1 Setup

1.1 header.h

```cpp
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
```

```cpp
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())
```

```cpp
20 #define in(el, cont) (find(cont.begin(), cont.end
    (), el) != cont.end())
21
22 constexpr int INF  = 2000000010;
23 constexpr ll LLINF = 9000000000000000010LL;
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++;
```

```cpp
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
        cout + printf
39 //  cin.tie(NULL);
40 //  cout << fixed << setprecision(12);
41 //  return 0;
42 // }
```

1.2 Bash for c++ compile with header.h

```bash
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++

```bash
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

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

1.4.1 Aux. helper C++

```cpp
1 #include "header.h"
2
3 int main() {
```

```cpp
4    // Read in a line including white space
5    string line;
6    getline(cin, line);
7    // When doing the above read numbers as
        follows:
8    int n;
9    getline(cin, line);
10   stringstream ss(line);
11   ss >> n;
12
13   // Count the number of 1s in binary
        represnatation of a number
14   ull number;
15   __builtin_popcountll(number);
16 }
```

1.4.2 Aux. helper python

```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

```python
1 from collections import deque
2 def bfs(g, roots, n):
3     q = deque(roots)
4     explored = set()
5     distances = [0 if v in roots else float('inf'
          ) for v in range(n)]
6
7     while len(q) != 0:
8         node = q.popleft()
9         if node in explored: continue
10        explored.add(node)
11        for neigh in g[node]:
12            if neigh not in explored:
```

```python
13                q.append(neigh)
14                distances[neigh] = distances[node
                     ] + 1
15    return distances
```

2.1.2 Dijkstra

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

2.1.3 Topological Sort

```python
1 #Python program to print topological sorting of a
     DAG
2 from collections import defaultdict
3
4 #Class to represent a graph
5 class Graph:
6     def __init__(self,vertices):
7         self.graph = defaultdict(list) #
              dictionary containing adjacency List
8         self.V = vertices #No. of vertices
9
10    # function to add an edge to graph
11    def addEdge(self,u,v):
12        self.graph[u].append(v)
13
```

```python
14    # A recursive function used by
          topologicalSort
15    def topologicalSortUtil(self,v,visited,stack)
          :
16
17        # Mark the current node as visited.
18        visited[v] = True
19
20        # Recur for all the vertices adjacent to
              this vertex
21        for i in self.graph[v]:
22            if visited[i] == False:
23                self.topologicalSortUtil(i,
                      visited,stack)
24
25        # Push current vertex to stack which
              stores result
26        stack.insert(0,v)
27
28    # The function to do Topological Sort. It
          uses recursive
29    # topologicalSortUtil()
30    def topologicalSort(self):
31        # Mark all the vertices as not visited
32        visited = [False]*self.V
33        stack =[]
34
35        # Call the recursive helper function to
              store Topological
36        # Sort starting from all vertices one by
              one
37        for i in range(self.V):
38            if visited[i] == False:
39                self.topologicalSortUtil(i,
                      visited,stack)
40
41        # Print contents of stack
42        return stack
43
44    def isCyclicUtil(self, v, visited, recStack):
45
46        # Mark current node as visited and
47        # adds to recursion stack
48        visited[v] = True
49        recStack[v] = True
50
51        # Recur for all neighbours
52        # if any neighbour is visited and in
53        # recStack then graph is cyclic
54        for neighbour in self.graph[v]:
55            if visited[neighbour] == False:
56                if self.isCyclicUtil(neighbour,
                      visited, recStack) == True:
57                    return True
58            elif recStack[neighbour] == True:
```

```python
59                return True
60
61        # The node needs to be popped from
62        # recursion stack before function ends
63        recStack[v] = False
64        return False
65
66    # Returns true if graph is cyclic else false
67    def isCyclic(self):
68        visited = [False] * (self.V + 1)
69        recStack = [False] * (self.V + 1)
70        for node in range(self.V):
71            if visited[node] == False:
72                if self.isCyclicUtil(node,
                      visited, recStack) == True:
73                    return True
74        return False
```

2.1.4   Kruskal (UnionFind)

```python
1   class UnionFind:
2       def __init__(self, n):
3           self.parent = [-1]*n
4
5       def find(self, x):
6           if self.parent[x] < 0:
7               return x
8           self.parent[x] = self.find(self.parent[x
                  ])
9           return self.parent[x]
10
11      def connect(self, a, b):
12          ra = self.find(a)
13          rb = self.find(b)
14          if ra == rb:
15              return False
16          if self.parent[ra] > self.parent[rb]:
17              self.parent[rb] += self.parent[ra]
18              self.parent[ra] = rb
19          else:
20              self.parent[ra] += self.parent[rb]
21              self.parent[rb] = ra
22          return True
23
24  # Full MST is len(spanning==n-1)
25  def kruskal(n, edges):
26      uf = UnionFind(n)
27      spanning = []
28      edges.sort(key = lambda d: -d[2])
29      while edges and len(spanning) < n-1:
30          u, v, w = edges.pop()
31          if not uf.connect(u, v):
32              continue
33          spanning.append((u, v, w))
```

```python
34      return spanning
35
36  # Example
37  edges = [(1, 2, 10), (2, 3, 20)]
```

2.2   Num. Th. / Comb.

2.2.1   nCk % prime

```python
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

```python
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

```python
def longestCommonSubsequence(text1, text2):  # O(
    m*n) time, O(m) space
    n = len(text1)
    m = len(text2)

    # Initializing two lists of size m
    prev = [0] * (m + 1)
    cur = [0] * (m + 1)

    for idx1 in range(1, n + 1):
        for idx2 in range(1, m + 1):
            # If characters are matching
            if text1[idx1 - 1] == text2[idx2 -
                1]:
                cur[idx2] = 1 + prev[idx2 - 1]
            else:
                # If characters are not matching
                cur[idx2] = max(cur[idx2 - 1],
                    prev[idx2])

        prev = cur.copy()

    return cur[m]
```

2.3.2 KMP

```python
class KMP:
    def partial(self, pattern):
        """ Calculate 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):
        """KMP search main algorithm: String ->
            String -> [Int]
        Return all the matching position of
            pattern string P in T"""
        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):
```

```python
            ret.append(i - (j - 1))
            j = partial[j - 1]
    return ret
```

2.3.3 Edit distance

```python
def editDistance(str1, str2):
  # Get the lengths of the input strings
  m = len(str1)
  n = len(str2)

  # Initialize a list to store the current row
  curr = [0] * (n + 1)

  # Initialize the first row with values from 0
      to n
  for j in range(n + 1):
    curr[j] = j

  # Initialize a variable to store the previous
      value
  previous = 0

  # Loop through the rows of the dynamic
      programming matrix
  for i in range(1, m + 1):
    # Store the current value at the beginning of
        the row
    previous = curr[0]
    curr[0] = i

    # Loop through the columns of the dynamic
        programming matrix
    for j in range(1, n + 1):
      # Store the current value in a temporary
          variable
      temp = curr[j]

      # Check if the characters at the current
          positions in str1 and str2 are the same
      if str1[i - 1] == str2[j - 1]:
        curr[j] = previous
      else:
        # Update the current cell with the
            minimum of the three adjacent cells
        curr[j] = 1 + min(previous, curr[j - 1],
            curr[j])

      # Update the previous variable with the
          temporary value
      previous = temp

  # The value in the last cell represents the
      minimum number of operations
```

```python
    return curr[n]
```

2.4 Other Algorithms

2.4.1 Rotate matrix

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

2.5 Geometry

2.5.1 Convex Hull

```python
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
```

2.5.2 Geometry

```python
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
```

2.6 Other Data Structures

2.6.1 Segment Tree

```python
N = 100000  # limit for array size
tree = [0] * (2 * N)  # Max size of tree

def build(arr, n):  # function to build the tree
    # insert leaf nodes in tree
    for i in range(n):
        tree[n + i] = arr[i]

    # build the tree by calculating parents
    for i in range(n - 1, 0, -1):
        tree[i] = tree[i << 1] + tree[i << 1 | 1]

def updateTreeNode(p, value, n):  # function to
    update a tree node
    # set value at position p
```

```python
    tree[p + n] = value
    p = p + n

    i = p  # move upward and update parents
    while i > 1:
        tree[i >> 1] = tree[i] + tree[i ^ 1]
        i >>= 1

def query(l, r, n):  # function to get sum on
    interval [l, r)
    res = 0
    # loop to find the sum in the range
    l += n
    r += n
    while l < r:
        if l & 1:
            res += tree[l]
            l += 1
        if r & 1:
            r -= 1
            res += tree[r]
        l >>= 1
        r >>= 1
    return res
```

2.6.2 Trie

```python
class TrieNode:
    def __init__(self):
        self.children = [None]*26
        self.isEndOfWord = False

class Trie:
    def __init__(self):
        self.root = self.getNode()

    def getNode(self):
        return TrieNode()

    def _charToIndex(self,ch):
        return ord(ch)-ord('a')


    def insert(self,key):
        pCrawl = self.root
        length = len(key)
        for level in range(length):
            index = self._charToIndex(key[level])
            if not pCrawl.children[index]:
                pCrawl.children[index] = self.
                    getNode()
            pCrawl = pCrawl.children[index]
        pCrawl.isEndOfWord = True
```

```python
    def search(self, key):
        pCrawl = self.root
        length = len(key)
        for level in range(length):
            index = self._charToIndex(key[level])
            if not pCrawl.children[index]:
                return False
            pCrawl = pCrawl.children[index]

        return pCrawl.isEndOfWord
```

3 C++

3.1 Graphs

3.1.1 BFS

```cpp
#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);
```

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

3.1.3 Dijkstra

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

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

3.1.4 Floyd-Warshall

```
1 #include "header.h"
2 // g[i][j] = infty 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] <
                       LLINF and g[i][j] > g[i][k]
                       + g[k][j]) {
9                     g[i][j] = g[i][k] + g[k][j];
10 }}}}}
```

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
     <ll, ll, ll>>& edges, ll n) {
5     set<pair<ll, ll>> ans;
6     ll cost = 0;
7
8     sort(edges.begin(), edges.end());
9     DisjointSet<ll> fs(n);
10
11    ll dist, i, j;
12    for (auto edge: edges) {
13        dist = get<0>(edge);
14        i = get<1>(edge);
15        j = get<2>(edge);
16
17        if (fs.find_set(i) != fs.find_set(j)) {
18            fs.union_sets(i, j);
19            ans.insert({i, j});
20            cost += dist;
21        }
22    }
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; }
4 /**
5  * Given J jobs and W workers (J <= W), computes
      the minimum cost to assign each
6  * prefix of jobs to distinct workers.
7  * @tparam T a type large enough to represent
      integers on the order of J *
8  * max(|C|)
9  * @param C a matrix of dimensions JxW such that
      C[j][w] = cost to assign j-th
10 * job to w-th worker (possibly negative)
11 *
12 * @return a vector of length J, with the j-th
      entry equaling the minimum cost
13 * to assign the first (j+1) jobs to distinct
      workers
14 */
15 template <class T> vector<T> hungarian(const
     vector<vector<T>> &C) {
16    const int J = (int)size(C), W = (int)size(C
         [0]);
17    assert(J <= W);
18    // job[w] = job assigned to w-th worker, or
         -1 if no job assigned
19    // note: a W-th worker was added for
         convenience
20    vector<int> job(W + 1, -1);
21    vector<T> ys(J), yt(W + 1);  // potentials
22    // -yt[W] will equal the sum of all deltas
23    vector<T> answers;
24    const T inf = numeric_limits<T>::max();
25    for (int j_cur = 0; j_cur < J; ++j_cur) {  //
          assign j_cur-th job
26        int w_cur = W;
27        job[w_cur] = j_cur;
28        // min reduced cost over edges from Z to
             worker w
29        vector<T> min_to(W + 1, inf);
30        vector<int> prv(W + 1, -1);  // previous
             worker on alternating path
31        vector<bool> in_Z(W + 1);    // whether
             worker is in Z
32        while (job[w_cur] != -1) {   // runs at
             most j_cur + 1 times
33            in_Z[w_cur] = true;
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]))
                   prv[w] = w_cur;
41             if (ckmin(delta, min_to[w]))
                   w_next = w;
42           }
43         }
44         // delta will always be non-negative,
45         // except possibly during the first
              time this loop runs
46         // if any entries of C[j_cur] are
              negative
47         for (int w = 0; w <= W; ++w) {
48             if (in_Z[w]) ys[job[w]] += delta,
                  yt[w] -= delta;
49             else min_to[w] -= delta;
50         }
51         w_cur = w_next;
52       }
53       // update assignments along alternating
            path
54       for (int w; w_cur != W; w_cur = w) job[
            w_cur] = job[w = prv[w_cur]];
55       answers.push_back(-yt[W]);
56     }
57     return answers;
58 }
```

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<
       int, pair<ld, int>>>
4  graph g;
5  const ld infty = 1e60l;  // Change if necessary
6  ld fill(int n, vld& potential) {  // Finds max
       flow, min cost
7    priority_queue<pair<ld, int>> pq;
8    vector<bool> visited(n+2, false);
9    vi parent(n+2, 0);
10   vld dist(n+2, infty);
11   dist[0] = 0.l;
12   pq.emplace(make_pair(0.l, 0));
13   while (not pq.empty()) {
14     int node = pq.top().second;
15     pq.pop();
16     if (visited[node]) continue;
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
              ] - potential[neigh];
24         if (d + 1e-10l < dist[neigh]) {
25           dist[neigh] = d;
26           pq.emplace(make_pair(-d, neigh));
27           parent[neigh] = node;
28   }}}}
29
30   for (int i = 0; i < n+2; i++) {
31     potential[i] = min(infty, potential[i] + dist
           [i]);
32   }
33   if (not parent[n+1]) return infty;
34   ld ans = 0.l;
35   for (int x = n+1; x; x=parent[x]) {
36     ans += g[parent[x]][x].first;
37     g[parent[x]][x].second--;
38     g[x][parent[x]].second++;
39   }
40   return ans;
41 }
```

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
            side 1's for other side
7      bool is_bipartite = true;   // becomes false
            if not bipartite
8      queue<int> q;
9      for (int st = 0; st < n; ++st) {
10         if (side[st] == -1) {
11             q.push(st);
12             side[st] = 0;
13             while (!q.empty()) {
14                 int v = q.front();
15                 q.pop();
16                 for (int u : adj[v]) {
17                     if (side[u] == -1) {
18                         side[u] = side[v] ^ 1;
19                         q.push(u);
20                     } else {
21                         is_bipartite &= side[u]
                               != side[v];
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;
                v = parent[v])
36             cycle.push_back(v);
37         cycle.push_back(cycle_start);
38         reverse(cycle.begin(), cycle.end());
39
40         cout << "Cycle␣Found:␣";
41         for (int v : cycle) cout << v << "␣";
42         cout << endl;
43     }
44 }
```

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;
```

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

3.1.11 Tarjan's SCC

```cpp
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) {
```

```cpp
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()
                   ) continue;
48             scc_graph[scc[u]].push_back(scc[v]);
49             rec.insert({scc[u], scc[v]});
50         }
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++) {
```

```cpp
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

```cpp
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

```cpp
#include "header.h"
int n; // number of nodes
vvi adj; // adjacency list of graph
vector<bool> visited;
vi tin, low;
int timer;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}
void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i]) dfs(i);
    }
}
```

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

```cpp
#include "header.h"
int n; // number of nodes
vvi adj; // adjacency list of graph
vector<bool> visited;
vi tin, low;
int timer;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
```

```cpp
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}
void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i]) dfs (i);
    }
}
```

3.1.15 Topological sort

```cpp
#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());
}
```

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

```cpp
#include "header.h"
// Switch vi and vvpi to vl and vvpl if necessary
void bellmann_ford_extended(vvpi &e, int source,
    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;
  for (int iter = 0; iter < e.size() - 1; ++iter)
      {
    bool relax = false;
    for (int u = 0; u < e.size(); ++u)
```

```cpp
      if (dist[u] == INF) continue;
      else for (auto &e : e[u])
        if(dist[u]+e.second < dist[e.first])
          dist[e.first] = dist[u]+e.second, relax
              = true;
    if(!relax) break;
  }
  bool ch = true;
  while (ch) {          // keep going untill no
      more changes
    ch = false;          // set dist to -INF when in
        cycle
    for (int u = 0; u < e.size(); ++u)
      if (dist[u] == INF) continue;
      else for (auto &e : e[u])
        if (dist[e.first] > dist[u] + e.second
          && !cyc[e.first]) {
          dist[e.first] = -INF;
          ch = true;     //return true for cycle
              detection only
          cyc[e.first] = true;
        }
  }
}
```

3.1.17 Ford-Fulkerson Basic Max. flow

```cpp
#include "header.h"
#define V 6  // Num. of vertices in given graph

/* Returns true if there is a path from source 's
    ' to sink
't' in residual graph. Also fills parent[] to
    store the
path */
bool bfs(int rGraph[V][V], int s, int t, int
    parent[]) {
  bool visited[V];
  memset(visited, 0, sizeof(visited));
  queue<int> q;
  q.push(s);
  visited[s] = true;
  parent[s] = -1;

  // Standard BFS Loop
  while (!q.empty()) {
    int u = q.front();
    q.pop();

    for (int v = 0; v < V; v++) {
      if (visited[v] == false && rGraph[u][v] >
          0) {
        if (v == t) {
          parent[v] = u;
```

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

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

```
1
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     t[v].emplace_back(u, t[u].size()-1, c, -
       cost);
19   }
20   void add_arc(int u, int v, F c, W cost = 0){
21     auto &t = *this;
22     t[u].emplace_back(v, t[v].size(), c, cost
       );
23     t[v].emplace_back(u, t[u].size()-1, 0, -
       cost);
24   }
25   void clear() { for (auto &E : *this) for (
       auto &e : E) e.f = 0LL; }
26 };
27 struct Dinic{
28   FlowGraph &edges; int V,s,t;
29   vi l; vector<vector<S>::iterator> its; //
       levels and iterators
30   Dinic(FlowGraph &edges, int s, int t) :
31     edges(edges), V(edges.size()), s(s), t(t)
       , l(V,-1), its(V) {}
32   ll augment(int u, F c) { // we reuse the same
       iterators
33     if (u == t) return c; ll r = 0LL;
34     for(auto &i = its[u]; i != edges[u].end()
       ; i++){
35       auto &e = *i;
36       if (e.res() && l[u] < l[e.v]) {
37         auto d = augment(e.v, min(c, e.
       res()));
38         if (d > 0) { e.f += d; edges[e.v
       ][e.r].f -= d; c -= d;
39           r += d; if (!c) break; }
40       }   }
41     return r;
42   }
43   ll run() {
44     ll flow = 0, f;
45     while(true) {
46       fill(l.begin(), l.end(),-1); l[s]=0;
       // recalculate the layers
47       queue<int> q; q.push(s);
48       while(!q.empty()){
49         auto u = q.front(); q.pop(); its[
       u] = edges[u].begin();
50         for(auto &&e : edges[u]) if(e.res
```

```
49         () && l[e.v]<0)
           l[e.v] = l[u]+1, q.push(e.v);
50       }
51       if (l[t] < 0) return flow;
52       while ((f = augment(s, INF)) > 0)
53         flow += f;
54     }   }
54 };
```

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] <
       nums[i]) {
10     path[i] = sub.empty() ? -1 : subIndex[sub.
       size() - 1];
11     sub.push_back(nums[i]);
12     subIndex.push_back(i);
13     } else {
14     int idx = lower_bound(sub.begin(), sub.end(),
       nums[i]) - sub.begin();
15     path[i] = idx == 0 ? -1 : subIndex[idx - 1];
16     sub[idx] = nums[i];
17     subIndex[idx] = i;
18     }
19   }
20   vector<T> ans;
21   int t = subIndex[subIndex.size() - 1];
22   while (t != -1) {
23     ans.push_back(t);
24     t = path[t];
25   }
26   reverse(ans.begin(), ans.end());
27   return ans;
28 }
29 // Length only
30 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
       possible distinct sums
3  int main() {
4    int n;
5    vi coins(n);  // all possible coins to use
6    int sum = 0;        // sum of the coins
7    vi dp(sum + 1, 0);           // dp[x] = 1 if sum
         x can be made
8    dp[0] = 1;                 // sum 0 can be
         made
9    for (int c = 0; c < n; ++c)         // first
         iteration: sums with first
10     for (int x = sum; x >= 0; --x)      // coin,
         next first 2 coins etc
11       if (dp[x]) dp[x + coins[c]] = 1;  // if sum
             x valid, x+c valid
12 }
```

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
       n coins of
3  // different denominations
4  int count(vi& coins, int n, int sum) {
5    // 2d dp array where n is the number of coin
6    // denominations and sum is the target sum
7    vector<vector<int> > dp(n + 1, vector<int>(
         sum + 1, 0));
8    dp[0][0] = 1;
9    for (int i = 1; i <= n; i++) {
10       for (int j = 0; j <= sum; j++) {
11
12           // without using the current coin,
13           dp[i][j] += dp[i - 1][j];
14
15           // using the current coin
16           if ((j - coins[i - 1]) >= 0)
17               dp[i][j] += dp[i][j - coins[i -
                   1]];
18       }
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
         the distance from the starting node to each
         node
7    for (auto u : adj[s]) { // for each adjacent
         node
8      if (u != e) dfs(u, s);  // don't move
           backwards in the tree
9    }
10 }
11 int main() {
12   // read input, create adj list
13   dfs(0, -1);                    // first dfs call
         to find farthest node from arbitrary node
14   dfs(distance(d, max_element(d, d + n)), -1);
         // second dfs call to find farthest node
         from that one
15   cout << *max_element(d, d + n) - 1 << '\n';  //
         distance from second node to farthest is
         the diameter
16 }
```

3.3.2   Tree Node Count

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

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

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];
```

```cpp
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

```cpp
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 - $O(n)$

```cpp
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

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

```cpp
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.

```cpp
#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

```cpp
#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 %
```

```cpp
            ]], 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);
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;
```

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

```cpp
57      return true;
58    }
59  };
```

3.7.2 Matrix Solve

```cpp
 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;
```

```cpp
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.

```cpp
 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

```cpp
 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();
```

```cpp
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

```cpp
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

```cpp
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 }
```

```cpp
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)
        . Slower but more precise.
29  //  root_cache(N);
30  //           , sh=p-1     , --sh
31    for (int m = 2; m <= N; m <<= 1) {
32      T w, w_m = T::root(inv ? -m : m);
33      for (int k = 0; k < N; k += m) {
34        w = T{1};
35        for (int j = 0; j < m/2; ++j) {
36  //        T w = (!inv ? roots[j<<sh] : roots[j<<
        sh].conj());
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 :
        A) x = x*inverse; }
45  }
46  // convolution leaves A and B in frequency domain
         state
47  // C may be equal to A or B for in-place
        convolution
48  void convolution(vector<T> &A, vector<T> &B,
        vector<T> &C){
49    int s = A.size() + B.size() - 1;
50    int q = 32 - __builtin_clz(s-1), N=1<<q;  //
        fails if s=1
51    A.resize(N,{}); B.resize(N,{}); C.resize(N,{});
52    fft(A, q, false); fft(B, q, false);
53    for (int i = 0; i < N; ++i) C[i] = A[i] * B[i];
54    fft(C, q, true); C.resize(s);
55  }
56  void square_inplace(vector<T> &A) {
57    int s = 2*A.size()-1, q = 32 - __builtin_clz(s
        -1), N=1<<q;
58    A.resize(N,{}); fft(A, q, false);
59    for(auto &x : A) x = x*x;
60    fft(A, q, true); A.resize(s);
```

```cpp
61  }
```

3.7.7   Polyn. inv. div.

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

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

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
       term of:
22 // A[n] = \sum_i C[i] * A[n-i-1]
23 T nth_term(const vector<T> &A, const vector<T> &C
       , ll n) {
24   int k = (int)A.size();
25   if (n < k) return A[n];
26
27   vector<T> f(k+1, T{1});
28   for (int i = 0; i < k; ++i)
29     f[i] = T{-1} * C[k-i-1];
30   f = xmod(f, n);
31
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];
```

```cpp
        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.

```cpp
#include "header.h"
#define maxn 200010
int t,n,m,tree[maxn],p[maxn];

void update(int k, int z) {
    while (k <= maxn) {
        tree[k] += z;
        k += k & (-k);
    }
}

int sum(int k) {
    int ans = 0;
    while(k) {
        ans += tree[k];
        k -= k & (-k);
    }
}
```

```cpp
    return ans;
}
```

3.8.3   Fenwick2d tree

```cpp
#include "header.h"
template <class T>
struct FenwickTree2D {
  vector< vector<T> > tree;
  int n;
  FenwickTree2D(int n) : n(n) { tree.assign(n +
      1, vector<T>(n + 1, 0)); }
  T query(int x1, int y1, int x2, int y2) {
    return query(x2,y2)+query(x1-1,y1-1)-query(x2
        ,y1-1)-query(x1-1,y2);
  }
  T query(int x, int y) {
    T s = 0;
    for (int i = x; i > 0; i -= (i & (-i)))
      for (int j = y; j > 0; j -= (j & (-j)))
        s += tree[i][j];
    return s;
  }
  void update(int x, int y, T v) {
    for (int i = x; i <= n; i += (i & (-i)))
      for (int j = y; j <= n; j += (j & (-j)))
        tree[i][j] += v;
  }
};
```

3.8.4   Trie

```cpp
#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);
    }
  }
```

```cpp
  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];
      }
  }
};
```

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

```cpp
#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);
```

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

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.

```
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 $1, \ldots, n-1$ 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

```
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

$$a^p \equiv a \mod p$$

Subfactorial

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

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

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 divisibleby $k'!$. 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. $n = n_k p^k + \cdots + n_1 p + n_0$ 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.
$n \mod 2^i = n\&(2^i - 1)$.
$\forall k : \ 1 \oplus 2 \oplus \ldots \oplus (4k-1) = 0$

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:

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

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

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

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

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

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

4.3 Geometry Formulas

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

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

$$s = \frac{a+b+c}{2} \qquad 2R = \frac{a}{\sin \alpha}$$

cosine rule: $\qquad c^2 = a^2 + b^2 - 2ab \cos \gamma$

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

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

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

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:

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

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

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

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.

Lucas Theorem If p is prime, then:

$$\frac{p^a}{k} \equiv 0 (\mod p)$$

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

$$\frac{m}{n} = \Pi_{i=0}^{k} \frac{m_i}{n_i} \mod p$$

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.

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