mod p)
*/

using ll = int; // Change type of data here

ll Pow(ll a, ll b, ll mod)
{
    ll ans(1);

    for (; b; b >>= 1)
    {
        if (b & 1)
            ans = ans * a % mod;
        a = a * a % mod;
    }

    return ans;
}

ll tonelli_shanks(ll n, ll p)
{
    ll s = 0;
    ll q = p - 1;
    while ((q & 1) == 0)
    {
        q /= 2;
        ++s;
    }
    if (s == 1)
    {
        ll r = Pow(n, (p + 1) / 4, p);
        if ((r * r) % p == n)
            return r;
        return 0;
    }
    // Find the first quadratic non-residue z by brute-force search
    ll z = 1;
    while (Pow(++z, (p - 1) / 2, p) != p - 1)
        ;
    ll c = Pow(z, q, p);
    ll r = Pow(n, (q + 1) / 2, p);
    ll t = Pow(n, q, p);
    ll m = s;
    while (t != 1)
    {
        ll tt = t;
        ll i = 0;
        while (tt != 1)
        {
            tt = (tt * tt) % p;
            ++i;
            if (i == m)
                return 0;
        }
        ll b = Pow(c, Pow(2, m - i - 1, p - 1), p);
        ll b2 = (b * b) % p;
        r = (r * b) % p;
        t = (t * b2) % p;
```

```
        c = b2;
        m = i;
    }
    if ((r * r) % p == n)
        return r;
    return -1; // Can't find
}
```

3.12 Discrete Logarithm (Find x that $a^x \equiv b\ (mod\ m)$)

```
// Returns minimum x for which a ^ x % m = b % m.
// Returns -1 if x isn't exist

using ll = int;

ll DiscreteLogarithm(ll a, ll b, ll m)
{
    a %= m, b %= m;
    ll k = 1, add = 0, g;
    while ((g = __gcd(a, m)) > 1)
    {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    ll n = sqrt((ld)m) + 1;
    ll an = 1;
    for (ll i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<ll, ll> vals;
    for (ll q = 0, cur = b; q <= n; ++q)
    {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (ll p = 1, cur = k; p <= n; ++p)
    {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur))
        {
            ll ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

3.13 Primitive Root (Exist k that $g^k \equiv a\ (mod\ n)$ for all a)

```
/*
g is a primitive root modulo n if and only if for any integer a such that gcd(a, n) = 1, there exists
      an integer k such that:
g^k = a (mod n).

Primitive root modulo n exists if and only if:
    - n is 1, 2, 4
    - n is power of an odd prime number (n = p ^ k)
    - n is twice power of an odd prime number (n = 2 * p ^ k)

This theorem was proved by Gauss in 1801.
*/

using ll = int; // Change type of data here

ll Pow(ll a, ll b, ll mod)
{
    ll ans(1);

    for (; b; b >>= 1)
    {
        if (b & 1)
            ans = ans * a % mod;
        a = a * a % mod;
    }
    return ans;
}

ll GetPhi(ll n)
{
    ll ans(1);

    for (ll i = 2; i * i <= n; ++i)
        if (n % i == 0)
        {
            while (n % i == 0)
            {
                n /= i;
                ans *= i;
            }

            ans = ans / i * (i - 1);
        }

    if (n != 1)
        ans *= n - 1;

    return ans;
}

ll PrimitiveRoot(ll p)
{
    vector<ll> fact;
    ll phi = GetPhi(p);
    ll n = phi;

    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0)
        {
            fact.push_back(i);
            while (n % i == 0)
                n /= i;
        }

    if (n > 1)
        fact.push_back(n);

    for (ll res = 2; res <= p; ++res)
    {
        bool ok = true;

        for (int i = 0; i < fact.size() && ok; ++i)
            ok &= Pow(res, phi / fact[i], p) != 1;

        if (ok)
            return res;
    }

    return -1; // can't find
}
```

3.14 Discrete Root (Find x that $x^k \equiv a\ (mod\ n)$, n is a prime)

```
/*
Given a prime n and two integers a and k, find all x for which:\
    - x ^ k = a (mod n)

Notice:
    - In case k = 2, let's use Tonelli - Shanks
    - You must insert my implementation of Discrete Logarithm and Primitive Root to run algorithm
*/

ll Pow(ll a, ll b, ll mod)
{
    ll ans(1);

    for (; b; b >>= 1)
    {
        if (b & 1)
            ans = ans * a % mod;
        a = a * a % mod;
    }

    return ans;
}

ll DiscreteRoot(ll a, ll k, ll n)
{
    ll g = PrimitiveRoot(n);
    ll v = Pow(g, k, n);
    ll ans = DiscreteLogarithm(v, a, n);
```

```
    if (ans == -1)
        return -1; // Can't find

    return Pow(g, ans, n);
}
```

4 Graph algorithms

4.1 Twosat (2-SAT)

```
// start from 0
// pos(V) is the vertex that represent V in graph
// neg(V) is the vertex that represent !V
// pos(V) ^ neg(V) = 1, use two functions below
// (U v V) <=> (!U -> V) <=> (!V -> U)
// You need do addEge(represent(U), represent(V))
// solve() == false mean no answer
// Want to get the answer ?
// color[pos(U)] = 1 means we choose U
// otherwise, we don't
constexpr int N = 1e5 + 5; // Keep N double of n
inline int pos(int u) { return u << 1; }
inline int neg(int u) { return u << 1 | 1; }
struct TwoSAT
{
    int n, numComp, cntTarjan;
    vector<int> adj[N], stTarjan;
    int low[N], num[N], root[N], color[N];
    TwoSAT(int n) : n(n * 2)
    {
        memset(root, -1, sizeof root);
        memset(low, -1, sizeof low);
        memset(num, -1, sizeof num);
        memset(color, -1, sizeof color);
        cntTarjan = 0;
        stTarjan.clear();
    }
    void addEdge(int u, int v)
    {
        adj[u ^ 1].push_back(v);
        adj[v ^ 1].push_back(u);
    }
    void tarjan(int u)
    {
        stTarjan.push_back(u);
        num[u] = low[u] = cntTarjan++;
        for (int v : adj[u])
        {
            if (root[v] != -1)
                continue;
            if (low[v] == -1)
                tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        if (low[u] == num[u])
        {
            while (1)
            {
                int v = stTarjan.back();
                stTarjan.pop_back();
                root[v] = numComp;
                if (u == v)
                    break;
            }
            numComp++;
        }
    }
    bool solve()
    {
        for (int i = 0; i < n; i++)
            if (root[i] == -1)
                tarjan(i);
        for (int i = 0; i < n; i += 2)
        {
            if (root[i] == root[i ^ 1])
                return 0;
            color[i] = (root[i] < root[i ^ 1]);
        }
        return 1;
    }
};
```

4.2 Eulerian Path

```
// Path that goes all edges
// Start from 1
struct EulerianGraph
{
    vector<vector<pair<int, int>>> a;
    int num_edges;

    EulerianGraph(int n)
    {
        a.resize(n + 1);
        num_edges = 0;
    }

    void add_edge(int u, int v, bool undirected = true)
    {
        a[u].push_back(make_pair(v, num_edges));
        if (undirected)
            a[v].push_back(make_pair(u, num_edges));
        num_edges++;
    }

    vector<int> get_eulerian_path()
    {
        vector<int> path, s;
        vector<bool> was(num_edges);

        s.push_back(1);
        // start of eulerian path
        // directed graph: deg_out - deg_in == 1
        // undirected graph: odd degree
        // for eulerian cycle: any vertex is OK

        while (!s.empty())
        {
            int u = s.back();
            bool found = false;
            while (!a[u].empty())
            {
                int v = a[u].back().first;
                int e = a[u].back().second;
                a[u].pop_back();
                if (was[e])
                    continue;
                was[e] = true;
                s.push_back(v);
                found = true;
                break;
            }
            if (!found)
            {
                path.push_back(u);
                s.pop_back();
            }
        }
        reverse(path.begin(), path.end());
        return path;
    }
};
```

4.3 Biconnected Component Tree

```
// Biconnected Component Tree
// 1 is the root of Tree
// n + i is the node that represent i-th bcc, its depth is even

const int N = 3e5 + 5; // Change size to n + number of bcc (For safety, set N >= 2 * n)
int n, nBicon, nTime;
int low[N], num[N];
vector<int> adj[N], nadj[N];
vector<int> s;

void dfs(int v, int p = -1)
{
    low[v] = num[v] = ++nTime;
    s.emplace_back(v);

    for (auto i : adj[v])
        if (i != p)
        {
            if (!num[i])
            {
                dfs(i, v);
                low[v] = min(low[v], low[i]);
```

```
            if (low[i] >= num[v])
            {
                ++nBicon;
                nadj[v].emplace_back(n + nBicon);

                int vertex;
                do
                {
                    vertex = s.back();
                    s.pop_back();

                    nadj[n + nBicon].emplace_back(vertex);
                } while (vertex != i);
            }
        }
        else
            low[v] = min(low[v], num[i]);
    }
}
```

5 String

5.1 Palindrome Tree

```
// base on idea odd palindrome, even palindrome
// 0-odd is the root of tree
struct node
{
    int len;
    node *child[26], *sufflink;
    node()
    {
        len = 0;
        for (int i = 0; i < 26; ++i)
            child[i] = NULL;
        sufflink = NULL;
    }
};
struct PalindromeTree
{
    node odd, even;
    PalindromeTree()
    {
        odd.len = -1;
        odd.sufflink = &odd;
        even.len = 0;
        even.sufflink = &odd;
    }
    void Assign(string &s)
    {
        node *last = &even;
        for (int i = 0; i < (int)s.size(); ++i)
        {
            node *tmp = last;
            while (s[i - tmp->len - 1] != s[i])
                tmp = tmp->sufflink;
            if (tmp->child[s[i] - 'a'])
            {
                last = tmp->child[s[i] - 'a'];
                continue;
            }
            tmp->child[s[i] - 'a'] = new node;
            last = tmp->child[s[i] - 'a'];
            last->len = tmp->len + 2;
            if (last->len == 1)
            {
                last->sufflink = &even;
                continue;
            }
            tmp = tmp->sufflink;
            while (s[i - tmp->len - 1] != s[i])
                tmp = tmp->sufflink;
            last->sufflink = tmp->child[s[i] - 'a'];
        }
    }
};
```

5.2 Suffix Array

```
// string and array pos start from 0
// but array sa and lcp start from 1

constexpr int N = 3e5 + 5; // change size to size of string;

struct SuffixArray
{
    string s;
    int n, c[N], p[N], rp[N], lcp[N];

    //p[] : suffix array
    // lcp[]: lcp array

    void Assign(const string &x)
    {
        s = x;
        s.push_back('$'); // Change character here due to range of charater in string
        n = s.size();
        Build();
        s.pop_back();
        n = s.size();
    }

    void Build()
    {
        vector<int> pn(N), cn(N), cnt(N);

        for (int i = 0; i < n; ++i)
            ++cnt[s[i]];
        for (int i = 1; i <= 256; ++i)
            cnt[i] += cnt[i - 1];
        for (int i = 0; i < n; ++i)
            p[--cnt[s[i]]] = i;

        for (int i = 1; i < n; ++i)
            c[p[i]] = c[p[i - 1]] + (s[p[i]] != s[p[i - 1]]);

        int maxn = c[p[n - 1]];

        for (int i = 0; (1 << i) < n; ++i)
        {
            for (int j = 0; j < n; ++j)
                p[j] = ((p[j] - (1 << i)) % n + n) % n;
            for (int j = 0; j <= maxn; ++j)
                cnt[j] = 0;

            for (int j = 0; j < n; ++j)
                ++cnt[c[p[j]]];

            for (int j = 1; j <= maxn; ++j)
                cnt[j] += cnt[j - 1];

            for (int j = n - 1; ~j; --j)
                pn[--cnt[c[p[j]]]] = p[j];

            for (int j = 1; j < n; ++j)
                cn[pn[j]] = cn[pn[j - 1]] + (c[pn[j]] != c[pn[j - 1]] || c[(pn[j] + (1 << i)) % n] !=
                      c[(pn[j - 1] + (1 << i)) % n]);

            maxn = cn[pn[n - 1]];

            for (int j = 0; j < n; ++j)
            {
                p[j] = pn[j];
                c[j] = cn[j];
            }
        }
    }

    void BuildLCP()
    {
        for (int i = 1; i <= n; ++i)
            rp[p[i]] = i;
        for (int i = 0; i < n; ++i)
        {
            if (i)
                lcp[i] = max(lcp[i - 1] - 1, 0);
            if (rp[i] == n)
                continue;

            while (lcp[i] < n - i && lcp[i] < n - p[rp[i] + 1] && s[i + lcp[i]] == s[p[rp[i] + 1] +
                  lcp[i]])
                ++lcp[i];
        }
    }
} g;
```

5.3 Aho Corasick - Extended KMP

```cpp
constexpr int ALPHABET_SIZE = 26;
constexpr int firstCharacter = 'a';

struct Node
{
    Node *to[ALPHABET_SIZE];
    Node *suflink;
    int ending_length; // 0 if is not ending

    Node()
    {
        for (int i = 0; i < ALPHABET_SIZE; ++i)
            to[i] = NULL;
        suflink = NULL;
        ending_length = false;
    }
};

struct AhoCorasick
{
    Node *root;

    AhoCorasick()
    {
        root = new Node();
    }

    void add(const string &s)
    {
        Node *cur_node = root;

        for (char c : s)
        {
            int v = c - firstCharacter;

            if (!cur_node->to[v])
                cur_node->to[v] = new Node();

            cur_node = cur_node->to[v];
        }

        cur_node->ending_length = s.size();
    }

    // if a->to[v] == NULL
    // for convinient a->to[v] = the node x->to[v] that a match x and x->to[v] != NULL
    // root -> suflink = root

    void build()
    {
        queue<Node *> Q;
        root->suflink = root;
        Q.push(root);

        while (!Q.empty())
        {
            Node *par = Q.front();
            Q.pop();
            for (int c = 0; c < ALPHABET_SIZE; ++c)
            {
                if (par->to[c])
                {
                    par->to[c]->suflink = par == root ? root : par->suflink->to[c];
                    Q.push(par->to[c]);
                }
                else
                {
                    par->to[c] = par == root ? root : par->suflink->to[c];
                }
            }
        }
    }
};
```

5.4 Suffix Tree (Template by Tran Khoi Nguyen)

```cpp
struct tk
{
    map<ll, ll> nxt;
    ll par, f, len;
    ll link;
    tk(ll par = -1, ll f = 0, ll len = 0) : par(par), f(f), len(len)
    {
        nxt.clear();
        link = -1;
    }
};

struct Suffix_Tree
{
    vector<tk> st;
    ll node;
    ll dis;
    s
        ll n;
    vector<ll> s;
    void init()
    {
        st.clear();
        node = 0;
        dis = 0;
        st.emplace_back(-1, 0, base);
        n = 0;
    }

    void go_edge()
    {
        while (dis > st[st[node].nxt[s[n - dis]]].len)
        {
            node = st[node].nxt[s[n - dis]];
            dis -= st[node].len;
        }
    }

    void add_char(ll c)
    {
        ll last = 0;
        s.pb(c);
        n = s.size();
        dis++;
        while (dis > 0)
        {
            go_edge();
            ll edge = s[n - dis];
            ll &v = st[node].nxt[edge];
            ll t = s[st[v].f + dis - 1];
            if (v == 0)
            {
                v = st.size();
                st.emplace_back(node, n - dis, base);
                st[last].link = node;
                last = 0;
            }
            else if (c == t)
            {
                st[last].link = node;
                return;
            }
            else
            {
                ll u = st.size();
                st.emplace_back(node, st[v].f, dis - 1);
                st[u].nxt[c] = st.size();
                st.emplace_back(u, n - 1, base);
                st[u].nxt[t] = v;
                st[v].f += (dis - 1);
                st[v].len -= (dis - 1);
                v = u;
                st[last].link = u;
                last = u;
            }
            if (node == 0)
                dis--;
            else
                node = st[node].link;
        }
    }
};
```

6 Data structures

6.1 Fenwick Tree (With Walk on tree)

```cpp
// This is equivalent to calculating lower_bound on prefix sums array
// LOGN = log2(N)

struct FenwickTree
{
    int n, LOGN;
    ll a[N]; // BIT array
```

```
    FenwickTree()
    {
        memset(a, 0, sizeof a);
    }

    void Update(int p, ll v)
    {
        for (; p <= n; p += p & -p)
            a[p] += v;
    }

    ll Get(int p)
    {
        ll ans(0);

        for (; p; p -= p & -p)
            ans += a[p];

        return ans;
    }

    int search(ll v)
    {
        ll sum = 0;
        int pos = 0;
        for (int i = LOGN; i >= 0; i--)
        {
            if (pos + (1 << i) <= n && sum + a[pos + (1 << i)] < v)
            {
                sum += a[pos + (1 << i)];
                pos += (1 << i);
            }
        }
        return pos + 1;
        //+1 because pos will be position of largest value less than v
    }
};
```

6.2 Convex Hull Trick (Min)

```
// If you want to get maximum, sort coef (A) not decreasing and change B[line.back()] > B[i] into B[
    line.back()] < B[i]

struct ConvexHullTrick
{
    vector<ll> A, B;
    vector<int> line;
    vector<ld> point;
    ConvexHullTrick(int n = 0)
    {
        A.resize(n + 2, 0);
        B.resize(n + 2, 0);
        point.emplace_back(-Inf);
    }

    ld ff(int x, int y)
    {
        return (ld)1.0 * (B[y] - B[x]) / (A[x] - A[y]);
    }

    void Add(int i)
    {
        while ((int)line.size() > 1 || ((int)line.size() == 1 && A[line.back()] == A[i]))
        {
            if (A[line.back()] == A[i])
            {
                if (B[line.back()] > B[i])
                {
                    line.pop_back();
                    if (!line.empty())
                        point.pop_back();
                }
                else
                    break;
            }
            else
            {
                if (ff(i, line.back()) <= ff(i, line[line.size() - 2]))
                {
                    line.pop_back();
                    if (!line.empty())
                        point.pop_back();
                }
                else
                    break;
            }
        }
        if (line.empty() || A[line.back()] != A[i])
        {
            if (!line.empty())
                point.emplace_back(ff(line.back(), i));
            line.emplace_back(i);
        }
    }
    ll Get(int x)
    {
        int j = lower_bound(point.begin(), point.end(), x) - point.begin();
        return A[line[j - 1]] * x + B[line[j - 1]];
    }
};
```

6.3 Dynamic Convex Hull Trick (Min)

```
struct Line
{
    mutable ll k, m, p;
    bool operator<(const Line& o) const
    {
        if (k==o.k) return m>o.m;
        return k > o.k;
    }
    bool operator<(ll x) const
    {
        return p < x;
    }
};
struct LineContainer : multiset<Line, less<>>
{
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b)
    {
        return a / b - ((a ^ b) < 0 && a % b);
    }
    bool isect(iterator x, iterator y)
    {
        if (y == end())
            return x->p = inf, 0;
        if (x->k == y->k)
            x->p = x->m < y->m ? inf : -inf;
        else
            x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m)
    {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z))
            z = erase(z);
        if (x != begin() && isect(--x, y))
            isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x)
    {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

6.4 SPlay Tree

```
struct KNode
{
    int Value;
    int Size;
    KNode *P, *L, *R;
};
using QNode = KNode *;
KNode No_thing_here;
QNode nil = &No_thing_here, root;

void Link(QNode par, QNode child, bool Right)
{
    child->P = par;
    if (Right)
        par->R = child;
    else
```

```
        par->L = child;
}

void Update(QNode &a)
{
    a->Size = a->L->Size + a->R->Size + 1;
}

void Init()
{
    nil->Size = 0;
    nil->P = nil->L = nil->R = nil;
    root = nil;
    for (int i = 1; i <= n; ++i)
    {
        QNode cur = new KNode;
        cur->P = cur->L = cur->R = nil;
        cur->Value = i;
        Link(cur, root, false);
        root = cur;
        Update(root);
    }
}

void Rotate(QNode x)
{
    QNode y = x->P;
    QNode z = y->P;
    if (x == y->L)
    {
        Link(y, x->R, false);
        Link(x, y, true);
    }
    else
    {
        Link(y, x->L, true);
        Link(x, y, false);
    }
    Update(y);
    Update(x);
    x->P = nil;
    if (z != nil)
        Link(z, x, z->R == y);
}

void Up_to_Root(QNode x)
{
    while (1){
        QNode y = x->P;
        QNode z = y->P;
        if(y == nil)
            break;
        if(z != nil){
            if((x == y->L) == (y == z->L))
                Rotate(y);
            else
                Rotate(x);
        }
        Rotate(x);
    }
}

QNode The_kth(QNode x, int k)
{
    while (true)
    {
        if (x->L->Size == k - 1)
            return x;
        if (x->L->Size >= k)
            x = x->L;
        else
        {
            k -= x->L->Size + 1;
            x = x->R;
        }
    }
    return nil;
}

void Split(QNode x, int k, QNode &a, QNode &b)
{
    if (k == 0)
    {
        a = nil;
        b = x;
        return;
    }
    QNode cur = The_kth(x, k);
    Up_to_Root(cur);
    a = cur;
    b = a->R;
    a->R = nil;
    b->P = nil;
    Update(a);
}

QNode Join(QNode a, QNode b)
{
    if (a == nil)
        return b;
    while (a->R != nil)
        a = a->R;
    Up_to_Root(a);
    Link(a, b, true);
    Update(a);
    return a;
}

void Print(QNode &a)
{
    if (a->L != nil)
        Print(a->L);
    cout << (a->Value) << " ";
    if (a->R != nil)
        Print(a->R);
}
```