

# CS381 Homework 5

Out: Nov. 14, Tuesday

Due: Nov. 21, Tuesday, at the beginning of (or before) class  
Submissions will not be accepted afterwards.

## Instructions:

**Read the course policy (including academic dishonesty) in the course webpage at <http://www.cs.purdue.edu/homes/gopal/cs381>.**

Text refers to the Introduction to Algorithms (second edition) book.

**Justify your answers. Show appropriate work.**

**Reading:** Appendices B.4 and B.5 (intro to graphs and trees) and Chapters 21 and 22 of Text.

## Problem 1

In this problem we will see an interesting use of disjoint set data structure. We are given a sequence of  $n$  numbers  $x_1, x_2, \dots, x_n$  and a set of queries of the following type: given  $x_i$  and  $x_j$  ( $i < j$ ) you want to return  $\min_{i \leq k \leq j} \{x_k\}$ , i.e., the minimum value between  $x_i$  and  $x_j$  in the sequence (including  $x_i$  and  $x_j$ ). We will call the above query as a *min-query* with parameters  $x_i$  and  $x_j$ . The goal is to design a data structure that takes  $O(n)$  space and answers a set of  $m$  min-queries in  $O((n+m) \log^* n)$  time. (Note that it is easy to take  $O(n^2)$  space to answer all the  $m$  queries in  $O(m)$  time — simply compute the minimum element for all  $\binom{n}{2}$  pairs and store the value in a 2-dimensional array of size  $n \times n$ .) We will not worry about the time needed to construct this data structure (this is the preprocessing cost). We only care that the data structure takes  $O(n)$  space and answers all  $m$  min-queries (which are given as part of the input) in  $O((n+m) \log^* n)$  time.

1. We will first build a binary tree  $T$  (let's call it the query tree) as described by the following recursive definition. The root is the minimum of all the elements. Let  $x_r$  be this value. (Clearly,  $x_r$  is the value that should

be returned for values  $x_i$  and  $x_j$  where  $i < r$  and  $j \geq r$ .) The left subtree is the query tree for the elements  $x_1, \dots, x_{r-1}$  and the right subtree is the query tree for the elements  $x_{r+1}, \dots, x_n$ .

(a) For the sequence: 5, 7, 3, 2, 9, 8, 4, 10, 1, 6, 5 build the query tree. (5 points)

(b) Describe how to answer a particular min-query using the above tree in  $O(h)$  time, where  $h$  is the height of the query tree  $T$ . Show that the time needed to answer a query can be as much as  $O(n)$  in the worst-case. (Hint: Give an example.) What is the space taken by this data structure (i.e., the query tree)? (10 points)

2. We next consider a strategy that uses disjoint set data structure to reduce the time taken to answer a query while keeping the space to be  $O(n)$ .

We run the following recursive procedure on the query tree  $T$  with the initial call  $query(root)$  (i.e., on the root node of  $T$ ). Initially we assume that all nodes are colored blue, i.e.,  $color[u] = \text{blue}$  for all  $u \in T$ .

**query**( $u$ ) //  $u$  is a node in  $T$

```

1 Make-set( $u$ )
2 ancestor[find-set( $u$ )] =  $u$  // ancestor is an array of size  $n$ 
3 for each child  $v$  of  $u$  in  $T$ 
    3.1 query( $v$ )
    3.2 union( $u, v$ )
    3.3 ancestor[find-set( $u$ )] =  $u$ 
4 color[ $u$ ] = red
5 for each node  $v$  such that there is a min-query with parameters  $u$  and  $v$ 
    5.1 if color[ $v$ ] = red then
        answer this min-query by returning ancestor[find-set( $v$ )]

```

(a) Show the disjoint set data structure created by the above algorithm when run on the query tree of example 1(a). Use union by weight strategy for union and path compression for doing find. Run the algorithm and show how it answers a min-query when invoked with parameters 7 and 10. (15 points)

(b) Argue that the total time to answer  $m$  queries on a sequence of  $n$  elements by the above algorithm is  $O((n + m) \log^* n)$  time. (10 points)

(c) Give an argument as to why the above algorithm correctly answers queries. (Hint: Use induction on the height of the tree  $T$ . Thus your induction hypothesis will be: The algorithm answers all queries correctly when run on a query tree of height  $< h$ . Argue the correctness for  $h$ . For the induction step you might want to consider three cases depending on the

parameters of the min-query —  $(u, v)$ : (1) one of them is the root (2) both of them are in the left subtree or both of them are in the right subtree (3) one is in the left subtree and the other is in the right subtree). (15 points)

## Problem 2

The biconnected components of a graph  $G = (V, E)$  is a partition of the edges into sets such that the graph formed by each set of edges is biconnected. (As mentioned in class, a graph is biconnected if it does not contain any cut-vertex.)

(a) Modify the algorithm that we studied in class that finds the cut-vertices to find biconnected components also. Note that you should first explain the algorithm that we studied in class for the cut-vertices and show how you will modify it so that it will output the biconnected components also. Note that your algorithm should run in  $O(|V| + |E|)$  time. Give a clear description of your algorithm and the pseudocode and argue the running time bound. (15 points)

(b) A cut-edge of a graph  $G = (V, E)$  is an edge whose removal partitions the graph into two or more connected components (note that some of these components can be single vertex). Give a  $O(|V| + |E|)$  time algorithm to find all the cut-edges of  $G$ . Give a clear description of your algorithm and the pseudocode and argue the running time bound. (Hint: Again this is a modification of the algorithm to find cut-vertices. What is the condition for an edge to be a cut-edge? Can it lie on a simple cycle of  $G$ ?) (15 points)

(c) Run your algorithms of parts (a) and (b) on the undirected graph given in Figure 22.3(a) of the Text (page 533). Show the main steps and output the cut vertices, the cut-edges, and the biconnected components of the graph. (15 points)