COMP3506/7505: Regular Exercise Set 10

Prepared by Yufei Tao

Problem 1. Let T be a (2, 3)-tree on a set S of integers. Suppose that each node in the tree stores a pointer to its parent. You are given the leftmost leaf z of T, and asked to report the k smallest integers in T. Describe an algorithm to do so in $O(k)$ time.

Solution. Recall that there is a natural ordering on the leaf nodes of T. Specifically, if a leaf node is to the left of another in the ordering, the data elements in the former leaf are strictly smaller than those in the latter. Let us first describe an operation called *move-right*(v) which, given a leaf node v, returns the leaf to the right of v in the ordering. Ascend from v to the lowest ancestor p that has a routing element e strictly larger than all the data elements in v. If p is not found, then v is already the right most leaf; and the operation returns nothing. Otherwise, the operation returns the left most leaf in the subtree of p that e corresponds to.

Returning to the original problem. Set v to z. Repeat the following until k integers have been reported:

- Report all the data elements in v.
- Move v to the leaf on its right by calling *move-right*(v).

Let us now analyze the query time. In total, we visit at most $k/2$ leaf nodes, at most $k/2^2$ level-1 nodes, at most $k/2^3$ level-2 nodes, ... Hence, the total number of nodes visited is $O(k)$. To account for the cost of *move-right*, every time we ascend from a node u to its parent p, we charge the cost on the edge between u and p; similarly, every time we descend from p to u, also charge the cost in the same way. Thus, the total cost is $O(k)$ because each node has at most 3 edges.

Problem 2. Let $G = (V, E)$ be a directed graph. Suppose that we perform BFS starting from a source vertex s, and obtain a BFS-tree T. For any vertex $v \in V$, denote by $l(v)$ the level of v in the BFS-tree. Prove that BFS en-queues the vertices v of V in non-descending order of $l(v)$.

Solution. Take any vertices u, v such that $l(u) > l(v)$. Let $v_1, v_2, ..., v_{l(v)}$ be the vertices on the path from the root to v in T; note that $v_1 = s$ and $v_{l(v)} = v$. Let $u_1, u_2, ..., u_{l(v)}$ be the last $l(v)$ vertices on the path from the root to u in T; note that $u_1 \neq s$ and $u_{l(v)} = u$. It thus follows that v_1 is en-queued before u_1. Remember that BFS en-queues v_2 when de-queuing v_1, and similarly, en-queues u_2 when de-queuing u_1. By the FIFO property of the queue, we know that v_2 is en-queued before u_2. By the same reasoning, v_3 is en-queued before u_3, v_4 before u_4, etc. This means that v is before u.

Problem 3. Let $G = (V, E)$ be a directed graph. Suppose that we perform BFS starting from a source vertex s, and obtain a BFS-tree T. For any vertex $v \in V$, prove that the path from s to v in T is a shortest path from s to v in G.

Solution. We will instead prove the following claim: *all the vertices with shortest path distance l from s are at level l* (recall that the root is at level 0). This will establish the conclusion in Problem 3 because the path from s to a level-l node v in T has length l.

We will prove the claim by induction on l. The base case where $l = 0$ is obviously true.

Assuming that the claim holds for all $l \le k-1$ ($k \ge 1$), next we prove that the claim is also true for $l = k$. Let v be a vertex with shortest path distance k from s. Consider all the shortest paths from s to v. From every such shortest path, take the vertex immediately before v (i.e., the predecessor of v in that path), and put that vertex into a set. Let S be the set of all such "predecessors of v" collected. Let u be the vertex in S that is the earliest one entering the queue. We know that the shortest path distance from s to u is $k-1$. It thus follows from the inductive assumption that u is at level $k-1$ of T.

Consider the moment when u is removed from the queue in BFS. We will argue that the color of v must be white. Hence, BFS makes v a child of u, thus making v at level k.

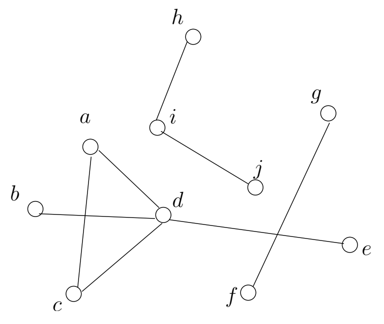
Suppose for contradiction that the color of v is gray or black. This means that v has been put into the queue when another vertex u' was de-queued earlier. From the conclusion of Problem 2 and the definition of u, we know that $l(u') < l(u)$. From the inductive assumption, this means that the shortest path distance of u' from s that is less than $k-1$, implying that the shortest path distance from s to v is less than k, thus giving a contradiction.

Problem 4. Let $G = (V, E)$ be an undirected graph. We will denote an edge between vertices u, v as $\{u, v\}$. Next, we define the single source shortest path (SSSP) problem on G. Define a *path* from s to t as a sequence of edges $\{v_1, v_2\}, \{v_2, v_3\}, ..., \{v_t, v_{t+1}\}$, where $t \ge 1$, $v_1 = s$, and $v_{t+1} = t$. The *length* of the path equals t. Then, the SSSP problem gives a source vertex s, and asks to find shortest paths from s to all the other vertices in G. Adapt BFS to solve this problem in $O(|V| + |E|)$ time. Once again, you need to produce a BFS tree where, for each vertex $v \in V$, the path from the root to v gives a shortest path from s to v.

Solution. Same as BFS, except that when a vertex v is de-queued, we inspect all its neighbors (as opposed to its out-neighbors as in the directed version).

Problem 5 (Connected Components). Let $G = (V, E)$ be an undirected graph. A *connected component* (CC) of G includes a set $S \subseteq V$ of vertices such that

- For any vertices $u, v \in S$, there is a path from u to v, and a path from v to u.
- (Maximality) It is not possible to add any vertex into S while still ensuring the previous property.



For example, in the above graph, $\{a, b, c, d, e\}$ is a CC, but $\{a, b, c, d\}$ is not, and neither is $\{g, f, e\}$.

Prove: Let S_1, S_2 be two CCs. Then, they must be disjoint, i.e., $S_1 \cap S_2 = \emptyset$.

Solution. Suppose that a vertex v is in $S_1 \cap S_2$. Then, for any vertex $u_1 \in S_1$ and $u_2 \in S_2$, we know:

- There is a path from u_1 to u_2 by way of v.
- There is a path from u_2 to u_1 by way of v.

This violates the fact that S_1 and S_2 must be maximal.

Problem 6. Let $G = (V, E)$ be an undirected graph. Describe an algorithm to divide V into a set of CCs. For example, in the example of Problem 5, your algorithm should return 3 CCs: $\{a, b, c, d, e, \}, \{g, f\}$, and $\{h, i, j\}$.

Solution. Run BFS starting from an arbitrary vertex in V. All the vertices in the BFS-tree constitute the first CC. Then, start another BFS from an arbitrary vertex that is still white. All the vertices in this BFS-tree constitute another CC. Repeat this until V has no more white vertices.

Problem 7. Recall that, in the DFS algorithm, after we have grown a DFS-tree, we may need to re-start from an arbitrary vertex that remains white. Multiple re-starts may be necessary throughout the algorithm. Describe how to find the re-starting white vertices efficiently so that the overall execution time of the algorithm is $O(|V| + |E|)$.

Solution. At the beginning of the algorithm, initialize an array C of size $|V|$, such that $C[i]$ remembers the color of vertex i ($i \in [1, |V|]$). Whenever a vertex changes its color in the algorithm, update the array. When the first re-start is needed, scan C from the beginning and stop as soon as finding the first white vertex. At the next re-start, continue scanning C from the last position, and stop as soon as the next white vertex is found. Overall, the array C is scanned only once. Thus, the additional overhead (which includes both the scanning and color updating) is $O(|V| + |E|)$.