Daffodil International University

DIU_EasyPeasy

_Taifu, sourov.cse, juhair300

Team Reference Document

Contents

1 Code 1
- 1.1 Build System Linux . . . 1
- 1.2 Build System Windows . . . 1
- 1.3 Stress Testing(check.sh) . . . 1
- 1.4 Stress Testing(gen.cpp) . . . 1
- 1.5 dbg . . . 2
- 1.6 pythonTemp . . . 2
- 1.7 2-SAT . . . 2
- 1.8 Aho Corasick . . . 2
- 1.9 Articulation Point and Bridges . . . 2
- 1.10 Bellman Ford . . . 3
- 1.11 Big Integer . . . 3
- 1.12 Centroid Decomposition Struct . . . 4
- 1.13 Centroid Decomposition . . . 4
- 1.14 Chinese Remainder Theorem . . . 4
- 1.15 Convex Hull . . . 4
- 1.16 ConvexHullTrick . . . 5
- 1.17 Custom Hash . . . 6
- 1.18 Custom Map(Pair Query) . . . 6
- 1.19 DSU(weighted) . . . 6
- 1.20 DSU . . . 6
- 1.21 Discrete Log . . . 6
- 1.22 Euler Phi . . . 7
- 1.23 Extended GCD . . . 7
- 1.24 Fenwick Tree . . . 7
- 1.25 Floyd Warshall . . . 7
- 1.26 GP Hash Table . . . 7
- 1.27 Geometric Sum . . . 7
- 1.28 Heavy-Light Decomposition . . . 7
- 1.29 HopcroftKarp . . . 8
- 1.30 Interval Set . . . 8
- 1.31 KMP . . . 8
- 1.32 LCA . . . 8
- 1.33 Linear Diophantine Equation . . . 8
- 1.34 Manacher . . . 9
- 1.35 Matrix Exponentiation . . . 9
- 1.36 Mint . . . 9
- 1.37 Mobius Function . . . 9
- 1.38 N-th Permutation . . . 9
- 1.39 PBDS . . . 9
- 1.40 Polynomial Interpolation . . . 9
- 1.41 Prefix Sum 3D . . . 10
- 1.42 Segment Tree(Persistent) . . . 10
- 1.43 Segment Tree(Special Variant) . . . 10
- 1.44 Segment Tree . . . 10
- 1.45 Seive upto 1e9 . . . 10
- 1.46 Seive(Linear) . . . 11
- 1.47 Seive(Segmented) . . . 11
- 1.48 Seive . . . 11
- 1.49 Sparse Table . . . 11
- 1.50 SquareRoot Decomposition . . . 11
- 1.51 String Hashing . . . 12
- 1.52 Strongly_Connected_Components(SCC) . . . 12
- 1.53 Submask Enumeration . . . 12
- 1.54 Suffix Array . . . 12
- 1.55 Suffix Automaton . . . 13
- 1.56 Ternary Search . . . 13
- 1.57 Topological Sorting . . . 13
- 1.58 Trie . . . 13
- 1.59 Z_algo . . . 14
- 1.60 int128 . . . 14
- 1.61 josephus problem . . . 14
- 1.62 nCr and nPr-1 . . . 14
- 1.63 nCr and nPr-2 . . . 14
- 1.64 notes . . . 14
- 1.65 ntt . . . 14

2 Geometry 15
- 2.1 Angular Sort . . . 15
- 2.2 CircleCircleIntersection . . . 15
- 2.3 CircleLineIntersection . . . 15
- 2.4 Closest Pair of Points . . . 15
- 2.5 ComputeCentroid . . . 15
- 2.6 ComputeCircleCenter . . . 15
- 2.7 ComputeLineIntersection . . . 15
- 2.8 ComputeSignedArea . . . 15
- 2.9 Convex Hull . . . 15
- 2.10 DistancePointPlane . . . 15
- 2.11 DistancePointSegment . . . 15
- 2.12 Half Plane Intersection . . . 15
- 2.13 IsSimple . . . 16
- 2.14 LinesCollinear . . . 16
- 2.15 LinesParallel . . . 16
- 2.16 Point . . . 16
- 2.17 PointInPolygon . . . 16
- 2.18 ProjectPointLine . . . 16
- 2.19 ProjectPointSegment . . . 16
- 2.20 SegmentsIntersect . . . 16

3 Notes 16
- 3.1 Geometry . . . 16
- 3.2 Binomial Coefficent . . . 17
- 3.3 Fibonacci Number . . . 17
- 3.4 Sums . . . 17
- 3.5 Series . . . 17
- 3.6 Pythagorean Triples . . . 17
- 3.7 Number Theory . . . 17
- 3.8 Permutations . . . 18
- 3.9 Partitions and subsets . . . 18
- 3.10 Coloring . . . 18
- 3.11 General purpose numbers . . . 19
- 3.12 Ballot Theorem . . . 19
- 3.13 Classical Problem . . . 19
- 3.14 Matching Formula . . . 19
- 3.15 Inequalities . . . 19
- 3.16 Games . . . 19
- 3.17 Tree Hashing . . . 19
- 3.18 Permutation . . . 19
- 3.19 String . . . 19
- 3.20 Bit . . . 19
- 3.21 Convolution . . . 19
- 3.22 Matrix Rotation . . . 19
- 3.23 Common Formulas . . . 19
- 3.24 Logarithms . . . 19
- 3.25 Common Series Sums . . . 19
- 3.26 Progressions . . . 19

1 Code

1.1 Build System Linux

```
{
  "cmd" : ["ulimit -s 268435456;g++ -std=c++20 $file_name -o $file_base_name && timeout 4s ./$file_base_name<input.txt>output.txt"],
  "selector" : "source.c",
  "shell": true,
  "working_dir" : "$file_path"
}
```

1.2 Build System Windows

```
{
  "cmd": ["g++.exe","-std=c++20", "${file}", "-o", "${file_base_name}.exe", "&&" , "${file_base_name}.exe<input.txt>output.txt"],
  "selector":"source.cpp",
  "shell":true,
  "working_dir":"$file_path"
}
```

1.3 Stress Testing(check.sh)

```
// chmod u+x check.sh
// ./check.sh
set -e
g++ gen.cpp -o gen
g++ code.cpp -o code
g++ brute.cpp -o brute
for ((i = 1; ; ++i)); do
  echo "Passed on TestCase: " $i
  ./gen $i > in
  ./code < in > out1
  ./brute < in > out2
  diff -Z out1 out2 || break
done
echo -e "WA on the following test:"
cat in
echo -e "\nExpected:"
cat out2
echo -e "\nFound:"
cat out1
```

1.4 Stress Testing(gen.cpp)

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
inline ll gen_random(ll l, ll r) {
  return uniform_int_distribution<ll>(l, r)(rng);
}
inline double gen_random_real(double l, double r) {
  return uniform_real_distribution<double>(l, r)(rng);
}
int main(int argc, char* args[]) {
  int _ = atoi(args[1]);
  rng.seed(_);
  int n = gen_random(1, 5);
  vector<int> per;
  for (int i = 0; i < n; ++i) {
    per.push_back(i + 1);
  }
  shuffle(per.begin(), per.end(), rng);
  return 0;
}
```

1.5 dbg

```
#include <bits/stdc++.h>
using namespace std;
string to_string(const char c) {
  return "'" + string(1, c) + "'";
}
string to_string(const string& s) {
  return '"' + s + '"';
}
string to_string(const char* s) {
  return to_string((string) s);
}
string to_string(bool b) {
  return (b ? "true" : "false");
}
template <size_t N>
string to_string(bitset<N> v) {
  return v.to_string();
}
template <typename A, typename B>
string to_string(pair<A, B> p) {
  return "(" + to_string(p.first) + ", " +
  ↪  to_string(p.second) + ")";
}
template <typename A>
string to_string(A v) {
  bool first = true;
  string res = "{";
  for (const auto &x : v) {
    if (!first) {
      res += ", ";
    }
    first = false;
    res += to_string(x);
  }
  res += "}";
  return res;
}
void dbg_out() { cerr << endl; }
template <typename Head, typename... Tail>
void dbg_out(Head H, Tail... T) {
  cerr << " " << to_string(H);
  dbg_out(T...);
}
#define dbg(...) cerr << "Line " << __LINE__ << ": " <<
↪  "[" << #__VA_ARGS__ << "]:", dbg_out(__VA_ARGS__)
/*
#include "dbg.h"
int main() {
  char c = 'a';
  int a = 2;
  string s = "diu";
  vector<int> v = {2, 1, 3};
  set<int> st = {2, 1, 3};
  map<int, int> cnt;
  cnt[0]++, cnt[1]++, cnt[0]++;
  dbg(c, a, s, v, st, cnt);
  dbg('c');
  dbg("diu");
  bitset<5> bs = 5;
  dbg(bs);
  dbg(int(bs[2]));
}
*/
```

1.6 pythonTemp

```
import math, sys
input = sys.stdin.buffer.readline
write = sys.stdout.write
tc = int(input())
for t in range (tc):
  h1, h2, b = map(int, input().split())
  h = math.log(h2 / h1)
  bb = math.log((b - 1) / b)
  ans = math.ceil(h / bb)
  print(ans)
```

1.7 2-SAT

```
struct _2SAT {
  int N;
  vector<bool> vis, value;
  vector<int> order, comp;
  vector<vector<int>> adj, adjT;
  _2SAT(int n) : N(n), adj(2 * n), adjT(2 * n), vis(2
  ↪  * n), comp(2 * n), value(2 * n) {}
  void dfs1(int u) {
    vis[u] = true;
    for (auto v: adj[u]) {
      if (!vis[v]) {
        dfs1(v);
      }
    }
    order.push_back(u);
  }
  void dfs2(int u, int cnt) {
    comp[u] = cnt;
    for (auto v: adjT[u]) {
      if (!comp[v]) {
        dfs2(v, cnt);
      }
    }
  }
  void Kosaraju() {
    for (int i = 0; i < 2 * N; ++i) {
      if (!vis[i]) dfs1(i);
    }
    reverse(order.begin(), order.end());
    int cnt = 1;
    for (auto u: order) {
      if (!comp[u]) {
        dfs2(u, cnt++);
      }
    }
  }
  bool assignment() {
    Kosaraju();
    for (int i = 0; i < N; ++i) {
      if (comp[i] == comp[i + N]) {
        return false;
      }
      value[i] = comp[i] < comp[i + N] ? 0 : 1;
    }
    return true;
  }
  void addDisjunction(int a, bool pos_a, int b, bool
  ↪  pos_b) { // a V b
    int neg_a = a + N, neg_b = b + N;
    if (!pos_a) swap(a, neg_a);
    if (!pos_b) swap(b, neg_b);
    adj[neg_a].push_back(b);
    adj[neg_b].push_back(a);
    adjT[a].push_back(neg_b);
    adjT[b].push_back(neg_a);
  }
};
```

1.8 Aho Corasick

```
const int N = 1e6 + 3, A = 26;
int trie[N][A], node[N], dp[N];
int total = 0;
void add(string& s, int i) {
  int u = 0;
  for (char c: s) {
    int k = c - 'a';
    if (!trie[u][k]) {
      trie[u][k] = ++total;
    }
    u = trie[u][k];
  }
  node[i] = u;
}
vector<int> ord;
int slink[N];
void build() {
  queue<int> q;
  q.push(0);
  while (q.size()) {
    int p = q.front();
    q.pop();
    ord.push_back(p);
    for (int c = 0; c < A; ++c) {
      int u = trie[p][c];
      if (!u) continue;
      q.push(u);
      if (!p) continue;
      int v = slink[p];
      while (v and !trie[v][c]) v = slink[v];
      if (trie[v][c]) slink[u] = trie[v][c];
    }
  }
}
void solve() {
  build();
  int u = 0;
  for (char c: text) {
    c -= 'a';
    while (u and !trie[u][c]) u = slink[u];
    u = trie[u][c];
    dp[u]++;
  }
  reverse(ord.begin(), ord.end());
  for (int u: ord) {
    dp[slink[u]] += dp[u];
  }
}
```

1.9 Articulation Point and Bridges

```
// Articulation point
vector<vector<int>> adj;
vector<int> tin, low;
vector<bool> vis;
int timer;
void is_cutpoint(int v) {
  // process the cutpoint
}
void dfs(int v, int p = -1) {
  vis[v] = true;
  tin[v] = low[v] = timer++;
  int children = 0;
  for (int u : adj[v]) {
    if (u == p) continue;
    if (vis[u]) {
      low[v] = min(low[v], tin[u]);
    } else {
      dfs(u, v);
```

```cpp
      low[v] = min(low[v], low[u]);
      if (low[u] >= tin[v] && p != -1) {
        is_cutpoint[v] = true;
      }
      ++children;
    }
  }
  if(p == -1 && children > 1) {
    is_cutpoint[v] = true;
  }
}
void find_cutpoints(int n) {
  timer = 0;
  vis.assign(n + 1, false);
  is_cutpoint.assign(n + 1, false);
  tin.assign(n + 1, -1);
  low.assign(n + 1, -1);
  for (int i = 1; i <= n; ++i) {
    if (!vis[i]) {
      dfs (i);
    }
  }
}
// Bridges
vector<vector<int>> adj;
vector<int> tin, low;
vector<bool> vis;
int timer;
void is_bridge(int v,int to) {
  //process the found bridge
}
void dfs(int v, int p = -1) {
  vis[v] = true;
  tin[v] = low[v] = timer++;
  bool parent_skipped = false;
  for (int u : adj[v]) {
    if (u == p && !parent_skipped) {
      parent_skipped = true;
      continue;
    }
    if (vis[u]) {
      low[v] = min(low[v], tin[u]);
    } else {
      dfs(u, v);
      low[v] = min(low[v], low[u]);
      if (low[u] > tin[v]) {
        is_bridge(v, u);
      }
    }
  }
}
void find_bridges() {
  timer = 0;
  vis.assign(n, false);
  tin.assign(n, -1);
  low.assign(n, -1);
  for (int i = 0; i < n; ++i) {
    if (!vis[i]) {
      dfs(i);
    }
  }
}
```

1.10 Bellman Ford

```cpp
const int INF = 1e9;
struct Edge {
  int u, v, w;
};
void solve() {
  int n, m;
  cin >> n >> m;
  vector<Edge> e(m);
  for (int i = 0; i < m; ++i) {
    cin >> e[i].u >> e[i].v >> e[i].w;
  }
  vector<int> d(n + 1, INF);
  d[1] = 0;   // distance of source node
  vector<int> p(n + 1, -1);   // parent vector
  int x;
  for (int i = 1; i <= n; ++i) {
    x = -1;
    for (auto [u, v, w]: e)
      if (d[u] < INF and d[u] + w < d[v]) {
        d[v] = d[u] + w;
        p[v] = u;
        x = v;
      }
  }
  if (x == -1) cout << "No negative cycle found\n";
  else {            // Path Printing
    int y = x;
    for (int i = 0; i < n; ++i) y = p[y];

    vector<int> path;
    for (int cur = y; ; cur = p[cur]) {
      path.push_back(cur);
      if (cur == y && path.size() > 1) break;
    }
    reverse(path.begin(), path.end());

    cout << "Negative cycle: ";
    for (int u : path) cout << u << " ";
    cout << "\n";
  }
}
```

1.11 Big Integer

```cpp
class BIG_INT {
private:
  string result;
public:
  string bigfinder(string a, string b){
    if(a.size() < b.size()) swap(a, b);
    string d = b;
    reverse(full(b));
    while(b.size() < a.size()) b.pb('0');
    reverse(full(b));
    int i = 0;
    while(a[i]){
      if(a[i] > b[i]) return a;
      else if(a[i] < b[i]) return d;
      i++;
    }
    return "same";
  }
  llu stringtonumber(string a){
    llu n = 0;
    for(llu i = 0; a[i]; i++) n = ( n*10 ) + (a[i]-48);
    return n;
  }
  string add(string a,string b){
    result.clear();
    reverse(full(a));
    reverse(full(b));
    if(a.size() < b.size()) swap(a, b);
    while(b.size() < a.size()) b.pb('0');
    llu i = 0, carry = 0;
    while(a[i]){
      carry = carry + a[i]-48 + b[i]-48;
      result.pb((carry %10) + 48);
      carry = carry / 10;
      i++;
    }
    while(carry > 9){
      result.pb((carry % 10) + 48);
      carry = carry / 10;
    }
    if(carry != 0) result.pb(carry + 48);
    reverse(full(result));
    return result;
  }
  string subtraction(string a, string b){
    result.clear();
    bool flag = true;
    if(bigfinder(a, b) == b){
      swap(a, b);
      flag = false;
    }
    reverse(full(a));
    reverse(full(b));
    while(b.size() < a.size()) b.pb('0');
    int i = 0, carry = 0, x = 0;
    while(a[i]){
      if(b[i] > a[i]) x = (a[i]-48) + 10;
      else x = a[i]-48;
      carry = x-(carry + (b[i]-48));
      result.pb(carry+48);
      carry = x / 10;
      i++;
    }
    while(result[result.size()-1] == '0' and result.size() > 1)
      result.erase(result.size()-1, 1);
    if(!flag) result.pb('-');
    reverse(full(result));
    return result;
  }
  string multiplication(string a, string b){
    if(b.size() > a.size()) swap(a, b);
    reverse(full(a));
    reverse(full(b));
    while(a.size() > b.size()) b.pb('0');
    vector < string > x;
    for(llu i = 0; b[i]; i++){
      llu carry = 0;
      string str;
      for(llu j = 0; a[j]; j++){
        str += ((((b[i]-48)*(a[j]-48))+carry)%10)+48;
        carry = (((b[i]-48)*(a[j]-48))+carry)/10;
      }
      if(carry > 0) str += carry + 48;
      reverse(full(str));
      llu zero = i;
      while(zero--) str += '0';
      x.pb(str); }
    llu len = x.size();
    if(len == 1) result = x[0];
    else{
      for(llu i = 0; i < len-1; i++){
        x[i+1] = add(x[i], x[i+1]);
      }
    }
    result = x[len-1];
    while(result[0] == '0' and result.size() > 1)
      result.erase(result.begin() + 0);
    return result;
  }
};
// Big Integer Division
void bigDivision() {
  string a = "50";
  ll b = 6;
  ll len = a.length(), mod = 0, d = Digit(b), lowest =
    0, i = 0;
  while (i < d or lowest < b) {
    lowest = (lowest * 10) + (a[i] - 48);
    i++;
  }
  while (i < len + 1) {
```

```
    mod = lowest % b;
    lowest = (mod * 10) + (a[i] - 48);
    if (b > lowest) {
      lowest = (lowest * 10) + (a[i] - 48);
      i++;
    }
    i++;
  }
  cout << mod << endl;
}
```

1.12 Centroid Decomposition Struct

```
struct CentroidDecomposition {
  set<int> adj[N];
  map<int, int> dis[N];
  int sz[N], par[N], ans[N];
  void init(int n) {
    for(int i = 1 ; i <= n ; ++i) {
      adj[i].clear(), dis[i].clear();
      ans[i] = inf;
    }
  }
  void addEdge(int u, int v) {
    adj[u].insert(v); adj[v].insert(u);
  }
  int dfs(int u, int p) {
    sz[u] = 1;
    for(auto v : adj[u]) if(v != p) {
      sz[u] += dfs(v, u);
    }
    return sz[u];
  }
  int centroid(int u, int p, int n) {
    for(auto v : adj[u]) if(v != p) {
      if(sz[v] > n / 2) return centroid(v, u, n);
    }
    return u;
  }
  void dfs2(int u, int p, int c, int d) {
    dis[c][u] = d;
    for(auto v : adj[u]) if(v != p) {
      dfs2(v, u, c, d + 1);
    }
  }
  void build(int u, int p) {
    int n = dfs(u, p);
    int c = centroid(u, p, n);
    if(p == -1) p = c;
    par[c] = p;
    dfs2(c, p, c, 0);
    vector<int> tmp(adj[c].begin(), adj[c].end());
    for(auto v : tmp) {
      adj[c].erase(v); adj[v].erase(c);
      build(v, c);
    }
  }
  void modify(int u) {
    for(int v = u ; v != 0 ; v = par[v]) {
      ans[v] = min(ans[v], dis[v][u]);
    }
  }
  int query(int u) {
    int mn = inf;
    for(int v = u ; v != 0 ; v = par[v]) {
      mn = min(mn, ans[v] + dis[v][u]);
    }
    return mn;
  }
} cd;
```

1.13 Centroid Decomposition

```
const int N = 2e5+5;
int n, k;
vector<int> adj[N];
int sz[N], cen[N];
ll ans = 0;
void dfs_sz(int u, int p) {
  sz[u] = 1;
  for (auto v: adj[u]) {
    if (v != p and !cen[v]) {
      dfs_sz(v, u);
      sz[u] += sz[v];
    }
  }
}
int get_cen(int u, int p, int s) {
  for (auto v: adj[u]) {
    if (v != p and !cen[v] and 2 * sz[v] > s)  return
    ↪ get_cen(v, u, s);
  }
  return u;
}
int t, tin[N], tout[N], nodes[N], dep[N];
void dfs(int u, int p) {
  nodes[t] = u;
  tin[u] = t++;
  for (auto v: adj[u]) {
    if (v != p and !cen[v]) {
      dep[v] = dep[u] + 1;
      dfs(v, u);
    }
  }
  tout[u] = t - 1;
}
void go(int u) {
  dfs_sz(u, u);
  int c = get_cen(u, u, sz[u]);
  cen[c] = 1;
  t = 0;
  dep[c] = 0;
  dfs(c, c);
  int cnt[t]{1};
  for (auto v: adj[c]) {
    if (!cen[v]) {
      for (int i = tin[v]; i <= tout[v]; ++i) {
        int w = nodes[i];
        int req = k - dep[w];
        if (req >= 0 and req < t) {
          ans += cnt[req];
        }
      }
      for (int i = tin[v]; i <= tout[v]; ++i) {
        int w = nodes[i];
        cnt[dep[w]]++;
      }
    }
  }
  for (auto v: adj[c]) {
    if (!cen[v]) {
      go(v);
    }
  }
}
void solve () {
  cin >> n >> k;
  for (int e = 0; e < n - 1; ++e) {
    int u, v;  cin >> u >> v;  u--, v--;
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  go(0);
  cout << ans << "\n";
}
```

1.14 Chinese Remainder Theorem

```
ll extended_euclidean(ll a, ll b, ll& x, ll& y) {
  x = 1, y = 0;
  ll x1 = 0, y1 = 1, a1 = a, b1 = b;
  while (b1) {
      ll q = a1 / b1;
      tie(x, x1) = make_tuple(x1, x - q * x1);
      tie(y, y1) = make_tuple(y1, y - q * y1);
      tie(a1, b1) = make_tuple(b1, a1 - q * b1);
  }
  return a1;
}
pair<ll, ll> CRT( vector<ll> A, vector<ll> M ) {
  ll n = A.size();
  ll a1 = A[0];
  ll m1 = M[0];
  for ( ll i = 1; i < n; i++ ) {
      ll a2 = A[i];
      ll m2 = M[i];
      ll g = __gcd(m1, m2);
      if ( a1 % g != a2 % g) return { -1, -1};
      ll p, q;
      extended_euclidean(m1 / g, m2 / g, p, q);
      ll mod = m1 / g * m2;
      if (mod > 1e10) return { -1, -1};
      ll x = (a1 * (m2 / g) * q + a2 * (m1 / g) * p) %
      ↪  mod;
      a1 = x;
      if (a1 < 0) a1 += mod;
      m1 = mod;
  }
  return {a1, m1};
}
```

1.15 Convex Hull

```
struct Point {
  ll x, y;
  Point () {
    this->x = 0;
    this->y = 0;
  }
  Point (ll x, ll y) {
    this->x = x;
    this->y = y;
  }
  bool operator ==(const Point& p) const {
    return (this->x == p.x and this->y == p.y);
  }
  bool operator <(const Point& p) const {
    return make_pair(this->x, this->y) <
    ↪  make_pair(p.x, p.y); // with respect to x-axis
    // // with respect to angle from (0, 0)
    // if (*this * p == 0) {
    //   return dis() < p.dis();
    // }
    // return (*this * p < 0);
  }
  void operator -=(const Point& p) {
    this->x -= p.x;
    this->y -= p.y;
  }
  Point operator -(const Point& p) const {
    Point q;
    q.x = this->x - p.x;
    q.y = this->y - p.y;
    return q;
  }
  ll operator *(const Point& p) const {
    return x * p.y - y * p.x;
  }
  ll triangle(const Point& a, const Point& b) {
```

```cpp
    return (a - *this) * (b - *this);
  }
  pair<double, double> rotate(double deg) {
    deg = deg * M_PI / 180.0;
    return {x * cos(deg) - y * sin(deg), x * sin(deg) + y * cos(deg)};
  }
  bool isInside(Point& a, Point& b) const { // if p is inside segment a-b
    if ((a - *this) * (b - *this) != 0) return false;
    bool d1 = this->x >= min(a.x, b.x) and this->x <= max(a.x, b.x);
    bool d2 = this->y >= min(a.y, b.y) and this->y <= max(a.y, b.y);
    return d1 and d2;
  }
  bool rayIntersect(Point a, Point b) {
    Point q(this->x, INT32_MAX);  // if p-q ray intersects segment a-b
    for (int rep = 0; rep < 2; ++rep) {
      if ((a - *this) * (q - *this) <= 0 and (b - *this) * (q - *this) > 0 and (a - *this) * (b - *this) < 0) {
        return true;
      }
      swap(a, b);
    }
    return false;
  }
  friend istream& operator >>(istream& cin, Point& p) {
    cin >> p.x >> p.y;
    return cin;
  }
  friend ostream& operator <<(ostream& cout, const Point& p) {
    cout << p.x << " " << p.y;
    return cout;
  }
};
// upper and lower part
void solve() {
  int n;
  cin >> n;
  vector<Point> v(n);
  for (int i = 0; i < n; ++i) {
    cin >> v[i];
  }
  sort(v.begin(), v.end());
  vector<Point> hull;
  for (int rep = 0; rep < 2; ++rep) {
    const int sz = hull.size();
    for (auto C: v) {
      while (hull.size() >= sz + 2) {
        Point A = hull.end()[-2];
        Point B = hull.end()[-1];
        if (((B - A) * (C - A)) <= 0) {
          break;
        }
        hull.pop_back();
      }
      hull.push_back(C);
    }
    hull.pop_back();
    reverse(v.begin(), v.end());
  }
  cout << hull.size() << "\n";
  for (auto p: hull) {
    cout << p << "\n";
  }
}
// sorting by angle
void solve() {
  int n;
  cin >> n;
  vector<Point> v(n);
  for (int i = 0; i < n; ++i) {
    cin >> v[i];
    if (make_pair(v[i].x, v[i].y) < make_pair(v[0].x, v[0].y)) {
      swap(v[i], v[0]);
    }
  }
  for (int i = 1; i < n; ++i) {
    v[i] -= v[0];
  }
  sort(v.begin() + 1, v.end());
  int j = n - 1;
  while (j >= 2 and v[j] * v[j - 1] == 0) {
    --j;
  }
  reverse(v.begin() + j, v.end());
  vector<Point> hull;
  hull.push_back(Point{0, 0});
  for (int i = 1; i < n; ++i) {
    auto C = v[i];
    while (hull.size() >= 2) {
      Point A = hull.end()[-2];
      Point B = hull.end()[-1];
      if (((B - A) * (C - A)) <= 0) {
        break;
      }
      hull.pop_back();
    }
    hull.push_back(C);
  }
  cout << hull.size() << "\n";
  for (auto& p: hull) {
    p += v[0];
    cout << p << "\n";
  }
}
```

1.16 ConvexHullTrick

```cpp
/*
 * Dynamic version of data structure
 * to be used in dynamic programming optimisation
 * called "Convex Hull trick"
 * 'Dynamic' means that there is no restriction on adding lines order
 */
class ConvexHullDynamic
{
  typedef long long coef_t;
  typedef long long coord_t;
  typedef long long val_t;

  /*
   * Line 'y=a*x+b' represented by 2 coefficients 'a' and 'b'
   * and 'xLeft' which is intersection with previous line in hull(first line has -INF)
   */
private:
  struct Line
  {
    coef_t a, b;
    double xLeft;

    enum Type {line, maxQuery, minQuery} type;
    coord_t val;

    explicit Line(coef_t aa = 0, coef_t bb = 0) :
      a(aa), b(bb), xLeft(-INFINITY), type(Type::line), val(0) {}
    val_t valueAt(coord_t x) const { return a * x + b; }
    friend bool areParallel(const Line& l1, const Line& l2) { return l1.a == l2.a; }
    friend double intersectX(const Line& l1, const Line& l2) { return areParallel(l1, l2) ? INFINITY : 1.0 * (l2.b - l1.b) / (l1.a - l2.a); }
    bool operator<(const Line& l2) const
    {
      if (l2.type == line)
        return this->a     < l2.a;
      if (l2.type == maxQuery)
        return this->xLeft < l2.val;
      if (l2.type == minQuery)
        return this->xLeft > l2.val;
    }
  };
private:
  bool            isMax; //whether or not saved envelope is top(search of max value)
  std::set<Line>  hull;  //envelope itself
private:
  /*
   * INFO:        Check position in hull by iterator
   * COMPLEXITY:  O(1)
   */
  bool hasPrev(std::set<Line>::iterator it) { return it != hull.begin(); }
  bool hasNext(std::set<Line>::iterator it) { return it != hull.end() && std::next(it) != hull.end(); }
  /*
   * INFO:        Check whether line l2 is irrelevant
   * NOTE:        Following positioning in hull must be true
   *              l1 is next left to l2
   *              l2 is right between l1 and l3
   *              l3 is next right to l2
   * COMPLEXITY:  O(1)
   */
  bool irrelevant(const Line& l1, const Line& l2, const Line& l3) { return intersectX(l1, l3) <= intersectX(l1, l2); }
  bool irrelevant(std::set<Line>::iterator it)
  {
    return hasPrev(it) && hasNext(it)
           && (    isMax && irrelevant(*std::prev(it), *it, *std::next(it))
                || !isMax && irrelevant(*std::next(it), *it, *std::prev(it)) );
  }
  /*
   * INFO:        Updates 'xValue' of line pointed by iterator 'it'
   * COMPLEXITY:  O(1)
   */
  std::set<Line>::iterator updateLeftBorder(std::set<Line>::iterator it)
  {
    if (isMax && !hasPrev(it) || !isMax && !hasNext(it))
      return it;

    double val = intersectX(*it, isMax ? *std::prev(it) : *std::next(it));
    Line buf(*it);
    it = hull.erase(it);
    buf.xLeft = val;
    it = hull.insert(it, buf);
    return it;
  }
```

```cpp
public:
  explicit ConvexHullDynamic(bool isMax): isMax(isMax)
   {}
  /*
   * INFO:        Adding line to the envelope
   *              Line is of type 'y=a*x+b' represented
   by 2 coefficients 'a' and 'b'
   * COMPLEXITY:  Adding N lines(N calls of function)
   takes O(N*log N) time
   */
  void addLine(coef_t a, coef_t b)
  {
    //find the place where line will be inserted in set
    Line l3 = Line(a, b);
    auto it = hull.lower_bound(l3);
    //if parallel line is already in set, one of them
     becomes irrelevant
    if (it != hull.end() && areParallel(*it, l3))
    {
      if (isMax && it->b < b || !isMax && it->b > b)
        it = hull.erase(it);
      else
        return;
    }
    //try to insert
    it = hull.insert(it, l3);
    if (irrelevant(it)) { hull.erase(it); return; }
    //remove lines which became irrelevant after
     inserting line
    while (hasPrev(it) && irrelevant(std::prev(it)))
     hull.erase(std::prev(it));
    while (hasNext(it) && irrelevant(std::next(it)))
     hull.erase(std::next(it));
    //refresh 'xLine'
    it = updateLeftBorder(it);
    if (hasPrev(it))
      updateLeftBorder(std::prev(it));
    if (hasNext(it))
      updateLeftBorder(std::next(it));
  }
  /*
   * INFO:        Query, which returns max/min(depends
     on hull type - see more info above) value in
     point with abscissa 'x'
   * COMPLEXITY:  O(log N), N-amount of lines in hull
   */
  val_t getBest(coord_t x) const
  {
    Line q;
    q.val = x;
    q.type = isMax ? Line::Type::maxQuery :
     Line::Type::minQuery;
    auto bestLine = hull.lower_bound(q);
    if (isMax) --bestLine;
    return bestLine->valueAt(x);
  }
};
```

1.17 Custom Hash

```cpp
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM = chrono::stead
     y_clock::now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};
unordered_map<long long int, int, custom_hash> mp; //
   this will work when the key is an int or long long
   int
// hash pair
struct hash_pair {
  template <class T1, class T2>
  size_t operator()(const pair<T1, T2>& p) const {
      custom_hash hasher; // Create an instance of
       custom_hash
      size_t hash1 = hasher(p.first);
      size_t hash2 = hasher(p.second);
      // Combine hashes
      if (hash1 != hash2) {
        return hash1 ^ (hash2 << 1); // Better mixing
      }
      return hash1;
  }
};
```

1.18 Custom Map(Pair Query)

```cpp
// a1 <= a2 <= a3 <= a4......
// b1 >= b2 >= b3 >= b4......
map<ll, ll> mp;
void insert(ll a, ll b) {
  auto it = mp.lower_bound(a);
  if (it != mp.end() and it->second >= b) return;
  it = mp.insert(it, {a, b});
  it->second = b;
  while (it != mp.begin() and prev(it)->second <= b) {
    mp.erase(prev(it));
  }
}
// returns the largest b among the a's that are greater
 than or equal to x
ll query(ll x) {
  auto it = mp.lower_bound(x);
  if (it == mp.end()) return 0;
  return it->second;
}
```

1.19 DSU(weighted)

```cpp
const int N = 2e5 + 3;
int par[N], sz[N];
long long w[N];
int find(int u) {
  if (par[u] == u) return u;
  int p = find(par[u]);
  w[u] += w[par[u]];
  return par[u] = p;
}
bool unite(int a, int b, ll d) {
  int u = find(a), v = find(b);
  if (u == v) return w[a] - w[b] == d;
  if (sz[u] < sz[v]) {
    w[u] += d + w[b] - w[a];
    par[u] = v;
    sz[v] += sz[u];
  } else {
    w[v] += w[a] - d - w[b];
    par[v] = u;
    sz[u] += sz[v];
  }
  return true;
}
void solve() {
  int n, q;
  cin >> n >> q;
  for (int i = 1; i <= n; ++i) {
    par[i] = i;
    sz[i] = 1;
  }
  for (int i = 1; i <= q; ++i) {
    int a, b, d;
    cin >> a >> b >> d;
    if (unite(a, b, d)) cout << i << " ";
  }
  cout << endl;
}
```

1.20 DSU

```cpp
const int N = 1e5 + 9;
int parent[N], sz[N];
void make_set(int v) {
  parent[v] = v;
  sz[v] = 1;
}
int find_set(int v) {
  if (v == parent[v]) return v;
  return parent[v] = find_set(parent[v]);
}
void union_sets(int a, int b) {
  a = find_set(a);
  b = find_set(b);
  if (a != b) {
    if (sz[a] < sz[b]) swap(a, b);
    parent[b] = a;
    sz[a] += sz[b];
  }
}
```

1.21 Discrete Log

```cpp
// Returns minimum x for which a ^ x % m = b % m, a and
 m are coprime.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;
    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }
    for (int p = 1, cur = 1; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur];
            return ans;
        }
    }
    return -1;
}
// a and m are not coprime:
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
```

```
    }
    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;
    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }
    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

1.22 Euler Phi

```
1. phi(n) = n * (p1 - 1) / p1 * (p2 - 1) / p2 . . . .
2. gcd d: phi(n / d)
3. Sum of coprime numbers of an integer = phi(n) * n /
 ↪  2
4. N = ∑phi(d) where, d | N
5. Code:
vector<int> phi(n + 1);
void prec(int n) {  //nlogn
  phi[1] = 1;
  for (int i = 2; i <= n; i++)
    phi[i] = i - 1;
  for (int i = 2; i <= n; i++)
    for (int j = 2 * i; j <= n; j += i)
      phi[j] -= phi[i];
}
int phi(int n) { //sqrt(n)
  int result = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      while (n % i == 0) n /= i;
      result -= result / i;
    }
  }
  if (n > 1) result -= result / n;
  return result;
}
```

1.23 Extended GCD

```
ll egcd(ll a, ll b, ll &x, ll &y) {
  if (b == 0) {
    x = 1; y = 0;
    return a;
  }
  ll x1, y1;
  ll d = egcd(b, a % b, x1, y1);
  x = y1; y = x1 - y1 * (a / b);
  return d;
}
```

1.24 Fenwick Tree

```
struct FenwickTree {
  vector<ll> bit;
  ll n;

  FenwickTree(int n) {
    this->n = n;
    bit.assign(n, 0);
  }

  FenwickTree(vector<ll> const &a) :
 ↪  FenwickTree(a.size()){
    for (int i = 0; i < n; i++) {
      bit[i] += a[i];
      int r = i | (i + 1);
      if (r < n) bit[r] += bit[i];
    }
  }

  ll sum(int r) {
    ll ret = 0;
    for (; r >= 0; r = (r & (r + 1)) - 1)
      ret += bit[r];
    return ret;
  }

  ll sum(int l, int r) {
    return sum(r) - sum(l - 1);
  }

  void add(int idx, ll delta) {
    for (; idx < n; idx = idx | (idx + 1))
      bit[idx] += delta;
  }
};
struct FenwickTree2D {
  vector<vector<int>> bit;
  int n, m;

  FenwickTree2D(int n, int m) {
    this->n = n;
    this->m = m;
    bit.assign(n, vector<int>(m, 0));
  }

  int sum(int x, int y) {
    int ret = 0;
    for (int i = x; i >= 0; i = (i & (i + 1)) - 1)
      for (int j = y; j >= 0; j = (j & (j + 1)) - 1)
        ret += bit[i][j];
    return ret;
  }

  void add(int x, int y, int delta) {
    for (int i = x; i < n; i = i | (i + 1))
      for (int j = y; j < m; j = j | (j + 1))
        bit[i][j] += delta;
  }
};
```

1.25 Floyd Warshall

```
const int N = 100, inf = 1e9 + 9;
int d[N][N], nextof[N][N];
int n;
void init() {
  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= n; ++j) {
      nextof[i][j] = j;
      d[i][j] = inf;
      if (i == j) d[i][j] = 0;
    }
  }
}
void cal() {
  for (int k = 1; k <= n; ++k) {
    for (int i = 1; i <= n; ++i) {
      for (int j = 1; j <= n; ++j) {
        if (d[i][k] + d[k][j] < d[i][j]) {
          d[i][j] = d[i][k] + d[k][j];
          nextof[i][j] = nextof[i][k];
        }
      }
    }
  }
}
vector<int> findPath(int i, int j) {
  vector<int> path = {i};
  while(i != j) {
    i = nextof[i][j];
    path.push_back(i);
  }
  return path;
}
```

1.26 GP Hash Table

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
const int RANDOM = chrono::high_resolution_clock::now(
 ↪  ).time_since_epoch().count();
struct custom_hash {
  int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<int, int, custom_hash> mp;
```

1.27 Geometric Sum

```
ll geometric_Sum() {
  ll a, r, n; cin >> a >> r >> n; //a = first value r =
 ↪  ratio, n = n-term
  ll res = BigMod(r, n);
  ll numara = (a * (1 - res)) % MOD;
  numara = (numara + MOD) % MOD;
  ll deno = ((1 - r) % MOD + MOD) % MOD;
  ll ans = (numara * BigMod(deno, MOD - 2)) % MOD;
  return ans;
}
// geometric sum for any MOD
const int mod = 1e9 + 7;
int geo_all(int n, int x) { //1 + x + x^2 + x^3 ....x^n
 ↪  = f(n, x)
  if (n == 0) return 0;
  int ret = 1ll * (1 + x) * geo_all(n / 2, 1ll * x * x
 ↪  % mod) % mod;
  if (n & 1) ret = (1 + 1ll * x * ret) % mod;
  return ret;
}
```

1.28 Heavy-Light Decomposition

```
vector<int> parent, depth, heavy, head, pos;
int cur_pos;
int dfs(int v, vector<vector<int>> const& adj) {
    int size = 1;
    int max_c_size = 0;
    for (int c : adj[v]) {
        if (c != parent[v]) {
            parent[c] = v, depth[c] = depth[v] + 1;
            int c_size = dfs(c, adj);
            size += c_size;
            if (c_size > max_c_size)
                max_c_size = c_size, heavy[v] = c;
        }
    }
    return size;
}
void decompose(int v, int h, vector<vector<int>>
 ↪  const& adj) {
    head[v] = h, pos[v] = cur_pos++;
    if (heavy[v] != -1)
        decompose(heavy[v], h, adj);
    for (int c : adj[v]) {
        if (c != parent[v] && c != heavy[v])
            decompose(c, c, adj);
    }
}
void init(vector<vector<int>> const& adj) {
```

```
    int n = adj.size();
    parent = vector<int>(n);
    depth = vector<int>(n);
    heavy = vector<int>(n, -1);
    head = vector<int>(n);
    pos = vector<int>(n);
    cur_pos = 0;
    dfs(0, adj);
    decompose(0, 0, adj);
}

int query(int a, int b) {
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]]) {
        if (depth[head[a]] > depth[head[b]])
            swap(a, b);
        int cur_heavy_path_max =
         ↪  segment_tree_query(pos[head[b]], pos[b]);
        res = max(res, cur_heavy_path_max);
    }
    if (depth[a] > depth[b])
        swap(a, b);
    int last_heavy_path_max =
     ↪  segment_tree_query(pos[a], pos[b]);
    res = max(res, last_heavy_path_max);
    return res;
}
```

1.29 HopcroftKarp

```
struct HopcroftKarp {
  const int INF = 1e9 + 7;
  vector <vector<int>> g;
  vector<int> match, dist;
  int nodes;
  void init(int _nodes) {
    nodes = _nodes;
    g.resize(_nodes);
    match.resize(_nodes);
    dist.resize(_nodes);
  }
  void add_edge(int u, int v) {
    g[u].push_back(v);
  }
  bool bfs(int n) {
    queue <int> q;
    for (int i = 1; i <= n; ++i) {
      if (!match[i]) dist[i] = 0, q.emplace(i);
      else dist[i] = INF;
    }
    dist[0] = INF;
    while (!q.empty()) {
      int u = q.front(); q.pop();
      if (!u) continue;
      for (int v : g[u]) {
        if (dist[match[v]] == INF) {
          dist[match[v]] = dist[u] + 1,
           ↪  q.emplace(match[v]);
        }
      }
    }
    return dist[0] != INF;
  }
  bool dfs (int u) {
    if (!u) return 1;
    for (int v : g[u]) {
      if (dist[match[v]] == dist[u] + 1 and
       ↪  dfs(match[v])) {
        match[u] = v, match[v] = u;
        return 1;
      }
    }
    dist[u] = INF;
    return 0;
  }
  int hopcroftKarp() {
    int n = nodes - 1;
    int ret = 0;
    while (bfs(n)) {
      for (int i = 1; i <= n; ++i) {
        ret += !match[i] and dfs(i);
      }
    }
    return ret;
  }
};
```

1.30 Interval Set

```
struct interval_set {
  set<array<ll, 2>> st;
  ll cnt = 0, d = 0;
  void init(ll _d) {
    d = _d;
  }
  void add(ll l, ll r, ll x) {
    cnt += x * (1 + (r - l) / d);
  }
  void insert(ll l, ll r) {
    auto it = st.lower_bound({l, INF});
    if (it != st.begin()) {
      it--;
      if (l <= (*it)[1]) {
        l = (*it)[0];
        r = max(r, (*it)[1]);
        add((*it)[0], (*it)[1], -1);
        st.erase(it);
      }
    }
    while (1) {
      auto it = st.lower_bound({l, -INF});
      if (it == st.end() || r < (*it)[0]) break;
      r = max(r, (*it)[1]);
      add((*it)[0], (*it)[1], -1);
      st.erase(it);
    }
    add(l, r, 1);
    st.insert({l, r});
  }
  ll count() {
    return cnt;
  }
};
```

1.31 KMP

```
// Longest Proper Prefix which is also a Suffix
vector<int> prefix_function(string &s) {
  int n = (int)s.length();
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    int j = pi[i - 1];
    while (j > 0 && s[i] != s[j])
      j = pi[j - 1];
    if (s[i] == s[j])
      j++;
    pi[i] = j;
  }
  return pi;
}
```

1.32 LCA

```
const int N = 1e6 + 9, LOG = 21;
int up[N][LOG], depth[N];
vector<int> children[N];
void dfs(int a) {
  for (auto b: children[a]) {
    depth[b] = depth[a] + 1;
    up[b][0] = a;  // a is parent of b
    for (int i = 1; i < LOG; ++i) {
      up[b][i] = up[up[b][i-1]][i-1];
    }
    dfs(b);
  }
}
int getKthAncestor(int node, int k) {
  if (depth[node] < k) return -1;
  for (int i = 0; i < LOG; ++i) {
    if (k & (1 << i)) {
      node = up[node][i];
    }
  }
  return node;
}
int getLCA(int u, int v) {
  if (depth[u] < depth[v]) swap(u, v);
  u = getKthAncestor(u, depth[u] - depth[v]);
  if (u == v) return v;
  for (int i = LOG - 1; i >= 0; --i) {
    if (up[u][i] != up[v][i]) {
      u = up[u][i];
      v = up[v][i];
    }
  }
  return up[v][0];
}
```

1.33 Linear Diophantine Equation

```
int gcd(int a, int b, int& x, int& y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  int x1, y1;
  int d = gcd(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
bool find_any_solution(int a, int b, int c, int &x0,
 ↪  int &y0, int &g) {
  g = gcd(abs(a), abs(b), x0, y0);
  if (c % g) {
    return false;
  }
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
void shift_solution(int & x, int & y, int a, int b,
 ↪  int cnt) {
  x += cnt * b;
  y -= cnt * a;
}
int find_all_solutions(int a, int b, int c, int minx,
 ↪  int maxx, int miny, int maxy) {
  int x, y, g;
  if (!find_any_solution(a, b, c, x, y, g))
    return 0;
  a /= g;
  b /= g;
```

```
  int sign_a = a > 0 ? +1 : -1;
  int sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x) / b);
  if (x < minx)
    shift_solution(x, y, a, b, sign_b);
  if (x > maxx)
    return 0;
  int lx1 = x;
  shift_solution(x, y, a, b, (maxx - x) / b);
  if (x > maxx)
    shift_solution(x, y, a, b, -sign_b);
  int rx1 = x;
  shift_solution(x, y, a, b, -(miny - y) / a);
  if (y < miny)
    shift_solution(x, y, a, b, -sign_a);
  if (y > maxy)
    return 0;
  int lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y) / a);
  if (y > maxy)
    shift_solution(x, y, a, b, sign_a);
  int rx2 = x;
  if (lx2 > rx2)
    swap(lx2, rx2);
  int lx = max(lx1, lx2);
  int rx = min(rx1, rx2);
  if (lx > rx)
    return 0;
  return (rx - lx) / abs(b) + 1;
}
```

1.34 Manacher

Description: pal[1][i] = longest odd (half rounded down) palindrome around pos i and starts at i - pal[1][i] and ends at i + pal[1][i]
pal[0][i] = half length of longest even palindrome around pos i, i + 1 and starts at i - par[0][i] + 1 and ends at i + pal[0][i]

```
int pal[2][N];
void manacher(string &s) {
  int n = s.size(), idx = 2;
  while (idx--) {
    for (int l=-1, r=-1, i=0; i<n-1; ++i){
      if (i > r) l = r = i;
      else {
        int k = min(r-i, pal[idx][l+r-i]);
        l = i - k,  r = i + k;
      }
      while (l - idx >= 0 and r + 1 < n and s[l - idx]
      ↪  == s[r + 1]) l--, r++;
      pal[idx][i] = r - i;
      // [l - 1 + idx : r] palindrome
    }
  }
}
```

1.35 Matrix Exponentiation

```
const int mod = 1e9 + 7;
struct Mat {
  int sz;
  vector<vector<int>> val;
  Mat(int sz) {
    this->sz = sz;
    val.resize(sz, vector<int>(sz, 0));
  }
  Mat(int sz, int v) {
    this->sz = sz;
    val.resize(sz, vector<int>(sz, 0));
    for (int i = 0; i < sz; ++i) {
      val[i][i] = v; // diagonal values
    }
  }
  Mat operator * (Mat m2) {
    Mat ans(sz);
    for (int i = 0; i < sz; ++i) {
      for (int j = 0; j < sz; ++j) {
        for (int k = 0; k < sz; ++k) {
          ans.val[i][j] = (ans.val[i][j] + (1LL *
          ↪  val[i][k] * m2.val[k][j]) % mod) % mod;
        }
      }
    }
    return ans;
  }
};
Mat Mat_Expo(Mat a, long long n) {
  Mat ans(a.sz, 1);  // identity matrix
  while (n) {
    if (n & 1) {
      ans = ans * a;
    }
    a = a * a;
    n >>= 1;
  }
  return ans;
}
```

1.36 Mint

```
struct Mint {
  int v;
  Mint(long long val = 0) {
    v = int(val % MOD);
    if (v < 0) v += MOD;
  }
  Mint operator+(const Mint &o) const { return Mint(v
  ↪  + o.v); }
  Mint operator-(const Mint &o) const { return Mint(v
  ↪  - o.v); }
  Mint operator*(const Mint &o) const { return
  ↪  Mint(1LL * v * o.v); }
  Mint operator/(const Mint &o) const { return *this *
  ↪  o.inv(); }
  Mint& operator+=(const Mint &o) { v += o.v; if (v >=
  ↪  MOD) v -= MOD; return *this; }
  Mint& operator-=(const Mint &o) { v -= o.v; if (v <
  ↪  0) v += MOD; return *this; }
  Mint& operator*=(const Mint &o) { v = int(1LL * v *
  ↪  o.v % MOD); return *this; }
  Mint pow(long long p) const {
    Mint a = *this, res = 1;
    while (p > 0) {
      if (p & 1) res *= a;
      a *= a;
      p >>= 1;
    }
    return res;
  }
  Mint inv() const { return pow(MOD - 2); }
  friend ostream& operator<<(ostream& os, const Mint&
  ↪  m) {
    os << m.v;
    return os;
  }
};
```

1.37 Mobius Function

```
const int N = 1E6 + 5;
int mu[N];
void pre() {
  mu[1] = 1;
  for (int i = 1; i < N; ++i) {
    for (int j = i + i; j < N; j += i) {
      mu[j] -= mu[i];
    }
  }
}
```

1.38 N-th Permutation

```
vector<ll> fact(21, 1);
//does not handle if given ff-th permutation does not
↪  exist
string n_th_Permutation(string s, ll ff){
  int n = s.size();
  for(int i=0; i<n; i++){
    sort(s.begin()+i, s.end());
    int pos = i+ff/fact[n-1-i];
    ff %= fact[n-1-i];
    swap(s[i], s[pos]);
  }
  return s;
}
```

1.39 PBDS

Description: *x.find_by_order(k) : iterator to the k-th index
x.order_of_key(k) : number of items smaller than k

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

1.40 Polynomial Interpolation

```
// P(x) = a0 + a1x + a2x^2 + ... + anx^n
// y[i] = P(i)
const int mod = 1e9 + 7;
ll BigMod(ll a, ll b) {
   ll res = 1;
   while (b) {
      if (b & 1) res = 1ll * res * a % mod;
      a = 1ll * a * a % mod;
      b >>= 1;
   }
   return res;
}
ll inv(ll x) {
   if (x < 0) x += mod;
   return BigMod(x, mod - 2);
}
ll add(ll &a, ll b) {
   a += b;
   if (a >= mod) a -= mod;
   return a;
}
ll eval (vector<ll> y, ll k) {
   int n = y.size() - 1;
   if (k <= n) {
      return y[k];
   }
   vector<ll> L(n + 1, 1);
   for (int x = 1; x <= n; ++x) {
      L[0] = L[0] * (k - x) % mod;
      L[0] = L[0] * inv(-x) % mod;
   }
   for (int x = 1; x <= n; ++x) {
      L[x] = L[x - 1] * inv(k - x) % mod * (k - (x -
      ↪  1)) % mod;
      L[x] = L[x] * ((x - 1) - n + mod) % mod * inv(x)
      ↪  % mod;
   }
   ll yk = 0;
   for (int x = 0; x <= n; ++x) {
      yk = add(yk, L[x] * y[x] % mod);
   }
   return yk;
}
```

1.41 Prefix Sum 3D

```
pref[x][y][z] = pref[x - 1][y][z] + pref[x][y - 1][z]
    + pref[x][y][z - 1] - pref[x - 1][y - 1][z] -
    pref[x - 1][y][z - 1] - pref[x][y - 1][z - 1] +
    pref[x - 1][y - 1][z - 1] + arr[x][y][z];
// from x1 to x2, y1 to y2, z1 to z2
ans = pref[x2][y2][z2] - pref[x1 - 1][y2][z2] -
    pref[x2][y1 - 1][z2] - pref[x2][y2][z1 - 1] +
    pref[x1 - 1][y1 - 1][z] + pref[x1 - 1][y][z1 - 1]
    + pref[x2][y1 - 1][z1 - 1] - pref[x1 - 1][y1 -
    1][z1 - 1];
```

1.42 Segment Tree(Persistent)

```
//the code returns k-th number in a range l to r if the
    range were sorted
struct PST {
#define lc t[cur].l
#define rc t[cur].r
  struct node {
    int l = 0, r = 0, val = 0;
  } t[20 * N];
  int T = 0;
  int build(int b, int e) {
    int cur = ++T;
    if(b == e) return cur;
    int mid = b + e >> 1;
    lc = build(b, mid);
    rc = build(mid + 1, e);
    t[cur].val = t[lc].val + t[rc].val;
    return cur;
  }
  int upd(int pre, int b, int e, int i, int v) {
    int cur = ++T;
    t[cur] = t[pre];
    if(b == e) {
      t[cur].val += v;
      return cur;
    }
    int mid = b + e >> 1;
    if(i <= mid) {
      rc = t[pre].r;
      lc = upd(t[pre].l, b, mid, i, v);
    } else {
      lc = t[pre].l;
      rc = upd(t[pre].r, mid + 1, e, i, v);
    }
    t[cur].val = t[lc].val + t[rc].val;
    return cur;
  }
  int query(int pre, int cur, int b, int e, int k) {
    if(b == e) return b;
    int cnt = t[lc].val - t[t[pre].l].val;
    int mid = b + e >> 1;
    if(cnt >= k) return query(t[pre].l, lc, b, mid, k);
    else return query(t[pre].r, rc, mid + 1, e, k -
      cnt);
  }
} t;
int V[N], root[N], a[N];
int32_t main() {
  map<int, int>mp;
  int n, q;
  cin >> n >> q;
  for(int i = 1; i <= n; i++) cin >> a[i], mp[a[i]];
  int c = 0;
  for(auto x : mp) mp[x.first] = ++c, V[c] = x.first;
  root[0] = t.build(1, n);
  for(int i = 1; i <= n; i++) {
    root[i] = t.upd(root[i - 1], 1, n, mp[a[i]], 1);
  }
  while(q--) {
    int l, r, k;
    cin >> l >> r >> k;
    cout << V[t.query(root[l - 1], root[r], 1, n, k)]
      << '\n';
  }
  return 0;
}
```

1.43 Segment Tree(Special Variant)

```
// Range increment and decrement (increment before
    decrement) and number of positive elements in the
    whole array
const int N = 2e6+6;
int st[4 * N], cnt[4 * N];
void add(int l, int r, ll x, int u = 1, int s = 0, int
    e = N - 1) {
  if (s > r or e < l)  return ;
  int v = u << 1, w = v | 1, m = s + e >> 1;
  if (l <= s and e <= r) {
    cnt[u] += x;
    if (cnt[u])  st[u] = e - s + 1;
    else  st[u] = (s == e)? 0: st[v] + st[w];
    return ;
  }
  add(l, r, x, v, s, m);
  add(l, r, x, w, m + 1, e);
  if (!cnt[u])  st[u] = st[v] + st[w];
}
```

1.44 Segment Tree

```
struct ST {
  #define lc (n << 1)
  #define rc ((n << 1) + 1)
  long long t[4 * N], lazy[4 * N];
  ST() {
    memset(t, 0, sizeof t);
    memset(lazy, 0, sizeof lazy);
  }
  inline void push(int n, int b, int e) { // change
    this
    if (lazy[n] == 0) return;
    t[n] = t[n] + lazy[n] * (e - b + 1);
    if (b != e) {
      lazy[lc] = lazy[lc] + lazy[n];
      lazy[rc] = lazy[rc] + lazy[n];
    }
    lazy[n] = 0;
  }
  inline long long combine(long long a,long long b) {
    // change this
    return a + b;
  }
  inline void pull(int n) { // change this
    t[n] = t[lc] + t[rc];
  }
  void build(int n, int b, int e) {
    lazy[n] = 0; // change this
    if (b == e) {
      t[n] = a[b];
      return;
    }
    int mid = (b + e) >> 1;
    build(lc, b, mid);
    build(rc, mid + 1, e);
    pull(n);
  }
  void upd(int n, int b, int e, int i, int j, long
    long v) {
    push(n, b, e);
    if (j < b || e < i) return;
    if (i <= b && e <= j) {
      lazy[n] = v; //set lazy
      push(n, b, e);
      return;
    }
    int mid = (b + e) >> 1;
    upd(lc, b, mid, i, j, v);
    upd(rc, mid + 1, e, i, j, v);
    pull(n);
  }
  long long query(int n, int b, int e, int i, int j) {
    push(n, b, e);
    if (i > e || b > j) return 0; //return null
    if (i <= b && e <= j) return t[n];
    int mid = (b + e) >> 1;
    return combine(query(lc, b, mid, i, j), query(rc,
      mid + 1, e, i, j));
  }
}t;
```

1.45 Seive upto 1e9

```
// credit: min_25
// takes 0.5s for n = 1e9
vector<int> sieve(const int N, const int Q = 17, const
  int L = 1 << 15) {
  static const int rs[] = {1, 7, 11, 13, 17, 19, 23,
    29};
  struct P {
    P(int p) : p(p) {}
    int p; int pos[8];
  };
  auto approx_prime_count = [] (const int N) -> int {
    return N > 60184 ? N / (log(N) - 1.1) : max(1., N
      / (log(N) - 1.11)) + 1;
  };
  const int v = sqrt(N), vv = sqrt(v);
  vector<bool> isp(v + 1, true);
  for (int i = 2; i <= vv; ++i) if (isp[i]) {
    for (int j = i * i; j <= v; j += i) isp[j] = false;
  }
  const int rsize = approx_prime_count(N + 30);
  vector<int> primes = {2, 3, 5}; int psize = 3;
  primes.resize(rsize);
  vector<P> sprimes; size_t pbeg = 0;
  int prod = 1;
  for (int p = 7; p <= v; ++p) {
    if (!isp[p]) continue;
    if (p <= Q) prod *= p, ++pbeg, primes[psize++] = p;
    auto pp = P(p);
    for (int t = 0; t < 8; ++t) {
      int j = (p <= Q) ? p : p * p;
      while (j % 30 != rs[t]) j += p << 1;
      pp.pos[t] = j / 30;
    }
    sprimes.push_back(pp);
  }
  vector<unsigned char> pre(prod, 0xFF);
  for (size_t pi = 0; pi < pbeg; ++pi) {
    auto pp = sprimes[pi]; const int p = pp.p;
    for (int t = 0; t < 8; ++t) {
      const unsigned char m = ~(1 << t);
      for (int i = pp.pos[t]; i < prod; i += p) pre[i]
        &= m;
    }
  }
  const int block_size = (L + prod - 1) / prod * prod;
  vector<unsigned char> block(block_size); unsigned
    char* pblock = block.data();
  const int M = (N + 29) / 30;
  for (int beg = 0; beg < M; beg += block_size, pblock
    -= block_size) {
```

```
    int end = min(M, beg + block_size);
    for (int i = beg; i < end; i += prod) {
      copy(pre.begin(), pre.end(), pblock + i);
    }
    if (beg == 0) pblock[0] &= 0xFE;
    for (size_t pi = pbeg; pi < sprimes.size(); ++pi) {
      auto& pp = sprimes[pi];
      const int p = pp.p;
      for (int t = 0; t < 8; ++t) {
        int i = pp.pos[t]; const unsigned char m = ~(1
        ↪ << t);
        for (; i < end; i += p) pblock[i] &= m;
        pp.pos[t] = i;
      }
    }
    for (int i = beg; i < end; ++i) {
      for (int m = pblock[i]; m > 0; m &= m - 1) {
        primes[psize++] = i * 30 +
        ↪ rs[__builtin_ctz(m)];
      }
    }
  }
  assert(psize <= rsize);
  while (psize > 0 && primes[psize - 1] > N) --psize;
  primes.resize(psize);
  return primes;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, a, b; cin >> n >> a >> b;
  auto primes = sieve(n);
  vector<int> ans;
  for (int i = b; i < primes.size() && primes[i] <= n;
  ↪ i += a) ans.push_back(primes[i]);
  cout << primes.size() << ' ' << ans.size() << '\n';
  for (auto x: ans) cout << x << ' '; cout << '\n';
  return 0;
}
```

1.46 Seive(Linear)

```
const int N = 10000000;
vector<int> spf(N+1);
vector<int> pr;
for (int i=2; i <= N; ++i) {
    if (spf[i] == 0) {
        spf[i] = i;
        pr.push_back(i);
    }
    for (int j = 0; i * pr[j] <= N; ++j) {
        spf[i * pr[j]] = pr[j];
        if (pr[j] == spf[i]) {
            break;
        }
    }
}
```

1.47 Seive(Segmented)

```
vector<char> segmentedSieve(long long L, long long R) {
    // generate all primes up to sqrt(R)
    long long lim = sqrt(R);
    vector<char> mark(lim + 1, false);
    vector<long long> primes;
    for (long long i = 2; i <= lim; ++i) {
        if (!mark[i]) {
            primes.emplace_back(i);
            for (long long j = i * i; j <= lim; j += i)
                mark[j] = true;
        }
    }
    vector<char> isPrime(R - L + 1, true);
    for (long long i : primes)
        for (long long j = max(i * i, (L + i - 1) / i
        ↪ * i); j <= R; j += i)
            isPrime[j - L] = false;
    if (L == 1)
        isPrime[0] = false;
    return isPrime;
}
int count_primes(int n) {
    const int S = 10000;
    vector<int> primes;
    int nsqrt = sqrt(n);
    vector<char> is_prime(nsqrt + 2, true);
    for (int i = 2; i <= nsqrt; i++) {
        if (is_prime[i]) {
            primes.push_back(i);
            for (int j = i * i; j <= nsqrt; j += i)
                is_prime[j] = false;
        }
    }
    int result = 0;
    vector<char> block(S);
    for (int k = 0; k * S <= n; k++) {
        fill(block.begin(), block.end(), true);
        int start = k * S;
        for (int p : primes) {
            int start_idx = (start + p - 1) / p;
            int j = max(start_idx, p) * p - start;
            for (; j < S; j += p)
                block[j] = false;
        }
        if (k == 0)
            block[0] = block[1] = false;
        for (int i = 0; i < S && start + i <= n; i++) {
            if (block[i])
                result++;
        }
    }
    return result;
}
```

1.48 Seive

```
const int N = 1e6 + 3;
bitset<N> isPrime;
vector<int> prime;
void seive() {
  isPrime[2] = 1;
  for (int i = 3; i <= N; i+=2) {
    isPrime[i] = 1;
  }
  for (int i = 3; i * i <= N; i += 2) {
    if(isPrime[i]) {
      for (int j = i * i; j <= N; j += (i + i)) {
        isPrime[j] = 0;
      }
    }
  }
  prime.push_back(2);
  for (int i = 3; i <= N; i+=2) {
    if(isPrime[i]) {
      prime.push_back(i);
    }
  }
}
```

1.49 Sparse Table

```
const int N = 2e5 + 3, M = __bit_width(N) + 1;
int maxTable[N][M], a[N];
void buildTable(int n) {
  for (int i = 0; i < n; ++i) {
    maxTable[i][0] = a[i];
  }
  for (int k = 1; k < M; ++k) {
    for (int i = 0; i + (1 << k) <= n; ++i) {
      maxTable[i][k] = max(maxTable[i][k - 1],
      ↪ maxTable[i + (1 << (k - 1))][k - 1]);
    }
  }
}
int maxQuery(int i, int j, int n) {
  if (j < 0 or i >= n) return INT32_MIN;
  int k = __bit_width(j - i + 1) - 1;
  return max(maxTable[i][k], maxTable[j - (1 << k) +
  ↪ 1][k]);
}
```

1.50 SquareRoot Decomposition

```
#include <bits/stdc++.h>
using namespace std;
struct Sqrt {
  int block_size;
  vector<int> nums;
  vector<long long> blocks;
  Sqrt(int sqrtn, vector<int> &arr) :
  ↪ block_size(sqrtn), blocks(sqrtn, 0) {
    nums = arr;
    for (int i = 0; i < nums.size(); i++) { blocks[i /
    ↪ block_size] += nums[i]; }
  }
  void update(int x, int v) {
    blocks[x / block_size] -= nums[x];
    nums[x] = v;
    blocks[x / block_size] += nums[x];
  }
  long long query(int r) {
    long long res = 0;
    for (int i = 0; i < r / block_size; i++) { res +=
    ↪ blocks[i]; }
    for (int i = (r / block_size) * block_size; i < r;
    ↪ i++) { res += nums[i]; }
    return res;
  }
  long long query(int l, int r) { return query(r) -
  ↪ query(l - 1); }
};
int main() {
  int n, q;
  cin >> n >> q;
  vector<int> arr(n);
  for (int i = 0; i < n; i++) { cin >> arr[i]; }
  Sqrt sq((int)ceil(sqrt(n)), arr);
  for (int i = 0; i < q; i++) {
    int t, l, r;
    cin >> t >> l >> r;
    if (t == 1) {
      sq.update(l - 1, r);
    } else {
      cout << sq.query(l, r) << "\n";
    }
  }
}
```

```
#include <bits/stdc++.h>
using namespace std;
struct Query {
  int l, r, idx;
};
```

```cpp
int main() {
  int n;
  cin >> n;
  vector<int> v(n);
  for (int i = 0; i < n; i++) { cin >> v[i]; }
  int q;
  cin >> q;
  vector<Query> queries;
  for (int i = 0; i < q; i++) {
    int x, y;
    cin >> x >> y;
    queries.push_back({--x, --y, i});
  }
  int block_size = (int)sqrt(n);
  auto mo_cmp = [&](Query a, Query b) {
    int block_a = a.l / block_size;
    int block_b = b.l / block_size;
    if (block_a == block_b) { return a.r < b.r; }
    return block_a < block_b;
  };
  sort(queries.begin(), queries.end(), mo_cmp);
  int different_values = 0;
  vector<int> values(VALMAX);
  auto remove = [&](int idx) {
    values[v[idx]]--;
    if (values[v[idx]] == 0) { different_values--; }
  };
  auto add = [&](int idx) {
    values[v[idx]]++;
    if (values[v[idx]] == 1) { different_values++; }
  };
  int mo_left = -1;
  int mo_right = -1;
  vector<int> ans(q);
  for (int i = 0; i < q; i++) {
    int left = queries[i].l;
    int right = queries[i].r;
    while (mo_left < left) { remove(mo_left++); }
    while (mo_left > left) { add(--mo_left); }
    while (mo_right < right) { add(++mo_right); }
    while (mo_right > right) { remove(mo_right--); }
    ans[queries[i].idx] = different_values;
  }
  for (int i = 0; i < q; i++) { cout << ans[i] << '\n'; }
}
```

1.51 String Hashing

```cpp
const int p1 = 137, mod1 = 127657753, p2 = 277, mod2 = 987654319; // 911382323, 972663749
const int N = 1e6 + 3;
array<int, 2> pref[N], rev[N];
int pw1[N], pw2[N], ipw1[N], ipw2[N];
int power(int a, int n, int mod) {
  int ans = 1 % mod;
  while (n) {
    if (n & 1) ans = 1LL * ans * a % mod;
    a = 1LL * a * a % mod;
    n >>= 1;
  }
  return ans;
}
void prec() {
  pw1[0] = pw2[0] = ipw1[0] = ipw2[0] = 1;
  int ip1 = power(p1, mod1 - 2, mod1);
  int ip2 = power(p2, mod2 - 2, mod2);
  for (int i = 1; i < N; ++i) {
    pw1[i] = 1LL * pw1[i - 1] * p1 % mod1;
    pw2[i] = 1LL * pw2[i - 1] * p2 % mod2;
    ipw1[i] = 1LL * ipw1[i - 1] * ip1 % mod1;
    ipw2[i] = 1LL * ipw2[i - 1] * ip2 % mod2;
  }
}
void build(string& s) {
  int n = s.size();
  for (int i = 0; i < n; ++i) {
    pref[i][0] = 1LL * s[i] * pw1[i] % mod1;
    if (i) pref[i][0] = (pref[i][0] + pref[i - 1][0]) % mod1;
    pref[i][1] = 1LL * s[i] * pw2[i] % mod2;
    if (i) pref[i][1] = (pref[i][1] + pref[i - 1][1]) % mod2;
    rev[i][0] = 1LL * s[i] * ipw1[i] % mod1;
    if (i) rev[i][0] = (rev[i][0] + rev[i - 1][0]) % mod1;
    rev[i][1] = 1LL * s[i] * ipw2[i] % mod2;
    if (i) rev[i][1] = (rev[i][1] + rev[i - 1][1]) % mod2;
  }
}
array<int, 2> get_hash(int i, int j) {
  array<int, 2> ans = {0, 0};
  ans[0] = pref[j][0];
  if (i) ans[0] = (pref[j][0] - pref[i - 1][0] + mod1) % mod1;
  ans[1] = pref[j][1];
  if (i) ans[1] = (pref[j][1] - pref[i - 1][1] + mod2) % mod2;
  ans[0] = 1LL * ans[0] * ipw1[i] % mod1;
  ans[1] = 1LL * ans[1] * ipw2[i] % mod2;
  return ans;
}
array<int, 2> get_rev_hash(int i, int j) {
  array<int, 2> ans = {0, 0};
  ans[0] = rev[j][0];
  if (i) ans[0] = (rev[j][0] - rev[i - 1][0] + mod1) % mod1;
  ans[1] = rev[j][1];
  if (i) ans[1] = (rev[j][1] - rev[i - 1][1] + mod2) % mod2;
  ans[0] = 1LL * ans[0] * pw1[j] % mod1;
  ans[1] = 1LL * ans[1] * pw2[j] % mod2;
  return ans;
}
```

1.52 Strongly_Connected_Components(SCC)

```cpp
const int N = 1e5 + 9;
int vis[N], id[N];
vector<int> adj[N], adj_t[N];
vector<int> order;
void dfs1(int v) {
  vis[v] = 1;
  for (int u: adj[v]) {
    if (!vis[u]) {
      dfs1(u);
    }
  }
  order.push_back(v);
}
void dfs2(int v, int cnt) {
  id[v] = cnt;
  for (int u: adj_t[v]) {
    if (!id[u]) {
      dfs2(u, cnt);
    }
  }
}
void solve() {
  int n, m;
  cin >> n >> m;
  for (int i = 0; i < m; ++i) {
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v);
    adj_t[v].push_back(u);
  }
  for (int i = 1; i <= n; ++i) {
    if (!vis[i]) {
      dfs1(i);
    }
  }
  reverse(order.begin(), order.end());
  int cnt = 1;
  for (auto v: order) {
    if (!id[v]) {
      dfs2(v, cnt++);
    }
  }
}
```

1.53 Submask Enumeration

```cpp
// Generate all submask of m
for (int s = m; ; s = (s-1) & m) {
 // you can use s ...
 if (s == 0)  break;
}
```

1.54 Suffix Array

Description: This funtion return two vectors (first vector is sorted suffix array position , second vector is longest common prefix with previous string)

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

```
  else x = len;
  if (x >= len) {
    if (len1 == len2) return 0;
    if (len1 < len2) return -1;
    return 1;
  }
  if (s[l1 + x] < s[l2 + x]) return -1;
  return 1;
};
## Kth Unique Substring
auto kth = [&] (ll k) {
  int i = 0;
  while (i + 1 < n and k > n - sa[i] - lcp[i]) {
    k -= n - sa[i] - lcp[i];
    i++;
  }
  k = min(k, 0ll + n - sa[i] - lcp[i]);
  array<int, 2> ret = {sa[i], k + lcp[i]};
  return ret;
};
## Several Consecutive Identical Substrings
for (int i = 1; i < n; ++i) {
  for (int j = i; j < n; j += i) {
    // Block = [j-i...j-1]
    int e1 = rmq(0, pos[j - i], pos[j]), e2 = 0;
    if (i < j) {
      e2 = rmq(1, rev_pos[j - i - 1], rev_pos[j - 1]);
    }
    int k = (e1 + e2) / i + 1;
    // [j-i-e2 ... j-1+e1] is periodic with period length = i
  }
}
```

1.55 Suffix Automaton

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

1.56 Ternary Search

```
double ternary_search(double l, double r) {
  double eps = 1e-9;      //set the error limit here
  while (r - l > eps) {
    double m1 = l + (r - l) / 3;
    double m2 = r - (r - l) / 3;
    double f1 = f(m1); //value of function at m1
    double f2 = f(m2); //value of function at m2
    if (f1 < f2)
      l = m1;
    else
      r = m2;
  }
  return f(l)      //return the maximum of f(x) in [l, r]
}
```

1.57 Topological Sorting

```
const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];
vector<int> ord;
void dfs(int u) {
  vis[u] = true;
  for (auto v: g[u]) {
    if (!vis[v]) {
      dfs(v);
    }
  }
  ord.push_back(u);
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m; cin >> n >> m;
  while (m--) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
  }
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
      dfs(i);
    }
  }
  reverse(ord.begin(), ord.end());
  // check if feasible
  vector<int> pos(n + 1);
  for (int i = 0; i < (int) ord.size(); i++) {
    pos[ord[i]] = i;
  }
  for (int u = 1; u <= n; u++) {
    for (auto v: g[u]) {
      if (pos[u] > pos[v]) {
        cout << "IMPOSSIBLE\n";
        return 0;
      }
    }
  }
  // print the order
  for (auto u: ord) cout << u << ' ';
  cout << '\n';
  return 0;
}
```

1.58 Trie

```
const int N = 1e6 + 3;
int nextof[N][26], cnt[N];
int tot = 1;
void add(string& s) {
  int u = 1;
  ++cnt[u];
  for (auto c: s) {
    int v = c - 'a';
    if (!nextof[u][v]) {
      nextof[u][v] = ++tot;
```

```
    }
    u = nextof[u][v];
    ++cnt[u];
  }
}
int countPref(string& s) {
  int u = 1;
  for (auto c: s) {
    int v = c - 'a';
    if (!nextof[u][v]) return 0;
    u = nextof[u][v];
  }
  return cnt[u];
}
```

1.59 Z_algo

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

1.60 int128

```
istream& operator >>(istream& cin, __int128& x) {
  string s;
  cin >> s;
  x = 0;
  for (int i = 0; i < s.size(); ++i) {
    x = x * 10 + (s[i] - '0');
  }
  return cin;
}
ostream& operator <<(ostream& cout, __int128 x) {
  if (x == 0) {
    cout << 0;
    return cout;
  }
  if (x < 0) {
    cout << "-";
    x *= -1;
  }
  string s;
  while (x) {
    s += (x % 10) + '0';
    x /= 10;
  }
  reverse(s.begin(), s.end());
  cout << s;
  return cout;
}
```

1.61 josephus problem

Description: Given natural numbers n and k, the numbers 1 to n are arranged in a circle. Starting from 1, every k-th number is removed in a circular manner. This continues until only one number remains. Find the last remaining number.

```
int josephus(int n, int k) {
  if (n == 1)
    return 0;
  if (k == 1)
    return n-1;
  if (k > n)
    return (josephus(n-1, k) + k) % n;
  int cnt = n / k;
  int res = josephus(n - cnt, k);
  res -= n % k;
  if (res < 0)
    res += n;
  else
    res += res / (k - 1);
  return res;
}
```

1.62 nCr and nPr-1

```
int fact[N], ifact[N];
void prec() {
  fact[0] = 1;
  for (int i = 1; i < N; i++) {
    fact[i] = 1LL * fact[i - 1] * i % mod;
  }
  ifact[N - 1] = power(fact[N - 1], -1);
  for (int i = N - 2; i >= 0; i--) {
    ifact[i] = 1LL * ifact[i + 1] * (i + 1) % mod;
  }
}
int nPr(int n, int r) {
  if (n < r) return 0;
  return 1LL * fact[n] * ifact[n - r] % mod;
}
int nCr(int n, int r) {
  if (n < r) return 0;
  return 1LL * fact[n] * ifact[r] % mod * ifact[n - r]
  ↪ % mod;
}
```

1.63 nCr and nPr-2

```
const int N = 2005, mod = 1e9 + 7;
int C[N][N], fact[N];
void prec() { // O(n^2)
  for (int i = 0; i < N; i++) {
    C[i][0] = C[i][i] = 1;
    for (int j = 1; j < i; j++) {
      C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
    }
  }
  fact[0] = 1;
  for (int i = 1; i < N; i++) {
    fact[i] = 1LL * fact[i - 1] * i % mod;
  }
}
int nCr(int n, int r) { // O(1)
  if (n < r) return 0;
  return C[n][r];
}
int nPr(int n, int r) { // O(1)
  if (n < r) return 0;
  return 1LL * nCr(n, r) * fact[r] % mod;
}
```

1.64 notes

```
Pick's Theorem:
Given a certain lattice polygon with non-zero area.
We denote its area by S, the number of points with
↪ integer coordinates lying strictly inside the
↪ polygon by I and the number of points lying on
↪ polygon sides by B.
Then, the Pick's formula states: S = I + (B / 2) - 1
```

1.65 ntt

```
const int MOD = 1e9 + 7;
const int PRIMITIVE_ROOT = 3;
int mod_exp(int base, int exp, int mod) {
  int result = 1;
  while (exp > 0) {
    if (exp % 2 == 1) {
      result = (1LL * result * base) % mod;
    }
    base = (1LL * base * base) % mod;
    exp /= 2;
  }
  return result;
}
void ntt(vector<int> &a, bool invert) {
  int n = a.size();
  int log_n = log2(n);
  for (int i = 1, j = 0; i < n; i++) {
    int bit = n / 2;
    while (j >= bit) {
      j -= bit;
      bit /= 2;
    }
    j += bit;
    if (i < j) swap(a[i], a[j]);
  }
  for (int len = 2; len <= n; len *= 2) {
    int wlen = mod_exp(PRIMITIVE_ROOT, (MOD - 1) /
    ↪ len, MOD);
    if (invert) wlen = mod_exp(wlen, MOD - 2, MOD);
    for (int i = 0; i < n; i += len) {
      int w = 1;
      for (int j = 0; j < len / 2; j++) {
        int u = a[i + j];
        int v = (1LL * a[i + j + len / 2] * w) % MOD;
        a[i + j] = (u + v) % MOD;
        a[i + j + len / 2] = (u - v + MOD) % MOD;
        w = (1LL * w * wlen) % MOD;
      }
    }
  }
  if (invert) {
    int n_inv = mod_exp(n, MOD - 2, MOD);
    for (int &x : a) {
      x = (1LL * x * n_inv) % MOD;
    }
  }
}
vector<int> multiply(vector<int> const &a, vector<int>
↪ const &b) {
  int n = 1;
  while (n < a.size() + b.size()) n *= 2;
  vector<int> fa(a.begin(), a.end()), fb(b.begin(),
  ↪ b.end());
  fa.resize(n);
  fb.resize(n);
  ntt(fa, false);
  ntt(fb, false);
  for (int i = 0; i < n; i++) {
    fa[i] = (1LL * fa[i] * fb[i]) % MOD;
  }
  ntt(fa, true);
  while (!fa.empty() && fa.back() == 0) fa.pop_back();
  return fa;
}
```

2 Geometry

2.1 Angular Sort

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

2.2 CircleCircleIntersection

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

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

2.3 CircleLineIntersection

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

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

2.4 Closest Pair of Points

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

2.5 ComputeCentroid

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

2.6 ComputeCircleCenter

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

2.7 ComputeLineIntersection

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

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

2.8 ComputeSignedArea

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

```
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";
}
```

2.9 Convex Hull

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

2.10 DistancePointPlane

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

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

2.11 DistancePointSegment

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

2.12 Half Plane Intersection

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

```
struct Halfplane {
  PT p, pq; ld angle;
  Halfplane() {}
  // Two points on line
  Halfplane(const PT& a, const PT& b) : p(a), pq(b -
  ↪  a) {
    angle = atan2l(pq.y, pq.x);
  }
  bool out(const PT& r) {
    return cross(pq, r - p) < -EPS;
  }
  bool operator < (const Halfplane& e) const {
    return angle < e.angle;
  }
  friend PT inter(const Halfplane& s, const Halfplane&
  ↪  t) {
    ld alpha = cross((t.p - s.p), t.pq) / cross(s.pq,
    ↪  t.pq);
    return s.p + (s.pq * alpha);
  }
```

```cpp
};
vector<PT> hp_intersect(vector<Halfplane>& H) {
  PT box[4] = {  // Bounding box in CCW order
    PT(INF, INF),  PT(-INF, INF),
    PT(-INF, -INF), PT(INF, -INF)
  };
  for(int i = 0; i<4; i++) { // Add bounding box
  ↪ half-planes.
    Halfplane aux(box[i], box[(i+1) % 4]);
    H.push_back(aux);
  }
  sort(H.begin(), H.end());
  deque<Halfplane> dq; int len = 0;
  for(int i = 0; i < int(H.size()); i++) {
    while (len > 1 && H[i].out(inter(dq[len-1],
    ↪ dq[len-2]))) {
      dq.pop_back(); --len;
    }
    while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
      dq.pop_front(); --len;
    }
    if (len > 0 && fabsl(cross(H[i].pq, dq[len-1].pq))
    ↪ < EPS) {
      if (dot(H[i].pq, dq[len-1].pq) < 0.0)
        return vector<PT>();
      if (H[i].out(dq[len-1].p)) {
        dq.pop_back(); --len;
      }
      else continue;
    }
    dq.push_back(H[i]); ++len;
  }
  while (len > 2 && dq[0].out(inter(dq[len-1],
  ↪ dq[len-2]))) {
    dq.pop_back(); --len;
  }
  while (len > 2 && dq[len-1].out(inter(dq[0],
  ↪ dq[1]))) {
    dq.pop_front(); --len;
  }
  // Report empty intersection if necessary
  if (len < 3) return vector<PT>();
  // Reconstruct the convex polygon from the remaining
  ↪ half-planes.
  vector<PT> ret(len);
  for(int i = 0; i+1 < len; i++) {
    ret[i] = inter(dq[i], dq[i+1]);
  }
  ret.back() = inter(dq[len-1], dq[0]);
  return ret;
}
```

2.13 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;
}
```

2.14 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;
}
```

2.15 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;
}
```

2.16 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); }
double abs(PT p) { return sqrt(p.x*p.x + p.y*p.y); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  return os << "(" << p.x << "," << p.y << ")";
}
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t),
  ↪ p.x*sin(t)+p.y*cos(t));
}
// angle (range [0, pi]) between two vectors
double angle(PT v, PT w) {
  return acos(clamp(dot(v,w) / abs(v) / abs(w), -1.0,
  ↪ 1.0));
}
```

2.17 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;
}
```

2.18 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);
}
```

2.19 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;
}
```

2.20 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;
}
```

3 Notes

3.1 Geometry

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

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

3.1.3 Spherical coordinates

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

3.1.4 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

3.1.5 Polygon

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

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

3.1.6 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.

3.2 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: $\sum_{k=0}\binom{n-k}{k} = F_{n+1}$
- 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.
- $\binom{n}{m} \equiv \lfloor\frac{n}{p}\rfloor (\mod p)$

3.3 Fibonacci Number

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

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

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

6.$\gcd(F_m, F_n) = F_{\gcd(m,n)}$ **7.**$\sum_{0\le k\le n}\left(\binom{n-k}{k}\right) = F_{n+1}$

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

3.4 Sums

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

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

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

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

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

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

3.5 Series

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

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

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

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

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

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

Generating Function

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

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

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

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

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

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

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

3.6 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.

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

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

3.7.2 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 $n < 1e19$.

3.7.3 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.

3.7.4 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 $n-1$.

3.7.5 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, $\sum_{i=1} nlcm(i,n) = \frac{n}{2}(\sum_{d|n}(d\phi(d)) + 1)$
- $\phi(1) = 1$ and $\phi(2) = 1$ which two are only odd ϕ
- $\phi(3) = 2$ and $\phi(4) = 2$ and $\phi(6) = 2$ which three are only prime ϕ
- Find minimum n such that $\frac{\phi(n)}{n}$ is maximum- Multiple of small primes- $2*3*5*7*11*13*...$

3.7.6 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 $\sum_{d|n} \mu(d) f(d) = \prod_{p|n}(1 - f(p))$, $\sum_{d|n} \mu(d)^2 f(d) = \prod_{p|n}(1 + f(p))$.

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

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

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

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

3.7.8 Jacobi symbol

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

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

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

3.7.10 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.

3.7.11 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 $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

3.7.12 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 $(a-1)(b-1) - 1 = ab - a - b$.

3.7.13 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.

3.8 Permutations

3.8.1 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		

3.8.2 Cycles

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

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

3.8.3 Derangements

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

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

3.8.4 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

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

3.9 Partitions and subsets

3.9.1 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

3.9.2 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

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

3.9.3 2nd Kaplansky's Lemma

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

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

3.10 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}

3.11 General purpose numbers

3.11.1 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. $\pi(j) > j$.

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

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

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

3.11.2 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,

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

3.11.3 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 $n \neq 1$.

3.11.4 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:

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

3.11.5 Lucas Number

Number of edge cover of a cycle graph C_n is L_n

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

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

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

3.14 Matching Formula

3.14.1 Normal Graph

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

3.14.2 Bipartite Graph

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

3.15 Inequalities

3.15.1 Titu's Lemma

For positive reals $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$,

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

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

3.16 Games

3.16.1 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 $G(x) = 0$.

3.16.2 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.

3.16.3 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*.

3.17 Tree Hashing

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

3.18 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.

3.19 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.

3.20 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)

3.21 Convolution

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

3.22 Matrix Rotation

3.22.1 Anti-Clockwise Rotation

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

3.22.2 Clockwise Rotation

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

3.23 Common Formulas

3.23.1 Permutation

$${}^nP_r = \frac{n!}{(n-r)!}$$

3.23.2 Combination

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

3.24 Logarithms

3.24.1 Change of Base Formula

$$\log_a x = \frac{\log_b x}{\log_b a}$$

3.25 Common Series Sums

3.25.1 Sum of first n positive integers

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

3.25.2 Sum of first n odd positive integers

$$1 + 3 + 5 + \cdots + (2n-1) = n^2$$

3.25.3 Sum of first n even positive integers

$$2 + 4 + 6 + \cdots + 2n = n(n+1)$$

3.25.4 Sum of first n squares

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

3.25.5 Sum of first n cubes

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \left[\frac{n(n+1)}{2}\right]^2$$

3.26 Progressions

3.26.1 Arithmetic Progression

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

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