

1 Setup	1	3.1.10 Find cycle directed .	7
1.1 header.h	1	3.1.11 Tarjan's SCC	7
1.2 Bash for c++ compile with header.h	1	3.1.12 SCC edges	8
1.3 Bash for run tests c++ . .	2	3.1.13 Find Bridges	8
1.4 Bash for run tests python .	2	3.1.14 Find articulation points	9
1.4.1 Auxiliary helper C++	2	3.1.15 Topological sort . .	9
1.4.2 Auxiliary helper python	2	3.2 Dynamic Programming . .	9
2 Python	2	3.2.1 Longest Increasing Subsequence	9
2.1 Graphs	2	3.2.2 0-1 Knapsack	10
2.1.1 BFS	2	3.3 Trees	10
2.1.2 Dijkstra	2	3.3.1 Tree diameter	10
2.2 Dynamic Programming . .	3	3.3.2 Tree Node Count . .	10
2.3 Trees	3	3.4 Number Theory / Combinatorics	10
2.4 Number Theory / Combinatorics	3	3.4.1 Modular exponentiation	10
2.4.1 nCk % prime	3	3.4.2 GCD	10
2.4.2 Sieve of Eratosthenes	3	3.4.3 Sieve of Eratosthenes	10
2.5 Strings	3	3.4.4 Fibonacci % prime .	11
2.5.1 LCS	3	3.4.5 nCk % prime	11
2.5.2 KMP	3	3.5 Strings	11
2.6 Geometry	3	3.5.1 Aho-Corasick algorithm	11
2.7 Other Algorithms	3	3.5.2 KMP	12
2.7.1 Rotate matrix	3	3.6 Geometry	12
2.8 Other Data Structures . .	4	3.6.1 essentials.cpp	12
2.8.1 Segment Tree	4	3.6.2 Convex Hull	12
3 C++	4	3.7 Other Algorithms	13
3.1 Graphs	4	3.8 Other Data Structures . .	13
3.1.1 BFS	4	3.8.1 Disjoint set	13
3.1.2 DFS	4	3.8.2 Fenwick tree	13
3.1.3 Dijkstra	5	4 Other Mathematics	13
3.1.4 Floyd-Warshall . . .	5	4.1 Helpful functions	13
3.1.5 Kruskal	5	4.1.1 Euler's Totient Function	13
3.1.6 Hungarian algorithm	5	4.2 Theorems and definitions .	14
3.1.7 Successive shortest path	6		
3.1.8 Bipartite check . . .	6		
3.1.9 Find cycle directed .	7		

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>
10 #define vl vector<ll>
11 #define vi vector<int>
12 #define vvi vector<vi>
13 #define vvl vector<vl>
14 #define vpl vector<pl>
15 #define vpi vector<pi>
16 #define vld vector<ld>
17 #define in_fast(el, cont) (cont.find(el) != cont.end())
18 #define in(el, cont) (find(cont.begin(), cont.end(), el) != cont.end())
19
20 constexpr int INF = 2000000010;
21 constexpr ll LLINF = 9000000000000000010LL;
22
23 template <typename T, template <typename ELEM, typename ALLOC = std::
    allocator<ELEM> > class Container>
24 std::ostream& operator<<(std::ostream& o, const Container<T>& container) {
25     typename Container<T>::const_iterator beg = container.begin();
26     if (beg != container.end()) {
27         o << *beg++;
28         while (beg != container.end()) {
29             o << " " << *beg++;
30         }
31     }
32     return o;
33 }
34
35 // int main() {
36 //     ios::sync_with_stdio(false); // do not use cout + printf
37 //     cin.tie(NULL);
38 //     cout << fixed << setprecision(12);
39 //     return 0;
40 // }

```

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 Auxiliary 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 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 Auxiliary helper python

```
1 # Read until EOF
2 while True:
3     try:
4         pattern = input()
5     except EOFError:
6         break
```

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] + 1
16    return distances
```

2.1.2 Dijkstra

```
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
```

2.2 Dynamic Programming

2.3 Trees

2.4 Number Theory / Combinatorics

2.4.1 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

```

2.4.2 Sieve of Eratosthenes $O(n)$ 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

```

2.5 Strings

2.5.1 LCS

```

1 def longestCommonSubsequence(text1, text2): # O(m*n) time, O(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 - 1]:
13                cur[idx2] = 1 + prev[idx2 - 1]
14            else:
15                # If characters are not matching
16                cur[idx2] = max(cur[idx2 - 1], prev[idx2])
17
18        prev = cur.copy()
19
20    return cur[m]

```

2.5.2 KMP

```

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

```

2.6 Geometry

2.7 Other Algorithms

2.7.1 Rotate matrix

```

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

```

2.8 Other Data Structures

2.8.1 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 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 interval [l, r)
24     res = 0
25     # loop to find the sum in the range
26     l += n
27     r += n
28     while l < r:
29         if l & 1:
30             res += tree[l]
31             l += 1
32         if r & 1:
33             r -= 1
34             res += tree[r]
35     l >>= 1
36     r >>= 1
37     return res

```

3 C++

3.1 Graphs

3.1.1 BFS

```

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

```

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 {
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    return pair<set<pair<ll, ll>>, ll> {ans, cost};
25 }

```

3.1.6 Hungarian algorithm

```

1 #include "header.h"
2
3 template <class T> bool ckmin(T &a, const T &b) { return b < a ? a = b, 1 :
4     0; }
5
6 /**
7  * Given J jobs and W workers (J <= W), computes the minimum cost to assign
8  * each
9  * prefix of jobs to distinct workers.
10 * @tparam T a type large enough to represent integers on the order of J *
11 * max(|C|)
12 * @param C a matrix of dimensions JxW such that C[j][w] = cost to assign j-
13 * th
14 * job to w-th worker (possibly negative)
15 *
16 * @return a vector of length J, with the j-th entry equaling the minimum
17 * cost
18 * to assign the first (j+1) jobs to distinct workers
19 */
20 template <class T> vector<T> hungarian(const vector<vector<T>> &C) {
21     const int J = (int)size(C), W = (int)size(C[0]);
22     assert(J <= W);
23     // job[w] = job assigned to w-th worker, or -1 if no job assigned
24     // note: a W-th worker was added for convenience
25     vector<int> job(W + 1, -1);
26     vector<T> ys(J), yt(W + 1); // potentials
27 }

```

```

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] - ys[j] - yt[w]))
40                     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 Successive 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 inf = 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, inf);
11    dist[0] = 0.1;
12    pq.emplace(make_pair(0.1, 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        }
31        for (int i = 0; i < n+2; i++) {
32            potential[i] = min(inf, potential[i] + dist[i]);
33        }
34        if (not parent[n+1]) return inf;
35        ld ans = 0.1;
36        for (int x = n+1; x; x = parent[x]) {
37            ans += g[parent[x]][x].first;
38            g[parent[x]][x].second--;
39            g[x][parent[x]].second++;
40        }
41        return ans;

```

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;

```

```

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])) 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         cout << "Cycle Found: ";
39         for (int v : cycle) cout << v << " ";
40         cout << endl;
41     }
42 }

```

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    {}
11    void visit(int u, vi &com) {
12        l[u] = n[u] = counter++;
13        st.push(u);
14        vs[u] = true;
15        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.size())
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 Find articulation 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) IS_CUTPOINT(v);
19            ++children;
20        }
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 }

```

3.2 Dynamic Programming

3.2.1 Longest Increasing Subsequence

```

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 }

```

3.3 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
7     // starting node to each node
8     for (auto u : adj[s]) { // for each adjacent node
9         if (u != e) dfs(u, s); // don't move backwards in the tree
10     }
11 }
12 int main() {
13     // read input, create adj list
14     dfs(0, -1); // first dfs call to find farthest node from
15     // arbitrary node
16     dfs(distance(d, max_element(d, d + n)), -1); // second dfs call to find
17     // farthest node from that one
18     cout << *max_element(d, d + n) - 1 << '\n'; // distance from second node
19     // to farthest is the diameter
20 }

```

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 Number Theory / Combinatorics

3.4.1 Modular 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 }

```

3.4.2 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 }

```

3.4.3 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      }}

```

3.4.4 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)
8         / 2)) % MOD;
9     return Fib[n];
10 }

```

3.4.5 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 }

```

3.5 Strings

3.5.1 Aho-Corasick algorithm 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 }

```

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 }

```

3.6 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 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 are in 'h', ccw.
6     const vector<P> &p;
7     ConvexHull(const vector<P> &p) : n(_p.size()), c(n), p(_p) {
8         std::iota(c.begin(), c.end(), 0);
9         std::sort(c.begin(), c.end(), [this](size_t l, size_t r) -> bool {
10             return p[l].x != p[r].x ? p[l].x < p[r].x : p[l].y < p[r].y; });
11         c.erase(std::unique(c.begin(), c.end(), [this](size_t l, size_t r) {
12             return p[l] == p[r]; })), c.end());
13         for (size_t s = 1, r = 0; r < 2; ++r, s = h.size()) {
14             for (size_t i : c) {
15                 while (h.size() > s && ccw(p[h.end()[-2]], p[h.end()[-1]], p[i]) <= 0)
16                     h.pop_back();
17                 h.push_back(i);
18             }
19             reverse(c.begin(), c.end());
20         }
21         if (h.size() > 1) h.pop_back();
22     }
23     size_t size() const { return h.size(); }
24     template <class T, void U(const P &, const P &, const P &, T &)>
25     void rotating_calipers(T &ans) {
26         if (size() <= 2)
27             U(p[h[0]], p[h.back()], p[h.back()], ans);
28         else

```

```

27     for (size_t i = 0, j = 1, s = size(); i < 2 * s; ++i) {
28         while (det(p[h[(i + 1) % s]] - p[h[i % s]], p[h[(j + 1) % s]] - p[h[
                j]]) >= 0)
29             j = (j + 1) % s;
30         U(p[h[i % s]], p[h[(i + 1) % s]], p[h[j]], ans);
31     }
32 }
33 };
34 // Example: furthest pair of points. Now set ans = 0LL and call
35 // 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 }

```

3.7 Other Algorithms

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          // cout << "k: " << k << endl;
10     }
11 }
12
13 int sum(int k) {
14     int ans = 0;
15     while(k) {
16         ans += tree[k];
17         k -= k & (-k);
18     }
19     return ans;
20 }

```

4 Other Mathematics

4.1 Helpful functions

4.1.1 Euler's Totient Function $n = p_1^{k_1-1} \cdot (p_1 - 1) \cdot \dots \cdot p_r^{k_r-1} \cdot (p_r - 1)$, 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.

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

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

4.2 Theorems and definitions

Fermat's little theorem $a^p \equiv a \pmod p$

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

Least common multiple $\text{lcm}(a, b) = a \cdot b / \gcd(a, b)$

Binomials and other partitionings We have $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} = \prod_{i=1}^k \frac{n-i+1}{i}$. 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 $\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}$.

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 $n_i < m_i \Rightarrow \binom{n_i}{m_i} = 0$.

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 $x \pmod{2^{k-1}} \geq 2^{k-1}$, or iff $x \pmod{2^{k-1}} - x \pmod{2^k} \neq 0$ (i.e. $= 2^{k-1}$)

It comes handy when you need to look at the bits of the numbers which are pair sums or subset sums etc.

$n \pmod{2^i} = n \& (2^i - 1)$.

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

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

Second kind: $S_2(n, k)$ count partitions of n distinct elements into exactly k non-empty groups. $S_2(n, k) = S_2(n-1, k-1) + kS_2(n-1, k)$ with $S_2(n, 1) = S_2(n, n) = 1$ and $S_2(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n$