COMP2007 Assignment 1

Example Solutions

September 10, 2017

1. a. Description of how your algorithm works (in plain English)

We load the input into an adjacency list then for each query (u, v) perform a breadth first search (BFS) rooted at u. Whenever that search visits a vertex for the first time check if that vertex is v; if it is then print 1 and terminate the BFS. If the BFS finishes normally without finding v then print 0.

1. b. Argue why your algorithm is correct

Suppose there is a path of length ℓ between u and v, say $u, p_1, p_2, \ldots, p_{\ell-1}, v$. Then in the first iteration of the BFS u will be visited and it's neighbors (including p_1) will be added to the queue of vertices to visit. When p_1 is visited it's neighbors (including p_2) will be added to the queue, and so on until visiting $p_{\ell-1}$ adds v to the queue. The algorithm will visit everything on the queue before terminating, so it will visit v and print 1 as required.

Suppose that our algorithm prints a 1 when considering u and v. Then v was visited by the BFS in some layer ℓ. Then v must share an edge with some vertex $p_{\ell-1}$ in layer $\ell - 1$, then $p_{\ell-1}$ must share an edge with some vertex $p_{\ell-2}$ in layer $\ell - 2$ and so on, until p_1 is adjacent to some vertex in layer 0. But u is the only vertex in layer 0. Hence $u, p_1, p_2, \ldots, p_{\ell-1}, v$ forms a path from u to v in G.

Hence our modified BFS prints a 1 if and only if there is a path between u and v.

1. c. Prove an upper bound on the complexity of your algorithm

We know from lectures that a BFS takes time $O(m + n)$, our modification of checking for v when we consider a vertex for the first time takes $O(1)$ per vertex for a total of $O(n)$. We must perform the modified BFS for each query and there are q queries. In summary our algorithm takes $O(q(m + n + n)) = O(q(m + n))$.

2. a. Description of how your algorithm works (in plain English)

One Approach

We will use a modified version of Kruskal's algorithm.

Initialize the total weight to zero and X to the empty set then begin by creating a UnionFind data structure F containing each vertex in V as a singleton set. For each edge (u_i, v_i) in A we add w_i to the total weight and check if u and v are in different sets in F; if they are, merge the two sets.

Now sort E by weight from smallest to largest and for each edge (u_i, v_i) in E check if u_i and v_i are in different sets in F. If they are, add w_i to the total weight, add (u_i, v_i) to X and merge the two sets.

Once we have considered each edge in E, terminate and return X and the total weight.

Another Approach

We will perform a reduction, then apply Prim's algorithm.

Initialize W_A to zero. For each edge (u_i, v_i) in A, add w_i to W_A, then set w_i to zero. Now run Prim's algorithm on the modified graph, which will return a set of edges X and a weight W. Return $X \setminus A$ and $W_A + W$.

2. b. Argue why your algorithm is correct

One Approach

Once every edge in A has been added (and included in the total weight) the sets of F are the connected components of the graph (V, A). We now need to show that our second loop connects the connected components of (V, A) in the cheapest way possible.

Consider a new graph G', where the vertices of G' are the connected components of (V, A) and the edges are the edges in E that connect two different connected components. In the case that there is more than one edge between two connected components discard all but the cheapest one. Applying Kruskal's algorithm to this graph will produce its minimum spanning tree - but this is exactly what our algorithm does in the second loop.

Hence by the correctness of Kruskal's algorithm our algorithm connects the connected components of (V, A) in the cheapest way possible, and hence forms the minimum spanning graph of G which includes A.

Another Approach

Let X be the set of edges returned by our algorithm. Observe that since G is a connected graph $(V, X \cup A)$ must also be connected, since Prim's algorithm will always return a tree on a connected graph.

Suppose there was a set Y such that $(V, Y \cup A)$ was connected. Then, by the correctness of Prim's algorithm, the weight of $Y \cup A$ must be greater or equal to the weight of the tree selected by Prim's. Since the edges in A have weight zero, the tree selected by Prim's must have the same total weight as X. But similarly $Y \cup A$ has the same weight as Y. Thus the weight of Y must be greater or equal to the weight of X. Thus X is the minimum weight set such that $(V, X \cup A)$ is connected.

2. c. Prove an upper bound on the time complexity of your algorithm

One Approach

We need to perform n MakeSet operations to initialize, and up to $|A|$ and m calls to Find and Union for our first and second loops respectively.

We know from lectures that using the *union by rank* heuristic we can perform all operations on a UnionFind in (amortized) $O(\log n)$ time. We must also sort E, which we can do in $O(m \log m) = O(m \log n)$ since $O(\log m) = O(\log(n^2)) = O(\log n)$. Hence the total running time is $O(n \log n) + O(|A| \log n) + O(m \log n) + O(m \log n) = O(m \log n)$.

Another Approach

Totaling the weights in A and setting them to zero takes time $|A| = O(m)$. Running Prim's algorithm takes time $O(m \log n)$, from lectures. Computing $X \setminus A$ can be done in $O(m)$ if the sets are sorted, which we can do in $O(m \log m) = O(m \log n)$. Hence the total running time is $O(m) + O(m \log n) + O(m \log n) + O(m) = O(m \log n)$.