Contents

1 During Contest

1.1 Contest begin

1. The typer gives statements to his teammates and keeps the PC password
2. Write the password in QWERTY
3. Set the language layout to AZERTY: `setxkbmap fr`
4. Login to PC2
5. Open standings
6. Open VSCode and `Ctrl` + `Shift` + `P` autosave
7. Write template

```
#include "bits/stdc++.h"
using namespace std;
typedef long long ll;
#define int ll
#define all(x) begin(x), end(x)
typedef pair<int, int> pii;
#define input "input.in"
#define output "output.out"

void solve() {}

signed main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  if (fopen(input, "r")) freopen(input, "r", stdin);
  if (fopen(output, "r")) freopen(output, "w+", stdout);
  int t = 1;
  cin >> t;
  while (t--) solve();
}
```

1. Check standings
2. Change `.bashrc`

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 -fsanitize=undefined,address'
e() { c "$1" -o "./.build/${1%.*}" && ./.build/"${1%.*}"; }
```

1.2 Troubleshooting

1.2.1 Pre-submit

1. Write a few simple test cases if sample is not enough.
2. Are time limits close? If so, generate max cases.
3. Is the memory usage fine? Could anything overflow?
4. Make sure to submit the right file.

1.2.2 Wrong answer

1. Print your solution! Print debug output, as well.
2. Are you clearing all data structures between test cases?
3. Can your algorithm handle the whole range of input?
4. Read the full problem statement again.
5. Do you handle all corner cases correctly?
6. Have you understood the problem correctly?
7. Any uninitialized variables?
8. Any overflows?
9. Confusing N and M, i and j, etc.?
10. Are you sure your algorithm works?
11. What special cases have you not thought of?
12. Are you sure the STL functions you use work as you think?
13. Add some assertions, maybe resubmit.
14. Create some testcases to run your algorithm on.
15. Go through the algorithm for a simple case.
16. Go through this list again.
17. Explain your algorithm to a teammate.
18. Ask the teammate to look at your code.
19. Go for a small walk, e.g. to the toilet.
20. Is your output format correct? (including whitespace)
21. Rewrite your solution from the start or let a teammate do it.

1.2.3 Runtime error

1. Have you tested all corner cases locally?
2. Any uninitialized variables?
3. Are you reading/writing outside the range of any vector?
4. Any assertions that might fail?
5. Any possible division by 0? (`mod 0` for example)
6. Any possible infinite recursion?
7. Invalidated pointers or iterators?
8. Are you using too much memory?
9. Debug with resubmits (e.g. remapped signals, see Various).

1.2.4 Time limit exceeded

1. Do you have any possible infinite loops?
2. What is the complexity of your algorithm?
3. Are you copying a lot of unnecessary data? (References)
4. How big is the input and output? (consider `scanf`)
5. Avoid `vector`, `map`. (use `arrays`/`unordered_map`)
6. What do your teammates think about your algorithm?

1.2.5 Memory limit exceeded

1. What is the max amount of memory your algorithm should need?
2. Are you clearing all data structures between test cases?

2 Data Structures

2.1 Segment Tree

```
/**
 * Description: Zero-indexed tree.
 * Bounds are inclusive to the left and to the right.
 * Can be changed by modifying T, merge and neutral.
 */

template <class T>
struct Seg {
  int n;
  T neutral;  // to change
  vector<T> t;
  Seg(int n = 0, T neutral = T()) : n(n), neutral(neutral) {
    t.assign(4 * n, neutral);
  }
  Seg(int n, vector<T> &a, T neutral = T()) : n(n), neutral(neutral) {
    t.assign(4 * n, neutral);
    build(a, 1, 0, n - 1);
  }
  virtual T merge(T a, T b) = 0;  // (any associative fn)
  void build(vector<T> &a, int v, int l, int r) {
    if (l == r)
      t[v] = a[l];
    else {
      int m = (l + r) / 2;
      build(a, 2 * v, l, m);
      build(a, 2 * v + 1, m + 1, r);
      t[v] = merge(t[2 * v], t[2 * v + 1]);
    }
  }
  // query for range [a,b]
  T query(int a, int b, int v, int l, int r) {
    if (a > r || b < l) return neutral;
    if (a == l && b == r) return t[v];
    int m = (l + r) / 2;
    auto ql = query(a, min(b, m), 2 * v, l, m);
    auto qr = query(max(a, m + 1), b, 2 * v + 1, m + 1, r);
    return merge(ql, qr);
  }
  T query(int a, int b) { return query(a, b, 1, 0, n - 1); }
  // update value at index idx to val
  void update(int idx, int val, int v, int l, int r) {
    if (l == r) {
      t[v] = val;
    } else {
      int m = (l + r) / 2;
      if (idx <= m)
        update(idx, val, 2 * v, l, m);
      else
        update(idx, val, 2 * v + 1, m + 1, r);
    }
  }
  void update(int idx, int val) { update(idx, val, 1, 0, n - 1); }
};
```

2.2 Lazy Segment Tree

```
template <class T>
struct LazySeg {
  int n;
  vector<T> t;
  vector<T> lazy;
  // 0 for sum/gcd/or/xor and 1 for
  // and/product and infinity for max/min
  static const T neutral;
  // 0 for sum/gcd/or/xor and 1 for and/product
  // and infinity for max/min
  static const T neutralLazy;
  LazySeg() = default;
  LazySeg(vector<T>& a) {
    n = a.size();
    t.resize(4 * n, neutral);
    lazy.resize(4 * n, neutralLazy);
    build(a, 1, 0, n - 1);
  }
  T merge(T a, T b) { return a | b; }
```

```
T conquer(T a, T b) { return a & b; }
void propagate(int v, int l, int r) {
  if (l == r) return;
  t[2 * v] = merge(t[2 * v], lazy[v]);
  lazy[2 * v] = merge(lazy[2 * v], lazy[v]);
  t[2 * v + 1] = merge(t[2 * v + 1], lazy[v]);
  lazy[2 * v + 1] = merge(lazy[2 * v + 1], lazy[v]);
  lazy[v] = neutralLazy;
}
void build(vector<T>& a, int v = 1, int l = 0, int r = n - 1) {
  if (l == r)
    t[v] = a[l];
  else {
    int m = (l + r) / 2;
    build(a, 2 * v, l, m);
    build(a, 2 * v + 1, m + 1, r);
    t[v] = conquer(t[2 * v], t[2 * v + 1]);
  }
}
void update(int a, int b, T val, int v, int l, int r) {
  propagate(v, l, r);
  if (a > r || b < l) return;
  if (l >= a && r <= b) {
    t[v] = merge(t[v], val);
    lazy[v] = merge(lazy[v], val);
    return;
  }
  int m = (l + r) / 2;
  update(a, b, val, 2 * v, l, m);
  update(a, b, val, 2 * v + 1, m + 1, r);
  t[v] = conquer(t[2 * v], t[2 * v + 1]);
}
void update(int a, int b, T val) { update(a, b, val, 1, 0, n - 1); }
// query for range [a,b]
T query(int a, int b, int v, int l, int r) {
  propagate(v, l, r);
  if (a > r || b < l) return neutral;
  if (l >= a && r <= b) return t[v];
  int m = (l + r) / 2;
  auto ql = query(a, b, 2 * v, l, m);
  auto qr = query(a, b, 2 * v + 1, m + 1, r);
  return conquer(ql, qr);
}
T query(int a, int b) { return query(a, b, 1, 0, n - 1); }
};
```

2.3 Matrix

```
template <class T>
struct Matrix {
  typedef Matrix M;
  int N;
  vector<vector<T>> d;
  Matrix(int n) : N(n), d(n, vector<T>(n)) {}
  M operator*(const M& m) const {
    M ans(N);
    rep(i, 0, N) rep(k, 0, N) if (d[i][k]) rep(j, 0, N) {
      ans.d[i][j] += (d[i][k] * m.d[k][j]) % MOD;
      ans.d[i][j] %= MOD;
    }
    return ans;
  }
  vector<T> operator*(const vector<T>& vec) const {
    vector<T> ans(N);
    rep(i, 0, N) rep(j, 0, N) {
      ans[i] += (d[i][j] * vec[j]) % MOD, ans[i] %= MOD;
    }
    return ans;
  }
  M operator^(ll p) const {
    assert(p >= 0);
    M a(N), b(*this);
    rep(i, 0, N) a.d[i][i] = 1;
    while (p) {
      if (p & 1) a = a * b;
      b = b * b;
      p >>= 1;
    }
    return a;
```

```
  }
};
```

2.4 Persistent Segment Tree

```
struct PersistentSegmentTree {
  struct Vertex {
    Vertex *l, *r;
    int sum;
    int xxor;
    Vertex(int val, int x)
        : l(nullptr), r(nullptr), sum(val), xxor(x) {}
    Vertex(Vertex* l, Vertex* r) : l(l), r(r), sum(0), xxor(0) {
      if (l) sum += l->sum;
      if (r) sum += r->sum;
      if (l) xxor ^= l->xxor;
      if (r) xxor ^= r->xxor;
    }
  };
  int conquer(int a, int b) { return a ^ b; }
  Vertex* build(int tl, int tr) {
    if (tl == tr) return new Vertex(0LL, 0LL);
    int tm = (tl + tr) / 2;
    return new Vertex(build(tl, tm), build(tm + 1, tr));
  }

  Vertex* update(Vertex* v, int tl, int tr, int pos, int val) {
    if (tl == tr) return new Vertex(1, val);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
      return new Vertex(update(v->l, tl, tm, pos, val), v->r);
    else
      return new Vertex(v->l, update(v->r, tm + 1, tr, pos, val));
  }

  int find_xor(Vertex* vl, Vertex* vr, int tl, int tr, int k) {
    if (tl == tr) return vr->xxor ^ vl->xxor;
    int tm = (tl + tr) / 2;
    int left_count = vr->l->sum - vl->l->sum;
    int left_res = vr->l->xxor ^ vl->l->xxor;
    if (left_count >= k) return find_xor(vl->l, vr->l, tl, tm, k);
    return left_res ^
           find_xor(vl->r, vr->r, tm + 1, tr, k - left_count);
  }
};
```

2.5 Fenwick Tree

```
/**
 * Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1],
 * and updates single elements a[i],
 * taking the difference between the old and new value.
 */

template <class T>
struct FT {
  vector<T> s;
  FT(int n) : s(n) {}
  void update(int pos, T dif) {  // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  // sum of values in [0, pos)
  T query(int pos) {
    T res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos - 1];
    return res;
  }
  // min pos st sum of [0, pos] >= sum
  // Returns n if no sum is >= sum, or -1 if empty sum is.
  int lower_bound(T sum) {
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw - 1] < sum)
        pos += pw, sum -= s[pos - 1];
    }
    return pos;
  }
};
```

2.6 Disjoint Set Union

```
/**
 * Description: Disjoint-set data structure.
 */

struct DSU {
  vector<int> e;
  DSU(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    e[a] += e[b];
    e[b] = a;
    return true;
  }
};
```

2.7 Rollback Disjoint Set Union

```
/**
 * Description: Disjoint-set data structure with rollback.
 */

struct DSU {
  vector<int> e;
  vector<pair<int, int>> st;
  DSU(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return st.size(); }
  void rollback(int t) {
    for (int i = time(); i-- > t;) e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b];
    e[b] = a;
    return true;
  }
};
```

3 Graph

3.1 Shortest Paths

```
/**
 * Shortest path
 * T is the distance metric (e.g. int or long long).
 */

template <class node, typename D>
struct Dijkstra {
  vector<D> dist;
  vector<vector<pair<int, D>>> adj;
  Dijkstra(vector<vector<pair<int, D>>> &_adj) : adj(_adj) {}
  void compute(int s) {
    dist.assign(adj.size(), numeric_limits<D>::max());
    typedef tuple<D, node> state;
    priority_queue<state, vector<state>, greater<state>> q;
    q.push({0, node(s)});
    dist[s] = 0;
    while (!q.empty()) {
      auto [d, cur] = q.top();
      q.pop();

      for (auto [w, c] : adj[cur])
```

```
        if (dist[c] > d + w) {
          dist[c] = d + w;
          q.push({dist[c], c});
        }
      }
    }
    D get(int idx) { return dist[idx]; }
};const int INF = 1e9 + 7;

template <class edge>
void bellmanford(int n, int v, vector<edge> &edges) {
  vector<int> d(n, INF);
  d[v] = 0;
  vector<int> p(n, -1);
  int x;
  for (int i = 0; i < n; ++i) {
    x = -1;
    for (auto e : edges)
      if (d[e.a] < INF)
        if (d[e.b] > d[e.a] + e.cost) {
          d[e.b] = max(-INF, d[e.a] + e.cost);
          p[e.b] = e.a;
          x = e.b;
        }
  }

  if (x == -1)
    cout << "No negative cycle from " << v;
  else {
    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 << ' ';
  }
} const int INF = 1e9 + 7;

void floydwarshall(int n, vector<vector<int>> &d) {
  for (int i = 0; i < n; i++) d[i][i] = 0;
  for (int k = 0; k < n; ++k)
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j)
        if (d[i][k] < INF && d[k][j] < INF) {
          d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
          d[j][i] = min(d[j][i], d[j][k] + d[k][i]);
        }
}
```

3.2 MST

```
struct Edge {
  int w = INT_MAX, to = -1;
  bool operator>(Edge const& other) const {
    return make_pair(w, to) > make_pair(other.w, other.to);
  }
};

struct Prim {
  int n;
  vector<vector<Edge>> adj;
```

```
  Prim(int n, vector<vector<Edge>>& adj) : n(n), adj(adj) {}
  vector<Edge> mst() {
    int cost = 0;
    vector<Edge> res;
    priority_queue<Edge, vector<Edge>, greater<Edge>> q;
    q.push({0, 0});
    vector<bool> vis(n, false);
    while (!q.empty()) {
      auto e = q.top();
      auto [w, v] = e;
      q.pop();
      cost += w;
      vis[v] = true;
      res.push_back(e);
      for (auto e : adj[v])
        if (!vis[e.to]) q.push(e);
    }
    return res;
  }
}; #include "../data-structures/DSU.h"

struct Edge {
  int u, v, weight;
  bool operator<(Edge const& other) { return weight < other.weight; }
};

struct Kruskal {
  int n;
  vector<Edge> edges;
  DSU uf;
  Kruskal(int n, vector<Edge>& edges) : n(n), edges(edges), uf(n) {}
  vector<Edge> mst() {
    vector<Edge> res;
    sort(edges.begin(), edges.end());
    int cost = 0;
    for (auto [u, v, w] : edges)
      if (uf.find(u) != uf.find(v)) {
        cost += w;
        res.push_back({u, v, w});
        uf.join(u, v);
      }
    return res;
  }
};
```

3.3 Tree

```
#include "../data-structures/SegmentTree.h"

template <class node>
struct HLD {
  int n, root, segsz = 0;
  vector<vector<int>> adj;
  vector<int> parent, depth, heavy, head, pos;
  Seg<node> t;
  HLD(int n, vector<vector<int>> &adj, vector<node> &vals, int root = 0)
      : n(n), adj(adj), root(root) {
    parent.resize(n);
    depth.resize(n);
    heavy.resize(n);
    head.resize(n);
    pos.resize(n);
    dfs(root, -1);
    decompose(root, root);
    t = Seg<node>(segsz, vals);
  }
  int dfs(int v, int p, int d = 0) {
```

```
    parent[v] = p, depth[v] = d, heavy[v] = -1;
    int sz = 1, mxsz = 0;
    for (auto c : adj[v])
      if (c != p) {
        auto csz = dfs(c, v, d + 1);
        sz += csz;
        if (csz > mxsz) mxsz = csz, heavy[v] = c;
      }
    return sz;
  }
  void decompose(int v, int curhead) {
    head[v] = curhead, pos[v] = segsz++;
    if (heavy[v] != -1) decompose(heavy[v], curhead);
    for (auto c : adj[v])
      if (c != parent[v] && c != heavy[v]) decompose(c, c);
  }
  node query(int u, int v) {
    node res = t.neutral;
    while (head[u] != head[v]) {
      if (depth[head[u]] < depth[head[v]]) swap(u, v);
      auto curans = t.query(pos[head[u]], pos[u]);
      res = t.merge(res, curans);
      u = parent[head[u]];
    }
    if (pos[u] > pos[v]) swap(u, v);
    auto curans = t.query(pos[u], pos[v]);
    res = t.merge(res, curans);
    return res;
  }
};
#include "../data-structures/SegmentTree.h"

struct RMQLCA {
  int n;
  vector<int> first, epath, etime;
  struct RMQ : Seg<int> {
    RMQ() : Seg<int>() { neutral = INT_MAX; }
    RMQ(int n, vector<int> &a) : Seg<int>(n, a) {}
    int merge(int l, int r) override { return min(l, r); }
  };
  RMQ t;
  RMQLCA(vector<vector<int>> &adj, int root = 0) {
    n = adj.size();
    first.resize(n);
    epath.reserve(n * 2);
    etime.reserve(n * 2);
    dfs(adj, root, root);
    int m = etime.size();
    t = RMQ(m, etime);
  }
  void dfs(vector<vector<int>> &adj, int v, int p, int h = 0) {
    first[v] = epath.size();
    for (auto c : adj[v])
      if (c != p) {
        epath.push_back(v), etime.push_back(first[v]);
        dfs(adj, c, v, h + 1);
      }
  }
  int lca(int u, int v) {
    auto [l, r] = minmax(first[u], first[v]);
    return epath[t.query(l, r)];
  }
};

struct BLLCA {
  int n, l, timer;
```

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

    VirtualTree(int _n) : n(_n) {
      LOG = __lg(n) + 2; // Use +2 for safety
      tin.resize(n), tout.resize(n), depth.resize(n);
      up.assign(n, vector<int>(LOG));
      adj.resize(n), vadj.resize(n);
      timer = 0;
    }

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

    void dfs_lca(int v, int p, int d) {
      tin[v] = timer++;
      up[v][0] = p;
      depth[v] = d;
      for (int i = 1; i < LOG; ++i)
        up[v][i] = up[up[v][i - 1]][i - 1];

      for (int u : adj[v])
        if (u != p)  dfs_lca(u, v, d + 1);
      tout[v] = timer++;
    }

    void build(int root = 0) { dfs_lca(root, root, 0); }

    bool isAncestor(int u, int v) {
      return tin[u] <= tin[v] && tout[u] >= tout[v];
    }

    int lca(int u, int v) {
      if (isAncestor(u, v)) return u;
      if (isAncestor(v, u)) return v;
```

```
  for (int i = LOG - 1; i >= 0; --i)
    if (!isAncestor(up[u][i], v)) u = up[u][i];
  return up[u][0];
}

void clearVirtualTree() {
  for (int v : used_nodes) vadj[v].clear();
  used_nodes.clear();
}

vector<int> buildVirtualTree(vector<int> &nodes) {
  if (nodes.size() <= 1)   return nodes;
  clearVirtualTree();

  auto cmpTin = [&](int a, int b) { return tin[a] < tin[b]; };
  sort(nodes.begin(), nodes.end(), cmpTin);

  int k = nodes.size();
  for (int i = 0; i < k - 1; ++i)
    nodes.push_back(lca(nodes[i], nodes[i + 1]));

  sort(nodes.begin(), nodes.end(), cmpTin);
  nodes.erase(unique(nodes.begin(), nodes.end()), nodes.end());

  vector<int> st;
  for (int v : nodes) {
    while (!st.empty() && !isAncestor(st.back(), v)) st.pop_back();
    if (!st.empty())  vadj[st.back()].push_back(v);
    st.push_back(v);
    used_nodes.push_back(v);
  }
  return nodes;
  }
};
```

3.4 Miscellaneous

```
vector<int> toposort(const vector<vector<int>>& adj) {
  vector<int> deg(adj.size()), q;
  for (int i = 0; i < adj.size(); i++) deg[i] = adj[i].size();
  for (int i = 0; i < adj.size(); i++)
    if (deg[i] == 0) q.push_back(i);
  for (int j = 0; j < q.size(); j++)
    for (auto x : adj[q[j]])
      if (--deg[x] == 0) q.push_back(x);
  return q;
}
```

3.4.1 Number of Spanning Trees

Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--, mat[b][b]++` (and `mat[b][a]--, mat[a][a]++` if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

3.4.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq ... \geq d_n$ exists iff $d_1 + ... + d_n$ is even and for every $k = 1...n$: $\Sigma\{i=1\}^k d\_i \leq k(k-1) + \Sigma\{i=k+1\}^n \min(d\_i, k)$

4 Strings

4.1 Hashing

```
/**
 * Return the prefix hashes of a string using a pool of prime numbers.
 */

struct Hashing {
```

```
const ll mod = 1e9 + 7;
vector<vector<ll>> p, pref;
// assuming s is only composed of a-z characters
Hashing(const string &s) {
  int n = s.length();
  // change if s contains other chars than a-z
  // primes should be greater than number of possible characters
  vector primes{37, 41, 43};
  p.assign(primes.size(), vector<ll>(n + 1));
  pref.assign(primes.size(), vector<ll>(n + 1));
  for (int i = 0; i < primes.size(); i++)
    p[i][0] = 1, p[i][1] = primes[i];
  for (int i = 0; i < p.size(); i++) {
    p[i].resize(n + 1);
    for (int j = 0; j < n; j++) {
      // change here if s contains other chars than a-z
      int c = s[j] - 'a' + 1;
      pref[i][j + 1] = (pref[i][j] + c * p[i][j]) % mod;
      p[i][j + 1] = p[i][j] * p[i][1] % mod;
    }
  }
}
// hash of s[l..r] inclusive for l & r
vector<ll> get(int l, int r) {
  int n = p[0].size() - 1;
  vector<ll> ans(p.size());
  for (int i = 0; i < p.size(); i++) {
    ans[i] = (pref[i][r + 1] - pref[i][l] + mod) % mod;
    ans[i] = ans[i] * p[i][n - l] % mod;
  }
  return ans;
}
bool compare(pair<int, int> r1, pair<int, int> r2) {
  auto res1 = get(r1.first, r1.second);
  auto res2 = get(r2.first, r2.second);
  for (int i = 0; i < p.size(); i++)
    if (res1[i] != res2[i]) return false;
  return true;
  }
};
```

4.2 KMP

```
/**
 * pi[x] computes the length of the longest prefix of s that ends
 * at x, other than s[0...x] itself (abacaba -> 0010123).
 * Can be used to find all occurrences of a string.
 */

vector<int> pi(const string& s) {
  vector<int> p(s.length());
  for (int i = 1; i < s.length(); i++) {
    int g = p[i - 1];
    while (g && s[i] != s[g]) g = p[g - 1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}

vector<int> match(const string& s, const string& pat) {
  vector<int> p = pi(pat + '\0' + s), res;
  int n = s.length(), m = pat.length();
  for (int i = m; i < n + m + 1; i++)
    if (p[i] == m) res.push_back(i - 2 * m);
  return res;
}
```

4.3 Manacher

```
/**
 * For each position in a string, computes p[0][i] = half length of
 * longest even palindrome around pos i (abaccaba -> 00004000),
 * p[1][i] = longest odd (half rounded down) (abacaba -> 1214121).
 */

struct Manacher {
  array<vector<int>, 2> p;
  Manacher(const string& s) {
```

```
    int n = s.length();
    p = {vector<int>(n), vector<int>(n)};
    for (int z = 0; z < 2; z++)
      for (int i = 0, l = 0, r = 0; i < n; i++) {
        int t = r - i + !z;
        if (i < r) p[z][i] = min(t, p[z][l + t]);
        int L = i - p[z][i], R = i + p[z][i] - !z;
        while (L >= !z && R + !z < n && s[L - !z] == s[R + !z])
          p[z][i]++, L--, R++;
        if (R > r) l = L, r = R;
      }
  }
  bool check(int l, int r) {
    int z = (r - l + 1) & 1;
    int m = (l + r + 1) / 2;
    return p[z][m] >= r - m + 1;
  }
};
```

5 Geometry

5.1 Point

```
/**
 * Description: Class to handle points in the plane.
 *    T can be e.g. double or long long. (Avoid int.)
 */

template <class T>
struct Point {
  typedef Point P;
  T x, y;
  Point(T x = 0, T y = 0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x, y) < tie(p.x, p.y); }
  bool operator==(P p) const { return tie(x, y) == tie(p.x, p.y); }
  P operator+(P p) const { return P(x + p.x, y + p.y); }
  P operator-(P p) const { return P(x - p.x, y - p.y); }
  P operator*(T d) const { return P(x * d, y * d); }
  P operator/(T d) const { return P(x / d, y / d); }
  T dot(P p) const { return x * p.x + y * p.y; }
  T dot(P a, P b) const { return (a - *this).dot(b - *this); }
  T cross(P p) const { return x * p.y - y * p.x; }
  T cross(P a, P b) const { return (a - *this).cross(b - *this); }
  T dist2() const { return x * x + y * y; }
  T dist2(P p) const { return dist2(*this - p); }
  double dist() const { return sqrt((double)dist2()); }
  double dist(P p) const { return sqrt((double)dist2(p)); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this / dist(); }  // makes dist()=1
  P perp() const { return P(-y, x); }        // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
  }
  friend istream &operator>>(istream &os, P &p) {
    return os >> p.x >> p.y;
  }
  friend ostream &operator<<(ostream &os, P p) {
    return os << "(" << p.x << "," << p.y << ")";
  }
  bool between(P b, P c) {
    P bl(min(b.x, c.x), min(b.y, c.y)),
      tr(max(b.x, c.x), max(b.y, c.y));
    return x >= bl.x && x <= tr.x && y >= bl.y && y <= tr.y;
  }
};
```

5.2 Line

Line $a \cdot x + b \cdot y + c = 0$:

- $\boldsymbol{n} = (a, b)$ is a normal vector, $\boldsymbol{d} = (-b, a)$ is a direction vector.
- Parallel lines at distance r have equations: $a \cdot x + b \cdot y + c \pm r \cdot \sqrt{a^2 + b^2} = 0$.
- Distance of a point (x, y) to the line: $\frac{a \cdot x + b \cdot y + c}{\sqrt{a^2 + b^2}}$.

- The sign of $a \cdot x + b \cdot y + c$ determines the point's side relative to the line (0 means the point lies on the line).

```cpp
#include "Point.h"

template <class T>
struct Line {
  using P = Point<T>;
  using L = Line;
  T a, b, c;
  Line() : a(0), b(0), c(0) {}
  Line(P u, P v) {
    a = u.y - v.y;
    b = v.x - u.x;
    c = -a * u.x - b * u.y;
    auto g = gcd(gcd(abs(a), abs(b)), abs(c));
    // for double: sqrt(a*a + b*b)
    assert(g != 0);
    // for double: abs(g) > EPS
    a /= g, b /= g, c /= g;
    if (a < 0 || (a == 0 && b < 0)) a *= -1, b *= -1, c *= -1;
    // for double: a < -EPS || (abs(a) <= EPS && b < -EPS)
  }
  Line(T a, T b, T c) : a(a), b(b), c(c) {}
  bool operator==(L &l) { return tie(a, b, c) == tie(l.a, l.b, l.c); }
  bool colinear(L &l) { return a * l.b == b * l.a; }
  double dist(P &p) {
    return (a * p.x + b * p.y + c) / sqrt(a * a + b * b);
  }
  P intersection(L &l) {
    auto v1 = P{a, l.a};
    auto v2 = P{b, l.b};
    auto v3 = P{c, l.c};
    // make sure T is double
    P p = {-v3.cross(v2) / v1.cross(v2),
           -v1.cross(v3) / v1.cross(v2)};
    return p;
  }
};
```

5.3 Segment

```cpp
#include "Point.h"

template <class T>
struct Segment {
  using P = Point<T>;
  using S = Segment;
  P p1, p2;
  Segment() : p1(), p2() {}
  Segment(P u, P v) : p1(u), p2(v) {}
  bool intersect(S &l) {
    auto [p3, p4] = l;
    // adjust with epsilon when using double
    auto sgn = [](T x) { return x > 0 ? 1 : x < 0 ? -1 : 0; };
    auto c1 = p3.cross(p4, p1);
    auto c2 = p3.cross(p4, p2);
    auto c3 = p1.cross(p2, p3);
    auto c4 = p1.cross(p2, p4);
    // adjust with epsilon when using double
    if (c1 == 0 && c2 == 0)
      return p1.between(p3, p4) || p2.between(p3, p4) ||
             p3.between(p1, p2) || p4.between(p1, p2);
    return sgn(c1) != sgn(c2) && sgn(c3) != sgn(c4);
  }
  bool has(P &p) {
    return p1.dot(p, p2) == 0 && p.between(p1, p2);
  }
};
```

5.4 Ray

```cpp
#include "Point.h"
#include "Line.h"

template <class T>
struct Ray {
  using P = Point<T>;
  using L = Line<T>;
  using R = Ray;
```

```
  P p1, p2;  // ray from p1 to p2
  L l;

  Ray(P p1, P p2) : p1(p1), p2(p2), l(p1, p2) {}

  double dist(P x) {
    if (p1.dot(p2, x) <= 0) {
      return (p1 - x).dist();
    } else {
      auto [a, b, c] = l;
      return abs(a * x.x + b * x.y + c) / sqrt(a * a + b * b);
    }
  }

  bool intersect(R &r) {
    if ((p1 - p2).cross(r.p1, r.p2) == 0) {
      if (l.c != r.l.c) return false;
      return p1.dot(p2, r.p1) >= 0 || r.p1.dot(r.p1, p1) >= 0;
    } else {
      auto i = l.intersection(r.l);
      return p1.dot(i, p2) >= 0 && r.p1.dot(i, r.p2) >= 0;
    }
  }

  double dist(R r) {
    if (intersect(r)) return 0;
    return min(dist(r.p1), r.dist(p1));
  }
};
```

5.5 Bisector

```
template <class P, class L>
L bisector(P x, P y, P z) {
  // make sure T is double
  y = (y - x), z = (z - x);
  P mid = (y * z.dist() + z * y.dist());
  return L(x, x + mid);
}
```

5.6 Circle

```
#include "Point.h"
#include "Line.h"

template <class T>
struct Circle {
  using P = Point<T>;
  using L = Line<T>;
  using C = Circle;
  T r;
  P o;
  Circle(P o, T r) : o(o), r(r) {}
  vector<P> intersection(L l) {
    P ab = b - a, p = a + ab * (c - a).dot(ab) / ab.dist2();
    double s = a.cross(b, c), h2 = r * r - s * s / ab.dist2();
    if (h2 < 0) return {};
    if (h2 == 0) return {p};
    P h = ab.unit() * sqrt(h2);
    return {p - h, p + h};
  }
};
```

5.7 Closest Pair

```
template <class P>
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> s;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (auto p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) s.erase(v[j++]);
    auto lo = s.lower_bound(p - d), hi = s.upper_bound(p + d);
    while (hi != lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}}), lo++;
    s.insert(p);
  }
```

```
  return ret.second;
}
```

5.8 Rotating Callipers

```
template <class T>
vector<pair<P, P>> rotatingcallipers(vector<P> &p) {
  int n = p.size();
  vector<pair<P, P>> res;
  // parallel edges shouldn't be visited twice
  vector<bool> vis(n, false);
  auto sgn = [](ll x) { return x < 0 ? -1 : x > 0; };
  auto nxt = [&](int idx) { return (idx + 1) % n; };
  auto prv = [&](int idx) { return (idx - 1 + n) % n; };
  for (int p1 = 0, p2 = 0; p1 < n; p1++) {
    auto u = p[nxt(p1)] - p[p1];
    auto v1 = p[nxt(p2)] - p[p2], v2 = p[p2] - p[prv(p2)];
    while (p1 == p2 || nxt(p1) == p2 ||
           sgn(u.cross(v1)) == sgn(u.cross(v2))) {
      p2 = nxt(p2);
      v1 = p[nxt(p2)] - p[p2], v2 = p[p2] - p[prv(p2)];
    }
    if (vis[p1]) continue;
    vis[p1] = true;
    res.push_back({p[p1], p[p2]});
    res.push_back({p[nxt(p1)], p[p2]});
    // if both edges from p1 and p2 are parallel to each other
    if (u.cross(v1) == 0) {
      res.push_back({p[p1], p[nxt(p2)]});
      res.push_back({p[nxt(p1)], p[nxt(p2)]});
      vis[p2] = true;
    }
  }
  return res;
}
```

5.9 Polygon Area

```
/**
 * Description: Returns twice the signed area of a polygon.
 *    Clockwise enumeration gives negative area.
 *    Watch out for overflow if using int as P!
 */

template <class P>
P polygonArea2(vector<Point<P>>& v) {
  P ans = v.back().cross(v[0]);
  for (int i = 0; i < v.size() - 1; i++) ans += v[i].cross(v[i + 1]);
  return ans;
};
```

5.10 In Polygon

```
/**
 * Returns 1 if p lies inside the polygon, 0 if on the boundary of
 * the ploygon and -1 if outside the polygon.
 * Time: O(n).
 */

template <class P>
int inPolygon(vector<P> &poly, P cur) {
  auto intersect = [](P cur, P p1, P p2) {
    return cur.cross(p1, p2) == 0 &&
           (min(p1.x, p2.x) <= cur.x && max(p1.x, p2.x) >= cur.x) &&
           (min(p1.y, p2.y) <= cur.y && max(p1.y, p2.y) >= cur.y);
  };

  int crossings = 0;
  bool isBoundary = false;
  int n = poly.size();

  for (int i = 0; !isBoundary && i < n; i++) {
    auto p1 = poly[i], p2 = poly[(i + 1) % n];
    if (p1.y > p2.y) swap(p1, p2);

    if (intersect(cur, p1, p2)) {
      isBoundary = true;
    } else {
      auto isTouch = cur.cross(p1, p2) >= 0;
      auto isBetween = (p1.y < cur.y) && (p2.y >= cur.y);
```

```
      if (isTouch && isBetween) crossings++;
    }
  }

  if (!isBoundary)
    return (crossings & 1 ? 1 : -1);
  else
    return 0;
}
```

5.11 Circle Tangents

```
/**
 * Finds the external tangents of two circles, or internal if r2 is
 * negated. Can return 0, 1, or 2 tangents -- 0 if one circle contains
 * the other (or overlaps it, in the internal case, or if the circles
 * are the same); 1 if the circles are tangent to each other (in which
 * case .first = .second and the tangent line is perpendicular to the
 * line between the centers). .first and .second give the tangency
 * points at circle 1 and 2 respectively. To find the tangents of a
 * circle with a point set r2 to 0.
 */

template <class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
  if (d2 == 0 || h2 < 0) return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

5.12 Convex Hull

```
/**
 * Returns a vector of the points of the convex hull in
 * counter-clockwise order.
 * Points on the edge of the hull between two other points are
 * considered part of the hull.
 */

template <class P>
vector<P> convexHull(vector<P> &v) {
  vector<P> ch;
  auto chk = [&](P z) {
    auto x = ch[ch.size() - 2], y = ch[ch.size() - 1];
    return x.cross(y, z) <= 0;
  };
  sort(v.begin(), v.end(),
       [](P a, P b) { return a.x == b.x ? a.y > b.y : a.x < b.x; });
  for (auto cur : v) {  // construct upper hull
    while (ch.size() >= 2 && !chk(cur)) ch.pop_back();
    ch.push_back(cur);
  }
  ch.pop_back();
  reverse(v.begin(), v.end());
  int s = ch.size();
  for (auto cur : v) {  // construct lower hull
    while (ch.size() - s >= 2 && !chk(cur)) ch.pop_back();
    ch.push_back(cur);
  }
  ch.pop_back();
  return ch;
}
```

5.13 Point 3D

```
template <class T>
struct Point3D {
  typedef Point3D P;
  T x, y, z;
  explicit Point3D(T x = 0, T y = 0, T z = 0) : x(x), y(y), z(z) {}
```

```
  bool operator<(P& p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z);
  }
  bool operator==(P& p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z);
  }
  P operator+(P& p) const { return P(x + p.x, y + p.y, z + p.z); }
  P operator-(P& p) const { return P(x - p.x, y - p.y, z - p.z); }
  P operator*(T d) const { return P(x * d, y * d, z * d); }
  P operator/(T d) const { return P(x / d, y / d, z / d); }
  T dot(P& p) const { return x * p.x + y * p.y + z * p.z; }
  P cross(P& p) const {
    return P(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y * p.x);
  }
  T dist2() const { return x * x + y * y + z * z; }
  double dist() const { return sqrt((double)dist2()); }
  // Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
  double phi() const { return atan2(y, x); }
  // Zenith angle (latitude) to the z-axis in interval [0, pi]
  double theta() const { return atan2(sqrt(x * x + y * y), z); }
  P unit() const { return *this / (T)dist(); }  // makes dist()=1
  // returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  // returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle);
    P u = axis.unit();
    return u * dot(u) * (1 - c) + (*this) * c - cross(u) * s;
  }
};
```

5.14 Point

```
/**
 * Description: Class to handle points in the plane.
 *    T can be e.g. double or long long. (Avoid int.)
 */

template <class T>
struct Point {
  typedef Point P;
  T x, y;
  Point(T x = 0, T y = 0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x, y) < tie(p.x, p.y); }
  bool operator==(P p) const { return tie(x, y) == tie(p.x, p.y); }
  P operator+(P p) const { return P(x + p.x, y + p.y); }
  P operator-(P p) const { return P(x - p.x, y - p.y); }
  P operator*(T d) const { return P(x * d, y * d); }
  P operator/(T d) const { return P(x / d, y / d); }
  T dot(P p) const { return x * p.x + y * p.y; }
  T dot(P a, P b) const { return (a - *this).dot(b - *this); }
  T cross(P p) const { return x * p.y - y * p.x; }
  T cross(P a, P b) const { return (a - *this).cross(b - *this); }
  T dist2() const { return x * x + y * y; }
  T dist2(P p) const { return dist2(*this - p); }
  double dist() const { return sqrt((double)dist2()); }
  double dist(P p) const { return sqrt((double)dist2(p)); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this / dist(); }  // makes dist()=1
  P perp() const { return P(-y, x); }        // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
  }
  friend istream &operator>>(istream &os, P &p) {
    return os >> p.x >> p.y;
  }
  friend ostream &operator<<(ostream &os, P p) {
    return os << "(" << p.x << "," << p.y << ")";
  }
  bool between(P b, P c) {
    P bl(min(b.x, c.x), min(b.y, c.y)),
      tr(max(b.x, c.x), max(b.y, c.y));
    return x >= bl.x && x <= tr.x && y >= bl.y && y <= tr.y;
  }
};
```

6 Mathematics

6.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -\frac{b}{2a}$.

$$\begin{cases} ax + by = e \\ cx + dy = f \end{cases} \Rightarrow \begin{cases} x = \frac{ed-bf}{ad-bc} \\ y = \frac{af-ec}{ad-bc} \end{cases}$$

In general, given an equation $Ax = b$, the solution to a variable x_i is given by,

$$x_i = \frac{\det(A_i')}{\det(A)}$$

where A_i' is A with the i'th column replaced by b.

6.2 Recurrences

If $a_n = c_1 a_{n-1} + ... + c_k a_{n-k}$ and $r_1, ..., r_k$ are distinct roots of

$$x^k - c_1 x^{k-1} - ... - c_k$$

there are $d_1, ..., d_k$ such that

$$a_n = d_1 r_1^n + ... + d_k r_k^n$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

6.3 Trigonometry

$$\sin(v + w) = \sin(v)\cos(w) + \cos(v)\sin(w)$$

$$\cos(v + w) = \cos(v)\cos(w) - \sin(v)\sin(w)$$

$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$

$$\sin v + \sin w = 2\sin\left(\frac{v+w}{2}\right)\cos\left(\frac{v-w}{2}\right)$$

$$\cos v + \cos w = 2\cos\left(\frac{v+w}{2}\right)\cos\left(\frac{v-w}{2}\right)$$

$$(V + W)\frac{\tan(v-w)}{2} = (V - W)\frac{\tan(v+w)}{2}$$

where V, W are lengths of sides opposite angles v, w.

$$a\cos x + b\sin x = r\cos(x - \varphi)$$

$$a\sin x + b\cos x = r\sin(x + \varphi)$$

where $r = \sqrt{a^2 + b^2}, \varphi = \text{atan2}(b, a)$.

6.4 Geometry

6.4.1 Triangles

Side lengths: a, b, c

Area: $A = \sqrt{p(p-a)(p-b)(p-c)}$

Circumradius: $R = ab\frac{c}{4A}$

Semiperimeter: $p = \frac{a+b+c}{2}$

Inradius: $r = \frac{A}{p}$

Length of median: $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Angle bisector: $s_a = \sqrt{bc\left(1 - \left(\frac{a}{b+c}\right)^2\right)}$

Law of sines: $\sin\frac{\alpha}{a} = \sin\frac{\beta}{b} = \sin\frac{\gamma}{c} = \frac{1}{2R}$

Law of cosines: $a^2 = b^2 + c^2 - 2bc\cos\alpha$

Law of tangents: $\frac{a+b}{a-b} = \frac{\tan\left(\frac{\alpha+\beta}{2}\right)}{\tan\left(\frac{\alpha-\beta}{2}\right)}$

6.4.2 Quadrilaterals

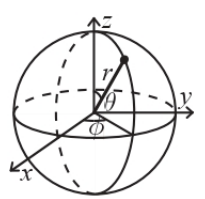
With side lengths a, b, c, d, diagonals e, f, diagonals angle θ, area A, and flux $F = b^2 + d^2 - a^2 - c^2$:

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

For cyclic quadrilaterals: opposite angles sum to 180°,

$ef = ac + bd$ and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

6.4.3 Spherical coordinates



$$x = r\sin(\theta)\cos(\varphi) \qquad r = \sqrt{x^2 + y^2 + z^2}$$

$$y = r\sin(\theta)\sin(\varphi) \qquad \theta = \text{acos}\left(\frac{z}{\sqrt{x^2+y^2+z^2}}\right)$$

$$z = r\cos(\theta) \qquad \varphi = \text{atan2}(y, x)$$

6.5 Derivatives/Integrals

$$\frac{d}{\mathrm{dx}}\arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{\mathrm{dx}}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{\mathrm{dx}}\tan x = 1+\tan^2 x \qquad \frac{d}{\mathrm{dx}}\arctan x = \frac{1}{1+x^2}$$

$$\int \tan(ax) = -\frac{\ln|\cos(ax)|}{a} \qquad \int e^{-x^2} = \frac{\sqrt{\pi}}{2}\,\mathrm{erf}(x)$$

$$\int x\sin(ax) = \frac{\sin(ax)-ax\cos(ax)}{a^2} \int xe^{ax} = \frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

$$\int_a^b f(x)g(x)\mathrm{dx} = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)\ \mathrm{dx}$$

6.6 Sums

$$\sum_{i=m}^{n} c^i = \frac{c^{n+1}-c^m}{c-1}, c\neq 1 \qquad \sum_{i=0}^{n}\binom{n}{i}a^{n-i}b^i = (a+b)^n$$

$$\sum_{i=0}^{n}\binom{n}{i} = 2^n \qquad \sum_{i=0}^{n}\binom{i}{k} = \binom{n+1}{k+1}$$

$$\sum_{i=0}^{n} i\binom{n}{i} = n2^{n-1} \qquad \sum_{i=0}^{n}\frac{\binom{n}{i}}{i+1} = \frac{2^{n+1}-1}{n+1}$$

$$\sum_{i=0}^{n} i!\binom{n}{i} = \lfloor n!e\rfloor \qquad \sum_{i=0}^{n} iP_k\binom{n}{i} = P_k\left(2^{n-k}\right)$$

$$\sum_{i=0}^{n} i\cdot i! = (n+1)!-1 \qquad \sum_{k=0}^{m}\binom{n+k}{n} = \binom{n+m+1}{n+1}$$

$$\sum_{i=0}^{n}\binom{n}{i}^2 = \binom{2n}{n} \qquad \sum_{i=0}^{n}\frac{1}{i!} = \frac{\lfloor n!e\rfloor}{n!}$$

$$\sum_{i=1}^{n} i + kP_{k+1} = \sum_{i=1}^{n}\prod_{j=0}^{k}(i+j) = \frac{(n+k+1)!}{(n-1)!(k+2)}$$

$$\sum_{i=0}^{n} i!\binom{n}{i} = \sum_{i=0}^{n} P_n i = \lfloor n!e\rfloor$$

6.7 Series

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

$$\ln(1+x) = x-\frac{x^2}{2}+\frac{x^3}{3}-...,(-1<x\leq 1)$$

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

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

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

6.8 Probability theory

Let X be a discrete random variable with distribution $p_X(x)$. Then:

Expected value (mean): $\mu = \mathbb{E}[X] = \sum_x xp_{X(x)}$

Variance: $\sigma^2 = V(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2 = \sum_x (x-E[X])^2 p_{X(x)}$

For continuous X, replace sums with integrals.

Linearity: $\mathbb{E}(aX+bY) = a\mathbb{E}(X)+b\mathbb{E}(Y)$

For independent X, Y: $V(aX+bY) = a^2V(X)+b^2V(Y)$

6.8.1 Discrete distributions

6.8.1.1 Binomial distribution

The number of successes in n independent yes/no experiments, each of which yields success with probability p is $\mathrm{Bin}(n,p), n=1,2,...,0\leq p\leq 1$.

$$p(k) = \binom{n}{k}p^k(1-p)^{n-k}$$

$$\mu = np, \sigma^2 = np(1-p)$$

$\mathrm{Bin}(n,p)$ is approximately $\mathrm{Po}(np)$ for small p.

6.8.1.2 First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each of which yields success with probability p, is $\mathrm{Fs}(p), 0\leq p\leq 1$.

$$p(k) = p(1-p)^{k-1}, k=1,2,...$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1-p}{p^2}$$

6.8.1.3 Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\text{Po}(\lambda), \lambda = t\kappa$.

$$p(k) = e^{-\lambda}\frac{\lambda^k}{k!}, k = 0, 1, 2, ...$$

$$\mu = \lambda, \sigma^2 = \lambda$$

6.8.2 Continuous distributions

6.8.2.1 Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $U(a, b), a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} a < x < b \\ 0 \quad \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

6.8.2.2 Exponential distribution

$$\text{Exp}(\lambda),$$

The time between events in a Poisson process is $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} x \geq 0 \\ 0 \qquad x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

6.8.2.3 Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $N(\mu, \sigma^2), \sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1 \sim N(\mu_1, \sigma_1^2)$ and $X_2 \sim N(\mu_2, \sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim N(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

6.9 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1, X_2, ...$ be a sequence of random variables generated by the Markov process. Then there is a transition matrix $\boldsymbol{P} = (p_{\text{ij}})$, with $p_{\text{ij}} = \Pr(X_n = i \mid X_{n-1} = j)$, and $\boldsymbol{p}^{(\text{n})} = \boldsymbol{P}^n\boldsymbol{p}^{(0)}$ is the probability distribution for X_n (i.e., $p_i^{(\text{n})} = \Pr(X_n = i)$), where $\boldsymbol{p}^{(0)}$ is the initial distribution.

6.9.1 Stationary distribution

π is a stationary distribution if $\pi = \pi\boldsymbol{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i. $\frac{\pi_j}{\pi_i}$ is the expected number of visits in state j between two visits in state i.

For a connected, undirected, and non-bipartite graph where the transition probability is uniform among all neighbors, π_i is proportional to node i's degree.

6.9.2 Ergodicity

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k\to\infty} \boldsymbol{P}^k = \boldsymbol{1}\pi$.

6.9.3 Absorption

A Markov chain is an A-chain if the states can be partitioned into two sets $\boldsymbol{A}$ and $\boldsymbol{G}$, such that all states in $\boldsymbol{A}$ are absorbing ($p_{\text{ii}} = 1$), and all states in $\boldsymbol{G}$ lead to an absorbing state in $\boldsymbol{A}$. The probability for absorption in state $i \in \boldsymbol{A}$, when the initial state is j, is

$$a_{\text{ij}} = p_{\text{ij}} + \sum_{k \in \boldsymbol{G}} a_{\text{ik}} p_{\text{kj}}$$

. The expected time until absorption, when the initial state is i, is

$$t_i = 1 + \sum_{k \in \boldsymbol{G}} p_{ki} t_k$$

.

7 Combinatorics

7.1 Permutations

7.1.1 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

7.1.2 Cycles

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

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

.

7.1.3 Derangements

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

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

7.1.4 Burnside's lemma

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

$$\left(\frac{1}{|G|}\right)\sum_{g\in G}|X^g|$$

where X^g are the elements fixed by g ($g \cdot x = x$).

If $f(n)$ counts "configurations" of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get $g(n) = \left(\frac{1}{n}\right)\sum_{k=0}^{n-1} f(\gcd(n,k)) = \left(\frac{1}{n}\right)\sum_{k\mid n} f(k)\varphi\left(\frac{n}{k}\right)$.

7.2 Partitions and subsets

7.2.1 Partition function

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

$$p(0)=1, \quad p(n) = \sum_{k\in\mathbb{Z};k\neq 0} (-1)^{k+1} p\left(n - \frac{k(3k-1)}{2}\right)$$

$$p(n) \sim \frac{0.145}{n} \cdot \exp\left(2.56\sqrt{n}\right)$$

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

7.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + \ldots + n_1 p + n_0$ and $m = m_k p^k + \ldots + m_1 p + m_0$. Then

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

.

```
#include "BinaryExponentiation.h"
vector<ll> fact;

ll nCk(ll n, ll k, ll p = mod) {
  if (k > n) return 0;
  if (k == 0) return 1;
   return ((fact[n] * inv(fact[k], p) % p) * inv(fact[n -
k], p)) % p;
}

ll nCk1(ll n, ll k, ll p = mod) {
  if (k > n) return 0;
  if (k == 0) return 1;
  return ((nCk(n % p, k % p, p) % mod) * (nCk1(n / p, k /
p, p)) %
          mod) %
         mod;
}

ll starsandbars(ll stars, ll bars) {
  return nCk(bars + stars, stars);
}

ll C(ll n, ll r) {
  ll p = 1, k = 1;
  if (r > n) return 0;
  if (n - r < r) r = n - r;
  if (r != 0)
    while (r) {
      p *= n, k *= r;
      ll m = gcd(p, k);
      p /= m, k /= m;
      n--, r--;
    }
  return p;
}
```

7.3 General purpose numbers

7.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t - 1}$ (FFT-able). $B[0, ...] = \left[1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, ...\right]$.

Sums of powers:

$$\sum_{i=1}^{n} n^m = \left(\frac{1}{m+1}\right) \sum_{k=0}^{m} \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_{m}^{\infty} f(x)\ \mathrm{dx} - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(\mathrm{k}\text{-}1)(m)}$$

$$\approx \int_{m}^{\infty} f(x)\ \mathrm{dx} + \frac{f(m)}{2} - f' \frac{m}{12} + f''' \frac{m}{720} + O\left(f^{(5)(m)}\right).$$

7.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n, k) = c(n-1, k-1) + (n-1)c(n-1, k), c(0, 0) = 1$$

$$\sum_{k=0}^{n} c(n, k) x^k = x(x+1)...(x+n-1)$$

$c(8, k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n, 2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, ...$

7.3.3 Eulerian numbers

Number of permutations $\pi \in S_n$ with exactly k descents (j such that $\pi(j) > \pi(j+1)$).

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

7.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k),$$
$$S(n, 1) = S(n, n) = 1$$
$$S(n, k) = \left(\frac{1}{k!}\right) \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n$$

7.3.5 Bell numbers

Total number of partitions of n distinct elements.

$$B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, ...$$

For prime p, $B(p^m + n) \equiv mB(n) + B(n+1) (\mathrm{mod}\, p)$.

7.3.6 Labeled unrooted trees

Number on n vertices: n^{n-2}

Number formed from k existing trees of sizes n_i: $n_1 n_2 ... n_k; n^{k-2}$

Number with degrees d_i: $\frac{(n-2)!}{(d_1-1)!...(d_n-1)!}$

7.3.7 Catalan numbers

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

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

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, ...$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parentheses, correctly nested.
- binary trees 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 subsequence.

```
const int mod = 1e9 + 7;
const int n = 1e6 + 1;
int catalan[n];

void gencatalan() {
  catalan[0] = catalan[1] = 1;
  for (int i = 2; i <= n; i++) {
    catalan[i] = 0;
    for (int j = 0; j < i; j++) {
      catalan[i] += (catalan[j] * catalan[i - j - 1]) % mod;
      if (catalan[i] >= mod) catalan[i] -= mod;
    }
  }
}
```

8 Number theory

8.1 Modular arithmetic

```
#include "Euclid.h"

const ll mod = 1e9 + 7;  // change to something else
struct Mod {
  ll x;
  Mod(ll x) : x(x) {}
  Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
  Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
  Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
  Mod operator/(Mod b) { return *this * invert(b); }
  Mod invert(Mod a) {
    ll x, y, g = euclid(a.x, mod, x, y);
    assert(g == 1);
    return Mod((x + mod) % mod);
  }
  Mod operator^(ll e) {
    if (!e) return Mod(1);
    Mod r = *this ^ (e / 2);
    r = r * r;
    return e & 1 ? *this * r : r;
  }
};

ll binpow(ll a, ll b, ll m = mod) {
  if (a == 0) return 0;
  ll res = 1;
  while (b > 0) {
    if (b & 1) res = res * a % m;
    a = (a * a) % m;
    b >>= 1;
  }
  return res;
}

ll inv(ll a, ll m = mod) { return binpow(a, m - 2, m); }
```

8.2 Primality

```
const int n = 1e6 + 1;
int lpd[n];
vector<int> primes;

void sieve() {
  for (int i = 2; i < n; ++i) {
    if (lpd[i] == 0) lpd[i] = i, primes.push_back(i);
    for (int j = 0; i * primes[j] < n; ++j) {
      lpd[i * primes[j]] = primes[j];
      if (primes[j] == lpd[i]) break;
    }
  }
}
```

8.3 Divisibility

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

8.3.1 Bézout's identity

For $a \neq 0$, $b \neq 0$, let $d = \gcd(a, b)$ be the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), k \in \mathbb{Z}$$

```
const int n = 1e6 + 1;
int phi[n];

int totient() {
  for (int i = 0; i < n; i++) phi[i] = i;
  for (int i = 1; i < n; i++)
    for (int j = 2 * i; j < n; j += i) phi[j] -= phi[i];
}
```

8.4 Fractions

```
/**
 * If fraction is negative, the nominator holds the sign.
 * Nominator and denominator values can go up to 1e9.
 * For values up to 1e18, replace ll with ld and int with ll
 * and add the proper implementation of gcd for ld
 */

struct Frac {
  ll a, b;
  Frac(int a) : a(a), b(1) {}
  Frac(int x, int y) {
    assert(y != 0);
    auto sgn = x && (x < 0) ^ (y < 0) ? -1 : 1;
    x = abs(x), y = abs(y);  // ensure positive denominator
    auto g = gcd(x, y);
    a = sgn * x / g, b = y / g;
  }
  Frac operator+(Frac f) { return Frac(a * f.b + b * f.a,
b * f.b); }
  Frac operator-(Frac f) { return Frac(a * f.b - b * f.a,
b * f.b); }
  Frac operator*(Frac f) { return Frac(a * f.a, b * f.b); }
  Frac operator/(Frac f) { return Frac(a * f.b, b * f.a); }
  Frac inverse() { return Frac(b, a); }
  bool operator<(Frac f) { return a * f.b < b * f.a; }
  bool operator<=(Frac f) { return a * f.b <= b * f.a; }
  bool operator>(Frac f) { return a * f.b > b * f.a; }
  bool operator>=(Frac f) { return a * f.b >= b * f.a; }
};
```

8.5 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

8.6 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 1000000.

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

8.7 Number / Sum / Product of Divisors

If the prime factorization of n is $p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$, where p_i are distinct prime numbers, then:

- Number of divisors: $d(n) = (e_1 + 1)(e_2 + 1) \cdots (e_k + 1)$
- Sum of divisors: $\sigma(n) = \frac{p_1^{e_1+1}-1}{p_1-1} \times \frac{p_2^{e_2+1}-1}{p_2-1} \times \cdots \times \frac{p_k^{e_k+1}-1}{p_k-1}$
- Product of divisors: $n^{\frac{d(n)}{2}}$

```
const ll mod = 1e9 + 7;

tuple<ll, ll, ll> numprodsum(vector<pll> &a) {
  int n = a.size();

  // num  is the number of divisors mod 1e9+7
  ll num = 1, sum = 1, prod = 1;
  for (auto [p, k] : a) {
    num *= (k + 1) % mod, num %= mod;
    sum *= (binpow(p, k + 1) - 1) * inv(p - 1) % mod, sum
%= mod;
  }
  // there is no way to compute inverse modulo of 2 mod (1e9
+ 7) - 1
  // or any prime modulus, since 2 and mod - 1 are always
not coprime
  vector<ll> precompute(n);
  for (ll i = 0, num2 = 1; i < n; i++) {
    auto [p, k] = *(a.begin() + i);
    precompute[i] =
        binpow(p, num2 * (k * (k + 1) / 2 % (mod - 1)) %
(mod - 1));
    num2 *= (k + 1) % (mod - 1), num2 %= mod - 1;
  }

  for (ll i = n - 1, num2 = 1; i >= 0; i--) {
    auto [p, k] = *(a.begin() + i);
    prod *= binpow(precompute[i], num2), prod %= mod;
    num2 *= (k + 1) % (mod - 1), num2 %= mod - 1;
  }

  return make_tuple(num, sum, prod);
}
```

8.8 Gray Code

```
int g(int n) { return n ^ (n >> 1); }

int revg(int g) {
  int n = 0;
  while (g) n ^= g, g >>= 1;
  return n;
}
```

8.9 Fibonacci

```
pair<int, int> fib(int n) {
  if (n == 0) return {0, 1};
  auto p = fib(n >> 1);
  auto c = p.first * (2 * p.second - p.first);
  auto d = p.first * p.first + p.second * p.second;
  if (n & 1)
    return {d, c + d};
  else
    return {c, d};
}
```

8.10 Estimates

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

The upper bound on the number of divisors is $n^{\frac{1}{3}}$. The number of divisors of n is at most around 100 for $n < 5 \times 10^4$, 500 for $n < 10^7$, 1344 for $n < 10^9$, 2000 for $n < 10^{10}$, $200,000$ for $n < 10^{19}$.

8.11 Möbius Function

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

Möbius Inversion:

$$g(n) = \sum_{d \mid n} f(d) \Leftrightarrow f(n) = \sum_{d \mid n} \mu(d) g\left(\frac{n}{d}\right)$$

Other useful formulas/forms:

$\sum_{d \mid n} \mu(d) = [n = 1]$ (very useful)

$$g(n) = \sum_{n \mid d} f(d) \Leftrightarrow f(n) = \sum_{n \mid d} \mu\left(\frac{d}{n}\right) g(d)$$

$$g(n) = \sum_{1 \le m \le n} f\left(\left\lfloor \frac{n}{m} \right\rfloor\right) \Leftrightarrow f(n) = \sum_{1 \le m \le n} \mu(m) g\left(\left\lfloor \frac{n}{m} \right\rfloor\right)$$

```cpp
const int n = 1e6 + 1;
bool prime[n];
int mob[n];

void mobius() {
  for (int i = 2; i < n; i++) mob[i] = -1, prime[i] = true;
  for (int i = 2; i < n; i++)
    if (prime[i]) {
      mob[i] = 1;
      for (int j = 2 * i; j < n; j += i) {
        prime[j] = false;
        mob[j] = j % (i * i) == 0 ? 0 : -mob[j];
      }
    }
}
```

9 Miscalenous

10 Optimization Tricks

10.1 Bit Hacks

- `n & -n` gives the least significant set bit of $n.
- `n | (1 << x)` sets the x-th bit in `n`.
- `n ^ (1 << x)` flips the x-th bit in `n`.
- `n & ~(1 << x)` clears the x-th bit in `n`.
- `n & (n + 1)` clears all trailing ones in `n`.
- `n | (n + 1)` sets the last cleared bit in `n`.
- To loop over all subset masks of m (excluding m itself): `for (int x = m; x;){ --x &= m; }`
- To find the next number after x with the same number of bits set, use: `c = x &-x, r = x + c; (((r^x) >> 2) / c) | r`
- For cumulative sums of subsets, use:

```cpp
for (int b = 0; b < k; b++)
  for (int i = 0; i < (1 << k); i++)
    if (i & 1 << b) D[i] += D[i ^ (1 << b)];
```

10.2 Language & Compiler Support

C++ supports some of the bit hacks since C++20 via the bit standard library:

1. `has_single_bit`: checks if the number is a power of two
2. `bit_ceil` / `bit_floor`: round up/down to the next power of two
3. `rotl` / `rotr`: rotate the bits in the number
4. `countl_zero` / `countr_zero` / `countl_one` / `countr_one`: count the leading/trailing zeros/ones
5. `popcount`: count the number of set bits

GCC defines a list at Built-in Functions Provided by GCC that also work in older versions of C++:

1. `__builtin_popcount(unsigned int)` returns the number of set bits (`__builtin_popcount(0b0001'0010'1100) == 4`)
2. `__builtin_ffs(int)` finds the index of the first (most right) set bit (`__builtin_ffs(0b0001'0010'1100) == 3`)
3. `__builtin_clz(unsigned int)` the count of leading zeros (`__builtin_clz(0b0001'0010'1100) == 23`)
4. `__builtin_ctz(unsigned int)` the count of trailing zeros (`__builtin_ctz(0b0001'0010'1100) == 2`)
5. `__builtin_parity(x)` the parity (even or odd) of the number of ones in the bit representation

Note that some of the operations (both the C++20 functions and the Compiler Built-in ones) might be quite slow in GCC if you don't enable a specific compiler target with `#pragma GCC target("popcnt")`.

10.3 Pragmas

1. Use `#pragma GCC optimize ("Ofast")` to enable GCC's auto-vectorization and better optimization for floating points (assumes associativity and turns off denormals).
2. Use `#pragma GCC target ("avx,avx2")` to potentially double performance of vectorized code, though it may cause crashes on older machines. Consider using `#pragma GCC target ("sse4")` for older architectures.
3. Use `#pragma GCC optimize ("trapv")` to kill the program on integer overflows (but it may slow down execution).