

## Contents

## 1 Data Structures

1.1	Binary Trie	1
1.2	Disjoint Set Union	1
1.3	Minimum Queue	1
1.4	Mo	2
1.5	Range Add Point Query	2
1.6	Range Add Range Query	2
1.7	Segment Tree	3
1.8	Segment Tree 2d	3
1.9	Sparse Table	3
1.10	Sparse Table 2d	3
1.11	Sqrt Decomposition	4

## 2 Dynamic Programming

2.1	Divide And Conquer	4
2.2	Edit Distance	4
2.3	Knapsack	4
2.4	Knuth Optimization	4
2.5	Longest Common Subsequence	4
2.6	Longest Increasing Subsequence	5
2.7	Max Sum	5
2.8	Subset Sum	5

## 3 Geometry

3.1	Areas	5
3.2	Basic Geometry	5
3.3	Circle Line Intersection	5
3.4	Convex Hull	6
3.5	Count Lattices	6
3.6	Line Intersection	6
3.7	Line Sweep	6
3.8	Minkowski Sum	6
3.9	Nearest Points	7
3.10	Point In Convex	7
3.11	Segment Intersection	7

## 4 Graph Theory

4.1	Articulation Point	8
4.2	Bellman Ford	8
4.3	Bridge	8
4.4	Centroid Decomposition	8
4.5	Dijkstra	8
4.6	Dinics	9
4.7	Edmonds Karp	9
4.8	Fast Second Mst	9
4.9	Find Cycle	10
4.10	Floyd Warshall	10
4.11	Ford Fulkerson	10
4.12	Hierholzer	10
4.13	Hungarian	10
4.14	Is Bipartite	11
4.15	Is Cyclic	11
4.16	Kahn	11
4.17	Kosaraju	11
4.18	Kruskals	11
4.19	Kuhn	11
4.20	Lowest Common Ancestor	12
4.21	Maximum Bipartite Matching	12
4.22	Min Cost Flow	12
4.23	Prim	12
4.24	Topological Sort	13
4.25	Zero One Bfs	13

## 5 Math

5.1	Chinese Remainder Theorem	13
5.2	Extended Euclidean	13
5.3	Factorial Modulo	13
5.4	Fast Fourier Transform	13
5.5	Fibonacci	13
5.6	Find All Solutions	14
5.7	Linear Sieve	14
5.8	Matrix	14
5.9	Miller Rabin	14
5.10	Modulo Inverse	15
5.11	Pollard Rho Brent	15
5.12	Range Sieve	15
5.13	Segmented Sieve	15
5.14	Sum Of Divisors	15
5.15	Tonelli Shanks	15

## 6 Miscellaneous

6.1	Gauss	16
6.2	Ternary Search	16

## 7 References

7.1	Ref	16
-----	-----	----

## 8 Strings

8.1	Count Unique Substrings	16
8.2	Finding Repetitions	16
8.3	Group Identical Substrings	17
8.4	Hashing	17
8.5	Knuth Morris Pratt	17
8.6	Longest Common Prefix	17
8.7	Manacher	17
8.8	Rabin Karp	17
8.9	Suffix Array	17
8.10	Z Function	18

## 1 Data Structures

## 1.1 Binary Trie

```

1 struct Node { struct Node* parent, child[2]; };
2 struct BinaryTrie {
3     Node* root;
4     BinaryTrie() {
5         root = new Node();
6         root->parent = NULL;
7         root->child[0] = NULL;
8         root->child[1] = NULL;
9     }
10    void insert_node(int x) {
11        Node* cur = root;
12        for (int place = 29; place >= 0; place--) {
13            int bit = x >> place & 1;
14            if (cur->child[bit] != NULL) cur = cur->child[bit];
15            else {
16                cur->child[bit] = new Node();
17                cur->child[bit]->parent = cur;
18                cur = cur->child[bit];
19                cur->child[0] = NULL;
20                cur->child[1] = NULL;
21            }
22        }
23    }
24    void remove_node(int x) {
25        Node* cur = root;
26        for (int place = 29; place >= 0; place--) {

```

```

13    int bit = x >> place & 1;
14    if (cur->child[bit] == NULL) return;
15    cur = cur->child[bit];
16    while (cur->parent != NULL && cur->child[0] ==
17        NULL && cur->child[1] == NULL) {
18        Node* temp = cur;
19        cur = cur->parent;
20        if (temp == cur->child[0]) cur->child[0] =
21            NULL;
22        else cur->child[1] = NULL;
23        delete temp;
24    }
25    int get_min_xor(int x) {
26        Node* cur = root;
27        int minXor = 0;
28        for (int place = 29; place >= 0; place--) {
29            int bit = x >> place & 1;
30            if (cur->child[bit] != NULL) cur = cur->child[bit];
31            else {
32                minXor ^= 1 << place;
33                cur = cur->child[1 ^ bit];
34            }
35        }
36        return minXor;
37    }
38    };

```

## 1.2 Disjoint Set Union

```

1 struct DSU {
2     vector<int> parent, size;
3     DSU(int n) {
4         parent.resize(n);
5         size.resize(n);
6         for (int i = 0; i < n; i++) make_set(i);
7     }
8     void make_set(int v) {
9         parent[v] = v;
10        size[v] = 1;
11    }
12    bool is_same(int a, int b) { return find_set(a) == find_set(b); }
13    int find_set(int v) { return v == parent[v] ? v : parent[v] = find_set(parent[v]); }
14    void union_sets(int a, int b) {
15        a = find_set(a);
16        b = find_set(b);
17        if (a != b) {
18            if (size[a] < size[b]) swap(a, b);
19            parent[b] = a;
20            size[a] += size[b];
21        }
22    }
23 };

```

## 1.3 Minimum Queue

```

1 ll get_minimum(stack<pair<ll, ll>> &s1, stack<pair<ll, ll>> &s2) {
2     if (s1.empty() || s2.empty()) {
3         return s1.empty() ? s2.top().second : s1.top().second;

```

```

4     } else {
5         return min(s1.top().second, s2.top().second);
6     }
7 }
8 void add_element(ll new_element, stack<pair<ll, ll
9 >> &s1) {
10     ll minimum = s1.empty() ? new_element : min(
11         new_element, s1.top().second);
12     s1.push({new_element, minimum});
13 }
14 ll remove_element(stack<pair<ll, ll>> &s1, stack<
15 pair<ll, ll>> &s2) {
16     if (s2.empty()) {
17         while (!s1.empty()) {
18             ll element = s1.top().first;
19             s1.pop();
20             ll minimum = s2.empty() ? element : min(
21                 element, s2.top().second);
22             s2.push({element, minimum});
23         }
24         ll removed_element = s2.top().first;
25         s2.pop();
26         return removed_element;
27     }
28 }

```

#### 1.4 Mo

```

1 void remove(idx); // TODO: remove value at idx
2 // from data structure
3 void add(idx); // TODO: add value at idx from
4 // data structure
5 int get_answer(); // TODO: extract the current
6 // answer of the data structure
7 int block_size;
8 struct Query {
9     int l, r, idx;
10     bool operator<(Query other) const {
11         return make_pair(l / block_size, r) < make_pair(
12             other.l / block_size, other.r);
13     }
14 };
15 vector<int> mo_s_algorithm(vector<Query> queries) {
16     vector<int> answers(queries.size());
17     sort(queries.begin(), queries.end());
18     // TODO: initialize data structure
19     int cur_l = 0, cur_r = -1;
20     // invariant: data structure will always reflect
21     // the range [cur_l, cur_r]
22     for (Query q : queries) {
23         while (cur_l > q.l) {
24             cur_l--;
25             add(cur_l);
26         }
27         while (cur_r < q.r) {
28             cur_r++;
29             add(cur_r);
30         }
31         while (cur_l < q.l) {
32             remove(cur_l);
33             cur_l++;
34         }
35         while (cur_r > q.r) {
36             remove(cur_r);
37             cur_r--;
38         }
39         answers[q.idx] = get_answer();
40     }
41 }

```

```

35 }
36 return answers;
37 }

```

#### 1.5 Range Add Point Query

```

1 template<typename T, typename InType = T>
2 class SegTreeNode {
3 public:
4     const T IDN = 0, DEF = 0;
5     int i, j;
6     T val;
7     SegTreeNode<T, InType>* lc, * rc;
8     SegTreeNode(int i, int j) : i(i), j(j) {
9         if (j - i == 1) {
10             lc = rc = nullptr;
11             val = DEF;
12             return;
13         }
14         int k = (i + j) / 2;
15         lc = new SegTreeNode<T, InType>(i, k);
16         rc = new SegTreeNode<T, InType>(k, j);
17         val = 0;
18     }
19     SegTreeNode(const vector<InType>& a, int i, int j
20 ) : i(i), j(j) {
21         if (j - i == 1) {
22             lc = rc = nullptr;
23             val = (T) a[i];
24             return;
25         }
26         int k = (i + j) / 2;
27         lc = new SegTreeNode<T, InType>(a, i, k);
28         rc = new SegTreeNode<T, InType>(a, k, j);
29         val = 0;
30     }
31     void range_add(int l, int r, T x) {
32         if (r <= i || j <= l) return;
33         if (l <= i && j <= r) {
34             val += x;
35             return;
36         }
37         lc->range_add(l, r, x);
38         rc->range_add(l, r, x);
39     }
40     T point_query(int k) {
41         if (k < i || j <= k) return IDN;
42         if (j - i == 1) return val;
43         return val + lc->point_query(k) + rc->
44             point_query(k);
45     }
46 };
47 template<typename T, typename InType = T>
48 class SegTree {
49 public:
50     SegTreeNode<T, InType> root;
51     SegTree(int n) : root(0, n) {}
52     SegTree(const vector<InType>& a) : root(a, 0, a.
53         size()) {}
54     void range_add(int l, int r, T x) { root.
55         range_add(l, r, x); }
56     T point_query(int k) { return root.point_query(k)
57         ; }
58 };

```

#### 1.6 Range Add Range Query

```

1 template<typename T, typename InType = T>
2 class SegTreeNode {
3 public:
4     const T IDN = 0, DEF = 0;
5     int i, j;
6     T val, to_add = 0;
7     SegTreeNode<T, InType>* lc, * rc;
8     SegTreeNode(int i, int j) : i(i), j(j) {
9         if (j - i == 1) {
10             lc = rc = nullptr;
11             val = DEF;
12             return;
13         }
14         int k = (i + j) / 2;
15         lc = new SegTreeNode<T, InType>(i, k);
16         rc = new SegTreeNode<T, InType>(k, j);
17         val = operation(lc->val, rc->val);
18     }
19     SegTreeNode(const vector<InType>& a, int i, int j
20 ) : i(i), j(j) {
21         if (j - i == 1) {
22             lc = rc = nullptr;
23             val = (T) a[i];
24             return;
25         }
26         int k = (i + j) / 2;
27         lc = new SegTreeNode<T, InType>(a, i, k);
28         rc = new SegTreeNode<T, InType>(a, k, j);
29         val = operation(lc->val, rc->val);
30     }
31     void propagate() {
32         if (to_add == 0) return;
33         val += to_add;
34         if (j - i > 1) {
35             lc->to_add += to_add;
36             rc->to_add += to_add;
37         }
38         to_add = 0;
39     }
40     void range_add(int l, int r, T delta) {
41         propagate();
42         if (r <= i || j <= l) return;
43         if (l <= i && j <= r) {
44             to_add += delta;
45             propagate();
46         } else {
47             lc->range_add(l, r, delta);
48             rc->range_add(l, r, delta);
49             val = operation(lc->val, rc->val);
50         }
51     }
52     T range_query(int l, int r) {
53         propagate();
54         if (l <= i && j <= r) return val;
55         if (j <= l || r <= i) return IDN;
56         return operation(lc->range_query(l, r), rc->
57             range_query(l, r));
58     }
59     T operation(T x, T y) {}
60 };
61 template<typename T, typename InType = T>
62 class SegTree {
63 public:
64     SegTreeNode<T, InType> root;
65     SegTree(int n) : root(0, n) {}
66 }

```

```

64 SegTree(const vector<InType>& a) : root(a, 0, a.
    size()) {}
65 void range_add(int l, int r, T delta) { root.
    range_add(l, r, delta); }
66 T range_query(int l, int r) { return root.
    range_query(l, r); }
67 };

```

## 1.7 Segment Tree

```

1 template<typename T, typename InType = T>
2 class SegTreeNode {
3 public:
4     const T IDN = 0, DEF = 0;
5     int i, j;
6     T val;
7     SegTreeNode<T, InType>* lc, * rc;
8     SegTreeNode(int i, int j) : i(i), j(j) {
9         if (j - i == 1) {
10             lc = rc = nullptr;
11             val = DEF;
12             return;
13         }
14         int k = (i + j) / 2;
15         lc = new SegTreeNode<T, InType>(i, k);
16         rc = new SegTreeNode<T, InType>(k, j);
17         val = op(lc->val, rc->val);
18     }
19     SegTreeNode(const vector<InType>& a, int i, int j)
20         : i(i), j(j) {
21         if (j - i == 1) {
22             lc = rc = nullptr;
23             val = (T) a[i];
24             return;
25         }
26         int k = (i + j) / 2;
27         lc = new SegTreeNode<T, InType>(a, i, k);
28         rc = new SegTreeNode<T, InType>(a, k, j);
29         val = op(lc->val, rc->val);
30     }
31     void set(int k, T x) {
32         if (k < i || j <= k) return;
33         if (j - i == 1) {
34             val = x;
35             return;
36         }
37         lc->set(k, x);
38         rc->set(k, x);
39         val = op(lc->val, rc->val);
40     }
41     T range_query(int l, int r) {
42         if (l <= i && j <= r) return val;
43         if (j <= l || r <= i) return IDN;
44         return op(lc->range_query(l, r), rc->
            range_query(l, r));
45     }
46     T op(T x, T y) {}
47 };
48 template<typename T, typename InType = T>
49 class SegTree {
50 public:
51     SegTreeNode<T, InType> root;
52     SegTree(int n) : root(0, n) {}
53     SegTree(const vector<InType>& a) : root(a, 0, a.
        size()) {}
54     void set(int k, T x) { root.set(k, x); }

```

```

54 T range_query(int l, int r) { return root.
    range_query(l, r); }
55 };

```

## 1.8 Segment Tree 2d

```

1 template<typename T, typename InType = T>
2 class SegTree2dNode {
3 public:
4     int i, j, tree_size;
5     SegTree<T, InType>* seg_tree;
6     SegTree2dNode<T, InType>* lc, * rc;
7     SegTree2dNode() {}
8     SegTree2dNode(const vector<vector<InType>>& a,
9         int i, int j) : i(i), j(j) {
10         tree_size = a[0].size();
11         if (j - i == 1) {
12             lc = rc = nullptr;
13             seg_tree = new SegTree<T, InType>(a[i]);
14             return;
15         }
16         int k = (i + j) / 2;
17         lc = new SegTree2dNode<T, InType>(a, i, k);
18         rc = new SegTree2dNode<T, InType>(a, k, j);
19         seg_tree = new SegTree<T, InType>(vector<T>(
20             tree_size));
21         operation_2d(lc->seg_tree, rc->seg_tree);
22     }
23     ~SegTree2dNode() {
24         delete lc;
25         delete rc;
26     }
27     void set_2d(int kx, int ky, T x) {
28         if (kx < i || j <= kx) return;
29         if (j - i == 1) {
30             seg_tree->set(ky, x);
31             return;
32         }
33         lc->set_2d(kx, ky, x);
34         rc->set_2d(kx, ky, x);
35         operation_2d(lc->seg_tree, rc->seg_tree);
36     }
37     T range_query_2d(int lx, int rx, int ly, int ry)
38     {
39         if (lx <= i && j <= rx) return seg_tree->
            range_query(ly, ry);
40         if (j <= lx || rx <= i) return -INF;
41         return max(lc->range_query_2d(lx, rx, ly, ry),
            rc->range_query_2d(lx, rx, ly, ry));
42     }
43     void operation_2d(SegTree<T, InType>* x, SegTree<
44         T, InType>* y) {
45         for (int k = 0; k < tree_size; k++) {
46             seg_tree->set(k, max(x->range_query(k, k + 1)
47                 , y->range_query(k, k + 1)));
48         }
49     }
50 };
51 template<typename T, typename InType = T>
52 class SegTree2d {
53 public:
54     SegTree2dNode<T, InType> root;
55     SegTree2d() {}
56     SegTree2d(const vector<vector<InType>>& mat) :
57         root(mat, 0, mat.size()) {}
58     void set_2d(int kx, int ky, T x) { root.set_2d(kx
59         , ky, x); }

```

```

53 T range_query_2d(int lx, int rx, int ly, int ry)
    { return root.range_query_2d(lx, rx, ly, ry)
    ; }
54 };

```

## 1.9 Sparse Table

```

1 ll log2_floor(ll i) {
2     return i ? __builtin_clzll(1) - __builtin_clzll(i)
3         : -1;
4 }
5 vector<vector<ll>> build_sum(ll N, ll K, vector<ll> &
    array) {
6     vector<vector<ll>> st(K + 1, vector<ll>(N + 1));
7     for (ll i = 0; i < N; i++) st[0][i] = array[i];
8     for (ll i = 1; i <= K; i++)
9         for (ll j = 0; j + (1 << i) <= N; j++)
10             st[i][j] = st[i - 1][j] + st[i - 1][j + (1 <<
11                 (i - 1))];
12     return st;
13 }
14 ll sum_query(ll L, ll R, ll K, vector<vector<ll>> &
    st) {
15     ll sum = 0;
16     for (ll i = K; i >= 0; i--) {
17         if ((1 << i) <= R - L + 1) {
18             sum += st[i][L];
19             L += 1 << i;
20         }
21     }
22     return sum;
23 }
24 vector<vector<ll>> build_min(ll N, ll K, vector<ll> &
    array) {
25     vector<vector<ll>> st(K + 1, vector<ll>(N + 1));
26     for (ll i = 0; i < N; i++) st[0][i] = array[i];
27     for (ll i = 1; i <= K; i++)
28         for (ll j = 0; j + (1 << i) <= N; j++)
29             st[i][j] = min(st[i - 1][j], st[i - 1][j + (1
30                 << (i - 1))]);
31     return st;
32 }
33 ll min_query(ll L, ll R, vector<vector<ll>> &st) {
34     ll i = log2_floor(R - L + 1);
35     return min(st[i][L], st[i][R - (1 << i) + 1]);
36 }

```

## 1.10 Sparse Table 2d

```

1 const int N = 100;
2 int matrix[N][N];
3 int table[N][N][(int)(log2(N) + 1)][(int)(log2(N) +
4     1)];
5 void build_sparse_table(int n, int m) {
6     for (int i = 0; i < n; i++)
7         for (int j = 0; j < m; j++)
8             table[i][j][0][0] = matrix[i][j];
9     for (int k = 1; k <= (int)(log2(n)); k++)
10         for (int i = 0; i + (1 << k) - 1 < n; i++)
11             for (int j = 0; j + (1 << k) - 1 < m; j++)
12                 table[i][j][k][0] = min(table[i][j][k -
13                     1][0], table[i + (1 << (k - 1))][j][k -
14                         1][0]);
15     for (int k = 1; k <= (int)(log2(m)); k++)
16         for (int i = 0; i < n; i++)

```

```

14     for (int j = 0; j + (1 << k) - 1 < m; j++)
15         table[i][j][0][k] = min(table[i][j][0][k -
16                                     1], table[i][j + (1 << (k - 1))][0][k
17                                     - 1]);
18     for (int k = 1; k <= (int)(log2(n)); k++)
19         for (int l = 1; l <= (int)(log2(m)); l++)
20             for (int i = 0; i + (1 << k) - 1 < n; i++)
21                 for (int j = 0; j + (1 << l) - 1 < m; j++)
22                     table[i][j][k][l] = min(
23                         min(table[i][j][k - 1][l - 1], table[i
24                             + (1 << (k - 1))][j][k - 1][l -
25                             1]),
26                         min(table[i][j + (1 << (l - 1))][k -
27                             1][l - 1], table[i + (1 << (k - 1)
28                             )][j + (1 << (l - 1))][k - 1][l -
29                             1]));
30     };
31 }

```

### 1.11 Sqrt Decomposition

```

1  int n;
2  vector<int> a (n);
3  int len = (int) sqrt (n + .0) + 1; // size of the
4  // block and the number of blocks
5  vector<int> b (len);
6  for (int i = 0; i < n; ++i) b[i / len] += a[i];
7  for (;;) {
8      int l, r;
9      // read input data for the next query
10     int sum = 0;
11     for (int i = l; i <= r; )
12         if (i % len == 0 && i + len - 1 <= r) {
13             // if the whole block starting at i belongs
14             // to [l, r]
15             sum += b[i / len];
16             i += len;
17         } else {
18             sum += a[i];
19             ++i;
20         }
21     // or
22     /*
23     int sum = 0;
24     int c_l = l / len, c_r = r / len;
25     if (c_l == c_r)
26         for (int i = l; i <= r; ++i)
27             sum += a[i];
28     else {
29         for (int i = l, end = (c_l + 1) * len - 1; i <= end; ++i)
30             sum += a[i];
31         for (int i = c_l + 1; i <= c_r - 1; ++i)
32             sum += b[i];
33         for (int i = c_r * len; i <= r; ++i)
34             sum += a[i];
35     }
36     */

```

```

35 }

```

## 2 Dynamic Programming

### 2.1 Divide And Conquer

```

1  ll m, n;
2  vector<ll> dp_before(n), dp_cur(n);
3  ll C(ll i, ll j);
4  void compute(ll l, ll r, ll optl, ll optpr) {
5      if (l > r) return;
6      ll mid = (l + r) >> 1;
7      pair<ll, ll> best = {LLONG_MAX, -1};
8      for (ll k = optl; k <= min(mid, optpr); k++)
9          best = min(best, {(k ? dp_before[k - 1] : 0) +
10                           C(k, mid), k});
11      dp_cur[mid] = best.first;
12      ll opt = best.second;
13      compute(l, mid - 1, optl, opt);
14      compute(mid + 1, r, opt, optpr);
15  }
16  ll solve() {
17      for (ll i = 0; i < n; ++i) dp_before[i] = C(0, i);
18      ;
19      for (ll i = 1; i < m; ++i) {
20          compute(0, n - 1, 0, n - 1);
21          dp_before = dp_cur;
22      }
23      return dp_before[n - 1];
24  }

```

### 2.2 Edit Distance

```

1  ll edit_distance(string x, string y, ll n, ll m) {
2      vector<vector<int>> dp(n + 1, vector<int>(m + 1,
3          INF));
4      dp[0][0] = 0;
5      for (int i = 1; i <= n; ++i) {
6          dp[i][0] = i;
7      }
8      for (int j = 1; j <= m; ++j) {
9          dp[0][j] = j;
10     }
11     for (int i = 1; i <= n; ++i) {
12         for (int j = 1; j <= m; ++j) {
13             dp[i][j] = min({dp[i - 1][j] + 1, dp[i][j -
14                 1] + 1, dp[i - 1][j - 1] + (x[i - 1] !=
15                 y[j - 1])});
16         }
17     }
18     return dp[n][m];
19 }

```

### 2.3 Knapsack

```

1  ll knapsack(ll W, vector<ll> &wt, vector<ll> &val,
2      ll n) {
3      vector<ll> dp(W + 1, 0);
4      for (ll i = 1; i <= n; ++i) {
5          for (ll w = W; w >= 0; w--) {
6              if (wt[i - 1] <= w) {

```

```

6              dp[w] = max(dp[w], dp[w - wt[i - 1]] + val[
7                  i - 1]);
8              }
9          }
10     return dp[W];
11 }

```

### 2.4 Knuth Optimization

```

1  ll solve() {
2      ll N;
3      ... // Read input
4      vector<vector<ll>> dp(N, vector<ll>(N)), opt(N,
5          vector<ll>(N));
6      auto C = [&](ll i, ll j) {
7          ... // Implement cost function C.
8      };
9      for (ll i = 0; i < N; ++i) {
10         opt[i][i] = i;
11         ... // Initialize dp[i][i] according to the
12             // problem
13     }
14     for (ll i = N - 2; i >= 0; i--) {
15         for (ll j = i + 1; j < N; ++j) {
16             ll mn = LL_MAX, cost = C(i, j);
17             for (ll k = opt[i][j - 1]; k <= min(j - 1,
18                 opt[i + 1][j]); k++) {
19                 if (mn >= dp[i][k] + dp[k + 1][j] + cost) {
20                     opt[i][j] = k;
21                     mn = dp[i][k] + dp[k + 1][j] + cost;
22                 }
23             }
24             dp[i][j] = mn;
25         }
26     }
27     cout << dp[0][N - 1] << '\n';
28 }

```

### 2.5 Longest Common Subsequence

```

1  ll LCS(string x, string y, ll n, ll m) {
2      vector<vector<ll>> dp(n + 1, vector<ll>(m + 1));
3      for (ll i = 0; i <= n; ++i) {
4          for (ll j = 0; j <= m; ++j) {
5              if (i == 0 || j == 0) {
6                  dp[i][j] = 0;
7              } else if (x[i - 1] == y[j - 1]) {
8                  dp[i][j] = dp[i - 1][j - 1] + 1;
9              } else {
10                 dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
11             }
12         }
13     }
14     ll index = dp[n][m];
15     vector<char> lcs(index + 1);
16     lcs[index] = '\0';
17     ll i = n, j = m;
18     while (i > 0 && j > 0) {
19         if (x[i - 1] == y[j - 1]) {
20             lcs[index - 1] = x[i - 1];
21             i--;
22             j--;
23             index--;
24         } else if (dp[i - 1][j] > dp[i][j - 1]) {

```

```

25     i--;
26 } else {
27     j--;
28 }
29 }
30 return dp[n][m];
31 }

```

## 2.6 Longest Increasing Subsequence

```

1 ll get_ceil_idx(vector<ll> &a, vector<ll> &T, ll l,
2     ll r, ll x) {
3     while (r - l > 1) {
4         ll m = l + (r - l) / 2;
5         if (a[T[m]] >= x) {
6             r = m;
7         } else {
8             l = m;
9         }
10    }
11    return r;
12 }
13 ll LIS(ll n, vector<ll> &a) {
14     ll len = 1;
15     vector<ll> T(n, 0), R(n, -1);
16     T[0] = 0;
17     for (ll i = 1; i < n; i++) {
18         if (a[i] < a[T[0]]) {
19             T[0] = i;
20         } else if (a[i] > a[T[len - 1]]) {
21             R[i] = T[len - 1];
22             T[len++] = i;
23         } else {
24             ll pos = get_ceil_idx(a, T, -1, len - 1, a[i]);
25             R[i] = T[pos - 1];
26             T[pos] = i;
27         }
28     }
29     return len;
}

```

## 2.7 Max Sum

```

1 int max_subarray_sum(vi arr) {
2     int x = 0, s = 0;
3     for (int k = 0; k < n; k++) {
4         s = max(arr[k], s+arr[k]);
5         x = max(x, s);
6     }
7     return x;
8 }

```

## 2.8 Subset Sum

```

1 bool subset_sum(ll n, vector<ll> &arr, ll sum) {
2     vector<vector<ll>> dp(n + 1, vector<ll>(sum + 1,
3         false));
4     dp[0][0] = true;
5     for (ll i = 1; i <= n; i++) {
6         for (ll j = 0; j <= sum; j++) {
7             dp[i][j] = dp[i - 1][j];

```

```

7         if (j >= arr[i]) {
8             dp[i][j] |= dp[i - 1][j - arr[i]];
9         }
10    }
11 }
12 return dp[n][sum];
13 }

```

## 3 Geometry

### 3.1 Areas

```

1 int signed_area_parallelogram(point2d p1, point2d
2     p2, point2d p3) {
3     return cross(p2 - p1, p3 - p1);
4 }
5 double triangle_area(point2d p1, point2d p2,
6     point2d p3) {
7     return abs(signed_area_parallelogram(p1, p2, p3))
8         / 2.0;
9 }
10 bool clockwise(point2d p1, point2d p2, point2d p3) {
11     return signed_area_parallelogram(p1, p2, p3) < 0;
12 }
13 bool counter_clockwise(point2d p1, point2d p2,
14     point2d p3) {
15     return signed_area_parallelogram(p1, p2, p3) > 0;
16 }
17 double area(const vector<point>& fig) {
18     double res = 0;
19     for (unsigned i = 0; i < fig.size(); i++) {
20         point p = i ? fig[i - 1] : fig.back();
21         point q = fig[i];
22         res += (p.x - q.x) * (p.y + q.y);
23     }
24     return fabs(res) / 2;
25 }

```

### 3.2 Basic Geometry

```

1 struct point2d {
2     ftype x, y;
3     point2d() {}
4     point2d(ftype x, ftype y): x(x), y(y) {}
5     point2d& operator+=(const point2d &t) {
6         x += t.x;
7         y += t.y;
8         return *this;
9     }
10    point2d& operator-=(const point2d &t) {
11        x -= t.x;
12        y -= t.y;
13        return *this;
14    }
15    point2d& operator*=(ftype t) {
16        x *= t;
17        y *= t;
18        return *this;
19    }
20    point2d& operator/=(ftype t) {
21        x /= t;
22        y /= t;
23        return *this;

```

```

24 }
25 point2d operator+(const point2d &t) const {
26     return point2d(*this) += t;
27 }
28 point2d operator-(const point2d &t) const {
29     return point2d(*this) -= t;
30 }
31 point2d operator*(ftype t) const { return point2d
32     (*this) *= t; }
33 point2d operator/(ftype t) const { return point2d
34     (*this) /= t; }
35 };
36 point2d operator*(ftype a, point2d b) { return b *
37     a; }
38 ftype dot(point2d a, point2d b) { return a.x * b.x
39     + a.y * b.y; }
40 ftype dot(point3d a, point3d b) { return a.x * b.x
41     + a.y * b.y + a.z * b.z; }
42 ftype norm(point2d a) { return sqrt(norm(a)); }
43 double abs(point2d a) { return sqrt(norm(a)); }
44 double proj(point2d a, point2d b) { return dot(a, b)
45     / abs(b); }
46 double angle(point2d a, point2d b) { return acos(
47     dot(a, b) / abs(a) / abs(b)); }
48 point3d cross(point3d a, point3d b) { return
49     point3d(a.y * b.z - a.z * b.y, a.z * b.x - a.x *
50     b.z, a.x * b.y - a.y * b.x); }
51 ftype triple(point3d a, point3d b, point3d c) {
52     return dot(a, cross(b, c)); }
53 ftype cross(point2d a, point2d b) { return a.x * b.y
54     - a.y * b.x; }
55 point2d intersect(point2d a1, point2d d1, point2d
56     a2, point2d d2) { return a1 + cross(a2 - a1,
57     d2) / cross(d1, d2) * d1; }
58 point3d intersect(point3d a1, point3d n1, point3d
59     a2, point3d n2, point3d a3, point3d n3) {
60     point3d x(n1.x, n2.x, n3.x);
61     point3d y(n1.y, n2.y, n3.y);
62     point3d z(n1.z, n2.z, n3.z);
63     point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
64     return point3d(triple(d, y, z), triple(x, d, z),
65         triple(x, y, d)) / triple(n1, n2, n3);
66 }
67 }

```

### 3.3 Circle Line Intersection

```

1 double r, a, b, c; // given as input
2 double x0 = -a * c / (a * a + b * b);
3 double y0 = -b * c / (a * a + b * b);
4 if (c * c > r * r * (a * a + b * b) + EPS) {
5     puts ("no points");
6 } else if (abs (c * c - r * r * (a * a + b * b)) <
7     EPS) {
8     puts ("1 point");
9     cout << x0 << ' ' << y0 << '\n';
10 } else {
11     double d = r * r - c * c / (a * a + b * b);
12     double mult = sqrt (d / (a * a + b * b));
13     double ax = x0 + b * mult;
14     double bx = x0 - b * mult;
15     double ay = y0 + a * mult;
16     double by = y0 - a * mult;
17     puts ("2 points");
18     cout << ax << ' ' << ay << '\n' << bx << ' ' <<
19         by << '\n';
20 }

```

### 3.4 Convex Hull

```
1 struct pt {
2     double x, y;
3 };
4 ll orientation(pt a, pt b, pt c) {
5     double v = a.x * (b.y - c.y) + b.x * (c.y - a.y)
6         + c.x * (a.y - b.y);
7     if (v < 0) {
8         return -1;
9     } else if (v > 0) {
10        return +1;
11    }
12    return 0;
13 }
14 bool cw(pt a, pt b, pt c, bool include_collinear) {
15     ll o = orientation(a, b, c);
16     return o < 0 || (include_collinear && o == 0);
17 }
18 bool collinear(pt a, pt b, pt c) {
19     return orientation(a, b, c) == 0;
20 }
21 void convex_hull(vector<pt>& a, bool
22     include_collinear = false) {
23     pt p0 = *min_element(a.begin(), a.end(), [](pt a,
24         pt b) {
25             return make_pair(a.y, a.x) < make_pair(b.y, b.x);
26         });
27     sort(a.begin(), a.end(), [&p0](const pt& a, const
28         pt& b) {
29         ll o = orientation(p0, a, b);
30         if (o == 0) {
31             return (p0.x - a.x) * (p0.x - a.x) + (p0.y -
32                 a.y) * (p0.y - a.y)
33                 < (p0.x - b.x) * (p0.x - b.x) + (p0.y -
34                     b.y) * (p0.y - b.y);
35         }
36         return o < 0;
37     });
38     if (include_collinear) {
39         ll i = (ll) a.size() - 1;
40         while (i >= 0 && collinear(p0, a[i], a.back())) {
41             i--;
42         }
43         reverse(a.begin() + i + 1, a.end());
44     }
45     vector<pt> st;
46     for (ll i = 0; i < (ll) a.size(); i++) {
47         while (st.size() > 1 && !cw(st[st.size() - 2],
48             st.back(), a[i], include_collinear)) {
49             st.pop_back();
50         }
51         st.push_back(a[i]);
52     }
53     a = st;
54 }
```

### 3.5 Count Lattices

```
1 int count_lattices(Fraction k, Fraction b, long
2     long n) {
3     auto fk = k.floor();
4     auto fb = b.floor();
5     auto cnt = 0LL;
6     if (k >= 1 || b >= 1) {
```

```
6         cnt += (fk * (n - 1) + 2 * fb) * n / 2;
7         k -= fk;
8         b -= fb;
9     }
10    auto t = k * n + b;
11    auto ft = t.floor();
12    if (ft >= 1) cnt += count_lattices(1 / k, (t - t.
13        floor()) / k, t.floor());
14    return cnt;
15 }
```

### 3.6 Line Intersection

```
1 struct pt { double x, y; };
2 struct line { double a, b, c; };
3 const double EPS = 1e-9;
4 double det(double a, double b, double c, double d) {
5     return a*d - b*c;
6 }
7 bool intersect(line m, line n, pt & res) {
8     double zn = det(m.a, m.b, n.a, n.b);
9     if (abs(zn) < EPS) return false;
10    res.x = -det(m.c, m.b, n.c, n.b) / zn;
11    res.y = -det(m.a, m.c, n.a, n.c) / zn;
12    return true;
13 }
14 bool parallel(line m, line n) { return abs(det(m.a,
15     m.b, n.a, n.b)) < EPS; }
16 bool equivalent(line m, line n) {
17     return abs(det(m.a, m.b, n.a, n.b)) < EPS
18         && abs(det(m.a, m.c, n.a, n.c)) < EPS
19         && abs(det(m.b, m.c, n.b, n.c)) < EPS;
20 }
```

### 3.7 Line Sweep

```
1 const double EPS = 1E-9;
2 struct pt { double x, y; };
3 struct seg {
4     pt p, q;
5     ll id;
6     double get_y(double x) const {
7         if (abs(p.x - q.x) < EPS) return p.y;
8         return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.
9             x);
10    }
11 }
12 bool intersectld(double l1, double r1, double l2,
13     double r2) {
14     if (l1 > r1) swap(l1, r1);
15     if (l2 > r2) swap(l2, r2);
16     return max(l1, l2) <= min(r1, r2) + EPS;
17 }
18 ll vec(const pt& a, const pt& b, const pt& c) {
19     double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y)
20         * (c.x - a.x);
21     return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
22 }
23 bool intersect(const seg& a, const seg& b) {
24     return intersectld(a.p.x, a.q.x, b.p.x, b.q.x) &&
25         intersectld(a.p.y, a.q.y, b.p.y, b.q.y) &&
26         vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <=
27             0 &&
28         vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <=
29             0;
30 }
```

```
26 bool operator<(const seg& a, const seg& b) {
27     double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.
28         x));
29     return a.get_y(x) < b.get_y(x) - EPS;
30 }
31 struct event {
32     double x;
33     ll tp, id;
34     event() {}
35     event(double x, ll tp, ll id) : x(x), tp(tp), id(
36         id) {}
37     bool operator<(const event& e) const {
38         if (abs(x - e.x) > EPS) return x < e.x;
39         return tp > e.tp;
40     }
41 };
42 set<seg> s;
43 vector<set<seg>::iterator> where;
44 set<seg>::iterator prev(set<seg>::iterator it) {
45     return it == s.begin() ? s.end() : --it;
46 }
47 set<seg>::iterator next(set<seg>::iterator it) {
48     return ++it;
49 }
50 pair<ll, ll> solve(const vector<seg>& a) {
51     ll n = (ll) a.size();
52     vector<event> e;
53     for (ll i = 0; i < n; ++i) {
54         e.push_back(event(min(a[i].p.x, a[i].q.x), +1,
55             i));
56         e.push_back(event(max(a[i].p.x, a[i].q.x), -1,
57             i));
58     }
59     sort(e.begin(), e.end());
60     s.clear();
61     where.resize(a.size());
62     for (size_t i = 0; i < e.size(); ++i) {
63         ll id = e[i].id;
64         if (e[i].tp == +1) {
65             set<seg>::iterator nxt = s.lower_bound(a[id])
66                 , prv = prev(nxt);
67             if (nxt != s.end() && intersect(*nxt, a[id]))
68                 return make_pair(nxt->id, id);
69             if (prv != s.end() && intersect(*prv, a[id]))
70                 return make_pair(prv->id, id);
71             where[id] = s.insert(nxt, a[id]);
72         } else {
73             set<seg>::iterator nxt = next(where[id]), prv
74                 = prev(where[id]);
75             if (nxt != s.end() && prv != s.end() &&
76                 intersect(*nxt, *prv)) return make_pair(
77                 prv->id, nxt->id);
78             s.erase(where[id]);
79         }
80     }
81     return make_pair(-1, -1);
82 }
```

### 3.8 Minkowski Sum

```
1 struct pt {
2     ll x, y;
3     pt operator + (const pt & p) const { return pt{x
4         + p.x, y + p.y}; }
5     pt operator - (const pt & p) const { return pt{x
6         - p.x, y - p.y}; }
```



```

5   ll cross(const pt & p) const { return x * p.y - y
    * p.x; }
6   };
7   void reorder_polygon(vector<pt> & P){
8       size_t pos = 0;
9       for (size_t i = 1; i < P.size(); i++){
10          if (P[i].y < P[pos].y || (P[i].y == P[pos].y &&
11              P[i].x < P[pos].x)) pos = i;
12      }
13      rotate(P.begin(), P.begin() + pos, P.end());
14  }
15  vector<pt> minkowski(vector<pt> P, vector<pt> Q){
16      // the first vertex must be the lowest
17      reorder_polygon(P);
18      reorder_polygon(Q);
19      // we must ensure cyclic indexing
20      P.push_back(P[0]);
21      P.push_back(P[1]);
22      Q.push_back(Q[0]);
23      Q.push_back(Q[1]);
24      // main part
25      vector<pt> result;
26      size_t i = 0, j = 0;
27      while (i < P.size() - 2 || j < Q.size() - 2){
28          result.push_back(P[i] + Q[j]);
29          auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] -
30              Q[j]);
31          if (cross >= 0 && i < P.size() - 2) ++i;
32          if (cross <= 0 && j < Q.size() - 2) ++j;
33      }
34      return result;
35  }

```

### 3.9 Nearest Points

```

1  struct pt {
2      ll x, y, id;
3  };
4  struct cmp_x {
5      bool operator()(const pt & a, const pt & b) const
6      {
7          return a.x < b.x || (a.x == b.x && a.y < b.y);
8      }
9  };
10 struct cmp_y {
11     bool operator()(const pt & a, const pt & b) const
12     { return a.y < b.y; }
13 };
14 ll n;
15 vector<pt> a;
16 double mindist;
17 pair<ll, ll> best_pair;
18 void upd_ans(const pt & a, const pt & b) {
19     double dist = sqrt((a.x - b.x) * (a.x - b.x) + (a
20         .y - b.y) * (a.y - b.y));
21     if (dist < mindist) {
22         mindist = dist;
23         best_pair = {a.id, b.id};
24     }
25 }
26 vector<pt> t;
27 void rec(ll l, ll r) {
28     if (r - l <= 3) {
29         for (ll i = 1; i < r; ++i)
30             for (ll j = i + 1; j < r; ++j)
31                 upd_ans(a[i], a[j]);
32         sort(a.begin() + 1, a.begin() + r, cmp_y());
33     }
34 }

```

```

30     return;
31 }
32 ll m = (l + r) >> 1, midx = a[m].x;
33 rec(l, m);
34 rec(m, r);
35 merge(a.begin() + 1, a.begin() + m, a.begin() + m
36     , a.begin() + r, t.begin(), cmp_y());
37 copy(t.begin(), t.begin() + r - 1, a.begin() + 1)
38 ;
39 ll tsz = 0;
40 for (ll i = 1; i < r; ++i) {
41     if (abs(a[i].x - midx) < mindist) {
42         for (ll j = tsz - 1; j >= 0 && a[i].y - t[j].
43             y < mindist; --j)
44             upd_ans(a[i], t[j]);
45         t[tsz++] = a[i];
46     }
47 }
48 t.resize(n);
49 sort(a.begin(), a.end(), cmp_x());
50 mindist = 1E20;
51 rec(0, n);

```

### 3.10 Point In Convex

```

1  struct pt {
2      long long x, y;
3  };
4  pt(long long _x, long long _y) : x(_x), y(_y) {}
5  pt operator+(const pt &p) const { return pt(x + p
6      .x, y + p.y); }
7  pt operator-(const pt &p) const { return pt(x - p
8      .x, y - p.y); }
9  long long cross(const pt &p) const { return x * p
10     .y - y * p.x; }
11 long long dot(const pt &p) const { return x * p.x
12     + y * p.y; }
13 long long cross(const pt &a, const pt &b) const {
14     return (a - *this).cross(b - *this); }
15 long long dot(const pt &a, const pt &b) const {
16     return (a - *this).dot(b - *this); }
17 long long sqrLen() const { return this->dot(*this
18     ); }
19 };
20 bool lexComp(const pt &l, const pt &r) { return l.x
21     < r.x || (l.x == r.x && l.y < r.y); }
22 int sgn(long long val) { return val > 0 ? 1 : (val
23     == 0 ? 0 : -1); }
24 vector<pt> seq;
25 pt translation;
26 int n;
27 bool pointInTriangle(pt a, pt b, pt c, pt point) {
28     long long s1 = abs(a.cross(b, c));
29     long long s2 = abs(point.cross(a, b)) + abs(point
30         .cross(b, c)) + abs(point.cross(c, a));
31     return s1 == s2;
32 }
33 void prepare(vector<pt> &points) {
34     n = points.size();
35     int pos = 0;
36     for (int i = 1; i < n; i++) {
37         if (lexComp(points[i], points[pos])) pos = i;
38     }
39     rotate(points.begin(), points.begin() + pos,
40         points.end());
41     n--;
42 }

```

```

31 seq.resize(n);
32 for (int i = 0; i < n; i++) seq[i] = points[i +
33     1] - points[0];
34 translation = points[0];
35 bool pointInConvexPolygon(pt point) {
36     point = point - translation;
37     if (seq[0].cross(point) != 0 && sgn(seq[0].cross(
38         point)) != sgn(seq[0].cross(seq[n - 1])))
39         return false;
40     if (seq[n - 1].cross(point) != 0 && sgn(seq[n -
41         1].cross(point)) != sgn(seq[n - 1].cross(seq
42             [0])))
43         return false;
44     if (seq[0].cross(point) == 0)
45         return seq[0].sqrLen() >= point.sqrLen();
46     int l = 0, r = n - 1;
47     while (r - l > 1) {
48         int mid = (l + r) / 2;
49         int pos = mid;
50         if (seq[pos].cross(point) >= 0) l = mid;
51         else r = mid;
52     }
53     int pos = l;
54     return pointInTriangle(seq[pos], seq[pos + 1], pt
55         (0, 0), point);
56 }

```

### 3.11 Segment Intersection

```

1  const double EPS = 1E-9;
2  struct pt {
3      double x, y;
4      bool operator<(const pt& p) const {
5          return x < p.x - EPS || (abs(x - p.x) < EPS &&
6              y < p.y - EPS);
7      }
8  };
9  struct line {
10     double a, b, c;
11     line() {}
12     line(pt p, pt q) {
13         a = p.y - q.y;
14         b = q.x - p.x;
15         c = -a * p.x - b * p.y;
16         norm();
17     }
18     void norm() {
19         double z = sqrt(a * a + b * b);
20         if (abs(z) > EPS) a /= z, b /= z, c /= z;
21     }
22     double dist(pt p) const { return a * p.x + b * p.
23         y + c; }
24 };
25 double det(double a, double b, double c, double d)
26 {
27     return a * d - b * c;
28 }
29 inline bool betw(double l, double r, double x) {
30     return min(l, r) <= x + EPS && x <= max(l, r) +
31         EPS;
32 }
33 inline bool intersect_ld(double a, double b, double
34     c, double d) {
35     if (a > b) swap(a, b);
36     if (c > d) swap(c, d);
37     return max(a, c) <= min(b, d) + EPS;
38 }

```

```

33 }
34 bool intersect(pt a, pt b, pt c, pt d, pt& left, pt
    & right) {
35     if (!intersect_ld(a.x, b.x, c.x, d.x) || !
        intersect_ld(a.y, b.y, c.y, d.y)) return
        false;
36     line m(a, b);
37     line n(c, d);
38     double zn = det(m.a, m.b, n.a, n.b);
39     if (abs(zn) < EPS) {
40         if (abs(m.dist(c)) > EPS || abs(n.dist(a)) >
            EPS) return false;
41         if (b < a) swap(a, b);
42         if (d < c) swap(c, d);
43         left = max(a, c);
44         right = min(b, d);
45         return true;
46     } else {
47         left.x = right.x = -det(m.c, m.b, n.c, n.b) /
            zn;
48         left.y = right.y = -det(m.a, m.c, n.a, n.c) /
            zn;
49         return betw(a.x, b.x, left.x) && betw(a.y, b.y,
            left.y) &&
50             betw(c.x, d.x, left.x) && betw(c.y, d.y,
            left.y);
51     }
52 }

```

## 4 Graph Theory

### 4.1 Articulation Point

```

1 void APUtil(vector<vector<ll>> &adj, ll u, vector<
    bool> &visited,
2 vector<ll> &disc, vector<ll> &low, ll &time, ll
    parent, vector<bool> &isAP) {
3     ll children = 0;
4     visited[u] = true;
5     disc[u] = low[u] = ++time;
6     for (auto v : adj[u]) {
7         if (!visited[v]) {
8             children++;
9             APUtil(adj, v, visited, disc, low, time, u,
                isAP);
10            low[u] = min(low[u], low[v]);
11            if (parent != -1 && low[v] >= disc[u]) {
12                isAP[u] = true;
13            }
14        } else if (v != parent) {
15            low[u] = min(low[u], disc[v]);
16        }
17    }
18    if (parent == -1 && children > 1) {
19        isAP[u] = true;
20    }
21 }
22 void AP(vector<vector<ll>> &adj, ll n) {
23     vector<ll> disc(n), low(n);
24     vector<bool> visited(n), isAP(n);
25     ll time = 0, par = -1;
26     for (ll u = 0; u < n; u++) {
27         if (!visited[u]) {
28             APUtil(adj, u, visited, disc, low, time, par,
                isAP);
29         }

```

```

30     }
31     for (ll u = 0; u < n; u++) {
32         if (isAP[u]) {
33             cout << u << " ";
34         }
35     }
36 }

```

---

### 4.2 Bellman Ford

```

1 struct Edge {
2     int a, b, cost;
3 };
4 int n, m, v;
5 vector<Edge> edges;
6 const int INF = 1000000000;
7 void solve() {
8     vector<int> d(n, INF);
9     d[v] = 0;
10    vector<int> p(n, -1);
11    int x;
12    for (int i = 0; i < n; ++i) {
13        x = -1;
14        for (Edge e : edges)
15            if (d[e.a] < INF)
16                if (d[e.b] > d[e.a] + e.cost) {
17                    d[e.b] = max(-INF, d[e.a] + e.cost);
18                    p[e.b] = e.a;
19                    x = e.b;
20                }
21    }
22    if (x == -1) cout << "No negative cycle from " <<
        v;
23    else {
24        int y = x;
25        for (int i = 0; i < n; ++i) y = p[y];
26        vector<int> path;
27        for (int cur = y; cur = p[cur]) {
28            path.push_back(cur);
29            if (cur == y && path.size() > 1) break;
30        }
31        reverse(path.begin(), path.end());
32        cout << "Negative cycle: ";
33        for (int u : path) cout << u << ' ';
34    }
35 }

```

### 4.3 Bridge

```

1 int n;
2 vector<vector<int>> adj;
3 vector<bool> visited;
4 vector<int> tin, low;
5 int timer;
6 void dfs(int v, int p = -1) {
7     visited[v] = true;
8     tin[v] = low[v] = timer++;
9     for (int to : adj[v]) {
10        if (to == p) continue;
11        if (visited[to]) {
12            low[v] = min(low[v], tin[to]);
13        } else {
14            dfs(to, v);
15            low[v] = min(low[v], low[to]);
16            if (low[to] > tin[v]) IS_BRIDGE(v, to);

```

```

17     }
18 }
19 }
20 void find_bridges() {
21     timer = 0;
22     visited.assign(n, false);
23     tin.assign(n, -1);
24     low.assign(n, -1);
25     for (int i = 0; i < n; ++i) {
26         if (!visited[i]) dfs(i);
27     }
28 }

```

### 4.4 Centroid Decomposition

```

1 vector<vector<int>> adj;
2 vector<bool> is_removed;
3 vector<int> subtree_size;
4 int get_subtree_size(int node, int parent = -1) {
5     subtree_size[node] = 1;
6     for (int child : adj[node]) {
7         if (child == parent || is_removed[
            child]) continue;
8         subtree_size[node] +=
            get_subtree_size(child, node);
9     }
10    return subtree_size[node];
11 }
12 int get_centroid(int node, int tree_size, int
    parent = -1) {
13     for (int child : adj[node]) {
14         if (child == parent || is_removed[
            child]) continue;
15         if (subtree_size[child] * 2 >
            tree_size) return get_centroid(
            child, tree_size, node);
16     }
17     return node;
18 }
19 void build_centroid_decomp(int node = 0) {
20     int centroid = get_centroid(node,
        get_subtree_size(node));
21     // do something
22     is_removed[centroid] = true;
23     for (int child : adj[centroid]) {
24         if (is_removed[child]) continue;
25         build_centroid_decomp(child);
26     }
27 }

```

### 4.5 Dijkstra

```

1 const int INF = 1000000000;
2 vector<vector<pair<int, int>>> adj;
3 void dijkstra(int s, vector<int> & d, vector<int> &
    p) {
4     int n = adj.size();
5     d.assign(n, INF);
6     p.assign(n, -1);
7     d[s] = 0;
8     using pii = pair<int, int>;
9     priority_queue<pii, vector<pii>, greater<pii>> q;
10    q.push({0, s});
11    while (!q.empty()) {
12        int v = q.top().second, d_v = q.top().first;

```



```

13     q.pop();
14     if (d_v != d[v]) continue;
15     for (auto edge : adj[v]) {
16         int to = edge.first, len = edge.second;
17         if (d[v] + len < d[to]) {
18             d[to] = d[v] + len;
19             p[to] = v;
20             q.push({d[to], to});
21         }
22     }
23 }
24 }

```

#### 4.6 Dinics

```

1 struct FlowEdge {
2     int v, u;
3     ll cap, flow = 0;
4     FlowEdge(int v, int u, ll cap) : v(v), u(u), cap(cap) {}
5 };
6 struct Dinic {
7     const ll flow_inf = 1e18;
8     vector<FlowEdge> edges;
9     vector<vector<int>> adj;
10    int n, m = 0, s, t;
11    vector<int> level, ptr;
12    queue<int> q;
13    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
14        adj.resize(n);
15        level.resize(n);
16        ptr.resize(n);
17    }
18    void add_edge(int v, int u, ll cap) {
19        edges.emplace_back(v, u, cap);
20        edges.emplace_back(u, v, 0);
21        adj[v].push_back(m);
22        adj[u].push_back(m + 1);
23        m += 2;
24    }
25    bool bfs() {
26        while (!q.empty()) {
27            int v = q.front();
28            q.pop();
29            for (int id : adj[v]) {
30                if (edges[id].cap - edges[id].flow < 1)
31                    continue;
32                if (level[edges[id].u] != -1) continue;
33                level[edges[id].u] = level[v] + 1;
34                q.push(edges[id].u);
35            }
36            return level[t] != -1;
37        }
38    }
39    ll dfs(int v, ll pushed) {
40        if (pushed == 0) return 0;
41        if (v == t) return pushed;
42        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
43            int id = adj[v][cid], u = edges[id].u;
44            if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1) continue;
45            ll tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
46            if (tr == 0) continue;
47            edges[id].flow += tr;
48            edges[id ^ 1].flow -= tr;

```

```

48        return tr;
49    }
50    return 0;
51 }
52 ll flow() {
53     ll f = 0;
54     while (true) {
55         fill(level.begin(), level.end(), -1);
56         level[s] = 0;
57         q.push(s);
58         if (!bfs()) break;
59         fill(ptr.begin(), ptr.end(), 0);
60         while (ll pushed = dfs(s, flow_inf)) f += pushed;
61     }
62     return f;
63 }
64 };

```

#### 4.7 Edmonds Karp

```

1 int n;
2 vector<vector<int>> capacity;
3 vector<vector<int>> adj;
4 int bfs(int s, int t, vector<int>& parent) {
5     fill(parent.begin(), parent.end(), -1);
6     parent[s] = -2;
7     queue<pair<int, int>> q;
8     q.push({s, INF});
9     while (!q.empty()) {
10        int cur = q.front().first, flow = q.front().second;
11        q.pop();
12        for (int next : adj[cur]) {
13            if (parent[next] == -1 && capacity[cur][next] > 0) {
14                parent[next] = cur;
15                int new_flow = min(flow, capacity[cur][next]);
16                if (next == t) return new_flow;
17                q.push({next, new_flow});
18            }
19        }
20    }
21    return 0;
22 }
23 int maxflow(int s, int t) {
24     int flow = 0;
25     vector<int> parent(n);
26     int new_flow;
27     while (new_flow = bfs(s, t, parent)) {
28         flow += new_flow;
29         int cur = t;
30         while (cur != s) {
31             int prev = parent[cur];
32             capacity[prev][cur] -= new_flow;
33             capacity[cur][prev] += new_flow;
34             cur = prev;
35         }
36     }
37     return flow;
38 }

```

#### 4.8 Fast Second Mst

```

1 struct edge {
2     int s, e, w, id;
3     bool operator<(const struct edge& other) {
4         return w < other.w; }
5 };
6 typedef struct edge Edge;
7 const int N = 2e5 + 5;
8 long long res = 0, ans = 1e18;
9 int n, m, a, b, w, id, l = 21;
10 vector<Edge> edges;
11 vector<int> h(N, 0), parent(N, -1), size(N, 0), present(N, 0);
12 vector<vector<pair<int, int>>> adj(N), dp(N, vector<pair<int, int>>(l));
13 vector<vector<int>> up(N, vector<int>(l, -1));
14 pair<int, int> combine(pair<int, int> a, pair<int, int> b) {
15     vector<int> v = {a.first, a.second, b.first, b.second};
16     int topTwo = -3, topOne = -2;
17     for (int c : v) {
18         if (c > topOne) {
19             topTwo = topOne;
20             topOne = c;
21         } else if (c > topTwo && c < topOne) topTwo = c;
22     }
23     return {topOne, topTwo};
24 }
25 void dfs(int u, int par, int d) {
26     h[u] = 1 + h[par];
27     up[u][0] = par;
28     dp[u][0] = {d, -1};
29     for (auto v : adj[u]) {
30         if (v.first != par) dfs(v.first, u, v.second);
31     }
32 }
33 pair<int, int> lca(int u, int v) {
34     pair<int, int> ans = {-2, -3};
35     if (h[u] < h[v]) swap(u, v);
36     for (int i = l - 1; i >= 0; i--) {
37         if (h[u] - h[v] >= (1 << i)) {
38             ans = combine(ans, dp[u][i]);
39             u = up[u][i];
40         }
41     }
42     if (u == v) return ans;
43     for (int i = l - 1; i >= 0; i--) {
44         if (up[u][i] != -1 && up[v][i] != -1 && up[u][i] != up[v][i]) {
45             ans = combine(ans, combine(dp[u][i], dp[v][i]));
46             u = up[u][i];
47             v = up[v][i];
48         }
49     }
50     ans = combine(ans, combine(dp[u][0], dp[v][0]));
51     return ans;
52 }
53 int main(void) {
54     cin >> n >> m;
55     for (int i = 1; i <= n; i++) {
56         parent[i] = i;
57         size[i] = 1;
58     }
59     for (int i = 1; i <= m; i++) {
60         cin >> a >> b >> w; // 1-indexed
61         edges.push_back({a, b, w, i - 1});

```

```

62 }
63 sort(edges.begin(), edges.end());
64 for (int i = 0; i <= m - 1; i++) {
65     a = edges[i].s;
66     b = edges[i].e;
67     w = edges[i].w;
68     id = edges[i].id;
69     if (unite_set(a, b)) {
70         adj[a].emplace_back(b, w);
71         adj[b].emplace_back(a, w);
72         present[id] = 1;
73         res += w;
74     }
75 }
76 dfs(1, 0, 0);
77 for (int i = 1; i <= m - 1; i++) {
78     for (int j = 1; j <= n; ++j) {
79         if (up[j][i - 1] != -1) {
80             int v = up[j][i - 1];
81             up[j][i] = up[v][i - 1];
82             dp[j][i] = combine(dp[j][i - 1], dp[v][i - 1]);
83         }
84     }
85 }
86 for (int i = 0; i <= m - 1; i++) {
87     id = edges[i].id;
88     w = edges[i].w;
89     if (!present[id]) {
90         auto rem = lca(edges[i].s, edges[i].e);
91         if (rem.first != w) {
92             if (ans > res + w - rem.first) ans = res + w - rem.first;
93         } else if (rem.second != -1) {
94             if (ans > res + w - rem.second) ans = res + w - rem.second;
95         }
96     }
97 }
98 cout << ans << "\n";
99 return 0;
100 }

```

#### 4.9 Find Cycle

```

1 bool dfs(ll v) {
2     color[v] = 1;
3     for (ll u : adj[v]) {
4         if (color[u] == 0) {
5             parent[u] = v;
6             if (dfs(u)) {
7                 return true;
8             }
9         } else if (color[u] == 1) {
10            cycle_end = v;
11            cycle_start = u;
12            return true;
13        }
14    }
15    color[v] = 2;
16    return false;
17 }
18 void find_cycle() {
19     color.assign(n, 0);
20     parent.assign(n, -1);
21     cycle_start = -1;
22     for (ll v = 0; v < n; v++) {

```

```

23         if (color[v] == 0 && dfs(v)) {
24             break;
25         }
26     }
27     if (cycle_start == -1) {
28         cout << "Acyclic" << endl;
29     } else {
30         vector<ll> cycle;
31         cycle.push_back(cycle_start);
32         for (ll v = cycle_end; v != cycle_start; v = parent[v]) {
33             cycle.push_back(v);
34         }
35         cycle.push_back(cycle_start);
36         reverse(cycle.begin(), cycle.end());
37         cout << "Cycle found: ";
38         for (ll v : cycle) {
39             cout << v << ' ';
40         }
41         cout << '\n';
42     }
43 }

```

#### 4.10 Floyd Warshall

```

1 void floyd_warshall(vector<vector<ll>> &dis, ll n)
2 {
3     for (ll k = 0; k < n; k++)
4         for (ll i = 0; i < n; i++)
5             for (ll j = 0; j < n; j++)
6                 if (dis[i][k] < INF && dis[k][j] < INF)
7                     dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
8     for (ll i = 0; i < n; i++)
9         for (ll j = 0; j < n; j++)
10            if (dis[k][k] < 0 && dis[i][k] < INF && dis[k][j] < INF)
11                dis[i][j] = -INF;
12 }

```

#### 4.11 Ford Fulkerson

```

1 bool bfs(ll n, vector<vector<ll>> &r_graph, ll s, ll t, vector<ll> &parent) {
2     vector<bool> visited(n, false);
3     queue<ll> q;
4     q.push(s);
5     visited[s] = true;
6     parent[s] = -1;
7     while (!q.empty()) {
8         ll u = q.front();
9         q.pop();
10        for (ll v = 0; v < n; v++) {
11            if (!visited[v] && r_graph[u][v] > 0) {
12                if (v == t) {
13                    parent[v] = u;
14                    return true;
15                }
16                q.push(v);
17                parent[v] = u;
18                visited[v] = true;
19            }
20        }
21    }

```

```

22     return false;
23 }
24 ll ford_fulkerson(ll n, vector<vector<ll>> graph, ll s, ll t) {
25     ll u, v;
26     vector<vector<ll>> r_graph;
27     for (u = 0; u < n; u++)
28         for (v = 0; v < n; v++)
29             r_graph[u][v] = graph[u][v];
30     vector<ll> parent;
31     ll max_flow = 0;
32     while (bfs(n, r_graph, s, t, parent)) {
33         ll path_flow = INF;
34         for (v = t; v != s; v = parent[v]) {
35             u = parent[v];
36             path_flow = min(path_flow, r_graph[u][v]);
37         }
38         for (v = t; v != s; v = parent[v]) {
39             u = parent[v];
40             r_graph[u][v] -= path_flow;
41             r_graph[v][u] += path_flow;
42         }
43         max_flow += path_flow;
44     }
45     return max_flow;
46 }

```

#### 4.12 Hierholzer

```

1 void print_circuit(vector<vector<ll>> &adj) {
2     map<ll, ll> edge_count;
3     for (ll i = 0; i < adj.size(); i++) {
4         edge_count[i] = adj[i].size();
5     }
6     if (!adj.size()) {
7         return;
8     }
9     stack<ll> curr_path;
10    vector<ll> circuit;
11    curr_path.push(0);
12    ll curr_v = 0;
13    while (!curr_path.empty()) {
14        if (edge_count[curr_v]) {
15            curr_path.push(curr_v);
16            ll next_v = adj[curr_v].back();
17            edge_count[curr_v]--;
18            adj[curr_v].pop_back();
19            curr_v = next_v;
20        } else {
21            circuit.push_back(curr_v);
22            curr_v = curr_path.top();
23            curr_path.pop();
24        }
25    }
26    for (ll i = circuit.size() - 1; i >= 0; i--) {
27        cout << circuit[i] << ' ';
28    }
29 }

```

#### 4.13 Hungarian

```

1 vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
2 for (int i=1; i<=n; ++i) {
3     p[0] = i;
4     int j0 = 0;

```

```

5  vector<int> minv (m+1, INF);
6  vector<bool> used (m+1, false);
7  do {
8      used[j0] = true;
9      int i0 = p[j0], delta = INF, j1;
10     for (int j=1; j<=m; ++j)
11         if (!used[j]) {
12             int cur = A[i0][j]-u[i0]-v[j];
13             if (cur < minv[j]) minv[j] = cur, way[j] =
14                 j0;
15             if (minv[j] < delta) delta = minv[j], j1 =
16                 j;
17         }
18     for (int j=0; j<=m; ++j)
19         if (used[j]) u[p[j]] += delta, v[j] -= delta
20         ;
21     else minv[j] -= delta;
22     j0 = j1;
23     while (p[j0] != 0);
24     do {
25         int j1 = way[j0];
26         p[j0] = p[j1];
27         j0 = j1;
28     } while (j0);
29     vector<int> ans (n+1);
30     for (int j=1; j<=m; ++j)
31         ans[p[j]] = j;
32     int cost = -v[0];

```

#### 4.14 Is Bipartite

```

1  bool is_bipartite(vector<ll> &col, vector<vector<ll>
2  >> &adj, ll n) {
3      queue<pair<ll, ll>> q;
4      for (ll i = 0; i < n; i++) {
5          if (col[i] == -1) {
6              q.push({i, 0});
7              col[i] = 0;
8              while (!q.empty()) {
9                  pair<ll, ll> p = q.front();
10                 q.pop();
11                 ll v = p.first, c = p.second;
12                 for (ll j : adj[v]) {
13                     if (col[j] == c) {
14                         return false;
15                     }
16                     if (col[j] == -1) {
17                         col[j] = (c ? 0 : 1);
18                         q.push({j, col[j]});
19                     }
20                 }
21             }
22         }
23     }
24     return true;

```

#### 4.15 Is Cyclic

```

1  bool is_cyclic_util(int u, vector<vector<int>> &adj
2  , vector<bool> &vis, vector<bool> &rec) {
3      vis[u] = true;
4      rec[u] = true;
5      for (auto v : adj[u]) {

```

```

5          if (!vis[v] && is_cyclic_util(v, adj, vis, rec)
6              ) return true;
7          else if (rec[v]) return true;
8      }
9      rec[u] = false;
10     return false;
11 }
12 bool is_cyclic(int n, vector<vector<int>> &adj) {
13     vector<bool> vis(n, false), rec(n, false);
14     for (int i = 0; i < n; i++)
15         if (!vis[i] && is_cyclic_util(i, adj, vis, rec)
16             ) return true;
17     return false;
18 }

```

#### 4.16 Kahn

```

1  void kahn(vector<vector<ll>> &adj) {
2      ll n = adj.size();
3      vector<ll> in_degree(n, 0);
4      for (ll u = 0; u < n; u++)
5          for (ll v : adj[u]) in_degree[v]++;
6      queue<ll> q;
7      for (ll i = 0; i < n; i++)
8          if (in_degree[i] == 0)
9              q.push(i);
10     ll cnt = 0;
11     vector<ll> top_order;
12     while (!q.empty()) {
13         ll u = q.front();
14         q.pop();
15         top_order.push_back(u);
16         for (ll v : adj[u])
17             if (--in_degree[v] == 0) q.push(v);
18         cnt++;
19     }
20     if (cnt != n) {
21         cout << -1 << '\n';
22         return;
23     }
24     // print top_order
25 }

```

#### 4.17 Kosaraju

```

1  void topo_sort(int u, vector<vector<int>> &adj,
2  vector<bool> &vis, stack<int> &stk) {
3      vis[u] = true;
4      for (int v : adj[u]) {
5          if (!vis[v]) {
6              topo_sort(v, adj, vis, stk);
7          }
8      }
9      stk.push(u);
10 }
11 vector<vector<int>> transpose(int n, vector<vector<
12 int>> &adj) {
13     vector<vector<int>> adj_t(n);
14     for (int u = 0; u < n; u++) {
15         for (int v : adj[u]) {
16             adj_t[v].push_back(u);
17         }
18     }
19     return adj_t;

```

```

19 }
20
21 void get_scc(int u, vector<vector<int>> &adj_t,
22 vector<bool> &vis, vector<int> &scc) {
23     vis[u] = true;
24     scc.push_back(u);
25     for (int v : adj_t[u]) {
26         if (!vis[v]) {
27             get_scc(v, adj_t, vis, scc);
28         }
29     }
30 }
31 void kosaraju(int n, vector<vector<int>> &adj,
32 vector<vector<int>> &sccs) {
33     vector<bool> vis(n, false);
34     stack<int> stk;
35     for (int u = 0; u < n; u++) {
36         if (!vis[u]) {
37             topo_sort(u, adj, vis, stk);
38         }
39     }
40     vector<vector<int>> adj_t = transpose(n, adj);
41     for (int u = 0; u < n; u++) {
42         vis[u] = false;
43     }
44     while (!stk.empty()) {
45         int u = stk.top();
46         stk.pop();
47         if (!vis[u]) {
48             vector<int> scc;
49             get_scc(u, adj_t, vis, scc);
50             sccs.push_back(scc);
51         }
52     }

```

#### 4.18 Kruskals

```

1  struct Edge {
2      int u, v, weight;
3      bool operator<(Edge const& other) {
4          return weight < other.weight;
5      }
6  };
7  int n;
8  vector<Edge> edges;
9  int cost = 0;
10 vector<Edge> result;
11 DSU dsu = DSU(n);
12 sort(edges.begin(), edges.end());
13 for (Edge e : edges) {
14     if (dsu.find_set(e.u) != dsu.find_set(e.v)) {
15         cost += e.weight;
16         result.push_back(e);
17         dsu.union_sets(e.u, e.v);
18     }
19 }

```

#### 4.19 Kuhn

```

1  int n, k;
2  vector<vector<int>> g;
3  vector<int> mt;
4  vector<bool> used;

```

```

5 bool try_kuhn(int v) {
6     if (used[v]) return false;
7     used[v] = true;
8     for (int to : g[v]) {
9         if (mt[to] == -1 || try_kuhn(mt[to])) {
10             mt[to] = v;
11             return true;
12         }
13     }
14     return false;
15 }
16 int main() {
17     mt.assign(k, -1);
18     vector<bool> used1(n, false);
19     for (int v = 0; v < n; ++v) {
20         for (int to : g[v]) {
21             if (mt[to] == -1) {
22                 mt[to] = v;
23                 used1[v] = true;
24                 break;
25             }
26         }
27     }
28     for (int v = 0; v < n; ++v) {
29         if (used1[v]) continue;
30         used.assign(n, false);
31         try_kuhn(v);
32     }
33     for (int i = 0; i < k; ++i)
34         if (mt[i] != -1)
35             printf("%d %d\n", mt[i] + 1, i + 1);
36 }

```

#### 4.20 Lowest Common Ancestor

```

1 struct LCA {
2     vector<ll> height, euler, first, segtree;
3     vector<bool> visited;
4     ll n;
5     LCA(vector<vector<ll>> &adj, ll root = 0) {
6         n = adj.size();
7         height.resize(n);
8         first.resize(n);
9         euler.reserve(n * 2);
10        visited.assign(n, false);
11        dfs(adj, root);
12        ll m = euler.size();
13        segtree.resize(m * 4);
14        build(1, 0, m - 1);
15    }
16    void dfs(vector<vector<ll>> &adj, ll node, ll h = 0) {
17        visited[node] = true;
18        height[node] = h;
19        first[node] = euler.size();
20        euler.push_back(node);
21        for (auto to : adj[node]) {
22            if (!visited[to]) {
23                dfs(adj, to, h + 1);
24                euler.push_back(node);
25            }
26        }
27    }
28    void build(ll node, ll b, ll e) {
29        if (b == e) segtree[node] = euler[b];
30        else {
31            ll mid = (b + e) / 2;

```

```

32        build(node << 1, b, mid);
33        build(node << 1 | 1, mid + 1, e);
34        ll l = segtree[node << 1], r = segtree[node
35            << 1 | 1];
36        segtree[node] = (height[l] < height[r]) ? l :
37            r;
38    }
39    ll query(ll node, ll b, ll e, ll L, ll R) {
40        if (b > R || e < L) return -1;
41        if (b >= L && e <= R) return segtree[node];
42        ll mid = (b + e) >> 1;
43        ll left = query(node << 1, b, mid, L, R);
44        ll right = query(node << 1 | 1, mid + 1, e, L,
45            R);
46        if (left == -1) return right;
47        if (right == -1) return left;
48        return height[left] < height[right] ? left :
49            right;
50    }
51    ll lca(ll u, ll v) {
52        ll left = first[u], right = first[v];
53        if (left > right) swap(left, right);
54        return query(1, 0, euler.size() - 1, left,
55            right);
56    }
57 }

```

#### 4.21 Maximum Bipartite Matching

```

1 bool bpm(ll n, ll m, vector<vector<bool>> &bpGraph,
2     ll u, vector<bool> &seen, vector<ll> &matchR) {
3     for (ll v = 0; v < m; v++) {
4         if (bpGraph[u][v] && !seen[v]) {
5             seen[v] = true;
6             if (matchR[v] < 0 || bpm(n, m, bpGraph,
7                 matchR[v], seen, matchR)) {
8                 matchR[v] = u;
9                 return true;
10            }
11        }
12    }
13    return false;
14 }
15 ll maxBPM(ll n, ll m, vector<vector<bool>> &bpGraph) {
16     vector<ll> matchR(m, -1);
17     ll result = 0;
18     for (ll u = 0; u < n; u++) {
19         vector<bool> seen(m, false);
20         if (bpm(n, m, bpGraph, u, seen, matchR)) {
21             result++;
22         }
23     }
24     return result;
25 }

```

#### 4.22 Min Cost Flow

```

1 struct Edge {
2     int from, to, capacity, cost;
3 };
4 vector<vector<int>> adj, cost, capacity;
5 const int INF = 1e9;

```

```

6 void shortest_paths(int n, int v0, vector<int> &d,
7     vector<int> &p) {
8     d.assign(n, INF);
9     d[v0] = 0;
10    vector<bool> inq(n, false);
11    queue<int> q;
12    q.push(v0);
13    p.assign(n, -1);
14    while (!q.empty()) {
15        int u = q.front();
16        q.pop();
17        inq[u] = false;
18        for (int v : adj[u]) {
19            if (capacity[u][v] > 0 && d[v] > d[u] + cost[
20                u][v]) {
21                d[v] = d[u] + cost[u][v];
22                p[v] = u;
23                if (!inq[v]) {
24                    inq[v] = true;
25                    q.push(v);
26                }
27            }
28        }
29    }
30    int min_cost_flow(int N, vector<Edge> edges, int K,
31        int s, int t) {
32        adj.assign(N, vector<int>());
33        cost.assign(N, vector<int>(N, 0));
34        capacity.assign(N, vector<int>(N, 0));
35        for (Edge e : edges) {
36            adj[e.from].push_back(e.to);
37            adj[e.to].push_back(e.from);
38            cost[e.from][e.to] = e.cost;
39            cost[e.to][e.from] = -e.cost;
40            capacity[e.from][e.to] = e.capacity;
41        }
42        int flow = 0;
43        int cost = 0;
44        vector<int> d, p;
45        while (flow < K) {
46            shortest_paths(N, s, d, p);
47            if (d[t] == INF) break;
48            int f = K - flow, cur = t;
49            while (cur != s) {
50                f = min(f, capacity[p[cur]][cur]);
51                cur = p[cur];
52            }
53            flow += f;
54            cost += f * d[t];
55            cur = t;
56            while (cur != s) {
57                capacity[p[cur]][cur] -= f;
58                capacity[cur][p[cur]] += f;
59                cur = p[cur];
60            }
61            if (flow < K) return -1;
62            else return cost;
63        }
64    }

```

#### 4.23 Prim

```

1 const int INF = 1000000000;
2 struct Edge {
3     int w = INF, to = -1;
4     bool operator<(Edge const& other) const {

```

```

5     return make_pair(w, to) < make_pair(other.w,
6         other.to);
7 }
8 int n;
9 vector<vector<Edge>> adj;
10 void prim() {
11     int total_weight = 0;
12     vector<Edge> min_e(n);
13     min_e[0].w = 0;
14     set<Edge> q;
15     q.insert({0, 0});
16     vector<bool> selected(n, false);
17     for (int i = 0; i < n; ++i) {
18         if (q.empty()) {
19             cout << "No MST!" << endl;
20             exit(0);
21         }
22         int v = q.begin()->to;
23         selected[v] = true;
24         total_weight += q.begin()->w;
25         q.erase(q.begin());
26         if (min_e[v].to != -1) cout << v << " " <<
27             min_e[v].to << endl;
28         for (Edge e : adj[v]) {
29             if (!selected[e.to] && e.w < min_e[e.to].w) {
30                 q.erase({min_e[e.to].w, e.to});
31                 min_e[e.to] = {e.w, v};
32                 q.insert({e.w, e.to});
33             }
34         }
35         cout << total_weight << endl;
36     }
}

```

#### 4.24 Topological Sort

```

1 void dfs(ll v) {
2     visited[v] = true;
3     for (ll u : adj[v]) {
4         if (!visited[u]) {
5             dfs(u);
6         }
7     }
8     ans.push_back(v);
9 }
10 void topological_sort() {
11     visited.assign(n, false);
12     ans.clear();
13     for (ll i = 0; i < n; ++i) {
14         if (!visited[i]) {
15             dfs(i);
16         }
17     }
18     reverse(ans.begin(), ans.end());
19 }

```

#### 4.25 Zero One Bfs

```

1 vector<int> d(n, INF);
2 d[s] = 0;
3 deque<int> q;
4 q.push_front(s);
5 while (!q.empty()) {
6     int v = q.front();

```

```

7     q.pop_front();
8     for (auto edge : adj[v]) {
9         int u = edge.first, w = edge.second;
10        if (d[v] + w < d[u]) {
11            d[u] = d[v] + w;
12            if (w == 1) q.push_back(u);
13            else q.push_front(u);
14        }
15    }
16 }

```

## 5 Math

### 5.1 Chinese Remainder Theorem

```

1 struct Congruence {
2     ll a, m;
3 };
4
5 ll chinese_remainder_theorem(vector<Congruence>
6     const& congruences) {
7     ll M = 1;
8     for (auto const& congruence : congruences) M *=
9         congruence.m;
10    ll solution = 0;
11    for (auto const& congruence : congruences) {
12        ll a_i = congruence.a;
13        ll M_i = M / congruence.m;
14        ll N_i = mod_inv(M_i, congruence.m);
15        solution = (solution + a_i * M_i % M * N_i) % M;
16    }
17    return solution;
18 }

```

### 5.2 Extended Euclidean

```

1 int gcd(int a, int b, int& x, int& y) {
2     if (b == 0) {
3         x = 1;
4         y = 0;
5         return a;
6     }
7     int x1, y1, d = gcd(b, a % b, x1, y1);
8     x = y1;
9     y = x1 - y1 * (a / b);
10    return d;
11 }

```

### 5.3 Factorial Modulo

```

1 int factmod(int n, int p) {
2     vector<int> f(p);
3     f[0] = 1;
4     for (int i = 1; i < p; ++i) f[i] = f[i - 1] * i %
5         p;
6     int res = 1;
7     while (n > 1) {
8         if ((n / p) % 2) res = p - res;
9         res = res * f[n % p] % p;
10        n /= p;
11    }

```

```

11     return res;
12 }

```

### 5.4 Fast Fourier Transform

```

1 using cd = complex<double>;
2 const double PI = acos(-1);
3 void fft(vector<cd>& a, bool invert) {
4     int n = a.size();
5     if (n == 1) return;
6     vector<cd> a0(n / 2), a1(n / 2);
7     for (int i = 0; 2 * i < n; i++) {
8         a0[i] = a[2 * i];
9         a1[i] = a[2 * i + 1];
10    }
11    fft(a0, invert);
12    fft(a1, invert);
13    double ang = 2 * PI / n * (invert ? -1 : 1);
14    cd w(1), wn(cos(ang), sin(ang));
15    for (int i = 0; 2 * i < n; i++) {
16        a[i] = a0[i] + w * a1[i];
17        a[i + n / 2] = a0[i] - w * a1[i];
18        if (invert) {
19            a[i] /= 2;
20            a[i + n / 2] /= 2;
21        }
22        w *= wn;
23    }
24 }
25 vector<int> multiply(vector<int> const& a, vector<
26     int> const& b) {
27     vector<cd> fa(a.begin(), a.end()), fb(b.begin()
28         , b.end());
29     int n = 1;
30     while (n < a.size() + b.size()) n <= 1;
31     fa.resize(n);
32     fb.resize(n);
33     fft(fa, false);
34     fft(fb, false);
35     for (int i = 0; i < n; i++) fa[i] *= fb[i];
36     fft(fa, true);
37     vector<int> result(n);
38     for (int i = 0; i < n; i++) result[i] = round(
39         fa[i].real());
40     return result;
41 }

```

### 5.5 Fibonacci

```

1 /*
2 Properties:
3 - Cassini's identity:  $f[n-1]f[n+1] - f[n]^2 = (-1)^n$ 
4 - d'Ocagne's identity:  $f[m]f[n+1] - f[m+1]f[n] = (-1)^n f[m-n]$ 
5 - Addition rule:  $f[n+k] = f[k]f[n+1] + f[k-1]f[n]$ 
6 -  $k = n$  case:  $f[2n] = f[n](f[n+1] + f[n-1])$ 
7 -  $f[n] \mid f[nk]$ 
8 -  $f[n] \mid f[m] \Rightarrow n \mid m$ 
9 - GCD rule:  $\text{gcd}(f[m], f[n]) = f[\text{gcd}(m, n)]$ 
10 -  $[1 \ 1], [1 \ 0]]^n = [[f[n+1] \ f[n]], [f[n] \ f[n-1]]]$ 
11 -  $f[2k+1] = f[k+1]^2 + f[k]^2$ 
12 -  $f[2k] = f[k](f[k+1] + f[k-1]) = f[k](2f[k+1] - f[k])$ 

```

```

13 - Periodic sequence modulo p
14 - sum[i=1..n]f[i] = f[n+2] - 1
15 - sum[i=0..n-1]f[2i+1] = f[2n]
16 - sum[i=1..n]f[2i] = f[2n+1] - 1
17 - sum[i=1..n]f[i]^2 = f[n]f[n+1]
18 Fibonacci encoding:
19 1. Iterate through the Fibonacci numbers from the
   largest to the
20 smallest until you find one less than or equal to n
   .
21 2. Suppose this number was F_i. Subtract F_i from
   n & and put a 1 &
22 in the i-2 position of the code word (indexing from
   0 from the
23 leftmost to the rightmost bit).
24 3. Repeat until there is no remainder.
25 4. Add a final 1 & to the codeword to indicate its
   end.
26 Closed-form: f[n] = ((1 + rt(5))/2)^n - ((1 - rt
   (5)) / 2)^n / rt(5)
27 */
28 struct matrix {
29     ll mat[2][2];
30     matrix friend operator *(const matrix &a, const
   matrix &b) {
31         matrix c;
32         for (int i = 0; i < 2; i++) {
33             for (int j = 0; j < 2; j++) {
34                 c.mat[i][j] = 0;
35                 for (int k = 0; k < 2; k++) c.mat[i][j] +=
   a.mat[i][k] * b.mat[k][j];
36             }
37         }
38         return c;
39     }
40 };
41 matrix matpow(matrix base, ll n) {
42     matrix ans{ {
43         {1, 0},
44         {0, 1}
45     } };
46     while (n) {
47         if (n & 1) ans = ans * base;
48         base = base * base;
49         n >>= 1;
50     }
51     return ans;
52 }
53 ll fib(int n) {
54     matrix base{ {
55         {1, 1},
56         {1, 0}
57     } };
58     return matpow(base, n).mat[0][1];
59 }
60 pair<int, int> fib (int n) {
61     if (n == 0) return {0, 1};
62     auto p = fib(n >> 1);
63     int c = p.first * (2 * p.second - p.first);
64     int d = p.first * p.first + p.second * p.second;
65     if (n & 1) return {d, c + d};
66     else return {c, d};
67 }

```

## 5.6 Find All Solutions

```

1 bool find_any_solution(ll a, ll b, ll c, ll &x0, ll

```

```

        &y0, ll &g) {
2     g = gcd_extended(abs(a), abs(b), x0, y0);
3     if (c % g) return false;
4     x0 *= c / g;
5     y0 *= c / g;
6     if (a < 0) x0 = -x0;
7     if (b < 0) y0 = -y0;
8     return true;
9 }
10 void shift_solution(ll &x, ll &y, ll a, ll b, ll
   cnt) {
11     x += cnt * b;
12     y -= cnt * a;
13 }
14 ll find_all_solutions(ll a, ll b, ll c, ll minx, ll
   maxx, ll miny, ll maxy) {
15     ll x, y, g;
16     if (!find_any_solution(a, b, c, x, y, g)) return
   0;
17     a /= g;
18     b /= g;
19     ll sign_a = a > 0 ? +1 : -1;
20     ll sign_b = b > 0 ? +1 : -1;
21     shift_solution(x, y, a, b, (minx - x) / b);
22     if (x < minx) shift_solution(x, y, a, b, sign_b);
23     if (x > maxx) return 0;
24     ll lx1 = x;
25     shift_solution(x, y, a, b, (maxx - x) / b);
26     if (x > maxx) shift_solution(x, y, a, b, -sign_b)
   ;
27     ll rx1 = x;
28     shift_solution(x, y, a, b, -(miny - y) / a);
29     if (y < miny) shift_solution(x, y, a, b, -sign_a)
   ;
30     if (y > maxy) return 0;
31     ll lx2 = x;
32     shift_solution(x, y, a, b, -(maxy - y) / a);
33     if (y > maxy) shift_solution(x, y, a, b, sign_a);
34     ll rx2 = x;
35     if (lx2 > rx2) swap(lx2, rx2);
36     ll lx = max(lx1, lx2), rx = min(rx1, rx2);
37     if (lx > rx) return 0;
38     return (rx - lx) / abs(b) + 1;
39 }

```

## 5.7 Linear Sieve

```

1 void linear_sieve(ll N, vector<ll> &lowest_prime,
   vector<ll> &prime) {
2     for (ll i = 2; i <= N; i++) {
3         if (lowest_prime[i] == 0) {
4             lowest_prime[i] = i;
5             prime.push_back(i);
6         }
7         for (ll j = 0; i * prime[j] <= N; j++) {
8             lowest_prime[i * prime[j]] = prime[j];
9             if (prime[j] == lowest_prime[i]) break;
10        }
11    }
12 }

```

## 5.8 Matrix

```

1 /*
2 Matrix exponentiation:

```

```

3 f[n] = af[n-1] + bf[n-2] + cf[n-3]
4 Use:
5 |f[n] | |a b c| |f[n-1]|
6 |f[n-1]| |1 0 0| |f[n-2]|
7 |f[n-2]| |0 1 0| |f[n-3]|
8 To get:
9 |f[n] | |a b c|^(n-2) |f[2]|
10 |f[n-1]| |1 0 0| |f[1]|
11 |f[n-2]| |0 1 0| |f[0]|
12 */
13 struct Matrix { int mat[MAX_N][MAX_N]; };
14 Matrix matrix_mul(Matrix a, Matrix b) {
15     Matrix ans; int i, j, k;
16     for (i = 0; i < MAX_N; i++)
17         for (j = 0; j < MAX_N; j++)
18             for (ans.mat[i][j] = k = 0; k < MAX_N; k++)
19                 ans.mat[i][j] += a.mat[i][k] * b.mat[k][j];
20     return ans;
21 }
22 Matrix matrix_pow(Matrix base, int p) {
23     Matrix ans; int i, j;
24     for (i = 0; i < MAX_N; i++)
25         for (j = 0; j < MAX_N; j++)
26             ans.mat[i][j] = (i == j);
27     while (p) {
28         if (p & 1) ans = matrix_mul(ans, base);
29         base = matrix_mul(base, base);
30         p >>= 1;
31     }
32     return ans;
33 }

```

## 5.9 Miller Rabin

```

1 using u64 = uint64_t;
2 using u128 = __uint128_t;
3 u64 binpower(u64 base, u64 e, u64 mod) {
4     u64 result = 1;
5     base %= mod;
6     while (e) {
7         if (e & 1) result = (u128) result * base % mod;
8         base = (u128) base * base % mod;
9         e >>= 1;
10    }
11    return result;
12 }
13 bool check_composite(u64 n, u64 a, u64 d, ll s) {
14     u64 x = binpower(a, d, n);
15     if (x == 1 || x == n - 1) return false;
16     for (ll r = 1; r < s; r++) {
17         x = (u128) x * x % n;
18         if (x == n - 1) return false;
19     }
20     return true;
21 }
22 bool miller_rabin(u64 n) {
23     if (n < 2) return false;
24     ll r = 0;
25     u64 d = n - 1;
26     while ((d & 1) == 0) {
27         d >>= 1;
28         r++;
29     }
30     for (ll a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
   31, 37}) {
31         if (n == a) return true;
32         if (check_composite(n, a, d, r)) return false;

```

```

33 }
34 return true;
35 }

```

## 5.10 Modulo Inverse

```

1 ll mod_inv(ll a, ll m) {
2     if (m == 1) return 0;
3     ll m0 = m, x = 1, y = 0;
4     while (a > 1) {
5         ll q = a / m, t = m;
6         m = a % m;
7         a = t;
8         t = y;
9         y = x - q * y;
10        x = t;
11    }
12    if (x < 0) x += m0;
13    return x;
14 }

```

## 5.11 Pollard Rho Brent

```

1 ll mult(ll a, ll b, ll mod) {
2     return (__int128_t) a * b % mod;
3 }
4 ll f(ll x, ll c, ll mod) {
5     return (mult(x, x, mod) + c) % mod;
6 }
7 ll pollard_rho_brent(ll n, ll x0 = 2, ll c = 1) {
8     ll x = x0, g = 1, q = 1, xs, y, m = 128, l = 1;
9     while (g == 1) {
10        y = x;
11        for (ll i = 1; i < l; i++) x = f(x, c, n);
12        ll k = 0;
13        while (k < l && g == 1) {
14            xs = x;
15            for (ll i = 0; i < m && i < l - k; i++) {
16                x = f(x, c, n);
17                q = mult(q, abs(y - x), n);
18            }
19            g = __gcd(q, n);
20            k += m;
21        }
22        l *= 2;
23    }
24    if (g == n) {
25        do {
26            xs = f(xs, c, n);
27            g = __gcd(abs(xs - y), n);
28        } while (g == 1);
29    }
30    return g;
31 }

```

## 5.12 Range Sieve

```

1 vector<bool> range_sieve(ll l, ll r) {
2     ll n = sqrt(r);
3     vector<bool> is_prime(n + 1, true);
4     vector<ll> prime;
5     is_prime[0] = is_prime[1] = false;

```

```

6     prime.push_back(2);
7     for (ll i = 4; i <= n; i += 2) is_prime[i] = false;
8     for (ll i = 3; i <= n; i += 2) {
9         if (is_prime[i]) {
10            prime.push_back(i);
11            for (ll j = i * i; j <= n; j += i) is_prime[j] = false;
12        }
13    }
14    vector<bool> result(r - l + 1, true);
15    for (ll i : prime)
16        for (ll j = max(i * i, (l + i - 1) / i * i); j
17              <= r; j += i)
18            result[j - l] = false;
19    if (l == 1) result[0] = false;
20    return result;

```

## 5.13 Segmented Sieve

```

1 vector<ll> segmented_sieve(ll n) {
2     const ll S = 10000;
3     ll nsqrt = sqrt(n);
4     vector<char> is_prime(nsqrt + 1, true);
5     vector<ll> prime;
6     is_prime[0] = is_prime[1] = false;
7     prime.push_back(2);
8     for (ll i = 4; i <= nsqrt; i += 2) {
9         is_prime[i] = false;
10    }
11    for (ll i = 3; i <= nsqrt; i += 2) {
12        if (is_prime[i]) {
13            prime.push_back(i);
14            for (ll j = i * i; j <= nsqrt; j += i) {
15                is_prime[j] = false;
16            }
17        }
18    }
19    vector<ll> result;
20    vector<char> block(S);
21    for (ll k = 0; k * S <= n; k++) {
22        fill(block.begin(), block.end(), true);
23        for (ll p : prime) {
24            for (ll j = max((k * S + p - 1) / p, p) * p -
25                  k * S; j < S; j += p) {
26                block[j] = false;
27            }
28            if (k == 0) {
29                block[0] = block[1] = false;
30            }
31            for (ll i = 0; i < S && k * S + i <= n; i++) {
32                if (block[i]) {
33                    result.push_back(k * S + i);
34                }
35            }
36        }
37    }
38    return result;

```

## 5.14 Sum Of Divisors

```

1 ll sum_of_divisors(ll num) {
2     ll total = 1;

```

```

3     for (int i = 2; (ll)i * i <= num; i++) {
4         if (num % i == 0) {
5             int e = 0;
6             do {
7                 e++;
8                 num /= i;
9             } while (num % i == 0);
10            ll sum = 0, pow = 1;
11            do {
12                sum += pow;
13                pow *= i;
14            } while (e-- > 0);
15            total *= sum;
16        }
17    }
18    if (num > 1) total *= (1 + num);
19    return total;
20 }

```

## 5.15 Tonelli Shanks

```

1 ll legendre(ll a, ll p) {
2     return bin_pow_mod(a, (p - 1) / 2, p);
3 }
4 ll tonelli_shanks(ll n, ll p) {
5     if (legendre(n, p) == p - 1) {
6         return -1;
7     }
8     if (p % 4 == 3) {
9         return bin_pow_mod(n, (p + 1) / 4, p);
10    }
11    ll Q = p - 1, S = 0;
12    while (Q % 2 == 0) {
13        Q /= 2;
14        S++;
15    }
16    ll z = 2;
17    for (; z < p; z++) {
18        if (legendre(z, p) == p - 1) {
19            break;
20        }
21    }
22    ll M = S, c = bin_pow_mod(z, Q, p), t =
23        bin_pow_mod(n, Q, p), R = bin_pow_mod(n, (Q
24        + 1) / 2, p);
25    while (t % p != 1) {
26        if (t % p == 0) {
27            return 0;
28        }
29        ll i = 1, t2 = t * t % p;
30        for (; i < M; i++) {
31            if (t2 % p == 1) {
32                break;
33            }
34            t2 = t2 * t2 % p;
35        }
36        ll b = bin_pow_mod(c, bin_pow_mod(2, M - i - 1,
37        p), p);
38        M = i;
39        c = b * b % p;
40        t = t * c % p;
41        R = R * b % p;
42    }
43    return R;

```



## 6 Miscellaneous

### 6.1 Gauss

```
1 const double EPS = 1e-9;
2 const ll INF = 2;
3 ll gauss(vector<vector<double>> a, vector<double>
  &ans) {
4     ll n = (ll) a.size(), m = (ll) a[0].size() - 1;
5     vector<ll> where (m, -1);
6     for (ll col = 0, row = 0; col < m && row < n; ++
      col) {
7         ll sel = row;
8         for (ll i = row; i < n; ++i) {
9             if (abs(a[i][col]) > abs(a[sel][col])) {
10                sel = i;
11            }
12        }
13        if (abs(a[sel][col]) < EPS) {
14            continue;
15        }
16        for (ll i = col; i <= m; ++i) {
17            swap(a[sel][i], a[row][i]);
18        }
19        where[col] = row;
20        for (ll i = 0; i < n; ++i) {
21            if (i != row) {
22                double c = a[i][col] / a[row][col];
23                for (ll j = col; j <= m; ++j) {
24                    a[i][j] -= a[row][j] * c;
25                }
26            }
27        }
28        ++row;
29    }
30    ans.assign(m, 0);
31    for (ll i = 0; i < m; ++i) {
32        if (where[i] != -1) {
33            ans[i] = a[where[i]][m] / a[where[i]][i];
34        }
35    }
36    for (ll i = 0; i < n; ++i) {
37        double sum = 0;
38        for (ll j = 0; j < m; ++j) {
39            sum += ans[j] * a[i][j];
40        }
41        if (abs(sum - a[i][m]) > EPS) {
42            return 0;
43        }
44    }
45    for (ll i = 0; i < m; ++i) {
46        if (where[i] == -1) {
47            return INF;
48        }
49    }
50    return 1;
51 }
```

### 6.2 Ternary Search

```
1 double ternary_search(double l, double r) {
2     double eps = 1e-9;
3     while (r - l > eps) {
4         double m1 = l + (r - l) / 3;
```

```
5         double m2 = r - (r - l) / 3;
6         double f1 = f(m1);
7         double f2 = f(m2);
8         if (f1 < f2) {
9             l = m1;
10        } else {
11            r = m2;
12        }
13    }
14    return f(l);
15 }
```

## 7 References

### 7.1 Ref

```
1 st.insert(4);
2 st.erase(4);
3 st.empty();
4 // permutations
5 do {
6     for (int num : nums) {
7         cout << num << " ";
8     }
9     cout << endl;
10 } while (next_permutation(nums.begin(), nums.end())
  );
11 // bitset
12 int num = 27; // Binary representation: 11011
13 bitset<10> s(string("0010011010")); // from right
  to left
14 bitset<sizeof(int) * 8> bits(num);
15 int setBits = bits.count();
16 // sort
17 sort(v.begin(), v.end());
18 sort(v.rbegin(), v.rend());
19 // custom sort
20 bool comp(string a, string b) {
21     if (a.size() != b.size()) return a.size() < b.size
      ();
22     return a < b;
23 }
24 sort(v.begin(), v.end(), comp);
25 // binary search
26 int a = 0, b = n-1;
27 while (a <= b) { int k = (a+b)/2; if (array[k] == x
  ) {
28     // x found at index k
29     if (array[k] > x) b = k-1; else a = k+1;
30 } // iterator
31 for (auto it = s.begin(); it != s.end(); it++) {
32     cout << *it << "\n";
33 }
34 // hamming distance
35 int hamming(int a, int b) {
36     return __builtin_popcount(a^b);
37 }
38 // custom comparator for pq
39 class Compare {
40 public:
41     bool operator() (T a, T b) {
42         if (cond) return true; // do not swap
43         return false;
44     }
45 };
46 priority_queue<PII, vector<PII>, Compare> ds;
```

## 8 Strings

### 8.1 Count Unique Substrings

```
1 int count_unique_substrings(string const& s) {
2     int n = s.size();
3     const int p = 31;
4     const int m = 1e9 + 9;
5     vector<long long> p_pow(n);
6     p_pow[0] = 1;
7     for (int i = 1; i < n; i++) p_pow[i] = (p_pow[i -
      1] * p) % m;
8     vector<long long> h(n + 1, 0);
9     for (int i = 0; i < n; i++) h[i + 1] = (h[i] + (s
      [i] - 'a' + 1) * p_pow[i]) % m;
10    int cnt = 0;
11    for (int l = 1; l <= n; l++) {
12        unordered_set<long long> hs;
13        for (int i = 0; i <= n - l; i++) {
14            long long cur_h = (h[i + l] + m - h[i]) % m;
15            cur_h = (cur_h * p_pow[n - i - 1]) % m;
16            hs.insert(cur_h);
17        }
18        cnt += hs.size();
19    }
20    return cnt;
21 }
```

### 8.2 Finding Repetitions

```
1 vector<int> z_function(string const& s) {
2     int n = s.size();
3     vector<int> z(n);
4     for (int i = 1, l = 0, r = 0; i < n; i++) {
5         if (i <= r) z[i] = min(r - i + 1, z[i - l]);
6         while (i + z[i] < n && s[z[i]] == s[i + z[i]])
7             z[i]++;
8         if (i + z[i] - 1 > r) {
9             l = i;
10            r = i + z[i] - 1;
11        }
12    }
13    return z;
14 }
15 int get_z(vector<int> const& z, int i) {
16     if (0 <= i && i < (int) z.size()) return z[i];
17     else return 0;
18 }
19 vector<pair<int, int>> repetitions;
20 void convert_to_repetitions(int shift, bool left,
  int cntr, int l, int k1, int k2) {
21     for (int ll = max(1, l - k2); ll <= min(l, k1);
      ll++) {
22         if (left && ll == 1) break;
23         int l2 = l - ll;
24         int pos = shift + (left ? cntr - ll : cntr - l
      - ll + 1);
25         repetitions.emplace_back(pos, pos + 2 * l - 1);
26     }
27 }
28 void find_repetitions(string s, int shift = 0) {
29     int n = s.size();
30     if (n == 1) return;
31     int nu = n / 2;
```

```

31 int nv = n - nu;
32 string u = s.substr(0, nu);
33 string v = s.substr(nu);
34 string ru(u.rbegin(), u.rend());
35 string rv(v.rbegin(), v.rend());
36 find_repetitions(u, shift);
37 find_repetitions(v, shift + nu);
38 vector<int> z1 = z_function(ru);
39 vector<int> z2 = z_function(v + '#' + u);
40 vector<int> z3 = z_function(ru + '#' + rv);
41 vector<int> z4 = z_function(v);
42 for (int cntr = 0; cntr < n; cntr++) {
43     int l, k1, k2;
44     if (cntr < nu) {
45         l = nu - cntr;
46         k1 = get_z(z1, nu - cntr);
47         k2 = get_z(z2, nv + 1 + cntr);
48     } else {
49         l = cntr - nu + 1;
50         k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
51         k2 = get_z(z4, (cntr - nu) + 1);
52     }
53     if (k1 + k2 >= l) convert_to_repetitions(shift,
54         cntr < nu, cntr, l, k1, k2);
55 }

```

### 8.3 Group Identical Substrings

```

1 vector<vector<int>> group_identical_strings(vector<
  string> const& s) {
2     int n = s.size();
3     vector<pair<long long, int>> hashes(n);
4     for (int i = 0; i < n; i++) hashes[i] = {
5         compute_hash(s[i]), i};
6     sort(hashes.begin(), hashes.end());
7     vector<vector<int>> groups;
8     for (int i = 0; i < n; i++) {
9         if (i == 0 || hashes[i].first != hashes[i - 1].
10             first) groups.emplace_back();
11         groups.back().push_back(hashes[i].second);
12     }
13     return groups;

```

### 8.4 Hashing

```

1 ll compute_hash(string const& s) {
2     const ll p = 31, m = 1e9 + 9;
3     ll hash_value = 0, p_pow = 1;
4     for (char c : s) {
5         hash_value = (hash_value + (c - 'a' + 1) *
6             p_pow) % m;
7         p_pow = (p_pow * p) % m;
8     }
9     return hash_value;

```

### 8.5 Knuth Morris Pratt

```

1 vector<ll> prefix_function(string s) {

```

```

2     ll n = (ll) s.length();
3     vector<ll> pi(n);
4     for (ll i = 1; i < n; i++) {
5         ll j = pi[i - 1];
6         while (j > 0 && s[i] != s[j]) j = pi[j - 1];
7         if (s[i] == s[j]) j++;
8         pi[i] = j;
9     }
10    return pi;
11 }
12 // count occurrences
13 vector<int> ans(n + 1);
14 for (int i = 0; i < n; i++)
15     ans[pi[i]]++;
16 for (int i = n - 1; i > 0; i--)
17     ans[pi[i - 1]] += ans[i];
18 for (int i = 0; i <= n; i++)
19     ans[i]++;

```

### 8.6 Longest Common Prefix

```

1 vector<int> lcp_construction(string const& s,
  vector<int> const& p) {
2     int n = s.size();
3     vector<int> rank(n, 0);
4     for (int i = 0; i < n; i++) rank[p[i]] = i;
5     int k = 0;
6     vector<int> lcp(n - 1, 0);
7     for (int i = 0; i < n; i++) {
8         if (rank[i] == n - 1) {
9             k = 0;
10            continue;
11        }
12        int j = p[rank[i] + 1];
13        while (i + k < n && j + k < n && s[i + k] == s[
14            j + k]) k++;
15        lcp[rank[i]] = k;
16        if (k) k--;
17    }
18    return lcp;

```

### 8.7 Manacher

```

1 vector<int> manacher_odd(string s) {
2     int n = s.size();
3     s = "$" + s + "^";
4     vector<int> p(n + 2);
5     int l = 1, r = 1;
6     for (int i = 1; i <= n; i++) {
7         p[i] = max(0, min(r - i, p[l + (r - i)]));
8         while (s[i - p[i]] == s[i + p[i]]) p[i]++;
9         if (i + p[i] > r) l = i - p[i], r = i + p[i];
10    }
11    return vector<int>(begin(p) + 1, end(p) - 1);
12 }
13 vector<int> manacher(string s) {
14     string t;
15     for (auto c : s) t += string("#") + c;
16     auto res = manacher_odd(t + "#");
17     return vector<int>(begin(res) + 1, end(res) - 1);
18 }

```

### 8.8 Rabin Karp

```

1 vector<ll> rabin_karp(string const& s, string const
  & t) {
2     const ll p = 31, m = 1e9 + 9;
3     ll S = s.size(), T = t.size();
4     vector<ll> p_pow(max(S, T));
5     p_pow[0] = 1;
6     for (ll i = 1; i < (ll) p_pow.size(); i++) p_pow[
7         i] = (p_pow[i - 1] * p) % m;
8     vector<ll> h(T + 1, 0);
9     for (ll i = 0; i < T; i++) h[i + 1] = (h[i] + (t[
10         i] - 'a' + 1) * p_pow[i]) % m;
11     ll h_s = 0;
12     for (ll i = 0; i < S; i++) h_s = (h_s + (s[i] - '
13         a' + 1) * p_pow[i]) % m;
14     vector<ll> occurrences;
15     for (ll i = 0; i + S - 1 < T; i++) {
16         ll cur_h = (h[i + S] + m - h[i]) % m;
17         if (cur_h == h_s * p_pow[i] % m) occurrences.
18             push_back(i);
19     }
20     return occurrences;

```

### 8.9 Suffix Array

```

1 vector<int> sort_cyclic_shifts(string const& s) {
2     int n = s.size();
3     const int alphabet = 256;
4     vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
5     for (int i = 0; i < n; i++) cnt[s[i]]++;
6     for (int i = 1; i < alphabet; i++) cnt[i] += cnt[
7         i - 1];
8     for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
9     c[p[0]] = 0;
10    int classes = 1;
11    for (int i = 1; i < n; i++) {
12        if (s[p[i]] != s[p[i - 1]]) classes++;
13        c[p[i]] = classes - 1;
14    }
15    vector<int> pn(n), cn(n);
16    for (int h = 0; (1 << h) < n; ++h) {
17        for (int i = 0; i < n; i++) {
18            pn[i] = p[i] - (1 << h);
19            if (pn[i] < 0)
20                pn[i] += n;
21        }
22        fill(cnt.begin(), cnt.begin() + classes, 0);
23        for (int i = 0; i < n; i++) cnt[c[pn[i]]]++;
24        for (int i = 1; i < classes; i++) cnt[i] += cnt[
25            i - 1];
26        for (int i = n - 1; i >= 0; i--) p[--cnt[c[pn[i]
27            ]]] = pn[i];
28        cn[p[0]] = 0;
29        classes = 1;
30        for (int i = 1; i < n; i++) {
31            pair<int, int> cur = {c[p[i]], c[(p[i] + (1
32                << h)) % n]};
33            pair<int, int> prev = {c[p[i - 1]], c[(p[i - 1] +
34                (1 << h)) % n]};
35            if (cur != prev) ++classes;
36            cn[p[i]] = classes - 1;
37        }
38    }
39    c.swap(cn);

```

```

34     }
35     return p;
36 }
37 vector<int> build_suff_arr(string s) {
38     s += "$";
39     vector<int> sorted_shifts = sort_cyclic_shifts(s)
40         ;
41     sorted_shifts.erase(sorted_shifts.begin());
42     return sorted_shifts;
43 }
44 // compare two substrings
45 int compare(int i, int j, int l, int k) {
46     pair<int, int> a = {c[k][i], c[k][(i + l - (1 <<

```

```

46         k)) % n]};
47     pair<int, int> b = {c[k][j], c[k][(j + l - (1 <<
48         k)) % n]};
49     return a == b ? 0 : a < b ? -1 : 1;
50 }

```

## 8.10 Z Function

```

1 vector<int> z_function(string s) {
2     int n = s.size();
3     vector<int> z(n);

```

```

4     for (int i = 1, l = 0, r = 0; i < n; i++) {
5         if (i < r) z[i] = min(r - i, z[i - l]);
6         while (i + z[i] < n && s[z[i]] == s[i + z[i]])
7             z[i]++;
8         if (i + z[i] > r) {
9             l = i;
10            r = i + z[i];
11        }
12    }
13    return z;

```