CP-Reference

Contents

1 Template

```
#include<bits/stdc++.h> //3alHady | ft. Reda , AbdoSa3d , Kareem

  using namespace std;

  #define all(v) v.begin(),v.end()
  #define ll long long
  #define endl "\n"

  void solve()
  {
  }

  signed
  main()
  {
    cin.tie(0)->sync_with_stdio(false);

     int t = 1;
      //cin >> t;
      while (t--)
         solve();

  }
```

2 Data Structures

2.1 Centroid Decomposition

```
class centroid_decomposition {
  vector<bool> centroidMarked;
  vector<int> size;

  void dfsSize(int node, int par) {
    size[node] = 1;
    for (int ch : adj[node])
      if (ch != par && !centroidMarked[ch]) {
        dfsSize(ch, node);
        size[node] += size[ch];
      }
  }

  int getCenter(int node, int par, int size_of_tree) {
    for (int ch : adj[node]) {
      if (ch == par || centroidMarked[ch]) continue;
      if (size[ch] * 2 > size_of_tree) return getCenter(ch, node, size_of_tree);
    }
    return node;
  }

  int getCentroid(int src) {
    dfsSize(src, -1);
    int centroid = getCenter(src, -1, size[src]);
    centroidMarked[centroid] = true;
    return centroid;
  }

  int decomposeTree(int root) {
    root = getCentroid(root);
    solve(root);
    for (int ch : adj[root]) {
      if (centroidMarked[ch]) continue;
      int centroid_of_subtree = decomposeTree(ch);
      // note: root and centroid_of_subtree probably not have a direct edge in
      // adj      // centroidTree[root].push_back(centroid_of_subtree);      //
↪  centroidParent[centroid_of_subtree] = root;    }
    return root;
  }

  void calc(int node, int par) {
    // TO-DO
    for (int ch : adj[node])
      if (ch != par && !centroidMarked[ch]) calc(ch, node);
  }

  void add(int node, int par) {
    // TO-DO
    for (int ch : adj[node])
      if (ch != par && !centroidMarked[ch]) add(ch, node);
  }

  void remove(int node, int par) {
    // TO-DO
    for (int ch : adj[node])
      if (ch != par && !centroidMarked[ch]) remove(ch, node);
  }

  void solve(int root) {
    // add root
    for (int ch : adj[root])
      if (!centroidMarked[ch]) {
        calc(ch, root);
        add(ch, root);
      }
    // TO-DO //remove root
    for (int ch : adj[root])
      if (!centroidMarked[ch]) remove(ch, root);
  }

 public:
  int n, root;
  vector<vector<int>> adj, centroidTree;
  vector<int> centroidParent;

  centroid_decomposition(vector<vector<int>> &adj) : adj(adj) {
    n = (int)adj.size() - 1;
    size = vector<int>(n + 1);
    centroidTree = vector<vector<int>>(n + 1);
    centroidParent = vector<int>(n + 1, -1);
    centroidMarked = vector<bool>(n + 1);
    root = decomposeTree(1);
  }
};
```

2.2 DSU

Minimal

```
struct DSU {
    std::vector<int> p;
    DSU(int n): p(n) { std::iota(p.begin(), p.end(), 0); }
    int find(int x) { return p[x]==x ? x : p[x]=find(p[x]); }
    void unite(int a, int b){ p[find(a)] = find(b); }
};
```

```
struct DSU {
    vector<int> rank, parent, size;
    vector<vector<int>> component;
    int forsets;

    DSU(int n) {
        size = rank = parent = vector<int>(n + 1, 1);
        component = vector<vector<int>>(n + 1);
        forsets = n;
        for (int i = 0; i <= n; i++) {
            parent[i] = i;
            component[i].push_back(i);
        }
    }

    int find_set(int v) {
        if (v == parent[v])
            return v;
        return parent[v] = find_set(parent[v]);
    }

    void link(int par, int node) {
        parent[node] = par;
        size[par] += size[node];
        for (const int &it: component[node])
            component[par].push_back(it);
        component[node].clear();
        if (rank[par] == rank[node])
            rank[par]++;
        forsets--;
    }

    bool union_sets(int v, int u) {
        v = find_set(v), u = find_set(u);
        if (v != u) {
            if (rank[v] < rank[u])
                swap(v, u);
            link(v, u);
        }
        return v != u;
    }

    bool same_set(int v, int u) {
        return find_set(v) == find_set(u);
    }
    int size_set(int v) {
        return size[find_set(v)];
    }
};
```

2.3 EETREE

```
/* @brief Palindromes in Dequeby adamant*/
#define PROBLEM "https://judge.yosupo.jp/problem/palindromes_in_deque"
#pragma GCC optimize("Ofast,unroll-loops")
#pragma GCC target("tune=native")
#include <bits/stdc++.h>
using namespace std;

template<int sigma = 26, char mch = 'a'>
struct eertree {
    eertree(size_t q) {
        q += 2;
        cnt = len = par = qlink = vector(q, 0);
        to.resize(q);
        link.resize(q);
        ranges::fill(link[0], 1);
        qlink[0] = 1;
        len[1] = -1;
    }

    template<bool back = 1>
    static int get(auto const &d, size_t idx) {
        if (idx >= size(d)) { return -1; } else { return back ? rbegin(d)[idx] : d[idx]; }
    }

    template<bool back = 1>
    static void push(auto &d, auto c) { back ? d.push_back(c) : d.push_front(c); }

    template<bool back = 1>
    static void pop(auto &d) { back ? d.pop_back() : d.pop_front(); }

    template<bool back = 1>
    void add_letter(char c) {
        c -= mch;
        push<back>(s, c);
        int pre = get<back>(states, 0);
        int last = make_to<back>(pre, c);
        active += !(cnt[last]++);
        int D = 2 + len[pre] - len[last];
        while (D + len[pre] <= len[last]) {
            pop<back>(states);
            if (!empty(states)) {
                pre = get<back>(states, 0);
                D += get<back>(diffs, 0);
                pop<back>(diffs);
            } else { break; }
        }
        if (!empty(states)) { push<back>(diffs, D); }
        push<back>(states, last);
    }

    template<bool back = 1>
    void pop_letter() {
        int last = get<back>(states, 0);
        active -= !(--cnt[last]);
        pop<back>(states);
        pop<back>(s);
        array cands = {pair{qlink[last], len[last] - len[qlink[last]]}, pair{par[last],
↪   0}};
        for (auto [state, diff]: cands) {
            if (empty(states)) {
                states = {state};
                diffs = {diff};
            } else {
                int D = get<back>(diffs, 0) - diff;
                int pre = get<back>(states, 0);
                if (D + len[state] > len[pre]) {
                    push<back>(states, state);
                    pop<back>(diffs);
                    push<back>(diffs, D);
                    push<back>(diffs, diff);
                }
            }
        }
        pop<back>(diffs);
    }

    void add_letter(char c, bool back) { back ? add_letter<1>(c) : add_letter<0>(c); }
    void pop_letter(bool back) { back ? pop_letter<1>() : pop_letter<0>(); }
    int distinct() { return active; }

    template<bool back = 1>
    int maxlen() { return len[get<back>(states, 0)]; }

private:
    vector<array<int, sigma> > to, link;
    vector<int> len, qlink, par, cnt;
    deque<char> s;
    deque<int> states = {0}, diffs;
    int sz = 2, active = 0;

    template<bool back = 1>
    int make_to(int last, int c) {
        if (c != get<back>(s, len[last] + 1)) { last = link[last][c]; }
        if (!to[last][c]) {
            int u = to[link[last][c]][c];
            qlink[sz] = u;
            link[sz] = link[u];
            link[sz][get<back>(s, len[u])] = u;
            len[sz] = len[last] + 2;
            par[sz] = last;
            to[last][c] = sz++;
        }
        return to[last][c];
    }
};

void solve() {
    int q;
    cin >> q;
    eertree me(q);
    for (int i = 0; i < q; i++) {
        int t;
        cin >> t;
        if (t / 2) { me.pop_letter(t % 2); } else {
            char c;
            cin >> c;
            me.add_letter(c, t % 2);
        }
        cout << me.distinct() << ' ' << me.maxlen<0>() << ' ' << me.maxlen<1>() << "\n";
    }
}

signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int t = 1;
    while (t--) { solve(); }
}
```

2.4 Fenwick 2d

```
template <typename T>
class FenwickTree2D {
 public:
  vector<vector<T>> tree;
  int n, m;
  void init(int n, int m) {
    tree.assign(n + 2, vector<T>(m + 2, 0));
    this->n = n;
    this->m = m;
  }
  T merge(T &x, T &y) { return x + y; }
  void update(int x, int y, T val) {
    for (; x <= n; x += x & -x) {
      for (int z = y; z <= m; z += z & -z) {
        tree[x][z] = merge(tree[x][z], val);
      }
    }
  }
  T getPrefix(int x, int y) {
    if (x <= 0) return 0;
    T ret = 0;
    for (; x; x -= x & -x) {
      for (int z = y; z; z -= z & -z) {
        ret = merge(ret, tree[x][z]);
      }
    }
    return ret;
  }
  T getSquare(int xl, int yl, int xr, int yr) {
    return getPrefix(xr, yr) + getPrefix(xl - 1, yl - 1) -
           getPrefix(xr, yl - 1) - getPrefix(xl - 1, yr);
  }
};
```

2.5 Fenwick

```
struct fenwik_tree
{
    int n;
    vector<int> fen;
    fenwik_tree(int _n)
    {
       fen = vector<int>(_n + 1);
       n = _n;
    }
    int sum(int p)
    {
       int s = 0;
       while (p >= 1)s += fen[p], p -= p & -p;
       return s;
    }
    int sum(int l, int r)
    {
       return sum(r) - (l > 1 ? sum(l - 1) : 0);
    }
    void add(int p, int x)
    {
       while (p <= n)fen[p] += x, p += p & -p;
    }
};
```

2.6 LCA o1 minimal

```
vector<int> dep(n), in(n), out(n), seq, pa(n, -1);

  auto dfs = [&](auto &&self, int u) -> void {
  in[u] = seq.size();
  seq.push_back(u);
  for (auto v : G[u])
    if (v != pa[u]) {
    pa[v] = u, dep[v] = dep[u] + 1;
    self(self, v);
    }
  out[u] = seq.size();
  };
  seq.reserve(n);
  dfs(dfs, 0);
  const int M = __lg(n);
  vector dp(M + 1, vector<int>(n));
  auto cmp = [&](int a, int b) { return dep[a] < dep[b] ? a : b; };

  dp[0] = seq;
  for (int i = 0; i < M; i++)
  for (int j = 0; j + (2 << i) <= n; j++)
    dp[i + 1][j] = cmp(dp[i][j], dp[i][j + (1 << i)]);

  auto lca = [&](int a, int b) {
  if (a == b)return a;
  a = in[a] + 1, b = in[b] + 1;
  if (a > b)swap(a, b);
  int h = __lg(b - a);
  return pa[cmp(dp[h][a], dp[h][b - (1 << h)])];
  };
```

2.7 LCA

```
int const N = 1e5 + 5, M = 20;
int dp[N][M + 1];
int lvl[N], n;
vector <vector<int>> adj;

void dfs(int u, int par) {
    dp[u][0] = par;
    for (auto i : adj[u]) {
        if (i != par) {
            lvl[i] = lvl[u] + 1;
            dfs(i, u);
        }
    }
}

void build() {
    dfs(1, -1);
    for (int i = 1; i <= M; i++) {
        for (int j = 1; j <= n; j++) {
            int u = dp[j][i - 1];
            if (u == -1)dp[j][i] = -1;
            else dp[j][i] = dp[u][i - 1];
        }
    }
}

int lca(int u, int v) {
    if (lvl[u] > lvl[v])swap(u, v);
    for (int i = M; i >= 0; i--) {
        if (lvl[v] - (1 << i) >= lvl[u])
            v = dp[v][i];
    }
    if (u == v)return v;
    for (int i = M; i >= 0; i--) {
        int cu = dp[u][i], cv = dp[v][i];
    if (min(cu, cv) != -1 && cu != cv)
        u = cu, v = cv;
    }
    return dp[u][0];
}

int shortestPath(int u, int v) {
    return lvl[u] + lvl[v] - 2 * lvl[lca(u, v)];
}
```

2.8 Mo (Saad)

```
const int N = 1e5 + 5;
vector<int> v(N);
int root;
struct query {
  int l, r, idx;
  bool operator<(query& oth) const {
    if (l / root == oth.l / root) return r < oth.r;
    return l < oth.l;
  }
};
vector<query> ask;
vector<ll> ans(N);
ll cur;
int n, q, l = 0, r = -1;
struct MO {
  void add(int idx) {}
  void remove(int idx) {}
  void change(int idx) {
    while (r < ask[idx].r) {
      r++;
      add(r);
    }
    while (r > ask[idx].r) {
      remove(r);
      r--;
    }
    while (l > ask[idx].l) {
      l--;
      add(l);
    }
    while (l < ask[idx].l) {
      remove(l);
      l++;
    }
    ans[ask[idx].idx] = cur;
  }
};
void solve() {
  cin >> n >> q;
  ask = vector<query>(q);
  root = sqrt(q) + 1;
  for (int i = 0; i < n; i++) cin >> v[i];
  for (int i = 0; i < q; i++)
    cin >> ask[i].l >> ask[i].r, --ask[i].l, --ask[i].r, ask[i].idx = i;
  sort(all(ask));
  MO mo;
  for (int i = 0; i < q; i++) mo.change(i);
  for (int i = 0; i < q; i++) cout << ans[i] << endl;
}
```

2.9 Mo Algorithm

```
int sqrtN;//use a constant value
struct query {
    int l, r, qindx, block;
    query() {}
    query(int l, int r, int qindx) : l(l), r(r), qindx(qindx), block(l / sqrtN) {}
    bool operator <(const query& q) {
        if (block != q.block)return block < q.block;
        return (block & 2 == 0 ? r<q.r : r>q.r);
    }

};
vector<query>q;
int curL, curR, ans;
void add(int indx);
void remove(int indx);
void solve(int l, int r) {
    while (curL > l)add(--curL);
    while (curR < r)add(++curR);
    while (curL < l)remove(curL++);
    while (curR > r)remove(curR--);
}
vector<int> MO() {
    vector<int>rt(q.size());
    ans = 0;
    curL = 1; curR = 0;
    add(0);
    sort(q.begin(), q.end());
    for (auto it : q) {
        solve(it.l, it.r);
        rt[it.qindx] = ans;
    }
    return rt;

}
```

2.10 Mo on trees

```
const int N = 1e5 + 5,lvls = 18;
vector<vector<int>>v;
int n,timer,root,q,l,r,seq[2*N],dp[N][lvls+1];
vector<int>depth,start,endd,ans,freqNode;
int lca(int x, int y)
{
    if (depth[x] < depth[y])swap(x, y);
    for (int k = lvls;k >= 0;k--)
    {
        if (depth[x] - (1 << k) >= depth[y])x = dp[x][k];
    }
    if (x == y)return x;
    for (int k = lvls;k >= 0;k--)
        if (dp[x][k] != dp[y][k])x = dp[x][k], y = dp[y][k];
    return dp[x][0];
}
void dfs(int node, int par)
{
    start[node] = timer;
    seq[timer] = node;
    timer++;
    dp[node][0] = par;
    depth[node] = depth[par] + 1;
    for (auto it : v[node])
    {
        if (it == par)continue;
        dfs(it, node);
    }
    endd[node] = timer;
    seq[timer] = node;
    timer++;
}
void process()
{
    for (int k = 1;k <= lvls;k++)
    {
        for (int j = 0;j < n;j++)
        {
            if (dp[j][k - 1] == -1)continue;
            dp[j][k] = dp[dp[j][k - 1]][k - 1];
        }
    }
}
struct query
{
    int l, r, idx,lc;
    bool operator<(query& oth)const {
        if (l / root == oth.l / root)
            return r < oth.r;
        return l < oth.l;
    }
};
vector<query>ask;

void add(int idx)
{
    freqNode[seq[idx]]^=1;
    if (freqNode[seq[idx]] & 1);
    else ;
}
```

```
void remove(int idx)
{
    freqNode[seq[idx]]^=1;
    if (freqNode[seq[idx]] & 1);
    else;
}
void change(int idx)
{
    while (r < ask[idx].r)
    {
        r++;
        add(r);
    }
    while (l > ask[idx].l)
    {
        l--;
        add(l);
    }

    while (r > ask[idx].r)
    {
        remove(r);
        r--;
    }
    while (l < ask[idx].l)
    {
        remove(l);
        l++;
    }
    ans[ask[idx].idx];
    if(~ask[idx].lc);
}

void solve()
{
    timer = 0;
    l = 0, r = -1;
    cin>>n>>q;
    v=vector<vector<int>>(n);
    freqNode=start=endd=depth=vector<int>(n+5);
    ans=vector<int>(q);
    ask = vector<query>(q);
    root = sqrt(2*n) + 5;
    for (int i = 1;i < n;i++)
    {
        int x, y;
        cin >> x >> y;
        --x, --y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
    dfs(0, n);
    process();
    for (int i = 0;i < q;i++)
    {
        int x, y;
        cin >> x >> y;
        --x, --y;
        int lc = lca(x, y);
        if (lc == x || lc == y)
        {
            ask[i].l = start[x];
            ask[i].r = start[y];
            ask[i].lc=-1;
        }
        else
        {
            if (start[x] > start[y])swap(x, y);
            ask[i].l = endd[x];
            ask[i].r = start[y];
            ask[i].lc=lc;
        }
        if (ask[i].l > ask[i].r)
            swap(ask[i].l, ask[i].r);
        ask[i].idx = i;
    }
    sort(all(ask));
    for (int i = 0;i < q;i++)change(i);
}
```

2.11 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<typename key>
using ordered_set = tree<key, null_type, less<key>, rb_tree_tag,
↪  tree_order_statistics_node_update>;

// order_of_key(k) : It returns to the number of items that are strictly smaller than
↪  our item k in O(logn) tme.
// find_by_order(k): It returns to an iterator to the kth element (counting from zero)
↪  in the set in O(logn) tme.
// To find the first element k must be zero.
```

2.12 Saad HLD

```
const int N = 1e4 + 5;
int sz[N], head[N];
vector<int>heavy(N, -1);
vector<int>seq, idx(N);
vector<vector<pair<int, int>>>v(N);
int val[N];
int depth[N];
int n;
template<typename T>
class segment_tree {//1-based
#define LEFT (idx<<1)
```

```
#define RIGHT (idx<<1|1)
#define MID ((start+end)>>1)
    int n;
    vector<T> tree;
    vector<T> lazy;

    T merge(const T& left, const T& right) {
        return max(left, right);
    }

    inline void pushup(int idx) {
        tree[idx] = merge(tree[LEFT], tree[RIGHT]);
    }

    void build(int idx, int start, int end) {
        if (start == end)
            return;
        build(LEFT, start, MID);
        build(RIGHT, MID + 1, end);
        pushup(idx);
    }

    void build(int idx, int start, int end, const vector<T>& arr) {
        if (start == end) {
            tree[idx] = arr[start];
            return;
        }
        build(LEFT, start, MID, arr);
        build(RIGHT, MID + 1, end, arr);
        pushup(idx);
    }

    T query(int idx, int start, int end, int from, int to) {
        if (from <= start && end <= to)
            return tree[idx];
        if (to<start || from>end)return 0;
        return merge(query(LEFT, start, MID, from, to), query(RIGHT, MID + 1, end, from, to));

    }

    void update(int idx, int start, int end, int lq, int rq,
        const T& val) {
        if (rq < start || end < lq)
            return;
        if (lq <= start && end <= rq) {
            tree[idx] = val;
            return;
        }

        update(LEFT, start, MID, lq, rq, val);
        update(RIGHT, MID + 1, end, lq, rq, val);
        pushup(idx);
    }
```

```
public:
    segment_tree(int n) : n(n), tree(n << 2), lazy(n << 2) {
    }

    segment_tree(const vector<T>& v) {
        n = v.size() - 1;
        tree = vector<T>(n << 2);
        lazy = vector<T>(n << 2);
        build(1, 1, n, v);
    }

    T query(int l, int r) {
        return query(1, 1, n, l, r);
    }

    void update(int l, int r, const T& val) {
        update(1, 1, n, l, r, val);
    }

#undef LEFT
#undef RIGHT
#undef MID
};
void dfsSz(int node, int par)
{
    int mx = 0;
    sz[node] = 1;
    for (auto it : v[node])
    {
        if (it.first == par)continue;
        dfsSz(it.first, node);
        val[it.first] = it.second;

        if (sz[it.first] > mx)mx = sz[it.first], heavy[node] = it.first;
        sz[node] += sz[it.first];
    }
}
int p[N];
void dfs(int node, int par, int h)
{
    p[node] = par;
    head[node] = h;
    idx[node] = seq.size();
    seq.push_back(node);

    depth[node] = depth[par] + 1;
    if (~heavy[node])
        dfs(heavy[node], node, h);
    for (auto it : v[node])
    {
        if (it.first == par || heavy[node] == it.first)continue;
        dfs(it.first, node, it.first);
    }
```

```
}
segment_tree<int> st(0);
int query(int a, int b) {
    int res = 0;
    for (; head[a] != head[b]; b = p[head[b]]) {
        if (depth[head[a]] > depth[head[b]])
            swap(a, b);
        int cur_heavy_path_max = st.query(idx[head[b]], idx[b]);
        res = max(res, cur_heavy_path_max);
    }
    if (depth[a] > depth[b])
        swap(a, b);
    if (a != b)
    {
        int last_heavy_path_max = st.query(idx[a] + 1, idx[b]);
        res = max(res, last_heavy_path_max);
    }
    return res;
}
void solve()
{
    seq.push_back(-1);
    cin >> n;
    vector<pair<int, int>>edges(n);
    heavy = vector<int>(n + 1, -1);
    for (int i = 1;i < n;i++)
    {
        int x, y, c;
        cin >> x >> y >> c;
        edges[i] = { x,y };
        v[x].push_back({ y,c });
        v[y].push_back({ x,c });
    }
    dfsSz(1, -1);
    dfs(1, N - 1, 1);
    st = segment_tree<int>(n+1);
    for (int i = 2;i <= n;i++)
        st.update(idx[i], idx[i], val[i]);
    while (true)
    {
        string q;
        cin >> q;
        if (q == "DONE")
        {
            seq.clear();
            for (int i = 1;i <= n;i++)
                v[i].clear();
            return;
        }
        if (q == "CHANGE")
        {
            int id, val;
            cin >> id >> val;
            int a = edges[id].first;
            int b = edges[id].second;
            if (p[a] == b)swap(a, b);
            st.update(idx[b], idx[b], val);

        }
        else
        {
            int a, b;
            cin >> a >> b;
            cout << query(a, b)<< endl;
        }
    }

}
```

2.13 Sack

```
vector<int> *vec[maxn];
int cnt[maxn];
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
           mx = sz[u], bigChild = u;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           dfs(u, v, 0);
    if(bigChild != -1)
        dfs(bigChild, v, 1), vec[v] = vec[bigChild];
    else
        vec[v] = new vector<int> ();
    vec[v]->push_back(v);
    cnt[ col[v] ]++;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           for(auto x : *vec[u]){
               cnt[ col[x] ]++;
               vec[v] -> push_back(x);
           }
    //now cnt[c] is the number of vertices in subtree of vertex v that has color c.
    // note that in this step *vec[v] contains all of the subtree of vertex v.
    if(keep == 0)
        for(auto u : *vec[v])
            cnt[ col[u] ]--;
}


int cnt[maxn];
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
          mx = sz[u], bigChild = u;
    for(auto u : g[v])
```

```
        if(u != p && u != bigChild)
            dfs(u, v, 0);  // run a dfs on small childs and clear them from cnt
    if(bigChild != -1)
        dfs(bigChild, v, 1);  // bigChild marked as big and not cleared from cnt
    for(auto u : g[v])
  if(u != p && u != bigChild)
      for(int p = st[u]; p < ft[u]; p++)
    cnt[ col[ ver[p] ] ]++;
    cnt[ col[v] ]++;
    //now cnt[c] is the number of vertices in subtree of vertex v that has color c. You can answer the queries easily.
    if(keep == 0)
        for(int p = st[v]; p < ft[v]; p++)
      cnt[ col[ ver[p] ] ]--;
}
```

2.14 SparseTable

```
template<typename T>
struct sparseTable {
    vector<vector<T>> table;
    vector<int> lg;
    int n, maxLog;
    sparseTable(vector<ll> &v) {
        n = v.size();
        lg.resize(n + 1);
        for (int i = 2; i <= n; ++i)lg[i] = lg[i / 2] + 1;
        maxLog = lg[n] + 1;
        table.resize(n, vector<T>(maxLog));
        for (int i = 0; i < n; i++) {
            table[i][0] = v[i];
        }
        for (int j = 1; j < maxLog; j++) {
            for (int i = 0; i <= n - (1 << j); i++) {
                table[i][j] = min(table[i][j - 1], table[i + (1 << (j - 1))][j - 1]);
            }
        }
    }
    T query(int l, int r) {
        assert(l <= r);
        int j = lg[r - l + 1];
        return min(table[l][j], table[r - (1 << j) + 1][j]);
    }
    T query2(int l, int r) {
        T mn = 2e9;
        for (int i = maxLog; i >= 0; i--) {
            if ((1 << i) <= r - l + 1) {
                mn = min(mn, table[l][i]);
                l += 1 << i;
            }
        }
        return mn;
    }
};
```

2.15 Sqrt Decomposition

```
int root = (int)sqrt(n + .0) + 1;
vector<int> v(n), bucket(root);
for (int i = 0; i < n; ++i)
    bucket[i / root] += v[i];

while (q--) {
    int l, r;
    int sum = 0;
    int c_l = l / root, c_r = r / root;
    if (c_l == c_r)
        for (int i = l; i <= r; ++i)
            sum += v[i];
    else {
        for (int i = l, end = (c_l + 1) * root - 1; i <= end; ++i)
            sum += v[i];
        for (int i = c_l + 1; i <= c_r - 1; ++i)
            sum += bucket[i];
        for (int i = c_r * root; i <= r; ++i)
            sum += v[i];
    }
}
```

2.16 Treap

```
enum DIR
{
  L, R
};

template<typename T>
struct cartesian_tree
{
  static cartesian_tree<T>* sentinel;
  T key;
  int priority = 0, size = 0;
  bool reverse = false;
  cartesian_tree* child[2];
  cartesian_tree* parent;
  cartesian_tree()
  {
    key = T();
    priority = 0;
    child[L] = child[R] = parent = this;
  }
  cartesian_tree(const T& x, int y) :
    key(x), priority(y)
  {
    size = 1;
    child[L] = child[R] = sentinel;
    parent = sentinel;
  }
  void push_down()
  {
    if (!reverse)
      return;
    reverse = 0;
    child[L]->doReverse();
    child[R]->doReverse();
  }
  void doReverse()
  {
    reverse ^= 1;
    swap(child[L], child[R]);
  }
  void push_up()
  {
    if (child[L] != sentinel)child[L]->parent = this;
    if (child[R] != sentinel)child[R]->parent = this;
    size = child[L]->size + child[R]->size + 1;
  }

};

template<typename T>
cartesian_tree<T>* cartesian_tree<T>::sentinel = new cartesian_tree<T>();

template<typename T, template<typename> class cartesian_tree>
class implicit_treap
{ //1 based
  typedef cartesian_tree<T> node;
  typedef cartesian_tree<T>* nodeptr;
#define emptyNode cartesian_tree<T>::sentinel
  nodeptr root;

  void split(nodeptr root, nodeptr& l, nodeptr& r, int firstXElment)
  {
    if (root == emptyNode)
    {
      l = r = emptyNode;
      return;
    }
    root->push_down();
    if (firstXElment <= root->child[L]->size)
    {
      split(root->child[L], l, root->child[L], firstXElment);
      r = root;
    }
    else
    {
      split(root->child[R], root->child[R], r,
        firstXElment - root->child[L]->size - 1);
      l = root;
    }
    root->push_up();
  }
  nodeptr merge(nodeptr l, nodeptr r)
  {
    l->push_down();
    r->push_down();
    if (l == emptyNode || r == emptyNode)
      return (l == emptyNode ? r : l);
    if (l->priority > r->priority)
    {
      l->child[R] = merge(l->child[R], r);
      l->push_up();
      return l;
    }
    r->child[L] = merge(l, r->child[L]);
    r->push_up();
    return r;
  }
  vector<nodeptr> split_range(int s, int e)
  { // [x<s,s<=x<=e,e<x]
    nodeptr l, m, r, tmp;
    split(root, l, tmp, s - 1);
    split(tmp, m, r, e - s + 1);
    return { l, m, r };
  }
  map<T, nodeptr> mp;
 public:
  implicit_treap() :
    root(emptyNode)
  {
  }
  int size()
  {
    return root->size;
  }
  void insert(int pos, const T& key)
  {

    nodeptr tmp = new node(key, rand());
    nodeptr l, r;
    split(root, l, r, pos - 1);
    root = merge(merge(l, tmp), r);
  }
  void push_back(const T& value)
  {
    nodeptr tmp = new node(value, rand());
    mp[value] = tmp;
    root = merge(root, tmp);
  }
  T getByIndex(int pos)
  {

    vector<nodeptr> tmp = split_range(pos, pos);
    nodeptr l = tmp[0], m = tmp[1], r = tmp[2];
    T rt = m->key;
    root = merge(merge(l, m), r);
    return rt;
  }
  void erase(int pos)
  {
    vector<nodeptr> tmp = split_range(pos, pos);
    nodeptr l = tmp[0], m = tmp[1], r = tmp[2];
    delete m;
    root = merge(l, r);
  }
  void cyclic_shift(int s, int e)
  { //to the right
    vector<nodeptr> tmp = split_range(s, e);
    nodeptr l = tmp[0], m = tmp[1], r = tmp[2];
    nodeptr first, second;
    split(m, first, second, e - s);
    root = merge(merge(merge(l, second), first), r);
  }
  void reverse_range(int s, int e)
  {
    vector<nodeptr> tmp = split_range(s, e);
    nodeptr l = tmp[0], m = tmp[1], r = tmp[2];
    m->doReverse();
    root = merge(merge(l, m), r);
  }
  node range_query(int s, int e)
  {
    vector<nodeptr> tmp = split_range(s, e);
    nodeptr l = tmp[0], m = tmp[1], r = tmp[2];
    node rt = *m;
    root = merge(merge(l, m), r);
  }
  int getIndexByValue(const T& value)
  {
    nodeptr cur = mp[value];

    vector<nodeptr> path;
    path.push_back(cur);
    while (cur != root)
    {
      cur = cur->parent;
      path.push_back(cur);
    }

    for (int i = path.size() - 1; i >= 0; i--)
      path[i]->push_down();
    for (auto& it : path)
      it->push_up();

    cur = mp[value];
    int cnt = cur->child[L]->size + 1;
    while (cur != root)
    {
      nodeptr par = cur->parent;
      if (par->child[R] == cur)cnt += par->child[L]->size + 1;
```

```
      cur = par;
    }
    return cnt;
  }
};
implicit_treap<int, cartesian_tree> tr;
```

2.17 Two SAT

```
//SCC

vector<vector<int>> adj, scc;
vector<int> dfs_num, dfs_low, compId;
vector<bool> inStack;
stack<int> stk;
int timer;
void dfs(int node) {
    dfs_num[node]=dfs_low[node]=++timer;
    stk.push(node);
    inStack[node]=1;
    for(int child : adj[node])
       if(!dfs_num[child]) {
          dfs(child);
          dfs_low[node]=min(dfs_low[node], dfs_low[child]);
       }
       else if(inStack[child])
          dfs_low[node]=min(dfs_low[node], dfs_num[child]);

//can be dfs_low[node] = min(dfs_low[node], dfs_low[child]);

    if(dfs_low[node]==dfs_num[node]) {
       scc.push_back(vector<int>());
       int v=-1;
       while(v!=node) {
          v=stk.top();
          stk.pop();
          inStack[v]=0;
          scc.back().push_back(v);
          compId[v]=scc.size()-1;
       }
    }
}
void SCC() {
    timer=0;
    dfs_num=dfs_low=compId=vector<int>(adj.size());
    inStack=vector<bool>(adj.size());
    scc=vector<vector<int >>();
    for(int i=1; i<adj.size(); i++)
       if(!dfs_num[i]) dfs(i);
}
int n;
int Not(int x) {
    return (x>n ? x-n : x+n);
}
```

```
void addEdge(int a, int b) {
    adj[Not(a)].push_back(b);
    adj[Not(b)].push_back(a);
}
void add_xor_edge(int a, int b) {
    addEdge(Not(a), Not(b));
    addEdge(a, b);
}
// this should be fixed
bool _2SAT(vector<int>& value) {
    SCC();
    for(int i=1; i<=n; i++)
       if(compId[i]==compId[Not(i)])
          return false;
    vector<int> assign(scc.size(), -1);
    for(int i=0; i<scc.size(); i++)
       if(assign[i]==-1) {
          assign[i]=true;
          assign[compId[Not(scc[i].back())]]=false;
       }
    for(int i=1; i<=n; i++)
       value[i]=assign[compId[i]];
    return true;
}
// to this but needs testing
bool _2SAT(vector<int> &value) {
    SCC();
    for (int i = 1; i <= n; i++)
        if (compId[i] == compId[Not(i)]) return false;

    value.assign(n + 1, 0);
    // Tarjan gives components in reverse topological order if you index by discovery
    // Using the standard rule:    for (int i = 1; i <= n; i++)
        value[i] = (compId[i] > compId[Not(i)]);
    return true;
}
```

2.18 TwoSAT F

```
class two_sat
{
 public:
  two_sat(int n) : answer(n, -1), start(2 * n + 1)
  {
  }
  void add_or_clause(int x, bool valx, int y, bool valy)
  {
    x = x * 2 + valx;
    y = y * 2 + valy;
    edges.push_back({ x ^ 1, y });
    edges.push_back({ y ^ 1, x });
    ++start[x ^ 1];
    ++start[y ^ 1];
  }
  pair<bool, vector<int>> solve()
  {
    for (int i = 0; i + 1 < start.size(); ++i)
    { start[i + 1] += start[i]; }
    proc();
    tos.resize(edges.size());
    for (const auto& p : edges)
    { tos[start[p.first]++] = p.second; }
    proc();
    for (int i = 0; i < answer.size(); ++i)
    {
      if (answer[i] == -1)
      {
        lastv.clear();
        if (!dfs(2 * i))
        {
          for (int v : lastv)
          { answer[v] = -1; }
          if (!dfs(2 * i + 1))
          { return { false, {}}; }
        }
      }
    }
    return { true, answer };
  }
 private:
  void proc()
  {
    for (int i = start.size() - 1; i; --i)
    { start[i] = start[i - 1]; }
    start[0] = 0;
  }
  bool dfs(int v)
  {
    lastv.push_back(v / 2);
    answer[v / 2] = v % 2;
    for (int i = start[v]; i < start[v + 1]; ++i)
    {
      const int to = tos[i];
      if ((answer[to / 2] != -1 && answer[to / 2] != to % 2) || (answer[to / 2] == -1 &&
↪   !dfs(to)))
      { return false; }
    }
    return true;
  }
  vector<int> answer, lastv, start, tos;
  vector<pair<int, int>> edges;
};
```

2.19 Segment Tree

2.19.1 Extended Segment Tree

```
struct segtree {
    segtree *left = nullptr, *right = nullptr;
    int mx = 0;

    segtree(int val = 0) :
            mx(val) {
    }

    void extend() {
        if (left == nullptr) {
            left = new segtree();
            right = new segtree();
        }
    }

    void pushup() {
        mx = max(left->mx, right->mx);
    }

    ~segtree() {
        if (left == nullptr)return;
        delete left;
        delete right;
    }
};

class extened_segment_tree {
#define MID ((start+end)>>1)

    void update(segtree *root, int start, int end, int pos, int val) {
        if (pos < start || end < pos)
            return;
        if (start == end) {
            root->mx = max(root->mx, val);
            return;
        }
        root->extend();
        update(root->left, start, MID, pos, val);
        update(root->right, MID + 1, end, pos, val);
        root->pushup();
    }

    int query(segtree *root, int start, int end, int l, int r) {
        if (root == nullptr || r < start || end < l)
            return 0;
        if (l <= start && end <= r)
            return root->mx;
        return max(query(root->left, start, MID, l, r),
                   query(root->right, MID + 1, end, l, r));
    }
public:
    int start, end;
    segtree *root;

    extened_segment_tree() {
    }

    ~extened_segment_tree() {
        delete root;
    }

    extened_segment_tree(int start, int end) : start(start), end(end) {
        root = new segtree();
    }

    void update(int pos, int val) {
        update(root, start, end, pos, val);
    }

    int query(int l, int r) {
        return query(root, start, end, l, r);
    }

#undef MID
};
```

2.19.2 Persistant Basic

```
struct Node {
  Node *ls{}, *rs{};
  int sum{};
} pool[int(1e6)];

int top;
Node *null = pool + 0;

void reset() {
  top = 1;
  null->sum = 0;
  null->ls = null->rs = null;
}

Node *newNode() {
  auto p = pool + (top++);
  *p = *null;
  return p;
}

// x is postion to insert
Node *add(Node *p, int l, int r, int x) {
  auto np = newNode();
  *np = *p;
  np->sum++;
  if (r - l == 1) {
    return np;
  }
  int m = (l + r) / 2;
  if (x < m) {
    np->ls = add(p->ls, l, m, x);
  } else {
    np->rs = add(p->rs, m, r, x);
  }
  return np;
}

void solve() {
  int n;
  const int C = (int)1E5;
  vector<Node *> root(n);

  // when add, w is postion on seg you want to add
  // root[v] = add(root[u], 1, C + 1, w);  auto Find = [&](Node *a, Node *b, int k) {
    int l = 1, r = C + 1;    while (r - l > 1) {      int m = (l + r) / 2;      int cnt
↪   = a->ls->sum - b->ls->sum;      if (cnt >= k) {        a = a->ls;        b = b->ls;
↪         r = m;      } else {        k -= cnt;        a = a->rs;        b = b->rs;
↪     l = m;      }    }    return l;  };
}
```

2.19.3 Persistent Segment Tree (Anas)

```
struct Node;
extern Node *emp;
struct Node {
  Node *left, *right;
  int val;
  Node() {
    left = right = this;
    val = 0;
  }
  Node(int val, Node *L = emp, Node *R = emp) : val(val), left(L), right(R) {}
};
Node *emp = new Node();
const int N = 2e5 + 5, MAX = 2e5 + 5;
Node *seg[N] = {emp};
Node *insert(Node *root, int start, int end, int idx) {
  if (idx > end || idx < start)
    return root;
  if (start == end) {
    return new Node(root->val + 1);
  }
  int mid = start + end >> 1;
  Node *L = insert(root->left, start, mid, idx);
  Node *R = insert(root->right, mid + 1, end, idx);
  return new Node(L->val + R->val, L, R);
}
int query(Node *L, Node *R, int start, int end, int k) {
  if (end <= k)
```

```
    return 0;
  if (start > k)
    return R->val - L->val;
  int mid = start + end >> 1;
  return query(L->left, R->left, start, mid, k) +
         query(L->right, R->right, mid + 1, end, k);
}
```

2.19.4 PST Kareem

```
struct Node {
    Node *left, *right;
    ll val;

    Node(ll val) : left(nullptr), right(nullptr), val(val) {}

    Node(Node *l, Node *r) : left(l), right(r), val(l->val + r->val) {}
};

struct PST {
    int n;
    vector<Node*>roots;
    Node *build(int s, int e) {
        if (s == e) {
            return new Node(0);
        }
        return new Node(build(s, (s + e) / 2), build((s + e) / 2 + 1, e));
    }

    Node *update(Node *prev, int s, int e, int idx, ll val) {
        if (s == e) {
            return new Node(val);
        }
        int mid = (s + e) / 2;
        if (idx <= mid) {
            return new Node(update(prev->left, s, mid, idx, val), prev->right);
        } else {
            return new Node(prev->left, update(prev->right, mid + 1, e, idx, val));
        }
    }

    ll get(Node *cur, int s, int e, int l, int r) {
        if (s > r || e < l)return 0;
        if (s >= l && e <= r)return cur->val;
        return get(cur->left, s, (s + e) / 2, l, r) + get(cur->right, (s + e) / 2 + 1,
↪   e, l, r);
    }
    PST(int n) : n(n){
        roots.push_back(build(0,n-1));
    }
    void update(int k, int idx, ll x){
        roots[k] = update(roots[k], 0, n - 1, idx, x);
    }
    ll get(int k,int l,int r){
        return get(roots[k],0,n-1,l,r);
    }
    void makeCopy(int k){
        roots.push_back(roots[k]);
    }
};
```

2.19.5 PST Path Kareem

```
// values on nodes
struct Node {
    Node *left, *right;
    ll frq, sum;
    Node(ll frq, ll sum) : left(nullptr), right(nullptr), frq(frq), sum(sum) {}

    Node(Node *l, Node *r) : left(l), right(r), frq(l->frq + r->frq), sum(l->sum + r->sum) {}

};

struct PST {
private:
    int n;
    vector<Node *> roots;

    Node *build(int s, int e) {
        if (s == e) {
            return new Node(0ll, 0ll);
        }
        return new Node(build(s, (s + e) / 2), build((s + e) / 2 + 1, e));
    }

    Node *update(Node *prev, int s, int e, int idx, ll val) {
        if (s == e) {
            //cout<<"HI ";cout<<idx<<' '<<val<<' '<<(prev->frq)<<' '<<(prev->sum)<<endl;
            return new Node(1 + prev->frq, val + prev->sum);
        }
        int mid = (s + e) / 2;
        if (idx <= mid) {
            return new Node(update(prev->left, s, mid, idx, val), prev->right);
        } else {
            return new Node(prev->left, update(prev->right, mid + 1, e, idx, val));
        }
    }
    //kth element
    ll getKth(Node *u, Node *v, Node *lc, Node *upLc, int s, int e, ll k) {
        if (s == e)return s;
        int lVal = u->left->frq + v->left->frq - upLc->left->frq - lc->left->frq;
        if (lVal >= k) {
            return getKth(u->left, v->left, lc->left, upLc->left, s, (s + e) / 2, k);
        }
        return getKth(u->right, v->right, lc->right, upLc->right, (s + e) / 2 + 1, e, k - lVal);
    }
    pair<ll, ll> merge(const pair<ll, ll> &a, const pair<ll, ll> &b) {
        return {a.first + b.first, a.second + b.second};
    }
    // number of elements, sum of them in range
    pair<ll, ll> getRange(Node *u, Node *v, Node *lc, Node *upLc, int s, int e, int l,
↪   int r) {
        if (s > r || e < l)return {0, 0};
        if (s >= l && e <= r) {
            return {u->frq + v->frq - (upLc->frq) - lc->frq, u->sum + v->sum -
↪   (upLc->sum) - lc->sum};
        }
        return merge(getRange(u->left, v->left, lc->left, upLc->left, s, (s + e) / 2, l,
↪   r),
                     getRange(u->right, v->right, lc->right, upLc->right, (s + e) / 2 +
↪   1, e, l, r));
    }

public:
    PST(int n) : n(n) {
        roots = vector<Node *>(n);
        roots[0] = build(0, n - 1);
    }

    void update(int u, int par, int idx, ll val) {
        roots[u] = update(roots[par], 0, n - 1, idx, val);
    }
    //kth element
    ll getKth(int u, int v, int lc, int upLC, ll k) {
        return getKth(roots[u], roots[v], roots[lc], roots[upLC], 0, n - 1, k);
    }
    // number of elements, sum of them in range
    pair<ll, ll> getRange(int u, int v, int lc, int upLc, int l, int r) {
        return getRange(roots[u], roots[v], roots[lc], roots[upLc], 0, n - 1, l, r);
    }
};
```

2.19.6 RST Lazy

```
#define outofrange l > e || s > r
#define inrange l <= s && e <= r
#define lchild p * 2, s, (s+e)/2
#define rchild p * 2 + 1, (s+e)/2 + 1, e

const int mod = 1e9 + 7;
static const int N = 3e5 + 500;
// if needed more than one thingy or complex merging
struct W
{
    int sum, sum2;
    int lz;
    W operator+(W he) const
    {
       // up
       return { (sum + he.sum) % mod, (sum2 + he.sum2) % mod, 0 };
    }

    void prop(int len)
    {
       // prop lazy
       lz %= mod;
       sum2 += lz * lz % mod * len + 2 * sum * lz;
       sum += len * lz;
       sum2 %= mod;
       sum %= mod;
       lz = 0;
    }
};
W seg[4 * N], def = { 0 };
int n, l, r, val;

void dostuff(int p, int s, int e)
{
    // down
    if (s != e)
       seg[p * 2].lz += seg[p].lz, seg[p * 2 + 1].lz += seg[p].lz;
    seg[p].prop(e - s + 1);
}
void update(int p = 1, int s = 1, int e = n)
{
    dostuff(p, s, e);
    if (outofrange)return;
    if (inrange)
    {
       seg[p].lz += val;
       dostuff(p, s, e);
       return;
    }
    update(lchild), update(rchild);
    seg[p] = seg[p * 2] + seg[p * 2 + 1];
}

W get(int p = 1, int s = 1, int e = n)
{
    dostuff(p, s, e);
    if (outofrange)
       return def;
    if (inrange)
       return seg[p];
    return get(lchild) + get(rchild);
}
```

2.19.7 RST simple

```
#define outofrange l > e || s > r
#define inrange l <= s && e <= r
#define lchild p * 2, s, (s+e)/2
#define rchild p * 2 + 1, (s+e)/2 + 1, e
```

```
static const int N = 3e5 + 500;
// if needed more than one thingy or complex merging
struct W
{
  int sum, mx, mn;
  W operator+(W he) const
  {
    return { sum + he.sum, max(mx, he.mx), min(mn, he.mn) };
  }
};
W seg[4 * N], def = { 0 }, val;
int n, l, r;

void update(int p = 1, int s = 1, int e = n)
{
  if (outofrange)return;
  if (inrange)
  {
    seg[p] = val;
    return;
  }
  update(lchild), update(rchild);
  seg[p] = seg[p * 2] + seg[p * 2 + 1];
}
W get(int p = 1, int s = 1, int e = n)
{
  if (outofrange)
    return def;
  if (inrange)
    return seg[p];
  return get(lchild) + get(rchild);
}
```

2.19.8 Segment Tree with lazy

```
Segment tree lazy
/*
for efficient memory (2*n)
#define LEFT (idx+1)
#define MID ((start+end)>>1)
#define RIGHT (idx+((MID-start+1)<<1))
*/
template<typename T>
class segment_tree {//1-based
#define LEFT (idx<<1)
#define RIGHT (idx<<1|1)
#define MID ((start+end)>>1)
    int n;
    vector<T> tree;
    vector<T> lazy;

    T merge(const T &left, const T &right) {
    }

    inline void pushdown(int idx, int start, int end) {
        if (lazy[idx] == 0)
            return;
        //update tree[idx] with lazy[idx]
        tree[idx] += lazy[idx];
        if (start != end) {
            lazy[LEFT] += lazy[idx];
            lazy[RIGHT] += lazy[idx];
        }
        //clear lazy
        lazy[idx] = 0;
    }

    inline void pushup(int idx) {
        tree[idx] = merge(tree[LEFT], tree[RIGHT]);
    }

    void build(int idx, int start, int end) {
        if (start == end)
            return;
        build(LEFT, start, MID);
        build(RIGHT, MID + 1, end);
        pushup(idx);
    }

    void build(int idx, int start, int end, const vector<T> &arr) {
        if (start == end) {
            tree[idx] = arr[start];
            return;
        }
        build(LEFT, start, MID, arr);
        build(RIGHT, MID + 1, end, arr);
        pushup(idx);
    }

    T query(int idx, int start, int end, int from, int to) {
        pushdown(idx, start, end);
        if (from <= start && end <= to)
            return tree[idx];
        if (to <= MID)
            return query(LEFT, start, MID, from, to);
        if (MID < from)
            return query(RIGHT, MID + 1, end, from, to);
        return merge(query(LEFT, start, MID, from, to),
                     query(RIGHT, MID + 1, end, from, to));
    }

    void update(int idx, int start, int end, int lq, int rq,
                const T &val) {
        pushdown(idx, start, end);
        if (rq < start || end < lq)
            return;
        if (lq <= start && end <= rq) {
            lazy[idx] += val;//update lazy
            pushdown(idx, start, end);
            return;
        }
        update(LEFT, start, MID, lq, rq, val);
        update(RIGHT, MID + 1, end, lq, rq, val);
        pushup(idx);
    }

public:
    segment_tree(int n) : n(n), tree(n << 2), lazy(n << 2) {
    }

    segment_tree(const vector<T> &v) {
        n = v.size() - 1;
        tree = vector<T>(n << 2);
        lazy = vector<T>(n << 2);
        build(1, 1, n, v);
    }

    T query(int l, int r) {
        return query(1, 1, n, l, r);
    }

    void update(int l, int r, const T &val) {
        update(1, 1, n, l, r, val);
    }

#undef LEFT
#undef RIGHT
#undef MID
};
```

2.19.9 SparseSegmentTree(setBinary-sum Range)

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

class SparseSegtree {
  private:
  struct Node {
    int freq = 0;
    int lazy = 0;
    Node *left = nullptr;
    Node *right = nullptr;
  };
  Node *root = new Node;
  const int n;

  int comb(int a, int b) { return a + b; }

  void apply(Node *cur, int len, int val) {
    if (val == 1) {
      (cur->lazy) = val;
      (cur->freq) = len * val;
    }
  }

  void push_down(Node *cur, int l, int r) {
    if ((cur->left) == nullptr) { (cur->left) = new Node; }
    if ((cur->right) == nullptr) { (cur->right) = new Node; }
    int m = (l + r) / 2;
    apply(cur->left, m - l + 1, cur->lazy);
    apply(cur->right, r - m, cur->lazy);
  }

  void range_set(Node *cur, int l, int r, int ql, int qr, int val) {
    if (qr < l || ql > r) { return; }
    if (ql <= l && r <= qr) {
      apply(cur, r - l + 1, val);
    } else {
      push_down(cur, l, r);
      int m = (l + r) / 2;
      range_set(cur->left, l, m, ql, qr, val);
      range_set(cur->right, m + 1, r, ql, qr, val);
      (cur->freq) = comb((cur->left)->freq, (cur->right)->freq);
    }
  }

  int range_sum(Node *cur, int l, int r, int ql, int qr) {
    if (qr < l || ql > r) { return 0; }
    if (ql <= l && r <= qr) { return cur->freq; }
    push_down(cur, l, r);
    int m = (l + r) / 2;
    return comb(range_sum(cur->left, l, m, ql, qr),
                range_sum(cur->right, m + 1, r, ql, qr));
  }

  public:
  SparseSegtree(int n) : n(n) {}

  void range_set(int ql, int qr, int val) { range_set(root, 0, n - 1, ql, qr, val); }

  int range_sum(int ql, int qr) { return range_sum(root, 0, n - 1, ql, qr); }
};

int main() {
  int query_num;
  cin >> query_num;
  const int RANGE_SIZE = 1e9;
  SparseSegtree st(RANGE_SIZE + 1);

  int c = 0;
  for (int i = 0; i < query_num; i++) {
    int type, x, y;
    cin >> type >> x >> y;
    if (type == 1) {
      c = st.range_sum(x + c, y + c);
      cout << c << '\n';
    } else if (type == 2) {
      st.range_set(x + c, y + c, 1);
    }
  }
}
```

2.19.10 SparseSegmentTree(xor-sum Range)

```
class SparseSegtree {
private:
    struct Node {
        int freq = 0;
        bool lazy = 0;
        Node *left = nullptr;
        Node *right = nullptr;
    };
    Node *root = new Node;
    const int n;

    int comb(int a, int b) { return a + b; }

    void apply(Node *cur, int len, int val) {
        if(val == 1){
            (cur->lazy) ^= val;
            cur->freq = len - cur->freq;
        }
    }

    void push_down(Node *cur, int l, int r) {
        if ((cur->left) == nullptr) { (cur->left) = new Node; }
        if ((cur->right) == nullptr) { (cur->right) = new Node; }
        int m = (l + r) / 2;
        apply(cur->left, m - l + 1, cur->lazy);
        apply(cur->right, r - m, cur->lazy);
        cur->lazy = 0;
    }

    void flip_range(Node *cur, int l, int r, int ql, int qr, int val) {
        if (qr < l || ql > r) { return; }
        if (ql <= l && r <= qr) {
            apply(cur, r - l + 1, val);
        } else {
            push_down(cur, l, r);
            int m = (l + r) / 2;
            flip_range(cur->left, l, m, ql, qr, val);
            flip_range(cur->right, m + 1, r, ql, qr, val);
            (cur->freq) = comb((cur->left)->freq, (cur->right)->freq);
        }
    }

    int range_sum(Node *cur, int l, int r, int ql, int qr) {
        if (qr < l || ql > r) { return 0; }
        if (ql <= l && r <= qr) { return cur->freq; }
        push_down(cur, l, r);
        int m = (l + r) / 2;
        return comb(range_sum(cur->left, l, m, ql, qr),
                    range_sum(cur->right, m + 1, r, ql, qr));
    }

public:
    SparseSegtree(int n) : n(n) {}

    void flip_range(int ql, int qr, int val) { flip_range(root, 0, n - 1, ql, qr, val); }

    int range_sum(int ql, int qr) { return range_sum(root, 0, n - 1, ql, qr); }
};
```

3 Flows

3.1 Dinic Reda get min cut edges

```
void dfs2(int u, int par) {
    if (reached[u])
        return;
    reached[u] = 1;
    for (int i = 0; i < (int) g[u].size(); i++) {
        edge &e = g[u][i];

        if (par == e.to)continue;
        if (e.flow == e.w) continue;
        dfs2(e.to, u);
    }
}

vector<ii > get_cut(int _s, int _t) {
    max_flow(_s, _t);
    reached = vector<int>(n);
    dfs2(s, -1);
    vector<ii > ret;
    for (int u = s; u <= t; u++) {
        if (reached[u]) {
            for (int i = 0; i < (int) g[u].size(); i++) {
                edge &e = g[u][i];
                if (!reached[e.to]) {
                    ret.push_back({u, e.to});
                }
            }
        }
    }
    return ret;
}
```

3.2 Dinic Reda with scaling

```
Dinic with capacity scaling runs in:
O(V * E * log C)
where
* V = number of vertices
* E = number of edges
* C = maximum edge capacity.

const ll inf = 1LL << 61;

struct Dinic {
    struct edge {
        int to, rev;
        ll flow, w;
        int id;
    };
    int n, s, t, mxid;
    vector<int> d, flow_through, done;
    vector<vector<edge>> g;

    Dinic() {}

    Dinic(int _n) {
        n = _n + 10;
        mxid = 0;
        g.resize(n);
    }

    void add_edge(int u, int v, ll w, int id = -1) {
        edge a = {v, (int)g[v].size(), 0, w, id};
        edge b = {u, (int)g[u].size(), 0, w, -2}; // for bidirectional edges cap(b) = w
        g[u].emplace_back(a);
        g[v].emplace_back(b);
        mxid = max(mxid, id);
    }

    bool bfs(ll scale) {
        d.assign(n, -1);
        d[s] = 0;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (auto &e : g[u]) {
                int v = e.to;
                if (d[v] == -1 && e.flow < e.w && e.w >= scale) {
                    d[v] = d[u] + 1;
                    q.push(v);
                }
            }
        }
        return d[t] != -1;
    }

    ll dfs(int u, ll flow, ll scale) {
        if (u == t) return flow;
        for (int &i = done[u]; i < (int)g[u].size(); i++) {
            edge &e = g[u][i];
            if (e.w < scale || e.flow >= e.w) continue;
            int v = e.to;
            if (d[v] == d[u] + 1) {
                ll nw = dfs(v, min(flow, e.w - e.flow), scale);
                if (nw > 0) {
                    e.flow += nw;
                    g[v][e.rev].flow -= nw;
                    return nw;
                }
            }
        }
        return 0;
    }

    ll max_flow(int _s, int _t) {
        s = _s;
        t = _t;
        ll flow = 0;

        // Determine the maximum capacity in the graph
        ll scale = 1;
        for (int i = 0; i < n; i++) {
            for (const auto &e : g[i]) {
                scale = max(scale, e.w);
            }
        }

        // Apply scaling
        for (scale = (scale + 1) / 2; scale > 0; scale /= 2) {
            while (bfs(scale)) {
                done.assign(n, 0);
                while (ll nw = dfs(s, inf, scale)) flow += nw;
            }
        }

        flow_through.assign(mxid + 10, 0);
        for (int i = 0; i < n; i++)
            for (auto e : g[i])
                if (e.id >= 0)
                    flow_through[e.id] = e.flow;

        return flow;
    }
};
```

3.3 Dinic Reda

```
struct Dinic
{
  struct edge
  {
    int to, rev;
    ll flow, w;
    int id;
  };
  int n, s, t, mxid;
  vector<int> d, flow_through, done;
  vector<vector<edge>> g;

  Dinic()
  {
  }

  Dinic(int _n)
  {
    n = _n + 10;
    mxid = 0;
    g.resize(n);
  }

  void add_edge(int u, int v, ll w, int id = -1)
  {
    edge a = { v, (int)g[v].size(), 0, w, id };
    edge b = { u, (int)g[u].size(), 0, 0, -2 };//for bidirectional edges cap(b) = w
    g[u].emplace_back(a);
    g[v].emplace_back(b);
    mxid = max(mxid, id);
  }

  bool bfs()
  {
    d.assign(n, -1);
    d[s] = 0;
    queue<int> q;
    q.push(s);
    while (!q.empty())
    {
      int u = q.front();
      q.pop();
      for (auto& e : g[u])
      {
        int v = e.to;
        if (d[v] == -1 && e.flow < e.w) d[v] = d[u] + 1, q.push(v);
      }
    }
    return d[t] != -1;
  }

  ll dfs(int u, ll flow)
  {
    if (u == t) return flow;
    for (int& i = done[u]; i < (int)g[u].size(); i++)
    {
      edge& e = g[u][i];
      if (e.w <= e.flow) continue;
      int v = e.to;
      if (d[v] == d[u] + 1)
      {
        ll nw = dfs(v, min(flow, e.w - e.flow));
        if (nw > 0)
        {
          e.flow += nw;
          g[v][e.rev].flow -= nw;
          return nw;
        }
      }
    }
    return 0;
  }

  ll max_flow(int _s, int _t)
  {
    s = _s;
    t = _t;
    ll flow = 0;
    while (bfs())
    {
      done.assign(n, 0);
      while (ll nw = dfs(s, inf)) flow += nw;
    }
    flow_through.assign(mxid + 10, 0);
    for (int i = 0; i < n; i++) for (auto e : g[i]) if (e.id >= 0) flow_through[e.id] =
↪   e.flow;
    return flow;
  }
};

void solve()
{

  int n, m;
  cin >> n >> m;
  Dinic F(n);
  for (int i = 0; i < m; ++i)
  {
    int a, b, c;
    cin >> a >> b >> c;
    F.add_edge(a, b, c);
  }
  cout << F.max_flow(1, n) << endl;
}
```

3.4 Edmonds Karp

```
//O( V * E * E)
#define INF 0x3f3f3f3f3f3f3f3fLL
int n;
int capacity[101][101];

int getPath(int src, int dest, vector<int> &parent) {
    parent = vector<int>(n + 1, -1);
    queue<pair<int, int>> q;
    q.push({src, INF});
    while (q.size()) {
        int cur = q.front().first, flow = q.front().second;
        q.pop();

        if (cur == dest) return flow;
        for (int i = 1; i <= n; i++)
            if (parent[i] == -1 && capacity[cur][i]) {
                parent[i] = cur;
                q.push({i, min(flow, capacity[cur][i])});
                if (i == dest) return q.back().second;
            }
    }
    return 0;
}
int Edmonds_Karp(int source, int sink) {
    int max_flow = 0;
    int new_flow = 0;
    vector<int> parent(n + 1, -1);
    while (new_flow = getPath(source, sink, parent)) {
        max_flow += new_flow;
        int cur = sink;
        while (cur != source) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        };
    }
    return max_flow;
}
```

3.5 Hopcroft Karp

```
//Hopcroft-Karp algorithm for maximum bipartite matching
//O(sqrt(V) * E)
struct Hopcroft_Karp {//1-based
#define NIL 0
#define INF INT_MAX
  int n, m;``
  vector<vector<int>> adj;
  vector<int> rowAssign, colAssign, dist;
  bool bfs() {
    queue<int> q;
```

```cpp
    dist = vector<int>(adj.size(), INF);
    for (int i = 1; i <= n; i++)
      if (rowAssign[i] == NIL) {
        dist[i] = 0;
        q.push(i);
      }
    while (!q.empty()) {
      int cur = q.front();
      q.pop();
      if (dist[cur] >= dist[NIL])break;
      for (auto& nxt : adj[cur]) {
        if (dist[colAssign[nxt]] == INF) {
          dist[colAssign[nxt]] = dist[cur] + 1;
          q.push(colAssign[nxt]);
        }
      }
    }
    return dist[NIL] != INF;
  }
  bool dfs(int i) {
    if (i == NIL)
      return true;
    for (int j : adj[i]) {
      if (dist[colAssign[j]] == dist[i] + 1 && dfs(colAssign[j])) {
        colAssign[j] = i;
        rowAssign[i] = j;
        return true;
      }
    }
    dist[i] = INF;
    return false;
  }
  Hopcroft_Karp(int n, int m)
    :n(n), m(m), adj(n + 1), rowAssign(n + 1), colAssign(m + 1) {
  }
  void addEdge(int u, int v) {
    adj[u].push_back(v);
  }
  int maximum_bipartite_matching() {
    int rt = 0;
    while (bfs()) {
      for (int i = 1; i <= n; i++)
        if (rowAssign[i] == NIL && dfs(i))
          rt++;
    }
    return rt;
  }
};
```

3.6 Hungarian

```
// n^3 or something
// nodes are 0-based  /* There are n workers and n tasks.  You know exactly how much you
↪   need to pay each worker to perform one or another task.  You also know that every
↪   worker can only perform one task.  Your goal is to assign each worker some a task,
↪   while minimizing your expenses.  */

// fill vector a with costs  // if you want maximizie final cost then you will multiply
↪   edges cost with -1  // this algorithm works only on bipartite graph  // the maximum
↪   matching must equal to n  template<typename T>
class hungarian
{
public:
    int n;
    int m;
    vector<vector<T> > a;
    vector<T> u;
    vector<T> v;
    vector<int> pa;
    vector<int> pb;
    vector<int> way;
    vector<T> minv;
    vector<bool> used;
    T inf;
    hungarian(int _n, int _m) : n(_n), m(_m)
    {
       assert(n <= m);
       a = vector<vector<T> >(n, vector<T>(m));
       u = vector<T>(n + 1);
       v = vector<T>(m + 1);
       pa = vector<int>(n + 1, -1);
       pb = vector<int>(m + 1, -1);
       way = vector<int>(m, -1);
       minv = vector<T>(m);
       used = vector<bool>(m + 1);
       inf = numeric_limits<T>::max();
    }
    inline void add_row(int i)
    {
       fill(minv.begin(), minv.end(), inf);
       fill(used.begin(), used.end(), false);
       pb[m] = i;
       pa[i] = m;
       int j0 = m;
       do
       {
          used[j0] = true;
          int i0 = pb[j0];
          T delta = inf;
          int j1 = -1;
          for (int j = 0; j < m; j++)
          {
             if (!used[j])
```

```
            {
               T cur = a[i0][j] - u[i0] - v[j];

               if (cur < minv[j])
               {
                  minv[j] = cur;
                  way[j] = j0;
               }
               if (minv[j] < delta)
               {
                  delta = minv[j];
                  j1 = j;
               }
            }
         }
         for (int j = 0; j <= m; j++)
         {
            if (used[j])
            {
               u[pb[j]] += delta;
               v[j] -= delta;
            }
            else
            {
               minv[j] -= delta;
            }
         }
         j0 = j1;
      } while (pb[j0] != -1);
      do
      {
         int j1 = way[j0];
         pb[j0] = pb[j1];
         pa[pb[j0]] = j0;
         j0 = j1;
      } while (j0 != m);
   }
   inline T current_score()
   {
      return -v[m];
   }
   inline T solve()
   {
      for (int i = 0; i < n; i++)
      {
         add_row(i);
      }
      return current_score();
   }
};
```

3.7 Max Bipartite Matching

```
//O(E*V)
vector<vector<int>> adj;
vector<int> rowAssign, colAssign, vis;//make vis array instance of vector
int test_id;

bool canMatch(int i) {
    if (vis[i] == test_id) return false;
    vis[i] = test_id;
    for (int j: adj[i])
        if (colAssign[j] == -1) {
            colAssign[j] = i;
            rowAssign[i] = j;
            return true;
        }
    for (int j: adj[i])
        if (canMatch(colAssign[j])) {
            colAssign[j] = i;
            rowAssign[i] = j;
            return true;
        }
    return false;
}
// O(rows * edges) //number of operation could by strictly less than order (1e5*1e5->AC)
int maximum_bipartite_matching(int rows, int cols) {
    int maxFlow = 0;
    rowAssign = vector<int>(rows, -1);
    colAssign = vector<int>(cols, -1);
    vis = vector<int>(rows);
    for (int i = 0; i < rows; i++) {
        test_id++;
        if (canMatch(i)) maxFlow++;
    }
    vector<pair<int, int>> matches;
    for (int j = 0; j < cols; j++)
        if (~colAssign[j]) matches.push_back({colAssign[j], j});
    return maxFlow;
}
```

3.8 MCMF

```
struct MCMF //0-based
{
    struct edge
    {
       int from, to, cost, cap, flow, backEdge;
       edge()
       {
          from = to = cost = cap = flow = backEdge = 0;
       }
       edge(int from, int to, int cost, int cap, int flow, int backEdge) :
          from(from), to(to), cost(cost), cap(cap), flow(flow),
          backEdge(
             backEdge)
       {
       }
       bool operator<(const edge& other) const
       {
          return cost < other.cost;
       }
    };
    int n, src, dest;
    vector<vector<edge>> adj;

    const int OO = 1e9;
    MCMF(int n, int src, int dest) : n(n), src(src), dest(dest), adj(n)
    {
    }
    void addEdge(int u, int v, int cost, int cap)
    {
       edge e1 = edge(u, v, cost, cap, 0, adj[v].size());
       edge e2 = edge(v, u, -cost, 0, 0, adj[u].size());
       adj[u].push_back(e1);
       adj[v].push_back(e2);
    }
    pair<int, int> minCostMaxFlow()
    {
       int maxFlow = 0, cost = 0;
       while (true)
       {
          vector<pair<int, int>> path = spfa();
          if (path.empty())
             break;
          int new_flow = OO;
          for (auto& it : path)
          {
             edge& e = adj[it.first][it.second];
             new_flow = min(new_flow, e.cap - e.flow);
          }
          for (auto& it : path)
          {
             edge& e = adj[it.first][it.second];
             e.flow += new_flow;
             cost += new_flow * e.cost;
             adj[e.to][e.backEdge].flow -= new_flow;
          }
          maxFlow += new_flow;
       }
       return { maxFlow, cost };
    }
    enum visit
    {
       finished, in_queue, not_visited
    };
    vector<pair<int, int>> spfa()
    {
       vector<int> dis(n, OO), prev(n, -1), from_edge(n), state(n,
          not_visited);
       deque<int> q;
       dis[src] = 0;
       q.push_back(src);
       while (!q.empty())
       {
          int u = q.front();

          q.pop_front();
          state[u] = finished;
          for (int i = 0; i < adj[u].size(); i++)
          {
             edge e = adj[u][i];
             if (e.flow >= e.cap || dis[e.to] <= dis[u] + e.cost)
                continue;
             dis[e.to] = dis[u] + e.cost;
             prev[e.to] = u;
             from_edge[e.to] = i;
             if (state[e.to] == in_queue)
                continue;
             if (state[e.to] == finished
                || (!q.empty() && dis[q.front()] > dis[e.to]))
                q.push_front(e.to);
             else
                q.push_back(e.to);
             state[e.to] = in_queue;
          }
       }
       if (dis[dest] == OO)
          return {};
       vector<pair<int, int>> path;
       int cur = dest;
       while (cur != src)
       {
          path.push_back({ prev[cur], from_edge[cur] });
          cur = prev[cur];
       }
       reverse(path.begin(), path.end());
       return path;
    }
};
```

3.9 Push Relable

```
// v^2 * sqrt_root(E)


#define sz(x) x.size()

struct PushRelabel
{
    struct Edge
    {
```

```
   int dest, back;
   ll f, c;
};
vector<vector<Edge>> g;
vector<ll> ec;
vector<Edge*> cur;
vector<vi > hs;
vi H;

PushRelabel(int n) : g(n), ec(n), cur(n), hs(2 * n), H(n)
{
}

void addEdge(int s, int t, ll cap, ll rcap = 0)
{
   if (s == t) return;
   g[s].push_back({ t, sz(g[t]), 0, cap });
   g[t].push_back({ s, sz(g[s]) - 1, 0, rcap });
}

void addFlow(Edge& e, ll f)
{
   Edge& back = g[e.dest][e.back];
   if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
   e.f += f;
   e.c -= f;
   ec[e.dest] += f;
   back.f -= f;
   back.c += f;
   ec[back.dest] -= f;
}

ll calc(int s, int t)
{
   int v = sz(g);
   H[s] = v;
   ec[t] = 1;
   vi co(2 * v);
   co[0] = v - 1;
   for (int i = 0; i < v; i++)
      cur[i] = g[i].data();
   for (Edge& e : g[s]) addFlow(e, e.c);

   for (int hi = 0;;)
   {
      while (hs[hi].empty()) if (!hi--) return -ec[s];
      int u = hs[hi].back();
      hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
   if (cur[u] == g[u].data() + sz(g[u]))
         {
            H[u] = 1e9;
            for (Edge& e : g[u])
               if (e.c && H[u] > H[e.dest] + 1)
```

```
                      H[u] = H[e.dest] + 1, cur[u] = &e;
                if (++co[H[u]], !--co[hi] && hi < v)
                   for (int i = 0; i < v; i++)
                      if (hi < H[i] && H[i] < v)
                         --co[H[i]], H[i] = v + 1;
                hi = H[u];
             }
             else if (cur[u]->c && H[u] == H[cur[u]->dest] + 1)
                addFlow(*cur[u], min(ec[u], cur[u]->c));
             else ++cur[u];
       }
    }

    bool leftOfMinCut(int a)
    {
       return H[a] >= sz(g);
    }
};
```

4 Geometry

4.1 Areas

```
ld areaOfTri(P a, P b, P c){
    return abs((vec(a, b)^vec(a, c)) / 2.0);
}

ld triArea(ot A, ot B, ot C){
    ld s = (A + B + C) * 0.5;
    return sqrt(s * (s - A) * (s - B) * (s - C));
}
//Area of a Triangle Formula in Coordinate Geometry
//If (x1, y1), (x2, y2), and (x3, y3) are the three vertices of a triangle on the
↪  coordinate plane, then //its area is calculated by the formula
//(1/2) |x1(y2 − y3) + x2(y3 − y1) + x3(y1 − y2)|

// tested
ld areaOfPolygon(vector<P> p){
    ot area = 0;
    for(int i = 1; i < p.size() - 1; i++)
       area += vec(p[0], p[i])^vec(p[0], p[i + 1]);
    return abs(area / 2.0);
}
```

4.2 Basic Funcs

```
bool collinear(P a, P b, P c)
{
    return (vec(a, b) ^ vec(a, c)) == 0;
}

**get acute directed angle from a to b**
// tested
```

```
ld gda(P a, P b)
{
    ld ang = abs(angle(a) - angle(b));
    ang = min(ang, 2 * PI - ang);
    return ang * ((a ^ b) > 0 ? 1 : -1);
}

// get's X given a line and y
ld getX(P a, P b, ll y) {
    ld slope = (b.y - a.y) / (b.x - a.x);
    return a.x + (y - a.y) / slope;
}

//tested
ld linePointDis(P l1, P l2, P p)
{
    ot area = abs(vec(p, l1) ^ vec(p, l2));
    ld base = len(vec(l1, l2));
    return area / base;
}


ld segmentPointDis(P a, P b, P p){
    P ab = vec(a,b), ap = vec(a,p);
    ld ab2 = ab*ab; // dot
    if (ab2 < eps) return len(vec(p,a)); // degenerate segment
    ld t = max<ld>(0, min<ld>(1, (ap*ab)/ab2));
    P proj = { a.x + ab.x*t, a.y + ab.y*t };
    return len(vec(p, proj));
}

struct cmp
{
    P about;
    cmp(P c)
    {
       about = c;
    }
    bool operator()(const P& a, const P& b) const
    {
       ld cr = vec(about, a) ^ vec(about, b);
       if (fabs(cr) < eps)
          return a < b;
       return cr > 0;
    }
};

void sortAntiClockWise(vector<P>& pnts)
{
    P mn(*min_element(all(pnts)));
    sort(pnts.begin(), pnts.end(), cmp(mn));
}
inline bool pibb(P const& a, P const& b1, P const& b2)
{
    return a.x >= min(b1.x, b2.x) &&
       a.x <= max(b1.x, b2.x) &&
       a.y >= min(b1.y, b2.y) &&
       a.y <= max(b1.y, b2.y);
}

bool lineIntersection(P p1, P p2, P p3, P p4, P& sec)
{
    ld denom = (p1.x - p2.x) * (p3.y - p4.y) - (p1.y - p2.y) * (p3.x -p4.x);

    if (abs(denom) < eps)
       return false;

    ld t = ((p1.x - p3.x) * (p3.y - p4.y) - (p1.y - p3.y) * (p3.x - p4.x)) / denom;

    sec = p1 + vec(p1, p2) * t;

    return true;
}

bool segmentsIntersection(P l1, P l2, P k1, P k2, P& sec)
{
    if (lineIntersection(l1, l2, k1, k2, sec))
    {
       if (pibb(sec, l1, l2) && pibb(sec, k1, k2))
          return true;
       return false;
    }
    return false;
}

bool isPointOnSegment(P const& a, P const& l1, P const& l2)
{
    return collinear(a, l1, l2) && pibb(a, l1, l2);
}
P getCircleCenter(P A, P B, P C) {
    if (isCollinear(A, B, C)) {
       collinear = true;
       return {0, 0};
    }
    collinear = false;
    ld D = 2 * (A.x * (B.y - C.y) + B.x * (C.y - A.y) + C.x * (A.y - B.y));
    ld Ux = ((A.x * A.x + A.y * A.y) * (B.y - C.y) + (B.x * B.x + B.y * B.y) * (C.y -
↪   A.y) + (C.x * C.x + C.y * C.y) * (A.y - B.y)) / D;
    ld Uy = ((A.x * A.x + A.y * A.y) * (C.x - B.x) + (B.x * B.x + B.y * B.y) * (A.x -
↪   C.x) + (C.x * C.x + C.y * C.y) * (B.x - A.x)) / D;
    return {Ux, Uy};
}

bool issqare(P a, P b, P c, P d)
{
    if (a == b || a == c || a == d || b == c || b == c || c == d)
       return false;
    vector<ld> ds = {
       a.dis(b), a.dis(c), a.dis(d),
       b.dis(c), b.dis(d), c.dis(d)
    };
    sort(all(ds));
    return ds[0] > 0 &&
       abs(ds[2] - ds[3]) < 1e-8 &&
       abs(ds[0] - ds[1]) < 1e-8 &&
       abs(ds[1] - ds[2]) < 1e-8 &&
       abs(ds[4] - ds[5]) < 1e-8;

}

P reflectPoint(P a, P b, P p) {
    P ab = vec(a, b), ap = vec(a, p);
    ld ab2 = ab * ab; // dot(ab, ab)
    if (ab2 < eps) return p; // degenerate line -> nothing to reflect
    // projection parameter on the INFINITE line (no clamp!)
    ld t = (ap * ab) / ab2;
    P proj = {a.x + ab.x * t, a.y + ab.y * t};
    // reflected point: proj + (proj - p)
    P r = {2 * proj.x - p.x, 2 * proj.y - p.y};
    return r;
}
```

4.3 ConvexHull

```
void convexHull(vector<P> p, vector<P>& hull)
{

    sort(all(p), [&](P& a, P& b)
    {
      if (a.x != b.x)
         return a.x < b.x;
      return a.y < b.y;
    });
    if (p.size() == 1)
    {
       hull.push_back(p[0]);
       return;
    }
    for (int rep = 0; rep < 2; rep++)
    {
       int s = hull.size();
       for (int i = 0; i < p.size(); i++)
       {
          while (hull.size() >= s + 2)
          {
             P p1 = hull.end()[-2];
             P p2 = hull.end()[-1];
             if ((vec(p1, p2) ^ vec(p1, p[i])) < -eps)
                break;

             hull.pop_back();
          }
          hull.push_back(p[i]);
       }
       reverse(all(p));
       hull.pop_back();
    }
}
```

4.4 CosineRule

```
c^2 = a^2 + b^2 - 2ab * cos(C)
a^2 = b^2 + c^2 - 2bc * cos(A)
b^2 = a^2 + c^2 - 2ac * cos(B)

cos(C) = (a^2 + b^2 - c^2) / (2ab)
cos(A) = (b^2 + c^2 - a^2) / (2bc)
cos(B) = (a^2 + c^2 - b^2) / (2ac)


C = cos^(-1) [(a^2 + b^2 - c^2) / (2ab)]
A = cos^(-1) [(b^2 + c^2 - a^2) / (2bc)]
B = cos^(-1) [(a^2 + c^2 - b^2) / (2ac)]
```

4.5 Defines 3D

```
// 3D geometry (compact, ACPC-friendly)

#define ot long long
#define ld long double
#define sq(x) ((x)*(x))

const ld eps = 1e-12L;
const ld PI  = acosl(-1.0L);

struct P3 {
    ld x, y, z;

    void read() { std::cin >> x >> y >> z; }

    P3 operator + (P3 const& o) const { return {x + o.x, y + o.y, z + o.z}; }
    P3 operator - (P3 const& o) const { return {x - o.x, y - o.y, z - o.z}; }
    void operator += (P3 const& o) { x += o.x; y += o.y; z += o.z; }
    void operator -= (P3 const& o) { x -= o.x; y -= o.y; z -= o.z; }

    P3 operator * (ld k) const { return {x * k, y * k, z * k}; }
    P3 operator / (ld k) const { return {x / k, y / k, z / k}; }
    friend P3 operator * (ld k, P3 a) { return a * k; }

    // dot
    ld operator * (P3 const& o) const { return x * o.x + y * o.y + z * o.z; }

    // cross
    P3 operator ^ (P3 const& o) const {
        return { y * o.z - z * o.y, z * o.x - x * o.z, x * o.y - y * o.x };
    }

    ld norm2() const { return (*this) * (*this); }
    ld len()   const { return sqrtl(norm2()); }

    P3 unit() const {
        ld l = len();
        return (l < eps ? P3{0, 0, 0} : (*this) / l);
    }

    // Rodrigues rotation: rotate this vector around 'axis' (through origin) by 'ang' radians
    P3 rotate(P3 axis, ld ang) const {
        axis = axis.unit();
        if (axis.len() < eps) return *this;
        ld c = cosl(ang), s = sinl(ang);
        return (*this) * c + (axis ^ (*this)) * s + axis * ((axis * (*this)) * (1 - c));
    }
};
```

4.6 Defines and Point all

```
#define ot long long
#define ld long double
#define sq(x) ((x)*(x))
#define eps 1e-8
#define angle(a) (atan2((a).y, (a).x))
#define slope(p) (((p).y)/((p).x))
#define vec(a, b) ((b)-(a))
#define len(v) (hypotl((v).y, (v).x))

struct P
{
    ot x, y;
    void read()
    {
       cin >> x >> y;
    }

    bool operator==(P const& he) const
    {
       return x == he.x && y == he.y;
    }
    P operator-(P const& he) const
    {
       return { x - he.x, y - he.y };
    }

    P operator+(P const& he) const
    {
       return { x + he.x, y + he.y };
    }
    void operator-=(P const& he)
    {
       x -= he.x, y -= he.y;
    }
    void operator+=(P const& he)
    {
       x += he.x, y += he.y;
    }
    // scalar multiplication
    P operator*(ot val) const
    {
       return { x * val, y * val };
    }

    // cross product
    ot operator^(P const& he) const
    {
       return x * he.y - y * he.x;
    }
    // dot product = x1x2+y1y2 = |a|*|b|*cos(theta)
    ot operator*(P const& he) const
    {
       return x * he.x + y * he.y;
    }
    //angle in radians
    P rotate(ot angle) const
    {
       ot cos_theta = cos(angle);
       ot sin_theta = sin(angle);
       return { x * cos_theta - y * sin_theta, x * sin_theta + y * cos_theta };
    }
};


struct Line {
    ld a, b, c;
};

Line lineFromVector(P p, P v) {
    Line L;
    L.a = v.y;
    L.b = -v.x;
    L.c = v.x * p.y - v.y * p.x;

    ld norm = sqrt(L.a * L.a + L.b * L.b);
    if (norm > 1e-12) L.a /= norm, L.b /= norm, L.c /= norm;

    // (optional) make a unique orientation
    if (L.a < 0 || (abs(L.a) < 1e-12 && L.b < 0))
        L.a = -L.a, L.b = -L.b, L.c = -L.c;

    return L;
}
```

4.7 Defines and Point simple

```
#define ot long long
#define ld long double
#define eps 1e-8
#define vec(a, b) ((b)-(a))
#define len(v) (hypotl((v).y, (v).x))

struct P
{
    ot x, y;
    void read()
    {
       cin >> x >> y;
    }
    P operator-(P const& he) const
    {
       return { x - he.x, y - he.y };
    }
    // cross product
  ot operator^(P const& he) const
    {
       return x * he.y - y * he.x;
    }
};
```

4.8 Helpers 3D

```
#define vec3(a,b) ((b)-(a))

ld dis(P3 a, P3 b) { return vec3(a, b).len(); }

// scalar triple product: a · (b × c)
ld mixed(P3 a, P3 b, P3 c) { return a * (b ^ c); }

bool collinear(P3 a, P3 b, P3 c){
    P3 ab = vec3(a,b), ac = vec3(a,c);
    return (ab ^ ac).len() <= eps * ab.len() * ac.len();
}
bool collinear(P3 a, P3 b, P3 c){
    P3 ab = vec3(a,b), ac = vec3(a,c);
    return (ab ^ ac).len() <= eps * ab.len() * ac.len();
}

bool coplanar(P3 a, P3 b, P3 c, P3 d){
    P3 ab=vec3(a,b), ac=vec3(a,c), ad=vec3(a,d);
    return fabsl(mixed(ab,ac,ad)) <= eps * ab.len() * ac.len() * ad.len();
}
bool coplanar(P3 a, P3 b, P3 c, P3 d){
    P3 ab=vec3(a,b), ac=vec3(a,c), ad=vec3(a,d);
    return fabsl(mixed(ab,ac,ad)) <= eps * ab.len() * ac.len() * ad.len();
}
```

```
ld triArea(P3 a, P3 b, P3 c) {
    return (vec3(a, b) ^ vec3(a, c)).len() / 2;
}

ld tetraVol(P3 a, P3 b, P3 c, P3 d) {
    return fabsl(mixed(vec3(a, b), vec3(a, c), vec3(a, d))) / 6;
}

// projection of p onto infinite line (a,b)
P3 projPointLine(P3 p, P3 a, P3 b) {
    P3 ab = vec3(a, b);
    ld ab2 = ab * ab;
    if (ab2 < eps) return a;
    ld t = (vec3(a, p) * ab) / ab2;
    return a + ab * t;
}

ld linePointDis(P3 a, P3 b, P3 p) {
    P3 ab = vec3(a, b), ap = vec3(a, p);
    ld den = ab.len();
    if (den < eps) return ap.len();
    return (ab ^ ap).len() / den;
}

ld segmentPointDis(P3 a, P3 b, P3 p) {
    P3 ab = vec3(a, b), ap = vec3(a, p);
    ld ab2 = ab * ab;
    if (ab2 < eps) return ap.len();
    ld t = (ap * ab) / ab2;
    t = std::max<ld>(0, std::min<ld>(1, t));
    return dis(p, a + ab * t);
}

ld pointPlaneDis(P3 p, P3 A, P3 B, P3 C) {
    P3 n = (vec3(A, B) ^ vec3(A, C));
    ld den = n.len();
    if (den < eps) return 0;
    return fabsl(vec3(A, p) * n) / den;
}


// projection of p onto plane (A,B,C)
P3 projPointPlane(P3 p, P3 A, P3 B, P3 C) {
    P3 n = (vec3(A, B) ^ vec3(A, C));
    ld nn = n * n;
    if (nn < eps) return p;
    ld t = (vec3(A, p) * n) / nn;
    return p - n * t;
}

// reflection of p across plane (A,B,C)
P3 reflectPointPlane(P3 p, P3 A, P3 B, P3 C) {
    P3 q = projPointPlane(p, A, B, C);
    return q * 2 - p;
}

ld angle3(P3 a, P3 b) { // [0, pi]
    ld la = a.len(), lb = b.len();
    if (la < eps || lb < eps) return 0;
    ld cs = (a * b) / (la * lb);
    cs = std::max<ld>(-1, std::min<ld>(1, cs));
    return acosl(cs);
}

// line (p1,p2) with plane (A,B,C) -> unique intersection point in 'sec'
bool linePlaneIntersection(P3 p1, P3 p2, P3 A, P3 B, P3 C, P3& sec) {
    P3 v = vec3(p1, p2);
    P3 n = (vec3(A, B) ^ vec3(A, C));
    ld denom = n * v;
    if (fabsl(denom) < eps) return false; // parallel or in-plane -> no unique hit
    ld t = (n * (A - p1)) / denom;
    sec = p1 + v * t;
    return true;
}

// segment [p1,p2] with plane (A,B,C) -> unique intersection in 'sec'
bool segmentPlaneIntersection(P3 p1, P3 p2, P3 A, P3 B, P3 C, P3& sec) {
    P3 v = vec3(p1, p2);
    P3 n = (vec3(A, B) ^ vec3(A, C));
    ld denom = n * v;
    if (fabsl(denom) < eps) return false;
    ld t = (n * (A - p1)) / denom;
    if (t < -eps || t > 1 + eps) return false;
    sec = p1 + v * t;
    return true;
}

// point-in-triangle in 3D (accepts points slightly off-plane within eps)
bool pointInTriangle(P3 p, P3 A, P3 B, P3 C) {
    P3 n = (vec3(A, B) ^ vec3(A, C));
    ld nl = n.len();
    if (nl < eps) return false; // degenerate triangle
    if (fabsl(vec3(A, p) * n) > eps * nl) return false; // not on plane

    P3 v0 = vec3(A, C), v1 = vec3(A, B), v2 = vec3(A, p);
    ld d00 = v0 * v0, d01 = v0 * v1, d11 = v1 * v1;
    ld d20 = v2 * v0, d21 = v2 * v1;
    ld denom = d00 * d11 - d01 * d01;
    if (fabsl(denom) < eps) return false;

    ld u = (d11 * d20 - d01 * d21) / denom;
    ld v = (d00 * d21 - d01 * d20) / denom;
    return u >= -eps && v >= -eps && u + v <= 1 + eps;
}
```

4.9 Intersections

```
// two segments


// To check if two segments intersect we will use the * signed area of the ABC triangle.
↪  This can be derived * from the cross product of the vectors AB and AC.

bool intersect(Segment a, Segment b)
{
    Point p1 = { a.xi, a.yi }, p2 = { a.xf, a.yf }, p3 = { b.xi, b.yi }, p4 = { b.xf,
↪   b.yf };

    return ((p4 - p1) ^ (p2 - p1)) * ((p2 - p1) ^ (p3 - p1)) >= 0 &&
       ((p2 - p3) ^ (p4 - p3)) * ((p4 - p3) ^ (p1 - p3)) >= 0 &&
       max(p1.x, p2.x) >= min(p3.x, p4.x) && max(p3.x, p4.x) >= min(p1.x, p2.x) &&
       max(p1.y, p2.y) >= min(p3.y, p4.y) && max(p3.y, p4.y) >= min(p1.y, p2.y);
}
```

4.10 Lattice Points

```
long long getLaticeBoundryPoints(vector<P>& p)
{
    long long boundry = 0;
    p.push_back(p[0]);
    for (int i = 0; i < p.size() - 1; i++)
    {
       P a = vec(p[i], p[i + 1]);
       boundry += __gcd(abs((int)a.x), abs((int)a.y));
    }
    p.pop_back();
    return boundry;
}
long long getLaticeInsidePoints(vector<P>& p)
{
    return (getAreat2(p) - getLaticeBoundryPoints(p)) / 2 + 1;
}
```

4.11 Polygon Centroid

```
P getCentroid(vector<P>& p)
{
    ld x, y;
    long long tarea = 0;
    x = 0;
    y = 0;
    for (int i = 1; i < p.size() - 2; i++)
    {
       long long area = (vec(p[0], p[i]) ^ vec(p[0], p[i + 1]));
       x += area * ((p[0].x + p[i].x + p[i + 1].x) / 3.0);
       y += area * ((p[0].y + p[i].y + p[i + 1].y) / 3.0);
       tarea += area;
    }
    x /= tarea;
    y /= tarea;
    return { x, y };
}
```

4.12 some Properties of Regular polygons

```
Properties of Regular polygons
Some of the properties of regular polygons are listed below.

All the sides of a regular polygon are equal
All the interior angles are equal
The perimeter of a regular polygon with n sides is equal to the n times of a side
↪  measure.
The sum of all the interior angles of a simple n-gon or regular polygon = (n − 2) × 180°
The number of diagonals in a polygon with n sides = n(n − 3)/2
The number of triangles formed by joining the diagonals from one corner of a polygon = n
↪  − 2
The measure of each interior angle of n-sided regular polygon = [(n − 2) × 180°]/n
The measure of each exterior angle of an n-sided regular polygon = 360°/n



area of reqular polygon = ((l^2)*n)/(4tan(PI/n))
l is the side length

n is the number of sides
```

5 Graph

5.1 Articulation Points Kareem

```
//undirected
vector<int>adj[N];
int dfn[N],low[N];
set<int>points;
int cnt;
void articulationPoint(int node, int parent){
    dfn[node] = low[node] = ++cnt;
    int childs = 0;
    for(auto&ch : adj[node]){
        if(ch == parent)continue;
        if(dfn[ch] == -1){
            articulationPoint(ch, node);
            low[node] = min(low[node],low[ch]);
            if(dfn[node]<=low[ch]&& ~parent)points.insert(node);
            childs++;
        }else{
            low[node] = min(low[node],dfn[ch]);
        }
    }
    if(childs >1 && parent == -1)points.insert(node);
}
```

5.2 Articulation Points Reda

```
#define vi vector<int>
#define vvi vector<vector<int>>
#define vsi vector<set<int>>
//SCC
vvi G;
vi dnum, dlow, pr;
set<int> arc_point;

int dfsroot, cntroot, id, n;
void articulation_points(int node)
{
    if (dnum[node] != -1) return;
    dnum[node] = dlow[node] = ++id;
    for (auto u : G[node])
    {
       if (dnum[u] == -1)
       {
          pr[u] = node;
          articulation_points(u);
          if (dnum[node] <= dlow[u])
             if (node == dfsroot)
                cntroot++;
             else
                arc_point.insert(node);
          dlow[node] = min(dlow[node], dlow[u]);
       }
       else if (u != pr[node])
       {
          dlow[node] = min(dlow[node], dnum[u]);
       }
    }
}
void solve()
{
    dlow = pr = vi(n + 1);
    dnum = vi(n + 1, -1);
    id = 0;
    arc_point.clear();
    G = vvi(n + 1);
// input graph
    for (int i = 0; i < n; i++)
    {
       dfsroot = i;
       cntroot = 0;
       articulation_points(i);
       if (cntroot > 1)
          arc_point.insert(i);
    }
}
```

5.3 Bellmanford

```cpp
struct edge {
    int from, to, weight;

    edge() { from = to = weight = 0; }

    edge(int from, int to, int weight) :
            from(from), to(to), weight(weight) {
    }

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

vector<edge> edgeList;

//O(V*E)
void bellmanford(int n, int src, int dest = -1) {
    vector<int> dis(n + 1, oo), prev(n + 1, -1);
    dis[src] = 0;
    bool negativeCycle = false;
    int last = -1, tmp = n;
    while (tmp--) {
        last = -1;
        for (edge e: edgeList)
            if (dis[e.to] > dis[e.from] + e.weight) {
                dis[e.to] = dis[e.from] + e.weight;
                prev[e.to] = e.from;
                last = e.to;
            }
        if (last == -1)
            break;
        if (tmp == 0)
            negativeCycle = true;
    }
    if (last != -1) {
        for (int i = 0; i < n; i++)
            last = prev[last];
        vector<int> cycle;
        for (int cur = last; cur != last || cycle.size() > 1; cur =
                                                                 prev[cur])
            cycle.push_back(cur);
        reverse(cycle.begin(), cycle.end());
    }
    vector<int> path;
    while (dest != -1) {
        path.push_back(dest);
        dest = prev[dest];
    }
    reverse(path.begin(), path.end());
}
```

5.4 Bridges Kareem

```
//undirected
vector<int>adj[N];
int dfn[N],low[N],timer;
void bridges(int node,int parent){
    dfn[node] = low[node] = ++timer;
    for(auto&ch : adj[node]){
        if(ch == parent)continue;
        if(dfn[ch] == -1){
            bridges(ch,node);
            low[node] = min(low[node],low[ch]);
            if(dfn[node] < low[ch]){
                // is_bridge
            }
        }else{
            low[node] = min(low[node],dfn[ch]);
        }
    }
}
```

5.5 Bridges Reda

```
vsi G;
vi dn, dlow, pr;
int id = 0, n;
set<pair<int, int>> out;
void bridges(int node)
{
    if (dn[node] != -1)
       return;
    dn[node] = dlow[node] = ++id;
    for (auto u : G[node])
    {
       if (dn[u] == -1)
       {
          pr[u] = node;
          bridges(u);
          if (dn[node] < dlow[u])
             out.insert({ min(node, u), max(node, u) });
          dlow[node] = min(dlow[node], dlow[u]);
       }
       else if (u != pr[node])
       {
          dlow[node] = min(dlow[node], dn[u]);
       }
    }
}
void solve()
{
    out.clear();
    dlow = vi(n + 1);
    dn = pr = vi(n + 1, -1);
    G = vsi(n + 1);
// input graph
    for (int i = 0; i < n; i++)if (dn[i] == -1)bridges(i);
// out -> set with all bridges
}
```

5.6 Euler tour

```
void euler(vector<vector<int> >& adjMax, vector<int>& ret,
    int n, int i, bool isDirected = false)
{
    for (int j = 0; j < n; j++)
    {
       if (adjMax[i][j])
       {
          adjMax[i][j]--;
          if (!isDirected)
             adjMax[j][i]--;
          euler(adjMax, ret, n, j, isDirected);
       }
    }
    ret.push_back(i);
}
```

5.7 Tarjan SCC Reda

```
//TARJAN

#define vi vector<int>
#define vvi vector<vector<int>>
#define usi vector<set<int>>
//SCC
int n, m;
vvi G;
vi dn, dlow, onstack;
stack<int> st;
int id;
void tarjan(int node)
{
  if (~dn[node])return;
  ++id;
  dn[node] = dlow[node] = id;
  onstack[node] = 1;
  st.push(node);
  for (auto u : G[node])
  {
    if (dn[u] == -1)
    {
      tarjan(u);
      dlow[node] = min(dlow[node], dlow[u]);
    }
    if (onstack[u])
      dlow[node] = min(dlow[node], dlow[u]);
  }
  if (dlow[node] == dn[node])
  {
    int u = -1;
    while (u != node)
    {
      u = st.top();
      st.pop();
      onstack[u] = 0;
      dlow[u] = dlow[node];
    }
  }
}
void solve()
{
  cin >> n >> m;
  st = stack<int>();
  G = vvi(n + 1);
  id = 0;
  dn = vi(n + 2, -1);
  onstack = dlow = vi(n + 2);

// input graph
  for (int i = 1; i <= n; i++)if (dn[i] == -1)tarjan(i);
// same dlow value means on same cycle
}
```

6 Math

6.1 Combinatorics

```
/*
* nCr = n!/((n-r)! * r!)
* nCr(n,r) = nCr(n,n-r)
* nPr = n!/(n-r)!
* nPr(circle) = nPr/r
* nCr(n,r) = pascal[n][r]
* catalan[n] = nCr(2n,n)/(n+1)
*/
ull nCr(int n, int r) {
    if (r > n)
        return 0;
    r = max(r, n - r);
    ull ans = 1, div = 1, i = r + 1;
    while (i <= n) {
        ans *= i++;
        ans /= div++;
    }
    return ans;
}

ull nPr(int n, int r) {
    if (r > n)
        return 0;
    ull p = 1, i = n - r + 1;
    while (i <= n)
        p *= i++;
    return p;
}

// return catalan number n-th using dp O(n^2)//max = 35 then overflow
vector<ull> catalanNumber(int n) {
    vector<ull> catalan(n + 1);
    catalan[0] = catalan[1] = 1;
    for (int i = 2; i <= n; i++) {
        ull &rt = catalan[i];
        for (int j = 0; j < n; j++)
            rt += catalan[j] * catalan[n - j - 1];
    }
    return catalan;
}

// count number of paths in matrix n*m // go to right or down only
ull countNumberOfPaths(int n, int m) {
    return nCr(n + m - 2, n - 1);
}
```

6.2 Equations

```
// (a^n)%p=result, return minimum n
int getPower(int a, int result, int mod) {
    int sq = sqrt(mod);
    map<int, int> mp;
    ll r = 1;
    for (int i = 0; i < sq; i++) {
        if (mp.find(r) == mp.end())
            mp[r] = i;
        r = (r * a) % mod;
    }
    ll tmp = modInverse(r, mod);
    ll cur = result;
    for (int i = 0; i <= mod; i += sq) {
        if (mp.find(cur) != mp.end())
            return i + mp[cur];
        cur = (cur * tmp) % mod;//val/(a^sq)
    }
    return INF;
}
// need testing
// return a ^ 1 + a ^ 2 + a ^ 3 + .... a ^ k
ll sumPower(ll a, ll k, int mod) {
    if (k == 1) return a % mod;
    ll half = sumPower(a, k / 2, mod);
    ll p = half * power(a, k / 2, mod) % mod;
    p = (p + half) % mod;
    if (k & 1) p = (p + power(a, k, mod)) % mod;
    return p;
}
// same function but faster (not tested)
```

```
int calci_xpi(int x, int n)
{
    int p1 = fix(fix(x * (1 - (modInv(Bpow(x, n))))) * modInv((x - 1) * (x - 1)));
    int p2 = fix(n * fix(modInv((x - 1) * Bpow(x, n))));
    return fix(p1 - p2);
}
//return sum of sequence a, a+x , a+2x .... b(not tested)
ll sumSequence(ll a, ll b, ll x) {
    a = ((a + x - 1) / x) * x;
    b = (b / x) * x;
    return (b + a) * (b - a + x) / (2 * x);
}
```

6.3 Fast Power

```
ll fpow(ll x, ll n, int mod) {
    if (n == 0)return 1 % mod;
    if (n == 1)return x % mod;
    ll ans = fpow(x, n / 2, mod);
    ans = ans * ans % mod;
    if (n & 1)ans = ans * (x % mod) % mod;
    return ans;
}


 // iterative
ll fpow(ll x, ll k, ll mod) {
  ll res = 1;
  for (x %= mod; k; k >>= 1, x = x * x % mod)
    if (k & 1) res = res * x % mod;
  return res;
}

// minimal
ll fast(ll b, ll e){
    if(!e)return 1;
    return fast(b * b % mod, e >> 1) * ((e&1) ? b : 1) % mod;
}

ll modInverse(ll b, ll mod) { // if mod is Prime
    return power(b, mod - 2, mod);
}
**if mod is not Prime,gcd(a,b) must be equal 1
ll modInverse(ll b, ll mod) {
    return power(b, phi_function(mod) - 1, mod);
}
```

6.4 Gauss

```
class Gauss {
    const int INF = 2;
    const double EPS = 1E-9;
public:
    int gauss(vector<vector<double> > a, vector<double> &ans) {
```

```
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where(m, -1);
    for (int col = 0, row = 0; col < m && row < n; ++col) {
        int sel = row;
        for (int i = row; i < n; ++i)
            if (abs(a[i][col]) > abs(a[sel][col]))
                sel = i;
        if (abs(a[sel][col]) < EPS)
            continue;
        for (int i = col; i <= m; ++i)
            swap(a[sel][i], a[row][i]);
        where[col] = row;

        for (int i = 0; i < n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j = col; j <= m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

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

    for (int i = 0; i < m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}

int gaussWithMod(vector<vector<ll>> a, vector<ll> &ans, ll p) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where(m, -1);
    for (int col = 0, row = 0; col < m && row < n; ++col) {
        int sel = row;
        for (int i = row; i < n; ++i)
            if (a[i][col] > a[sel][col])
                sel = i;
        if (a[sel][col] == 0)
            continue;
        for (int i = col; i <= m; ++i)
            swap(a[sel][i], a[row][i]);
        where[col] = row;

        ll inv = modInverse(a[row][col], p);
        for (int i = 0; i < n; ++i) {
            if (i != row && a[i][col] != 0) {
                ll c = a[i][col] * inv % p;
                for (int j = col; j <= m; ++j) {
                    a[i][j] = (a[i][j] - a[row][j] * c % p + p) % p;
                }
            }
        }
        ++row;
    }

    ans.assign(m, 0);
    for (int i = 0; i < m; ++i)
        if (where[i] != -1) {
            ll inv = modInverse(a[where[i]][i], p);
            ans[i] = (a[where[i]][m] * inv) % p;
        }


    for (int i = 0; i < n; ++i) {
        ll sum = 0;
        for (int j = 0; j < m; ++j)
            sum = (sum + ans[j] * a[i][j]) % p;
        if (sum != a[i][m])
            return 0; // No solution
    }

    for (int i = 0; i < m; ++i)
        if (where[i] == -1)
            return INF; // Infinite solutions

    return 1; // Unique solution
}

int gauss(vector<bitset<N>> a, int n, int m, bitset<N> &ans) {
    vector<int> where(m, -1);
    for (int col = 0, row = 0; col < m && row < n; ++col) {
        for (int i = row; i < n; ++i) {
            if (a[i][col]) {
                swap(a[i], a[row]);
                break;
            }
        }
        if (!a[row][col])
            continue;

        where[col] = row;
        for (int i = 0; i < n; ++i) {
            if (i != row && a[i][col])
                a[i] ^= a[row];  // XOR the rows
        }
        ++row;
    }

    ans.reset();
    for (int i = 0; i < m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m];

    for (int i = 0; i < n; ++i) {
        bool sum = 0;
        for (int j = 0; j < m; ++j)
            sum ^= (ans[j] & a[i][j]);  // XOR the known values
        if (sum != a[i][m])
            return 0;  // No solution
    }

    for (int i = 0; i < m; ++i)
        if (where[i] == -1)
            return INF;  // Infinite solutions

    return 1;  // Unique solution
}

int compute_rank(vector<vector<double>> A) {
    int n = A.size();
    int m = A[0].size();

    int rank = 0;
    vector<bool> row_selected(n, false);
    for (int i = 0; i < m; ++i) {
        int j;
        for (j = 0; j < n; ++j) {
            if (!row_selected[j] && abs(A[j][i]) > EPS)
                break;
        }
        if (j != n) {
            ++rank;
            row_selected[j] = true;
            for (int p = i + 1; p < m; ++p)
                A[j][p] /= A[j][i];
            for (int k = 0; k < n; ++k) {
                if (k != j && abs(A[k][i]) > EPS) {
                    for (int p = i + 1; p < m; ++p)
                        A[k][p] -= A[j][p] * A[k][i];
                }
            }
        }
    }
    return rank;
}
```

```
};
```

6.5 Linear Diophantine Equation CRT

```
ll exgcd(ll a,ll b, ll  &x, ll &y){
    if (a<0||b<0){
        ll g=exgcd(abs(a),abs(b),x,y);
        if (a<0)x*=-1;
        if (b<0)y*=-1;
        return g;
    }
    if (b == 0){
        x=1,y=0;
        return a;
    }
    ll g= exgcd(b,a%b,y,x);
    y-=(a/b)*x;
    return g;
}
ll ldioph(ll a,ll b, ll c,ll &x,ll &y, bool &found){
    ll g=exgcd(a,b,x,y);
    if (c%g){
        found=false;
        return g;
    }
    found=true;
    x*=c/g;
    y*=c/g;
    return g;
}
pair<ll,ll>CRT(const vector<ll>&a, const vector<ll>&m){
    ll rem=a[0],mod=m[0];
    int n=a.size();
    for (int i=1;i<n;i++){
        ll x,y;
        bool found=0;
        ll g= ldioph(mod,-m[i],a[i]-rem,x,y,found);

        if (!found)
            return {-1,-1};
        rem+=mod*x;
        mod=mod/g*m[i];
        rem=(rem%mod+mod)%mod;
    }
    return {rem,mod};

}
```

6.6 Lucus Theorem

```
int N = 1e6 + 3, mod = 1e6 + 3;
// pre in O(mod),
struct combi
{
  int n;
  vector<int> facts, finvs, invs;
  combi(int _n) : n(_n), facts(_n), finvs(_n), invs(_n)
  {
    facts[0] = finvs[0] = 1;
    invs[1] = 1;
    for (int i = 2; i < n; i++)
    {
      invs[i] = 1LL * invs[mod % i] * (mod - mod / i) % mod;
    }
    for (int i = 1; i < n; i++)
    {
      facts[i] = 1LL * facts[i - 1] * i % mod;
      finvs[i] = 1LL * finvs[i - 1] * invs[i] % mod;
    }
  }
  inline int ncr(int x, int y)
  {
    if (y > x || y < 0) return 0;
    return 1LL * facts[x] * finvs[y] % mod * finvs[x - y] % mod;
  }
};
combi C(N);
// Computes nCr % mod using Lucas' Theorem when mod is a prime
int lucas(ll n, ll r)
{
  if (r > n) return 0;
  if (n < mod) return C.ncr(n, r);
  return 1LL * lucas(n / mod, r / mod) * lucas(n % mod, r % mod) % mod;
}
```

6.7 Matrix power

```
// const int M = 2;
// typedef array<array<int, M>, M> matrix;

int mod = 1e9+7;
class matrix{
public:
    vector<vector<ll>>v;
    int n;
    matrix(int sz1){
        n = sz1;
        v = vector<vector<ll>>(n, vector<ll>(n));
    }
    matrix operator *(const matrix&a){
        matrix res(n);
        for(int i=0;i<n;i++){
            for(int k=0;k<n;k++){
                //if(v[i][k] == 0)continue;
                for(int j=0;j<n;j++){
                    res.v[i][j] +=  v[i][k] * a.v[k][j];
                    res.v[i][j]%=mod;
                }
            }
        }
        return res;
    }
    void ones(){
        for(int i=0;i<n;i++)v[i][i] = 1;
    }
};
matrix mpow(matrix a,ll k){
    matrix ans(a.n);
    ans.ones();
    while(k){
        if(k&1){
            ans = ans*a;
        }
        a = a*a;
        k>>=1;
    }
    return ans;
}
```

6.8 MillerRabin

```
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n-2] if
//The largest known gap between consecutive primes  1e9 is 282,   1e12   1132
using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128) result * base % mod;
        base = (u128) base * base % mod;
        e >>= 1;
    }
    return result;
}
bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128) x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
}
```

```cpp
bool MillerRabin(u64 n) { // returns true if n is prime, else returns false.
    if (n < 2)
        return false;

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

    for (int a: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

6.9 Mobius Function

```cpp
const int N = 2e6 + 10, MOD = 1e9 + 7;
int mob[N];
bool prime[N];
void mobius() {
  // fix this ya 3amy @abdosa3d
    memset(mob, 1, sizeof mob);
    memset(prime + 2, 1, sizeof(prime) - 2);
    mob[0] = 0;
    mob[2] = -1;
    for (int i = 4; i < N; i += 2) {
      // fix this too
        mob[i] *= (i & 3) ? -1 : 0;
        prime[i] = 0;
    }
    for (int i = 3; i < N; i += 2)
        if (prime[i]) {
            mob[i] = -1;
            for (int j = 2 * i; j < N; j += i) {
                mob[j] *= (j % (1LL * i * i)) ? -1 : 0;
                prime[j] = 0;
            }
        }
}


const int N = 4e7 + 10;
char mu[N];
vector<bool> comp(N);

void mobius_linear() {
    vector<int> ps;
    ps.reserve(N / 10);
    mu[1] = 1;
    for (int i = 2; i < N; ++i) mu[i] = 0; // optional, will be set

    for (int i = 2; i < N; ++i) {
        if (!comp[i]) {
            ps.push_back(i);
            mu[i] = -1;
        }
        for (int p: ps) {
            long long v = 1LL * i * p;
            if (v >= N) break;
            comp[v] = true;
            if (i % p == 0) {
                mu[v] = 0;
                break;
            } // square divides
            mu[v] = -mu[i];
        }
    }
}
```

6.10 NCR Preprocessing

```
const int N = 1e6 + 100;
const int mod = 1e9 + 7;
ll fact[N];
ll inv[N]; //mod inverse for i
ll invfact[N]; //mod inverse for i!
void factInverse() {
    fact[0] = inv[1] = fact[1] = invfact[0] = invfact[1] = 1;
    for (long long i = 2; i < N; i++) {
        fact[i] = (fact[i - 1] * i) % mod;
        inv[i] = mod - (inv[mod % i] * (mod / i) % mod);
        invfact[i] = (inv[i] * invfact[i - 1]) % mod;
    }
}

ll nCr(int n, int r) {
    if (r > n) return 0;
    return (((fact[n] * invfact[r]) % mod) * invfact[n - r]) %
           mod;
}
```

6.11 Phi

```
const int N=1e6+5;
int phi[N];
void pre(){
    for (int i=0;i<N;i++)
        phi[i]=i;
    for (int i=2;i<N;i++){
        if (phi[i]==i){
            for (int j=i;j<N;j+=i)
                phi[j]-=phi[j]/i;
        }
    }

}

ll phi(ll n){
    ll p_to_k, relative_primes=1;
    for (ll i=2,d=1;i*i<=n;i+=d,d=2){
        if (!(n%i)){
            p_to_k=1;
            while(!(n%i)){
                p_to_k*=i,n/=i;
            }
            relative_primes*=(p_to_k/i)*(i-1);

        }

    }
    if (n!=1){
        relative_primes*=(n-1);
    }
    return relative_primes;
}

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

6.12 Sieve w e5wato

```
get prime factors using sieve (less than sqrt(n))
vector<int> prime_fact(int n)
{
  vector<int>temp;
  for (int i = 0;primes[i] * 1LL * primes[i] <= n;i++)
  {
    while (n % primes[i] == 0)
      temp.push_back(primes[i]), n /= primes[i];
  }
  if (n > 1)temp.push_back(n);
  return temp;
}

segmented_sieve --> get primes in range ex(1e9:1e9+1e6) o[(r-l)*loglog(r)+ o(sieve)]
```

```
 r
 const int N=1e5+5;
 bool composite[N];
void segmented_sieve()
{
  for(auto i:primes)
  {
    for(int j=max(i*i,(l+i-1)/i*i);j<=r;j+=i)
        composite[j-l]=1;                   //
  }
  if(l==1)composite[0]=1;
}

//Linear Sieve
const int N = 1e7;
int lpf[N + 1];
vector<int> prime;

void sieve() {
    for (int i = 2; i <= N; i++) {
        if (lpf[i] == 0) {
            lpf[i] = i;
            prime.push_back(i);
        }
        for (int j: prime) {
            if (j > lpf[i] || 1LL * i * j > N)break;
            lpf[i * j] = j;
        }
    }
}
```

6.13 SieveUpTo1e9

```
//about 5e7 primes up to 1e9
vector<int> sieve(const int N = int(1e9), const int Q = 17, const int L = 1 << 15) {
    static const int rs[] = {1, 7, 11, 13, 17, 19, 23, 29};
    struct P {
        P(int p) : p(p) {}
        int p;
        int pos[8];
    };
    auto approx_prime_count = [](const int N) -> int {
        return N > 60184 ? N / (log(N) - 1.1) : max(1., N / (log(N) - 1.11)) + 1;
    };
    const int v = sqrt(N), vv = sqrt(v);
    vector<bool> isp(v + 1, true);
    for (int i = 2; i <= vv; ++i)
        if (isp[i]) {
            for (int j = i * i; j <= v; j += i)
                isp[j] = false;
        }
    const int rsize = approx_prime_count(N + 30);
    vector<int> primes = {2, 3, 5};
    int psize = 3;
```

```
primes.resize(rsize);
vector<P> sprimes;
size_t pbeg = 0;
int prod = 1;
for (int p = 7; p <= v; ++p) {
    if (!isp[p]) continue;
    if (p <= Q)prod *= p, ++pbeg, primes[psize++] = p;
    auto pp = P(p);
    for (int t = 0; t < 8; ++t) {
        int j = (p <= Q) ? p : p * p;
        while (j % 30 != rs[t]) j += p << 1;
        pp.pos[t] = j / 30;
    }
    sprimes.push_back(pp);
}
vector<unsigned char> pre(prod, 0xFF);
for (size_t pi = 0; pi < pbeg; ++pi) {
    auto pp = sprimes[pi];
    const int p = pp.p;
    for (int t = 0; t < 8; ++t) {
        const unsigned char m = ~(1 << t);
        for (int i = pp.pos[t]; i < prod; i += p)pre[i] &= m;
    }
}
const int block_size = (L + prod - 1) / prod * prod;
vector<unsigned char> block(block_size);
unsigned char *pblock = block.data();
const int M = (N + 29) / 30;
for (int beg = 0; beg < M; beg += block_size, pblock -= block_size) {
    int end = min(M, beg + block_size);
    for (int i = beg; i < end; i += prod) {
        copy(pre.begin(), pre.end(), pblock + i);
    }
    if (beg == 0) pblock[0] &= 0xFE;
    for (size_t pi = pbeg; pi < sprimes.size(); ++pi) {
        auto &pp = sprimes[pi];
        const int p = pp.p;
        for (int t = 0; t < 8; ++t) {
            int i = pp.pos[t];
            const unsigned char m = ~(1 << t);
            for (; i < end; i += p)pblock[i] &= m;
            pp.pos[t] = i;
        }
    }
    for (int i = beg; i < end; ++i) {
        for (int m = pblock[i]; m > 0; m &= m - 1) {
            primes[psize++] = i * 30 + rs[__builtin_ctz(m)];
        }
    }
}
assert(psize <= rsize);
while (psize > 0 && primes[psize - 1] > N)--psize;
primes.resize(psize);
return primes;
```

```
}
```

6.14 SumRangeDivisors

```
// return sum of divisors for all number from 1 to n //O(n)
ll sumRangeDivisors(int n) {
    ll ans = 0;

    for (int x = 1; x <= n; x++)
        ans += (n / x) * x;
    return ans;
}
// calc 1e9 in 42ms,can calc more but need big integer
ll sumRangeDivisors(ll x) {
    ll ans = 0, left = 1, right;
    for (; left <= x; left = right + 1) {
        right = x / (x / left);
        ans += (x / left) * (left + right) * (right - left + 1) / 2;
    }
    return ans;
}
```

6.15 FFT w e5wato

6.15.1 FFT Iterative

```
const double PI = acos(-1);
typedef complex<double> cd;

void fft(vector<cd> &a, bool invert) {
    int n = a.size();
    // bit reversal permutation
    for (int i = 0, j = 0; i < n; i++) {
        if (i < j)swap(a[i], a[j]);
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)j ^= bit;
        j ^= bit;
    }
    for (int ln = 2; ln <= n; ln <<= 1) {
        double angle = 2 * PI / ln;
        cd wln(cos(angle), sin(angle) * (invert ? -1 : 1));
        for (int j = 0; j < n; j += ln) {
            cd w(1);
            for (int i = 0; i < ln / 2; i++) {
                cd temp = a[i + j];
                a[i + j] = a[i + j] + w * a[i + j + ln / 2];
                a[i + j + ln / 2] = temp - w * a[i + j + ln / 2];
                w *= wln;
                if (invert) {
                    a[i + j] /= 2;
                    a[i + j + ln / 2] /= 2;
                }
            }
        }
    }
}
vector<ll> mul(vector<ll> &a, vector<ll> &b) {
    int n = 1;
    while (n < a.size() + b.size())n <<= 1;
    vector<cd> fa(all(a)), fb(all(b));
    fa.resize(n);
    fb.resize(n);
    fft(fa, 0);
    fft(fb, 0);
    for (int i = 0; i < n; i++) {
        fa[i] *= fb[i];
    }
    fft(fa, 1);
    vector<ll> res(n);
    for (int i = 0; i < n; i++) {
        res[i] = round(fa[i].real());
    }
    return res;
}
```

6.15.2 FFT MOD

```
#define rep(aa, bb, cc) for(int aa = bb; aa < cc;aa++)
#define sz(a) (int)a.size()
#define vi vector<int>
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C> &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1);  // (^ 10% faster if double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n);
        rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
    }
    vi rev(n);
    rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k)
            rep(j, 0, k) {
                // C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)
                // include-line
                auto x = (double *) &rt[j + k], y = (double *) &a[i + j + k];
↪              /// exclude-line
                C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] * y[0]);
↪              /// exclude-line
                a[i + j + k] = a[i + j] - z;
                a[i + j] += z;
            }
}
```

```
template<int M>
vi convMod(const vi &a, const vi &b) {
    if (a.empty() || b.empty()) return {};
    vi res(sz(a) + sz(b) - 1);
    int B = 32 - __builtin_clz(sz(res)), n = 1 << B, cut = int(sqrt(M));
    vector<C> L(n), R(n), outs(n), outl(n);
    rep(i, 0, sz(a)) L[i] = C((int) a[i] / cut, (int) a[i] % cut);
    rep(i, 0, sz(b)) R[i] = C((int) b[i] / cut, (int) b[i] % cut);
    fft(L), fft(R);
    rep(i, 0, n) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
    }
    fft(outl), fft(outs);
    rep(i, 0, sz(res)) {
        ll av = int64_t(real(outl[i]) + .5), cv = int64_t(imag(outs[i]) + .5);
        ll bv = int64_t(imag(outl[i]) + .5) + int64_t(real(outs[i]) + .5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res;
}
```

6.15.3 FFT string Matching

```
using cd = complex<double>;
const double PI = acos(-1), eps = 5e-4; // If you get a wrong answer you can change the
↪  eps lower of higher till you pass

void fft(vector<cd> & a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }

    if (invert) {
        for (cd & x : a)
            x /= n;
    }
}

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

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

    return fa;
}

void solve(int tc) {

    string s, patt; cin >> s >> patt;
    int n = (int)s.length(), m = (int)patt.length();

    vector<cd> poly1(n), poly2(m);

    for (int i = 0; i < n; ++i) {
        double angle = 2*PI*(s[i]-'a')/26;
        poly1[i] = cd(cos(angle), sin(angle));
    }
    for (int i = 0; i < m; ++i) {
        if(patt[m-i-1] == '*') poly2[i] = cd(0,0);
        else {
            double angle = 2*PI*(patt[m-i-1]-'a')/26;
            poly2[i] = cd(cos(angle), -sin(angle));
        }
    }

    vector<cd> ans = multiply(poly1, poly2);
    int wild_cnt = (int)count(patt.begin(), patt.end(), '*');

    int tot = 0;
    vector<int> pos;
    for (int i = 0; i < n; ++i) {
        if(fabs(ans[m-1+i].real() - (m - wild_cnt)) < eps && fabs(ans[m-1+i].imag()) <
↪   eps) {
            ++tot;
            pos.push_back(i);
        }
    }

    cout << tot << "\n";
    for(auto & p : pos) cout << p << " ";
    cout << "\n";

}
```

6.15.4 FFT

```
const double PI = acos(-1);
typedef complex<double> cd;
void fft(vector<cd>&a,bool invert){
    int n = a.size();
    if(n == 1)return;
    vector<cd>a0(n/2),a1(n/2);
    for(int i=0;i*2<n;i++){
        a0[i] = a[i*2];
        a1[i] = a[i*2+1];
    }
    fft(a0,invert);
    fft(a1,invert); // a(x) = a0(x^2) + x * a0(x^2)
    double angle = 2*PI/n * (invert ? -1 : 1);
    cd w = 1, wn(cos(angle),sin(angle));
    for(int i=0;i<n/2;i++){
        a[i] = a0[i] + w * a1[i];
        a[i + n/2] = a0[i] - w * a1[i];
        w*= wn;
        if(invert){
            a[i]/=2;
            a[i + n/2]/=2;
        }
    }
}
vector<ll> multiply(vector<ll>&a,vector<ll>&b){
    int n = 1;
    while(n < sz(a) + sz(b))n<<=1;
    vector<cd>fa(all(a)),fb(all(b));
    fa.resize(n);
    fb.resize(n);
    fft(fa,0);
    fft(fb,0);
    for(int i=0;i<n;i++){
        fa[i]*=fb[i];
    }
    fft(fa,1);
    vector<ll>res(n);
    for(int i=0;i<n;i++){
        res[i] = round(fa[i].real());
    }
    return res;
}
```

6.15.5 FWHT

```
const int mod = 1e9 + 7;

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

const int inv2 = (mod + 1) >> 1;
#define M (1 << 20)
#define OR 0
#define AND 1
#define XOR 2

struct FWHT {
    int P1[M], P2[M];

    void wt(int *a, int n, int flag = XOR) {
        if (n == 0) return;
        int m = n / 2;
        wt(a, m, flag);
        wt(a + m, m, flag);
        for (int i = 0; i < m; i++) {
            int x = a[i], y = a[i + m];
            if (flag == OR) a[i] = x, a[i + m] = (x + y) % mod;
            if (flag == AND) a[i] = (x + y) % mod, a[i + m] = y;
            if (flag == XOR) a[i] = (x + y) % mod, a[i + m] = (x - y + mod) % mod;
        }
    }

    void iwt(int *a, int n, int flag = XOR) {
        if (n == 0) return;
        int m = n / 2;
        iwt(a, m, flag);
        iwt(a + m, m, flag);
        for (int i = 0; i < m; i++) {
            int x = a[i], y = a[i + m];
            if (flag == OR) a[i] = x, a[i + m] = (y - x + mod) % mod;
            if (flag == AND) a[i] = (x - y + mod) % mod, a[i + m] = y;
            if (flag == XOR) a[i] = 1LL * (x + y) * inv2 % mod, a[i + m] = 1LL * (x - y
↪   + mod) * inv2 % mod;
            // replace inv2 by >>1 if not required
        }
    }

    vector<int> multiply(int n, vector<int> A, vector<int> B, int flag = XOR) {
        assert(__builtin_popcount(n) == 1);
        A.resize(n);
        B.resize(n);
        for (int i = 0; i < n; i++) P1[i] = A[i];
        for (int i = 0; i < n; i++) P2[i] = B[i];
        wt(P1, n, flag);
        wt(P2, n, flag);
        for (int i = 0; i < n; i++) P1[i] = 1LL * P1[i] * P2[i] % mod;
        iwt(P1, n, flag);
        return vector<int>(P1, P1 + n);
    }

    vector<int> pow(int n, vector<int> A, long long k, int flag = XOR) {
        assert(__builtin_popcount(n) == 1);
        A.resize(n);
        for (int i = 0; i < n; i++) P1[i] = A[i];
        wt(P1, n, flag);
        for (int i = 0; i < n; i++) P1[i] = POW(P1[i], k);
        iwt(P1, n, flag);
        return vector<int>(P1, P1 + n);
    }
} t;
```

6.15.6 NTT

```
const ll mod = (119 << 23) + 1, root = 3; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
ll modpow(ll b, ll e) {
    ll ans = 1;
    for (; e; b = b * b % mod, e /= 2)
        if (e & 1) ans = ans * b % mod;
    return ans;
}

// Primitive Root of the mod of form 2^a * b + 1
int generator() {
    vector<int> fact;
    int phi = mod - 1, 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 (int res = 2; res <= mod; ++res) {
        bool ok = true;
        for (size_t i = 0; i < fact.size() && ok; ++i)
            ok &= modpow(res, phi / fact[i]) != 1;
        if (ok) return res;
    }
    return -1;
}

ll modpow(ll b, ll e, ll m) {
    ll ans = 1;
    for (; e; b = b * b % m, e /= 2)
        if (e & 1) ans = ans * b % m;
    return ans;
}

void ntt(vector<ll> &a) {
    int n = (int) a.size(), L = 31 - __builtin_clz(n);
    static vector<ll> rt(2, 1); // erase the static if you want to use two moduli;
    for (static int k = 2, s = 2; k < n; k *= 2, s++) { // erase the static if you want
↪   to use two moduli;
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s, mod)};
        for (int i = k; i < 2 * k; ++i) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    vector<int> rev(n);
    for (int i = 0; i < n; ++i) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    for (int i = 0; i < n; ++i) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; ++j) {
                ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
                a[i + j + k] = ai - z + (z > ai ? mod : 0);
                ai += (ai + z >= mod ? z - mod : z);
            }
        }
    }
}

vector<ll> conv(const vector<ll> &a, const vector<ll> &b) {
    if (a.empty() || b.empty()) return {};
    int s = (int) a.size() + (int) b.size() - 1, B = 32 - __builtin_clz(s), n = 1 << B;
    int inv = modpow(n, mod - 2, mod);
    vector<ll> L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    for (int i = 0; i < n; ++i) out[-i & (n - 1)] = (ll) L[i] * R[i] % mod * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}

ll CRT(ll a, ll m1, ll b, ll m2) {
    __int128 m = m1 * m2;
    ll ans = a * m2 % m * modpow(m2, m1 - 2, m1) % m + m1 * b % m * modpow(m1, m2 - 2,
↪   m2) % m;
    return ans % m;
}
```

7 misc

7.1 Base Conversion

```
string letters = "0123456789ABCDEF";

int toInt(char c)
{
    return letters.find(c);
}

int FromAnyBasetoDecimal(string in, int base)
{
    int res = 0;
    for (int i = 0; i < in.size(); ++i)
        res *= base, res += toInt(in[i]);
    return res;
}

string FromDecimaltoAnyBase(int number, int base)
{
    if (number == 0)
        return "0";
    string res = "";
    for (; number; number /= base)
        res = letters[number % base] + res;
    return res;
}

string toNegativeBase(int n, int negBase)
{
    if (n == 0)
        return "0";
    string ans = "";
    while (n != 0)
    {
        int rem = n % negBase;
        n /= negBase;
        if (rem < 0)
        {
            rem += (-negBase);
            n += 1;
        }
        ans += to_string(rem);
    }
    reverse(all(ans));
    return ans;
}

void print(int x)
{
    if (x <= 1)
    {
        cout << x;
        return;    }
    print(x >> 1);
    cout << (x & 1);
}
```

7.2 BuiltIn functions

```
ll mxb(ll x)
{
   if (!x)
      return x;
   return 1LL << (63 - __builtin_clzll(x));
}
// count leading zeros
__builtin_clzll(x)

// count on bits
__builtin_popcountll(x)

// to get lowbit (x&-x)
```

7.3 CheckTime

```
auto start = chrono::high_resolution_clock::now();
  // code    auto stop = chrono::high_resolution_clock::now();
  auto duration = chrono::duration_cast<chrono::nanoseconds>(stop - start);
  cerr << "Time taken :" << ((ld)duration.count()) / ((ld)1e9) << " s" << endl;
```

7.4 CompareFunction for ds

```
class comp {
    public:
       bool operator()(T a, T b){
           if(cond){
               return true;
           }
           return false;
      }
};

priority_queue<data_type, container, comparator> ds;
```

7.5 DP digits

```
int n;
vector<int>v1, v2;
ll dp[20][2][2];
ll fun(int idx, bool l,bool r)
{

  if (idx == n)return 1;
  ll& ret = dp[idx][l][r];
  if (~ret)return ret;
  ret = 0;
  int lim1 = l? 0 : v1[idx];
  int lim2 = r? 9 : v2[idx];

  for (int i = lim1;i <=lim2;i++)
  {
    bool temp1 = i > v1[idx];
    bool temp2 = i < v2[idx];
      ret += fun(idx + 1, l|temp1, r|temp2);
  }
  return ret;
}
void solve()
{
  memset(dp, -1, sizeof dp);
  ll x, y;
  cin >> x >> y;
  while (x)v1.push_back(x % 10), x /= 10;
  while (y)v2.push_back(y % 10), y /= 10;
  while (v1.size() < v2.size())v1.push_back(0);
  while (v2.size() < v1.size())v2.push_back(0);
  reverse(all(v1));
  reverse(all(v2));
  n = v1.size();
  cout << fun(0, 0, 0, 0) << endl;
  v1.clear();
  v2.clear();
}
```

7.6 DP SOS Kareem

```
const int LG = 22;
const int M = 1 << LG;

// subset contribute to its superset
void forward1(vector<ll>&dp) {
    for (int bt = 0; bt < LG; ++bt) {
        for (int m = 0; m < M; ++m) {
            if (m >> bt & 1){
                dp[m] += dp[m ^ (1 << bt)];
            }
        }
    }
}

// superset contribute to its subset
void forward2(vector<ll>&dp) {
    for (int bt = 0; bt < LG; ++bt) {
        for (int m = M - 1; m >= 0; m--) {
            if (m >> bt &1){
                dp[m ^ (1 << bt)] += dp[m];
            }
        }
    }
}
```

```
// remove subset contribution from superset
void backward1(vector<ll>&dp) {
    for (int bt = 0; bt < LG; bt++){
        for (int m = M - 1; m >= 0; m--){
            if (m >> bt &1){
                dp[m] -= dp[m ^ (1 << bt)];
            }
        }
    }
}

// remove superset contribution from subset
void backward2(vector<ll> &dp) {
    for (int bt = 0; bt < LG; bt++){
        for (int m = 0; m < M; m++){
            if (m >> bt &1){
                dp[m ^ (1 << bt)] -= dp[m];
            }
        }
    }
}
```

7.7 Generate All Submasks

```
void genAllSubmask(int mask) {
    for (int subMask = mask;; subMask = (subMask - 1) & mask) {
//code
        if (subMask == 0)
            break;
    }
}
```

7.8 Li-Chao

```
const int LC_N = (int)1e6 + 1;
const long long LC_INF = (long long)1e17;
vector<array<long long,2>> lc_tree(4 * LC_N, {0, LC_INF});

long long f_line(const array<long long,2>& line, int x) {
    return line[0] * x + line[1];
}

void lc_insert(array<long long,2> line, int lo = 1, int hi = LC_N, int i = 1) {
    int m = (lo + hi) / 2;
    bool left = f_line(line, lo) < f_line(lc_tree[i], lo);
    bool mid  = f_line(line, m)  < f_line(lc_tree[i], m);

    if (mid) swap(lc_tree[i], line);
    if (hi - lo == 1) return;

    if (left != mid)
        lc_insert(line, lo, m, 2 * i);
    else
```

```
        lc_insert(line, m, hi, 2 * i + 1);
}

long long lc_query(int x, int lo = 1, int hi = LC_N, int i = 1) {
    int m = (lo + hi) / 2;
    long long curr = f_line(lc_tree[i], x);

    if (hi - lo == 1) return curr;

    if (x < m)
        return min(curr, lc_query(x, lo, m, 2 * i));
    else
        return min(curr, lc_query(x, m, hi, 2 * i + 1));
}
```

7.9 LIS Onlogn

```
int LIS(const vector<int> &v) {
    vector<int> lis(v.size());//put value less than zero if needed
    int l = 0;
    for (int i = 0; i < sz(v); i++) {
        int idx = lower_bound(lis.begin(), lis.begin() + l, v[i]) - lis.begin();
        if (idx == l)
            l++;
        lis[idx] = v[i];
    }
    return l;
}
```

7.10 misereNim

```
string misereNim(const vector<int>& heaps) {
    int ones = 0;
    int moreThanOne = 0;
    int nimSum = 0;

    for (int h : heaps) {
        if (h == 1) ones++;
        else moreThanOne++;
        nimSum ^= h;
    }

    if (moreThanOne == 0) {
        // All heaps are 1
        return (ones % 2 == 0 ? "Win" : "Lose");
    } else if (moreThanOne == 1) { // remove this condition @saad
        // One heap > 1
        return (ones % 2 == 1 ? "Win" : "Lose");
    } else {
        // General case
        return (nimSum == 0 ? "Lose" : "Win");
    }
```

7.11 Next element

```
int n;
const int N = 1e5+5;
vector<int>v(N);
  //
vector<int>next_mx()
{
  stack<int>st;
  st.push(n + 1);
  vector<int>suf;
  v[n + 1] = 2e9;
  for (int i = n;i > 0;i--)
  {
    while (v[i] >= v[st.top()])st.pop();
    suf.push_back(st.top());
    st.push(i);
  }
  suf.push_back(0);
  reverse(all(suf));
  return suf;
}

vector<int>next_mn()
{
  stack<int>st;
  st.push(n + 1);
  vector<int>suf;
  v[n + 1] = -2e9;
  for (int i = n;i > 0;i--)
  {
    while (v[i] <= v[st.top()])st.pop();
    suf.push_back(st.top());
    st.push(i);
  }
  suf.push_back(0);
  reverse(all(suf));
  return suf;
}

vector<int>prev_mx()
{
  vector<int>pre;
  stack<int>st;
  pre.push_back(0);
  st.push(0);
  v[0] = 2e9;
  for (int i = 1;i <= n;i++)
  {
    while (v[i] >= v[st.top()])st.pop();
    pre.push_back(st.top());
    st.push(i);
  }
  return pre;
}

vector<int>prev_mn()
{
  vector<int>pre;
  stack<int>st;
  pre.push_back(0);
  st.push(0);
  v[0] = -2e9;
  for (int i = 1;i <= n;i++)
  {
    while (v[i] <= v[st.top()])st.pop();
    pre.push_back(st.top());
    st.push(i);
  }
  return pre;
}
```

7.12 Random

```
-#include <random>

static random_device rd;
static mt19937 gen(rd());

int randomgen(int x)
{
    uniform_int_distribution<> dis(0, x - 1);
    int rndnum = dis(gen);
    return rndnum;
}

#include <chrono>
#include <random>

//write this line once in top
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count() *
               ((uint64_t) new char | 1));

// use this instead of rand()
template<typename T>
T Rand(T low, T high) {
    return uniform_int_distribution<T>(low, high)(rng);
}
```

7.13 Ternary search

```
// 1 d ternary search

bool can(int mid) {}
int ternary_search(int l, int r)
{
  int ans;
  while (l <= r)
  {
    int mid1 = l + (r - l) / 3;
    int mid2 = r - (r - l) / 3;
    if (can(mid1) > can(mid2))ans=mid1, r = mid2;
    else ans=mid2, l = mid1;
  }
  return ans;
}
// 2d, beware if dealing with double remove the +-1
double get(int row,int mid);
double ans = 1e18;
double ternary1(int row)
{
    int l = -1e9 - 5, r = 1e9 + 5, mid1, mid2;
    double res;
    while (l <= r) {
        mid1 = l + (r - l) / 3;
        mid2 = r - (r - l) / 3;
        double res1 = get(row, mid1);
        double res2 = get(row, mid2);
        if (res1 > res2)
            l = mid1 + 1, res = res2;
        else
            r = mid2 - 1, res = res1;
    }
    return res;
}
void ternary2()
{
    int l = -1e9 - 5, r = 1e9 + 5, mid1, mid2;
    while (l <= r) {
        mid1 = l + (r - l) / 3;
        mid2 = r - (r - l) / 3;
        double res1 = ternary1(mid1);
        double res2 = ternary1(mid2);
        if (res1 > res2)
        {
            ans = min(ans, res2);
            l = mid1 + 1;
        }
        else
        {
            ans = min(ans, res1);
            r = mid2 - 1;
        }
    }

}
```

7.14 VectorHasher

```
struct VectorHasher {
    int operator()(const vector<int> &V) const {
        int hash = V.size();
        for(auto &i : V) {
            hash ^= i + 0x9e3779b9 + (hash << 6) + (hash >> 2);
        }
        return hash;
    }
};

unordered_map<vector <int> , int, VectorHasher> used;
```

7.15 XOR basis Range

```
const int LOG = 21;
struct Basis {
    int basis[LOG];
    int lst_idx[LOG];
    int sz;

    Basis() {
        sz = 0;
        for (int i = LOG - 1; i >= 0; --i) {
            basis[i] = 0;
            lst_idx[i] = -1;
        }
    }
    void insert(int x, int idx) {
        for (int i = LOG - 1; i >= 0; --i) {
            if ((x & (1ll << i)) == 0) continue;
            if(lst_idx[i] < idx)
            {
                swap(x, basis[i]);
                swap(lst_idx[i], idx);
                ++sz;
            }
            x ^= basis[i];
        }
    }
    int get_max(int l) {
        int ans = 0;
        for (int i = LOG - 1; i >= 0; --i) {
            if(basis[i] && !(ans & (1ll<<i)) && lst_idx[i] >= l)
                ans ^= basis[i];
        }
        return ans;
    }
};

void solve()
{

    Basis B = Basis();
    int n; cin >> n;
    vector<Basis> arr(n);
    for (int i = 0; i < n; ++i) {
        int x; cin >> x;
        B.insert(x, i);
        arr[i] = B;
    }

    int q; cin >> q;
    while (q--)
    {
        int l, r; cin >> l >> r;
        l--, r--;
        cout << arr[r].get_max(l) << endl;
    }

}
```

7.16 XOR basis

```
const int d=31;
int basis[d];
int sz;
bool insertVector(int mask) {
    for (ll i = d-1;i>=0; i--){
        if ((mask & 1<< i) == 0) continue;
        if (!basis[i]) {
            basis[i] = mask;
            sz++;
            return true;
        }
        mask ^= basis[i];
    }
    return false;
}
```

7.17 Xor from 1 to n

```
//xor from 1 to x
ll getXor(ll x) {
    if (x % 4 == 0)return x;
    if (x % 4 == 1)return 1;
    if (x % 4 == 2)return x + 1;
    return 0;
}
```

8 Problem codes

8.1 Lca Persistant code problem

```
#include<bits/stdc++.h> //Silence | ft. Reda , AbdoSa3d , Nourhan

using namespace std;

#define all(v) v.begin(),v.end()
#define ll long long
#define endl "\n"
```

```
struct Node
{
  Node* ls{}, * rs{};
  int sum{};
} pool[int(1e6)];
int top;
Node* null = pool + 0;
vector<Node*> rot;
void reset()
{
  top = 1;
  null->sum = 0;
  null->ls = null->rs = null;
}
Node* newNode()
{
  auto p = pool + (top++);
  *p = *null;
  return p;
}

// x is postion to insert
Node* add(Node* p, int l, int r, int x)
{
  auto np = newNode();
  *np = *p;
  np->sum++;
  if (r - l == 1)
  {
    return np;
  }
  int m = (l + r) / 2;
  if (x < m)
  {
    np->ls = add(p->ls, l, m, x);
  }
  else
  {
    np->rs = add(p->rs, m, r, x);
  }
  return np;
}

int R = 1e5 + 1;
int const N = 1e5 + 5, M = 20;
int dp[N][M + 1];
int lvl[N], n;
vector<vector<pair<int, int>>> G;

void dfs(int u, int par)
{
  dp[u][0] = par;
  for (auto [i, w] : G[u])
  {
    if (i != par)
    {
      lvl[i] = lvl[u] + 1;
      rot[i] = add(rot[u], 1, R, w);
      dfs(i, u);
    }
  }
}

int lca(int u, int v)
{
  if (lvl[u] > lvl[v])swap(u, v);
  for (int i = M; i >= 0; i--)
    if (lvl[v] - (1 << i) >= lvl[u])
      v = dp[v][i];

  if (u == v)return v;
  for (int i = M; i >= 0; i--)
  {
    int cu = dp[u][i], cv = dp[v][i];
    if (min(cu, cv) != -1 && cu != cv)
      u = cu, v = cv;
  }
  return dp[u][0];
}

void solve()
{
  reset();
  cin >> n;
  rot = vector<Node*>(n + 1);
  G = vector<vector<pair<int, int>>>(n + 1);
  rot[1] = null;
  for (int i = 1; i < n; i++)
  {
    int u, v, w;
    cin >> u >> v >> w;
    G[u].push_back({ v, w });
    G[v].push_back({ u, w });
  }
  dfs(1, -1);
  for (int i = 1; i <= M; i++)
  {
    for (int j = 1; j <= n; j++)
    {
      int u = dp[j][i - 1];
      if (u == -1)dp[j][i] = -1;
      else dp[j][i] = dp[u][i - 1];
    }
  }
  auto get = [&](int a, int b, int c, int k)
  {
    int l = 1, r = R;
    Node* at = rot[a];
    Node* bt = rot[b];
    Node* pt = rot[c];
    while (r - l > 1)
    {
      int sum = at->ls->sum + bt->ls->sum - 2 * pt->ls->sum;
      if (sum >= k)
      {
        at = at->ls;
        bt = bt->ls;
        pt = pt->ls;
        r = (r + l) / 2;
      }
      else
      {
        k -= sum;
        at = at->rs;
        bt = bt->rs;
        pt = pt->rs;
        l = (r + l) / 2;
      }
    }
    return l;
  };

  int q;
  cin >> q;
  cout << fixed << setprecision(1);
  while (q--)
  {
    int a, b;
    cin >> a >> b;
    int c = lca(a, b);
    int cnt = lvl[a] + lvl[b] - 2 * lvl[c];
    if (cnt & 1)
    {
      int x = get(a, b, c, cnt / 2 + 1);
      cout << x * 1.0 << endl;
    }
    else
    {
      int x = get(a, b, c, cnt / 2);
      int y = get(a, b, c, cnt / 2 + 1);
      cout << (x + y) / 2.0 << endl;
    }

  }
}

int
main()
{
  cin.tie(0)->sync_with_stdio(0);
  int t = 1;
  cin >> t;
  while (t--)
    solve();

}
```

8.2 Line Sweep Mostafa Saad

```
/*
   *
   *  Created on: Oct 29, 2016
   *      Author: mostafa saad
   *
   *
   *  tested on timus_1469_no_smoking
   *      http://acm.timus.ru/problem.aspx?space=1&num=1469
   */

  #include <iostream>
  #include <cmath>
  #include <complex>
  #include <cassert>
  #include <bits/stdc++.h>
  using namespace std;

  const double PI = acos(-1.0);
  const double EPS = 1e-8;

  int dcmp(double a, double b) {
    return fabs(a - b) <= EPS ? 0 : a < b ? -1 : 1;
  }

  typedef complex<int> point;

  #define X real()
  #define Y imag()
  #define angle(a)                (atan2((a).imag(), (a).real()))
  #define vec(a,b)                ((b)-(a))
  #define same(p1,p2)             (dp(vec(p1,p2),vec(p1,p2)) < EPS)
  #define dp(a,b)                 ( (conj(a)*(b)).real() )  // a*b cos(T), if zero ->
↪   prep
  #define cp(a,b)                 ( (conj(a)*(b)).imag() )  // a*b sin(T), if zero ->
↪   parllel
  #define length(a)               (hypot((a).imag(), (a).real()))
  #define normalize(a)            (a)/length(a)
  #define rotate0(p,ang)          ((p)*exp(point(0,ang)))
  #define rotateA(p,ang,about)  (rotate0(vec(about,p),ang)+about)
  #define reflect0(v,m)  (conj((v)/(m))*(m))

  point reflect(point p, point p0, point p1) {
    point z = p - p0, w = p1 - p0;
    return conj(z / w) * w + p0;  // Refelect point p1 around p0p1
  }
```

```
#define all(v)      ((v).begin()), ((v).end())
#define sz(v)     ((int)((v).size()))
#define clr(v, d)   memset(v, d, sizeof(v))
#define rep(i, v)   for(int i=0;i<sz(v);++i)
#define lp(i, n)    for(int i=0;i<(int)(n);++i)
#define lpi(i, j, n)  for(int i=(j);i<(int)(n);++i)
#define lpd(i, j, n)  for(int i=(j);i>=(int)(n);--i)

int ccw(point a, point b, point c) {
  point v1(b - a), v2(c - a);
  double t = cp(v1, v2);

  if (t > +EPS)
    return +1;
  if (t < -EPS)
    return -1;
  if (v1.X * v2.X < -EPS || v1.Y * v2.Y < -EPS)
    return -1;
  if (norm(v1) < norm(v2) - EPS)
    return +1;
  return 0;
}

bool intersect(point p1, point p2, point p3, point p4) {
  // special case handling if a segment is just a point
  bool x = (p1 == p2), y = (p3 == p4);
  if (x && y)
    return p1 == p3;
  if (x)
    return ccw(p3, p4, p1) == 0;
  if (y)
    return ccw(p1, p2, p3) == 0;

  return ccw(p1, p2, p3) * ccw(p1, p2, p4) <= 0 && ccw(p3, p4, p1) * ccw(p3, p4, p2)
↪  <= 0;
}

/////////////////////////////////////////////////////////////

bool operator <(point &a, point &b) {
  if (dcmp(a.X, b.X) != 0)
    return dcmp(a.X, b.X) < 0;
  return dcmp(a.Y, b.Y) < 0;
}

struct segment {
  point p, q;
  int seg_idx;

  segment() {seg_idx = -1;}
  segment(point p_, point q_, int seg_idx_) {
    if (q_ < p_)
      swap(p_, q_);
    p = p_, q = q_, seg_idx = seg_idx_;
  }

  double CY(int x) const {
    if (dcmp(p.X, q.X) == 0)
      return p.Y; // horizontal

    double t = 1.0 * (x - p.X)/(q.X - p.X);
    return p.Y + (q.Y - p.Y)*t;
  }
  // operator< is very tricky and can cause 100 WAs.
  bool operator<(const segment& rhs) const {
    if(same(p, rhs.p) && same(q, rhs.q))
      return false;

    int maxX = max(p.X, rhs.p.X);
    int yc = dcmp(CY(maxX), rhs.CY(maxX));

    if (yc == 0) // critical condition
      return seg_idx < rhs.seg_idx;
    return yc < 0;
  }
};

////////////////////////////////////////////////////////////

int ENTRY = +1, EXIT = -1;          // entry types
const int MAX_SEGMENTS = 50000 + 9;
const int MAX_EVENTS = MAX_SEGMENTS * 2;

struct event {
  point p;
  int type, seg_idx;
  // smaller X first. If tie: ENTRY event first. Last on smaller Y
  bool operator <(const event & rhs) const {
    if (dcmp(p.X, rhs.p.X) != 0)
      return dcmp(p.X, rhs.p.X) < 0;
    if (type != rhs.type)
      return type > rhs.type;
    return dcmp(p.Y, rhs.p.Y) < 0;
  }
};

int n;
segment segments[MAX_SEGMENTS];
event events[MAX_EVENTS];
set<segment> sweepSet;
typedef set<segment>::iterator ITER;

////////////////////////////////////////////////////////////

bool intersectSeg(ITER seg1Iter, ITER seg2Iter) {
  if (seg1Iter == sweepSet.end() || seg2Iter == sweepSet.end())
    return false;
  return intersect(seg1Iter->p, seg1Iter->q, seg2Iter->p, seg2Iter->q);
}

ITER after(ITER cur) {
  return cur == sweepSet.end() ? sweepSet.end() : ++cur;
}

ITER before(ITER cur) {
  return cur == sweepSet.begin() ? sweepSet.end() : --cur;
}

void FoundIntersection(int i, int j) {
  printf("%d %d\n", i + 1, j + 1);
}

void bentleyOttmann_lineSweep() {   // O( (k+n) logn )
  // Prepare events
  lp(i, n)
  {
    events[2*i] = {segments[i].p, ENTRY, i};
    events[2*i+1] = {segments[i].q, EXIT, i};
  }
  sort(events, events+2*n);

  lp(i, 2*n) {
    if (events[i].type == ENTRY) {
      auto status = sweepSet.insert(segments[events[i].seg_idx]);
      ITER cur = status.first, below = before(cur), above = after(cur);

      if(!status.second) {
        FoundIntersection(cur->seg_idx, events[i].seg_idx); // Duplicate
      } else {
        if(intersectSeg(cur, above))
          FoundIntersection(cur->seg_idx, above->seg_idx);
        if(intersectSeg(cur, below))
          FoundIntersection(cur->seg_idx, below->seg_idx);
      }
    } else {
      ITER cur = sweepSet.find(segments[events[i].seg_idx]);

      if(cur == sweepSet.end())
        continue; // e.g. Duplicate

      ITER below = before(cur), above = after(cur);

      if(intersectSeg(above, below))
        FoundIntersection(above->seg_idx, below->seg_idx);
      sweepSet.erase(cur);
    }
  }
}

////////////////////////////////////////////////////////////
```

```
int main() {
#ifndef ONLINE_JUDGE
  freopen("test.txt", "rt", stdin);
#endif

  int x, y;

  cin >> n;
  lp(i, n)
  {
    cin >> x >> y;      point p1 = point(x, y);
    cin >> x >> y;      point p2 = point(x, y);

    segments[i] = segment(p1, p2, i);
  }
  bentleyOttmann_lineSweep();

  return 0;
}
```

9 Strings

9.1 Aho Corasick

```
struct aho_corasick{
    struct trie_node {

        vector<int> pIdxs; //probably take memory limit
        map<char, int> next;
        int fail;
        trie_node() : fail(0) {}
        bool have_next(char ch) {
            return next.find(ch) != next.end();
        }
        int &operator[](char ch) {
            return next[ch];
        }
    };
    vector<trie_node> t;
    vector<string> patterns;
    vector<int> end_of_pattern;
    vector<vector<int>> adj;
    int insert(const string &s, int patternIdx) {
        int root = 0;
        for (const char &ch: s) {
            if (!t[root].have_next(ch)) {
                t.push_back(trie_node());
                t[root][ch] = t.size() - 1;
            }
            root = t[root][ch];
        }
        t[root].pIdxs.push_back(patternIdx);
        return root;
    }
    int next_state(int cur, char ch) {
        while (cur > 0 && !t[cur].have_next(ch))
            cur = t[cur].fail;
        if (t[cur].have_next(ch))
            return t[cur][ch];
        return 0;
    }
    void buildAhoTree() {
        queue<int> q;
        for (auto &child: t[0].next)
            q.push(child.second);
        while (!q.empty()) {

            int cur = q.front();
            q.pop();
            for (auto &child: t[cur].next) {
                int k = next_state(t[cur].fail, child.first);
                t[child.second].fail = k;
                vector<int> &idxs = t[child.second].pIdxs;
                //dp[child.second] = max(dp[child.second],dp[k]);
                idxs.insert(idxs.end(), all(t[k].pIdxs));
                q.push(child.second);
            }
        }
    }
    void buildFailureTree() {
        adj = vector<vector<int>>(t.size());
        for (int i = 1; i < t.size(); i++)
            adj[t[i].fail].push_back(i);
    }
    aho_corasick(const vector<string> &_patterns) {
        t.push_back(trie_node());
        patterns = _patterns;
        end_of_pattern = vector<int>(patterns.size());
        for (int i = 0; i < patterns.size(); i++)
            end_of_pattern[i] = insert(patterns[i], i);
        buildAhoTree();
        //buildFailureTree();
    }
    vector<vector<int>> match(const string &str) {
        int k = 0;
        vector<vector<int>> rt(patterns.size());
        for (int i = 0; i < str.size(); i++) {
            k = next_state(k, str[i]);
            for (auto &it: t[k].pIdxs)
                rt[it].push_back(i);
        }
        return rt;
    }
};
```

9.2 Anas Suffix Array

```
struct SuffixArray {
    const static int alpha = 128, LOG = 20;
    vector<int> suf, order, newOrder, lcp, logs;
    vector<vector<int>> table;
    string s;
    int n;

    SuffixArray(const string& _s) : n(sz(_s) + 1), s(_s) {
        s += ' ';
        suf = order = newOrder = vector<int>(n);
        vector<int> bucket_idx(n), newOrder(n), new_suf(n);
        vector<int> prev(n), head(alpha, -1);

        auto getOrder = [&](const int& a) -> int {
            return a < n ? order[a] : 0;
        };

        for (int i = 0; i < n; i++) {
            prev[i] = head[s[i]];
            head[s[i]] = i;
        }
        for (int i = 0, buc = -1, idx = 0; i < alpha; i++) {
            if(head[i] == -1) continue;
            bucket_idx[++buc] = idx;
            for (int j = head[i]; ~j; j = prev[j]){
                suf[idx++] = j; order[j] = buc;
            }
        }

        for (int len = 1; order[suf[n - 1]] != n - 1; len <<= 1) {
            auto comp = [&](const int &a, const int &b) -> bool {
                if (order[a] != order[b]) return order[a] < order[b];
                return getOrder(a + len) < getOrder(b + len);
            };
            for (int i = 0; i < n; i++) {
                int j = suf[i] - len;
                if(j < 0) continue;
                new_suf[bucket_idx[order[j]]++] = j;
            }
            for(int i = 1; i < n; i++){
                suf[i] = new_suf[i];
                bool newGroup = comp(suf[i - 1], suf[i]);
                newOrder[suf[i]] = newOrder[suf[i - 1]] + newGroup;
                if(newGroup){
                    bucket_idx[newOrder[suf[i]]] = i;
                }
            }
            order = newOrder;
        }

        lcp = vector<int>(n);
        int k = 0;
        for (int i = 0; i < n - 1; i++) {
            int pos = order[i];
            int j = suf[pos - 1];
            while (s[i + k] == s[j + k]) k++;
            lcp[pos] = k;
            k = max(0, k - 1);
        }
        buildTable();
    }

    void buildTable() {
        table = vector<vector<int>>(n + 1, vector<int>(LOG));
        logs = vector<int>(n + 1);
        logs[1] = 0;
        for (int i = 2; i <= n; i++)
            logs[i] = logs[i >> 1] + 1;
        for (int i = 0; i < n; i++) {
            table[i][0] = lcp[i];
        }
        for (int j = 1; j <= logs[n]; j++) {
            for (int i = 0; i <= n - (1 << j); i++) {
                table[i][j] = min(table[i][j - 1], table[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    int LCP(int i, int j) {
        if (i == j) return n - i - 1;
        int l = order[i], r = order[j];
        if (l > r) swap(l, r);
        l++;
        int sz = logs[r - l + 1];
        return min(table[l][sz], table[r - (1 << sz) + 1][sz]);
    }

    int LCP_Order(int l, int r){
        if (l == r) return n-suf[l]-1;
        if (l > r) swap(l, r);
        l++;
        int sz = logs[r - l + 1];
        return min(table[l][sz], table[r - (1 << sz) + 1][sz]);

    }

    int compare_substrings(int l1, int r1, int l2, int r2) {
        int k = min({LCP(l1, l2), r1 - l1 + 1, r2 - l2 + 1});
        l1 += k; l2 += k;
        if (l1 > r1 && l2 > r2) return 0;
        if (l1 > r1) return -1;
        if (l2 > r2) return 1;
        return (s[l1] > s[l2] ? 1 : -1);
    }
};
```

9.3 Hashing Kareem

```
class Hashing {
    const ll MOD = (1ll << 61) - 1;
    vector<ll> p, h;
    static ll base;
public:
    Hashing(const string &a) {
        p = h = vector<ll>(a.size() + 1);
        p[0] = 1;
        for (int i = 0; i < a.size(); i++) {
            p[i+1] = (__int128_t) p[i] * base % MOD;
            h[i+1] = ((__int128_t) h[i] * base + a[i]) % MOD;
        }
    }
    ll getHash(int l, int r) { //base 0
        return ((h[r + 1] - (__int128_t) h[l] * p[r - l + 1] % MOD) + MOD)%MOD;
    }
};
ll rng(ll l = (1ll << 40), ll r = (1ll << 60)) {
    static std::mt19937 gen(
            std::chrono::steady_clock::now().time_since_epoch().count());
    return std::uniform_int_distribution<long long>(l, r)(gen);
}
ll Hashing::base = rng();
```

9.4 Hashing

```
const int N = 1e5 + 5, MOD1 = 1e9 + 7, MOD2 = 1e9 + 9;
  int pw1[N], inv1[N], pw2[N], inv2[N], BASE;

bool isPrime(int x) {
    for (int i = 2; i * i <= x; i++) {
        if (x % i == 0) return 0;
    }
    return x > 1;
}

int fix(ll x, int M) {
    return (x % M + M) % M;
}

int fpow(int a, int b, int mod) {
    if (!b) return 1;
    int ret = fpow(a, b >> 1, mod);
    ret = fix(1ll * ret * ret, mod);
    if (b & 1) ret = fix(1ll * ret * a, mod);
    return ret;
}

void init() {
    static bool done = false;
    if (done) return;
    done = true;
    mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
    uniform_int_distribution<int> dist(257, 10007);
    do{
        BASE = dist(rng);
    }while (!isPrime(BASE));

    pw1[0] = inv1[0] = pw2[0] = inv2[0] = 1;
    int iv1 = fpow(BASE, MOD1 - 2, MOD1);
    int iv2 = fpow(BASE, MOD2 - 2, MOD2);
    for (int i = 1; i < N; ++i) {
        pw1[i] = fix(1ll * pw1[i - 1] * BASE, MOD1);
        pw2[i] = fix(1ll * pw2[i - 1] * BASE, MOD2);
        inv1[i] = fix(1ll * inv1[i - 1] * iv1, MOD1);
        inv2[i] = fix(1ll * inv2[i - 1] * iv2, MOD2);
    }
}

struct Hash {
    vector<pair<int, int>> pre;

    Hash(const string &s) {
        init();
        pre.assign(sz(s) + 1, {0, 0});
        for (int i = 0; i < sz(s); i++) {
            pre[i + 1] = make_pair(fix(1ll * pw1[i] * s[i] + pre[i].first, MOD1),
                                   fix(1ll * pw2[i] * s[i] + pre[i].second, MOD2));
        }
    }

    pair<int, int> getRange(int l, int r) const { // 0-based
        return make_pair(fix(1ll * inv1[l] * (pre[r + 1].first - pre[l].first), MOD1),
                         fix(1ll * inv2[l] * (pre[r + 1].second - pre[l].second), MOD2));
    }
};
```

9.5 KMP

```
//KMP
vector<int>KMP(const string&s){
    int n = sz(s);
    vector<int>fail(n);
    for(int i = 1;i<n;i++){
        int j = fail[i-1];
        while(j && s[i]!=s[j]){
            j = fail[j-1];
        }
        if(s[i] == s[j])j++;
        fail[i] = j;
    }
    return fail;
}
```

9.6 Manacher

```
vector<int> manacher_odd(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 1, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}
vector<int> manacher(string s) {
    string t;
    for(auto c: s) {
        t += string("#") + c;
    }
    auto res = manacher_odd(t + "#");
    return vector<int>(begin(res) + 1, end(res) - 1);
}

// Returns vector `d2` where d2[i] is the max radius of even-length
// palindrome centered between s[i-1] and s[i]
vector<int> manacher_even(const string &s) {
    int n = s.size();
    vector<int> d2(n); // For even-length palindromes
    int l = 0, r = -1;

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

        while (i - k - 1 >= 0 && i + k < n && s[i - k - 1] == s[i + k])
            ++k;

        d2[i] = k;
        if (i + k - 1 > r) {
            l = i - k;
            r = i + k - 1;
        }
    }

    return d2;
}
```

9.7 Saad Trie

```
struct trie
{
    trie* nxt[26]{};
    bool endOfWord = false;
    void insert(const string& s)
    {
        trie* current = this;
        for (auto ch : s)
        {
            int i = ch - 'a';
            if (current->nxt[i] == nullptr) current->nxt[i] = new trie;
            current = current->nxt[i];
        }
        current->endOfWord = true;
    }
    bool search(const string& s)
    {
        trie* current = this;
        for (auto ch : s)
        {
            int i = ch - 'a';
            if (current->nxt[i]==nullptr)return false;
            current = current->nxt[i];
        }
        return current->endOfWord;
    }
};
```

```
struct trie
{
    trie* nxt[2]{};
    void insert(int val)
    {
        trie* current = this;
        for (int i=30;i>=0;i--)
        {
            bool bit = val >> i & 1;
            if (current->nxt[bit] == nullptr) current->nxt[bit] = new trie;
            current = current->nxt[bit];
        }
    }
    int search(int val)
    {
        int ans = 0;
        trie* current = this;
        for (int i = 30;i >= 0;i--)
        {
            bool bit = val >> i & 1;
            if (current->nxt[!bit] == nullptr)
                current = current->nxt[bit];
            else
                ans += (1 << i), current=current->nxt[!bit];
        }
        return ans;
    }
};
```

9.8 Suffix Automation

```
struct suffix_automaton {
    struct state {
        int len, link = 0, cnt = 0;
        bool terminal = false, is_clone = false;
        map<char, int> next;
        state(int len = 0) : len(len) {}
        bool have_next(char ch) {
            return next.find(ch) != next.end();
        }
        void clone(const state &other, int nlen) {
            len = nlen;
            next = other.next;
            link = other.link;
            is_clone = true;
        }
    };
    vector<state> st;
    int last = 0;
    suffix_automaton() {
        st.push_back(state());
        st[0].link = -1;
    }
    suffix_automaton(const string &s) : suffix_automaton() {
        for (char ch: s)
            extend(ch);
        for (int cur = last; cur > 0; cur = st[cur].link)
            st[cur].terminal = true;
    }
    void extend(char c) {

        int cur = st.size();
        st.push_back(state(st[last].len + 1));
        st[cur].cnt = 1;
        int p = last;
        last = cur;
        while (p != -1 && !st[p].have_next(c)) {
            st[p].next[c] = cur;
            p = st[p].link;
        }
        if (p == -1)
            return;
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
            return;
        }
        int clone = st.size();
        st.push_back(state());
        st[clone].clone(st[q], st[p].len + 1);
        while (p != -1 && st[p].next[c] == q) {
            st[p].next[c] = clone;
            p = st[p].link;
        }
        st[q].link = st[cur].link = clone;
    }
    void calc_number_of_occurrences() {
        vector<vector<int>> lvl(st[last].len + 1);
        for (int i = 1; i < st.size(); i++)
            lvl[st[i].len].push_back(i);
        for (int i = st[last].len; i >= 0; i--)
            for (auto cur: lvl[i])
                st[st[cur].link].cnt += st[cur].cnt;
    }
    vector<ll> dp;
    ll Count(int cur) //count number of paths
    {
        ll &rt = dp[cur];
        if (rt)
            return rt;
        rt = 1;
        for (auto ch: st[cur].next)
            rt += Count(ch.second);
        return rt;

    }
    string kth_substring(ll k) //1-based,different substring,0 = ""
    {
        assert(k <= Count(0));
        string rt;
        int cur = 0;
        while (k > 0) {
            for (auto ch: st[cur].next) {
                if (Count(ch.second) < k)
                    k -= Count(ch.second);
                else {
                    rt += ch.first;
                    cur = ch.second;
                    k--;
                    break;
                }
            }
        }
        return rt;
    }
    string longest_common_substring(const string &t) {
        int cur = 0, l = 0, mx = 0, idx = 0;
        for (int i = 0; i < t.size(); i++) {
            while (cur > 0 && !st[cur].have_next(t[i])) {
                cur = st[cur].link;
                l = st[cur].len;
            }
            if (st[cur].have_next(t[i])) {
                cur = st[cur].next[t[i]];
                l++;
            }
            if (l > mx) {
                mx = l;
                idx = i;
            }
        }
        return t.substr(idx - mx + 1, mx);
    }
};
```

9.9 Trie 1d vector

```
class Trie{
private:
    struct Node{
        map<char,int>mp;
        int leaf;
        bool have_next(char c){
            return mp.find(c)!=mp.end();
        }
        int& operator[](char c){
            return mp[c];
        }
        Node(){
            leaf = 0;
        }
    };
public:
    vector<Node>v;
    Trie(){
        v.push_back(Node());
    }
    void update(const string&s,int op){
        int cur = 0;
        for(auto&ch : s){
            if(!v[cur].have_next(ch)){
                v.push_back(Node());
                v[cur][ch] = v.size() - 1;
            }
            cur = v[cur][ch];
        }
        v[cur].leaf+=op;
    }
    int count(const string&s){
        int cur = 0;
        for(auto&it: s){
            if(!v[cur].have_next(it)){
                return 0;
            }
            cur = v[cur][it];
        }
        return v[cur].leaf;
    }
};
```

9.10 Trie pointers

```
class Trie{
private:

    struct Node{
        map<char,Node*>mp;
        int c;
        Node(){
            c = 0;
        }
    };
    Node*root;
    void destroy(Node*cur){
        for(auto&it :cur->mp){
            destroy(it.second);
        }
        delete cur;
    }
public:
    Trie(){
        root = new Node;
    }
    ~Trie(){
        destroy(root);
    }
    void update(const string&s,int op){
        Node*tmp = root;
        for(auto&it : s){
            if(tmp->mp.count(it) == 0){
                tmp->mp[it] = new Node;
            }
            tmp = tmp->mp[it];
            tmp->c+=op;
        }
    }
};
```

9.11 Z algorithm

```
/* z[i] equal the length of the longest substring starting from s[i]
 which is also a prefix of s */
 vector<int> z_algo(string s) {
    int n = s.size();
    vector<int> z(n);
    z[0] = n;
    for (int i = 1, L = 1, R = 1; i < n; i++) {
        int k = i - L;
        if (z[k] + i >= R) {
            L = i;
            R = max(R, i);
            while (R < n && s[R - L] == s[R]) R++;
            z[i] = R - L;
        } else z[i] = z[k];
    }
    return z;
}


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