

## Contents

<b>1 Data Structures</b>	1
1.1 Disjoint Set Union	1
1.2 Minimum Queue	1
1.3 Range Add Point Query	1
1.4 Range Add Range Query	2
1.5 Segment Tree	2
1.6 Segment Tree 2d	2
1.7 Sparse Table	3
1.8 Sparse Table 2d	3
<b>2 Dynamic Programming</b>	3
2.1 Divide And Conquer	3
2.2 Edit Distance	3
2.3 Knapsack	3
2.4 Knuth Optimization	3
2.5 Longest Common Subsequence	4
2.6 Longest Increasing Subsequence	4
2.7 Max Sum	4
2.8 Subset Sum	4
<b>3 Geometry</b>	4
3.1 Areas	4
3.2 Basic Geometry	4
3.3 Circle Line Intersection	5
3.4 Convex Hull	5
3.5 Count Lattices	5
3.6 Line Intersection	5
3.7 Line Sweep	5
3.8 Minkowski Sum	6
3.9 Nearest Points	6
3.10 Point In Convex	6
3.11 Segment Intersection	7
<b>4 Graph Theory</b>	7
4.1 Articulation Point	7
4.2 Bellman Ford	7
4.3 Bridge	7
4.4 Centroid Decomposition	7
4.5 Dijkstra	8
4.6 Dinics	8
4.7 Edmonds Karp	8
4.8 Fast Second Mst	8
4.9 Find Cycle	9
4.10 Floyd Warshall	9
4.11 Ford Fulkerson	9
4.12 Hierholzer	10
4.13 Hungarian	10
4.14 Is Bipartite	10
4.15 Is Cyclic	10
4.16 Kahn	10
4.17 Kosaraju	10
4.18 Kruskals	11
4.19 Kuhn	11
4.20 Lowest Common Ancestor	11
4.21 Maximum Bipartite Matching	11
4.22 Min Cost Flow	11
4.23 Prim	12
4.24 Topological Sort	12
4.25 Zero One Bfs	12
<b>5 Math</b>	12
5.1 Chinese Remainder Theorem	12
5.2 Extended Euclidean	12

5.3 Factorial Modulo	12
5.4 Fast Fourier Transform	12
5.5 Fibonacci	13
5.6 Find All Solutions	13
5.7 Linear Sieve	13
5.8 Matrix	13
5.9 Miller Rabin	14
5.10 Modulo Inverse	14
5.11 Pollard Rho Brent	14
5.12 Range Sieve	14
5.13 Segmented Sieve	14
5.14 Sum Of Divisors	14
5.15 Tonelli Shanks	15
<b>6 Miscellaneous</b>	15
6.1 Gauss	15
6.2 Ternary Search	15
<b>7 References</b>	15
7.1 Ref	15
<b>8 Strings</b>	15
8.1 Count Unique Substrings	15
8.2 Finding Repetitions	16
8.3 Group Identical Substrings	16
8.4 Hashing	16
8.5 Knuth Morris Pratt	16
8.6 Longest Common Prefix	16
8.7 Manacher	16
8.8 Rabin Karp	16
8.9 Suffix Array	17
8.10 Z Function	17

## 1 Data Structures

## 1.1 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 };

```

## 1.2 Minimum Queue

```

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

```

## 1.3 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) : i(i), j(j) {
20        if (j - i == 1) {
21            lc = rc = nullptr;
22            val = (T) a[i];
23            return;
24        }
25        int k = (i + j) / 2;
26        lc = new SegTreeNode<T, InType>(a, i, k);
27        rc = new SegTreeNode<T, InType>(a, k, j);
28        val = 0;
29    }
30    void range_add(int l, int r, T x) {
31        if (r <= i || j <= l) return;
32        if (l <= i && j <= r) {
33            val += x;

```

```

34     return;
35 }
36 lc->range_add(l, r, x);
37 rc->range_add(l, r, x);
38 }
39 T point_query(int k) {
40     if (k < i || j <= k) return IDN;
41     if (j - i == 1) return val;
42     return val + lc->point_query(k) + rc->
        point_query(k);
43 }
44 };
45 template<typename T, typename InType = T>
46 class SegTree {
47 public:
48     SegTreeNode<T, InType> root;
49     SegTree(int n) : root(0, n) {}
50     SegTree(const vector<InType>& a) : root(a, 0, a.
        size()) {}
51     void range_add(int l, int r, T x) { root.
        range_add(l, r, x); }
52     T point_query(int k) { return root.point_query(k)
        ; }
53 };

```

#### 1.4 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
        ) : i(i), j(j) {
20         if (j - i == 1) {
21             lc = rc = nullptr;
22             val = (T) a[i];
23             return;
24         }
25         int k = (i + j) / 2;
26         lc = new SegTreeNode<T, InType>(a, i, k);
27         rc = new SegTreeNode<T, InType>(a, k, j);
28         val = operation(lc->val, rc->val);
29     }
30     void propagate() {
31         if (to_add == 0) return;
32         val += to_add;
33         if (j - i > 1) {
34             lc->to_add += to_add;
35             rc->to_add += to_add;
36         }
37         to_add = 0;
38     }
39     void range_add(int l, int r, T delta) {

```

```

40     propagate();
41     if (r <= i || j <= l) return;
42     if (l <= i && j <= r) {
43         to_add += delta;
44         propagate();
45     } else {
46         lc->range_add(l, r, delta);
47         rc->range_add(l, r, delta);
48         val = operation(lc->val, rc->val);
49     }
50 }
51 T range_query(int l, int r) {
52     propagate();
53     if (l <= i && j <= r) return val;
54     if (j <= l || r <= i) return IDN;
55     return operation(lc->range_query(l, r), rc->
        range_query(l, r));
56 }
57 T operation(T x, T y) {}
58 };
59 template<typename T, typename InType = T>
60 class SegTree {
61 public:
62     SegTreeNode<T, InType> root;
63     SegTree(int n) : root(0, n) {}
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 };

```

#### 1.5 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
        ) : i(i), j(j) {
20         if (j - i == 1) {
21             lc = rc = nullptr;
22             val = (T) a[i];
23             return;
24         }
25         int k = (i + j) / 2;
26         lc = new SegTreeNode<T, InType>(a, i, k);
27         rc = new SegTreeNode<T, InType>(a, k, j);
28         val = op(lc->val, rc->val);
29     }
30     void set(int k, T x) {
31         if (k < i || j <= k) return;

```

```

32     if (j - i == 1) {
33         val = x;
34         return;
35     }
36     lc->set(k, x);
37     rc->set(k, x);
38     val = op(lc->val, rc->val);
39 }
40 T range_query(int l, int r) {
41     if (l <= i && j <= r) return val;
42     if (j <= l || r <= i) return IDN;
43     return op(lc->range_query(l, r), rc->
        range_query(l, r));
44 }
45 T op(T x, T y) {}
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.
        size()) {}
53     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 };

```

#### 1.6 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,
        int i, int j) : i(i), j(j) {
9          tree_size = a[0].size();
10         if (j - i == 1) {
11             lc = rc = nullptr;
12             seg_tree = new SegTree<T, InType>(a[i]);
13             return;
14         }
15         int k = (i + j) / 2;
16         lc = new SegTree2dNode<T, InType>(a, i, k);
17         rc = new SegTree2dNode<T, InType>(a, k, j);
18         seg_tree = new SegTree<T, InType>(vector<T>(
            tree_size));
19         operation_2d(lc->seg_tree, rc->seg_tree);
20     }
21     ~SegTree2dNode() {
22         delete lc;
23         delete rc;
24     }
25     void set_2d(int kx, int ky, T x) {
26         if (kx < i || j <= kx) return;
27         if (j - i == 1) {
28             seg_tree->set(ky, x);
29             return;
30         }
31         lc->set_2d(kx, ky, x);
32         rc->set_2d(kx, ky, x);
33         operation_2d(lc->seg_tree, rc->seg_tree);
34     }

```

```

35 T range_query_2d(int lx, int rx, int ly, int ry)
36 {
37     if (lx <= i && j <= rx) return seg_tree->
38         range_query(ly, ry);
39     if (j <= lx || rx <= i) return -INF;
40     return max(lc->range_query_2d(lx, rx, ly, ry),
41         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 template<typename T, typename InType = T>
51 class SegTree2d {
52 public:
53     SegTree2dNode<T, InType> root;
54     SegTree2d() {}
55     SegTree2d(const vector<vector<InType>>& mat) :
56         root(mat, 0, mat.size()) {}
57     void set_2d(int kx, int ky, T x) { root.set_2d(kx
58         , ky, x); }
59     T range_query_2d(int lx, int rx, int ly, int ry)
60     { return root.range_query_2d(lx, rx, ly, ry)
61         ; }
62 };

```

## 1.7 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>
6     &array) {
7     vector<vector<ll>> st(K + 1, vector<ll>(N + 1));
8     for (ll i = 0; i < N; i++) st[0][i] = array[i];
9     for (ll i = 1; i <= K; i++)
10         for (ll j = 0; j + (1 << i) <= N; j++)
11             st[i][j] = st[i - 1][j] + st[i - 1][j + (1 <<
12                 (i - 1))];
13     return st;
14 }
15 ll sum_query(ll L, ll R, ll K, vector<vector<ll>> &
16     st) {
17     ll sum = 0;
18     for (ll i = K; i >= 0; i--) {
19         if ((1 << i) <= R - L + 1) {
20             sum += st[i][L];
21             L += 1 << i;
22         }
23     }
24     return sum;
25 }
26 vector<vector<ll>> build_min(ll N, ll K, vector<ll>
27     &array) {
28     vector<vector<ll>> st(K + 1, vector<ll>(N + 1));
29     for (ll i = 0; i < N; i++) st[0][i] = array[i];
30     for (ll i = 1; i <= K; i++)
31         for (ll j = 0; j + (1 << i) <= N; j++)
32             st[i][j] = min(st[i - 1][j], st[i - 1][j + (1
33                 << (i - 1))]);
34     return st;
35 }

```

```

30 ll min_query(ll L, ll R, vector<vector<ll>> &st) {
31     ll i = log2_floor(R - L + 1);
32     return min(st[i][L], st[i][R - (1 << i) + 1]);
33 }

```

## 1.8 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++)
17             for (int j = 0; j + (1 << k) - 1 < m; j++)
18                 table[i][j][0][k] = min(table[i][j][0][k -
19                     1], table[i][j + (1 << (k - 1))][0][k -
20                         1]);
21     for (int k = 1; k <= (int)(log2(n)); k++)
22         for (int l = 1; l <= (int)(log2(m)); l++)
23             for (int i = 0; i + (1 << k) - 1 < n; i++)
24                 for (int j = 0; j + (1 << l) - 1 < m; j++)
25                     table[i][j][k][l] = min(
26                         min(table[i][j][k - 1][l - 1], table[i
27                             + (1 << (k - 1))][j][k - 1][l -
28                                 1]),
29                         min(table[i][j + (1 << (l - 1))][k -
30                             1][l - 1], table[i + (1 << (k - 1)
31                                 )][j + (1 << (l - 1))][k - 1][l -
32                                     1]));
33 }
34 int rmq(int x1, int y1, int x2, int y2) {
35     int k = log2(x2 - x1 + 1), l = log2(y2 - y1 + 1);
36     return max(
37         max(table[x1][y1][k][l], table[x2 - (1 << k) +
38             1][y1][k][l]),
39         max(table[x1][y2 - (1 << l) + 1][k][l], table[
40             x2 - (1 << k) + 1][y2 - (1 << l) + 1][k][l]
41         ));
42 }

```

## 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 opttr) {
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, opttr); 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, opttr);
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         for (int j = 1; j <= m; j++) {
8             dp[i][j] = j;
9         }
10        for (int i = 1; i <= n; i++) {
11            for (int j = 1; j <= m; j++) {
12                dp[i][j] = min({dp[i - 1][j] + 1, dp[i][j -
13                    1] + 1, dp[i - 1][j - 1] + (x[i - 1] !=
14                        y[j - 1])});
15            }
16        }
17        return dp[n][m];
18    }

```

### 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) {
7                 dp[w] = max(dp[w], dp[w - wt[i - 1]] + val[
8                     i - 1]);
9             }
10        }
11        return dp[W];
12    }

```

### 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));

```

```

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

```

## 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) {

```

```

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

```

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

```

## 3 Geometry

### 3.1 Areas

```

1  int signed_area_parallelogram(point2d p1, point2d
2      p2, point2d p3) {
3      return cross(p2 - p1, p3 - p2);

```

```

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

```

## 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 * a; }
37 ftype dot(point2d a, point2d b) { return a.x * b.x
38     + a.y * b.y; }
39 ftype dot(point3d a, point3d b) { return a.x * b.x
40     + a.y * b.y + a.z * b.z; }
41 ftype norm(point2d a) { return sqrt(dot(a, a)); }
42 double abs(point2d a) { return sqrt(norm(a)); }

```

```

35 double proj(point2d a, point2d b) { return dot(a, b)
    } / abs(b); }
36 double angle(point2d a, point2d b) { return acos(
    dot(a, b) / abs(a) / abs(b)); }
37 point3d cross(point3d a, point3d b) { return
    point3d(a.y * b.z - a.z * b.y, a.z * b.x - a.x
    * b.z, a.x * b.y - a.y * b.x); }
38 ftype triple(point3d a, point3d b, point3d c) {
    return dot(a, cross(b, c)); }
39 ftype cross(point2d a, point2d b) { return a.x * b.
    y - a.y * b.x; }
40 point2d intersect(point2d a1, point2d d1, point2d
    a2, point2d d2) { return a1 + cross(a2 - a1,
    d2) / cross(d1, d2) * d1; }
41 point3d intersect(point3d a1, point3d n1, point3d
    a2, point3d n2, point3d a3, point3d n3) {
42     point3d x(n1.x, n2.x, n3.x);
43     point3d y(n1.y, n2.y, n3.y);
44     point3d z(n1.z, n2.z, n3.z);
45     point3d d(dot(a1, n1), dot(a2, n2), dot(a3, n3));
46     return point3d(triple(d, y, z), triple(x, d, z),
    triple(x, y, d)) / triple(n1, n2, n3);
47 }

```

### 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)) <
    EPS) {
7     puts ("1 point");
8     cout << x0 << ' ' << y0 << '\n';
9 } else {
10     double d = r * r - c * c / (a * a + b * b);
11     double mult = sqrt (d / (a * a + b * b));
12     double ax, ay, bx, by;
13     ax = x0 + b * mult;
14     bx = x0 - b * mult;
15     ay = y0 + a * mult;
16     by = y0 - a * mult;
17     puts ("2 points");
18     cout << ax << ' ' << ay << '\n' << bx << ' ' <<
    by << '\n';
19 }

```

### 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)
    + c.x * (a.y - b.y);
6     if (v < 0) {
7         return -1;
8     } else if (v > 0) {
9         return +1;
10    }
11    return 0;
12 }
13 bool cw(pt a, pt b, pt c, bool include_collinear) {
14     ll o = orientation(a, b, c);

```

```

15     return o < 0 || (include_collinear && o == 0);
16 }
17 bool collinear(pt a, pt b, pt c) {
18     return orientation(a, b, c) == 0;
19 }
20 void convex_hull(vector<pt>& a, bool
    include_collinear = false) {
21     pt p0 = *min_element(a.begin(), a.end(), [](pt a,
    pt b) {
22         return make_pair(a.y, a.x) < make_pair(b.y, b.x
    );
23     });
24     sort(a.begin(), a.end(), [&p0](const pt& a, const
    pt& b) {
25         ll o = orientation(p0, a, b);
26         if (o == 0) {
27             return (p0.x - a.x) * (p0.x - a.x) + (p0.y -
    a.y) * (p0.y - a.y)
28                 < (p0.x - b.x) * (p0.x - b.x) + (p0.y -
    b.y) * (p0.y - b.y);
29         }
30         return o < 0;
31     });
32     if (include_collinear) {
33         ll i = (ll) a.size() - 1;
34         while (i >= 0 && collinear(p0, a[i], a.back()))
35             i--;
36         reverse(a.begin() + i + 1, a.end());
37     }
38     vector<pt> st;
39     for (ll i = 0; i < (ll) a.size(); i++) {
40         while (st.size() > 1 && !cw(st[st.size() - 2],
    st.back(), a[i], include_collinear)) {
41             st.pop_back();
42         }
43         st.push_back(a[i]);
44     }
45     a = st;

```

### 3.5 Count Lattices

```

1 int count_lattices(Fraction k, Fraction b, long
    long n) {
2     auto fk = k.floor();
3     auto fb = b.floor();
4     auto cnt = 0LL;
5     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.
    floor()) / k, t.floor());
13    return cnt;
14 }

```

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

```

### 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
    .x);
9     }
10 }
11 bool intersectld(double l1, double r1, double l2,
    double r2) {
12     if (l1 > r1) swap(l1, r1);
13     if (l2 > r2) swap(l2, r2);
14     return max(l1, l2) <= min(r1, r2) + EPS;
15 }
16 ll vec(const pt& a, const pt& b, const pt& c) {
17     double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y)
    * (c.x - a.x);
18     return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
19 }
20 bool intersect(const seg& a, const seg& b) {
21     return intersectld(a.p.x, a.q.x, b.p.x, b.q.x) &&
    intersectld(a.p.y, a.q.y, b.p.y, b.q.y) &&
22         vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <=
    0 &&
23         vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <=
    0;
24 }
25 }
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.
    x));
28     return a.get_y(x) < b.get_y(x) - EPS;
29 }
30 struct event {
31     double x;
32     ll tp, id;
33     event() {}
34     event(double x, ll tp, ll id) : x(x), tp(tp), id(
    id) {}
35 }
36 bool operator<(const event& e) const {
37     if (abs(x - e.x) > EPS) return x < e.x;
38     return tp > e.tp;
39 }
40 set<seg> s;
41 vector<set<seg>::iterator> where;

```

```

42 set<seg>::iterator prev(set<seg>::iterator it) {
43     return it == s.begin() ? s.end() : --it;
44 }
45 set<seg>::iterator next(set<seg>::iterator it) {
46     return ++it;
47 }
48 pair<ll, ll> solve(const vector<seg>& a) {
49     ll n = (ll) a.size();
50     vector<event> e;
51     for (ll i = 0; i < n; ++i) {
52         e.push_back(event(min(a[i].p.x, a[i].q.x), +1,
53                             i));
54         e.push_back(event(max(a[i].p.x, a[i].q.x), -1,
55                             i));
56     }
57     sort(e.begin(), e.end());
58     s.clear();
59     where.resize(a.size());
60     for (size_t i = 0; i < e.size(); ++i) {
61         ll id = e[i].id;
62         if (e[i].tp == +1) {
63             set<seg>::iterator nxt = s.lower_bound(a[id])
64             , prv = prev(nxt);
65             if (nxt != s.end() && intersect(*nxt, a[id]))
66                 return make_pair(nxt->id, id);
67             if (prv != s.end() && intersect(*prv, a[id]))
68                 return make_pair(prv->id, id);
69             where[id] = s.insert(nxt, a[id]);
70         } else {
71             set<seg>::iterator nxt = next(where[id]), prv
72             = prev(where[id]);
73             if (nxt != s.end() && prv != s.end() &&
74                 intersect(*nxt, *prv)) return make_pair(
75                 prv->id, nxt->id);
76             s.erase(where[id]);
77         }
78     }
79     return make_pair(-1, -1);
80 }

```

### 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}; }
7     ll cross(const pt & p) const { return x * p.y - y
8         * p.x; }
9 };
10 void reorder_polygon(vector<pt> & P) {
11     size_t pos = 0;
12     for (size_t i = 1; i < P.size(); i++) {
13         if (P[i].y < P[pos].y || (P[i].y == P[pos].y &&
14             P[i].x < P[pos].x)) pos = i;
15     }
16     rotate(P.begin(), P.begin() + pos, P.end());
17 }
18 vector<pt> minkowski(vector<pt> P, vector<pt> Q) {
19     // the first vertex must be the lowest
20     reorder_polygon(P);
21     reorder_polygon(Q);
22     // we must ensure cyclic indexing
23     P.push_back(P[0]);
24     P.push_back(P[1]);
25     Q.push_back(Q[0]);

```

```

22 Q.push_back(Q[1]);
23 // main part
24 vector<pt> result;
25 size_t i = 0, j = 0;
26 while (i < P.size() - 2 || j < Q.size() - 2) {
27     result.push_back(P[i] + Q[j]);
28     auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] -
29         Q[j]);
30     if (cross >= 0 && i < P.size() - 2) ++i;
31     if (cross <= 0 && j < Q.size() - 2) ++j;
32 }
33 return result;

```

### 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     {
13         return a.y < b.y; }
14 };
15 ll n;
16 vector<pt> a;
17 double mindist;
18 pair<ll, ll> best_pair;
19 void upd_ans(const pt & a, const pt & b) {
20     double dist = sqrt((a.x - b.x) * (a.x - b.x) + (a
21         .y - b.y) * (a.y - b.y));
22     if (dist < mindist) {
23         mindist = dist;
24         best_pair = {a.id, b.id};
25     }
26 }
27 vector<pt> t;
28 void rec(ll l, ll r) {
29     if (r - l <= 3) {
30         for (ll i = l; i < r; ++i)
31             for (ll j = i + 1; j < r; ++j)
32                 upd_ans(a[i], a[j]);
33         sort(a.begin() + l, a.begin() + r, cmp_y());
34         return;
35     }
36     ll m = (l + r) >> 1, midx = a[m].x;
37     rec(l, m);
38     rec(m, r);
39     merge(a.begin() + l, a.begin() + m, a.begin() + m
40         , a.begin() + r, t.begin(), cmp_y());
41     copy(t.begin(), t.begin() + r - l, a.begin() + l)
42     ;
43     ll tsz = 0;
44     for (ll i = l; i < r; ++i) {
45         if (abs(a[i].x - midx) < mindist) {
46             for (ll j = tsz - 1; j >= 0 && a[i].y - t[j].
47                 y < mindist; --j)
48                 upd_ans(a[i], t[j]);
49             t[tsz++] = a[i];
50         }
51     }
52 }

```

```

46 t.resize(n);
47 sort(a.begin(), a.end(), cmp_x());
48 mindist = 1E20;
49 rec(0, n);

```

### 3.10 Point In Convex

```

1 struct pt {
2     long long x, y;
3     pt() {}
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 sqLen() 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     seq.resize(n);
43     for (int i = 0; i < n; i++) seq[i] = points[i +
44         1] - points[0];
45     translation = points[0];
46 }
47 bool pointInConvexPolygon(pt point) {
48     point = point - translation;
49     if (seq[0].cross(point) != 0 && sgn(seq[0].cross(
50         point)) != sgn(seq[0].cross(seq[n - 1])))
51         return false;
52     if (seq[n - 1].cross(point) != 0 && sgn(seq[n -
53         1].cross(point)) != sgn(seq[n - 1].cross(seq
54         [0])))
55         return false;
56     if (seq[0].cross(point) == 0)
57         return seq[0].sqLen() >= point.sqLen();
58     int l = 0, r = n - 1;
59     while (r - l > 1) {
60         int mid = (l + r) / 2;

```

```

46     int pos = mid;
47     if (seq[pos].cross(point) >= 0) l = mid;
48     else r = mid;
49 }
50 int pos = l;
51 return pointInTriangle(seq[pos], seq[pos + 1], pt
    (0, 0), point);
52 }

```

### 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.y
23         + c; }
24 };
25 double det(double a, double b, double c, double d) {
26     return a * d - b * c;
27 }
28 inline bool betw(double l, double r, double x) {
29     return min(l, r) <= x + EPS && x <= max(l, r) +
30     EPS;
31 }
32 inline bool intersect_ld(double a, double b, double
33     c, double d) {
34     if (a > b) swap(a, b);
35     if (c > d) swap(c, d);
36     return max(a, c) <= min(b, d) + EPS;
37 }
38 bool intersect(pt a, pt b, pt c, pt d, pt& left, pt
39     & right) {
40     if (!intersect_ld(a.x, b.x, c.x, d.x) || !
41         intersect_ld(a.y, b.y, c.y, d.y)) return
42         false;
43     line m(a, b);
44     line n(c, d);
45     double zn = det(m.a, m.b, n.a, n.b);
46     if (abs(zn) < EPS) {
47         if (abs(m.dist(c)) > EPS || abs(n.dist(a)) >
48             EPS) return false;
49         if (b < a) swap(a, b);
50         if (d < c) swap(c, d);
51         left = max(a, c);
52         right = min(b, d);
53         return true;
54     } else {

```

```

47     left.x = right.x = -det(m.c, m.b, n.c, n.b) /
48         zn;
49     left.y = right.y = -det(m.a, m.c, n.a, n.c) /
50         zn;
51     return betw(a.x, b.x, left.x) && betw(a.y, b.y,
52         left.y) &&
53         betw(c.x, d.x, left.x) && betw(c.y, d.y,
54         left.y);
55 }
56 }

```

## 4 Graph Theory

### 4.1 Articulation Point

```

1  void APUtil(vector<vector<ll>> &adj, ll u, vector<
2      bool> &visited,
3      vector<ll> &disc, vector<ll> &low, ll &time, ll
4      parent, vector<bool> &isAP) {
5      ll children = 0;
6      visited[u] = true;
7      disc[u] = low[u] = ++time;
8      for (auto v : adj[u]) {
9          if (!visited[v]) {
10             children++;
11             APUtil(adj, v, visited, disc, low, time, u,
12                 isAP);
13             low[u] = min(low[u], low[v]);
14             if (parent != -1 && low[v] >= disc[u]) {
15                 isAP[u] = true;
16             }
17             } else if (v != parent) {
18                 low[u] = min(low[u], disc[v]);
19             }
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,
29                 isAP);
30         }
31     }
32     for (ll u = 0; u < n; u++) {
33         if (isAP[u]) {
34             cout << u << " ";
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 " <<
23         v;
24     else {
25         int y = x;
26         for (int i = 0; i < n; ++i) y = p[y];
27         vector<int> path;
28         for (int cur = y; cur = p[cur]) {
29             path.push_back(cur);
30             if (cur == y && path.size() > 1) break;
31         }
32         reverse(path.begin(), path.end());
33         cout << "Negative cycle: ";
34         for (int u : path) cout << u << ' ';
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[
8              child]) continue;
9          subtree_size[node] +=
10             get_subtree_size(child, node);
11     }
12     return subtree_size[node];
13 }
14 int get_centroid(int node, int tree_size, int
15     parent = -1) {
16     for (int child : adj[node]) {
17         if (child == parent || is_removed[
18             child]) continue;
19         if (subtree_size[child] * 2 >
20             tree_size) return get_centroid
21                 (child, tree_size, node);
22     }
23     return node;
24 }
25 void build_centroid_decomp(int node = 0) {
26     int centroid = get_centroid(node,
27         get_subtree_size(node));
28     // do something
29     is_removed[centroid] = true;
30     for (int child : adj[centroid]) {
31         if (is_removed[child]) continue;
32         build_centroid_decomp(child);
33     }
34 }

```

#### 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> &
4      p) {
5      int n = adj.size();
6      d.assign(n, INF);
7      p.assign(n, -1);
8      d[s] = 0;
9      using pii = pair<int, int>;
10     priority_queue<pii, vector<pii>, greater<pii>> q;
11     q.push({0, s});
12     while (!q.empty()) {
13         int v = q.top().second, d_v = q.top().first;
14         q.pop();
15         if (d_v != d[v]) continue;
16         for (auto edge : adj[v]) {
17             int to = edge.first, len = edge.second;
18             if (d[v] + len < d[to]) {
19                 d[to] = d[v] + len;
20                 p[to] = v;
21                 q.push({d[to], to});
22             }
23         }
24     }
25 }

```

#### 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(
5          cap) {}
6  };
7  struct Dinic {
8      const ll flow_inf = 1e18;
9      vector<FlowEdge> edges;
10     vector<vector<int>> adj;
11     int n, m = 0, s, t;
12     vector<int> level, ptr;
13     queue<int> q;
14     Dinic(int n, int s, int t) : n(n), s(s), t(t) {
15         adj.resize(n);
16         level.resize(n);
17         ptr.resize(n);
18     }
19     void add_edge(int v, int u, ll cap) {
20         edges.emplace_back(v, u, cap);
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         }
37         return level[t] != -1;
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
43             (); cid++) {
44             int id = adj[v][cid], u = edges[id].u;
45             if (level[v] + 1 != level[u] || edges[id].cap
46                 - edges[id].flow < 1) continue;
47             ll tr = dfs(u, min(pushed, edges[id].cap -
48                 edges[id].flow));
49             if (tr == 0) continue;
50             edges[id].flow += tr;
51             edges[id ^ 1].flow -= tr;
52             return tr;
53         }
54         return 0;
55     }
56     ll flow() {
57         ll f = 0;
58         while (true) {
59             fill(level.begin(), level.end(), -1);
60             level[s] = 0;
61             q.push(s);
62             if (!bfs()) break;
63             fill(ptr.begin(), ptr.end(), 0);
64             while (ll pushed = dfs(s, flow_inf)) f +=
65                 pushed;
66         }
67         return f;
68     }
69 }

```

#### 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().
11             second;
12         q.pop();
13         for (int next : adj[cur]) {
14             if (parent[next] == -1 && capacity[cur][next
15                 ]) {
16                 parent[next] = cur;
17                 int new_flow = min(flow, capacity[cur][next
18                     ]);
19                 if (next == t) return new_flow;
20                 q.push({next, new_flow});
21             }
22         }
23     }
24     return 0;
25 }
26 int maxflow(int s, int t) {
27     int flow = 0;
28     vector<int> parent(n);
29     int new_flow;
30     while (new_flow = bfs(s, t, parent)) {
31         flow += new_flow;
32         int cur = t;
33         while (cur != s) {
34             int prev = parent[cur];
35             capacity[prev][cur] -= new_flow;
36             capacity[cur][prev] += new_flow;
37             cur = prev;
38         }
39     }
40     return flow;
41 }

```

#### 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),
12     present(N, 0);
13 vector<vector<pair<int, int>>> adj(N), dp(N, vector
14     <pair<int, int>>(1));
15 vector<vector<int>> up(N, vector<int>(1, -1));
16 pair<int, int> combine(pair<int, int> a, pair<int,
17     int> b) {
18     vector<int> v = {a.first, a.second, b.first, b.
19         second};
20 }

```



```

15 int topTwo = -3, topOne = -2;
16 for (int c : v) {
17     if (c > topOne) {
18         topTwo = topOne;
19         topOne = c;
20     } else if (c > topTwo && c < topOne) topTwo = c;
21 }
22 return {topOne, topTwo};
23 }
24 void dfs(int u, int par, int d) {
25     h[u] = 1 + h[par];
26     up[u][0] = par;
27     dp[u][0] = {d, -1};
28     for (auto v : adj[u]) {
29         if (v.first != par) dfs(v.first, u, v.second);
30     }
31 }
32 pair<int, int> lca(int u, int v) {
33     pair<int, int> ans = {-2, -3};
34     if (h[u] < h[v]) swap(u, v);
35     for (int i = 1 - 1; i >= 0; i--) {
36         if (h[u] - h[v] >= (1 << i)) {
37             ans = combine(ans, dp[u][i]);
38             u = up[u][i];
39         }
40     }
41     if (u == v) return ans;
42     for (int i = 1 - 1; i >= 0; i--) {
43         if (up[u][i] != -1 && up[v][i] != -1 && up[u][i]
44             != 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 <= 1 - 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 +
93                 w - rem.first;
94             else if (rem.second != -1) {
95                 if (ans > res + w - rem.second) ans = res +
96                     w - rem.second;
97             }
98         }
99     }
100     cout << ans << "\n";
101     return 0;

```

#### 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 =
33             parent[v]) {
34             cycle.push_back(v);
35         }
36         cycle.push_back(cycle_start);
37         reverse(cycle.begin(), cycle.end());
38         cout << "Cycle found: ";
39         for (ll v : cycle) {
40             cout << v << ' ';

```

```

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] +
8                         dis[k][j]);
9     for (ll i = 0; i < n; i++)
10         for (ll j = 0; j < n; j++)
11             for (ll k = 0; k < n; k++)
12                 if (dis[k][k] < 0 && dis[i][k] < INF && dis
13                     [k][j] < INF)
14                     dis[i][j] = -INF;

```

#### 4.11 Ford Fulkerson

```

1 bool bfs(ll n, vector<vector<ll>> &r_graph, ll s,
2     ll t, vector<ll> &parent) {
3     vector<bool> visited(n, false);
4     queue<ll> q;
5     q.push(s);
6     visited[s] = true;
7     parent[s] = -1;
8     while (!q.empty()) {
9         ll u = q.front();
10        q.pop();
11        for (ll v = 0; v < n; v++) {
12            if (!visited[v] && r_graph[u][v] > 0) {
13                if (v == t) {
14                    parent[v] = u;
15                    return true;
16                }
17                q.push(v);
18                parent[v] = u;
19                visited[v] = true;
20            }
21        }
22    }
23    return false;
24 }
25 ll ford_fulkerson(ll n, vector<vector<ll>> graph,
26     ll s, ll t) {
27     ll u, v;
28     vector<vector<ll>> r_graph;
29     for (u = 0; u < n; u++)
30         for (v = 0; v < n; v++)
31             r_graph[u][v] = graph[u][v];
32     vector<ll> parent;
33     ll max_flow = 0;
34     while (bfs(n, r_graph, s, t, parent)) {
35         ll path_flow = INF;
36         for (v = t; v != s; v = parent[v]) {
37             u = parent[v];
38             path_flow = min(path_flow, r_graph[u][v]);
39         }
40         for (v = t; v != s; v = parent[v]) {
41             u = parent[v];
42             r_graph[u][v] -= path_flow;
43             r_graph[v][u] += path_flow;
44         }
45         max_flow += path_flow;
46     }
47     return max_flow;

```

```

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] = j0;
14                if (minv[j] < delta) delta = minv[j], j1 = j;
15            }
16        for (int j=0; j<=m; ++j)
17            if (used[j]) u[p[j]] += delta, v[j] -= delta;
18        else minv[j] -= delta;
19        j0 = j1;
20    } while (p[j0] != 0);

```

```

21 do {
22     int j1 = way[j0];
23     p[j0] = p[j1];
24     j0 = j1;
25 } while (j0);
26 }
27 vector<int> ans (n+1);
28 for (int j=1; j<=m; ++j)
29     ans[p[j]] = j;
30 int cost = -v[0];

```

#### 4.14 Is Bipartite

```

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

```

#### 4.15 Is Cyclic

```

1 bool is_cyclic_util(int u, vector<vector<int>> &adj, vector<bool> &vis, vector<bool> &rec) {
2     vis[u] = true;
3     rec[u] = true;
4     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, vector<bool>& vis, stack<int>& stk) {
2     vis[u] = true;
3     for (int v : adj[u]) {
4         if (!vis[v]) {
5             topo_sort(v, adj, vis, stk);
6         }
7     }
8     stk.push(u);
9 }
10
11 vector<vector<int>> transpose(int n, vector<vector<int>>& adj) {
12     vector<vector<int>> adj_t(n);
13     for (int u = 0; u < n; u++) {
14         for (int v : adj[u]) {
15             adj_t[v].push_back(u);
16         }
17     }
18     return adj_t;
19 }
20
21 void get_scc(int u, vector<vector<int>>& adj_t, vector<bool>& vis, vector<int>& scc) {
22     vis[u] = true;
23     scc.push_back(u);
24     for (int v : adj_t[u]) {
25         if (!vis[v]) {
26             get_scc(v, adj_t, vis, scc);
27         }
28     }
29 }
30
31 void kosaraju(int n, vector<vector<int>>& adj, vector<vector<int>>& sccs) {
32     vector<bool> vis(n, false);
33     stack<int> stk;
34     for (int u = 0; u < n; u++) {
35         if (!vis[u]) {

```

```

36     topo_sort(u, adj, vis, stk);
37 }
38 }
39 vector<vector<int>> adj_t = transpose(n, adj);
40 for (int u = 0; u < n; u++) {
41     vis[u] = false;
42 }
43 while (!stk.empty()) {
44     int u = stk.top();
45     stk.pop();
46     if (!vis[u]) {
47         vector<int> scc;
48         get_scc(u, adj_t, vis, scc);
49         sccs.push_back(scc);
50     }
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 << 1 | 1];
35            segtree[node] = (height[l] < height[r]) ? l : r;
36        }
37    }
38    ll query(ll node, ll b, ll e, ll L, ll R) {
39        if (b > R || e < L) return -1;
40        if (b >= L && e <= R) return segtree[node];
41        ll mid = (b + e) >> 1;
42        ll left = query(node << 1, b, mid, L, R);
43        ll right = query(node << 1 | 1, mid + 1, e, L, R);
44        if (left == -1) return right;
45        if (right == -1) return left;
46        return height[left] < height[right] ? left : right;

```

```

47 }
48 ll lca(ll u, ll v) {
49     ll left = first[u], right = first[v];
50     if (left > right) swap(left, right);
51     return query(1, 0, euler.size() - 1, left, right);
52 }
53 }

```

#### 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     {
4         for (ll v = 0; v < m; v++) {
5             if (bpGraph[u][v] && !seen[v]) {
6                 seen[v] = true;
7                 if (matchR[v] < 0 || bpm(n, m, bpGraph,
8                     matchR[v], seen, matchR)) {
9                     matchR[v] = u;
10                    return true;
11                }
12            }
13        }
14        return false;
15    }
16 ll maxBPM(ll n, ll m, vector<vector<bool>> &bpGraph)
17 {
18     vector<ll> matchR(m, -1);
19     ll result = 0;
20     for (ll u = 0; u < n; u++) {
21         vector<bool> seen(m, false);
22         if (bpm(n, m, bpGraph, u, seen, matchR)) {
23             result++;
24         }
25     }
26     return result;
27 }

```

#### 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[v0] = 0;
9     vector<bool> inq(n, false);
10    queue<int> q;
11    q.push(v0);
12    p.assign(n, -1);
13    while (!q.empty()) {
14        int u = q.front();
15        q.pop();
16        inq[u] = false;
17        for (int v : adj[u]) {
18            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
19                d[v] = d[u] + cost[u][v];
20                p[v] = u;
21                if (!inq[v]) {
22                    inq[v] = true;

```

```

23     q.push(v);
24 }
25 }
26 }
27 }
28 }
29 int min_cost_flow(int N, vector<Edge> edges, int K,
30     int s, int t) {
31     adj.assign(N, vector<int>());
32     cost.assign(N, vector<int>(N, 0));
33     capacity.assign(N, vector<int>(N, 0));
34     for (Edge e : edges) {
35         adj[e.from].push_back(e.to);
36         adj[e.to].push_back(e.from);
37         cost[e.from][e.to] = e.cost;
38         cost[e.to][e.from] = -e.cost;
39         capacity[e.from][e.to] = e.capacity;
40     }
41     int flow = 0;
42     int cost = 0;
43     vector<int> d, p;
44     while (flow < K) {
45         shortest_paths(N, s, d, p);
46         if (d[t] == INF) break;
47         int f = K - flow, cur = t;
48         while (cur != s) {
49             f = min(f, capacity[p[cur]][cur]);
50             cur = p[cur];
51         }
52         flow += f;
53         cost += f * d[t];
54         cur = t;
55         while (cur != s) {
56             capacity[p[cur]][cur] -= f;
57             capacity[cur][p[cur]] += f;
58             cur = p[cur];
59         }
60     }
61     if (flow < K) return -1;
62     else return cost;
63 }

```

#### 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  };
9  int n;
10 vector<vector<Edge>> adj;
11 void prim() {
12     int total_weight = 0;
13     vector<Edge> min_e(n);
14     min_e[0].w = 0;
15     set<Edge> q;
16     q.insert({0, 0});
17     vector<bool> selected(n, false);
18     for (int i = 0; i < n; ++i) {
19         if (q.empty()) {
20             cout << "No MST!" << endl;
21             exit(0);
22         }
23         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     }
36     cout << total_weight << endl;
37 }

```

#### 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     }
12     return res;
13 }

```

#### 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<
    int> const& b) {
26      vector<cd> fa(a.begin(), a.end()), fb(b.begin()
        , b.end());
27      int n = 1;
28      while (n < a.size() + b.size()) n <= 1;
29      fa.resize(n);
30      fb.resize(n);
31      fft(fa, false);
32      fft(fb, false);
33      for (int i = 0; i < n; i++) fa[i] *= fb[i];
34      fft(fa, true);
35      vector<int> result(n);
36      for (int i = 0; i < n; i++) result[i] = round(
        fa[i].real());
37      return result;
38  }

```

## 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:  $\gcd(f[m], f[n]) = f[\gcd(m, n)]$ 
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 -  $\text{sum}[i=1..n]f[i] = f[n+2] - 1$ 
15 -  $\text{sum}[i=0..n-1]f[2i+1] = f[2n]$ 
16 -  $\text{sum}[i=1..n]f[2i] = f[2n+1] - 1$ 
17 -  $\text{sum}[i=1..n]f[i]^2 = f[n]f[n+1]$ 
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 in
22 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:  $f[n] = ((1 + \sqrt{5})/2)^n - ((1 - \sqrt{5})/2)^n / \sqrt{5}$ 
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   $|f[n] \mid |a \ b \ c| |f[n-1]|$ 
6   $|f[n-1]| = |1 \ 0 \ 0| |f[n-2]|$ 
7   $|f[n-2]| = |0 \ 1 \ 0| |f[n-3]|$ 
8  To get:
9   $|f[n] \mid |a \ b \ c|^{(n-2)} |f[2]|$ 
10  $|f[n-1]| = |1 \ 0 \ 0| \mid f[1]|$ 
11  $|f[n-2]| = |0 \ 1 \ 0| \mid f[0]|$ 
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                 31, 37}) {
32         if (n == a) return true;
33         if (check_composite(n, a, d, r)) return false;
34     }
35     return true;
36 }

```

## 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;
21 }

```

```

20 }

```

## 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;
39 }

```

## 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 }

```

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 =
        bin_pow_mod(n, Q, p), R = bin_pow_mod(n, (Q
        + 1) / 2, p);
23     while (t % p != 1) {
24         if (t % p == 0) {
25             return 0;
26         }
27         ll i = 1, t2 = t * t % p;
28         for (; i < M; i++) {
29             if (t2 % p == 1) {
30                 break;
31             }
32             t2 = t2 * t2 % p;
33         }
34         ll b = bin_pow_mod(c, bin_pow_mod(2, M - i - 1,
        p), p);
35         M = i;
36         c = b * b % p;
37         t = t * c % p;
38         R = R * b % p;
39     }
40     return R;
41 }

```

## 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         ++row;
28     }
29     ans.assign(m, 0);
30     for (ll i = 0; i < m; ++i) {
31         if (where[i] != -1) {
32             ans[i] = a[where[i]][m] / a[where[i]][i];
33         }
34     }
35     for (ll i = 0; i < n; ++i) {
36         double sum = 0;
37         for (ll j = 0; j < m; ++j) {
38             sum += ans[j] * a[i][j];
39         }
40         if (abs(sum - a[i][m]) > EPS) {
41             return 0;
42         }
43     }
44     for (ll i = 0; i < m; ++i) {
45         if (where[i] == -1) {
46             return INF;
47         }
48     }
49     return 1;
50 }

```

### 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,
21     int cntr, int l, int k1, int k2) {
22     for (int ll = max(1, l - k2); ll <= min(l, k1);
23         ll++) {
24         if (left && ll == 1) break;
25         int l2 = l - ll;
26         int pos = shift + (left ? cntr - ll : cntr - 1
27             - ll + 1);
28         repetitions.emplace_back(pos, pos + 2 * l - 1);
29     }
30 }
31 void find_repetitions(string s, int shift = 0) {
32     int n = s.size();
33     if (n == 1) return;
34     int nu = n / 2;
35     int nv = n - nu;
36     string u = s.substr(0, nu);
37     string v = s.substr(nu);
38     string ru(u.rbegin(), u.rend());
39     string rv(v.rbegin(), v.rend());
40     find_repetitions(u, shift);
41     find_repetitions(v, shift + nu);
42     vector<int> z1 = z_function(ru);
43     vector<int> z2 = z_function(v + '#' + u);
44     vector<int> z3 = z_function(ru + '#' + rv);
45     vector<int> z4 = z_function(v);
46     for (int cntr = 0; cntr < n; cntr++) {
47         int l, k1, k2;
48         if (cntr < nu) {
49             l = nu - cntr;
50             k1 = get_z(z1, nu - cntr);
51             k2 = get_z(z2, nv + 1 + cntr);
52         } else {

```

```

49         l = cntr - nu + 1;
50         k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu))
51             ;
52         k2 = get_z(z4, (cntr - nu) + 1);
53         if (k1 + k2 >= 1) 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<
2     string> const& s) {
3     int n = s.size();
4     vector<pair<long long, int>> hashes(n);
5     for (int i = 0; i < n; i++) hashes[i] = {
6         compute_hash(s[i]), i};
7     sort(hashes.begin(), hashes.end());
8     vector<vector<int>> groups;
9     for (int i = 0; i < n; i++) {
10        if (i == 0 || hashes[i].first != hashes[i - 1].
11            first) groups.emplace_back();
12        groups.back().push_back(hashes[i].second);
13    }
14    return groups;
15 }

```

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

```

```

14     if (cur_h == h_s * p_pow[i] % m) occurences.
        push_back(i);
15 }
16 return occurences;
17 }

```

## 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)

```

```

19            pn[i] += n;
20        }
21        fill(cnt.begin(), cnt.begin() + classes, 0);
22        for (int i = 0; i < n; i++) cnt[c[pn[i]]]++;
23        for (int i = 1; i < classes; i++) cnt[i] += cnt[
24            i - 1];
25        for (int i = n-1; i >= 0; i--) p[--cnt[c[pn[i]
26            ]]] = pn[i];
27        cn[p[0]] = 0;
28        classes = 1;
29        for (int i = 1; i < n; i++) {
30            pair<int, int> cur = {c[p[i]], c[(p[i] + (1
31                << h)) % n]};
32            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] +
33                (1 << h)) % n]};
34            if (cur != prev) ++classes;
35            cn[p[i]] = classes - 1;
36        }
37        c.swap(cn);
38        return p;
39    }
40    vector<int> build_suff_arr(string s) {
41        s += "$";
42        vector<int> sorted_shifts = sort_cyclic_shifts(s)

```

```

43 // compare two substrings
44 int compare(int i, int j, int l, int k) {
45     pair<int, int> a = {c[k][i], c[k][(i + 1 - (1 <<
46         k)) % n]};
47     pair<int, int> b = {c[k][j], c[k][(j + 1 - (1 <<
48         k)) % n]};
49     return a == b ? 0 : a < b ? -1 : 1;

```

## 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;

```

$f(n) = O(g(n))$	iff $\exists$ positive $c, n_0$ such that $0 \leq f(n) \leq cg(n) \forall n \geq n_0$ .	$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}.$
$f(n) = \Omega(g(n))$	iff $\exists$ positive $c, n_0$ such that $f(n) \geq cg(n) \geq 0 \forall n \geq n_0$ .	In general:
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ .	$\sum_{i=1}^n i^m = \frac{1}{m+1} \left[ (n+1)^{m+1} - 1 - \sum_{i=1}^n ((i+1)^{m+1} - i^{m+1} - (m+1)i^m) \right]$
$f(n) = o(g(n))$	iff $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$ .	$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k}.$
$\lim_{n \rightarrow \infty} a_n = a$	iff $\forall \epsilon > 0, \exists n_0$ such that $ a_n - a  < \epsilon, \forall n \geq n_0$ .	Geometric series:
$\sup S$	least $b \in \mathbb{R}$ such that $b \geq s, \forall s \in S$ .	$\sum_{i=0}^n c^i = \frac{c^{n+1} - 1}{c - 1}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1 - c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1 - c}, \quad  c  < 1,$
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \leq s, \forall s \in S$ .	$\sum_{i=0}^n ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad  c  < 1.$
$\liminf_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \inf \{a_i \mid i \geq n, i \in \mathbb{N}\}.$	Harmonic series:
$\limsup_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \sup \{a_i \mid i \geq n, i \in \mathbb{N}\}.$	$H_n = \sum_{i=1}^n \frac{1}{i}, \quad \sum_{i=1}^n iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$
$\binom{n}{k}$	Combinations: Size $k$ sub-sets of a size $n$ set.	$\sum_{i=1}^n H_i = (n+1)H_n - n, \quad \sum_{i=1}^n \binom{i}{m} H_i = \binom{n+1}{m+1} \left( H_{n+1} - \frac{1}{m+1} \right).$
$[n]$	Stirling numbers (1st kind): Arrangements of an $n$ element set into $k$ cycles.	1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}, \quad 2. \sum_{k=0}^n \binom{n}{k} = 2^n, \quad 3. \binom{n}{k} = \binom{n}{n-k},$
$\left\{ \begin{matrix} n \\ k \end{matrix} \right\}$	Stirling numbers (2nd kind): Partitions of an $n$ element set into $k$ non-empty sets.	4. $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}, \quad 5. \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$
$\langle \begin{matrix} n \\ k \end{matrix} \rangle$	1st order Eulerian numbers: Permutations $\pi_1 \pi_2 \dots \pi_n$ on $\{1, 2, \dots, n\}$ with $k$ ascents.	6. $\binom{n}{m} \binom{m}{k} = \binom{n}{k} \binom{n-k}{m-k}, \quad 7. \sum_{k=0}^n \binom{r+k}{k} = \binom{r+n+1}{n},$
$\langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle$	2nd order Eulerian numbers.	8. $\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}, \quad 9. \sum_{k=0}^n \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n},$
$C_n$	Catalan Numbers: Binary trees with $n+1$ vertices.	10. $\binom{n}{k} = (-1)^k \binom{k-n-1}{k}, \quad 11. \left\{ \begin{matrix} n \\ 1 \end{matrix} \right\} = \left\{ \begin{matrix} n \\ n \end{matrix} \right\} = 1,$
14. $\left[ \begin{matrix} n \\ 1 \end{matrix} \right] = (n-1)!,$	15. $\left[ \begin{matrix} n \\ 2 \end{matrix} \right] = (n-1)!H_{n-1},$	16. $\left[ \begin{matrix} n \\ n \end{matrix} \right] = 1, \quad 17. \left[ \begin{matrix} n \\ k \end{matrix} \right] \geq \left\{ \begin{matrix} n \\ k \end{matrix} \right\},$
18. $\left[ \begin{matrix} n \\ k \end{matrix} \right] = (n-1) \left[ \begin{matrix} n-1 \\ k \end{matrix} \right] + \left[ \begin{matrix} n-1 \\ k-1 \end{matrix} \right],$	19. $\left\{ \begin{matrix} n \\ n-1 \end{matrix} \right\} = \left[ \begin{matrix} n \\ n-1 \end{matrix} \right] = \binom{n}{2},$	20. $\sum_{k=0}^n \left[ \begin{matrix} n \\ k \end{matrix} \right] = n!, \quad 21. C_n = \frac{1}{n+1} \binom{2n}{n},$
22. $\langle \begin{matrix} n \\ 0 \end{matrix} \rangle = \langle \begin{matrix} n \\ n-1 \end{matrix} \rangle = 1,$	23. $\langle \begin{matrix} n \\ k \end{matrix} \rangle = \langle \begin{matrix} n \\ n-1-k \end{matrix} \rangle,$	24. $\langle \begin{matrix} n \\ k \end{matrix} \rangle = (k+1) \langle \begin{matrix} n-1 \\ k \end{matrix} \rangle + (n-k) \langle \begin{matrix} n-1 \\ k-1 \end{matrix} \rangle,$
25. $\langle \begin{matrix} 0 \\ k \end{matrix} \rangle = \begin{cases} 1 & \text{if } k=0, \\ 0 & \text{otherwise} \end{cases}$	26. $\langle \begin{matrix} n \\ 1 \end{matrix} \rangle = 2^n - n - 1,$	27. $\langle \begin{matrix} n \\ 2 \end{matrix} \rangle = 3^n - (n+1)2^n + \binom{n+1}{2},$
28. $x^n = \sum_{k=0}^n \langle \begin{matrix} n \\ k \end{matrix} \rangle \binom{x+k}{n},$	29. $\langle \begin{matrix} n \\ m \end{matrix} \rangle = \sum_{k=0}^m \binom{n+1}{k} (m+1-k)^n (-1)^k,$	30. $m! \left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_{k=0}^n \langle \begin{matrix} n \\ k \end{matrix} \rangle \binom{k}{n-m},$
31. $\langle \begin{matrix} n \\ m \end{matrix} \rangle = \sum_{k=0}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} \binom{n-k}{m} (-1)^{n-k-m} k!,$	32. $\langle \langle \begin{matrix} n \\ 0 \end{matrix} \rangle \rangle = 1,$	33. $\langle \langle \begin{matrix} n \\ n \end{matrix} \rangle \rangle = 0 \quad \text{for } n \neq 0,$
34. $\langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle = (k+1) \langle \langle \begin{matrix} n-1 \\ k \end{matrix} \rangle \rangle + (2n-1-k) \langle \langle \begin{matrix} n-1 \\ k-1 \end{matrix} \rangle \rangle,$	35. $\sum_{k=0}^n \langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle = \frac{(2n)n}{2^n},$	
36. $\left\{ \begin{matrix} x \\ x-n \end{matrix} \right\} = \sum_{k=0}^n \langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle \binom{x+n-1-k}{2n},$	37. $\left\{ \begin{matrix} n+1 \\ m+1 \end{matrix} \right\} = \sum_k \binom{n}{k} \left\{ \begin{matrix} k \\ m \end{matrix} \right\} = \sum_{k=0}^n \left\{ \begin{matrix} k \\ m \end{matrix} \right\} (m+1)^{n-k},$	

The Chinese remainder theorem: There exists a number  $C$  such that:

$$C \equiv r_1 \pmod{m_1}$$

$$\vdots \quad \vdots \quad \vdots$$

$$C \equiv r_n \pmod{m_n}$$

if  $m_i$  and  $m_j$  are relatively prime for  $i \neq j$ .

Euler's function:  $\phi(x)$  is the number of positive integers less than  $x$  relatively prime to  $x$ . If  $\prod_{i=1}^n p_i^{e_i}$  is the prime factorization of  $x$  then

$$\phi(x) = \prod_{i=1}^n p_i^{e_i-1} (p_i - 1).$$

Euler's theorem: If  $a$  and  $b$  are relatively prime then

$$1 \equiv a^{\phi(b)} \pmod{b}.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \pmod{p}.$$

The Euclidean algorithm: if  $a > b$  are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If  $\prod_{i=1}^n p_i^{e_i}$  is the prime factorization of  $x$  then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers:  $x$  is an even perfect number iff  $x = 2^{n-1}(2^n - 1)$  and  $2^n - 1$  is prime.

Wilson's theorem:  $n$  is a prime iff

$$(n-1)! \equiv -1 \pmod{n}.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of } r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\left(\frac{a}{d}\right).$$

Prime numbers:

$$p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n}$$

$$+ O\left(\frac{n}{\ln n}\right),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3}$$

$$+ O\left(\frac{n}{(\ln n)^4}\right).$$

Definitions:

*Loop* An edge connecting a vertex to itself.

*Directed* Each edge has a direction.

*Simple* Graph with no loops or multi-edges.

*Walk* A sequence  $v_0 e_1 v_1 \dots e_\ell v_\ell$ .

*Trail* A walk with distinct edges.

*Path* A trail with distinct vertices.

*Connected* A graph where there exists a path between any two vertices.

*Component* A maximal connected subgraph.

*Tree* A connected acyclic graph.

*Free tree* A tree with no root.

*DAG* Directed acyclic graph.

*Eulerian* Graph with a trail visiting each edge exactly once.

*Hamiltonian* Graph with a cycle visiting each vertex exactly once.

*Cut* A set of edges whose removal increases the number of components.

*Cut-set* A minimal cut.

*Cut edge* A size 1 cut.

*k-Connected* A graph connected with the removal of any  $k-1$  vertices.

*k-Tough*  $\forall S \subseteq V, S \neq \emptyset$  we have  $k \cdot c(G-S) \leq |S|$ .

*k-Regular* A graph where all vertices have degree  $k$ .

*k-Factor* A  $k$ -regular spanning subgraph.

*Matching* A set of edges, no two of which are adjacent.

*Clique* A set of vertices, all of which are adjacent.

*Ind. set* A set of vertices, none of which are adjacent.

*Vertex cover* A set of vertices which cover all edges.

*Planar graph* A graph which can be embedded in the plane.

*Plane graph* An embedding of a planar graph.

$$\sum_{v \in V} \deg(v) = 2m.$$

If  $G$  is planar then  $n - m + f = 2$ , so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree  $\leq 5$ .

Notation:

$E(G)$  Edge set

$V(G)$  Vertex set

$c(G)$  Number of components

$G[S]$  Induced subgraph

$\deg(v)$  Degree of  $v$

$\Delta(G)$  Maximum degree

$\delta(G)$  Minimum degree

$\chi(G)$  Chromatic number

$\chi_E(G)$  Edge chromatic number

$G^c$  Complement graph

$K_n$  Complete graph

$K_{n_1, n_2}$  Complete bipartite graph

$r(k, \ell)$  Ramsey number

Geometry

Projective coordinates: triples  $(x, y, z)$ , not all  $x, y$  and  $z$  zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

Cartesian Projective

$$(x, y) \quad (x, y, 1)$$

$$y = mx + b \quad (m, -1, b)$$

$$x = c \quad (1, 0, -c)$$

Distance formula,  $L_p$  and  $L_\infty$  metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

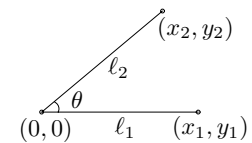
$$[|x_1 - x_0|^p + |y_1 - y_0|^p]^{1/p},$$

$$\lim_{p \rightarrow \infty} [|x_1 - x_0|^p + |y_1 - y_0|^p]^{1/p}.$$

Area of triangle  $(x_0, y_0), (x_1, y_1)$  and  $(x_2, y_2)$ :

$$\frac{1}{2} \text{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{l_1 l_2}.$$

Line through two points  $(x_0, y_0)$  and  $(x_1, y_1)$ :

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \quad V = \frac{4}{3} \pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.

– Issac Newton

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \dots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\begin{aligned} \frac{1}{1-x} &= 1 + x + x^2 + x^3 + x^4 + \dots = \sum_{i=0}^{\infty} x^i, \\ \frac{1}{1-cx} &= 1 + cx + c^2x^2 + c^3x^3 + \dots = \sum_{i=0}^{\infty} c^i x^i, \\ \frac{1}{1-x^n} &= 1 + x^n + x^{2n} + x^{3n} + \dots = \sum_{i=0}^{\infty} x^{ni}, \\ \frac{x}{(1-x)^2} &= x + 2x^2 + 3x^3 + 4x^4 + \dots = \sum_{i=0}^{\infty} ix^i, \\ x^k \frac{d^n}{dx^n} \left( \frac{1}{1-x} \right) &= x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \dots = \sum_{i=0}^{\infty} i^n x^i, \\ e^x &= 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \dots = \sum_{i=0}^{\infty} \frac{x^i}{i!}, \\ \ln(1+x) &= x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \dots = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i}, \\ \ln \frac{1}{1-x} &= x + \frac{1}{2}x^2 + \frac{1}{3}x^3 + \frac{1}{4}x^4 + \dots = \sum_{i=1}^{\infty} \frac{x^i}{i}, \\ \sin x &= x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!}, \\ \cos x &= 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}, \\ \tan^{-1} x &= x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)}, \\ (1+x)^n &= 1 + nx + \frac{n(n-1)}{2}x^2 + \dots = \sum_{i=0}^{\infty} \binom{n}{i} x^i, \\ \frac{1}{(1-x)^{n+1}} &= 1 + (n+1)x + \binom{n+2}{2}x^2 + \dots = \sum_{i=0}^{\infty} \binom{i+n}{i} x^i, \\ \frac{x}{e^x - 1} &= 1 - \frac{1}{2}x + \frac{1}{12}x^2 - \frac{1}{720}x^4 + \dots = \sum_{i=0}^{\infty} \frac{B_i x^i}{i!}, \\ \frac{1}{2x}(1 - \sqrt{1-4x}) &= 1 + x + 2x^2 + 5x^3 + \dots = \sum_{i=0}^{\infty} \frac{1}{i+1} \binom{2i}{i} x^i, \\ \frac{1}{\sqrt{1-4x}} &= 1 + x + 2x^2 + 6x^3 + \dots = \sum_{i=0}^{\infty} \binom{2i}{i} x^i, \\ \frac{1}{\sqrt{1-4x}} \left( \frac{1 - \sqrt{1-4x}}{2x} \right)^n &= 1 + (2+n)x + \binom{4+n}{2}x^2 + \dots = \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i, \\ \frac{1}{1-x} \ln \frac{1}{1-x} &= x + \frac{3}{2}x^2 + \frac{11}{6}x^3 + \frac{25}{12}x^4 + \dots = \sum_{i=1}^{\infty} H_i x^i, \\ \frac{1}{2} \left( \ln \frac{1}{1-x} \right)^2 &= \frac{1}{2}x^2 + \frac{3}{4}x^3 + \frac{11}{24}x^4 + \dots = \sum_{i=2}^{\infty} \frac{H_{i-1} x^i}{i}, \\ \frac{x}{1-x-x^2} &= x + x^2 + 2x^3 + 3x^4 + \dots = \sum_{i=0}^{\infty} F_i x^i, \\ \frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2} &= F_n x + F_{2n} x^2 + F_{3n} x^3 + \dots = \sum_{i=0}^{\infty} F_{ni} x^i. \end{aligned}$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y) \sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\alpha A(x) + \beta B(x) = \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,$$

$$x^k A(x) = \sum_{i=k}^{\infty} a_{i-k} x^i,$$

$$\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} = \sum_{i=0}^{\infty} a_{i+k} x^i,$$

$$A(cx) = \sum_{i=0}^{\infty} c^i a_i x^i,$$

$$A'(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,$$

$$xA'(x) = \sum_{i=1}^{\infty} i a_i x^i,$$

$$\int A(x) dx = \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,$$

$$\frac{A(x) + A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i} x^{2i},$$

$$\frac{A(x) - A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.$$

Summation: If  $b_i = \sum_{j=0}^i a_j$  then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty} \left( \sum_{j=0}^i a_j b_{i-j} \right) x^i.$$

God made the natural numbers;  
all the rest is the work of man.  
– Leopold Kronecker