University of Dhaka

DU_Primordius

fahimcp495, _timelord, Shakil17

Team Reference Document for ICPC Dhaka Regional 2024

Contents

1 Code

1.1 2SAT

```cpp
//CNF: (a | b) ^ (c | d) means (!a -> b) ^ (!b -> a)
// (!a or b) = (-a, b), 1-based indexing
string two_sat(int n, vector<array<int, 2>> clauses) {
  vector<int> adj[2 * n];
  for (auto [a, b]: clauses) {
    if (a > 0)  a = 2 * a - 2;
    else  a = 2 * -a - 1;
    if (b > 0)  b = 2 * b - 2;
    else  b = 2 * -b - 1;
    adj[a ^ 1].push_back(b), adj[b ^ 1].push_back(a);
  }
  vector<vector<int>> sccs = get_sccs(2 * n, adj);
  int tot_scc = sccs.size();
  vector<int> scc_no(2 * n);
  for (int i = 0; i < tot_scc; ++i) {
    for (int u: sccs[i]) {
      scc_no[u] = i;
    }
  }
  string assignment;
  for (int u = 0; u < n; u++) {
    if (scc_no[2 * u] == scc_no[2 * u + 1]) {
      return "";
    }
    if (scc_no[2 * u] < scc_no[2 * u + 1]) {
      assignment += '-';
    }
    else {
      assignment += '+';
    }
  }
  return assignment;
}
```

1.2 Aho Corasick

```cpp
const int N = 1e5+5, A = 26;
int to[N][A], cnt[N], node[N], tot = 1;
vector<int> idx[N];
void add(string &p, int i) {
  int u = 0;
  for (auto c: p) {
    c -= 'a';
    if (!to[u][c]) to[u][c] = tot++;
    u = to[u][c];
  }
  cnt[u]++;
  node[i] = u;
  idx[u].push_back(i);
}
vector<int> slnk(N), olnk(N), adj[N], order;
void build() {
  queue<int> q; q.push(0);
  while (!q.empty()) {
    int u = q.front(); q.pop();
    order.push_back(u);
    for (int c = 0; c < A; ++c) {
      int v = to[u][c];
      if (!v) continue;
      q.push(v);
      dp2[v] = cnt[v];
      if (!u) continue;
      int cur = slnk[u];
      while (cur and !to[cur][c]) cur = slnk[cur];
      slnk[v] = to[cur][c];
      if (cnt[lnk[v]]) olnk[v] = lnk[v];
      else olnk[v] = olnk[lnk[v]];
      adj[to[cur][c]].push_back(v);
      dp2[v] += dp2[slnk[v]];
    }
  }
}
vector<int> idx2[N];
void trav(string &s) {
  int u = 0;
  vector<int> dp(tot);
  for (int i = 0; i < s.size(); ++i) {
    int c = s[i] - 'a';
    while (u and !to[u][c]) u = slnk[u];
    u = to[u][c];
    dp[u]++, idx2[u].push_back(i);
    ans[i] = dp2[i];
    for (int v = u; v; v = olnk[v]) {
```

```
      for (auto j: idx[v]) {
        // jth pattern ends at ith position of s
      }
    }
  }
  // Count positions
  reverse(order.begin(), order.end());
  for (auto u: order) {
    dp[slnk[u]] += dp[u];
  }
}
// Find positions of a pattern
int pos[N], f;
void go(int u) {
  for (auto i: idx2[u]) pos[f++] = i;
  for (auto v: adj[u]) go(v);
}
// add(p[i], i); build(); trav(s);
// dp[node[i]] = counts of occurrence of the ith
  pattern in the text
// go(node[i]) finds the occurrences of the ith pattern
  in the text
// dp2[i] = number of patterns that ends at ith
  position of the Text
```

1.3 AlienTrick

Description: Divide n elements into k groups and minimize the sum of a cost function for each group. If $f_n(k) \geq f_n(k+1)$ (monotone) and $f_n(k-1) - f_n(k) \geq f_n(k) - f_n(k+1)$ (convex) we can apply alien trick. It means with the increase of k, answer becomes more optimal, but the rate of becoming optimal slows down. We define $g_n = f_n(k) + kC$. That means we need to add a penalty C for each group we use. We can binary search on C as g is convex. Compute g_n and the optimal k on each iteration – k decreases when C increases.
Time: $\mathcal{O}(n \log \max)$

1.4 Articulation Bridge Online

```
// O(n+m)*logn, Can be maintained 2ECC
vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
int ab, lca_itr;
vector<int> last_visit;
void init(int n) {
  par.resize(n);
  dsu_2ecc.resize(n);
  dsu_cc.resize(n);
  dsu_cc_size.resize(n);
  lca_itr = 0;
  last_visit.assign(n, 0);
  for (int i=0; i<n; ++i) {
    dsu_2ecc[i] = dsu_cc[i] = i;
    dsu_cc_size[i] = par[i] = -1;
  }
  ab = 0;
}
int find_2ecc(int v) {
  if (v == -1) return -1;
  return dsu_2ecc[v] == v ? v : dsu_2ecc[v] =
    find_2ecc(dsu_2ecc[v]);
}
int find_cc(int v) {
  v = find_2ecc(v);
  return dsu_cc[v] == v ? v : dsu_cc[v] =
    find_cc(dsu_cc[v]);
}
void make_root(int v) {
  v = find_2ecc(v);
  int root = v, child = -1;
  while (v != -1) {
    int p = find_2ecc(par[v]);
    par[v] = child; dsu_cc[v] = root;
    child = v; v = p;
  }
  dsu_cc_size[root] = dsu_cc_size[child];
}
void merge_path (int a, int b) {
  ++lca_itr;
  vector<int> path_a, path_b;
  int lca = -1;
  while (lca == -1) {
    if (a != -1) {
      a = find_2ecc(a);
      path_a.push_back(a);
      if (last_visit[a] == lca_itr){
        lca = a; break;
      }
      last_visit[a] = lca_itr;
      a = par[a];
    }
    if (b != -1) {
      b = find_2ecc(b);
      path_b.push_back(b);
      if (last_visit[b] == lca_itr){
        lca = b; break;
      }
      last_visit[b] = lca_itr;
      b = par[b];
    }
  }
  for (int v : path_a) {
    dsu_2ecc[v] = lca;
    if (v == lca) break;
    --ab;
  }
  for (int v : path_b) {
    dsu_2ecc[v] = lca;
    if (v == lca) break;
    --ab;
  }
}
void add_edge(int a, int b) {
  a = find_2ecc(a); b = find_2ecc(b);
  if (a == b) return;
  int ca = find_cc(a);
  int cb = find_cc(b);
  if (ca != cb) {
    ++ab;
    if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
      swap(a, b); swap(ca, cb);
    }
    make_root(a);
    par[a] = dsu_cc[a] = b;
    dsu_cc_size[cb] += dsu_cc_size[a];
  } else {
    merge_path(a, b);
  }
}
```

1.5 Articulation Bridge

```
void dfs(int u, int p) {
  tin[u] = lo[u] = ++t; int cnt = 0;
  for (auto [v, f]: adj[u]) {
    if (v == p and ++cnt <= 1) continue;
    if (tin[v]) lo[u] = min(lo[u], tin[v]);
    else {
      dfs(v, u);
      lo[u] = min(lo[u], lo[v]);
      if (tin[u] < lo[v]) {
        ab.insert({u, v});
      }}}}
```

1.6 Articulation Point

```
vector<int> adj[N];
int t = 0;
vector<int> tin(N, -1), low(N), ap;
void dfs (int u, int p) {
  tin[u] = low[u] = t++;
  int is_ap = 0, child = 0;
  for (int v: adj[u]) {
    if (v != p) {
      if (tin[v] != -1) {
        low[u] = min(low[u], tin[v]);
      }
      else {
        child++;
        dfs(v, u);
        if (tin[u] <= low[v]) {
          is_ap = 1;
        }
        low[u] = min(low[u], low[v]);
      }
    }
  }
  if ((p != -1 or child > 1) and is_ap)
    ap.push_back(u);
}
dfs(0, -1);
```

1.7 Auxiliary Tree

```
int n, l, dt[N];
vector<vector<int>> adj;
int timer;
vector<int> tin, tout;
vector<vector<int>> up;
void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
      up[v][i] = up[up[v][i-1]][i-1];
    for (int u : adj[v]) {
      if (u != p)
          dfs(u, v);
    }
    tout[v] = ++timer;
}
bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}
int lca(int u, int v)
{
    if (is_ancestor(u, v))
      return u;
    if (is_ancestor(v, u))
      return v;
    for (int i = l; i >= 0; --i) {
      if (!is_ancestor(up[u][i], v))
          u = up[u][i];
    }
    return up[u][0];
}
void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

```cpp
bool cmp(int u,int v) {
    return tin[u] < tin[v];
}
vector<int>  g[N];
int virtual_tree(vector<int> vert){
    sort(vert.begin(),vert.end(),cmp);
    int k= vert.size();
    for(int i=0;i<k-1;i++){
        int nv= lca(vert[i],vert[i+1]);
        vert.push_back(nv);
    }
    sort(vert.begin(),vert.end(),cmp);
    vert.erase(unique(vert.begin(),vert.end()),
     ↪  vert.end());
    for(auto v:vert) g[v].clear();
    vector<int> stk;
    stk.push_back(vert[0]);
    for(int i=0;i<(int)vert.size();i++){
        int u= vert[i];
        while((int)stk.size() >=2 &&
         ↪  !is_ancestor(stk.back(),u)){
            int sz= stk.size();
            g[stk[sz-2]].push_back(stk.back());
            stk.pop_back();
        }
        stk.push_back(u);
    }
    while((int)stk.size()>=2){
        int sz= stk.size();
        g[stk[sz-2]].push_back(stk.back());
        stk.pop_back();
    }
    return stk[0];
}
```

1.8 BCC Edge

```cpp
vector<array<int, 2>> edges, adj[N];
vector <int> tin(N), lo(N), is_ap(N), bcc[N],
 ↪  bcc_ed[N];
int t = 0, tot = 0;
stack<int> stk;
void pop_bcc(int e) {
  do {
    bcc_ed[tot].push_back(stk.top());  stk.pop();
  } while (bcc_ed[tot].back() != e);
  tot++;
}
void dfs(int u, int p = -1) {
  int ch = 0;
  tin[u] = lo[u] = t++;
  for(auto [v, e] : adj[u]) {
    if (v == p)  continue;
    if (tin[v] != -1) {
      if (tin[u] > tin[v]) {
        lo[u] = min(lo[u], tin[v]);
        stk.push(e);
      }
    }
    else {
      ch++;
      stk.push(e);
      dfs(v, u);
      if ((p != -1 or ch > 1) and tin[u] <= lo[v]) {
        is_ap[u] = 1;
        pop_bcc(e);
      }
      lo[u] = min(lo[u], lo[v]);
    }
  }
}
void procces_bcc(int n) {
  for (int i = 0; i < n; ++i) {
    tin[i] = -1, is_ap[i] = 0;
    bcc_ed[i].clear();
    bcc[i].clear();
  }
  t = tot = 0;
  for (int u = 0; u < n; ++u) {
    if (tin[u] == -1) {
      dfs(u, -1);
      if (!stk.empty()) {
        while (!stk.empty()) {
          bcc_ed[tot].push_back(stk.top());  stk.pop();
        }
        tot++;
      }
    }
  }
  for (int i = 0; i < tot; ++i) {
    for (auto e: bcc_ed[i]) {
      auto [u, v] = edges[e];
      bcc[i].push_back(u);
      bcc[i].push_back(v);
    }
  }
  for (int i = 0; i < tot; ++i) {
    sort(bcc[i].begin(), bcc[i].end());
    bcc[i].erase(unique(bcc[i].begin(), bcc[i].end()),
     ↪  bcc[i].end());
  }
}
```

1.9 BCC

```cpp
struct graph {
  int n,t=0,cno=0;
  vector<vector<int>> g;
  vector<int> tin, lo, bcomp;
  stack <int> st;
  graph(int n):n(n),g(n),lo(n),bcomp(n){}
  void add_edge(int u, int v){
    g[u].push_back(v);
    g[v].push_back(u);
  }
  void dfs(int v, int p=-1){
    lo[v]=tin[v]=++t;
    st.push(v);
    for(int u:g[v]){
      if(u==p)    continue;
      if(!tin[u]){
        dfs(u, v);
        lo[v]=min(lo[v],lo[u]);
      } else{
        lo[v]=min(lo[v],tin[u]);
      }
    }
    if(tin[v]==lo[v]){
      while (!st.empty()){
        int tp=st.top(); st.pop();
        bcomp[tp]=cno;
        if(tp==v)    break;
      }
      cno++;
    }
  }
  vector<int> bcc(){
    tin.assign(n, 0);
    for (int i = 0; i < n; ++i){
      if(!tin[i])
        dfs(i);
    }
    return bcomp;
  }
};
```

1.10 Block Cut Tree

```cpp
vector<int> adj[N];
vector<int> tin(N, -1), lo(N), is_ap(N), bcc[N];
stack<int> stk;
int t = 0, tot = 0;
void pop_bcc(int u, int v) {
  bcc[tot].push_back(u);
  while (bcc[tot].back() != v) {
    bcc[tot].push_back(stk.top());
    stk.pop();
  }
  tot++;
}
void dfs (int u, int p) {
  tin[u] = lo[u] = t++;
  stk.push(u);
  int ch = 0;
  for (auto v: adj[u]) {
    if (v != p) {
      if (tin[v] != -1) {
        lo[u] = min(lo[u], tin[v]);
      }
      else {
        ch++;
        dfs(v, u);
        if ((p != -1 or ch > 1) and tin[u] <= lo[v]) {
          // is_ap[u] = 1;
          pop_bcc(u, v);
        }
        lo[u] = min(lo[u], lo[v]);
      }
    }
  }
}
void process_bcc (int n) {
  for (int u = 0; u < n; ++u) {
    tin[u] = -1;
    is_ap[u] = 0;
    bcc[u].clear();
  }
  t = tot = 0;
  for (int u = 0; u < n; ++u) {
    if (tin[u] == -1) {
      dfs(u, -1);
      if (!stk.empty()) {
        while (!stk.empty()) {
          bcc[tot].push_back(stk.top());
          stk.pop();
        }
        tot++;
      }
    }
  }
}
int nn;
vector<int> comp_num(N), bct_adj[N];
void build_bct(int n) {
  process_bcc(n);
  int nn = tot;
  for (int u = 0; u < n; ++u) {
    if (is_ap[u]) {
      comp_num[u] = nn++;
    }
  }
  for (int i = 0; i < tot; ++i) {
    for (auto u: bcc[i]) {
      if (is_ap[u]) {
        u = comp_num[u];
```

```
        bct_adj[i].push_back(u);
        bct_adj[u].push_back(i);
      }
      else {
        comp_num[u] = i;
      }
    }
  }
}
```

1.11 Bridge Tree

Description: finds all edge-biconnected components and compresses them into a tree.
Time: $\mathcal{O}(V+E)$

```
vector <int> g[N], tree[N];
int n, m, in[N], low[N], ptr, compID[N];
void go (int u, int par = -1) {
  in[u] = low[u] = ++ptr;
  for (int v : g[u]) {
    if (in[v]) {
      if (v == par) par = -1;
      else low[u] = min(low[u], in[v]);
    } else {
      go(v, u); low[u] = min(low[u], low[v]);
    }
  }
}
void shrink (int u, int id) {
  compID[u] = id;
  for (int v : g[u]) if (!compID[v]) {
    if (low[v] > in[u]) {
      tree[id].emplace_back(++ptr);
      shrink(v, ptr);
    } else { shrink(v, id); }
  }
}
int main() {
  cin >> n >> m;
  while (m--) {
    int u, v;
    scanf("%d %d", &u, &v);
    g[u].emplace_back(v);
    g[v].emplace_back(u);
  }
  for (int i = 1; i <= n; ++i) if (!in[i]) go(i);
  vector <int> roots; ptr = 0;
  for (int i = 1; i <= n; ++i) if (!compID[i]) {
    roots.emplace_back(++ptr);
    shrink(i, ptr);
  }
}
```

1.12 CHT

Description:

- If m is decreasing:
  - for min : bad(s-3, s-2, s-1), x increasing
  - for max : bad(s-1, s-2, s-3), x decreasing
- If m is increasing:
  - for max : bad(s-3, s-2, s-1), x increasing
  - for min : bad(s-1, s-2, s-3), x decreasing
- If x isn't monotonic, then do Ternary Search or keep intersections and do binary search

```
struct CHT {
  vector<ll> m, b;
  int ptr = 0;
  bool bad(int l1, int l2, int l3) { // returns
   ↪ intersect(l1, l3) <= intersect(l1, l2)
    return 1.0 * (b[l3] - b[l1]) * (m[l1] - m[l2]) <=
     ↪ 1.0 * (b[l2] - b[l1]) * (m[l1] - m[l3]);
  }
  void insert_line(ll _m, ll _b) {
    m.push_back(_m);
    b.push_back(_b);
    int s = m.size();
    while (s >= 3 && bad(s - 1, s - 2, s - 3)) {
      s--;
      m.erase(m.end() - 2);
      b.erase(b.end() - 2);
    }
  }
  ll f(int i, ll x) { return m[i] * x + b[i]; }
  ll eval(ll x) {
    if (ptr >= m.size()) ptr = m.size() - 1;
    while (ptr < m.size() - 1 && f(ptr + 1, x) >
     ↪ f(ptr, x)) ptr++;
    return f(ptr, x);
  }
};
```

1.13 Centroid Decomposition

```
vector<int> adj[N];
int sz[N], cen[N];
void dfs_sz(int u, int p) {
  sz[u] = 1;
  for (auto v: adj[u]) {
    if (v != p and !cen[v]) {
      dfs_sz(v, u); sz[u] += sz[v];
    }
  }
}
int get(int u, int p, int k) {
  for (auto v: adj[u]) {
    if (v != p and !cen[v] and sz[v] > k) return
     ↪ get(v, u, k);
  }
  cen[u] = 1; return u;
}
int cpar[N], cdep[N], dis[N][K];
vector<int> cnt[N], cntp[N];
void dfs(int u, int p, int d) {
  for (auto v: adj[u]) {
    if (v != p and !cen[v]) {
      dis[v][d] = dis[u][d] + 1; dfs(v, u, d);
    }
  }
}
int go(int u, int d) {
  dfs_sz(u, u);
  int tot = sz[u];
  u = get(u, u, sz[u] >> 1);
  cdep[u] = d, dis[u][d] = 0;
  dfs(u, u, d);
  cnt[u].resize(tot);
  cntp[u].resize(tot + 1);
  for (auto v: adj[u]) {
    if (!cen[v]) {
      int w = go(v, d + 1);
      cpar[w] = u;
    }
  }
  return u;
}
int n, k;
ll ans = 0;
void add(int u) {
  int pv = -1, v = u;
  while (v != -1) {
    int rem = k - dis[u][cdep[v]];
    if (rem >= 0 and rem < cnt[v].size()) {
      ans += cnt[v][rem];
    }
    if (pv != -1) {
      if (rem >= 0 and rem < cntp[pv].size()) {
        ans -= cntp[pv][rem];
      }
    }
    pv = v;
    v = cpar[v];
  }
  pv = -1, v = u;
  while (v != -1) {
    int d = dis[u][cdep[v]];
    cnt[v][d]++;
    if (pv != -1) {
      cntp[pv][d]++;
    }
    pv = v;
    v = cpar[v];
  }
}
```

1.14 DP on Tree

```
// Rerooting Technique
int down[N], dp[N];
// This dfs is supposed to calculate down[u] from its
 ↪ childs down value
void dfs(int u, int p) {
  for (auto v: adj[u]) {
    if (v != p) {
      dfs(v, u);
      // down[u] <- down[v]
    }
  }
}
// This dfs is supposed to calculate its childs up
 ↪ value
// Assuming up[u] is already calculated
void dfs2(int u, int p) {
  // Add contribution of down[siblings] to up[v]
  ll pref = ?;
  for (auto v: adj[u]) {
    // up[v] <- pref
    // Update pref
  }
  reverse(adj[u].begin(), adj[u].end());
  ll suf = ?;
  for (auto v: adj[u]) {
    // up[v] <- suf
    // Update suf
  }
  for (auto v: adj[u]) {
    // Add contribution of up[u] to up[v]
    // up[v] <- up[u]
    dfs2(v, u);
  }
}
```

1.15 DSU On Tree

```
void dfs(int u, int p) {
  node[t] = u, tin[u] = t++, sz[u] = 1, hc[u] = -1;
  for (auto v: adj[u]) {
    if (v != p) {
      dfs(v, u);
      sz[u] += sz[v];
      if (hc[u] == -1 or sz[hc[u]] < sz[v]) hc[u] = v;
    }
  }
```

```cpp
  tout[u] = t - 1;
}
void dsu(int u, int p, int keep) {
  for (int v: adj[u]) {
    if (v != p and v != hc[u]) dsu(v, u, 0);
  }
  if (hc[u] != -1) dsu(hc[u], u, 1);
  for (auto v: adj[u]) {
    if (v != p and v != hc[u]) {
      for (int i = tin[v]; i <= tout[v]; ++i) {
        int w = node[i];
        // Update ans if ans is related to path/pair
      }
      for (int i = tin[v]; i <= tout[v]; ++i) {
        int w = node[i];
        // Add contribution of node w
      }
    }
  }
  // Add contribution of node u
  // Update ans if ans is related to subtree
  if (!keep) {
    for (int i = tin[u]; i <= tout[u]; ++i) {
      int w = node[i];
      // Remove contribution of node w
    }
    // Data structure is empty now
  }
}
dfs(0, 0);  dsu(0, 0, 0);
```

1.16 Debug

```cpp
string to_string(const string& s) {
  return '"' + s + '"';
}
string to_string(const char* s) {
  return to_string(string(s));
}
string to_string(const char c) {
  return "'" + string(1, c) + "'";
}
string to_string(bool b) {
  return b ? "true" : "false";
}
template <typename A, typename B>
string to_string(pair<A, B> p) {
  return "(" + to_string(p.first) + ", " +
   ↪ to_string(p.second) + ")";
}
template <typename A>
string to_string(A v) {
  string res = "{";
  for (const auto &x : v) {
    res += to_string(x) + ", ";
  }
  res += "}";
  return res;
}
void debug_out() { cerr << endl; }
template <typename Head, typename... Tail>
void debug_out(Head H, Tail... T) {
  cerr << " " << to_string(H);
  debug_out(T...);
}
#define dbg(...) cerr << "[" << #__VA_ARGS__ << "]:",
 ↪ debug_out(__VA_ARGS__)
```

1.17 Determinant

```cpp
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));
double det = 1;
for (int i=0; i<n; ++i) {
  int k = i;
  for (int j=i+1; j<n; ++j)
    if (abs (a[j][i]) > abs (a[k][i]))
      k = j;
  if (abs (a[k][i]) < EPS) {
    det = 0;
    break;
  }
  swap (a[i], a[k]);
  if (i != k)
    det = -det;
  det *= a[i][i];
  for (int j=i+1; j<n; ++j)
    a[i][j] /= a[i][i];
  for (int j=0; j<n; ++j)
    if (j != i && abs (a[j][i]) > EPS)
      for (int k=i+1; k<n; ++k)
        a[j][k] -= a[i][k] * a[j][i];
}
```

1.18 Dinic

Description: Lower bound on capacity – create a supersource, a supersink. If $u \to v$ has a lower bound of L, give an edge from supersource to v with capacity L. Give an edge from u to supersink with capacity L. Give an edge from normal sink to normal source with capacity infinity. If max flow here is equal to L, then the lower bound can be satisfied. For minimum flow satisfying lower bounds, binary search on the capacity from normal sink to normal source (instead of assigning inf). For maximum flow satisfying bounds, just add another source to normal source and binary search on capacity.

Time: $\mathcal{O}(V^2E)$, (on unit graphs $\mathcal{O}(E\sqrt{V})$)

```cpp
// Effective flows are adj[u][3] where adj[u][3] > 0
ll get_max_flow(vector<array<int, 3>> edges, int n,
 ↪ int s, int t) {
  vector<array<ll, 4>> adj[n];
  for (auto [u, v, c]: edges) {
    adj[u].push_back({v, (int)adj[v].size(), c, 0});
    adj[v].push_back({u, (int)adj[u].size() - 1, 0,
     ↪ 0});
  }
  ll max_flow = 0;
  while (true) {
    queue<int> q;  q.push(s);
    vector<int> dis(n, -1);  dis[s] = 0;
    while (!q.empty()) {
      int u = q.front();  q.pop();
      for (auto [v, idx, c, f]: adj[u]) {
        if (dis[v] == -1 and c > f) {
          q.push(v);
          dis[v] = dis[u] + 1;
        }
      }
    }
    if (dis[t] == -1)  break;
    vector<int> next(n);
    function<ll(int, ll)> dfs = [&] (int u, ll flow) {
      if (u == t)  return flow;
      while (next[u] < adj[u].size()) {
        auto &[v, idx, c, f] = adj[u][next[u]++];
        if (c > f and dis[v] == dis[u] + 1) {
          ll bn = dfs(v, min(flow, c - f));
          if (bn > 0) {
            f += bn;
            adj[v][idx][3] -= bn;
            return bn;
          }
        }
      }
      return 0ll;
    };
    while (ll flow = dfs(s, LLONG_MAX)) {
      max_flow += flow;
    }
  }
  return max_flow;
}
```

1.19 Diophantine

Description: For any solution (x_0, y_0), all solutions are of the form $x = x_0 + k\frac{b}{g}, y = y_0 + k\frac{a}{g}$

```cpp
// (d, x, y) s.t ax + by = gcd(a, b) = d
tuple<ll, ll, ll> exgcd(ll a, ll b) {
  if(b == 0) return {a, 1, 0};
  auto [d, _x, _y] = exgcd(b, a % b);
  ll x = _y, y = _x - (a / b) * _y;
  return {d, x, y};
}
tuple<bool, ll, ll> diophantine(ll a, ll b, ll c) {
  auto [d, _x, _y] = exgcd(a, b);
  if(c % d) return {false, 0, 0};
  else {
    ll x = (c / d) * _x, y = (c / d) * _y;
    return {true, x, y};
  }
}
void shift_solution(ll &x, ll &y, ll a, ll b, ll cnt) {
  x += cnt * b;
  y -= cnt * a;
}
// returns the number of solutions where x is in the
 ↪ range[minx, maxx] and y is in the range[miny, maxy]
ll find_all_solutions(ll a, ll b, ll c, ll minx, ll
 ↪ maxx, ll miny,ll maxy) {
  ll g = __gcd(a, b);
  auto [res, x, y] = diophantine(a, b, c);
  if (res == false) return 0;
  if (a == 0 and b == 0) {
    assert(c == 0);
    return 1LL * (maxx - minx + 1) * (maxy - miny + 1);
  }
  if (a == 0) {
    return (maxx - minx + 1) * (miny <= c / b and c /
     ↪ b <= maxy);
  }
  if (b == 0) {
    return (maxy - miny + 1) * (minx <= c / a and 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;
}
```

1.20 DnC Optimization

Description:

- $\text{dp}[i][j] = \min_{k<j}\{\text{dp}[i-1][k] + C[k][j]\}$
- $A[i][j] \le A[i][j+1]$
- $O(kn^2)$ to $O(kn\log n)$
- sufficient: $C[a][c]+C[b][d] \le C[a][d]+C[b][c]$, $a \le b \le c \le d$ (QI)

```
// Divide an array into k parts
// Minimize the sum of squre of each subarray
ll pref[N], dp[N][N];
void compute(int l, int r, int j, int kl, int kr) {
  if (l > r)  return ;
  int m = (l + r) / 2;
  array<ll, 2> best = {LLONG_MAX, -1};
  for (int k = kl; k <= min(m - 1, kr); ++k) {
    best = min(best, {dp[k][j - 1] + (pref[m] -
    ↪  pref[k]) * (pref[m] - pref[k]), k});
  }
  dp[m][j] = best[0];
  compute(l, m - 1, j, kl, best[1]);
  compute(m + 1, r, j, best[1], kr);
}
```

1.21 Dominator Tree

Description: A node u is ancestor of node v in the dominator tree if all the the paths from source to node v contain node u. If a problem asks for edge disjoint paths, for every edge, take a new node w and turn the edge $(u \to v)$ to $(u \to w \to v)$ and find node disjoint path now. 1-based directed graph input. dtree is the edge list of the dominator tree. Clear everything at the start of each test case. Only the nodes reachable from source will be in the dominator tree.
Time: construction $\mathcal{O}(V+E)$

```
vector <int> g[sz], rg[sz], dtree[sz], bucket[sz];
int sdom[sz], par[sz], dom[sz], dsu[sz], label[sz];
int arr[sz], rev[sz], ts, source;
void dfs(int u) {
   ts++; arr[u] = ts; rev[ts] = u;
   label[ts] = sdom[ts] = dsu[ts] = ts;
   for(int &v : g[u]) {
      if(!arr[v]) { dfs(v); par[arr[v]] = arr[u]; }
      rg[arr[v]].push_back(arr[u]);
   }
}
inline int root(int u, int x = 0) {
   if(u == dsu[u]) return x ? -1 : u;
   int v = root(dsu[u], x + 1);
   if(v < 0) return u;
   if(sdom[label[dsu[u]]] < sdom[label[u]]) label[u] =
   ↪  label[dsu[u]];
   dsu[u] = v; return x ? v : label[u];
}
void build(int n) {
   dfs(source);
   for(int i=n; i; i--) {
      for(int j : rg[i]) sdom[i] =
      ↪  min(sdom[i],sdom[root(j)]);
      if(i > 1) bucket[sdom[i]].push_back(i);
      for(int w : bucket[i]) {
         int v = root(w);
         if(sdom[v] == sdom[w]) dom[w] = sdom[w];
         else dom[w] = v;
      } if(i > 1) dsu[i] = par[i];
   }
   for(int i=2; i<=n; i++) {
      int &dm = dom[i];
      if(dm ^ sdom[i]) dm = dom[dm];
      dtree[rev[i]].push_back(rev[dm]);
      dtree[rev[dm]].push_back(rev[i]);
   }
}
int main() {
  // input graph of n nodes in g[], assign "source"
  ↪  vertex
  ts = 0; build(n); // clear stuff before calling
}
```

1.22 Dynamic CHT

```
const ll IS_QUERY = -(1LL << 62);
struct line {
  ll m, b;
  mutable function <const line*()> succ;
  bool operator < (const line &rhs) const {
    if (rhs.b != IS_QUERY) return m < rhs.m;
    const line *s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s -> b < (s -> m - m) * x;
  }
};
struct CHT : public multiset <line> {
  bool bad (iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y -> m == z -> m && y -> b <= z -> b;
    }
    auto x = prev(y);
    if (z == end()) return y -> m == x -> m && y -> b
    ↪  <= x -> b;
    return 1.0 * (x -> b - y -> b) * (z -> m - y -> m)
    ↪  >= 1.0 * (y -> b - z -> b) * (y -> m - x -> m);
  }
  void add (ll m, ll b) {
    auto y = insert({m, b});
    y -> succ = [=] {return next(y) == end() ? 0 :
    ↪  &*next(y);};
    if (bad(y)) {erase(y); return;}
    while (next(y) != end() && bad(next(y)))
    ↪  erase(next(y));
    while (y != begin() && bad(prev(y)))
    ↪  erase(prev(y));
  }
  ll eval (ll x) {
    auto l = *lower_bound((line) {x, IS_QUERY});
    return l.m * x + l.b;
  }
};
// To find maximum
CHT cht;
cht.add(m, c);
y_max = cht.eval(x);
// To find minimum
CHT cht;
cht.add(-m, -c);
y_min = -cht.eval(x);
```

1.23 Dynamic Connectivity

```
const int Q = 1e5+5;
vector<array<int, 2>> t[4 * Q];
vector<int> ans(Q);
int q;
struct DSU {
  int n, comps;
  vector<int> par, rnk;
  stack<array<int, 4>> ops;
  DSU(){}
  DSU(int n): n(n), comps(n), par(n), rnk(n) {
    iota(par.begin(), par.end(), 0);
  }
  int find(int u) {
    return (par[u] == u)? u: find(par[u]);
  }
  bool unite(int u, int v) {
    u = find(u), v = find(v);
    if (u == v)  return false;
    comps--;
    if (rnk[u] > rnk[v])  swap(u, v);
    ops.push({u, rnk[u], v, rnk[v]});
    par[u] = v;
    if (rnk[u] == rnk[v])  rnk[v]++;
    return true;
  }
  void rollback() {
    if (ops.empty())  return ;
    auto [u, rnku, v, rnkv] = ops.top();  ops.pop();
    par[u] = u, rnk[u] = rnku;
    par[v] = v, rnk[v] = rnkv;
    comps++;
  }
} dsu;
void add(int l, int r, array<int, 2> ed, int u = 1,
↪  int s = 0, int e = q) {
  if (r < s or e < l)  return ;
  if (l <= s and e <= r) {
    t[u].push_back(ed);
    return ;
  }
  int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
  add(l, r, ed, v, s, m);
  add(l, r, ed, w, m + 1, e);
}
void go(int u = 1, int s = 0, int e = q) {
  int rmv = 0;
  for (auto &ed: t[u])  rmv += dsu.unite(ed[0], ed[1]);
  if (s == e)  ans[s] = dsu.comps;
  else {
    int v = 2 * u, w = 2 * u + 1, m = (s + e) / 2;
    go(v, s, m);
    go(w, m + 1, e);
  }
  while (rmv--)  dsu.rollback();
}
```

1.24 Edge Coloring

Description: Given a simple, undirected graph with max degree D, computes a $(D+1)$-coloring of the edges such that no neighboring edges share a color. (D-coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)
Time: $\mathcal{O}(NM)$

```
vi edgeColoring(int N, vector<pii> eds) {
 vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
 for (pii e : eds) ++cc[e.first], ++cc[e.second];
 int u, v, ncols = *max_element(all(cc)) + 1;
 vector<vi> adj(N, vi(ncols, -1));
 for (pii e : eds) {
  tie(u, v) = e;
  fan[0] = v;
  loc.assign(ncols, 0);
  int at = u, end = u, d, c = free[u], ind = 0, i = 0;
  while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
   loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
  cc[loc[d]] = c;
```

```cpp
  for (int cd = d; at != -1; cd ^= c ^ d, at =
    adj[at][cd])
   swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
  while (adj[fan[i]][d] != -1) {
   int left = fan[i], right = fan[++i], e = cc[i];
   adj[u][e] = left;
   adj[left][e] = u;
   adj[right][e] = -1;
   free[right] = e;
  }
  adj[u][d] = fan[i];
  adj[fan[i]][d] = u;
  for (int y : {fan[0], u, end})
   for (int& z = free[y] = 0; adj[y][z] != -1; z++);
 }
 rep(i,0,sz(eds))
  for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;)
    ++ret[i];
 return ret;
}
```

1.25 Euler Walk

Description: On directed graph, circuit (or edge disjoint directed cycles) exists iff each node satisfies in_degree = out_degree and the graph is strongly connected; path exists iff at most one vertex has in_degree - out_degree = 1 and at most one vertex has out_degree - in_degree = 1 and all other vertices have in_degree = out_degree, and graph is weakly connected. Push edge ID in circ if edges needed.
Time: $\mathcal{O}(V+E)$

```cpp
// Directed graph
vector<int> euler_cycle(vector<int> *adj, int s = 0) {
  vector<int> cycle;
  function<void(int)> dfs = [&] (int u) {
    while (!adj[u].empty()) {
      int v = adj[u].back();
      adj[u].pop_back();
      dfs(v);
    }
    cycle.push_back(u);
  };
  dfs(s);
  reverse(cycle.begin(), cycle.end());
  return cycle;
}
// Undirected graph
vector<int> euler_cycle(vector<int> *adj, vector<int>
  *des_idx, vector<int> *done, int s = 0) {
  vector<int> cycle;
  function<void(int)> dfs = [&] (int u) {
    while (!adj[u].empty()) {
      int i = adj[u].size() - 1;
      if (done[u][i]) {
        adj[u].pop_back();
        continue;
      }
      int v = adj[u][i];
      adj[u].pop_back();
      done[u][i] = 1;
      done[v][des_idx[u][i]] = 1;
      dfs(v);
    }
    cycle.push_back(u);
  };
  dfs(s);
  return cycle;
}
int n, m;  cin >> n >> m;
vector<int> adj[n], des_idx[n], done[n];
vector<int> deg(n);
for (int e = 0; e < m; ++e) {
  int u, v;  cin >> u >> v;  u--, v--;
  des_idx[u].push_back(adj[v].size());
  des_idx[v].push_back(adj[u].size());
  adj[u].push_back(v);
  adj[v].push_back(u);
  done[u].push_back(0);
  done[v].push_back(0);
  deg[u]++, deg[v]++;
}
for (int u = 0; u < n; ++u) {
  if (deg[u] & 1) {
    cout << "IMPOSSIBLE\n";
    return ;
  }
}
vector<int> cycle = euler_cycle(adj, des_idx, done, 0);
if (cycle.size() != m + 1) {
  cout << "IMPOSSIBLE\n";
  return ;
}
```

1.26 FFT

```cpp
typedef double ld; // USE DOUBLE, MUCH FASTER AND
  ALMOST ALWAYS WORKS
struct cplx {
  ld a, b;
  cplx(ld a = 0, ld b = 0) : a(a), b(b) {}
  const cplx operator +(const cplx &c) const {
    return cplx(a + c.a, b + c.b);
  }
  const cplx operator -(const cplx &c) const {
    return cplx(a - c.a, b - c.b);
  }
  const cplx operator *(const cplx &c) const {
    return cplx(a * c.a - b * c.b, a * c.b + b * c.a);
  }
  const cplx conj() const {
    return cplx(a, -b);
  }
};
const ld PI = acosl(-1);
const int MOD = 1e9 + 7;
const int N = (1 << 20) + 5;
int rev[N]; cplx w[N];
void prepare(int n) {
  int sz = __builtin_ctz(n);
  for (int i = 1; i < n; ++i) rev[i] = (rev[i >> 1] >>
    1) | ((i & 1) << (sz - 1));
  w[0] = 0, w[1] = 1, sz = 1;
  while (1 << sz < n) {
    ld ang = 2 * PI / (1 << (sz + 1));
    cplx w_n = cplx(cosl(ang), sinl(ang));
    for (int i = 1 << (sz - 1); i < (1 << sz); ++i) {
      w[i << 1] = w[i], w[i << 1 | 1] = w[i] * w_n;
    } ++sz;
  }
}
void fft(cplx *a, int n) {
  for (int i = 1; i < n - 1; ++i) {
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  }
  for (int h = 1; h < n; h <<= 1) {
    for (int s = 0; s < n; s += h << 1) {
      for (int i = 0; i < h; ++i) {
        cplx &u = a[s + i], &v = a[s + i + h], t = v *
          w[h + i];
        v = u - t, u = u + t;
      }
    }
  }
}
static cplx f[N], g[N], u[N], v[N];
vector<int> multiply(vector<int> &a, vector<int> &b) {
  int n = a.size(), m = b.size();
  int sz = n + m - 1;
  while (sz & (sz - 1)) sz = (sz | (sz - 1)) + 1;
  prepare(sz);
  for (int i = 0; i < sz; ++i) f[i] = cplx(i < n ?
    a[i] : 0, i < m ? b[i] : 0);
  fft(f, sz);
  for (int i = 0; i <= (sz >> 1); ++i) {
    int j = (sz - i) & (sz - 1);
    cplx x = (f[i] * f[i] - (f[j] * f[j]).conj()) *
      cplx(0, -0.25);
    f[j] = x, f[i] = x.conj();
  }
  fft(f, sz);
  vector<int> c(sz);
  for (int i = 0; i < sz; ++i) c[i] = f[i].a / sz +
    0.5;
  return c;
}
vector<int> multiplyMod(vector<int> &a, vector<int>
  &b) {
  int n = a.size(), m = b.size();
  int sz = 1;
  while (sz < n + m - 1) sz <<= 1;
  prepare(sz);
  for (int i = 0; i < sz; ++i) {
    f[i] = i < n ? cplx(a[i] & 32767, a[i] >> 15) :
      cplx(0, 0);
    g[i] = i < m ? cplx(b[i] & 32767, b[i] >> 15) :
      cplx(0, 0);
  }
  fft(f, sz), fft(g, sz);
  for (int i = 0; i < sz; ++i) {
    int j = (sz - i) & (sz - 1);
    static cplx da, db, dc, dd;
    da = (f[i] + f[j].conj()) * cplx(0.5, 0);
    db = (f[i] - f[j].conj()) * cplx(0, -0.5);
    dc = (g[i] + g[j].conj()) * cplx(0.5, 0);
    dd = (g[i] - g[j].conj()) * cplx(0, -0.5);
    u[j] = da * dc + da * dd * cplx(0, 1);
    v[j] = db * dc + db * dd * cplx(0, 1);
  }
  fft(u, sz), fft(v, sz);
  vector<int> c(sz);
  for (int i = 0; i < sz; ++i) {
    int da = (ll) (u[i].a / sz + 0.5) % MOD;
    int db = (ll) (u[i].b / sz + 0.5) % MOD;
    int dc = (ll) (v[i].a / sz + 0.5) % MOD;
    int dd = (ll) (v[i].b / sz + 0.5) % MOD;
    c[i] = (da + ((ll) (db + dc) << 15) + ((ll) dd <<
      30)) % MOD;
  }
  return c;
}
```

1.27 FWHT

Description: AND, OR works for any modulo, XOR works for only prime. All works without mod. Size must be power of two
Time: $\mathcal{O}(nlogn)$

```cpp
const int mod = 998244353;
void fwht(vector<int> &a, int inv, int f) {
  int sz = a.size();
  for (int len = 1; 2 * len <= sz; len <<= 1) {
    for (int i = 0; i < sz; i += 2 * len) {
      for (int j = 0; j < len; j++) {
```

```cpp
        int x = a[i + j];
        int y = a[i + j + len];
        if (f == 0) {
          if (!inv)  a[i + j] = y, a[i + j + len] = add(x,  y);
          else  a[i + j] = sub(y, x), a[i + j + len] = x;
        }
        else if (f == 1) {
          if (!inv)  a[i + j + len] = add(x, y);
          else  a[i + j + len] = sub(y, x);
        }
        else {
          a[i + j] = add(x, y);
          a[i + j + len] = sub(x, y);
        }
      }
    }
  }
}
vector<int> mul(vector<int> a, vector<int> b, int f) { // 0:AND, 1:OR, 2:XOR
  int sz = a.size();
  fwht(a, 0, f);  fwht(b, 0, f);
  vector<int> c(sz);
  for (int i = 0; i < sz; ++i) {
    c[i] = 1ll * a[i] * b[i] % mod;
  }
  fwht(c, 1, f);
  if (f) {
    int sz_inv = poww(sz, mod - 2, mod);
    for (int i = 0; i < sz; ++i) {
      c[i] = 1ll * c[i] * sz_inv % mod;
    }
  }
  return c;
}
```

1.28 Fenwick Tree

```cpp
## Range Update Range Sum
ll ft[2][N];
void add(int no, int i, ll x) {
  i++;
  while (i<=n){
    ft[no][i] += x;
    i += i & -i;
  }
}
void radd(int l, int r, ll x){
  add(0, l, x);
  add(0, r+1, -x);
  add(1, l, x*(l-1));
  add(1, r+1, -x*r);
}
ll csumft(int no, int i) {
  i++;
  ll ret = 0;
  while (i>0){
    ret += ft[no][i];
    i -= i & -i;
  }
  return ret;
}
ll csum(int i) {
  return csumft(0, i)*i - csumft(1, i);
}
ll rsum(int l, int r) {
  return csum(r)- csum(l-1);
}
## Binary Search on Prefix Sum
int lb(ll x) {
  int i = 0;
  for (int k = K - 1; k >= 0; --k) {
    int j = i + (1 << k);
    if (j <= n and ft[j] < x) {
      i = j, x -= ft[j];
    }
  }
  return i;
}
```

1.29 Floor Sum

```cpp
ll floor_sum(ll n, ll m, ll a, ll b) {
  ll ans = 0;
  if (a >= m) {
    ans += (n - 1) * n * (a / m) / 2;
    a %= m;
  }
  if (b >= m) {
    ans += n * (b / m);
    b %= m;
  }
  ll y_max = (a * n + b) / m, x_max = (y_max * m - b);
  if (y_max == 0) return ans;
  ans += (n - (x_max + a - 1) / a) * y_max;
  ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
  return ans;
}1
```

1.30 Gaussian Elimination

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef long double ld;
const int N = 505;
const ld EPS = 1e-10;
const int MOD = 998244353;
ll bigMod (ll a, ll e, ll mod) {
  if (e == -1) e = mod - 2;
  ll ret = 1;
  while (e) {
    if (e & 1) ret = ret * a % mod;
    a = a * a % mod, e >>= 1;
  }
  return ret;
}
pair <int, ld> gaussJordan (int n, int m, ld eq[N][N], ld res[N]) {
  ld det = 1;
  vector <int> pos(m, -1);
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (fabs(eq[k][j]) > fabs(eq[piv][j])) piv = k;
    if (fabs(eq[piv][j]) < EPS) continue; pos[j] = i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k], eq[i][k]);
    if (piv ^ i) det = -det; det *= eq[i][j];
    for (int k = 0; k < n; ++k) if (k ^ i) {
      ld x = eq[k][j] / eq[i][j];
      for (int l = j; l <= m; ++l) eq[k][l] -= x * eq[i][l];
    } ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[pos[i]][m] / eq[pos[i]][i];
  }
  for (int i = 0; i < n; ++i) {
    ld cur = -eq[i][m];
    for (int j = 0; j < m; ++j) cur += eq[i][j] * res[j];
    if (fabs(cur) > EPS) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
pair <int, int> gaussJordanModulo (int n, int m, int eq[N][N], int res[N], int mod) {
  int det = 1;
  vector <int> pos(m, -1);
  const ll mod_sq = (ll) mod * mod;
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (eq[k][j] > eq[piv][j]) piv = k;
    if (!eq[piv][j]) continue; pos[j] = i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k], eq[i][k]);
    if (piv ^ i) det = det ? MOD - det : 0; det = (ll) det * eq[i][j] % MOD;
    for (int k = 0; k < n; ++k) if (k ^ i and eq[k][j]) {
      ll x = eq[k][j] * bigMod(eq[i][j], -1, mod) % mod;
      for (int l = j; l <= m; ++l) if (eq[i][l])
        eq[k][l] = (eq[k][l] + mod_sq - x * eq[i][l]) % mod;
    } ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 :
        res[i] = eq[pos[i]][m] * bigMod(eq[pos[i]][i], -1, mod) % mod;
  }
  for (int i = 0; i < n; ++i) {
    ll cur = -eq[i][m];
    for (int j = 0; j < m; ++j) cur += (ll) eq[i][j] * res[j], cur %= mod;
    if (cur) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
pair <int, int> gaussJordanBit (int n, int m, bitset <N> eq[N], bitset <N> &res) {
  int det = 1;
  vector <int> pos(m, -1);
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (eq[k][j]) {
      piv = k; break;
    }
    if (!eq[piv][j]) continue; pos[j] = i,
    swap(eq[piv], eq[i]), det &= eq[i][j];
    for (int k = 0; k < n; ++k) if (k ^ i and eq[k][j]) eq[k] ^= eq[i]; ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 : res[i] = eq[pos[i]][m];
  }
  for (int i = 0; i < n; ++i) {
    int cur = eq[i][m];
    for (int j = 0; j < m; ++j) cur ^= eq[i][j] & res[j];
    if (cur) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}
```

1.31 HLD

```
vector<int> adj[N];
int tin[N], tout[N], sz[N], dep[N], par[N][K], hc[N];
void dfs(int u) {
  tin[u] = t++;
  for (int k = 1; k < K; ++k)
    par[u][k] = par[par[u][k - 1]][k - 1];
  sz[u] = 1, hc[u] = -1;
  for (auto v: adj[u]) {
    if (v == par[u][0])  continue;
    par[v][0] = u;
    dep[v] = dep[u] + 1;
    dfs(v);
    sz[u] += sz[v];
    if (hc[u] == -1 or sz[v] > sz[hc[u]])
      hc[u] = v;
  }
  tout[u] = t++;
}
bool is_anc(int u, int v) {
  return tin[u]<=tin[v] and tout[v]<=tout[u];
}
int get_lca(int u, int v) {
  if (is_anc(u, v))  return u;
  if (is_anc(v, u))  return v;
  for (int k = K - 1; k >= 0; --k)
    if (!is_anc(par[u][k], v))
      u = par[u][k];
  return par[u][0];
}
int idx, in[N], out[N], hd[N];
void hld(int u) {
  in[u] = idx++;
  if (hd[u] == -1)  hd[u] = u;
  if (hc[u] != -1)  hd[hc[u]] = hd[u], hld(hc[u]);
  for (auto v: adj[u]) {
    if (v != par[u][0] and v != hc[u])  hld(v);
  }
  out[u] = ptr - 1;
}
void pupdate(int u, int v, int x) {
  while (hd[u] != hd[v]) {
    if(dep[hd[u]]>dep[hd[v]]) swap(u, v);
    add(in[hd[v]], in[v], x);
    v = par[hd[v]][0];
  }
  if (dep[u] > dep[v]) swap(u, v);
  add(in[u], in[v], x);
  // u is the lca
}
int psum(int u, int v) {
  int ret = 0;
  while (hd[u] != hd[v]) {
    if(dep[hd[u]]>dep[hd[v]]) swap(u, v);
    ret += rsum(in[hd[v]], in[v]);
    v = par[hd[v]][0];
  }
  if (dep[u] > dep[v]) swap(u, v);
  ret += rsum(in[u], in[v]);  // if weight in edges,
  ↪  then exclude lca, by query on (in[u] + 1, in[v])
  // u is the lca
  return ret;
}
// Query from u to root
ll query (int u) {
  ll ret = 0;
  while (1) {
    ret += rsum(pos[hd[u]], pos[u]);
    if (hd[u] == 0) break;
    u = par[hd[u]];
  }
  return ret;
}
void init(int n, int r) {
  par[r] = r;
  for (int u = 0; u < n; ++u) {
    par[u][0] = 0;
  }
  dfs(r);
  idx = 0;
  fill(hd, hd + n, -1);
  hld(r);
}
```

1.32 Hashing

```
const int mod = 1e9 + 7;
const ll P[] = {127, 1000003};
ll p[2][N], inv[2][N];
void init () {
  for (int it = 0; it < 2; ++it) {
    p[it][0] = inv[it][0] = 1;
    ll IP = poww(P[it], -1);
    for (int i = 1; i < N; ++i) {
      p[it][i] = p[it][i - 1] * P[it] % mod;
      inv[it][i] = inv[it][i - 1] * IP % mod;
    }
  }
}
struct RangeHash {
  vector <ll> h[2], rev[2];
  RangeHash (const string s, bool f = 0) {
    for (int it = 0; it < 2; ++it) {
      h[it].resize(s.size() + 1, 0);
      for (int i = 0; i < s.size(); ++i) {
        h[it][i + 1] = (h[it][i] + p[it][i + 1] *
        ↪  s[i]) % mod;
      }
      if (f) {
        rev[it].resize(s.size() + 1, 0);
        for (int i = 0; i < s.size(); ++i) {
          rev[it][i + 1] = (rev[it][i] + inv[it][i +
          ↪  1] * s[i]) % mod;
        }
      }
    }
  }
  inline ll get (int l, int r) {
    ll z = (h[0][r + 1] - h[0][l] + mod) * inv[0][l +
    ↪  1] % mod;
    ll o = (h[1][r + 1] - h[1][l] + mod) * inv[1][l +
    ↪  1] % mod;
    return o << 31 | z;
  }
  inline ll getReverse (int l, int r) {
    ll z = (rev[0][r + 1] - rev[0][l] + mod) * p[0][r
    ↪  + 1] % mod;
    ll o = (rev[1][r + 1] - rev[1][l] + mod) * p[1][r
    ↪  + 1] % mod;
    return o << 31 | z;
  }
};
ll get (string &s) {
  int n = s.size();
  ll z = 0, o = 0;
  for (int i = 0; i < s.size(); ++i) {
    z = (z + p[0][i] * s[i]) % mod;
    o = (o + p[1][i] * s[i]) % mod;
  }
  return o << 31 | z;
}
// Point Update
void update (int i, int x, int u = 1, int s = 0, int e
↪  = n - 1) {
  if (s == e) {
    st[0][u] = x * p[0][i] % mod;
    st[1][u] = x * p[1][i] % mod;
    return ;
  }
  int v = u << 1, w = v | 1, m = (s + e) >> 1;
  if (i <= m) update(i, x, v, s, m);
  else update(i, x, w, m + 1, e);
  st[0][u] = addr(st[0][v], st[0][w]);
  st[1][u] = addr(st[1][v], st[1][w]);
}
array<ll, 2> query (int l, int r, int u = 1, int s =
↪  0, int e = n - 1) {
  if (e < l or r < s) return {0, 0};
  if (l <= s and e <= r) return {st[0][u] * inv[0][l]
  ↪  % mod, st[1][u] * inv[1][l] % mod};
  int v = u << 1, w = v | 1, m = (s + e) >> 1;
  auto ql = query(l, r, v, s, m);
  auto qr = query(l, r, w, m + 1, e);
  ll ret0 = addr(ql[0], qr[0]);
  ll ret1 = addr(ql[1], qr[1]);
  return {ret0, ret1};
}
```

1.33 Hopcroft Karp

```
// 1-based
const int N = 1e5+5, INF = 1e8 + 5;
vector <int> g[N];
int n, e, match[N], dist[N];
bool bfs() {
  queue <int> q;
  for (int i = 1; i <= n; ++i) {
    if (!match[i]) dist[i] = 0, q.emplace(i);
    else dist[i] = INF;
  }
  dist[0] = INF;
  while (!q.empty()) {
    int u = q.front(); q.pop();
    if (!u) continue;
    for (int v : g[u]) {
      if (dist[match[v]] == INF) {
        dist[match[v]] = dist[u] + 1,
        q.emplace(match[v]);
      }
    }
  }
  return dist[0] != INF;
}
bool dfs (int u) {
  if (!u) return 1;
  for (int v : g[u]) {
    if (dist[match[v]] == dist[u] + 1 and
    ↪  dfs(match[v])) {
      match[u] = v, match[v] = u;
      return 1;
    }
  }
  dist[u] = INF;
  return 0;
}
int hopcroftKarp() {
  int ret = 0;
  while (bfs()) {
    for (int i = 1; i <= n; ++i) {
      ret += !match[i] and dfs(i);
    }
  }
  return ret;
}
```

1.34 Hungarian Algorithm

```
template<typename T>
pair<T, vector<int>> MinAssignment(const
↪  vector<vector<T>> &c) {
```

```
  int n = c.size(), m = c[0].size();        // assert(n <= m);
  vector<T> v(m), dist(m);                  // v: potential
  vector<int> L(n, -1), R(m, -1);           // matching pairs
  vector<int> idx(m), prev(m);
  iota(idx.begin(), idx.end(), 0);
  auto residue = [&](int i, int j) { return c[i][j] - v[j]; };
  for (int f = 0; f < n; ++f) {
    for (int j = 0; j < m; ++j) {
      dist[j] = residue(f, j); prev[j] = f;
    }
    T w; int j, l;
    for (int s = 0, t = 0;;) {
      if (s == t) {
        l = s; w = dist[idx[t++]];
        for (int k = t; k < m; ++k) {
          j = idx[k]; T h = dist[j];
          if (h <= w) {
            if (h < w) { t = s; w = h; }
            idx[k] = idx[t]; idx[t++] = j;
          }
        }
        for (int k = s; k < t; ++k) {
          j = idx[k];
          if (R[j] < 0) goto aug;
        }
      }
      int q = idx[s++], i = R[q];
      for (int k = t; k < m; ++k) {
        j = idx[k];
        T h = residue(i,j) - residue(i,q) + w;
        if (h < dist[j]) {
          dist[j] = h; prev[j] = i;
          if (h == w) {
            if (R[j] < 0) goto aug;
            idx[k] = idx[t]; idx[t++] = j;
          }
        }
      }
    }
  aug:
    for(int k = 0; k < l; ++k)
      v[idx[k]] += dist[idx[k]] - w;
    int i;
    do {
      R[j] = i = prev[j];
      swap(j, L[i]);
    } while (i != f);
  }
  T ret = 0;
  for (int i = 0; i < n; ++i)
    ret += c[i][L[i]]; // (i, L[i]) is a solution
  return {ret, L};
}
```

1.35 Interval Container

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$

```
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
 if (L == R) return is.end();
 auto it = is.lower_bound({L, R}), before = it;
 while (it != is.end() && it->first <= R) {
  R = max(R, it->second); before = it = is.erase(it);
 }
 if (it != is.begin() && (--it)->second >= L) {
  L = min(L, it->first); R = max(R, it->second);
  is.erase(it);
 } return is.insert(before, {L,R});
}
void removeInterval(set<pii>& is, int L, int R) {
 if (L == R) return;
 auto it = addInterval(is, L, R); auto r2 = it->second;
 if (it->first == L) is.erase(it); else (int&)it->second = L;
 if (R != r2) is.emplace(R, r2);
}
```

1.36 Interval Cover

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$

```
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
 vi S(sz(I)), R; iota(all(S), 0);
 sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
 T cur = G.first; int at = 0;
 while (cur < G.second) { // (A)
  pair<T, int> mx = make_pair(cur, -1);
  while (at < sz(I) && I[S[at]].first <= cur) {
   mx = max(mx, make_pair(I[S[at]].second, S[at]));
   at++;
  }
  if (mx.second == -1) return {};
  cur = mx.first; R.push_back(mx.second);
 } return R;
}
```

1.37 KMP

```
vector<int> get_pi(string& s){
  int n = s.size();
  vector<int> pi(n);
  for (int k = 0, i = 1; i < n; ++i){
    if(s[i] == s[k])  pi[i] = ++k;
    else if(k == 0) pi[i] = 0;
    else   k = pi[k-1], --i;
 }
 return pi;
}
// KMP DP
int dpi[N][C];
int get_pi(int k, int c) {
  int &ret = dpi[k][c];
  if (ret != -1) return ret;
  if (c == s[k] - 'a') ret = k + 1;
  else if (k == 0) ret = 0;
  else ret = get_pi(pi[k - 1], c);
  return ret;
}
// Period =  n % (n - pi.back() == 0)? n - pi.back(): n
// Borders = pi.back(), pi[pi.back() - 1], ...
// Prefix palindrome: s + "#" + rev(s)
// Number of occurrences of each prefix:
vector<int> pref_occur(vector<int> &pi) {
  int n = pi.size();
  vector<int> pref_occur(n + 1);
  for (int i = 0; i < n; ++i) {
    pref_occur[pi[i]]++;
  }
  for (int len = n; len > 0; --len) {
    pref_occur[pi[len - 1]] += pref_occur[len];
    pref_occur[len]++;
  }
  return pref_occur;
}
// Find the length of the longest proper suffix of a suffix which also its prefix
// Reverse -> Find prefix function -> Reverse
// Find minimum length string such that given strings occur as substring
```

1.38 Knuth Optimization

Description: When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i,j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b,c) \le f(a,d)$ and $f(a,c)+f(b,d) \le f(a,d)+f(b,c)$ for all $a \le b \le c \le d$. Consider also: LineContainer, monotone queues, ternary search.
Time: $\mathcal{O}(N^2)$

```
// Divide an array into n parts.
// Cost of each division is subarray sum
// Minimize the cost
ll dp[n][n], opt[n][n];
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    dp[i][j] = LLONG_MAX;
  }
  opt[i][i] = i;
  dp[i][i] = 0;
}
for (int i = n - 2; i >= 0; --i) {
  for (int j = i + 1; j < n; ++j) {
    for (int k = opt[i][j - 1]; k <= min(j - 1ll, opt[i + 1][j]); ++k) {
      if (dp[i][j] >= dp[i][k] + dp[k + 1][j] + (pref[j + 1] - pref[i])) {
        dp[i][j] = dp[i][k] + dp[k + 1][j] + (pref[j + 1] - pref[i]);
        opt[i][j] = k;
      }
    }
  }
}
cout << dp[0][n - 1] << "\n";
```

1.39 Li Chao Tree

Description: Add line segment, query minimum y at some x. Provide list of all query x points to constructor (offline solution). Use add_segment(line, l,r) to add a line segment $y = ax + b$ defined by $x \in [l, r)$. Use query(x) to get min at x.
Time: $\mathcal{O}(\log n)$

```
struct LiChaoTree {
  using Line = pair <ll, ll>;
  const ll linf = numeric_limits<ll>::max();
  int n; vector<ll> xl; vector<Line> dat;
  LiChaoTree(const vector<ll>& _xl) : xl(_xl) {
    n = 1; while(n < xl.size())n <<= 1;
    xl.resize(n,xl.back());
    dat = vector<Line>(2 * n - 1,Line(0,linf));
  }
  ll eval(Line f,ll x){return f.first * x + f.second;}
  void _add_line(Line f,int k,int l,int r){
    while (l != r) {
      int m = (l + r) / 2;
      ll lx = xl[l],mx = xl[m],rx = xl[r - 1];
```

```
      Line &g = dat[k];
      if(eval(f,lx) < eval(g,lx) && eval(f,rx) <
       ↪ eval(g,rx)){
        g = f; return;
      }
      if(eval(f,lx) >= eval(g,lx) && eval(f,rx) >=
       ↪ eval(g,rx))return;
      if(eval(f,mx) < eval(g,mx))swap(f,g);
      if(eval(f,lx) < eval(g,lx)) k = k * 2 + 1, r = m;
      else k = k * 2 + 2, l = m;
    }
  }
  void add_line(Line f){_add_line(f,0,0,n);}
  void add_segment(Line f,ll lx,ll rx){
    int l = lower_bound(xl.begin(), xl.end(),lx) -
     ↪ xl.begin();
    int r = lower_bound(xl.begin(), xl.end(),rx) -
     ↪ xl.begin();
    int a0 = l, b0 = r, sz = 1; l += n;r += n;
    while(l < r){
      if(r & 1) r--, b0 -= sz, _add_line(f,r - 1,b0,b0
       ↪ + sz);
      if(l & 1) _add_line(f,l - 1,a0,a0 + sz), l++, a0
       ↪ += sz;
      l >>= 1, r >>= 1, sz <<= 1;
    }
  }
  ll query(ll x) {
    int i = lower_bound(xl.begin(), xl.end(),x) -
     ↪ xl.begin();
    i += n - 1; ll res = eval(dat[i],x);
    while (i) i = (i - 1) / 2, res = min(res,
     ↪ eval(dat[i], x));
    return res;
  }
};
```

1.40 Linear Recurrence

Description: Get k-th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \ldots \geq n-1]$ and $tr[0 \ldots n-1]$.

Usage: `linearRec({0, 1}, {1, 1}, k) // k-th Fibonacci number`

Time: $\mathcal{O}\left(n^2 \log k\right)$

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
  int n = tr.size();
  auto combine = [&](Poly a, Poly b) {
    Poly res(n * 2 + 1);
    for (int i = 0; i <= n; ++i)
      for (int j = 0; j <= n; ++j)
        res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
    for (int i = 2 * n; i > n; --i)
      for (int j = 0; j < n; ++j)
        res[i-1-j] = (res[i-1-j] + res[i] * tr[j]) %
         ↪ mod;
    res.resize(n + 1); return res;
  };
  Poly pol(n + 1), e(pol); pol[0] = e[1] = 1;
  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }
  ll res = 0;
  for (int i = 0; i < n; ++i)
    res = (res + pol[i + 1] * S[i]) % mod;
  return res;
}
```

1.41 MST Boruvka

Description: While there are more than one components, Find the closest weight edge that connects this component to any other component and Add this closest edge to MST if not already added.

1.42 Manacher

```
// pal[1][i] = longest odd (half rounded down)
    palindrome around pos i and starts at i - pal[1][i]
 ↪ and ends at i + pal[1][i]
// pal[0][i] = half length of longest even palindrome
    around pos i, i + 1 and starts at i - par[0][i] + 1
 ↪ and ends at i + pal[0][i]
int pal[2][N];
void manacher(string &s) {
  int n = s.size(), idx = 2;
  while (idx--) {
    for (int l=-1, r=-1, i=0; i<n-1; ++i){
      if (i > r) l = r = i;
      else {
        int k = min(r-i, pal[idx][l+r-i]);
        l = i - k,  r = i + k;
      }
      while (l - idx >= 0 and r + 1 < n and s[l - idx]
       ↪ == s[r + 1]) l--, r++;
      pal[idx][i] = r - i;
      // [l - 1 + idx : r] palindrome
  }}}
```

1.43 Matrix Expo

```
using row = vector<int>;
using matrix = vector<row>;
matrix unit_mat(int n) {
  matrix I(n, row(n));
  for (int i = 0; i < n; ++i){
    I[i][i] = 1;
  }
  return I;
}
matrix mat_mul(matrix a, matrix b) {
  int m = a.size(), n = a[0].size();
  int p = b.size(), q = b[0].size();
  // assert(n==p);
  matrix res(m, row(q));
  for (int i = 0; i < m; ++i){
    for (int j = 0; j < q; ++j){
      for (int k = 0; k < n; ++k){
        res[i][j] = (res[i][j] + 1ll *
         ↪ a[i][k]*b[k][j]) % mod;
      }
    }
  }
  return res;
}
matrix mat_exp(matrix a, int p) {
  int m = a.size(), n = a[0].size(); // assert(m==n);
  matrix res = unit_mat(m);
  while (p) {
    if (p&1)  res = mat_mul(a, res);
    a = mat_mul(a, a), p >>= 1;
  }
  return res;
}
```

1.44 Matrix Inverse

Description: Inverts matrix A, stores in A. Returns rank.

Time: $\mathcal{O}\left(n^3\right)$

```
int matInv(vector<vector<ll>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<ll>> tmp(n, vector<ll>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;
  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n) if (A[j][k]) {
      r = j; c = k; goto found;
    }
    return i;
  found:
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n) swap(A[j][i], A[j][c]), swap(tmp[j][i],
     ↪ tmp[j][c]);
    swap(col[i], col[c]);
    ll v = bigMod(A[i][i], -1);
    rep(j,i+1,n) {
      ll f = A[j][i] * v % mod;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) %
       ↪ mod;
      rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k])
       ↪ % mod;
    }
    rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
    rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
    A[i][i] = 1;
  }
  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    ll v = A[j][i];
    rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) %
     ↪ mod;
  }
  rep(i,0,n) rep(j,0,n)
    A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] <
     ↪ 0 ? mod : 0);
  return n;
}
```

1.45 Max Suffix Query

```
## Max Suffix Query
// a1 < a2 < ... < and b1 > b2 > ... > bn
map<ll, ll> mp;
void ins(ll a, ll b) {
  auto it = mp.lb(a);
  if (it != end(mp) && it->s >= b) return;
  it = mp.insert(it, {a, b});  it->s = b;
  while (it != begin(mp) && prev(it)->s <= b)
   ↪ mp.erase(prev(it));
}
// Return max b for for a >= x
ll query(ll x) {
  auto it = mp.lb(x);
  return it == end(mp)? 0: it->s;
}
## Max Prefix Query
// a1 < a2 < ... < and b1 < b2 < ... < bn
map<ll, ll> mp;
void ins(ll a, ll b) {
  auto it = mp.ub(a);
  if (it != begin(mp) and prev(it)->s >= b)  return;
  it = mp.insert(it, {a, b});  it->s = b;
  while (next(it) != end(mp) and next(it)->s <= b)
   ↪ mp.erase(next(it));
}
// Return max b for all a <= x
ll query(ll x) {
  auto it = mp.ub(x);
  return it == begin(mp)? 0: prev(it)->s;
}
```

1.46 Maximum Independent Set

Description: To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertexCover.

1.47 Min Cost Flow

```
struct MCF {
  int n;
  vector<vector<array<ll, 5>>> adj;    // v, pos of u in v, cap, cost, flow
  vector<ll> dis, par, pos;
  MCF(int n): n(n), adj(n), dis(n), par(n), pos(n) {}
  void add_edge(int u, int v, int cap, int cost) {
    adj[u].push_back({v, adj[v].size(), cap, cost, 0});
    adj[v].push_back({u, adj[u].size() - 1, 0, -cost, 0});
  }
  ll spfa(int s, int t) {
    dis.assign(n, INF);
    vector<ll> mn_cap(n, INF), inq(n);
    queue<int> q;
    q.push(s), inq[s] = 1, dis[s] = 0;
    while (!q.empty()) {
      int u = q.front();  q.pop();
      inq[u] = 0;
      for (int i = 0; i < adj[u].size(); ++i) {
        auto [v, idx, cap, cost, flow] = adj[u][i];
        if (cap > flow and dis[v] > dis[u] + cost) {
          dis[v] = dis[u] + cost;
          par[v] = u;
          pos[v] = i;
          mn_cap[v] = min(mn_cap[u], cap - flow);
          q.push(v);
          inq[v] = 1;
        }
      }
    }
    return (mn_cap[t] == INF? 0: mn_cap[t]);
  }
  array<ll, 2> get(int s, int t, ll max_flow = INF) {
    ll flow = 0, mc = 0;
    while (ll f = min(spfa(s, t), max_flow - flow)) {
      flow += f;
      mc += f * dis[t];
      int u = t;
      while (u != s) {
        int p = par[u];
        adj[p][pos[u]][4] += f;
        adj[u][adj[p][pos[u]][1]][4] -= f;
        u = p;
      }
    }
    return {flow, mc};
  }
};
```

1.48 Min Cut

Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s, only traversing edges with positive residual capacity.

1.49 MinRotation

Description: Finds the lexicographically smallest rotation of a string.
Usage: `rotate(v.begin(), v.begin()+minRotation(v), v.end());`
Time: $\mathcal{O}(N)$

```
int minRotation(string s) {
  int a = 0, N = sz(s); s += s;
  rep(b, 0, N) rep(k, 0, N) {
    if (a + k == b || s[a + k] < s[b + k]) {b += max(0, k - 1); break;}
    if (s[a + k] > s[b + k]) { a = b; break; }
  } return a;
}
```

1.50 Minimum Vertex Cover

Description: Suppose you have left and right parts in bipartite graph and you found the maximum matching M. Let's define orientation of edges. Those edges that belong to M will go from right to left, all other edges will go from left to right. Now run DFS starting at all left vertices that are not incident to edges in M. Some vertices of the graph will become visited during this DFS and some not-visited. To get minimum vertex cover you take all visited right vertices of M, and all not-visited left vertices of M.

1.51 Misc

```
## Build Command
g++ -std=c++17 -Wshadow -Wall -o t t.cpp -g -fsanitize=address -fsanitize=undefined -D_GLIBCXX_DEBUG && ./t <in> out
## Pragmas
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("O3,unroll-loops,Ofast,fast-math")
#pragma GCC target("avx,avx2,fma")
#pragma GCC optimize("O3", "unroll-loops")
#pragma GCC target("avx2", "popcnt")
## bitset
BS._Find_first()
BS._Find_next(x) //Return first set bit after xth bit, x on failure
## Gray Code, G(0) = 000, G(1) = 001, G(2) = 011, G(3) = 010
inline int g(int n) { return n ^ (n >> 1); }
## Inverse Gray Code
int rev_g(int g) {
  int n = 0;
  for ( ; g; g >>= 1) n ^= g;
  return n;
}
## Only for non-negative integers
## Returns the immediate next number with same count of one bits, -1 on failure
long long hakmemItem175(long long n) {
  if (!n) return -1;
  long long x = (n & -n);
  long long left = (x + n);
  long long right = ((n ^ left) / x) >> 2;
  long long res = (left | right);
  return res;
}
## Returns the immediate previous number with same count of one bits, -1 on failure
long long lol(long long n) {
  if (n < 2) return -1;
  long long res = ~hakmemItem175(~n);
  return (!res) ? -1 : res;
}
int __builtin_clz(int x);// number of leading zero
int __builtin_ctz(int x);// number of trailing zero
int __builtin_clzll(long long x);// number of leading zero
int __builtin_ctzll(long long x);// number of trailing zero
int __builtin_popcount(int x);// number of 1-bits in x
int __builtin_popcountll(long long x);// number of 1-bits in x
lsb(n): (n & -n); // last bit (smallest)
floor(log2(n)): 31 - __builtin_clz(n | 1);
floor(log2(n)): 63 - __builtin_clzll(n | 1);
## compute next perm. ex) 00111, 01011, 01101, 01110, 10011, 10101..
long long next_perm(long long v){
  long long t = v | (v-1);
  return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1));
}
## Next Combination Mask
int next_combs_mask(int mask) {
  int lsb = -mask & mask;
  return (((mask + lsb) ^ mask) / (lsb << 2)) | (mask + lsb);
}
## Iterate over submask in decreasing order
for (int s=mask; s > 0; s = (s-1)&mask) {}
## GNU Pbds
#include <ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using oset = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
template <typename T, typename R> using omap = tree<T, R, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
## unordered map
struct chash{
  size_t operator()(const pair<int,int>&x)const{
    return hash<long long>()(((long long)x.first)^(((long long)x.second)<<32));
  }
};
struct chash {
  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_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};
gp_hash_table<ii, int, chash> cnt;
## set custom operator
struct comp {
  bool operator()(const int& a, const int& b) const {
    return a < b;
  }
};
set<int, comp> st;
priority_queue<int, vector<int>, comp> pq;
## Random Number
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int x = rng() % 495;
## Running time
clock_t st = clock();
double t = (clock() - st) / (1.0 * CLOCKS_PER_SEC);
string line;  getline(cin, line);
istringstream iss(line);
string word;
while (iss >> word) {
  cout << word << "\n";
```

```
}
merge(t[v].begin(), t[v].end(), t[w].begin(),
 ↪  t[w].end(), back_inserter(t[u]));
ulimit -s 65532
## Rollback
vector<pair<int*, int>> cng[N];
cng[i].push_back({&x, x});
for (auto [x, y]: cng[i]) {
  *x = y;
}
```

1.52 Mo On Tree

Description: Build Euler order of $2N$ size - write node ID when entering AND exiting. Path (u,v) with $\text{in}[u] < \text{in}[v]$ is now range. If u is LCA, then range is $[\text{in}[u], \text{in}[v]]$. If not, then range is $[\text{out}[u], \text{in}[v]] \cup [\text{in}[LCA], \text{in}[LCA]]$. Nodes that appear exactly once (not 0 or 2 times) on these ranges are relevant, maintain them during Mo.

Time: $\mathcal{O}\left(N\sqrt{Q}\right)$

1.53 Mo With Updates

Description: Sort queries by tuple $(\lfloor l/B \rfloor, \lfloor r/B \rfloor, t)$ where t is the time of query. If DS has answer for $[L,R]$ at time T now, then we have to adjust range (add/remove like usual Mo). For time, we need to make some updates if $t < T$ and rollback some updates if $t > T$.

Time: $\mathcal{O}\left(QN^{2/3}\right)$

1.54 Mo's Algorithm

```
vector<array<int, 4>> cu(m);
for (int i = 0; i < m; ++i) {
  auto &[b, l, r, idx] = cu[i];
  cin >> l >> r;  l--;
  b = r / B;
  idx = i;
}
sort(cu.begin(), cu.end());
int s = 0, e = -1;
for (auto [b, l, r, i]: cu) {
  while (l < s)  add(--s);
  while (e < r)  add(++e);
  while (s < l)  remove(s++);
  while (r < e)  remove(e--);
  ans[i] = cur_ans;
}
```

1.55 NTT

```
const int G = 3;
const int MOD = 998244353;
const int N = (1 << 20) + 5;
int rev[N], w[N], inv_n;
void prepare (int n) {
  int sz = abs(31 - __builtin_clz(n));
  int r = bigMod(G, (MOD - 1) / n, MOD);
  inv_n = bigMod(n, MOD - 2, MOD), w[0] = w[n] = 1;
  for (int i = 1; i < n; ++i) w[i] = (ll) w[i - 1] * r
   ↪  % MOD;
  for (int i = 1; i < n; ++i) rev[i] = (rev[i >> 1] >>
   ↪  1) | ((i & 1) << (sz - 1));
}
void ntt (int *a, int n, int dir) {
  for (int i = 1; i < n - 1; ++i) {
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  }
  for (int m = 2; m <= n; m <<= 1) {
    for (int i = 0; i < n; i += m) {
      for (int j = 0; j < (m >> 1); ++j) {
        int &u = a[i + j], &v = a[i + j + (m >> 1)];
        int t = (ll) v * w[dir ? n - n / m * j : n / m
         ↪  * j] % MOD;
        v = u - t < 0 ? u - t + MOD : u - t;
        u = u + t >= MOD ? u + t - MOD : u + t;
      }
    }
  } if (dir) for (int i = 0; i < n; ++i) a[i] = (ll)
   ↪  a[i] * inv_n % MOD;
}
int f_a[N], f_b[N];
vector <int> multiply (vector <int> a, vector <int> b)
 ↪  {
  int sz = 1, n = a.size(), m = b.size();
  while (sz < n + m - 1) sz <<= 1; prepare(sz);
  for (int i = 0; i < sz; ++i) f_a[i] = i < n ? a[i] :
   ↪  0;
  for (int i = 0; i < sz; ++i) f_b[i] = i < m ? b[i] :
   ↪  0;
  ntt(f_a, sz, 0); ntt(f_b, sz, 0);
  for (int i = 0; i < sz; ++i) f_a[i] = (ll) f_a[i] *
   ↪  f_b[i] % MOD;
  ntt(f_a, sz, 1); return vector <int> (f_a, f_a + n +
   ↪  m - 1);
}
// G = primitive_root(MOD)
int primitive_root (int p) {
  vector <int> factor;
  int tmp = p - 1;
  for (int i = 2; i * i <= tmp; ++i) {
    if (tmp % i == 0) {
      factor.emplace_back(i);
      while (tmp % i == 0) tmp /= i;
    }
  }
  if (tmp != 1) factor.emplace_back(tmp);
  for (int root = 1; ; ++root) {
    bool flag = true;
    for (int i = 0; i < (int) factor.size(); ++i) {
      if (bigMod(root, (p - 1) / factor[i], p) == 1) {
        flag = false; break;
      }
    }
    if (flag) return root;
  }
}
```

1.56 Number Theory

```
## Floor
ll floor (ll n, ll k) {
  if (n >= 0)  return n / k;
  return  (n - (k - 1)) / k;
}
## Ceil
ll ceil (ll n, ll k) {
  if (n >= 0)  return (n + k - 1) / k;
  return n / k;
}
## Modular Inverse O(N)
inv[1] = 1;
for(int i = 2; i < N; ++i) {
  inv[i] = -(mod / i) * inv[mod % i] % mod;
  inv[i] += mod;
}
## Harmonic Lemma (ceill)
ll i = 1;
while (i < n) {
  ll cval = (n + i - 1) / i;
  ll j = (n + cval - 2) / (cval - 1);
  // ceil(n/i)...ceil(n/(j - 1)) = cval
  i = j;
}
ll egcd(ll a, ll b, ll &x, ll &y){
  if(b == 0) { x=1, y=0; return a;}
  ll g = egcd(b, a%b, y, x);
  y -= a/b*x;  return g;
}
ll mod_inv(ll a, ll m){
  ll x, y;
  ll g = egcd(a, m, x, y);
  if(g != 1)  return -1;  //no solution
  return (x%m+m)%m;
}
## Linear-sieve
int lpf[N], pm[N], pcnt = 0;
for (int i = 2; i < N; ++i) {
  if (!lpf[i])  lpf[i] = i, pm[pcnt++] = i;
  for (int j = 0; j < sz; ++j) {
    int p = pm[j];
    if (lpf[i] < p or i * p >= N)  break;
    lpf[i * p] = p;
  }
}
## Miller-Rabin
bool isp(ll n){
  if(n==2 || n == 3)  return 1;
  if(n<=1 || n%2==0)  return 0;
  for (int k = 0; k < 10; ++k){
    ll a = 2+rand()%(n-2);
    ll s = n-1;
    while(!(s&1)) s>>=1;
    if(powmod(a, s, n) == 1)  continue;
    int iscomp = 1;
    while(s!=n-1){
      if(powmod(a, s, n)==n-1){
        iscomp = 0;
        break;
      }
      s=s<<1;
    }
    if(iscomp) return 0;
  }
  return 1;
}
## Miller-Rabin Deterministic:
bool check_composite(u64 n, u64 a, u64 d, int s) {
  u64 x = binpower(a, d, n);
  if (x == 1 || x == n - 1)
    return false;
  for (int r = 1; r < s; r++) {
    x = (u128)x * x % n;
    if (x == n - 1)
      return false;
  }
  return true;
}
bool isp(u64 n) {
  if (n < 2)
    return false;
  int r = 0;
  u64 d = n - 1;
  while ((d & 1) == 0) {
    d >>= 1;
    r++;
  }
  for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
   ↪  31, 37}) {
    if (n == a)
      return true;
    if (check_composite(n, a, d, r))
      return false;
  }
```

```
  return true;
}
## Prime Factorize of large number(Pollard Rho):
ll f(ll x, ll c, ll n){
  return (mulmod(x,x,n)+c)%n;
}
ll pollard_rho(ll n){
  if(n == 1)  return 1;
  if(n%2 == 0)  return 2;
  ll x = rand()%(n-2)+2;
  ll y = x;
  ll c = rand()%(n-1)+1;
  ll g = 1;
  while (g == 1){
    x = f(x, c, n);
    y = f(y, c, n);
    y = f(y, c, n);
    g = __gcd(abs(x-y), n);
  }
  return g;
}
vector<ll> prime_factorize(ll n){
  if(n<=1)  return vector<ll>();
  if(isp(n))  return vector<ll> ({n});
  ll d = pollard_rho(n);
  vector<ll> v = factorize(d);
  vector<ll> w = factorize(n/d);
  v.insert(v.end(), w.begin(), w.end());
  sort(v.begin(), v.end());
  return v;
}
// auto pf = prime_factorize(n);
## Number of divisors of n O(n^1/3):
int nod(ll n){
  sieve();
  int ret = 1;
  for (int i = 2; 1LL*i*i*i <= n; ++i){
    if(isp[i]){
      int e = 0;
      while(n%i == 0){
        e++;
        n /= i;
      }
      ret *= e+1;
    }
  }
  ll sq = sqrt(1.0L*n);
  if(isprime(n))  ret *= 2;
  else if(n == sq*sq and isprime(sq))  ret *= 3;
  else if(n!=1) ret *= 4;
  return ret;
}
## Smallest inverse phi
ll inv_phi(ll phi, ll n, int pc) {
  if (phi == 1)  return n;
  if (pc == -1)  return INF;
  ll ret = inv_phi(phi, n, pc - 1);
  if (phi % (p[pc] - 1) == 0) {
    phi /= (p[pc] - 1);
    n = n / (p[pc] - 1) * p[pc];
    while (phi % p[pc] == 0) {
      phi /= p[pc];
    }
    ret = min(ret, inv_phi(phi, n, pc - 1));
  }
  return ret;
}
ll phi;  cin >> phi;
if (phi & 1) {
  cout << (phi == 1) << "\n";
}
else {
  for (int i = 1; i * i <= phi; ++i) {
    if (phi % i == 0) {
      if (isp(i + 1)) {
        p.push_back(i + 1);
      }
      if (i * i != phi and isp(phi / i + 1)){
        p.push_back(phi / i + 1);
      }
    }
  }
  sort(p.begin(), p.end());
  ll ans = inv_phi(phi, phi, p.size() - 1);
  cout << (ans == INF? 0: ans) << "\n";
}
## Count elements ai s.t. gcd(x, ai) = g
for (auto d: divs[x / g]) {
  f += mu[d] * cnt[d * g];
}
## CRT
ll crt(ll r1, ll m1, ll r2, ll m2){
  if(m1<m2) swap(r1, r2), swap(m1, m2);
  ll p, q, g = egcd(m1, m2, p, q);
  if((r2-r1)%g !=0 )  return -1;  //no solution
  ll x = (r2-r1)%m2*p%m2*m1/g + r1;
  return x<0? x+m1*m2/g: x;
}
ll crt(vector<ll>& r, vector<ll>& m){
  ll x = r[0], M=m[0];
  for (int i = 1; i < r.size(); ++i){
    x = crt(x, M, r[i], m[i]);
    ll g = __gcd(M, m[i]);
    M = (M/g)*(m[i]/g);
  }
  return x;
}
## Discrete Logarithm
ll discrete_log(ll a, ll b, ll m) {
  a %= m, b %= m;
  if(a == 0){
    return (b == 0? 1: -1);
  }
  ll k = 1, add = 0, g;
  while ((g = __gcd(a, m)) > 1) {
    if (b == k)  return add;
    if (b % g)  return -1;
    b /= g, m /= g, k = (k * a / g) % m, ++add;
  }
  int n = sqrt(m) + 1;
  unordered_map<int, int> vals;
  for (ll q = 0, cur = b; q <= n; ++q) {
    vals[cur] = q;
    cur = (cur * a) % m;
  }
  ll an = 1;
  for (int i = 0; i < n; ++i) {
    an = (an * a) % m;
  }
  for (ll p = 1, cur = k; p <= n; ++p) {
    cur = (cur * an) % m;
    if (vals.count(cur)) {
      return n * p - vals[cur] + add;
    }
  }
  return -1;
}
## Primitive Root
int get_gen(int p) {
  int n = p - 1;
  vector<int> pfs;
  for (int i = 2; i * i <= n; ++i) {
    if (n % i == 0) {
      pfs.push_back(i);
      while (n % i == 0) n /= i;
    }
  }
  if (n > 1) pfs.push_back(n);
  n = p - 1;
  for (int g = 2; g < p; ++g) {
    int ok = 1;
    for (auto pf: pfs) {
      if (poww(g, n / pf, p) == 1) {
        ok = 0;
        break;
      }
    }
    if (ok) return g;
  }
  return -1;
}
```

1.57 Online FFT

Description: $a_1, a_2, \ldots, a_{n-1}$ and b_0 are given, find $b_1, b_2, \ldots, b_{n-1}$, where $b_i = \sum_{j=1}^{i} a_j \cdot b_{i-j}$

Time: $\mathcal{O}\left(n \log n^2\right)$

```
for (int i = 1; i < n; ++i) {
  for (int p = 1; (i & (p - 1)) == 0; p <<= 1) {
    vector<int> aa(a + p, a + min(p << 1, n));
    vector<int> bb(b + i - p, b + i);
    auto c = mul(aa, bb);
    for (int j = 0; j < c.size(); ++j) {
      if (i + j >= n)  break;
      add(b[i + j], c[j]);
    }
  }
}
```

1.58 Palindromic Tree

```
int to[N][A], len[N], lnk[N], u, cnt;
int node[N], occ[N], dep[N];
void init() {
  while (cnt >= 0) {
    memset(to[cnt], 0, sizeof(to[cnt]));
    occ[cnt] = 0; cnt--;
  }
  len[1] = -1; lnk[1] = lnk[2] = 1;
  u = cnt = 2;
}
void add(int i) {
  while (s[i-1-len[u]] != s[i]) u=lnk[u];
  int c = s[i] - 'a', v = lnk[u];
  while (s[i-1-len[v]] != s[i]) v=lnk[v];
  if (!to[u][c]) {
    to[u][c] = ++cnt;
    len[cnt] = len[u] + 2;
    lnk[cnt] = len[cnt] == 1? 2: to[v][c];
    dep[cnt] = dep[lnk[cnt]] + 1;
  }
  u = to[u][c]; node[i] = u; occ[u]++;
}
s = " " + s;
init();
for (int i = 1; i < s.size(); ++i) add(i);
for (int u = cnt; u > 2; --cnt) {
  occ[lnk[u]] += occ[u];
}
// The number of palindromic substrings end at i-th
↪ position = dep[node[i]]
```

1.59 Persistent Segment Tree

```
const int N = 2e5+5, K = 4 + (1 + __lg(Q)); // Q for
↪ number of times add() is called
int n, a[N], L[K * N], R[K * N], cur;
ll st[K * N];
```

```cpp
int copy(int u) {
  ++cur; st[cur] = st[u];
  L[cur] = L[u]; R[cur] = R[u];
  return cur;
}
int build(int s = 0, int e = n - 1) {
  int u = ++cur;
  if (s == e) { st[u] = a[s]; return u;}
  int m = s + e >> 1;
  L[u]=build(s, m); R[u]=build(m + 1, e);
  st[u] = st[L[u]] + st[R[u]]; return u;
}
int add(int i, ll x, int u, int s = 0, int e = n - 1) {
  u = copy(u);
  if (s == e) { st[u] += x; return u; }
  int &v = L[u], &w = R[u], m = s + e >> 1;
  if (i <= m) v = add(i, x, v, s, m);
  else w = add(i, x, w, m + 1, e);
  st[u] = st[v] + st[w]; return u;
}
ll rsum(int l, int r, int u, int s = 0, int e = n - 1)
  {
  if (e < l or r < s) return 0;
  if (l <= s and e <= r) return st[u];
  int v = L[u], w = R[u], m = s + e >> 1;
  return rsum(l, r, v, s, m) + rsum(l, r, w, m + 1, e);
}
// Return count of [l...r] in (ul...ur] subarray
int rcnt(int l, int r, int ul, int ur, int s = 0, int
  e = n - 1) {
  if (l > r or e < l or r < s) return 0;
  if (l <= s and e <= r) return st[ur] - st[ul];
  int m = s + e >> 1;
  return rcnt(l, r, L[ul], L[ur], s, m) + rcnt(l, r,
  R[ul], R[ur], m + 1, e);
}
// Return kth smallest number in [l...r] subarray
int kth(int k, int ul, int ur, int s = 0, int e = n -
  1) {
  if (s == e) return s;
  int m = s + e >> 1;
  int x = st[L[ur]] - st[L[ul]];
  if (x >= k) return kth(k, L[ul], L[ur], s, m);
  else return kth(k - x, R[ul], R[ur], m + 1, e);
}
// while finding kth smallest number among union of
  some disjoint subarrays
// Specially, while finding kth smallest number in a
  path using HLD
int kth(int k, vector<array<int, 2>> u, int s = 0, int
  e = n - 1) {
  if (s == e) return s;
  int m = s + e >> 1;
  int sz = u.size(), x = 0;
  vector<array<int, 2>> v(sz), w(sz);
  for (int i = 0; i < sz; ++i) {
    v[i] = {L[u[i][0]], L[u[i][1]]};
    w[i] = {R[u[i][0]], R[u[i][1]]};
    x += st[v[i][1]] - st[v[i][0]];
  }
  if (x >= k) return kth(k, v, s, m);
  else return kth(k - x, w, m + 1, e);
}
// With Lazy
const int N = 1e5+5, K = 4 + 4 * (__lg(N) + 1);
int n, a[N], L[K * N], R[K * N], cur;
ll st[K * N], lz[K * N];
int copy(int u) {
  st[++cur] = st[u]; lz[cur] = lz[u];
  L[cur] = L[u]; R[cur] = R[u];
  return cur;
}
void push(int u, int s, int e) {
  if (!lz[u])  return ;
  int v = L[u], w = R[u], m = s + e >> 1;
  st[v] += (m - s + 1) * lz[u];
  st[w] += (e - m) * lz[u];
  lz[v]+=lz[u]; lz[w]+=lz[u]; lz[u]=0;
}
int build(int s = 0, int e = n - 1) {
  int u = ++cur;
  if (s == e) { st[u] = a[s]; return u;}
  int m = s + e >> 1;
  L[u] = build(s, m); R[u] = build(m + 1, e);
  st[u] = st[L[u]] + st[R[u]]; return u;
}
void add(int l, int r, ll x, int u, int s = 0, int e =
  n - 1) {
  if (e < l or r < s) return;
  if (l <= s and e <= r) {
    st[u] += (e - s + 1) * x;
    lz[u] += x; return; }
  int &v = L[u], &w = R[u], m = s + e >> 1;
  v = copy(v), w = copy(w);
  push(u, s, e); add(l,r,x,v,s,m);
  add(l,r,x,w,m+1,e); st[u]=st[v]+st[w];
}
ll rsum(int l, int r, int u, int s = 0, int e = n - 1)
  {
  if (e < l or r < s) return 0;
  if (l <= s and e <= r) return st[u];
  int &v = L[u], &w = R[u], m = s + e >> 1;
  if (lz[u]){ v=copy(v), w=copy(w);
    push(u, s, e); }
  return rsum(l, r, v, s, m) + rsum(l, r, w, m + 1, e);
}
cur = 0; root[0] = build();
root[i + 1] = copy(root[i]);
add(l, r, x, root[i + 1])
cout << rsum(l, r, ver[i]) << "\n";
```

1.60 Persistent Trie

```cpp
int trie[N * IDX][2], cnt[N * IDX], root[N], tot = 1;
void init() {
  cnt[tot]=0; root[0]=tot;
}
int add(int u, int x) {
  int uu = ++tot; int ret = uu;
  cnt[u] = cnt[uu];
  for (int idx = IDX - 1; idx >= 0; --idx) {
    int f = x >> idx & 1;
    trie[uu][!f] = trie[u][!f];
    trie[uu][f] = ++tot;
    uu = trie[uu][f]; u = trie[u][f];
    cnt[uu] = cnt[u] + 1;
  }
  return ret;
}
int max_xor(int u, int x) {
  int ret = 0;
  for (int idx = IDX - 1; idx >= 0; --idx) {
    int f = x >> idx & 1;
    if (cnt[trie[u][!f]])  ret |= 1 << idx, u =
      trie[u][!f];
    else  u = trie[u][f];
  }
  return ret;
}
```

1.61 Polynomial Interpolation

```cpp
// P(x) = a0 + a1x + a2x^2 + ... + anx^n
// y[i] = P(i)
ll eval (vector<ll> y, ll k) {
  int n = y.size() - 1;
  if (k <= n) {
    return y[k];
  }
  vector<ll> L(n + 1, 1);
  for (int x = 1; x <= n; ++x) {
    L[0] = L[0] * (k - x) % mod;
    L[0] = L[0] * inv(-x) % mod;
  }
  for (int x = 1; x <= n; ++x) {
    L[x] = L[x - 1] * inv(k - x) % mod * (k - (x - 1))
      % mod;
    L[x] = L[x] * ((x - 1) - n + mod) % mod * inv(x) %
      mod;
  }
  ll yk = 0;
  for (int x = 0; x <= n; ++x) {
    yk = add(yk, L[x] * y[x] % mod);
  }
  return yk;
}
```

1.62 SCC

```cpp
void dfs1(int u, vector<int> *adj, vector<int> &vis,
  vector<int> &order) {
  vis[u] = 1;
  for (int &v: adj[u]) {
    if (!vis[v]) {
      dfs1(v, adj, vis, order);
    }
  }
  order.emplace_back(u);
}
void dfs2(int u, vector<int> *rev_adj, vector<int>
  &vis, vector<int> &scc) {
  scc.emplace_back(u);
  vis[u] = 1;
  for (int &v: rev_adj[u]) {
    if (!vis[v]) {
      dfs2(v, rev_adj, vis, scc);
    }
  }
}
vector<vector<int>> get_sccs(int n, vector<int> *adj) {
  vector<int> vis(n), order;
  for (int u = 0; u < n; ++u) {
    if (!vis[u]) {
      dfs1(u, adj, vis, order);
    }
  }
  vector<int> rev_adj[n];
  for (int u = 0; u < n; ++u) {
    for (int v: adj[u]) {
      rev_adj[v].emplace_back(u);
    }
  }
  vector<vector<int>> sccs;
  reverse(order.begin(), order.end());
  vis.assign(n, 0);
  for (int u: order) {
    if (!vis[u]) {
      sccs.emplace_back(0);
      dfs2(u, rev_adj, vis, sccs.back());
    }
  }
  return sccs;
}
vector<vector<int>> sccs = get_sccs(n, adj);
int tot_scc = sccs.size();
vector<int> scc_no(n);
for (int i = 0; i < tot_scc; ++i) {
  for (int u: sccs[i]) {
    scc_no[u] = i;
  }
}
```

1.63 SOS DP

```
// f(i, mask) = Sum over subsets of mask which are identical to the mask considering the bits from MSB to the (i + 1)th least significant bit.
## Count Over Subset
// f(i, mask) = f(i - 1, mask)  // ith bit = 0
// f(i, mask) = f(i - 1, mask) + f(i - 1, mask ^ (1 << i)) // ith bit = 1
for (int i = 0; i < K; ++i) {
  for (int mask = MASK - 1; mask >= 0; --mask) {
    if (mask >> i & 1) {
      dp[mask] += dp[mask ^ 1 << i];
    }
  }
}
## Count Over Superset
// f(i, mask) = f(i - 1, mask) + f(i - 1, mask ^ (1 << i)) // ith bit = 0
// f(i, mask) = f(i - 1, mask)  // ith bit = 1
for (int i = 0; i < K; ++i) {
  for (int mask = 0; mask < MASK; ++mask) {
    if (mask >> i & 1 ^ 1) {
      dp[mask] += dp[mask ^ 1 << i];
    }
  }
}
## Count Over Disjoint-set
  - Sum over submask of its complement
## Number of subsequences of with bitwise OR = mask
for (int i = 0; i < K; ++i) {
  for (int mask = MASK - 1; mask >= 0; --mask) {
    if (mask >> i & 1) {
      dp[mask] += dp[mask ^ 1 << i];
    }
  }
}
for (int mask = 0; mask < MASK; ++mask) {
  dp[mask] = two[dp[mask]];
}
for (int i = 0; i < K; ++i) {
  for (int mask = MASK - 1; mask >= 0; --mask) {
    if (mask >> i & 1) {
      sub(dp[mask], dp[mask ^ 1 << i]);
    }
  }
}
// Now, dp[mask] = number of subsequences with OR = mask
## Number of subsequences of with bitwise AND = mask
- Flip the bits, then find number of subsequences with bitwise OR
```

1.64 Segment Tree Beats

Description: For update $A_i \to A_i \bmod x$ and similar, keep range min, max in node and lazily update whenever min = max. For update $A_i \to \min(A_i, x)$ and similar, keep range max, second max in node and lazily update whenever $x >$ second max.

Time: $\mathcal{O}(\log^2 N)$, $\mathcal{O}(\log N)$

1.65 Segment Tree

```
## Range Minimum Query
// 0-indexed, point updates, commutative merge
namespace Special {
  int n, a[N], tree[N << 1];
  void init() {
    for (int i = 0; i < n; ++i) {
      tree[n + i] = a[i];
    }
    for (int i = n - 1; i >= 0; --i) {
      tree[i] = min(tree[i << 1], tree[i << 1 | 1]);
    }
  }
  // assign a[p] = v
  void update (int p, int v) {
    for (tree[p += n] = v; p > 1; p >>= 1) {
      tree[p >> 1] = min(tree[p], tree[p ^ 1]);
    }
  }
  // range [l, r) sum
  int query (int l, int r) {
    int ret = INT_MAX;
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) ret = min(ret, tree[l++]);
      if (r & 1) ret = min(ret, tree[--r]);
    }
    return ret;
  }
}
## 1-indexed, range updates, no restriction on merge
namespace General {
  int n, a[N], tree[N << 2], lazy[N << 2];
  void init (int u = 1, int b = 1, int e = n) {
    lazy[u] = 0;
    if (b == e) return void(tree[u] = a[b]);
    int mid = b + e >> 1;
    init(u << 1, b, mid), init(u << 1 | 1, mid + 1, e);
    tree[u] = min(tree[u << 1], tree[u << 1 | 1]);
  }
  inline void push (int u, int b, int e) {
    tree[u] += lazy[u];
    if (b ^ e) lazy[u << 1] += lazy[u], lazy[u << 1 | 1] += lazy[u];
    lazy[u] = 0;
  }
  // add v on range [l, r]
  void update (int l, int r, int v, int u = 1, int b = 1, int e = n) {
    if (lazy[u]) push(u, b, e);
    if (b > r or e < l) return;
    if (b >= l and e <= r) {
      lazy[u] += v;
      return push(u, b, e);
    }
    int mid = b + e >> 1;
    update(l, r, v, u << 1, b, mid), update(l, r, v, u << 1 | 1, mid + 1, e);
    tree[u] = min(tree[u << 1], tree[u << 1 | 1]);
  }
  // range [l, r] sum
  int query (int l, int r, int u = 1, int b = 1, int e = n) {
    if (b > r or e < l) return INT_MAX;
    if (lazy[u]) push(u, b, e);
    if (b >= l and e <= r) return tree[u];
    int mid = b + e >> 1;
    return min(query(l, r, u << 1, b, mid), query(l, r, u << 1 | 1, mid + 1, e));
  }
}
```

1.66 Sparse Table

```
## 1D
void build() {
  for(int i = 0; i < n; ++i) st[i][0]=a[i];
  for(int k = 0; k + 1 < K; ++k)
    for(int i = 0; i + (2 << k) <= n; ++i)
      st[i][k + 1] = min(st[i][k], st[i + (1 << k)][k]);
}
ll rmq(int l, int r) {
  int k = lg[r - l + 1];
  return min(st[l][k], st[r - (1 << k) + 1][k]);
}
## 2D
int st[N][N][LG][LG];
int a[N][N], lg2[N];
int yo(int x1, int y1, int x2, int y2) {
  x2++;
  y2++;
  int a = lg2[x2 - x1], b = lg2[y2 - y1];
  return max(
        max(st[x1][y1][a][b], st[x2 - (1 << a)][y1][a][b]),
        max(st[x1][y2 - (1 << b)][a][b], st[x2 - (1 << a)][y2 - (1 << b)][a][b])
      );
}
void build(int n, int m) { // 0 indexed
  for (int i = 2; i < N; i++) lg2[i] = lg2[i >> 1] + 1;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      st[i][j][0][0] = a[i][j];
    }
  }
  for (int a = 0; a < LG; a++) {
    for (int b = 0; b < LG; b++) {
      if (a + b == 0) continue;
      for (int i = 0; i + (1 << a) <= n; i++) {
        for (int j = 0; j + (1 << b) <= m; j++) {
          if (!a) {
            st[i][j][a][b] = max(st[i][j][a][b - 1], st[i][j + (1 << (b - 1))][a][b - 1]);
          } else {
            st[i][j][a][b] = max(st[i][j][a - 1][b], st[i + (1 << (a - 1))][j][a - 1][b]);
          }
        }
      }
    }
  }
}
```

1.67 Stirling

Description:

- Number of permutations of n elements with k disjoint cycles = Str1(n,k) = (n-1) * Str1(n-1,k) + Str1(n-1,k-1).
- Str1(0,0) = 1
- $\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\dots(x+n-1)$
- $Str1(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
- $Str1(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots$
- n! = Sum(Str1(n,k)) (for all 0 <= k <= n).
- Ways to partition n labelled objects into k unlabelled subsets = Str2(n,k) = k * Str2(n-1,k) + Str2(n-1,k-1).
- $Str2(n,1) = Str2(n,n) = 1$
- $Str2(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$
- Parity of Str2(n,k) : ((n-k) & Floor((k-1)/2)) == 0).
- Ways to partition n labelled objects into k unlabelled subsets, with each subset containing at least r elements: SR(n,k) = k * SR(n-1,k) + C(n-1,r-1) * SR(n-r,k-1).
- Number of ways to partition n labelled objects 1,2,3, ... n into k non-empty subsets so that for any integers i and j in a given subset |i-j| >= d: Str2(n-d+1, k-d+1), n >= k >= d.
- Number of ways to color a $1 \times n$ grid using k colors such that each color is used at least once = $k!.Str2(n,k)$
- Denote the n objects to partition by the integers $1, 2, \dots, n$. Define the reduced Stirling numbers of the second kind, denoted $S^d(n,k)$, to be the number of ways to partition the integers

1,2,…,n into k nonempty subsets such that all elements in each subset have pairwise distance at least d. That is, for any integers i and j in a given subset, it is required that $|i-j| \ge d$. It has been shown that these numbers satisfy, $S^d(n,k) = S(n-d+1, k-d+1), \quad n \ge k \ge d$.

```cpp
// O(k*log(n))
ll get_sn2(int n, int k) {
  ll sn2 = 0;
  for (int i = 0; i <= k; ++i) {
    ll now = nCr(k, i) * poww(k - i, n, mod) % mod;
    if (i & 1) now = now * (mod - 1) % mod;
    add(sn2, now);
  }
  sn2 = sn2 * ifact[k] % mod;
  return sn2;
}
NTT ntt(mod);
vector<ll> v[MAX];
//Stirling1 (n,k) = co-eff of x^k in
   x*(x+1)*(x+2)*....(x+n-1)
int Stirling1(int n, int r) {
  int nn = 1;
  while (nn < n) nn <<= 1;
  for (int i = 0; i < n; ++i) {v[i].push_back(i);
    v[i].push_back(1);}
  for (int i = n; i < nn; ++i) v[i].push_back(1);
  for (int j = nn; j > 1; j >>= 1) {
    int hn = j >> 1;
    for (int i = 0; i < hn; ++i) ntt.multiply(v[i],
      v[i + hn], v[i]);
  }
  return v[0][r];
}
NTT ntt(mod);
vector<ll> a, b, res;
//Stirling2 (n,k) = co-eff of x^k in product of
   polynomials A & B
//where A(i) = (-1)^i / i!  and B(i) = i^n / i!
int Stirling2(int n, int r) {
  a.resize(n + 1); b.resize(n + 1);
  for (int i = 0; i <= n; i++) {
    a[i] = invfct[i];
    if (i % 2 == 1) a[i] = mod - a[i];
  }
  for (int i = 0; i <= n; i++) {
    b[i] = bigMod(i, n, mod);
    b[i] = (b[i] * invfct[i]) % mod;
  }
  NTT ntt(mod);
  ntt.multiply(a, b, res);
  return res[r];
}
```

1.68 Stress Testing

```bash
set -e
g++ -std=c++17 gen.cpp -o gen
g++ -std=c++17 main.cpp -o main
g++ -std=c++17 brute.cpp -o brute
for((i = 1; ; ++i)); do
  echo $i
  ./gen $i > in
  ./main < in > out
  ./brute < in > out2
  diff -w out out2 || break
done
```

1.69 Subset Convolution

Description: $c_i = \sum_{j \subseteq i} a_j b_{i \oplus j}$.

Time: $\mathcal{O}(n * \log^2(n))$

```cpp
vector<int> subrset_conv (vector<int> a, vector<int>
   b) {
  int n = a.size();
  int lg = log2(n);
  vector<int> cnt(n);
  vector<vector<int>> fa(lg + 1, vector<int> (n)),
     fb(lg + 1, vector<int> (n)), g(lg + 1,
     vector<int> (n));
  for (int i = 0; i < n; ++i) {
    cnt[i] = cnt[i >> 1] + (i & 1);
    fa[cnt[i]][i] = a[i] % mod;
    fb[cnt[i]][i] = b[i] % mod;
  }
  for (int k = 0; k <= lg; ++k) {
    fwht(fa[k], 0, 1);  fwht(fb[k], 0, 1);
  }
  for (int k = 0; k <= lg; ++k) {
    for (int j = 0; j <= k; ++j) {
      for (int i = 0; i < n; ++i) {
        g[k][i] = addr(g[k][i], 1ll * fa[j][i] * fb[k
          - j][i] % mod);
      }
    }
  }
  for (int k = 0; k <= lg; ++k) {
    fwht(g[k], 1, 1);
  }
  vector<int> c(n);
  for (int i = 0; i < n; ++i) {
    c[i] = g[cnt[i]][i];
  }
  return c;
}
```

1.70 Suffix Array

```cpp
array<vector<int>, 2> get_sa(string& s, int lim=128) {
    // for integer, just change string to vector<int>
     and minimum value of vector must be >= 1
  int n = s.size() + 1, k = 0, a, b;
  vector<int> x(begin(s), end(s)+1), y(n), sa(n),
    lcp(n), ws(max(n, lim)), rank(n);
  x.back() = 0;
  iota(begin(sa), end(sa), 0);
  for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim
    = p) {
    p = j, iota(begin(y), end(y), n - j);
    for (int i = 0; i < n; ++i) if (sa[i] >= j) y[p++]
      = sa[i] - j;
    fill(begin(ws), end(ws), 0);
    for (int i = 0; i < n; ++i) ws[x[i]]++;
    for (int i = 1; i < lim; ++i) ws[i] += ws[i - 1];
    for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
    swap(x, y), p = 1, x[sa[0]] = 0;
    for (int i = 1; i < n; ++i) a = sa[i - 1], b =
      sa[i], x[b] =
        (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 :
          p++;
  }
  for (int i = 1; i < n; ++i) rank[sa[i]] = i;
  for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
    for (k && k--, j = sa[rank[i] - 1]; s[i + k] ==
      s[j + k]; k++);
  sa.erase(sa.begin()), lcp.erase(lcp.begin());
  return {sa, lcp};
}
## Comparing Two Substrings
auto query = [&] (int l1, int r1, int l2, int r2) {
  int len1 = r1 - l1 + 1, len2 = r2 - l2 + 1;
  int len = min(len1, len2);
  int i = pos[l1], j = pos[l2], x;
  if (l1 != l2) x = st.query(i, j);
  else x = len;
  if (x >= len) {
    if (len1 == len2) return 0;
    if (len1 < len2) return -1;
    return 1;
  }
  if (s[l1 + x] < s[l2 + x]) return -1;
  return 1;
};
## Kth Unique Substring
auto kth = [&] (ll k) {
  int i = 0;
  while (i + 1 < n and k > n - sa[i] - lcp[i]) {
    k -= n - sa[i] - lcp[i];
    i++;
  }
  k = min(k, 0ll + n - sa[i] - lcp[i]);
  array<int, 2> ret = {sa[i], k + lcp[i]};
  return ret;
};
## Several Consecutive Identical Substrings
for (int i = 1; i < n; ++i) {
  for (int j = i; j < n; j += i) {
    // Block = [j-i...j-1]
    int e1 = rmq(0, pos[j - i], pos[j]), e2 = 0;
    if (i < j) {
      e2 = rmq(1, rev_pos[j - i - 1], rev_pos[j - 1]);
    }
    int k = (e1 + e2) / i + 1;
    // [j-i-e2 ... j-1+e1] is periodic with period
      length = i
  }
}
```

1.71 Suffix Automaton with Rollback

```cpp
int len[N], lnk[N], sz, last;
int nxt[N][A];
vector<pair<int*, int>> cng[N];
int R;
void update(int &x, int y) {
  cng[R].push_back({&x, x});
  x = y;
}
void rollback() {
  R--;
  while (!cng[R].empty()) {
    auto [x, y] = cng[R].back();  cng[R].pop_back();
    *x = y;
  }
}
void init (int n) {
  len[0] = 0, lnk[0] = -1, last = 0, sz = 1;
  for (int i = 0; i <= 2 * n; ++i) {
    memset(nxt[i], -1, sizeof nxt[i]);
  }
  while (R > 0)  rollback();
}
void add (int c) {
  int new_sz = sz;
  int cur = new_sz++;
  update(len[cur], len[last] + 1);
  int u = last;
  while (u != -1 and nxt[u][c] == -1) {
    update(nxt[u][c], cur);
    u = lnk[u];
  }
  if (u == -1) {
    update(lnk[cur], 0);
  }
  else {
    int v = nxt[u][c];
```

```
    if (len[u] + 1 == len[v]) {
      update(lnk[cur], v);
    }
    else {
      int w = new_sz++;
      update(len[w], len[u] + 1);
      update(lnk[w], lnk[v]);
      for (int a = 0; a < A; ++a) {
        update(nxt[w][a], nxt[v][a]);
      }
      while (u != -1 and nxt[u][c] == v) {
        update(nxt[u][c], w);
        u = lnk[u];
      }
      update(lnk[cur], w);
      update(lnk[v], w);
    }
  }
  update(last, cur);
  update(sz, new_sz);
}
```

1.72 Suffix Automaton

```
int len[2 * N], lnk[2 * N], last, sz = 1;
unordered_map<char, int> to[2 * N]; // Use map during
  finding kth substring
int deg[2 * N], focc[2 * N];  // First Occurrence
ll cnt[2 * N], dp[2 * N];
void init(int n) {
  fill(deg, deg + sz, 0);
  fill(cnt, cnt + sz, 0);
  while (sz) to[--sz].clear();
  lnk[0] = -1, last = 0, sz = 1;
}
void add (char c, int i) {
  int cur = sz++;
  len[cur] = len[last] + 1;
  cnt[cur] = 1; dp[cur] = i;
  focc[cur] = i;
  int u = last;
  last = cur;
  while (u != -1 and !to[u].count(c)) {
    to[u][c] = cur;
    u = lnk[u];
  }
  if (u == -1) {
    lnk[cur] = 0;
  }
  else {
    int v = to[u][c];
    if (len[u] + 1 == len[v]) {
      lnk[cur] = v;
    }
    else {
      int w = sz++;
      len[w] = len[u] + 1, lnk[w] = lnk[v], to[w] =
        to[v];
      focc[w] = focc[v];
      while (u != -1 and to[u][c] == v) {
        to[u][c] = w, u = lnk[u];
      }
      lnk[cur] = lnk[v] = w;
    }
  }
}
bool exist (string &p) {
  int u = 0;
  for (auto c: p) {
    if (!to[u].count(c)) return false;
    u = to[u][c];
  }
  return true;
}
void build() {
  deg[0] = 1;
  for (int u = 1; u < sz; ++u) {
    deg[lnk[u]]++;
  }
  queue<int> q;
  for (int u = 0; u < sz; ++u) {
    if (!deg[u]) q.push(u);
  }
  while (!q.empty()) {
    int u = q.front(); q.pop();
    int v = lnk[u];
    cnt[v] += cnt[u]; // DP on suffix link tree
    for (auto [c, v]: to[u]) { // DP on DAG
      dp[u] = max(dp[u], dp[v]);
    }
    deg[v]--;
    if (!deg[v]) q.push(v);
  }
}
## Count number of occurrence for each k length
  substring of s in SA
ll count (string s, int k) {
  ll ret = 0;
  int u = 0, L = 0;
  for (auto c: s) {
    while (u and !to[u].count(c)) u = lnk[u], L =
      len[u];
    if (!to[u].count(c)) continue;
    u = to[u][c], L++;
    while (len[lnk[u]] >= k) u = lnk[u], L = len[u];
    if (L >= k) ret += cnt[u];
  }
  return ret;
}
## Kth substring (not distinct)
ll dp[2 * N];
ll dfs (int u) {
  if (dp[u] != -1) return dp[u];
  dp[u] = cnt[u]; // For distinct dp[u] = 1
  for (auto [c, v]: to[u]) {
    dp[u] += dfs(v);
  }
  return dp[u];
}
void yo (int u, ll k, string &s) {
  if (k <= 0) return ;
  for (auto [c, v]: to[u]) {
    if (k > dfs(v)) k -= dfs(v);
    else {
      s += c;
      k -= cnt[v];  // For distinct k -= 1
      yo(v, k, s);
      return ;
    }
  }
}
```

1.73 Treap

```
## Typical TEAP
struct node {
  ll val, prior, sz, sum;
  node *l, *r;
  node(int val, int prior, int sz) : val(val),
      prior(prior), sz(sz), sum(0), l(nullptr),
      r(nullptr){}
};
using pnode = node*;
pnode root;
pnode new_node(ll val){
  return new node(val, rand(), 1);
}
int get_sz(pnode u){
  return u? u->sz: 0;
}
void update(pnode u){
  if (!u)  return ;
  u->sz = get_sz(u->l) + 1 + get_sz(u->r);
  u->sum = u->val + (u->l? u->l->sum: 0) + (u->r?
    u->r->sum: 0);
}
void split(pnode u, pnode &l, pnode &r, ll val){
  if(!u)  l = r = NULL;
  else if(val > u->val) split(u->r, u->r, r, val), l =
    u;
  else split(u->l, l, u->l, val), r = u;
  update(u);
}
void merge(pnode &u, pnode l, pnode r){
  if(!l or !r) u = l? l: r;
  if(l->prior > r->prior) merge(l->r, l->r, r),  u = l;
  else merge(r->l, l, r->l),  u = r;
  update(u);
}
void insert(pnode &u, pnode it){
  if(!u)  u = it;
  else if(it->prior > u->prior) split(u, it->l, it->r,
    it->val), u = it;
  else insert(it->val <= u->val ? u->l: u->r, it);
  update(u);
}
void erase(pnode &u, ll val){
  if(!u)  return ;
  if(val == u->val)  merge(u, u->l, u->r);
  else  erase(val < u->val ? u->l: u->r, val);
  update(u);
}
bool present(pnode u, int x){
  if(!u)  return false;
  if(u->val == x)  return true;
  if(u->val < x) return present(u->r, x);
  return present(u->l, x);
}
ll kth(pnode u, int k){
  if(get_sz(u) < k) return INT_MIN;
  if(get_sz(u->l) == k-1)  return u->val;
  if(get_sz(u->l) < k-1) return kth(u->r, k -
    get_sz(u->l) - 1);
  return kth(u->l, k);
}
int cnt_less(pnode u, ll x){
  if(!u)  return 0;
  if(x <= u->val)  return cnt_less(u->l, x);
  return get_sz(u->l) + 1 + cnt_less(u->r, x);
}
ll sum_less(pnode u, ll x) {
  if (!u)  return 0;
  if (x <= u->val)  return sum_less(u->l, x);
  return u->val + (u->l? u->l->sum: 0) +
    sum_less(u->r, x);
}
## Implicit TREAP
struct node {
  ll val, sum;
  int prior, sz, rev;
  node *l, *r;
  node(){}
  node(ll val): val(val), sum(val), prior(rand()),
    sz(1), rev(0), l(nullptr), r(nullptr) {}
};
using pnode = node*;
pnode root;
int get_sz(pnode t) {
  return t? t->sz: 0;
```

```
}
ll get_sum(pnode t) {
  return t? t->sum: 0;
}
void update(pnode &t) {
  if (!t)  return ;
  t->sz = get_sz(t->l) + 1 + get_sz(t->r);
  t->sum = get_sum(t->l) + t->val + get_sum(t->r);
}
void push(pnode t) {
  if (t and t->rev) {
    swap(t->l, t->r);
    t->rev = 0;
    if (t->l) {
      t->l->rev ^= 1;
    }
    if (t->r) {
      t->r->rev ^= 1;
    }
  }
}
void merge(pnode &t, pnode l, pnode r){
  push(l);
  push(r);
  if(!l or !r)  t=l?l:r;
  else if(l->prior > r->prior)  merge(l->r, l->r, r), t=l;
  else  merge(r->l,l,r->l) , t=r;
  update(t);
}
void split(pnode t, pnode &l, pnode &r, int pos, int add=0) {
  push(t);
  if(!t)  return void(r=l=NULL);
  int cur_pos = get_sz(t->l)+add;
  if(pos > cur_pos) split(t->r, t->r, r, pos, cur_pos+1), l = t;
  else   split(t->l, l, t->l, pos, add), r=t;
  update(t);
}
void insert(pnode &t, pnode it, int i) {
  pnode t1, t2;
  split(t, t1, t2, i);
  merge(t1, t1, it);
  merge(t, t1, t2);
}
void reverse(pnode &t, int l, int r) {
  pnode lt, mt, rt;
  split(t, t, rt, r + 1);
  split(t, lt, mt, l);
  mt->rev = 1;
  merge(mt, mt, rt);
  merge(t, lt, mt);
}
ll rsum(pnode& t, int l, int r) {
  pnode lt, mt, rt;
  split(t, t, rt, r + 1);
  split(t, lt, mt, l);
  ll ret = mt->sum;
  merge(mt, mt, rt);
  merge(t, lt, mt);
  return ret;
}
int n, q;  cin >> n >> q;
vector<ll> a(n);
for (auto &ai: a) {
  cin >> ai;
}
for (int i = 0; i < n; ++i) {
  insert(root, new node(a[i]), i);
}
while (q--) {
  int tp, l, r;  cin >> tp >> l >> r;  l--, r--;
  if (tp == 1) {
    reverse(root, l, r);
  }
  else {
    cout << rsum(root, l, r) << "\n";
  }
}
```

1.74 Two Edge CC

```
struct graph {
  int n, t, sz;
  vector<vector<int>> adj;
  vector<int> tin, low, cmp;
  graph(int n): n(n),adj(n),tin(n),low(n),cmp(n){}
  void add_edge(int u, int v){
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  void dfs(int u, int p){
    tin[u]=low[u]=t++;
    int cnt=0;
    for(int v: adj[u]){
      if(v==p and ++cnt <= 1) continue;
      if(tin[v]!=-1)  low[u] = min(low[u], tin[v]);
      else {
        dfs(v,u);
        low[u] = min(low[u], low[v]);
      }
    }
  }
  void dfs2(int u, int p){
    if(p!=-1 and tin[p]>=low[u]) cmp[u] = cmp[p];
    else  cmp[u] = sz++;
    for(int v: adj[u]){
      if(cmp[v]==-1)  dfs2(v,u);
    }
  }
  void process_2ecc(){
    t = 0, sz = 0;
    for (int i = 0; i < n; ++i){
      tin[i] = low[i] = cmp[i] = -1;
    }
    for (int i = 0; i < n; ++i){
      if(tin[i]==-1)  dfs(i,-1);
    }
    for (int i = 0; i < n; ++i){
      if(cmp[i]==-1)  dfs2(i,-1);
    }
  }
};
```

1.75 XOR Basis

```
using Basis = array<ll, D>;
bool add(Basis &b, ll x) {
  for (int i = D - 1; i >= 0; --i) {
    if (x >> i & 1 ^ 1) continue;
    if (!b[i]) return b[i] = x, true;
    x ^= b[i];
  } return false;
}
void reduce(Basis &b, ll &x) {
  for (int d = D - 1; d >= 0; --d) {
    x = min(x, x ^ b[d]);
  }
}
bool exist(Basis &b, ll x) {
  return reduce(b, x), x == 0;
}
ll max_xor(Basis &b, ll x = 0) {
  for (int i = D - 1; i >= 0; --i) {
    x = max(x, x ^ b[i]);
  } return x;
}
ll kth(Basis &b, ll k) {
  ll ret = 0, rem = rnk;
  for (int i = D - 1; i >= 0; --i) {
    if (!b[i]) continue;
    rem--;
    if (ret >> i & 1) ret ^= b[i];
    if ((1ll << rem) >= k) continue;
    ret ^= b[i]; k -= 1 << rem;
  } return ret;
}
int cnt_supermask(Basis &b, int x) {
  Basis c{0}; int rnk = 0;
  for (auto bi: b) {
    bi &= x, rnk += add(c, bi);
  }
  int ret = 0;
  if (exist(c, x)) {
    ret = two[n - rnk]; // 2^(n-rnk)
    if (x == 0) sub(ret, 1);  // When counting
     ↪ non-empty subsets
  } return ret;
}
Basis b{0};
## Static Range XOR-basis Query
Basis b[N], idx[N];
void build () {
  b[0] = idx[0] = Basis{0};
  for (int i = 0; i < n; ++i) {
    if (i) b[i] = b[i - 1], idx[i] = idx[i - 1];
    ll x = a[i], j = i;
    for (int d = D - 1; d >= 0; --d) {
      if (x >> d & 1 ^ 1)  continue;
      if (!b[i][d]) b[i][d] = x, idx[i][d] = j;
      if (idx[i][d] < j) swap(b[i][d], x),
       ↪ swap(idx[i][d], j);
      x ^= b[i][d];
    }
  }
}
Basis rbasis (int l, int r) {
  Basis ret{0};
  for (int d = D - 1; d >= 0; --d) {
    if (b[r][d] and l <= idx[r][d]) ret[d] = b[r][d];
  }
  return ret;
}
## Maximum XOR Subset Printing
Basis b{0};
vector<int> idx[D];
bool add(ll x, int i) {
  vector<int> cur = {i};
  for (int i = D - 1; i >= 0; --i) {
    if (x >> i & 1 ^ 1) continue;
    if (!b[i]) return b[i] = x, idx[i] = cur, true;
    x ^= b[i], cur.push_back(i);
  } return false;
}
pair<ll, vector<int>> max_xor(ll x = 0) {
  vector<int> ret, cnt(D);
  for (int i = D - 1; i >= 0; --i) {
    if ((x ^ b[i]) > x) {
      x ^= b[i];
      cnt[i] ^= 1;
    }
  }
  for (int i = 0; i < D; ++i) {
    if (cnt[i]) {
      ret.push_back(idx[i][0]);
      for (int j = 1; j < idx[i].size(); ++j) {
        cnt[idx[i][j]] ^= 1;
      }
    }
```

```cpp
  }
  return make_pair(x, ret);
}
## Reduced row echelon form (unique).
void reduce(vector<ll> &b, ll &x) {
  for (auto bi: b) x = min(x, x ^ bi);
}
void add(vector<ll> &b, ll x) {
  reduce(b, x);
  if (x) { for (auto &bi: b) {
      bi = min(bi, bi ^ x);
    }
    b.push_back(x);
  }
}
## Previous of upper bound of k with initial value = x
ll ub(vector<ll> &b, ll k, ll x = 0) {
  reduce(b, x);
  sort(b.rbegin(), b.rend());
  for (auto bi: b) {
    if (x > k) x = min(x, x ^ bi);
    else if ((x ^ bi) <= k) x ^= bi;
  }
  if (x > k) x = 0;
  return x;
}
```

1.76 Z Algorithm

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

2 Geometry

2.1 Angular Sort

```cpp
inline bool up (point p) {
  return p.y > 0 or (p.y == 0 and p.x >= 0);
}
sort(v.begin(), v.end(), [] (point a, point b) {
  return up(a) == up(b) ? a.x * b.y > a.y * b.x :
   up(a) < up(b);
});
inline int quad (point p) {
  if (p.y >= 0)  return p.x < 0;
  return 2 + (p.x >= 0);
}
sort(pt.begin(), pt.end(), [] (point a, point b) {
  return quad(a) == quad(b) ? a.x * b.y > a.y * b.x :
   quad(a) < quad(b);
});
```

2.2 CircleCircleIntersection

Description: compute intersection of circle centered at a with radius r with circle centered at b with radius R.

```cpp
vector<PT> CircleCircleIntersection(PT a, PT b, double
 r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}
```

2.3 CircleLineIntersection

Description: Compute intersection of line through points a and b with circle centered at c with radius $r > 0$.

```cpp
vector<PT> CircleLineIntersection(PT a, PT b, PT c,
 double r) {
  vector<PT> ret;
  b = b-a; a = a-c;
  double A = dot(b, b); double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}
```

2.4 Closest Pair of Points

```cpp
ll min_dis(vector<array<int, 2>> &pts, int l, int r) {
  if (l + 1 >= r)  return LLONG_MAX;
  int m = (l + r) / 2;
  ll my = pts[m-1][1];
  ll d = min(min_dis(pts, l, m), min_dis(pts, m, r));
  inplace_merge(pts.begin()+l, pts.begin()+m,
   pts.begin()+r);
  for (int i = l; i < r; ++i) {
    if ((pts[i][1] - my) * (pts[i][1] - my) < d) {
      for (int j = i + 1; j < r and (pts[i][0] -
        pts[j][0]) * (pts[i][0] - pts[j][0]) < d;
       ++j) {
        ll dx = pts[i][0] - pts[j][0], dy = pts[i][1]
         - pts[j][1];
        d = min(d, dx * dx + dy * dy);
      }
    }
  }
  return d;
}
vector<array<int, 2>> pts(n);
sort(pts.begin(), pts.end(), [&] (array<int, 2> a,
 array<int, 2> b){
  return make_pair(a[1], a[0]) < make_pair(b[1], b[0]);
});
```

2.5 ComputeCentroid

```cpp
// centroid of a (possibly nonconvex) polygon.
PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y -
     p[j].x*p[i].y);
  }
  return c / scale;
}
```

2.6 ComputeCircleCenter

```cpp
// compute center of circle passing through three
 points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b),
   c, c+RotateCW90(a-c));
}
```

2.7 ComputeLineIntersection

Description: compute intersection of line passing through a and b with line passing through c and d, assuming that unique intersection exists; for segment intersection, check if segments intersect first.

```cpp
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}
```

2.8 ComputeSignedArea

Description: Computes the area of a (possibly nonconvex) polygon, assuming that the coordinates are listed in a clockwise or counterclockwise fashion.

```cpp
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}
double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}
```

2.9 Convex Hull

```cpp
vector <PT> convexHull (vector <PT> p) {
  int n = p.size(), m = 0;
  if (n < 3) return p;
  vector <PT> hull(n + n);
  sort(p.begin(), p.end(), [&] (PT a, PT b) {
    return (a.x==b.x? a.y<b.y: a.x<b.x);
  });
  for (int i = 0; i < n; ++i) {
    while (m > 1 and cross(hull[m - 2] - p[i], hull[m
     - 1] - p[i]) <= 0) --m;
    hull[m++] = p[i];
  }
  for (int i = n - 2, j = m + 1; i >= 0; --i) {
    while (m >= j and cross(hull[m - 2] - p[i], hull[m
     - 1] - p[i]) <= 0) --m;
    hull[m++] = p[i];
  }
  hull.resize(m - 1); return hull;
}
```

2.10 DistancePointPlane

Description: compute distance between point (x, y, z) and plane $ax + by + cz = d$

```cpp
double DistancePointPlane(double x, double y, double
 z, double a, double b, double c, double d) {
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}
```

2.11 DistancePointSegment

```cpp
// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}
```

2.12 Geo3D

```
const ld kPi = 2 * acos(0);
const ld kEps = 1e-9;
struct P {
  ld x, y, z;
  P(ld a = 0, ld b = 0, ld c = 0) : x(a), y(b), z(c) {}
  P(const P& a) : x(a.x), y(a.y), z(a.z) {}
  void operator=(const P& a) { x = a.x; y = a.y; z = a.z; }
  P operator+(const P& a) const { P p(x + a.x, y + a.y, z + a.z); return p; }
  P operator-(const P& a) const { P p(x - a.x, y - a.y, z - a.z); return p; }
  P operator*(ld a) const { P p(x * a, y * a, z * a); return p; }
  P operator/(ld a) const { assert(a > kEps); P p(x / a, y / a, z / a); return p; }
  P& operator+=(const P& a) { x += a.x; y += a.y; z += a.z; return *this; }
  P& operator-=(const P& a) { x -= a.x; y -= a.y; z -= a.z; return *this; }
  P& operator*=(ld a) { x *= a; y *= a; z *= a; return *this;}
  P& operator/=(ld a) { assert(a > kEps); x /= a; y /= a; z /= a; return *this; }
  ld& operator[](int a) {
    if(a == 0) return x;
    if(a == 1) return y;
    if(a == 2) return z;
    assert(false);
  }
  bool IsZero() const {
    return fabs(x) < kEps && fabs(y) < kEps && fabs(z) < kEps;
  }
  bool operator==(const P& a) const {
    return (*this - a).IsZero();
  }
  ld DotProd(const P& a) const {
    return x * a.x + y * a.y + z * a.z;
  }
  ld Norm() const { return sqrt(x*x+y*y+z*z); }
  void NormalizeSelf() { *this /= Norm(); }
  P Normalize() {
    P res(*this);
    res.NormalizeSelf();
    return res;
  }
  ld Dis(const P& a) const { return (*this - a).Norm(); }
  pair<ld, ld> SphericalAngles() const {
    return {atan2(z, sqrt(x * x + y * y)), atan2(y, x)};
  }
  ld Area(const P& p) const {
    return Norm() * p.Norm() * sin(Angle(p)) / 2;
  }
  ld Angle(const P& p) const {
    ld a = Norm();
    ld b = p.Norm();
    ld c = Dis(p);
    return acos((a * a + b * b - c * c) / (2 * a * b));
  }
  static ld Angle(P& p, P& q) { return p.Angle(q); }
  P CrossProd(P p) {
    P q(*this);
    return {q[1] * p[2] - q[2] * p[1], q[2] * p[0] - q[0] * p[2], q[0] * p[1] - q[1] * p[0]};
  }
  static bool LexCmp(const P& a, const P& b) {
    if (fabs(a.x - b.x) > kEps) return a.x < b.x;
    if (fabs(a.y - b.y) > kEps) return a.y < b.y;
    return a.z < b.z;
  }
};
struct Line {
  P p[2]; bool is_seg;
  Line(P a, P b, bool is_seg_ = false) {
    p[0] = a;
    p[1] = b;
    is_seg = is_seg_;
  }
  Line() {}
  P& operator[](int a) { return p[a]; }
};
struct Plane {
  P p[3];
  Plane(P a, P b, P c) {
    p[0] = a; p[1] = b; p[2] = c;
  }
  P& operator[](int a) { return p[a]; }
  P GetNormal() {
    P cross = (p[1] - p[0]).CrossProd(p[2] - p[0]);
    return cross.Normalize();
  }
  void GetPlaneEq(ld& A, ld& B, ld& C, ld& D) {
    P normal = GetNormal();
    A = normal[0]; B = normal[1];
    C = normal[2]; D = normal.DotProd(p[0]);
    assert(abs(D - normal.DotProd(p[1])) < kEps);
    assert(abs(D - normal.DotProd(p[2])) < kEps);
  }
};
struct Utils {
  static P ProjPtToLine(P p, Line l) { // ok
    P diff = l[1] - l[0]; diff.NormalizeSelf();
    return l[0] + diff * (p - l[0]).DotProd(diff);
  }
  static ld DisPtLine(P p, Line l) { // ok
//    ld area = Area(p, l[0], l[1]);
//     ld dis1 = 2 * area / l[0].Dis(l[1]);
    ld dis2 = p.Dis(ProjPtToLine(p, l));
//     assert(abs(dis1 - dis2) < kEps);
    return dis2;
  }
  static ld DisPtPlane(P p, Plane pl) {
    P normal = pl.GetNormal();
    return abs(normal.DotProd(p - pl[0]));
  }
  static P ProjPtToPlane(P p, Plane pl) {
    P normal = pl.GetNormal();
    return p - normal * normal.DotProd(p - pl[0]);
  }
  static bool PtBelongToPlane(P p, Plane pl) {
    return DisPtPlane(p, pl) < kEps;
  }
  static bool PtBelongToLine(P& p, Line& l) {
    return DisPtLine(p, l) < kEps;
  }
  static ld Det(P a, P b, P d) { // ok
    P pts[3] = {a, b, d};
    ld res = 0;
    for (int sign : {-1, 1}) {
     for(int st_col=0;st_col<3;++st_col) {
        int c = st_col;
        ld prod = 1;
        for(int r=0;r<3;++r) {
          prod *= pts[r][c];
          c = (c + sign + 3) % 3;
        }
        res += sign * prod;
      }
    }
    return res;
  }
  static ld Area(P p, P q, P r) { // ok
    q -= p; r -= p;
    return q.Area(r);
  }
};
int main() { return 0; }
/*  double x = sin(lng/180*PI)*cos(lat/180*PI)*alt;
    double y = cos(lng/180*PI)*cos(lat/180*PI)*alt;
    double z = sin(lat/180*PI)*alt; */
```

2.13 Half Plane Intersection

Description: Calculates the intersection of halfplanes, assuming every half-plane allows the region to the left of its line.

```
struct Halfplane {
  PT p, pq; ld angle;
  Halfplane() {}
  // Two points on line
  Halfplane(const PT& a, const PT& b) : p(a), pq(b - a) {
    angle = atan2l(pq.y, pq.x);
  }
  bool out(const PT& r) {
    return cross(pq, r - p) < -EPS;
  }
  bool operator < (const Halfplane& e) const {
    return angle < e.angle;
  }
  friend PT inter(const Halfplane& s, const Halfplane& t) {
    ld alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
    return s.p + (s.pq * alpha);
  }
};
vector<PT> hp_intersect(vector<Halfplane>& H) {
  PT box[4] = {  // Bounding box in CCW order
    PT(INF, INF),  PT(-INF, INF),
    PT(-INF, -INF), PT(INF, -INF)
  };
  for(int i = 0; i<4; i++) { // Add bounding box half-planes.
    Halfplane aux(box[i], box[(i+1) % 4]);
    H.push_back(aux);
  }
  sort(H.begin(), H.end());
  deque<Halfplane> dq; int len = 0;
  for(int i = 0; i < int(H.size()); i++) {
    while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))) {
      dq.pop_back(); --len;
    }
    while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
      dq.pop_front(); --len;
    }
    if (len > 0 && fabsl(cross(H[i].pq, dq[len-1].pq)) < EPS) {
      if (dot(H[i].pq, dq[len-1].pq) < 0.0)
        return vector<PT>();
      if (H[i].out(dq[len-1].p)) {
        dq.pop_back(); --len;
      }
      else continue;
    }
    dq.push_back(H[i]); ++len;
  }
  while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
```

```cpp
    dq.pop_back(); --len;
  }
  while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
    dq.pop_front(); --len;
  }
  // Report empty intersection if necessary
  if (len < 3) return vector<PT>();
  // Reconstruct the convex polygon from the remaining half-planes.
  vector<PT> ret(len);
  for(int i = 0; i+1 < len; i++) {
    ret[i] = inter(dq[i], dq[i+1]);
  }
  ret.back() = inter(dq[len-1], dq[0]);
  return ret;
}
```

2.14 IsSimple

```cpp
// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}
```

2.15 LinesCollinear

```cpp
bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}
```

2.16 LinesParallel

```cpp
// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}
```

2.17 Minkowski Sum

```cpp
void reorder_polygon(vector<PT>& P){
  size_t pos = 0;
  for(size_t i = 1; i < P.size(); i++){
    if(pii(P[i].y, P[i].x) < pii(P[pos].y, P[pos].x))
      pos = i;
  }
  rotate(P.begin(), P.begin() + pos, P.end());
}
vector<PT> minkowski(vector<PT> &P, vector<PT> &Q){
  reorder_polygon(P); reorder_polygon(Q);
  P.push_back(P[0]); P.push_back(P[1]);
  Q.push_back(Q[0]); Q.push_back(Q[1]);
  vector<PT> result;
  size_t i = 0, j = 0;
  while(i < P.size() - 2 or j < Q.size() - 2){
    result.push_back(P[i] + Q[j]);
    auto vprod = cross((P[i + 1] - P[i]), (Q[j + 1] - Q[j]));
    if(vprod >= 0 && i < P.size() - 2) ++i;
    if(vprod <= 0 && j < Q.size() - 2) ++j;
  } return result;
}
```

2.18 Point

```cpp
double INF = 1e100;
double EPS = 1e-12;
struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
  PT operator * (double c)     const { return PT(x*c, y*c  ); }
  PT operator / (double c)     const { return PT(x/c, y/c  ); }
};
double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double abs(PT p) { return sqrt(p.x*p.x + p.y*p.y); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  return os << "(" << p.x << "," << p.y << ")";
}
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}
// angle (range [0, pi]) between two vectors
double angle(PT v, PT w) {
  return acos(clamp(dot(v,w) / abs(v) / abs(w), -1.0, 1.0));
}
```

2.19 PointInPolygon

Description: -1 = strictly inside, 0 = on, 1 = strictly outside.

```cpp
int PointInPolygon(vector<PT> &P, PT a) {
  int cnt = 0, n = P.size();
  for(int i = 0; i < n; ++i) {
    PT q = P[(i + 1) % n];
    if (onSegment(P[i], q, a)) return 0;
    cnt ^= ((a.y < P[i].y) - (a.y < q.y)) * cross(P[i] - a, q - a) > 0;
  } return cnt > 0 ? -1 : 1;
}
int PointInConvexPolygon(vector<PT> &P, const PT& q) { // O(log n)
  int n = P.size();
  ll a = cross(P[0] - q, P[1] - q), b = cross(P[0] - q, P[n - 1] - q);
  if (a < 0 or b > 0) return 1;
  int l = 1, r = n - 1;
  while (l + 1 < r) {
    int mid = l + r >> 1;
    if (cross(P[0] - q, P[mid] - q) >= 0) l = mid;
    else r = mid;
  }
  ll k = cross(P[l] - q, P[r] - q);
  if (k <= 0) return k < 0 ? 1 : 0;
  if (l == 1 and a == 0) return 0;
  if (r == n - 1 and b == 0) return 0;
  return -1;
}
```

2.20 ProjectPointLine

```cpp
// project point c onto line through a and b, assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}
```

2.21 ProjectPointSegment

```cpp
// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}
```

2.22 SegmentsIntersect

```cpp
// determine if line segment from a to b intersects with line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}
```

2.23 UnitTest

```cpp
// expected: (-5,2)
RotateCCW90(PT(2,5))
// expected: (5,-2)
RotateCW90(PT(2,5));
// expected: (-5,2)
RotateCCW(PT(2,5),M_PI/2);
// expected: (5,2)
ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7));
// expected: (5,2) (7.5,3) (2.5,1)
ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7))
ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7))
ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7))
// expected: 6.78903
DistancePointPlane(4,-4,3,2,-2,5,-8);
// expected: 1 0 1
LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5))
LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5))
LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13));
// expected: 0 0 1
LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5))
LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5))
LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13));
// expected: 1 1 1 0
SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3))
SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5))
SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1))
SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7));
// expected: (1,2)
```

```
ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1),
  PT(-1,3));
// expected: (1,1)
ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5));
vector<PT> v({PT(0,0), PT(5,0), PT(5,5), PT(0,5)});
// expected: 1 1 1 0 0
PointInPolygon(v, PT(2,2))
PointInPolygon(v, PT(2,0))
PointInPolygon(v, PT(0,2))
PointInPolygon(v, PT(5,2))
PointInPolygon(v, PT(2,5))
// expected: 0 1 1 1 1
PointOnPolygon(v, PT(2,2))
PointOnPolygon(v, PT(2,0))
PointOnPolygon(v, PT(0,2))
PointOnPolygon(v, PT(5,2))
PointOnPolygon(v, PT(2,5))
// expected: {(1,6)}, {(5,4) (4,5)}, {}, {(4,5) (5,4)},
  {}, {(4,5) (5,4)}
CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10,
  sqrt(2.0)/2.0);
CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5,
  sqrt(2.0)/2.0);
// area = 5.0, centroid = (1.1666666, 1.166666)
vector<PT> p({PT(0,0), PT(5,0), PT(1,1), PT(0,5)});
ComputeCentroid(p), ComputeArea(p);
```

3 Notes

3.1 General

- $|a-b|+|b-c|+|c-a| = 2(\max(a,b,c)-\min(a,b,c))$
- $a\cdot b \le c \to a \le \lfloor \frac{c}{b} \rfloor$ is correct
- $a\cdot b < c \to a < \lfloor \frac{c}{b} \rfloor$ is incorrect
- $a\cdot b \ge c \to a \ge \lfloor \frac{c}{b} \rfloor$ is correct
- $a\cdot b > c \to a > \lfloor \frac{c}{b} \rfloor$ is correct
- For positive integer n, and arbitrary real numbers m, x:

$$\left\lfloor \frac{\lfloor \frac{x}{m} \rfloor}{n} \right\rfloor = \left\lfloor \frac{x}{mn} \right\rfloor \text{ and } \left\lceil \frac{\lceil \frac{x}{m} \rceil}{n} \right\rceil = \left\lceil \frac{x}{mn} \right\rceil$$

3.2 Probabilty and Expected Value

- *Bayes Theorem:* $P(A\mid B) = \frac{P(A\cap B)}{P(B)} = \frac{P(B|A)\cdot P(A)}{P(B)}$

3.3 Geometry

3.3.1 Triangles

Circumradius: $R = \dfrac{abc}{4A}$, Inradius: $r = \dfrac{A}{s}$

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2+2c^2-a^2}$

Length of bisector (divides angles in two): $s_a = \sqrt{bc\left[1-\left(\dfrac{a}{b+c}\right)^2\right]}$

Law of tangents: $\dfrac{a+b}{a-b} = \dfrac{\tan\dfrac{\alpha+\beta}{2}}{\tan\dfrac{\alpha-\beta}{2}}$

3.3.2 Quadrilaterals

With side lengths a,b,c,d, diagonals e,f, diagonals angle θ, area A and magic flux $F = b^2+d^2-a^2-c^2$:

$$4A = 2ef\cdot\sin\theta = F\tan\theta = \sqrt{4e^2f^2-F^2}$$

For cyclic quadrilaterals the sum of opposite angles is 180°, $ef = ac+bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

3.3.3 Spherical coordinates

$$\begin{aligned} x &= r\sin\theta\cos\phi & r &= \sqrt{x^2+y^2+z^2} \\ y &= r\sin\theta\sin\phi & \theta &= \operatorname{acos}(z/\sqrt{x^2+y^2+z^2}) \\ z &= r\cos\theta & \phi &= \operatorname{atan2}(y,x) \end{aligned}$$

3.3.4 3D figures

Sphere	Volume $V = \frac{4}{3}\pi r^3$, surface area $S = 4\pi r^2$
Spherical sect.	Volume $V = \pi h^2(r-h/3)$, surface area $S = 2\pi rh$
Pyramid	Volume $V = \frac{1}{3}hS_{base}$
Cone	Volume $V = \frac{1}{3}\pi r^2 h$, lat. surf. area $S = \pi r\sqrt{r^2+h^2}$

3.4 Binomial Coefficent

- Factoring in: $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$
- Sum over k: $\sum_{k=0}^{n}\binom{n}{k} = 2^n$
- Alternating sum: $\sum_{k=0}^{n}(-1)^k\binom{n}{k} = 0$
- Even and odd sum: $\sum_{k=0}^{n}\binom{n}{2k} = \sum_{k=0}^{n}\binom{n}{2k+1}2^{n-1}$
- The Hockey Stick Identity
  - (Left to right) Sum over n and k: $\sum_{k=0}^{m}\binom{n+k}{k} = \binom{n+m-1}{m}$
  - (Right to left) Sum over n: $\sum_{m=0}^{n}\binom{m}{k} = \binom{n+1}{k+1}$
- Sum of the squares: $\sum_{k=0}^{n}(\binom{n}{k})^2 = \binom{2n}{n}$
- Weighted sum: $\sum_{k=1}^{n}k\binom{n}{k} = n2^{n-1}$
- Connection with the fibonacci numbers: $\sum_{k=0}\binom{n-k}{k} = F_{n+1}$
- Vandermonde's Identity: $\sum_{i=0}^{k}\binom{m}{i}\binom{n}{k-i} = \binom{m+n}{n}$
- If $f(n,k) = C(n,0)+C(n,1)+\ldots+C(n,k)$, Then $f(n+1,k) = 2*f(n,k)-C(n,k)$ [For multiple $f(n,k)$ queries, use Mo's algo]

Lucas Theorem

$$\binom{m}{n} \equiv \prod_{i=0}^{k}\binom{m_i}{n_i} (\mod p)$$

- $\binom{m}{n}$ is divisible by p if and only if at least one of the base-p digits of n is greater than the corresponding base-p digit of m.
- The number of entries in the nth row of Pascal's triangle that are not divisible by $p = \prod_{i=0}^{k}(n_i+1)$
- All entries in the $(p^k-1)th$ row are not divisible by p.
- $\binom{n}{m} \equiv \lfloor\frac{n}{p}\rfloor (\mod p)$

3.5 Fibonacci Number

1. $k = A-B, F_AF_B = F_{k+1}F_A^2 + F_kF_AF_{A-1}$
2. $\sum_{i=0}^{n}F_i^2 = F_{n+1}F_n$
3. $\sum_{i=0}^{n}F_iF_{i+1} = F_{n+1}^2-(-1)^n$
4. $\sum_{i=0}^{n}F_iF_{i+1} = F_{n+1}^2-(-1)^n$
5. $\sum_{i=0}^{n}F_iF_{i-1} = \sum_{i=0}^{n-1}F_iF_{i+1}$
6. $\gcd(F_m,F_n) = F_{\gcd(m,n)}$
7. $\sum_{0\le k\le n}\left(\binom{n-k}{k}\right) = F_{n+1}$
8. $\gcd(F_n,F_{n+1}) = \gcd(F_n,F_{n+2}) = \gcd(F_{n+1},F_{n+2}) = 1$

3.6 Sums

$1^4+2^4+3^4+\cdots+n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$

$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1}-1-\sum_{i=1}^{n}\left((i+1)^{m+1}-i^{m+1}-(m+1)i^m\right)\right]$

$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k n^{m+1-k}$

$\sum_{k=0}^{n} kx^k = (x-(n+1)x^{n+1}+nx^{n+2})/(x-1)^2$

$\sum_{i=0}^{n} i\times i! = (n+1)!-1$

3.7 Polynomials and Series

- $x^n-y^n = (x-y)(\sum_{i=0} n-1x^{n-i-1}y^i)$
- $\sum_{i=1,3,5,\ldots}^{i\le n}\binom{n}{i}a^{n-i}b^i = \frac{1}{2}((a+b)^n-(a-b)^n)$
- $e^x = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots,(-\infty<x<\infty)$
- $\ln(1+x) = x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots,(-1<x\le 1)$
- $\sqrt{1+x} = 1+\frac{x}{2}-\frac{x^2}{8}+\frac{2x^3}{32}-\frac{5x^4}{128}+\ldots,(-1\le x\le 1)$
- $\sin x = x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots,(-\infty<x<\infty)$
- $\cos x = 1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots,(-\infty<x<\infty)$
- $(x+a)^{-n} = \sum_{k=0}^{\infty}(-1)^k\binom{n+k-1}{k}x^ka^{-n-k}$
- $1/(1-x) = 1+x+x^2+x^3+\ldots$
- $1/(1-ax) = 1+ax+(ax)^2+(ax)^3+\ldots$
- $1/(1-x)^2 = 1+2x+3x^2+4x^3+\ldots$
- $1/(1-x)^3 = C(2,2)+C(3,2)x+C(4,2)x^2+C(5,2)x^3+\ldots$
- $1/(1-ax)^{k+1} = 1+C(1+k,k)(ax)+C(2+k,k)(ax)^2+C(3+k,k)(ax)^3+\ldots$
- $x(x+1)(1-x)^{-3} = 1+x+4x^2+9x^3+16x^4+25x^5+\ldots$

3.8 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k\cdot(m^2-n^2),\quad b = k\cdot(2mn),\quad c = k\cdot(m^2+n^2),$$

with $m>n>0$, $k>0$, $m\bot n$, and either m or n even.

3.9 Number Theory

- HCN: 1e6(240), 1e9(1344), 1e12(6720), 1e14(17280), 1e15(26880), 1e16(41472)
- $gcd(a,b,c,d,\ldots) = gcd(a,b-a,c-b,d-c,\ldots)$
- $gcd(a+k,b+k,c+k,d+k,\ldots) = gcd(a+k,b-a,c-b,d-c,\ldots)$
- Primitive root exists iff $n = 1,2,4,p^k,2\times p^k$, where p is an odd prime.
- If primtive root exists, there are $\phi(\phi(n))$ primtive roots of n.
- The numbers from 1 to n have in total $O(n\log\log n)$ unique prime factors.
- $x \equiv r_1 \mod m1$ and $x \equiv r_2 \mod m2$ has a solution iff $\gcd(m_1,m_2)|(r_1-r_2)$ Solution of $x^2 \equiv a(\mod p)$
- $ca \equiv cb \pmod m \iff a \equiv b \pmod{\frac{n}{\gcd(n,c)}}$
- $ax \equiv b \pmod m$ has a solution $\iff \gcd(a,m)|b$
- If $ax \equiv b \pmod m$ has a solution, then it has $gcd(a,m)$ solutions and they are separated by $\frac{m}{\gcd(a,m)}$
- $ax \equiv 1 \pmod m$ has a solution or a is invertible $\pmod m$ $\iff$ $\gcd(a,m) = 1$
- $x^2 \equiv 1 \pmod p$ then $x \equiv \pm 1 \pmod p$
- There are $\frac{p-1}{2}$ has no solution.
- There are $\frac{p-1}{2}$ has exaclty two solutions.
- When $p\%4 = 3$, $x \equiv \pm a^{\frac{p+1}{4}}$
- When $p\%8 = 5$, $x \equiv a^{\frac{p+3}{8}}$ *or* $x \equiv 2^{\frac{p-1}{4}}a^{\frac{p+3}{8}}$

3.9.1 Primes

$p = 962592769$ is such that $2^{21} \mid p-1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^{\times}$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

3.9.2 Estimates

$\sum_{d|n} d = O(n\log\log n)$.

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

3.9.3 Perfect numbers

$n > 1$ is called perfect if it equals sum of its proper divisors and 1. Even n is perfect iff $n = 2^{p-1}(2^p - 1)$ and $2^p - 1$ is prime (Mersenne's). No odd perfect numbers are yet found.

3.9.4 Carmichael numbers

A positive composite n is a Carmichael number ($a^{n-1} \equiv 1 \pmod n$ for all a: $\gcd(a, n) = 1$), iff n is square-free, and for all prime divisors p of n, $p - 1$ divides $n - 1$.

3.9.5 Totient

- If p is a prime $\phi(p^k) = p^k - p^{k-1} = p^k(1 - \frac{1}{p})$
- If a and b are relatively prime, $\phi(ab) = \phi(a)\phi(b)$
- $\phi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})(1 - \frac{1}{p_3})...(1 - \frac{1}{p_k})$
- Sum of coprime to $n = n * \frac{\phi(n)}{2}$
- If $n = 2^k, \phi(n) = 2^{k-1} = \frac{n}{2}$
- For a and b, $\phi(ab) = \phi(a)\phi(b)\frac{d}{\phi(d)}$
- $\phi(ip) = p\phi(i)$ whenever p is a prime and it divides i
- The number of $a(1 <= a <= N)$ such that $gcd(a, N) = d$ is $\phi(\frac{n}{d})$
- If $n > 2$, $\phi(n)$ is always even
- Sum of gcd, $\sum_{i=1}^{n} gcd(i, n) = \sum_{d|n} d\phi(\frac{n}{d})$
- Sum of lcm, $\sum_{i=1}^{n} nlcm(i, n) = \frac{n}{2}(\sum_{d|n}(d\phi(d)) + 1)$
- $\phi(1) = 1$ and $\phi(2) = 1$ which two are only odd ϕ
- $\phi(3) = 2$ and $\phi(4) = 2$ and $\phi(6) = 2$ which three are only prime ϕ
- Find minimum n such that $\frac{\phi(n)}{n}$ is maximum- Multiple of small primes- $2 * 3 * 5 * 7 * 11 * 13 * ...$

3.9.6 Mobius function

- $\sum_{d|n} \phi(d) = n$
- $\sum_{d|n} \mu(d) = [n = 1]$
- $\phi(n) = \sum_{d|n} \mu(d)\frac{n}{d}$.
- *Number of coprime tuples* $= \sum_i \mu(i) \cdot \mathrm{cnt}_i$
- *Sum of gcd of all tuples* $= \sum_i \phi(i) \cdot \mathrm{cnt}_i$
- *Sum of lcm of all tuples* $= \sum_i f(i) \cdot \mu(i) \cdot \mathrm{cnt}_i$
  - $f(i) = \frac{1}{i}\sum_{d|i} d \cdot \mu(d)$
- If for all $n \in N$, $F(n) = \sum_{d|n} f(d)$, then $f(n) = \sum_{d|n} \mu(d)F(\frac{n}{d})$, and vice versa.
- If f is multiplicative, then $\sum_{d|n} \mu(d)f(d) = \prod_{p|n}(1 - f(p))$, $\sum_{d|n} \mu(d)^2 f(d) = \prod_{p|n}(1 + f(p))$.

3.9.7 Legendre symbol

If p is an odd prime, $a \in \mathbb{Z}$, then $\left(\frac{a}{p}\right)$ equals 0, if $p|a$; 1 if a is a quadratic residue modulo p; and -1 otherwise. Euler's criterion: $\left(\frac{a}{p}\right) = a^{\left(\frac{p-1}{2}\right)} \pmod p$.

3.9.8 Jacobi symbol

If $n = p_1^{a_1} \cdots p_k^{a_k}$ is odd, then $\left(\frac{a}{n}\right) = \prod_{i=1}^{k}\left(\frac{a}{p_i}\right)^{k_i}$.

3.9.9 Primitive roots

If the order of g modulo m (min $n > 0$: $g^n \equiv 1 \pmod m$) is $\phi(m)$, then g is called a primitive root. If Z_m has a primitive root, then it has $\phi(\phi(m))$ distinct primitive roots. Z_m has a primitive root iff m is one of 2, 4, p^k, $2p^k$, where p is an odd prime. If Z_m has a primitive root g, then for all a coprime to m, there exists unique integer $i = \mathrm{ind}_g(a)$ modulo $\phi(m)$, such that $g^i \equiv a \pmod m$. $\mathrm{ind}_g(a)$ has logarithm-like properties: $\mathrm{ind}(1) = 0$, $\mathrm{ind}(ab) = \mathrm{ind}(a) + \mathrm{ind}(b)$.

If p is prime and a is not divisible by p, then congruence $x^n \equiv a \pmod p$ has $\gcd(n, p-1)$ solutions if $a^{(p-1)/\gcd(n,p-1)} \equiv 1 \pmod p$, and no solutions otherwise. (Proof sketch: let g be a primitive root, and $g^i \equiv a \pmod p$, $g^u \equiv x \pmod p$. $x^n \equiv a \pmod p$ iff $g^{nu} \equiv g^i \pmod p$ iff $nu \equiv i \pmod p$.)

3.9.10 Discrete logarithm problem

Find x from $a^x \equiv b \pmod m$. Can be solved in $O(\sqrt{m})$ time and space with a meet-in-the-middle trick. Let $n = \lceil\sqrt{m}\rceil$, and $x = ny - z$. Equation becomes $a^{ny} \equiv ba^z \pmod m$. Precompute all values that the RHS can take for $z = 0, 1, \ldots, n-1$, and brute force y on the LHS, each time checking whether there's a corresponding value for RHS.

3.9.11 Postage stamps/McNuggets problem

Let a, b be relatively-prime integers. There are exactly $\frac{1}{2}(a-1)(b-1)$ numbers *not* of form $ax + by$ $(x, y \ge 0)$, and the largest is $(a-1)(b-1) - 1 = ab - a - b$.

3.9.12 Fermat's two-squares theorem

Odd prime p can be represented as a sum of two squares iff $p \equiv 1 \pmod 4$. A product of two sums of two squares is a sum of two squares. Thus, n is a sum of two squares iff every prime of form $p = 4k + 3$ occurs an even number of times in n's factorization.

3.10 Twelve-fold Way

Balls	Urns	unrestricted	≤ 1	≥ 1
labeled	labeled	u^b	$(u)_b$	$u!S(b,u)$
unlabeled	labeled	$\binom{u+b-1}{b}$	$\binom{u}{b}$	$\binom{b-1}{u-1}$
labeled	unlabeled	$\sum_{i=1}^{u} S(b,i)$	$[b \le u]$	$S(b,u)$
unlabeled	unlabeled	$\sum_{i=1}^{u} p_i(b)$	$[b \le u]$	$p_u(b)$

3.11 Permutations

3.11.1 Cycles

Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n)\frac{x^n}{n!} = \exp\left(\sum_{n \in S}\frac{x^n}{n}\right)$$

3.11.2 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor\frac{n!}{e}\right\rfloor$$

3.11.3 Involutions

An involution is a permutation with maximum cycle length 2, and it is its own inverse.

$$a(n) = a(n-1) + (n-1)a(n-2)$$

$$a(0) = a(1) = 1$$

$$1, 1, 2, 4, 10, 26, 76, 232, 764, 2620, 9496, 35696, 140152$$

3.11.4 Burnside's 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

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

3.12 Partitions and subsets

3.12.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1, \; p(n) = \sum_{k \in \mathbb{Z}\setminus\{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

3.12.2 Partition Number

- Time Complexity: $O(n\sqrt{n})$

```
for (int i = 1; i <= n; ++i) {
  pent[2 * i - 1] = i * (3 * i - 1) / 2;
  pent[2 * i] = i * (3 * i + 1) / 2;
}
p[0] = 1;
for (int i = 1; i <= n; ++i) {
  p[i] = 0;
  for (int j = 1, k = 0; pent[j] <= i; ++j) {
    if (k < 2) p[i] = add(p[i], p[i - pent[j]]);
    else p[i] = sub(p[i], p[i - pent[j]]); ++k, k &= 3;
  }
}
```

- The number of partitions of a positive integer n into exactly k parts equals the number of partitions of n whose largest part equals k

$$p_k(n) = p_k(n-k) + p_{k-1}(n-1)$$

3.12.3 2nd Kaplansky's Lemma

The number of ways of selecting k objects, no two consecutive,from n labelled objects arrayed in a circle is $\frac{n}{k}\binom{n-k-1}{k-1} = \frac{n}{n-k}\binom{n-k}{k}$

3.12.4 Distinct Objects into Distinct Bins

- n distinct objects into r distinct bins $= r^n$

- Among n distinct objects, exactly k of them into r distincts bins $= \binom{n}{k}r^k$

- n distinct objects into r distinct bins such that each bin contains at least one object $= \sum_{i=0}^{r}(-1)^i\binom{r}{i}(r-i)^n$

3.13 Coloring

- The number of labeled undirected graphs with n vertices, $G_n = 2^{\binom{n}{2}}$
- The number of labeled directed graphs with n vertices, $G_n = 2^{n(n1)}$
- The number of connected labeled undirected graphs with n vertices, $C_n = 2^{\binom{n}{2}} - \frac{1}{n}\sum_{k=1}^{n-1} k\binom{n}{k}2^{\binom{n-k}{2}}C_k = 2^{\binom{n}{2}} - \sum_{k=1}^{n-1}\binom{n-1}{k-1}2^{\binom{n-k}{2}}C_k$
- The number of k-connected labeled undirected graphs with n vertices, $D[n][k] = \sum_{s=1}^{n}\binom{n-1}{s-1}C_s D[n-s][k-1]$
- Cayley's formula: the number of trees on n labeled vertices = the number of spanning trees of a complete graph with n labeled vertices $= n^{n-2}$
- Number of ways to color a graph using k color such that no two adjacent nodes have same color
  - Complete graph $= k(k-1)(k-2)...(k-n+1)$
  - Tree $= k(k-1)^{n-1}$
  - Cycle $= (k-1)^n + (-1)^n(k-1)$
- Number of trees with n labeled nodes: n^{n-2}

3.14 General purpose numbers

3.14.1 Eulerian numbers

a Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

$$E(n,k) = \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n$$

3.14.2 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1,1,2,5,15,52,203,877,4140,21147,\ldots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod p$$

3.14.3 Bernoulli numbers

$\sum_{j=0}^{m} \binom{m+1}{j} B_j = 0$. $B_0 = 1$, $B_1 = -\frac{1}{2}$. $B_n = 0$, for all odd $n \neq 1$.

3.14.4 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2} C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

- $C_n = 1,1,2,5,14,42,132,429,1430,4862,16796,58786,\ldots$
- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.
- **Non-Crossing Partitions:** Ways to partition a set of n elements such that no two blocks of the partition "cross" when visualized on a circle.
- Find the count of balanced parentheses sequences consisting of $n+k$ pairs of parentheses where the first k symbols are open brackets.

$$C_n^{(k)} = \frac{k+1}{n+k+1}\binom{2n+k}{n}$$

- Recursive formula of Catalan Numbers:

$$C_n^{(k)} = \frac{(2n+k-1)\cdot(2n+k)}{n\cdot(n+k+1)} C_{n-1}^{(k)}$$

3.14.5 Super Catalan numbers

The number of monotonic lattice paths of a $n \times n$-grid that do not touch the diagonal.

$$S(n) = \frac{3(2n-3)S(n-1) - (n-3)S(n-2)}{n}$$

$$S(1) = S(2) = 1$$

$$1,1,3,11,45,197,903,4279,20793,103049,518859$$

3.14.6 Motzkin numbers

Number of ways of drawing any number of nonintersecting chords among n points on a circle. Number of lattice paths from $(0,0)$ to $(n,0)$ never going below the x-axis, using only steps NE, E, SE.

$$M(n) = \frac{3(n-1)M(n-2) + (2n+1)M(n-1)}{n+2}$$

$$M(0) = M(1) = 1$$

$$1,1,2,4,9,21,51,127,323,835,2188,5798,15511,41835,113634$$

3.15 Narayana numbers

$$N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$$

- n pairs of balanced parenthesis with k nestings.
- number of lattice paths from $(0,0)$ to $(2n,0)$, with steps only north-east and southeast and k peaks.
- number of unlabeled ordered rooted trees with n edges and k leaves.
- number of non-crossing partition of a set with n elements into exactly k blocks.

3.15.1 Schröder numbers

Number of lattice paths from $(0,0)$ to (n,n) using only steps N, NE, E, never going above the diagonal. Number of lattice paths from $(0,0)$ to $(2n,0)$ using only steps NE, SE and double east EE, never going below the x-axis. Twice the Super Catalan number, except for the first term.

$$1,2,6,22,90,394,1806,8558,41586,206098$$

3.15.2 Lucas Number

Number of edge cover of a cycle graph C_n is L_n

$$L(n) = L(n-1) + L(n-2); L(0) = 2, L(1) = 1$$

3.16 Ballot Theorem

Suppose that in an election, candidate A receives a votes and candidate B receives b votes, where a kb for some positive integer k. Compute the number of ways the ballots can be ordered so that A maintains more than k times as many votes as B throughout the counting of the ballots.
The solution to the ballot problem is $\frac{a-kb}{a+b} \times C(a+b,a)$

3.17 Classical Problems

- $F(n,k) =$ number of ways to color n objects using exactly k colors. Let $G(n,k)$ be the number of ways to color n objects using no more than k colors. Then, $F(n,k) = G(n,k) - C(k,1) * G(n,k-1) + C(k,2) * G(n,k-2) - C(k,3) * G(n,k-3)$...
- Number of ways to divide n persons into $\frac{n}{k}$ equal groups, each having size k is: $\frac{n!}{k!^{\frac{n}{k}}\left(\frac{n}{k}\right)!} = \prod_{n \geq k}^{n-=k} \binom{n-1}{k-1}$
- Number of ways to choose n ids from 1 to b such that every id has distance at least k is $\binom{b-(n-1)(k-1)}{n}$

Determining G(n, k) :
Suppose, we are given a 1 * n grid. Any two adjacent cells can not have same color. Then, $G(n,k) = k * ((k-1)^(n-1))$
If no such condition on adjacent cells. Then, $G(n,k) = k^n$

3.18 Matching Formula

3.18.1 Normal Graph

MM + MEC = n (exculding vertex), IS + VC = G, MIS + MVC = G

3.18.2 Bipartite Graph

MIS = n - MBM, MVC = MBM, MEC = n - MBM

3.18.3 Grundy numbers

For a two-player, normal-play (last to move wins) game on a graph (V,E): $G(x) = \text{mex}(\{G(y) : (x,y) \in E\})$. x is losing iff $G(x) = 0$.

3.18.4 Sums of games

- *Player chooses a game and makes a move in it.* Grundy number of a position is xor of grundy numbers of positions in summed games.
- *Player chooses a non-empty subset of games (possibly, all) and makes moves in all of them.* A position is losing iff each game is in a losing position.
- *Player chooses a proper subset of games (not empty and not all), and makes moves in all chosen ones.* A position is losing iff grundy numbers of all games are equal.
- *Player must move in all games, and loses if can't move in some game* A position is losing if any of the games is in a losing position.

3.18.5 Misère Nim

A position with pile sizes $a_1, a_2, \ldots, a_n \geq 1$, not all equal to 1, is losing iff $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$ (like in normal nim.) A position with n piles of size 1 is losing iff n is *odd*.

3.19 Tree Hashing

$f(u) = sz[u] * \sum_{i=0} f(v) * p^i$; $f(v)$ are sorted $f(child) = 1$

3.20 Permutation

To maximize the sum of adjacent differences of a permuation, it is necessary and sufficient to place the smallest half numbers in odd position and the greatest half numbers in even position. Or, vice versa.

3.21 String

- If the sum of length of some strings is N, there can be at most $\sqrt{N}$ distinct length.
- A Text can have at most $O(N \times \sqrt{N})$ distinct substrings that match with given patterns where the sum of the length of the given patterns is N.
- Period = n % (n - pi.back() == 0)? n - pi.back(): n
- The first $(period)$ cyclic rotations of a string are distinct. Further cyclic rotations repeat the previous strings.
- S is a palindrome if and only if it's period is a palindrome.
- If S and T are palindromes, then the periods of S and T are same if and only if $S+T$ is a palindrome.

3.22 Bit

- (a xor b) and (a + b) has the same parity
- (a + b) = (a xor b) + 2 (a & b) = (a | b) + (a & b)
- gcd(a, b) <= a - b <= xor(a, b)

3.23 Convolution

- Hamming Distance: Replace 0 with -1
- SQRT Decomposition: Find block size, B = sqrt(8 * n)
- Pattern Matching:

1. $A(x) = a_0x^0 + a_1x^1 + \cdots + a_{n-1}x^{n-1}$, $n = |T|, a_i = \cos(\alpha_i) + i\sin(\alpha_i)$, $\alpha_i = \frac{2\pi T[i]}{26}$.
2. $B(x) = b_0x^0 + b_1x^1 + \cdots + b_{m-1}x^{m-1}$, $m = |P|, b_i = \cos(\beta_i) - i\sin(\beta_i)$, $\beta_i = \frac{2\pi P[m-i-1]}{26}$.
3. $C(x) = A(x) \times B(x)$, if $c_{m-1+i} = m$ then pattern appears in the text at position i.

- Pattern Matching with Wildcards: set $b_i = 0$ if $P[m-i-1] = *$. If x is the number of wildcards in P, then we will have a match of P in T at index i if $c_{m-1+i} = m - x$.