Branch: **master** ▾ | **Find file** | **Copy path**

blog-codes / src / codechef_FLIPCOIN.cpp

kartikkukreja Create codechef_FLIPCOIN.cpp

`eb6d92f` on Jan 10, 2015

1 contributor

Raw | **Blame** | **History**

165 lines (134 sloc) 4.07 KB

```
1   #include <cstdio>
2
3   struct SegmentTreeNode {
4           int start, end; // this node is responsible for the segment [start...end]
5           int count;
6           bool pendingUpdate;
7
8           SegmentTreeNode() : count(0), pendingUpdate(false) {}
9
10          void assignLeaf(bool value) {}
11
12          void merge(SegmentTreeNode& left, SegmentTreeNode& right) {
13                  count = (left.pendingUpdate ? (left.end - left.start + 1 - left.cou
14                              + (right.pendingUpdate ? (right.end - right.start + 1 -
15          }
16
17          int query() {
18                  return count;
19          }
20
21          bool hasPendingUpdate() {
22                  return pendingUpdate;
23          }
24
25          void applyPendingUpdate() {
26                  count = (end - start + 1) - count;
27                  pendingUpdate = false;
28          }
29
30          void addUpdate(bool value) {
31                  pendingUpdate = !pendingUpdate;
32          }
33
34          bool getPendingUpdate() {
```

```
35                return true;
36        }
37};
38
39template<class InputType, class UpdateType, class OutputType>
40class SegmentTree {
41        SegmentTreeNode* nodes;
42        int N;
43
44public:
45        SegmentTree(InputType arr[], int N) {
46                this->N = N;
47                nodes = new SegmentTreeNode[getSegmentTreeSize(N)];
48                buildTree(arr, 1, 0, N-1);
49        }
50
51        ~SegmentTree() {
52                delete[] nodes;
53        }
54
55        // get the value associated with the segment [start...end]
56        OutputType query(int start, int end) {
57                SegmentTreeNode result = query(1, start, end);
58                return result.query();
59        }
60
61        // range update: update the range [start...end] by value
62        // Exactly what is meant by an update is determined by the
63        // problem statement and that logic is captured in segment tree node
64        void update(int start, int end, UpdateType value) {
65                update(1, start, end, value);
66        }
67
68private:
69        void buildTree(InputType arr[], int stIndex, int start, int end) {
70                // nodes[stIndex] is responsible for the segment [start...end]
71                nodes[stIndex].start = start, nodes[stIndex].end = end;
72
73                if (start == end) {
74                        // a leaf node is responsible for a segment containing only
75                        nodes[stIndex].assignLeaf(arr[start]);
76                        return;
77                }
78
79                int mid = (start + end) / 2,
80                        leftChildIndex = 2 * stIndex,
81                        rightChildIndex = leftChildIndex + 1;
82
```

```
83                buildTree(arr, leftChildIndex, start, mid);
84                buildTree(arr, rightChildIndex, mid + 1, end);
85                nodes[stIndex].merge(nodes[leftChildIndex], nodes[rightChildIndex])
86        }
87
88        int getSegmentTreeSize(int N) {
89                int size = 1;
90                for (; size < N; size <<= 1);
91                return size << 1;
92        }
93
94        SegmentTreeNode query(int stIndex, int start, int end) {
95                if (nodes[stIndex].start == start && nodes[stIndex].end == end) {
96                        SegmentTreeNode result = nodes[stIndex];
97                        if (result.hasPendingUpdate())
98                                result.applyPendingUpdate();
99                        return result;
100               }
101
102               int mid = (nodes[stIndex].start + nodes[stIndex].end) >> 1,
103                       leftChildIndex = stIndex << 1,
104                       rightChildIndex = leftChildIndex + 1;
105               SegmentTreeNode result;
106
107               if (start > mid)
108                       result = query(rightChildIndex, start, end);
109               else if (end <= mid)
110                       result = query(leftChildIndex, start, end);
111               else {
112                       SegmentTreeNode leftResult = query(leftChildIndex, start, m
113                                                          rightResult = query(rightCh
114                       result.start = leftResult.start;
115                       result.end = rightResult.end;
116                       result.merge(leftResult, rightResult);
117               }
118
119               if (nodes[stIndex].hasPendingUpdate()) {
120                       result.addUpdate(nodes[stIndex].getPendingUpdate());
121                       result.applyPendingUpdate();
122               }
123               return result;
124       }
125
126       void update(int stIndex, int start, int end, UpdateType value) {

127               if (nodes[stIndex].start == start && nodes[stIndex].end == end) {
128                       nodes[stIndex].addUpdate(value);
129                       return;
130               }
```

```
131
132                 int mid = (nodes[stIndex].start + nodes[stIndex].end) >> 1,
133                         leftChildIndex = stIndex << 1,
134                         rightChildIndex = leftChildIndex + 1;
135
136                 if (start > mid)
137                         update(rightChildIndex, start, end, value);
138                 else if (end <= mid)
139                         update(leftChildIndex, start, end, value);
140                 else {
141                         update(leftChildIndex, start, mid, value);
142                         update(rightChildIndex, mid+1, end, value);
143                 }
144                 nodes[stIndex].merge(nodes[leftChildIndex], nodes[rightChildIndex])
145         }
146 };
147
148 bool tailsUp[100005];
149
150 int main() {
151         int N, Q, cmd, A, B;
152
153         scanf("%d %d", &N, &Q);
154         SegmentTree<bool,bool,int> st(tailsUp, N);
155         while (Q--) {
156                 scanf("%d %d %d", &cmd, &A, &B);
157                 if (cmd == 0)
158                         st.update(A, B, true);
159                 else
160                         printf("%d\n", st.query(A, B));
161         }
162
163         return 0;
164 }
```