Hello, BOJ 2025!

riroan

March 2, 2025

Contents

ARE YOU TYPE CORRECTLY?
CHECK TIME COMPLEXITY OF YOUR ALGORITHM!
CHECK YOUR MAXIMUM ARRAY SIZE!

1 Graph

1.1 Dijkstra

```
import heapq
def dijkstra(start):
    distances = [0] * n
    for i in range(n):
        distances[i] = INF
    distances[start] = 0
    q = []
    heapq.heappush(q, [distances[start], start])
    while q:
        current_distance, current_destination = heapq.heappop(q)
        if distances[current_destination] < current_distance:
            continue
        for new_destination in g[current_destination]:
            new_distance = 1
            distance = current_distance + new_distance
            if distance < distances[new_destination]:
                distances[new_destination] = distance
                heapq.heappush(q, [distance, new_destination])
    return distances
```

1.2 Floyd Warshall

```
n,m = map(int, input().split()) # n : #vertex, m : #edge
arr = [[INF] * n for i in range(n)]
for i in range(n):
    arr[i][i] = 0
for i in range(m):
    a,b,c=map(int, input().split())
    arr[a-1][b-1] = c
    arr[b-1][a-1] = c

for k in range(n):
    for i in range(n):
        for j in range(n):
            arr[i][j] = min(arr[i][j],arr[i][k]+arr[k][j])
```

1.3 Topological Sort

```
from graphlib import TopologicalSorter, CycleError
N, M = mis()
# (Node, Preceding-Node)
g = {x+1:[] for x in range(N)}
for _ in range(M):
    a, b = mis()
    g[b].append(a)
try:
    ts = TopologicalSorter(g)
    print(*[x for x in ts.static_order()])
except CycleError:
    print(0)
```

1.4 Euler Tour Technique(C++)

```
    vector<int> start(n), end(n);
    int cur = -1, root = 0;
    auto dfs = [&](auto self, int x, int p) -> void
    {
        start[x] = ++cur;
        for (auto i : arr[x])
        {
            if (i == p)
                continue;
            self(self, i, x);
        }
        end[x] = cur;
    };
    dfs(dfs, root, root);
```

1.5 SCC(C++)

```
struct SCC
{
    int n;
    vector<vector<int>> g;
    vector<int> stk;
    vector<int> dfn, low, iscc;
    vector<vector<int>> scc;
    vector<pair<int, int>> edges;
    vector<vector<int>> arr; // SCC graph
    int cur, cnt;

    SCC() {}
    SCC(int n)
    {
        init(n);
    }

    void init(int n)
    {
        this->n = n;
        g.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        iscc.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

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

    int dfs(int x)
    {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);
        for (auto y : g[x])
            if (dfn[y] == -1)
                low[x] = min(low[x], dfs(y));
            else if (iscc[y] == -1)
                low[x] = min(low[x], dfn[y]);

        if (dfn[x] == low[x])
        {
            int y;
            do
            {
                y = stk.back();
                iscc[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
        return low[x];
    }

    void build()
    {
        for (int i = 0; i < n * 2; i++)
            if (dfn[i] == -1)
                dfs(i);

        scc.resize(cnt);
        for (int i = 0; i < n; i++)
            scc[iscc[i]].push_back(i);
        sort(scc.begin(), scc.end());
        arr.resize(cnt);

        for (auto [x, y] : edges)
            if (iscc[x] != iscc[y])
                arr[iscc[x]].push_back(iscc[y]);
    }
};

struct SAT : public SCC
{
    SAT(int n)
    {
        n++;
        init(n);
    }
    void init(int n)
    {
        this->n = n;
        g.assign(n * 2, {});
        dfn.assign(n * 2, -1);
        low.resize(n * 2);
        iscc.assign(n * 2, -1);
        stk.clear();
        cur = cnt = 0;
    }
    int apply_not(int a)
    {
        return a % 2 ? a - 1 : a + 1;
    }

    void add(int u, int v)
    {
        u = (u < 0 ? -(u + 1) * 2 : u * 2 - 1);
        v = (v < 0 ? -(v + 1) * 2 : v * 2 - 1);
        g[apply_not(u)].push_back(v);
        g[apply_not(v)].push_back(u);
    }

    auto check()
    {
        for (int i = 0; i < n; i++)
            if (iscc[i * 2] == iscc[i * 2 + 1])
                return 0;
        return 1;
    }
};
```

1.6 BCC(C++)

```
struct BCC
{
    int n, cur, cpiv;
    vector<int> dfn, low, par, vis;
    vector<vector<int>> g, bcc, ibcc;
    BCC(int n)
    {
        this->n = n;
        dfn.resize(n);
        low.resize(n);
        par.resize(n);
        vis.resize(n);
        g.resize(n, {});
        bcc.resize(n, {});
        cur = 0;
        cpiv = 0;
    }

    void add(int a, int b)
    {
        g[a].push_back(b);
        g[b].push_back(a);
    }

    int dfs(int x, int p)
    {
        dfn[x] = low[x] = ++cur;
        par[x] = p;
        for (auto w : g[x])
        {
            if (w == p)
                continue;
            if (!dfn[w])
```

```
                low[x] = min(low[x], dfs(w, x));
            else
                low[x] = min(low[x], dfn[w]);
        }
        return low[x];
    }
    void color(int x, int c)
    {
        if (c)
            bcc[x].push_back(c);
        vis[x] = 1;
        for (auto w : g[x])
        {
            if (vis[w])
                continue;
            if (dfn[x] <= low[w])
            {
                bcc[x].push_back(++cpiv);
                color(w, cpiv);
            }
            else
                color(w, c);
        }
    }

    void build()
    {
        for (int i = 0; i < n; i++)
            if (!dfn[i])
                dfs(i, 0);
        for (int i = 0; i < n; i++)
            if (!vis[i])
                color(i, 0);
        ibcc.resize(cpiv);
        for (int i = 0; i < n;i++)
            for(auto j : bcc[i])
                ibcc[j - 1].push_back(i);
    }

    auto get_articulation_point()
    {
        vector<int> res;
        for (int i = 0; i < n; i++)
            if (bcc[i].size() > 1)
                res.push_back(i);
        return res;
    }

    auto get_articulation_bridge()
    {
        vector<pair<int, int>> res;
        for (auto i : ibcc)
            if (i.size() == 2)
                res.push_back(minmax(i[0], i[1]));
        sort(res.begin(), res.end());
        return res;
```

```
    }
};
```

1.7 Dinic(C++)

```
int N, M, S, E, lv[MAX], w[MAX], ans;
struct Edge {
  int to, c, rev;
  Edge(int to, int c, int rev)
    :to(to), c(c), rev(rev) {}
};
vector<Edge> v[MAX];
void addEdge(int s, int e, int c) {
  v[s].emplace_back(e, c, v[e].size());
  v[e].emplace_back(s, 0, v[s].size() - 1);
}
bool bfs() {
  memset(lv, -1, sizeof(lv));
  lv[S] = 0;
  queue<int> q;
  q.push(S);
  while (!q.empty()) {
    int cur = q.front();
    q.pop();
    for (auto i : v[cur]) {
      if (i.c && lv[i.to] == -1) {
        lv[i.to] = lv[cur] + 1;
        q.push(i.to);
      }
    }
  }
  return lv[E] != -1;
}
int dfs(int cur, int c) {
  if (cur == E)return c;
  for (; w[cur] < v[cur].size(); w[cur]++) {
    Edge& e = v[cur][w[cur]];
    if (!e.c || lv[e.to] != lv[cur] + 1)
      continue;
    int f = dfs(e.to, min(c, e.c));
    if (f > 0) {
      e.c -= f;
      v[e.to][e.rev].c += f;
      return f;
    }
  }
  return 0;
}
```

1.8 Bipartite Matching(C++)

```
int N, M, d[MAX];
bool used[MAX];
vector<int> v[MAX];
bool dfs(int x) {
  for (auto i : v[x]) {
    if (used[i])
      continue;
    used[i] = true;
    if (!d[i] || dfs(d[i])) {
      d[i] = x;
      return true;
    }
  }
  return false;
}
```

1.9 Dijkstra + DP(C++)

```
# BOJ 10217 KCM Travel
int N, M, K, d[MAX][MAXC];  // cost memoization
vector<pii> v[MAX];
int main() {
  cin.tie(0);
  cout.tie(0);
  ios::sync_with_stdio(false);

  int t;
  cin >> t;
  while (t--) {
    cin >> N >> M >> K;
    for (auto& i : v) {
      i.clear();
    }
    for (int i = 0; i < K; i++) {
      int s, e, cost, time;
      cin >> s >> e >> cost >> time;
      v[s].push_back({ {time, e}, cost });
    }
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({ {0, 1}, 0 });
    for (int i = 0; i < MAX; i++) {
      for (int j = 0; j < MAXC; j++) {
        d[i][j] = INF;
      }
    }
    d[1][0] = 0;
    while (!pq.empty()) {
      int time = pq.top().first.first;
      int cur = pq.top().first.second;
      int cost = pq.top().second;
      pq.pop();
      if (cost > M || d[cur][cost] < time)
      continue;
      for (auto i : v[cur]) {
        int nTime = i.first.first + time;
        int nCost = i.second + cost;
        int next = i.first.second;
        if (nCost <= M && nTime < d[next][nCost]) {
          // No -> 3120ms / Yes -> 260ms
          for (int j = nCost + 1; j <= M; j++) {
            if (d[next][j] <= nTime)
            break;
            d[next][j] = nTime;
          }
          d[next][nCost] = nTime;
          pq.push({ {nTime, next}, nCost });
        }
      }
    }
    int ans = INF;
    for (int i = 0; i <= M; i++) {
      ans = min(ans, d[N][i]);
    }
    if (ans >= INF)
    cout << "Poor KCM\n";
    else
    cout << ans << "\n";
  }
}
```

1.10 Check Bipartite Graph(C++)

```
int N, M, p[MAX];
map<int, int> m;
int find(int a) {
  if (a == p[a])return a;
  return p[a] = find(p[a]);
}

bool merge(int a, int b) {
  a = find(a);
  b = find(b);
  if (a == b)return false;
  if (a > b)swap(a, b);
  p[b] = a;
  return true;
}

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin >> N >> M;
  for (int i = 1; i <= N * 2; i++)p[i] = i;
  while (M--) {
    char ch;
    int n1, n2;
    cin >> ch >> n1 >> n2;
    if (ch == 'S') {
      merge(n1, n2);
      merge(n1 + N, n2 + N);
    }
    else {
      merge(n1, n2 + N);
      merge(n2, n1 + N);
    }
  }
  for (int i = 1; i <= N; i++) {
    if (find(i) == find(i + N)) {
      cout << 0;
      return 0;
    }
  }
  for (int i = 1; i <= N; i++) {
    merge(i, i + N);
  }
  for (int i = 1; i <= N; i++) {
    m[find(i)]++;
  }
  cout << 1;
  for (int i = 0; i < m.size(); i++) {
    cout << 0;
  }
}
```

1.11 Bellman-Ford(C++)

```
vector<pair<int, ll>> v[501];
ll d[501];
int main(){
  cin.tie(0);
  cout.tie(0);
  ios::sync_with_stdio(false);

  int n, m;
  cin>>n>>m;
  for(int i=0;i<m;i++){
    int a,b;
    ll c;
    cin>>a>>b>>c;
    v[a].push_back({b,c});
  }
  for(int i=2;i<=n;i++){
    d[i]=INF;
  }
  bool mCycle=false;
  for(int i=1;i<=n;i++){
    for(int j=1;j<=n;j++){
      for(pair<int, ll> p: v[j]){
        int next=p.first;
        ll dis=d[j]+p.second;
        if(d[j]!=INF&&d[next]>dis){
          d[next]=dis;
          if(i==n)
          mCycle=true;
        }
      }
    }
  }
  if(mCycle)
  cout<<"-1\n";
  else{
    for(int i=2;i<=n;i++){
      if(d[i]==INF)
      cout<<"-1\n";
      else
      cout<<d[i]<<"\n";
    }
  }
}
```

1.12 HLD(C++)

```
struct HLD
{
    vector<int> sz, d, p, top, in, out;
    vector<vector<int>> arr;
    SegmentTree<int> st;
    map<pair<int, int>, int> edges;
    int n, pv = 0;

    HLD(int n)
    {
        this->n = n;
        sz.resize(n);
        d.resize(n);
        p.resize(n);
        top.resize(n);
        in.resize(n);
        out.resize(n);
        arr.resize(n);
    }

    void add(int u, int v, int w = 1)
    {
        arr[u].push_back(v);
        arr[v].push_back(u);
        edges[minmax(u, v)] = w;
    }

    void make(int root = 0)
    {
        top[root] = root;
        d[root] = 0;
        p[root] = -1;
        dfs1(root);
        dfs2(root);
        vector<int> brr(n);
        for (int i = 0; i < n; i++)
            for (auto j : arr[i])
                brr[in[j]] = edges[minmax(i, j)];
        st = SegmentTree<int>(
            brr, [](int a, int b)
            { return a + b; },
            0LL);
    }

    void dfs1(int x = 0)
    {
        if (p[x] != -1)
            arr[x].erase(find(arr[x].begin(), arr[x].end(), p[x]));

        sz[x] = 1;
        for (auto &i : arr[x])
        {
            d[i] = d[x] + 1;
            p[i] = x;
            dfs1(i);
            sz[x] += sz[i];
            if (sz[i] > sz[arr[x][0]])
                swap(i, arr[x][0]);
        }
    }

    void dfs2(int x = 0)
    {
        in[x] = pv++;
        for (auto i : arr[x])
        {
            top[i] = i == arr[x][0] ? top[x] : i;
            dfs2(i);
        }
        out[x] = pv;
    }

    auto lca(int a, int b)
    {
        int ret = 0;
        while (top[a] ^ top[b])
        {
            if (d[top[a]] > d[top[b]])
                a = p[top[a]];
            else
                b = p[top[b]];
        }
        return d[a] < d[b] ? a : b;
    }

    auto unit_dist(int a, int b)
    {
        return d[a] + d[b] - 2 * d[lca(a, b)];
    }

    auto dist(int a, int b)
    {
        int ret = 0;
        while (top[a] != top[b])
        {
            if (d[top[a]] < d[top[b]])
                swap(a, b);
            ret += st.query(in[top[a]], in[a]);
            a = p[top[a]];
        }
        if (d[a] > d[b])
            swap(a, b);
        ret += st.query(in[a] + 1, in[b]);
        return ret;
    }

    auto query(int a, int b){
        // do something
    }

    bool isAncestor(int a, int b)
    {
        return in[a] <= in[b] && in[b] < out[a];
    }

    auto rootedLCA(int a, int b, int c)
    {
        return lca(a, b) ^ lca(b, c) ^ lca(c, a);
    }
};
```

1.13 Push Relabel(C++)

```
class PushRelabel
{
public:
    const int INF = 1LL << 60;
    int n;
    vector<vector<int>> flow, capacity;
    queue<int> q;
    vector<int> ex, h;

    PushRelabel(int n)
    {
        this->n = n;
        flow.resize(n);
        capacity.resize(n);
        ex.resize(n);
        h.resize(n);
        for (int i = 0; i < n; i++)
        {
            flow[i].resize(n);
            capacity[i].resize(n);
        }
    }

    void add(int u, int v, int t)
    {
        capacity[u][v] = t;
    }

    void push(int u, int v)
    {
        int d = min(ex[u], capacity[u][v] - flow[u][v]);
        flow[u][v] += d;
        flow[v][u] -= d;
        ex[u] -= d;
        ex[v] += d;
        if (d > 0 && ex[v] == d)
            q.push(v);
    }

    void relabel(int u, int &v)
    {
        int d = INF;
        for (int i = 0; i < n; i++)
            if (capacity[u][i] - flow[u][i] > 0)
                d = min(d, h[i]);
        if (d < INF)
            h[u] = d + 1;
        v = 0;
    }

    int max_flow(int s, int t)
    {
        h[s] = n;
        ex[s] = INF;
        for (int i = 0; i < n; i++)
            if (i != s)
                push(s, i);
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            if (u == s || u == t)
                continue;
            int v = 0;
            while (ex[u])
            {
                if (v < n)
                    if (capacity[u][v] - flow[u][v] > 0 && h[u] > h[v])
                        push(u, v);
                    else
                        v++;
                else
                    relabel(u, v);
            }
        }
        int ret = 0;
        for (int i = 0; i < n; i++)
            ret += flow[i][t];
        return ret;
    }
};
```

1.14 MCMF(C++)

```
template <class T>
struct MinCostFlow
{
    struct _Edge
    {
        int to;
        T cap;
        T cost;
        _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_), cost(cost_) {}
    };
    int n;
    vector<_Edge> e;
    vector<vector<int>> g;
    vector<T> h, dis;
    vector<int> pre;
    bool dijkstra(int s, int t)
    {
        dis.assign(n, numeric_limits<T>::max());
        pre.assign(n, -1);
        priority_queue<pair<T, int>, vector<pair<T, int>>, greater<pair<T, int>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty())
        {
            T d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] != d)
            {
                continue;
            }
            for (int i : g[u])
            {
                int v = e[i].to;
                T cap = e[i].cap;
                T cost = e[i].cost;
                if (cap > 0 && dis[v] > d + h[u] - h[v] + cost)
                {
                    dis[v] = d + h[u] - h[v] + cost;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != numeric_limits<T>::max();
    }
    MinCostFlow() {}
    MinCostFlow(int n_)
    {
        init(n_);
    }
    void init(int n_)
    {
        n = n_;
        e.clear();
        g.assign(n, {});
    }
    void addEdge(int u, int v, T cap, T cost)
    {
        g[u].push_back(e.size());
        e.emplace_back(v, cap, cost);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -cost);
    }
    pair<T, T> flow(int s, int t)
    {
        T flow = 0;
        T cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t))
        {
            for (int i = 0; i < n; ++i)
                h[i] += dis[i];
            T aug = numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].to)
                aug = min(aug, e[pre[i]].cap);
            for (int i = t; i != s; i = e[pre[i] ^ 1].to)
            {
                e[pre[i]].cap -= aug;
                e[pre[i] ^ 1].cap += aug;
            }
            flow += aug;
            cost += aug * h[t];
        }
        return make_pair(flow, cost);
    }
    struct Edge
    {
        int from;
        int to;
        T cap;
        T cost;
        T flow;
    };
    vector<Edge> edges()
    {
        vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2)
        {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.cost = e[i].cost;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

2 Data Structure

2.1 Disjoint Set

```
class DisjointSet:
    def __init__(self, n):
        self.f = [i for i in range(n)]
        self.siz = [1]*n

    def get(self, x):
        while x!=self.f[x]:
            self.f[x] = self.f[self.f[x]]
            x = self.f[x]
        return x

    def unite(self, x, y):
        x = self.get(x)
        y = self.get(y)
        if x == y:
            return False
        self.siz[x] += self.siz[y]
        self.f[y] = x
        return True

    def size(self, x):
        return self.siz[self.get(x)]
```

2.2 MergeSort Tree(C++)

```
struct Node
{
    int n;
    vector<int> arr;
    Node() : n(0) {}
    Node(const vector<int> &_arr) : n(_arr.size()), arr(_arr) {}
    Node(const vector<int> &&_arr) : n(_arr.size()), arr(_arr) {}

    Node operator+(const Node &other)
    {
        vector<int> res;
        int l = 0, r = 0;
        while (l < n && r < other.n)
            res.push_back(arr[l] < other.arr[r] ? arr[l++] : other.arr[r++]);
        while (l < n)
            res.push_back(arr[l++]);
        while (r < other.n)
            res.push_back(other.arr[r++]);
        return Node(res);
    }

    int count_greater_than_k(int k)
    {
        // greater_than_equal: upper_bound -> lower_bound
        return n - (ranges::upper_bound(arr, k) - arr.begin());
    }

    int count_less_than_k(int k)
    {
        // less_than_equal: lower_bound -> upper_bound
        return ranges::lower_bound(arr, k) - arr.begin();
    }
};

struct MergeSortTree
{
    int n;
    vector<Node> arr;
    MergeSortTree(const vector<int> &_arr)
    {
        n = _arr.size();
        arr = vector<Node>(4 * n);
        init(_arr, 1, 0, n - 1);
    }

    Node init(const vector<int> &_arr, int node, int l, int r)
    {
        if (l == r)
```

```
            return arr[node] = Node(vector<int>({_arr[l]}));
        int mid = l + r >> 1;
        return arr[node] = init(_arr, node * 2, l, mid) + init(_arr, node * 2 + 1, mid + 1,
        r);
    }

    int count_greater_than_k(int left, int right, int k, int node = 1, int l = 0, int r =
    -1)
    {
        if (r == -1)
            r = n - 1;
        if (right < l || r < left)
            return 0;
        if (left <= l && r <= right)
            return arr[node].count_greater_than_k(k);
        int mid = l + r >> 1;
        return count_greater_than_k(left, right, k, node * 2, l, mid) +
        count_greater_than_k(left, right, k, node * 2 + 1, mid + 1, r);
    }

    int count_less_than_k(int left, int right, int k, int node = 1, int l = 0, int r = -1)
    {
        if (r == -1)
            r = n - 1;
        if (right < l || r < left)
            return 0;
        if (left <= l && r <= right)
            return arr[node].count_less_than_k(k);
        int mid = l + r >> 1;
        return count_less_than_k(left, right, k, node * 2, l, mid) + count_less_than_k(left,
        right, k, node * 2 + 1, mid + 1, r);
    }

    int find_kth(int left, int right, int k)
    {
        int l = -(1LL << 62), r = -1;
        while (l + 1 < r)
        {
            int x = l + r >> 1;
            if (count_less_than_k(left, right, x) < k)
                l = x;
            else
                r = x;
        }

        return l;
    }
};
```

2.3 Trie

```
class Trie:
    def __init__(self):
        self.root = {}

    def insert(self, s):
        cur_node = self.root
        for c in s:
            if c not in cur_node:
                cur_node[c] = {}
            cur_node = cur_node[c]
        cur_node["*"] = s

    def search(self, s):
        cur_node = self.root
        for c in s:
            if c in s:
                cur_node = cur_node[c]
            else:
                return False
        return "*" in cur_node
```

2.4 XOR Trie

```
ans=0
class Trie:
    def __init__(self):
        self.children = [None, None]
        self.cnt = 0
        self.end = False

    def insert(self, x, ix=0):
        self.cnt += 1
        if ix == m:
            self.end = True
            return
        if self.children[x[ix]] == None:
            self.children[x[ix]] = Trie()
        self.children[x[ix]].insert(x, ix + 1)

  # change below
    def query(self, x, ix=0): # #(less than x)
        global ans
        if self.end:
            return
        if k[ix] == 1:
            if self.children[x[ix]] != None:
                ans += self.children[x[ix]].cnt
            if self.children[1 - x[ix]] != None:
                self.children[1 - x[ix]].query(x, ix + 1)
        else:
            if self.children[x[ix]] != None:
                self.children[x[ix]].query(x, ix + 1)
```

2.5 Policy Based Data Structure(C++)

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace ¨gnu˙pbds;
typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>
```

```
ordered_set;

int main(){
    ordered_set X;

    X.insert(16);
    X.insert(1);
    X.insert(4);
    X.insert(2);

    cout<<*X.find_by_order(0)<<endl; // 1
    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<*X.find_by_order(2)<<endl; // 4
    cout<<*X.find_by_order(3)<<endl; // 16
    cout<<*X.find_by_order(-1)<<endl; // 0 : invalid index
    cout<<*X.find_by_order(5)<<endl;

    cout<<X.order_of_key(1)<<endl; // #(less than 1) : 0
    cout<<X.order_of_key(4)<<endl; // #(less than 4) : 2
    cout<<X.order_of_key(400)<<endl; // #(less than 400) : 4
}
```

2.6 Segment Tree(C++)

```
template <typename T>
class SegmentTree
{
public:
    int n;
    vector<T> arr;
    function<T(T, T)> func;
    T basis;
    SegmentTree(vector<T> &brr, function<T(T, T)> f, T b)
    {
        n = brr.size();
        arr = vector<T>(n * 2);
        func = f;
        basis = b;
        init(brr, 0, n - 1, 1);
    }
    void init(vector<T> &brr, int left, int right, int node)
    {
        for (int i = 0; i < n; i++)
            arr[i + n] = brr[i];
        for (int i = n - 1; i > 0; --i)
            arr[i] = func(arr[i << 1], arr[i << 1 | 1]);
    }
    T query(int left, int right)
    {
        int res = basis;
        for (left += n, right += n + 1; left < right; left >>= 1, right >>= 1)
        {
            if (left & 1)
                res = func(res, arr[left++]);
            if (right & 1)
                res = func(res, arr[--right]);
        }
        return res;
    }

    void update(int p, T newValue)
    {
        for (arr[p += n] = newValue; p > 1; p >>= 1)
            arr[p >> 1] = func(arr[p], arr[p ^ 1]);
    }
};
```

2.7 Lazy Segment Tree(C++)

```
class Node{
public:
    int value=0;
    int lazy=0;
};

class LST
{
public:
    int n;
    vector<Node> tree;

    LST(const vector<int> &arr)
    {
        n = arr.size();
        tree = vector<Node>(4 * n);
        init(arr, 0, n - 1, 1);
    }

    auto func(int a, int b)
    {
        return a + b;
    }

    Node init(const vector<int> &arr, int left, int right, int node)
    {
        if (left == right){
            tree[node].value = arr[left];
            return tree[node];
        }
        int mid = (left + right) / 2;
        Node l = init(arr, left, mid, node * 2);
        Node r = init(arr, mid + 1, right, node * 2 + 1);
        tree[node].value = func(l.value, r.value);
        return tree[node];
    }

    void propagate(int node, int nodeLeft, int nodeRight)
    {
        if (tree[node].lazy)
        {
            if (nodeLeft != nodeRight)
            {
                tree[node * 2].lazy = func(tree[node * 2].lazy, tree[node].lazy);
                tree[node * 2 + 1].lazy = func(tree[node * 2 + 1].lazy, tree[node].lazy);
            }
            tree[node].value = func(tree[node].value, tree[node].lazy * (nodeRight -
            nodeLeft + 1));
            tree[node].lazy = 0;
        }
    }

    int query(int left, int right)
    {
        return query(left, right, 1, 0, n - 1);
    }

    int query(int left, int right, int node, int nodeLeft, int nodeRight)
    {
        propagate(node, nodeLeft, nodeRight);
        if (right < nodeLeft || nodeRight < left)
            return 0;
        if (left <= nodeLeft && nodeRight <= right)
            return tree[node].value;
        int mid = (nodeLeft + nodeRight) / 2;
        return func(query(left, right, node * 2, nodeLeft, mid), query(left, right, node * 2
        + 1, mid + 1, nodeRight));
    }

    void update(int left, int right, int newValue)
    {
        update(left, right, newValue, 1, 0, n - 1);
    }

    void update(int left, int right, int newValue, int node, int nodeLeft, int nodeRight)
    {
        propagate(node, nodeLeft, nodeRight);
        if (right < nodeLeft || nodeRight < left)
            return;
        if (left <= nodeLeft && nodeRight <= right)
        {
            tree[node].lazy = func(tree[node].lazy, newValue);
            propagate(node, nodeLeft, nodeRight);
            return;
        }
        int mid = (nodeLeft + nodeRight) / 2;
        update(left, right, newValue, node * 2, nodeLeft, mid);
        update(left, right, newValue, node * 2 + 1, mid + 1, nodeRight);
        tree[node].value = func(tree[node * 2].value, tree[node * 2 + 1].value);
    }
};
```

2.8 Fenwick Tree + Inversion Counting(C++)

```
int N,a[MAX],tree[MAX];
ll query(int i){
  ll ret=0;
  for(;i;i-=i&-i){
    ret+=1LL*tree[i];
  }
  return ret;
}
void update(int i, int val){
  for(;i<=N;i+=i&-i){
    tree[i]+=val;
  }
}
int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin>>N;
  ll ans=0;
  for(int i=1;i<=N;i++){
    cin>>a[i];
    ans+=query(N)-query(a[i]);
    update(a[i],1);
  }
  cout<<ans;
}
```

2.9 Splay Tree(C++)

```
struct Node
{
    Node *p;
    array<Node *, 2> child{};
    int cnt, value, sum, ma, mi, lazy;
    bool inv;
    Node(int value = 0)
        : cnt(1), value(value), sum(value), ma(value), mi(value), inv(false), lazy(0)
    {
        p = nullptr;
    }

    inline bool is_root()
    {
        return p == nullptr;
    }

    inline bool pos()
    {
        return p->child[1] == this;
    }

    inline void rev()
    {
        swap(child[0], child[1]);
    }
};

struct SplayTree
{
    Node *root;
    SplayTree()
    {
        root = nullptr;
    }
    void pull(Node *x)
    {
        x->cnt = 1;
        x->sum = x->value;
        x->ma = x->value;
        x->mi = x->value;
        for (Node *i : x->child)
        {
            if (i)
            {
                x->cnt += i->cnt;
                x->sum += i->sum;
                x->ma = max(x->ma, i->ma);
                x->mi = min(x->mi, i->mi);
            }
        }
    }

    void push(Node *x)
    {
        if (!x)
            return;
        if (x->inv)
        {
            x->rev();
            x->inv = false;
            for (Node *i : x->child)
                if (i)
                    i->inv ^= 1;
        }
        if (x->lazy)
        {
            x->value += x->lazy;
            for (Node *i : x->child)
                if (i)
                {
                    i->lazy += x->lazy;
                    i->sum += i->cnt * x->lazy;
                }
            x->lazy = 0;
        }
    }

    void rotate(Node *x)
    {
        auto p = x->p;
        Node *y;
        push(p);
        push(x);
        bool ix = x->pos();
        p->child[ix] = y = x->child[!ix];
        x->child[!ix] = p;

        x->p = p->p;
        p->p = x;
        if (y)
            y->p = p;
        if (x->p)
            x->p->child[p != x->p->child[0]] = x;
        else
            root = x;
        pull(p);
        pull(x);
    }

    void splay(Node *x, Node *y = nullptr)
    {
        if (!x)
            return;
        while (x->p != y)
        {
            Node *p = x->p;
            if (p->p == y)
            {
                rotate(x);
                break;
            }
            if (p->pos() == x->pos())
                rotate(p);
            rotate(x);
        }
        if (!y)
            root = x;
    }

    void reverse(int l, int r)
    {
        Node *x = gather(++l, ++r);
        if (x)
            x->inv ^= 1;
    }

    Node *gather(int s, int e)
    {
        find_kth(e + 1);
        auto tmp = root;
        find_kth(s - 1);
        splay(tmp, root);
        return root->child[1]->child[0];
    }

    void print()
    {
        if (root == nullptr)
            return;
        int sz = root->cnt;
        cout << "VALUES-" << sz << ": ";
        for (int i = 0; i < sz; i++)
        {
            Node *x = find_kth(i);
            cout << x->value << " ";
        }
        cout << endl;
    }

    void insert(int v, int pos = -1)
    {
        Node *node = new Node(v);
        if (root == nullptr)
        {
            root = node;
            return;
        }
        Node *cur = nullptr;
        if (pos == -1)
            cur = find_kth(root->cnt - 1);
        else
        {
            cur = find_kth(pos);
            if (cur->child[0])
                cur = cur->child[0];
            else
            {
                cur->child[0] = node;
                node->p = cur;
                splay(node);
                return;
            }
        }

        while (cur->child[1] != nullptr)
            cur = cur->child[1];
        cur->child[1] = node;
        node->p = cur;
        splay(node);
    }

    void erase(int k)
    {
        Node *x = find_kth(k);
        Node *l = x->child[0];
        Node *r = x->child[1];
        delete x;
        if (!l && !r)
            root = nullptr;
        else if (l && r)
        {
            r->p = nullptr;
            Node *cur = r;
            while (cur->child[0])
                cur = cur->child[0];
            l->p = cur;
            cur->child[0] = l;
            splay(l);
        }
        else if (l)
        {
            l->p = nullptr;
```

```
            splay(l);
        }
        else
        {
            r->p = nullptr;
            splay(r);
        }
    }

    Node *find_kth(int k)
    {
        k++;
        Node *x = root;
        push(x);
        while (1)
        {
            while (x->child[0] && x->child[0]->cnt >= k)
            {
                x = x->child[0];
                push(x);
            }

            if (x->child[0])
                k -= x->child[0]->cnt;

            if (!--k)
                break;
            x = x->child[1];
            push(x);
        }
        splay(x);
        return root;
    }

    void add(int l, int r, int v)
    {
        Node *x = gather(++l, ++r);
        x->sum += x->cnt * v;
        x->lazy += v;
    }

    void add(int ix, int v)
    {
        Node *x = find_kth(++ix);
        x->value += v;
        x->sum += v;
    }

    void set(int ix, int v)
    {
        Node *x = find_kth(++ix);
        v -= x->value;
        x->value += v;
        x->sum += v;
    }
    void shift_right(int l, int r, int k)
    {
        k %= (r - l + 1);
        if (k == 0)
            return;
        reverse(l, r);
        reverse(l, l + k - 1);
        reverse(l + k, r);
    }

    void shift_left(int l, int r, int k)
    {
        k %= (r - l + 1);
        if (k == 0)
            return;
        reverse(l, r);
        reverse(r - k + 1, r);
        reverse(l, r - k);
    }
};
```

2.10 LiChao Tree(C++)

```
constexpr int inf = 2e18;

struct Line
{
    int a, b;
    inline int get(int x)
    {
        return a * x + b;
    }
};

struct Node
{
    int l, r;
    int s, e;
    Line line;

    Node(int _s, int _e)
    {
        l = -1;
        r = -1;
        s = _s;
        e = _e;
        line = {0, -inf};
    }

    inline int get(int x)
    {
        return line.get(x);
    }

    int &operator[](int ix)
    {
        assert(ix == 0 || ix == 1);
        if (ix == 0)
            return l;
        return r;
    }
};

struct LiChao
{
    vector<Node> nodes;
    LiChao()
    {
        nodes.emplace_back(-2e12, 2e12);
    }

    void add(int a, int b)
    {
        add(Line(a, b));
    }

    void add(Line line, int v = 0)
    {
        Node &node = nodes[v];
        int s = node.s, e = node.e;
        int m = s + e >> 1;

        Line &low = node.line, high = line;
        if (low.get(s) > high.get(s))
            swap(low, high);
        if (low.get(e) <= high.get(e))
        {
            node.line = high;
            return;
        }
        int ix = low.get(m) < high.get(m);
        vector<int> left({s, m + 1}), right({m, e});
        vector<Line> lines({low, high});
        node.line = lines[ix];
        if (node[ix] == -1)
        {
            node[ix] = nodes.size();
            nodes.emplace_back(left[ix], right[ix]);
        }
        add(lines[!ix], nodes[v][ix]);
    }

    int get(int x, int v = 0)
    {
        if (v == -1)
            return -inf;
        Node node = nodes[v];
        int l = node.s, r = node.e;
        int m = l + r >> 1;
        return max(node.get(x), get(x, node[x > m]));
    }
};
```

3 Math

3.1 Linear Sieve

```
n = 1000010   # max number
sieve = [0]*n # sieve
primes = []   # prime array
for i in range(2, n):
    if sieve[i] == 0:
        primes.append(i)
    for j in primes:
        if i*j>=n: break
        sieve[i*j] = 1
        if i%j==0: break
```

3.2 FFT

```
import math
pi = math.pi

def FFT(a, inv):
    n = len(a)
    j = 0
    roots = [0] * (n // 2)
    for i in range(1, n):
        bit = n >> 1
        while j >= bit:
            j -= bit
            bit >>= 1
        j += bit
        if i < j:
            a[i], a[j] = a[j], a[i]
    ang = 2 * pi / n * (-1 if inv else 1)
    for i in range(n // 2):
        roots[i] = complex(math.cos(ang * i), math.sin(ang * i))
    i = 2
    while i <= n:
        step = n // i
        for j in range(0, n, i):
            for k in range(i // 2):
                u = a[j + k]
                v = a[j + k + i // 2] * roots[step * k]
                a[j + k] = u + v
                a[j + k + i // 2] = u - v
        i <<= 1
    if inv:
        for i in range(n):
            a[i] /= n

def multiply(arr, brr):
    n = 2
    while n<len(arr) + len(brr):
        n<<=1
    arr = arr+[0]*(n-len(arr))
    brr = brr+[0]*(n-len(brr))
    FFT(arr, 0)
    FFT(brr, 0)
    for i in range(n):
        arr[i] *= brr[i]
    FFT(arr, 1)
    ret = [0]*n
    for i in range(n):
        ret[i] = round(arr[i].real)
    return ret
```

3.3 Berlekamp Massey + Kitamasa

```
mod = 10**9+7


def berlekamp_massey(x):
    if len(x) <= 1:
        return []
    a, b = x[:2]
    x = [i % mod for i in x]
    f, cur, d = 1, [1], [0]
    if a != b:
        cur, d = [b * pow(a, mod - 2, mod)], [1]

    def get(c, ix):
        res = 0
        for i in range(len(c)):
            res = (res + c[i] * x[ix - i]) % mod
        return res

    for i in range(2, len(x)):
        t = get(cur, i - 1)
        if t == x[i]:
            continue
        delta = (x[i] - t) % mod
        d = [1] + [mod - i for i in d]
        mul = delta * pow(get(d, f), mod - 2, mod) % mod
        d = [0] * (i - f - 1) + [(j * mul) % mod for j in d]
        for j in range(len(cur)):
            d[j] = (d[j] + cur[j]) % mod
        cur, d, f = d, cur, i

    return cur


def get_nth(rec, dp, n):
    m = len(rec)
    s, t = [0] * m, [0] * m
    s[0] = 1
    if m != 1:
        t[1] = 1
    else:
        t[0] = rec[0]

    def mul(v, w, rec):
        m = len(v)
        t = [0] * (2 * m)
        for j in range(m):
            for k in range(m):
                t[j + k] += v[j] * w[k] % mod
                if t[j + k] >= mod:
                    t[j + k] -= mod
        for j in range(2 * m - 1, m - 1, -1):
            for k in range(1, m + 1):
                t[j - k] += t[j] * rec[k - 1] % mod
                if t[j - k] >= mod:
                    t[j - k] -= mod
        t = t[:m]
        return t

    while n:
        if n & 1:
            s = mul(s, t, rec)
        t = mul(t, t, rec)
        n >>= 1
    ret = 0
    for i in range(m):
        ret += s[i] * dp[i] % mod
    return ret % mod


def guess_nth_term(x, n):
    if n < len(x):
        return x[n]
    v = berlekamp_massey(x)
    if len(v) == 0:
        return 0
    return get_nth(v, x, n)
```

3.4 Combination

```
def inverseEuler(n, mod):
    return pow(n, mod-2, mod)

def C(n, r, mod):
    f = [1] * (n+1)
    for i in range(2, n+1):
        f[i] = (f[i-1]*i) % mod
    return (f[n]*((inverseEuler(f[r], mod)*inverseEuler(f[n-r], mod)) % mod)) % mod
```

3.5 Lucas Theorem

```
# (nCr)%mod (mod is prime)
arr,brr = [],[]
while n:
    arr.append(n%mod)
    n//=mod

while r:
    brr.append(r%mod)
    r//=mod

if len(arr) < len(brr):
    arr, brr = brr, arr

brr+=[0]*(len(arr) - len(brr))

def fact(n): # or preprocess
    r = 1
    for i in range(1, n + 1):
        r*=i
    return r

def C(n,r):
    if n<r:
        return 0
    return fact(n) // (fact(r) * fact(n-r))

l = len(arr)
ans = 1
for i in range(l):
    ans *= C(arr[i], brr[i]) % mod
```

3.6 Extended Euclidean Algorithm

```
def EED(a, b):
    if a < b:
        a, b = b, a
    if b == 0:
        return a, 1, 0
    g, x1, y1 = EED(b, a % b)
    return g, y1, x1 - a // b * y1
```

3.7 Euler totient Function

```
N = 1000010
s = [1] * N # Eratosthenes Sieve

# ... linear sieve

def phi(arr): # arr : factorization order of n
    r = 1
    for i in range(len(arr)):
        if arr[i]:
            r *= p[i] ** arr[i] - p[i] ** (arr[i] - 1) # p^k - p^(k-1)
    return r
```

3.8 All Euler totient value

```
n = 1001
sieve = [i for i in range(n+1)]
for i in range(2, n+1):
    if sieve[i] == i:
        for j in range(i, n+1, i):
            sieve[j] -= sieve[j] // i
```

3.9 Partition Number

```
mod = 998244353
p = [1]
g = []
k = 1
kc = 0
for n in range(1, T+2): # O(n sqrt(n))
    p.append(0)
    q = p[-1]
    if kc:
        if k * (3 * k + 1) == 2 * n:
            g.append(k * (3 * k + 1) // 2)
            kc = 0
            k += 1
    else:
        if k * (3 * k - 1) == 2 * n:
            g.append(k * (3 * k - 1) // 2)
            kc = 1
    for i in range(len(g)):
        if i & 3 < 2:
            q = (q + p[n - g[i]]) % mod
        else:
            q = (q + mod - p[n - g[i]]) % mod
    p[-1] = q
```

3.10 Mobius inversion

```
n = 10**7+100
prime = [1]*n
mu = [1]*n
for i in range(2, n):
    if not prime[i]:
        continue
    mu[i] = -1
    for j in range(i*2, n, i):
        prime[j] = 0
        mu[j] = -mu[j]
        if (j//i) % i == 0:
            mu[j] = 0
```

3.11 Pollard Rho + Miller Rabin Test(C++)

```
ll mul(ll x, ll y, ll mod) {
  return (__int128)x * y % mod;
}

ll ipow(ll x, ll y, ll p) {
  ll ret = 1, piv = x % p;
  while (y) {
    if (y & 1) ret = mul(ret, piv, p);
    piv = mul(piv, piv, p);
    y >>= 1;
  }
  return ret;
}

bool miller_rabin(ll x, ll a) {
  if (x % a == 0) return 0;
  ll d = x - 1;
  while (1) {
    ll tmp = ipow(a, d, x);
    if (d & 1) return(tmp != 1 && tmp != x - 1);
    else if (tmp == x - 1) return 0;
    d >>= 1;
  }
}

bool isprime(ll x) {
  for (auto& i : { 2,3,5,7,11,13,17,19,23,29,31,37 }) {
    if (x == i) return 1;
    if (x > 40 && miller_rabin(x, i)) return 0;
  }
  if (x <= 40) return 0;
  return 1;
}

ll f(ll x, ll n, ll c) {
  return(c + mul(x, x, n)) % n;
}

ll myAbs(ll a) {
  return a > 0 ? a : (-a);
}

ll gcd(ll a, ll b) {
  if (b == 0)
    return a;
  return gcd(b, a % b);
}

void rec(ll n, vector<ll>& v) {
  if (n == 1) return;
  if (n % 2 == 0) {
    v.push_back(2);
    rec(n / 2, v);
    return;
  }
  if (isprime(n)) {
    v.push_back(n);
    return;
  }
  ll a, b, c;
  while (1) {
    a = rand() % (n - 2) + 2;
    b = a;
    c = rand() % 20 + 1;
    do {
      a = f(a, n, c);
      b = f(f(b, n, c), n, c);
    } while (gcd(myAbs(a - b), n) == 1);
    if (a != b)
      break;
  }
  ll x = gcd(myAbs(a - b), n);
  rec(x, v);
  rec(n / x, v);
}

auto factorize(ll n) {
  vector<ll> ret;
  rec(n, ret);
  sort(ret.begin(), ret.end());
  return ret;
}
```

3.12 Catalan Number(C++)

```
ll N, d[MAX] = { 1,1,2,5 };
int main() {
  cin.tie(0)->sync_with_stdio(0);

  int t;
  cin>>t;
  for (int i = 4; i < MAX; i++) {
    for (int j = 0; j < i; j++) {
      d[i] += d[j] * d[i - j - 1];
      d[i] %= MOD;
    }
  }
  while(t--){
    cin >> N;
    if(N%2){
      cout<<0<<"\n";
      continue;
    }
    N/=2;
    cout << d[N] << "\n";
  }
}
```

4 Geometry

4.1 CCW

```
def ccw(a, b, c):
    return a[0]*b[1] + b[0]*c[1] + c[0]*a[1] - \
           (b[0]*a[1] + c[0]*b[1] + a[0]*c[1])
```

4.2 Line Cross

```
def cross(a, b, c, d):
    return ccw(a, b, c) * ccw(a, b, d) < 0 and ccw(c, d, a) * ccw(c, d, b) < 0
```

4.3 Convex Hull

```
def ConvexHull(points):
    upper = []
    lower = []
    for p in sorted(points):
        while len(upper) > 1 and ccw(upper[-2], upper[-1], p) >= 0:
            upper.pop()
        while len(lower) > 1 and ccw(lower[-2], lower[-1], p) <= 0:
            lower.pop()
        upper.append(p)
        lower.append(p)
    return upper, lower
```

4.4 Rotating Calipers

```
def sub(a,b):
    return[a[0]-b[0], a[1]-b[1]]

def norm(p):
    return (p[0]**2+p[1]**2)**0.5

def dot(p1, p2):
    return p1[0] * p2[0] + p1[1] * p2[1]

def diameter(p):
    n = len(p)
    left, right = 0, 0
    for i in range(1, n):
        if p[i] < p[left]:
            left = i
            p[left] = p[i]
        if p[i] > p[right]:
            right = i
            p[right] = p[i]
    calipersA = [0,1]
    ret = norm(sub(p[right], p[left]))
    toNext = [None] * n
    for i in range(n):
        toNext[i] = sub(p[(i+1)%n],p[i])
        tmp = norm(toNext[i])+eps
        toNext[i] = [toNext[i][0]/tmp, toNext[i][1]/tmp]
    a = left
    b = right
    while a != right or b != left:
        cosThetaA = dot(calipersA, toNext[a])
        cosThetaB = -dot(calipersA, toNext[b])
        if cosThetaA > cosThetaB:
            calipersA = toNext[a]
            a = (a + 1) % n
        else:
            calipersA = [-toNext[b][0], -toNext[b][1]]
            b = (b + 1) % n
        ret = max(ret, norm(sub(p[b], p[a])))
    return ret
```

5 String

5.1 KMP

```
def make_fail(s):
    pi = [0] * len(s)
    j = 0
    for i in range(1, len(s)):
        while s[i] != s[j] and j > 0:
            j = pi[j-1]
        if s[i] == s[j]:
            j += 1
            pi[i] = j
    return pi

def KMP(string, pattern):
    pi = make_fail(pattern)
    indices = []
    j = 0
    for i in range(len(string)):
        while string[i] != pattern[j] and j > 0:
            j = pi[j-1]
        if string[i] == pattern[j]:
            if j == len(pattern) - 1: # found
                indices.append(i - len(pattern) + 2)
                j = pi[j]
            else:
                j += 1
    return indices
```

5.2 Manacher

```
# s = list(input())
s = '#'.join(s)
s = '#' + s + '#'
def manacher(s):
    n = len(s)
    A = [0] * n
    r = 0
    p = 0
    for i in range(n):
        if i <= r:
            A[i] = min(A[2 * p - i], r - i)
        else:
            A[i] = 0
        while i - A[i] - 1 >= 0 and i + A[i] + 1 < n and s[i - A[i] - 1] == s[i + A[i] + 1]:
            A[i] += 1
        if r < i + A[i]:
            r = i + A[i]
            p = i
    return A
```

5.3 Aho Corasick(C++)

```
struct Trie {
  Trie* next[26];
  Trie* fail;
  bool output;
  Trie() : output(false) {
    fill(next, next + 26, nullptr);
  }
  ~Trie() {
    for (int i = 0; i < 26; i++) {
      if (next[i])
        delete next[i];
    }
  }
  void insert(string& s, int idx) {
    if (idx >= s.length()) {
      output = true;
      return;
    }
    int x = s[idx] - 'a';
    if (!next[x]) {
      next[x] = new Trie();
    }
    next[x]->insert(s, idx + 1);
  }
};
void fail(Trie* root) {
  queue<Trie*> q;
  root->fail = root;
  q.push(root);
  while (!q.empty()) {
    Trie* cur = q.front();
    q.pop();
    for (int i = 0; i < 26; i++) {
      Trie* nxt = cur->next[i];
      if (!nxt)
        continue;
      if (root == cur)
        nxt->fail = root;
      else {
        Trie* tmp = cur->fail;
        while (tmp != root && !tmp->next[i])
          tmp = tmp->fail;
        if (tmp->next[i])
          tmp = tmp->next[i];
        nxt->fail = tmp;
      }
      if (nxt->fail->output)
        nxt->output = true;
      q.push(nxt);
    }
  }
}
string solve(string s, Trie* root) {
  vector<pair<int, int>> ret;
  Trie* cur = root;
  for (int i = 0; i < s.length(); i++) {
    int nxt = s[i] - 'a';
    while (cur != root && !cur->next[nxt])
      cur = cur->fail;
    if (cur->next[nxt])
      cur = cur->next[nxt];
    if (cur->output) {
      return "YES";
    }
  }
  return "NO";
}
```

5.4 Suffix Array(C++)

```
struct Comparator {
  const vector<int>& group;
  int t;
  Comparator(const vector<int>& _group, int _t) :group(_group), t(_t) {
  }

  bool operator() (int a, int b) {
    if (group[a] != group[b]) return group[a] < group[b];
    return group[a + t] < group[b + t];
  }

};

vector<int> getSuffixArray(const string& s) {
  int t = 1;
  int n = s.size();
  vector<int> group(n + 1);
  for (int i = 0; i < n; i++)
    group[i] = s[i];
  group[n] = -1;
  vector<int> perm(n);
  for (int i = 0; i < n; i++)perm[i] = i;
  while (t < n) {
    Comparator compareUsing2T(group, t);
    sort(perm.begin(), perm.end(), compareUsing2T);
    t <<= 1;
    if (t >= n) break;
    vector<int> newGroup(n + 1);
    newGroup[n] = -1;
    newGroup[perm[0]] = 0;
    for (int i = 1; i < n; i++) {
      if (compareUsing2T(perm[i - 1], perm[i]))
        newGroup[perm[i]] = newGroup[perm[i - 1]] + 1;
      else

        newGroup[perm[i]] = newGroup[perm[i - 1]];
    }
    group = newGroup;
  }
  return perm;
}
```

5.5 Suffix Automaton(C++)

```
struct State
{
    signed len, link;
    int cnt = 0LL, d = 0LL;
    map<char, signed> next;
    vector<int> inv_link;
};

struct SuffixAutomaton
{
    vector<State> st;
    set<int> terminals;
    SegmentTree<int> segtree;
    int sz, last, l;
    SuffixAutomaton()
    {
        l = 400001;
        init();
    }
    SuffixAutomaton(string s)
    {
        l = s.size() * 2 + 1;
        init();
        build(s);
        postprocessing();
    }
    void init()
    {
        st.resize(l);
        sz = 0;
        last = 0;
        st[0].len = 0;
        st[0].link = -1;
        sz++;
        vector<int> brr(l);
        segtree = SegmentTree<int>(
            brr, [](int a, int b)
            { return a + b; },
            0LL);
    }

    void postprocessing()
    {
        for (int i = 0; i < sz; i++)
            for (auto [x, y] : st[i].next)
                st[y].cnt += st[i].cnt + (i == 0);
        for (int i = 1; i < sz; i++)
            st[st[i].link].inv_link.push_back(i);
        get_d(0);
        st[0].d--;
    }

    int get_d(int ix)
    {
        if (st[ix].d > 0)
            return st[ix].d;
        int r = 1;
        for (auto [x, y] : st[ix].next)
            r += get_d(y);
        return st[ix].d = r;
    }

    void build(string s)
    {
        for (auto i : s)
            sa_extend(i);
        int p = last;
        while (p > 0)
        {
            terminals.insert(p);
            p = st[p].link;
        }
    }

    void update(int ix)
    {
        segtree.update(ix, st[ix].len - st[st[ix].link].len);
    }

    void sa_extend(char c)
    {
        int cur = sz++;
        st[cur].len = st[last].len + 1;
        int p = last;
        while (p != -1 && !st[p].next.count(c))
    {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1)
    {
        st[cur].link = 0;
    }
    else
    {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len)
        {
            st[cur].link = q;
        }
        else
        {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            update(clone);
            while (p != -1 && st[p].next[c] == q)
            {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
            update(q);
        }
    }
    update(cur);
    last = cur;
}

int get_diff_strings()
{
    return segtree.query(0, sz - 1);
}

int get_tot_len_diff_substrings()
{
    int tot = 0;
    for (int i = 1; i < sz; i++)
    {
        int shortest = st[st[i].link].len + 1;
        int longest = st[i].len;

        int num_strings = longest - shortest + 1;
        int cur = num_strings * (longest + shortest) / 2;
        tot += cur;
    }
    return tot;
}

string get_lexicographically_kth_string(int k)
{
    // TODO
    return "";
}

int go(string w)
{
    int cur = 0;
    for (auto i : w)
    {
        cur = st[cur].next[i];
        if (cur == 0)
            return 0;
    }
    return cur;
}

bool is_substring(string w)
{
    return go(w) > 0;
}

bool is_suffix(string w)
{
    return terminals.contains(go(w));
}

int count(string w)
{
    // TODO
    return 1;
}

string lcs(string w)
{
    int v = 0, l = 0, best = 0, bestpos = 0;
    for (int i = 0; i < w.size(); i++)
    {
        while (v && !st[v].next.count(w[i]))
        {
            v = st[v].link;
            l = st[v].len;
        }
        if (st[v].next.count(w[i]))
        {
            v = st[v].next[w[i]];
            l++;
        }
        if (l > best)
        {
            best = l;
            bestpos = i;
        }
    }
    return w.substr(bestpos - best + 1, best);
}
```

```
};
```

6 Sequence

6.1 Fibonacci Sequence

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, \ldots$
$a_1 = a_2 = 1$
$a_n = a_{n-1} + a_{n-2} (n \geq 3)$

6.2 Catalan numbers

$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, \ldots$
$C(n) = \frac{(2n)!}{(n!(n+1)!)}$

6.3 Partition Number

$1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42, 56, 77, 101, 135, 176, 231, 297, 385, 490, 627, 792, 1002, 1255, 1575, 1958, \ldots$

6.4 Derangement

$1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, 14684570, 176214841, 2290792932, 32071101049, \ldots$
$der(0) = 1, der(1) = 0$
$der(n) = (n-1)(der(n-1) + der(n-2)))$

7 Formulas or Theorems

7.1 Cayley Formula

n개의 완전 그래프는 n^{n-2}개의 스패닝 트리를 갖는다.

7.2 Erdos-Gallai Theorem

정수 수열 $d_1 \geq d_2 \geq \cdots \geq d_n$이 정점이 n개인 단순 그래프의 차수 수열이 될 필요충분조건은 $\sum_{i=1}^{n} d_i$가 짝수이고 $\sum_{i=1}^{n} d_i \leq k(k-1) + \sum_{i=k+1}^{n} min(d_i, k)$ 가 $1 \leq k \leq n$에서 성립하는 것이다.

7.3 Planar Graph Lemma

평면 그래프에서 V-E+F=2가 성립한다.
여기서 F(face)는 어떤 사이클 안에 간선이 없는 사이클이다.
평면 그래프는 간선이 교차하지 않는 그래프

7.4 Moser's Circle

$g(n)$: 원주상에서 n개의 점을 현으로 연결하는데 세 현이 원 안의 한 점에서 만나지 않도록 할 때 원이 나눠지는 조각의 수
$g(n) =_n \mathrm{C}_4 +_n \mathrm{C}_2 + 1$

7.5 Pick's Theorem

다각형 내부의 격자점의 개수를 I, 면적을 A, 다각형 경계 위 격자 점의 개수를 B라고 하면 $A = I + \frac{B}{2} - 1$이다.

7.6 Complete Bipartite Graph Lemma

$K_{n,m}$의 스패닝 트리의 개수는 $m^{n-1}n^{m-1}$이다.

7.7 Small to Large Trick

두 집합을 합칠 때 작은 집합을 큰 집합에 합치는게 시간이 적게 든다.

8 Miscellaneous

8.1 O(nlogn) LIS

```
import bisect

def lis(n, arr):
    brr = [-9876543210]
    for i in range(n):
        if arr[i] > brr[-1]:
            brr.append(arr[i])
            continue
        t = bisect.bisect_left(brr, arr[i])
        brr[t] = arr[i]
    return brr
```

8.2 Hanoi Tower

```
def hanoi(n): # n : #(disk)
    rHanoi(n, 1, 2, 3)

def rHanoi(n, f, a, t):
    if n == 1:
        print(f, t)
        return
    rHanoi(n - 1, f, t, a)
    print(f, t)
    rHanoi(n - 1, a, f, t)
```

8.3 Hackenbush Score

```
# W : 1
# B : -1
score = 0
f = 1
flag = 1
for i in range(len(s)): # s : (W*B*)*
    if i and s[i] != s[i - 1]:
        flag = 2
    f /= flag
    if s[i] == 'W':
        score += f
    else:
        score -= f
```

8.4 LCS

```
def LCS(a, b): # O(n^2)
    arr = [[0] * (len(a) + 1) for _ in range((len(b) + 1))]

    la = len(a)
    lb = len(b)

    for i in range(1, lb + 1):
        for j in range(1, la + 1):
            if a[j - 1] == b[i - 1]:
                arr[i][j] = arr[i - 1][j - 1] + 1
            else:
                arr[i][j] = max(arr[i - 1][j], arr[i][j - 1])
    l = arr[-1][-1]
    a,b=b,a
    i = len(a)
    j = len(b)
    s = []
    while i and j:
        if a[i - 1] == b[j - 1]:
            s.append(b[j - 1])
            i -= 1
            j -= 1
        else:
            if arr[i - 1][j] > arr[i][j - 1]:
                i -= 1
            else:
                j -= 1
    return l, ''.join(s[::-1]) # length, one of LCS string
```

8.5 2D Prefix-sum

```
def get_ps(arr):
    n = len(arr)
    m = len(arr[0])
    s = [[0] * (m + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            s[i][j] = arr[i - 1][j - 1] + s[i - 1][j] + \
                s[i][j - 1] - s[i - 1][j - 1]

    def get(a, b, c, d):
        nonlocal arr, s
        c += 1
        d += 1
        return s[c][d] - s[a][d] - s[c][b] + s[a][b]
    return get
```

8.6 1D-Knapsack

```
dp = [0]*(k+1)
for cost, value in brr:
    for i in range(k, cost-1, -1):
        dp[i] = max(dp[i], dp[i-cost] + value)
```

8.7 Ternary Search(C++)

```
ll s = 0, e = T;
while (s + 3 <= e) {
  ll p = (s * 2 + e) / 3, q = (s + e * 2) / 3;
  if (solve(p) > solve(q))
  s = p;
  else
  e = q;
}
ll ans = INF, idx = 0;
for (int i = s; i <= e; i++) {
  ll dis = solve(i);
  if (ans > dis) {
    idx = i;
    ans = dis;
  }
}
```

8.8 O(nlogn) LIS(C++)

```
for (int i = 0; i < N; i++) {
  int cur = lower_bound(ans.begin(), ans.end(), v[i]) - ans.begin();
  if (cur < ans.size())
  ans[cur] = v[i];
  else
  ans.push_back(v[i]);
}
cout << ans.size() << "\n";
```

8.9 FastIO Python

```
import os, io, ¨pypy¨ # underscore

class FastIO:
  def __init__(self):
    self.r = io.BytesIO(os.read(0, os.fstat(0).st_size)).read()
    self.w = __pypy__.builders.StringBuilder()
    self.i = 0
  def Flush(self): os.write(1, self.w.build().encode())
  def ReadInt(self):
    ret = 0
    while self.r[self.i] & 16: ret = 10 * ret + (self.r[self.i] & 15); self.i += 1
    self.i += 1
    return ret
  def Write(self, x): self.w.append(x)

IO = FastIO()
n = IO.ReadInt()
IO.Write('\n'.join(map(str, [IO.ReadInt() + IO.ReadInt() for _ in range(n)])));
IO.Flush()
```

8.10 Fast C++ Template

```
// compile : g++ a.cpp -std=c++17 && ./a.out
#include<bits/stdc++.h>
#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")
#define sz(v) (int)v.size()
#define int long long
#define all(v) (v).begin(), (v).end()
#define press(v) (v).erase(unique(all(v)), (v).end())
#define endl '\n'
using namespace std;
typedef pair<int, int> pi;
typedef pair<int,pi> pii;
const int MAX = 1e5+7;
const int INF = 0x3f3f3f3f3f3f3f3f;
const int MOD = 1e9 + 7;
int N,a[MAX];
int32_t main(){
  cin.tie(0)->sync_with_studio(0);
}
```