

## CSCI3160: Regular Exercise Set 4

Prepared by Yufei Tao

**Problem 1.** Recall that a *tree* is a connected graph without cycles. Prove:

- Every tree has at least a leaf node, i.e., a node with degree 1 (i.e., a node incident to only one edge).
- Every tree with  $n$  nodes has precisely  $n - 1$  edges.

**Solution.** Proof of the first statement: Start from an arbitrary node  $u$ . If  $u$  is not a leaf, then walk across one of its edges to reach a neighbor node, and delete the edge that was crossed. Then, set  $u$  to that neighbor node, and repeat the process. In this process, every node will be encountered at most once (if a node is seen twice, there must be a cycle, and hence cause a contradiction). Since the tree has a finite number of nodes, the process must come to an end eventually. The last node reached must be a leaf.

Proof of the second statement: We will prove the claim by induction on  $n$ . When  $n = 2$ , the tree has only one edge; and the claim is clearly true. Next, assuming the claim's correctness for  $n = k$ , we will prove that it also holds for any tree  $T$  with  $n = k + 1$  nodes. From the first statement, we know that there must be a leaf node  $u$  in  $T$ . Remove  $u$  from  $T$  and the only edge incident to  $u$ . The remaining tree has  $k$  nodes which, by the inductive assumption, must have  $k - 1$  edges. It thus follows that  $T$  has  $k$  edges.

**Problem 2.** Let  $G$  be a simple graph with  $n$  vertices and  $n - 1$  edges. Prove: if  $G$  is connected (i.e., a path exists between any two vertices in  $G$ ), then  $G$  must be a tree.

**Solution.** Consider an arbitrary spanning tree  $T$  of  $G$ . Because  $G$  is connected,  $T$  must include all the  $n$  vertices of  $G$ . From the statements of Problem 1, we know that  $T$  must have  $n - 1$  edges. This means that  $T$  has all the edges of  $G$  and, hence,  $G = T$ .

**Problem 3 (one for one, still a tree).** Let  $T$  be a tree. Add a new edge between two vertices in  $T$ ; this gives us a graph  $G$  with a cycle  $cyc$ . Now, remove from  $G$  an arbitrary edge  $e'$  of  $cyc$ ; let  $G'$  be the graph thus obtained. Prove:  $G'$  is a tree.

**Solution.** Let  $n$  be the number of vertices in  $T$ . It is clear that  $G'$  has  $n - 1$  edges. Next, we will prove that  $G'$  is connected (i.e., a path exists between any two of its vertices), which (by the statement of Problem 2) shows that  $G'$  is a tree.

Let  $u$  and  $v$  be two arbitrary vertices in  $G'$ . Consider an arbitrary path  $\pi$  from  $u$  to  $v$  in  $G$  (this path must exist because  $G$  is connected). If  $\pi$  does not use edge  $e'$  (i.e., the edge deleted), then  $\pi$  exists in  $G'$  and, hence,  $u$  and  $v$  are connected in  $G'$ . Now, consider the case where  $e'$  is in  $\pi$ . Assume, without loss of generality, that  $e' = \{u', v'\}$  and that  $\pi$  goes from  $u$  to  $u'$ , crosses  $e'$  to  $v'$ , and then continues onto  $v$ . This means that, in  $G'$ ,  $u$  is connected to  $u'$  and  $v$  is connected to  $v'$ . It remains to prove that  $u'$  is connected to  $v'$  in  $G'$ , which will tell us that  $u$  is connected to  $v$  in  $G'$ .

Remember that  $e'$  is in the cycle  $cyc$ . This implies that, in  $cyc$ , we can find a path from  $u'$  to  $v'$  that does not pass through  $e'$ . This path must still remain in  $G'$ . Therefore, we conclude that  $u'$  is connected to  $v'$  in  $G'$ .

**Problem 4.** Let  $S$  be a set of integer pairs of the form  $(id, v)$ . We will refer to the first field as the *id* of the pair, and the second as the *key* of the pair. Design a data structure that supports the following operations:

- Insert: add a new pair  $(id, v)$  to  $S$  (you can assume that  $S$  does not already have a pair with the same id).
- Delete: given an integer  $t$ , delete the pair  $(id, v)$  from  $S$  where  $t = id$ , if such a pair exists.
- DeleteMin: remove from  $S$  the pair with the smallest key, and return it. .

Your structure must consume  $O(n)$  space, and support all operations in  $O(\log n)$  time where  $n = |S|$ .

**Solution.** Maintain  $S$  in two binary search trees  $T_1$  and  $T_2$ , where the pairs are indexed on ids in  $T_1$ , and on keys in  $T_2$ . We support the three operations as follows:

- Insert: simply insert the new pair  $(id, v)$  into both  $T_1$  and  $T_2$ .
- Delete: first find the pair with id  $t$  in  $T_1$ , from which we know the key  $v$  of the pair. Now, delete the pair  $(t, v)$  from both  $T_1$  and  $T_2$ .
- DeleteMin: find the pair with the smallest key  $v$  from  $T_2$  (which can be found by continuously descending into left child nodes). Now we have its id  $t$  as well. Remove  $(t, v)$  from  $T_1$  and  $T_2$ .

**Problem 5.** Prove: in a weighted undirected graph  $G = (V, E)$  where all the edges have distinct weights, the minimum spanning tree (MST) is unique.

**Solution.** We will prove that the tree  $T$  returned by the Prim's algorithm is the only MST. Set  $n = |V|$ . Let  $e_1, e_2, \dots, e_{n-1}$  be the sequence of edges that the algorithm adds to  $T$ . Suppose, on the contrary, that there is another MST  $T'$ . Let  $k$  be the smallest  $i$  such that  $e_i$  is not in  $T'$ .

- Case 1:  $k = 1$ . This means that  $e_1$ , which is the edge with the smallest weight, is not in  $T'$ . Add  $e_1$  to  $T'$  to create a cycle, and remove from the cycle the edge with the largest weight. This create another spanning tree whose cost is strictly smaller than  $T'$  (remember: all the edges are distinct), contradicting the fact that  $T'$  is an MST.
- Case 2:  $k > 1$ . Recall that edges  $e_1, e_2, \dots, e_{k-1}$  form a tree. Let  $S$  be the set of vertices in this tree. Add  $e_k = \{u, v\}$  into  $T'$  to create a cycle. Suppose  $u \in S$ ; it follows that  $v \notin S$ . Let us walk on the cycle from  $v$ , by going into  $S$ , traveling within  $S$ , and stopping as soon as we exit  $S$ . Let  $\{u', v'\}$  be the last edge crossed (namely, one of  $u', v'$  is in  $S$ , while the other one is not). By the way Prim's algorithm runs and the fact that all edges have distinct weights, we know that  $\{u, v\}$  has a smaller weight than  $\{u', v'\}$ . Thus, removing  $\{u', v'\}$  from  $T'$  gives spanning tree with strictly smaller cost, which creates a contradiction.

**Problem 6.** Describe how to implement the Prim's algorithm on a graph  $G = (V, E)$  in  $O((|V| + |E|) \cdot \log |V|)$  time.

**Solution.** Remember that the algorithm incrementally grows a tree  $T$  which in the end becomes an MST. Let  $S$  be the set of vertices that are currently in  $T$ . At all times, the algorithm maintains, for every vertex  $v \in V \setminus S$ , its lightest cross edge  $best\_cross(v)$  and the weight of this edge.

We maintain a set  $P$  of triples, one for every vertex  $u \in V \setminus S$ . Specifically, the triple of  $u$  has the form  $(u, v, t)$ , indicating that  $best\_cross(u)$  is the edge  $\{u, v\}$  (i.e.,  $v \in S$ ), whose weight is  $t$ . We need the following operations on  $P$ :

- Insert: add a triple  $(u, v, t)$  to  $P$ .
- DecreaseKey: given a vertex  $v' \in S$  and a cross edge  $\{u, v'\}$  (i.e.,  $u \notin S$ ), this operation does the following. First, fetch the triple  $(u, v, t)$ . Then, compare  $t$  to the weight  $t'$  of  $\{u, v'\}$ . If  $t' < t$ , update the triple  $(u, v, t)$  to  $(u, v', t')$ ; otherwise, do nothing.
- DeleteMin: Remove from  $P$  the triple  $(u, v, t)$  with the smallest  $t$ .

We can store  $P$  in a data structure of Problem 2 which supports all operations in  $O(\log |V|)$  time (note: DecreaseKey can be implemented as a Delete followed by an Insert). Besides the above structure, we also store an array  $A$  of length  $|V|$  to so that we can query in constant time, for any vertex  $v \in V$ , whether  $v$  is in  $S$  currently.

Now we can implement the algorithm as follows. Let  $\{v_1, v_2\}$  be an edge with the smallest weight in  $G$ . The set  $S$  contains only  $v_1$  and  $v_2$  at this point. For every vertex  $u \in V \setminus S$  where  $S = \{v_1, v_2\}$ , we check whether  $u$  has cross edges to  $v_1$  and  $v_2$ . If neither edge exists, insert triple  $(u, nil, \infty)$  to  $P$ . Otherwise, suppose without loss of generality that  $\{u, v_1\}$  is the lighter cross edge of  $u$  with weight  $t$ ; insert a triple  $(u, v_1, t)$  into  $P$ .

Repeat the following until  $P$  is empty:

- Perform a DeleteMin to obtain a triple  $(u, v, t)$ .
- Recall that  $u$  should be added to  $S$ , which may need to change the cross edges of some other vertices. To implement this, for every edge  $(u, u')$  of  $u$  where  $u' \notin S$ , perform DecreaseKey with  $u'$  and  $\{u, u'\}$ .