Daffodil International University

Anup Barman

Personal Reference Document

khun_   bee_   anup_barman

Contents

1 1.Miscellaneous

1.1 Lazy Propagation

```cpp
class stree {
    vector<int> seg, lazy;
public:
    segtree(int n) {
        seg.resize(4 * n + 5);
        lazy.resize(4 * n + 5);
    }
    void propagate(int i, int low, int high) {
        if (lazy[i] != 0) {
            seg[i] += (high - low + 1) * lazy[i];
            if (low != high) {
                lazy[2 * i + 1] += lazy[i];
                lazy[2 * i + 2] += lazy[i];
            }
            lazy[i] = 0;
        }
    }
    void build(int i, int low, int high, int arr[]) {
        if (low == high) {
            seg[i] = arr[low];
            return;
        }
        int mid = (low + high) >> 1;
        build(2 * i + 1, low, mid, arr);
        build(2 * i + 2, mid + 1, high, arr);
        seg[i] = seg[2 * i + 1] + seg[2 * i + 2];
    }
    void update(int i, int low, int high, int l, int r,
      int val) {
        propagate(i, low, high);
        if (high < l or r < low) return;
        if (low >= l and high <= r) {
            seg[i] += (high - low + 1) * val;
            if (low != high) {
                // has children
                lazy[2 * i + 1] += val;
                lazy[2 * i + 2] += val;
            }
            return;
        }
        int mid = (low + high) >> 1;
        update(2 * i + 1, low, mid, l, r, val);
        update(2 * i + 2, mid + 1, high, l, r, val);
        seg[i] = seg[2 * i + 1] + seg[2 * i + 2];
    }
    int query(int i, int low, int high, int l, int r) {
        propagate(i, low, high);
        if (high < l or r < low) return 0;
        if (low >= l and high <= r) return seg[i];
        int mid = (low + high) >> 1;
        int left = query(2 * i + 1, low, mid, l, r);
        int right = query(2 * i + 2, mid + 1, high, l, r);
        return left + right;
    }
};
```

1.2 PBDS

```cpp
#include "ext/pb_ds/assoc_container.hpp"
#include "ext/pb_ds/tree_policy.hpp"
using namespace __gnu_pbds;
template <class T>
using oset = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
Note: Use less_equal for multiset like behaviour
Usage:
st.find_by_order(k) :: returns iterator of the k-th
  smallest element
st.order_of_key(x) :: returns index of x (number of
  elements less than x)
```

1.3 Segment Tree :: Binary search on Unsorted array

```cpp
// 1 based indexing
struct SegTree {
    int n;
    vector<int> seg;
    SegTree(int _n) {
        n = _n;
        seg.assign(4 * n + 5, 0);
    }
    void build(int node, int l, int r) {
        if (l == r) {
            seg[node] = 1;
            return;
        } // each position initially present
        int mid = (l + r) >> 1;
        build(node * 2, l, mid);
        build(node * 2 + 1, mid + 1, r);
        seg[node] = seg[node * 2] + seg[node * 2 + 1];
    }
    // set position pos to value val (0 or 1)
    void update(int node, int l, int r, int pos, int
      val) {
        if (l == r) {
            seg[node] = val;
            return;
        }
        int mid = (l + r) >> 1;
        if (pos <= mid)
            update(node * 2, l, mid, pos, val);
        else
            update(node * 2 + 1, mid + 1, r, pos, val);
        seg[node] = seg[node * 2] + seg[node * 2 + 1];
    }
    // find index of k-th "present" element (1-based k)
    int kth(int node, int l, int r, int k) {
        if (l == r) return l;
        int leftCnt = seg[node * 2];
        int mid = (l + r) >> 1;
        if (k <= leftCnt)
            return kth(node * 2, l, mid, k);
        else
            return kth(node * 2 + 1, mid + 1, r, k -
              leftCnt);
    }
};

void solve() {
    int n;
    cin >> n;
    vector<ll> a(n + 1), p(n + 1);
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }
    for (int i = 1; i <= n; ++i) {
        cin >> p[i];
    }
    SegTree st(n);
    st.build(1, 1, n);
    // For each removal request p[i], find the p[i]-th
    //   present element,
    // print it and mark that position as removed (set to
    //   0).
    for (int i = 1; i <= n; ++i) {
        int k = p[i];
        int idx = st.kth(1, 1, n, k);  // index in original
          array
```

```cpp
    cout << a[idx] << (i == n ? '\n' : ' ');
    st.update(1, 1, n, idx, 0);
  }
}
```

1.4 Segment Tree

```cpp
class stree {
  vector<int> seg;
public:
  segtree(int n) {
    seg.assign(4 * n + 5, 0);
  }
  void build(int ind, int low, int high, int arr[]) {
    if (low == high) {
      seg[ind] = arr[low];
      return;
    }
    int mid = (low + high) >> 1;
    build(2 * ind + 1, low, mid, arr);
    build(2 * ind + 2, mid + 1, high, arr);
    seg[ind] = min(seg[2 * ind + 1], seg[2 * ind + 2]);
  }
  int query(int ind, int low, int high, int l, int r) {
    if (r < low or high < l) return INT_MAX;
    if (low >= l and high <= r) return seg[ind];
    int mid = (low + high) / 2;
    int left = query(2 * ind + 1, low, mid, l, r);
    int right = query(2 * ind + 2, mid + 1, high, l,
    ↪   r);
    return min(left, right);
  }
  void update(int ind, int low, int high, int i, int
  ↪   val) {
    if (low == high) {
      seg[ind] = val;
      return;
    }
    int mid = (low + high) / 2;
    if (i <= mid) update(2 * ind + 1, low, mid, i,
    ↪   val);
    else update(2 * ind + 2, mid + 1, high, i, val);
    seg[ind] = min(seg[2 * ind + 1], seg[2 * ind + 2]);
  }
};
```

1.5 Sparse Table

```cpp
const int MX = 2e5 + 10;
int n, arr[MX], st[25][MX];
int log2Floor(int i) {
  return 31 - __builtin_clz(i);
}
void build() {
  int k = log2Floor(n);
  copy(arr, arr + n, st[0]);
  for (int i = 1; i <= k; ++i) {
    for (int j = 0; j + (1 << i) <= n; j++) {
      st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 <<
      ↪   (i - 1))]);
    }
  }
}
int query(int l, int r) {
  int i = log2Floor(r - l + 1);
  return min(st[i][l], st[i][r - (1 << i) + 1]);
}
```

1.6 Unique Prime Factorization using Sieve

```cpp
const int MX = 2e5 + 10;
vector<int> pfac[MX];
void factorize() {
```

```cpp
  for (int i = 2; i < MX; i++){
    if (!pfac[i].empty()) continue;
    for (int j = i; j < MX; j += i)
      pfac[j].push_back(i);
  }
}
```

1.7 nCr

```cpp
const int MX = 1e6 + 10;
const int M = 1e9 + 7;
int fact[MX], inv_fact[MX];
int modPow(int a, int b) {
  int ans = 1;
  while (b) {
    if (b & 1) ans = (1LL * ans * a) % M;
    a = (1LL * a * a) % M;
    b >>= 1;
  }
  return ans;
}
void precalFact() {
  fact[0] = inv_fact[0] = 1;
  for (int i = 1; i < MX; i++) {
    fact[i] = (1LL * fact[i - 1] * i) % M;
  }
  inv_fact[MX - 1] = modPow(fact[MX - 1], M - 2);
  for (int i = MX - 2; i >= 1; i--) {
    inv_fact[i] = (1LL * inv_fact[i + 1] * (i + 1)) %
    ↪   M;
  }
}
int nCr(int n, int r) {
  if (r < 0 or r > n) return 0;
  return 1LL * fact[n] * inv_fact[r] % M * inv_fact[n
  ↪   - r] % M;
}
```

2 2. Graph

2.1 Articulation Point

```cpp
int n;                       // number of nodes
vector<vector<int>> lst;     // adjacency list of graph
vector<bool> vis;
vector<int> tin, low;
int timer;
void dfs(int u, int p = -1) {
  vis[u] = true;
  tin[u] = low[u] = timer++;
  int children = 0;
  for (int v : lst[u]) {
    if (v == p) continue;
    if (vis[v]) {
      low[u] = min(low[u], tin[v]);
    } else {
      dfs(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] >= tin[u] && p != -1)
        IS_CUTPOINT(u);
      ++children;
    }
  }
  // if no vertex below v can reach u or higher
  ↪   removing u disconnects that subtree
  if (p == -1 && children > 1)
    IS_CUTPOINT(u);
}
void find_cutpoints() {
  timer = 0;
  vis.assign(n, false);
  tin.assign(n, -1);
  low.assign(n, -1);
  for (int i = 0; i < n; ++i) {
    if (!vis[i])
```

```cpp
      dfs(i);
  }
}
```

2.2 Bridge Finding Algorithm

```cpp
const int MX = 1e5 + 10;
int n, m, timer = 0;
vector<int> adj[MX];
vector<int> tin(MX, -1), low(MX, -1);
vector<bool> vis(MX, false);
void is_bridge(int u, int v) {
  // do something with the edge
}
void dfs(int u, int p = -1) {
  vis[u] = true;
  tin[u] = low[u] = timer++;
  for (int v : adj[u]) {
    if (v == p) continue;
    if (vis[v]) {
      low[u] = min(low[u], tin[v]);
    } else {
      dfs(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] > tin[u]) {
        is_bridge(u, v);
      }
    }
  }
}
```

2.3 Cycle Detection in DAG

```cpp
const int MX = 1e5 + 10;
bool vis[MX], pathVis[MX];
vector<int> lst[MX];
bool dfs(int u) {
  vis[u] = true;
  pathVis[u] = true;
  for (auto v : lst[u]) {
    if (!vis[v]) {
      if (dfs(v))
        return true;
    } else if (pathVis[v]) {
      return true;
    }
  }
  pathVis[u] = false;
  return false;
}
void solve() {
  // take graph input
  for (int i = 0; i < n; ++i) {
    if (!vis[i])
      dfs(i);
  }
}
```

2.4 DSU on Trees

```cpp
int n, color[MX], ans[MX];
vector<int> g[MX];
set<int> bucket[MX];
int merge(int a, int b) {
  if (bucket[a].size() < bucket[b].size()) swap(a, b);
  bucket[a].insert(bucket[b].begin(), bucket[b].end());
  bucket[b].clear();
  return a;
}
int dfs(int u, int p = -1) {
  int cur = u;
  for (int v : g[u])
    if (v != p)
      cur = merge(cur, dfs(v, u));
```

```cpp
    ans[u] = (int)bucket[cur].size();
    return cur;
}
void solve() {
    cin >> n;
    for (int i = 0; i < n; ++i) {
        cin >> color[i];
        bucket[i].insert(color[i]);
    }
    // graph input
    dfs(0);
    // print output
}
```

2.5 DSU

```cpp
const int MX = 1e5 + 10;
int par[MX], sz[MX];
void init() {
    for (int i = 1; i < MX; i++) {
        par[i] = i;
        sz[i] = 1;
    }
}
int findpar(int x) {
    if (par[x] == x) return x;
    return par[x] = findpar(par[x]);
}
void unite(int u, int v) {
    u = findpar(u);
    v = findpar(v);
    if (u != v) {
        if (sz[u] < sz[v]) {
            swap(u, v);
        }
        sz[u] += sz[v];
        par[v] = u;
    }
}
```

2.6 Euler Tour

```cpp
const int MX = 2e5 + 10;
int timer = -1;
// s = start pos, e = end pos
int val[MX], s[MX], e[MX], flat[MX];
vector<int> lst[MX];
void dfs(int u, int p) {
    s[u] = ++timer;
    flat[timer] = val[u];
    for (auto v : lst[u]) {
        if (v != p)
            dfs(v, u);
    }
    e[u] = timer;
}
```

2.7 Floyd Warshall

```cpp
vector<vector<int>> d(n, vector<int> (n, INF));
// take graph input into d
for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (d[i][k] < INF && d[k][j] < INF)
                d[i][j] = min(d[i][j], d[i][k] +
                ↪ d[k][j]);
        }
    }
}
```

2.8 LCA using Binary Lifting

```cpp
int n, l;
vector<vector<int>> adj;
```

```cpp
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);
}
```

2.9 MST

```cpp
// DSU first
void solve() {
    int n, m;
    cin >> n >> m;
    vector<tuple<int, int, int>> edges;
    for (int i = 0; i < m; ++i) {
        int u, v, wt;
        cin >> u >> v >> wt;
        edges.push_back({wt, u, v});
    }
    sort(edges.begin(), edges.end());
    init(n);
    int cost = 0;
    for (tuple& [ wt, u, v ] : edges) {
        if (findpar(u) == findpar(v)) continue;
        unite(u, v);
        cost += wt;
    }
    cout << cost << endl;
}
```

2.10 Max Bipartite Matching [Hopcroft-Karp]

```cpp
const int INF = 1e9;
void hopcroftCarp() {
    int n, m, e;
    cin >> n >> m >> e;
    vector<int> adj[n];
    for (int i = 0; i < e; ++i) {
        int u, v;
        cin >> u >> v;
        --u;
        --v;
        adj[u].push_back(v);
```

```cpp
}
vector<int> ml(m, -1), mr(n, -1), dist(n);
auto bfs = [&]() -> bool {
    queue<int> q;
    for (int u = 0; u < n; ++u) {
        if (mr[u] == -1) {
            dist[u] = 0;
            q.push(u);
        } else {
            dist[u] = INF;
        }
    }
    bool foundAugmenting = false;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v : adj[u]) {
            int pairedLeft = ml[v];
            if (pairedLeft == -1) {
                foundAugmenting = true;
            } else if (dist[pairedLeft] == INF) {
                dist[pairedLeft] = dist[u] + 1;
                q.push(pairedLeft);
            }
        }
    }
    return foundAugmenting;
};
function<bool(int)> dfs = [&](int u) -> bool {
    for (int v : adj[u]) {
        int pairedLeft = ml[v];
        if (pairedLeft == -1 or (dist[pairedLeft] ==
        ↪ dist[u] + 1 and dfs(pairedLeft))) {
            mr[u] = v;
            ml[v] = u;
            return true;
        }
    }
    dist[u] = INF;
    return false;
};
int matching = 0;
while (bfs()) {
    for (int u = 0; u < n; ++u) {
        if (mr[u] == -1) {
            if (dfs(u)) matching++;
        }
    }
}
cout << matching << el;
for (int u = 0; u < n; ++u) {
    if (mr[u] != -1) {
        cout << u << " " << mr[u] << el;
    }
}
}
```

2.11 Max Bipartite Matching [Kuhn's]

```cpp
// left set size, right set size, edge count
int n, k, m, visToken = 1;
vector<int> lst[MX];
int mr[MX], ml[MX], vis[MX];
bool try_kuhn(int u) {
    if (vis[u] == visToken)
        return false;
    vis[u] = visToken;
    for (auto v : lst[u]) {
        if (ml[v] == -1 or try_kuhn(ml[v])) {
            ml[v] = u;
            mr[u] = v;
            return true;
```

```cpp
      }
    }
    return false;
  }
}
void solve() {
  cin >> n >> k >> m;
  for (int i = 0; i < m; ++i) {
    int u, v;
    cin >> u >> v;
    --u, --v;
    lst[u].push_back(v);
  }
  fill(mr, mr + n, -1);
  fill(ml, ml + k, -1);
  int ans = 0;
  for (int u = 0; u < n; ++u) {
    for (auto v : lst[u]) {
      if (ml[v] == -1) {
        ml[v] = u;
        mr[u] = v;
        ans++;
        break;
      }
    }
  }
  for (int u = 0; u < n; ++u) {
    if (mr[u] != -1) continue;
    visToken++;
    if (try_kuhn(u))
      ans++;
  }
  cout << ans << el;
  for (int v = 0; v < k; ++v) {
    if (ml[v] != -1) {
      cout << ml[v] + 1 << " " << v + 1 << el;
    }
  }
}
```

2.12 Topological Sorting

```cpp
const int N = 1e5 + 10;
vector<int> g[N], indegree(N, 0);
vector<int> topSort(int n) {
  queue<int> q;
  vector<int> order;
  for (int i = 1; i <= n; i++) {
    if (indegree[i] == 0) {
      q.push(i);
    }
  }
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    order.push_back(u);
    for (int v : g[u]) {
      indegree[v]--;
      if (indegree[v] == 0) {
        q.push(v);
      }
    }
  }
  return order;
}
```

2.13 Weighted Union Find

```cpp
const int MX = 2e5 + 10;
int par[MX], sz[MX];
ll d[MX];
void init() {
  for (int i = 0; i < MX; ++i) {
    par[i] = i;
    sz[i] = 1;
    d[i] = 0;
  }
}
```

```cpp
}
int findpar(int x) {
  if (par[x] == x) return x;
  int p = par[x];
  par[x] = findpar(p);
  d[x] += d[p];
  return par[x];
}
bool unite(int a, int b, ll w) {
  int ra = findpar(a);
  int rb = findpar(b);
  if (ra == rb) {
    return (d[b] - d[a] == w);
  }
  if (sz[ra] < sz[rb]) {
    swap(a, b);
    swap(ra, rb);
    w = -w;
  }
  par[rb] = ra;
  d[rb] = d[a] + w - d[b];
  sz[ra] += sz[rb];
  return true;
}
ll dist(int a, int b) {
  findpar(a), findpar(b);
  return d[b] - d[a];
}
```

3 3.Geometry
3.1 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);
});
```

3.2 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;
}
```

3.3 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;
```

```cpp
  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;
}
```

3.4 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]);
});
```

3.5 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;
}
```

3.6 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));
}
```

3.7 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);
}
```

3.8 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));
}
// integer area
void computeIntArea() {
  int n; cin >> n;
  point arr[n];
  for (int i = 0; i < n; i++) {
    arr[i].read();
  }
  point a = {0, 0};
  ll ans = 0;
  for (int i = 0; i + 1 < n; i++) {
    ans += a.triangle(arr[i], arr[i + 1]);
  }
  ans += a.triangle(arr[n - 1],  arr[0]);
  cout << abs(ans) << "\n";
}
```

3.9 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;
}
```

3.10 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);
}
```

3.11 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)));
}
```

3.12 Half Plane Intersection

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

```cpp
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;
  }
```

```cpp
  while (len > 2 && dq[0].out(inter(dq[len-1],
     dq[len-2]))) {
    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;
}
```

3.13 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;
}
```

3.14 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;
}
```

3.15 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;
}
```

3.16 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); }
```

```cpp
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));
}
```

3.17 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;
}
```

3.18 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);
}
```

3.19 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;
}
```

3.20 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;
}
```

4 Notes

4.1 Geometry

4.1.1 Triangles

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

The area of a triangle using two sides and the included angle can be given as:

$$A = \frac{1}{2}ab\sin\angle C$$

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

4.1.2 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 $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

4.1.3 Spherical coordinates

$$x = r\sin\theta\cos\phi \qquad r = \sqrt{x^2 + y^2 + z^2}$$
$$y = r\sin\theta\sin\phi \qquad \theta = \mathrm{acos}(z/\sqrt{x^2+y^2+z^2})$$
$$z = r\cos\theta \qquad \phi = \mathrm{atan2}(y, x)$$

4.1.4 Pick's Theorem:

Given a lattice polygon with non-zero area, we define: S as the area of the polygon, I as the number of integer-coordinate points strictly inside the polygon, B as the number of integer-coordinate points on the boundary of the polygon. Then, Pick's Theorem states:

$$S = I + \frac{B}{2} - 1$$

The number of lattice points on segments (x1, y1) to (x2 , y2) is: gcd(abs(x2 - x1) , abs(y2 - y1)) + 1

4.1.5 Polygon

For a regular polygon with n sides and side length a, the circumradius R is given by:

$$R = \frac{a}{2\sin\left(\frac{\pi}{n}\right)}$$

4.1.6 Area of a Circular Segment

The area of a circular segment, which is the region enclosed by a chord and the corresponding arc, can be calculated using the formula:

$$A = \frac{R^2}{2}(\theta - \sin\theta)$$

where: R is the radius of the circle, θ is the central angle subtended by the chord, in radians.

4.2 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: $\sum_{k=0}^{n}\binom{n-k}{k} = F_{n+1}$

- Vandermonde's Identity: $\sum_{i=0}^{k}\binom{m}{i}\binom{n}{k-i} = \binom{m+n}{k}$

- If $f(n,k) = C(n,0) + C(n,1) + ... + C(n,k)$, 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 divisble by p.

- $\binom{n}{m} \equiv \lfloor\frac{n}{p}\rfloor(\mod p)$

4.3 Fibonacci Number

1. $k = A - B, F_A F_B = F_{k+1}F_A^2 + F_k F_A F_{A-1}$

2. $\sum_{i=0}^{n}F_i^2 = F_{n+1}F_n$ **3.** $\sum_{i=0}^{n}F_i F_{i+1} = F_{n+1}^2 - (-1)^n$

4. $\sum_{i=0}^{n}F_i F_{i+1} = F_{n+1}^2 - (-1)^n$ **5.** $\sum_{i=0}^{n}F_i F_{i-1} = \sum_{i=0}^{n-1}F_i F_{i+1}$

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. $\gcd(F_n, F_{n+1}) = \gcd(F_n, F_{n+2}) = \gcd(F_{n+1}, F_{n+2}) = 1$

4.4 Sums

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

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

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

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

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

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

4.5 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, (-1 < x \le 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, (-1 \le x \le 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, (-\infty < x < \infty)$$

$$(x+a)^{-n} = \sum_{k=0}^{\infty}(-1)^k\binom{n+k-1}{k}x^k a^{-n-k}$$

Generating Function

$$1/(1-x) = 1 + x + x^2 + x^3 + \dots$$

$$1/(1-ax) = 1 + ax + (ax)^2 + (ax)^3 + \dots$$

$$1/(1-x)^2 = 1 + 2x + 3x^2 + 4x^3 + \dots$$

$$1/(1-x)^3 = C(2,2) + C(3,2)x + C(4,2)x^2 + C(5,2)x^3 + \dots$$

$$1/(1-ax)^{(k+1)} = 1 + C(1+k,k)(ax) + C(2+k,k)(ax)^2 + C(3+k,k)(ax)^3 + \dots$$

$$x(x+1)(1-x)^-3 = 1 + x + 4x^2 + 9x^3 + 16x^4 + 25x^5 + \dots$$

$$e^x = 1 + x + (x^2)/2! + (x^3)/3! + (x^4)/4! + \dots$$

4.6 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

4.7 Number Theory

- HCN: 1e6(240), 1e9(1344), 1e12(6720), 1e14(17280), 1e15(26880), 1e16(41472)

- $gcd(a,b,c,d,...) = gcd(a, b-a, c-b, d-c, ...)$

- $gcd(a+k, b+k, c+k, d+k, ...) = gcd(a+k, b-a, c-b, d-c, ...)$

- 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 $x \equiv 2^{\frac{p-1}{4}}a^{\frac{p+3}{8}}$

4.7.1 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 $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

4.7.2 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 $n < 1e19$.

4.7.3 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.

4.7.4 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 $n-1$.

4.7.5 Totient

- If p is a prime $(p^k) = p^k - p^{k-1}$
- If a 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 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- $2*3*5*7*11*13*...$

4.7.6 Mobius function

$\mu(1) = 1$. $\mu(n) = 0$, if n is not squarefree. $\mu(n) = (-1)^s$, if n is the product of s distinct primes. Let f, F be functions on positive integers. 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. $\phi(n) = \sum_{d|n} \mu(d)\frac{n}{d}$. $\sum_{d|n} \mu(d) = 1$.
If f is multiplicative, then $\sum_{d|n} \mu(d)f(d) = \prod_{p|n}(1 - f(p))$, $\sum_{d|n} \mu(d)^2 f(d) = \prod_{p|n}(1 + f(p))$.

$$\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j)=1] = \sum_{k=1}^{n}\mu(k)\lfloor\frac{n}{k}\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}gcd(i,j) = \sum_{k=1}^{n}k\sum_{l=1}^{\lfloor\frac{n}{k}\rfloor}\mu(l)\lfloor\frac{n}{kl}\rfloor^2$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}gcd(i,j) = \sum_{k=1}^{n}(\frac{\lfloor\frac{n}{k}\rfloor(1+\lfloor\frac{n}{k}\rfloor)}{2})^2\sum_{d|k}\mu(d)kd$$

4.7.7 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: $\left(\frac{a}{p}\right) = a^{\left(\frac{p-1}{2}\right)} \pmod{p}$.

4.7.8 Jacobi symbol

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

4.7.9 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 $i = ind_g(a)$ modulo $\phi(m)$, such that $g^i \equiv a \pmod{m}$. $ind_g(a)$ has logarithm-like properties: $ind(1) = 0$, $ind(ab) = ind(a) + ind(b)$.
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 $nu \equiv i \pmod{p}$.)

4.7.10 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.

4.7.11 Pythagorean triples

Integer solutions of $x^2 + y^2 = z^2$ All relatively prime triples are given by: $x = 2mn, y = m^2 - n^2, z = m^2 + n^2$ where $m > n, \gcd(m,n) = 1$ and $m \not\equiv n \pmod 2$. All other triples are multiples of these. Equation $x^2 + y^2 = 2z^2$ is equivalent to $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

4.7.12 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 $ax + by$ $(x, y \geq 0)$, and the largest is $(a-1)(b-1) - 1 = ab - a - b$.

4.7.13 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.

4.8 Permutations
4.8.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800

n	11	12	13	14	15	16	17
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14

n	20	25	30	40	50	100	150	171
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX

4.8.2 Cycles

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

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

4.8.3 Derangements

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

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

4.8.4 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

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

4.9 Partitions and subsets
4.9.1 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

4.9.2 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

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

4.9.3 2nd Kaplansky's Lemma

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

4.9.4 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 $= \sum_{i=0}^{r} (-1)^i \binom{r}{i}(r-i)^n$

4.10 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}

4.11 General purpose numbers
4.11.1 Eulerian numbers

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. $\pi(j) > j$.

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

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

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

4.11.2 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,

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

4.11.3 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 $n \neq 1$.

4.11.4 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.
- 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:

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

4.11.5 Lucas Number

Number of edge cover of a cycle graph C_n is L_n

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

4.12 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 $\frac{a-kb}{a+b} \times C(a+b, a)$

4.13 Classical Problem

$F(n,k)$ = 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, $F(n,k) = G(n,k) - C(k,1)*G(n,k-1) + C(k,2)*G(n,k-2) - C(k,3)*G(n,k-3)...$

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, $G(n,k) = k^n$

4.14 Matching Formula
4.14.1 Normal Graph
MM + MEC = n (excluding vertex), IS + VC = G, MIS + MVC = G
4.14.2 Bipartite Graph
MIS = n - MBM, MVC = MBM, MEC = n - MBM

4.15 Inequalities
4.15.1 Titu's Lemma
For positive reals $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$,

$$\frac{a_1^2}{b_1} + \frac{a_2^2}{b_2} + \ldots + \frac{a_n^2}{b_n} \geq \frac{a_1 + a_2 + \ldots + a_n}{b_1 + b_2 + \ldots + b_n}$$

Equality holds if and only if $a_i = kb_i$ for a non-zero real constant k.

4.16 Games
4.16.1 Grundy numbers
For a two-player, normal-play (last to move wins) game on a graph (V,E): $G(x) = \text{mex}(\{G(y) : (x,y) \in E\})$, where $\text{mex}(S) = \min\{n \geq 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

4.16.2 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.

4.16.3 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*.

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

4.18 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.

4.19 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 T are same if and only if $S + T$ is a palindrome.

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

4.21 Convolution
- Hamming Distance: Replace 0 with -1 - SQRT Decomposition: Find block size, B = $\sqrt{(8*n)}$

4.22 Matrix Rotation
4.22.1 Anti-Clockwise Rotation
$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

4.22.2 Clockwise Rotation
$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

4.23 Common Formulas
4.23.1 Permutation
$$^nP_r = \frac{n!}{(n-r)!}$$

4.23.2 Combination
$$^nC_r = \frac{n!}{r!(n-r)!}$$

4.24 Logarithms
4.24.1 Change of Base Formula
$$\log_a x = \frac{\log_b x}{\log_b a}$$

4.25 Common Series Sums
4.25.1 Sum of first n positive integers
$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

4.25.2 Sum of first n odd positive integers
$$1 + 3 + 5 + \cdots + (2n-1) = n^2$$

4.25.3 Sum of first n even positive integers
$$2 + 4 + 6 + \cdots + 2n = n(n+1)$$

4.25.4 Sum of first n squares
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

4.25.5 Sum of first n cubes
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \left[\frac{n(n+1)}{2}\right]^2$$

4.26 Progressions
4.26.1 Arithmetic Progression
- **Sequence:** $a, a+d, a+2d, \ldots, a+(n-1)d$

- **Sum of first n terms:** $S_n = \frac{n}{2}[2a + (n-1)d]$

4.26.2 Geometric Progression
- **Sequence:** $a, ar, ar^2, \ldots, ar^{n-1}$

- **Sum (for $r > 1$):** $S_n = \frac{a(r^n - 1)}{r-1}$

- **Sum (for $r < 1$):** $S_n = \frac{a(1-r^n)}{1-r}$