Team Notebook

UITS-O(struggle) - University of Information Technology and Sciences

November 3, 2023

Contents

1 Data Structures

1.1 1D Segment Tree

```cpp
/** 1. segment_tree<long long> seg_tree(a);
  * 2. seg_tree.build(1, 0, n - 1);
  * 3. Note : Make sure that the segment tree type and the
       vector type must mathc. E.g If, struct segment_tree <
          long long>
    then, vector must be vector<long long>
  * 4. Note : While using index for answer. Make sure to use
       them as (0 based)
  * 5. now, you're good to go.
 **/
template <typename T>
struct segment_tree {
  int n;
  vector<T> a, tree;
  /* Used to create the tree array */
  segment_tree (vector<T> cpy) {
    a = cpy;
    n = (int) a.size();
    tree.assign(n << 2, 0);
  };
  /* used to build the tree */
  void build (int node, int l, int r) {
    if (l == r) {
      tree[node] = a[l];
      return;
    }
    int mid = (l + r) >> 1;
    build(node << 1, l, mid);
    build((node << 1) + 1, mid + 1, r);
    tree[node] = tree[node << 1] + tree[(node << 1) + 1];
  }
  /* point sum or range sum */
  T sum (int node, int start, int end, int l, int r) {
    if (end < l or r < start) {
      return 0;
    }
    if (l <= start and end <= r) {
      return tree[node];
    }
    int mid = (start + end) >> 1;
    T left_sum = sum(node << 1, start, mid, l, r);
    T right_sum = sum((node << 1) + 1, mid + 1, end, l, r);
    return left_sum + right_sum;
  }
  /* point updating value / adding value */
  void update (int node, int start, int end, int id, T val)
      {
    if (start == end) {
      tree[node] = val;
      return;
    }
    int mid = (start + end) >> 1;
    if (id <= mid) {
      update(node << 1, start, mid, id, val);
    } else {
      update((node << 1) + 1, mid + 1, end, id, val);
    }
    tree[node] = tree[node << 1] + tree[(node << 1) + 1];
  }
};
```

1.2 2D Segment Tree

```cpp
/** 1. _2D_segment_tree<long long> _2D_seg_tree(a);
  * 2. _2D_seg_tree.buildx(1, 0, n - 1);
  * 3. Note : Make sure that the segment tree type and the
       vector type must mathc. E.g If, struct
          _2D_segment_tree <long long>
    then, vector must be vector<long long>
  * 4. Note : While using index for answer. Make sure to use
       them as (0 based)
  * 5. now, you're good to go.
 **/
template <typename T>
struct _2D_segment_tree {
  int n, m;
  vector<vector<T>> a, t;
  _2D_segment_tree (vector<vector<T>> a) {
    this -> a = a;
    this -> n = (int) a.size();
    this -> m = (int) a[0].size();
    t.assign(n << 2, vector<T>(m << 2));
  }
  void build_y (int vx, int lx, int rx, int vy, int ly, int
       ry) {
    if (ly == ry) {
      if (lx == rx) {
        t[vx][vy] = a[lx][ly];
      } else {
        t[vx][vy] = t[(vx << 1)][vy] + t[(vx << 1) + 1][vy];
      }
    } else {
      int my = (ly + ry) >> 1;
      build_y(vx, lx, rx, (vy << 1), ly, my);
      build_y(vx, lx, rx, (vy << 1) + 1, my + 1, ry);
      t[vx][vy] = t[vx][(vy << 1)] + t[vx][(vy << 1) + 1];
    }
  }
  /* Prepares the _2D segment tree
   * _2D_seg_tree.build_x(1, 0, n - 1); */
  void build_x (int vx, int lx, int rx) {
    if (lx != rx) {
      int mx = (lx + rx) >> 1;
      build_x((vx << 1), lx, mx);
      build_x((vx << 1) + 1, mx + 1, rx);
    }
    build_y(vx, lx, rx, 1, 0, m - 1);
  }
  T sum_y (int vx, int vy, int tly, int try_, int ly, int ry
       ) {
    if (ly > ry) {
      return (T) 0;
    }
    if (ly == tly && try_ == ry) {
      return t[vx][vy];
    }
    int tmy = (tly + try_) >> 1;
    return sum_y(vx, (vy << 1), tly, tmy, ly, min(ry, tmy))
       + sum_y(vx, (vy << 1) + 1, tmy + 1, try_, max(ly, tmy
           + 1), ry);
  }
  /* Returns the sum of a sub-matrix from,
   * [(left_x, left_y) top_left corner] to [(right_x,
       right_y) bottom_right corner]
   * _2D_seg_tree.sum_x(1, 0, n - 1, --left_x, --right_x, --
       left_y, --right_y) -> 0 based indexing */
  T sum_x (int vx, int tlx, int trx, int lx, int rx, int ly,
       int ry) {
    if (lx > rx) {
      return 0;
    }
    if (lx == tlx && trx == rx) {
      return sum_y(vx, 1, 0, m - 1, ly, ry);
    }
    int tmx = (tlx + trx) >> 1;
    return sum_x((vx << 1), tlx, tmx, lx, min(rx, tmx), ly,
        ry)
       + sum_x((vx << 1) + 1, tmx + 1, trx, max(lx, tmx + 1),
           rx, ly, ry);
  }
  void update_y (int vx, int lx, int rx, int vy, int ly, int
       ry, int x, int y, T new_val) {
    if (ly == ry) {
      if (lx == rx) {
```

```
      t[vx][vy] = new_val;
    } else {
      t[vx][vy] = t[(vx << 1)][vy] + t[(vx << 1) + 1][vy];
    }
  } else {
    int my = (ly + ry) >> 1;
    if (y <= my) {
      update_y(vx, lx, rx, (vy << 1), ly, my, x, y, new_val
          );
    } else {
      update_y(vx, lx, rx, (vy << 1) + 1, my + 1, ry, x, y,
          new_val);
    }
    t[vx][vy] = t[vx][(vy << 1)] + t[vx][(vy << 1) + 1];
  }
}
/* Updates a particular cell of the matrix - (x_axis,
    y_axis)
 * _2D_seg_tree.update_x(1, 0, n - 1, --left_x, --left_y,
     new_val); */
void update_x (int vx, int lx, int rx, int x, int y, T
    new_val) {
  if (lx != rx) {
    int mx = (lx + rx) >> 1;
    if (x <= mx) {
      update_x((vx << 1), lx, mx, x, y, new_val);
    } else {
      update_x((vx << 1) + 1, mx + 1, rx, x, y, new_val);
    }
  }
  update_y(vx, lx, rx, 1, 0, m - 1, x, y, new_val);
}
};
```

1.3 Disjoint Set Union

```
/* disjoint_set<int> dsu(n + 1); */
/* dsu.make_set(u, v); */
vector<int> par, siz;
template<typename T>
struct disjoint_set {
  int n;
  T find_set(T v) {
    if (par[v] == v) {
      return v;
    }
    return par[v] = find_set(par[v]);
  }
  void init(T v) {
    par[v] = v;
    siz[v] = 1;
  }
  disjoint_set (int n) {
    this -> n = n;
    siz.assign(n + 1, 0);
    par.assign(n + 1, 0);
    for (int u = 1; u <= n; ++u) {
      init(u);
    }
  }
  void make_set(T a, T b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
      if (siz[a] < siz[b]) {
        swap(a, b);
      }
      par[b] = a;
      siz[a] += siz[b];
    }
  }
  T find_group_size(T a) {
    a = find_set(a);
    return siz[a];
  }
};
```

1.4 Lazy Propagation

```
/** 1. struct lazy_propagation <int64_t>lazy_prop(a);
  * 2. lazy_prop.build(1, 0, n - 1);
  * 3. now, you're good to go.
 **/
template <typename T>
struct lazy_propagation {
  struct info {
    T sum = 0, prop = 0;
  };
  int n;
  vector<T> a;
  vector<info> tree;
  lazy_propagation (vector<T> cpy) {
    a = cpy;
    n = (int) a.size();
    tree.resize(4 * n);
  };
  void build (int node, int l, int r) {
    if (l == r) {
      tree[node].sum = a[l];
      return;
    }
    int mid = (l + r) >> 1;
    build(node << 1, l, mid);
    build((node << 1) + 1, mid + 1, r);
    tree[node].sum = tree[node << 1].sum + tree[(node << 1) +
        1].sum;
  }
  T sum (int node, int start, int end, int l, int r, T carry
      = 0) {
    if (end < l or r < start) {
      return 0;
    }
    if (l <= start and end <= r) {
      return tree[node].sum + (((end - start) + 1) * carry);
    }
    int mid = (start + end) >> 1;
    T left_sum = sum(node << 1, start, mid, l, r, carry +
        tree[node].prop);
    T right_sum = sum((node << 1) + 1, mid + 1, end, l, r,
        carry + tree[node].prop);
    return left_sum + right_sum;
  }
  void update (int node, int start, int end, int l, int r, T
      val) {
    if (end < l or r < start) {
      return;
    }
    if (l <= start and end <= r) {
      tree[node].sum += ((end - start) + 1) * val;
      tree[node].prop += val;
      return;
    }
    int mid = (start + end) >> 1;
    update(node << 1, start, mid, l, r, val);
    update((node << 1) + 1, mid + 1, end, l, r, val);
    tree[node].sum = tree[node << 1].sum + tree[(node << 1) +
        1].sum + (((end - start) + 1) * tree[node].prop);
  }
};
```

1.5 Monotonic Stack (Increasing)

```
#include <bits/stdc++.h>
using namespace std;

// Function to build Monotonic
// increasing stack
```

```cpp
void increasingStack(int arr[], int N) {
  // Initialise stack
  stack<int> stk;
  for (int i = 0; i < N; i++) {
  // Either stack is empty or
  // all bigger nums are popped off
    while (stk.size() > 0 && stk.top() > arr[i]) {
      stk.pop();
    }
    stk.push(arr[i]);
  }
  int N2 = stk.size();
  int ans[N2] = { 0 };
  int j = N2 - 1;
  // Empty Stack
  while (!stk.empty()) {
    ans[j] = stk.top();
    stk.pop();
    j--;
  }
  // Displaying the original array
  cout << "The Array: ";
  for (int i = 0; i < N; i++) {
    cout << arr[i] << " ";
  }
  cout << endl;
  // Displaying Monotonic increasing stack
  cout << "The Stack: ";
  for (int i = 0; i < N2; i++) {
    cout << ans[i] << " ";
  }
  cout << endl;
}

// Driver code
int main() {
  int arr[] = { 1, 4, 5, 3, 12, 10 };
  int N = sizeof(arr) / sizeof(arr[0]);
  // Function Call
  increasingStack(arr, N);
  return 0;
}
```

1.6 PBDS with Deletion in Multiset

```cpp
/* Necessary includes */
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
/* The data structure */
typedef tree<ll, null_type, less< ll >, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
/* Functionalities */
ordered_set s; //declaring pbds
s.insert(x); //taking input
s.find(x)==s.end() // search for a present or not
s.order_of_key(x) //postion of x in sorted set
*s.find_by_order(r); // value presnt at index r
s1.insert({x,cnt++}); //insert in multiset
s1.erase(s1.lower_bound({x,-1})); //erase in multiset
s1.find_by_order(x)->first //value of index x in multiset
```

1.7 Policy Based Data Structure

```cpp
/** 1. Firstly, place the header files and namespace and set
     the data type and comparator.
 * 2. ordered_set X;
 * 3. X.insert(8);
 * 4. *X.find_by_order(1)
 *       Note : finds the kth largest or the kth smallest
     element (counting from zero)
 *            i.e. The element at the position i (powerful)
 * 5. X.order_of_key(3)
 *      Note : finds the number of items in a set that are
     strictly smaller than our item
 *            i.e. The position of the current element (
     powerful)
 * Note : This will exactly work like set, multiset, map [
     also can use their functinalities.
 *    -> Not possible : erasing elements with their value (in
      multiset)
 */
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree <
  int, // type
  null_type, // use mapped_type for map
  less<int>, // comparator (less/greater) & type [less_equal
       for multiset]
  rb_tree_tag,
  tree_order_statistics_node_update
> ordered_set;
// Returns the position if found, else returns size
auto pbds_lower_bound = [&] (int el) {
  return (int) ms.order_of_key(el);
};
// Returns the position if found, else returns size
auto pbds_upper_bound = [&] (int el) {
  return (int) ms.order_of_key(el + 1);
};
```

1.8 Sparse Table

```cpp
/* Used for answering queries, but can answer R(Min/Max)Q in
     O(1) */
/* sparse_table<int> st(a, N); */
/* cout << st.min_query(l, r) << '\n'; */
template <typename T>
struct sparse_table {
  int N;
  int n, k;
  vector<T> a;
  vector<T> logs;
  vector<vector<T>> st;
  void gen_logs () {
    logs[1] = 0;
    for (int i = 2; i <= N; i++) {
      logs[i] = logs[i/2] + 1;
    }
  }
  void proc () {
    st[0] = a;
    for (int i = 1; i <= k; ++i) {
      for (int j = 0; j + (1 << i) - 1 < n; ++j) {
        /* Change this line according the question */
        st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i -
             1))]);
      }
    }
  }
  sparse_table (vector<T> a, int N) {
    this -> a = a;
    this -> N = N;
    n = (int) a.size();
    logs.assign(N + 1, 0);
    gen_logs();
    k = logs[n];
    st.assign(k + 1, vector<T>(n + 1, 0));
    proc();
  }
  /* This function is used for only min/max query O(1); */
  T min_query (int l, int r) {
    int p = logs[r - l + 1];
    return min(st[p][l], st[p][r - (1 << p) + 1]);
  }
```

```
  /* This function is used for the rest of the query O(log(n
      )) */
  T sum_query (int l, int r) {
    T sum = 0;
    for (int i = k; i >= 0; --i) {
      if ((1 << i) <= r - l + 1) {
        sum += st[i][l];
        l += 1 << i;
      }
    }
    return sum;
  }
};
```

2 Digit DP

2.1 Find the unluckiest number in range

```
/* Let's define the luckiness of a number x as the
     difference between the largest and smallest digits of
     that number. For example, 142857 has 8 as its largest
     digit and 1 as its smallest digit, so its luckiness is
     81=7. And the number 111 has all digits equal to 1, so
     its luckiness is zero.
   Now, you've to find the unluckiest number in the range (l
       , r) */

#include <bits/stdc++.h>
using namespace std;

string str_a, str_b;
long long overall_ans;
short overall_min_dist = 9;
short dp[19][2][2][10][10]; // This is the time complexity
     as well

short sol (int pos = 0, bool bigger_a = false, bool
     smaller_b = false, short max_dig = 0, short min_dig =
     9, long long new_seq = 0) {
  if (pos == (int) str_b.size()) {
    if (max_dig - min_dig <= overall_min_dist) {
      overall_min_dist = max_dig - min_dig;
      overall_ans = new_seq;
    }
    return max_dig - min_dig;
  }
  short& tmp_dist = dp[pos][bigger_a][smaller_b][max_dig][
      min_dig];
  if (tmp_dist != -1) {
    return tmp_dist;
  }
  tmp_dist = 9;
  short left_cand = bigger_a ? 0 : str_a[pos] - '0';
  short right_cand = smaller_b ? 9 : str_b[pos] - '0';
  for (short tmp_dig = left_cand; tmp_dig <= right_cand; ++
       tmp_dig) {
    tmp_dist = min(tmp_dist, sol(pos + 1, bigger_a | str_a[
         pos] - '0' < tmp_dig, smaller_b | tmp_dig < str_b[
         pos] - '0', max(max_dig, tmp_dig), min(min_dig,
         tmp_dig), new_seq * 10 + tmp_dig));
  }
  return tmp_dist;
}

int main () {
  ios::sync_with_stdio(false);
  cin.tie(0);
  int tt;
  cin >> tt;
  while (tt--) {
    long long a, b;
    cin >> a >> b;
    str_a = to_string(a);
    str_b = to_string(b);
    if ((int) str_a.size() < (int) str_b.size()) {
      cout << string((int) str_a.size(), '9') << '\n';
      continue;
    }
    overall_min_dist = 9;
    memset(dp, -1, sizeof dp);
    sol();
    cout << overall_ans << '\n';
  }
  return 0;
}
```

3 Dynamic Programming

3.1 Count all subset having K set bits after bitwise AND

```
// Count all subsets having K set bits after bitwise AND

#include <bits/stdc++.h>
using namespace std;

const int maxn = (int) 2e5;
const int mod = (int) 1e9 + 7;
const int all_subset = 1 << 6;
typedef vector<int> custom;

int add_mod (int a, int b) {
  int res = (a + b) % mod;
  res += (res < 0 ? mod : 0);
  return res;
}

int main () {
  ios::sync_with_stdio(false);
  cin.tie(0);
  int tt;
  cin >> tt;
  while (tt--) {
    int n, k;
    cin >> n >> k;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) {
      cin >> a[i];
    }
    vector<custom> dp(n + 1, custom(all_subset));
    for (int i = 0; i < n; ++i) {
      dp[i + 1][a[i]] = add_mod(dp[i + 1][a[i]], 1); // Start
           a subsequence from the (ith) element
      for (int mask = 0; mask < all_subset; ++mask) {
        dp[i + 1][mask] = add_mod(dp[i + 1][mask], dp[i][mask
             ]); // Don't take the (ith) element into the
             current subsequence
        dp[i + 1][mask & a[i]] = add_mod(dp[i + 1][mask & a[i
             ]], dp[i][mask]); // Take the (ith) element into
              the current subsequence
      }
    }
    long long ans = 0;
    for (int mask = 0; mask < all_subset; ++mask) {
      if (__builtin_popcount(mask) == k) {
        ans = add_mod(ans, dp[n][mask]);
      }
    }
    cout << ans << '\n';
  }
  return 0;
}
```

3.2 LCS in O(n) space

```
// Space Optimized LCS
// Space - O(N) | Time - (N * M)

#include <bits/stdc++.h>
using namespace std;

int main() {
  string s = "AGGTAB";
  string t = "GXTXAYB"; // Ans = 4
  int n = (int) s.size();
  int m = (int) t.size();
  vector<vector<int>> dp(2, vector<int>(m + 1));
  for (int i = 0; i <= n; ++i) {
    int k = i & 1;
    for (int j = 0; j <= m; ++j) {
      if (i == 0 or j == 0) {
        dp[k][j] = 0;
      } else if (s[i - 1] == t[j - 1]) {
        dp[k][j] = 1 + dp[k ^ 1][j - 1];
      } else {
        dp[k][j] = max(dp[k ^ 1][j], dp[k][j - 1]);
      }
    }
  }
  cout << max(dp[1][m], dp[0][m]) << '\n';
  return 0;
}
```

3.3 LIS in O(nlogn) time

```
// LIS in O(nlog(n))
int LIS (vector<int> const& a) {
  int n = a.size();
  const int inf = 1e9;
  vector<int> dp(n + 1, inf);
  dp[0] = -inf;
  for (int i = 0; i < n; i++) {
    int l = upper_bound(dp.begin(), dp.end(), a[i]) - dp.
        begin();
    if (dp[l - 1] < a[i] and a[i] < dp[l]) {
      dp[l] = a[i];
    }
  }
  int ans = 0;
  for (int l = 0; l <= n; l++) {
    if (dp[l] < inf) {
      ans = l;
    }
  }
  return ans;
}
```

4 Geometry

4.1 Convex Hull (CSES)

```
/* Given a set of n points in the two-dimensional plane,
your task is to determine the convex hull of the points
.
Input
The first input line has an integer n: the number of points.
After this, there are n lines that describe the points. Each
line has two integers x and y: the coordinates of a
point.
You may assume that each point is distinct, and the area of
the hull is positive.
Output
First print an integer k: the number of points in the convex
hull.
After this, print k lines that describe the points. You can
print the points in any order. Print all points that
lie on the convex hull.
Constraints
* 3 <= n <= 2*10^5
* -10^9 <= x,y <= 10^9
Example Input:
6
2 1
2 5
3 3
4 3
4 4
6 3
Example Output:
4
2 1
2 5
4 4
6 3 */
C++ Solution:
#define int long long
#define P complex<int>
#define X real()
#define Y imag()
int cross(P &a, P &b, P &c) {
  P u = c - b;
  P v = a - b;
  int cp = (conj(u) * v).Y;
  return (cp > 0) - (cp < 0);
}
vector<P> hull(vector<P> &v) {
  vector<P> ans = {v[0]};
  for (int i = 1; i < v.size(); i++) {
    while (ans.size() > 1) {
      P b = ans.back();
      ans.pop_back();
      P a = ans.back();
      P c = v[i];
      if (cross(a, b, c) != 1) {
        ans.push_back(b);
        break;
      }
    }
    ans.push_back(v[i]);
  }
  return ans;
}
signed main() {
  int n;
  cin >> n;
  vector<P> v(n);
  for (int i = 0; i < n; i++) {
    int x, y;
    cin >> x >> y;
    v[i] = {x, y};
  }
  sort(v.begin(), v.end(), [] (const P &a, const P &b) {
    return (a.X == b.X) ? (a.Y < b.Y) : (a.X < b.X);
  });
  vector<P> v1 = hull(v);
  sort(v.begin(), v.end(), [] (const P &a, const P &b) {
    return (a.X == b.X) ? (a.Y > b.Y) : (a.X > b.X);
  });
  vector<P> v2 = hull(v);
  for (int i = 1; i < v2.size(); i++) {
    if (v2[i] == v1[0])
      break;
    v1.push_back(v2[i]);
  }
  cout << v1.size() << endl;
  for (auto i: v1)
    cout << i.X << " " << i.Y << endl;
}
```

4.2 Line Segment Intersection (CSES)

```
/* There are two line segments: the first goes through the
points (x1,y1) and (x2,y2), and the second goes through
the points (x3,y3) and (x4,y4).
Your task is to determine if the line segments intersect, i.
e., they have at least one common point.
Input
The first input line has an integer t: the number of tests.
After this, there are t lines that describe the tests. Each
line has eight integers x1, y1, x2, y2, x3, y3, x4 and
y4.
Output
For each test, print "YES" if the line segments intersect
and "NO" otherwise.
Constraints
* 1 <= t <= 10^5
* -10^9 <= x1,y1,x2,y2,x3,y3,x4,y4 <= 10^9
* (x1,y1) != (x2,y2)
* (x3,y3) != (x4,y4)
Example Input:
5
1 1 5 3 1 2 4 3
1 1 5 3 1 1 4 3
1 1 5 3 2 3 4 1
1 1 5 3 2 4 4 1
1 1 5 3 3 2 7 4
Example Output:
NO
YES
YES
YES
YES */
C++ Solution:
#define int long long
#define P complex<int>
#define X real()
#define Y imag()
int cross(P a, P b, P c) {
  P u = b - a;
  P v = c - a;
  return (conj(u)*v).Y;
}
bool comp(const P &a, const P &b) {
  return (a.X == b.X) ? (a.Y < b.Y) : (a.X < b.X);
}
bool mid(P a, P b, P c) {
  vector<P> v = {a, b, c};
  sort(v.begin(), v.end(), comp);
  return (v[1] == c);
}
int sgn(int x) {
  return (x > 0) - (x < 0);
}
bool check(P a, P b, P c, P d) {
  int cp1 = cross(a, b, c);
  int cp2 = cross(a, b, d);
  int cp3 = cross(c, d, a);
  int cp4 = cross(c, d, b);
  if (cp1 == 0 && mid(a, b, c))
    return 1;
  if (cp2 == 0 && mid(a, b, d))
    return 1;
  if (cp3 == 0 && mid(c, d, a))
    return 1;
  if (cp4 == 0 && mid(c, d, b))
    return 1;
  if (sgn(cp1) != sgn(cp2) && sgn(cp3) != sgn(cp4))
    return
      1;
  return 0;
}
signed main() {
  int t;
  cin>>t;
  while (t--) {
    int x, y;
    P a, b, c, d;
    cin >> x >> y;
    a = {x, y};
    cin >> x >> y;
    b = {x, y};
    cin >> x >> y;
    c = {x, y};
    cin >> x >> y;
    d = {x, y};
    cout << (check(a, b, c, d) ? "YES" : "NO") << endl;
  }
}
```

4.3 Minimum Euclidean Distance (CSES)

```
/* Given a set of points in the two-dimensional plane, your
task is to find the minimum Euclidean distance between
two distinct points.
The Euclidean distance of points (x1,y1) and (x2,y2) is sqrt
((x1-x2)^2 + (y1-y2)^2).
Input
The first input line has an integer n: the number of points.
After this, there are n lines that describe the points. Each
line has two integers x and y. You may assume that
each point is distinct.
Output
Print one integer: d^2 where d is the minimum Euclidean
distance (this ensures that the result is an integer).
Constraints
* 2 <= n <= 2*10^5
* -10^9 <= x,y <= 10^9
Example Input:
4
2 1
4 4
1 2
6 3
Example Output:
2 */
C++ Solution:
#define int long long
#define P pair<int, int>
#define X first
#define Y second
int norm(P a, P b) {
return (b.X - a.X) * (b.X - a.X) + (b.Y - a.Y) * (b.Y - a.
Y);
}
signed main(){
int n; cin>>n;
vector<P> v(n);
int d = LLONG_MAX;
for (int i = 0; i < n; i++) {
int x, y; cin >> x >> y;
v[i] = {x, y};
}
sort(v.begin(), v.end());
set<P> s = {{v[0].Y, v[0].X}};
int j = 0;
for (int i = 1; i < n; i++) {
auto it = s.begin();
int dd = ceil(sqrt(d));
while (j < i && v[j].X < v[i].X - dd) {
s.erase({v[j].Y, v[j].X});
j++;
}
auto l = s.lower_bound({v[i].Y - dd, 0});
auto r = s.upper_bound({v[i].Y + dd, 0});
for (auto it = l; it != r; it++) {
d = min(d, norm({it->Y, it->X}, v[i]));
}
s.insert({v[i].Y, v[i].X});
```

```
}
cout << d;
}
```

4.4 Point Location Test (CSES)

```
/* Your task is to calculate the area of a given polygon.
The polygon consists of n vertices (x1,y1),(x2,y2),...,(xn,
yn). The vertices (xi,yi) and (xi+1,yi+1) are adjacent
for i=1,2,...,n1, and the vertices (x1,y1) and (xn,yn)
are also adjacent.
Input
The first input line has an integer n: the number of
vertices.
After this, there are n lines that describe the vertices.
The ith such line has two integers xi and yi.
You may assume that the polygon is simple, i.e., it does not
intersect itself.
Output
Print one integer: 2a where the area of the polygon is a (
this ensures that the result is an integer).
Constraints
* 3 <= n <= 1000
* -10^9 <= xi,yi <= 10^9
Example Input:
4
1 1
4 2
3 5
1 4
Example Output:
16 */
C++ Solution:
#define int long long
#define X first
#define Y second
signed main() {
  int n;
  cin>>n;
  pair<int,int> a[n];
  for (int i = 0; i < n; i++)
    cin>>a[i].X>>a[i].Y;
  int ans = 0;
//shoelace formula
  for (int i = 0; i < n; i ++) {
    ans += (a[i].X*a[(i+1)%n].Y - a[(i+1)%n].X*a[i].Y);
  }
  cout<<abs(ans);
}
```

4.5 Point in Polygon (CSES)

```
/* You are given a polygon of n vertices and a list of m
points. Your task is to determine for each point if it
is inside, outside or on the boundary of the polygon.
The polygon consists of n vertices (x1,y1),(x2,y2),...,(xn,
yn). The vertices (xi,yi) and (xi+1,yi+1) are adjacent
for i=1,2,...,n1, and the vertices (x1,y1) and (xn,yn)
are also adjacent.
Input
The first input line has two integers n and m: the number of
vertices in the polygon and the number of points.
After this, there are n lines that describe the polygon. The
ith such line has two integers xi and yi.
You may assume that the polygon is simple, i.e., it does not
intersect itself.
Finally, there are m lines that describe the points. Each
line has two integers x and y.
Output
For each point, print "INSIDE", "OUTSIDE" or "BOUNDARY".
Constraints
* 3 <= n,m <= 1000
* 1 <= m <= 1000
* -10^9 <= xi,yi <= 10^9
* -10^9 <= x,y <= 10^9
Example Input:
4 3
1 1
4 2
3 5
1 4
2 3
3 1
1 3
Example Output:
INSIDE
OUTSIDE
BOUNDARY */
C++ Solution:
#define int long long
#define P complex<int>
#define X real()
#define Y imag()
const int INF = 1e9 + 7;
int cross(P a, P b, P c) {
  P u = b - a;
  P v = c - a;
  int cp = (conj(u)*v).Y;
  return (cp > 0) - (cp < 0);
}
bool comp(const P &a, const P &b) {
  return (a.X == b.X) ? (a.Y < b.Y) : (a.X < b.X);
}
bool mid(P a, P b, P c) {
  vector<P> v = {a, b, c};
  sort(v.begin(), v.end(), comp);
  return (v[1] == c);
}
bool check(P a, P b, P c, P d) {
  int cp1 = cross(a, b, c);
  int cp2 = cross(a, b, d);
  int cp3 = cross(c, d, a);
  int cp4 = cross(c, d, b);
  if (cp1 * cp2 < 0 && cp3 * cp4 < 0)
    return 1;
  if (cp3 == 0 && mid(c, d, a) && cp4 < 0)
    return 1;
  if (cp4 == 0 && mid(c, d, b) && cp3 < 0)
    return 1;
  return 0;
}
signed main() {
// https://en.wikipedia.org/wiki/Point_in_polygon#
  Ray_casting_algorithm
  int n, m;
  cin >> n >> m;
  vector<P> v;
  for (int i = 0; i < n; i++) {
    int x, y;
    cin >> x >> y;
    v.push_back({x, y});
  }
  v.push_back(v[0]);
  while (m--) {
    int x, y;
    cin >> x >> y;
    P a = {x, y};
    P b = {INF, INF};
    int cnt = 0;
    int flag = 0;
    for (int i = 0; i < n; i++) {
      int cp = cross(v[i], v[i+1], a);
      if (cp == 0 && mid(v[i], v[i+1], a)) {
        flag = 1;
        break;
      }
      cnt += check(v[i], v[i+1], a, b);
```

```
    }
    if (flag)
      cout << "BOUNDARY" << endl;
    else
      cout << (cnt & 1 ? "INSIDE" : "OUTSIDE") << endl;
  }
}
```

4.6 Polygon Area (CSES)

```
/* Your task is to calculate the area of a given polygon.
The polygon consists of n vertices (x1,y1),(x2,y2),...,(xn,
yn). The vertices (xi,yi) and (xi+1,yi+1) are adjacent
for i=1,2,...,n1, and the vertices (x1,y1) and (xn,yn)
are also adjacent.
Input
The first input line has an integer n: the number of
vertices.
After this, there are n lines that describe the vertices.
The ith such line has two integers xi and yi.
You may assume that the polygon is simple, i.e., it does not
intersect itself.
Output
Print one integer: 2a where the area of the polygon is a (
this ensures that the result is an integer).
Constraints
* 3 <= n <= 1000
* -10^9 <= xi,yi <= 10^9
Example Input:
4
1 1
4 2
3 5
1 4
Example Output:
16 */
C++ Solution:
#define int long long
#define X first
#define Y second
signed main() {
  int n;
  cin>>n;
  pair<int,int> a[n];
  for (int i = 0; i < n; i++)
    cin>>a[i].X>>a[i].Y;
  int ans = 0;
//shoelace formula
  for (int i = 0; i < n; i ++) {
    ans += (a[i].X*a[(i+1)%n].Y - a[(i+1)%n].X*a[i].Y);
  }
  cout<<abs(ans);
}
```

4.7 Polygon Lattice Points (CSES)

```
/* Given a polygon, your task is to calculate the number of
lattice points inside the polygon and on its boundary.
A lattice point is a point whose coordinates are
integers.
The polygon consists of n vertices (x1,y1),(x2,y2),...,(xn,
yn). The vertices (xi,yi) and (xi+1,yi+1) are adjacent
for i=1,2,...,n1, and the vertices (x1,y1) and (xn,yn)
are also adjacent.
Input
The first input line has an integer n: the number of
vertices.
After this, there are n lines that describe the vertices.
The ith such line has two integers xi and yi.
You may assume that the polygon is simple, i.e., it does not
intersect itself.
Output
Print two integers: the number of lattice points inside the
polygon and on its boundary.
Constraints
* 3 <= n <= 10^5
* -10^6 <= xi,yi <= 10^6
Example Input:
4
1 1
5 3
3 5
1 4
Example Output:
6 8 */
C++ Solution:
#define int long long
#define P complex<int>
#define X real()
#define Y imag()
signed main() {
//  picks  theorem + https://math.stackexchange.com/a
     /301895/530789
  int n;
  cin >> n;
  vector<P> v(n);
  for (int i = 0; i < n; i++) {
    int x, y;
    cin >> x >> y;
    v[i] = {x, y};
  }
  v.push_back(v[0]);
  int area = 0;
  int b = 0;
  for (int i = 0; i < n; i++) {
    P x = v[i], y = v[i+1];
    area += (conj(x) * y).Y;
    P z = y - x;
    int g = __gcd(z.X, z.Y);
    b += abs(g);
  }
// 2*area = 2*a + b - 2
  int a = abs(area) - b + 2;
  cout << a/2 <<      << b;
}
```

5 Graph Theory

5.1 Bellman Ford

```
/* Time complexity: O(V * E) */
/* Add edges both ways for undirected graph */
/* g.push_back({u, v, w}); */
/* g.push_back({v, u, w}); */
int cyc_node = -1;
const int inf = (int) 1e9;
vector<int> dist(n + 1, inf);
vector<int> par(n + 1, -1);
auto check_neg_cycle = [&] () {
  if (cyc_node == -1) {
    return printf("No neg cycle\n"), 0;
  }
  int u = cyc_node;
  for (int i = 1; i <= n; ++i) {
    u = par[u];
  }
  vector<int> path;
  for (int cur = u; ; cur = par[cur]) {
    path.push_back(cur);
    if (1 < (int) path.size() and cur == u) {
   break;
    }
  }
  reverse(path.begin(), path.end());
  for (auto p : path) {
    printf("%d ", p);
```

```
  }
  printf("\n");
  return 0;
};
auto bellman_ford = [&] (int src) {
  dist[src] = 0;
  for (int i = 1; i <= n; ++i) {
    cyc_node = -1;
    for (int j = 0; j < m; ++j) {
   int u = g[j][0];
   int v = g[j][1];
   int w = g[j][2];
   if (dist[u] < inf) {
     if (dist[u] + w < dist[v]) {
    dist[v] = max(-inf, dist[u] + w);
    par[v] = u;
    cyc_node = v;
     }
   }
    }
  }
  check_neg_cycle();
};
bellman_ford(1);
```

5.2 Bipartite Check

```
int n;
vector<vector<int>> adj;

vector<int> side(n, -1);
bool is_bipartite = true;
queue<int> q;
for (int st = 0; st < n; ++st) {
    if (side[st] == -1) {
        q.push(st);
        side[st] = 0;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int u : adj[v]) {
                if (side[u] == -1) {
                    side[u] = side[v] ^ 1;
                    q.push(u);
                } else {
                    is_bipartite &= side[u] != side[v];
                }
            }
        }
    }
}

cout << (is_bipartite ? "YES" : "NO") << endl;
```

5.3 Dijkstras algorithm

```
/* Used to extract the shortest path from source(u) to
     destination(v) */
/* Time complexity: O(V + E log V) */
/* dijkstra<int> dij(g, n, --src); */
/* dij.proc_tab(); */
/* cout << dij.get_dist(--v) << '\n'; */
/* auto path = dij.get_path(v); */
/* Follows 0-based indexing */
template <typename T>
struct dijkstra {
  int n;
  int src;
  // Change this (inf) according to the question
  const T inf = (T) 1e16;
  vector<int> par, seen;
  vector<T> dist;
  vector<vector<array<int, 2>>> g;
  dijkstra (vector<vector<array<int, 2>>> g, int n, int src)
       {
    this -> g = g;
    this -> n = n;
    this -> src = src;
    // Remove this (par) if not needed
    par.assign(n, -1);
    seen.assign(n, false);
    dist.assign(n, inf);
  }
  void proc_tab () {
    multiset<array<T, 2>> ms;
    dist[src] = 0;
    ms.insert({0, src});
    while (!ms.empty()) {
      auto u = *ms.begin();
      ms.erase(ms.begin());
      if (!seen[u[1]]) {
        seen[u[1]] = true;
        for (auto ch : g[u[1]]) {
          if (dist[u[1]] + ch[1] < dist[ch[0]]) {
            dist[ch[0]] = dist[u[1]] + ch[1];
            /* Here saving the previous node as parent if
                 this is giving less cost */
            par[ch[0]] = u[1];
            ms.insert({dist[ch[0]], ch[0]});
          }
        }
      }
    }
  }
  T get_dist (int dest) {
    return dist[dest];
  }
  vector<int> get_path (int dest) {
    vector<int> path;
    for (int v = dest; v != -1; v = par[v]) {
      path.push_back(v + 1);
    }
    reverse(path.begin(), path.end());
    return path;
  }
};
```

5.4 Find farthest leaf distance from each node

```
/* This code finds the farthes nodes/leaves from each node
   IDEA:
     1. At first maintain 1st and 2nd farthest node from each
          node in it's own subtree, while climbing up the
         tree levels
       - denoted as (down[v].first | down[v].second)
     2. Then, find the farthest node outside of it's subtree,
          while climbing down the tree levels
       - denoted as (up[v])
     3. Then, farthest dist would be farthest_dist[v] = max(
         up[v], down[v].first)
   Time complexity: Linear */

#include <bits/stdc++.h>
using namespace std;

const int N = (int) 2e5 + 1;
vector<int> parent(N);
vector<int> nei[N], depth_ver[N];

struct value {
  int val = 0;
  int ver = 0;
};

void dfs (int u, int p = -1, int l = 0) {
```

```
    parent[u] = p;
    depth_ver[l].push_back(u);
    for (auto v : nei[u]) {
      if (p != v) {
        dfs(v, u, l + 1);
      }
    }
}

int main(){
  ios::sync_with_stdio(0);
  cin.tie(0);
  int tt;
  cin >> tt;
  while (tt--) {
    int n, k, c;
    cin >> n >> k >> c;
    for (int i = 0; i < n - 1; ++i) {
      int u, v;
      cin >> u >> v;
      nei[u].push_back(v);
      nei[v].push_back(u);
    }
    dfs(1);
    vector<pair<value, value>> down(n + 1);
    for (int l = n - 1; 0 <= l; --l) {
      for (auto u : depth_ver[l]) {
        for (auto v : nei[u]) {
          if (v != parent[u]) {
            if (down[u].first.val < down[v].first.val + 1) {
              down[u].first.val = down[v].first.val + 1;
              down[u].first.ver = v;
            }
          }
        }
        for (auto v : nei[u]) {
          if (v != parent[u] and v != down[u].first.ver) {
            if (down[u].second.val < down[v].first.val + 1) {
              down[u].second.val = down[v].first.val + 1;
              down[u].second.ver = v;
            }
          }
        }
      }
    }
    vector<int> up(n + 1);
    for (int l = 1; l <= n - 1; ++l) {
      for (auto u : depth_ver[l]) {
        int p = parent[u];
        up[u] = up[p] + 1;
        if (down[p].first.ver == u) {
          up[u] = max(up[u], down[p].second.val + 1);
        } else {
          up[u] = max(up[u], down[p].first.val + 1);
        }
      }
    }
    vector<int> farthest_node_dist(n + 1); // This vector
        contains the farthest node/leaf dist
    for (int u = 1; u <= n; ++u) {
      farthest_node_dist[u] = max(up[u], down[u].first.val);
      cout << "farthest_node[" << u << "] = " <<
          farthest_node_dist[u] << '\n';
    }
    cout << "\n";
    for (int u = 1; u <= n; ++u) {
      nei[u].clear();
      int level = u - 1;
      depth_ver[level].clear();
    }
  }
  return 0;
}
```

5.5 Floyd Warshall

```
/* Time complexity: O(n ^ 3) */
auto floyd_warshall = [&] () {
  /* init the (d) array, if there doesn't exists a path b/w
      u-v then set them to infinity */
  const int inf = (int) 1e9;
  for (int k = 1; k <= n; ++k) {
    for (int i = 1; i <= n; ++i) {
      for (int j = 1; j <= n; ++j) {
       /* if there exists both of these path or not */
        if (d[i][k] < inf and d[k][j] < inf) {
          d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
        }
      }
    }
  }
};
```

5.6 Johnsons algorithm

```
// Johnson Algorithm (graph) all pair shortest path with
    negative cost.
/// Johnson's algorithm for all pair shortest paths in
    sparse graphs
/// Complexity: O(N * M) + O(N * M * log(N))

#define MAX 90
#define clr(ar) memset(ar, 0, sizeof(ar))
typedef long long ll;
const ll INF = (1LL << 60) - 666;
struct Edge{ // u to v edge
    int u, v;
    ll w;
    Edge(){}
    Edge(int u, int v, ll w) : u(u), v(v), w(w){}
};
bool bellman_ford(int n, int src, vector <Edge> E, vector <
    ll>& dis){
    dis[src] = 0;
    for (int i = 0; i <= n; i++){
        int flag = 0;
        for (auto e: E){
            if ((dis[e.u] + e.w) < dis[e.v]){
                flag = 1;
                dis[e.v] = dis[e.u] + e.w;
            }
        }
        if (flag == 0) return true;
    }
    return false;
}
vector <ll> dijkstra(int n, int src, vector <Edge> E, vector
      <ll> potential){
    set<pair<ll, int> > S;
    vector <ll> dis(n + 1, INF);
    vector <ll> temp(n + 1, INF);
    vector <pair<int, ll> > adj[n + 1];
    dis[src] = temp[src] = 0;
    S.insert(make_pair(temp[src], src));
    for (auto e: E){
        adj[e.u].push_back(make_pair(e.v, e.w));
    }
    int __sigh = 0;
    while (!S.empty()){
        pair<ll, int> cur = *(S.begin());
        S.erase(cur);
        int u = cur.second;
        for (int i = 0; i < adj[u].size(); i++){
            int v = adj[u][i].first;
            ll w = adj[u][i].second;
            if ((temp[u] + w) < temp[v]){
                S.erase(make_pair(temp[v], v));
```

```
            temp[v] = temp[u] + w;
            dis[v] = dis[u] + w;
            S.insert(make_pair(temp[v], v));
         }
      }
   }
   return dis;
}
void johnson(int n, ll ar[MAX][MAX], vector <Edge> E){
   vector <ll> potential(n + 1, INF);
   for (int i = 1; i <= n; i++) E.push_back(Edge(0, i, 0));

   assert(bellman_ford(n, 0, E, potential));
   for (int i = 1; i <= n; i++) E.pop_back();

   for (int i = 1; i <= n; i++){
      vector <ll> dis = dijkstra(n, i, E, potential);
      for (int j = 1; j <= n; j++){
         ar[i][j] = dis[j];
      }
   }
}
ll ar[MAX][MAX];// output all pair shortest distance
vector <Edge> E; // input graph
```

5.7 Kruskals MST

```
/* Before this, make sure to write the dsu algo */
/* Time complexity: O(mlog(m)) - only for sorting, others
    done in constant time */
disjoint_set<int> dsu(n + 1);
auto kruskals = [&] () {
  long long min_cost = 0;
  /* The edges must be sorted in asc according to their
      weights */
  sort(p.begin(), p.end());
  for (int i = 0; i < m; ++i) {
    /* p[i][0] = cost, p[i][1] = u, p[i][2] = v; */
    if (dsu.find_set(p[i][1]) != dsu.find_set(p[i][2])) {
      min_cost += p[i][0];
      dsu.make_set(p[i][1], p[i][2]);
    }
  }
  return min_cost;
};
```

5.8 Lowest Common Ancestor

```
/* Time complexity: Build O(nlog(n)), Query O(log(n)); */
/* binary_lifting bl(n, m, g); */
/* bl.lca(u, v); */
/* bl.get_dist(u, v); */
struct binary_lifting {
  int n, m;
  vector<int> lvl;
  vector<vector<int>> g;
  vector<vector<int>> par;
  void dfs (int v, int l, int p) {
    lvl[v] = l;
    par[v][0] = p;
    for (auto ch : g[v]) {
      if (ch != p) {
        dfs(ch, l + 1, v);
      }
    }
  }
  void init () {
    dfs(1, 0, -1);
    const int logn = __lg(n);
    for (int i = 1; i <= logn; ++i) {
      for (int j = 1; j <= n; ++j) {
        if (par[j][i - 1] != -1) {
          int p = par[j][i - 1];
          par[j][i] = par[p][i - 1];
        }
      }
    }
  }
  binary_lifting (int n, int m, vector<vector<int>> g) {
    this -> n = n;
    this -> m = m;
    this -> g = g;
    lvl.assign(n + 1, 0);
    const int logn = __lg(n);
    par.assign(n + 1, vector<int>(logn + 1, -1));
    init();
  }
  int lca (int u, int v) {
    if (lvl[v] < lvl[u]) {
      swap(v, u);
    }
    int d = lvl[v] - lvl[u];
    while (d) {
      int logd = __lg(d);
      v = par[v][logd];
      d -= (1 << logd);
    }
    if (u == v) {
      return u;
    }
    int logn = __lg(n);
    for (int i = logn; i >= 0; --i) {
      if (par[u][i] != -1 and par[u][i] != par[v][i]) {
        u = par[u][i];
        v = par[v][i];
      }
    }
    return par[u][0];
  }
  int get_dist (int u, int v) {
    int com_ances = lca(u, v);
    return lvl[u] + lvl[v] - (lvl[com_ances] << 1);
  }
};
```

5.9 Topological Sort

```
/* A topological sort of a directed acyclic graph is a
linear ordering of its vertices such that for every
directed edge (u -> v) from vertex (u) to vertex (v), (
u) comes before (v) in the ordering.
This code includes:
1. Topological ordering of a Directed Acyclic Graph (DAG)
2. How to check if a topological ordering is valid or not
*/

#define pb push_back
int N; // Number of nodes
// Assume that this graph is a DAG
vector<int> graph[100000], top_sort;
bool visited[100000];
void dfs(int node) {
  for (int i : graph[node]) {
    if (!visited[i]) {
      visited[i] = true;
      dfs(i);
    }
  }
  top_sort.pb(node);
}
void compute() {
  for (int i = 0; i < N; i++) {
    if (!visited[i]) {
      visited[i] = true;
      dfs(i);
    }
  }
```

```
  reverse(begin(top_sort), end(top_sort));
// The vector  top_sort  is now topologically sorted
}
int main() {
  int M;
  cin >> N >> M;
  for (int i = 0; i < M; ++i) {
    int a, b;
    cin >> a >> b;
    graph[a - 1].pb(b - 1);
  }
  compute();
  vector<int> ind(N);
  for (int i = 0; i < N; i++)
    ind[top_sort[i]] = i;
  for (int i = 0; i < N; i++) {
    for (int j : graph[i])
      if (ind[j] <= ind[i]) {
        cout << "IMPOSSIBLE\n"; // topological sort wasnt
        valid
        exit(0);
      }
  }
  for (int i : top_sort)
    cout << i + 1 << " ";
  cout << "\n";
}
```

6 Miscellaneous

6.1 Direction Arrays

```
int dx[] = { +1, 0, -1, 0, +1, +1, -1, -1};
int dy[] = {0, +1, 0, -1, +1, -1, +1, -1};
int dx[] = { +0, +0, +1, -1, -1, +1, -1, +1}; ///King's Move
int dy[] = { -1, +1, +0, +0, +1, +1, -1, -1}; ///king's Move
int dx[] = { -2, -2, -1, -1, +1, +1, +2, +2}; ///knight's
    Move
int dy[] = { -1, +1, -2, +2, -2, +2, -1, +1}; ///knight's
    Mov
```

6.2 First K digits of N!

```
// first k digit of N!
// sometime give error in kth digit.
ll leadingDigitFact ( ll n, ll k ) {
  long double fact = 0;
  for ( ll i = 1; i <= n; i++ ) {
    fact += log10 ( i );
  }
  long double q = fact - floor ( fact + eps );
  long double B = pow ( 10, q );
  for ( ll i = 0; i < k - 1; i++ ) {
    B *= 10LL;
  }
  return floor(B + eps);
}
```

6.3 Number of digits in N!

```
Number of digit in N! ::
int factorialDigit ( int n ) {
  double x = 0;
  for ( int i = 1; i <= n; i++ ) {
    x += log10 ( i );
  }
  int res = x + 1 + eps;
  return res;
}
```

6.4 Number of digits in N

```
// Number of digit in n ::
const double eps = 1e-9;
int numberDigit ( int n ) {
  int rightAnswer = log10(n) + 1 + eps;
  return rightAnswer;
}
```

6.5 Number of trailing zeroes in N!

```
// Number of trailing Zeros in N! ::
ll zeros = 0;
for (ll i = 5; i <= n; i *= 5) {
  zeros += (n / i);
}
```

6.6 PBDS and Modular Arithmetic

```
//Policy based data-structure
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef tree< long long, null_type, less_equal<long long>,
        rb_tree_tag, tree_order_statistics_node_update >
        ordered_set;
//change ll to any data type
//less_equal for multiset increasing order
//less for set increasing order
//greater_equal for multiset decreasing order
//greater for set decreasing order
//cout<<*X.find_by_order(1)<<endl; // iterator to the k-th
     largest element
//cout<<X.order_of_key(-5)<<endl; // number of items in a
     set that are strictly smaller than our item

//Number theory related
const int MOD = 1e9+7;
int gcd ( int a, int b ) { return __gcd ( a, b ); }
int lcm ( int a, int b ) { return a * ( b / gcd ( a, b ) );
     }
inline void normal(int &a) { a %= MOD; (a < 0) && (a += MOD)
     ; }
inline int modMul(int a, int b) { a %= MOD, b %= MOD; normal
     (a), normal(b); return (a*b)%MOD; }
inline int modAdd(int a, int b) { a %= MOD, b %= MOD; normal
     (a), normal(b); return (a+b)%MOD; }
inline int modSub(int a, int b) { a %= MOD, b %= MOD; normal
     (a), normal(b); a -= b; normal(a); return a; }
inline int modPow(int b, int p) { int r = 1; while(p) { if(p
     &1) r = modMul(r, b); b = modMul(b, b); p >>= 1; }
     return r; }
inline int modInverse(int a) { return modPow(a, MOD-2); }
inline int modDiv(int a, int b) { return modMul(a,
     modInverse(b)); }
```

6.7 Prime Factors of N!

```
// Prime Factors of N!::
map<int, int>mp;
while (prime[i] <= n) {
  int x = n, j = 1, ans = 0;
  while (x > 0) {
 ans += x / prime[i];
 x /= prime[i];
  }
  mp[prime[i]] = ans;
  i++;
```

```
}
```

6.8 Product of Divisors (POD)

```
POD(n) = pow(n,NOD(n)/2); // product of divisors of n.
```

6.9 SNOD

```
int SNOD ( int n ){
  int sq = sqrt ( n );
  int ret = 0;
  for ( int i = 1; i <= sq; i++ ) {
 ret += ( n / i ) - i;
  }
  ret *= 2; // for a > b.
  ret += sq; // for a == b.
  return ret;
}
```

6.10 Space slicing (strings)

```
// Words in a string consists space:
void print () {
  string s, w;
  getline(cin, s);
  istringstream iS(s);
  while (iS >> w) {
    //w is the is the desired words.
    //it only removes space
    //',' , ';' , '?' these are not removed
    cout << w << endl;
  }
}
```

6.11 Sum of Divisors (SOD)

```
ll SOD (ll n) { // sum of divisors of n
  ll ret = 1;
  for ( auto p : prime ) {
    if ( 1LL * p * p > n ) break;
    if ( n % p == 0 ) {
      long long pwP = p;
      while ( n % p == 0 ) {
        pwP *= p;
        n /= p;
      }
      ret *= ( ( pwP - 1 ) / ( p - 1 ) );
    }
  }
  if ( n > 1 ) {
    ret *= ( n + 1 );
  }
  return ret;
}
```

6.12 Sum of every (I) - LCM(i, n)

```
// Ans = lcm(1,n) + lcm(2,n) + lcm(3,n) + lcm(4,n) + . + lcm
     (n,n).
ll phi[N],r[N];
ll totalLcm(ll n){
  ll ans = 0;
  for (int i = 1; i < mx; i++) {
    for (int j = i; j < mx; j += i) {
      r[j] += phi[i] * i;
    }
  }
  ans = r[n] + 1;
  ans *= n;
  ans /= 2;
  return ans;
}
```

7 Number Theory

7.1 Binary Exponentiation

```
/* binary_expo<int>(2, n - 1, m); */
template<typename T, typename X>
T binary_expo (T val, T power, X m) {
  T output = 1;
  while (power) {
    if (power & 1) {
      output = T((output * 1LL * val) % m);
    }
    val = (val * 1LL * val) % m;
    power >>= 1;
  }
  return output;
}
```

7.2 Binomial Coefficients (nCr)

```
/* bin_coeff<int> bcoef(N, M); */
/* bcoef.nCr(n, r); */
template <typename T>
struct bin_coeff {
  T n, m;
  vector<T> fact;
  void gen_fact () {
    fact[0] = fact[1] = 1;
    for (int i = 2; i <= n; ++i) {
      fact[i] = (1LL * fact[i - 1] * i) % m;
    }
  }
  bin_coeff (T n, T m) {
    this -> n = n;
    this -> m = m;
    fact.resize(n + 1);
    gen_fact();
  }
  T inv (T val, T power) {
    T output = 1;
    while (power) {
      if (power & 1) {
        output = T((output * 1LL * val) % m);
      }
      val = (val * 1LL * val) % m;
      power >>= 1;
    }
    return output;
  }
  T nCr (T N, T R) {
    return (fact[N] * 1LL * inv((fact[R] * 1LL * fact[N - R])
         % m, m - 2)) % m;
  }
};
```

7.3 Catalan Number

```
long long catalan[n + 1];
// Initialize first two values in table
catalan[0] = catalan[1] = 1;
// Fill entries in catalan[] using recursive formula
for (int i = 2; i <= n; i++) {
  catalan[i] = 0;
  for (int j = 0; j < i; j++) {
    catalan[i] += catalan[j] * catalan[i - j - 1];
  }
}
```

7.4 Chinese Remainder Theorem

```
struct Congruence {
  long long a, m;
};
long long chinese_remainder_theorem(vector<Congruence> const
                                      & congruences) {
  long long M = 1;
  for (auto const& congruence : congruences) {
    M *= congruence.m;
  }
  long long solution = 0;
  for (auto const& congruence : congruences) {
    long long a_i = congruence.a;
    long long M_i = M / congruence.m;
    long long N_i = mod_inv(M_i, congruence.m);
    solution = (solution + a_i * M_i % M * N_i) % M;
  }
  return solution;
}
```

7.5 Euler Totient (Precomputed)

```
/* Time complexity: O(n*log*log(n)) */
void phi_1_to_n(int n) {
  vector<int> phi(n + 1);
  for (int i = 0; i <= n; i++) {
    phi[i] = i;
  }
  for (int i = 2; i <= n; i++) {
    if (phi[i] == i) {
      for (int j = i; j <= n; j += i) {
        phi[j] -= phi[j] / i;
      }
    }
  }
}
```

7.6 Euler Totient (Single)

```
/* Time complexity: O(sqrt(n))
* Returns phi(n) */
int phi (int n) {
  int result = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      while (n % i == 0) {
        n /= i;
      }
      result -= result / i;
    }
  }
  if (n > 1) {
    result -= result / n;
  }
  return result;
}
```

7.7 Extended GCD

```
int gcd(int a, int b, int& x, int& y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  int x1, y1;
  int d = gcd(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
```

7.8 Fermats Primality Test

```
template <typename T>
bool fermat (T n, int iter=5) {
  if (n < 4) {
    return n == 2 or n == 3;
  }
  for (int i = 0; i < iter; i++) {
    T a = 2 + rand() % (n - 3);
    if (binary_expo<T>(a, n - 1, n) != 1) {
      return false;
    }
  }
  return true;
}
```

7.9 Lucas Theorem

```
// Lucas's Theorem calculates nCr % p in log(n). And if p <
    n and if p is not prime, multiple of two prime (mod =
    prime*prime)


// mint data-type is a custom data type which does mod own
    it's own.
// So, wherever "mint" has been used, we just need to use
    manual mod for the calculations when using it's values

#include<bits/stdc++.h>
using namespace std;

const int N = 1e6 + 3, mod = 1e6 + 3;
using ll = long long;

struct combi{
  int n; vector<mint> facts, finvs, invs;
  combi(int _n): n(_n), facts(_n), finvs(_n), invs(_n){
    facts[0] = finvs[0] = 1;
    invs[1] = 1;
    for (int i = 2; i < n; i++) invs[i] = invs[mod % i] * (-
        mod / i);
    for(int i = 1; i < n; i++){
      facts[i] = facts[i - 1] * i;
      finvs[i] = finvs[i - 1] * invs[i];
    }
  }
  inline mint fact(int n) { return facts[n]; }
  inline mint finv(int n) { return finvs[n]; }
  inline mint inv(int n) { return invs[n]; }
  inline mint ncr(int n, int k) { return n < k or k < 0 ? 0
      : facts[n] * finvs[k] * finvs[n-k]; }
};

combi C(N);

// returns nCr modulo mod where mod is a prime
// Complexity: log(n)
mint lucas(ll n, ll r) {
  if (r > n) return 0;
  if (n < mod) return C.ncr(n, r);
  return lucas(n / mod, r / mod) * lucas(n % mod, r % mod);
}

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
```

```
  cout << lucas(100000000, 2322) << '\n';
  return 0;
}
```

7.10 Matrix Exponentiation

```
// C++ program to find value of f(n) where f(n)
// is defined as,
// F(n) = F(n-1) + F(n-2) + F(n-3), n >= 3
// Base Cases :
// F(0) = 0, F(1) = 1, F(2) = 1
// Time Complexity: O(logN)
// Step 1: findNthTerm(n)
// A utility function to multiply two matrices
// a[][] and b[][]. Multiplication result is
// stored back in b[][]
void multiply(int a[3][3], int b[3][3]) {
// Creating an auxiliary matrix to store elements
// of the multiplication matrix
  int mul[3][3];
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      mul[i][j] = 0;
      for (int k = 0; k < 3; k++)
        mul[i][j] += a[i][k]*b[k][j];
    }
  }
// storing the multiplication result in a[][]
  for (int i=0; i<3; i++)
    for (int j=0; j<3; j++)
      a[i][j] = mul[i][j]; // Updating our matrix
}
// Function to compute F raise to power n-2.
int power(int F[3][3], int n) {
  int M[3][3] = {{1,1,1}, {1,0,0}, {0,1,0}};
// Multiply it with initial values i.e with
// F(0) = 0, F(1) = 1, F(2) = 1
  if (n==1)
    return F[0][0] + F[0][1];
  power(F, n/2);
  multiply(F, F);
  if (n%2 != 0)
    multiply(F, M);
// Multiply it with initial values i.e with
// F(0) = 0, F(1) = 1, F(2) = 1
  return F[0][0] + F[0][1] ;
}
// Return  nth  term of a series defined using below
// recurrence relation.
// f(n) is defined as
// f(n) = f(n-1) + f(n-2) + f(n-3), n>=3
// Base Cases :
// f(0) = 0, f(1) = 1, f(2) = 1
int findNthTerm(int n) {
  int F[3][3] = {{1,1,1}, {1,0,0}, {0,1,0}} ;
//Base cases
  if(n==0)
    return 0;
  if(n==1 || n==2)
    return 1;
  return power(F, n-2);
}
```

7.11 Mobius Function

```
for (i = 0; i < MU_MAX; i++) {
  mu[i] = 1;
}
for (i = 2; i <= sqroot; i++) {
  if (mu[i] == 1) {
// for each factor found, swap (+) and (-)
    for (j = i; j <= MU_MAX; j += i) {
      mu[j] *= (-1LL);
    }
// square factor = 0
    for (j = i * i; j <= MU_MAX; j += i * i) {
      mu[j] = 0;
    }
  }
}
```

7.12 Pollard Rho

```
#include<bits/stdc++.h>
using namespace std;

using ll = long long;
namespace PollardRho {
  mt19937 rnd(chrono::steady_clock::now().time_since_epoch()
      .count());
  const int P = 1e6 + 9;
  ll seq[P];
  int primes[P], spf[P];
  inline ll add_mod(ll x, ll y, ll m) {
    return (x += y) < m ? x : x - m;
  }
inline ll mul_mod(ll x, ll y, ll m) {
  ll res = __int128(x) * y % m;
  return res;
  // ll res = x * y - (ll)((long double)x * y / m + 0.5) *
      m;
  // return res < 0 ? res + m : res;
}
inline ll pow_mod(ll x, ll n, ll m) {
  ll res = 1 % m;
  for (; n; n >>= 1) {
    if (n & 1) res = mul_mod(res, x, m);
    x = mul_mod(x, x, m);
  }
  return res;
}
// O(it * (logn)^3), it = number of rounds performed
inline bool miller_rabin(ll n) {
  if (n <= 2 || (n & 1 ^ 1)) return (n == 2);
  if (n < P) return spf[n] == n;
  ll c, d, s = 0, r = n - 1;
  for (; !(r & 1); r >>= 1, s++) {}
  // each iteration is a round
  for (int i = 0; primes[i] < n && primes[i] < 32; i++) {
    c = pow_mod(primes[i], r, n);
    for (int j = 0; j < s; j++) {
      d = mul_mod(c, c, n);
      if (d == 1 && c != 1 && c != (n - 1)) return false;
      c = d;
    }
    if (c != 1) return false;
  }
  return true;
}
void init() {
  int cnt = 0;
  for (int i = 2; i < P; i++) {
    if (!spf[i]) primes[cnt++] = spf[i] = i;
    for (int j = 0, k; (k = i * primes[j]) < P; j++) {
      spf[k] = primes[j];
      if (spf[i] == spf[k]) break;
    }
  }
}
// returns O(n^(1/4))
ll pollard_rho(ll n) {
  while (1) {
    ll x = rnd() % n, y = x, c = rnd() % n, u = 1, v, t =
        0;
    ll *px = seq, *py = seq;
    while (1) {
```

```cpp
        *py++ = y = add_mod(mul_mod(y, y, n), c, n);
        *py++ = y = add_mod(mul_mod(y, y, n), c, n);
        if ((x = *px++) == y) break;
        v = u;
        u = mul_mod(u, abs(y - x), n);
        if (!u) return __gcd(v, n);
        if (++t == 32) {
          t = 0;
          if ((u = __gcd(u, n)) > 1 && u < n) return u;
        }
      }
      if (t && (u = __gcd(u, n)) > 1 && u < n) return u;
    }
  }
  vector<ll> factorize(ll n) {
    if (n == 1) return vector <ll>();
    if (miller_rabin(n)) return vector<ll> {n};
    vector <ll> v, w;
    while (n > 1 && n < P) {
      v.push_back(spf[n]);
      n /= spf[n];
    }
    if (n >= P) {
      ll x = pollard_rho(n);
      v = factorize(x);
      w = factorize(n / x);
      v.insert(v.end(), w.begin(), w.end());
    }
    return v;
  }
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  PollardRho::init();
  int t; cin >> t;
  while (t--) {
    ll n; cin >> n;
    auto f = PollardRho::factorize(n);
    sort(f.begin(), f.end());
    cout << f.size() << ' ';
    for (auto x: f) cout << x << ' '; cout << '\n';
  }
  return 0;
}
// https://judge.yosupo.jp/problem/factorize
```

7.13 Sum of The Number of Divisors in cbrt(n)

```cpp
#include<bits/stdc++.h>
using namespace std;

using uint32 = unsigned int;
using uint64 = unsigned long long;
using uint128 = __uint128_t;

// credit: zimpha
// compute \sum_{i=1}^{n} sigma0(i) in ~O(n^{1/3}) time.
// it is also equal to \sum_{i=1}^{n} floor(n / i)
// takes ~100 ms for n = 1e18
uint128 sum_sigma0(uint64 n) {
  auto out = [n] (uint64 x, uint32 y) {
    return x * y > n;
  };
  auto cut = [n] (uint64 x, uint32 dx, uint32 dy) {
    return uint128(x) * x * dy >= uint128(n) * dx;
  };
  const uint64 sn = sqrtl(n);
  const uint64 cn = pow(n, 0.34); //cbrtl(n);
  uint64 x = n / sn;
  uint32 y = n / x + 1;
  uint128 ret = 0;
  stack<pair<uint32, uint32>> st;
  st.emplace(1, 0);
  st.emplace(1, 1);
  while (true) {
    uint32 lx, ly;
    tie(lx, ly) = st.top();
    st.pop();
    while (out(x + lx, y - ly)) {
      ret += x * ly + uint64(ly + 1) * (lx - 1) / 2;
      x += lx, y -= ly;
    }
    if (y <= cn) break;
    uint32 rx = lx, ry = ly;
    while (true) {
      tie(lx, ly) = st.top();
      if (out(x + lx, y - ly)) break;
      rx = lx, ry = ly;
      st.pop();
    }
    while (true) {
      uint32 mx = lx + rx, my = ly + ry;
      if (out(x + mx, y - my)) {
        st.emplace(lx = mx, ly = my);
      }
      else {
        if (cut(x + mx, lx, ly)) break;
        rx = mx, ry = my;
      }
    }
  }
  for (--y; y > 0; --y) ret += n / y;
  return ret * 2 - sn * sn;
}

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int t; cin >> t;
  while (t--) {
    long long n; cin >> n;
    auto ans = sum_sigma0(n);
    string s = "";
    while (ans > 0) {
      s += char('0' + ans % 10);
      ans /= 10;
    }
    reverse(s.begin(), s.end());
    cout << s << '\n';
  }
  return 0;
}
// https://www.spoj.com/problems/DIVCNT1/en/
```

7.14 Trivial nCr

```cpp
/* Used when theres no mod used
* Time complexity: O(k)
* Step 1: ncr_triv<int>(n, r) */
template <typename T>
T ncr_triv (T n, T k) {
  T ncr = 1;
  if (n - k < k) {
    k = n - k;
  }
  for (T i = 0; i < k; ++i) {
    ncr *= (n - i);
    ncr /= (i + 1);
  }
  return ncr;
}
```

7.15 nCr Table

```
long long ncr[maxn][maxn] = {0};
const int mod = (int) 1e9 + 7;
void init () {
  ncr[0][0] = 1;
  for (int i = 1; i < maxn; i++) {
    ncr[i][0] = 1;
    for (int j = 1; j < i + 1; j++) {
      ncr[i][j] = (ncr[i - 1][j - 1] + ncr[i - 1][j]) % mod;
    }
  }
}
```

8 Strings

8.1 Aho Corasick

```
const int K = 26;
struct Vertex {
  int next[K];
  bool leaf = false;
  int p = -1;
  char pch;
  int link = -1;
  int go[K];
  Vertex(int p=-1, char ch= $ ) : p(p), pch(ch) {
    fill(begin(next), end(next), -1);
    fill(begin(go), end(go), -1);
  }
};
vector<Vertex> t(1);
void add_string(string const& s) {
  int v = 0;
  for (char ch : s) {
    int c = ch -  a  ;
    if (t[v].next[c] == -1) {
      t[v].next[c] = t.size();
      t.emplace_back(v, ch);
    }
    v = t[v].next[c];
  }
  t[v].leaf = true;
}
int go(int v, char ch);
int get_link(int v) {
  if (t[v].link == -1) {
    if (v == 0 || t[v].p == 0) {
      t[v].link = 0;
    } else {
      t[v].link = go(get_link(t[v].p), t[v].pch);
    }
  }
  return t[v].link;
}
int go(int v, char ch) {
  int c = ch -  a  ;
  if (t[v].go[c] == -1) {
    if (t[v].next[c] != -1) {
      t[v].go[c] = t[v].next[c];
    } else {
      t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
  }
  return t[v].go[c];
}
```

8.2 Knuth Morris Pratt

```
/* Total complexity: O(n + m) */
/* Application (string problems) : */
/*  1. Used to extract matched positions */
/*  2. Used to know if we have a match or not */
/* kmp<int> km(full_string, pattern_to_search_for); */
/* auto ids = km.pos(); */
template <typename T>
struct kmp {
  int n, m;
  string s, t;
  vector<T> tab;
  // Creating the prefix length table
  void proc () {
    int i = 0;
    for (int j = 1; j < m;) {
      if (t[i] == t[j]) {
        tab[j] = i + 1;
        i += 1, j += 1;
      } else {
        if (i) {
          i = tab[i - 1];
        } else {
          j += 1;
        }
      }
    }
  }
  // initializing everything
  kmp (string s, string t) {
    this -> s = s;
    this -> t = t;
    n = (T) s.size();
    m = (T) t.size();
    tab.assign(m, 0);
    proc();
  }
  // Returns all the starting positions where we have a
      match
  // If we have a match we continue,
  // Otherwise, we look in the previous index of the table
      to save time.
  vector<T> pos () {
    int i = 0;
    int j = 0;
    vector<T> ids;
    while (i < n) {
      if (s[i] == t[j]) {
        i += 1, j += 1;
      } else {
        if (j) {
          j = tab[j - 1];
        } else {
          i += 1;
        }
      }
      // If pattern found take the index
      if (j == m) {
        ids.push_back(i - m);
        j = tab[j - 1];
      }
    }
    return ids;
  }
};
```

8.3 Manachers

```
/* Time complexity: O(N) */
/* While solving this problem always try to solve this on
    the basis of the generated answer it is returning */
/* Which means, always try to solve on the basis of
    converted string -> #a#b#a# */
/* Return the length of a palindrome from left side,
    defining (i) as the middle of that palindrome*/
/* manachers<int> man(s); */
/* auto ans = man.ret_ans(); */
template <typename T>
```

```
struct manachers {
  int n;
  vector<int> p;
  void manac_odd (string s) {
    n = (int) s.size();
    s = "(" + s + ")";
    p.assign(n + 2, 0);
    int l = 1, r = 1;
    for (int i = 1; i <= n; ++i) {
      p[i] = max(0, min(r - i, p[l + (r - i)]));
      while (s[i - p[i]] == s[i + p[i]]) {
        p[i] += 1;
      }
      if (r < i + p[i]) {
        l = i - p[i];
        r = i + p[i];
      }
    }
  }
  manachers (string t) {
    string s = "";
    for (auto c : t) {
      s += string("#") + c;
    }
    manac_odd(s + "#");
  }
  vector<T> ret_ans () {
    return vector<T>(p.begin() + 1, p.end() - 1);
  }
};
```

8.4 String Hashing (double)

```
#include <bits/stdc++.h>
using namespace std;
const int mod = (int) 1e9 + 7;
int add_mod (int a, int b) {
  int res = (a + b) % mod;
  res += (res < 0 ? mod : 0);
  return res;
}
int sub_mod (int a, int b) {
  int res = (a - b) % mod;
  res += (res < 0 ? mod : 0);
  return res;
}
int mult_mod (int a, int b) {
  int res = (a * 1LL * b) % mod;
  res += (res < 0 ? mod : 0);
  return res;
}
template<typename T, typename X>
T binary_expo (T val, T power, X m) {
  T output = 1;
  while (power) {
    if (power & 1) {
      output = T((output * 1LL * val) % m);
    }
    val = (val * 1LL * val) % m;
    power >>= 1;
  }
  return output;
}
int main () {
  /* ios::sync_with_stdio(false); */
  /* cin.tie(0); */
  string s;
  cin >> s;
  /* This block of code completely double hashes the string
  S */
  int p1 = 31, p2 = 53;
  int n = (int) s.size();
  vector<int> pref_hash1(n);
  vector<int> pref_hash2(n);
  pref_hash1[0] = (s[0] -   a  ) + 1;
  pref_hash2[0] = (s[0] -   a  ) + 1;
  /* The inverse array is needed to substract the substring
  s hash */
  vector<int> p_pow1(n), inv1(n);
  vector<int> p_pow2(n), inv2(n);
  p_pow1[0] = inv1[0] = 1;
  p_pow2[0] = inv2[0] = 1;
  for (int i = 1; i < n; i++) {
    p_pow1[i] = (p_pow1[i - 1] * 1LL * p1) % mod;
    p_pow2[i] = (p_pow2[i - 1] * 1LL * p2) % mod;
    inv1[i] = binary_expo<int>(p_pow1[i], mod - 2, mod);
    inv2[i] = binary_expo<int>(p_pow2[i], mod - 2, mod);
    pref_hash1[i] = add_mod(pref_hash1[i - 1], mult_mod((s[i]
                          -    a    + 1), p_pow1[i]));
    pref_hash2[i] = add_mod(pref_hash2[i - 1], mult_mod((s[i]
                          -    a    + 1), p_pow2[i]));
  }
  /* This function returns the hash-1 of the substring of
  string s
  * Moreover, this function also uses 0 based indesing */
  auto substring_hash1 = [&] (int l, int r) {
    int res = pref_hash1[r];
    if (0 < l) {
      res -= pref_hash1[l - 1];
    }
    res = mult_mod(res, inv1[l]);
    return res;
  };
  /* This function returns the hash-1 of the substring of
  string s
  * Moreover, this function also uses 0 based indesing */
  auto substring_hash2 = [&] (int l, int r) {
    int res = pref_hash2[r];
    if (0 < l) {
      res -= pref_hash2[l - 1];
    }
    res = mult_mod(res, inv2[l]);
    return res;
  };
  /* This block of code quering for each substring hash*/
  int q;
  cin >> q;
  while (q--) {
    int l, r;
    cin >> l >> r;
    --l, --r;
    cout << substring_hash1(l, r) << \ n ;
    cout << substring_hash2(l, r) << \ n ;
  }
  return 0;
}
```

8.5 Suffix Array

```
// Always set this to max length
#define N ((int)15e2+5)
/// suffixarray
int cmp_for_sa(int*r,int a,int b,int l) {
  return(r[a]==r[b])&&(r[a+l]==r[b+l]);
}
int wa[N],wb[N],wws[N],wv[N],rnk[N],lcp[N],sa[N],Data[N];
void DA (int*r,int*sa,int n,int m) {
  int i,j,p,*x=wa,*y=wb,*t;
  for(i=0; i<m; i++)
    wws[i]=0;
  for(i=0; i<n; i++)
    wws[x[i]=r[i]]++;
  for(i=1; i<m; i++)
    wws[i]+=wws[i-1];
  for(i=n-1; i>=0; i--)
    sa[--wws[x[i]]]=i;
  for(j=1,p=1; p<n; j*=2,m=p) {
    for(p=0,i=n-j; i<n; i++)
```

```
      y[p++]=i;
    for(i=0; i<n; i++)
      if(sa[i]>=j)
        y[p++]=sa[i]-j;
    for(i=0; i<n; i++)
      wv[i]=x[y[i]];
    for(i=0; i<m; i++)
      wws[i]=0;
    for(i=0; i<n; i++)
      wws[wv[i]]++;
    for(i=1; i<m; i++)
      wws[i]+=wws[i-1];
    for(i=n-1; i>=0; i--)
      sa[--wws[wv[i]]]=y[i];
    for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1; i<n; i++)
      x[sa[i]] = cmp_for_sa(y,sa[i-1],sa[i],j) ? p-1 : p++;
  }
}
void cal_lcp(int* r,int* sa,int n) {
  int i,j,k=0;
  for(i=1; i<=n; i++)
    rnk[sa[i]]=i;
  for(i=0; i<n; lcp[rnk[i++]]=k)
    for(k?k--:0,j=sa[rnk[i]-1]; r[i+k]==r[j+k]; k++);
}
void suffix_array(char* A) {
  int n=strlen(A);
  for(int i=0; i<=128; i++) {
    wa[i]=wb[i]=wws[i]=wv[i]=rnk[i]=lcp[i]=sa[i]=Data[i]=0;
  }
  for(int i=0; i<=n; i++) {
    wa[i]=wb[i]=wws[i]=wv[i]=rnk[i]=lcp[i]=sa[i]=Data[i]=0;
    if(i<n)
      Data[i]=A[i];
  }
  DA(Data,sa,n+1,128);
  cal_lcp(Data,sa,n);
// transforming it into 0-based SA
  for(int i=0; i<n; i++) {
    sa[i]=sa[i+1];
    lcp[i]=lcp[i+2];
    rnk[i]--;
  }
}
int main() {
  char str[50];
  scanf("%s", str);
  suffix_array(str);
  return 0;
}
```

8.6 Trie

```
/* Time complexity (construction): */
/*   * number of nodes. Which depends on matched prefix. */
/*   * the more prefix-match, the better. */
/* Time complexity (per query): */
/*   * length of the asked string */
/* Trie trie; */
/* trie.insert(s); */
/* cout << (trie.search(t) ? "YES\n" : "NO\n") << '\n'; */
/* trie.del(); */
/* Note: useful for searching a string is present or not */
struct Trie {
  struct node {
    bool endmark;
 /* Change the size to (10) if working with digits */
    node* next[26];
    node () {
      endmark = false;
   /* Change the limit to 10 if working with digits */
      for (int i = 0; i < 26; ++i) {
        next[i] = NULL;
      }
    }
  } * root;
  /* Trie tri; */
  Trie () {
    root = new node();
  }
  /* tri.insert(s); */
  /* inserts a string the the Trie */
  void insert (string s) {
    node* curr = root;
    for (auto ch : s) {
      /* change 'a' according to the problem statement */
      int id = ch - 'a';
      if (curr -> next[id] == NULL) {
        curr -> next[id] = new node();
      }
      curr = curr -> next[id];
    }
    curr -> endmark = true;
  }
  /* return if a string is present in the list or not */
  /* cout << (tri.search(t) ? "YES\n" : "NO\n") << '\n'; */
  bool search (string s) {
    node* curr = root;
    for (auto ch : s) {
      /* change 'a' according to the problem statement */
      int id = ch - 'a';
      if (curr -> next[id] == NULL) {
        return false;
      }
      curr = curr -> next[id];
    }
    return curr -> endmark;
  }
  void del_node (node* curr) {
 /* Change the limit to 10 if working with digits */
    for (int i = 0; i < 26; ++i) {
      if (curr -> next[i]) {
        del_node(curr -> next[i]);
      }
    }
    delete(curr);
  }
  /* tri.del(); */
  /* deletes, all the nodes. Useful in reducing memory */
  void del () {
    del_node(root);
  }
};
```

Lemma’s OR Concepts

1. If d is a divisor of n, then there are Phi(n/d) numbers i <= n for which gcd(i,n)=d
2. An integer which is divisible by 3, the sum of its digits is also divisible by 3.
3. Only the power of twos has even divisors. Otherwise, every number has at least 1 odd divisor.
4. Sum of an arithmetic progression is = $a_l + (a_l+1) + \ldots + a_r = .5 * (a_l+a_r) * (r-l+1)$.
5. The number of odd elements in the n^{th} row of the Pascal Triangle, Answer is 2^x, where x is the number of 1 bit in the binary representation of n (pop-count).
6. An unimodal permutation is a permutation that increases up to a certain point following which it starts decreasing. The number of unimodal permutations of a permutation is 2^(n - 1).

Tricks AND Techniques

1. Use llabs() for long long
2. Use __lg(n) instead of log2(n)
3. Use sqrtl(n) instead of sqrt(n)
4. The last element in a multiset/set can be accessed using st.rbegin() or st.end() -1

Topics to TRY when STUCK

1. Look at the constraints
2. Pigeon-hole principle
3. Binary Search
4. Write brute force for small constraints (to find patterns or stress testing)
5. Equation formulating (left/right changing)
6. Sliding window | Two pointer
7. Simulate the test cases on the notebook for pattern
8. Solve the reversed problem

When facing ERROR

1. Check data type
2. Function calling error
3. Clearing vectors/containers
4. Array out of bound/overflow
5. Mod or dividing by zero
6. Look for corner cases for small input