TIN093 Algorithms - Assignment 6

Mirco Ghadri

September 1, 2024

Problem 10

If we perform BFS and color each layer alternating colors, such as red and blue, our graph is bipartite if and only if there are no edges between nodes in the same layer, i.e. nodes that have the same color. If there is an edge between nodes in the same layer, the edge will be part of an odd cycle. This can be proved as follows. Let $e = (u, v)$ be the edge that connects 2 nodes u and v which are part of the same layer i. There is some node z in a layer j, which is a common ancestor to the nodes u and v. How do we know this? Because the node we started BFS from is a common ancestor to all nodes. Furthermore, there can be more common ancestors in larger layer numbers. Select the z in the largest such layer j. Then the path from z to u has $i - j$ edges and the path from z to v has $i - j$ edges. And we also have the path from u to v which consists of the single edge $e = (u, v)$. This gives us a cycle of length $2(j - i) + 1$ edges, which is odd. But we must prove equivalence, i.e assume we have an odd cycle in the graph. We need to prove that one of its edges will be between nodes of the same layer. Lets analyze what happens when we perform BFS on any odd cycle of length/number of edges $n = 2k + 1$. Select any node in this odd cycle to be the root node and perform BFS from it. In the end, no matter which node we select as root node and perform BFS from it, we will have an edge between 2 nodes of the same layer. So indeed all odd cycles in the graph will have an edge that is part of nodes in the same layer after BFS is performed on the graph. And all edges between nodes of the same layer are part of an odd cycle as we proved. We are now ready to formulate our algorithm.

1. Perform BFS on the graph starting from any node s. Store the nodes of all layers in lists for that specific layer such as $L_1, L_2, L_3...L_n$. The node s naturally belongs to the list for layer 0, i.e. L_0.

2. Starting from Layer1, see if there is an edge connecting any pair of nodes in this layer. If not, increase the layer counter by 1 and see if there is an edge connecting any nodes belonging to layer 2.

3. Once you have found an edge $e = (u, v)$ that connects 2 nodes of the same layer i, find the largest common ancestor of the nodes u and v. How? Check the parent of u and check the parent of v using the BFS tree that was generated by the BFS algorithm. This is possible because the BFS algorithm also generates a BFS tree besides the obvious list for the different layers. If they have the same parent, you have found their largest common ancestor. If not, check the grandparent of u and the grandparent of v. If they are the same you have found the solution. If not, repeatedly go 1 step up in the BFS tree until you find the first common ancestor which will be the largest common ancestor. How long does this take? This depends on the depth of the BFS tree which can be at most n so it takes $O(n)$ time.

4. If you find an odd cycle with length 3 you stop immediately since this is the smallest possible size of any odd cycle. Else you find the length of each odd cycle using the method in 3 and store the length in a list together with the 2 nodes in the same layer which form an edge in the odd cycle. This is so that we can reproduce the odd cycle afterwards. After the algorithm has terminated, you find the smallest item in the list, i.e smallest length. You can do this using linear search. The question becomes, how many items can there be? I.e. how many odd cycles? Since we proved that each odd cycle has an edge that is part of 2 nodes in the same layer, we can upper bound the number odd cycles by the number of edges in the graph. Let m denote the number of edges in the graph. So we can say that linear search to find the smallest odd cycle takes $O(m)$ time. Once you found the smallest odd cycle, you look at the 2 nodes of the same layer which are part of the odd cycle and simply reconstruct it.

So what is the total running time of this algorithm? We know that BFS takes $O(m)$ time because it has to discover all the edges of the graph. Finding the largest common ancestor takes $O(n)$ time. Finding the smallest odd cycle takes $O(m)$ time. So we have a total time complexity of:

$$O(m + n) \tag{1}$$

Question to Examiner: Does BFS take O(m+n) or O(m) time? I found that the book mentions O(m+n) but lecture notes mentions O(m) time. Personally I think you create arrays to store all nodes in which should take O(n) time, and exploring all edges should take O(2m) = O(m) time. So BFS should take $O(m + n)$ time? But maybe the lecture notes assumes you only discover the edges and build a BFS tree but not that you create a list of BFS layers and store the nodes in it as the book does? Or maybe it has to do with the fact that the graph is connected? I.e. the number of edges in a connected graph is $m \geq n - 1$ so $O(n) \leq O(m)$?

Problem 11

To understand why this is true we need to look at how BFS and DFS works. A BFS tree will never contain an edge between nodes that are in the same layer because these nodes have already been discovered. So if we have a graph G that is not bipartite/contains odd cycles/has edges between nodes in the same BFS layer, then we know that G will not look

like the BFS tree. However if we have a graph G which has no edges between 2 nodes in the same BFS layer, then G will indeed be identical to the BFS tree. Why? Because there is only 1 path from the source node to every other node.
Now the question becomes, if the BFS tree is equal to the DFS tree of the graph G, then why does it mean that the graph does not contain any edges between 2 nodes of the same BFS layer? To prove this, we note that DFS works by following a path as deep as possible from the source node u until the path can not continue,i.e reaches a node with no more outgoing edges to undiscovered nodes. First lets assume the graph does not contain any edge between 2 nodes of the same BFS layer and show that DFS tree will look like BFS tree. Since there are no edges between nodes of the same layer, there is only 1 path to each node. The DFS tree will show this path, which will also be the same path as the BFS tree. Now assume the graph does contain an edge between 2 nodes of the same BFS layer and show that DFS tree will not look like BFS tree. Indeed the edge that connects these 2 nodes will be a part of the DFS tree. Lets call the edge $e = (u, v)$.