

Competitive Programming Notebook

Programadores Roblox

Contents

1 DP	2	7 General	11
1.1 Lis	2	7.1 Struct	11
1.2 Knapsack	2	7.2 Bitwise	11
1.3 Lcs	2	8 Search and sort	11
2 DS	2	8.1 Mergeandcount	11
2.1 Ordered Set E Map	2	8.2 Dfs	12
2.2 Segtree Iterativa	2	8.3 Bfs	12
2.3 Dsu	2	9 String copy	12
2.4 Segtree Sum	3	9.1 Z Function	12
2.5 Psum 2d	3	9.2 Trie Ponteiros	12
2.6 Segtree Gcd	3	9.3 Kmp	13
2.7 Bit	4	9.4 Hashing	13
3 Graph	4	9.5 Lcs	13
3.1 Lca	4	9.6 Countpermutations	13
3.2 Floyd Warshall	5	10 Primitives	13
3.3 Kruskal	5		
3.4 Topological Sort	5		
3.5 Bellman Ford	5		
3.6 Dfs	6		
3.7 Lca Jc	6		
3.8 Dijkstra	6		
4 String	6		
4.1 Z Function	6		
4.2 Trie Ponteiros	6		
4.3 Trie	7		
4.4 Kmp	7		
4.5 Hashing	7		
4.6 Lcs	7		
4.7 Countpermutations	8		
5 Geometry	8		
5.1 Lattice Points	8		
5.2 Inside Polygon	8		
5.3 Point Location	8		
5.4 Convex Hull	9		
6 Math	10		
6.1 Divisores	10		
6.2 Equacao Diofantina	10		
6.3 Crivo	10		
6.4 Mod Inverse	10		
6.5 Fexp	10		
6.6 Discrete Log	10		
6.7 Exgcd	10		
6.8 Segment Sieve	10		

1 DP

1.1 Lis

1.2 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-
14 items[i].first] + items[i].second);
15         }
16 }
```

1.3 Lcs

2 DS

2.1 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
17     elemento
18     cout << st.order_of_key(2) << endl; ///numero de
19     elementos menores que k
20     o_map<int, int>mp;
21     mp.insert({1, 10});
22     mp.insert({2, 20});
23     cout << mp.find_by_order(0)->second << endl; /// k-
24     esimo elemento
25     cout << mp.order_of_key(2) << endl; /// numero de
26     elementos (chave) menores que k
27     return 0;
28 }
```

2.2 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
23 + 1]);
24    }
25
26    T range_query(int l, int r) {
27        T res_l = neutral_value, res_r =
28        neutral_value;
29
30        for (l += n, r += n + 1; l < r; l >>= 1, r
31 >>= 1) {
32            if (l & 1) res_l = combine(res_l, tree[l
33 + 1]);
34            if (r & 1) res_r = combine(tree[--r],
35 res_r);
36        }
37
38        return combine(res_l, res_r);
39    }
40
41    void update(int pos, T new_val) {
42        tree[pos += n] = new_val;
43
44        for (pos >>= 1; pos > 0; pos >>= 1)
45            tree[pos] = combine(tree[2 * pos], tree[2
46 * pos + 1]);
47    }
48};
```

2.3 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 +
5 1, 1), c(n) {
6         for (int i = 1; i <= n; ++i) par[i] = i;
7     }
8     int find(int i) {
9         return (par[i] == i ? i : (par[i] = find(par[
10 i])));
11     }
12     bool same(int i, int j) {
13         return find(i) == find(j);
14     }
15     int get_size(int i) {
16         return sz[find(i)];
17     }
18     int count() {
19         return c; // quantos componentes conexos
20     }
21     int merge(int i, int j) {
```

```

20     if ((i = find(i)) == (j = find(j))) return
21     -1;
22     else --c;
23     if (rank[i] > rank[j]) swap(i, j);
24     par[i] = j;
25     sz[j] += sz[i];
26     if (rank[i] == rank[j]) rank[j]++;
27     return j;
28 };

```

2.4 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>
29     &v) {
30         if (l == r) {
31             t[p] = v[l];
32         } else {
33             int mid = (l + r) / 2;
34             build(lc(p), l, mid, v);
35             build(rc(p), mid + 1, r, v);
36             t[p] = merge(t[lc(p)], t[rc(p)]);
37         }
38     }
39     void build(int _n) {
40         n = _n;
41         t.assign(n * 4, neutral);
42         lazy.assign(n * 4, 0);
43         replace.assign(n * 4, false);
44     }
45     void build(const vector<ll> &v) {
46         n = (int)v.size();
47         t.assign(n * 4, neutral);
48         lazy.assign(n * 4, 0);
49         replace.assign(n * 4, false);
50         build(1, 0, n - 1, v);
51     }
52     void build(ll *bg, ll *en) {
53         build(vector<ll>(bg, en));
54     }
55     ll query(int p, int l, int r, int L, int R) {
56         push(p, l, r);
57         if (l > R || r < L) return neutral;
58         if (l >= L && r <= R) return t[p];
59         int mid = (l + r) / 2;
60         auto ql = query(lc(p), l, mid, L, R);

```

```

61         auto qr = query(rc(p), mid + 1, r, L, R);
62         return merge(ql, qr);
63     }
64     ll query(int l, int r) { return query(1, 0, n -
65     1, l, r); }
66     void update(int p, int l, int r, int L, int R, ll
67     val, bool repl = 0) {
68         push(p, l, r);
69         if (l > R || r < L) return;
70         if (l >= L && r <= R) {
71             lazy[p] = val;
72             replace[p] = repl;
73             push(p, l, r);
74         } else {
75             int mid = (l + r) / 2;
76             update(lc(p), l, mid, L, R, val, repl);
77             update(rc(p), mid + 1, r, L, R, val, repl
78             );
79             t[p] = merge(t[lc(p)], t[rc(p)]);
80         }
81     }
82     void sumUpdate(int l, int r, ll val) { update(1,
83     0, n - 1, l, r, val, 0); }
84     void assignUpdate(int l, int r, ll val) { update
85     (1, 0, n - 1, l, r, val, 1); }
86 } segsum;

```

2.5 Psum 2d

```

1 vector<vector<int>> psum(h+1, vector<int>(w+1, 0));
2
3 for (int i=1; i<=h; i++){
4     for (int j=1; j<=w; j++){
5         cin >> psum[i][j];
6         psum[i][j] += psum[i-1][j]+psum[i][j-1]-psum[
7         i-1][j-1];
8     }
9 }
10 // retorna a psum2d do intervalo inclusivo [(a, b), (
11 c, d)]
12 int retangulo(int a, int b, int c, int d){
13     c = min(c, h), d = min(d, w);
14     a = max(0LL, a-1), b = max(0LL, b-1);
15     return v[c][d]-v[a][d]-v[c][b]+v[a][b];
16 }

```

2.6 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
13    start, int end) {
14        if (start == end) {
15            tree[node] = arr[start];
16        } else {
17            int mid = (start + end) / 2;
18            build(arr, 2 * node + 1, start, mid);
19            build(arr, 2 * node + 2, mid + 1, end);
20            tree[node] = gcd(tree[2 * node + 1], tree
21            [2 * node + 2]);
22        }
23    }
24 }

```

```

21     }
22
23     void update(int node, int start, int end, int idx
24     , int value) {
25         if (start == end) {
26             tree[node] = value;
27         } else {
28             int mid = (start + end) / 2;
29             if (idx <= mid) {
30                 update(2 * node + 1, start, mid, idx,
31                 value);
32             } else {
33                 update(2 * node + 2, mid + 1, end,
34                 idx, value);
35             }
36             tree[node] = gcd(tree[2 * node + 1], tree
37             [2 * node + 2]);
38         }
39     }
40
41     int query(int node, int start, int end, int l,
42     int r) {
43         if (r < start || l > end) {
44             return 0;
45         }
46         if (l <= start && end <= r) {
47             return tree[node];
48         }
49         int mid = (start + end) / 2;
50         int left_gcd = query(2 * node + 1, start, mid
51         , l, r);
52         int right_gcd = query(2 * node + 2, mid + 1,
53         end, l, r);
54         return gcd(left_gcd, right_gcd);
55     }
56
57     public:
58     SegmentTreeGCD(const vector<int>& arr) {
59         n = arr.size();
60         tree.resize(4 * n);
61         build(arr, 0, 0, n - 1);
62     }
63
64     void update(int idx, int value) {
65         update(0, 0, n - 1, idx, value);
66     }
67
68     int query(int l, int r) {
69         return query(0, 0, n - 1, l, r);
70     }
71 };

```

2.7 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 Graph

3.1 Lca

```

1  // LCA - CP algorithm
2  // preprocessing O(NlogN)
3  // lca O(logN)
4  // Uso: criar LCA com a quantidade de vÃrtices (n) e
5  // lista de adjacÃncia (adj)
6  // chamar a funÃÃo preprocess com a raiz da Ãrvore
7
8  struct LCA {
9      int n, l, timer;
10     vector<vector<int>> adj;
11     vector<int> tin, tout;
12     vector<vector<int>> up;
13
14     LCA(int n, const vector<vector<int>>& adj) : n(n)
15     , adj(adj) {}
16
17     void dfs(int v, int p) {
18         tin[v] = ++timer;
19         up[v][0] = p;
20         for (int i = 1; i <= l; ++i)
21             up[v][i] = up[up[v][i-1]][i-1];
22
23         for (int u : adj[v]) {
24             if (u != p)
25                 dfs(u, v);
26         }
27         tout[v] = ++timer;
28     }
29
30     bool is_ancestor(int u, int v) {
31         return tin[u] <= tin[v] && tout[u] >= tout[v]
32     };
33
34     int lca(int u, int v) {
35         if (is_ancestor(u, v))
36             return u;
37         if (is_ancestor(v, u))
38             return v;
39         for (int i = l; i >= 0; --i) {
40             if (!is_ancestor(up[u][i], v))
41                 u = up[u][i];
42         }
43         return up[u][0];
44     }
45
46     void preprocess(int root) {
47         tin.resize(n);
48         tout.resize(n);
49         timer = 0;
50         l = ceil(log2(n));
51         up.assign(n, vector<int>(l + 1));

```

```

51     dfs(root, root);
52 }
53 };

```

3.2 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 }

```

3.3 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 }

```

3.4 Topological Sort

```

1 vector<int> adj[MAXN];
2 vector<int> estado(MAXN); // 0: nao visitado 1:
   // processamento 2: processado
3 vector<int> ordem;
4 bool temCiclo = false;
5
6 void dfs(int v) {
7     if(estado[v] == 1) {
8         temCiclo = true;
9         return;
10    }
11    if(estado[v] == 2) return;
12    estado[v] = 1;
13    for(auto &nei : adj[v]) {
14        if(estado[v] != 2) dfs(nei);
15    }
16    estado[v] = 2;
17    ordem.push_back(v);
18    return;

```

3.5 Bellman Ford

```

1 struct Edge {
2     int u, v, w;
3 };
4
5 // se x = -1, não 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());

```

3.6 Dfs

```

1 int dfs(int x, int p) {
2     for (auto e : adj[x]) {
3         if (e != p) {
4             dfs(e, x);
5         }
6     }
7 }

```

3.7 Lca Jc

```

1 int LOG;
2
3 int get_lca(int a, int b) {
4     if (profundidade[b] > profundidade[a]) {
5         swap(a, b);
6     }
7     int k = profundidade[a] - profundidade[b]; //
8     // tanto que tenho que subir
9     for (int j = LOG-1; j >= 0; j--) {
10         if ((1 << j) & k) {
11             a = cima[a][j];
12         }
13     }
14     if (a == b) return a; // ja to no lca
15
16     for (int j = LOG-1; j >= 0; j--) { // subo com os
17         // dois até chegar no lca fazendo binary lifting
18         if (cima[a][j] != cima[b][j]) {
19             a = cima[a][j];
20             b = cima[b][j];
21         }
22     }
23     return cima[a][0];
24 }
25
26 void dfs(int v, int p) {
27     if (v != 1) profundidade[v] = profundidade[p] + 1;
28     cima[v][0] = p;
29     for (int j = 1; j < LOG; j++) {
30         if (cima[v][j-1] != -1) {
31             cima[v][j] = cima[cima[v][j-1]][j-1];
32         } else {
33             cima[v][j] = -1;
34         }
35     }
36     for (auto &nei : adj[v]) {
37         if (nei != p) {
38             dfs(nei, v);
39         }
40     }
41 }

```

```

38     }
39 }
40
41 while ((1 << LOG) <= n) LOG++;

```

3.8 Dijkstra

```

1 // SSP com pesos positivos.
2 // O((V + E) log V).
3
4 vector<int> dijkstra(int S) {
5     vector<bool> vis(MAXN, 0);
6     vector<ll> dist(MAXN, LLONG_MAX);
7     dist[S] = 0;
8     priority_queue<pii, vector<pii>, greater<pii>> pq;
9     pq.push({0, S});
10    while (pq.size()) {
11        ll v = pq.top().second;
12        pq.pop();
13        if (vis[v]) continue;
14        vis[v] = 1;
15        for (auto &[peso, vizinho] : adj[v]) {
16            if (dist[vizinho] > dist[v] + peso) {
17                dist[vizinho] = dist[v] + peso;
18                pq.push({dist[vizinho], vizinho});
19            }
20        }
21    }
22    return dist;
23 }

```

4 String

4.1 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        if (i + z[i] > r) {
12            l = i;
13            r = i + z[i];
14        }
15    }
16    return z;
17 }
18 }

```

4.2 Trie Ponteiros

```

1 // Trie por ponteiros
2 // Inserir, buscar e consultar 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 Ã 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 }

```

4.3 Trie

```

1 // Trie por array
2 // Inserir, busca e consulta de prefixo em O(N)
3
4 int trie[MAXN][26];
5 int tot_nos = 0;
6 vector<bool> acaba(MAXN, false);
7 vector<int> contador(MAXN, 0);
8
9 void insere(string s) {
10     int no = 0;
11     for(auto &c : s) {
12         if(trie[no][c - 'a'] == 0) {
13             trie[no][c - 'a'] = ++tot_nos;
14         }
15         no = trie[no][c - 'a'];
16         contador[no]++;
17     }
18     acaba[no] = true;
19 }
20
21 bool busca(string s) {
22     int no = 0;
23     for(auto &c : s) {
24         if(trie[no][c - 'a'] == 0) {
25             return false;
26         }
27         no = trie[no][c - 'a'];
28     }
29     return acaba[no];
30 }
31
32 int isPref(string s) {
33     int no = 0;
34     for(auto &c : s) {

```

```

35         if(trie[no][c - 'a'] == 0){
36             return -1;
37         }
38         no = trie[no][c - 'a'];
39     }
40     return contador[no];
41 }

```

4.4 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 }

```

4.5 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-1
        +1] % MOD : 0));
26        return hash < 0 ? hash + MOD : hash;
27    }
28 };

```

4.6 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 }

```

4.7 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 {
7     int n = t.size();
8     int ans = 0;
9
10    vector<int> fact(n + 1, 1), invfact(n + 1, 1);
11    for (int i = 1; i <= n; i++)
12        fact[i] = (fact[i - 1] * i) % MOD;
13    invfact[n] = fexp(fact[n], MOD - 2, MOD);
14    for (int i = n - 1; i >= 0; i--)
15        invfact[i] = (invfact[i + 1] * (i + 1)) % MOD;
16
17    // For each position in t, try placing a letter
18    // smaller than t[i] that is in freq
19    for (int i = 0; i < n; i++) {
20        for (char c = 'a'; c < t[i]; c++) {
21            if (freq[c - 'a'] > 0) {
22                freq[c - 'a']--;
23                int ways = fact[n - i - 1];
24                for (int f : freq)
25                    ways = (ways * invfact[f]) % MOD;
26                ans = (ans + ways) % MOD;
27                freq[c - 'a']++;
28            }
29        }
30        if (freq[t[i] - 'a'] == 0) break;
31        freq[t[i] - 'a']--;
32    }
33    return ans;
34 }

```

5 Geometry

5.1 Lattice Points

```

1 ll gcd(ll a, ll b) {
2     return b == 0 ? a : gcd(b, a % b);
3 }
4 ll area_trianguelo(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_trianguelo(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 }

```

5.2 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 // Any O(n)
28
29 int inside(vp &p, point pp){
30     // 1 - inside / 0 - boundary / -1 - outside
31     int n = p.size();
32     for(int i=0; i<n; i++){
33         int j = (i+1)%n;
34         if(line({p[i], p[j]}).inside_seg(pp))
35             return 0;
36     }
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 }

```

5.3 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), deltax1 = (y1-y2);
12
13         int compx = (x1-x3), compy = (y1-y3);
14
15         int ans = (deltax1*compy) - (compx*deltax1);
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 }

```

5.4 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             // Ãl -1 pq eu nÃo quero excluir os
colineares
L.pop_back();
70         }
71         L.push_back(pp);
72     }
73
74     reverse(p.begin(), p.end());
75
76     //Upper
77     for(auto pp : p)
78     {
79         while(U.size() >= 2 and ccw(U[U.size()-2], U
.back(), pp) == -1)
80         {
81             U.pop_back();
82         }
83         U.push_back(pp);
84     }
85
86     L.pop_back();
87     L.insert(L.end(), U.begin(), U.end()-1);
88     return L;
89 }
90
91 cod area(vector<point> v)
92 {
93     int ans = 0;
94     int aux = (int)v.size();
95     for(int i = 2; i < aux; i++)
96     {
97         ans += ((v[i] - v[0])^(v[i-1] - v[0]))/2;
98     }
99     ans = abs(ans);
100     return ans;
101 }
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 vector <point> ch = convex_hull(v);
122
123 cout << ch.size() << '\n';
124 for(auto p : ch) cout << p.x << " " << p.y << "\n";
125
126 return 0;
127 }
128

```

6 Math

6.1 Divisores

```

1 // Retorna um vetor com os divisores de x
2 // eh preciso ter o crivo implementado
3 // 0(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 }

```

6.2 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 }

```

6.3 Crivo

```

1 // 0(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 }

```

6.4 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 }

```

6.5 Fexp

```

1 // a^e mod m
2 // 0(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 }

```

6.6 Discrete Log

```

1 // Returns minimum x for which a^x = b (mod m), a and
2 // m are coprime.
3 // if the answer dont need to be greater than some
4 // value, the vector<int> can be removed
5 int discrete_log(int a, int b, int m) {
6     a %= m, b %= m;
7     int n = sqrt(m) + 1;
8
9     int an = 1;
10    for (int i = 0; i < n; ++i)
11        an = (an * 1ll * a) % m;
12
13    unordered_map<int, vector<int>> vals;
14    for (int q = 0, cur = b; q <= n; ++q) {
15        vals[cur].push_back(q);
16        cur = (cur * 1ll * a) % m;
17    }
18
19    int res = LLONG_MAX;
20
21    for (int p = 1, cur = 1; p <= n; ++p) {
22        cur = (cur * 1ll * an) % m;
23        if (vals.count(cur)) {
24            for (int q: vals[cur]){
25                int ans = n * p - q;
26                res = min(res, ans);
27            }
28        }
29    }
30    return res;
31 }

```

6.7 Exgcd

```

1 // 0 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 }

```

6.8 Segment Sieve

```

1 // Retorna quantos primos tem entre [l, r] (inclusivo)

```

```

2 // precisa de um vetor com os primos até sqrt(r)
3 int seg_sieve(int l, int r){
4     if (l > r) return 0;
5     vector<bool> is_prime(r - l + 1, true);
6     if (l == 1) is_prime[0] = false;
7
8     for (int p : primos){
9         if (p * p > r) break;
10        int start = max(p * p, (l + p - 1) / p * p);
11        for (int j = start; j <= r; j += p){
12            if (j >= l) {
13                is_prime[j - l] = false;
14            }
15        }
16    }
17
18    return accumulate(all(is_prime), 0ll);
19 }

```

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 Search and sort

8.1 Mergeandcount

```

1 // Realiza a mesclagem de dois subarrays e conta o
2 // número de trocas necessárias.
3 int mergeAndCount(vector<int>& v, int l, int m, int r
4 ) {
5     int x = m - l + 1; // Tamanho do subarray
6     // esquerdo.
7     int y = r - m; // Tamanho do subarray direito.
8
9     // Vetores temporários para os subarray esquerdo
10    // e direito.
11    vector<int> left(x), right(y);
12
13    for (int i = 0; i < x; i++) left[i] = v[l + i];
14    for (int j = 0; j < y; j++) right[j] = v[m + 1 +
15    j];
16
17    int i = 0, j = 0, k = l;
18    int swaps = 0;
19
20    while (i < x && j < y) {
21        if (left[i] <= right[j]) {
22            // Se o elemento da esquerda for menor ou
23            // igual, coloca no vetor original.
24            v[k++] = left[i++];
25        } else {
26            // Caso contrario, coloca o elemento da
27            // direita e conta as trocas.
28            v[k++] = right[j++];
29            swaps += (x - i);
30        }
31    }
32
33    // Adiciona os elementos restantes do subarray
34    // esquerdo (se houver).
35    while (i < x) v[k++] = left[i++];
36
37    // Adiciona os elementos restantes do subarray
38    // direito (se houver).
39    while (j < y) v[k++] = right[j++];
40
41    return swaps; // Retorna o numero total de
42    // trocas realizadas.
43 }
44
45 int mergeSort(vector<int>& v, int l, int r) {
46     int swaps = 0;
47
48     if (l < r) {
49         // Encontra o ponto medio para dividir o
50         // vetor.
51         int m = l + (r - l) / 2;
52
53         // Chama merge sort para a metade esquerda.
54         swaps += mergeSort(v, l, m);
55         // Chama merge sort para a metade direita.
56         swaps += mergeSort(v, m + 1, r);

```

```

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
53 trocas no vetor.

```

8.2 Dfs

```

1 // Printa os nos na ordem em que sÃ£o 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.
23 end()){
24                     pilha.push(vizinho);
25                 }
26             }
27         }
28     }

```

8.3 Bfs

```

1 // Printa os nos na ordem em que sÃ£o 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 nÃo atual
19
20         for(int vizinho: grafo[cur]){
21             if(visited.find(vizinho) == visited.end())
22             {
23                 fila.push(vizinho);
24                 visited.insert(vizinho);
25             }
26         }
27     }

```

9 String copy

9.1 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         {
11             z[i]++;
12         }
13         if(i + z[i] > r) {
14             l = i;
15             r = i + z[i];
16         }
17     }
18     return z;

```

9.2 Trie Ponteiros

```

1 // Trie por ponteiros
2 // InserÃ§Ã£o, 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 Ã prefixo e quantas strings tem s como
38 prefixo
39 int isPref(string s, Node *raiz) {
40     Node *cur = raiz;
41     for(auto &c : s) {
42         if (cur->filhos[c - 'a'] != NULL) {
43             cur = cur->filhos[c - 'a'];
44             continue;
45         }
46         return -1;

```

```

46     }
47     return cur->contador;
48 }

```

9.3 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 }

```

9.4 Hashing

```

1 // String Hash template
2 // constructor(s) - O(|s|)
3 // query(l, r) - returns the hash of the range [l,r]
4 // from left to right - O(1)
5 // query_inv(l, r) from right to left - O(1)
6 // patrocinado por tiagodfs
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
14    (n) {
15        for (int i=0;i<n;i++) p[i] = (i ? X*p[i-1]:1)
16        % MOD;
17        for (int i=0;i<n;i++)
18            h[i] = (s[i] + (i ? h[i-1]:0) * X) % MOD;
19        for (int i=n-1;i>=0;i--)
20            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * X)
21            % MOD;
22    }
23    int query(int l, int r) {
24        int hash = (h[r] - (l ? h[l-1]*p[r-l+1]:0)%MOD :
25        0));
26        return hash < 0 ? hash + MOD : hash;
27    }
28    int query_inv(int l, int r) {
29        int hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l
30        +1] % MOD : 0));
31        return hash < 0 ? hash + MOD : hash;
32    }
33 };

```

9.5 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,
6     0));
7
8     for (int i = 1; i <= m; ++i) {
9         for (int j = 1; j <= n; ++j) {
10            if (s1[i - 1] == s2[j - 1])
11                dp[i][j] = dp[i - 1][j - 1] + 1;
12            else
13                dp[i][j] = max(dp[i - 1][j], dp[i][j
14            - 1]);
15        }
16    }
17    return dp[m][n];
18 }

```

9.6 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
4 // characters
5 // O(n*freq.size())
6 int countPermLess(vector<int> freq, const string &t)
7 {
8     int n = t.size();
9     int ans = 0;
10
11    vector<int> fact(n + 1, 1), invfact(n + 1, 1);
12    for (int i = 1; i <= n; i++)
13        fact[i] = (fact[i - 1] * i) % MOD;
14    invfact[n] = fexp(fact[n], MOD - 2, MOD);
15    for (int i = n - 1; i >= 0; i--)
16        invfact[i] = (invfact[i + 1] * (i + 1)) % MOD;
17
18    // For each position in t, try placing a letter
19    // smaller than t[i] that is in freq
20    for (int i = 0; i < n; i++) {
21        for (char c = 'a'; c < t[i]; c++) {
22            if (freq[c - 'a'] > 0) {
23                freq[c - 'a']--;
24                int ways = fact[n - i - 1];
25                for (int f : freq)
26                    ways = (ways * invfact[f]) % MOD;
27                ans = (ans + ways) % MOD;
28                freq[c - 'a']++;
29            }
30        }
31        if (freq[t[i] - 'a'] == 0) break;
32        freq[t[i] - 'a']--;
33    }
34    return ans;
35 }

```

10 Primitives