1 Setup

1.0.1 Tips

Test session: Check `__int128`, GNU builtins, and end of line whitespace requirements.
C++ var. limits: `int` -2^{31}, $2^{31}-1$
`ll` -2^{63}, $2^{63}-1$
`ull` 0, $2^{64}-1$
`__int128` -2^{127}, $2^{127}-1$
`ld` $-1.7e308$, $1.7e308$, 18 digits precision

1.0.2 Xmodmap setup

```
remove Lock = Caps_Lock
keysym Escape = Caps_Lock
keysym Caps_Lock = Escape
add Lock = Caps_Lock
```

1.0.3 header.h

```cpp
#pragma once
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define ull unsigned ll
#define ld long double
#define pl pair<ll, ll>
#define pi pair<int, int>
#define vl vector<ll>
#define vi vector<int>
#define vb vector<bool>
#define vvi vector<vi>
#define vvl vector<vl>
#define vpl vector<pl>
#define vpi vector<pi>
#define vld vector<ld>
#define vvpi vector<vpi>
#define in(el, cont) (cont.find(el) != cont.end())//sets/maps
#define all(x) x.begin(), x.end()
#define rep(i, a, b) for(int i = a; i < (b); ++i)

constexpr int INF  = INT_MAX;
constexpr ll LLINF = LONG_LONG_MAX;

// int main() {
//  ios::sync_with_stdio(false);  // do not use cout + printf
//  cin.tie(NULL);
//  cout << fixed << setprecision(12);
//  return 0;
// }
```

1.0.4 Aux. helper C++

```cpp
#include "header.h"
int main() {
    // Read in a line including white space
    string line;
    getline(cin, line);
    // When doing the above read numbers as follows:
    int n;
    getline(cin, line);
    stringstream ss(line);
    ss >> n;

    // Count the number of 1s in binary represnatation of a number
    ull number;
    __builtin_popcountll(number);
}

// __int128
using lll = __int128;
ostream& operator<<( ostream& o, __int128 n ) {
  auto t = n<0 ? -n : n; char b[128], *d = end(b);
  do *--d = '0'+t%10, t /= 10; while (t);
  if(n<0) *--d = '-';
  o.rdbuf()->sputn(d,end(b)-d);
  return o;
}
```

1.0.5 Aux. helper python

```python
from functools import lru_cache

# Read until EOF
while True:
    try:
        pattern = input()
    except EOFError:
        break

@lru_cache(maxsize=None)
def smth_memoi(i, j, s):
    # Example in-built cache
    return "sol"

# Fast I
import io, os
def fast_io():
    finput = io.BytesIO(os.read(0,
        os.fstat(0).st_size)).readline
    s = finput().decode()
    return s

# Fast O
import sys
def fast_out():
    n = 5
    sys.stdout.write(str(n)+"\n")
```

2 Python

2.1 Graphs

2.1.1 BFS

```python
from collections import deque
def bfs(g, roots, n):
    q = deque(roots)
    explored = set()
    distances = [0 if v in roots else float('inf') for v in range(n)]
    while len(q) != 0:
        node = q.popleft()
        if node in explored: continue
        explored.add(node)
        for neigh in g[node]:
            if neigh not in explored:
                q.append(neigh)
                if distances[neigh] == float('inf'):
                    distances[neigh] = distances[node] + 1
    return distances
```

2.1.2 Dijkstra

```python
from heapq import *
def dijkstra(n, root, g):  # g = {node: (cost, neigh)}
  dist = [float("inf")]*n
  dist[root] = 0
  prev = [-1]*n

  pq = [(0, root)]
  heapify(pq)
  visited = set([])

  while len(pq) != 0:
    _, node = heappop(pq)

    if node in visited: continue
    visited.add(node)

    # In case of disconnected graphs
    if node not in g:
      continue

```

```
  for cost, neigh in g[node]:
   alt = dist[node] + cost
   if alt < dist[neigh]:
    dist[neigh] = alt
    prev[neigh] = node
    heappush(pq, (alt, neigh))
 return dist
```

2.1.3 Topological Sort topological sorting of a DAG

```
from collections import defaultdict
class Graph:
  def __init__(self,vertices):
    self.graph = defaultdict(list) #adjacency List
    self.V = vertices #No. V

  def addEdge(self,u,v):
    self.graph[u].append(v)

  def topologicalSortUtil(self,v,visited,stack):
    visited[v] = True
    # Recur for all the vertices adjacent to this vertex
    for i in self.graph[v]:
      if visited[i] == False:
        self.topologicalSortUtil(i,visited,stack)
    stack.insert(0,v)

  def topologicalSort(self):
    visited = [False]*self.V
    stack =[]
    for i in range(self.V):
      if visited[i] == False:
        self.topologicalSortUtil(i,visited,stack)
    return stack

  def isCyclicUtil(self, v, visited, recStack):
    visited[v] = True
    recStack[v] = True
    for neighbour in self.graph[v]:
      if visited[neighbour] == False:
        if self.isCyclicUtil(neighbour, visited, recStack) == True:
          return True
      elif recStack[neighbour] == True:
        return True
    recStack[v] = False
    return False

  def isCyclic(self):
    visited = [False] * (self.V + 1)
    recStack = [False] * (self.V + 1)
    for node in range(self.V):
      if visited[node] == False:
        if self.isCyclicUtil(node, visited, recStack) == True:
          return True
    return False
```

2.1.4 Kruskal (UnionFind) Min. span. tree

```
class UnionFind:
  def __init__(self, n):
    self.parent = [-1]*n

  def find(self, x):
    if self.parent[x] < 0:
      return x
    self.parent[x] = self.find(self.parent[x])
    return self.parent[x]

  def connect(self, a, b):
    ra = self.find(a)
    rb = self.find(b)
    if ra == rb:
      return False
    if self.parent[ra] > self.parent[rb]:
      self.parent[rb] += self.parent[ra]
      self.parent[ra] = rb
    else:
      self.parent[ra] += self.parent[rb]
      self.parent[rb] = ra
    return True

# Full MST is len(spanning==n-1)
def kruskal(n, edges):
  uf = UnionFind(n)
  spanning = []
  # Sort edges by asc. weight (check+-)
  edges.sort(key = lambda d: -d[2])
  while edges and len(spanning) < n-1:
    u, v, w = edges.pop()
    if not uf.connect(u, v):
      continue
    spanning.append((u, v, w))
  return spanning
```

2.1.5 Prim Min. span. tree - good for dense graphs

```
from heapq import heappush, heappop, heapify
def prim(G, n):
 s = next(iter(G.keys()))
 V = set([s])
 M = []
 c = 0

 E = [(w,s,v) for v,w in G[s].items()]
 heapify(E)

 while E and len(M) < n-1:
  w,u,v = heappop(E)
  if v in V: continue
  V.add(v)
  M.append((u,v))
  c += w
  u = v
  [heappush(E,(w,u,v)) for v,w in G[u].items() if v not in V]

 if len(M) == n-1:
  return M, c
 else:
  return None, None
```

2.2 Num. Th. / Comb.

2.2.1 nCk % prime p must be prime and k < p

```
def fermat_binom(n, k, p):
  if k > n:
    return 0
  num = 1
  for i in range(n-k+1, n+1):
    num *= i % p
  num %= p
  denom = 1
  for i in range(1,k+1):
    denom *= i % p
  denom %= p
  # numerator * denominator^(p-2) (mod p)
  return (num * pow(denom, p-2, p)) % p
```

2.2.2 Sieve of E. $O(n)$ so actually faster than C++ version, but more memory

```
MAX_SIZE = 10**8+1
isprime = [True] * MAX_SIZE
prime = []
SPF = [None] * (MAX_SIZE)
def manipulated_seive(N):  # Up to N (not included)
 isprime[0] = isprime[1] = False
 for i in range(2, N):
  if isprime[i] == True:
   prime.append(i)
   SPF[i] = i
  j = 0
```

```
    while (j < len(prime) and
      i * prime[j] < N and
        prime[j] <= SPF[i]):
      isprime[i * prime[j]] = False
      SPF[i * prime[j]] = prime[j]
      j += 1
```

2.2.3 Modular Inverse of a mod b

```
def modinv(a, b):
  if b == 1: return 1
  b0, x0, x1 = b, 0, 1
  while a > 1:
    q, a, b = a//b, b, a%b
    x0, x1 = x1 - q * x0, x0
  if x1 < 0: x1 += b0
  return x1
```

2.2.4 Chinese rem. an x such that ∀ y,m: yx = 1 mod m requires all m,m' to be >=1 and coprime

```
def chinese_remainder(ys, ms):
  N, x = 1, 0
  for m in ms: N*=m
  for y,m in zip(ys,ms):
    n = N // m
    x += n * y * modinv(n, m)
  return x % N
```

2.2.5 Bezout

```
def bezout_id(a, b):
    r,x,s,y,t,z = b,a,0,1,1,0
    while r:
        q = x // r
        x, r = r, x % r
        y, s = s, y - q * s
        z, t = t, z - q * t
    return y % (b // x), z % (-a // x)
```

2.2.6 Gen. chinese rem.

```
def general_chinese_remainder(a,b,m,n):
  g = gcd(m,n)

  if a == b and m == n:
    return a, m
  if (a % g) != (b % g):
    return None, None

  u,v = bezout_id(m,n)
  x = (a*v*n + b*u*m) // g
  return int(x) % lcm(m,n), int(lcm(m,n))
```

2.3 Strings

2.3.1 Longest common substr. (Consecutive) $O(mn)$ time, $O(m)$ space

```
from functools import lru_cache
@lru_cache
def lcs(s1, s2):
    if len(s1) == 0 or len(s2) == 0:
        return 0
    return max(
      lcs(s1[:-1], s2), lcs(s1, s2[:-1]),
      (s1[-1] == s2[-1]) + lcs(s1[:-1], s2[:-1])
    )
```

2.3.2 Longest common subseq. (Non-consecutive)

```
def longestCommonSubsequence(text1, text2):
    n = len(text1)
    m = len(text2)
    prev = [0] * (m + 1)
    cur = [0] * (m + 1)
    for idx1 in range(1, n + 1):
        for idx2 in range(1, m + 1):
            # matching
            if text1[idx1 - 1] == text2[idx2 - 1]:
                cur[idx2] = 1 + prev[idx2 - 1]
            else:
                # not matching
                cur[idx2] = max(cur[idx2 - 1], prev[idx2])
        prev = cur.copy()
    return cur[m]
```

2.3.3 KMP Return all matching pos. of P in T

```
class KMP:
    def partial(self, pattern):
        """ Calc. partial match table: String -> [Int]"""
        ret = [0]
        for i in range(1, len(pattern)):
            j = ret[i - 1]
            while j > 0 and pattern[j] != pattern[i]: j = ret[j - 1]
            ret.append(j + 1 if pattern[j] == pattern[i] else j)
        return ret

    def search(self, T, P):
        """KMPString -> String -> [Int]"""
        partial, ret, j = self.partial(P), [], 0
        for i in range(len(T)):
            while j > 0 and T[i] != P[j]: j = partial[j - 1]
            if T[i] == P[j]: j += 1
            if j == len(P):
                ret.append(i - (j - 1))
                j = partial[j - 1]
        return ret
```

2.3.4 Longest common pref. with the suffix array built we can do, e.g., longest common prefix of x, y with suffixarray where x,y are suffixes of the string used $O(\log n)$

```
def lcp(x, y, P):
    res = 0
    if x == y:
        return n - x
    for k in range(len(P) - 1, -1, -1):
        if x >= n or y >= n:
            break
        if P[k][x] == P[k][y]:
            x += 1 << k
            y += 1 << k
            res += 1 << k
    return res
```

2.3.5 Edit distance

```
def editDistance(str1, str2):
  m = len(str1)
  n = len(str2)
  curr = [0] * (n + 1)
  for j in range(n + 1):
    curr[j] = j
  previous = 0
  # dp rows
  for i in range(1, m + 1):
    previous = curr[0]
    curr[0] = i

    # dp cols
    for j in range(1, n + 1):
      temp = curr[j]
      if str1[i - 1] == str2[j - 1]:
        curr[j] = previous
      else:
```

```
        curr[j] = 1 + min(previous, curr[j - 1], curr[j])
      previous = temp
  return curr[n]
```

2.3.6 Bitstring

Slower than a set for many elements, but hashable. Also see `Hashing`

```
def add_element(bit_string, index):
    return bit_string | (1 << index)
def remove_element(bit_string, index):
    return bit_string & ~(1 << index)
def contains_element(bit_string, index):
    return (bit_string & (1 << index)) != 0
```

2.4 Geometry

2.4.1 Convex Hull

```
def vec(a,b):
    return (b[0]-a[0],b[1]-a[1])
def det(a,b):
    return a[0]*b[1] - b[0]*a[1]
def convexhull(P):
    if (len(P) == 1):
        return [(p[0][0], p[0][1])]

    h = sorted(P)
    lower = []
    i = 0
    while i < len(h):
        if len(lower) > 1:
            a = vec(lower[-2], lower[-1])
            b = vec(lower[-1], h[i])
            if det(a,b) <= 0 and len(lower) > 1:
                lower.pop()
                continue
        lower.append(h[i])
        i += 1

    upper = []
    i = 0
    while i < len(h):
        if len(upper) > 1:
            a = vec(upper[-2], upper[-1])
            b = vec(upper[-1], h[i])
            if det(a,b) >= 0:
                upper.pop()
                continue
        upper.append(h[i])
        i += 1

    reversedupper = list(reversed(upper[1:-1:]))
    reversedupper.extend(lower)
    return reversedupper
```

2.4.2 Geometry

```

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

def det(a,b):
    return a[0]*b[1] - b[0]*a[1]

    lower = []
    i = 0
    while i < len(h):
        if len(lower) > 1:
            a = vec(lower[-2], lower[-1])
            b = vec(lower[-1], h[i])
            if det(a,b) <= 0 and len(lower) > 1:
                lower.pop()
                continue
        lower.append(h[i])
        i += 1

    # find upper hull
    # det <= 0 -> replace
    upper = []
    i = 0
    while i < len(h):
        if len(upper) > 1:
            a = vec(upper[-2], upper[-1])
            b = vec(upper[-1], h[i])
            if det(a,b) >= 0:
                upper.pop()
                continue
        upper.append(h[i])
        i += 1
```

3 C++

3.1 Graphs

3.1.1 BFS

```
#include "header.h"
#define graph unordered_map<ll, unordered_set<ll>>
vi bfs(int n, graph& g, vi& roots) {
    vi parents(n+1, -1); // nodes are 1..n
    unordered_set<int> visited;
    queue<int> q;
    for (auto x: roots) {
        q.emplace(x);
        visited.insert(x);
    }
    while (not q.empty()) {
        int node = q.front();
        q.pop();

        for (auto neigh: g[node]) {
            if (not in(neigh, visited)) {
                parents[neigh] = node;
                q.emplace(neigh);
                visited.insert(neigh);
            }
        }
    }
    return parents;
}
vi reconstruct_path(vi parents, int start, int goal) {
    vi path;
    int curr = goal;
    while (curr != start) {
        path.push_back(curr);
        if (parents[curr] == -1) return vi(); // No path, empty vi
        curr = parents[curr];
    }
    path.push_back(start);
    reverse(path.begin(), path.end());
    return path;
}
```

3.1.2 DFS

Cycle detection / removal

```
#include "header.h"
void removeCyc(ll node, unordered_map<ll, vector<pair<ll, ll>>>& neighs, vector<bool>& visited,
vector<bool>& recStack, vector<ll>& ans) {
    if (!visited[node]) {
        visited[node] = true;
        recStack[node] = true;
        auto it = neighs.find(node);
        if (it != neighs.end()) {
            for (auto util: it->second) {
                ll nnode = util.first;
                if (recStack[nnode]) {
                    ans.push_back(util.second);
                } else if (!visited[nnode]) {
                    removeCyc(nnode, neighs, visited, recStack, ans);
                }
            }
        }
```

```
    }
    recStack[node] = false;
}
```

3.1.3 Dijkstra

```
#include "header.h"
vector<int> dijkstra(int n, int root, map<int, vector<pair<int, int>>>& g) {
  unordered_set<int> visited;
  vector<int> dist(n, INF);
    priority_queue<pair<int, int>> pq;
    dist[root] = 0;
    pq.push({0, root});
    while (!pq.empty()) {
        int node = pq.top().second;
        int d = -pq.top().first;
        pq.pop();

        if (in(node, visited)) continue;
        visited.insert(node);

        for (auto e : g[node]) {
            int neigh = e.first;
            int cost = e.second;
            if (dist[neigh] > dist[node] + cost) {
                dist[neigh] = dist[node] + cost;
                pq.push({-dist[neigh], neigh});
            }
        }
    }
    return dist;
}
```

3.1.4 Floyd-Warshall

```
#include "header.h"
// g[i][j] = infty if not path from i to j
// if g[i][i] < 0, i is contained in a negative cycle
void warshall(vvl& g) {
    for (int k=0; k<g.size(); ++k) {
        for (int i=0; i<g.size(); ++i) {
            for (int j=0; j<g.size(); ++j) {
                if (g[i][k] < LLONG_MAX and g[k][j] < LLONG_MAX and g[i][j] > g[i][k] + g[k][j]) {
                    g[i][j] = g[i][k] + g[k][j];
}}}}}
```

3.1.5 Kruskal

Minimum spanning tree of undirected weighted graph. $O(E \log E)$

```
#include "header.h"
#include "disjoint_set.h"
pair<set<pair<ll, ll>>, ll>  kruskal(vector<tuple<ll, ll, ll>>& edges, ll n) {
    set<pair<ll, ll>> ans;
    ll cost = 0;

    sort(edges.begin(), edges.end());
    DisjointSet<ll> fs(n);

    ll dist, i, j;
    for (auto edge: edges) {
        dist = get<0>(edge);
        i = get<1>(edge);
        j = get<2>(edge);

        if (fs.find_set(i) != fs.find_set(j)) {
            fs.union_sets(i, j);
            ans.insert({i, j});
            cost += dist;
        }
    }
    return pair<set<pair<ll, ll>>, ll> {ans, cost};
}
```

3.1.6 Hungarian algorithm

Given J jobs and W workers (J <= W), computes the minimum cost to assign each prefix of jobs to distinct workers.

```
#include "header.h"
template <class T> bool ckmin(T &a, const T &b) { return b < a ? a = b, 1 : 0; }
/**
 * @tparam T: type large enough to represent integers of O(J * max(|C|))
 * @param C: JxW matrix such that C[j][w] = cost to assign j-th
 * job to w-th worker (possibly negative)
 * @return a vector (length J), with the j-th entry = min. cost
 * to assign the first (j+1) jobs to distinct workers
 */
template <class T> vector<T> hungarian(const vector<vector<T>> &C) {
    const int J = (int)size(C), W = (int)size(C[0]);
    assert(J <= W);
    // a W-th worker added for convenience
    vector<int> job(W + 1, -1);
    vector<T> ys(J), yt(W + 1);  // potentials
    vector<T> answers;
    const T inf = numeric_limits<T>::max();
    for (int j_cur = 0; j_cur < J; ++j_cur) {
        int w_cur = W;
        job[w_cur] = j_cur;
        vector<T> min_to(W + 1, inf);
        vector<int> prv(W + 1, -1);
        vector<bool> in_Z(W + 1);
        while (job[w_cur] != -1) {   // runs at most j_cur + 1 times
            in_Z[w_cur] = true;
            const int j = job[w_cur];
            T delta = inf;
            int w_next;
            for (int w = 0; w < W; ++w) {
                if (!in_Z[w]) {
                    if (ckmin(min_to[w], C[j][w] - ys[j] - yt[w]))
                        prv[w] = w_cur;
                    if (ckmin(delta, min_to[w]))
                        w_next = w;
                }
            }
            for (int w = 0; w <= W; ++w) {
                if (in_Z[w]) ys[job[w]] += delta, yt[w] -= delta;
                else min_to[w] -= delta;
            }
            w_cur = w_next;
        }
        for (int w; w_cur != W; w_cur = w) job[w_cur] = job[w = prv[w_cur]];
        answers.push_back(-yt[W]);
    }
    return answers;
}
```

3.1.7 Suc. shortest path

Calculates max flow, min cost

```
#include "header.h"
// map<node, map<node, pair<cost, capacity>>>
#define graph unordered_map<int, unordered_map<int, pair<ld, int>>>
graph g;
const ld infty = 1e60l;  // Change if necessary
ld fill(int n, vld& potential) {  // Finds max flow, min cost
  priority_queue<pair<ld, int>> pq;
  vector<bool> visited(n+2, false);
  vi parent(n+2, 0);
  vld dist(n+2, infty);
  dist[0] = 0.l;
  pq.emplace(make_pair(0.l, 0));
  while (not pq.empty()) {
```

```cpp
    int node = pq.top().second;
    pq.pop();
    if (visited[node]) continue;
    visited[node] = true;
    for (auto& x : g[node]) {
      int neigh = x.first;
      int capacity = x.second.second;
      ld cost = x.second.first;
      if (capacity and not visited[neigh]) {
        ld d = dist[node] + cost + potential[node] - potential[neigh];
        if (d + 1e-10l < dist[neigh]) {
          dist[neigh] = d;
          pq.emplace(make_pair(-d, neigh));
          parent[neigh] = node;
  }}}}

  for (int i = 0; i < n+2; i++) {
    potential[i] = min(infty, potential[i] + dist[i]);
  }
  if (not parent[n+1]) return infty;
  ld ans = 0.l;
  for (int x = n+1; x; x=parent[x]) {
    ans += g[parent[x]][x].first;
    g[parent[x]][x].second--;
    g[x][parent[x]].second++;
  }
  return ans;
}
```

3.1.8 Bipartite check

```cpp
#include "header.h"
int main() {
    int n;
    vvi adj(n);

    vi side(n, -1);    // will have 0's for one side 1's for other side
    bool is_bipartite = true;   // becomes false if not bipartite
    queue<int> q;
    for (int st = 0; st < n; ++st) {
        if (side[st] == -1) {
            q.push(st);
            side[st] = 0;
            while (!q.empty()) {
                int v = q.front();
                q.pop();
                for (int u : adj[v]) {
                    if (side[u] == -1) {
                        side[u] = side[v] ^ 1;
                        q.push(u);
                    } else {
                        is_bipartite &= side[u] != side[v];
                    }
}}}}}
```

3.1.9 Bipartite matching (Hopcroft-Karp)

Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched. Time: $O(\sqrt{V}E)$

```cpp
// Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);

bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
  if (A[a] != L) return 0;
  A[a] = -1;
  for (int b : g[a]) if (B[b] == L + 1) {
    B[b] = 0;
    if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
      return btoa[b] = a, 1;}
  return 0;}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
  int res = 0;
  vi A(g.size()), B(btoa.size()), cur, next;
  for (;;) {
    fill(all(A), 0); fill(all(B), 0);
    /// Find the starting nodes for BFS (i.e. layer 0).
    cur.clear();
    for (int a : btoa) if(a != -1) A[a] = -1;
    rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
    /// Find all layers using bfs.
    for (int lay = 1;; lay++) {
      bool islast = 0;
      next.clear();
      for (int a : cur) for (int b : g[a]) {
        if (btoa[b] == -1) {
          B[b] = lay; islast = 1;}
        else if (btoa[b] != a && !B[b]) {
          B[b] = lay;
          next.push_back(btoa[b]);}}
      if (islast) break;
      if (next.empty()) return res;
      for (int a : next) A[a] = lay;
      cur.swap(next);
    }
    /// Use DFS to scan for augmenting paths.
    rep(a,0,sz(g))
      res += dfs(a, 0, g, btoa, A, B);
  }
}
```

3.1.10 Find cycle directed

```cpp
#include "header.h"
int n;
const int mxN = 2e5+5;
vvi adj(mxN);
vector<char> color;
vi parent;
int cycle_start, cycle_end;
bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u)) return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}
void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;
    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))break;
    }
    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle Found: ";
        for (int v : cycle) cout << v << " ";
        cout << endl;
    }
}
```

3.1.11 Find cycle undirected

```
#include "header.h"
int n;
const int mxN = 2e5 + 5;
vvi adj(mxN);
vector<bool> visited;
vi parent;
int cycle_start, cycle_end;
bool dfs(int v, int par) { // passing vertex and its parent vertex
    visited[v] = true;
    for (int u : adj[v]) {
        if(u == par) continue; // skipping edge to parent vertex
        if (visited[u]) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
        parent[u] = v;
        if (dfs(u, parent[u]))
            return true;
    }
    return false;
}
void find_cycle() {
    visited.assign(n, false);
    parent.assign(n, -1);
    cycle_start = -1;
    for (int v = 0; v < n; v++) {
        if (!visited[v] && dfs(v, parent[v])) break;
    }
    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        cout << "Cycle Found: ";
        for (int v : cycle) cout << v << " ";
        cout << endl;
    }
}
```

3.1.12 Tarjan's SCC

```
#include "header.h"
struct Tarjan {
  vvi &edges;
  int V, counter = 0, C = 0;
  vi n, l;
  vector<bool> vs;
  stack<int> st;
  Tarjan(vvi &e) : edges(e), V(e.size()), n(V, -1), l(V, -1), vs(V, false) {}
  void visit(int u, vi &com) {
    l[u] = n[u] = counter++;
    st.push(u);
    vs[u] = true;
    for (auto &&v : edges[u]) {
      if (n[v] == -1) visit(v, com);
      if (vs[v]) l[u] = min(l[u], l[v]);
    }
    if (l[u] == n[u]) {
      while (true) {
        int v = st.top();
        st.pop();
        vs[v] = false;
        com[v] = C;  //<== ACT HERE
        if (u == v) break;
      }
      C++;
    }
  }
  int find_sccs(vi &com) {  // component indices will be stored in 'com'
    com.assign(V, -1);
    C = 0;
    for (int u = 0; u < V; ++u)
      if (n[u] == -1) visit(u, com);
    return C;
  }
  // scc is a map of the original vertices of the graph to the vertices of the SCC graph, scc_graph is its adjacency list. SCC indices and edges are stored in 'scc' and 'scc_graph'.
  void scc_collapse(vi &scc, vvi &scc_graph) {
    find_sccs(scc);
    scc_graph.assign(C, vi());
    set<pi> rec;  // recorded edges
    for (int u = 0; u < V; ++u) {
      assert(scc[u] != -1);
      for (int v : edges[u]) {
        if (scc[v] == scc[u] ||
          rec.find({scc[u], scc[v]}) != rec.end())
            continue;
        scc_graph[scc[u]].push_back(scc[v]);
        rec.insert({scc[u], scc[v]});
      }
    }
  }
  // The number of edges needed to be added is max(sources.size(), sinks.())
  void findSourcesAndSinks(const vvi &scc_graph, vi &sources, vi &sinks) {
    vi in_degree(C, 0), out_degree(C, 0);
    for (int u = 0; u < C; u++) {
      for (auto v : scc_graph[u]) {
        in_degree[v]++;
        out_degree[u]++;
      }
    }
    for (int i = 0; i < C; ++i) {
      if (in_degree[i] == 0) sources.push_back(i);
      if (out_degree[i] == 0) sinks.push_back(i);
    }
  }
};
```

3.1.13 SCC edges

Prints out the missing edges to make the input digraph strongly connected

```
#include "header.h"
const int N=1e5+10;
int n,a[N],cnt[N],vis[N];
vector<int> hd,tl;
int dfs(int x){
    vis[x]=1;
    if(!vis[a[x]])return vis[x]=dfs(a[x]);
    return vis[x]=x;
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
        cnt[a[i]]++;
    }
    int k=0;
    for(int i=1;i<=n;i++){
        if(!cnt[i]){
            k++;
            hd.push_back(i);
            tl.push_back(dfs(i));
        }
    }
    int tk=k;
    for(int i=1;i<=n;i++){
        if(!vis[i]){
            k++;
            hd.push_back(i);
            tl.push_back(dfs(i));
        }
    }
    if(k==1&&!tk)k=0;
    printf("%d\n",k);
    for(int i=0;i<k;i++)printf("%d %d\n",tl[i],hd[(i+1)%k]);
    return 0;
}
```

3.1.14 Topological sort

```
#include "header.h"
int n; // number of vertices
vvi adj; // adjacency list of graph
vector<bool> visited;
vi ans;
void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u]) dfs(u);
    }
    ans.push_back(v);
}
void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i]) dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

3.1.15 Bellmann-Ford

Same as Dijkstra but allows neg. edges

```
#include "header.h"
// Switch vi and vvpi to vl and vvpl if necessary
void bellmann_ford_extended(vvpi &e, int source, int goal, vi &dist, vb &cyc) {
    dist.assign(e.size(), INF);
    cyc.assign(e.size(), false); // true when u is in a <0 cycle
    dist[source] = 0;

    // Perform n-1 relaxations
    for (int iter = 0; iter < e.size() - 1; ++iter) {
        bool relax = false;
        for (int u = 0; u < e.size(); ++u) {
            if (dist[u] == INF) continue;
            for (auto &edge : e[u]) {
                int v = edge.first, w = edge.second;
                if (dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                    relax = true;
                }
            }
        }
        if (!relax) break;
    }
    // Step to detect any reachable negative cycles
    for (int u = 0; u < e.size(); ++u) {
        if (dist[u] == INF) continue;
        for (auto &edge : e[u]) {
            int v = edge.first, w = edge.second;
            if (dist[u] + w < dist[v]) {
                // If we can still relax, mark the node in the negative cycle
                dist[v] = -INF;
                cyc[v] = true;
            }
        }
    }
    // Propagate neg. cycle detection to all reachable nodes (if necessary)
    bool change = true;
    while (change) {
        change = false;
        for (int u = 0; u < e.size(); ++u) {
            if (!cyc[u]) continue;
            for (auto &edge : e[u]) {
                int v = edge.first;
                if (!cyc[v]) {
                    cyc[v] = true;
                    dist[v] = -INF;
                    change = true;
                }
            }
        }
    }
}
```

3.1.16 Dinic max flow $O(V^2E)$, $O(Ef)$

```
#include "header.h"
using F = ll; using W = ll; // types for flow and weight/cost
struct S{
    const int v;    // neighbour
    const int r;    // index of the reverse edge
    F f;            // current flow
    const F cap;    // capacity
    const W cost;   // unit cost
    S(int v, int ri, F c, W cost = 0) :
        v(v), r(ri), f(0), cap(c), cost(cost) {}
    inline F res() const { return cap - f; }
};
struct FlowGraph : vector<vector<S>> {
    FlowGraph(size_t n) : vector<vector<S>>(n) {}
    void add_edge(int u, int v, F c, W cost = 0){
        auto &t = *this;
        t[u].emplace_back(v, t[v].size(), c, cost);
        t[v].emplace_back(u, t[u].size()-1, c, -cost);
    }
    void add_arc(int u, int v, F c, W cost = 0){
        auto &t = *this;
        t[u].emplace_back(v, t[v].size(), c, cost);
        t[v].emplace_back(u, t[u].size()-1, 0, -cost);
    }
    void clear() { for (auto &E : *this) for (auto &e : E) e.f = 0LL; }
};
struct Dinic{
    FlowGraph &edges; int V,s,t;
    vi l; vector<vector<S>::iterator> its; // levels and iterators
    Dinic(FlowGraph &edges, int s, int t) :
        edges(edges), V(edges.size()), s(s), t(t), l(V,-1), its(V) {}
    ll augment(int u, F c) { // we reuse the same iterators
        if (u == t) return c; ll r = 0LL;
        for(auto &i = its[u]; i != edges[u].end(); i++){
            auto &e = *i;
            if (e.res() && l[u] < l[e.v]) {
                auto d = augment(e.v, min(c, e.res()));
                if (d > 0) { e.f += d; edges[e.v][e.r].f -= d; c -= d;
                    r += d; if (!c) break; }
        }   }
        return r;
    }
    ll run() {
        ll flow = 0, f;
        while(true) {
            fill(l.begin(), l.end(),-1); l[s]=0;
            queue<int> q; q.push(s);
            while(!q.empty()){
                auto u = q.front(); q.pop(); its[u] = edges[u].begin();
                for(auto &&e : edges[u]) if(e.res() && l[e.v]<0)
                    l[e.v] = l[u]+1, q.push(e.v);
            }
            if (l[t] < 0) return flow;
            while ((f = augment(s, INF)) > 0) flow += f;
        }   }
};
```

3.1.17 Edmonds-Karp

(Max) flow algorithm with time $O(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only.

```
#include "header.h"
template<class T> T edmondsKarp(vector<unordered_map<int, T>>&
```

```
    graph, int source, int sink) {
  assert(source != sink);
  T flow = 0;
  vi par(sz(graph)), q = par;

  for (;;) {
    fill(all(par), -1);
    par[source] = 0;
    int ptr = 1;
    q[0] = source;

    rep(i,0,ptr) {
      int x = q[i];
      for (auto e : graph[x]) {
        if (par[e.first] == -1 && e.second > 0) {
          par[e.first] = x;
          q[ptr++] = e.first;
          if (e.first == sink) goto out;
        }
      }
    }
    return flow;
out:
    T inc = numeric_limits<T>::max();
    for (int y = sink; y != source; y = par[y])
      inc = min(inc, graph[par[y]][y]);

    flow += inc;
    for (int y = sink; y != source; y = par[y]) {
      int p = par[y];
      if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
      graph[y][p] += inc;
    }
  }
}
```

3.2 Dynamic Programming

3.2.1 Longest Incr. Subseq.

```
#include "header.h"
template<class T>
vector<T> index_path_lis(vector<T>& nums) {
  int n = nums.size();
  vector<T> sub;
    vector<int> subIndex;
  vector<T> path(n, -1);
  for (int i = 0; i < n; ++i) {
      if (sub.empty() || sub[sub.size() - 1] < nums[i]) {
    path[i] = sub.empty() ? -1 : subIndex[sub.size() - 1];
    sub.push_back(nums[i]);
    subIndex.push_back(i);
      } else {
    int idx = lower_bound(sub.begin(), sub.end(), nums[i]) - sub.begin();
    path[i] = idx == 0 ? -1 : subIndex[idx - 1];
    sub[idx] = nums[i];
    subIndex[idx] = i;
      }
  }
  vector<T> ans;
  int t = subIndex[subIndex.size() - 1];
  while (t != -1) {
      ans.push_back(t);
      t = path[t];
  }
  reverse(ans.begin(), ans.end());
  return ans;
}
// Length only
template<class T>
int length_lis(vector<T> &a) {
  set<T> st;
  typename set<T>::iterator it;
  for (int i = 0; i < a.size(); ++i) {
    it = st.lower_bound(a[i]);
    if (it != st.end()) st.erase(it);
    st.insert(a[i]);
  }
  return st.size();
}
```

3.2.2 0-1 Knapsack

Given a number of coins, calculate all possible distinct sums

```
#include "header.h"
int main() {
  int n;
  vi coins(n); // possible coins to use
  int sum = 0; // their sum of the coins
  vi dp(sum + 1, 0); // dp[x] = 1 if sum x can be made
  dp[0] = 1;
  for (int c = 0; c < n; ++c)
    for (int x = sum; x >= 0; --x)
      if (dp[x]) dp[x + coins[c]] = 1;
}
```

3.2.3 Coin change

Total distinct ways to make sum using n coins of different vals

```
#include "header.h"
int count(vi& coins, int n, int sum) {
    vvi dp(n + 1, vi(sum + 1, 0));
    dp[0][0] = 1;
    for (int i = 1; i <= n; i++) {
        for (int j = 0; j <= sum; j++) {
            // without using the current coin,
            dp[i][j] += dp[i - 1][j];
            // using the current coin
            if ((j - coins[i - 1]) >= 0)
                dp[i][j] += dp[i][j - coins[i - 1]];
        }
    }
    return dp[n][sum];
}
```

3.2.4 Longest common subseq.

Optimization for each unique element appearing k-times

```
#include "../header.h"
#include "../Data Structures/fenwick_tree.cpp"
int lcs(int k, vector<int>& A, vector<int>& B) {
    int lenA = A.size();
    int lenB = B.size();

    // Determine the number of distinct elements from max element in A and B
    int n = max(*max_element(A.begin(), A.end()), *max_element(B.begin(), B.end())) + 1;

    vector<vector<int>> C(n);
    for (int j = 0; j < lenB; ++j) {
        C[B[j]].push_back(j);
    }

    int ans = 0;
    FenwickTree<int> fenwick(lenB + 1);
    for (int i = 0; i < lenA; ++i) {
        int a = A[i];
        for (int j = C[a].size() - 1; j >= 0; --j) {
            int pos = C[a][j];
            int x = fenwick.query(pos) + 1;
            fenwick.update(pos + 1, x);  // Convert to 1-based index
            ans = max(ans, x);
        }
    }
    return ans;
}
```

3.3 Numerical

3.3.1 Template (for this section)

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
```

3.3.2 Polynomial

```
#include "template.cpp"
struct Poly {
  vector<double> a;
  double operator()(double x) const {
    double val = 0;
    for (int i = sz(a); i--;) (val *= x) += a[i];
    return val;
  }
  void diff() {
    rep(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
    a.pop_back();
  }
};
```

3.3.3 Poly Roots

Finds the real roots to a polynomial. $O(n^2 \log(1/\epsilon))$

```
// Usage: polyRoots({{2,-3,1}},-1e9,1e9) = solve x^2-3x+2 = 0
#include "Polynomial.h"
#include "template.cpp"
vector<double> polyRoots(Poly p, double xmin, double xmax) {
  if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
  vector<double> ret;
  Poly der = p;
  der.diff();
  auto dr = polyRoots(der, xmin, xmax);
  dr.push_back(xmin-1);
  dr.push_back(xmax+1);
  sort(all(dr));
  rep(i,0,sz(dr)-1) {
    double l = dr[i], h = dr[i+1];
    bool sign = p(l) > 0;
    if (sign ^ (p(h) > 0)) {
      rep(it,0,60) { // while (h - l > 1e-8)
        double m = (l + h) / 2, f = p(m);
        if ((f <= 0) ^ sign) l = m;
        else h = m;
      }
      ret.push_back((l + h) / 2);
    }
  }
  return ret;
}
```

3.3.4 Golden Section Search

Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum and no local maximum. The maximum error in the result is *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version. $O(\log((b-a)/\epsilon))$

```
/** Usage:
  double func(double x) { return 4+x+.3*x*x; }
  double xmin = gss(-1000,1000,func); */
#include "template.cpp"
// It is important for r to be precise, otherwise we don't necessarily maintain the inequality a < x1 < x2 < b.
double gss(double a, double b, double (*f)(double)) {
  double r = (sqrt(5)-1)/2, eps = 1e-7;
  double x1 = b - r*(b-a), x2 = a + r*(b-a);
  double f1 = f(x1), f2 = f(x2);
  while (b-a > eps)
    if (f1 < f2) { //change to > to find maximum
      b = x2; x2 = x1; f2 = f1;
      x1 = b - r*(b-a); f1 = f(x1);
    } else {
      a = x1; x1 = x2; f1 = f2;
      x2 = a + r*(b-a); f2 = f(x2);
    }
  return a;
}
```

3.3.5 Hill Climbing

Poor man's optimization for unimodal functions.

```
#include "template.cpp"
typedef array<double, 2> P;
template<class F> pair<double, P> hillClimb(P start, F f) {
  pair<double, P> cur(f(start), start);
  for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
    rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
      P p = cur.second;
      p[0] += dx*jmp;
      p[1] += dy*jmp;
      cur = min(cur, make_pair(f(p), p));
    }
  }
  return cur;
}
```

3.3.6 Integration

Simple integration of a function over an interval using Simpson's rule. The error should be proportional to h^4, although in practice you will want to verify that the result is stable to desired precision when epsilon changes.

```
#include "template.cpp"
template<class F>
double quad(double a, double b, F f, const int n = 1000) {
  double h = (b - a) / 2 / n, v = f(a) + f(b);
  rep(i,1,n*2)
    v += f(a + i*h) * (i&1 ? 4 : 2);
  return v * h / 3;
}
```

3.3.7 Integration Adaptive

Fast integration using an adaptive Simpson's rule.

```
/** Usage:
double sphereVolume = quad(-1, 1, [](double x) {
return quad(-1, 1, [\&](double y) {
return quad(-1, 1, [\&](double z) {
return x*x + y*y + z*z < 1; });});}); */
#include "template.cpp"
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6
template <class F>
d rec(F& f, d a, d b, d eps, d S) {
  d c = (a + b) / 2;
  d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
  if (abs(T - S) <= 15 * eps || b - a < 1e-10)
    return T + (T - S) / 15;
  return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
  return rec(f, a, b, eps, S(a, b));
}
```

3.4 Num. Th. / Comb.

3.4.1 Basic stuff

```
#include "header.h"
ll gcd(ll a, ll b) { while (b) { a %= b; swap(a, b);
    } return a; }
ll lcm(ll a, ll b) { return (a / gcd(a, b)) * b; }
ll mod(ll a, ll b) { return ((a % b) + b) % b; }
// Finds x, y s.t. ax + by = d = gcd(a, b).
void extended_euclid(ll a, ll b, ll &x, ll &y, ll &d
    ) {
  ll xx = y = 0;
  ll yy = x = 1;
  while (b) {
    ll q = a / b;
    ll t = b; b = a % b; a = t;
    t = xx; xx = x - q * xx; x = t;
    t = yy; yy = y - q * yy; y = t;
  }
  d = a;
}
// solves ab = 1 (mod n), -1 on failure
ll mod_inverse(ll a, ll n) {
  ll x, y, d;
  extended_euclid(a, n, x, y, d);
  return (d > 1 ? -1 : mod(x, n));
}
// All modular inverses of [1..n] mod P in O(n) time
    .
vi inverses(ll n, ll P) {
  vi I(n+1, 1LL);
  for (ll i = 2; i <= n; ++i)
    I[i] = mod(-(P/i) * I[P%i], P);
  return I;
}
// (a*b)%m
ll mulmod(ll a, ll b, ll m){
  ll x = 0, y=a%m;
  while(b>0){
    if(b&1) x = (x+y)%m;
    y = (2*y)%m, b /= 2;
  }
  return x % m;
}
// Finds b^e % m in O(lg n) time, ensure that b < m
    to avoid overflow!
ll powmod(ll b, ll e, ll m) {
  ll p = e<2 ? 1 : powmod((b*b)%m,e/2,m);
  return e&1 ? p*b%m : p;
}
// Solve ax + by = c, returns false on failure.
bool linear_diophantine(ll a, ll b, ll c, ll &x, ll
    &y) {
  ll d = gcd(a, b);
  if (c % d) {
    return false;
  } else {
    x = c / d * mod_inverse(a / d, b / d);
    y = (c - a * x) / b;
    return true;
  }
}

// Description: Tonelli-Shanks algorithm for modular
     square roots. Finds $x$ s.t. $x^2 = a \pmod p$
    ($-x$ gives the other solution). O(\log^2 p)
    worst case, O(\log p) for most $p$
ll sqrtmod(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(powmod(a, (p-1)/2, p) == 1); // else no
      solution
  if (p % 4 == 3) return powmod(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p %
       8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  /// find a non-square mod p
  while (powmod(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = powmod(a, (s + 1) / 2, p);
  ll b = powmod(a, s, p), g = powmod(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = powmod(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

3.4.2 Mod. exponentiation Or use pow() in python

```
#include "header.h"
ll mod_pow(ll base, ll exp, ll mod) {
  if (mod == 1) return 0;
    if (exp == 0) return 1;
    if (exp == 1) return base;

  ll res = 1;
  base %= mod;
  while (exp) {
    if (exp % 2 == 1) res = (res * base) % mod;
    exp >>= 1;
    base = (base * base) % mod;
  }

  return res % mod;
}
```

3.4.3 GCD Or math.gcd in python, std::gcd in C++

```
#include "header.h"
ll gcd(ll a, ll b) {
  if (a == 0) return b;
  return gcd(b % a, a);
}
```

3.4.4 Sieve of Eratosthenes

```
#include "header.h"
vl primes;
void getprimes(ll n) {  // Up to n (not included)
    vector<bool> p(n, true);
    p[0] = false;
    p[1] = false;
    for(ll i = 0; i < n; i++) {
        if(p[i]) {
            primes.push_back(i);
            for(ll j = i*2; j < n; j+=i) p[j] =
                false;
}}}
```

3.4.5 Fibonacci % prime Starting $1, 1, 2, 3, \ldots$

```
#include "header.h"
const ll MOD = 1000000007;
unordered_map<ll, ll> Fib;
ll fib(ll n) {
    if (n < 2) return 1;
    if (Fib.find(n) != Fib.end()) return Fib[n];
    Fib[n] = (fib((n + 1) / 2) * fib(n / 2) + fib((n
         - 1) / 2) * fib((n - 2) / 2)) % MOD;
    return Fib[n];
}
```

3.4.6 nCk % prime

```
#include "header.h"
ll binom(ll n, ll k) {
    ll ans = 1;
    for(ll i = 1; i <= min(k,n-k); ++i) ans = ans*(n
        +1-i)/i;
    return ans;
}
```

```cpp
ll mod_nCk(ll n, ll k, ll p ){
    ll ans = 1;
    while(n){
        ll np = n%p, kp = k%p;
        if(kp > np) return 0;
        ans *= binom(np,kp);
        n /= p; k /= p;
    }
    return ans;
}
```

3.5 Strings

3.5.1 Z alg. KMP alternative (same complexities)

```cpp
#include "../header.h"
void Z_algorithm(const string &s, vi &Z) {
  Z.assign(s.length(), -1);
  int L = 0, R = 0, n = s.length();
  for (int i = 1; i < n; ++i) {
    if (i > R) {
      L = R = i;
      while (R < n && s[R - L] == s[R]) R++;
      Z[i] = R - L; R--;
    } else if (Z[i - L] >= R - i + 1) {
      L = i;
      while (R < n && s[R - L] == s[R]) R++;
      Z[i] = R - L; R--;
    } else Z[i] = Z[i - L];
  }
}
```

3.5.2 KMP

```cpp
#include "header.h"
void compute_prefix_function(string &w, vi &prefix)
    {
  prefix.assign(w.length(), 0);
  int k = prefix[0] = -1;

  for(int i = 1; i < w.length(); ++i) {
    while(k >= 0 && w[k + 1] != w[i]) k = prefix[k];
    if(w[k + 1] == w[i]) k++;
    prefix[i] = k;
  }
}
vi knuth_morris_pratt(string &s, string &w) {
  int q = -1;
  vi prefix, positions;
  compute_prefix_function(w, prefix);
  for(int i = 0; i < s.length(); ++i) {
    while(q >= 0 && w[q + 1] != s[i]) q = prefix[q];
    if(w[q + 1] == s[i]) q++;
    if(q + 1 == w.length()) {
      // Match at position (i - w.length() + 1)
        positions.push_back(i - w.length() + 1);
      q = prefix[q];
    }
  }
  return positions;
}
```

3.5.3 Aho-Corasick Also can be used as Knuth-Morris-Pratt algorithm

```cpp
#include "header.h"
map<char, int> cti;
int cti_size;
template <int ALPHABET_SIZE, int (*mp)(char)>
struct AC_FSM {
  struct Node {
    int child[ALPHABET_SIZE], failure = 0, match_par
         = -1;
    vi match;
    Node() { for (int i = 0; i < ALPHABET_SIZE; ++i)
         child[i] = -1; }
  };
  vector<Node> a;
  vector<string> &words;
  AC_FSM(vector<string> &words) : words(words) {
    a.push_back(Node());
    construct_automaton();
  }
  void construct_automaton() {
    for (int w = 0, n = 0; w < words.size(); ++w, n
        = 0) {
      for (int i = 0; i < words[w].size(); ++i) {
        if (a[n].child[mp(words[w][i])] == -1) {
          a[n].child[mp(words[w][i])] = a.size();
          a.push_back(Node());
        }
        n = a[n].child[mp(words[w][i])];
      }
      a[n].match.push_back(w);
    }
    queue<int> q;
    for (int k = 0; k < ALPHABET_SIZE; ++k) {
      if (a[0].child[k] == -1) a[0].child[k] = 0;
      else if (a[0].child[k] > 0) {
        a[a[0].child[k]].failure = 0;
        q.push(a[0].child[k]);
      }
    }
    while (!q.empty()) {
      int r = q.front(); q.pop();
      for (int k = 0, arck; k < ALPHABET_SIZE; ++k)
          {
        if ((arck = a[r].child[k]) != -1) {
          q.push(arck);
          int v = a[r].failure;
          while (a[v].child[k] == -1) v = a[v].
              failure;
          a[arck].failure = a[v].child[k];
          a[arck].match_par = a[v].child[k];
          while (a[arck].match_par != -1
              && a[a[arck].match_par].match.empty())
            a[arck].match_par = a[a[arck].match_par
                ].match_par;
        }
      }
    }
  }
  void aho_corasick(string &sentence, vvi &matches){
    matches.assign(words.size(), vi());
    int state = 0, ss = 0;
    for (int i = 0; i < sentence.length(); ++i, ss =
         state) {
      while (a[ss].child[mp(sentence[i])] == -1)
        ss = a[ss].failure;
      state = a[state].child[mp(sentence[i])]
          = a[ss].child[mp(sentence[i])];
      for (ss = state; ss != -1; ss = a[ss].
          match_par)
        for (int w : a[ss].match)
          matches[w].push_back(i + 1 - words[w].
              length());
    }
  }
};
int char_to_int(char c) {
  return cti[c];
}
int main() {
  ll n;
  string line;
  while(getline(cin, line)) {
    stringstream ss(line);
    ss >> n;

    vector<string> patterns(n);
    for (auto& p: patterns) getline(cin, p);

    string text;
    getline(cin, text);

    cti = {}, cti_size = 0;
    for (auto c: text) {
      if (not in(c, cti)) {
        cti[c] = cti_size++;
      }
    }
    for (auto& p: patterns) {
```

```
   for (auto c: p) {
     if (not in(c, cti)) {
       cti[c] = cti_size++;
     }
   }
  }

  vvi matches;
  AC_FSM <128+1, char_to_int> ac_fms(patterns);
  ac_fms.aho_corasick(text, matches);
  for (auto& x: matches) cout << x << endl;
 }

}
```

3.5.4 Long. palin. subs Manacher - $O(n)$

```
#include "header.h"
void manacher(string &s, vi &pal) {
  int n = s.length(), i = 1, l, r;
  pal.assign(2 * n + 1, 0);
  while (i < 2 * n + 1) {
    if ((i&1) && pal[i] == 0) pal[i] = 1;
    l = i / 2 - pal[i] / 2; r = (i-1) / 2 + pal[i] / 2;

    while (l - 1 >= 0 && r + 1 < n && s[l - 1] == s[r + 1])
      --l, ++r, pal[i] += 2;

    for (l = i - 1, r = i + 1; l >= 0 && r < 2 * n + 1; --l, ++r) {
      if (l <= i - pal[i]) break;
      if (l / 2 - pal[l] / 2 > i / 2 - pal[i] / 2)
        pal[r] = pal[l];
      else {  if (l >= 0)
          pal[r] = min(pal[l], i + pal[i] - r);
        break;
      }
    }
    i = r;
} }
```

3.6 Geometry

3.6.1 essentials.cpp

```
#include "../header.h"
using C = ld; // could be ll or ld
constexpr C EPS = 1e-10; // change to 0 for C=ll
struct P {     // may also be used as a 2D vector
  C x, y;
  P(C x = 0, C y = 0) : x(x), y(y) {}
  P operator+ (const P &p) const { return {x + p.x, y + p.y}; }
  P operator- (const P &p) const { return {x - p.x, y - p.y}; }
  P operator* (C c) const { return {x * c, y * c}; }
  P operator/ (C c) const { return {x / c, y / c}; }
  C operator* (const P &p) const { return x*p.x + y*p.y; }
  C operator^ (const P &p) const { return x*p.y - p.x*y; }
  P perp() const { return P{y, -x}; }
  C lensq() const { return x*x + y*y; }
  ld len() const { return sqrt((ld)lensq()); }
  static ld dist(const P &p1, const P &p2) {
    return (p1-p2).len(); }
  bool operator==(const P &r) const {
    return ((*this)-r).lensq() <= EPS*EPS; }
};
C det(P p1, P p2) { return p1^p2; }
C det(P p1, P p2, P o) { return det(p1-o, p2-o); }
C det(const vector<P> &ps) {
  C sum = 0; P prev = ps.back();
  for(auto &p : ps) sum += det(p, prev), prev = p;
  return sum;
}
// Careful with division by two and C=ll
C area(P p1, P p2, P p3) { return abs(det(p1, p2, p3))/C(2); }
C area(const vector<P> &poly) { return abs(det(poly))/C(2); }
int sign(C c){ return (c > C(0)) - (c < C(0)); }
int ccw(P p1, P p2, P o) { return sign(det(p1, p2, o)); }

// Only well defined for C = ld.
P unit(const P &p) { return p / p.len(); }
P rotate(P p, ld a) { return P{p.x*cos(a)-p.y*sin(a), p.x*sin(a)+p.y*cos(a)}; }
```

3.6.2 Two segs. itersec.

```
#include "header.h"
#include "essentials.cpp"
bool intersect(P a1, P a2, P b1, P b2) {
  if (max(a1.x, a2.x) < min(b1.x, b2.x)) return false;
  if (max(b1.x, b2.x) < min(a1.x, a2.x)) return false;
  if (max(a1.y, a2.y) < min(b1.y, b2.y)) return false;
  if (max(b1.y, b2.y) < min(a1.y, a2.y)) return false;
  bool l1 = ccw(a2, b1, a1) * ccw(a2, b2, a1) <= 0;
  bool l2 = ccw(b2, a1, b1) * ccw(b2, a2, b1) <= 0;
  return l1 && l2;
}
```

3.6.3 Convex Hull

```
#include "header.h"
#include "essentials.cpp"
struct ConvexHull {  // O(n lg n) monotone chain.
  size_t n;
  vector<size_t> h, c;  // Indices of the hull are in `h`, ccw.
  const vector<P> &p;
  ConvexHull(const vector<P> &_p) : n(_p.size()), c(n), p(_p) {
    std::iota(c.begin(), c.end(), 0);
    std::sort(c.begin(), c.end(), [this](size_t l, size_t r) -> bool { return p[l].x != p[r].x ? p[l].x < p[r].x : p[l].y < p[r].y; });
    c.erase(std::unique(c.begin(), c.end(), [this](size_t l, size_t r) { return p[l] == p[r]; }), c.end());
    for (size_t s = 1, r = 0; r < 2; ++r, s = h.size()) {
      for (size_t i : c) {
        while (h.size() > s && ccw(p[h.end()[-2]], p[h.end()[-1]], p[i]) <= 0)
          h.pop_back();
        h.push_back(i);
      }
      reverse(c.begin(), c.end());
    }
    if (h.size() > 1) h.pop_back();
  }
  size_t size() const { return h.size(); }
  template <class T, void U(const P &, const P &, const P &, T &)>
  void rotating_calipers(T &ans) {
    if (size() <= 2)
      U(p[h[0]], p[h.back()], p[h.back()], ans);
    else
      for (size_t i = 0, j = 1, s = size(); i < 2 * s; ++i) {
        while (det(p[h[(i + 1) % s]] - p[h[i % s]], p[h[(j + 1) % s]] - p[h[j]]) >= 0)
          j = (j + 1) % s;
        U(p[h[i % s]], p[h[(i + 1) % s]], p[h[j]], ans);
      }
  }
};
// Example: furthest pair of points. Now set ans = 0LL and call
// ConvexHull(pts).rotating_calipers<ll, update>(ans);
```

```cpp
void update(const P &p1, const P &p2, const P &o, ll &ans) {
  ans = max(ans, (ll)max((p1 - o).lensq(), (p2 - o).lensq()));
}
int main() {
  ios::sync_with_stdio(false);  // do not use cout + printf
  cin.tie(NULL);

  int n;
  cin >> n;
  while (n) {
    vector<P> ps;
        int x, y;
    for (int i = 0; i < n; i++) {
            cin >> x >> y;
            ps.push_back({x, y});
        }

        ConvexHull ch(ps);
        cout << ch.h.size() << endl;
        for(auto& p: ch.h) {
            cout << ps[p].x << " " << ps[p].y << endl;
        }
    cin >> n;
  }

  return 0;
}
```

3.7 Other Algorithms

3.7.1 2-sat

```cpp
#include "../header.h"
#include "../Graphs/tarjan.cpp"
struct TwoSAT {
  int n;
  vvi imp; // implication graph
  Tarjan tj;

  TwoSAT(int _n) : n(_n), imp(2 * _n, vi()), tj(imp) { }

  // Only copy the needed functions:
  void add_implies(int c1, bool v1, int c2, bool v2) {
    int u = 2 * c1 + (v1 ? 1 : 0),
      v = 2 * c2 + (v2 ? 1 : 0);
    imp[u].push_back(v);    //  u =>  v
    imp[v^1].push_back(u^1);  // -v => -u
  }
  void add_equivalence(int c1, bool v1, int c2, bool v2) {
    add_implies(c1, v1, c2, v2);
    add_implies(c2, v2, c1, v1);
  }
  void add_or(int c1, bool v1, int c2, bool v2) {
    add_implies(c1, !v1, c2, v2);
  }
  void add_and(int c1, bool v1, int c2, bool v2) {
    add_true(c1, v1); add_true(c2, v2);
  }
  void add_xor(int c1, bool v1, int c2, bool v2) {
    add_or(c1, v1, c2, v2);
    add_or(c1, !v1, c2, !v2);
  }
  void add_true(int c1, bool v1) {
    add_implies(c1, !v1, c1, v1);
  }

  // on true: a contains an assignment.
  // on false: no assignment exists.
  bool solve(vb &a) {
    vi com;
    tj.find_sccs(com);
    for (int i = 0; i < n; ++i)
      if (com[2 * i] == com[2 * i + 1])
        return false;

    vvi bycom(com.size());
    for (int i = 0; i < 2 * n; ++i)
      bycom[com[i]].push_back(i);

    a.assign(n, false);
    vb vis(n, false);
    for(auto &&component : bycom){
      for (int u : component) {
        if (vis[u / 2]) continue;
        vis[u / 2] = true;
        a[u / 2] = (u % 2 == 1);
      }
    }
    return true;
  }
};
```

3.7.2 Finite field For FFT

```cpp
#include "header.h"
#include "../Number Theory/elementary.cpp"
template<ll p,ll w> // prime, primitive root
struct Field { using T = Field; ll x; Field(ll x=0) : x{x} {}
  T operator+(T r) const { return {(x+r.x)%p}; }
  T operator-(T r) const { return {(x-r.x+p)%p}; }
  T operator*(T r) const { return {(x*r.x)%p}; }
  T operator/(T r) const { return (*this)*r.inv(); }
  T inv() const { return {mod_inverse(x,p)}; }
  static T root(ll k) { assert( (p-1)%k==0 );   // (p-1)%k == 0?
    auto r = powmod(w,(p-1)/abs(k),p);      // k-th root of unity
    return k>=0 ? T{r} : T{r}.inv();
  }
  bool zero() const { return x == 0LL; }
};
using F1 = Field<1004535809,3 >;
using F2 = Field<1107296257,10>; // 1<<30 + 1<<25 + 1
using F3 = Field<2281701377,3 >; // 1<<31 + 1<<27 + 1
```

3.7.3 Complex field For FFR

```cpp
#include "header.h"
const double m_pi = M_PIf64x;
struct Complex { using T = Complex; double u,v;
  Complex(double u=0, double v=0) : u{u}, v{v} {}
  T operator+(T r) const { return {u+r.u, v+r.v}; }
  T operator-(T r) const { return {u-r.u, v-r.v}; }
  T operator*(T r) const { return {u*r.u - v*r.v, u*r.v + v*r.u}; }
  T operator/(T r) const {
    auto norm = r.u*r.u+r.v*r.v;
    return {(u*r.u + v*r.v)/norm, (v*r.u - u*r.v)/norm};
  }
  T operator*(double r) const { return T{u*r, v*r}; }
  T operator/(double r) const { return T{u/r, v/r}; }
  T inv() const { return T{1,0}/ *this; }
  T conj() const { return T{u, -v}; }
  static T root(ll k){ return {cos(2*m_pi/k), sin(2*m_pi/k)}; }
  bool zero() const { return max(abs(u), abs(v)) < 1e-6; }
};
```

3.7.4 FFT

```cpp
#include "header.h"
#include "complex_field.cpp"
#include "fin_field.cpp"
void brinc(int &x, int k) {
  int i = k - 1, s = 1 << i;
  x ^= s;
  if ((x & s) != s) {
```

```cpp
    --i; s >>= 1;
    while (i >= 0 && ((x & s) == s))
      x = x &~ s, --i, s >>= 1;
    if (i >= 0) x |= s;
  }
}
using T = Complex;  // using T=F1,F2,F3
vector<T> roots;
void root_cache(int N) {
  if (N == (int)roots.size()) return;
  roots.assign(N, T{0});
  for (int i = 0; i < N; ++i)
    roots[i] = ((i&-i) == i)
      ? T{cos(2.0*m_pi*i/N), sin(2.0*m_pi*i/N)}
      : roots[i&-i] * roots[i-(i&-i)];
}
void fft(vector<T> &A, int p, bool inv = false) {
  int N = 1<<p;
  for(int i = 0, r = 0; i < N; ++i, brinc(r, p))
    if (i < r) swap(A[i], A[r]);
//  Uncomment to precompute roots (for T=Complex). Slower but more precise.
//  root_cache(N);
//          , sh=p-1     , --sh
  for (int m = 2; m <= N; m <<= 1) {
    T w, w_m = T::root(inv ? -m : m);
    for (int k = 0; k < N; k += m) {
      w = T{1};
      for (int j = 0; j < m/2; ++j) {
//          T w = (!inv ? roots[j<<sh] : roots[j<<sh].conj());
        T t = w * A[k + j + m/2];
        A[k + j + m/2] = A[k + j] - t;
        A[k + j] = A[k + j] + t;
        w = w * w_m;
      }
    }
  }
  if(inv){ T inverse = T(N).inv(); for(auto &x : A) x = x*inverse; }
}
// convolution leaves A and B in frequency domain state
// C may be equal to A or B for in-place convolution
void convolution(vector<T> &A, vector<T> &B, vector<T> &C){
  int s = A.size() + B.size() - 1;
  int q = 32 - __builtin_clz(s-1), N=1<<q;  // fails if s=1
  A.resize(N,{}); B.resize(N,{}); C.resize(N,{});
  fft(A, q, false); fft(B, q, false);
  for (int i = 0; i < N; ++i) C[i] = A[i] * B[i];
  fft(C, q, true); C.resize(s);
}
void square_inplace(vector<T> &A) {
  int s = 2*A.size()-1, q = 32 - __builtin_clz(s-1), N=1<<q;
  A.resize(N,{}); fft(A, q, false);
  for(auto &x : A) x = x*x;
  fft(A, q, true); A.resize(s);
}
```

3.7.5 Polyn. inv. div.

```cpp
#include "header.h"
#include "fft.cpp"
vector<T> &rev(vector<T> &A) { reverse(A.begin(), A.end()); return A; }
void copy_into(const vector<T> &A, vector<T> &B, size_t n) {
  std::copy(A.begin(), A.begin()+min({n, A.size(), B.size()}), B.begin());
}
// Multiplicative inverse of A modulo x^n. Requires A[0] != 0!!
vector<T> inverse(const vector<T> &A, int n) {
  vector<T> Ai{A[0].inv()};
  for (int k = 0; (1<<k) < n; ++k) {
    vector<T> As(4<<k, T(0)), Ais(4<<k, T(0));
    copy_into(A, As, 2<<k); copy_into(Ai, Ais, Ai.size());
    fft(As, k+2, false); fft(Ais, k+2, false);
    for (int i = 0; i < (4<<k); ++i) As[i] = As[i]*Ais[i]*Ais[i];
    fft(As, k+2, true); Ai.resize(2<<k, {});
    for (int i = 0; i < (2<<k); ++i) Ai[i] = T(2) * Ai[i] - As[i];
  }
  Ai.resize(n);
  return Ai;
}
// Polynomial division. Returns {Q, R} such that A = QB+R, deg R < deg B.
// Requires that the leading term of B is nonzero.
pair<vector<T>, vector<T>> divmod(const vector<T> &A, const vector<T> &B) {
  size_t n = A.size()-1, m = B.size()-1;
  if (n < m) return {vector<T>(1, T(0)), A};

  vector<T> X(A), Y(B), Q, R;
  convolution(rev(X), Y = inverse(rev(Y), n-m+1), Q);
  Q.resize(n-m+1); rev(Q);

  X.resize(Q.size()), copy_into(Q, X, Q.size());
  Y.resize(B.size()), copy_into(B, Y, B.size());
  convolution(X, Y, X);

  R.resize(m), copy_into(A, R, m);
  for (size_t i = 0; i < m; ++i) R[i] = R[i] - X[i];
  while (R.size() > 1 && R.back().zero()) R.pop_back();
  return {Q, R};
}
vector<T> mod(const vector<T> &A, const vector<T> &B) {
  return divmod(A, B).second;
}
```

3.7.6 Linear recurs.

Given a linear recurrence of the form

$$a_n = \sum_{i=0}^{k-1} c_i a_{n-i-1}$$

this code computes a_n in $O(k \log k \log n)$ time.

```cpp
#include "header.h"
#include "poly.cpp"
// x^k mod f
vector<T> xmod(const vector<T> f, ll k) {
  vector<T> r{T(1)};
  for (int b = 62; b >= 0; --b) {
    if (r.size() > 1)
      square_inplace(r), r = mod(r, f);
    if ((k>>b)&1) {
      r.insert(r.begin(), T(0));
      if (r.size() == f.size()) {
        T c = r.back() / f.back();
        for (size_t i = 0; i < f.size(); ++i)
          r[i] = r[i] - c * f[i];
        r.pop_back();
      }
    }
  }
  return r;
}
// Given A[0,k) and C[0, k), computes the n-th term of:
// A[n] = \sum_i C[i] * A[n-i-1]
T nth_term(const vector<T> &A, const vector<T> &C, ll n) {
  int k = (int)A.size();
  if (n < k) return A[n];

  vector<T> f(k+1, T{1});
  for (int i = 0; i < k; ++i)
    f[i] = T{-1} * C[k-i-1];
  f = xmod(f, n);

  T r = T{0};
  for (int i = 0; i < k; ++i)
    r = r + f[i] * A[i];
```

```
  return r;
}
```

3.7.7 Convolution Precise up to 9e15

```
#include "header.h"
#include "fft.cpp"
void convolution_mod(const vi &A, const vi &B, ll
    MOD, vi &C) {
  int s = A.size() + B.size() - 1; ll m15 = (1LL
      <<15)-1LL;
  int q = 32 - __builtin_clz(s-1), N=1<<q;  // fails
       if s=1
  vector<T> Ac(N), Bc(N), R1(N), R2(N);
  for (size_t i = 0; i < A.size(); ++i) Ac[i] = T{A[
      i]&m15, A[i]>>15};
  for (size_t i = 0; i < B.size(); ++i) Bc[i] = T{B[
      i]&m15, B[i]>>15};
  fft(Ac, q, false); fft(Bc, q, false);
  for (int i = 0, j = 0; i < N; ++i, j = (N-1)&(N-i)
      ) {
    T as = (Ac[i] + Ac[j].conj()) / 2;
    T al = (Ac[i] - Ac[j].conj()) / T{0, 2};
    T bs = (Bc[i] + Bc[j].conj()) / 2;
    T bl = (Bc[i] - Bc[j].conj()) / T{0, 2};
    R1[i] = as*bs + al*bl*T{0,1}, R2[i] = as*bl + al
        *bs;
  }
  fft(R1, q, true); fft(R2, q, true);
  ll p15 = (1LL<<15)%MOD, p30 = (1LL<<30)%MOD; C.
      resize(s);
  for (int i = 0; i < s; ++i) {
    ll l = llround(R1[i].u), m = llround(R2[i].u), h
        = llround(R1[i].v);
    C[i] = (l + m*p15 + h*p30) % MOD;
  }
}
```

3.7.8 Partitions of n Finds all possible partitions of a number

```
#include "header.h"
void printArray(int p[], int n) {
  for (int i = 0; i < n; i++)
    cout << p[i] << " ";
  cout << endl;
}
void printAllUniqueParts(int n) {
  int p[n]; // array to store a partition
  int k = 0;  // idx of last element in a partition
  p[k] = n;

  // The loop stops when the current partition has
      all 1s
  while (true) {
    printArray(p, k + 1);
    int rem_val = 0;
    while (k >= 0 && p[k] == 1) {
      rem_val += p[k];
      k--;
    }
    // no more partitions
    if (k < 0) return;

    p[k]--;
    rem_val++;

    // sorted order is violated (fix)
    while (rem_val > p[k]) {
      p[k + 1] = p[k];
      rem_val = rem_val - p[k];
      k++;
    }

    p[k + 1] = rem_val;
    k++;
  }
}
```

3.7.9 Ternary search Find the smallest i in $[a,b]$ that maximizes $f(i)$, assuming that $f(a) < \cdots < f(i) \geq \cdots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the $<$ marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to $>$, also at (B). $O(\log(b-a))$

```
// Usage: int ind = ternSearch(0,n-1,[\&](int i){
    return a[i];});
#include "../Numerical/template.cpp"
template<class F>
int ternSearch(int a, int b, F f) {
  assert(a <= b);
  while (b - a >= 5) {
    int mid = (a + b) / 2;
    if (f(mid) < f(mid+1)) a = mid; // (A)
    else b = mid+1;
  }
  rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

3.7.10 Hashing Also see `Primes` in `Other Mathematics`. For a proper rolling hash over a string, fix the modulus, and draw the base b uniformly at random from $\{0, 1, \ldots, p-1\}$. Note that when comparing rolling hashes of strings of different lengths, it is useful to hash the empty character to 0, and hash all actual characters to nonzero values. Some primes:

$$10^3 + \{-9, -3, 9, 13\},$$
$$10^6 + \{-17, 3, 33\}, 10^9 + \{7, 9, 21, 33, 87\}$$

```
#include "../header.h"
template <class T>
size_t hash_combine(size_t seed, const T &v) {
  return seed ^ (std::hash<T>()(v) + 0x9e3779b9 + (
      seed << 6) + (seed >> 2));
}
namespace std {
template <typename U, typename V>
struct hash<pair<U, V>> {
  size_t operator()(const pair<U, V> &k) const {
    return hash_combine(hash_combine(0, k.first), k.
        second);
  }
};
}
```

3.8 Other Data Structures

3.8.1 Disjoint set (i.e. union-find)

```
template <typename T>
class DisjointSet {
    typedef T * iterator;
    T *parent, n, *rank;
    public:
        // O(n), assumes nodes are [0, n)
        DisjointSet(T n) {
            this->parent = new T[n];
            this->n = n;
            this->rank = new T[n];
            for (T i = 0; i < n; i++) {
                parent[i] = i;
                rank[i] = 0;
            }
        }

        // O(log n)
        T find_set(T x) {
            if (x == parent[x]) return x;
            return parent[x] = find_set(parent[x]);
        }

```

```
        // O(log n)
        void union_sets(T x, T y) {
            x = this->find_set(x);
            y = this->find_set(y);

            if (x == y) return;
            if (rank[x] < rank[y]) {
                T z = x;
                x = y;
                y = z;
            }
            parent[y] = x;
            if (rank[x] == rank[y]) rank[x]++;
        }
};
```

3.8.2 Fenwick tree

(i.e. BIT) eff. update + prefix sum calc. Can be generalized to arbitrary dimensions by duplicating loops.

```
// #include "header.h"
template < class T >
struct FenwickTree { // use 1 based indices !!!
    int n ; vector <T > tree ;
    FenwickTree ( int n ) : n ( n ) { tree . assign
        ( n + 1 , 0) ; }
    T query ( int l , int r ) { return query ( r ) -
        query ( l - 1) ; }
    T query ( int r ) {
        T s = 0;
        for (; r > 0; r -= ( r & ( - r ) ) ) s +=
            tree [ r ];
        return s ;
    }
    void update ( int i , T v ) {
        for (; i <= n ; i += ( i & ( - i ) ) ) tree
            [ i ] += v ;
    }
};
```

3.8.3 Trie

```
#include "header.h"
const int ALPHABET_SIZE = 26;
inline int mp(char c) { return c - 'a'; }
struct Node {
  Node* ch[ALPHABET_SIZE];
  bool isleaf = false;
  Node() {
    for(int i = 0; i < ALPHABET_SIZE; ++i) ch[i] =
        nullptr;
  }

  void insert(string &s, int i = 0) {
    if (i == s.length()) isleaf = true;
    else {
      int v = mp(s[i]);
      if (ch[v] == nullptr)
        ch[v] = new Node();
      ch[v]->insert(s, i + 1);
    }
  }

  bool contains(string &s, int i = 0) {
    if (i == s.length()) return isleaf;
    else {
      int v = mp(s[i]);
      if (ch[v] == nullptr) return false;
      else return ch[v]->contains(s, i + 1);
    }
  }

  void cleanup() {
    for (int i = 0; i < ALPHABET_SIZE; ++i)
      if (ch[i] != nullptr) {
        ch[i]->cleanup();
        delete ch[i];
      }
  }
};
```

3.8.4 Treap

A binary tree whose nodes contain two values, a key and a priority, such that the key keeps the BST property

```
#include "header.h"
struct Node {
  ll v;
  int sz, pr;
  Node *l = nullptr, *r = nullptr;
  Node(ll val) : v(val), sz(1) { pr = rand(); }
};
int size(Node *p) { return p ? p->sz : 0; }
void update(Node* p) {
  if (!p) return;
  p->sz = 1 + size(p->l) + size(p->r);
  // Pull data from children here
}
void propagate(Node *p) {
  if (!p) return;
  // Push data to children here
}
void merge(Node *&t, Node *l, Node *r) {
  propagate(l), propagate(r);
  if (!l)     t = r;
  else if (!r)  t = l;
  else if (l->pr > r->pr)
      merge(l->r, l->r, r), t = l;
  else  merge(r->l, l, r->l), t = r;
  update(t);
}
void spliti(Node *t, Node *&l, Node *&r, int index)
    {
  propagate(t);
  if (!t) { l = r = nullptr; return; }
  int id = size(t->l);
  if (index <= id) // id \in [index, \infty), so
      move it right
    spliti(t->l, l, t->l, index), r = t;
  else
    spliti(t->r, t->r, r, index - id), l = t;
  update(t);
}
void splitv(Node *t, Node *&l, Node *&r, ll val) {
  propagate(t);
  if (!t) { l = r = nullptr; return; }
  if (val <= t->v) // t->v \in [val, \infty), so
      move it right
    splitv(t->l, l, t->l, val), r = t;
  else
    splitv(t->r, t->r, r, val), l = t;
  update(t);
}
void clean(Node *p) {
  if (p) { clean(p->l), clean(p->r); delete p; }
}
```

3.8.5 Segment tree

```
#include "../header.h"
// example: SegmentTree<int, min> st(n, INT_MAX);
const int& addOp(const int& a, const int& b) {
    static int result;
    result = a + b;
    return result;
}
template <class T, const T&(*op)(const T&, const T&)
    >
struct SegmentTree {
  int n; vector<T> tree; T id;
  SegmentTree(int _n, T _id) : n(_n), tree(2 * n,
      _id), id(_id) { }
  void update(int i, T val) {
    for (tree[i+n] = val, i = (i+n)/2; i > 0; i /=
        2)
      tree[i] = op(tree[2*i], tree[2*i+1]);
  }
  T query(int l, int r) {
    T lhs = T(id), rhs = T(id);
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
```

```cpp
      if (  l&1 ) lhs = op(lhs, tree[l++]);
      if (!(r&1)) rhs = op(tree[--r], rhs);
    }
    return op(l == r ? op(lhs, tree[l]) : lhs, rhs);
  }
};
```

3.8.6 Lazy segment tree Uptimizes range updates

```cpp
#include "../header.h"
using T=int; using U=int; using I=int;    // exclusive right bounds
T t_id; U u_id;
T op(T a, T b){ return a+b; }
void join(U &a, U b){ a+=b; }
void apply(T &t, U u, int x){ t+=x*u; }
T convert(const I &i){ return i; }
struct LazySegmentTree {
  struct Node { int l, r, lc, rc; T t; U u;
    Node(int l, int r, T t=t_id):l(l),r(r),lc(-1),rc(-1),t(t),u(u_id){}
  };
  int N; vector<Node> tree; vector<I> &init;
  LazySegmentTree(vector<I> &init) : N(init.size()), init(init){
    tree.reserve(2*N-1); tree.push_back({0,N}); build(0, 0, N);
  }
  void build(int i, int l, int r) { auto &n = tree[i];
    if (r > l+1) { int m = (l+r)/2;
      n.lc = tree.size();     n.rc = n.lc+1;
      tree.push_back({l,m});    tree.push_back({m,r});
      build(n.lc,l,m);      build(n.rc,m,r);
      n.t = op(tree[n.lc].t, tree[n.rc].t);
    } else n.t = convert(init[l]);
  }
  void push(Node &n, U u){ apply(n.t, u, n.r-n.l); join(n.u,u); }
  void push(Node &n){push(tree[n.lc],n.u);push(tree[n.rc],n.u);n.u=u_id;}
  T query(int l, int r, int i = 0) { auto &n = tree[i];
    if(r <= n.l || n.r <= l) return t_id;
    if(l <= n.l && n.r <= r) return n.t;
    return push(n), op(query(l,r,n.lc),query(l,r,n.rc));
  }
  void update(int l, int r, U u, int i = 0) { auto &n = tree[i];
    if(r <= n.l || n.r <= l) return;
    if(l <= n.l && n.r <= r) return push(n,u);
    push(n); update(l,r,u,n.lc); update(l,r,u,n.rc);
    n.t = op(tree[n.lc].t, tree[n.rc].t);
  }
};
```

3.8.7 Dynamic segment tree Sparse, i.e., larges values, i.e., not storred as an array

```cpp
#include "../header.h"
using T=ll; using U=ll;           // exclusive right bounds
T t_id; U u_id;
T op(T a, T b){ return a+b; }
void join(U &a, U b){ a+=b; }
void apply(T &t, U u, int x){ t+=x*u; }
T part(T t, int r, int p){ return t/r*p; }
struct DynamicSegmentTree {
  struct Node { int l, r, lc, rc; T t; U u;
    Node(int l, int r):l(l),r(r),lc(-1),rc(-1),t(t_id),u(u_id){}
  };
  vector<Node> tree;
  DynamicSegmentTree(int N) { tree.push_back({0,N}); }
  void push(Node &n, U u){ apply(n.t, u, n.r-n.l); join(n.u,u); }
  void push(Node &n){push(tree[n.lc],n.u);push(tree[n.rc],n.u);n.u=u_id;}
  T query(int l, int r, int i = 0) { auto &n = tree[i];
    if(r <= n.l || n.r <= l) return t_id;
    if(l <= n.l && n.r <= r) return n.t;
    if(n.lc < 0) return part(n.t, n.r-n.l, min(n.r,r)-max(n.l,l));
    return push(n), op(query(l,r,n.lc),query(l,r,n.rc));
  }
  void update(int l, int r, U u, int i = 0) { auto &n = tree[i];
    if(r <= n.l || n.r <= l) return;
    if(l <= n.l && n.r <= r) return push(n,u);
    if(n.lc < 0) { int m = (n.l + n.r) / 2;
      n.lc = tree.size();      n.rc = n.lc+1;
      tree.push_back({tree[i].l, m}); tree.push_back({m, tree[i].r});
    }
    push(tree[i]); update(l,r,u,tree[i].lc); update(l,r,u,tree[i].rc);
    tree[i].t = op(tree[tree[i].lc].t, tree[tree[i].rc].t);
  }
};
```

3.8.8 Suffix array

```cpp
#include "../header.h"
struct SuffixArray {
  string s;
  int n;
  vvi P;
  SuffixArray(string &_s) : s(_s), n(_s.length()) { construct(); }
  void construct() {
    P.push_back(vi(n, 0));
    compress();
    vi occ(n + 1, 0), s1(n, 0), s2(n, 0);
    for (int k = 1, cnt = 1; cnt / 2 < n; ++k, cnt *= 2) {
      P.push_back(vi(n, 0));
      fill(occ.begin(), occ.end(), 0);
      for (int i = 0; i < n; ++i)
        occ[i+cnt<n ? P[k-1][i+cnt]+1 : 0]++;
      partial_sum(occ.begin(), occ.end(), occ.begin());
      for (int i = n - 1; i >= 0; --i)
        s1[--occ[i+cnt<n ? P[k-1][i+cnt]+1 : 0]] = i;
      fill(occ.begin(), occ.end(), 0);
      for (int i = 0; i < n; ++i)
        occ[P[k-1][s1[i]]]++;
      partial_sum(occ.begin(), occ.end(), occ.begin());
      for (int i = n - 1; i >= 0; --i)
        s2[--occ[P[k-1][s1[i]]]] = s1[i];
      for (int i = 1; i < n; ++i) {
        P[k][s2[i]] = same(s2[i], s2[i - 1], k, cnt) ? P[k][s2[i - 1]] : i;
      }
    }
  }
  bool same(int i, int j, int k, int l) {
    return P[k - 1][i] == P[k - 1][j]
      && (i + l < n ? P[k - 1][i + l] : -1)
      == (j + l < n ? P[k - 1][j + l] : -1);
  }
  void compress() {
    vi cnt(256, 0);
    for (int i = 0; i < n; ++i) cnt[s[i]]++;
    for (int i = 0, mp = 0; i < 256; ++i)
      if (cnt[i] > 0) cnt[i] = mp++;
    for (int i = 0; i < n; ++i) P[0][i] = cnt[s[i]];
  }
  const vi &get_array() { return P.back(); }
  int lcp(int x, int y) {
    int ret = 0;
    if (x == y) return n - x;
    for (int k = P.size() - 1; k >= 0 && x < n && y < n; --k)
      if (P[k][x] == P[k][y]) {
        x += 1 << k;
```

```
        y += 1 << k;
        ret += 1 << k;
      }
    return ret;
  }
};
```

3.8.9 Suffix tree

```
#include "../header.h"
using T = char;
using M = map<T,int>; // or array<T,ALPHABET_SIZE>
using V = string; // could be vector<T> as well
using It = V::const_iterator;
struct Node{
  It b, e; M edges; int link; // end is exclusive
  Node(It b, It e) : b(b), e(e), link(-1) {}
  int size() const { return e-b; }
};
struct SuffixTree{
  const V &s; vector<Node> t;
  int root,n,len,remainder,llink; It edge;
  SuffixTree(const V &s) : s(s) { build(); }
  int add_node(It b, It e){ return t.push_back({b,e
      }), t.size()-1; }
  int add_node(It b){ return add_node(b,s.end()); }
  void link(int node){ if(llink) t[llink].link =
      node; llink = node; }
  void build(){
    len = remainder =  0; edge = s.begin();
    n = root = add_node(s.begin(), s.begin());
    for(auto i = s.begin(); i != s.end(); ++i){
      ++remainder; llink = 0;
      while(remainder){
        if(len == 0) edge = i;
        if(t[n].edges[*edge] == 0){
          t[n].edges[*edge] = add_node(i); link(n);
        } else {
          auto x = t[n].edges[*edge];
          if(len >= t[x].size()){
            len -= t[x].size(); edge += t[x].size();
                 n = x;
            continue;
          }
          if(*(t[x].b + len) == *i){
            ++len; link(n); break;
          }
          auto split = add_node(t[x].b, t[x].b+len);
          t[n].edges[*edge] = split;
          t[x].b += len;
          t[split].edges[*i] = add_node(i);
          t[split].edges[*t[x].b] = x;
          link(split);
        }
        --remainder;
        if(n == root && len > 0)
          --len, edge = i - remainder + 1;
        else n = t[n].link > 0? t[n].link: root;
      }
    }
  }
};
```

3.8.10 Suffix automaton

```
#include "../header.h"
using T = char; using M = map<T,int>; using V =
    string;
struct Node {   // s: start, len: length, link:
    suffix link, e: edges
  int s, len, link; M e; bool term;       // term:
      terminal node?
  Node(int s, int len, int link=-1):s(s), len(len),
      link(link), term(0) {}
};
struct SuffixAutomaton{
  const V &s; vector<Node> t; int l;  // string;
      tree; last added state
  SuffixAutomaton(const V &s) : s(s) { build(); }
  void build(){
    l = t.size(); t.push_back({0,-1});         //
        root node
    for(auto c : s){
      int p=l, x=t.size(); t.push_back({0,t[l].len +
          1}); // new node
      while(p>=0 && t[p].e[c] == 0) t[p].e[c] = x,
          p= t[p].link;
      if(p<0) t[x].link = 0;             // at root
      else {
        int q = t[p].e[c];             // the c-child
            of q
        if(t[q].len == t[p].len + 1) t[x].link = q;
        else {                    // cloning of q
          int cl = t.size(); t.push_back(t[q]);
          t[cl].len = t[p].len + 1;
          t[cl].s = t[q].s + t[q].len - t[p].len -
              1;
          t[x].link = t[q].link = cl;
          while(p >= 0 && t[p].e.count(c) > 0 && t[p
              ].e[c] == q)
            t[p].e[c] = cl, p = t[p].link;  //
                relink suffix
        }
      }
      l = x;                      // update last
    }
    while(l>=0) t[l].term = true, l = t[l].link;
  }
};
```

3.8.11 UnionFind

```
#include "header.h"
struct UnionFind {
  std::vector<int> par, rank, size;
  int c;
  UnionFind(int n) : par(n), rank(n, 0), size(n, 1),
      c(n) {
    for(int i = 0; i < n; ++i) par[i] = i;
  }
  int find(int i) { return (par[i] == i ? i : (par[i
      ] = find(par[i]))); }
  bool same(int i, int j) { return find(i) == find(j
      ); }
  int get_size(int i) { return size[find(i)]; }
  int count() { return c; }
  int merge(int i, int j) {
    if((i = find(i)) == (j = find(j))) return -1;
    --c;
    if(rank[i] > rank[j]) swap(i, j);
    par[i] = j;
    size[j] += size[i];
    if(rank[i] == rank[j]) rank[j]++;
    return j;
  }
};
```

3.8.12 Indexed set

Similar to set, but allows accessing elements by index using `find_by_order()` in $O(\log n)$

```
#include "../header.h"
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using namespace std;
typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> indexed_set;
```

3.8.13 Order Statistics Tree

A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change `null_type`.$O(\log N)$

```
#include <bits/extc++.h> // !!!!
using namespace __gnu_pbds;
using namespace std;

template<class T>
using Tree = tree<T, null_type, less<T>,rb_tree_tag,
```

```
    tree_order_statistics_node_update>;

void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
  assert(t.order_of_key(10) == 1);
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2
       into t
}
```

3.8.14 Range minimum queries

Answers range minimum queries in constant time after $O(V \log V)$ preproc.

```
template<class T>
struct RMQ {
  vector<vector<T>> jmp;
  RMQ(const vector<T>& V) : jmp(1, V) {
    for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *=
         2, ++k) {
      jmp.emplace_back(sz(V) - pw * 2 + 1);
      rep(j,0,sz(jmp[k]))
        jmp[k][j]=min(jmp[k-1][j],jmp[k-1][j+pw]);
    }
  }
  T query(int a, int b) { // returns min(V[a], ...,
      V[b-1])
    assert(a<b); // or return inf if a == b
    int dep = 31 - __builtin_clz(b-a);
    return min(jmp[dep][a],jmp[dep][b-(1<<dep)]);
  }
};
```

3.8.15 Pareto Front

```
#include "../header.h"
struct pareto_front {
  map<ll, ll> m;
  void insert(ll a, ll b) {
    auto it = m.lower_bound(a);
    if (it != m.end() && it->second >= b)
      return;
    while (!m.empty() && (it = m.upper_bound(a)) !=
         m.begin())
      if ((--it)->first <= a && it->second <= b)
        m.erase(it); else break;
    m[a] = b;
  }
  // max { b | (a, b) \in m, a >= u }, or -LLINF
      otherwise.
  ll max_tail(ll u) {
    auto it = m.lower_bound(u);
    return (it != m.end() ? it->second : -LLINF);
  }
};
```

4 Other Mathematics

4.1 Helpful functions

4.1.1 Euler's Totient Fucntion

$n = p_1^{k_1-1} \cdot (p_1 - 1) \cdot \ldots \cdot p_r^{k_r-1} \cdot (p_r - 1)$, where $p_1^{k_1} \cdot \ldots \cdot p_r^{k_r}$ is the prime factorization of n.

```
# include "header.h"
ll phi(ll n) {  // \Phi(n)
    ll ans = 1;
    for (ll i = 2; i*i <= n; i++) {
        if (n % i == 0) {
            ans *= i-1;
            n /= i;
            while (n % i == 0) {
                ans *= i;
                n /= i;
            }
        }
    }
    if (n > 1) ans *= n-1;
    return ans;
}
vi phis(int n) {  // All \Phi(i) up to n
  vi phi(n + 1, 0LL);
  iota(phi.begin(), phi.end(), 0LL);
  for (ll i = 2LL; i <= n; ++i)
    if (phi[i] == i)
      for (ll j = i; j <= n; j += i)
        phi[j] -= phi[j] / i;
  return phi;
}
```

Formulas $\Phi(n)$ counts all numbers in $1, \ldots, n-1$ coprime to n.
$a^{\varphi(n)} \equiv 1 \mod n$, a and n are coprimes.
$\forall e > \log_2 m :\ n^e \mod m = n^{\Phi(m)+e \mod \Phi(m)} \mod m$.
$\gcd(m,n) = 1 \Rightarrow \Phi(m \cdot n) = \Phi(m) \cdot \Phi(n)$.

4.1.2 Pascal's trinagle

$\binom{n}{k}$ is k-th element in the n-th row, indexing both from 0

```
#include "header.h"
void printPascal(int n) {
    for (int line = 1; line <= n; line++) {
        int C = 1; // used to represent C(line, i)
        for (int i = 1; i <= line; i++) {
            cout << C << " ";
            C = C * (line - i) / i;
        }
        cout << "\n";
    }
}
```

4.2 Theorems and definitions

Subfactorial (Derangements) Permutations of a set such that none of the elements appear in their original position:

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

$$!(0) = 1,\ !n = n \cdot !(n-1) + (-1)^n$$

$$!n = (n-1)(!(n-1) + !(n-2)) = \left[\frac{n!}{e}\right] \tag{1}$$

$$!n = 1 - e^{-1},\ n \to \infty \tag{2}$$

Binomials and other partitionings

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} = \prod_{i=1}^{k} \frac{n-i+1}{i}$$

This last product may be computed incrementally since any product of k' consecutive values is divisibleby $k'!$. Basic identities: The hockeystick identity:

$$\sum_{k=r}^{n} \binom{k}{r} = \binom{n+1}{r+1}$$

or

$$\sum_{k \le n} \binom{r+k}{k} = \binom{r+n+1}{n}$$

Also

$$\sum_{k=0}^{n} \binom{k}{m} = \binom{n+1}{m+1}$$

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$

For $n, m \geq 0$ and p prime: write n, m in base p, i.e. $n = n_k p^k + \cdots + n_1 p + n_0$ and $m = m_k p^k + \ldots m_1 p + m_0$. Then by Lucas theorem we have $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \mod p$, with the convention that $n_i < m_i \implies \binom{n_i}{m_i} = 0$.

Fibonacci (See also number theory section)

$$\sum_{0 \leq k \leq n} \binom{n-k}{k} = F_{n+1}$$

$$F_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n$$

$\sum_{i=1}^{n} F_i = F_{n+2} - 1$, $\sum_{i=1}^{n} F_i^2 = F_n F_{n+1}$

$$\gcd(F_m, F_n) = F_{\gcd(m,n)}$$

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

Bit stuff $a + b = a \oplus b + 2(a\&b) = a|b + a\&b$.
kth bit is set in x iff $x \mod 2^{k-1} \geq 2^k$, or iff $x \mod 2^{k-1} - x \mod 2^k \neq 0$ (i.e. $= 2^k$) It comes handy when you need to look at the bits of the numbers which are pair sums or subset sums etc.
$n \mod 2^i = n\&(2^i - 1)$.
$\forall k: \; 1 \oplus 2 \oplus \ldots \oplus (4k-1) = 0$

4.3 Geometry Formulas

Euler: $\quad 1 + CC = V - E + F$

Pick: $\quad \text{Area} = \text{itr pts} + \frac{\text{bdry pts}}{2} - 1$

Given a non-self-intersecting closed polygon on n vertices, given as (x_i, y_i), its centroid (C_x, C_y) is given as:

$$C_x = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i),$$

$$C_y = \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) = \text{polygon area}$$

Inclusion-Exclusion For appropriate f compute $\sum_{S \subseteq T} (-1)^{|T \setminus S|} f(S)$, or if only the size of S matters, $\sum_{s=0}^{n} (-1)^{n-s} \binom{n}{s} f(s)$. In some contexts we might use Stirling numbers, not binomial coefficients!

Some useful applications:

Graph coloring Let $I(S)$ count the number of independent sets contained in $S \subseteq V$ ($I(\emptyset) = 1$, $I(S) = I(S \setminus v) + I(S \setminus N(v))$). Let $c_k = \sum_{S \subseteq V} (-1)^{|V \setminus S|} I(S)$. Then V is k-colorable iff $v > 0$. Thus we can compute the chromatic number of a graph in $O^*(2^n)$ time.

Burnside's lemma Given a group G acting on a set X, the number of elements in X up to symmetry is

$$\frac{1}{|G|} \sum_{g \in G} |X^g|$$

with X^g the elements of X invariant under g. For example, if $f(n)$ counts "configurations" of some sort of length n, and we want to count them up to rotational symmetry using $G = \mathbb{Z}/n\mathbb{Z}$, then

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

I.e. for coloring with c colors we have $f(k) = k^c$.

Relatedly, in Pólya's enumeration theorem we imagine X as a set of n beads with G permuting the beads (e.g. a necklace, with G all rotations and reflections of the n-cycle, i.e. the dihedral group D_n). Suppose further that we had Y colors, then the number of G-invariant colorings Y^X/G is counted by

$$\frac{1}{|G|} \sum_{g \in G} |Y|^{c(g)}$$

with $c(g)$ counting the number of cycles of g when viewed as a permutation of X. We can generalize this to a weighted version: if the color i can occur exactly r_i times, then this is counted by the coefficient of $t_1^{r_1} \ldots t_n^{r_n}$ in the polynomial

$$Z(t_1, \ldots, t_n) = \frac{1}{|G|} \sum_{g \in G} \prod_{m \geq 1} (t_1^m + \cdots + t_n^m)^{c_m(g)}$$

where $c_m(g)$ counts the number of length m cycles in g acting as a permutation on X. Note we get the original formula by setting all $t_i = 1$. Here Z is the cycle index. Note: you can cleverly deal with even/odd sizes by setting some t_i to -1.

4.4 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k - c_1 x^{k-1} - \cdots - c_k$, there are $d_1, \ldots, d_k$ s.t.

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

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

4.5 Sums

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

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

4.6 Series

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

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

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

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

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

4.7 Quadrilaterals

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

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

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

4.8 Triangles

Side lengths: a, b, c

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

Area:

$$[ABC] = rp = \frac{1}{2} ab \sin\gamma$$

$$= \frac{abc}{4R} = \sqrt{p(p-a)(p-b)(p-c)} = \frac{1}{2}\left|(B-A, C-A)^T\right|$$

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

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

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

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

4.9 Trigonometry

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

$$\sin v + \sin w = 2\sin\frac{v+w}{2}\cos\frac{v-w}{2}$$

$$\cos v + \cos w = 2\cos\frac{v+w}{2}\cos\frac{v-w}{2}$$

$$(V+W)\tan(v-w)/2 = (V-W)\tan(v+w)/2$$

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

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

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

4.10 Combinatorics

Combinations and Permutations
$P(n,r) = \frac{n!}{(n-r)!}$
$C(n,r) = \binom{n}{r} = \frac{n!}{r!(n-r)!}$
$C(n,r) = C(n, n-r)$

4.11 Cycles

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

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

4.12 Labeled unrooted trees

\# on n vertices: n^{n-2}
\# on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
\# with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

4.13 Partition function

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

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

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

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

4.14 Numbers

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

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

Euler-Maclaurin formula for infinite sums:

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

$$\approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

Stirling's numbers **First kind:** $S_1(n,k)$ count permutations on n items with k cycles. $S_1(n,k) = S_1(n-1,k-1) + (n-1)S_1(n-1,k)$ with $S_1(0,0) = 1$. Note:

$$\sum_{k=0}^{n} S_1(n,k)x^k = x(x+1)\ldots(x+n-1)$$

$$\sum_{k=0}^{n} S_1(n,k) = n!$$

$S_1(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$S_1(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$
Second kind: $S_2(n,k)$ count partitions of n distinct elements into exactly k non-empty groups.

$$S_2(n,k) = S_2(n-1,k-1) + kS_2(n-1,k)$$

$$S_2(n,1) = S_2(n,n) = 1$$

$$S_2(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i} i^n$$

Narayana numbers The number of expressions containing n pairs of parentheses, which are correctly matched and which contain k distinct nestings.

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

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. $\pi(j) > j$.

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

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

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

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,

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

Catalan numbers Number of correct bracket sequence consisting of n opening and n closing brackets.
The number of ways to completely parenthesize $n+1$ factors.
The number of triangulations of a convex polygon with $n+2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
The number of ways to connect the $2n$ points on a circle to form n disjoint i.e. non-intersecting chords.

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

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

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

- 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.

4.15 Probability

Stochastic variables
$P(X = r) = C(n, r) \cdot p^r \cdot (1-p)^{n-r}$

Bayes' Theorem $P(B|A) = \frac{P(A|B)P(B)}{P(A)}$
$P(B|A) = \frac{P(A|B)P(B)}{P(A|B)P(B)+P(A|\bar{B})P(\bar{B})}$
$P(B_k|A) = \frac{P(A|B_k)P(B_k)}{P(A|B_1)P(B_1)\cdot\ldots\cdot P(A|B_n)P(B_n)}$

Expectation Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

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

For independent X and Y,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

4.16 Number Theory

Bezout's Theorem

$$a, b \in \mathbb{Z}^+ \implies \exists s, t \in \mathbb{Z} : \gcd(a, b) = sa + tb$$

Bézout's identity For $a \neq$, $b \neq 0$, then $d = gcd(a, b)$ is 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

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

Partial Coprime Divisor Property

$$(\gcd(a, b) = 1) \wedge (a \mid bc) \implies (a \mid c)$$

Coprime Modulus Equivalence Property

$$(\gcd(c, m) = 1) \wedge (ac \equiv bc \mod m) \implies (a \equiv b \mod m)$$

Fermat's Little Theorem

$$(\text{prime}(p)) \wedge (p \nmid a) \implies (a^{p-1} \equiv 1 \mod p)$$

$$(\text{prime}(p)) \implies (a^p \equiv a \mod p)$$

Euler's Theorem

$$a^{\phi(m)-1} \equiv a^{-1} \mod m, \text{ if } \gcd(a, m) = 1$$

$$a^{-1} \equiv a^{m-2} \mod m, \text{ if } m \text{ is prime}$$

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.

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

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 $n < 1e19$.

Mobius Function

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

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d)g(n/d)$$

Other useful formulas/forms:
$\sum_{d|n} \mu(d) = [n = 1]$ (very useful)
$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n)g(d)$
$g(n) = \sum_{1 \le m \le n} f(\left\lfloor \frac{n}{m} \right\rfloor) \Leftrightarrow f(n) = \sum_{1 \le m \le n} \mu(m)g(\left\lfloor \frac{n}{m} \right\rfloor)$

4.17 Discrete distributions

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

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

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

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

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

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

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

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 $\mathrm{Po}(\lambda)$, $\lambda = t\kappa$.

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

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

4.18 Continuous distributions

Uniform distribution If the probability density function is constant between a and b and 0 elsewhere it is $\mathrm{U}(a,b)$, $a < b$.

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

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

Exponential distribution The time between events in a Poisson process is $\mathrm{Exp}(\lambda)$, $\lambda > 0$.

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

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

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

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

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

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