```
#include <chrono>
1
     using namespace std;
2
     auto t1 = chrono::high_resolution_clock::now();
3
     auto t2 = std::chrono::high_resolution_clock::now();
4
5
     auto duration = std::chrono::duration_cast<std::chrono::microseconds>( t2 - t1 ).count();
6
7
     cout << duration;</pre>
8
9
10
     void countSmallerRight(int A[], int len)
11
     {
12
         set<int> s;
13
         int countSmaller[len];
14
         for (int i = len - 1; i >= 0; i--) {
15
              s.insert(A[i]);
16
              auto it = s.lower_bound(A[i]);
17
             countSmaller[i] = distance(s.begin(), it);
18
         }
19
20
         for (int i = 0; i < len; i++)</pre>
21
              cout << countSmaller[i] << " ";</pre>
22
     }
23
24
25
     //DSU
26
     ll r[limit];
27
     ll parent[limit];
28
29
     ll find(ll x){
30
         if(parent[x]==x) return x;
31
         return parent[x] = find(parent[x]);
32
     }
33
34
     void un(ll x, ll y){
35
         ll\ xRoot = find(x);
36
         ll yRoot = find(y);
37
         if(r[xRoot]<r[yRoot]){</pre>
38
              ll temp = yRoot;
39
             yRoot = xRoot;
40
             xRoot = temp;
41
         }
42
         parent[yRoot] = xRoot;
43
         if(r[xRoot] == r[yRoot]) r[xRoot]++;
44
45
     }
```

46

```
47
     //Hash
48
     struct VectorHasher {
49
         int operator()(const vector<int> &V) const {
50
              int hash = V.size();
51
              for(auto &i : V) {
52
                  hash ^= i + 0x9e3779b9 + (hash << 6) + (hash >> 2);
              }
53
54
              return hash;
55
         }
56
     };
57
58
     //Sqrt Decomposition
59
     #define limit 1000100
60
     ll arr[limit];
61
     ll bloki[1100];
     ll dolzinaBloka = sqrt(limit);
62
63
     void update(int i, ll val){
64
         arr[i] = min(val, arr[i]);
65
         int blokId = i / dolzinaBloka;
66
         bloki[blokId] = min(bloki[blokId], val);
67
     }
68
69
70
     int query(int L, int R){
71
         ll res = inf*1LL*inf;
72
73
         int p = L;
74
         //leva stran
75
         while(p < R and p % dolzinaBloka){</pre>
76
              res = min(res, arr[p]);
77
             p++;
         }
78
         //bloki
79
         while(p + dolzinaBloka <= R){</pre>
80
              res = min(res, bloki[p/dolzinaBloka]);
81
82
              p++;
         }
83
84
         //desno
85
         while(p <= R){</pre>
86
              res = min(res, arr[p]);
87
88
              p++;
89
         }
90
         //printf("query %lld %lld -> %lld\n", L, R, res);
91
         return res;
92
     }
93
94
95
     //Matrika
96
     struct Matrix{
```

```
97
          long long a[2][2];
          Matrix operator *(Matrix other){
98
               Matrix res = \{\{\{0,0\},\{0,0\}\}\}\};
99
               loop(i, 2){
100
                   loop(k, 2){}
101
102
                        loop(j, 2){}
103
                            res.a[i][k] += a[i][j] * other.a[j][k];
104
                            res.a[i][k] %= mod;
105
                       }
                   }
106
107
               }
108
               return res;
          }
109
110
      };
111
112
113
      //Teorija Stevil
114
115
      const int MOD = 998244353;
      const int N = 200043;
116
117
118
      int add(int x, int y)
119
120
          return (x + y) \% MOD;
121
122
123
      int sub(int x, int y)
124
          return add(x, MOD - y);
125
126
      }
127
      int mul(int x, int y)
128
129
130
          return (x * 1ll * y) % MOD;
131
      }
132
133
      int binpow(int x, int y)
134
      {
135
          int z = 1;
136
          while(y > 0)
137
138
               if(y % 2 == 1)
                   z = mul(z, x);
139
140
              x = mul(x, x);
              y /= 2;
141
          }
142
143
          return z;
144
      }
145
146
      int inv(int x)
```

```
147
      {
          return binpow(x, MOD - 2);
148
149
      }
150
151
      int fact[N];
152
      int C(int n, int k)
153
154
          return mul(fact[n], inv(mul(fact[k], fact[n - k])));
155
156
      }
157
158
159
      //Stringi
160
      vector<int> z_function(string s) {
161
          int n = (int) s.length();
162
          vector<int> z(n);
          for (int i = 1, l = 0, r = 0; i < n; ++i) {
163
164
              if (i <= r)
                  z[i] = min (r - i + 1, z[i - l]);
165
              while (i + z[i] < n \&\& s[z[i]] == s[i + z[i]])
166
                  ++z[i];
167
              if (i + z[i] - 1 > r)
168
                  l = i, r = i + z[i] - 1;
169
170
          }
171
          return z;
      }
172
173
      //Template
174
175
      using namespace std;
      typedef long long ll;
176
177
      #define pll pair<ll, ll>
      #define loop(i, n) for(ll i = 0; i < n; i++)</pre>
178
179
      #define FOR(i,n,m) for(ll i = n; i <= m; i++)</pre>
      #define isIn(vec, item) find(vec.begin(), vec.end(), item) != vec.end()
180
181
      #define fs first
      #define sc second
182
      #define pb push_back
183
      #define mp make_pair
184
      #define all(v) v.begin(),v.end()
185
186
      #define inf 1000000005
187
      #define mod 1000000007
      #define print(v) for(auto e : v) cout << e << " "; cout << endl;</pre>
188
189
190
      //Treap (Ne uporabljaj, ce se imas rad)
191
192
193
194
195
      /*
      VNAPREJ DOLIČI VELIKOST
196
```

```
197
     VEKTORJA, KAMOR SHRANJUJEŠ
198
     TREAPE
     */
199
     mt19937 mt_rand(time(0));
200
201
202
     struct Treap {
203
          int data, priority;
204
          vector<Treap*> kids;
          int subtreeSize, sum, toProp;
205
206
207
          Treap(int data);
208
     };
209
210
     int size(Treap *me) {
211
          return me == NULL ? 0 : me->subtreeSize;
212
     }
213
214
     void recalc(Treap *me) {
215
          if (me==NULL) return;
          me->subtreeSize = 1;
216
          me->sum = me->data + me->toProp*size(me);
217
          for (Treap* t:me->kids) if (t != NULL) me->subtreeSize += t->subtreeSize;
218
219
          for (Treap* t:me->kids) if (t != NULL) me->sum += t->sum+t->toProp*size(t);
220
     }
221
222
     void prop(Treap *me) {
223
          if (me==NULL) return;
          if (me->toProp == 0) return;
224
225
          for (Treap *t:me->kids) if (t != NULL) t->toProp += me->toProp;
          me->data+=me->toProp;
226
227
          me->toProp=0;
          recalc(me);
228
229
     }
230
231
     Treap* merge(Treap *l, Treap *r) {
          if (l==NULL) return r;
232
233
          if (r==NULL) return l;
          prop(l); prop(r);
234
          if (l->priority < r->priority) {
235
236
              l->kids[1]=merge(l->kids[1], r);
237
              recalc(l);
              return l;
238
          }
239
240
          else {
              r->kids[0]=merge(l, r->kids[0]);
241
              recalc(r);
242
243
              return r;
          }
244
245
     }
246
```

```
247
      vector<Treap*> split(Treap *me, int nInLeft) {
          if (me == NULL) return {NULL, NULL};
248
          prop(me);
249
          if (size(me->kids[0])>=nInLeft) {
250
251
              vector<Treap*> leftRes=split(me->kids[0], nInLeft);
252
              me->kids[0]=leftRes[1];
              recalc(me);
253
254
              return {leftRes[0], me};
255
          }
          else {
256
              nInLeft = nInLeft - size(me->kids[0]) - 1;
257
258
              vector<Treap*> rightRes = split(me->kids[1], nInLeft);
259
              me->kids[1] = rightRes[0];
260
              recalc(me);
261
              return {me, rightRes[1]};
          }
262
263
          return {NULL, NULL};
264
      }
265
      Treap::Treap(int _data) {
266
          kids={NULL, NULL};
267
          data = _data;
268
          toProp = 0;
269
270
          recalc(this);
271
272
          priority = mt_rand();
273
      }
274
275
      Treap* rangeAdd(Treap* t, int l, int r, int toAdd) {
          vector<Treap*> a=split(t, l), b=split(a[1], r-l+1);
276
277
          b[0]->toProp+=toAdd;
          return merge(a[0], merge(b[0], b[1]));
278
279
      }
280
      void inOrderTraversal(Treap *t){
281
          if(t == NULL) return;
282
          //Left, Node, Right
283
          inOrderTraversal(t->kids[0]);
284
          cout << t->data << endl;</pre>
285
          inOrderTraversal(t->kids[1]);
286
      }
287
288
289
      //HEAVY-LIGHT DECOMPOSITION
290
      vector<int> parent, heavy, head, pos, depth;
291
      vector<int> children[limit];
292
293
      int curPos=1;
294
295
      int dfs(int node){
296
          int size = 1;
```

```
297
          int maxS = 0;
298
          int best = -1;
          for(int c : children[node]){
299
              depth[c] = depth[node] + 1;
300
              int ret = dfs(c);
301
              size += ret;
302
303
              if(maxS < ret){</pre>
304
                  best = c;
305
                  maxS = ret;
306
              }
307
          }
308
          //nastavi heavy edge
309
          heavy[node] = best;
310
311
312
          return size;
313
      }
314
315
      void decompose(int node, int h){
          if(DEBUG)printf("decom %d, heavy %d\n", node, heavy[node]);
316
317
          head[node] = h;
          pos[node] = curPos++;
318
319
          if(heavy[node] != -1) decompose(heavy[node], h); //isti component
320
          for(int c : children[node]){
321
322
              if(c == heavy[node]) continue;
              decompose(c, c);
          }
      }
```

PDF document made with CodePrint using Prism