Intro to Algorithms HW 6

Greg Stewart

March 23, 2018

Q1.

Given an undirected graph G, describe a linear time algorithm to find the number of distinct shortest paths between two given vertices u and v. Note that two shortest paths are distinct if they have at least one edge taht is different.

This can be achieved by using BFS in addition to two new arrays corresponding to the vertices in G. The first array is one which counts the number of shortest paths up to a vertex x from u; let's call it count[]. The second array is one which tracks the length of the shortest path up to a vertex x from u; let's call it len[]. Starting BFS from u, at each new node visited, the len[x] is just the length from the previous node plus 1, and count[x] is equal to the previous node's count. There are two important cases to consider here if the vertex x is already visited.

First, if the the len[x] value is equal to that of the previous vertex plus 1, then both paths count as shortest distances to this node from u. Thus, we increase the count[x] value by the count of the previous node.

Second, if the len[x] value is greater than that of the previous vertex plus 1, then the previously found way of reaching x is better than the current way, so len[x] is equal to the value in len of the previous vertex plus 1, and count[x] is equal to that of the previous vertex.

There is also a third case when the current path has greater distance than the previous, but the values in the array need not be changed then. Continue with this until BFS is done, and the count[v] value will be the number of distinct shortest paths to v.

Q2.

Given a weighted directed graph with positive weights, give a $O(|V|^3)$ algorithm to find the length of the shortest cycle or report that the graph is acyclic.

To find the shortest cycle from a node v in graph G, one can use a slightly modified version of Dijkstra's algorithm, where the destination node is also the source node. This can be used to find the shortest cycle. Initialize a variable to infinity to track the length of a cycle. Going through all vertices in the graph, use Dijkstra's algorithm in this way to find the length of the shortest cycle, and if it is less than the currently stored value, assign the length to that variable. If there is not a cycle, Dijkstra's algorithm will search all vertices and fail. Thus if the shortest cycle length is unchanged at the end, then the graph is acyclic. This algorithm must search over every node, so the running time boils down to $O(|V|) \cdot O(|V|^2) = O(|V|^3)$.

Q3.

Give a directed weighted graph G, with positive weights on the edges, let us also add positive weights on the nodes. Let $l(x, y)$ denote the weight of an edge (x, y), and let $w(x)$ denote the weight of a vertex x. Define the cost of a path as the sum of the weights of all edges and vertices on that path. Give an efficient algorithm to find all the smallest cost paths (as defined above) from a source vertex to all other vertices. Analyze and report the running time of your algorithm.

For this, we can use a modified version of Dijkstra's algorithm—when adding the length or cost to the path from one vertex to another, also include the weight $w(v)$ of the vertex v in the path. When performing

the search for the optimum tentative path length, include the weight of the node in addition to the weight of the edge. This is $O(|V|^2)$ for any path between two nodes u and v in the worst case, just like the original algorithm. However, this needs to be done between each vertex and every other vertex in the graph.

The algorithm is that described above, but applied from each vertex in the graph to every other vertex in the graph: a nested loop goes over every possible combination, and the modified Dijkstra's algorithm is performed. This means that we have a running time of $O(|V|^2) \cdot O(|V|^2) = O(|V|^4)$.

Q4.

Given a directed graph G with possibly negative edge weights, consider the following algorithm to find the shortest paths form a source vertex to all other vertices. Pick some large constant c, and add it to the weight of each edge, so that there are no negative weights. Now, just run Dijkstra's algorithm to find all the shortest paths. Is this method correct? If yes, reason why; if not, give a counterexample.

This will not work as it does not treat all paths the same way. If the best path contains more edges than another path (but is the best path due to some negative weights), then the best path will have the constant c added to it n times, where n is the number of edges in the path.

For example, say the total cost of the best path is 2, and it contains 4 edges. Then when adjusting the edge weights, it will have total weight $2 + 4c$. Say another path originally has total weight 4, but only one edge. Then adjusted it has weight $4 + c$. Since c is a large constant, $4 + c < 2 + 4c$, so the real best path will no longer be found by Dijkstra's algorithm.