Competitive Programming Notebook

Programadores Roblox

Contents

1 String

1.1 Trie

```cpp
// Trie por array
// InserÃ§Ãčo, busca e consulta de prefixo em O(N)

int trie[MAXN][26];
int tot_nos = 0;
vector<bool> acaba(MAXN, false);
vector<int> contador(MAXN, 0);

void insere(string s) {
    int no = 0;
    for(auto &c : s) {
        if(trie[no][c - 'a'] == 0) {
            trie[no][c - 'a'] = ++tot_nos;
        }
        no = trie[no][c - 'a'];
        contador[no]++;
    }
    acaba[no] = true;
}

bool busca(string s) {
    int no = 0;
    for(auto &c : s) {
        if(trie[no][c - 'a'] == 0) {
            return false;
        }
        no = trie[no][c - 'a'];
    }
    return acaba[no];
}

int isPref(string s) {
    int no = 0;
    for(auto &c : s) {
        if(trie[no][c - 'a'] == 0){
            return -1;
        }
        no = trie[no][c - 'a'];
    }
    return contador[no];
}
```

1.2 Countpermutations

```cpp
// Returns the number of distinct permutations
// that are lexicographically less than the string t
// using the provided frequency (freq) of the
    characters
// O(n*freq.size())
int countPermLess(vector<int> freq, const string &t)
    {
    int n = t.size();
    int ans = 0;

    vector<int> fact(n + 1, 1), invfact(n + 1, 1);
    for (int i = 1; i <= n; i++)
        fact[i] = (fact[i - 1] * i) % MOD;
    invfact[n] = fexp(fact[n], MOD - 2, MOD);
    for (int i = n - 1; i >= 0; i--)
        invfact[i] = (invfact[i + 1] * (i + 1)) % MOD
    ;

    // For each position in t, try placing a letter
    smaller than t[i] that is in freq
    for (int i = 0; i < n; i++) {
        for (char c = 'a'; c < t[i]; c++) {
            if (freq[c - 'a'] > 0) {
                freq[c - 'a']--;
```

```cpp
                int ways = fact[n - i - 1];
                for (int f : freq)
                    ways = (ways * invfact[f]) % MOD;
                ans = (ans + ways) % MOD;
                freq[c - 'a']++;
            }
        }
        if (freq[t[i] - 'a'] == 0) break;
        freq[t[i] - 'a']--;
    }
    return ans;
}
```

1.3 Trie Ponteiros

```cpp
// Trie por ponteiros
// InserÃ§Ãčo, busca e consulta de prefixo em O(N)

struct Node {
    Node *filhos[26] = {};
    bool acaba = false;
    int contador = 0;
};

void insere(string s, Node *raiz) {
    Node *cur = raiz;
    for(auto &c : s) {
        cur->contador++;
        if(cur->filhos[c - 'a'] != NULL) {
            cur = cur->filhos[c - 'a'];
            continue;
        }
        cur->filhos[c - 'a'] = new Node();
        cur = cur->filhos[c - 'a'];
    }
    cur->contador++;
    cur->acaba = true;
}

bool busca(string s, Node *raiz) {
    Node *cur = raiz;
    for(auto &c : s) {
        if (cur->filhos[c - 'a'] != NULL) {
            cur = cur->filhos[c - 'a'];
            continue;
        }
        return false;
    }
    return cur->acaba;
}

// Retorna se Ãľ prefixo e quantas strings tem s como
     prefixo
int isPref(string s, Node *raiz) {
    Node *cur = raiz;
    for(auto &c : s) {
        if (cur->filhos[c - 'a'] != NULL) {
            cur = cur->filhos[c - 'a'];
            continue;
        }
        return -1;
    }
    return cur->contador;
}
```

1.4 Z Function

```cpp
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
```

```
6          if(i < r) {
7              z[i] = min(r - i, z[i - 1]);
8          }
9          while(i + z[i] < n && s[z[i]] == s[i + z[i]])
    {
10              z[i]++;
11          }
12          if(i + z[i] > r) {
13              l = i;
14              r = i + z[i];
15          }
16      }
17      return z;
18 }
```

1.5 Hashing

```
1  // String Hash template
2  // constructor(s) - O(|s|)
3  // query(l, r) - returns the hash of the range [l,r]
       from left to right - O(1)
4  // query_inv(l, r) from right to left - O(1)
5  // patrocinado por tiagodfs
6
7  struct Hash {
8      const int X = 2147483647;
9      const int MOD = 1e9+7;
10     int n; string s;
11     vector<int> h, hi, p;
12     Hash() {}
13     Hash(string s): s(s), n(s.size()), h(n), hi(n), p
       (n) {
14         for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1)
       % MOD;
15         for (int i=0;i<n;i++)
16             h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
17         for (int i=n-1;i>=0;i--)
18             hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X)
       % MOD;
19     }
20     int query(int l, int r) {
21         int hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
       0));
22         return hash < 0 ? hash + MOD : hash;
23     }
24     int query_inv(int l, int r) {
25         int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
       +1] % MOD : 0));
26         return hash < 0 ? hash + MOD : hash;
27     }
28 };
```

1.6 Kmp

```
1  vector<int> kmp(string s) {
2      int n = (int)s.length();
3      vector<int> p(n+1);
4      p[0] = -1;
5      for (int i = 1; i < n; i++) {
6          int j = p[i-1];
7          while (j >= 0 && s[j] != s[i-1])
8              j = p[j-1];
9          p[i] = j+1;
10     }
11     return p;
12 }
```

1.7 Lcs

```
1  int lcs(string &s1, string &s2) {
2      int m = s1.size();
3      int n = s2.size();
```

```
4
5      vector<vector<int>> dp(m + 1, vector<int>(n + 1,
       0));
6
7      for (int i = 1; i <= m; ++i) {
8          for (int j = 1; j <= n; ++j) {
9              if (s1[i - 1] == s2[j - 1])
10                 dp[i][j] = dp[i - 1][j - 1] + 1;
11             else
12                 dp[i][j] = max(dp[i - 1][j], dp[i][j
       - 1]);
13         }
14     }
15
16     return dp[m][n];
17 }
```

2 String copy

2.1 Countpermutations

```
1  // Returns the number of distinct permutations
2  // that are lexicographically less than the string t
3  // using the provided frequency (freq) of the
       characters
4  // O(n*freq.size())
5  int countPermLess(vector<int> freq, const string &t)
       {
6      int n = t.size();
7      int ans = 0;
8
9      vector<int> fact(n + 1, 1), invfact(n + 1, 1);
10     for (int i = 1; i <= n; i++)
11         fact[i] = (fact[i - 1] * i) % MOD;
12     invfact[n] = fexp(fact[n], MOD - 2, MOD);
13     for (int i = n - 1; i >= 0; i--)
14         invfact[i] = (invfact[i + 1] * (i + 1)) % MOD
       ;
15
16     // For each position in t, try placing a letter
       smaller than t[i] that is in freq
17     for (int i = 0; i < n; i++) {
18         for (char c = 'a'; c < t[i]; c++) {
19             if (freq[c - 'a'] > 0) {
20                 freq[c - 'a']--;
21                 int ways = fact[n - i - 1];
22                 for (int f : freq)
23                     ways = (ways * invfact[f]) % MOD;
24                 ans = (ans + ways) % MOD;
25                 freq[c - 'a']++;
26             }
27         }
28         if (freq[t[i] - 'a'] == 0) break;
29         freq[t[i] - 'a']--;
30     }
31     return ans;
32 }
```

2.2 Trie Ponteiros

```
1  // Trie por ponteiros
2  // Inserção, busca e consulta de prefixo em O(N)
3
4  struct Node {
5      Node *filhos[26] = {};
6      bool acaba = false;
7      int contador = 0;
8  };
9
10 void insere(string s, Node *raiz) {
11     Node *cur = raiz;
```

```
12      for(auto &c : s) {
13          cur->contador++;
14          if(cur->filhos[c - 'a'] != NULL) {
15              cur = cur->filhos[c - 'a'];
16              continue;
17          }
18          cur->filhos[c - 'a'] = new Node();
19          cur = cur->filhos[c - 'a'];
20      }
21      cur->contador++;
22      cur->acaba = true;
23  }
24
25  bool busca(string s, Node *raiz) {
26      Node *cur = raiz;
27      for(auto &c : s) {
28          if (cur->filhos[c - 'a'] != NULL) {
29              cur = cur->filhos[c - 'a'];
30              continue;
31          }
32          return false;
33      }
34      return cur->acaba;
35  }
36
37  // Retorna se Ãľ prefixo e quantas strings tem s como
        prefixo
38  int isPref(string s, Node *raiz) {
39      Node *cur = raiz;
40      for(auto &c : s) {
41          if (cur->filhos[c - 'a'] != NULL) {
42              cur = cur->filhos[c - 'a'];
43              continue;
44          }
45          return -1;
46      }
47      return cur->contador;
48  }
```

2.3 Z Function

```
1  vector<int> z_function(string s) {
2      int n = s.size();
3      vector<int> z(n);
4      int l = 0, r = 0;
5      for(int i = 1; i < n; i++) {
6          if(i < r) {
7              z[i] = min(r - i, z[i - l]);
8          }
9          while(i + z[i] < n && s[z[i]] == s[i + z[i]])
    {
10             z[i]++;
11         }
12         if(i + z[i] > r) {
13             l = i;
14             r = i + z[i];
15         }
16     }
17     return z;
18  }
```

2.4 Hashing

```
1  // String Hash template
2  // constructor(s) - O(|s|)
3  // query(l, r) - returns the hash of the range [l,r]
      from left to right - O(1)
4  // query_inv(l, r) from right to left - O(1)
5  // patrocinado por tiagodfs
6
7  struct Hash {
8      const int X = 2147483647;
9      const int MOD = 1e9+7;
10     int n; string s;
11     vector<int> h, hi, p;
12     Hash() {}
13     Hash(string s): s(s), n(s.size()), h(n), hi(n), p
    (n) {
14         for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1)
    % MOD;
15         for (int i=0;i<n;i++)
16             h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
17         for (int i=n-1;i>=0;i--)
18             hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X)
    % MOD;
19     }
20     int query(int l, int r) {
21         int hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD :
     0));
22         return hash < 0 ? hash + MOD : hash;
23     }
24     int query_inv(int l, int r) {
25         int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
    +1] % MOD : 0));
26         return hash < 0 ? hash + MOD : hash;
27     }
28  };
```

2.5 Kmp

```
1  vector<int> kmp(string s) {
2      int n = (int)s.length();
3      vector<int> p(n+1);
4      p[0] = -1;
5      for (int i = 1; i < n; i++) {
6          int j = p[i-1];
7          while (j >= 0 && s[j] != s[i-1])
8              j = p[j-1];
9          p[i] = j+1;
10     }
11     return p;
12  }
```

2.6 Lcs

```
1  int lcs(string &s1, string &s2) {
2      int m = s1.size();
3      int n = s2.size();
4
5      vector<vector<int>> dp(m + 1, vector<int>(n + 1,
    0));
6
7      for (int i = 1; i <= m; ++i) {
8          for (int j = 1; j <= n; ++j) {
9              if (s1[i - 1] == s2[j - 1])
10                 dp[i][j] = dp[i - 1][j - 1] + 1;
11             else
12                 dp[i][j] = max(dp[i - 1][j], dp[i][j
    - 1]);
13         }
14     }
15
16     return dp[m][n];
17  }
```

3 DS

3.1 Segtree Gcd

```
1  int gcd(int a, int b) {
2      if (b == 0)
3          return a;
4      return gcd(b, a % b);
```

```
5  }
6
7  class SegmentTreeGCD {
8  private:
9      vector<int> tree;
10     int n;
11
12     void build(const vector<int>& arr, int node, int
       start, int end) {
13         if (start == end) {
14             tree[node] = arr[start];
15         } else {
16             int mid = (start + end) / 2;
17             build(arr, 2 * node + 1, start, mid);
18             build(arr, 2 * node + 2, mid + 1, end);
19             tree[node] = gcd(tree[2 * node + 1], tree
       [2 * node + 2]);
20         }
21     }
22
23     void update(int node, int start, int end, int idx
       , int value) {
24         if (start == end) {
25             tree[node] = value;
26         } else {
27             int mid = (start + end) / 2;
28             if (idx <= mid) {
29                 update(2 * node + 1, start, mid, idx,
        value);
30             } else {
31                 update(2 * node + 2, mid + 1, end,
       idx, value);
32             }
33             tree[node] = gcd(tree[2 * node + 1], tree
       [2 * node + 2]);
34         }
35     }
36
37     int query(int node, int start, int end, int l,
       int r) {
38         if (r < start || l > end) {
39             return 0;
40         }
41         if (l <= start && end <= r) {
42             return tree[node];
43         }
44         int mid = (start + end) / 2;
45         int left_gcd = query(2 * node + 1, start, mid
       , l, r);
46         int right_gcd = query(2 * node + 2, mid + 1,
       end, l, r);
47         return gcd(left_gcd, right_gcd);
48     }
49
50 public:
51     SegmentTreeGCD(const vector<int>& arr) {
52         n = arr.size();
53         tree.resize(4 * n);
54         build(arr, 0, 0, n - 1);
55     }
56     void update(int idx, int value) {
57         update(0, 0, n - 1, idx, value);
58     }
59     int query(int l, int r) {
60         return query(0, 0, n - 1, l, r);
61     }
62 };
```

3.2   Bit

```
1  class BIT {
2      vector<int> bit;
3      int n;
4      int sum(int idx) {
5          int result = 0;
6          while (idx > 0) {
7              result += bit[idx];
8              idx -= idx & -idx;
9          }
10         return result;
11     }
12
13 public:
14     BIT(int size) {
15         n = size;
16         bit.assign(n + 1, 0);   // BIT indexada em 1
17     }
18     void update(int idx, int delta) {
19         while (idx <= n) {
20             bit[idx] += delta;
21             idx += idx & -idx;
22         }
23     }
24     int query(int idx) {
25         return sum(idx);
26     }
27     int range_query(int l, int r) {
28         return sum(r) - sum(l - 1);
29     }
30 };
31
32 BIT fenwick(n);
33 for(int i = 1; i <= n; i++) {
34     fenwick.update(i, arr[i]);
35 }
```

3.3   Psum 2d

```
1  // retangulo retorna a psum2d do intervalo inclusivo
2  vector<vector<int>> psum(n+1, vector<int>(m+1, 0));
3
4  for (int i=1; i<n+1; i++){
5      for (int j=1; j<m+1; j++){
6          cin >> psum[i][j];
7          psum[i][j] += psum[i-1][j]+psum[i][j-1]-psum[
       i-1][j-1];
8      }
9  }
10
11 // y1 eh variavel reservada
12 int retangulo(int x1, int yy1, int x2, int yy2){
13     x2 = min(x2, n), yy2 = min(yy2, m);
14     x1 = max(0LL, x1-1), yy1 = max(0LL, yy1-1);
15
16     return psum[x2][yy2]-psum[x1][yy2]-psum[x2][yy1]+
       psum[x1][yy1];
17 }
```

3.4   Ordered Set E Map

```
1
2  #include<ext/pb_ds/assoc_container.hpp>
3  #include<ext/pb_ds/tree_policy.hpp>
4  using namespace __gnu_pbds;
5  using namespace std;
6
7  template<typename T> using ordered_multiset = tree<T,
        null_type, less_equal<T>, rb_tree_tag,
        tree_order_statistics_node_update>;
8  template <typename T> using o_set = tree<T, null_type
       , less<T>, rb_tree_tag,
        tree_order_statistics_node_update>;
9  template <typename T, typename R> using o_map = tree<
       T, R, less<T>, rb_tree_tag,
        tree_order_statistics_node_update>;
```

```
10
11   int main() {
12     int i, j, k, n, m;
13     o_set<int>st;
14     st.insert(1);
15     st.insert(2);
16     cout << *st.find_by_order(0) << endl; /// k-esimo
          elemento
17     cout << st.order_of_key(2) << endl; ///numero de
          elementos menores que k
18     o_map<int, int>mp;
19     mp.insert({1, 10});
20     mp.insert({2, 20});
21     cout << mp.find_by_order(0)->second << endl; /// k-
          esimo elemento
22     cout << mp.order_of_key(2) << endl; /// numero de
          elementos (chave) menores que k
23     return 0;
24   }
```

3.5 Dsu

```
1    struct DSU {
2        vector<int> par, rank, sz;
3        int c;
4        DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n +
         1, 1), c(n) {
5            for (int i = 1; i <= n; ++i) par[i] = i;
6        }
7        int find(int i) {
8            return (par[i] == i ? i : (par[i] = find(par[
             i])));
9        }
10       bool same(int i, int j) {
11           return find(i) == find(j);
12       }
13       int get_size(int i) {
14           return sz[find(i)];
15       }
16       int count() {
17           return c;  // quantos componentes conexos
18       }
19       int merge(int i, int j) {
20           if ((i = find(i)) == (j = find(j))) return
             -1;
21           else --c;
22           if (rank[i] > rank[j]) swap(i, j);
23           par[i] = j;
24           sz[j] += sz[i];
25           if (rank[i] == rank[j]) rank[j]++;
26           return j;
27       }
28   };
```

3.6 Segtree Iterativa

```
1    // Exemplo de uso:
2    // SegTree<int> st(vetor);
3    // range query e point update
4
5    template <typename T>
6    struct SegTree {
7        int n;
8        vector<T> tree;
9        T neutral_value = 0;
10       T combine(T a, T b) {
11           return a + b;
12       }
13
14       SegTree(const vector<T>& data) {
15           n = data.size();
16           tree.resize(2 * n, neutral_value);
```

```
17
18           for (int i = 0; i < n; i++)
19               tree[n + i] = data[i];
20
21           for (int i = n - 1; i > 0; --i)
22               tree[i] = combine(tree[i * 2], tree[i * 2
             + 1]);
23       }
24
25       T range_query(int l, int r) {
26           T result = neutral_value;
27
28           for (l += n, r += n + 1; l < r; l >>= 1, r
             >>= 1) {
29               if (l & 1) result = combine(result, tree[
             l++]);
30               if (r & 1) result = combine(result, tree
             [--r]);
31           }
32
33           return result;
34       }
35
36       void update(int pos, T new_val) {
37           tree[pos += n] = new_val;
38
39           for (pos >>= 1; pos > 0; pos >>= 1)
40               tree[pos] = combine(tree[2 * pos], tree[2
             * pos + 1]);
41       }
42   };
```

3.7 Segtree Sum

```
1    struct SegTree {
2        ll merge(ll a, ll b) { return a + b; }
3        const ll neutral = 0;
4        int n;
5        vector<ll> t, lazy;
6        vector<bool> replace;
7        inline int lc(int p) { return p * 2; }
8        inline int rc(int p) { return p * 2 + 1; }
9        void push(int p, int l, int r) {
10           if (replace[p]) {
11               t[p] = lazy[p] * (r - l + 1);
12               if (l != r) {
13                   lazy[lc(p)] = lazy[p];
14                   lazy[rc(p)] = lazy[p];
15                   replace[lc(p)] = true;
16                   replace[rc(p)] = true;
17               }
18           } else if (lazy[p] != 0) {
19               t[p] += lazy[p] * (r - l + 1);
20               if (l != r) {
21                   lazy[lc(p)] += lazy[p];
22                   lazy[rc(p)] += lazy[p];
23               }
24           }
25           replace[p] = false;
26           lazy[p] = 0;
27       }
28       void build(int p, int l, int r, const vector<ll>
         &v) {
29           if (l == r) {
30               t[p] = v[l];
31           } else {
32               int mid = (l + r) / 2;
33               build(lc(p), l, mid, v);
34               build(rc(p), mid + 1, r, v);
35               t[p] = merge(t[lc(p)], t[rc(p)]);
36           }
37       }
38       void build(int _n) {
```

```
39              n = _n;
40              t.assign(n * 4, neutral);
41              lazy.assign(n * 4, 0);
42              replace.assign(n * 4, false);
43          }
44      void build(const vector<ll> &v) {
45              n = (int)v.size();
46              t.assign(n * 4, neutral);
47              lazy.assign(n * 4, 0);
48              replace.assign(n * 4, false);
49              build(1, 0, n - 1, v);
50          }
51      void build(ll *bg, ll *en) {
52              build(vector<ll>(bg, en));
53          }
54      ll query(int p, int l, int r, int L, int R) {
55              push(p, l, r);
56              if (l > R || r < L) return neutral;
57              if (l >= L && r <= R) return t[p];
58              int mid = (l + r) / 2;
59              auto ql = query(lc(p), l, mid, L, R);
60              auto qr = query(rc(p), mid + 1, r, L, R);
61              return merge(ql, qr);
62          }
63      ll query(int l, int r) { return query(1, 0, n -
    1, l, r); }
64      void update(int p, int l, int r, int L, int R, ll
     val, bool repl = 0) {
65              push(p, l, r);
66              if (l > R || r < L) return;
67              if (l >= L && r <= R) {
68                  lazy[p] = val;
69                  replace[p] = repl;
70                  push(p, l, r);
71              } else {
72                  int mid = (l + r) / 2;
73                  update(lc(p), l, mid, L, R, val, repl);
74                  update(rc(p), mid + 1, r, L, R, val, repl
    );
75                  t[p] = merge(t[lc(p)], t[rc(p)]);
76              }
77          }
78      void sumUpdate(int l, int r, ll val) { update(1,
    0, n - 1, l, r, val, 0); }
79      void assignUpdate(int l, int r, ll val) { update
    (1, 0, n - 1, l, r, val, 1); }
80 } segsum;
```

4 Primitives

5 Geometry

5.1 Point Location

```
1
2 int32_t main(){
3      sws;
4
5      int t; cin >> t;
6
7      while(t--){
8
9          int x1, y1, x2, y2, x3, y3; cin >> x1 >> y1
    >> x2 >> y2 >> x3 >> y3;
10
11          int deltax1 = (x1-x2), deltay1 = (y1-y2);
12
13          int compx = (x1-x3), compy = (y1-y3);
14
15          int ans = (deltax1*compy) - (compx*deltay1);
16
```

```
17          if(ans == 0){cout << "TOUCH\n"; continue;}
18          if(ans < 0){cout << "RIGHT\n"; continue;}
19          if(ans > 0){cout << "LEFT\n"; continue;}
20      }
21      return 0;
22 }
```

5.2 Convex Hull

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 #define int long long
5 typedef int cod;
6
7 struct point
8 {
9      cod x,y;
10      point(cod x = 0, cod y = 0): x(x), y(y)
11      {}
12
13      double modulo()
14      {
15          return sqrt(x*x + y*y);
16      }
17
18      point operator+(point o)
19      {
20          return point(x+o.x, y+o.y);
21      }
22      point operator-(point o)
23      {
24          return point(x - o.x , y - o.y);
25      }
26      point operator*(cod t)
27      {
28          return point(x*t, y*t);
29      }
30      point operator/(cod t)
31      {
32          return point(x/t, y/t);
33      }
34
35      cod operator*(point o)
36      {
37          return x*o.x + y*o.y;
38      }
39      cod operator^(point o)
40      {
41          return x*o.y - y * o.x;
42      }
43      bool operator<(point o)
44      {
45          if( x != o.x) return x < o.x;
46          return y < o.y;
47      }
48
49 };
50
51 int ccw(point p1, point p2, point p3)
52 {
53      cod cross = (p2-p1) ^ (p3-p1);
54      if(cross == 0) return 0;
55      else if(cross < 0) return -1;
56      else return 1;
57 }
58
59 vector <point> convex_hull(vector<point> p)
60 {
61      sort(p.begin(), p.end());
62      vector<point> L,U;
63
64      //Lower
```

```
65      for(auto pp : p)
66      {
67          while(L.size() >= 2 and ccw(L[L.size() - 2],
        L.back(), pp) == -1)
68          {
69              // Ã© -1 pq eu nÃčo quero excluir os
        colineares
70              L.pop_back();
71          }
72          L.push_back(pp);
73      }
74
75      reverse(p.begin(), p.end());
76
77      //Upper
78      for(auto pp : p)
79      {
80          while(U.size() >= 2 and ccw(U[U.size()-2], U
        .back(), pp) == -1)
81          {
82              U.pop_back();
83          }
84          U.push_back(pp);
85      }
86
87      L.pop_back();
88      L.insert(L.end(), U.begin(), U.end()-1);
89      return L;
90 }
91
92 cod area(vector<point> v)
93 {
94      int ans = 0;
95      int aux = (int)v.size();
96      for(int i = 2; i < aux; i++)
97      {
98          ans += ((v[i] - v[0])^(v[i-1] - v[0]))/2;
99      }
100     ans = abs(ans);
101     return ans;
102 }
103
104 int bound(point p1 , point p2)
105 {
106     return __gcd(abs(p1.x-p2.x), abs(p1.y-p2.y));
107 }
108 //teorema de pick [pontos = A - (bound+points)/2 + 1]
109
110 int32_t main()
111 {
112
113     int n;
114     cin >> n;
115
116     vector<point> v(n);
117     for(int i = 0; i < n; i++)
118     {
119         cin >> v[i].x >> v[i].y;
120     }
121
122     vector <point> ch = convex_hull(v);
123
124     cout << ch.size() <<  '\n';
125     for(auto p : ch) cout << p.x << " " << p.y << "\n
        ";
126
127     return 0;
128 }
```

5.3   Lattice Points

```
1 ll gcd(ll a, ll b) {
2      return b == 0 ? a : gcd(b, a % b);
```

```
3 }
4 ll area_triangulo(ll x1, ll y1, ll x2, ll y2, ll x3,
   ll y3) {
5      return abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 *
        (y1 - y2));
6 }
7 ll pontos_borda(ll x1, ll y1, ll x2, ll y2) {
8      return gcd(abs(x2 - x1), abs(y2 - y1));
9 }
10
11 int32_t main() {
12      ll x1, y1, x2, y2, x3, y3;
13      cin >> x1 >> y1;
14      cin >> x2 >> y2;
15      cin >> x3 >> y3;
16      ll area = area_triangulo(x1, y1, x2, y2, x3, y3);
17      ll tot_borda = pontos_borda(x1, y1, x2, y2) +
        pontos_borda(x2, y2, x3, y3) + pontos_borda(x3,
        y3, x1, y1);
18
19      ll ans = (area - tot_borda) / 2 + 1;
20      cout << ans << endl;
21
22      return 0;
23 }
```

5.4   Inside Polygon

```
1 // Convex O(logn)
2
3 bool insideT(point a, point b, point c, point e){
4      int x = ccw(a, b, e);
5      int y = ccw(b, c, e);
6      int z = ccw(c, a, e);
7      return !((x==1 or y==1 or z==1) and (x==-1 or y
        ==-1 or z==-1));
8 }
9
10 bool inside(vp &p, point e){ // ccw
11      int l=2, r=(int)p.size()-1;
12      while(l<r){
13          int mid = (l+r)/2;
14          if(ccw(p[0], p[mid], e) == 1)
15              l=mid+1;
16          else{
17              r=mid;
18          }
19      }
20      // bordo
21      // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)
        ==0) return false;
22      // if(r==2 and ccw(p[0], p[1], e)==0) return
        false;
23      // if(ccw(p[r], p[r-1], e)==0) return false;
24      return insideT(p[0], p[r-1], p[r], e);
25 }
26
27
28 // Any O(n)
29
30 int inside(vp &p, point pp){
31      // 1 - inside / 0 - boundary / -1 - outside
32      int n = p.size();
33      for(int i=0;i<n;i++){
34          int j = (i+1)%n;
35          if(line({p[i], p[j]}).inside_seg(pp))
36              return 0;
37      }
38      int inter = 0;
39      for(int i=0;i<n;i++){
40          int j = (i+1)%n;
41          if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p
        [i], p[j], pp)==1)
```

```
42          inter++; // up
43      else if(p[j].x <= pp.x and pp.x < p[i].x and
        ccw(p[i], p[j], pp)==-1)
44              inter++; // down
45      }
46
47      if(inter%2==0) return -1; // outside
48      else return 1; // inside
49 }
```

6 DP

6.1 Lis

6.2 Knapsack

```
1 // dp[i][j] => i-esimo item com j-carga sobrando na
     mochila
2 // O(N * W)
3
4 for(int j = 0; j < MAXN; j++) {
5     dp[0][j] = 0;
6 }
7 for(int i = 1; i <= N; i++) {
8     for(int j = 0; j <= W; j++) {
9         if(items[i].first > j) {
10             dp[i][j] = dp[i-1][j];
11         }
12         else {
13             dp[i][j] = max(dp[i-1][j], dp[i-1][j-
        items[i].first] + items[i].second);
14         }
15     }
16 }
```

6.3 Lcs

7 General

7.1 Struct

```
1 struct Pessoa{
2     // Atributos
3     string nome;
4     int idade;
5
6     // Comparador
7     bool operator<(const Pessoa& other) const{
8         if(idade != other.idade) return idade > other
        .idade;
9         else return nome > other.nome;
10     }
11 }
```

7.2 Bitwise

```
1 int check_kth_bit(int x, int k) {
2   return (x >> k) & 1;
3 }
4
5 void print_on_bits(int x) {
6   for (int k = 0; k < 32; k++) {
7     if (check_kth_bit(x, k)) {
8       cout << k << ' ';
9     }
10   }
11   cout << '\n';
```

```
12 }
13
14 int count_on_bits(int x) {
15   int ans = 0;
16   for (int k = 0; k < 32; k++) {
17     if (check_kth_bit(x, k)) {
18       ans++;
19     }
20   }
21   return ans;
22 }
23
24 bool is_even(int x) {
25   return ((x & 1) == 0);
26 }
27
28 int set_kth_bit(int x, int k) {
29   return x | (1 << k);
30 }
31
32 int unset_kth_bit(int x, int k) {
33   return x & (~(1 << k));
34 }
35
36 int toggle_kth_bit(int x, int k) {
37   return x ^ (1 << k);
38 }
39
40 bool check_power_of_2(int x) {
41   return count_on_bits(x) == 1;
42 }
```

8 Graph

8.1 Bellman Ford

```
1 struct Edge {
2     int u, v, w;
3 };
4
5 // se x = -1, nÃčo tem ciclo
6 // se x != -1, pegar pais de x pra formar o ciclo
7
8 int n, m;
9 vector<Edge> edges;
10 vector<int> dist(n);
11 vector<int> pai(n, -1);
12
13     for (int i = 0; i < n; i++) {
14         x = -1;
15         for (Edge &e : edges) {
16             if (dist[e.u] + e.w < dist[e.v]) {
17                 dist[e.v] = max(-INF, dist[e.u] + e.w
        );
18                 pai[e.v] = e.u;
19                 x = e.v;
20             }
21         }
22     }
23
24 // achando caminho (se precisar)
25 for (int i = 0; i < n; i++) x = pai[x];
26
27 vector<int> ciclo;
28 for (int v = x;; v = pai[v]) {
29     cycle.push_back(v);
30     if (v == x && ciclo.size() > 1) break;
31 }
32 reverse(ciclo.begin(), ciclo.end());
```

8.2 Lca

```cpp
// LCA - CP algorithm
// preprocessing O(NlogN)
// lca O(logN)
// Uso: criar LCA com a quantidade de vÃⓘrtices (n) e
    lista de adjacÃⓘncia (adj)
// chamar a funÃⓖÃⓒo preprocess com a raiz da Ãⓐrvore

struct LCA {
    int n, l, timer;
    vector<vector<int>> adj;
    vector<int> tin, tout;
    vector<vector<int>> up;

    LCA(int n, const vector<vector<int>>& adj) : n(n)
    , adj(adj) {}

    void dfs(int v, int p) {
        tin[v] = ++timer;
        up[v][0] = p;
        for (int i = 1; i <= l; ++i)
            up[v][i] = up[up[v][i-1]][i-1];

        for (int u : adj[v]) {
            if (u != p)
                dfs(u, v);
        }

        tout[v] = ++timer;
    }

    bool is_ancestor(int u, int v) {
        return tin[u] <= tin[v] && tout[u] >= tout[v
];
    }

    int lca(int u, int v) {
        if (is_ancestor(u, v))
            return u;
        if (is_ancestor(v, u))
            return v;
        for (int i = l; i >= 0; --i) {
            if (!is_ancestor(up[u][i], v))
                u = up[u][i];
        }
        return up[u][0];
    }

    void preprocess(int root) {
        tin.resize(n);
        tout.resize(n);
        timer = 0;
        l = ceil(log2(n));
        up.assign(n, vector<int>(l + 1));
        dfs(root, root);
    }
};
```

8.3   Dfs

```cpp
int dfs(int x, int p) {
    for (auto e : adj[x]) {
        if (e != p) {
            dfs(e, x);
        }
    }
}
```

8.4   Topological Sort

```cpp
vector<int> adj[MAXN];
vector<int> estado(MAXN); // 0: nao visitado 1:
    processamento 2: processado
vector<int> ordem;
bool temCiclo = false;

void dfs(int v) {
    if(estado[v] == 1) {
        temCiclo = true;
        return;
    }
    if(estado[v] == 2) return;
    estado[v] = 1;
    for(auto &nei : adj[v]) {
        if(estado[v] != 2) dfs(nei);
    }
    estado[v] = 2;
    ordem.push_back(v);
    return;
}
```

8.5   Dijkstra

```cpp
// SSP com pesos positivos.
// O((V + E) log V).

vector<int> dijkstra(int S) {
    vector<bool> vis(MAXN, 0);
    vector<ll> dist(MAXN, LLONG_MAX);
    dist[S] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> pq
;
    pq.push({0, S});
    while(pq.size()) {
        ll v = pq.top().second;
        pq.pop();
        if(vis[v]) continue;
        vis[v] = 1;
        for(auto &[peso, vizinho] : adj[v]) {
            if(dist[vizinho] > dist[v] + peso) {
                dist[vizinho] = dist[v] + peso;
                pq.push({dist[vizinho], vizinho});
            }
        }
    }
    return dist;
}
```

8.6   Lca Jc

```cpp
int LOG;

int get_lca(int a, int b) {
    if(profundidade[b] > profundidade[a]) {
        swap(a, b);
    }
    int k = profundidade[a] - profundidade[b]; //
    tanto que tenho que subir
    for(int j = LOG-1; j >= 0; j--) {
        if((1 << j) & k) {
            a = cima[a][j];
        }
    }
    if(a == b) return a; // ja to no lca

    for(int j = LOG-1; j >= 0; j--) { // subo com os
    dois atÃⓔ chegar no lca fazendo binary lifting
        if(cima[a][j] != cima[b][j]) {
            a = cima[a][j];
            b = cima[b][j];
        }
    }
    return cima[a][0];
}

void dfs(int v, int p) {
```

```
25        if(v != 1) profundidade[v] = profundidade[p] + 1;
26        cima[v][0] = p;
27        for(int j = 1; j < LOG; j++) {
28            if (cima[v][j-1] != -1) {
29                cima[v][j] = cima[cima[v][j-1]][j-1];
30            } else {
31                cima[v][j] = -1;
32            }
33        }
34        for(auto &nei : adj[v]) {
35            if(nei != p) {
36                dfs(nei, v);
37            }
38        }
39 }
40
41 while((1 << LOG) <= n) LOG++;
```

8.7   Kruskal

```
1 // Ordena as arestas por peso,  insere se ja nao
      estiver no mesmo componente
2 // O(E log E)
3
4 struct DSU {
5     vector<int> par, rank, sz;
6     int c;
7     DSU(int n) : par(n + 1), rank(n + 1, 0), sz(n +
   1, 1), c(n) {
8         for (int i = 1; i <= n; ++i) par[i] = i;
9     }
10    int find(int i) {
11        return (par[i] == i ? i : (par[i] = find(par[
   i])));
12    }
13    bool same(int i, int j) {
14        return find(i) == find(j);
15    }
16    int get_size(int i) {
17        return sz[find(i)];
18    }
19    int count() {
20        return c;  // quantos componentes conexos
21    }
22    int merge(int i, int j) {
23        if ((i = find(i)) == (j = find(j))) return
   -1;
24        else --c;
25        if (rank[i] > rank[j]) swap(i, j);
26        par[i] = j;
27        sz[j] += sz[i];
28        if (rank[i] == rank[j]) rank[j]++;
29        return j;
30    }
31 };
32
33 struct Edge {
34     int u, v, w;
35     bool operator <(Edge const & other) {
36         return weight <other.weight;
37     }
38 }
39
40 vector<Edge> kruskal(int n, vector<Edge> edges) {
41     vector<Edge> mst;
42     DSU dsu = DSU(n + 1);
43     sort(edges.begin(), edges.end());
44     for (Edge e : edges) {
45         if (dsu.find(e.u) != dsu.find(e.v)) {
46             mst.push_back(e);
47             dsu.join(e.u, e.v);
48         }
49     }
50     return mst;
51 }
```

8.8   Floyd Warshall

```
1 // SSP e acha ciclos.
2 // Bom com constraints menores.
3 // O(n^3)
4
5 int dist[501][501];
6
7 void floydWarshall() {
8     for(int k = 0; k < n; k++) {
9         for(int i = 0; i < n; i++) {
10            for(int j = 0; j < n; j++) {
11                dist[i][j] = min(dist[i][j], dist[i][
   k] + dist[k][j]);
12            }
13        }
14    }
15 }
16 void solve() {
17     int m, q;
18     cin >> n >> m >> q;
19     for(int i = 0; i < n; i++) {
20         for(int j = i; j < n; j++) {
21             if(i == j) {
22                 dist[i][j] = dist[j][i] = 0;
23             } else {
24                 dist[i][j] = dist[j][i] = linf;
25             }
26         }
27     }
28     for(int i = 0; i < m; i++) {
29         int u, v, w;
30         cin >> u >> v >> w; u--; v--;
31         dist[u][v] = min(dist[u][v], w);
32         dist[v][u] = min(dist[v][u], w);
33     }
34     floydWarshall();
35     while(q--) {
36         int u, v;
37         cin >> u >> v; u--; v--;
38         if(dist[u][v] == linf) cout << -1 << '\n';
39         else cout << dist[u][v] << '\n';
40     }
41 }
```

9   Search and sort

9.1   Mergeandcount

```
1
2 // Realiza a mesclagem de dois subarrays e conta o
      nÃžmero de trocas necessÃąrias.
3 int mergeAndCount(vector<int>& v, int l, int m, int r
   ) {
4     int x = m - l + 1;  // Tamanho do subarray
   esquerdo.
5     int y = r - m;  // Tamanho do subarray direito.
6
7     // Vetores temporarios para os subarray esquerdo
   e direito.
8     vector<int> left(x), right(y);
9
10    for (int i = 0; i < x; i++) left[i] = v[l + i];
11    for (int j = 0; j < y; j++) right[j] = v[m + 1 +
   j];
12
13    int i = 0, j = 0, k = l;
14    int swaps = 0;
```

```
15        while (i < x && j < y) {
16            if (left[i] <= right[j]) {
17                // Se o elemento da esquerda for menor ou
     igual, coloca no vetor original.
19                v[k++] = left[i++];
20            } else {
21                // Caso contrario, coloca o elemento da
     direita e conta as trocas.
22                v[k++] = right[j++];
23                swaps += (x - i);
24            }
25        }
26
27        // Adiciona os elementos restantes do subarray
     esquerdo (se houver).
28        while (i < x) v[k++] = left[i++];
29
30        // Adiciona os elementos restantes do subarray
     direito (se houver).
31        while (j < y) v[k++] = right[j++];
32
33        return swaps;  // Retorna o numero total de
     trocas realizadas.
34 }
35
36 int mergeSort(vector<int>& v, int l, int r) {
37        int swaps = 0;
38
39        if (l < r) {
40            // Encontra o ponto medio para dividir o
     vetor.
41            int m = l + (r - l) / 2;
42
43            // Chama merge sort para a metade esquerda.
44            swaps += mergeSort(v, l, m);
45            // Chama merge sort para a metade direita.
46            swaps += mergeSort(v, m + 1, r);
47
48            // Mescla as duas metades e conta as trocas.
49            swaps += mergeAndCount(v, l, m, r);
50        }
51
52        return swaps;  // Retorna o numero total de
     trocas no vetor.
53 }
```

9.2 Bfs

```
1 // Printa os nos na ordem em que sÃčo visitados
2 // Explora em largura (camadas)
3 // Complexidade: O(V+A) V = vertices e A = arestas
4 // Espaco: O(V)
5 // Uso: busca pelo caminho mais curto
6
7 void bfs(vector<vector<int>>&grafo, int inicio){
8        set<int> visited;
9        queue<int> fila;
10
11        fila.push(inicio);
12        visited.insert(inicio);
13
14        while(!fila.empty()){
15            int cur = fila.front();
16            fila.pop();
17
18            cout << cur << " "; // printa o nÃş atual
19
20            for(int vizinho: grafo[cur]){
21                if(visited.find(vizinho) == visited.end()
     ){
22                    fila.push(vizinho);
23                    visited.insert(vizinho)
```

```
24            }
25        }
26    }
27 }
```

9.3 Dfs

```
1 // Printa os nos na ordem em que sÃčo visitados
2 // Explora em profundidade
3 // Complexidade: O(V+A) V = vertices e A = arestas
4 // Espaco: O(V)
5 // Uso: explorar caminhos e backtracking
6
7 void dfs(vector<vector<int>>& grafo, int inicio){
8        set<int> visited;
9        stack<int> pilha;
10
11        pilha.push(inicio);
12
13        while(!pilha.empty()){
14            int cur = pilha.top();
15            pilha.pop();
16
17            if(visited.find(cur) == visited.end()){
18                cout << cur << " ";
19                visited.insert(cur);
20
21                for(int vizinho: grafo[cur]){
22                    if(visited.find(vizinho) == visited.
     end()){
23                        pilha.push(vizinho);
24                    }
25                }
26            }
27        }
28 }
```

10 Math

10.1 Equacao Diofantina

```
1 // resolve equacao ax + by = c
2 // retorno {existe sol., x, y, g}
3 array<ll, 4> find_any_solution(ll a, ll b, ll c) {
4        auto[x, y, g] = exgcd(a, b);
5        if (c % g) return {false, 0, 0, 0};
6        x *= c / g;
7        y *= c / g;
8        return {true, x, y, g};
9 }
```

10.2 Crivo

```
1 // O(n*log(log(n)))
2 bool composto[MAX]
3 for(int i = 1; i <= n; i++) {
4        if(composto[i]) continue;
5        for(int j = 2*i; j <= n; j += i)
6            composto[j] = 1;
7 }
```

10.3 Divisores

```
1 // Retorna um vetor com os divisores de x
2 // eh preciso ter o crivo implementado
3 // O(divisores)
4
5 vector<int> divs(int x){
6        vector<int> ans = {1};
7        vector<array<int, 2>> primos; // {primo, expoente
     }
8
```

```
9      while (x > 1) {
10         int p = crivo[x], cnt = 0;
11         while (x % p == 0) cnt++, x /= p;
12         primos.push_back({p, cnt});
13     }
14
15     for (int i=0; i<primos.size(); i++){
16         int cur = 1, len = ans.size();
17
18         for (int j=0; j<primos[i][1]; j++){
19             cur *= primos[i][0];
20             for (int k=0; k<len; k++)
21                 ans.push_back(cur*ans[k]);
22         }
23     }
24
25     return ans;
26 }
```

10.4   Fexp

```
1  // a^e mod m
2  // O(log n)
3
4  int fexp(int a, int e, int m) {
5      a %= m;
6      int ans = 1;
7      while (e > 0){
8          if (e & 1) ans = ans*a % m;
9          a = a*a % m;
```

```
10         e /= 2;
11     }
12     return ans%m;
13 }
```

10.5   Exgcd

```
1  // O retorno da funcao eh {n, m, g}
2  // e significa que gcd(a, b) = g e
3  // n e m sao inteiros tais que an + bm = g
4  array<ll, 3> exgcd(int a, int b) {
5      if(b == 0) return {1, 0, a};
6      auto [m, n, g] = exgcd(b, a % b);
7      return {n, m - a / b * n, g};
8  }
```

10.6   Mod Inverse

```
1  array<int, 2> extended_gcd(int a, int b) {
2      if (b == 0) return {1, 0};
3      auto [x, y] = extended_gcd(b, a % b);
4      return {y, x - (a / b) * y};
5  }
6
7  int mod_inverse(int a, int m) {
8      auto [x, y] = extended_gcd(a, m);
9      return (x % m + m) % m;
10 }
```