UPLB Eliens - Pegaraw Notebook

Contents

1 Data Structures

1.1 Binary Trie

```cpp
struct Node { struct Node* parent, child[2]; };
struct BinaryTrie {
  Node* root;
  BinaryTrie() {
    root = new Node();
    root->parent = NULL;
    root->child[0] = NULL;
    root->child[1] = NULL;
  }
  void insert_node(int x) {
    Node* cur = root;
    for (int place = 29; place >= 0; place--) {
      int bit = x >> place & 1;
      if (cur->child[bit] != NULL) cur = cur->child
          [bit];
      else {
        cur->child[bit] = new Node();
        cur->child[bit]->parent = cur;
        cur = cur->child[bit];
        cur->child[0] = NULL;
        cur->child[1] = NULL;
      }
    }
  }
  void remove_node(int x) {
    Node* cur = root;
    for (int place = 29; place >= 0; place--) {
      int bit = x >> place & 1;
      if (cur->child[bit] == NULL) return;
      cur = cur->child[bit];
    }
    while (cur->parent != NULL && cur->child[0] ==
        NULL && cur->child[1] == NULL) {
      Node* temp = cur;
      cur = cur->parent;
      if (temp == cur->child[0]) cur->child[0] =
          NULL;
      else cur->child[1] = NULL;
      delete temp;
    }
  }
  int get_min_xor(int x) {
    Node* cur = root;
    int minXor = 0;
    for (int place = 29; place >= 0; place--) {
      int bit = x >> place & 1;
      if (cur->child[bit] != NULL) cur = cur->child
          [bit];
      else {
        minXor ^= 1 << place;
        cur = cur->child[1 ^ bit];
      }
    }
    return minXor;
  }
};
```

1.2 Disjoint Set Union

```cpp
struct DSU {
  vector<int> parent, size;
  DSU(int n) {
    parent.resize(n);
    size.resize(n);
    for (int i = 0; i < n; i++) make_set(i);
  }
  void make_set(int v) {
    parent[v] = v;
    size[v] = 1;
  }
  bool is_same(int a, int b) { return find_set(a)
      == find_set(b); }
  int find_set(int v) { return v == parent[v] ? v :
      parent[v] = find_set(parent[v]); }
  void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
      if (size[a] < size[b]) swap(a, b);
      parent[b] = a;
      size[a] += size[b];
    }
  }
};
```

1.3 Minimum Queue

```cpp
ll get_minimum(stack<pair<ll, ll>> &s1, stack<pair<
    ll, ll>> &s2) {
  if (s1.empty() || s2.empty()) {
```

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

1.4   Mo

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

```cpp
33       }
34       answers[q.idx] = get_answer();
35    }
36    return answers;
37 }
```

1.5   Range Add Point Query

```cpp
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 };
```

1.6   Range Add Range Query

```cpp
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  };
```

1.7 Segment Tree

```
1  template<typename T, typename InType = T>
2  class SegTreeNode {
3  public:
4    const T IDN = 0, DEF = 0;
5    int i, j;
6    T val;
7    SegTreeNode<T, InType>* lc, * rc;
8    SegTreeNode(int i, int j) : i(i), j(j) {
9      if (j - i == 1) {
10       lc = rc = nullptr;
11       val = DEF;
12       return;
13     }
14     int k = (i + j) / 2;
15     lc = new SegTreeNode<T, InType>(i, k);
16     rc = new SegTreeNode<T, InType>(k, j);
17     val = op(lc->val, rc->val);
18   }
19   SegTreeNode(const vector<InType>& a, int i, int j
         ) : 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); }
```

1.8 Segment Tree 2d

```
1  template<typename T, typename InType = T>
2  class SegTree2dNode {
3  public:
4    int i, j, tree_size;
5    SegTree<T, InType>* seg_tree;
6    SegTree2dNode<T, InType>* lc, * rc;
7    SegTree2dNode() {}
8    SegTree2dNode(const vector<vector<InType>>& a,
         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     if (lx <= i && j <= rx) return seg_tree->
           range_query(ly, ry);
37     if (j <= lx || rx <= i) return -INF;
38     return max(lc->range_query_2d(lx, rx, ly, ry),
             rc->range_query_2d(lx, rx, ly, ry));
39   }
40   void operation_2d(SegTree<T, InType>* x, SegTree<
         T, InType>* y) {
41     for (int k = 0; k < tree_size; k++) {
42       seg_tree->set(k, max(x->range_query(k, k + 1)
             , y->range_query(k, k + 1)));
43     }
44   }
45 };
46 template<typename T, typename InType = T>
47 class SegTree2d {
48 public:
49   SegTree2dNode<T, InType> root;
50   SegTree2d() {}
51   SegTree2d(const vector<vector<InType>>& mat) :
         root(mat, 0, mat.size()) {}
52   void set_2d(int kx, int ky, T x) { root.set_2d(kx
         , ky, x); }
```

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

1.9 Sparse Table

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

1.10 Sparse Table 2d

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

```
14        for (int j = 0; j + (1 << k) - 1 < m; j++)
15          table[i][j][0][k] = min(table[i][j][0][k -
                1], table[i][j + (1 << (k - 1))][0][k
                - 1]);
16    for (int k = 1; k <= (int)(log2(n)); k++)
17      for (int l = 1; l <= (int)(log2(m)); l++)
18        for (int i = 0; i + (1 << k) - 1 < n; i++)
19          for (int j = 0; j + (1 << l) - 1 < m; j++)
20            table[i][j][k][l] = min(
21              min(table[i][j][k - 1][l - 1], table[i
                  + (1 << (k - 1))][j][k - 1][l -
                  1]),
22              min(table[i][j + (1 << (l - 1))][k -
                  1][l - 1], table[i + (1 << (k - 1)
                  )][j + (1 << (l - 1))][k - 1][l -
                  1])
23            );
24  }
25  int rmq(int x1, int y1, int x2, int y2) {
26    int k = log2(x2 - x1 + 1), l = log2(y2 - y1 + 1);
27    return max(
28      max(table[x1][y1][k][l], table[x2 - (1 << k) +
          1][y1][k][l]),
29      max(table[x1][y2 - (1 << l) + 1][k][l], table[
          x2 - (1 << k) + 1][y2 - (1 << l) + 1][k][l
          ])
30    );
31  }
```

1.11 Sqrt Decomposition

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

2 Dynamic Programming

2.1 Divide And Conquer

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

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,
          INF));
3    dp[0][0] = 0;
4    for (int i = 1; i <= n; i++) {
5      dp[i][0] = i;
6    }
7    for (int j = 1; j <= m; j++) {
8      dp[0][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 -
            1] + 1, dp[i - 1][j - 1] + (x[i - 1] !=
            y[j - 1])});
13     }
14   }
15   return dp[n][m];
16  }
```

2.3 Knapsack

```
1  ll knapsack(ll W, vector<ll> &wt, vector<ll> &val,
        ll n) {
2    vector<ll> dp(W + 1, 0);
3    for (ll i = 1; i <= n; i++) {
4      for (ll w = W; w >= 0; w--) {
5        if (wt[i - 1] <= w) {
6          dp[w] = max(dp[w], dp[w - wt[i - 1]] + val[
              i - 1]);
7        }
8      }
9    }
10   return dp[W];
11  }
```

2.4 Knuth Optimization

```
1  ll solve() {
2    ll N;
3    ... // Read input
4    vector<vector<ll>> dp(N, vector<ll>(N)), opt(N,
          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,
              opt[i + 1][j]); k++) {
16          if (mn >= dp[i][k] + dp[k + 1][j] + cost) {
17            opt[i][j] = k;
18            mn = dp[i][k] + dp[k + 1][j] + cost;
19          }
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,
       ll r, ll x) {
2      while (r - l > 1) {
3          ll m = l + (r - l) / 2;
4          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,
           false));
3      dp[0][0] = true;
4      for (ll i = 1; i <= n; i++) {
5          for (ll j = 0; j <= sum; j++) {
6              dp[i][j] = dp[i - 1][j];
7              if (j >= arr[i]) {
8                  dp[i][j] |= dp[i - 1][j - arr[i]];
9              }
10         }
11     }
12     return dp[n][sum];
13 }
```

3 Geometry

3.1 Areas

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

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; }
26     point2d operator-(const point2d &t) const {
           return point2d(*this) -= t; }
27     point2d operator*(ftype t) const { return point2d
           (*this) *= t; }
28     point2d operator/(ftype t) const { return point2d
           (*this) /= t; }
29 };
30 point2d operator*(ftype a, point2d b) { return b *
       a; }
31 ftype dot(point2d a, point2d b) { return a.x * b.x
       + a.y * b.y; }
32 ftype dot(point3d a, point3d b) { return a.x * b.x
       + a.y * b.y + a.z * b.z; }
33 ftype norm(point2d a) { return dot(a, a); }
34 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()))
           i--;
35     reverse(a.begin()+i+1, a.end());
36   }
37   vector<pt> st;
38   for (ll i = 0; i < (ll) a.size(); i++) {
39     while (st.size() > 1 && !cw(st[st.size() - 2],
           st.back(), a[i], include_collinear)) {
40       st.pop_back();
41     }
42     st.push_back(a[i]);
43   }
44   a = st;
45 }
```

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 intersect1d(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 intersect1d(a.p.x, a.q.x, b.p.x, b.q.x) &&
22     intersect1d(a.p.y, a.q.y, b.p.y, b.q.y) &&
23     vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <=
           0 &&
24     vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <=
           0;
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   bool operator<(const event& e) const {
36     if (abs(x - e.x) > EPS) return x < e.x;
37     return tp > e.tp;
38   }
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,
           i));
53     e.push_back(event(max(a[i].p.x, a[i].q.x), -1,
           i));
54   }
55   sort(e.begin(), e.end());
56   s.clear();
57   where.resize(a.size());
58   for (size_t i = 0; i < e.size(); ++i) {
59     ll id = e[i].id;
60     if (e[i].tp == +1) {
61       set<seg>::iterator nxt = s.lower_bound(a[id])
             , prv = prev(nxt);
62       if (nxt != s.end() && intersect(*nxt, a[id]))
             return make_pair(nxt->id, id);
63       if (prv != s.end() && intersect(*prv, a[id]))
             return make_pair(prv->id, id);
64       where[id] = s.insert(nxt, a[id]);
65     } else {
66       set<seg>::iterator nxt = next(where[id]), prv
             = prev(where[id]);
67       if (nxt != s.end() && prv != s.end() &&
             intersect(*nxt, *prv)) return make_pair(
             prv->id, nxt->id);
68       s.erase(where[id]);
69     }
70   }
71   return make_pair(-1, -1);
72 }
```

3.8 Minkowski Sum

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

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

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

```
30      return;
31    }
32    ll m = (l + r) >> 1, midx = a[m].x;
33    rec(l, m);
34    rec(m, r);
35    merge(a.begin() + l, a.begin() + m, a.begin() + m
          , a.begin() + r, t.begin(), cmp_y());
36    copy(t.begin(), t.begin() + r - l, a.begin() + l)
          ;
37    ll tsz = 0;
38    for (ll i = l; i < r; ++i) {
39      if (abs(a[i].x - midx) < mindist) {
40        for (ll j = tsz - 1; j >= 0 && a[i].y - t[j].
              y < mindist; --j)
41          upd_ans(a[i], t[j]);
42        t[tsz++] = a[i];
43      }
44    }
45  }
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
        .x, y + p.y); }
 6    pt operator-(const pt &p) const { return pt(x - p
        .x, y - p.y); }
 7    long long cross(const pt &p) const { return x * p
        .y - y * p.x; }
 8    long long dot(const pt &p) const { return x * p.x
        + y * p.y; }
 9    long long cross(const pt &a, const pt &b) const {
          return (a - *this).cross(b - *this); }
10    long long dot(const pt &a, const pt &b) const {
          return (a - *this).dot(b - *this); }
11    long long sqrLen() const { return this->dot(*this
        ); }
12  };
13  bool lexComp(const pt &l, const pt &r) { return l.x
        < r.x || (l.x == r.x && l.y < r.y); }
14  int sgn(long long val) { return val > 0 ? 1 : (val
        == 0 ? 0 : -1); }
15  vector<pt> seq;
16  pt translation;
17  int n;
18  bool pointInTriangle(pt a, pt b, pt c, pt point) {
19    long long s1 = abs(a.cross(b, c));
20    long long s2 = abs(point.cross(a, b)) + abs(point
        .cross(b, c)) + abs(point.cross(c, a));
21    return s1 == s2;
22  }
23  void prepare(vector<pt> &points) {
24    n = points.size();
25    int pos = 0;
26    for (int i = 1; i < n; i++) {
27      if (lexComp(points[i], points[pos])) pos = i;
28    }
29    rotate(points.begin(), points.begin() + pos,
          points.end());
30    n--;
```

```
31    seq.resize(n);
32    for (int i = 0; i < n; i++) seq[i] = points[i +
          1] - points[0];
33    translation = points[0];
34  }
35  bool pointInConvexPolygon(pt point) {
36    point = point - translation;
37    if (seq[0].cross(point) != 0 && sgn(seq[0].cross(
          point)) != sgn(seq[0].cross(seq[n - 1])))
38      return false;
39    if (seq[n - 1].cross(point) != 0 && sgn(seq[n -
          1].cross(point)) != sgn(seq[n - 1].cross(seq
          [0])))
40      return false;
41    if (seq[0].cross(point) == 0)
42      return seq[0].sqrLen() >= point.sqrLen();
43    int l = 0, r = n - 1;
44    while (r - l > 1) {
45      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 &&
            y < p.y - EPS);
 6    }
 7  };
 8  struct line {
 9    double a, b, c;
10    line() {}
11    line(pt p, pt q) {
12      a = p.y - q.y;
13      b = q.x - p.x;
14      c = -a * p.x - b * p.y;
15      norm();
16    }
17    void norm() {
18      double z = sqrt(a * a + b * b);
19      if (abs(z) > EPS) a /= z, b /= z, c /= z;
20    }
21    double dist(pt p) const { return a * p.x + b * p.
          y + c; }
22  };
23  double det(double a, double b, double c, double d)
          {
24    return a * d - b * c;
25  }
26  inline bool betw(double l, double r, double x) {
27    return min(l, r) <= x + EPS && x <= max(l, r) +
          EPS;
28  }
29  inline bool intersect_1d(double a, double b, double
          c, double d) {
30    if (a > b) swap(a, b);
31    if (c > d) swap(c, d);
32    return max(a, c) <= min(b, d) + EPS;
```

```
33   }
34   bool intersect(pt a, pt b, pt c, pt d, pt& left, pt
         & right) {
35     if (!intersect_1d(a.x, b.x, c.x, d.x) || !
             intersect_1d(a.y, b.y, c.y, d.y)) return
             false;
36     line m(a, b);
37     line n(c, d);
38     double zn = det(m.a, m.b, n.a, n.b);
39     if (abs(zn) < EPS) {
40       if (abs(m.dist(c)) > EPS || abs(n.dist(a)) >
             EPS) return false;
41       if (b < a) swap(a, b);
42       if (d < c) swap(c, d);
43       left = max(a, c);
44       right = min(b, d);
45       return true;
46     } else {
47       left.x = right.x = -det(m.c, m.b, n.c, n.b) /
             zn;
48       left.y = right.y = -det(m.a, m.c, n.a, n.c) /
             zn;
49       return betw(a.x, b.x, left.x) && betw(a.y, b.y,
             left.y) &&
50         betw(c.x, d.x, left.x) && betw(c.y, d.y,
             left.y);
51     }
52   }
```

4   Graph Theory

4.1   Articulation Point

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

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

4.2   Bellman Ford

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

4.3   Bridge

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

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

4.4   Centroid Decomposition

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

4.5   Dijkstra

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

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

4.6   Dinics

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

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

4.7   Edmonds Karp

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

4.8   Fast Second Mst

```
1   struct edge {
2     int s, e, w, id;
3     bool operator<(const struct edge& other) {
4         return w < other.w; }
4   };
5   typedef struct edge Edge;
6   const int N = 2e5 + 5;
7   long long res = 0, ans = 1e18;
8   int n, m, a, b, w, id, l = 21;
9   vector<Edge> edges;
10  vector<int> h(N, 0), parent(N, -1), size(N, 0),
        present(N, 0);
11  vector<vector<pair<int, int>>> adj(N), dp(N, vector
        <pair<int, int>>(l));
12  vector<vector<int>> up(N, vector<int>(l, -1));
13  pair<int, int> combine(pair<int, int> a, pair<int,
        int> b) {
14    vector<int> v = {a.first, a.second, b.first, b.
          second};
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 = l - 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 = l - 1; i >= 0; i--) {
43      if (up[u][i] != -1 && up[v][i] != -1 && up[u][i
            ] != up[v][i]) {
44        ans = combine(ans, combine(dp[u][i], dp[v][i
              ]));
45        u = up[u][i];
46        v = up[v][i];
47      }
48    }
49    ans = combine(ans, combine(dp[u][0], dp[v][0]));
50    return ans;
51  }
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 <= l - 1; i++) {
78      for (int j = 1; j <= n; ++j) {
79        if (up[j][i - 1] != -1) {
80          int v = up[j][i - 1];
81          up[j][i] = up[v][i - 1];
82          dp[j][i] = combine(dp[j][i - 1], dp[v][i -
                1]);
83        }
84      }
85    }
86    for (int i = 0; i <= m - 1; i++) {
87      id = edges[i].id;
88      w = edges[i].w;
89      if (!present[id]) {
90        auto rem = lca(edges[i].s, edges[i].e);
91        if (rem.first != w) {
92          if (ans > res + w - rem.first) ans = res +
                w - rem.first;
93        } else if (rem.second != -1) {
94          if (ans > res + w - rem.second) ans = res +
                w - rem.second;
95        }
96      }
97    }
98    cout << ans << "\n";
99    return 0;
100 }
```

4.9 Find Cycle

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

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

4.10 Floyd Warshall

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

4.11 Ford Fulkerson

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

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

4.12 Hierholzer

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

4.13 Hungarian

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

```
5    vector<int> minv (m+1, INF);
6    vector<bool> used (m+1, false);
7    do {
8      used[j0] = true;
9      int i0 = p[j0],  delta = INF,  j1;
10     for (int j=1; j<=m; ++j)
11       if (!used[j]) {
12         int cur = A[i0][j]-u[i0]-v[j];
13         if (cur < minv[j]) minv[j] = cur,  way[j] =
                 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)
           ) return true;
6      else if (rec[v]) return true;
7    }
8    rec[u] = false;
9    return false;
10 }
11 bool is_cyclic(int n, vector<vector<int>> &adj) {
12   vector<bool> vis(n, false), rec(n, false);
13   for (int i = 0; i < n; i++)
14     if (!vis[i] && is_cyclic_util(i, adj, vis, rec)
           ) return true;
15   return false;
16 }
```

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

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

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

4.24   Topological Sort

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

4.25   Zero One Bfs

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

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

5   Math

5.1   Chinese Remainder Theorem

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

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 %
             p;
5      int res = 1;
6      while (n > 1) {
7        if ((n / p) % 2) res = p - res;
8        res = res * f[n % p] % p;
9        n /= p;
10     }
```

```
11     return res;
12   }
```

5.4   Fast Fourier Transform

```
1    using cd = complex<double>;
2    const double PI = acos(-1);
3    void fft(vector<cd>& a, bool invert) {
4      int n = a.size();
5      if (n == 1) return;
6      vector<cd> a0(n / 2), a1(n / 2);
7      for (int i = 0; 2 * i < n; i++) {
8        a0[i] = a[2 * i];
9        a1[i] = a[2 * i + 1];
10     }
11     fft(a0, invert);
12     fft(a1, invert);
13     double ang = 2 * PI / n * (invert ? -1 : 1);
14     cd w(1), wn(cos(ang), sin(ang));
15     for (int i = 0; 2 * i < n; i++) {
16       a[i] = a0[i] + w * a1[i];
17       a[i + n / 2] = a0[i] - w * a1[i];
18       if (invert) {
19         a[i] /= 2;
20         a[i + n / 2] /= 2;
21       }
22       w *= wn;
23     }
24   }
25   vector<int> multiply(vector<int> const& a, vector<
         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] | f[nk]
8    - f[n] | f[m] => n | 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])
```

```
- Periodic sequence modulo p
- sum[i=1..n]f[i] = f[n+2] - 1
- sum[i=0..n-1]f[2i+1] = f[2n]
- sum[i=1..n]f[2i] = f[2n+1] - 1
- sum[i=1..n]f[i]^2 = f[n]f[n+1]
Fibonacci encoding:
1. Iterate through the Fibonacci numbers from the
    largest to the
smallest until you find one less than or equal to n
    .
2. Suppose this number was F_i. Subtract F_i from
    n Ł and put a 1 Ł
in the i-2 position of the code word (indexing from
    0 from the
leftmost to the rightmost bit).
3. Repeat until there is no remainder.
4. Add a final 1 Ł to the codeword to indicate its
    end.
Closed-form: f[n] = (((1 + rt(5))/2)^n - ((1 - rt
    (5)) / 2)^n)/rt(5)
*/
struct matrix {
  ll mat[2][2];
  matrix friend operator *(const matrix &a, const
        matrix &b){
    matrix c;
    for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
        c.mat[i][j] = 0;
        for (int k = 0; k < 2; k++) c.mat[i][j] +=
            a.mat[i][k] * b.mat[k][j];
      }
    }
    return c;
  }
};
matrix matpow(matrix base, ll n) {
  matrix ans{ {
    {1, 0},
    {0, 1}
  } };
  while (n) {
    if (n & 1) ans = ans * base;
    base = base * base;
    n >>= 1;
  }
  return ans;
}
ll fib(int n) {
  matrix base{ {
    {1, 1},
    {1, 0}
  } };
  return matpow(base, n).mat[0][1];
}
pair<int, int> fib (int n) {
  if (n == 0) return {0, 1};
  auto p = fib(n >> 1);
  int c = p.first * (2 * p.second - p.first);
  int d = p.first * p.first + p.second * p.second;
  if (n & 1) return {d, c + d};
  else return {c, d};
}
```

5.6   Find All Solutions

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

```
    &y0, ll &g) {
  g = gcd_extended(abs(a), abs(b), x0, y0);
  if (c % g) return false;
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
void shift_solution(ll & x, ll & y, ll a, ll b, ll
    cnt) {
  x += cnt * b;
  y -= cnt * a;
}
ll find_all_solutions(ll a, ll b, ll c, ll minx, ll
    maxx, ll miny, ll maxy) {
  ll x, y, g;
  if (!find_any_solution(a, b, c, x, y, g)) return
      0;
  a /= g;
  b /= g;
  ll sign_a = a > 0 ? +1 : -1;
  ll sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x) / b);
  if (x < minx) shift_solution(x, y, a, b, sign_b);
  if (x > maxx) return 0;
  ll lx1 = x;
  shift_solution(x, y, a, b, (maxx - x) / b);
  if (x > maxx) shift_solution(x, y, a, b, -sign_b)
      ;
  ll rx1 = x;
  shift_solution(x, y, a, b, -(miny - y) / a);
  if (y < miny) shift_solution(x, y, a, b, -sign_a)
      ;
  if (y > maxy) return 0;
  ll lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y) / a);
  if (y > maxy) shift_solution(x, y, a, b, sign_a);
  ll rx2 = x;
  if (lx2 > rx2) swap(lx2, rx2);
  ll lx = max(lx1, lx2), rx = min(rx1, rx2);
  if (lx > rx) return 0;
  return (rx - lx) / abs(b) + 1;
}
```

5.7   Linear Sieve

```
void linear_sieve(ll N, vector<ll> &lowest_prime,
    vector<ll> &prime) {
  for (ll i = 2; i <= N; i++) {
    if (lowest_prime[i] == 0) {
      lowest_prime[i] = i;
      prime.push_back(i);
    }
    for (ll j = 0; i * prime[j] <= N; j++) {
      lowest_prime[i * prime[j]] = prime[j];
      if (prime[j] == lowest_prime[i]) break;
    }
  }
}
```

5.8   Matrix

```
/*
Matrix exponentation:
```

```
f[n] = af[n-1] + bf[n-2] + cf[n-3]
Use:
|f[n]   | |a b c||f[n-1]|
|f[n-1]|=|1 0 0||f[n-2]|
|f[n-2]| |0 1 0||f[n-3]|
To get:
|f[n]   | |a b c|^(n-2)|f[2]|
|f[n-1]|=|1 0 0|        |f[1]|
|f[n-2]| |0 1 0|        |f[0]|
*/
struct Matrix { int mat[MAX_N][MAX_N]; };
Matrix matrix_mul(Matrix a, Matrix b) {
  Matrix ans; int i, j, k;
  for (i = 0; i < MAX_N; i++)
    for (j = 0; j < MAX_N; j++)
      for (ans.mat[i][j] = k = 0; k < MAX_N; k++)
        ans.mat[i][j] += a.mat[i][k] * b.mat[k][j];
  return ans;
}
Matrix matrix_pow(Matrix base, int p) {
  Matrix ans; int i, j;
  for (i = 0; i < MAX_N; i++)
    for (j = 0; j < MAX_N; j++)
      ans.mat[i][j] = (i == j);
  while (p) {
    if (p & 1) ans = matrix_mul(ans, base);
    base = matrix_mul(base, base);
    p >>= 1;
  }
  return ans;
}
```

5.9   Miller Rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpower(u64 base, u64 e, u64 mod) {
  u64 result = 1;
  base %= mod;
  while (e) {
    if (e & 1) result = (u128) result * base % mod;
    base = (u128) base * base % mod;
    e >>= 1;
  }
  return result;
}
bool check_composite(u64 n, u64 a, u64 d, ll s) {
  u64 x = binpower(a, d, n);
  if (x == 1 || x == n - 1) return false;
  for (ll r = 1; r < s; r++) {
    x = (u128) x * x % n;
    if (x == n - 1) return false;
  }
  return true;
}
bool miller_rabin(u64 n) {
  if (n < 2) return false;
  ll r = 0;
  u64 d = n - 1;
  while ((d & 1) == 0) {
    d >>= 1;
    r++;
  }
  for (ll a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
      31, 37}) {
    if (n == a) return true;
    if (check_composite(n, a, d, r)) return false;
```

```
33        }
34        return true;
35   }
```

5.10   Modulo Inverse

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

5.11   Pollard Rho Brent

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

5.12   Range Sieve

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

```
6        prime.push_back(2);
7        for (ll i = 4; i <= n; i += 2) is_prime[i] =
             false;
8        for (ll i = 3; i <= n; i += 2) {
9            if (is_prime[i]) {
10               prime.push_back(i);
11               for (ll j = i * i; j <= n; j += i) is_prime[j
                    ] = false;
12           }
13       }
14       vector<bool> result(r - l + 1, true);
15       for (ll i : prime)
16           for (ll j = max(i * i, (l + i - 1) / i * i); j
                <= r; j += i)
17               result[j - l] = false;
18       if (l == 1) result[0] = false;
19       return result;
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 -
                    k * S; j < S; j += p) {
25                   block[j] = false;
26               }
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       return result;
38   }
```

5.14   Sum Of Divisors

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

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

5.15   Tonelli Shanks

```
1    ll legendre(ll a, ll p) {
2        return bin_pow_mod(a, (p - 1) / 2, p);
3    }
4    ll tonelli_shanks(ll n, ll p) {
5        if (legendre(n, p) == p - 1) {
6            return -1;
7        }
8        if (p % 4 == 3) {
9            return bin_pow_mod(n, (p + 1) / 4, p);
10       }
11       ll Q = p - 1, S = 0;
12       while (Q % 2 == 0) {
13           Q /= 2;
14           S++;
15       }
16       ll z = 2;
17       for (; z < p; z++) {
18           if (legendre(z, p) == p - 1) {
19               break;
20           }
21       }
22       ll M = S, c = bin_pow_mod(z, Q, p), t =
             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

```cpp
const double EPS = 1e-9;
const ll INF = 2;
ll gauss(vector <vector<double>> a, vector<double>
    &ans) {
  ll n = (ll) a.size(), m = (ll) a[0].size() - 1;
  vector<ll> where (m, -1);
  for (ll col = 0, row = 0; col < m && row < n; ++
      col) {
    ll sel = row;
    for (ll i = row; i < n; ++i) {
      if (abs(a[i][col]) > abs(a[sel][col])) {
        sel = i;
      }
    }
    if (abs (a[sel][col]) < EPS) {
      continue;
    }
    for (ll i = col; i <= m; ++i) {
      swap(a[sel][i], a[row][i]);
    }
    where[col] = row;
    for (ll i = 0; i < n; ++i) {
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (ll j = col; j <= m; ++j) {
          a[i][j] -= a[row][j] * c;
        }
      }
    }
    ++row;
  }
  ans.assign(m, 0);
  for (ll i = 0; i < m; ++i) {
    if (where[i] != -1) {
      ans[i] = a[where[i]][m] / a[where[i]][i];
    }
  }
  for (ll i = 0; i < n; ++i) {
    double sum = 0;
    for (ll j = 0; j < m; ++j) {
      sum += ans[j] * a[i][j];
    }
    if (abs (sum - a[i][m]) > EPS) {
      return 0;
    }
  }
  for (ll i = 0; i < m; ++i) {
    if (where[i] == -1) {
      return INF;
    }
  }
  return 1;
}
```

6.2 Techniques

```
/*
Dynamic Programming
- Bitmask
- Range
- Digit
- Knapsack
Graph Theory
- Tree diameter
- Reversing edges
- Tree re-rooting
- DP on trees
- DFS tree
- Euler tour
- Binary Jumping
- Centroid
- DAG
- Condense
Data Structures
- Multiple information
- Binary searching on the tree
- 2D range query
- SQRT decomposition
- Small-to-large
Sorting and searching
- Sliding window
- Two pointers
- Binary search on the answer
*/
```

6.3 Ternary Search

```cpp
double ternary_search(double l, double r) {
  double eps = 1e-9;
  while (r - l > eps) {
    double m1 = l + (r - l) / 3;
    double m2 = r - (r - l) / 3;
    double f1 = f(m1);
    double f2 = f(m2);
    if (f1 < f2) {
      l = m1;
    } else {
      r = m2;
    }
  }
  return f(l);
}
```

7 References

7.1 Ref

```
// vector
push_back()
pop_back()
size()
clear()
erase()
empty()
Iterator lower_bound(Iterator first, Iterator last,
    const val)
Iterator upper_bound(Iterator first, Iterator last,
    const val)
// stack
push()
pop()
top()
empty()
size()
// queue
push()
pop()
front()
empty()
back()
size()
// priority_queue
push()
pop()
size()
empty()
top()
// set
insert()
begin()
end()
size()
find()
count()
empty()
// multiset
begin()
end()
size()
max_size()
empty()
insert(x) // O(log n)
clear()
erase(x)
// map
begin()
end()
size()
max_size()
empty()
pair insert(keyvalue, mapvalue)
erase(iterator position)
erase(const g)
clear()
// ordered_set
find_by_order(k)
order_of_key(k)
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set                              \
    tree<int, null_type, less<int>, rb_tree_tag, \
        tree_order_statistics_node_update>
// tuple
get<i>(tuple)
make_tuple(a1, a2, ...)
tuple_size<decltype(tuple)>::value
tuple1.swap(tuple2)
tie(a1, a2, ...) = tuple
tuple_cat(tuple1, tuple2)
// iterator
for (auto it = s.begin(); it != s.end(); it++) cout
    << *it << "\n";
begin()
end()
advance(ptr, k)
next(ptr, k)
prev(ptr, k)
// permutations
do {} while (next_permutation(nums.begin(), nums.
    end()));
// bitset
```

```cpp
int num = 27; // Binary representation: 11011
bitset<10> s(string("0010011010")); // from right
    to left
bitset<sizeof(int) * 8> bits(num);
int setBits = bits.count();
bits.set(index, val);
bits.reset();
bits.flip();
bits.all();
bits.any();
bits.none();
bits.test();
to_string();
to_ulong();
to_ullong();
[], &, |, !, >>=, <<=, &=, |=, ^=, ~;
// sort
sort(v.begin(),v.end());
sort(v.rbegin(),v.rend());
// custom sort
bool comp(string a, string b) {
  if (a.size() != b.size()) return a.size() < b.
      size();
  return a < b; }
sort(v.begin(), v.end(), comp);
// hamming distance
int hamming(int a, int b) { return
    __builtin_popcount(a ^ b); }
// custom comparator for pq
class Compare {
public:
bool operator() (T a, T b) {
if(cond) return true; // do not swap
return false; } };
priority_queue<PII, vector<PII>, Compare> ds;
// gcc compiler
__builtin_popcount(x)
__builtin_parity(x)
__builtin_clz(x) // leading
__builtin_ctz(x) // trailing
```

8 Strings

8.1 Count Unique Substrings

```cpp
int count_unique_substrings(string const& s) {
  int n = s.size();
  const int p = 31;
  const int m = 1e9 + 9;
  vector<long long> p_pow(n);
  p_pow[0] = 1;
  for (int i = 1; i < n; i++) p_pow[i] = (p_pow[i -
      1] * p) % m;
  vector<long long> h(n + 1, 0);
  for (int i = 0; i < n; i++) h[i + 1] = (h[i] + (s
      [i] - 'a' + 1) * p_pow[i]) % m;
  int cnt = 0;
  for (int l = 1; l <= n; l++) {
    unordered_set<long long> hs;
    for (int i = 0; i <= n - l; i++) {
      long long cur_h = (h[i + l] + m - h[i]) % m;
      cur_h = (cur_h * p_pow[n - i - 1]) % m;
      hs.insert(cur_h);
    }
    cnt += hs.size();
  }
```

```cpp
  return cnt;
}
```

8.2 Finding Repetitions

```cpp
vector<int> z_function(string const& s) {
  int n = s.size();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; i++) {
    if (i <= r) z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]])
        z[i]++;
    if (i + z[i] - 1 > r) {
      l = i;
      r = i + z[i] - 1;
    }
  }
  return z;
}
int get_z(vector<int> const& z, int i) {
  if (0 <= i && i < (int) z.size()) return z[i];
  else return 0;
}
vector<pair<int, int>> repetitions;
void convert_to_repetitions(int shift, bool left,
    int cntr, int l, int k1, int k2) {
  for (int l1 = max(1, l - k2); l1 <= min(l, k1);
      l1++) {
    if (left && l1 == l) break;
    int l2 = l - l1;
    int pos = shift + (left ? cntr - l1 : cntr - l
        - l1 + 1);
    repetitions.emplace_back(pos, pos + 2 * l - 1);
  }
}
void find_repetitions(string s, int shift = 0) {
  int n = s.size();
  if (n == 1) return;
  int nu = n / 2;
  int nv = n - nu;
  string u = s.substr(0, nu);
  string v = s.substr(nu);
  string ru(u.rbegin(), u.rend());
  string rv(v.rbegin(), v.rend());
  find_repetitions(u, shift);
  find_repetitions(v, shift + nu);
  vector<int> z1 = z_function(ru);
  vector<int> z2 = z_function(v + '#' + u);
  vector<int> z3 = z_function(ru + '#' + rv);
  vector<int> z4 = z_function(v);
  for (int cntr = 0; cntr < n; cntr++) {
    int l, k1, k2;
    if (cntr < nu) {
      l = nu - cntr;
      k1 = get_z(z1, nu - cntr);
      k2 = get_z(z2, nv + 1 + cntr);
    } else {
      l = cntr - nu + 1;
      k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu))
          ;
      k2 = get_z(z4, (cntr - nu) + 1);
    }
    if (k1 + k2 >= l) convert_to_repetitions(shift,
        cntr < nu, cntr, l, k1, k2);
  }
}
```

8.3 Group Identical Substrings

```cpp
vector<vector<int>> group_identical_strings(vector<
    string> const& s) {
  int n = s.size();
  vector<pair<long long, int>> hashes(n);
  for (int i = 0; i < n; i++) hashes[i] = {
      compute_hash(s[i]), i};
  sort(hashes.begin(), hashes.end());
  vector<vector<int>> groups;
  for (int i = 0; i < n; i++) {
    if (i == 0 || hashes[i].first != hashes[i - 1].
        first) groups.emplace_back();
    groups.back().push_back(hashes[i].second);
  }
  return groups;
}
```

8.4 Hashing

```cpp
ll compute_hash(string const& s) {
  const ll p = 31, m = 1e9 + 9;
  ll hash_value = 0, p_pow = 1;
  for (char c : s) {
    hash_value = (hash_value + (c - 'a' + 1) *
        p_pow) % m;
    p_pow = (p_pow * p) % m;
  }
  return hash_value;
}
```

8.5 Knuth Morris Pratt

```cpp
vector<ll> prefix_function(string s) {
  ll n = (ll) s.length();
  vector<ll> pi(n);
  for (ll i = 1; i < n; i++) {
    ll j = pi[i - 1];
    while (j > 0 && s[i] != s[j]) j = pi[j - 1];
    if (s[i] == s[j]) j++;
    pi[i] = j;
  }
  return pi;
}
// count occurences
vector<int> ans(n + 1);
for (int i = 0; i < n; i++)
  ans[pi[i]]++;
for (int i = n-1; i > 0; i--)
  ans[pi[i-1]] += ans[i];
for (int i = 0; i <= n; i++)
  ans[i]++;
```

8.6 Longest Common Prefix

```cpp
vector<int> lcp_construction(string const& s,
    vector<int> const& p) {
  int n = s.size();
  vector<int> rank(n, 0);
  for (int i = 0; i < n; i++) rank[p[i]] = i;
```

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

8.7   Manacher

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

8.8   Rabin Karp

```
1  vector<ll> rabin_karp(string const& s, string const
     & t) {
2    const ll p = 31, m = 1e9 + 9;
3    ll S = s.size(), T = t.size();
4    vector<ll> p_pow(max(S, T));
5    p_pow[0] = 1;
```

```
6    for (ll i = 1; i < (ll) p_pow.size(); i++) p_pow[
       i] = (p_pow[i-1] * p) % m;
7    vector<ll> h(T + 1, 0);
8    for (ll i = 0; i < T; i++) h[i + 1] = (h[i] + (t[
       i] - 'a' + 1) * p_pow[i]) % m;
9    ll h_s = 0;
10   for (ll i = 0; i < S; i++) h_s = (h_s + (s[i] - '
       a' + 1) * p_pow[i]) % m;
11   vector<ll> occurences;
12   for (ll i = 0; i + S - 1 < T; i++) {
13     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[
       i - 1];
7    for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
8    c[p[0]] = 0;
9    int classes = 1;
10   for (int i = 1; i < n; i++) {
11     if (s[p[i]] != s[p[i-1]]) classes++;
12     c[p[i]] = classes - 1;
13   }
14   vector<int> pn(n), cn(n);
15   for (int h = 0; (1 << h) < n; ++h) {
16     for (int i = 0; i < n; i++) {
17       pn[i] = p[i] - (1 << h);
18       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
         [i - 1];
24     for (int i = n-1; i >= 0; i--) p[--cnt[c[pn[i
         ]]]] = pn[i];
25     cn[p[0]] = 0;
```

```
26     classes = 1;
27     for (int i = 1; i < n; i++) {
28       pair<int, int> cur = {c[p[i]], c[(p[i] + (1
           << h)) % n]};
29       pair<int, int> prev = {c[p[i-1]], c[(p[i-1] +
           (1 << h)) % n]};
30       if (cur != prev) ++classes;
31       cn[p[i]] = classes - 1;
32     }
33     c.swap(cn);
34   }
35   return p;
36 }
37 vector<int> build_suff_arr(string s) {
38   s += "$";
39   vector<int> sorted_shifts = sort_cyclic_shifts(s)
       ;
40   sorted_shifts.erase(sorted_shifts.begin());
41   return sorted_shifts;
42 }
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 + l - (1 <<
       k)) % n]};
46   pair<int, int> b = {c[k][j], c[k][(j + l - (1 <<
       k)) % n]};
47   return a == b ? 0 : a < b ? -1 : 1;
48 }
```

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]])
         z[i]++;
7      if (i + z[i] > r) {
8        l = i;
9        r = i + z[i];
10     }
11   }
12   return z;
13 }
```