

ICPC - Notebook

Contents

1 data structures	2	7.8 Fast Exp	14
1.1 Ordered Set Gnu Pbds	2	7.9 Lcm-using-factorization	14
1.2 Segtree Rmq Lazy Max Update	2	7.10 Euler-phi	14
1.3 Segtree Rmq Lazy Range	2	7.11 Polynomial	14
1.4 Segtree Point Rmq	2	7.12 Integer Mod	14
1.5 Segtree Rsq Lazy Range Sum	3	7.13 Count Divisors Memo	15
1.6 Segtree Rxq Lazy Range Xor	3	7.14 Lcm	15
1.7 Dsu	4	7.15 Factorial-factorization	15
1.8 Dsu	4	7.16 Factorization-with-primes	15
1.9 Sparse Table Rmq	4	7.17 Modular-inverse-using-phi	16
2 graphs	4	7.18 Factorization	16
2.1 Scc-nodes-(kosajaru)	4	7.19 Gcd	16
2.2 2-sat-(struct)	5	7.20 Combinatorics With Repetitions	16
2.3 Floyd Warshall	5	8 strings	16
2.4 Topological-sorting	5	8.1 Rabin-karp	16
2.5 Lowest Common Ancestor Sparse Table	6	8.2 Trie-naive	16
2.6 Count-scc-(kosajaru)	6	8.3 String-psum	17
2.7 Kruskal	6		
2.8 Scc-(struct)	7		
2.9 Check-bipartite	7		
2.10 Dijkstra	7		
3 extras	7		
3.1 Binary To Gray	7		
3.2 Bigint	7		
3.3 Get-permutation-cycles	10		
4 dynamic programming	10		
4.1 Edit Distance	10		
4.2 Money Sum Bottom Up	10		
4.3 Knapsack Dp Values 01	11		
4.4 Tsp	11		
5 trees	11		
5.1 Binary-lifting	11		
5.2 Maximum-distances	11		
5.3 Heavy-light-decomposition	12		
5.4 Tree Diameter	12		
6 searching	13		
6.1 Ternary Search Recursive	13		
7 math	13		
7.1 Power-sum	13		
7.2 Sieve-list-primes	13		
7.3 Factorial	13		
7.4 Permutation-count	13		
7.5 N-choose-k-count	13		
7.6 Gcd-using-factorization	13		
7.7 Is-prime	14		

1 data structures

1.1 Ordered Set Gnu Pbds

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 template <typename T>
5 // using ordered_set = tree<T, null_type, less<T>,
6 // rb_tree_tag,
7 // tree_order_statistics_node_update>;
8 // if you want to find the elements less or equal :p
9 using ordered_set = tree<T, null_type, less_equal<T>,
10 rb_tree_tag,
11 tree_order_statistics_node_update>;
```

1.2 Segtree Rmq Lazy Max Update

```
1 struct SegmentTree {
2     int N;
3     vll ns, lazy;
4     SegmentTree(const vll &xs) : N(xs.size()), ns(4 *
5     N, 0), lazy(4 * N, 0) {
6         for (size_t i = 0; i < xs.size(); ++i) {
7             update(i, i, xs[i]);
8         }
9     }
10    void update(int a, int b, ll value) { update(1,
11    0, N - 1, a, b, value); }
12    void update(int node, int L, int R, int a, int b,
13    ll value) {
14        if (lazy[node]) {
15            ns[node] = max(ns[node], lazy[node]);
16            if (L < R) {
17                lazy[2 * node] = max(lazy[2 * node],
18                lazy[node]);
19                lazy[2 * node + 1] = max(lazy[2 *
20                node + 1], lazy[node]);
21            }
22            lazy[node] = 0;
23        }
24        if (a > R or b < L) return;
25        if (a <= L and R <= b) {
26            ns[node] = max(ns[node], value);
27            if (L < R) {
28                lazy[2 * node] = max(value, lazy[2 *
29                node]);
30                lazy[2 * node + 1] = max(value, lazy
31                [2 * node + 1]);
32            }
33            return;
34        }
35        update(2 * node, L, (L + R) / 2, a, b, value);
36        update(2 * node + 1, (L + R) / 2 + 1, R, a, b
37        , value);
38        ns[node] = max(ns[2 * node], ns[2 * node +
39        1]);
40    }
41    ll RMQ(int a, int b) { return RMQ(1, 0, N - 1, a,
42    b); }
43    ll RMQ(int node, int L, int R, int a, int b) {
44        if (lazy[node]) {
45            ns[node] = max(ns[node], lazy[node]);
46            if (L < R) {
47                lazy[2 * node] = max(lazy[2 * node],
48                lazy[node]);
49                lazy[2 * node + 1] = max(lazy[2 *
50                node + 1], lazy[node]);
51            }
52            lazy[node] = 0;
53        }
54    }
```

```
44     if (a > R or b < L) return 0;
45     if (a <= L and R <= b) return ns[node];
46     ll x = RMQ(2 * node, L, (L + R) / 2, a, b);
47     ll y = RMQ(2 * node + 1, (L + R) / 2 + 1, R,
48     a, b);
49     return max(x, y);
50 };
```

1.3 Segtree Rmq Lazy Range

```
1 struct SegmentTree {
2     int N;
3     vll ns, lazy;
4     SegmentTree(const vll &xs) : N(xs.size()), ns(4 *
5     N, 0) {
6         for (size_t i = 0; i < xs.size(); ++i) update
7         (i, i, xs[i]);
8     }
9     void update(int a, int b, ll value) { update(1,
10    0, N - 1, a, b, value); }
11    void update(int node, int L, int R, int a, int b,
12    ll value) {
13        if (lazy[node]) {
14            ns[node] = ns[node] == INT_MAX ? lazy[
15            node] : ns[node] + lazy[node];
16            if (L < R) {
17                lazy[2 * node] += lazy[node];
18                lazy[2 * node + 1] += lazy[node];
19            }
20            lazy[node] = 0;
21        }
22        if (a > R or b < L) return;
23        if (a <= L and R <= b) {
24            ns[node] = ns[node] == INT_MAX ? value :
25            ns[node] + value;
26            if (L < R) {
27                lazy[2 * node] += value;
28                lazy[2 * node + 1] += value;
29            }
30            return;
31        }
32        update(2 * node, L, (L + R) / 2, a, b, value);
33        update(2 * node + 1, (L + R) / 2 + 1, R, a, b
34        , value);
35        ns[node] = min(ns[2 * node], ns[2 * node +
36        1]);
37    }
38    ll RMQ(int a, int b) { return RMQ(1, 0, N - 1, a,
39    b); }
40    ll RMQ(int node, int L, int R, int a, int b) {
41        if (lazy[node]) {
42            ns[node] = ns[node] == INT_MAX ? lazy[
43            node] : ns[node] + lazy[node];
44            if (L < R) {
45                lazy[2 * node] += lazy[node];
46                lazy[2 * node + 1] += lazy[node];
47            }
48            lazy[node] = 0;
49        }
50        if (a > R or b < L) return INT_MAX;
51        if (a <= L and R <= b) return ns[node];
52        ll x = RMQ(2 * node, L, (L + R) / 2, a, b);
53        ll y = RMQ(2 * node + 1, (L + R) / 2 + 1, R,
54        a, b);
55        return min(x, y);
56    }
57 };
```

1.4 Segtree Point Rmq

```
1 class SegTree {
2     public:
3     int n;
```

```

4     vector<ll> st;
5     SegTree(const vector<ll> &v) : n((int)v.size()),
6     st(n * 4 + 1, LLONG_MAX) {
7         for (int i = 0; i < n; ++i) update(i, v[i]);
8     }
9     void update(int p, ll v) { update(1, 0, n - 1, p,
10     v); }
11     ll RMQ(int l, int r) { return RMQ(1, 0, n - 1, l,
12     r); }
13 private:
14     void update(int node, int l, int r, int p, ll v)
15     {
16         if (p < l or p > r) return; // fora do
17         intervalo.
18         if (l == r) {
19             st[node] = v;
20             return;
21         }
22         int mid = l + (r - l) / 2;
23         update(node * 2, l, mid, p, v);
24         update(node * 2 + 1, mid + 1, r, p, v);
25         st[node] = min(st[node * 2], st[node * 2 +
26     1]);
27     }
28     ll RMQ(int node, int L, int R, int l, int r) {
29         if (l <= L and r >= R) return st[node];
30         if (L > r or R < l) return LLONG_MAX;
31         if (L == R) return st[node];
32         int mid = L + (R - L) / 2;
33         return min(RMQ(node * 2, L, mid, l, r),
34         RMQ(node * 2 + 1, mid + 1, R, l, r));
35     }
36 }
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67

```

1.5 Segtree Rsq Lazy Range Sum

```

1 struct SegTree {
2     int N;
3     vector<ll> ns, lazy;
4
5     SegTree(const vector<ll> &xs) : N(xs.size()), ns
6     (4 * N, 0), lazy(4 * N, 0) {
7         for (size_t i = 0; i < xs.size(); ++i) update
8     (i, i, xs[i]);
9     }
10     void update(int a, int b, ll value) { update(1,
11     0, N - 1, a, b, value); }
12     void update(int node, int L, int R, int a, int b,
13     ll value) {
14         // Lazy propagation
15         if (lazy[node]) {
16             ns[node] += (R - L + 1) * lazy[node];
17             if (L < R) // Se o ón ãno é uma folha,
18             propaga
19             {
20                 lazy[2 * node] += lazy[node];
21                 lazy[2 * node + 1] += lazy[node];
22             }
23             lazy[node] = 0;
24         }
25         if (a > R or b < L) return;
26         if (a <= L and R <= b) {
27

```

```

        ns[node] += (R - L + 1) * value;
        if (L < R) {
            lazy[2 * node] += value;
            lazy[2 * node + 1] += value;
        }
        return;
    }
    update(2 * node, L, (L + R) / 2, a, b, value)
    ;
    update(2 * node + 1, (L + R) / 2 + 1, R, a, b
    , value);
    ns[node] = ns[2 * node] + ns[2 * node + 1];
}
ll RSQ(int a, int b) { return RSQ(1, 0, N - 1, a,
b); }
ll RSQ(int node, int L, int R, int a, int b) {
    if (lazy[node]) {
        ns[node] += (R - L + 1) * lazy[node];
        if (L < R) {
            lazy[2 * node] += lazy[node];
            lazy[2 * node + 1] += lazy[node];
        }
        lazy[node] = 0;
    }
    if (a > R or b < L) return 0;
    if (a <= L and R <= b) return ns[node];
    ll x = RSQ(2 * node, L, (L + R) / 2, a, b);
    ll y = RSQ(2 * node + 1, (L + R) / 2 + 1, R,
a, b);
    return x + y;
}
};

```

1.6 Segtree Rxq Lazy Range Xor

```

1 struct SegTree {
2     int N;
3     vector<ll> ns, lazy;
4
5     SegTree(const vector<ll> &xs) : N(xs.size()), ns
6     (4 * N, 0), lazy(4 * N, 0) {
7         for (size_t i = 0; i < xs.size(); ++i) update
8     (i, i, xs[i]);
9     }
10     void update(int a, int b, ll value) { update(1,
11     0, N - 1, a, b, value); }
12     void update(int node, int L, int R, int a, int b,
13     ll value) {
14         // Lazy propagation
15         if (lazy[node]) {
16             ns[node] ^= lazy[node];
17             if (L < R) // Se o ón ãno é uma folha,
18             propaga
19             {
20                 lazy[2 * node] ^= lazy[node];
21                 lazy[2 * node + 1] ^= lazy[node];
22             }
23             lazy[node] = 0;
24         }
25         if (a > R or b < L) return;

```

```

26
27     if (a <= L and R <= b) {
28         ns[node] ^= value;
29
30         if (L < R) {
31             lazy[2 * node] ^= value;
32             lazy[2 * node + 1] ^= value;
33         }
34
35         return;
36     }
37
38     update(2 * node, L, (L + R) / 2, a, b, value)
39 ;
40     update(2 * node + 1, (L + R) / 2 + 1, R, a, b
41 , value);
42
43     ns[node] = ns[2 * node] ^ ns[2 * node + 1];
44
45 ll rxq(int a, int b) { return RSQ(1, 0, N - 1, a,
46 b); }
47
48 ll rxq(int node, int L, int R, int a, int b) {
49     if (lazy[node]) {
50         ns[node] ^= lazy[node];
51
52         if (L < R) {
53             lazy[2 * node] ^= lazy[node];
54             lazy[2 * node + 1] ^= lazy[node];
55         }
56
57         lazy[node] = 0;
58     }
59
60     if (a > R or b < L) return 0;
61
62     if (a <= L and R <= b) return ns[node];
63
64     ll x = rxq(2 * node, L, (L + R) / 2, a, b);
65     ll y = rxq(2 * node + 1, (L + R) / 2 + 1, R,
66 a, b);
67
68     return x ^ y;
69 }
70 };

```

1.7 Dsu

```

1 class DSU:
2     def __init__(self, n):
3         self.n = n
4         self.p = [x for x in range(0, n + 1)]
5         self.size = [0 for i in range(0, n + 1)]
6
7     def find_set(self, x): # log n
8         if self.p[x] == x:
9             return x
10        else:
11            self.p[x] = self.find_set(self.p[x])
12            return self.p[x]
13
14    def same_set(self, x, y): # log n
15        return bool(self.find_set(x) == self.find_set
16(y))
17
18    def union_set(self, x, y): # log n
19        px = self.find_set(x)
20        py = self.find_set(y)
21
22        if px == py:
23            return
24
25        size_x = self.size[px]
26        size_y = self.size[py]
27
28        if size_x > size_y:

```

```

28        self.p[py] = self.p[px]
29        self.size[px] += self.size[py]
30    else:
31        self.p[px] = self.p[py]
32        self.size[py] += self.size[px]

```

1.8 Dsu

```

1 struct DSU {
2     vector<int> ps;
3     vector<int> size;
4     DSU(int N) : ps(N + 1), size(N + 1, 1) { iota(ps.
5 begin(), ps.end(), 0); }
6     int find_set(int x) { return ps[x] == x ? x : ps[
7 x] = find_set(ps[x]); }
8     bool same_set(int x, int y) { return find_set(x)
9 == find_set(y); }
10    void union_set(int x, int y) {
11        if (same_set(x, y)) return;
12
13        int px = find_set(x);
14        int py = find_set(y);
15
16        if (size[px] < size[py]) swap(px, py);
17
18        ps[py] = px;
19        size[px] += size[py];
20    }
21 };

```

1.9 Sparse Table Rmq

```

1 /*
2     Sparse table implementation for rmq.
3     build: O(NlogN)
4     query: O(1)
5 */
6 int fastlog2(ll x) {
7     ull i = x;
8     return i ? __builtin_clzll(1) - __builtin_clzll(i
9 ) : -1;
10 }
11 template <typename T>
12 class SparseTable {
13 public:
14     int N;
15     int K;
16     vector<vector<T>> st;
17     SparseTable(vector<T> vs)
18         : N((int)vs.size()), K(fastlog2(N) + 1), st(K
19 + 1, vector<T>(N + 1)) {
20         copy(vs.begin(), vs.end(), st[0].begin());
21
22         for (int i = 1; i <= K; ++i)
23             for (int j = 0; j + (1 << i) <= N; ++j)
24                 st[i][j] = min(st[i - 1][j], st[i -
25 1][j + (1 << (i - 1))]);
26     }
27     T RMQ(int l, int r) { // [l, r], 0 indexed
28         int i = fastlog2(r - l + 1);
29         return min(st[i][l], st[i][r - (1 << i) + 1])
30 };

```

2 graphs

2.1 Scc-nodes-(kosajaru)

```

1 /*
2 * O(n+m)
3 * Returns a pair <a, b>
4 *     a: number of SCCs
5 *     b: vector of size n, where b[i] is the SCC id
6         of node i

```

```

6  * */
7  void dfs(ll u, vchar &visited, const vll2d &g, vll &
    scc, bool buildScc, ll id,
8      vll &sccid) {
9      visited[u] = true;
10     sccid[u] = id;
11     for (auto &v : g[u])
12         if (!visited[v]) dfs(v, visited, g, scc,
            buildScc, id, sccid);
13
14     // if it's the first pass, add the node to the
    scc
15     if (buildScc) scc.eb(u);
16 }
17
18 pair<ll, vll> kosajaru(vll2d &g) {
19     ll n = len(g);
20     vll scc;
21     vchar vis(n);
22     vll sccid(n);
23     for (ll i = 0; i < n; i++)
24         if (!vis[i]) dfs(i, vis, g, scc, true, 0,
            sccid);
25
26     // build the transposed graph
27     vll2d gt(n);
28     for (int i = 0; i < n; ++i)
29         for (auto &v : g[i]) gt[v].eb(i);
30
31     // run the dfs on the previous scc order
32     ll id = 1;
33     vis.assign(n, false);
34     for (ll i = len(scc) - 1; i >= 0; i--)
35         if (!vis[scc[i]]) {
36             dfs(scc[i], vis, gt, scc, false, id++,
                sccid);
37         }
38     return {id - 1, sccid};
39 }

```

2.2 2-sat-(struct)

```

1  struct SAT2 {
2      ll n;
3      vll2d adj, adj_t;
4      vc used;
5      vll order, comp;
6      vc assignment;
7      bool solvable;
8      SAT2(ll _n)
9          : n(2 * _n),
10            adj(n),
11            adj_t(n),
12            used(n),
13            order(n),
14            comp(n, -1),
15            assignment(n / 2) {}
16  void dfs1(int v) {
17      used[v] = true;
18      for (int u : adj[v]) {
19          if (!used[u]) dfs1(u);
20      }
21      order.push_back(v);
22  }
23
24  void dfs2(int v, int cl) {
25      comp[v] = cl;
26      for (int u : adj_t[v]) {
27          if (comp[u] == -1) dfs2(u, cl);
28      }
29  }
30
31  bool solve_2SAT() {
32      // find and label each SCC
33      for (int i = 0; i < n; ++i) {
34          if (!used[i]) dfs1(i);
35      }

```

```

36     reverse(all(order));
37     ll j = 0;
38     for (auto &v : order) {
39         if (comp[v] == -1) dfs2(v, j++);
40     }
41
42     assignment.assign(n / 2, false);
43     for (int i = 0; i < n; i += 2) {
44         // x and !x belong to the same SCC
45         if (comp[i] == comp[i + 1]) {
46             solvable = false;
47             return false;
48         }
49
50         assignment[i / 2] = comp[i] > comp[i +
51             1];
52     }
53     solvable = true;
54     return true;
55 }
56
57 void add_disjunction(int a, bool na, int b, bool
    nb) {
58     a = (2 * a) ^ na;
59     b = (2 * b) ^ nb;
60     int neg_a = a ^ 1;
61     int neg_b = b ^ 1;
62     adj[neg_a].push_back(b);
63     adj[neg_b].push_back(a);
64     adj_t[b].push_back(neg_a);
65     adj_t[a].push_back(neg_b);
66 }

```

2.3 Floyd Warshall

```

1  vector<vll> floyd_warshall(const vector<vll> &adj, ll
    n) {
2      auto dist = adj;
3
4      for (int i = 0; i < n; ++i) {
5          for (int j = 0; j < n; ++j) {
6              for (int k = 0; k < n; ++k) {
7                  dist[j][k] = min(dist[j][k], dist[j][i]
9                      + dist[i][k]);
10             }
11         }
12     }
13     return dist;
14 }

```

2.4 Topological-sorting

```

1  /*
2  * O(V)
3  * assumes:
4  *     * vertices have index [0, n-1]
5  *     * if is a DAG:
6  *         * returns a topological sorting
7  *     * else:
8  *         * returns an empty vector
9  * */
10 enum class state { not_visited, processing, done };
11 bool dfs(const vector<vll> &adj, ll s, vector<state>
    &states, vll &order) {
12     states[s] = state::processing;
13     for (auto &v : adj[s]) {
14         if (states[v] == state::not_visited) {
15             if (not dfs(adj, v, states, order))
16                 return false;
17         } else if (states[v] == state::processing)
18             return false;
19     }
20     states[s] = state::done;
21     order.pb(s);
22     return true;
23 }

```

```

23 vll topologicalSorting(const vector<vll> &adj) {
24     ll n = len(adj);
25     vll order;
26     vector<state> states(n, state::not_visited);
27     for (int i = 0; i < n; ++i) {
28         if (states[i] == state::not_visited) {
29             if (not dfs(adj, i, states, order))
30                 return {};
31         }
32     }
33     reverse(all(order));
34     return order;
35 }

```

2.5 Lowest Common Ancestor Sparse Table

```

1 int fastlog2(ll x) {
2     ull i = x;
3     return i ? __builtin_clzll(1) - __builtin_clzll(i)
4         : -1;
5 }
6 template <typename T>
7 class SparseTable {
8 public:
9     int N;
10    int K;
11    vector<vector<T>> st;
12    SparseTable(vector<T> vs)
13        : N((int)vs.size()), K(fastlog2(N) + 1), st(K
14        + 1, vector<T>(N + 1)) {
15        copy(vs.begin(), vs.end(), st[0].begin());
16
17        for (int i = 1; i <= K; ++i)
18            for (int j = 0; j + (1 << i) <= N; ++j)
19                st[i][j] = min(st[i - 1][j], st[i -
20                1][j + (1 << (i - 1))]);
21    }
22    SparseTable() {}
23    T RMQ(int l, int r) {
24        int i = fastlog2(r - l + 1);
25        return min(st[i][l], st[i][r - (1 << i) + 1]);
26    }
27 };
28
29 class LCA {
30 public:
31     int p;
32     int n;
33     vi first;
34     vector<char> visited;
35     vi vertices;
36     vi height;
37     SparseTable<int> st;
38
39     LCA(const vector<vi> &g)
40         : p(0),
41           n((int)g.size()),
42           first(n + 1),
43           visited(n + 1, 0),
44           height(n + 1) {
45         build_dfs(g, 1, 1);
46         st = SparseTable<int>(vertices);
47     }
48
49     void build_dfs(const vector<vi> &g, int u, int hi) {
50         visited[u] = true;
51         height[u] = hi;
52         first[u] = vertices.size();
53         vertices.push_back(u);
54         for (auto uv : g[u]) {
55             if (!visited[uv]) {
56                 build_dfs(g, uv, hi + 1);
57                 vertices.push_back(uv);
58             }
59         }
60     }
61 }

```

```

57
58 int lca(int a, int b) {
59     int l = min(first[a], first[b]);
60     int r = max(first[a], first[b]);
61     return st.RMQ(l, r);
62 }
63 };

```

2.6 Count-scc-(kosajaru)

```

1 void dfs(ll u, vchar &visited, const vll2d &g, vll &
2     scc, bool buildScc) {
3     visited[u] = true;
4     for (auto &v : g[u])
5         if (!visited[v]) dfs(v, visited, g, scc,
6             buildScc);
7
8     // if it's the first pass, add the node to the
9     scc
10    if (buildScc) scc.pb(u);
11 }
12
13 ll kosajaru(vll2d &g) {
14     ll n = len(g);
15     vll scc;
16     vchar vis(n);
17     for (ll i = 0; i < n; ++i)
18         if (!vis[i]) dfs(i, vis, g, scc, true);
19
20     // build the transposed graph
21     vll2d gt(n);
22     for (int i = 0; i < n; ++i)
23         for (auto &v : g[i]) gt[v].pb(i);
24
25     // run the dfs on the previous scc order
26     ll scccnt = 0;
27     vis.assign(n, false);
28     for (ll i = len(scc) - 1; i >= 0; i--)
29         if (!vis[scc[i]]) dfs(scc[i], vis, gt, scc,
30             false), scccnt++;
31     return scccnt;
32 }

```

2.7 Kruskal

```

1 class DSU:
2     def __init__(self, n):
3         self.n = n
4         self.p = [x for x in range(0, n + 1)]
5         self.size = [0 for i in range(0, n + 1)]
6
7     def find_set(self, x):
8         if self.p[x] == x:
9             return x
10        else:
11            self.p[x] = self.find_set(self.p[x])
12            return self.p[x]
13
14    def same_set(self, x, y):
15        return bool(self.find_set(x) == self.find_set
16            (y))
17
18    def union_set(self, x, y):
19        px = self.find_set(x)
20        py = self.find_set(y)
21
22        if px == py:
23            return
24
25        size_x = self.size[px]
26        size_y = self.size[py]
27
28        if size_x > size_y:
29            self.p[py] = self.p[px]
30            self.size[px] += self.size[py]
31        else:
32            self.p[px] = self.p[py]

```

```

32         self.size[py] += self.size[px]
33
34
35 def kruskal(gv, n):
36     """
37     Receives te list of edges as a list of tuple in
38     the form:
39         d, u, v
40         d: distance between u and v
41     And also n as the total of verties.
42     """
43     dsu = DSU(n)
44
45     c = 0
46     for e in gv:
47         d, u, v = e
48         if not dsu.same_set(u, v):
49             c += d
50             dsu.union_set(u, v)
51
52     return c

```

2.8 Scc-(struct)

```

1 struct SCC {
2     ll N;
3     vll2d adj, tadj;
4     vll todo, comps, comp;
5     vector<set<ll>> sccadj;
6     vchar vis;
7     SCC(ll _N) : N(_N), adj(_N), tadj(_N), comp(_N,
8         -1), sccadj(_N), vis(_N) {}
9
10 void add_edge(ll x, ll y) { adj[x].eb(y), tadj[y]
11     .eb(x); }
12
13 void dfs(ll x) {
14     vis[x] = 1;
15     for (auto &y : adj[x])
16         if (!vis[y]) dfs(y);
17     todo.pb(x);
18 }
19
20 void dfs2(ll x, ll v) {
21     comp[x] = v;
22     for (auto &y : tadj[x])
23         if (comp[y] == -1) dfs2(y, v);
24 }
25
26 void gen() {
27     for (ll i = 0; i < N; ++i)
28         if (!vis[i]) dfs(i);
29     reverse(all(todo));
30     for (auto &x : todo)
31         if (comp[x] == -1) {
32             dfs2(x, x);
33             comps.pb(x);
34         }
35 }
36
37 void genSCCGraph() {
38     for (ll i = 0; i < N; ++i) {
39         for (auto &j : adj[i]) {
40             if (comp[i] != comp[j]) {
41                 sccadj[comp[i]].insert(comp[j]);
42             }
43         }
44     }
45 }
46
47 };

```

2.9 Check-bipartite

```

1 // O(V)
2 bool checkBipartite(const ll n, const vector<vll> &
3     adj) {
4     ll s = 0;
5     queue<ll> q;
6     q.push(s);

```

```

6     vll color(n, INF);
7     color[s] = 0;
8     bool isBipartite = true;
9     while (!q.empty() && isBipartite) {
10         ll u = q.front();
11         q.pop();
12         for (auto &v : adj[u]) {
13             if (color[v] == INF) {
14                 color[v] = 1 - color[u];
15                 q.push(v);
16             } else if (color[v] == color[u]) {
17                 return false;
18             }
19         }
20     }
21     return true;
22 }

```

2.10 Dijkstra

```

1 ll __inf = LLONG_MAX >> 5;
2 vll dijkstra(const vector<vector<pll>> &g, ll n) {
3     priority_queue<pll, vector<pll>, greater<pll>> pq
4     ;
5     vll dist(n, __inf);
6     vector<char> vis(n);
7     pq.emplace(0, 0);
8     dist[0] = 0;
9     while (!pq.empty()) {
10         auto [d1, v] = pq.top();
11         pq.pop();
12         if (vis[v]) continue;
13         vis[v] = true;
14
15         for (auto [d2, u] : g[v]) {
16             if (dist[u] > d1 + d2) {
17                 dist[u] = d1 + d2;
18                 pq.emplace(dist[u], u);
19             }
20         }
21     }
22     return dist;
23 }

```

3 extras

3.1 Binary To Gray

```

1 string binToGray(string bin) {
2     string gray(bin.size(), '0');
3     int n = bin.size() - 1;
4     gray[0] = bin[0];
5     for (int i = 1; i <= n; i++) {
6         gray[i] = '0' + (bin[i - 1] == '1') ^ (bin[i]
7             == '1');
8     }
9     return gray;
10 }

```

3.2 Bigint

```

1 const int maxn = 1e2 + 14, lg = 15;
2 const int base = 1000000000;
3 const int base_digits = 9;
4 struct bigint {
5     vector<int> a;
6     int sign;
7
8     int size() {
9         if (a.empty()) return 0;
10        int ans = (a.size() - 1) * base_digits;
11        int ca = a.back();
12        while (ca) ans++, ca /= 10;
13        return ans;
14    }

```

```

15  bigint operator~(const bigint &v) {
16      bigint ans = 1, a = *this, b = v;
17      while (!b.isZero()) {
18          if (b % 2) ans *= a;
19          a *= a, b /= 2;
20      }
21      return ans;
22  }
23  string to_string() {
24      stringstream ss;
25      ss << *this;
26      string s;
27      ss >> s;
28      return s;
29  }
30  int sumof() {
31      string s = to_string();
32      int ans = 0;
33      for (auto c : s) ans += c - '0';
34      return ans;
35  }
36  /*</arpa>*/
37  bigint() : sign(1) {}
38
39  bigint(long long v) { *this = v; }
40
41  bigint(const string &s) { read(s); }
42
43  void operator=(const bigint &v) {
44      sign = v.sign;
45      a = v.a;
46  }
47
48  void operator=(long long v) {
49      sign = 1;
50      a.clear();
51      if (v < 0) sign = -1, v = -v;
52      for (; v > 0; v = v / base) a.push_back(v %
base);
53  }
54
55  bigint operator+(const bigint &v) const {
56      if (sign == v.sign) {
57          bigint res = v;
58
59          for (int i = 0, carry = 0;
60              i < (int)max(a.size(), v.a.size())
61              || carry; ++i) {
62              if (i == (int)res.a.size()) res.a.
push_back(0);
63              res.a[i] += carry + (i < (int)a.size
64              () ? a[i] : 0);
65              carry = res.a[i] >= base;
66              if (carry) res.a[i] -= base;
67          }
68          return res;
69      }
70      return *this - (-v);
71  }
72
73  bigint operator-(const bigint &v) const {
74      if (sign == v.sign) {
75          if (abs() >= v.abs()) {
76              bigint res = *this;
77              for (int i = 0, carry = 0; i < (int)v.
a.size() || carry; ++i) {
78                  res.a[i] -= carry + (i < (int)v.
a.size() ? v.a[i] : 0);
79                  carry = res.a[i] < 0;
80                  if (carry) res.a[i] += base;
81              }
82              res.trim();
83              return res;
84          }
85          return -(v - *this);
86      }
87      return *this + (-v);
88  }
89
90  void operator*=(int v) {
91      if (v < 0) sign = -sign, v = -v;
92      for (int i = 0, carry = 0; i < (int)a.size()
93      || carry; ++i) {
94          if (i == (int)a.size()) a.push_back(0);
95          long long cur = a[i] * (long long)v +
96          carry;
97          carry = (int)(cur / base);
98          a[i] = (int)(cur % base);
99          // asm("divl %%ecx" : "=a"(carry), "=d"(a
[i]) :
100              // "A"(cur), "c"(base));
101          }
102      trim();
103  }
104
105  bigint operator*(int v) const {
106      bigint res = *this;
107      res *= v;
108      return res;
109  }
110
111  void operator*=(long long v) {
112      if (v < 0) sign = -sign, v = -v;
113      if (v > base) {
114          *this = *this * (v / base) * base + *this
115          * (v % base);
116          return;
117      }
118      for (int i = 0, carry = 0; i < (int)a.size()
119      || carry; ++i) {
120          if (i == (int)a.size()) a.push_back(0);
121          long long cur = a[i] * (long long)v +
122          carry;
123          carry = (int)(cur / base);
124          a[i] = (int)(cur % base);
125          // asm("divl %%ecx" : "=a"(carry), "=d"(a
[i]) :
126              // "A"(cur), "c"(base));
127          }
128      trim();
129  }
130
131  bigint operator*(long long v) const {
132      bigint res = *this;
133      res *= v;
134      return res;
135  }
136
137  friend pair<bigint, bigint> divmod(const bigint &
a1, const bigint &b1) {
138      int norm = base / (b1.a.back() + 1);
139      bigint a = a1.abs() * norm;
140      bigint b = b1.abs() * norm;
141      bigint q, r;
142      q.a.resize(a.a.size());
143
144      for (int i = a.a.size() - 1; i >= 0; i--) {
145          r *= base;
146          r += a.a[i];
147          int s1 = r.a.size() <= b.a.size() ? 0 : r
.a[b.a.size()];
148          int s2 = r.a.size() <= b.a.size() - 1 ? 0
: r.a[b.a.size() - 1];
149          int d = ((long long)base * s1 + s2) / b.a
.back();
150          r -= b * d;
151          while (r < 0) r += b, --d;
152          q.a[i] = d;
153      }
154
155      q.sign = a1.sign * b1.sign;
156      r.sign = a1.sign;
157      q.trim();
158      r.trim();

```



```

152     return make_pair(q, r / norm);
153 }
154
155 bigint operator/(const bigint &v) const { return
divmod(*this, v).first; }
156
157 bigint operator%(const bigint &v) const { return
divmod(*this, v).second; }
158
159 void operator/=(int v) {
160     if (v < 0) sign = -sign, v = -v;
161     for (int i = (int)a.size() - 1, rem = 0; i >=
0; --i) {
162         long long cur = a[i] + rem * (long long)
base;
163         a[i] = (int)(cur / v);
164         rem = (int)(cur % v);
165     }
166     trim();
167 }
168
169 bigint operator/(int v) const {
170     bigint res = *this;
171     res /= v;
172     return res;
173 }
174
175 int operator%(int v) const {
176     if (v < 0) v = -v;
177     int m = 0;
178     for (int i = a.size() - 1; i >= 0; --i)
179         m = (a[i] + m * (long long)base) % v;
180     return m * sign;
181 }
182
183 void operator+=(const bigint &v) { *this = *this
+ v; }
184 void operator-=(const bigint &v) { *this = *this
- v; }
185 void operator*=(const bigint &v) { *this = *this
* v; }
186 void operator/=(const bigint &v) { *this = *this
/ v; }
187
188 bool operator<(const bigint &v) const {
189     if (sign != v.sign) return sign < v.sign;
190     if (a.size() != v.a.size())
191         return a.size() * sign < v.a.size() * v.
sign;
192     for (int i = a.size() - 1; i >= 0; i--)
193         if (a[i] != v.a[i]) return a[i] * sign <
v.a[i] * sign;
194     return false;
195 }
196
197 bool operator>(const bigint &v) const { return v
< *this; }
198 bool operator<=(const bigint &v) const { return
!(v < *this); }
199 bool operator>=(const bigint &v) const { return
!(*this < v); }
200 bool operator==(const bigint &v) const {
201     return !(*this < v) && !(v < *this);
202 }
203 bool operator!=(const bigint &v) const { return
*this < v || v < *this; }
204
205 void trim() {
206     while (!a.empty() && !a.back()) a.pop_back();
207     if (a.empty()) sign = 1;
208 }
209
210 bool isZero() const { return a.empty() || (a.size
() == 1 && !a[0]); }
211
212 bigint operator-() const {
213     bigint res = *this;
214     res.sign = -sign;
215     return res;
216 }
217
218 bigint abs() const {
219     bigint res = *this;
220     res.sign *= res.sign;
221     return res;
222 }
223
224 long long longValue() const {
225     long long res = 0;
226     for (int i = a.size() - 1; i >= 0; i--) res =
res * base + a[i];
227     return res * sign;
228 }
229
230 friend bigint gcd(const bigint &a, const bigint &
b) {
231     return b.isZero() ? a : gcd(b, a % b);
232 }
233 friend bigint lcm(const bigint &a, const bigint &
b) {
234     return a / gcd(a, b) * b;
235 }
236
237 void read(const string &s) {
238     sign = 1;
239     a.clear();
240     int pos = 0;
241     while (pos < (int)s.size() && (s[pos] == '-'
|| s[pos] == '+')) {
242         if (s[pos] == '-') sign = -sign;
243         ++pos;
244     }
245     for (int i = s.size() - 1; i >= pos; i -=
base_digits) {
246         int x = 0;
247         for (int j = max(pos, i - base_digits +
1); j <= i; j++)
248             x = x * 10 + s[j] - '0';
249         a.push_back(x);
250     }
251     trim();
252 }
253
254 friend istream &operator>>(istream &stream,
bigint &v) {
255     string s;
256     stream >> s;
257     v.read(s);
258     return stream;
259 }
260
261 friend ostream &operator<<(ostream &stream, const
bigint &v) {
262     if (v.sign == -1) stream << '-';
263     stream << (v.a.empty() ? 0 : v.a.back());
264     for (int i = (int)v.a.size() - 2; i >= 0; --i)
265         stream << setw(base_digits) << setfill('0
') << v.a[i];
266     return stream;
267 }
268
269 static vector<int> convert_base(const vector<int>
&a, int old_digits,
270                                int new_digits) {
271     vector<long long> p(max(old_digits,
new_digits) + 1);
272     p[0] = 1;
273     for (int i = 1; i < (int)p.size(); i++) p[i]
= p[i - 1] * 10;
274     vector<int> res;
275     long long cur = 0;
276     int cur_digits = 0;
277     for (int i = 0; i < (int)a.size(); i++) {

```

```

278         cur += a[i] * p[cur_digits];
279         cur_digits += old_digits;
280         while (cur_digits >= new_digits) {
281             res.push_back(int(cur % p[new_digits
]));
282         cur /= p[new_digits];
283         cur_digits -= new_digits;
284     }
285 }
286 res.push_back((int)cur);
287 while (!res.empty() && !res.back()) res.
pop_back();
288 return res;
289 }
290
291 typedef vector<long long> vll;
292
293 static vll karatsubaMultiply(const vll &a, const
vll &b) {
294     int n = a.size();
295     vll res(n + n);
296     if (n <= 32) {
297         for (int i = 0; i < n; i++)
298             for (int j = 0; j < n; j++) res[i + j
] += a[i] * b[j];
299         return res;
300     }
301
302     int k = n >> 1;
303     vll a1(a.begin(), a.begin() + k);
304     vll a2(a.begin() + k, a.end());
305     vll b1(b.begin(), b.begin() + k);
306     vll b2(b.begin() + k, b.end());
307
308     vll a1b1 = karatsubaMultiply(a1, b1);
309     vll a2b2 = karatsubaMultiply(a2, b2);
310
311     for (int i = 0; i < k; i++) a2[i] += a1[i];
312     for (int i = 0; i < k; i++) b2[i] += b1[i];
313
314     vll r = karatsubaMultiply(a2, b2);
315     for (int i = 0; i < (int)a1b1.size(); i++) r[
i] -= a1b1[i];
316     for (int i = 0; i < (int)a2b2.size(); i++) r[
i] -= a2b2[i];
317
318     for (int i = 0; i < (int)r.size(); i++) res[i
+ k] += r[i];
319     for (int i = 0; i < (int)a1b1.size(); i++)
res[i] += a1b1[i];
320     for (int i = 0; i < (int)a2b2.size(); i++)
res[i + n] += a2b2[i];
321     return res;
322 }
323
324 bigint operator*(const bigint &v) const {
325     vector<int> a6 = convert_base(this->a,
base_digits, 6);
326     vector<int> b6 = convert_base(v.a,
base_digits, 6);
327     vll a(a6.begin(), a6.end());
328     vll b(b6.begin(), b6.end());
329     while (a.size() < b.size()) a.push_back(0);
330     while (b.size() < a.size()) b.push_back(0);
331     while (a.size() & (a.size() - 1)) a.push_back
(0), b.push_back(0);
332     vll c = karatsubaMultiply(a, b);
333     bigint res;
334     res.sign = sign * v.sign;
335     for (int i = 0, carry = 0; i < (int)c.size();
i++) {
336         long long cur = c[i] + carry;
337         res.a.push_back((int)(cur % 1000000));
338         carry = (int)(cur / 1000000);
339     }
340     res.a = convert_base(res.a, 6, base_digits);
341     res.trim();

```

```

342         return res;
343     }
344 };

```

3.3 Get-permutation-cicles

```

1  /*
2  * receives a permutation [0, n-1]
3  * returns a vector of cicles
4  * for example: [ 1, 0, 3, 4, 2] -> [[0, 1], [2, 3,
5  * */
6  vector<vll> getPermutationCicles(const vll &ps) {
7      ll n = len(ps);
8      vector<char> visited(n);
9      vector<vll> cicles;
10     for (int i = 0; i < n; ++i) {
11         if (visited[i]) continue;
12
13         vll cicle;
14         ll pos = i;
15         while (!visited[pos]) {
16             cicle.pb(pos);
17             visited[pos] = true;
18             pos = ps[pos];
19         }
20
21         cicles.push_back(vll(all(cicle)));
22     }
23     return cicles;
24 }

```

4 dynamic programming

4.1 Edit Distance

```

1  int edit_distance(const string &a, const string &b) {
2      int n = a.size();
3      int m = b.size();
4      vector<vi> dp(n + 1, vi(m + 1, 0));
5
6      int ADD = 1, DEL = 1, CHG = 1;
7      for (int i = 0; i <= n; ++i) {
8          dp[i][0] = i * DEL;
9      }
10     for (int i = 1; i <= m; ++i) {
11         dp[0][i] = ADD * i;
12     }
13
14     for (int i = 1; i <= n; ++i) {
15         for (int j = 1; j <= m; ++j) {
16             int add = dp[i][j - 1] + ADD;
17             int del = dp[i - 1][j] + DEL;
18             int chg = dp[i - 1][j - 1] + (a[i - 1] ==
b[j - 1] ? 0 : 1) * CHG;
19             dp[i][j] = min({add, del, chg});
20         }
21     }
22
23     return dp[n][m];
24 }

```

4.2 Money Sum Bottom Up

```

1  /*
2  find every possible sum using
3  the given values only once.
4  */
5  set<int> money_sum(const vi &xs) {
6      using vc = vector<char>;
7      using vvc = vector<vc>;
8      int _m = accumulate(all(xs), 0);
9      int _n = xs.size();
10     vvc _dp(_n + 1, vc(_m + 1, 0));
11     set<int> _ans;

```

```

12     _dp[0][xs[0]] = 1;
13     for (int i = 1; i < _n; ++i) {
14         for (int j = 0; j <= _m; ++j) {
15             if (j == 0 or _dp[i - 1][j]) {
16                 _dp[i][j + xs[i]] = 1;
17                 _dp[i][j] = 1;
18             }
19         }
20     }
21
22     for (int i = 0; i < _n; ++i)
23         for (int j = 0; j <= _m; ++j)
24             if (_dp[i][j]) _ans.insert(j);
25     return _ans;
26 }

```

4.3 Knapsack Dp Values 01

```

1  const int MAX_N = 1001;
2  const int MAX_S = 100001;
3  array<array<int, MAX_S>, MAX_N> dp;
4  bool check[MAX_N][MAX_S];
5  pair<int, vi> knapsack(int S, const vector<pii> &xs)
6  {
7      int N = (int)xs.size();
8
9      for (int i = 0; i <= N; ++i) dp[i][0] = 0;
10
11     for (int m = 0; m <= S; ++m) dp[0][m] = 0;
12
13     for (int i = 1; i <= N; ++i) {
14         for (int m = 1; m <= S; ++m) {
15             dp[i][m] = dp[i - 1][m];
16             check[i][m] = false;
17
18             auto [w, v] = xs[i - 1];
19
20             if (w <= m and (dp[i - 1][m - w] + v) >=
21                 dp[i][m]) {
22                 dp[i][m] = dp[i - 1][m - w] + v;
23                 check[i][m] = true;
24             }
25         }
26     }
27
28     int m = S;
29     vi es;
30
31     for (int i = N; i >= 1; --i) {
32         if (check[i][m]) {
33             es.push_back(i);
34             m -= xs[i - 1].first;
35         }
36     }
37
38     reverse(es.begin(), es.end());
39
40     return {dp[N][S], es};
41 }

```

4.4 Tsp

```

1  using vi = vector<int>;
2  vector<vi> dist;
3  vector<vi> memo;
4  /* 0 (N^2 * 2^N) */
5  int tsp(int i, int mask, int N) {
6      if (mask == (1 << N) - 1) return dist[i][0];
7      if (memo[i][mask] != -1) return memo[i][mask];
8      int ans = INT_MAX << 1;
9      for (int j = 0; j < N; ++j) {
10         if (mask & (1 << j)) continue;
11         auto t = tsp(j, mask | (1 << j), N) + dist[i
12             ][j];
13         ans = min(ans, t);
14     }
15     return memo[i][mask] = ans;
16 }

```

5 trees

5.1 Binary-lifting

```

1  /*
2   * far[h][i] = the node that 2^h far from node i
3   * sometimes is useful invert the order of loops
4   * time : O(nlogn)
5   */
6  const int maxlog = 20;
7  int far[maxlog + 1][n + 1];
8  int n;
9  for (int h = 1; h <= maxlog; h++) {
10     for (int i = 1; i <= n; i++) {
11         far[h][i] = far[h - 1][far[h - 1][i]];
12     }
13 }

```

5.2 Maximum-distances

```

1  /*
2   * Returns the maximum distance from every node to
3   * any other node in the tree.
4   */
5  pll mostDistantFrom(const vector<vll> &adj, ll n, ll
6      root) {
7      // 0 indexed
8      ll mostDistantNode = root;
9      ll nodeDistance = 0;
10     queue<pll> q;
11     vector<char> vis(n);
12     q.emplace(root, 0);
13     vis[root] = true;
14     while (!q.empty()) {
15         auto [node, dist] = q.front();
16         q.pop();
17         if (dist > nodeDistance) {
18             nodeDistance = dist;
19             mostDistantNode = node;
20         }
21         for (auto u : adj[node]) {
22             if (!vis[u]) {
23                 vis[u] = true;
24                 q.emplace(u, dist + 1);
25             }
26         }
27     }
28     return {mostDistantNode, nodeDistance};
29 }
30
31 ll twoNodesDist(const vector<vll> &adj, ll n, ll a,
32     ll b) {
33     queue<pll> q;
34     vector<char> vis(n);
35     q.emplace(a, 0);
36     while (!q.empty()) {
37         auto [node, dist] = q.front();
38         q.pop();
39         if (node == b) return dist;
40         for (auto u : adj[node]) {
41             if (!vis[u]) {
42                 vis[u] = true;
43                 q.emplace(u, dist + 1);
44             }
45         }
46     }
47     return -1;
48 }
49
50 tuple<ll, ll, ll> tree_diameter(const vector<vll> &
51     adj, ll n) {
52     // returns two points of the diameter and the
53     // diameter itself
54     auto [node1, dist1] = mostDistantFrom(adj, n, 0);
55 }

```

```

50     auto [node2, dist2] = mostDistantFrom(adj, n,
51     node1);
52     auto diameter = twoNodesDist(adj, n, node1, node2
53     );
54     return make_tuple(node1, node2, diameter);
55 }
56
57 vll everyDistanceFromNode(const vector<vll> &adj, ll
58 n, ll root) {
59     // Single Source Shortest Path, from a given root
60     queue<pair<ll, ll>> q;
61     vll ans(n, -1);
62     ans[root] = 0;
63     q.emplace(root, 0);
64     while (!q.empty()) {
65         auto [u, d] = q.front();
66         q.pop();
67         for (auto w : adj[u]) {
68             if (ans[w] != -1) continue;
69             ans[w] = d + 1;
70             q.emplace(w, d + 1);
71         }
72     }
73     return ans;
74 }
75
76 vll maxDistances(const vector<vll> &adj, ll n) {
77     auto [node1, node2, diameter] = tree_diameter(adj
78     , n);
79     auto distances1 = everyDistanceFromNode(adj, n,
80     node1);
81     auto distances2 = everyDistanceFromNode(adj, n,
82     node2);
83     vll ans(n);
84     for (int i = 0; i < n; ++i) ans[i] = max(
85     distances1[i], distances2[i]);
86     return ans;
87 }

```

```

35 #define rsz(____x, ____n) resize(____x, ____n)
36
37 const ll INF = 1e18;
38
39 struct HLD {
40     int n;
41     vi sizes;
42     vi2d g;
43     vi groups;
44     vi heavy;
45     HLD(int n) : n(n), sizes(n + 1), g(n + 1), groups
46     (n + 1), heavy(n + 1) {}
47     void get_sizes(int u, int p) {
48         int sz = 1;
49         int bigc = -1;
50         for (auto &v : g[u])
51             if (v != p) {
52                 get_sizes(p, u);
53                 if (bigc == -1 or sizes[bigc] < sizes
54                 [v])
55                     bigc = v, heavy[u] = v;
56                 sz += sizes[v];
57             }
58         sizes[u] = sz;
59     }
60     void decompose(int u, int p) {
61         groups[u] = p;
62         for (auto &v : g[u])
63             if (v != p) {
64                 if (v == heavy[u])
65                     decompose(v, p);
66                 else
67                     decompose(v, v);
68             }
69     }
70     void run() {}
71     int32_t main(void) {
72         fastio;
73         int t;
74         t = 1;
75         // cin >> t;
76         while (t--) run();
77     }

```

5.3 Heavy-light-decomposition

```

1 // iagorrr ;)
2 #include <bits/stdc++.h>
3 using namespace std;
4 #ifdef DEBUG
5 #include "debug.cpp"
6 #else
7 #define dbg(...) 666
8 #endif
9 #define endl '\n'
10 #define fastio \
11     ios_base::sync_with_stdio(false); \
12     cin.tie(0); \
13     cout.tie(0);
14 #define rep(i, l, r) for (int i = (l); i < (r); i++)
15 #define len(__x) (ll) __x.size()
16 using ll = long long;
17 using vll = vector<ll>;
18 using pll = pair<ll, ll>;
19 using vll2d = vector<vll>;
20 using vi = vector<int>;
21 using vi2d = vector<vi>;
22 using pii = pair<int, int>;
23 using vii = vector<pii>;
24 using vc = vector<char>;
25 #define all(a) a.begin(), a.end()
26 #define INV(____x) \
27     for (auto &xxxx : xxxx) cin >> xxx;
28 #define PRINTV(____x) \
29     for_each(all(____x), [](ll &____x) { cout << ____x
30     << ' '; }, cout << '\n');
31 #define snd second
32 #define fst first
33 #define pb(____x) push_back(____x)
34 #define mp(____a, ____b) make_pair(____a, ____b)
35 #define eb(____x) emplace_back(____x)

```

5.4 Tree Diameter

```

1 pll mostDistantFrom(const vector<vll> &adj, ll n, ll
2 root) {
3     // 0 indexed
4     ll mostDistantNode = root;
5     ll nodeDistance = 0;
6     queue<pll> q;
7     vector<char> vis(n);
8     q.emplace(root, 0);
9     vis[root] = true;
10    while (!q.empty()) {
11        auto [node, dist] = q.front();
12        q.pop();
13        if (dist > nodeDistance) {
14            nodeDistance = dist;
15            mostDistantNode = node;
16        }
17        for (auto u : adj[node]) {
18            if (!vis[u]) {
19                vis[u] = true;
20                q.emplace(u, dist + 1);
21            }
22        }
23    }
24    return {mostDistantNode, nodeDistance};
25 }
26
27 ll twoNodesDist(const vector<vll> &adj, ll n, ll a,
28 ll b) {
29     // 0 indexed
30     queue<pll> q;
31     vector<char> vis(n);

```

```

29     q.emplace(a, 0);
30     while (!q.empty()) {
31         auto [node, dist] = q.front();
32         q.pop();
33         if (node == b) {
34             return dist;
35         }
36         for (auto u : adj[node]) {
37             if (!vis[u]) {
38                 vis[u] = true;
39                 q.emplace(u, dist + 1);
40             }
41         }
42     }
43     return -1;
44 }
45 ll tree_diameter(const vector<vll> &adj, ll n) {
46     // 0 indexed !!!
47     auto [node1, dist1] = mostDistantFrom(adj, n, 0);
48     auto [node2, dist2] = mostDistantFrom(adj, n,
49     node1);
50     auto diameter = twoNodesDist(adj, n, node1, node2
51 );
52     return diameter;
53 }

```

6 searching

6.1 Ternary Search Recursive

```

1 const double eps = 1e-6;
2
3 // IT MUST BE AN UNIMODAL FUNCTION
4 double f(int x) { return x * x + 2 * x + 4; }
5
6 double ternary_search(double l, double r) {
7     if (fabs(f(l) - f(r)) < eps) return f((l + (r - l
8     ) / 2.0));
9
10    auto third = (r - l) / 3.0;
11    auto m1 = l + third;
12    auto m2 = r - third;
13
14    // change the signal to find the maximum point.
15    return m1 < m2 ? ternary_search(m1, r) :
16    ternary_search(l, m2);
17 }

```

7 math

7.1 Power-sum

```

1 // calculates K^0 + K^1 ... + K^n
2 ll fastpow(ll a, int n) {
3     if (n == 1) return a;
4     ll x = fastpow(a, n / 2);
5     return x * x * (n & 1 ? a : 1);
6 }
7 ll powersum(ll n, ll k) { return (fastpow(n, k + 1) -
8     1) / (n - 1); }

```

7.2 Sieve-list-primes

```

1 // list every prime until MAXN
2 const ll MAXN = 1e5;
3 vll list_primes(ll n) { // Nlog * log N
4     vll ps;
5     bitset<MAXN> sieve;
6     sieve.set();
7     sieve.reset(1);
8     for (ll i = 2; i <= n; ++i) {
9         if (sieve[i]) ps.push_back(i);
10        for (ll j = i * 2; j <= n; j += i) {
11            sieve.reset(j);

```

```

12        }
13    }
14    return ps;
15 }

```

7.3 Factorial

```

1 const ll MAX = 18;
2 vll fv(MAX, -1);
3 ll factorial(ll n) {
4     if (fv[n] != -1) return fv[n];
5     if (n == 0) return 1;
6     return n * factorial(n - 1);
7 }

```

7.4 Permutation-count

```

1 const ll MAX = 18;
2 vll fv(MAX, -1);
3 ll factorial(ll n) {
4     if (fv[n] != -1) return fv[n];
5     if (n == 0) return 1;
6     return n * factorial(n - 1);
7 }
8
9 template <typename T>
10 ll permutation_count(vector<T> xs) {
11     map<T, ll> h;
12     for (auto xi : xs) h[xi]++;
13     ll ans = factorial((ll)xs.size());
14     dbg(ans);
15     for (auto [v, cnt] : h) {
16         dbg(cnt);
17         ans /= cnt;
18     }
19
20     return ans;
21 }

```

7.5 N-choose-k-count

```

1 /*
2  * O(nm) time, O(m) space
3  * equal to n choose k
4  */
5 ll binom(ll n, ll k) {
6     if (k > n) return 0;
7     vll dp(k + 1, 0);
8     dp[0] = 1;
9     for (ll i = 1; i <= n; ++i)
10        for (ll j = k; j > 0; j--) dp[j] = dp[j] + dp
11        [j - 1];
12    return dp[k];

```

7.6 Gcd-using-factorization

```

1 // O(sqrt(n))
2 map<ll, ll> factorization(ll n) {
3     map<ll, ll> ans;
4     for (ll i = 2; i * i <= n; ++i) {
5         ll count = 0;
6         for (; n % i == 0; count++, n /= i)
7             ;
8         if (count) ans[i] = count;
9     }
10    if (n > 1) ans[n]++;
11    return ans;
12 }
13
14 ll gcd_with_factorization(ll a, ll b) {
15     map<ll, ll> fa = factorization(a);
16     map<ll, ll> fb = factorization(b);
17     ll ans = 1;
18     for (auto fai : fa) {
19         ll k = min(fai.second, fb[fai.first]);

```

```

20     while (k--) ans *= fai.first;
21 }
22 return ans;
23 }

```

7.7 Is-prime

```

1 bool isprime(ll n) { // O(sqrt(n))
2     if (n < 2) return false;
3     if (n == 2) return true;
4     if (n % 2 == 0) return false;
5     for (ll i = 3; i * i < n; i += 2)
6         if (n % i == 0) return false;
7     return true;
8 }

```

7.8 Fast Exp

```

1 /*
2  Fast exponentiation algorithm,
3  compute a^n in O(log(n))
4 */
5 ll fexp(ll a, int n) {
6     if (n == 0) return 1;
7     if (n == 1) return a;
8     ll x = fexp(a, n / 2);
9     return x * x * (n & 1 ? a : 1);
10 }

```

7.9 Lcm-using-factorization

```

1 map<ll, ll> factorization(ll n) {
2     map<ll, ll> ans;
3     for (ll i = 2; i * i <= n; i++) {
4         ll count = 0;
5         for (; n % i == 0; count++, n /= i)
6             ;
7         if (count) ans[i] = count;
8     }
9     if (n > 1) ans[n]++;
10    return ans;
11 }
12
13 ll lcm_with_factorization(ll a, ll b) {
14     map<ll, ll> fa = factorization(a);
15     map<ll, ll> fb = factorization(b);
16     ll ans = 1;
17     for (auto fai : fa) {
18         ll k = max(fai.second, fb[fai.first]);
19         while (k--) ans *= fai.first;
20     }
21     return ans;
22 }

```

7.10 Euler-phi

```

1 const ll MAXN = 1e5;
2 vll list_primes(ll n) { // Nlog * log N
3     vll ps;
4     bitset<MAXN> sieve;
5     sieve.set();
6     sieve.reset(1);
7     for (ll i = 2; i <= n; ++i) {
8         if (sieve[i]) ps.push_back(i);
9         for (ll j = i * 2; j <= n; j += i) {
10             sieve.reset(j);
11         }
12     }
13     return ps;
14 }
15
16 vector<pll> factorization(ll n, const vll &primes) {
17     vector<pll> ans;
18     for (auto &p : primes) {
19         if (n == 1) break;
20         ll cnt = 0;

```

```

21         while (n % p == 0) {
22             cnt++;
23             n /= p;
24         }
25         if (cnt) ans.emplace_back(p, cnt);
26     }
27     return ans;
28 }
29
30 ll phi(ll n, vector<pll> factors) {
31     if (n == 1) return 1;
32     ll ans = n;
33
34     for (auto [p, k] : factors) {
35         ans /= p;
36         ans *= (p - 1);
37     }
38
39     return ans;
40 }

```

7.11 Polynomial

```

1 using polynomial = vector<ll>;
2 int degree(const polynomial &xs) { return xs.size() - 1; }
3 ll horner_evaluate(const polynomial &xs, ll x) {
4     ll ans = 0;
5     ll n = degree(xs);
6     for (int i = n; i >= 0; --i) {
7         ans *= x;
8         ans += xs[i];
9     }
10    return ans;
11 }
12 polynomial operator+(const polynomial &a, const
13     polynomial &b) {
14     int n = degree(a);
15     int m = degree(b);
16     polynomial r(max(n, m) + 1, 0);
17
18     for (int i = 0; i <= n; ++i) r[i] += a[i];
19     for (int j = 0; j <= m; ++j) r[j] += b[j];
20     while (!r.empty() and r.back() == 0) r.pop_back();
21     if (r.empty()) r.push_back(0);
22     return r;
23 }
24 polynomial operator*(const polynomial &p, const
25     polynomial &q) {
26     int n = degree(p);
27     int m = degree(q);
28     polynomial r(n + m + 1, 0);
29     for (int i = 0; i <= n; ++i)
30         for (int j = 0; j <= m; ++j) r[i + j] += (p[i]

```

7.12 Integer Mod

```

1 const ll INF = 1e18;
2 const ll mod = 998244353;
3 template <ll MOD = mod>
4 struct Modular {
5     ll value;
6     static const ll MOD_value = MOD;
7
8     Modular(ll v = 0) {
9         value = v % MOD;
10        if (value < 0) value += MOD;
11    }
12
13    Modular(ll a, ll b) : value(0) {
14        *this += a;
15        *this /= b;
16    }

```

```

17 Modular& operator+=(Modular const& b) {
18     value += b.value;
19     if (value >= MOD) value -= MOD;
20     return *this;
21 }
22 Modular& operator--(Modular const& b) {
23     value -= b.value;
24     if (value < 0) value += MOD;
25     return *this;
26 }
27 Modular& operator*=(Modular const& b) {
28     value = (ll)value * b.value % MOD;
29     return *this;
30 }
31
32 friend Modular mexp(Modular a, ll e) {
33     Modular res = 1;
34     while (e) {
35         if (e & 1) res *= a;
36         a *= a;
37         e >>= 1;
38     }
39     return res;
40 }
41 friend Modular inverse(Modular a) { return mexp(a
, MOD - 2); }
42
43 Modular& operator/=(Modular const& b) { return *
this *= inverse(b); }
44 friend Modular operator+(Modular a, Modular const
b) { return a += b; }
45 Modular operator++(int) { return this->value = (
this->value + 1) % MOD; }
46 Modular operator++() { return this->value = (this
->value + 1) % MOD; }
47 friend Modular operator-(Modular a, Modular const
b) { return a -= b; }
48 friend Modular operator--(Modular const a) {
return 0 - a; }
49 Modular operator--(int) {
50     return this->value = (this->value - 1 + MOD)
% MOD;
51 }
52
53 Modular operator--() { return this->value = (this
->value - 1 + MOD) % MOD; }
54 friend Modular operator*(Modular a, Modular const
b) { return a *= b; }
55 friend Modular operator/(Modular a, Modular const
b) { return a /= b; }
56 friend std::ostream& operator<<(std::ostream& os,
Modular const& a) {
57     return os << a.value;
58 }
59 friend bool operator==(Modular const& a, Modular
const& b) {
60     return a.value == b.value;
61 }
62 friend bool operator!=(Modular const& a, Modular
const& b) {
63     return a.value != b.value;
64 }
65 };

```

7.13 Count Divisors Memo

```

1 const ll mod = 1073741824;
2 const ll maxd = 100 * 100 * 100 + 1;
3 vector<ll> memo(maxd, -1);
4 ll countdivisors(ll x) {
5     ll ox = x;
6     ll ans = 1;
7     for (ll i = 2; i <= x; ++i) {
8         if (memo[x] != -1) {
9             ans *= memo[x];
10            break;
11        }

```

```

12        ll count = 0;
13        while (x and x % i == 0) {
14            x /= i;
15            count++;
16        }
17        ans *= (count + 1);
18    }
19    memo[ox] = ans;
20    return ans;
21 }

```

7.14 Lcm

```

1 ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }
2 ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }

```

7.15 Factorial-factorization

```

1 // O(logN) greater k that p^k | n
2 ll E(ll n, ll p) {
3     ll k = 0, b = p;
4     while (b <= n) {
5         k += n / b;
6         b *= p;
7     }
8     return k;
9 }
10
11 // lsit every prime until MAXN O(Nlog * log N)
12 const ll MAXN = 1e5;
13 vll list_primes(ll n) {
14     vll ps;
15     bitset<MAXN> sieve;
16     sieve.set();
17     sieve.reset(1);
18     for (ll i = 2; i <= n; ++i) {
19         if (sieve[i]) ps.push_back(i);
20         for (ll j = i * 2; j <= n; j += i) sieve.
reset(j);
21     }
22     return ps;
23 }
24
25 // O(pi(N)*logN)
26 map<ll, ll> factorial_factorization(ll n, const vll &
primes) {
27     map<ll, ll> fs;
28     for (const auto &p : primes) {
29         if (p > n) break;
30         fs[p] = E(n, p);
31     }
32     return fs;
33 }

```

7.16 Factorization-with-primes

```

1 // Nlog * log N
2 const ll MAXN = 1e5;
3 vll list_primes(ll n) {
4     vll ps;
5     bitset<MAXN> sieve;
6     sieve.set();
7     sieve.reset(1);
8     for (ll i = 2; i <= n; ++i) {
9         if (sieve[i]) ps.push_back(i);
10        for (ll j = i * 2; j <= n; j += i) sieve.
reset(j);
11    }
12    return ps;
13 }
14
15 // O(pi(sqrt(n)))
16 map<ll, ll> factorization(ll n, const vll &primes) {
17     map<ll, ll> ans;
18     for (auto p : primes) {
19         if (p * p > n) break;

```



```

20         ll count = 0;
21         for (; n % p == 0; count++, n /= p)
22             ;
23         if (count) ans[p] = count;
24     }
25     return ans;
26 }

```

7.17 Modular-inverse-using-phi

```

1 map<ll, ll> factorization(ll n) {
2     map<ll, ll> ans;
3     for (ll i = 2; i * i <= n; i++) {
4         ll count = 0;
5         for (; n % i == 0; count++, n /= i)
6             ;
7         if (count) ans[i] = count;
8     }
9     if (n > 1) ans[n]++;
10    return ans;
11 }
12
13 ll phi(ll n) {
14     if (n == 1) return 1;
15
16     auto fs = factorization(n);
17     auto res = n;
18
19     for (auto [p, k] : fs) {
20         res /= p;
21         res *= (p - 1);
22     }
23
24     return res;
25 }
26
27 ll fexp(ll a, ll n, ll mod) {
28     if (n == 0) return 1;
29     if (n == 1) return a;
30     ll x = fexp(a, n / 2, mod);
31     return x * x * (n & 1 ? a : 1) % mod;
32 }
33
34 ll inv(ll a, ll mod) { return fexp(a, phi(mod) - 1, mod); }

```

7.18 Factorization

```

1 // O(sqrt(n))
2 map<ll, ll> factorization(ll n) {
3     map<ll, ll> ans;
4     for (ll i = 2; i * i <= n; i++) {
5         ll count = 0;
6         for (; n % i == 0; count++, n /= i)
7             ;
8         if (count) ans[i] = count;
9     }
10    if (n > 1) ans[n]++;
11    return ans;
12 }

```

7.19 Gcd

```

1 ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }

```

7.20 Combinatorics With Repetitions

```

1 void combinations_with_repetition(int n, int k,
2                                   function<void(const
3                                   vector<int> &> process) {
4     vector<int> v(k, 1);
5     int pos = k - 1;
6
7     while (true) {
8         process(v);

```

```

9         v[pos]++;
10
11         while (pos > 0 and v[pos] > n) {
12             --pos;
13             v[pos]++;
14         }
15
16         if (pos == 0 and v[pos] > n) break;
17
18         for (int i = pos + 1; i < k; ++i) v[i] = v[
19             pos];
20         pos = k - 1;
21     }
22 }

```

8 strings

8.1 Rabin-karp

```

1 vi rabin_karp(string const &s, string const &t) {
2     ll p = 31;
3     ll m = 1e9 + 9;
4     int S = s.size(), T = t.size();
5
6     vll p_pow(max(S, T));
7     p_pow[0] = 1;
8     for (int i = 1; i < (int)p_pow.size(); i++)
9         p_pow[i] = (p_pow[i - 1] * p) % m;
10
11     vll h(T + 1, 0);
12     for (int i = 0; i < T; i++)
13         h[i + 1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i
14             ]) % m;
15     ll h_s = 0;
16     for (int i = 0; i < S; i++) h_s = (h_s + (s[i] -
17         'a' + 1) * p_pow[i]) % m;
18
19     vi occurrences;
20     for (int i = 0; i + S - 1 < T; i++) {
21         ll cur_h = (h[i + S] + m - h[i]) % m;
22         // IT DON'T CONSIDERE CONLIIONS !
23         if (cur_h == h_s * p_pow[i] % m) occurrences.
24         push_back(i);
25     }
26     return occurrences;
27 }

```

8.2 Trie-naive

```

1 // time: O(n^2) memory: O(n^2)
2 using Node = map<char, int>;
3 using vi = vector<int>;
4 using Trie = vector<Node>;
5
6 Trie build(const string &s) {
7     int n = (int)s.size();
8     Trie trie(1);
9     string suffix;
10
11     for (int i = n - 1; i >= 0; --i) {
12         suffix = s.substr(i) + '#';
13
14         int v = 0; // root
15         for (auto c : suffix) {
16             if (c == '#') { // makrs the poistion of
17                 an occurrence
18                 trie[v][c] = i;
19                 break;
20             }
21             if (trie[v][c])
22                 v = trie[v][c];
23             else {
24                 trie.push_back({});
25                 trie[v][c] = trie.size() - 1;
26                 v = trie.size() - 1;

```



```

26     }
27 }
28 }
29 return trie;
30 }
31
32 vi search(Trie &trie, string s) {
33     int p = 0;
34     vi occ;
35     for (auto &c : s) {
36         p = trie[p][c];
37         if (!p) return occ;
38     }
39
40     queue<int> q;
41     q.push(0);
42     while (!q.empty()) {
43         auto cur = q.front();
44         q.pop();
45         for (auto [c, v] : trie[cur]) {
46             if (c == '#')
47                 occ.push_back(v);
48             else
49                 q.push(v);
50         }
51     }
52     return occ;
53 }
54
55 ll distinct_substr(const Trie &trie) {
56     ll cnt = 0;
57     queue<int> q;
58     q.push(0);
59     while (!q.empty()) {

```

```

60         auto u = q.front();
61         q.pop();
62
63         for (auto [c, v] : trie[u]) {
64             if (c != '#') {
65                 cnt++;
66                 q.push(v);
67             }
68         }
69     }
70     return cnt;
71 }

```

8.3 String-psum

```

1 struct strPsum {
2     ll n;
3     ll k;
4     vector<vll> psum;
5     strPsum(const string &s) : n(s.size()), k(100),
6         psum(k, vll(n + 1)) {
7         for (ll i = 1; i <= n; ++i) {
8             for (ll j = 0; j < k; ++j) {
9                 psum[j][i] = psum[j][i - 1];
10            }
11            psum[s[i - 1]][i]++;
12        }
13    }
14    ll qtd(ll l, ll r, char c) { // [0,n-1]
15        return psum[c][r + 1] - psum[c][l];
16    }
17 }

```