

<b>1 Setup</b>	<b>1</b>	3.1.6 Hungarian algorithm . . . . .	5	3.5.2 KMP . . . . .	11
1.1 header.h . . . . .	1	3.1.7 Suc. shortest path . . . . .	5	3.5.3 Aho-Corasick . . . . .	11
1.2 Bash for c++ compile with header.h . . . .	2	3.1.8 Bipartite check . . . . .	6	3.5.4 Long. palin. subs . . . . .	12
1.3 Bash for run tests c++ . . . . .	2	3.1.9 Find cycle directed . . . . .	6	3.6 Geometry . . . . .	12
1.4 Bash for run tests python . . . . .	2	3.1.10 Find cycle directed . . . . .	6	3.6.1 essentials.cpp . . . . .	12
1.4.1 Aux. helper C++ . . . . .	2	3.1.11 Tarjan's SCC . . . . .	6	3.6.2 Two segs. itersec. . . . .	12
1.4.2 Aux. helper python . . . . .	2	3.1.12 SCC edges . . . . .	7	3.6.3 Convex Hull . . . . .	13
<b>2 Python</b>	<b>2</b>	3.1.13 Find Bridges . . . . .	7	3.7 Other Algorithms . . . . .	13
2.1 Graphs . . . . .	2	3.1.14 Artic. points . . . . .	7	3.7.1 2-sat . . . . .	13
2.1.1 BFS . . . . .	2	3.1.15 Topological sort . . . . .	8	3.7.2 Matrix Solve . . . . .	14
2.1.2 Dijkstra . . . . .	2	3.1.16 Bellmann-Ford . . . . .	8	3.7.3 Matrix Exp. . . . .	14
2.2 Num. Th. / Comb. . . . .	2	3.1.17 Ford-Fulkerson . . . . .	8	3.7.4 Finite field . . . . .	14
2.2.1 nCk % prime . . . . .	2	3.1.18 Dinic max flow . . . . .	8	3.7.5 Complex field . . . . .	14
2.2.2 Sieve of E. . . . .	3	3.2 Dynamic Programming . . . . .	9	3.7.6 FFT . . . . .	14
2.3 Strings . . . . .	3	3.2.1 Longest Incr. Subseq. . . . .	9	3.7.7 Polyn. inv. div. . . . .	15
2.3.1 LCS . . . . .	3	3.2.2 0-1 Knapsack . . . . .	9	3.7.8 Linear recurs. . . . .	15
2.3.2 KMP . . . . .	3	3.2.3 Coin change . . . . .	9	3.7.9 Convolution . . . . .	16
2.3.3 Edit distance . . . . .	3	3.3 Trees . . . . .	10	3.7.10 Partitions of $n$ . . . . .	16
2.4 Other Algorithms . . . . .	3	3.3.1 Tree diameter . . . . .	10	3.8 Other Data Structures . . . . .	16
2.4.1 Rotate matrix . . . . .	3	3.3.2 Tree Node Count . . . . .	10	3.8.1 Disjoint set . . . . .	16
2.5 Other Data Structures . . . . .	3	3.4 Num. Th. / Comb. . . . .	10	3.8.2 Fenwick tree . . . . .	16
2.5.1 Segment Tree . . . . .	3	3.4.1 Basic stuff . . . . .	10	3.8.3 Fenwick2d tree . . . . .	17
2.5.2 Trie . . . . .	4	3.4.2 Mod. exponentiation . . . . .	10	3.8.4 Trie . . . . .	17
<b>3 C++</b>	<b>4</b>	3.4.3 GCD . . . . .	10	3.8.5 Treap . . . . .	17
3.1 Graphs . . . . .	4	3.4.4 Sieve of Eratosthenes . . . . .	10	<b>4 Other Mathematics</b>	<b>17</b>
3.1.1 BFS . . . . .	4	3.4.5 Fibonacci % prime . . . . .	11	4.1 Helpful functions . . . . .	17
3.1.2 DFS . . . . .	4	3.4.6 nCk % prime . . . . .	11	4.1.1 Euler's Totient Fucntion . . . . .	17
3.1.3 Dijkstra . . . . .	4	3.4.7 Chin. rem. th. . . . .	11	4.1.2 Pascal's trinagle . . . . .	18
3.1.4 Floyd-Warshall . . . . .	5	3.5 Strings . . . . .	11	4.2 Theorems and definitions . . . . .	19
3.1.5 Kruskal . . . . .	5	3.5.1 Z alg. . . . .	11	4.3 Geometry Formulas . . . . .	19

## 1 Setup

### 1.1 header.h

```

1 #pragma once // Delete this when copying this
  file
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 #define ll long long
6 #define ull unsigned ll
7 #define ld long double
8 #define pl pair<ll, ll>
9 #define pi pair<int, int> // use pl where
  possible/necessary

```

```

10 #define vl vector<ll>
11 #define vi vector<int> // change to vl where
    possible/necessary
12 #define vb vector<bool>
13 #define vvi vector<vi>
14 #define vvl vector<vl>
15 #define vpl vector<pl>
16 #define vpi vector<pi>
17 #define vld vector<ld>
18 #define vvpi vector<vpi>
19 #define in_fast(el, cont) (cont.find(el) != cont.
    end())
20 #define in(el, cont) (find(cont.begin(), cont.end
    (), el) != cont.end())
21
22 constexpr int INF = 2000000010;
23 constexpr ll LLINF = 900000000000000010LL;

```

```

24
25 template <typename T, template <typename ELEM,
    typename ALLOC = std::allocator<ELEM> > class
    Container>
26 std::ostream& operator<< (std::ostream& o, const
    Container<T>& container) {
27     typename Container<T>::const_iterator beg =
    container.begin();
28     if (beg != container.end()) {
29         o << *beg++;
30         while (beg != container.end()) {
31             o << " " << *beg++;
32         }
33     }
34     return o;
35 }
36

```

---

```

37 // int main() {
38 //   ios::sync_with_stdio(false); // do not use
39 //   cout + printf
40 //   cin.tie(NULL);
41 //   cout << fixed << setprecision(12);
42 //   return 0;
43 // }

```

---

## 1.2 Bash for c++ compile with header.h

---

```

1 #!/bin/bash
2 if [ $# -ne 1 ];then echo "Usage: $0 <input_file
>"; exit 1;fi
3 f="$1";d=code/o=a.out
4 [ -f $d/$f ] || { echo "Input file not found: $f
"; exit 1; }
5 g++ -I$d $d/$f -o $o && echo "Compilation
successful. Executable '$o' created." || echo
"Compilation failed."

```

---

## 1.3 Bash for run tests c++

---

```

1 g++ $1/$1.cpp -o $1/$1.out
2 for file in $1/*.in; do diff <($1/$1.out < "$file
") "${file%.in}.ans"; done

```

---

## 1.4 Bash for run tests python

---

```

1 for file in $1/*.in; do diff <(python3 $1/$1.py <
"$file") "${file%.in}.ans"; done

```

---

### 1.4.1 Aux. helper C++

---

```

1 #include "header.h"
2
3 int main() {
4     // Read in a line including white space
5     string line;
6     getline(cin, line);
7     // When doing the above read numbers as
8     // follows:
9     int n;
10    getline(cin, line);

```

---



---

```

10    stringstream ss(line);
11    ss >> n;
12
13    // Count the number of 1s in binary
14    // represnatation of a number
15    ull number;
16    __builtin_popcountll(number);

```

---

### 1.4.2 Aux. helper python

---

```

1 from functools import lru_cache
2
3 # Read until EOF
4 while True:
5     try:
6         pattern = input()
7     except EOFError:
8         break
9
10 @lru_cache(maxsize=None)
11 def smth_memoi(i, j, s):
12     # Example in-built cache
13     return "sol"

```

---

## 2 Python

### 2.1 Graphs

#### 2.1.1 BFS

---

```

1 from collections import deque
2 def bfs(g, roots, n):
3     q = deque(roots)
4     explored = set(roots)
5     distances = [float("inf")]*n
6     distances[0][0] = 0
7
8     while len(q) != 0:
9         node = q.popleft()
10        if node in explored: continue
11        explored.add(node)
12        for neigh in g[node]:
13            if neigh not in explored:
14                q.append(neigh)
15                distances[neigh] = distances[node]
16                + 1
17
18    return distances

```

---

#### 2.1.2 Dijkstra

---

```

1 from heapq import *
2 def dijkstra(n, root, g): # g = {node: (cost,
3     # neigh)}
4     dist = [float("inf")]*n
5     dist[root] = 0
6     prev = [-1]*n
7
8     pq = [(0, root)]
9     heapify(pq)
10    visited = set([])
11
12    while len(pq) != 0:
13        _, node = heappop(pq)
14
15        if node in visited: continue
16        visited.add(node)
17
18        # In case of disconnected graphs
19        if node not in g:
20            continue
21
22        for cost, neigh in g[node]:
23            alt = dist[node] + cost
24            if alt < dist[neigh]:
25                dist[neigh] = alt
26                prev[neigh] = node
27                heappush(pq, (alt, neigh))
28
29    return dist

```

---

### 2.2 Num. Th. / Comb.

#### 2.2.1 nCk % prime

---

```

1 # Note: p must be prime and k < p
2 def fermat_binom(n, k, p):
3     if k > n:
4         return 0
5     # calculate numerator
6     num = 1
7     for i in range(n-k+1, n+1):
8         num *= i % p
9     num %= p
10    # calculate denominator
11    denom = 1
12    for i in range(1, k+1):
13        denom *= i % p
14    denom %= p
15    # numerator * denominator^(p-2) (mod p)
16    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

---

```

1 MAX_SIZE = 10**8+1
2 isprime = [True] * MAX_SIZE
3 prime = []
4 SPF = [None] * (MAX_SIZE)
5
6 def manipulated_seive(N): # Up to N (not
    included)
7     isprime[0] = isprime[1] = False
8     for i in range(2, N):
9         if isprime[i] == True:
10             prime.append(i)
11             SPF[i] = i
12             j = 0
13             while (j < len(prime) and
14                    i * prime[j] < N and
15                    prime[j] <= SPF[i]):
16                 isprime[i * prime[j]] = False
17                 SPF[i * prime[j]] = prime[j]
18                 j += 1

```

---

## 2.3 Strings

### 2.3.1 LCS

---

```

1 def longestCommonSubsequence(text1, text2): # 0(
    m*n) time, 0(m) space
2     n = len(text1)
3     m = len(text2)
4
5     # Initializing two lists of size m
6     prev = [0] * (m + 1)
7     cur = [0] * (m + 1)
8
9     for idx1 in range(1, n + 1):
10         for idx2 in range(1, m + 1):
11             # If characters are matching
12             if text1[idx1 - 1] == text2[idx2 -
13                 1]:
14                 cur[idx2] = 1 + prev[idx2 - 1]
15             else:
16                 # If characters are not matching
17                 cur[idx2] = max(cur[idx2 - 1],
18                                prev[idx2])
19
20         prev = cur.copy()
21
22     return cur[m]

```

---

### 2.3.2 KMP

---

```

1 class KMP:
2     def partial(self, pattern):
3         """ Calculate partial match table: String
4             -> [Int]"""
5         ret = [0]
6         for i in range(1, len(pattern)):
7             j = ret[i - 1]
8             while j > 0 and pattern[j] != pattern
9                 [i]: j = ret[j - 1]
10            ret.append(j + 1 if pattern[j] ==
11                pattern[i] else j)
12        return ret
13
14    def search(self, T, P):
15        """KMP search main algorithm: String ->
16            String -> [Int]
17        Return all the matching position of
18        pattern string P in T"""
19        partial, ret, j = self.partial(P), [], 0
20        for i in range(len(T)):
21            while j > 0 and T[i] != P[j]: j =
22                partial[j - 1]
23            if T[i] == P[j]: j += 1
24            if j == len(P):
25                ret.append(i - (j - 1))
26                j = partial[j - 1]
27        return ret

```

---

### 2.3.3 Edit distance

---

```

1 def editDistance(str1, str2):
2     # Get the lengths of the input strings
3     m = len(str1)
4     n = len(str2)
5
6     # Initialize a list to store the current row
7     curr = [0] * (n + 1)
8
9     # Initialize the first row with values from 0
10    to n
11    for j in range(n + 1):
12        curr[j] = j
13
14    # Initialize a variable to store the previous
15    value
16    previous = 0
17
18    # Loop through the rows of the dynamic
19    programming matrix
20    for i in range(1, m + 1):
21        # Store the current value at the beginning of
22        the row
23        previous = curr[0]
24        curr[0] = i

```

---



---

```

21
22    # Loop through the columns of the dynamic
23    programming matrix
24    for j in range(1, n + 1):
25        # Store the current value in a temporary
26        variable
27        temp = curr[j]
28
29        # Check if the characters at the current
30        positions in str1 and str2 are the same
31        if str1[i - 1] == str2[j - 1]:
32            curr[j] = previous
33        else:
34            # Update the current cell with the
35            minimum of the three adjacent cells
36            curr[j] = 1 + min(previous, curr[j - 1],
37                             curr[j])
38
39        # Update the previous variable with the
40        temporary value
41        previous = temp
42
43    # The value in the last cell represents the
44    minimum number of operations
45    return curr[n]

```

---

## 2.4 Other Algorithms

### 2.4.1 Rotate matrix

---

```

1 def rotate_matrix(m):
2     return [[m[j][i] for j in range(len(m))] for
3             i in range(len(m[0])-1,-1,-1)]

```

---

## 2.5 Other Data Structures

### 2.5.1 Segment Tree

---

```

1 N = 100000 # limit for array size
2 tree = [0] * (2 * N) # Max size of tree
3
4 def build(arr, n): # function to build the tree
5     # insert leaf nodes in tree
6     for i in range(n):
7         tree[n + i] = arr[i]
8
9     # build the tree by calculating parents
10    for i in range(n - 1, 0, -1):
11        tree[i] = tree[i << 1] + tree[i << 1 | 1]
12
13 def updateTreeNode(p, value, n): # function to
14     update a tree node

```

---

```

14 # set value at position p
15 tree[p + n] = value
16 p = p + n
17
18 i = p # move upward and update parents
19 while i > 1:
20     tree[i >> 1] = tree[i] + tree[i ^ 1]
21     i >>= 1
22
23 def query(l, r, n): # function to get sum on
24     interval [l, r]
25     res = 0
26     # loop to find the sum in the range
27     l += n
28     r += n
29     while l < r:
30         if l & 1:
31             res += tree[l]
32             l += 1
33         if r & 1:
34             r -= 1
35             res += tree[r]
36         l >>= 1
37         r >>= 1
38     return res

```

### 2.5.2 Trie

```

1 class TrieNode:
2     def __init__(self):
3         self.children = [None]*26
4         self.isEndOfWord = False
5
6 class Trie:
7     def __init__(self):
8         self.root = self.getNode()
9
10    def getNode(self):
11        return TrieNode()
12
13    def _charToIndex(self, ch):
14        return ord(ch)-ord('a')
15
16
17    def insert(self, key):
18        pCrawl = self.root
19        length = len(key)
20        for level in range(length):
21            index = self._charToIndex(key[level])
22            if not pCrawl.children[index]:
23                pCrawl.children[index] = self.
24                    getNode()
25            pCrawl = pCrawl.children[index]
26            pCrawl.isEndOfWord = True

```

```

26
27 def search(self, key):
28     pCrawl = self.root
29     length = len(key)
30     for level in range(length):
31         index = self._charToIndex(key[level])
32         if not pCrawl.children[index]:
33             return False
34         pCrawl = pCrawl.children[index]
35
36     return pCrawl.isEndOfWord

```

## 3 C++

### 3.1 Graphs

#### 3.1.1 BFS

```

1 #include "header.h"
2 #define graph unordered_map<ll, unordered_set<ll
3 >>
4 vi bfs(int n, graph& g, vi& roots) {
5     vi parents(n+1, -1); // nodes are 1..n
6     unordered_set<int> visited;
7     queue<int> q;
8     for (auto x: roots) {
9         q.emplace(x);
10        visited.insert(x);
11    }
12    while (not q.empty()) {
13        int node = q.front();
14        q.pop();
15
16        for (auto neigh: g[node]) {
17            if (not in(neigh, visited)) {
18                parents[neigh] = node;
19                q.emplace(neigh);
20                visited.insert(neigh);
21            }
22        }
23    }
24    return parents;
25
26 vi reconstruct_path(vi parents, int start, int
27 goal) {
28     vi path;
29     int curr = goal;
30     while (curr != start) {
31         path.push_back(curr);
32         if (parents[curr] == -1) return vi(); //
33             No path, empty vi
34         curr = parents[curr];
35     }
36 }

```

```

33 path.push_back(start);
34 reverse(path.begin(), path.end());
35 return path;
36 }

```

#### 3.1.2 DFS Cycle detection / removal

```

1 #include "header.h"
2 void removeCyc(ll node, unordered_map<ll, vector<
3 pair<ll, ll>>>& neighs, vector<bool>& visited
4 ,
5 vector<bool>& recStack, vector<ll>& ans) {
6     if (!visited[node]) {
7         visited[node] = true;
8         recStack[node] = true;
9         auto it = neighs.find(node);
10        if (it != neighs.end()) {
11            for (auto util: it->second) {
12                ll nnode = util.first;
13                if (recStack[nnode]) {
14                    ans.push_back(util.second);
15                } else if (!visited[nnode]) {
16                    removeCyc(nnode, neighs,
17                        visited, recStack, ans);
18                }
19            }
20        }
21        recStack[node] = false;
22    }
23 }

```

#### 3.1.3 Dijkstra

```

1 #include "header.h"
2 vector<int> dijkstra(int n, int root, map<int,
3 vector<pair<int, int>>>& g) {
4     vector<int> dist(n, INF);
5     unordered_set<int> visited;
6     priority_queue<pair<int, int>> pq;
7     dist[root] = 0;
8     pq.push({0, root});
9     while (!pq.empty()) {
10        int node = pq.top().second;
11        int d = -pq.top().first;
12        pq.pop();
13
14        if (in(node, visited)) continue;
15        visited.insert(node);
16
17        for (auto e: g[node]) {
18            int neigh = e.first;
19            int cost = e.second;

```

```

19         if (dist[neigh] > dist[node] + cost)
20             {
21                 dist[neigh] = dist[node] + cost;
22                 pq.push({-dist[neigh], neigh});
23             }
24     }
25     return dist;
26 }

```

### 3.1.4 Floyd-Warshall

```

1 #include "header.h"
2 // g[i][j] = inf if not path from i to j
3 // if g[i][i] < 0, i is contained in a negative
  cycle
4 void warshall(vvl g) {
5     for (int i=0; i<g.size(); ++i) {
6         for (int j=0; j<g.size(); ++j) {
7             for (int k=0; k<g.size(); ++k) {
8                 if (g[i][k] < LLINF and g[k][j] <
9                     LLINF and g[i][j] > g[i][k]
10                    + g[k][j]) {
11                     g[i][j] = g[i][k] + g[k][j];
12                 }
13             }
14         }
15     }
16 }

```

### 3.1.5 Kruskal Minimum spanning tree of undirected weighted graph

```

1 #include "header.h"
2 #include "disjoint_set.h"
3 // O(E log E)
4 pair<set<pair<ll, ll>>, ll> kruskal(vector<tuple<
5     <ll, ll, ll>>& edges, ll n) {
6     set<pair<ll, ll>> ans;
7     ll cost = 0;
8
9     sort(edges.begin(), edges.end());
10    DisjointSet<ll> fs(n);
11
12    ll dist, i, j;
13    for (auto edge: edges) {
14        dist = get<0>(edge);
15        i = get<1>(edge);
16        j = get<2>(edge);
17
18        if (fs.find_set(i) != fs.find_set(j)) {
19            fs.union_sets(i, j);
20            ans.insert({i, j});
21            cost += dist;
22        }
23    }
24 }

```

```

23     return pair<set<pair<ll, ll>>, ll> {ans, cost};
24 }

```

### 3.1.6 Hungarian algorithm

```

1 #include "header.h"
2
3 template <class T> bool ckmin(T &a, const T &b) {
4     return b < a ? a = b, 1 : 0; }
5
6 /**
7  * Given J jobs and W workers (J <= W), computes
8  * the minimum cost to assign each
9  * prefix of jobs to distinct workers.
10 * @tparam T a type large enough to represent
11 * integers on the order of J *
12 * max(|C|)
13 * @param C a matrix of dimensions JxW such that
14 * C[j][w] = cost to assign j-th
15 * job to w-th worker (possibly negative)
16 *
17 * @return a vector of length J, with the j-th
18 * entry equaling the minimum cost
19 * to assign the first (j+1) jobs to distinct
20 * workers
21 */
22 template <class T> vector<T> hungarian(const
23     vector<vector<T>> &C) {
24     const int J = (int)size(C), W = (int)size(C[0]);
25     assert(J <= W);
26     // job[w] = job assigned to w-th worker, or
27     // -1 if no job assigned
28     // note: a W-th worker was added for
29     // convenience
30     vector<int> job(W + 1, -1);
31     vector<T> ys(J), yt(W + 1); // potentials
32     // -yt[W] will equal the sum of all deltas
33     vector<T> answers;
34     const T inf = numeric_limits<T>::max();
35     for (int j_cur = 0; j_cur < J; ++j_cur) { //
36         assign j_cur-th job
37         int w_cur = W;
38         job[w_cur] = j_cur;
39         // min reduced cost over edges from Z to
40         // worker w
41         vector<T> min_to(W + 1, inf);
42         vector<int> prv(W + 1, -1); // previous
43         // worker on alternating path
44         vector<bool> in_Z(W + 1); // whether
45         // worker is in Z
46         while (job[w_cur] != -1) { // runs at
47             most j_cur + 1 times
48             in_Z[w_cur] = true;
49         }
50     }
51 }

```

```

34     const int j = job[w_cur];
35     T delta = inf;
36     int w_next;
37     for (int w = 0; w < W; ++w) {
38         if (!in_Z[w]) {
39             if (ckmin(min_to[w], C[j][w]
40                 - ys[j] - yt[w]))
41                 prv[w] = w_cur;
42             if (ckmin(delta, min_to[w]))
43                 w_next = w;
44         }
45     }
46     // delta will always be non-negative,
47     // except possibly during the first
48     // time this loop runs
49     // if any entries of C[j_cur] are
50     // negative
51     for (int w = 0; w <= W; ++w) {
52         if (in_Z[w]) ys[job[w]] += delta,
53             yt[w] -= delta;
54         else min_to[w] -= delta;
55     }
56     w_cur = w_next;
57 }
58 // update assignments along alternating
59 // path
60 for (int w; w_cur != W; w_cur = w) job[
61     w_cur] = job[w = prv[w_cur]];
62 answers.push_back(-yt[W]);
63 return answers;
64 }

```

### 3.1.7 Suc. shortest path Calculates max flow, min cost

```

1 #include "header.h"
2 // map<node, map<node, pair<cost, capacity>>>
3 #define graph unordered_map<int, unordered_map<
4     int, pair<ld, int>>>
5 graph g;
6 const ld inf = 1e60; // Change if necessary
7 ld fill(int n, vld& potential) { // Finds max
8     flow, min cost
9     priority_queue<pair<ld, int>> pq;
10    vector<bool> visited(n+2, false);
11    vi parent(n+2, 0);
12    vld dist(n+2, inf);
13    dist[0] = 0.1;
14    pq.emplace(make_pair(0.1, 0));
15    while (not pq.empty()) {
16        int node = pq.top().second;
17        pq.pop();
18        if (visited[node]) continue;
19    }
20 }

```

```

17 visited[node] = true;
18 for (auto& x : g[node]) {
19     int neigh = x.first;
20     int capacity = x.second.second;
21     ld cost = x.second.first;
22     if (capacity and not visited[neigh]) {
23         ld d = dist[node] + cost + potential[node
24             ] - potential[neigh];
25         if (d + 1e-10l < dist[neigh]) {
26             dist[neigh] = d;
27             pq.emplace(make_pair(-d, neigh));
28             parent[neigh] = node;
29         }
30     }
31     for (int i = 0; i < n+2; i++) {
32         potential[i] = min(infty, potential[i] + dist
33             [i]);
34     }
35     if (not parent[n+1]) return infty;
36     ld ans = 0.1;
37     for (int x = n+1; x; x=parent[x]) {
38         ans += g[parent[x]][x].first;
39         g[parent[x]][x].second--;
40         g[x][parent[x]].second++;
41     }
42     return ans;
43 }

```

### 3.1.8 Bipartite check

```

1 #include "header.h"
2 int main() {
3     int n;
4     vvi adj(n);
5
6     vi side(n, -1); // will have 0's for one
7                     // side 1's for other side
8     bool is_bipartite = true; // becomes false
9     if not bipartite
10     queue<int> q;
11     for (int st = 0; st < n; ++st) {
12         if (side[st] == -1) {
13             q.push(st);
14             side[st] = 0;
15             while (!q.empty()) {
16                 int v = q.front();
17                 q.pop();
18                 for (int u : adj[v]) {
19                     if (side[u] == -1) {
20                         side[u] = side[v] ^ 1;
21                         q.push(u);
22                     } else {
23                         is_bipartite &= side[u]
24                             != side[v];
25                     }
26                 }
27             }
28         }
29     }
30 }

```

```

22     }
23 }

```

### 3.1.9 Find cycle directed

```

1 #include "header.h"
2 int n;
3 const int mxN = 2e5+5;
4 vvi adj(mxN);
5 vector<char> color;
6 vi parent;
7 int cycle_start, cycle_end;
8 bool dfs(int v) {
9     color[v] = 1;
10    for (int u : adj[v]) {
11        if (color[u] == 0) {
12            parent[u] = v;
13            if (dfs(u)) return true;
14        } else if (color[u] == 1) {
15            cycle_end = v;
16            cycle_start = u;
17            return true;
18        }
19    }
20    color[v] = 2;
21    return false;
22 }
23 void find_cycle() {
24     color.assign(n, 0);
25     parent.assign(n, -1);
26     cycle_start = -1;
27     for (int v = 0; v < n; v++) {
28         if (color[v] == 0 && dfs(v)) break;
29     }
30     if (cycle_start == -1) {
31         cout << "Acyclic" << endl;
32     } else {
33         vector<int> cycle;
34         cycle.push_back(cycle_start);
35         for (int v = cycle_end; v != cycle_start;
36             v = parent[v])
37             cycle.push_back(v);
38         cycle.push_back(cycle_start);
39         reverse(cycle.begin(), cycle.end());
40
41         cout << "Cycle Found: ";
42         for (int v : cycle) cout << v << " ";
43         cout << endl;
44     }
45 }

```

### 3.1.10 Find cycle directed

```

1 #include "header.h"
2 int n;
3 const int mxN = 2e5 + 5;
4 vvi adj(mxN);
5 vector<bool> visited;
6 vi parent;
7 int cycle_start, cycle_end;
8 bool dfs(int v, int par) { // passing vertex and
9     // its parent vertex
10    visited[v] = true;
11    for (int u : adj[v]) {
12        if (u == par) continue; // skipping edge
13        // to parent vertex
14        if (visited[u]) {
15            cycle_end = v;
16            cycle_start = u;
17            return true;
18        }
19        parent[u] = v;
20        if (dfs(u, parent[u]))
21            return true;
22    }
23    return false;
24 }
25 void find_cycle() {
26     visited.assign(n, false);
27     parent.assign(n, -1);
28     cycle_start = -1;
29     for (int v = 0; v < n; v++) {
30         if (!visited[v] && dfs(v, parent[v]))
31             break;
32     }
33     if (cycle_start == -1) {
34         cout << "Acyclic" << endl;
35     } else {
36         vector<int> cycle;
37         cycle.push_back(cycle_start);
38         for (int v = cycle_end; v != cycle_start;
39             v = parent[v])
40             cycle.push_back(v);
41         cycle.push_back(cycle_start);
42         cout << "Cycle Found: ";
43         for (int v : cycle) cout << v << " ";
44         cout << endl;
45     }
46 }

```

### 3.1.11 Tarjan's SCC

```

1 #include "header.h"
2
3 struct Tarjan {
4     vvi &edges;
5     int V, counter = 0, C = 0;

```

```

6  vi n, l;
7  vector<bool> vs;
8  stack<int> st;
9  Tarjan(vvi &e) : edges(e), V(e.size()), n(V,
    -1), l(V, -1), vs(V, false) {}
10 void visit(int u, vi &com) {
11     l[u] = n[u] = counter++;
12     st.push(u);
13     vs[u] = true;
14     for (auto &&v : edges[u]) {
15         if (n[v] == -1) visit(v, com);
16         if (vs[v]) l[u] = min(l[u], l[v]);
17     }
18     if (l[u] == n[u]) {
19         while (true) {
20             int v = st.top();
21             st.pop();
22             vs[v] = false;
23             com[v] = C; //<== ACT HERE
24             if (u == v) break;
25         }
26         C++;
27     }
28 }
29 int find_sccs(vi &com) { // component indices
    will be stored in 'com'
30     com.assign(V, -1);
31     C = 0;
32     for (int u = 0; u < V; ++u)
33         if (n[u] == -1) visit(u, com);
34     return C;
35 }
36 // scc is a map of the original vertices of the
    graph to the vertices
37 // of the SCC graph, scc_graph is its adjacency
    list.
38 // SCC indices and edges are stored in 'scc'
    and 'scc_graph'.
39 void scc_collapse(vi &scc, vvi &scc_graph) {
40     find_sccs(scc);
41     scc_graph.assign(C, vi());
42     set<pi> rec; // recorded edges
43     for (int u = 0; u < V; ++u) {
44         assert(scc[u] != -1);
45         for (int v : edges[u]) {
46             if (scc[v] == scc[u] ||
47                 rec.find({scc[u], scc[v]}) != rec.end())
48                 continue;
49             scc_graph[scc[u]].push_back(scc[v]);
50             rec.insert({scc[u], scc[v]});
51         }
52     }
53 // Function to find sources and sinks in the
    SCC graph

```

```

54 // The number of edges needed to be added is
    max(sources.size(), sinks.())
55 void findSourcesAndSinks(const vvi &scc_graph,
    vi &sources, vi &sinks) {
56     vi in_degree(C, 0), out_degree(C, 0);
57     for (int u = 0; u < C; u++) {
58         for (auto v : scc_graph[u]) {
59             in_degree[v]++;
60             out_degree[u]++;
61         }
62     }
63     for (int i = 0; i < C; ++i) {
64         if (in_degree[i] == 0) sources.push_back(i);
65         if (out_degree[i] == 0) sinks.push_back(i);
66     }
67 }
68 };

```

### 3.1.12 SCC edges Prints out the missing edges to make the input digraph strongly connected

```

1 #include "header.h"
2 const int N=1e5+10;
3 int n,a[N],cnt[N],vis[N];
4 vector<int> hd,tl;
5 int dfs(int x){
6     vis[x]=1;
7     if(!vis[a[x]])return vis[x]=dfs(a[x]);
8     return vis[x]=x;
9 }
10 int main(){
11     scanf("%d",&n);
12     for(int i=1;i<=n;i++){
13         scanf("%d",&a[i]);
14         cnt[a[i]]++;
15     }
16     int k=0;
17     for(int i=1;i<=n;i++){
18         if(!cnt[i]){
19             k++;
20             hd.push_back(i);
21             tl.push_back(dfs(i));
22         }
23     }
24     int tk=k;
25     for(int i=1;i<=n;i++){
26         if(!vis[i]){
27             k++;
28             hd.push_back(i);
29             tl.push_back(dfs(i));
30         }
31     }
32     if(k==1&&!tk)k=0;

```

```

33     printf("%d\n",k);
34     for(int i=0;i<k;i++)printf("%d□%d\n",tl[i],hd
        [(i+1)%k]);
35     return 0;
36 }

```

### 3.1.13 Find Bridges

```

1 #include "header.h"
2 int n; // number of nodes
3 vvi adj; // adjacency list of graph
4 vector<bool> visited;
5 vi tin, low;
6 int timer;
7 void dfs(int v, int p = -1) {
8     visited[v] = true;
9     tin[v] = low[v] = timer++;
10    for (int to : adj[v]) {
11        if (to == p) continue;
12        if (visited[to]) {
13            low[v] = min(low[v], tin[to]);
14        } else {
15            dfs(to, v);
16            low[v] = min(low[v], low[to]);
17            if (low[to] > tin[v])
18                IS_BRIDGE(v, to);
19        }
20    }
21 }
22 void find_bridges() {
23     timer = 0;
24     visited.assign(n, false);
25     tin.assign(n, -1);
26     low.assign(n, -1);
27     for (int i = 0; i < n; ++i) {
28         if (!visited[i]) dfs(i);
29     }
30 }

```

### 3.1.14 Artic. points (i.e. cut off points)

```

1 #include "header.h"
2 int n; // number of nodes
3 vvi adj; // adjacency list of graph
4 vector<bool> visited;
5 vi tin, low;
6 int timer;
7 void dfs(int v, int p = -1) {
8     visited[v] = true;
9     tin[v] = low[v] = timer++;
10    int children=0;
11    for (int to : adj[v]) {
12        if (to == p) continue;

```



```

13     if (visited[to]) {
14         low[v] = min(low[v], tin[to]);
15     } else {
16         dfs(to, v);
17         low[v] = min(low[v], low[to]);
18         if (low[to] >= tin[v] && p!=-1)
19             IS_CUTPOINT(v);
20         ++children;
21     }
22     if(p == -1 && children > 1)
23         IS_CUTPOINT(v);
24 }
25 void find_cutpoints() {
26     timer = 0;
27     visited.assign(n, false);
28     tin.assign(n, -1);
29     low.assign(n, -1);
30     for (int i = 0; i < n; ++i) {
31         if (!visited[i]) dfs(i);
32     }
33 }

```

### 3.1.15 Topological sort

```

1 #include "header.h"
2 int n; // number of vertices
3 vvi adj; // adjacency list of graph
4 vector<bool> visited;
5 vi ans;
6 void dfs(int v) {
7     visited[v] = true;
8     for (int u : adj[v]) {
9         if (!visited[u]) dfs(u);
10    }
11    ans.push_back(v);
12 }
13 void topological_sort() {
14     visited.assign(n, false);
15     ans.clear();
16     for (int i = 0; i < n; ++i) {
17         if (!visited[i]) dfs(i);
18     }
19     reverse(ans.begin(), ans.end());
20 }

```

### 3.1.16 Bellmann-Ford Same as Dijkstra but allows neg. edges

```

1 #include "header.h"
2 // Switch vi and vvpi to vl and vvpl if necessary
3 void bellmann_ford_extended(vvpi &e, int source,
4     vi &dist, vb &cyc) {

```

```

4     dist.assign(e.size(), INF);
5     cyc.assign(e.size(), false); // true when u is
6     in a <0 cycle
7     dist[source] = 0;
8     for (int iter = 0; iter < e.size() - 1; ++iter)
9     {
10        bool relax = false;
11        for (int u = 0; u < e.size(); ++u)
12            if (dist[u] == INF) continue;
13            else for (auto &e : e[u])
14                if (dist[u]+e.second < dist[e.first])
15                    dist[e.first] = dist[u]+e.second, relax
16                    = true;
17            if(!relax) break;
18        }
19        bool ch = true;
20        while (ch) { // keep going untill no
21            more changes
22            ch = false; // set dist to -INF when in
23            cycle
24            for (int u = 0; u < e.size(); ++u)
25                if (dist[u] == INF) continue;
26                else for (auto &e : e[u])
27                    if (dist[e.first] > dist[u] + e.second
28                        && !cyc[e.first]) {
29                        dist[e.first] = -INF;
30                        ch = true; //return true for cycle
31                        detection only
32                        cyc[e.first] = true;
33                    }
34        }
35    }

```

### 3.1.17 Ford-Fulkerson Basic Max. flow

```

1 #include "header.h"
2 #define V 6 // Num. of vertices in given graph
3
4 /* Returns true if there is a path from source 's
5  ' to sink
6  't' in residual graph. Also fills parent[] to
7  store the
8  path */
9 bool bfs(int rGraph[V][V], int s, int t, int
10     parent[]) {
11     bool visited[V];
12     memset(visited, 0, sizeof(visited));
13     queue<int> q;
14     q.push(s);
15     visited[s] = true;
16     parent[s] = -1;

```

```

17     int u = q.front();
18     q.pop();
19
20     for (int v = 0; v < V; v++) {
21         if (visited[v] == false && rGraph[u][v] >
22             0) {
23             if (v == t) {
24                 parent[v] = u;
25                 return true;
26             }
27             q.push(v);
28             parent[v] = u;
29             visited[v] = true;
30         }
31     }
32     return false;
33 }
34
35 // Returns the maximum flow from s to t in the
36 // given graph
37 int fordFulkerson(int graph[V][V], int s, int t)
38 {
39     int u, v;
40     int rGraph[V][V];
41     for (u = 0; u < V; u++)
42         for (v = 0; v < V; v++)
43             rGraph[u][v] = graph[u][v];
44
45     int parent[V]; // This array is filled by BFS
46     // and to
47     // store path
48     int max_flow = 0; // There is no flow initially
49     while (bfs(rGraph, s, t, parent)) {
50         int path_flow = INT_MAX;
51         for (v = t; v != s; v = parent[v]) {
52             u = parent[v];
53             path_flow = min(path_flow, rGraph[u][v]);
54         }
55         for (v = t; v != s; v = parent[v]) {
56             u = parent[v];
57             rGraph[u][v] -= path_flow;
58             rGraph[v][u] += path_flow;
59         }
60         max_flow += path_flow;
61     }
62     return max_flow;

```

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



```

2 using F = ll; using W = ll; // types for flow and
  weight/cost
3 struct S{
4     const int v;           // neighbour
5     const int r;           // index of the reverse edge
6     F f;                   // current flow
7     const F cap;           // capacity
8     const W cost;          // unit cost
9     S(int v, int ri, F c, W cost = 0) :
10         v(v), r(ri), f(0), cap(c), cost(cost) {}
11     inline F res() const { return cap - f; }
12 };
13 struct FlowGraph : vector<vector<S>> {
14     FlowGraph(size_t n) : vector<vector<S>>(n) {}
15     void add_edge(int u, int v, F c, W cost = 0){
16         auto &t = *this;
17         t[u].emplace_back(v, t[v].size(), c, cost
18         );
19         t[v].emplace_back(u, t[u].size()-1, c, -
20         cost);
21     }
22 }
23 void add_arc(int u, int v, F c, W cost = 0){
24     auto &t = *this;
25     t[u].emplace_back(v, t[v].size(), c, cost
26     );
27     t[v].emplace_back(u, t[u].size()-1, 0, -
28     cost);
29 }
30 void clear() { for (auto &E : *this) for (
31     auto &e : E) e.f = 0LL; }
32 };
33 struct Dinic{
34     FlowGraph &edges; int V,s,t;
35     vi l; vector<vector<S>::iterator> its; //
36     levels and iterators
37     Dinic(FlowGraph &edges, int s, int t) :
38         edges(edges), V(edges.size()), s(s), t(t)
39         , l(V,-1), its(V) {}
40     ll augment(int u, F c) { // we reuse the same
41         iterators
42         if (u == t) return c; ll r = 0LL;
43         for(auto &i = its[u]; i != edges[u].end()
44             ; i++){
45             auto &e = *i;
46             if (e.res() && l[u] < l[e.v]) {
47                 auto d = augment(e.v, min(c, e.
48                 res()));
49                 if (d > 0) { e.f += d; edges[e.v
50                 ][e.r].f -= d; c -= d;
51                 r += d; if (!c) break; }
52             }
53         }
54         return r;
55     }
56     ll run() {
57         ll flow = 0, f;

```

```

43 while(true) {
44     fill(l.begin(), l.end(),-1); l[s]=0;
45     // recalculate the layers
46     queue<int> q; q.push(s);
47     while(!q.empty()){
48         auto u = q.front(); q.pop(); its[
49         u] = edges[u].begin();
50         for(auto &&e : edges[u]) if(e.res
51         () && l[e.v]<0)
52             l[e.v] = l[u]+1, q.push(e.v);
53     }
54     if (l[t] < 0) return flow;
55     while ((f = augment(s, INF)) > 0)
56         flow += f;
57 }
58 };

```

## 3.2 Dynamic Programming

### 3.2.1 Longest Incr. Subseq.

```

1 #include "header.h"
2 template<class T>
3 vector<T> index_path_lis(vector<T>& nums) {
4     int n = nums.size();
5     vector<T> sub;
6     vector<int> subIndex;
7     vector<T> path(n, -1);
8     for (int i = 0; i < n; ++i) {
9         if (sub.empty() || sub[sub.size() - 1] <
10             nums[i]) {
11             path[i] = sub.empty() ? -1 : subIndex[sub.
12             size() - 1];
13             sub.push_back(nums[i]);
14             subIndex.push_back(i);
15         } else {
16             int idx = lower_bound(sub.begin(), sub.end(),
17             nums[i]) - sub.begin();
18             path[i] = idx == 0 ? -1 : subIndex[idx - 1];
19             sub[idx] = nums[i];
20             subIndex[idx] = i;
21         }
22     }
23     vector<T> ans;
24     int t = subIndex[subIndex.size() - 1];
25     while (t != -1) {
26         ans.push_back(t);
27         t = path[t];
28     }
29     reverse(ans.begin(), ans.end());
30     return ans;
31 }
32 // Length only
33 template<class T>

```

```

31 int length_lis(vector<T> &a) {
32     set<T> st;
33     typename set<T>::iterator it;
34     for (int i = 0; i < a.size(); ++i) {
35         it = st.lower_bound(a[i]);
36         if (it != st.end()) st.erase(it);
37         st.insert(a[i]);
38     }
39     return st.size();
40 }

```

### 3.2.2 0-1 Knapsack

```

1 #include "header.h"
2 // given a number of coins, calculate all
3 // possible distinct sums
4 int main() {
5     int n;
6     vi coins(n); // all possible coins to use
7     int sum = 0; // sum of the coins
8     vi dp(sum + 1, 0); // dp[x] = 1 if sum
9     x can be made
10    dp[0] = 1; // sum 0 can be
11    made
12    for (int c = 0; c < n; ++c) // first
13        iteration: sums with first
14        coin,
15        next first 2 coins etc
16        if (dp[x]) dp[x + coins[c]] = 1; // if sum
17        x valid, x+c valid
18 }

```

### 3.2.3 Coin change Number of coins required to achieve a given value

```

1 #include "header.h"
2 // Returns total distinct ways to make sum using
3 // n coins of
4 // different denominations
5 int count(vi& coins, int n, int sum) {
6     // 2d dp array where n is the number of coin
7     // denominations and sum is the target sum
8     vector<vector<int>> dp(n + 1, vector<int>(
9     sum + 1, 0));
10    dp[0][0] = 1;
11    for (int i = 1; i <= n; i++) {
12        for (int j = 0; j <= sum; j++) {
13            // without using the current coin,
14            dp[i][j] += dp[i - 1][j];
15            // using the current coin
16            if ((j - coins[i - 1]) >= 0)

```

```

17         dp[i][j] += dp[i][j - coins[i -
18             1]];
19     }
20     return dp[n][sum];
21 }

```

### 3.3 Trees

#### 3.3.1 Tree diameter

```

1 #include "header.h"
2 const int mxN = 2e5 + 5;
3 int n, d[mxN]; // distance array
4 vi adj[mxN]; // tree adjacency list
5 void dfs(int s, int e) {
6     d[s] = 1 + d[e]; // recursively calculate
7     // the distance from the starting node to each
8     // node
9     for (auto u : adj[s]) { // for each adjacent
10         // node
11         if (u != e) dfs(u, s); // don't move
12         // backwards in the tree
13     }
14 }
15 int main() {
16     // read input, create adj list
17     dfs(0, -1); // first dfs call
18     // to find farthest node from arbitrary node
19     dfs(distance(d, max_element(d, d + n)), -1);
20     // second dfs call to find farthest node
21     // from that one
22     cout << *max_element(d, d + n) - 1 << '\n'; //
23     // distance from second node to farthest is
24     // the diameter
25 }

```

#### 3.3.2 Tree Node Count

```

1 #include "header.h"
2 // calculate amount of nodes in each node's
3 // subtree
4 const int mxN = 2e5 + 5;
5 int n, cnt[mxN];
6 vi adj[mxN];
7 void dfs(int s = 0, int e = -1) {
8     cnt[s] = 1; // count leaves as one
9     for (int u : adj[s]) {
10         dfs(u, s);
11         cnt[s] += cnt[u]; // add up nodes of the
12         // subtrees
13     }
14 }

```

### 3.4 Num. Th. / Comb.

#### 3.4.1 Basic stuff

```

1 #include "header.h"
2 ll gcd(ll a, ll b) { while (b) { a %= b; swap(a,
3     b); } return a; }
4 ll lcm(ll a, ll b) { return (a / gcd(a, b)) * b;
5     }
6 ll mod(ll a, ll b) { return ((a % b) + b) % b; }
7 // Finds x, y s.t. ax + by = d = gcd(a, b).
8 void extended_euclid(ll a, ll b, ll &x, ll &y, ll
9     &d) {
10     ll xx = y = 0;
11     ll yy = x = 1;
12     while (b) {
13         ll q = a / b;
14         ll t = b; b = a % b; a = t;
15         t = xx; xx = x - q * xx; x = t;
16         t = yy; yy = y - q * yy; y = t;
17     }
18     d = a;
19 }
20 // solves ab = 1 (mod n), -1 on failure
21 ll mod_inverse(ll a, ll n) {
22     ll x, y, d;
23     extended_euclid(a, n, x, y, d);
24     return (d > 1 ? -1 : mod(x, n));
25 }
26 // All modular inverses of [1..n] mod P in O(n)
27 // time.
28 vi inverses(ll n, ll P) {
29     vi I(n+1, 1LL);
30     for (ll i = 2; i <= n; ++i)
31         I[i] = mod(-(P/i) * I[P%i], P);
32     return I;
33 }
34 // (a*b)%m
35 ll mulmod(ll a, ll b, ll m){
36     ll x = 0, y=a%m;
37     while(b>0){
38         if(b&1) x = (x+y)%m;
39         y = (2*y)%m, b /= 2;
40     }
41     return x % m;
42 }
43 // Finds b^e % m in O(lg n) time, ensure that b <
44 // m to avoid overflow!
45 ll powmod(ll b, ll e, ll m) {
46     ll p = e<2 ? 1 : powmod((b*b)%m,e/2,m);
47     return e&1 ? p*b%m : p;
48 }
49 // Solve ax + by = c, returns false on failure.
50 bool linear_diophantine(ll a, ll b, ll c, ll &x,
51     ll &y) {
52     ll d = gcd(a, b);

```

```

47     if (c % d) {
48         return false;
49     } else {
50         x = c / d * mod_inverse(a / d, b / d);
51         y = (c - a * x) / b;
52         return true;
53     }
54 }

```

#### 3.4.2 Mod. exponentiation Or use pow() in python

```

1 #include "header.h"
2 ll mod_pow(ll base, ll exp, ll mod) {
3     if (mod == 1) return 0;
4     if (exp == 0) return 1;
5     if (exp == 1) return base;
6
7     ll res = 1;
8     base %= mod;
9     while (exp) {
10         if (exp % 2 == 1) res = (res * base) % mod;
11         exp >>= 1;
12         base = (base * base) % mod;
13     }
14
15     return res % mod;
16 }

```

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

```

1 #include "header.h"
2 ll gcd(ll a, ll b) {
3     if (a == 0) return b;
4     return gcd(b % a, a);
5 }

```

#### 3.4.4 Sieve of Eratosthenes

```

1 #include "header.h"
2 vl primes;
3 void getprimes(ll n) { // Up to n (not included)
4     vector<bool> p(n, true);
5     p[0] = false;
6     p[1] = false;
7     for (ll i = 0; i < n; i++) {
8         if (p[i]) {
9             primes.push_back(i);
10             for (ll j = i*2; j < n; j+=i) p[j] =
11                 false;
12         }
13     }
14 }

```

### 3.4.5 Fibonacci % prime

```
1 #include "header.h"
2 const ll MOD = 1000000007;
3 unordered_map<ll, ll> Fib;
4 ll fib(ll n) {
5     if (n < 2) return 1;
6     if (Fib.find(n) != Fib.end()) return Fib[n];
7     Fib[n] = (fib((n + 1) / 2) * fib(n / 2) + fib(
        ((n - 1) / 2) * fib((n - 2) / 2)) % MOD;
8     return Fib[n];
9 }
```

### 3.4.6 nCk % prime

```
1 #include "header.h"
2 ll binom(ll n, ll k) {
3     ll ans = 1;
4     for(ll i = 1; i <= min(k, n-k); ++i) ans = ans
        *(n+1-i)/i;
5     return ans;
6 }
7 ll mod_nCk(ll n, ll k, ll p){
8     ll ans = 1;
9     while(n){
10         ll np = n%p, kp = k%p;
11         if(kp > np) return 0;
12         ans *= binom(np, kp);
13         n /= p; k /= p;
14     }
15     return ans;
16 }
```

### 3.4.7 Chin. rem. th.

```
1 #include "header.h"
2 #include "elementary.cpp"
3 // Solves x = a1 mod m1, x = a2 mod m2, x is
    unique modulo lcm(m1, m2).
4 // Returns {0, -1} on failure, {x, lcm(m1, m2)}
    otherwise.
5 pair<ll, ll> crt(ll a1, ll m1, ll a2, ll m2) {
6     ll s, t, d;
7     extended_euclid(m1, m2, s, t, d);
8     if (a1 % d != a2 % d) return {0, -1};
9     return {mod(s*a2 % m2 * m1 + t*a1 % m1 * m2, m1 *
        m2) / d, m1 / d * m2};
10 }
11
12 // Solves x = ai mod mi. x is unique modulo lcm
    mi.
13 // Returns {0, -1} on failure, {x, lcm mi}
    otherwise.
```

```
14 pair<ll, ll> crt(vector<ll> &a, vector<ll> &m) {
15     pair<ll, ll> res = {a[0], m[0]};
16     for (ull i = 1; i < a.size(); ++i) {
17         res = crt(res.first, res.second, mod(a[i], m[
            i]), m[i]);
18         if (res.second == -1) break;
19     }
20     return res;
21 }
```

## 3.5 Strings

### 3.5.1 Z alg. KMP alternative

```
1 #include "../header.h"
2 void Z_algorithm(const string &s, vi &Z) {
3     Z.assign(s.length(), -1);
4     int L = 0, R = 0, n = s.length();
5     for (int i = 1; i < n; ++i) {
6         if (i > R) {
7             L = R = i;
8             while (R < n && s[R - L] == s[R]) R++;
9             Z[i] = R - L; R--;
10        } else if (Z[i - L] >= R - i + 1) {
11            L = i;
12            while (R < n && s[R - L] == s[R]) R++;
13            Z[i] = R - L; R--;
14        } else Z[i] = Z[i - L];
15    }
16 }
```

### 3.5.2 KMP

```
1 #include "header.h"
2 void compute_prefix_function(string &w, vi &
    prefix) {
3     prefix.assign(w.length(), 0);
4     int k = prefix[0] = -1;
5
6     for(int i = 1; i < w.length(); ++i) {
7         while(k >= 0 && w[k + 1] != w[i]) k = prefix[
            k];
8         if(w[k + 1] == w[i]) k++;
9         prefix[i] = k;
10    }
11 }
12 void knuth_morris_pratt(string &s, string &w) {
13     int q = -1;
14     vi prefix;
15     compute_prefix_function(w, prefix);
16     for(int i = 0; i < s.length(); ++i) {
17         while(q >= 0 && w[q + 1] != s[i]) q = prefix[
            q];
```

```
18         if(w[q + 1] == s[i]) q++;
19         if(q + 1 == w.length()) {
20             // Match at position (i - w.length() + 1)
21             q = prefix[q];
22         }
23     }
24 }
```

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

```
1 #include "header.h"
2
3 map<char, int> cti;
4 int cti_size;
5 template <int ALPHABET_SIZE, int (*mp)(char)>
6 struct AC_FSM {
7     struct Node {
8         int child[ALPHABET_SIZE], failure = 0,
            match_par = -1;
9         vi match;
10        Node() { for (int i = 0; i < ALPHABET_SIZE;
            ++i) child[i] = -1; }
11    };
12    vector<Node> a;
13    vector<string> &words;
14    AC_FSM(vector<string> &words) : words(words) {
15        a.push_back(Node());
16        construct_automaton();
17    }
18    void construct_automaton() {
19        for (int w = 0, n = 0; w < words.size(); ++w,
            n = 0) {
20            for (int i = 0; i < words[w].size(); ++i) {
21                if (a[n].child[mp(words[w][i])] == -1) {
22                    a[n].child[mp(words[w][i])] = a.size();
23                    a.push_back(Node());
24                }
25                n = a[n].child[mp(words[w][i])];
26            }
27            a[n].match.push_back(w);
28        }
29        queue<int> q;
30        for (int k = 0; k < ALPHABET_SIZE; ++k) {
31            if (a[0].child[k] == -1) a[0].child[k] = 0;
32            else if (a[0].child[k] > 0) {
33                a[a[0].child[k]].failure = 0;
34                q.push(a[0].child[k]);
35            }
36        }
37        while (!q.empty()) {
38            int r = q.front(); q.pop();
39            for (int k = 0, arck; k < ALPHABET_SIZE; ++
                k) {
```

```

40     if ((arck = a[r].child[k]) != -1) {
41         q.push(arck);
42         int v = a[r].failure;
43         while (a[v].child[k] == -1) v = a[v].
            failure;
44         a[arck].failure = a[v].child[k];
45         a[arck].match_par = a[v].child[k];
46         while (a[arck].match_par != -1
47             && a[a[arck].match_par].match.empty
                ())
48             a[arck].match_par = a[a[arck].
                match_par].match_par;
49     }
50 }
51 }
52 }
53 void aho_corasick(string &sentence, vvi &
    matches){
54     matches.assign(words.size(), vi());
55     int state = 0, ss = 0;
56     for (int i = 0; i < sentence.length(); ++i,
        ss = state) {
57         while (a[ss].child[mp(sentence[i])] == -1)
58             ss = a[ss].failure;
59         state = a[state].child[mp(sentence[i])]
60             = a[ss].child[mp(sentence[i])];
61         for (ss = state; ss != -1; ss = a[ss].
            match_par)
62             for (int w : a[ss].match)
63                 matches[w].push_back(i + 1 - words[w].
                    length());
64     }
65 }
66 };
67 int char_to_int(char c) {
68     return cti[c];
69 }
70 int main() {
71     ll n;
72     string line;
73     while(getline(cin, line)) {
74         stringstream ss(line);
75         ss >> n;
76
77         vector<string> patterns(n);
78         for (auto& p: patterns) getline(cin, p);
79
80         string text;
81         getline(cin, text);
82
83         cti = {}, cti_size = 0;
84         for (auto c: text) {
85             if (not in(c, cti)) {
86                 cti[c] = cti_size++;
87             }

```

```

88     }
89     for (auto& p: patterns) {
90         for (auto c: p) {
91             if (not in(c, cti)) {
92                 cti[c] = cti_size++;
93             }
94         }
95     }
96
97     vvi matches;
98     AC_FSM <128+1, char_to_int> ac_fms(patterns);
99     ac_fms.aho_corasick(text, matches);
100     for (auto& x: matches) cout << x << endl;
101 }
102
103 }

```

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

```

1 #include "header.h"
2 void manacher(string &s, vi &pal) {
3     int n = s.length(), i = 1, l, r;
4     pal.assign(2 * n + 1, 0);
5     while (i < 2 * n + 1) {
6         if ((i&1) && pal[i] == 0) pal[i] = 1;
7         l = i / 2 - pal[i] / 2; r = (i-1) / 2 + pal[i]
            / 2;
8
9         while (l - 1 >= 0 && r + 1 < n && s[l - 1] ==
            s[r + 1])
10             --l, ++r, pal[i] += 2;
11
12         for (l = i - 1, r = i + 1; l >= 0 && r < 2 *
            n + 1; --l, ++r) {
13             if (l <= i - pal[i]) break;
14             if (l / 2 - pal[l] / 2 > i / 2 - pal[i] /
                2)
15                 pal[r] = pal[l];
16             else { if (l >= 0)
17                 pal[r] = min(pal[l], i + pal[i] - r);
18                 break;
19             }
20         }
21         i = r;
22 } }

```

## 3.6 Geometry

### 3.6.1 essentials.cpp

```

1 #include "../header.h"
2 using C = ld; // could be long long or long
    double

```

```

3 constexpr C EPS = 1e-10; // change to 0 for C=ll
4 struct P { // may also be used as a 2D vector
5     C x, y;
6     P(C x = 0, C y = 0) : x(x), y(y) {}
7     P operator+ (const P &p) const { return {x + p.
        x, y + p.y}; }
8     P operator- (const P &p) const { return {x - p.
        x, y - p.y}; }
9     P operator* (C c) const { return {x * c, y * c
        }; }
10    P operator/ (C c) const { return {x / c, y / c
        }; }
11    C operator* (const P &p) const { return x*p.x +
        y*p.y; }
12    C operator^ (const P &p) const { return x*p.y -
        p.x*y; }
13    P perp() const { return P{y, -x}; }
14    C lensq() const { return x*x + y*y; }
15    ld len() const { return sqrt((ld)lensq()); }
16    static ld dist(const P &p1, const P &p2) {
17        return (p1-p2).len(); }
18    bool operator==(const P &r) const {
19        return ((*this)-r).lensq() <= EPS*EPS; }
20 };
21 C det(P p1, P p2) { return p1^p2; }
22 C det(P p1, P p2, P o) { return det(p1-o, p2-o); }
23 C det(const vector<P> &ps) {
24     C sum = 0; P prev = ps.back();
25     for(auto &p : ps) sum += det(p, prev), prev = p
        ;
26     return sum;
27 }
28 // Careful with division by two and C=ll
29 C area(P p1, P p2, P p3) { return abs(det(p1, p2,
    p3))/C(2); }
30 C area(const vector<P> &poly) { return abs(det(
    poly))/C(2); }
31 int sign(C c){ return (c > C(0)) - (c < C(0)); }
32 int ccw(P p1, P p2, P o) { return sign(det(p1, p2
    , o)); }
33
34 // Only well defined for C = ld.
35 P unit(const P &p) { return p / p.len(); }
36 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.

```

1 #include "header.h"
2 #include "essentials.cpp"
3 bool intersect(P a1, P a2, P b1, P b2) {
4     if (max(a1.x, a2.x) < min(b1.x, b2.x)) return
        false;

```

```

5  if (max(b1.x, b2.x) < min(a1.x, a2.x)) return
    false;
6  if (max(a1.y, a2.y) < min(b1.y, b2.y)) return
    false;
7  if (max(b1.y, b2.y) < min(a1.y, a2.y)) return
    false;
8  bool l1 = ccw(a2, b1, a1) * ccw(a2, b2, a1) <=
    0;
9  bool l2 = ccw(b2, a1, b1) * ccw(b2, a2, b1) <=
    0;
10 return l1 && l2;
11 }

```

### 3.6.3 Convex Hull

```

1 #include "header.h"
2 #include "essentials.cpp"
3 struct ConvexHull { // O(n lg n) monotone chain.
4     size_t n;
5     vector<size_t> h, c; // Indices of the hull
6                             // are in 'h', ccw.
7     const vector<P> &p;
8     ConvexHull(const vector<P> &p) : n(p.size()),
9                                     c(n), p(p) {
10         std::iota(c.begin(), c.end(), 0);
11         std::sort(c.begin(), c.end(), [this](size_t l
12         , size_t r) -> bool { return p[l].x != p[r
13         ].x ? p[l].x < p[r].x : p[l].y < p[r].y;
14         });
15         c.erase(std::unique(c.begin(), c.end(), [this
16         ](size_t l, size_t r) { return p[l] == p[r
17         ]; }));
18         for (size_t s = 1, r = 0; r < 2; ++r, s = h
19         .size()) {
20             for (size_t i : c) {
21                 while (h.size() > s && ccw(p[h.end()
22                 [-2]], p[h.end()[-1]], p[i]) <= 0)
23                     h.pop_back();
24                 h.push_back(i);
25             }
26             reverse(c.begin(), c.end());
27         }
28         if (h.size() > 1) h.pop_back();
29     }
30     size_t size() const { return h.size(); }
31     template <class T, void U(const P &, const P &,
32                             const P &, T &)>
33     void rotating_calipers(T &ans) {
34         if (size() <= 2)
35             U(p[h[0]], p[h.back()], p[h.back()], ans);
36         else
37             for (size_t i = 0, j = 1, s = size(); i < 2
38                 * s; ++i) {
39                 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
    % s]], ans);
    }
    }
    };
    // Example: furthest pair of points. Now set ans
    = 0LL and call
    // ConvexHull(pts).rotating_calipers<ll, update>(
    ans);
36 void update(const P &p1, const P &p2, const P &o,
    ll &ans) {
37     ans = max(ans, (ll)max((p1 - o).lensq(), (p2 -
    o).lensq()));
38 }
39 int main() {
40     ios::sync_with_stdio(false); // do not use
41     cout << printf
42     cin.tie(NULL);
43     int n;
44     cin >> n;
45     while (n) {
46         vector<P> ps;
47         int x, y;
48         for (int i = 0; i < n; i++) {
49             cin >> x >> y;
50             ps.push_back({x, y});
51         }
52         ConvexHull ch(ps);
53         cout << ch.h.size() << endl;
54         for(auto& p: ch.h) {
55             cout << ps[p].x << " " << ps[p].y <<
56             endl;
57         }
58         cin >> n;
59     }
60     return 0;
61 }
62 }

```

## 3.7 Other Algorithms

### 3.7.1 2-sat

```

1 #include "../header.h"
2 #include "../Graphs/tarjan.cpp"
3 struct TwoSAT {
4     int n;
5     vvi imp; // implication graph
6     Tarjan tj;

```

```

7
8     TwoSAT(int _n) : n(_n), imp(2 * _n, vi()), tj(
9     imp) {}
10
11     // Only copy the needed functions:
12     void add_implies(int c1, bool v1, int c2, bool
13     v2) {
14         int u = 2 * c1 + (v1 ? 1 : 0),
15         v = 2 * c2 + (v2 ? 1 : 0);
16         imp[u].push_back(v); // u => v
17         imp[v^1].push_back(u^1); // -v => -u
18     }
19     void add_equivalence(int c1, bool v1, int c2,
20     bool v2) {
21         add_implies(c1, v1, c2, v2);
22         add_implies(c2, v2, c1, v1);
23     }
24     void add_or(int c1, bool v1, int c2, bool v2) {
25         add_implies(c1, !v1, c2, v2);
26     }
27     void add_and(int c1, bool v1, int c2, bool v2)
28     {
29         add_true(c1, v1); add_true(c2, v2);
30     }
31     void add_xor(int c1, bool v1, int c2, bool v2)
32     {
33         add_or(c1, v1, c2, v2);
34         add_or(c1, !v1, c2, !v2);
35     }
36     void add_true(int c1, bool v1) {
37         add_implies(c1, !v1, c1, v1);
38     }
39     // on true: a contains an assignment.
40     // on false: no assignment exists.
41     bool solve(vb &a) {
42         vi com;
43         tj.find_sccs(com);
44         for (int i = 0; i < n; ++i)
45             if (com[2 * i] == com[2 * i + 1])
46                 return false;
47         vvi bycom(com.size());
48         for (int i = 0; i < 2 * n; ++i)
49             bycom[com[i]].push_back(i);
50         a.assign(n, false);
51         vb vis(n, false);
52         for(auto &&component : bycom){
53             for (int u : component) {
54                 if (vis[u / 2]) continue;
55                 vis[u / 2] = true;
56                 a[u / 2] = (u % 2 == 1);
57             }
58         }
59     }

```

```

57     return true;
58 }
59 };

```

### 3.7.2 Matrix Solve

```

1 #include "header.h"
2 #define REP(i, n) for(auto i = decltype(n)(0); i
   < (n); i++)
3 using T = double;
4 constexpr T EPS = 1e-8;
5 template<int R, int C>
6 using M = array<array<T,C>,R>; // matrix
7 template<int R, int C>
8 T ReducedRowEchelonForm(M<R,C> &m, int rows) {
   // return the determinant
9     int r = 0; T det = 1; // MODIFIES
   the input
10    for(int c = 0; c < rows && r < rows; c++) {
11        int p = r;
12        for(int i=r+1; i<rows; i++) if(abs(m[i][c]) >
   abs(m[p][c])) p=i;
13        if(abs(m[p][c]) < EPS){ det = 0; continue; }
14        swap(m[p], m[r]); det = -det;
15        T s = 1.0 / m[r][c], t; det *= m[r][c];
16        REP(j,C) m[r][j] *= s; // make leading
   term in row 1
17        REP(i,rows) if (i!=r){ t = m[i][c]; REP(j,C)
   m[i][j] -= t*m[r][j]; }
18        ++r;
19    }
20    return det;
21 }
22 bool error, inconst; // error => multiple or
   inconsistent
23 template<int R,int C> // Mx = a; M:R*R, v:R*C =>
   x:R*C
24 M<R,C> solve(const M<R,R> &m, const M<R,C> &a,
   int rows){
25     M<R,R+C> q;
26     REP(r,rows){
27         REP(c,rows) q[r][c] = m[r][c];
28         REP(c,C) q[r][R+c] = a[r][c];
29     }
30     ReducedRowEchelonForm<R,R+C>(q,rows);
31     M<R,C> sol; error = false, inconst = false;
32     REP(c,C) for(auto j = rows-1; j >= 0; --j){
33         T t=0; bool allzero=true;
34         for(auto k = j+1; k < rows; ++k)
35             t += q[j][k]*sol[k][c], allzero &= abs(q[j]
   [k]) < EPS;
36         if(abs(q[j][j]) < EPS)
37             error = true, inconst |= allzero && abs(q[j]
   [R+c]) > EPS;

```

```

38     else sol[j][c] = (q[j][R+c] - t) / q[j][j];
   // usually q[j][j]=1
39 }
40 return sol;
41 }

```

### 3.7.3 Matrix Exp.

```

1 #include "header.h"
2 #define ITERATE_MATRIX(w) for (int r = 0; r < (w)
   ; ++r) \
3         for (int c = 0; c < (w); ++c)
4 template <class T, int N>
5 struct M {
6     array<array<T,N>,N> m;
7     M() { ITERATE_MATRIX(N) m[r][c] = 0; }
8     static M id() {
9         M I; for (int i = 0; i < N; ++i) I.m[i][i] =
   1; return I;
10    }
11    M operator*(const M &rhs) const {
12        M out;
13        ITERATE_MATRIX(N) for (int i = 0; i < N; ++i)
14            out.m[r][c] += m[r][i] * rhs.m[i][c];
15        return out;
16    }
17    M raise(ll n) const {
18        if(n == 0) return id();
19        if(n == 1) return *this;
20        auto r = (*this**this).raise(n / 2);
21        return (n%2 ? *this*r : r);
22    }
23 };

```

### 3.7.4 Finite field For FFT

```

1 #include "header.h"
2 #include "../Number_Theory/elementary.cpp"
3 template<ll p,ll w> // prime, primitive root
4 struct Field { using T = Field; ll x; Field(ll x
   =0) : x{x} {}
5     T operator+(T r) const { return {(x+r.x)%p}; }
6     T operator-(T r) const { return {(x-r.x+p)%p};
   }
7     T operator*(T r) const { return {(x*r.x)%p}; }
8     T operator/(T r) const { return (*this)*r.inv()
   ; }
9     T inv() const { return {mod_inverse(x,p)}; }
10    static T root(ll k) { assert( (p-1)%k==0 );
   // (p-1)%k == 0?
11        auto r = powmod(w,(p-1)/abs(k),p); // k-
   th root of unity
12    return k>=0 ? T{r} : T{r}.inv();

```

```

13 }
14 bool zero() const { return x == 0LL; }
15 };
16 using F1 = Field<1004535809,3 >;
17 using F2 = Field<1107296257,10>; // 1<<30 + 1<<25
   + 1
18 using F3 = Field<2281701377,3 >; // 1<<31 + 1<<27
   + 1

```

### 3.7.5 Complex field For FFR

```

1 #include "header.h"
2 const double m_pi = M_PIf64x;
3 struct Complex { using T = Complex; double u,v;
4     Complex(double u=0, double v=0) : u{u}, v{v} {}
5     T operator+(T r) const { return {u+r.u, v+r.v};
   }
6     T operator-(T r) const { return {u-r.u, v-r.v};
   }
7     T operator*(T r) const { return {u*r.u - v*r.v,
   u*r.v + v*r.u}; }
8     T operator/(T r) const {
9         auto norm = r.u*r.u+r.v*r.v;
10        return {(u*r.u + v*r.v)/norm, (v*r.u - u*r.v)
   /norm};
11    }
12    T operator*(double r) const { return T{u*r, v*r
   }; }
13    T operator/(double r) const { return T{u/r, v/r
   }; }
14    T inv() const { return T{1,0}/ *this; }
15    T conj() const { return T{u, -v}; }
16    static T root(ll k){ return {cos(2*m_pi/k), sin
   (2*m_pi/k)}; }
17    bool zero() const { return max(abs(u), abs(v))
   < 1e-6; }
18 };

```

### 3.7.6 FFT

```

1 #include "header.h"
2 #include "complex_field.cpp"
3 #include "fin_field.cpp"
4 void brinc(int &x, int k) {
5     int i = k - 1, s = 1 << i;
6     x ^= s;
7     if ((x & s) != s) {
8         --i; s >= 1;
9         while (i >= 0 && ((x & s) == s))
10             x = x &~ s, --i, s >= 1;
11         if (i >= 0) x |= s;
12     }
13 }

```



```

14 using T = Complex; // using T=F1,F2,F3
15 vector<T> roots;
16 void root_cache(int N) {
17     if (N == (int)roots.size()) return;
18     roots.assign(N, T{0});
19     for (int i = 0; i < N; ++i)
20         roots[i] = ((i&-i) == i)
21             ? T{cos(2.0*m_pi*i/N), sin(2.0*m_pi*i/N)}
22             : roots[i&-i] * roots[i-(i&-i)];
23 }
24 void fft(vector<T> &A, int p, bool inv = false) {
25     int N = 1<<p;
26     for(int i = 0, r = 0; i < N; ++i, brinc(r, p))
27         if (i < r) swap(A[i], A[r]);
28     // Uncomment to precompute roots (for T=Complex)
29     // . Slower but more precise.
30     // root_cache(N);
31     // , sh=p-1 , --sh
32     for (int m = 2; m <= N; m <= 1) {
33         T w, w_m = T::root(inv ? -m : m);
34         for (int k = 0; k < N; k += m) {
35             w = T{1};
36             for (int j = 0; j < m/2; ++j) {
37                 T t = w * A[k + j + m/2];
38                 A[k + j + m/2] = A[k + j] - t;
39                 A[k + j] = A[k + j] + t;
40                 w = w * w_m;
41             }
42         }
43     }
44     if(inv){ T inverse = T(N).inv(); for(auto &x :
45         A) x = x*inverse; }
46 // convolution leaves A and B in frequency domain
47 // state
48 // C may be equal to A or B for in-place
49 // convolution
50 void convolution(vector<T> &A, vector<T> &B,
51     vector<T> &C){
52     int s = A.size() + B.size() - 1;
53     int q = 32 - __builtin_clz(s-1), N=1<<q; //
54     fails if s=1
55     A.resize(N,{}); B.resize(N,{}); C.resize(N,{});
56     fft(A, q, false); fft(B, q, false);
57     for (int i = 0; i < N; ++i) C[i] = A[i] * B[i];
58     fft(C, q, true); C.resize(s);
59 }
60 void square_inplace(vector<T> &A) {
61     int s = 2*A.size()-1, q = 32 - __builtin_clz(s
62         -1), N=1<<q;
63     A.resize(N,{}); fft(A, q, false);
64     for(auto &x : A) x = x*x;
65     fft(A, q, true); A.resize(s);

```

```

61 }

```

### 3.7.7 Polyn. inv. div.

```

1 #include "header.h"
2 #include "fft.cpp"
3 vector<T> &rev(vector<T> &A) { reverse(A.begin(),
4     A.end()); return A; }
5 void copy_into(const vector<T> &A, vector<T> &B,
6     size_t n) {
7     std::copy(A.begin(), A.begin()+min({n, A.size()
8         }, B.size()}), B.begin());
9 }
10 // Multiplicative inverse of A modulo x^n.
11 // Requires A[0] != 0!!
12 vector<T> inverse(const vector<T> &A, int n) {
13     vector<T> Ai{A[0].inv()};
14     for (int k = 0; (1<<k) < n; ++k) {
15         vector<T> As(4<<k, T(0)), Ais(4<<k, T(0));
16         copy_into(A, As, 2<<k); copy_into(Ai, Ais, Ai
17             .size());
18         fft(As, k+2, false); fft(Ais, k+2, false);
19         for (int i = 0; i < (4<<k); ++i) As[i] = As[i]
20             *Ais[i]*Ais[i];
21         fft(As, k+2, true); Ai.resize(2<<k, {});
22         for (int i = 0; i < (2<<k); ++i) Ai[i] = T(2)
23             * Ai[i] - As[i];
24     }
25     Ai.resize(n);
26     return Ai;
27 }
28 // Polynomial division. Returns {Q, R} such that
29 // A = QB+R, deg R < deg B.
30 // Requires that the leading term of B is nonzero
31 .
32 pair<vector<T>, vector<T>> divmod(const vector<T>
33     &A, const vector<T> &B) {
34     size_t n = A.size()-1, m = B.size()-1;
35     if (n < m) return {vector<T>(1, T(0)), A};
36     vector<T> X(A), Y(B), Q, R;
37     convolution(rev(X), Y = inverse(rev(Y), n-m+1),
38         Q);
39     Q.resize(n-m+1); rev(Q);
40     X.resize(Q.size()), copy_into(Q, X, Q.size());
41     Y.resize(B.size()), copy_into(B, Y, B.size());
42     convolution(X, Y, X);
43     R.resize(m), copy_into(A, R, m);
44     for (size_t i = 0; i < m; ++i) R[i] = R[i] - X[
45         i];
46     while (R.size() > 1 && R.back().zero()) R.
47         pop_back();

```

```

39     return {Q, R};
40 }
41 vector<T> mod(const vector<T> &A, const vector<T>
42     &B) {
43     return divmod(A, B).second;
44 }

```

**3.7.8 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.

```

1 #include "header.h"
2 #include "poly.cpp"
3 // x^k mod f
4 vector<T> xmod(const vector<T> f, ll k) {
5     vector<T> r{T(1)};
6     for (int b = 62; b >= 0; --b) {
7         if (r.size() > 1)
8             square_inplace(r), r = mod(r, f);
9         if ((k>>b)&1) {
10             r.insert(r.begin(), T(0));
11             if (r.size() == f.size()) {
12                 T c = r.back() / f.back();
13                 for (size_t i = 0; i < f.size(); ++i)
14                     r[i] = r[i] - c * f[i];
15                 r.pop_back();
16             }
17         }
18     }
19     return r;
20 }
21 // Given A[0,k) and C[0, k), computes the n-th
22 // term of:
23 // A[n] = \sum_i C[i] * A[n-i-1]
24 T nth_term(const vector<T> &A, const vector<T> &C
25     , ll n) {
26     int k = (int)A.size();
27     if (n < k) return A[n];
28     vector<T> f(k+1, T{1});
29     for (int i = 0; i < k; ++i)
30         f[i] = T{-1} * C[k-i-1];
31     f = xmod(f, n);
32     T r = T{0};
33     for (int i = 0; i < k; ++i)
34         r = r + f[i] * A[i];
35     return r;
36 }

```



**3.7.9 Convolution** Precise up to 9e15

---

```

1 #include "header.h"
2 #include "fft.cpp"
3 void convolution_mod(const vi &A, const vi &B, ll
    MOD, vi &C) {
4     int s = A.size() + B.size() - 1; ll m15 = (1LL
        <<15)-1LL;
5     int q = 32 - __builtin_clz(s-1), N=1<<q; //
        fails if s=1
6     vector<T> Ac(N), Bc(N), R1(N), R2(N);
7     for (size_t i = 0; i < A.size(); ++i) Ac[i] = T
        {A[i]&m15, A[i]>>15};
8     for (size_t i = 0; i < B.size(); ++i) Bc[i] = T
        {B[i]&m15, B[i]>>15};
9     fft(Ac, q, false); fft(Bc, q, false);
10    for (int i = 0, j = 0; i < N; ++i, j = (N-1)&(N
        -i)) {
11        T as = (Ac[i] + Ac[j].conj()) / 2;
12        T al = (Ac[i] - Ac[j].conj()) / T{0, 2};
13        T bs = (Bc[i] + Bc[j].conj()) / 2;
14        T bl = (Bc[i] - Bc[j].conj()) / T{0, 2};
15        R1[i] = as*bs + al*bl*T{0,1}, R2[i] = as*bl +
            al*bs;
16    }
17    fft(R1, q, true); fft(R2, q, true);
18    ll p15 = (1LL<<15)%MOD, p30 = (1LL<<30)%MOD; C.
        resize(s);
19    for (int i = 0; i < s; ++i) {
20        ll l = llround(R1[i].u), m = llround(R2[i].u)
            , h = llround(R1[i].v);
21        C[i] = (l + m*p15 + h*p30) % MOD;
22    }
23 }
```

---

**3.7.10 Partitions of  $n$**  Finds all possible partitions of a number

---

```

1 #include "header.h"
2 void printArray(int p[], int n) {
3     for (int i = 0; i < n; i++)
4         cout << p[i] << " ";
5     cout << endl;
6 }
7
8 void printAllUniqueParts(int n) {
9     int p[n]; // An array to store a partition
10    int k = 0; // Index of last element in a
        partition
11    p[k] = n; // Initialize first partition as
        number itself
12
13    // This loop first prints current partition
        then generates next
```

---

```

14 // partition. The loop stops when the current
    partition has all 1s
15 while (true) {
16     printArray(p, k + 1);
17
18     // Find the rightmost non-one value in p[].
        Also, update the
19     // rem_val so that we know how much value can
        be accommodated
20     int rem_val = 0;
21     while (k >= 0 && p[k] == 1) {
22         rem_val += p[k];
23         k--;
24     }
25
26     // if k < 0, all the values are 1 so there
        are no more partitions
27     if (k < 0) return;
28
29     // Decrease the p[k] found above and adjust
        the rem_val
30     p[k]--;
31     rem_val++;
32
33     // If rem_val is more, then the sorted order
        is violated. Divide
34     // rem_val in different values of size p[k]
        and copy these values at
35     // different positions after p[k]
36     while (rem_val > p[k]) {
37         p[k + 1] = p[k];
38         rem_val = rem_val - p[k];
39         k++;
40     }
41
42     // Copy rem_val to next position and
        increment position
43     p[k + 1] = rem_val;
44     k++;
45 }
46 }
```

---

**3.8 Other Data Structures****3.8.1 Disjoint set** (i.e. union-find)

---

```

1 template <typename T>
2 class DisjointSet {
3     typedef T * iterator;
4     T *parent, n, *rank;
5 public:
6     // O(n), assumes nodes are [0, n)
7     DisjointSet(T n) {
8         this->parent = new T[n];
```

---

```

9         this->n = n;
10        this->rank = new T[n];
11
12        for (T i = 0; i < n; i++) {
13            parent[i] = i;
14            rank[i] = 0;
15        }
16    }
17
18    // O(log n)
19    T find_set(T x) {
20        if (x == parent[x]) return x;
21        return parent[x] = find_set(parent[x]
            );
22    }
23
24    // O(log n)
25    void union_sets(T x, T y) {
26        x = this->find_set(x);
27        y = this->find_set(y);
28
29        if (x == y) return;
30
31        if (rank[x] < rank[y]) {
32            T z = x;
33            x = y;
34            y = z;
35        }
36
37        parent[y] = x;
38        if (rank[x] == rank[y]) rank[x]++;
39    }
40 }
```

---

**3.8.2 Fenwick tree** (i.e. BIT) eff. update + prefix sum calc.

---

```

1 #include "header.h"
2 #define maxn 200010
3 int t,n,m,tree[maxn],p[maxn];
4
5 void update(int k, int z) {
6     while (k <= maxn) {
7         tree[k] += z;
8         k += k & (-k);
9     }
10 }
11
12 int sum(int k) {
13     int ans = 0;
14     while(k) {
15         ans += tree[k];
16         k -= k & (-k);
17     }
```

---

```

18     return ans;
19 }

```

### 3.8.3 Fenwick2d tree

```

1 #include "header.h"
2 template <class T>
3 struct FenwickTree2D {
4     vector< vector<T> > tree;
5     int n;
6     FenwickTree2D(int n) : n(n) { tree.assign(n +
7         1, vector<T>(n + 1, 0)); }
8     T query(int x1, int y1, int x2, int y2) {
9         return query(x2,y2)+query(x1-1,y1-1)-query(x2
10             ,y1-1)-query(x1-1,y2);
11     }
12     T query(int x, int y) {
13         T s = 0;
14         for (int i = x; i > 0; i -= (i & (-i)))
15             for (int j = y; j > 0; j -= (j & (-j)))
16                 s += tree[i][j];
17         return s;
18     }
19     void update(int x, int y, T v) {
20         for (int i = x; i <= n; i += (i & (-i)))
21             for (int j = y; j <= n; j += (j & (-j)))
22                 tree[i][j] += v;
23     }
24 };

```

### 3.8.4 Trie

```

1 #include "header.h"
2 const int ALPHABET_SIZE = 26;
3 inline int mp(char c) { return c - 'a'; }
4
5 struct Node {
6     Node* ch[ALPHABET_SIZE];
7     bool isleaf = false;
8     Node() {
9         for(int i = 0; i < ALPHABET_SIZE; ++i) ch[i]
10             = nullptr;
11     }
12     void insert(string &s, int i = 0) {
13         if (i == s.length()) isleaf = true;
14         else {
15             int v = mp(s[i]);
16             if (ch[v] == nullptr)
17                 ch[v] = new Node();
18             ch[v]->insert(s, i + 1);
19         }
20     }

```

```

21
22 bool contains(string &s, int i = 0) {
23     if (i == s.length()) return isleaf;
24     else {
25         int v = mp(s[i]);
26         if (ch[v] == nullptr) return false;
27         else return ch[v]->contains(s, i + 1);
28     }
29 }
30
31 void cleanup() {
32     for (int i = 0; i < ALPHABET_SIZE; ++i)
33         if (ch[i] != nullptr) {
34             ch[i]->cleanup();
35             delete ch[i];
36         }
37 }
38 };

```

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

```

1 #include "header.h"
2 struct Node {
3     ll v;
4     int sz, pr;
5     Node *l = nullptr, *r = nullptr;
6     Node(ll val) : v(val), sz(1) { pr = rand(); }
7 };
8 int size(Node *p) { return p ? p->sz : 0; }
9 void update(Node* p) {
10     if (!p) return;
11     p->sz = 1 + size(p->l) + size(p->r);
12     // Pull data from children here
13 }
14 void propagate(Node *p) {
15     if (!p) return;
16     // Push data to children here
17 }
18 void merge(Node *&t, Node *l, Node *r) {
19     propagate(l), propagate(r);
20     if (!l) t = r;
21     else if (!r) t = l;
22     else if (l->pr > r->pr)
23         merge(l->r, l->r, r), t = l;
24     else merge(r->l, l, r->l), t = r;
25     update(t);
26 }
27 void spliti(Node *t, Node *&l, Node *&r, int
28     index) {
29     propagate(t);
30     if (!t) { l = r = nullptr; return; }
31     int id = size(t->l);

```

```

31     if (index <= id) // id \in [index, \infty), so
32         move it right
33         spliti(t->l, l, t->l, index), r = t;
34     else
35         spliti(t->r, t->r, r, index - id), l = t;
36     update(t);
37 }
38 void splitv(Node *t, Node *&l, Node *&r, ll val)
39 {
40     propagate(t);
41     if (!t) { l = r = nullptr; return; }
42     if (val <= t->v) // t->v \in [val, \infty), so
43         move it right
44         splitv(t->l, l, t->l, val), r = t;
45     else
46         splitv(t->r, t->r, r, val), l = t;
47     update(t);
48 }
49 void clean(Node *p) {
50     if (p) { clean(p->l), clean(p->r); delete p; }
51 }

```

## 4 Other Mathematics

### 4.1 Helpful functions

**4.1.1 Euler's Totient Fucntion**  $n = p_1^{k_1-1} \cdot (p_1 - 1) \cdot \dots \cdot p_r^{k_r-1} \cdot (p_r - 1)$ , where  $p_1^{k_1} \cdot \dots \cdot p_r^{k_r}$  is the prime factorization of  $n$ .

```

1 # include "header.h"
2 ll phi(ll n) { // \Phi(n)
3     ll ans = 1;
4     for (ll i = 2; i*i <= n; i++) {
5         if (n % i == 0) {
6             ans *= i-1;
7             n /= i;
8             while (n % i == 0) {
9                 ans *= i;
10                 n /= i;
11             }
12         }
13     }
14     if (n > 1) ans *= n-1;
15     return ans;
16 }
17 vi phis(int n) { // All \Phi(i) up to n
18     vi phi(n + 1, 0LL);
19     iota(phi.begin(), phi.end(), 0LL);
20     for (ll i = 2LL; i <= n; ++i)
21         if (phi[i] == i)
22             for (ll j = i; j <= n; j += i)

```

---

```

23     phi[j] -= phi[j] / i;
24     return phi;
25 }

```

---

**Formulas**  $\Phi(n)$  counts all numbers in  $1, \dots, n-1$  coprime to  $n$ .

$a^{\varphi(n)} \equiv 1 \pmod n$ ,  $a$  and  $n$  are coprimes.

$\forall e > \log_2 m : n^e \pmod m = n^{\Phi(m)+e \pmod{\Phi(m)}} \pmod m$ .

$\gcd(m, n) = 1 \Rightarrow \Phi(m \cdot n) = \Phi(m) \cdot \Phi(n)$ .

**4.1.2 Pascal's trinagle**  $\binom{n}{k}$  is  $k$ -th element in the  $n$ -th row, indexing both from 0

---

```

1  #include "header.h"
2  void printPascal(int n) {
3      for (int line = 1; line <= n; line++) {
4          int C = 1; // used to represent C(line, i)
5          for (int i = 1; i <= line; i++) {
6
7              // The first value in a line is
              always 1
8              cout << C << "□";
9              C = C * (line - i) / i;
10         }
11         cout << "\n";
12     }
13 }

```

---

## 4.2 Theorems and definitions

### Fermat's little theorem

$$a^p \equiv a \pmod{p}$$

### Subfactorial

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

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

### 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 divisible by  $k'!$ .

Basic identities: The hockeystick identity:

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

or

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

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.  $n = n_k p^k + \dots + n_1 p + n_0$  and  $m = m_k p^k + \dots + m_1 p + m_0$ . Then by Lucas theorem we have  $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$ , 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**  $a + b = a \oplus b + 2(a \& b) = a|b + a \& b$ .

$k$ th bit is set in  $x$  iff  $x \bmod 2^{k-1} \geq 2^k$ , or iff  $x \bmod 2^{k-1} - x \bmod 2^k \neq 0$  (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.

$$n \bmod 2^i = n \& (2^i - 1).$$

$$\forall k: 1 \oplus 2 \oplus \dots \oplus (4k-1) = 0$$

**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:

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

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

**Second kind:**  $S_2(n, k)$  count partitions of  $n$  distinct elements into exactly  $k$  non-empty groups.

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

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

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

## 4.3 Geometry Formulas

$$[ABC] = rs = \frac{1}{2} ab \sin \gamma = \frac{abc}{4R} = \sqrt{s(s-a)(s-b)(s-c)} = \frac{1}{2} |(B - A)_{\text{Graph 4}}^T| \cdot \phi^*(2^n) \text{ time.}$$

$$s = \frac{a+b+c}{2}$$

$$2R = \frac{a}{\sin \alpha}$$

cosine rule:

$$c^2 = a^2 + b^2 - 2ab \cos \gamma$$

Euler:

$$1 + CC = V - E + F$$

Pick:

$$\text{Area} = \text{interior points} + \frac{\text{boundary points}}{2}$$

$$p \cdot q = |p||q| \cos(\theta) \quad |p \times q| = |p||q| \sin(\theta)$$

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:

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

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

**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  $\mathcal{O}^*(2^n)$  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

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

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  $t_1^{r_1} \dots t_n^{r_n}$  in the polynomial

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

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  $-1$ .

**Lucas Theorem** If  $p$  is prime, then:

$$\frac{p^a}{k} \equiv 0 \pmod{p}$$

Thus for non-negative integers  $m = m_k p^k + \dots + m_1 p + m_0$  and  $n = n_k p^k + \dots + n_1 p + n_0$ :

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

Note: The fraction's mean integer division.

**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.

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

$$C_0 = 1, C_1 = 1, C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

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

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