KUET_Xtinction Team Notebook

Md Umar Faruk, Raihan Arefin,
Md. Arifur Rahman

Contents

1 combinatorics

1.1 ncr dp [11 lines] - 4096d04440

```
int const N = 10000, mod = 12345;
int C[N][N];
void pre() {
    C[0][0] = 1;
    for(int n = 1; n < N; n++){
        C[n][0] = 1;
        for (int k = 1; k <= n; k++){
            C[n][k] = (C[n - 1][k - 1] +
                C[n - 1][k]) % mod;
        }
    }
}
```

1.2 ncr modinv [23 lines] - eb48f866fa

```
int f[N], invf[N];
void pre() {
    f[0] = 1;
    for (int i = 1; i < N; i++) {
        f[i] = 1LL * i * f[i - 1] % mod;
    }
    invf[N - 1] = power(f[N - 1], mod -
        2);
    for (int i = N - 2; i >= 0; i--) {
        invf[i] = 1LL * invf[i + 1] * (i
            + 1) % mod;
    }
}
int nCr(int n, int r) {
    if (n < r or n < 0) return 0;
    return 1LL * f[n] * invf[r] % mod *
        invf[n - r] % mod;
}
ll nCr(ll n, int r) {
    if(n < r || n < 0) return 0;
    ll ans = 1;
    for(int c = 1; c <= r; c++) {
        ans = (1LL * ans * n--) % mod;
    }
    return (1LL * ans * invf[r]) % mod;

}
```

1.3 ncr upto 1e18 [10 lines] - 9ec10be91c

```
long long C(int n, int r) {
    if(r > n - r) r = n - r;
    long long ans = 1;
    int i;
    for(i = 1; i <= r; i++) {
        ans *= n - r + i;
        ans /= i;
    }
    return ans;
}
```

2 dynamic programming

2.1 LIS with full path [35 lines] - e29917a0bc

```
// Function to compute and return LIS as
    a vector
vector<int> LIS(const vector<int>& num) {
    int n = num.size();

    vector<int> array(n + 1, INF);     //
        array[k] = min ending value of
        LIS of length k
    vector<int> mem(n + 1);            //
        mem[k] = index in `num` where LIS
        of length k ends
    vector<int> prev_idx(n, -1);       //
        prev_idx[i] = previous element's
        index in LIS
    vector<int> res;                   //
        result vector to store LIS

    array[0] = -INF;                   //
        Sentinel

    int maxlen = 1;

    for (int i = 0; i < n; i++) {
        // Binary search for the first
            position k where array[k] >=
            num[i]
        int k =
            lower_bound(array.begin(),
            array.begin() + maxlen + 1,
            num[i]) - array.begin();

        array[k] = num[i];     // Update
            LIS ending at length k
        mem[k] = i;            // Store
            current index

        if (k > 1)
            prev_idx[i] = mem[k - 1]; //
                Link to previous index in
                LIS

        if (k > maxlen)
```

```
            maxlen = k;         // Update
                LIS length
    }

    // Reconstruct LIS
    for (int i = mem[maxlen]; i != -1; i
        = prev_idx[i])
        res.push_back(num[i]);

    reverse(res.begin(), res.end()); //
        To get correct order

    return res;
}
```

2.2 all node max dis [39 lines] - cf57f4a333

```
void solve() {
    int n;
    cin >> n;
    vector<vector<int>> g(n + 1);
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }

    vector<int> down(n + 1);
    vector<int> best1(n + 1, -1), best2(n
        + 1, -1);
    function<int(int, int)> dfs1 =
        [&](int u, int p) -> int {
        for(auto v : g[u]) if(v != p) {
            int cand = 1 + dfs1(v, u);

            if (cand > best1[u]) {
                best2[u] = best1[u];
                best1[u] = cand; }
            else if (cand > best2[u])
                best2[u] = cand;
            down[u] = max(down[u],
                cand);
        }
        return down[u];
    };
    dfs1(1, 0);

    vector<int> up(n + 1);
    function<void(int, int)> dfs2 =
        [&](int u, int p) -> void {
        for(auto v : g[u]) if(v != p) {
            int val = (down[v] + 1 ==
                best1[u]) ? best2[u] :
                best1[u];
            up[v] = 1 + max(up[u], val);
            dfs2(v, u);
        }

    };
    dfs2(1, 0);

    for(int i = 1; i <= n; i++) {
        cout << max(up[i], down[i]) << "
            \n"[i == n];
    }
}
```

2.3 digit dp [14 lines] - d3853d7797

```
ll dp[11][2][2];
string l,r;//l<r must
ll rec(ll ind, bool tightL, bool tightR){
    if(ind==l.size())return 0;
    if(dp[ind][tightL][tightR]!=-1)return
        dp[ind][tightL][tightR];
    char lo=tightL?l[ind]:'0',hi=
        tightR?r[ind]:'9';
    ll ans=LLONG_MAX;
    for(char i=lo;i<=hi;i++){
        //Cost change kora lagbe
            problemwise
        ll cost=(ll)(l[ind]==i)+(r[ind]=
            =i);
        ans=min(ans,cost+rec(ind+1,tightL
            & l[ind]==i, tightR &
            r[ind]==i));
    }
    return dp[ind][tightL][tightR]=ans;
}
```

2.4 longest path [27 lines] - b1bd570b6f

```
queue<int> Q;
for(int i = 1; i <= n; i++) {
    if(indeg[i] == 0) Q.push(i);
}
vector<int> topo;
while(!Q.empty()) {
    int u = Q.front(); Q.pop();
    topo.push_back(u);
    for(auto v : g[u]) {
        if(--indeg[v] == 0) {
            Q.push(v);
        }
    }
}
vector<int> dp(n + 1, -1e9);
vector<int> par(n + 1, 0);
dp[1] = 1;
for(auto u : topo) {
    if(dp[u] == -1e9) continue;
    for(auto v : g[u]) {
        if(dp[u] + 1 > dp[v]) {
            dp[v] = dp[u] + 1;
            par[v] = u;
        }
    }
}
```

2.5 longest zig zag path dp [92 lines] - 2ddaf1640b

```
// find Longest zig zag path usnig BIT
    and DP
// red means peak ^
// white means vally v
// \/\/\/\/

struct BIT{                          //
    stores (length , last-pos)
    using Node = pair<int,int>;
    int n; vector<Node> t;          //
        (0,-1) = "nothing"
    BIT() {}
    BIT(int _n) : n(_n), t(n + 2, {0,-1})
        {}
    Node query(int i) {             //
        prefix-max
        Node res{0,-1};
        for (; i >= 1; i -= i & -i)
            if (t[i].first > res.first)
                res = t[i];
        return res;
    }
    void upd(int i, Node v) {
        for (; i <= n; i += i & -i)
            if (v.first > t[i].first)
                t[i] = v;
    }
};

void solve() {
    int n;
    cin >> n;
    vector<int> v(n);
    for(auto &x : v) cin >> x;

    vector<int> tmp = v;
    sort(tmp.begin(), tmp.end());
    tmp.erase(unique(tmp.begin(),
        tmp.end()), tmp.end());
    int m = tmp.size();
    vector<int> id(n), rid(n);
    for(int i = 0; i < n; i++) {
        id[i] = lower_bound(tmp.begin(),
            tmp.end(), v[i]) -
            tmp.begin() + 1;
        rid[i] = m - id[i] + 1;
    }

    BIT red(n), white(n);

    vector<vector<int>> dp(2,
        vector<int>(n + 1, 0)); // 0 ->
        red 1 -> white
    vector<vector<int>> pre(2,
        vector<int>(n + 1, -1)); // 0 ->
        red 1 -> white

    for(int i = 0; i < n; i++) {
        // current is red
        auto [lenw, posw] =
            white.query(id[i] - 1);
        dp[0][i] = lenw + 1;
        pre[0][i] = posw;
        red.upd(rid[i], {dp[0][i], i});
        // current is white
        auto [lenr, posr] =
            red.query(rid[i] - 1);
        dp[1][i] = lenr + 1;
        pre[1][i] = posr;
        white.upd(id[i], {dp[1][i], i});
    }

    int mx = 0, pos = 0, f = 0;
    for(int i = 0; i < n; i++) {
        if(mx < dp[0][i]) {
            mx = dp[0][i];
            pos = i;
            f = 0;
        }
        if(mx < dp[1][i]) {
            mx = dp[1][i];
            pos = i;
            f = 1;
        }
    }

    vector<int> ans;
    while(pos >= 0) {
        ans.push_back(v[pos]);
        pos = pre[f][pos];
        f ^= 1;
    }
    reverse(ans.begin(), ans.end());
    cout << ans.size() << "\n";
    for(auto x : ans) {
        cout << x << " ";
    }
    cout << "\n";
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    int tc = 1;
    //cin >> tc;
    while (tc--) solve();
    return 0;
}
```

2.6 sibling dp [54 lines] - 473736aa00

```
/*
  Problem:
  Divide a rooted tree into the minimum
      number of groups such that
  the total cost (edge weights) within
      each group does not exceed `k`.

  Definitions:
  - dp[u][remk]: Minimum number of groups
      required to cover subtree rooted at
      node `u`,
    where `remk` is the remaining cost
        capacity of the current group.
  - adj[par][idx]: The idx-th child of
      node `par`, along with the edge
      cost.
*/

ll n, k;
ll dp[mx][mx];
vector<pair<ll, ll>> adj[mx]; // adj[u] =
    {v, cost} edges in rooted tree

// Recursive function to compute
    dp[par][remk]
// par = current parent node
// idx = index of child being processed
// remk = remaining budget in the current
    group
ll sibling_dp(ll par, ll idx, ll remk) {
  if (remk < 0) return inf; // invalid
      case, over budget
```

```
    // No more children to process
    if (adj[par].size() <= idx) return 0;

    ll u = adj[par][idx].first; // current
        child node

    // Memoization check
    if (dp[u][remk] != -1)
      return dp[u][remk];

    ll ret = inf;
    ll under = 0, sibling = 0;

    // Option 1: Create a new group for
        this child (if not the root)
    if (par != 0) {
      under = 1 + dfs(u, 0, k);
          // Entire subtree of `u` in a new
          group
      sibling = dfs(par, idx + 1, remk);
          // Process next siblings with
          same group
      ret = min(ret, under + sibling);
          // Total groups = groups under +
          groups from siblings
    }

    // Option 2: Try including this child
        in current group if cost allows
    ll temp = remk - adj[par][idx].second;
        // Remaining cost after adding this
        edge

    // Try all possible cost splits between
        `under` (child) and `sibling`
        (rest)
    for (ll chk = temp; chk >= 0; chk--) {
      ll siblingk = temp - chk;
      under = dfs(u, 0, chk);
          // Child's subtree with budget
          `chk`
      sibling = dfs(par, idx + 1,
          siblingk); // Remaining siblings
          with remaining budget
      ret = min(ret, under + sibling);
    }

    return dp[u][remk] = ret; // Memoize
        and return result
}
```

2.7 sos dp [19 lines] - e6cfca2350

```
// SUBSET SOS
// dp[mask] = sum of dp[submask] where
    submask ⊆ mask
void sos_subset(int M, vector<ll>& dp) {
    int N = 1 << M;
    for(int i = 0; i < M; i++)
        for(int mask = 0; mask < N;
            mask++)
            if(mask & (1 << i))
                dp[mask] += dp[mask ^ (1
                    << i)];
}
```

```
// SUPERSET SOS
// dp[mask] = sum of dp[supermask] where
    supermask ⊇ mask
void sos_superset(int M, vector<ll>& dp)
    {
    int N = 1 << M;
    for(int i = 0; i < M; i++)
        for(int mask = N-1; mask >= 0;
            mask--)
            if(mask & (1 << i))
                dp[mask ^ (1 << i)] +=
                    dp[mask];
}
```

2.8 sum of dis of u [38 lines] - 6facf7b133

```
void solve() {
    int n;
    cin >> n;
    vector<vector<int>> g(n + 1);
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    vector<int> sz(n + 1);
    vector<ll> dp(n + 1);

    function<void(int, int)> dfs1 =
        [&](int u, int p) -> void {
        sz[u] = 1;
        for(auto v : g[u]) {
            if(v != p) {
                dfs1(v, u);
                sz[u] += sz[v];
                dp[1] += sz[v]; // root
                    is 1
            }
        }
    };
    dfs1(1, 0);

    function<void(int, int)> dfs2 =
        [&](int u, int p) -> void {
        for(auto v : g[u]) if(v != p) {
            dp[v] = dp[u] - sz[v] + (n -
                sz[v]);
            dfs2(v, u);
        }
    };
    dfs2(1, 0);

    for(int i = 1; i <= n; i++) {
        cout << dp[i] << " \n"[i == n];
    }

}
```

3 game theory

3.1 note [14 lines] - 9730900ce0

```
>[First Write a Brute Force solution]
>Nim = all xor
>Misere Nim = Nim + corner case: if all
    piles are 1, reverse(nim)
>Bogus Nim = Nim
>Staircase Nim = Odd indexed pile Nim
    (Even indexed pile doesnt matter, as
    one player can give bogus moves to
    drop all even piles to ground)
>Sprague Grundy: [Every impartial game
    under the normal play convention is
    equivalent to a one-heap game of nim]

Every tree = one nim pile = tree root
    value; tree leaf value = 0; tree node
    value = mex of all child nodes.
[Careful: one tree node can become
    multiple new tree roots(multiple
    elements in one node), then the value
    of that node = xor of all those root
    values]
>Hackenbush(Given a rooted tree; cut an
    edge in one move; subtree under that
    edge gets removed; last player to cut
    wins):
 Colon:  //G(u) =
    (G(v1)+1) ⊕ (G(v2)+1) ⊕ ··· [v1, v2, ···
    are childs of u]
For multiple trees ans is their xor
>Hackenbush on graph (instead of tree
    given an rooted graph):
fusion: All edges in a cycle can be fused
    to get a tree structure; build a
    super node, connect some single nodes
    with that super node, number of
    single nodes is the number of edges
    in the cycle.
Sol: [Bridge component tree] mark all
    bridges, a group of edges that are
    not bridges, becomes one component
    and contributes number of edges to
    the hackenbush. (even number of edges
    contributes 0, odd number of edges
    contributes 1)
```

4 strings

4.1 hash [50 lines] - 79b8bf4eea

```
struct SimpleHash {
    int len;
    long long base, mod;
    vector<int> P, H, R;
    SimpleHash() {}
    SimpleHash(string str, long long b,
        long long m) {
        base = b, mod = m, len =
            str.size();
        P.resize(len + 4, 1),
            H.resize(len + 3, 0),
            R.resize(len + 3, 0);
        for (int i = 1; i <= len + 3;
            i++)
            P[i] = (P[i - 1] * base) %
                mod;
        for (int i = 1; i <= len; i++)
            H[i] = (H[i - 1] * base +
                str[i - 1] + 1007) %
                mod;
        for (int i = len; i >= 1; i--)
            R[i] = (R[i + 1] * base +
                str[i - 1] + 1007) %
                mod;
    }
    inline int range_hash(int l, int r) {
        int hashval = H[r + 1] - ((long
            long)P[r - l + 1] * H[l] %
            mod);
        return (hashval < 0 ? hashval +
            mod : hashval);
    }
    inline int reverse_hash(int l, int r)
        {
        int hashval = R[l + 1] - ((long
            long)P[r - l + 1] * R[r + 2]
            % mod);
        return (hashval < 0 ? hashval +
            mod : hashval);
    }
};
struct DoubleHash {
    SimpleHash sh1, sh2;
    DoubleHash() {}
    DoubleHash(string str) {
        sh1 = SimpleHash(str,
            1949313259, 2091573227);
        sh2 = SimpleHash(str,
            1997293877, 2117566807);
    }
    long long concate(DoubleHash& B , int
        l1 , int r1 , int l2 , int r2) {
        int len1 = r1 - l1+1 , len2 = r2
            - l2+1;
        long long x1 =
            sh1.range_hash(l1, r1) ,
        x2 = B.sh1.range_hash(l2, r2);
        x1 = (x1 * B.sh1.P[len2]) %
            2091573227;
        long long newx1 = (x1 + x2) %
            2091573227;
        x1 = sh2.range_hash(l1, r1);
        x2 = B.sh2.range_hash(l2, r2);
        x1 = (x1 * B.sh2.P[len2]) %
            2117566807;
        long long newx2 = (x1 + x2) %
            2117566807;
        return (newx1 << 32) ^ newx2;
    }
    inline long long range_hash(int l,
        int r) {
        return ((long
            long)sh1.range_hash(l, r) <<
            32) ^ sh2.range_hash(l, r);
    }
    inline long long reverse_hash(int l,
        int r) {
        return ((long
            long)sh1.reverse_hash(l, r)
            << 32) ^ sh2.reverse_hash(l,
            r);
    }
};
```

4.2 kmp [13 lines] - 5a11b8b81b

```
string t, p;
string s = p + "#" + t;
int n = s.size();
// pi[0...i] returns longest suffix which
    is prefix
vector<int> pi(n, 0);
for(int i = 1, j = 0; i < n; i++) {
    while(j >= 0 && s[i] != s[j]) {
        if(j >= 1) j = pi[j - 1];
        else j = -1;
    }
    j++;
    pi[i] = j;
}
```

4.3 manachar [25 lines] - 934f373c7c

```
struct Manacher {
  vector<int> p[2];
  // p[1][i] = (max odd length palindrome
      centered at i) / 2 [floor division]
  // p[0][i] = same for even, it
      considers the right center
  // e.g. for s = "abbabba", p[1][3] = 3,
      p[0][2] = 2
  Manacher(string s) {
    int n = s.size();
    p[0].resize(n + 1);
    p[1].resize(n);
    for (int z = 0; z < 2; z++) {
      for (int i = 0, l = 0, r = 0; i <
          n; i++) {
        int t = r - i + !z;
        if (i < r) p[z][i] = min(t,
            p[z][l + t]);
        int L = i - p[z][i], R = i +
            p[z][i] - !z;
        while (L >= 1 && R + 1 < n && s[L
            - 1] == s[R + 1])
          p[z][i]++, L--, R++;
        if (R > r) l = L, r = R;
      }
    }
  }
  bool is_palindrome(int l, int r) {
    int mid = (l + r + 1) / 2, len = r -
        l + 1;
    return 2 * p[len % 2][mid] + len % 2
        >= len;
  }
};
```

4.4 palindromic tree [67 lines] - 8e5a153d34

```
struct palindromic_tree {
    struct node {
        int len, link;
        int occ, occ1;      // occ: total
            in s+s, occ1: only in first
            n0
        array<int, 10> next;
        node(int l = 0) : len(l),
            link(0), occ(0), occ1(0) {
            next.fill(-1);
        }
    };
    string s;
    vector<node> t;
    int last;
    int n0;                 // original
        length (first-half cutoff)

    palindromic_tree() {}
    palindromic_tree(string _s, int _n0)
        {
        s = _s; n0 = _n0;
        t.clear();
        t.reserve((int)s.size() + 3);
        t.push_back(node(-1));
        t.push_back(node(0));
        t[0].link = 0;
        t[1].link = 0;
        last = 1;
    }
    int getlink(int u, int i) {
        while (true) {
            int idx = i - 1 - t[u].len;
            if (idx >= 0 && s[idx] ==
                s[i]) return u;
            u = t[u].link;
        }
    }
    int add(int i) {
        int u = getlink(last, i);
        int ch = s[i] - '0';
        if (t[u].next[ch] != -1) {
            last = t[u].next[ch];
            t[last].occ++;
            if (i < n0) t[last].occ1++;
                // count only in first n0
                for non-wrap
            return 0;
        }
        int llen = t[u].len + 2;
        node nu(llen);
        t.push_back(nu);
        int v = (int)t.size() - 1;
        t[u].next[ch] = v;

        if (t[v].len == 1) {
            t[v].link = 1;
        } else {
            int w = t[u].link;
            w = getlink(w, i);
            t[v].link = t[w].next[ch];
        }
        last = v;
        t[last].occ = 1;
        if (i < n0) t[last].occ1 = 1;
        return 1;
    }
    void pushOcc() {
        for (int v = (int)t.size() - 1; v
            >= 2; --v) {
            int p = t[v].link;
            t[p].occ  += t[v].occ;
            t[p].occ1 += t[v].occ1;
        }
    }
};
```

4.5 suffix array [231 lines] - 1d9adcaccf

```
struct SuffixArray {
    string s;
    int n;
    vector<int> sa;
    vector<int> lcp, lg;
    vector<vector<int>> st;

    // Constructor: builds SA, LCP, and
        Sparse Table
    SuffixArray(const string& str) :
        s(str), n(str.size()) {
        buildSA();
        buildLCP();
        buildSparse();
    }

    // Builds the suffix array using the
        doubling method in O(n log n)
    void buildSA() {
        sa.resize(n);
        vector<int> rank(n), tmp(n);
        iota(sa.begin(), sa.end(), 0);
        for (int i = 0; i < n; i++)
            rank[i] = s[i];

        for (int k = 1; k < n; k *= 2) {
            auto cmp = [&](int i, int j)
                {
                if (rank[i] != rank[j])
                    return rank[i] <
                    rank[j];
                int ri = (i + k < n) ?
                    rank[i + k] : -1;
                int rj = (j + k < n) ?
                    rank[j + k] : -1;
                return ri < rj;
            };
            sort(sa.begin(), sa.end(),
                cmp);
            tmp[sa[0]] = 0;
            for (int i = 1; i < n; i++)
                tmp[sa[i]] = tmp[sa[i -
                    1]] + (cmp(sa[i -
                    1], sa[i]) ? 1 : 0);
            rank = tmp;
        }
    }

    // Builds the LCP (Longest Common
        Prefix) array in O(n)
    void buildLCP() {
        lcp.resize(n);
        vector<int> inv(n);
        for (int i = 0; i < n; i++)
            inv[sa[i]] = i;

        int k = 0;
        lcp[n - 1] = 0;
        for (int i = 0; i < n; i++) {
            if (inv[i] == n - 1) {
                k = 0;
                continue;
            }
            int j = sa[inv[i] + 1];
            while (i + k < n && j + k < n
                && s[i + k] == s[j + k])
                k++;
            lcp[inv[i]] = k;
            if (k) k--;
        }
    }

    // Builds sparse table over sa for
        rangeMin queries
    void buildSparse() {
        lg.resize(n + 1); lg[0] = -1;
        for (int i = 1; i <= n; ++i)
            lg[i] = lg[i >> 1] + 1;

        int K = lg[n] + 1;
        st.assign(K, vector<int>(n));
        st[0] = sa;
        for (int k = 1; k < K; ++k)
            for (int i = 0; i + (1 << k)
                <= n; ++i)
                st[k][i] = min(st[k -
                    1][i], st[k - 1][i +
                    (1 << (k - 1))]);
    }

    // Range minimum over sa[l..r] using
        sparse table
    int rangeMin(int l, int r) {
        int k = lg[r - l + 1];
        return min(st[k][l], st[k][r - (1
            << k) + 1]);
    }

    // Binary search for pattern range in
        suffix array
    pair<int, int> rangeSearch(const
        string& pat, int &L, int &R, int
        pos) {
        //cout « L « " " « R « "\n";
        int lo = L, hi = R, mid, ans =
            -1;

        while(lo <= hi) {
            mid = (lo + hi) >> 1;
            if(sa[mid] + pos >= n ||
                s[sa[mid] + pos] <
                pat[pos]) {
                lo = mid + 1;
            }
            else {
                ans = mid;
                hi = mid - 1;
            }
        }

        if(ans == -1) return {-1, -1};

        int left = ans;
        lo = ans, hi = R, ans = -1;
        while(lo <= hi) {
            mid = (lo + hi) >> 1;
            if(sa[mid] + pos >= n ||
                s[sa[mid] + pos] <=
                pat[pos]) {
                ans = mid;
                lo = mid + 1;
```

```
        }
        else {
            hi = mid - 1;
        }
    }
    if(ans == -1) return {-1, -1};
    int right = ans;
    return {left, right};
}

ll query(const string &p, int &L, int
    &R, int pos) {
    auto [l, r] = rangeSearch(p, L,
        R, pos);
    if(l == -1) return -1;
    L = l, R = r;
    //cout << p << " " << l << " " << r <<
        "\n";
    return (r - l + 1);
}

ll occurence_as_substring(const
    string &t) {
    string p;
    int ans = 0;
    int l = 0, r = n - 1;
    for(int i = 0; i < t.size(); i++)
        {
        p.push_back(t[i]);
        ans = query(p, l, r, i);
        if(ans == -1) break;
    }
    if(ans == -1) ans = 0;
    return ans;
}

// Returns the number of distinct
    substrings of each length
vector<int> countDistinctByLength() {
    vector<int> diff(n + 2, 0);
    for (int i = 0; i < n; i++) {
        int len = n - sa[i];
        int lcp_prev = (i == 0 ? 0 :
            lcp[i - 1]);
        int low = lcp_prev + 1;
        int high = len;
        if (low <= high) {
            diff[low]++;
            diff[high + 1]--;
        }
    }
    vector<int> result(n + 1);
    for (int i = 1; i <= n; i++)
        result[i] = result[i - 1] +
            diff[i];
    return result;
}

// Returns the k-th lexicographically
    smallest distinct substring
string getKthDistinctSubstring(long
    long k) {
    for (int i = 0; i < n; i++) {
        int start = sa[i];
        int lcp_prev = (i == 0 ? 0 :
            lcp[i - 1]);
        int total = n - start -
            lcp_prev;
        if (k <= total)
            return s.substr(start,
                lcp_prev + k);
        k -= total;
    }
    return "";
}

// Returns the longest substring that
    appears more than once
string longestRepeatingSubstring() {
    int max_len = 0, index = -1;
    for (int i = 1; i < n; i++) {
        if (lcp[i] > max_len) {
            max_len = lcp[i];
            index = sa[i];
        }
    }
    if (max_len == 0) return "-1";
    return s.substr(index, max_len);
}

// Returns the longest common
    substring between s and another
    string s2
pair<int, string>
    longestCommonSubstring(const
    string& s2) {
    string combined = s + '#' + s2;
    int len1 = s.size();
    SuffixArray combinedSA(combined);
    int max_len = 0, index = -1;
    for (int i = 1; i <
        combinedSA.n; i++) {
        int a = combinedSA.sa[i], b
            = combinedSA.sa[i - 1];
        if ((a < len1) != (b < len1))
            {
            if (combinedSA.lcp[i - 1]
                > max_len) {
                max_len =
                    combinedSA.lcp[i
                    - 1];
                index =
                    combinedSA.sa[i];

            }
        }
    }
    if (max_len == 0) return {0, ""};
    return {max_len,
        combined.substr(index,
        max_len)};
}

// Returns the longest palindromic
    substring in s
string longestPalindromicSubstring()
    {
    string rev = s;
    reverse(rev.begin(), rev.end());
    string joined = s + '#' + rev;
    SuffixArray pal(joined);
    int max_len = 0, index = -1;
    for (int i = 1; i < pal.n; i++) {
        int a = pal.sa[i], b =
            pal.sa[i - 1];
        bool in_s = (a < n), in_rev
            = (b > n);
        bool in_s2 = (b < n), in_rev2
            = (a > n);
        if ((in_s && in_rev) ||
            (in_s2 && in_rev2)) {
            int lcp_len = pal.lcp[i
                - 1];
            int pos = in_s ? a : b;
            if (lcp_len > max_len &&
                pos + lcp_len <= n) {
                max_len = lcp_len;
                index = pos;
            }
        }
    }
    if (max_len == 0) return "";
    return s.substr(index, max_len);
}

// Lexicographically compares
    s[l1..r1] and s[l2..r2]
int compareSubstrings(int l1, int
    r1, int l2, int r2) {
    int len1 = r1 - l1 + 1, len2 = r2
        - l2 + 1;
    int min_len = min(len1, len2);
    for (int i = 0; i < min_len; i++)
        {
        if (s[l1 + i] != s[l2 + i])
            return (s[l1 + i] < s[l2
                + i]) ? -1 : 1;
    }
    if (len1 == len2) return 0;
    return (len1 < len2) ? -1 : 1;
}
};
```

4.6 z [14 lines] - 3133135f30

```
// z[i] returns the longest substring of
    length k from i which is also a
    prefix
function<vector<int>()> Zf = [&]() ->
    vector<int> {
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n;
        ++i) {
        if (i <= r) {
            z[i] = min (r - i + 1, z[i -
                l]);
        }
        while (i + z[i] < n && s[z[i]]
            == s[i + z[i]]) ++z[i];
        if (i + z[i] - 1 > r) {
            l = i, r = i + z[i] - 1;
        }
    }
    return z;
};
```

5 flow

5.1 Blossom [86 lines] - 1c51297a78

```
// Finds Maximum Matching in a General
    Graph using Edmonds' Blossom
    Algorithm
// Returns: vector<int> mate where
    mate[i] = j means node i is matched
    with node j
// Complexity: O(N * M), where N = number
    of nodes, M = number of edges
vector<int> Blossom(vector<vector<int>>&
    graph) {
  int n = graph.size();             //
      Number of nodes
  int timer = -1;                   // Used
      for tracking during LCA search

  // mate[i] = j means i is matched with
      j (or -1 if unmatched)
  vector<int> mate(n, -1);
  vector<int> label(n);             //
      Labels: -1 = unused, 0 = even
      level, 1 = odd level
  vector<int> parent(n);            // BFS
      tree parent
  vector<int> orig(n);              //
      Original base of blossom for each
      node
  vector<int> aux(n, -1);           // Aux
      array used in LCA finding
  vector<int> q;                    // BFS
      queue

  // Finds the Lowest Common Ancestor
      (LCA) of x and y in the alternating
      tree
  auto lca = [&](int x, int y) {
    timer++;  // Increase visit timestamp
    for (; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;  //
          Found LCA
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 :
          orig[parent[mate[x]]]);  //
          Move up the alternating tree
    }
  };

  // Contracts a blossom (odd-length
      cycle) found during BFS
  auto blossom = [&](int v, int w, int a)
      {
    // v and w are two endpoints; a is
        the LCA
    while (orig[v] != a) {
      parent[v] = w;
      w = mate[v];
      if (label[w] == 1) {
        label[w] = 0;
        q.push_back(w);  // Re-add to BFS
            queue
      }
      orig[v] = orig[w] = a;  // Update
          base of blossom
```

```
    v = parent[w];
  }
};

// Augments along the alternating path
    ending at v to increase the
    matching
auto augment = [&](int v) {
  while (v != -1) {
    int pv = parent[v], nv = mate[pv];
    mate[v] = pv;       // Match v with
        its parent
    mate[pv] = v;
    v = nv;             // Continue on
        the unmatched end
  }
};

// Breadth-First Search to find an
    augmenting path starting from root
auto bfs = [&](int root) {
  fill(label.begin(), label.end(),
      -1);    // Reset labels
  iota(orig.begin(), orig.end(), 0);
      // Each node is its own blossom
      initially
  q.clear();
      // Reset BFS queue

  label[root] = 0;
      // Start root at even level
  q.push_back(root);

  for (int i = 0; i < (int)q.size();
      ++i) {
    int v = q[i];
    for (auto x : graph[v]) {
      if (label[x] == -1) {
        // If x is unvisited
        label[x] = 1;       // Mark as
            odd level
        parent[x] = v;
        if (mate[x] == -1)  // Found an
            augmenting path
          return augment(x), 1;
        label[mate[x]] = 0;  // Next
            level (even)
        q.push_back(mate[x]);
      }
      else if (label[x] == 0 && orig[v]
          != orig[x]) {
        // Found a blossom (x and v
            have the same label and
            different bases)
        int a = lca(orig[v], orig[x]);
        blossom(x, v, a);
        blossom(v, x, a);
      }
    }
  }
  return 0;  // No augmenting path
      found
};

// Try to find augmenting paths
    starting from unmatched nodes
  for (int i = 0; i < n; i++) {
    if (mate[i] == -1)
```

5.2 Hopcroft [69 lines] - 1867a462ee

```
struct HopcroftKarp {
    vector<vector<int>> g;
    vector<int> pairU, pairV, dis;
    int n, m;

    HopcroftKarp(int n, int m) : n(n),
        m(m) {
        g.resize(n + 1);
        dis.resize(n + 1);
        pairU.assign(n + 1, -1);
        pairV.assign(m + 1, -1);
    }

    void add_edge(int u, int v) {
        g[u].push_back(v);
    }

    bool bfs() {
        queue<int> Q;
        for(int u = 1; u <= n; u++) {
            if(pairU[u] == -1) {
                dis[u] = 0;
                Q.push(u);
            }
            else {
                dis[u] = -1;
            }
        }
        bool flag = false;
        while(not Q.empty()) {
            int u = Q.front(); Q.pop();
            for(auto v : g[u]) {
                int u2 = pairV[v];
                if(u2 == -1) {
                    flag = true;
                }
                else if(dis[u2] == -1) {
                    dis[u2] = dis[u] + 1;
                    Q.push(u2);
                }
            }
        }
        return flag;
    }

    bool dfs(int u) {
        for(auto v : g[u]) {
            int u2 = pairV[v];
            if(u2 == -1 || (dis[u2] ==
                dis[u] + 1 && dfs(u2))) {
                pairU[u] = v;
                pairV[v] = u;
                return true;
            }
        }
        dis[u] = -1;
        return false;
    }

    int run() {
        int match = 0;
        while(bfs()) {
            for(int u = 1; u <= n; u++) {
                if(pairU[u] == -1 &&
                    dfs(u)) {
                    match++;
                }
            }
        }
        return match;
    }
};
```

5.3 MCMF [150 lines] - a17481cbb4

```
/* Minimum Cost Maximum Flow (MCMF)
 * Works for both directed and undirected
     graphs, and supports negative edge
     costs.
 * Does NOT work with negative cost
     cycles.
 * For undirected edges, set `directed =
     false`.
 * Complexity: O(min(E^2 * V log V, E log
     V * flow))
 * Usage:
 *   MCMF mcmf(n);                    //
     Initialize with n nodes (0-indexed)
 *   mcmf.add_edge(u, v, cap, cost);  //
     Add edge from u to v with capacity
     and cost
 *   auto [maxFlow, minCost] =
     mcmf.solve(source, sink);
 */

using T = long long;
const T inf = 1LL << 61;

struct MCMF {
  struct edge {
    int u, v;
    T cap, cost;
    int id;
    edge(int _u, int _v, T _cap, T
        _cost, int _id)
      : u(_u), v(_v), cap(_cap),
          cost(_cost), id(_id) {}
  };

  int n, s, t, mxid;
  T flow, cost;
  bool neg;

  vector<vector<int>> g;     // Adjacency
      list: g[u] = indices of edges
  vector<edge> e;            // Edge list
  vector<T> d, potential;    // d =
      distances in Dijkstra; potential =
      reduced costs
  vector<T> flow_through;    // Stores
      flow pushed through original edges
      by ID
  vector<int> par;           // Parent
      edge index in path

  // Default constructor
  MCMF() {}

  // Initialize graph with n nodes
      (0-based indexing)
  MCMF(int _n) {
    n = _n + 10;
    g.assign(n, vector<int>());
    neg = false;
    mxid = 0;
  }

  // Adds an edge from u to v with given
      capacity and cost
  // If directed = false, adds edge in
      both directions
  void add_edge(int u, int v, T cap, T
      cost, int id = -1, bool directed =
      true) {
    if (cost < 0) neg = true;
    g[u].push_back(e.size());
    e.emplace_back(u, v, cap, cost, id);
    g[v].push_back(e.size());
    e.emplace_back(v, u, 0, -cost, -1);
        // Reverse edge
    mxid = max(mxid, id);
    if (!directed) add_edge(v, u, cap,
        cost, -1, true); // Add reverse
        only once
  }

  // Dijkstra with potentials (Johnson's
      algorithm) to find shortest path
  bool dijkstra() {
    par.assign(n, -1);
    d.assign(n, inf);
    priority_queue<pair<T, T>,
        vector<pair<T, T>>, greater<>>
        q;

    d[s] = 0;
    q.emplace(0, s);

    while (!q.empty()) {
      int u = q.top().second;
      T nw = q.top().first;
      q.pop();
      if (nw != d[u]) continue;

      for (int i : g[u]) {
        int v = e[i].v;
        T cap = e[i].cap;
        T w = e[i].cost + potential[u] -
            potential[v]; // Reduced cost
        if (cap > 0 && d[u] + w < d[v]) {
          d[v] = d[u] + w;
          par[v] = i;
          q.emplace(d[v], v);
        }
      }
    }

    for (int i = 0; i < n; i++) {
      if (d[i] < inf) potential[i] +=
          d[i]; // Update potentials
    }
```

```
  return d[t] != inf;  // True if sink
      is reachable
}

// DFS-style recursive flow sending
    function
T send_flow(int v, T cur) {
  if (par[v] == -1) return cur;
  int id = par[v];
  int u = e[id].u;
  T w = e[id].cost;
  T f = send_flow(u, min(cur,
      e[id].cap));
  cost += f * w;
  e[id].cap -= f;
  e[id ^ 1].cap += f;  // Update
      reverse edge
  return f;
}

// Main function to compute minimum
    cost max flow from s to t
// Optionally limit total flow with
    `goal`
// Returns {maximum flow, minimum cost}
pair<T, T> solve(int _s, int _t, T goal
    = inf) {
  s = _s; t = _t;
  flow = 0; cost = 0;
  potential.assign(n, 0);

  // If negative costs exist, use
      Bellman-Ford to initialize
      potentials
  if (neg) {
    d.assign(n, inf);
    d[s] = 0;
    for (int i = 0, relax = true; i < n
        && relax; ++i) {
      relax = false;
      for (int u = 0; u < n; ++u) {
        if (d[u] == inf) continue;
        for (int k : g[u]) {
          int v = e[k].v;
          T cap = e[k].cap, w =
              e[k].cost;
          if (cap > 0 && d[v] > d[u] +
              w) {
            d[v] = d[u] + w;
            relax = true;
          }
        }
      }
    }
    for (int i = 0; i < n; ++i) if
        (d[i] < inf) potential[i] =
        d[i];
  }

  // Repeatedly find augmenting paths
      using Dijkstra + potential
  while (flow < goal && dijkstra()) {
    flow += send_flow(t, goal - flow);
  }

    // Track flow through original edges
        (by ID)
    flow_through.assign(mxid + 10, 0);
    for (int u = 0; u < n; ++u) {
      for (int idx : g[u]) {
        if (e[idx].id >= 0) {
          flow_through[e[idx].id] = e[idx
              ^ 1].cap;
        }
      }
    }

    return {flow, cost};
  }
};
```

5.4 dinic [91 lines] - 1ea72f97f9

```
// Dinic's Algorithm for Maximum Flow
// Complexity: O(V^2 * E) in general
    graphs
// Usage:
//   1. Initialize with Dinic
    dinic(num_nodes);
//   2. Add edges with addEdge(u, v,
    capacity);
//   3. Compute maximum flow with
    maxFlow(source, sink);

#define eb emplace_back

struct Dinic {
  // Structure to represent an edge in
      the flow graph
  struct Edge {
    int u, v;         // from node u to
        node v
    ll cap, flow = 0; // capacity and
        current flow

    Edge() {}
    Edge(int u, int v, ll cap) : u(u),
        v(v), cap(cap) {}
  };

  int N;  // Number of nodes in the graph
  vector<Edge> edge;              // List
      of all edges
  vector<vector<int>> adj;        //
      Adjacency list of edge indices
  vector<int> d, pt;              //
      Distance (level) and pointer arrays

  // Constructor: Initializes flow
      network for N nodes
  Dinic(int N) : N(N), edge(0), adj(N),
      d(N), pt(N) {}

  // Adds a directed edge from u to v
      with given capacity
  // Adds reverse edge with 0 capacity
      for residual graph
  void addEdge(int u, int v, ll cap) {
    if (u == v) return;  // No self-loops
    edge.eb(u, v, cap);
    adj[u].eb(edge.size() - 1);
    edge.eb(v, u, 0);    // Reverse edge
    adj[v].eb(edge.size() - 1);
  }

  // Constructs level graph using BFS
      from source s
  // Returns true if sink t is reachable
      from s
  bool bfs(int s, int t) {
    queue<int> q({s});
    fill(d.begin(), d.end(), N + 1);  //
        Initialize all distances to
        "infinity"
    d[s] = 0;

    while (!q.empty()) {
      int u = q.front(); q.pop();
      if (u == t) break;  // Early stop
          if sink is reached
      for (int k : adj[u]) {
        Edge& e = edge[k];
        if (e.flow < e.cap && d[e.v] >
            d[u] + 1) {
          d[e.v] = d[u] + 1;
          q.emplace(e.v);
        }
      }
    }
    return d[t] != N + 1;  // If sink is
        reachable
  }

  // DFS to find augmenting path from u
      to T with available flow
  // Returns amount of flow pushed
      through the path
  ll dfs(int u, int T, ll flow = -1) {
    if (u == T || flow == 0) return flow;
    for (int& i = pt[u]; i <
        adj[u].size(); i++) {
      Edge& e = edge[adj[u][i]];
      Edge& oe = edge[adj[u][i] ^ 1];  //
          Reverse edge
      if (d[e.v] == d[u] + 1) {        //
          Valid level edge
        ll amt = e.cap - e.flow;       //
            Residual capacity
        if (flow != -1 && amt > flow) amt
            = flow; // Limit to available
            flow
        if (ll pushed = dfs(e.v, T, amt))
            {
          e.flow += pushed;
          oe.flow -= pushed;           //
              Adjust reverse flow
          return pushed;
        }
      }
    }
    return 0;  // No augmenting path
        found
  }

  // Computes the maximum flow from
      source s to sink t
  // Returns total flow value
  ll maxFlow(int s, int t) {
    ll total = 0;
    while (bfs(s, t)) {              //
        While there exists an augmenting
        path
      fill(pt.begin(), pt.end(), 0); //
          Reset current edge pointers
      while (ll flow = dfs(s, t)) {  //
          Push as much flow as possible
        total += flow;
      }
    }
    return total;
  }
};
```

5.5 flow [6 lines] - 071f5a23a9

```
Covering Problems:
> Maximum Independent Set(Bipartite):
    Largest set of nodes which do not
    have any edge between them. sol:
    V-(MaxMatching)
> Minimum Vertex Cover(Bipartite):
    -Smallest set of nodes to cover all
    the edges -sol: MaxMatching
> Minimum Edge Cover(General graph):
    -Smallest set of edges to cover all
    the nodes -sol: V-(MaxMatching) (if
    edge cover exists, does not exit for
    isolated nodes)
> Minimum Path Cover(Vertex disjoint)
    DAG: -Minimum number of vertex
    disjoint paths that visit all the
    nodes -sol: make a bipartite graph
    using same nodes in two sides, one
    side is "from" other is "to", add
    edges from "from" to "to", then ans
    is V-(MaxMatching)
> Minimum Path Cover(Vertex Not Disjoint)
    General graph: -Minimum number of
    paths that visit all the nodes -sol:
    consider cycles as nodes then it will
    become a path cover problem with
    vertex disjoint on DAG
```

5.6 hungerian [60 lines] - 699d538345

```
struct Hungerian {
    vector<vector<int>> cost;
    vector<int> worker, job, parentjob,
        matchjob, assign;
    int n;
    Hungerian(int n) : n(n) {
        cost.resize(n, vector<int>(n));
        worker.resize(n + 1);
        job.resize(n + 1);
        parentjob.resize(n + 1);
        matchjob.resize(n + 1);
    }
    int solve() {
        for(int w = 1; w <= n; w++) {
            matchjob[0] = w;
            vector<int> minslack(n + 1,
                INT_MAX);
            vector<bool> used(n + 1, 0);
            int job0 = 0;
```

```
        do {
            used[job0] = true;
            int worker0 =
                matchjob[job0], delta
                = INT_MAX, job1 = 0;
            for(int j = 1; j <= n;
                j++) {
                if(!used[j]) {
                    int curcost =
                        cost[worker0
                        - 1][j - 1]
                        -
                        worker[worker0]
                        - job[j];
                    if(curcost <
                        minslack[j])
                        {
                        minslack[j]
                            =
                            curcost;
                        parentjob[j]
                            = job0;
                    }
                    if(minslack[j] <
                        delta) {
                        delta =
                            minslack[j];

                        job1 = j;
                    }
                }
            }
            for(int j = 0; j <= n;
                j++) {
                if(used[j]) {
                worker[matchjob[j]]
                    += delta;
                job[j] -= delta;
                }
                else {
                    minslack[j] -=
                        delta;
                }
            }
            job0 = job1;
        } while(matchjob[job0] != 0);

        do {
            int job1 =
                parentjob[job0];
            matchjob[job0] =
                matchjob[job1];
            job0 = job1;
        } while(job0);
    }

    assign.assign(n + 1, 0);
    for(int j = 1; j <= n; j++) {
        assign[matchjob[j]] = j;
    }

    return -job[0];
  }
};
```

6 geometry

6.1 basic geo [101 lines] - 187dfc5049

```
// basic geometry
// be careful! [th in radians]
Chord Length of a Circle
L = 2r * sin(th / 2)
Arc Length of a Circle
L = r * th
Sector Area of a Circle
A = (1/2) * r^2 * th
Area of Segment = Area of Sector - Area
    of Triangle
A = (1/2) * r^2 * (th - sin(th))
Length of a Common Chord of Two Circles
L = 2 * sqrt( R^2 - ((d^2 + R^2 - r^2) /
    (2d))^2 )

kissing circles
The plus sign gives the curvature of the
    inner circle tangent to the three.
The minus sign gives the curvature of the
    outer circle enclosing the three.
k4 = k1 + k2 + k3 ± 2 * sqrt(k1*k2 +
    k2*k3 + k3*k1)
// where ki = 1 / ri (curvature), ri =
    radius of circle i

// Given medians ma, mb, mc
s = (ma + mb + mc) / 2
D = sqrt(s * (s - ma) * (s - mb) * (s -
    mc))
A = (4 / 3) * D  // Area of original
    triangle

//common formula is Area
= (1/2) * perimeter * apothem

// General Polygon
Each_Interior_Angle = ((n - 2) * pi) / n
Number_of_Diagonals = n * (n - 3) / 2

// Regular Polygon
// a = length of one side
// R = circumradius (radius of
    circumscribed circle)
// r = apothem (inradius, perpendicular
    from center to a side)
Side, a = 2 * R * sin(pi / n)
Apothem, r = a / (2 * tan(pi / n))
Circumradius = a / (2 * sin(pi / n))
Central_Angle = 360 / n
Area_from_side = (n * a^2) / (4 * tan(pi
    / n))
Area_from_apothem = (1/2) * n * a * r
Area_from_R = (1/2) * n * R^2 * sin(2pi /
    n)

// Volume of a spherical cap of height h
V_cap = (pi * h^2 * (3*R - h)) / 3
// Total volume of sphere
V_total = (4.0 / 3.0) * pi * R^3

// s = semi-perimeter = (a + b + c) / 2
// R = circumradius (radius of
    circumcircle)
// r = inradius (radius of incircle)
// Ra, Rb, Rc = exradii opposite to a, b,
    c respectively

// Incircle (circle tangent to all sides
    inside triangle)
Inradius = r = A / s //onto
// Circumcircle (passes through all
    triangle vertices)
Circumradius = R = (a * b * c) / (4 * A)
    //pori
// Excircle (touches one side and
    extensions of other two)
Exradius_Opposite_a = Ra = A / (s - a)



// Triangle Area Formulas
// Equilateral Triangle (all sides equal)
Area = (sqrt(3) / 4) * a^2
// Isosceles Triangle (two sides equal)
Area = (b / 4) * sqrt(4a^2 - b^2)


// 3D Volume Formulas
// Symbol Definitions:
// l = length, w = width, h = height
// a = edge length
// r = radius, d = diameter
// R = base radius (for cone/cylinder), H
    = height
// Cylinder
Volume = pi * r^2 * h
// Cone
Volume = (1/3) * pi * r^2 * h
// Sphere
Volume = (4/3) * pi * r^3
// Hemisphere
Volume = (2/3) * pi * r^3
// Triangular Prism
Volume = (1/2) * b * h * l
// where b = triangle base, h = triangle
    height, l = length/depth
// Pyramid (square or rectangular base)
Volume = (1/3) * base_area * height
// Tetrahedron (regular, edge = a)
Volume = (a^3) / (6 * sqrt(2))
// Frustum (of cone or pyramid)
Volume = (1/3) * pi * h * (R^2 + r^2 +
    R*r)
// R = larger base radius, r = smaller
    radius, h = vertical height
// Torus (donut shape)
Volume = 2 * pi^2 * R * r^2
// R = distance from center of tube to
    center of torus
// r = radius of tube
```

6.2 custom geo [404 lines] - aa41681d08

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;
const int N = 1e7 + 9, mod = 1e9 + 7;
const ld inf = 1e100;
const ld eps = 1e-9;
const ld PI = acos((ld)-1.0);
int sign(ld x) { return (x > eps) - (x <
    -eps); }
struct PT {
    ll x, y;
    PT() { x = 0, y = 0; }
    PT(ll x, ll y) : x(x), y(y) {}
    PT operator - (const PT &a) const {
        return PT(x - a.x, y - a.y); }
    PT operator * (const ll a) const {
        return PT(x * a, y * a); }
    friend PT operator * (const ll &a,
        const PT &b) { return PT(a *
        b.x, a * b.y); }
    PT operator / (const ll a) const {
        return PT(x / a, y / a); }
    bool operator == (PT a) const {
        return sign(a.x - x) == 0 &&
        sign(a.y - y) == 0; }
    bool operator != (PT a) const {
        return !(*this == a); }
    bool operator < (PT a) const { return
        sign(a.x - x) == 0 ? y < a.y : x
        < a.x; }
    bool operator > (PT a) const { return
        sign(a.x - x) == 0 ? y > a.y : x
        > a.x; }
};
istream &operator >> (istream &in, PT &p)
    { return in >> p.x >> p.y; }
ostream &operator << (ostream &out, PT
    &p) { return out << "(" << p.x << ","
    << p.y << ")"; }
inline ll dot(PT a, PT b) { return a.x *
    b.x + a.y * b.y; }
inline ll dist2(PT a, PT b) { return
    dot(a - b, a - b); }
inline double dist(PT a, PT b) { return
    sqrt(dot(a - b, a - b)); }
inline double cross(PT a, PT b) { return
    a.x * b.y - a.y * b.x; }
inline double cross2(PT a, PT b, PT c) {
    return cross(b - a, c - a); }
inline int orientation(PT a, PT b, PT c)
    { return sign(cross(b - a, c - a)); }

// intersection point between segment ab
    and segment cd assuming unique
    intersection exists
bool seg_seg_intersection(PT a, PT b, PT
    c, PT d, PT &ans) {
    ld oa = cross2(c, d, a), ob =
        cross2(c, d, b);
    ld oc = cross2(a, b, c), od =
        cross2(a, b, d);
    if (oa * ob < 0 && oc * od < 0){
        ans = (a * ob - b * oa) / (ob -
            oa);
        return 1;
    }
```

```
    else return 0;
}
// returns true if  point p is on line
    segment ab
bool is_point_on_seg(PT a, PT b, PT p) {
    if (fabs(cross(p - b, a - b)) < eps)
        {
        if (p.x < min(a.x, b.x) - eps ||
            p.x > max(a.x, b.x) + eps)
            return false;
        if (p.y < min(a.y, b.y) - eps ||
            p.y > max(a.y, b.y) + eps)
            return false;
        return true;
    }
    return false;
}
// intersection point between segment ab
    and segment cd assuming unique
    intersection exists
set<PT> seg_seg_intersection_inside(PT
    a,  PT b,  PT c,  PT d) {
    PT ans;
    if (seg_seg_intersection(a, b, c, d,
        ans)) return {ans};
    set<PT> se;
    if (is_point_on_seg(c, d, a))
        se.insert(a);
    if (is_point_on_seg(c, d, b))
        se.insert(b);
    if (is_point_on_seg(a, b, c))
        se.insert(c);
    if (is_point_on_seg(a, b, d))
        se.insert(d);
    return se;
}

ld area(vector<PT> &p) {
    ld ans = 0; int n = p.size();
    for (int i = 0; i < n; i++) ans +=
        cross(p[i], p[(i + 1) % n]);
    return fabs(ans) * 0.5;
}

ll area2(const vector<PT>& p) {
    ll ans = 0;
    int n = p.size();
    for (int i = 0; i < n; i++) {
        ans += cross(p[i], p[(i + 1) %
            n]);
    }
    return abs(ans); // not devided by 2
        for integers
}

bool is_point_on_polygon(vector<PT> &p,
    const PT& z) {
    int n = p.size();
    for (int i = 0; i < n; i++) {
        if (is_point_on_seg(p[i], p[(i +
            1) % n], z)) return 1;
    }
    return 0;
}
```

```
// returns 1e9 if the point is on the
    polygon
int winding_number(vector<PT> &p, const
    PT& z) { // O(n)
    if (is_point_on_polygon(p, z)) return
        1e9;
    int n = p.size(), ans = 0;
    for (int i = 0; i < n; ++i) {
        int j = (i + 1) % n;
        bool below = p[i].y < z.y;
        if (below != (p[j].y < z.y)) {
            auto orient = orientation(z,
                p[j], p[i]);
            if (orient == 0) return 0;
            if (below == (orient > 0))
                ans += below ? 1 : -1;
        }
    }
    return ans;
}

// -1 if strictly inside, 0 if on the
    polygon, 1 if strictly outside
int is_point_in_polygon(vector<PT> &p,
    const PT& z) { // O(n)
    int k = winding_number(p, z);
    return k == 1e9 ? 0 : k == 0 ? 1 :
        -1;
}

// -1 if strictly inside, 0 if on the
    polygon, 1 if strictly outside
// it must be strictly convex, otherwise
    make it strictly convex first
int is_point_in_convex(vector<PT> &p,
    const PT& x) { // O(log n)
    int n = p.size(); assert(n >= 3);
    int a = orientation(p[0], p[1], x), b
        = orientation(p[0], p[n - 1],
        x);
    if (a < 0 || b > 0) return 1;
    int l = 1, r = n - 1;
    while (l + 1 < r) {
        int mid = l + r >> 1;
        if (orientation(p[0], p[mid], x)
            >= 0) l = mid;
        else r = mid;
    }
    int k = orientation(p[l], p[r], x);
    if (k <= 0) return -k;
    if (l == 1 && a == 0) return 0;
    if (r == n - 1 && b == 0) return 0;
    return -1;
}

// Count boundary lattice points using
    GCD on each edge
ll boundary_lattice_points(const
    vector<PT>& p) {
    ll b = 0;
    int n = p.size();
    for (int i = 0; i < n; i++) {
        PT a = p[i], c = p[(i + 1) % n];
        b += __gcd(abs(a.x - c.x),
            abs(a.y - c.y));
    }
    return b;
}

// Computes (interior, boundary) lattice
    points using Pick's Theorem
pair<ll, ll> polygon_lattice_points(const
    vector<PT>& p) {
    // Pick's Theorem:
    // Area = I + B/2 - 1
    // I = the number of interior lattice
        points
    // B = the number of boundary lattice
        points
    ll A2 = area2(p);               //
        Twice the area
    ll B = boundary_lattice_points(p);
    ll I = (A2 - B + 2) / 2;
    return {I, B};
}

// distance, p1, p2
pair<ll, pair<PT, PT>>
    closest_pair_recursive(vector<PT>&
    pts, vector<PT>& tmp, int l, int r) {
    if (r - l <= 3) {
        pair<ll, pair<PT, PT>> res =
            {LLONG_MAX, {PT(), PT()}};
        for (int i = l; i < r; ++i) {
            for (int j = i + 1; j < r;
                ++j) {
                ll d = dist2(pts[i],
                    pts[j]);
                if (d < res.first) {
                    res = {d, {pts[i],
                        pts[j]}};
                }
            }
        }
        sort(pts.begin() + l, pts.begin()
            + r, [](const PT& a, const
            PT& b) {
            return a.y < b.y;
        });
        return res;
    }

    int m = (l + r) / 2;
    ll midx = pts[m].x;

    auto left =
        closest_pair_recursive(pts, tmp,
        l, m);
    auto right =
        closest_pair_recursive(pts, tmp,
        m, r);
    auto res = min(left, right);

    merge(pts.begin() + l, pts.begin() +
        m,
          pts.begin() + m, pts.begin() +
              r,
          tmp.begin() + l, [](const PT&
              a, const PT& b) {
              return a.y < b.y;
          });
    copy(tmp.begin() + l, tmp.begin() +
        r, pts.begin() + l);

    vector<PT> strip;
    for (int i = l; i < r; ++i) {
        if ((pts[i].x - midx) * (pts[i].x
            - midx) >= res.first)
            continue;
        for (int j = strip.size() - 1; j
            >= 0; --j) {
            ll dy = pts[i].y -
                strip[j].y;
            if (dy * dy >= res.first)
                break;
            ll d = dist2(pts[i],
                strip[j]);
            if (d < res.first) {
                res = {d, {pts[i],
                    strip[j]}};
            }
        }
        strip.push_back(pts[i]);
    }

    return res;
}

// distance, point1, point2
pair<ll, pair<PT, PT>>
    minimum_euclidean_distance(vector
    <PT>& pts) {
    int n = pts.size();
    sort(pts.begin(), pts.end(), [](const
        PT& a, const PT& b) {
        return a.x < b.x;
    });
    vector<PT> tmp(n);
    return closest_pair_recursive(pts,
        tmp, 0, n);
}

// returns the boundary points of the
    convex hull.
vector<PT> ConvexHull(vector<PT>&p, int
    n) {
    int sz = 0;
    vector<PT> hull(n + n);
    sort(p.begin(), p.end());
  for(int i = 0; i < n; ++i) {
        while (sz > 1 and cross2(hull[sz
            - 2], hull[sz - 1], p[i]) <
            0) --sz;
        hull[sz++] = p[i];
    }
    for(int i = n - 2, j = sz + 1; i >=
        0; --i) {
        while (sz >= j and cross2(hull[sz
            - 2], hull[sz - 1], p[i]) <
            0) --sz;
        hull[sz++] = p[i];
    }
    hull.resize(sz - 1);
    return hull;
```

```
}


// keep only the corners that form a
    strictly-convex polygon
vector<PT> make_strict_convex(const
    vector<PT>& P) {
    int n = (int)P.size();
    vector<PT> R;
    for (int i = 0; i < n; ++i) {
        PT prv = P[(i + n - 1) % n], cur
            = P[i], nxt = P[(i + 1) %
            n];
        if (orientation(prv, cur, nxt)
            != 0) R.push_back(cur);
    }
    return R;
}

// return true if a polygon strictly
    insider other
bool
    polygon_polygon_intersect(vector<PT>
    v1, vector<PT> v2) {
    auto H1 = ConvexHull(v1, v1.size());
    auto H2 = ConvexHull(v2, v2.size());
    vector<PT> all = v1;
    for(auto it : v2) {
        all.push_back(it);
    }
    auto H = ConvexHull(all, all.size());
    bool ok = (H == H1 || H == H2);
    return ok;
    /*v1 = make_strict_convex(v1);
    reverse(v1.begin(), v1.end());
    v2 = make_strict_convex(v2);
    reverse(v2.begin(), v2.end());
    if (v1.size() < 3 || v2.size() < 3) {
        return false;
    }
    bool ok = true;
    for (const auto &pt : v2)
        if (is_point_in_convex(v1, pt) !=
            -1) { ok = false; break; }

    if (!ok) {

        ok = true;
        for (const auto &pt : v1)
            if (is_point_in_convex(v2,
                pt) != -1) { ok = false;
                break; }
    }
    return ok;*/
}

// Projection of point c onto line ab
PT project_from_point_to_line(PT a, PT
    b, PT c) {
    return a + (b - a) * dot(c - a, b -
        a) / dot(b - a, b - a);
}

// Reflection of point c across line ab
PT reflection_from_point_to_line(PT a, PT
    b, PT c) {
    PT p = project_from_point_to_line(a,
        b, c);
    return p + (p - c);
}

// Minimum distance from point c to line
    ab
ld dist_from_point_to_line(PT a, PT b, PT
    c) {
    return fabs(cross(b - a, c - a)) /
        sqrtl(dist2(a, b));
}

// Minimum distance from point c to
    segment ab
ld dist_from_point_to_seg(PT a, PT b, PT
    c) {
    ll r = dist2(a, b);
    if (r == 0) return dist(c, a);
    ld t = max(0.0L, min(1.0L, (ld)dot(c
        - a, b - a) / r));
    PT proj = a + (b - a) * t;
    return dist(c, proj);
}

// Minimum distance between two segments
    ab and cd
ld dist_from_seg_to_seg(PT a, PT b, PT
    c, PT d) {
    PT dummy;
    if (seg_seg_intersection(a, b, c, d,
        dummy)) return 0;
    return min({
        dist_from_point_to_seg(a, b, c),
        dist_from_point_to_seg(a, b, d),
        dist_from_point_to_seg(c, d, a),
        dist_from_point_to_seg(c, d, b)
    });
}

// Checks if a polygon is strictly convex
bool is_convex(const vector<PT>& p) {
    int n = p.size();
    bool has_pos = false, has_neg =
        false;
    for (int i = 0; i < n; i++) {
        ll z = cross2(p[i], p[(i + 1) %
            n], p[(i + 2) % n]);
        has_pos |= (z > 0);
        has_neg |= (z < 0);
    }
    return !(has_pos && has_neg);
}

// Centroid (center of mass) of a polygon
PT centroid(const vector<PT>& p) {
    ld A = 0;
    PT c(0, 0);
    int n = p.size();
    for (int i = 0; i < n; i++) {
        ll cross_val = cross(p[i], p[(i
            + 1) % n]);
        A += cross_val;
        c.x += (p[i].x + p[(i + 1) %
            n].x) * cross_val;
        c.y += (p[i].y + p[(i + 1) %
            n].y) * cross_val;
    }
    A *= 0.5;
    c.x /= (6.0 * A);
    c.y /= (6.0 * A);
    return c;
}

// Angle between vectors (in radians)
ld get_angle(PT a, PT b) {
    ld costheta = (ld)dot(a, b) /
        sqrtl(dist2(a, {0, 0}) *
        dist2(b, {0, 0}));
    return acos(max((ld)-1.0,
        min((ld)1.0, costheta)));
}

// Intersection point of two infinite
    lines (ab and cd), returns false if
    parallel
bool line_line_intersection(PT a, PT b,
    PT c, PT d, PT &ans) {
    ld a1 = a.y - b.y, b1 = b.x - a.x, c1
        = cross(a, b);
    ld a2 = c.y - d.y, b2 = d.x - c.x, c2
        = cross(c, d);
    ld det = a1 * b2 - a2 * b1;
    if (fabs(det) < eps) return false;
    ans = PT((b1 * c2 - b2 * c1) / det,
        (c1 * a2 - a1 * c2) / det);
    return true;
}

// Check if lines ab and cd are parallel
    or collinear
// 0 = not parallel, 1 = parallel, 2 =
    collinear
int is_parallel(PT a, PT b, PT c, PT d) {
    ld cross1 = fabs(cross(b - a, d -
        c));
    if (cross1 < eps) {
        if (fabs(cross(a - b, a - c)) <
            eps && fabs(cross(c - d, c -
            a)) < eps) return 2;
        else return 1;
    }
    return 0;
}

// Angle bisector vector of angle <abc
PT angle_bisector(PT a, PT b, PT c) {
    PT p = a - b, q = c - b;
    return p + q * sqrtl((ld)dot(p, p) /
        dot(q, q));
}

// Point at distance d from a towards b
    (along line ab)
PT point_along_line(PT a, PT b, ld d) {
    assert(a != b);
    PT v = b - a;
    ld len = sqrtl(dot(v, v));
    return a + v * (d / len);
}

// Rotate a point counter-clockwise by
    angle t (radians) around origin
PT rotateccw(PT a, ld t) {
    return PT(a.x * cos(t) - a.y *
        sin(t), a.x * sin(t) + a.y *
        cos(t));
}

// Rotate a point clockwise by angle t
    (radians) around origin
PT rotatecw(PT a, ld t) {
    return PT(a.x * cos(t) + a.y *
        sin(t), -a.x * sin(t) + a.y *
        cos(t));
}

// 90-degree counter-clockwise rotation
PT rotateccw90(PT a) { return PT(-a.y,
    a.x); }

// 90-degree clockwise rotation
PT rotatecw90(PT a) { return PT(a.y,
    -a.x); }

// Check point in triangle (a, b, c)
// -1 = inside, 0 = on edge, 1 = outside
int is_point_in_triangle(PT a, PT b, PT
    c, PT p) {
    if (sign(cross(b - a, c - a)) < 0)
        swap(b, c);
    int c1 = sign(cross(b - a, p - a));
    int c2 = sign(cross(c - b, p - b));
    int c3 = sign(cross(a - c, p - c));
    if (c1 < 0 || c2 < 0 || c3 < 0)
        return 1;
    if (c1 + c2 + c3 != 3) return 0;
    return -1;
}
```

6.3 my geo [77 lines] - 0869646f06

```
struct pt {
    double x, y;
    pt() {}
    pt(double _x, double _y) {
        x = _x;
        y = _y;
    }
    pt operator + (const pt &o) const {
        return pt(x + o.x, y + o.y); }
    pt operator - (const pt &o) const {
        return pt(x - o.x, y - o.y); }
    ld operator | (const pt &o) const {
        return x * o.x + y * o.y; }
    ld operator * (const pt &o) const {
        return x * o.y - y * o.x; }
    ld dis() { return hypotl(x, y); }
    bool operator < (const pt &o) const {
        return (x < o.x || (x == o.x && y
            < o.y));
    }
    bool operator == (const pt &o) const
        {
```

```
        return (x == o.x && y == o.y);
    }
};
ld pt_to_seg_dis(pt p, pt a, pt b) {
    if(((p - a) | (b - a)) < 0) {
        return (p - a).dis();
    }
    else {
        if(((p - b) | (a - b)) < 0) {
            return (p - b).dis();
        }
        else {
            return fabs(((p - b) * (b -
                a)) / (b - a).dis());
        }
    }
}
ld dis_bet_seg(pt a, pt b, pt c, pt d) {
    ld dot1 = ((b - a) * (d - a)) * ((b
        - a) * (c - a));
    ld dot2 = ((d - c) * (a - c)) * ((d
        - c) * (b - c));
    if(dot1 <= 0 && dot2 <= 0) {
        cout << 0 << "\n";
        return;
    }
    ld ans = min(pt_to_seg_dis(c, a, b),
        pt_to_seg_dis(d, a, b));
    ans = min(ans, pt_to_seg_dis(a, c,
        d));
    ans = min(ans, pt_to_seg_dis(b, c,
        d));
    return ans;
}
int orientation(pt a, pt b, pt c) {
    ld val = (b - a) * (c - a);
    if (val > 0) return 1;
    if (val < 0) return -1;
    return 0;
}
vector<pt> convex_hull(vector<pt> v) {
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()),
        v.end());
    int n = v.size();
    if (n <= 2) return v;
    vector<pt> lower, upper;
    for (int i = 0; i < n; i++) {
        int j = lower.size();
        while (j >= 2 &&
            orientation(lower[j-2],
            lower.back(), v[i]) != 1) {
            lower.pop_back();
            j--;
        }
        lower.push_back(v[i]);
    }
    for (int i = n-1; i >= 0; i--) {
        int j = upper.size();
        while (j >= 2 &&
            orientation(upper[j-2],
            upper.back(), v[i]) != 1) {
            upper.pop_back();
            j--;
        }
        upper.push_back(v[i]);
    }
    lower.pop_back();
    upper.pop_back();
    lower.insert(lower.end(),
        upper.begin(), upper.end());
    return lower;
}
```

7 graphs

7.1 articulations [22 lines] - 075d74c576

```
const int N=1e5+5;
vector<int> g[N];
int vis[N],dis[N],lo[N],isAP[N];
int timer;
vector<pair<int,int>> ans;
void dfs(int src, int par)
{
    int child = 0;
    vis[src] = true;
    dis[src] = lo[src] = ++timer;
    for (auto v : g[src]) {
        if (!vis[v]) {
            child++;
            dfs(v,src);
            lo[src] = min(lo[src],
                lo[v]);
            if (par != -1 && lo[v] >=
                dis[src]) isAP[src] =
                true;
            if(lo[v]>dis[src])
                ans.push_back({min(v,
                src), max(v,src)});
        }
        else if (v != par) lo[src] =
            min(lo[src], dis[v]);
    }
    if (par == -1 && child > 1) isAP[src]
        = true;
}
```

7.2 bellman full [82 lines] - 646df1f54a

```
// if reach can possible from 1 to n
vector<array<int, 3>> edges;
vector<vector<int>> g(n + 1);
edges.push_back({u, v, -w});
g[v].push_back(u);
vector<ll> dis(n + 1, inf);
dis[1] = 0;
// n - 1 relaxation
for(int c = 1; c < n; c++) {
    bool relax = false;
    for(auto [u, v, w] : edges) {
        if(dis[u] != inf && dis[u] + w <
            dis[v]) {
            dis[v] = dis[u] + w;
            relax = true;
        }
    }
    if(!relax) false;
}
vector<bool> bad(n);
for(auto [u, v, w] : edges) {
    if(dis[u] != inf && dis[u] + w <
        dis[v]) {
        bad[v] = 1;
    }
}
bool flag = 1;
queue<int> Q;
Q.push(n);
vector<bool> vis(n + 1);
while(not Q.empty()) {
    int u = Q.front(); Q.pop();
    if(bad[u]) {
        flag = 0;
        break;
    }
    if(vis[u]) continue;
    vis[u] = 1;
    for(auto v : g[u]) {
        Q.push(v);
    }
}
// negatative cycle print
edges.push_back({u, v, w});
vector<ll> dis(n + 1, inf);
vector<int> par(n + 1, -1);
int node = -1;
for(int i = 1; i <= n; i++) {
    if(dis[i] != inf) continue;
    dis[i] = 0;
    for(int c = 1; c <= n; c++) {
        bool relax = 1;
        for(auto [u, v, w] : edges) {
            if(dis[u] != inf && dis[u] +
                w < dis[v]) {
                dis[v] = dis[u] + w;
                par[v] = u;
                if(c == n) node = v;
                relax = 0;
            }
        }
        if(relax) break;
    }
    if(node != -1) break;
}
if(node == -1) {
    cout << "NO\n";
}
else {
    // Get a node in the cycle by walking
        back n times
    for(int i = 0; i < n; i++) node =
        par[node];

    vector<int> path;
    int cur = node;
    do {
        path.push_back(cur);
        cur = par[cur];
    } while(cur != node);
    path.push_back(node);
    reverse(path.begin(), path.end());

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

7.3 centroid [124 lines] - 6d12aab948

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

/* ----------  Centroid-decomposition
    with fast depth counting  ----------
    */
struct CentroidTree {
    int n;
    vector<vector<int>> adj, ctree;
    vector<int>         subsize, cpar;
    vector<char>        blocked;
        // faster than vector<bool>

    /* reusable frequency table */
    vector<int> cnt;
        // size = K+1, filled on demand
    vector<int> touched;
        // indices of cnt we modified

    CentroidTree(int nodes) : n(nodes) {
        adj.resize(n + 1);
        ctree.resize(n + 1);
        subsize.resize(n + 1);
        cpar.assign(n + 1, -1);
        blocked.assign(n + 1, 0);
    }

    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    /* -------------  centroid
        decomposition boiler-plate
        ------------- */
    void calculate_size(int u, int p) {
        subsize[u] = 1;
        for (int v : adj[u])
            if (v != p && !blocked[v]) {
                calculate_size(v, u);
                subsize[u] += subsize[v];
            }
    }
    int get_centroid(int u, int p, int
        sz) {
        for (int v : adj[u])
            if (v != p && !blocked[v] &&
                subsize[v] > sz / 2)
                return get_centroid(v,
                    u, sz);
        return u;
    }

    /* ----------------  fast
        depth-multiset helpers
        ---------------- */
    inline void add_depth(int d) {
        if (cnt[d] == 0)
            touched.push_back(d);
        ++cnt[d];
    }
    inline void clear_touched() {
        for (int d : touched) cnt[d] = 0;
        touched.clear();
    }
```

```
void collect_depths(int u, int p, int
    d, vector<int>& depths, int K) {
    if (d > K) return;
    depths.push_back(d);
    for (int v : adj[u])
        if (v != p && !blocked[v])
            collect_depths(v, u, d +
                1, depths, K);
}

/* count all paths of exact length K
    that go through centroid c */
void process_centroid(int c, int K,
    long long& ans) {
    add_depth(0);
        // centroid itself
    for (int v : adj[c]) if
        (!blocked[v]) {
        vector<int> depths;
        collect_depths(v, c, 1,
            depths, K);

        for (int d : depths)
            // pairs: one in this
            subtree,
            if (d <= K)
                // other in previous
                ones / root
                ans += cnt[K - d];

        for (int d : depths)
            // merge current subtree
            depths
            if (d <= K) add_depth(d);
    }
    clear_touched();
        // reset counter
}

void decompose(int u, int p, int K,
    long long& ans) {
    calculate_size(u, -1);
    int c = get_centroid(u, -1,
        subsize[u]);
    blocked[c] = 1;
    cpar[c] = p;

    process_centroid(c, K, ans);

    if (p != -1) {
        ctree[p].push_back(c);
        ctree[c].push_back(p);
    }
    for (int v : adj[c])
        if (!blocked[v])
            decompose(v, c, K, ans);
}

long long count_paths_of_length_k(int
    K) {
    cnt.assign(K + 1, 0);
        // initialise once
    touched.clear();
    long long ans = 0;
    decompose(1, -1, K, ans);
        return ans;
    }

    /* optional helpers kept intact */
    int get_centroid_parent(int u) {
        return cpar[u]; }
    vector<int>&
        get_centroid_children(int u) {
        return ctree[u]; }
};

/* -------------------------------  main
    ------------------------------- */
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

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

    CentroidTree ct(n);
    for (int i = 0; i < n - 1; ++i) {
        int u, v; cin >> u >> v;
        ct.add_edge(u, v);
    }
    cout << ct.count_paths_of_length_k(k)
        << '\n';
    return 0;
}
```

7.4 dsu on tree [97 lines] - 94c0401be0

```
struct DSUTree {
    int n; // Number of nodes in the tree

    // Graph structure and metadata
    vector<vector<int>> adj;            //
        Original tree (adjacency list)
    vector<int> color, sz, res;         //
        color[i]: color of node i, sz[i]:
        subtree size, res[i]: result for
        node i
    vector<map<int, int>*> cnt;         //
        cnt[i]: pointer to map storing
        color frequencies in subtree of
        node i

    // Constructor: initialize vectors
        and process input + solve
    DSUTree(int n) : n(n) {
        adj.resize(n + 1);
        color.resize(n + 1);
        sz.resize(n + 1);
        res.resize(n + 1);
        cnt.resize(n + 1, nullptr);
        take_input(); // Read input
        build();      // Run DSU on tree
    }

    // Read color input and edges
    void take_input() {
        for (int i = 1; i <= n; i++) {
            cin >> color[i];
        }
        for (int i = 1; i < n; i++) {
            int u, v;
            cin >> u >> v;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
    }

    // Step 1: Compute sizes of all
        subtrees
    void compute_subtree_sizes(int u, int
        p) {
        sz[u] = 1;
        for (int v : adj[u]) {
            if (v != p) {
                compute_subtree_sizes(v,
                    u);
                sz[u] += sz[v];
            }
        }
    }

    // Step 2: Perform DSU on Tree to
        count distinct colors
    void dfs(int u, int p) {
        int big = -1, max_size = -1;

        // Find the heavy child (largest
            subtree)
        for (int v : adj[u]) {
            if (v != p && sz[v] >
                max_size) {
                max_size = sz[v];
                big = v;
            }
        }

        // Process all light children
            first (their data will be
            deleted)
        for (int v : adj[u]) {
            if (v != p && v != big)
                dfs(v, u);
        }

        // Process heavy child and reuse
            its color map
        if (big != -1) {
            dfs(big, u);
            cnt[u] = cnt[big]; // Inherit
                heavy child's map
        } else {
            cnt[u] = new map<int,
                int>(); // Create new map
                for leaf
        }

        // Add current node's color
        (*cnt[u])[color[u]]++;

        // Merge all light child maps
            into current node's map
        for (int v : adj[u]) {
            if (v == p || v == big)
                continue;
            for (auto &[col, freq] :
                *cnt[v]) {
                (*cnt[u])[col] += freq;
            }

        // Store the number of distinct
            colors in the subtree of u
        res[u] = cnt[u]->size();
    }

    // Entry point: preprocess and call
        DFS
    void build() {
        compute_subtree_sizes(1, 0); //
            Root the tree at node 1
        dfs(1, 0);                   //
            Start DFS from root
    }

    // Output the result
    void print() {
        for (int i = 1; i <= n; i++) {
            cout << res[i] << " "; //
                Output result for node i
        }
        cout << '\n';
    }
};
```

7.5 euler graph [99 lines] - 9ea81ac57c

```
/*
all the edges should be in the same
    connected component
#undirected graph: euler path/trail: all
    degrees are even or exactly two of
    them are odd.
#undirected graph: euler
    circuit/cylce[start == end]: all
    degrees are even
#directed graph: euler path: for all ->
    indeg = outdeg or
in - out == 1 && out - in == 1
#directed graph: euler circuit: for all
    -> indeg = outdeg
*/
struct EulerGraph {
    vector<vector<pair<int, int>>> g;
    vector<int> done, path, edges,
        indeg, outdeg;
    vector<bool> visedge;
    bool isdirected;
    int n, m;
    EulerGraph(int n, int m, bool isd =
        1) : n(n), m(m) {
        g.resize(n + 1);
        done.assign(n + 1, 0);
        indeg.assign(n + 1, 0);
        outdeg.assign(n + 1, 0);
        visedge.assign(m + 1, 0);
        isdirected = isd;
        take_input();
    }
    void take_input() {
        for(int i = 1; i <= m; i++) {
            int u, v;
            cin >> u >> v;
            u++, v++;
```

```
        g[u].push_back({v, i});
        indeg[v]++, outdeg[u]++;
        if(!isdirected) {
            g[v].push_back({u, i});
            indeg[u]++;
            outdeg[v]++;
        }
    }
}
void dfs(int u) {
    while(done[u] < g[u].size()) {
        auto [v, eid] =
            g[u][done[u]++];
        if(!isdirected) {
            if(visedge[eid])
                continue;
            visedge[eid] = 1;
        }
        dfs(v);
        edges.push_back(eid);
    }
    path.push_back(u);
}
pair<int, bool> find_start() { //
    true means any otherwise fixed
    if(isdirected) {
        bool allsame = 1;
        for(int i = 1; i <= n; i++) {
            if(indeg[i] != outdeg[i])
                {
                allsame = 0;
                break;
            }
        }
        if(allsame) return {-1, 1};
            // any can be root

        int incnt = 0, outcnt = 0,
            node = -1; // must be 1 1
            and remaining same
        for(int i = 1; i <= n; i++) {
            if((outdeg[i] - indeg[i])
                > 1 or (indeg[i] -
                outdeg[i]) > 1) {
                return {-1, 0};
            }
            if((outdeg[i] - indeg[i])
                == 1) {
                node = i;
                outcnt++;
            }
            if((outdeg[i] - indeg[i])
                == -1) incnt++;
        }
        if((incnt == 1 && outcnt ==
            1)) return {node, 1};
        return {-1, 0};
    }
    else {
        bool alleven = 1;
        for(int i = 1; i <= n; i++) {
            if(indeg[i] & 1) {
                alleven = 0;
                break;
            }
        }
        if(alleven) return {-1, 1};
            // any can be root

        int odd = 0, node = -1;
        for(int i = 1; i <= n; i++) {
            if(indeg[i] & 1) {
                odd++;
                node = i;
            }
        }
        if(odd == 2) return {node,
            1};
        return {-1, 0};
    }
}
void solve() {
    auto [root, flag] = find_start();
}
};
```

7.6 floyed warshall with negative cycle [29 lines] - 6e9cf92edd

```
vector<vector<ll>> dis(n, vector<ll>(n,
    inf));
for (int i = 0; i < n; i++) dis[i][i] =
    0;
for(int i = 1; i <= m; i++) {
    int u, v, w; cin >> u >> v >> w;
    dis[u][v] = min(dis[u][v], 1LL * w);
}
for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dis[i][k] < inf &&
                dis[k][j] < inf) {
                dis[i][j] =
                    min(dis[i][j],
                    dis[i][k] +
                    dis[k][j]);
            }
        }
    }
}
/* if (d[i][k] + d[k][j] < d[i][j] - EPS)
      d[i][j] = d[i][k] + d[k][j]; */
for (int k = 0; k < n; k++) {
    if(dis[k][k] >= 0) continue;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dis[i][k] < inf &&
                dis[k][j] < inf) {
                dis[i][j] = -inf; //
                    negative cycle found
            }
        }
    }
}
// dis[u][v] == inf -> not path, -inf ->
    negative cycle
```

7.7 khun algo [31 lines] - 9cf33812ac

```
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;

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

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

    mt.assign(k, -1);
    for (int v = 0; v < n; ++v) {
        used.assign(n, false);
        try_kuhn(v);
    }

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

7.8 kth sortest path [49 lines] - 6449754e4d

```
Time Complexity = O(m * K * log(n * K))

int m, n, deg[MM], source, sink, K;
int val[MM][12]; // val[u][k] = k-th
    shortest path distance to node u
    (0-based k)

// Edge representation: destination node
    and weight
struct edge {
    int v, w;
} adj[MM][500]; // Adjacency list as a 2D
    array: adj[u][i] = i-th neighbor of u

// Info stored in priority queue for
    Dijkstra
struct info {
    int v, w, k; // v = current node, w =
        total weight, k = current path
        rank
    bool operator<(const info &b) const {
        return w > b.w; // Min-heap by
            weight
    }
};

priority_queue<info, vector<info>> Q;

void kthBestShortestPath() {
    // Step 1: Initialize all distances
        to INF
    for (int i = 0; i < n; i++)
        for (int j = 0; j < K; j++)
            val[i][j] = inf;
    // Step 2: Start from source
    Q.push({source, 0, 0});
    val[source][0] = 0;

    // Step 3: Modified Dijkstra
    while (!Q.empty()) {
        info u = Q.top(); Q.pop();

        // For each neighbor of u
        for (int i = 0; i < deg[u.v];
            i++) {
            int to = adj[u.v][i].v;
            int cost = adj[u.v][i].w +
                u.w;

            // Try to insert this new
                cost into
                val[to][0...K-1]
            for (int k = 0; k < K; k++) {
                if (cost < val[to][k]) {
                    swap(cost,
                        val[to][k]); //
                        Insert and shift
                        worse value
                        forward
                    Q.push({to,
                        val[to][k], k});
                }
            }
        }
    }
}
```

7.9 lca [74 lines] - 41fcde7445

```
class LCA {
public:
    int n;
    vector<vector<int>> g, par;
    vector<int> dep, sz;
    int lg;
    LCA(){}
    LCA(int n) {
        this -> n = n;
        g.assign(n + 1, vector<int>());
        lg = log2(n) + 1;
        par.assign(n + 1, vector<int>(lg
            + 1, 0));
        dep.assign(n + 1, 0);
        sz.assign(n + 1, 0);
        take_input();
    }
    void take_input() {
        for(int i = 1; i < n; i++) {
            int u, v;
            cin >> u >> v;
            u++, v++;
            g[u].push_back(v);
            g[v].push_back(u);
        }
    }
    void dfs(int u = 1, int p = 0) {
        par[u][0] = p;
        dep[u] = dep[p] + 1;
        sz[u] = 1;
```

```
        for(int i = 1; i <= lg; i++) {
            par[u][i] = par[par[u][i -
                1]][i - 1];
        }
        for(auto v : g[u]) {
            if(v != p) {
                dfs(v, u);
                sz[u] += sz[v];
            }
        }
    }
    int lca(int u, int v) {
        if(dep[u] < dep[v]) swap(u, v);
            // make sure that dep[u] >
            dep[v]
        for(int k = lg; k >= 0; k--) {
            if(dep[par[u][k]] >= dep[v])
                u = par[u][k];
        }
        if(u == v) return v;
        for(int k = lg; k >= 0; k--) {
            if(par[u][k] != par[v][k]) {
                u = par[u][k];
                v = par[v][k];
            }
        }
        return par[u][0];
    }
    int kth(int u, int k) {
        for(int i = 0; i <= lg; i++) {
            if(k & (1 << i)) {
                u = par[u][i];
            }
        }
        return u;
    }
    int dist(int u, int v) {
        int l = lca(u, v);
        return dep[u] + dep[v] - (2 *
            dep[l]);
    }
    // kth node from u to v, 0th node is
        u
    // k < dist(u, v)
    int go(int u, int v, int k) {
        int l = lca(u, v);
        if(dep[l] + k <= dep[u]) return
            kth(u, k);
        k -= dep[u] - dep[l];
        return kth(v, dep[v] - dep[l] -
            k);
    }
} lca_ob;
```

7.10 reachability tree [92 lines] - db920f5986

```
/*
- Find the minimal/maximal weight of the
    edges
when traversing from vertex u to vertex v
- merge the nodes with new nodes using
    DSU
- max/min weight will be the root of the
    final tree
*/
struct Reachability_tree {
    vector<vector<int>> g, par;
    vector<int> dep, dsupar, cost,
        subtree;
    vector<ll> tot;
    int id, n, m, lg, timer;
    Reachability_tree(int n, int m) :
        id(n), n(n), m(m), g(n + m + 1),
            dsupar(n + m + 1), dep(n + m
                + 1), cost(n + m + 1),
            tot(n + m + 1), subtree(n + m
                + 1) {
        iota(dsupar.begin(),
            dsupar.end(), 0);
        lg = log2(n + m + 1) + 1;
        par.assign(n + m + 1,
            vector<int>(lg + 1));
        timer = 0;
        build();
    }
    int find(int v) {
        if (dsupar[v] == v) return v;
        return dsupar[v] =
            find(dsupar[v]);
    }
    void join(int u, int v, int w) {
        u = find(u);
        v = find(v);
        if(u == v) return;
        int new_node = ++id;
        g[new_node].push_back(u);
        g[u].push_back(new_node);
        g[new_node].push_back(v);
        g[v].push_back(new_node);
        cost[new_node] = max(cost[u],
            cost[v]);
        dsupar[u] = new_node;
        dsupar[v] = new_node;
    }
    void dfs(int u, int p) {
        par[u][0] = p;
        dep[u] = dep[p] + 1;
        subtree[u] = (u <= n);
        for(int i = 1; i <= lg; i++) {
            par[u][i] = par[par[u][i -
                1]][i - 1];
        }
        for(auto v : g[u]) if(v != p) {
            dfs(v, u);
            subtree[u] += subtree[v];
        }
    }
    int lca(int u, int v) {
        if(dep[u] < dep[v]) swap(u, v);
            // make sure that dep[u] >
            dep[v]
        for(int k = lg; k >= 0; k--) {
            if(dep[par[u][k]] >= dep[v])
                u = par[u][k];
        }
        if(u == v) return v;
        for(int k = lg; k >= 0; k--) {
            if(par[u][k] != par[v][k]) {
                u = par[u][k];
                v = par[v][k];
            }
        }
        return par[u][0];
    }
    void f(int u, int p) {
        for(auto v : g[u]) if(v != p) {
            tot[v] = tot[u] + 1LL *
                (subtree[u] - subtree[v])
                * cost[u];
            f(v, u);
        }
    }
    void build() {
        for(int i = 1; i <= n; i++) {
            cin >> cost[i];
        }
        vector<array<int, 3>> edges;
        for(int i = 1; i <= m; i++) {
            int u, v;
            cin >> u >> v;

            edges.push_back({max(cost[u],
                cost[v]), u, v});
        }
        sort(edges.begin(), edges.end());
        for(auto [w, u, v] : edges) {
            if(find(u) != find(v)) {
                join(u, v, w);
            }
        }
        dfs(id, 0);
        f(id, 0);
        for (int i = 1; i <= n; i++) {
            tot[i] += cost[i];
        }
    }
};
```

7.11 scc [44 lines] - 928d336921

```
/*components: number of SCC.
 s *z: size of each SCC.
 comp: component number of each node.
 Create reverse graph.
 Run find_scc() to find SCC.
 Might need to create condensation graph
     by create_condensed().
 Think about indeg/outdeg
 for multiple test cases- clear adj/radj
     /comp/vis/sz/topo/condensed.*/
vector<int>adj[mx], radj[mx];

int comp[mx], vis[mx], sz[mx],
    components;
vector<int>topo;
void dfs(int u) {
    vis[u] = 1;
    for (int v : adj[u])
        if (!vis[v]) dfs(v);
        topo.push_back(u);
}
void dfs2(int u, int val) {
    comp[u] = val;
    sz[val]++;
    for (int v : radj[u])
        if (comp[v] == -1)
            dfs2(v, val);
}
void find_scc(int n) {
    memset(vis, 0, sizeof vis);
    memset(comp, -1, sizeof comp);
    for (int i = 1;i <= n;i++)
        if (!vis[i])
            dfs(i);
    reverse(topo.begin(), topo.end());
    for (int u : topo)
        if (comp[u] == -1)
            dfs2(u, ++components);
}
vector<int>condensed[mx];
void create_condensed(int n) {
    for (int i = 1;i <= n;i++)
        for (int v : adj[i])
            if (comp[i] != comp[v])
                condensed[comp[i]]
                .push_back(comp[v]);
}
```

8 number theory

8.1 linear sieve [16 lines] - 5bf8f49c51

```
vector<int> pri;
vector<int> lp;   // lowest prime factor
void sieve(int n) {
    lp.assign(n + 1, 0);
    pri.clear();
    for (int i = 2; i <= n; i++) {
        if (lp[i] == 0) {
            lp[i] = i;
            pri.push_back(i);
        }
        for (int p : pri) {
            if (p > lp[i] || 1LL * p * i
                > n) break;
            lp[p * i] = p;
        }
    }
}
```

8.2 mobious [9 lines] - bc823ec9cd

```
//mobius O(nlogn)
int mob[N];
void mobius()
{
    for(int i=0;i<N;i++)mob[i]=0;
    mob[1]=1;
    for(int i=1;i<N;i++)
        for(int j=i+i;j<N;j+=i)mob[j]-=
            mob[i];
}
```

8.3 pollard rho [94 lines] - 3bab19be72

```
namespace PollardRho {
mt19937
rnd(chrono::steady_clock::
now().time_since_epoch().count());
  const int P = 1e6 + 9;
  ll seq[P];
  int primes[P], spf[P];
  inline ll add_mod(ll x, ll y, ll m) {
    return (x += y) < m ? x : x - m;
  }
  inline ll mul_mod(ll x, ll y, ll m) {
    ll res = __int128(x) * y % m;
    return res;
```

```
  // ll res = x * y - (ll)((long
      double)x * y / m + 0.5) * m;
  // return res < 0 ? res + m : res;
}
inline ll pow_mod(ll x, ll n, ll m) {
  ll res = 1 % m;
  for (; n; n >>= 1) {
    if (n & 1) res = mul_mod(res, x,
        m);
    x = mul_mod(x, x, m);
  }
  return res;
}
// O(it * (logn)^3), it = number of
    rounds performed
inline bool miller_rabin(ll n) {
  if (n <= 2 || (n & 1 ^ 1)) return (n
      == 2);
  if (n < P) return spf[n] == n;
  ll c, d, s = 0, r = n - 1;
  for (; !(r & 1); r >>= 1, s++) {}
  // each iteration is a round
  for (int i = 0; primes[i] < n &&
      primes[i] < 32; i++) {
    c = pow_mod(primes[i], r, n);
    for (int j = 0; j < s; j++) {
      d = mul_mod(c, c, n);
      if (d == 1 && c != 1 && c != (n
          - 1)) return false;
      c = d;
    }
    if (c != 1) return false;
  }
  return true;
}
void init() {
  int cnt = 0;
  for (int i = 2; i < P; i++) {
    if (!spf[i]) primes[cnt++] = spf[i]
        = i;
    for (int j = 0, k; (k = i *
        primes[j]) < P; j++) {
      spf[k] = primes[j];
      if (spf[i] == spf[k]) break;
    }
  }
}
// returns O(n^(1/4))
ll pollard_rho(ll n) {
  while (1) {
    ll x = rnd() % n, y = x, c = rnd()
        % n, u = 1, v, t = 0;
    ll *px = seq, *py = seq;
    while (1) {
      *py++ = y = add_mod(mul_mod(y,
          y, n), c, n);
      *py++ = y = add_mod(mul_mod(y,
          y, n), c, n);
      if ((x = *px++) == y) break;
      v = u;
      u = mul_mod(u, abs(y - x), n);
      if (!u) return __gcd(v, n);
      if (++t == 32) {
        t = 0;
        if ((u = __gcd(u, n)) > 1 && u
            < n) return u;
      }
      if (t && (u = __gcd(u, n)) > 1 && u
          < n) return u;
    }
  }
  vector<ll> factorize(ll n) {
    if (n == 1) return vector <ll>();
    if (miller_rabin(n)) return
        vector<ll> {n};
    vector <ll> v, w;
    while (n > 1 && n < P) {
      v.push_back(spf[n]);
      n /= spf[n];
    }
    if (n >= P) {
      ll x = pollard_rho(n);
      v = factorize(x);
      w = factorize(n / x);
      v.insert(v.end(), w.begin(),
          w.end());
    }
    return v;
  }
}
void solve() {
    PollardRho::init();
    ll p; cin >> p;
    auto get1 = PollardRho::factorize(p);
}
```

8.4 segmented sieve [23 lines] - 2314fd45fd

```
vector<ll> segmented_sieve(ll l, ll r) {
    vector<ll> segpr;
    vector<bool> pr(r - l + 5, 1);
    if (l == 1) {
        pr[0] = false;
    }
    for (ll i = 0; svp[i] * svp[i] <= r;
        i++) {
        ll cur = svp[i];
        ll base = cur * cur;
        if (base < l) {
            base = ((l + cur - 1) / cur)
                * cur;
        }
        for (ll j = base; j <= r; j +=
            cur) {
            pr[j - l] = false;
        }
    }
    for (ll i = 0; i <= r - l; i++) {
        if (pr[i]) {
            segpr.push_back(l + i);
        }
    }
    return segpr;
}
```

8.5 totient phi [28 lines] - c41619b4ad

```
//all of (1-10^6) -> O(nlogn)
int phi[N];
void totient() {
    for (int i = 0; i < N; i++) phi[i] =
        i;
    for (int i = 2; i < N; i++) {
        if (phi[i] != i) continue;
        for (int j = i; j < N; j += i)
            phi[j] -= phi[j] / i;
    }
}
//10^16 range->O(sqrt(n))
int phiValue(int n)
{
    int ans=1;
    int q=sqrt(n);
    for(int i=2;i<=q;i++)
    {
        if(n%i==0)
        {
            int tem=1;
            while(n%i==0) tem*=i, n/=i;
            ans=ans*tem/i*(i-1);
            q=sqrt(n);
        }
    }
    if(n>1)ans=ans*(n-1);
    return ans;
}
```

9 math

9.1 FFT [59 lines] - db73a70650

```
//Multiply returns ans[k]=∑ ai*bj % MOD
    [such that i+j==k]
const double PI = acos(-1);
struct cd {
    double x, y;
    cd(double x = 0, double y = 0):
        x(x), y(y) {}
    cd operator+(cd o) const { return
        cd(x + o.x, y + o.y); }
    cd operator-(cd o) const { return
        cd(x - o.x, y - o.y); }
    cd operator*(cd o) const { return
        cd(x*o.x - y*o.y, x*o.y +
        y*o.x); }
    cd operator*(double d) const { return
        cd(x * d, y * d); }
    cd conj() const { return cd(x, -y); }
};
void fft(vector<cd> &a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; ++i) {
        int bit = n >> 1;
        while (j & bit) {
            j ^= bit;
            bit >>= 1;
        }
        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];
                cd v = a[i + j + len/2]
                    * w;
                a[i + j] = u + v;
                a[i + j + len/2] = u - v;
                w = w * wlen;
            }
        }
    }
    if (invert) {
        for (cd &x : a)
            x = x * (1.0 / n);
    }
}
vector<long long> multiply(const
    vector<int> &a, const vector<int> &b)
    {
    vector<cd> fa(a.begin(), a.end()),
        fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);
    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; ++i)
        fa[i] = fa[i] * fb[i];
    fft(fa, true);
    vector<long long> result(n);
    for (int i = 0; i < n; ++i)
        result[i] = round(fa[i].x);
    return result;
}
```

9.2 NTT [55 lines] - 80b38224a2

```
//Multiply returns ans[k]=∑ ai*bj % MOD
    [such that i+j==k]
//Frequency arrays are often used
const int MOD=998244353, ROOT=3;
int lim, inv_lim;
vector<int> rev, wn, w;
int modexp(int x,int e){
  int r=1;
  while(e){
    if(e&1) r=(long long)r*x%MOD;
    x=(long long)x*x%MOD;
    e>>=1;
  }
  return r;
}
void precompute(int n){
  lim=1; int L=0;
  while(lim<n) lim<<=1, ++L;
  rev.assign(lim,0);
  for(int i=0;i<lim;i++)
    rev[i]=(rev[i>>1]>>1)|((i&1)<<(L-1));
  wn.assign(lim,1);
  int g=modexp(ROOT,(MOD-1)/lim);
  for(int i=1;i<lim;i++) wn[i]=(long
      long)wn[i-1]*g%MOD;
  inv_lim=modexp(lim,MOD-2);
  w.resize(lim);
}
```

```
void ntt(vector<int>&a,bool invert){
  for(int i=0;i<lim;i++) if(i<rev[i])
      swap(a[i],a[rev[i]]);
  for(int len=1;len<lim;len<<=1){
    int step=lim/(len<<1);
    for(int i=0;i<len;i++)
        w[i]=wn[i*step];
    for(int i=0;i<lim;i+=len<<1)
      for(int j=0;j<len;j++){
        int u=a[i+j];
        int v=(long
            long)a[i+j+len]*w[j]%MOD;
        a[i+j]=u+v<MOD?u+v:u+v-MOD;
        a[i+j+len]=u-v>=0?u-v:u-v+MOD;
      }
  }
  if(invert){
    reverse(a.begin()+1,a.begin()+lim);
    for(int&i:a) i=(long
        long)i*inv_lim%MOD;
  }
}
vector<int>
    multiply(vector<int>a,vector<int>b){
  if(a.empty()||b.empty()) return {};
  int need=a.size()+b.size()-1;
  precompute(need);
  a.resize(lim); b.resize(lim);
  ntt(a,false); ntt(b,false);
  for(int i=0;i<lim;i++) a[i]=(long
      long)a[i]*b[i]%MOD;
  ntt(a,true);
  a.resize(need);
  return a;
}
```

9.3 No of Digits in n! in base B [14 lines] - 94373fb11a

```
ll NoOfDigitInNFactInBaseB(ll N, ll B) {
    ll i;
    double ans = 0;

    // Sum of logarithms: log(N!) =
        log(1) + log(2) + ... + log(N)
    for (i = 1; i <= N; i++) ans +=
        log(i);

    // Convert log from base e to base B,
        then add 1 to get digit count
    ans = ans / log(B);
    ans = ans + 1;

    // Return the result as an integer
        (floor value)
    return (ll)ans;
}
```

9.4 Xor basis [35 lines] - ad03dee2a0

```
const int N = 1505;
bitset<N> bit[N];        // input vectors
    (1-indexed)
int basis[N];            // basis[i] =
    vector responsible for bit i
vector<int> g[N];        // for
    reconstruction: which vectors XORed
    to form current
int pivot[N];            // pivot[j] =
    leading bit index of row j

void build_linear_basis(int n, int
    maxBit) {
    fill(pivot, pivot + n + 1, -1);
    fill(basis, basis + maxBit + 1, -1);

    for(int i = maxBit; i >= 0; i--) {
        int pivotRow = -1;
        for(int j = 1; j <= n; j++) {
            if(pivot[j] == -1) continue;
            if(pivotRow != -1 &&
                bit[j][i]) {
                bit[j] ^= bit[pivotRow];
                    // eliminate bit i

                g[j].push_back(pivotRow);
                    // record operation
            }
            else if(bit[j][i]) {
                pivotRow = j;
                    // new pivot
                pivot[j] = i;
                basis[i] = j;
            }
        }
    }
}

// Example: maximum XOR of any subset
int maxXOR(int maxBit) {
    bitset<N> res;
    for(int i = maxBit; i >= 0; i--) {
        if(basis[i] != -1 && !res[i]) res
            ^= bit[basis[i]];
    }
    return (int)(res.to_ullong()); //
        works if N <= 64, else handle
        manually
}
```

9.5 inverse and mul mat [94 lines] - 7ddc88f99f

```
struct Matrix {
    vector<vector<double>> a;
    int n, m;

    Matrix(int r = 0, int c = 0) : n(r),
        m(c), a(r, vector<double>(c)) {}

    static Matrix identity(int size) {
        Matrix I(size, size);
        for (int i = 0; i < size; ++i)
            I.a[i][i] = 1;
        return I;
    }

    Matrix operator*(const Matrix &B)
        const {
        assert(m == B.n);
        Matrix res(n, B.m);
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < B.m; ++j)
                for (int k = 0; k < m;
                    ++k)
                    res.a[i][j] +=
                        a[i][k] *
                        B.a[k][j];
        return res;
    }

    Matrix pow(ll exp) const {
        assert(n == m);
        Matrix base = *this, res =
            identity(n);
        while (exp) {
            if (exp & 1) res = res *
                base;
            base = base * base;
            exp >>= 1;
        }
        return res;
    }

    Matrix transpose() const {
        Matrix res(m, n);
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < m; ++j)
                res.a[j][i] = a[i][j];
        return res;
    }

    double determinant() const {
        assert(n == m);
        Matrix tmp = *this;
        double det = 1;
        for (int i = 0; i < n; ++i) {
            int pivot = i;
            for (int j = i + 1; j < n;
                ++j)
                if (fabs(tmp.a[j][i]) >
                    fabs(tmp.a[pivot][i]))
                    pivot = j;
            if (fabs(tmp.a[pivot][i]) <
                1e-9) return 0;
            if (i != pivot)
                swap(tmp.a[i],
                tmp.a[pivot]), det *=
                -1;
            det *= tmp.a[i][i];
            for (int j = i + 1; j < n;
                ++j) {
                double f = tmp.a[j][i] /
                    tmp.a[i][i];
                for (int k = i; k < n;
                    ++k)
                    tmp.a[j][k] -= f *
                        tmp.a[i][k];
            }
        }
        return det;
    }

    bool inverse(Matrix &inv) const {
        if (n != m) return false;
        inv = identity(n);
        Matrix tmp = *this;

        for (int i = 0; i < n; ++i) {
            int pivot = i;
            for (int j = i + 1; j < n;
                ++j)
                if (fabs(tmp.a[j][i]) >
                    fabs(tmp.a[pivot][i]))
                    pivot = j;
            if (fabs(tmp.a[pivot][i]) <
                1e-9) return false;
            swap(tmp.a[i], tmp.a[pivot]);
            swap(inv.a[i], inv.a[pivot]);

            double f = tmp.a[i][i];
            for (int j = 0; j < n; ++j) {
                tmp.a[i][j] /= f;
                inv.a[i][j] /= f;
            }

            for (int j = 0; j < n; ++j) {
                if (i == j) continue;
                double fac = tmp.a[j][i];
                for (int k = 0; k < n;
                    ++k) {
                    tmp.a[j][k] -= fac *
                        tmp.a[i][k];
                    inv.a[j][k] -= fac *
                        inv.a[i][k];
                }
            }
        }
        return true;
    }
};
```

9.6 lenear diophentine eqn [73 lines] - e732e8bcad

```
ll extended_euclid(ll a, ll b, ll &x, ll
    &y) {
    ll xx = y = 0, yy = x = 1;
    while (b) {
        ll q = a / b;
        ll t = b; b = a % b; a = t;
        t = xx; xx = x - q * xx; x = t;
        t = yy; yy = y - q * yy; y = t;
    }
    return a;
}

// Solves a*x + b*y = c. Finds any
    solution (x0, y0)
bool find_any_solution(ll a, ll b, ll c,
    ll &x0, ll &y0, ll &g) {
    if (a == 0 && b == 0) {
        if (c) return false;
        x0 = y0 = g = 0;
        return true;
    }
    g = extended_euclid(abs(a), abs(b),
        x0, y0);
    if (c % g != 0) return false;
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 *= -1;
    if (b < 0) y0 *= -1;
    return true;
}
```

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

// Counts the number of solutions to a*x
    + b*y = c with x in [minx, maxx] and
    y in [miny, maxy]
ll find_all_solutions(ll a, ll b, ll c,
    ll minx, ll maxx, ll miny, ll maxy) {
    ll x, y, g;
    if (!find_any_solution(a, b, c, x,
        y, g)) return 0;

    if (a == 0 && b == 0) {
        assert(c == 0);
        return 1LL * (maxx - minx + 1) *
            (maxy - miny + 1);
    }
    if (a == 0) return (maxx - minx + 1)
        * (miny <= c / b && c / b <=
        maxy);
    if (b == 0) return (maxy - miny + 1)
        * (minx <= c / a && c / a <=
        maxx);

    a /= g, b /= g;
    ll sign_a = a > 0 ? +1 : -1;
    ll sign_b = b > 0 ? +1 : -1;

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

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

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

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

    if (lx2 > rx2) swap(lx2, rx2);
    ll lx = max(lx1, lx2);
    ll rx = min(rx1, rx2);
    if (lx > rx) return 0;

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

10 misc

10.1 2d pref sum [26 lines] - 0d9b46b715

```
struct PrefixSum2D {
    vector<vector<int>> prefix;
    int n, m;
    PrefixSum2D(const
        vector<vector<int>>& grid) {
        n = grid.size();
        m = grid[0].size();
        prefix.assign(n, vector<int>(m,
            0));

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                prefix[i][j] =
                    grid[i][j];

                if (i > 0) prefix[i][j]
                    += prefix[i - 1][j];
                    // Top
                if (j > 0) prefix[i][j]
                    += prefix[i][j - 1];
                    // Left
                if (i > 0 && j > 0)
                    prefix[i][j] -=
                    prefix[i - 1][j -
                    1]; // Top-left
                    overlap
            }
        }
    }
    int getSum(int x1, int y1, int x2,
        int y2) {
        int sum = prefix[x2][y2];
        if (x1 > 0) sum -= prefix[x1 -
            1][y2];
        if (y1 > 0) sum -= prefix[x2][y1
            - 1];
        if (x1 > 0 && y1 > 0) sum +=
            prefix[x1 - 1][y1 - 1];
        return sum;
    }
};
```

10.2 bit hacks [17 lines] - 13d6b87d3b

```
#define ckbit(n, k) (((n)>>(k)) & 1)
#define toggle(n, k) ((n) ^= (1LL << (k)))
#define setbit(n, k) ((n) |= (1LL << (k)))
#define unsetbit(n, k) ((n) &= ~(1LL <<
    (k)))
#define lowbit(n) ((n) & -(n))
#define highbit(n) (63 -
    __builtin_clzll(n)) // = floor
    log2(n)
// a|b = a^b + a&b
// a ^ (a&b) = b ^ (a|b)
// (a&b) ^ (a|b) = a^b

// a+b = a|b + a&b
// a+b = a^b + 2(a&b)

// a-b = (a^(a&b))-((a|b)^a)
// a-b = ((a|b)^b)-((a|b)^a)
// a-b = (a^(a&b))-(b^(a&b))
// a-b = ((a|b)^b)-(b^(a&b))
```

10.3 job with 2 deadline [62 lines] - 546d97e7c2

```
/*
given n workers for a company
for each worker given 3 parameter
k, l, r
we need to assign each worker for a day i
    for all n days
and no two worker can be assigned on same
    date
if workder i is assigned on jth day and j
    <= k then the profit 'l' otherwise
    'r'
assign such that maximize profit
*/

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;
const ld inf = 1e100;
const ld eps = 1e-18;

void solve() {
    int n;
    cin >> n;
    vector<vector<int>> front(n + 1),
        back(n + 1);
    ll tot = 0;
    for(int i = 1; i <= n; i++) {
        int k, l, r;
        cin >> k >> l >> r;
        tot += min(l, r);
        if(l > r) {
            front[k].push_back(l - r);
        }
        else if(l < r && (n - k) > 0) {
            back[n - k].push_back(r - l);
        }
    }
    auto add = [&](vector<vector<int>>
        &vec) {
        priority_queue<int> pq;
        for(int i = 1; i <= n; i++) {
            for(auto g : vec[i]) {
                pq.push(-g);
                if(pq.size() > i)
                    pq.pop();
            }
        }
        ll res = 0;
        while(not pq.empty()) {
            res += -pq.top(); pq.pop();
        }
        return res;
    };

    cout << tot + add(front) + add(back)
        << "\n";
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    int tc = 1, cs = 1;
    cin >> tc;
    while (tc--) {
        //cout << "Case " << cs++ << ": ";
        solve();
    }
    return 0;
}
```

10.4 pairs rearrange made palindrome [24 lines] - 449d38aff6

```
// palindrome count
// odd <= 1
// so Xor = 0 means odd = 0
// Xor = 1 means odd = 1 for this
// i need to fllip exactly one bit of a-z
map<int, ll> freq;
int mask = 0;
// before processing any characters, we
    have one occurrence of mask 0
freq[mask] = 1;
ll result = 0;
// iterate over each character in the
    string
for (char c : s) {
    // Toggle the bit corresponding to
        the current character
    mask ^= (1 << (c - 'a'));
    // count substrings ending here that
        are already balanced (exact
        match)
    result += freq[mask];
    // count substrings ending here that
        differ by exactly one bit
    for (int b = 0; b < 26; ++b) {
        result += freq[mask ^ (1 << b)];
    }
    // update frequency for the current
        mask
    freq[mask]++;
}
cout << result << "\n";
```

10.5 run [16 lines] - be27096b28

```
# Open & set C++ build system in Sublime
    (Linux)

1) Open Sublime Text
2) Menu → Tools → Build System → New
    Build System
3) Paste this code:

{
  "cmd": ["bash", "-c", "g++ \"$file\"
      -std=gnu++17 -O2 -o a.out &&
      ./a.out < input.txt"],
  "selector": "source.c, source.c++",
  "working_dir": "$file_path"
}

4) Ctrl + S → save as: cpp.sublime-build
5) Menu → Tools → Build System → cpp
6) Open main.cpp → Ctrl + B
# input from input.txt, output in
    terminal
```

10.6 sliding windows min-max using deque [13 lines] - 52c44b3137

```
deque<int> dq;
for(int i = 1; i <= n; i++) {
    while(!dq.empty() && v[dq.back()] <
        v[i]) {
        dq.pop_back();
    }
    dq.push_back(i);
    if(dq.front() <= i - k) {
        dq.pop_front();
    }
    if(i >= k) {
        cout << v[dq.front()] << " ";
    }
}
```

10.7 unique OR all subarray [22 lines] - b574681b4e

```
int ans = 0;
map<int, bool> has;
for(int i = 1; i <= n; i++) {
    int cur = 0, next = i, cnt = 0;
    while(next <= n) {
        cur |= tree.query(1, 1, n, i,
            next).val;
        if(!has.count(cur)) cnt++;
        has[cur] = 1;
        int mn = n + 2;
        for(int j = 0; j < 30; j++) {
            if(!(cur & (1 << j))) {
                if(bitv[j].size() > 0) {
                    auto it =
                        lower_bound(bitv[j].begin(),
                         bitv[j].end(),
                        i);
                    if(it !=
                        bitv[j].end()) {
                        mn = min(mn,
                            *it);
                    }
                }
            }
        }
        next = mn;
    }
}
```

11 Data structure

11.1 2dbit [97 lines] - 089f12d2d7

```
struct _2dbit {
    struct BITmx {
        vector<int> v;
        int n;
        BITmx(){}
        BITmx(int n) : n(n) {
            v.resize(n + 1);
        }
        void upd(int i, int val) {
            for(; i <= n; i += (i & -i))
                {
                v[i] = max(v[i], val);
            }
        }
        int query(int i) {
            int ans = 0;
            for(; i > 0; i -= (i & -i)) {
                ans = max(ans, v[i]);
            }
            return ans;
        }
    };
    vector<vector<int>> cords;
    vector<BITmx> cnt;
    int n;
    _2dbit(int n) : n(n) {
        cnt.resize(n + 1);
        cords.resize(n + 1);
    }
    void add(int i, int val) {
        for(; i <= n; i += (i & -i)) {
            cords[i].push_back(val);
        }
    }
    void build() {
        for(int i = 1; i <= n; i++) {
            auto &v = cords[i];
            sort(v.begin(), v.end());
            v.erase(unique(v.begin(),
                v.end()), v.end());
            cnt[i] = BITmx(v.size());
        }
    }
    int getidx(int p, int val) {
        auto &v = cords[p];
        return (lower_bound(v.begin(),
            v.end(), val) - v.begin() +
            1);
    }
    void upd(int x, int y, int val) {
        for(int i = x; i <= n; i += (i &
            -i)) {
            int id = getidx(i, y);
            cnt[i].upd(id, val);
        }
    }
    int query(int x, int y) {
        int ans = 0;
        for(int i = x; i > 0; i -= (i &
            -i)) {
            int id = getidx(i, y);
            ans = max(ans,
                cnt[i].query(id - 1));
        }
        return ans;
    }
};

void solve() {
    int n;
    cin >> n;
    vector<array<int, 2>> v(n + 1);
    vector<int> allx, ally;
    for(int i = 1; i <= n; i++) {
        cin >> v[i][0] >> v[i][1];
        allx.push_back(v[i][0]);
        ally.push_back(v[i][1]);
    }
    sort(allx.begin(), allx.end());
    allx.erase(unique(allx.begin(),
        allx.end()), allx.end());
    auto getidxX = [&](int x) {
        return
            (lower_bound(allx.begin(),
            allx.end(), x) - allx.begin()
            + 1);
    };
    sort(ally.begin(), ally.end());
    ally.erase(unique(ally.begin(),
        ally.end()), ally.end());
    auto getidxY = [&](int x) {
        return
            (lower_bound(ally.begin(),
            ally.end(), x) - ally.begin()
            + 1);
    };
    int m = allx.size();
    _2dbit bit(m);
    for(int i = 1; i <= n; i++){
        int x = getidxX(v[i][0]), y =
            getidxY(v[i][1]);
        bit.add(x, y);
    }
    bit.build();
    int ans = 0;
    for(int i = 1; i <= n; i++) {
        int x = getidxX(v[i][0]), y =
            getidxY(v[i][1]);
        int val = 1 + bit.query(x - 1,
            y);
        ans = max(ans, val);
        bit.upd(x, y, val);
    }
    cout << ans << "\n";
}
```

11.2 bit [24 lines] - a82760cfe8

```
template <class T>
struct BIT { //1-indexed
  int n; vector<T> t;
  BIT() {}
  BIT(int _n) {
    n = _n; t.assign(n + 1, 0);
  }
  T prefixSum(int i) {
    T ans = 0;
    for (; i >= 1; i -= (i & -i)) ans +=
        t[i];
    return ans;
  }
  void upd(int i, T val) {
    if (i <= 0) return;
    for (; i <= n; i += (i & -i)) t[i]
        += val;
  }
};
// inversion
BIT<ll> bit(n);
for(int i = 1; i <= n; i++) {
    int x; cin >> x;
    ans[i] = bit.query(n) - bit.query(x
        - 1);
    bit.upd(x, 1);
}
```

11.3 dsu with rollback [33 lines] - aa5a0e5006

```
struct DSU
{
    vector<int> parent, size;
    vector<pair<int, int>> history;
    int componentCount;
    DSU(int n) : parent(n + 1), size(n +
        1, 1),
        componentCount(n){iota(parent.begin(),
         parent.end(), 0);}
    int find(int v)
    {
        if (parent[v] != v)return
            find(parent[v]);
        return parent[v];
    }
    bool merge(int u, int v)
    {
        u = find(u),v = find(v);
        if (u == v){return false;}
        if (size[u] < size[v])swap(u, v);
        history.push_back({v, size[v]});
        history.push_back({u, size[u]});
        parent[v] = u,size[u] +=
            size[v],--componentCount;
        return true;
    }
    void rollback()
    {
        auto [u, oldSizeU] =
            history.back();
        history.pop_back();
        auto [v, oldSizeV] =
            history.back();
        history.pop_back();
        parent[v] = v,size[u] =
            oldSizeU,size[v] =
            oldSizeV,componentCount++;
    }
    int getComponentCount(){return
        componentCount;}
    bool same(int u, int v){return
        find(u) == find(v);}
    int getSize(int v){return
        size[find(v)];}
};
```

11.4 dsu [97 lines] - 4c9e4f01e0

```
/*
sometimes i need to modify the find()
    function
first of all add all expreinece as
    team[u]
then when join() make self[v] = team[v]
    - team[u]
so for v the ans still team[v] = self[v]
    + team[u]
and when we find some other parents
then the added points add to all players
    of a team
so we do self[v] += self[p] so here
    self[p]
*/
struct DSU {
    vector<ll> par, rank, sz;
    int n;
    DSU(int n) : n(n), par(n + 1), rank(n
        + 1, 0), sz(n + 1, 1) {
```

```
        iota(par.begin(), par.end(), 0);
    }
    int find(int v) {
        return (par[v] == v ? v : (par[v]
            = find(par[v])));
    }
    bool same(int u, int v) {
        return find(u) == find(v);
    }
    int get_size(int v) {
        return sz[find(v)];
    }
    int count() {
        return n;
    }
    void join(int u, int v) {
        u = find(u);
        v = find(v);
        n--;
        if (rank[v] > rank[u]) swap(u,
            v);
        par[v] = u;
        sz[u] += sz[v];
        if (rank[u] == rank[v])
            rank[u]++;
        // u is the parent;
    }
};
struct DSU {
    vector<ll> par, rank, sz, sum,
        elemwhichnode;
    int n, nextId;
    DSU(int n, int m)
        : par(n + m + 1), rank(n + m +
            1, 0), sz(n + m + 1, 1),
          sum(n + m + 1), elemwhichnode(n
              + 1), n(n), nextId(n) {
        iota(par.begin(), par.end(), 0);
        for (int i = 1; i <= n; ++i) {
            elemwhichnode[i] = i;
            sum[i] = i;
        }
    }
    void join(int u, int v) {
        u = find(u);
        v = find(v);
        if(u == v) return;
        n--;
        if (rank[v] > rank[u]) swap(u,
            v);
        par[v] = u;
        sz[u] += sz[v];
        sum[u] += sum[v];
        if (rank[u] == rank[v])
            rank[u]++;
        // u is the parent;
    }

    void erase(int v) {
        int nv = elemwhichnode[v];
        if (nv == 0) return;
            // already deleted
        int pv = find(nv);
        sz [pv] -= 1;
        sum[pv] -= v;

        elemwhichnode[v] = 0;
            // mark deleted
    }
    /* move p -> set(q) */
    void move(int u, int v) {
        if(elemwhichnode[u] == 0) return;

        int pu = find(elemwhichnode[u]);
        int pv = find(elemwhichnode[v]);
        if (pu == pv) return;

        erase(u);

        nextId++;
        par [nextId] = nextId;
        rank[nextId] = 0;
        sz  [nextId] = 1;
        sum [nextId] = u;

        elemwhichnode[u] = nextId;

        join(nextId, pv);
    }

    pair<int, ll> query(int p) {
        int r = find(elemwhichnode[p]);
        return { (int)sz[r], sum[r] };
    }
};
```

11.5 dynamic segtree [95 lines] - 651c502af4

```
class SegTlazy {
public:
    struct node {
        ll mn, mncnt, lazy;
        bool haslazy;
        node *l, *r;
        node() : mn(inf), mncnt(0),
            lazy(0), haslazy(0),
            l(nullptr), r(nullptr) {}
        node(ll mn, ll mncnt) : mn(mn),
            mncnt(mncnt), lazy(0),
            haslazy(0), l(nullptr),
            r(nullptr) {}
    };

    node *root;
    ll n, m; // max and min range
     // default 1 to max
    explicit SegTlazy(ll _n, ll _m = 1) :
        n(_n), m(_m) { root = new
        node(); }
    inline void merge(node *nd) {
        nd->mncnt = 0;
        if(nd->l && nd->r) {
            nd->mn = min(nd->l->mn,
                nd->r->mn);
            if(nd->mn == nd->l->mn)
                nd->mncnt +=
                nd->l->mncnt;
            if(nd->mn == nd->r->mn)
                nd->mncnt +=
                nd->r->mncnt;
        }
        else if(nd -> l) {
            nd->mn = nd->l->mn;
            if(nd->mn == nd->l->mn)
                nd->mncnt +=
                nd->l->mncnt;
        }
        else if(nd -> r) {
            nd->mn = nd->r->mn;
            if(nd->mn == nd->r->mn)
                nd->mncnt +=
                nd->r->mncnt;
        }
    }
    void apply(node *nd, ll b, ll e, ll
        lazy) {
        nd->mn += lazy;
    }
    inline void push(node *nd, ll b, ll
        e) {
        if (!nd->haslazy) return;
        apply(nd, b, e, nd->lazy);
        if (b != e) {
            if (!nd->l) nd->l = new
                node();
            if (!nd->r) nd->r = new
                node();
            nd->l->lazy += nd->lazy;
            nd->r->lazy += nd->lazy;
            nd->l->haslazy = 1;
            nd->r->haslazy = 1;
        }
        nd->lazy = 0;
        nd->haslazy = 0;
    }
    void build(node *nd, ll b, ll e) {
        if(b == e) {
            nd->mn = 0;
            nd->mncnt = 1;
            return;
        }
        ll mid = (b + e) >> 1;
        if (!nd->l) nd->l = new node();
        if (!nd->r) nd->r = new node();
        build(nd->l, b, mid);
        build(nd->r, mid + 1, e);
        merge(nd);
    }
    void build() { build(root, m, n); }
    void upd(node *nd, ll b, ll e, ll i,
        ll j, ll v) {
        push(nd, b, e);
        if (j < b || e < i) return;

        if (i <= b && e <= j) {
            nd->lazy += v;
            nd->haslazy = 1;
            push(nd, b, e);
            return;
        }
        ll mid = (b + e) >> 1;
        if (!nd->l) nd->l = new node();
        if (!nd->r) nd->r = new node();
        upd(nd->l, b, mid, i, j, v);
        upd(nd->r, mid + 1, e, i, j, v);
        merge(nd);
    }
    void upd(ll l, ll r, ll v) {
        upd(root, m, n, l, r, v); }
    node query(node *nd, ll b, ll e, ll
        i, ll j) {
        if (j < b || e < i) return
            node();
        if(!nd) return node();
        if (i <= b && e <= j) return *nd;
        push(nd, b, e);
        ll mid = (b + e) >> 1;
        node left  = query(nd->l, b,
            mid, i, j);
        node right = query(nd->r, mid +
            1, e, i, j);
        node res;
        res.mn = min(left.mn, right.mn);
        if(res.mn == left.mn) res.mncnt
            += left.mncnt;
        if(res.mn == right.mn) res.mncnt
            += right.mncnt;
        return res;
    }
    node query(ll l, ll r) { return
        query(root, m, n, l, r); }
};
```

11.6 gp hash table [19 lines] - 5b4a28d281

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
struct custom_hash {
  static uint64_t splitmix64(uint64_t x)
    {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) *
        0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) *
        0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM =
        chrono::steady_clock::now().time_since

    return splitmix64(x + FIXED_RANDOM);
  }
};
//pair (a, b) er jonne a * MOD + b
gp_hash_table<int, int, custom_hash> mp;
unordered_set<ll, custom_hash> mp;
mp.reserve(n);
```

11.7 hld [109 lines] - 14d49fef41

```
class HLD : public SegTlazy {
public:
    vector<vector<pair<int, int>>> g;
    vector<vector<int>> par;
    vector<int> heavy, head, pos, depth,
        val;
    int timer, n, height;

    HLD(int n) : n(n), SegTlazy(n) {
        timer = 0;
        height = 31 - __builtin_clz(n);
        heavy.assign(n + 1, 0);
        head.assign(n + 1, 0);
```

```
    pos.assign(n + 1, 0);
    par.assign(n + 1,
        vector<int>(height + 1));
    depth.assign(n + 1, 0);
    val.assign(n + 1, 0);
    g.resize(n + 1);
}

void add_edge(int u, int v, int w) {
    g[u].push_back({v, w});
    g[v].push_back({u, w});
}

int dfs(int u = 1, int p = 0) {
    int sub = 1, big = 0;
    par[u][0] = p;
    for(int j = 1; j <= height; j++)
        {
        par[u][j] = par[par[u][j -
            1]][j - 1];
    }
    for (auto [v, w] : g[u]) if(v !=
        p) {
        depth[v] = depth[u] + 1;
        val[v] = w;
        int subsize = dfs(v, u);
        if (subsize > big) big =
            subsize, heavy[u] = v;
        sub += subsize;
    }
    return sub;
}
void decompose(int u, int h) {
    head[u] = h,
    pos[u] = ++timer;
    if (heavy[u])
        decompose(heavy[u], h);
    for (auto [v, w] : g[u]) {
        if (v != par[u][0] && v !=
            heavy[u]) decompose(v,
            v);
    }
}
void makeHLD(int root = 1) {
    dfs(root);
    decompose(root, root);
    for(int i = 1; i <= n; i++) {
        SegTlazy::a[pos[i]] = val[i];
    }
    SegTlazy::build(1, 1, n);
}
// if value on edge then call isEdge
    = 1
ll Query(int u, int v, bool isEdge =
    1) {
    node ret;
    for (; head[u] != head[v]; v =
        par[head[v]][0]) {
        if (depth[head[u]] >
            depth[head[v]]) swap(u,
            v);
        node tmp =
            SegTlazy::query(1, 1, n,
            pos[head[v]], pos[v]);
        SegTlazy::merge(ret, ret,
            tmp);
        }
        if (depth[u] > depth[v]) swap(u,
            v);
        node tmp = SegTlazy::query(1, 1,
            n, pos[u] + isEdge, pos[v]);
        SegTlazy::merge(ret, ret, tmp);
        return ret.val;
    }
    void Update(int u, int v, int val,
        bool isEdge = 0) {
        for (; head[u] != head[v]; v =
            par[head[v]][0]) {
            if (depth[head[u]] >
                depth[head[v]]) swap(u,
                v);
          //  cout<<"Updating:"<<v<<'
              '<<head[v]<<' '<<val<<endl;
            SegTlazy::upd(1, 1, n,
                pos[head[v]], pos[v],
                val);
        }
        if (depth[u] > depth[v]) swap(u,
            v);
        //cout<<"Updating:"<<v<<' '<<u<<'
            '<<val<<endl;
        SegTlazy::upd(1, 1, n, pos[u] +
            isEdge, pos[v], val);
    }
};

void solve() {
    int n;
    cin>> n;
    HLD hld(n);
    // for vertex cin >> v[i]
    vector<pair<int, int>> edge;
    for(int i=1;i<n;i++){
        int u, v, w;
        cin >> u >> v >> w;
        hld.add_edge(u, v, w);
        edge.push_back({u, v});
    }
    hld.makeHLD();
    int q; cin >> q;
    while(q--){
        int ty; cin >> ty;
        if(ty == 1) {
            int id, val;
            cin >> id >> val;
            auto [u, v] = edge[id - 1];
            hld.Update(u, v, val, 1);
        }
        else {
            int u, v;
            cin >> u >> v;
            cout << hld.Query(u, v, 1) <<
                "\n";
        }
    }
}
```

11.8 mex using trie in logmax [26 lines] - 8195ee80bc

```
struct node{
    node *ch[2]; int cnt;
    node() {ch[0] = ch[1] = NULL, cnt =
        0; }
} *root;

void insert(int x) {
    node *curr = root;
    for(int i = 20; i >= 0; i--) {
        int bit = (x >> i) & 1;
        if(curr -> ch[bit] == NULL)
            curr -> ch[bit] = new
                node();
        curr -> ch[bit] -> cnt++;
        curr = curr -> ch[bit];
    }
}
int mex() {
    node *curr = root; int ret = 0;
    for(int i = 20; i >= 0; i--) {
        if(curr == NULL || curr -> ch[0]
            == NULL) return ret;
        if(curr -> ch[0] -> cnt >= (1 <<
            i)) {
            curr = curr -> ch[1];
            ret |= (1 << i);
        } else curr = curr -> ch[0];
    }
    return ret;
}
```

11.9 mo with update [76 lines] - 6f0115f479

```
const ll N = 1e6 + 9;
const ll B = 1000;

struct query {
    ll l, r, t, id;
    bool operator < (const query &x)
        const {
        if(l / B == x.l / B) {
            if(r / B == x.r / B) return t
                < x.t;
            return r / B < x.r / B;
        }
        return l / B < x.l / B;
    }
} Q[N];
struct upd {
    ll pos, old, cur;
} U[N];

ll a[N];
ll cnt[N], ans[N], l, r, t;
ll tot;
inline void add(int x) {
    ++cnt[x];
    if(cnt[x]==1)tot++;
}
inline void del(int x) {
    --cnt[x];
    if(cnt[x]==0)tot--;
}
inline void update(int pos, int x) {
    if (l <= pos && pos <= r) {
        add(x);
        del(a[pos]);
    }
    a[pos] = x;
}
map <ll, ll> mp;
ll nxt = 0;
ll get(ll x) {
    return mp.count(x) ? mp[x] : mp[x] =
        ++nxt;
}

void solve(){
    ll n, q;
    cin >> n >> q;
    for (ll i = 1; i <= n; i++) {
        cin >> a[i];
        a[i] = get(a[i]);
    }
    ll nq = 0, nu = 0;
    for (ll i = 1; i <= q; i++) {
        char ty;ll l, r;
        cin >> ty >> l >> r;
        if (ty == 'Q') {l++;++nq, Q[nq]
            = {l, r, nu, nq};}
        else{l++; ++nu, U[nu].pos = l,
            U[nu].old = a[l], a[l] =
            get(r), U[nu].cur = a[l];}
    }
    sort(Q + 1, Q + nq + 1);
    t = nu, l = 1, r = 0;
    for (ll i = 1; i <= nq; i++) {
        ll L = Q[i].l, R = Q[i].r, T =
            Q[i].t;
        while(t < T){ t++,
            update(U[t].pos, U[t].cur);}
        while(t > T) {update(U[t].pos,
            U[t].old), t--;}
        if(R < l) {
            while(l > L) add(a[--l]);
            while(l < L) del(a[l++]);
            while(r < R) add(a[++r]);
            while(r > R) del(a[r--]);
        } else {
            while(r < R) add(a[++r]);
            while(r > R) del(a[r--]);
            while(l > L) add(a[--l]);
            while(l < L) del(a[l++]);
        }
        ans[Q[i].id] = tot;
    }
    for (ll i = 1; i <= nq; i++) cout <<
        ans[i] << '\n';
}
```

11.10 pbds [23 lines] - c8bf03595b

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
template <typename T> using o_set =
    tree<T, null_type,
less_equal<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
template <typename T, typename R> using
    o_map = tree<T, R,
less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
```

```cpp
int main() {
  o_set<int>se;
  se.insert(1);
  se.insert(2);
  cout << *se.find_by_order(0) << endl;
      //k th element
  cout << se.order_of_key(2) << endl;
      //number of elements less than k
  o_map<int, int>mp;
  mp.insert({1, 10});
  mp.insert({2, 20});
  cout << mp.find_by_order(0)->second <<
      endl;
  ///k th element
  cout << mp.order_of_key(2) << endl;
  ///number of first elements less than k
}
```

11.11 persistance seg tree [68 lines] - c7856e9a5a

```cpp
class SegTlazy {
public:
    struct node {
        int cursum, premn, premx;
        node *left, *right;
        node(int _cur = 0, int mn = 1e9,
            int mx = -1e9)
            : cursum(_cur), premn(mn),
                premx(mx),
                left(nullptr),
                right(nullptr) {}
    };
    vector<int> a;
    vector<node*> roots; // persistent
        versions
    int n;
    SegTlazy() {
        n = 0;
        a.clear();
        roots.clear();
    }
    void init(int _n) {
        n = _n;
        a.assign(n + 1, 1); // 1-based
            intitially all 1
        roots.clear();
    }
    void merge(node* nd, node* l, node*
        r) {
        nd->cursum = l->cursum +
            r->cursum;
        nd->premn = min(l->premn,
            l->cursum + r->premn);
        nd->premx = max(l->premx,
            l->cursum + r->premx);
    }
    node* build(int b, int e) {
        if (b == e) {
            return new node(a[b], a[b],
                a[b]);
        }
        int mid = (b + e) >> 1;
        node* cur = new node();
        cur->left = build(b, mid);
        cur->right = build(mid + 1, e);
        merge(cur, cur->left,
            cur->right);
        return cur;
    }
    node* update(node* prev, int b, int
        e, int pos, int val) {
        if (b == e) {
            return new node(val, val,
                val);
        }
        int mid = (b + e) >> 1;
        node* cur = new node();
        if (pos <= mid) {
            cur->left =
                update(prev->left, b,
                mid, pos, val);
            cur->right = prev->right;
        } else {
            cur->left = prev->left;
            cur->right =
                update(prev->right, mid
                + 1, e, pos, val);
        }
        merge(cur, cur->left,
            cur->right);
        return cur;
    }
    node query(node* nd, int b, int e,
        int i, int j) {
        if (!nd || j < b || e < i) {
            return node(0, 1e9, -1e9);
        }
        if (i <= b && e <= j) {
            return *nd;
        }
        int mid = (b + e) >> 1;
        node l = query(nd->left, b, mid,
            i, j);
        node r = query(nd->right, mid +
            1, e, i, j);
        node res;
        merge(&res, &l, &r);
        return res;
    }
};
```

11.12 segtree [124 lines] - df44addd3e

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

struct LazySegTree {
    vector<ll> tree, lazyAdd, assignVal;
    vector<char> assignFlag; // 0 = no
        assign pending, 1 = assign
        pending
    ll n;

    LazySegTree(ll arr[], ll sz) {
        n = sz;
        tree.assign(4*n, 0);
        lazyAdd.assign(4*n, 0);
        assignVal.assign(4*n, 0);
        assignFlag.assign(4*n, 0);
        build(arr, 0, 0, n-1);
    }

    void build(ll arr[], ll ind, ll st,
        ll ed) {
        if (st == ed) { tree[ind] =
            arr[st]; return; }
        ll mid = (st + ed) / 2;
        build(arr, 2*ind+1, st, mid);
        build(arr, 2*ind+2, mid+1, ed);
        tree[ind] = tree[2*ind+1] +
            tree[2*ind+2];
    }

    // Push pending operations at node
        ind to children and apply to
        node's tree
    void push(ll ind, ll st, ll ed) {
        if (st > ed) return;
        // First handle assignment (it
            overrides additions)
        if (assignFlag[ind]) {
            tree[ind] = assignVal[ind] *
                (ed - st + 1);
            if (st != ed) {
                ll L = 2*ind+1, R =
                    2*ind+2;
                // set children to this
                    assignment, clear
                    their adds
                assignFlag[L] =
                    assignFlag[R] = 1;
                assignVal[L] =
                    assignVal[R] =
                    assignVal[ind];
                lazyAdd[L] = lazyAdd[R]
                    = 0;
            }
            assignFlag[ind] = 0; //
                consumed at this node
                (tree already updated)
            assignVal[ind] = 0;
        }
        // Then handle addition
        if (lazyAdd[ind] != 0) {
            tree[ind] += lazyAdd[ind] *
                (ed - st + 1);
            if (st != ed) {
                ll L = 2*ind+1, R =
                    2*ind+2;
                // if child has assign
                    pending, adding
                    modifies assignVal
                if (assignFlag[L])
                    assignVal[L] +=
                    lazyAdd[ind];
                else lazyAdd[L] +=
                    lazyAdd[ind];
                if (assignFlag[R])
                    assignVal[R] +=
                    lazyAdd[ind];
                else lazyAdd[R] +=
                    lazyAdd[ind];
            }
            lazyAdd[ind] = 0;
        }
    }

    // Range add: add val to every
        element in [l,r]
    void updateAdd(ll val, ll ind, ll l,
        ll r, ll st, ll ed) {
        push(ind, st, ed);
        if (r < st || ed < l) return;
        if (l <= st && ed <= r) {
            lazyAdd[ind] += val;
            push(ind, st, ed);
            return;
        }
        ll mid = (st + ed) / 2;
        updateAdd(val, 2*ind+1, l, r,
            st, mid);
        updateAdd(val, 2*ind+2, l, r,
            mid+1, ed);
        tree[ind] = tree[2*ind+1] +
            tree[2*ind+2];
    }

    // Range assign: set every element in
        [l,r] to val
    void updateAssign(ll val, ll ind, ll
        l, ll r, ll st, ll ed) {
        push(ind, st, ed);
        if (r < st || ed < l) return;
        if (l <= st && ed <= r) {
            // mark assignment on this
                node and apply
                immediately
            assignFlag[ind] = 1;
            assignVal[ind] = val;
            lazyAdd[ind] = 0; //
                assignment overrides
                previous adds
            push(ind, st, ed);
            return;
        }
        ll mid = (st + ed) / 2;
        updateAssign(val, 2*ind+1, l, r,
            st, mid);
        updateAssign(val, 2*ind+2, l, r,
            mid+1, ed);
        tree[ind] = tree[2*ind+1] +
            tree[2*ind+2];
    }

    // Range sum query [l,r]
    ll sum(ll ind, ll l, ll r, ll st, ll
        ed) {
        push(ind, st, ed);
        if (r < st || ed < l) return 0;
        if (l <= st && ed <= r) return
            tree[ind];
        ll mid = (st + ed) / 2;
        return sum(2*ind+1, l, r, st,
            mid) + sum(2*ind+2, l, r,
            mid+1, ed);
    }

    // convenient wrappers:
    void rangeAdd(ll l, ll r, ll val) {
        updateAdd(val, 0, l, r, 0, n-1);
        }
```

```
    void rangeAssign(ll l, ll r, ll val)
        { updateAssign(val, 0, l, r, 0,
        n-1); }
    ll rangeSum(ll l, ll r) { return
        sum(0, l, r, 0, n-1); }
};

// Example usage
int main() {
    ll a[] = {1,2,3,4,5};
    ll sz = 5;
    LazySegTree seg(a, sz);

    // add 10 to range [1,3]
    seg.rangeAdd(1,3,10); // array ->
        {1,12,13,14,5}

    // assign range [2,4] to 7
    seg.rangeAssign(2,4,7); // array ->
        {1,12,7,7,7}

    // add 5 to [0,2]
    seg.rangeAdd(0,2,5); // array ->
        {6,17,12,7,7}

    cout << seg.rangeSum(0,4) << "\n"; //
        prints 6+17+12+7+7 = 49
    cout << seg.rangeSum(1,2) << "\n"; //
        prints 17+12 = 29
    return 0;
}
```

11.13 sparse [34 lines] - 60d93b1d24

```
struct Spares_table { // now 1-based
    vector<int> a;
    vector<vector<int>> t;
    int n, q, lg;
    Spares_table() {}
    Spares_table(int n, int q) : n(n),
        q(q) {
        lg = __lg(max(1, n)) + 1;
        a.assign(n + 1, 0);
        t.assign(n + 2, vector<int>(lg +
            1, 0));
        take_input();
    }
    void take_input() {
        for (int i = 1; i <= n; i++) cin
            >> a[i];
        build();
    }
    void build() {
        if (n <= 1) return;
        // Build on diffs d[i] = |a[i] -
            a[i-1]| for i = 2..n
        for (int i = 2; i <= n; i++)
            t[i][0] = abs(a[i] - a[i -
            1]);
        for (int j = 1; (1 << j) <= n;
            j++) {
            for (int i = 2; i + (1 << j)
                - 1 <= n; i++) {
                t[i][j] = gcd(t[i][j -
                    1], t[i + (1 << (j -
                    1))][j - 1]);
            }
        }
    }
    // Query on the original array [l..r]
        (1-based).
    // Returns GCD of |a[i] - a[i-1]| for
        i in (l..r], and 0 if l == r.
    int query(int l, int r) {
        if (l >= r) return 0;
        int len = r - l;
        int k = __lg(len);
        return gcd(t[l + 1][k], t[r - (1
            << k) + 1][k]);
    }
};
```

11.14 trie [190 lines] - 8b1049f1f2

```
/*
if you need dfs on trie
at first make the trie an unique tree
inserting an unique id on each node
then just treat as node of a tree
*/
class Trie {
public:

    class node { // 1 based
    public:
        node *child[27];
        int leaf, sz, id;
        node() {
            for(int i = 0; i < 27; i++) {
                child[i] = nullptr;
            }
            sz = 0;
            leaf = 0;
            id = 0;
        }
    } *root;

    vector<int> dis;
    int unqid = 1;

    Trie () {
        root = new node();

    }

    void insert(string s) {
        auto cur = root;
        for(auto x : s) {
            int i = x - 'a' + 1; //
                1-based
            if(!cur->child[i]) {
                cur->child[i] = new
                    node(); // edge dilam
            }
            cur = cur->child[i];
            cur->sz++; // new edge e
                ashalam ebar increment
                korbo
            cur->id = unqid++;
        }
        cur->leaf++;
    }

    int set(node *cur) {
        dis[cur->id] = cur->leaf;
        for(int i = 1; i <= 26; i++) {
            if(cur->child[i]) {
                set(cur->child[i]);
                dis[cur->id] =
                    max(dis[cur->id],
                    dis[cur->child[i]->
                    id]);
            }
        }
        return dis[cur->id];
    }
};
Trie mytrie;
for(int i = 1; i <= n; i++) {
    string t; cin >> t;
    mytrie.insert(t);
}
mytrie.dis.assign(mytrie.unqid + 1, 0);
mytrie.set(mytrie.root);
/*
count how many subarray Xor < k
*/
struct Trie {
    const int B = 20;
    struct node { // 1 based
        node *child[2];
        int cnt;
        node() {
            child[0] = child[1] = 0;
            cnt = 0;
        }
    } *root;

    Trie () {
        root = new node();
    }

    void insert(int x) {
        auto cur = root;
        for(int i = B - 1; i >= 0; i--) {
            int id = (x >> i) & 1;
            if(!cur->child[id])
                cur->child[id] = new
                node();
            cur = cur->child[id];
            cur->cnt++;
        }
    }

    ll countLess(int y, int k) { // to
        count greater n - countless
        auto cur = root;
        ll ans = 0;
        for(int i = B - 1; i >= 0 &&
            cur; i--) {
            int yb = (y >> i) & 1;
            int kb = (k >> i) & 1;
            if(kb == 1) {
                if(cur->child[yb]) {
                    ans += cur->
                        child[yb]->cnt;
                }
                cur = cur->child[!yb];
            }
            else {
                cur = cur->child[yb];
            }
        }
        return ans;
    }
    int occurence(int x) {
        auto cur = root;
        for(int i = B - 1; i >= 0; i--) {
            int id = (x >> i) & 1;
            if(!cur->child[id]) return 0;
            cur = cur->child[id];
        }
        return cur->leaf;
    }

    void del(int x) {
        stack<pair<node *, int>> stck;
        auto cur = root;
        for(int i = B - 1; i >= 0; i--) {
            int id = (x >> i) & 1;
            stck.push({cur, id});
            cur = cur->child[id];
        }
        while(not stck.empty()) {
            auto [par, id] = stck.top();
            stck.pop();
            auto child = par->child[id];
            if(!child->child[0] &&
                !child->child[1]) {
                delete child;
                par->child[id] = nullptr;
            }
            else {
                break;
            }
        }
    }

    void remove(int x) {
        if(occurence(x) == 1) {
            del(x);
            return;
        }
        auto cur = root;
        for(int i = B - 1; i >= 0; i--) {
            int id = (x >> i) & 1;
            cur = cur->child[id];
        }
        cur->leaf--;
    }

    int get_max(int x) {
        auto cur = root;
        int ans = 0;
        for(int i = B - 1; i >= 0; i--) {
            int id = (x >> i) & 1;
            if(cur->child[!id]) {
                //cout << i << " " << id << "
                    " << sz << "\n";
                ans += (1 << i);
                cur = cur->child[!id];
            }
            else if(cur->child[id]){
                cur = cur->child[id];
```

```
            }
            else break;
        }
    }
    void clear(node *cur) {
        if(!cur) return;
        for(int i = 0; i < 2; i++) {
            if(cur->child[i])
                clear(cur->child[i]);
        }
        delete(cur);
    }
    ~Trie() {
        clear(root);
    }
};
ll ans = 0;
int pref = 0;
Trie trie;
trie.insert(0);
for(auto x : v) {
    pref ^= x;
    ans += trie.countLess(pref, k);
    trie.insert(pref);
}
```

11.15 wavelet tree [77 lines] - 90f623bba4

```
// Wavelet Tree structure used for
    answering range queries efficiently
struct wavelet_tree {
    int lo, hi;             // range of
        values represented at this node
    wavelet_tree *l, *r;    // left and
        right children
    int *b;
    ll *c;              // b: bitmap of
        counts going left, c: prefix sums
    int bsz, csz;           // sizes of b
        and c arrays

    // Constructor: initializes members
    wavelet_tree() : lo(1), hi(0),
        l(nullptr), r(nullptr),
                     b(nullptr), bsz(0),
                         c(nullptr),
                         csz(0) {}

    // Initializes wavelet tree on range
        [from, to) with values in [x, y]
    void init(int *from, int *to, int x,
        int y) {
        lo = x, hi = y;
        if(from >= to) return;

        int mid = (lo + hi) >> 1;
        auto f = [mid](int x) {
            return x <= mid;
        };

        int n = to - from;
        b = (int*)malloc((n + 2) *
            sizeof(int));
        c = (ll*)malloc((n + 2) *
            sizeof(ll));
        bsz = csz = 0;
        b[bsz++] = 0;
        c[csz++] = 0;

        for(auto it = from; it != to;
            it++) {
            b[bsz] = b[bsz - 1] +
                f(*it);     // Count of
                elements ≤ mid
            c[csz] = c[csz - 1] + (*it);
                // Prefix sum of elements
            bsz++;
            csz++;
        }

        if(hi == lo) return;

        auto pivot =
            stable_partition(from, to,
            f);  // Separate elements
        l = new wavelet_tree();
            l->init(from, pivot, lo,
            mid);       // Left subtree
        r = new wavelet_tree();
            r->init(pivot, to, mid + 1,
            hi);     // Right subtree
    }

    // Returns count of numbers ≤ k in
        range [l, r]
    int LTE(int l, int r, int k) {
        if(l > r || k < lo) return 0;
        if(hi <= k) return r - l + 1;

        int lb = b[l - 1], rb = b[r];
        return this->l->LTE(lb + 1, rb,
            k) + this->r->LTE(l - lb, r
            - rb, k);
    }

    // Returns sum of elements ≤ k in
        range [l, r]
    ll sum(int l, int r, int k) {
        if(l > r || k < lo) return 0;
        if(hi <= k) return c[r] - c[l -
            1];

        int lb = b[l - 1], rb = b[r];
        return this->l->sum(lb + 1, rb,
            k) + this->r->sum(l - lb, r
            - rb, k);
    }
    //kth smallest element in [l, r]
    //for array [1,2,1,3,5] 2nd smallest
        is 1 and 3rd smallest is 2
    int kth(int l, int r, int k) {
        if(l > r) return 0;
        if(lo == hi) return lo;
        int inLeft = b[r] - b[l - 1], lb
            = b[l - 1], rb = b[r];
        if(k <= inLeft) return
            this->l->kth(lb + 1, rb, k);
        return this->r->kth(l - lb, r -
            rb, k - inLeft);
    }
    // Destructor to free memory
    ~wavelet_tree() {
        delete l;
        delete r;
        if(b) free(b);
        if(c) free(c);
    }
};
```

12 Random

12.1 Combinatorics

- $\sum_{k=0}^{n} \binom{n-k}{k} = Fib_{n+1}$
- $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$
- $k\binom{n}{k} = n\binom{n-1}{k-1}$
- Number of binary sequences of length n such that no two 0's are adjacent = Fib_{n+1}
- Number of non-negative solution of $x_1 + x_2 + x_3 + ... + x_k = n$ is $\binom{n+k-1}{n}$

12.1.1 Catalan Number

- $C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$
- $C_0 = 1, C_1 = 1, C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
- $1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786$
- Number of correct bracket sequences consisting of n opening brackets.
- Number of ways to completely parenthesize n+1 factors.
- The number of triangulations of a convex polygon with +2 sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- The number of ways to connect the 2n points on a circle to form n disjoint i.e. non-intersecting chords.
- The number of monotonic lattice paths from point (0,0) to point (n,n) in a square lattice of size $n \times n$, which do not pass above the main diagonal
- Number of permutation of length n that can be stack sorted.
- The number of non-crossing partitions of a set of n elements.
- The number of rooted full binary tree with n+1 leaves.
- The number of Dyck words of length 2n. A string consisting of n X's and n Y's such that no string prefix has more Y's than X's.
- Number of permutation of length n with no three-term increasing subsequence.
- Number of ways to tile a stairstep shape of height n with n rectangle.
- $C_n^k = \frac{k+1}{n+1}\binom{2n-k}{n-k}$ denote the number of bracket sequences of size 2n with the first k elements being (.
- $N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$
- The number of expressions containing n pairs of correct parentheses, which contain k distinct nestings. $N(4,2) = 6$ $()((()))$, $(())(())$, $(()(()))$, $((()()))$, $(((())))$, $(((()))$
- The number of paths from (0,0) to (2n, 0) with steps only northeast and southeast, not staying below the x-axis with k peaks. And sum of all number of peaks is Catalan number.

12.1.2 Stirling Number of the First Kind

- Count permutation according to their number of cycles.
- $S(n,k)$ count the number of permutation of n elements with k disjoint cycles.
- $S(n,k) = (n-1) \times S(n-1,k) + S(n-1,k-1), S(0,0) = 1, S(n,0) = S(0,n) = 0$
- $S(n,1) = (n-1)!$
- $S(n,n-1) = \binom{n}{2}$
- $\sum_{k=0}^{n} S(n,k) = n!$

12.1.3 Stirling Numbers of the Second Kind

- Number of ways to partition a set of n objects into k non-empty subsets.
- $S(n,k) = k * S(n-1,k) + S(n-1,k-1), S(0,0) = 1, S(n,0) = S(0,n) = 0$
- $S(n,2) = 2^{n-1} - 1$
- $S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$
- $S(n,k) * k!$ = number of ways to color n nodes using colors from 1 to k such that each color is used at least once.

12.1.4 Bell Number

- Counts the number of partitions of a set.
- $B_{n+1} = \sum_{k=0}^{n}\binom{n}{k} * B_k$
- $B_n = \sum_{k=0}^{n} S(n,k)$, where S is Stirling number of second kind.
- The number of multiplicative partitions of a square free number with i prime factors is the i-th Bell number.
- $B(p^m+n) \equiv mB(n)+B(n+1)(\mod p)$
- If a deck is shuffled by removing and reinserting the top card n times, there are n^n possible shuffles. The number of shuffles that return the deck to its original order is B_n, so the probability of returning to the original order is B_n/n^n.

12.1.5 Lucas Theorem

- If p is prime then $\binom{p^a}{k} \equiv 0 \mod p$
- For non-negative integers m and n and a prime p:
  $\binom{m}{n} = \prod_{i=0}^{k}\binom{m_i}{n_i}(\mod p)$ where
  $m = m_k p^k + m_{k-1}p^{k-1} + ... + m_1 p + m_0$
  $n = n_k p^k + n_{k-1}p^{k-1} + ... + n_1 p + n_0$
  are the base p expansion.

12.1.6 Derangement

- A permutation such that no element appears in its original position.
- $d(n) = (n-1) * (d(n-1) + d(n-2)), d(0) = 1, d(1) = 0$
- $d(n) = nd(n-1) + (-1)^n = \lfloor \frac{n!}{e} \rfloor, n \geq 1$

12.1.7 Burnside Lemma

Given a group G of symmetries and a set X, the number of elements of X up to symmetry equals

$$\frac{1}{|G|}\sum_{g \in G}|X^g|$$

where X^g are the elements fixed by $g(g.x = x)$ If f(n) counts "configurations" of some sort of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n}\sum_{k=0}^{n-1} f(gcd(n,k)) = \frac{1}{n}\sum_{k|n} f(k)\phi(n/k)$$

12.1.8 Eulerian Number

- $E(n,k)$ is the number of permutations of the numbers 1 to n in which exactly k elements are greater than the previous element.
- $E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k), E(n,0) = E(n,n-1) = 1$
- $E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$
- $E(n,k) = E(n,n-1-k)$
- $E(0,k) = [k=0]$
- $E(n,1) = 2^n - n - 1$

12.2 Number Theory

12.2.1 Mobius Function and Inversion

- define $\mu(n)$ as the sum of the primitive nth roots of unity depending on the factorization of n into prime factors:

$$\mu(x) = \begin{cases} 0 & \text{n is not square free} \\ 1 & \text{n has even number of prime factors} \\ -1 & \text{n has odd number of prime factors} \end{cases}$$

- Mobius Inversion:
  $g(n) = \sum_{d|n} f(d) \leftrightarrow f(n) = \sum_{d|n}\mu(d)g(n/d)$
- $\sum_{d|n}\mu(d) = [n=1]$
- $\phi(n) = \sum_{d|n}\mu(d).\frac{n}{d} = n\sum_{d|n}\frac{\mu(d)}{d} = \sum_{d|n} d.\mu(\frac{n}{d})$
- $a|b \rightarrow \phi(a)|\phi(b)$
- $\phi(mn) = \phi(m).\phi(n).\frac{d}{\phi(d)}$ where $d = gcd(m,n)$
- $\phi(n^m) = n^{m-1}\phi(n)$
- $\sum_{i=1}^{n}[gcd(i,n) = k] = \phi(\frac{n}{k})$
- $\sum_{i=1}^{n} gcd(i,n) = \sum_{d|n} d.\phi(\frac{n}{d})$
- $\sum_{i=1}^{n}\frac{1}{gcd(i,n)} = \sum_{d|n}\frac{1}{d}.\phi(\frac{n}{d}) = \frac{1}{n}\sum_{d|n} d.\phi(d)$
- $\sum_{i=1}^{n}\frac{i}{gcd(i,n)} = \frac{n}{2}.\sum_{d|n}\frac{1}{d}.\phi(\frac{n}{d}) = \frac{n}{2}.\frac{1}{n}\sum_{d|n} d.\phi(d)$
- $\sum_{i=1}^{n}\frac{n}{gcd(i,n)} = 2.\sum_{i=1}^{n}\frac{i}{gcd(i,n)} - 1$

12.2.2 GCD and LCM

- gcd(a,b) = gcd(b, a mod b)
- If $a|b.c$, and gcd(a,b) = d, then $(a/d)|c$.
- GCD is a multiplicative function.
- gcd(a, lcm(b,c)) = lcm(gcd(a,b), gcd(a,c))
- $gcd(n^a - 1, n^b - 1) = n^{gcd(a,b)} - 1$

12.2.3 Gauss Circle Theorem

- Determine the number of lattice points in a circle centered at the origin with radius r.
- number of pairs (m,n) such that $m^2 + n^2 \leq r^2$
- $N(r) = 1 + 4\sum_{i=0}^{\infty}(\lfloor\frac{r^2}{4i+1}\rfloor - \lfloor\frac{r^2}{4i+3}\rfloor)$

12.2.4 Pick's Theorem

According to Pick's Theorem We can calculate the area of any polygon by just counting the number of Interior and Boundary lattice points of that polygon. If number of interior points are I and number of boundary lattice points are B then Area (A) of polygon will be:

$$Area = I + B/2 - 1$$

where I is the number of points in the interior shape, B stands for the number of points on the boundary of the shape.

12.2.5 Formula Cheatsheet

- $\sum_{i=1}^{n} = \frac{1}{m+1}[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}((i+1)^{m+1} - i^{m+1} - (m+1)i^m)]$
- $\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, c \neq 1$
- $\sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, |c| < 1$
- $H_n = \sum_{i=1}^{n}\frac{1}{n}, \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}$
- $\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$