



Notebook - Maratona de Programação

Cabo HDMI, VGA, USB

Contents

| | | |
|----------|-----------------------------|----------|
| 1 | Segtree | 2 |
| 1.1 | Standard SegTree | 2 |
| 1.2 | Lazy SegTree | 2 |
| 2 | Graph | 3 |
| 2.1 | Dinic | 3 |
| 2.2 | Bellman Ford | 4 |
| 2.3 | Topological Sort | 4 |
| 2.4 | Kth Ancestor | 4 |
| 3 | Parser | 5 |
| 3.1 | Parsing Functions | 5 |
| 4 | String | 5 |
| 4.1 | Z function | 5 |
| 4.2 | String hash | 5 |
| 5 | Set | 6 |
| 5.1 | Ordered Set | 6 |
| 5.2 | Set | 6 |
| 5.3 | Multiset | 6 |

1 Segtree

1.1 Standard SegTree

Complexity: $O(\log(n))$ query and update
Sum segment tree with point update.

```
1 // TITLE: Standard SegTree
2 // COMPLEXITY:  $O(\log(n))$  query and update
3 // DESCRIPTION: Sum segment tree with point update.
4
5 using type = int;
6
7 type iden = 0;
8 vector<type> seg;
9 int segsize;
10
11 type func(type a, type b)
12 {
13     return a + b;
14 }
15
16 // query do intervalo [l, r)
17 type query(int l, int r, int no = 0, int lx = 0, int
    rx = segsize)
18 {
19     // l lx rx r
20     if (r <= lx or rx <= l)
21         return iden;
22     if (l <= lx and rx <= r)
23         return seg[no];
24
25     int mid = lx + (rx - lx) / 2;
26     return func(query(l, r, 2 * no + 1, lx, mid),
27         query(l, r, 2 * no + 2, mid, rx));
28 }
29
30 void update(int dest, type val, int no = 0, int lx =
    0, int rx = segsize)
31 {
32     if (dest < lx or dest >= rx)
33         return;
34     if (rx - lx == 1)
35     {
36         seg[no] = val;
37         return;
38     }
39
40     int mid = lx + (rx - lx) / 2;
41     update(dest, val, 2 * no + 1, lx, mid);
42     update(dest, val, 2 * no + 2, mid, rx);
43     seg[no] = func(seg[2 * no + 1], seg[2 * no + 2]);
44 }
45
46 signed main()
47 {
48     ios_base::sync_with_stdio(0);
49     cin.tie(0);
50     cout.tie(0);
51     int n;
52     cin >> n;
53     segsize = n;
54     if (__builtin_popcount(n) != 1)
55     {
56         segsize = 1 + (int)log2(segsize);
57         segsize = 1 << segsize;
58     }
59     seg.assign(2 * segsize - 1, iden);
60
61     loop(i, 0, n)
62     {
63         int x;
64         cin >> x;
```

```
65         update(i, x);
66     }
67 }
```

1.2 Lazy SegTree

Complexity: $O(\log(n))$ query and update
Sum segment tree with range sum update.

```
1 // TITLE: Lazy SegTree
2 // COMPLEXITY:  $O(\log(n))$  query and update
3 // DESCRIPTION: Sum segment tree with range sum
    update.
4 vector<int> seg, lazy;
5 int segsize;
6
7 // change 0s to -1s if update is
8 // set instead of add. also,
9 // remove the +=s
10 void prop(int no, int lx, int rx) {
11     if (lazy[no] == 0) return;
12
13     seg[no] += (rx - lx) * lazy[no];
14     if (rx - lx > 1) {
15         lazy[2 * no + 1] += lazy[no];
16         lazy[2 * no + 2] += lazy[no];
17     }
18
19     lazy[no] = 0;
20 }
21
22 void update(int l, int r, int val, int no = 0, int lx = 0,
    int rx = segsize) {
23     // l r lx rx
24     prop(no, lx, rx);
25     if (r <= lx or rx <= l) return;
26     if (l <= lx and rx <= r) {
27         lazy[no] = val;
28         prop(no, lx, rx);
29         return;
30     }
31
32     int mid = lx + (rx - lx) / 2;
33     update(l, r, val, 2 * no + 1, lx, mid);
34     update(l, r, val, 2 * no + 2, mid, rx);
35     seg[no] = seg[2 * no + 1] + seg[2 * no + 2];
36 }
37
38 int query(int l, int r, int no = 0, int lx = 0, int rx =
    segsize) {
39     prop(no, lx, rx);
40     if (r <= lx or rx <= l) return 0;
41     if (l <= lx and rx <= r) return seg[no];
42
43     int mid = lx + (rx - lx) / 2;
44     return query(l, r, 2 * no + 1, lx, mid) +
45         query(l, r, 2 * no + 2, mid, rx);
46 }
47
48 signed main() {
49     ios_base::sync_with_stdio(0); cin.tie(0); cout.tie
    (0);
50
51     int n; cin >> n;
52     segsize = n;
53     if (__builtin_popcount(n) != 1) {
54         segsize = 1 + (int)log2(segsize);
55         segsize = 1 << segsize;
56     }
57
58     seg.assign(2 * segsize - 1, 0);
59     // use -1 instead of 0 if
```

```

60 // update is set instead of add
61 lazy.assign(2*segsz-1, 0);
62 }

```

2 Graph

2.1 Dinic

Complexity: $O(V*V*E)$, Bipartite is $O(\sqrt{V} E)$

Dinic is a strongly polynomial maximum flow algorithm, doesn't depend on capacity values good for matching

```

1 // TITLE: Dinic
2 // COMPLEXITY:  $O(V*V*E)$ , Bipartite is  $O(\sqrt{V} E)$ 
3 // DESCRIPTION: Dinic is a strongly polynomial
  maximum flow algorithm, doesn't depend on capacity
  values good for matching
4
5 const int oo = 0x3f3f3f3f3f3f3f3f;
6 // Edge structure
7 struct Edge
8 {
9     int from, to;
10    int flow, capacity;
11
12    Edge(int from_, int to_, int flow_, int capacity_)
13        : from(from_), to(to_), flow(flow_), capacity_
      (capacity_)
14    {}
15 };
16
17 struct Dinic
18 {
19     vector<vector<int>> graph;
20     vector<Edge> edges;
21     vector<int> level;
22     int size;
23
24     Dinic(int n)
25     {
26         graph.resize(n);
27         level.resize(n);
28         size = n;
29         edges.clear();
30     }
31
32     void add_edge(int from, int to, int capacity)
33     {
34         edges.emplace_back(from, to, 0, capacity);
35         graph[from].push_back(edges.size() - 1);
36
37         edges.emplace_back(to, from, 0, 0);
38         graph[to].push_back(edges.size() - 1);
39     }
40
41     int get_max_flow(int source, int sink)
42     {
43         int max_flow = 0;
44         vector<int> next(size);
45         while(bfs(source, sink)) {
46             next.assign(size, 0);
47             for (int f = dfs(source, sink, next, oo);
48                  f != 0; f = dfs(source, sink, next, oo)) {
49                 max_flow += f;
50             }
51             return max_flow;
52         }
53     }

```

```

54 bool bfs(int source, int sink)
55 {
56     level.assign(size, -1);
57     queue<int> q;
58     q.push(source);
59     level[source] = 0;
60
61     while(!q.empty()) {
62         int a = q.front();
63         q.pop();
64
65         for (int & b: graph[a]) {
66             auto edge = edges[b];
67             int cap = edge.capacity - edge.flow;
68             if (cap > 0 && level[edge.to] == -1)
69             {
70                 level[edge.to] = level[a] + 1;
71                 q.push(edge.to);
72             }
73         }
74     }
75     return level[sink] != -1;
76 }
77
78 int dfs(int curr, int sink, vector<int> & next,
79 int flow)
80 {
81     if (curr == sink) return flow;
82     int num_edges = graph[curr].size();
83
84     for (; next[curr] < num_edges; next[curr]++)
85     {
86         int b = graph[curr][next[curr]];
87         auto & edge = edges[b];
88         auto & rev_edge = edges[b^1];
89
90         int cap = edge.capacity - edge.flow;
91         if (cap > 0 && (level[curr] + 1 == level[
92             edge.to])) {
93             int bottle_neck = dfs(edge.to, sink,
94             next, min(flow, cap));
95             if (bottle_neck > 0) {
96                 edge.flow += bottle_neck;
97                 rev_edge.flow -= bottle_neck;
98                 return bottle_neck;
99             }
100         }
101     }
102     return 0;
103 }
104
105 // Example on how to use
106 void solve()
107 {
108     int n, m;
109     cin >> n >> m;
110     int N = n + m + 2;
111
112     int source = N - 2;
113     int sink = N - 1;
114
115     Dinic flow(N);
116
117     for (int i = 0; i < n; i++) {
118         int q; cin >> q;
119         while(q--) {
120             int b; cin >> b;
121             flow.add_edge(i, n + b - 1, 1);
122         }
123     }
124     for (int i = 0; i < n; i++) {
125         flow.add_edge(source, i, 1);
126     }

```

```

122     }
123     for (int i = 0; i < m; i++) {
124         flow.add_edge(i + n, sink, 1);
125     }
126
127     cout << m - flow.get_max_flow(source, sink) << endl;
128
129     // Getting participant edges
130     for (auto & edge: flow.edges) {
131         if (edge.capacity == 0) continue; // This
means is a reverse edge
132         if (edge.from == source || edge.to == source)
continue;
133         if (edge.from == sink || edge.to == sink)
continue;
134         if (edge.flow == 0) continue; // Is not
participant
135
136         cout << edge.from + 1 << " " << edge.to - n +
1 << endl;
137     }
138 }

```

2.2 Bellman Ford

Complexity: $O(n * m)$ | $n = |\text{nodes}|$, $m = |\text{edges}|$
Finds shortest paths from a starting node to all nodes of the graph. The node can have negative cycle and belman-ford will detected

```

1 // TITLE: Bellman Ford
2 // COMPLEXITY:  $O(n * m)$  |  $n = |\text{nodes}|$ ,  $m = |\text{edges}|$ 
3 // DESCRIPTION: Finds shortest paths from a starting
node to all nodes of the graph. The node can have
negative cycle and belman-ford will detected
4
5 // a and b vertices, c cost
6 // [{a, b, c}, {a, b, c}]
7 vector<tuple<int, int, int>> edges;
8 int N;
9
10 void bellman_ford(int x){
11     for (int i = 0; i < N; i++){
12         dist[i] = oo;
13     }
14     dist[x] = 0;
15
16     for (int i = 0; i < N - 1; i++){
17         for (auto [a, b, c]: edges){
18             if (dist[a] == oo) continue;
19             dist[b] = min(dist[b], dist[a] + w);
20         }
21     }
22 }
23 // return true if has cycle
24 bool check_negative_cycle(int x){
25     for (int i = 0; i < N; i++){
26         dist[i] = oo;
27     }
28     dist[x] = 0;
29
30     for (int i = 0; i < N - 1; i++){
31         for (auto [a, b, c]: edges){
32             if (dist[a] == oo) continue;
33             dist[b] = min(dist[b], dist[a] + w);
34         }
35     }
36
37     for (auto [a, b, c]: edges){
38         if (dist[a] == oo) continue;

```

```

39         if (dist[a] + w < dist[b]){
40             return true;
41         }
42     }
43     return false;
44 }
45 """

```

2.3 Topological Sort

Complexity: $O(N + M)$, N : Vertices, M : Arestas

Retorna no do grapho em ordem topologica, se a quantidade de nos retornada nao for igual a quantidade de nos e impossivel

```

1 // TITLE: Topological Sort
2 // COMPLEXITY:  $O(N + M)$ ,  $N$ : Vertices,  $M$ : Arestas
3 // DESCRIPTION: Retorna no do grapho em ordem
topologica, se a quantidade de nos retornada nao
for igual a quantidade de nos e impossivel
4
5 typedef vector<vector<int>> Adj_List;
6 typedef vector<int> Indegree_List; // How many nodes
depend on him
7 typedef vector<int> Order_List; // The order in
which the nodes appears
8
9 Order_List kahn(Adj_List adj, Indegree_List indegree)
10 {
11     queue<int> q;
12     // priority_queue<int> q; // If you want in
lexicografic order
13     for (int i = 0; i < indegree.size(); i++) {
14         if (indegree[i] == 0)
15             q.push(i);
16     }
17     vector<int> order;
18
19     while (not q.empty()) {
20         auto a = q.front();
21         q.pop();
22
23         order.push_back(a);
24         for (auto b: adj[a]) {
25             indegree[b]--;
26             if (indegree[b] == 0)
27                 q.push(b);
28         }
29     }
30     return order;
31 }
32
33 int32_t main()
34 {
35
36     Order_List = kahn(adj, indegree);
37     if (Order_List.size() != N) {
38         cout << "IMPOSSIBLE" << endl;
39     }
40     return 0;
41 }

```

2.4 Kth Ancestor

Complexity: $O(n * \log(n))$

Preprocess, then find in $\log n$

```

1 // TITLE: Kth Ancestor
2 // COMPLEXITY:  $O(n * \log(n))$ 
3 // DESCRIPTION: Preprocess, then find in  $\log n$ 
4
5 const int LOG_N = 30;

```

```

6 int get_kth_ancestor(vector<vector<int>> & up, int v,
    int k)
7 {
8     for (int j = 0; j < LOG_N; j++) {
9         if (k & ((int)1 << j)) {
10             v = up[v][j];
11         }
12     }
13     return v;
14 }
15
16 void solve()
17 {
18     vector<vector<int>> up(n, vector<int>(LOG_N));
19
20     for (int i = 0; i < n; i++) {
21         up[i][0] = parents[i];
22         for (int j = 1; j < LOG_N; j++) {
23             up[i][j] = up[up[i][j-1]][j-1];
24         }
25     }
26     cout << get_kth_ancestor(up, x, k) << endl;
27
28 }

```

3 Parser

3.1 Parsing Functions

Complexity:

```

1 // TITLE: Parsing Functions
2
3 vector<string> split_string(const string & s, const
    string & sep = " ") {
4     int w = sep.size();
5     vector<string> ans;
6     string curr;
7
8     auto add = [&](string a) {
9         if (a.size() > 0) {
10             ans.push_back(a);
11         }
12     };
13
14     for (int i = 0; i + w < s.size(); i++) {
15         if (s.substr(i, w) == sep) {
16             i += w-1;
17             add(curr);
18             curr.clear();
19             continue;
20         }
21         curr.push_back(s[i]);
22     }
23     add(curr);
24     return ans;
25 }
26
27 vector<int> parse_vector_int(string & s)
28 {
29     vector<int> nums;
30     for (string x: split_string(s)) {
31         nums.push_back(stoi(x));
32     }
33     return nums;
34 }
35
36 vector<float> parse_vector_float(string & s)
37 {

```

```

    vector<float> nums;
39     for (string x: split_string(s)) {
40         nums.push_back(stof(x));
41     }
42     return nums;
43 }
44
45 void solve()
46 {
47     cin.ignore();
48     string s;
49     getline(cin, s);
50
51     auto nums = parse_vector_float(s);
52     for (auto x: nums) {
53         cout << x << endl;
54     }
55 }

```

4 String

4.1 Z function

Complexity: Z function complexity

z function

```

1 // TITLE: Z function
2 // COMPLEXITY: Z function complexity
3 // DESCRIPTION: z function
4
5 void z_function(string& s)
6 {
7     return;
8 }

```

4.2 String hash

Complexity: $O(n)$ preprocessing, $O(1)$ query

Computes the hash of arbitrary substrings of a given string s.

```

1 // TITLE: String hash
2 // COMPLEXITY:  $O(n)$  preprocessing,  $O(1)$  query
3 // DESCRIPTION: Computes the hash of arbitrary
    substrings of a given string s.
4
5 struct hashes
6 {
7     string s;
8     int m1, m2, n, p;
9     vector<int> p1, p2, sum1, sum2;
10
11     hashes(string s) : s(s), n(s.size()), p1(n + 1),
        p2(n + 1), sum1(n + 1), sum2(n + 1)
12     {
13         srand(time(0));
14         p = 31;
15         m1 = rand() / 10 + 1e9; // 1000253887;
16         m2 = rand() / 10 + 1e9; // 1000546873;
17
18         p1[0] = p2[0] = 1;
19         loop(i, 1, n + 1)
20         {
21             p1[i] = (p * p1[i - 1]) % m1;
22             p2[i] = (p * p2[i - 1]) % m2;
23         }
24
25         sum1[0] = sum2[0] = 0;
26         loop(i, 1, n + 1)
27         {

```

```

28         sum1[i] = (sum1[i - 1] * p) % m1 + s[i - 20] }
29     1];
30         sum2[i] = (sum2[i - 1] * p) % m2 + s[i -
31     1];
32         sum1[i] %= m1;
33         sum2[i] %= m2;
34     }
35 }
36 // hash do intervalo [l, r)
37 int gethash(int l, int r)
38 {
39     int c1 = m1 - (sum1[l] * p1[r - 1]) % m1;
40     int c2 = m2 - (sum2[l] * p2[r - 1]) % m2;
41     int h1 = (sum1[r] + c1) % m1;
42     int h2 = (sum2[r] + c2) % m2;
43     return (h1 << 30) ^ h2;
44 };

```

5 Set

5.1 Ordered Set

Complexity: $O(\log(n))$

```

1 // TITLE: Ordered Set
2 // COMPLEXITY:  $O(\log(n))$ 
3 // DESCRIPTION: Set but you can look witch elements is
4 // in position (k)
5 #include <ext/pb_ds/assoc_container.hpp>
6 #include <ext/pb_ds/tree_policy.hpp>
7 using namespace __gnu_pbds;
8
9 #define ordered_set tree<int, null_type, less<int>,
10 rb_tree_tag, tree_order_statistics_node_update>
11
12 int32_t main() {
13     ordered_set o_set;
14
15     o_set.insert(5);
16     o_set.insert(1);
17     o_set.insert(2);
18     // o_set = {1, 2, 5}
19     5 == *(o_set.find_by_order(2));
20     2 == o_set.order_of_key(4); // {1, 2}

```

5.2 Set

Complexity: Insertion $\log(n)$

Keeps elements sorted, remove duplicates, upper_bound, lower_bound, find, count

```

1 // TITLE: Set
2 // COMPLEXITY: Insertion  $\log(n)$ 
3 // Description: Keeps elements sorted, remove
4 // duplicates, upper_bound, lower_bound, find, count
5 int main() {
6     set<int> set1;
7
8     set1.insert(1); //  $O(\log(n))$ 
9     set1.erase(1); //  $O(\log(n))$ 
10
11     set1.upper_bound(1); //  $O(\log(n))$ 
12     set1.lower_bound(1); //  $O(\log(n))$ 
13     set1.find(1); //  $O(\log(n))$ 
14     set1.count(1); //  $O(\log(n))$ 
15
16     set1.size(); //  $O(1)$ 
17     set1.empty(); //  $O(1)$ 
18
19     set1.clear() //  $O(1)$ 
20     return 0;

```

5.3 Multiset

Complexity: $O(\log(n))$

Same as set but you can have multiple elements with same values

```

1 // TITLE: Multiset
2 // COMPLEXITY:  $O(\log(n))$ 
3 // DESCRIPTION: Same as set but you can have multiple
4 // elements with same values
5 int main() {
6     multiset<int> set1;
7 }

```