Assignment 6

Erik Norlin, CID: norliner

October 8, 2023

Problem 10

A cycle exists in the graph if there turns out to be an edge between two nodes that are already visited when running the BFS algorithm. In order for this cycle to be odd the edge between two already visited nodes has to be on the same level in the BFS tree (see figure 1 for example).

For this problem a modified version of the BFS algorithm is constructed (see appendix). In this version, for all nodes, the parent node of a node u is saved as a property of u to be able to track back to the first common parent of two connecting nodes forming an odd cycle. The reason for this is because the first common parent of two nodes u and v on the same level i with edge (u, v) in a BFS tree forms the shortest odd cycle including u and v given layer 0 to i.

When running the algorithm, if two nodes are visited, has an edge between them and are on the same level in the BFS tree, then their first common parent node is searched for recursively in the *Find_shortest_cycle_length* function to find the shortest length of the cycle. The initial input in this function are the two connecting nodes u, v that form the odd cycle and k, the number of edges forming it, initially 3 edges which counts for the minimum amount of edges to form the odd cycle. If the parent nodes of u and v are the same node then k is simply returned as it is, namely 3 edges. Otherwise, the function continues to search recursively for the first common parent of those parents. For every recursive call, k increments with 2 edges because there is one edge connecting u to its' parent, and one edge connecting v to its' parent. It is then possible to back trace to find the the specific path the odd cycle takes without increasing the cost of running time.

For the running time of BFS, every node is visited once and every edge is considered twice so the running time is in $O(n + m)$. In addition to this in the modified BFS, in the worst case the number of recursive calls to the first common parent between two connecting nodes forming an odd cycle is $(n - 1)/2$ (see figure 1). For all nodes this results in $O(n(n - 1)/2) \in O(n^2)$, so in total the running time results in $O(n + m + n^2) \in O(m + n^2)$.

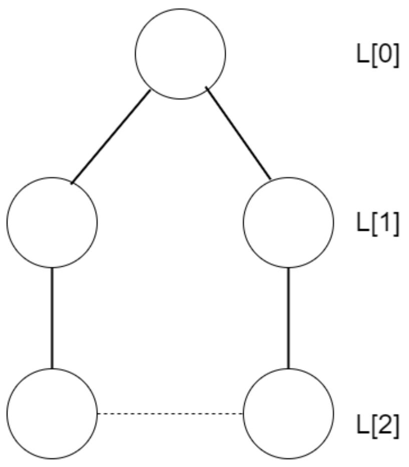

Figure 1: BFS tree of 5 nodes where the dashed line is an edge in the original graph that forms an odd cycle. The graph is constructed as a worst case scenario that maximizes the length of the shortest odd cycle given n nodes. The number of recursive calls to reach the first common parent of the two nodes in L[2] is $(n-1)/2$, and this is used in the time complexity analysis. In this tree, $n = 5$ and the two nodes on layer 2 are connected in the original graph. The number of recursive calls to find the first common parent, i.e., the root node, is (5-1)/2=2.

We prove that the algorithm cannot fail to find some shortest odd cycle by contradiction. Assume we have an arbitrary graph containing odd cycles and that the algorithm fails to find some shortest odd cycle. We also assume that the algorithm finds an odd cycle with a length larger than 3 edges at some point. At an arbitrary layer i further down in the BFS tree after this discovery, a node s is picked from L[i], then all edges to s are considered, and all nodes with edges to s are put in L[i+1]. After this, a node u is picked in L[i+1], then all edges to u are considered, and suppose that u has an edge to another node v in L[i+1], another odd cycle is found. s is the parent node to both u and v so the odd cycle has 3 edges in length, hence the minimum length of found odd cycles is updated to 3. This is some shortest odd cycle in the graph because it is shorter than the previously found odd cycle and also the shortest odd cycle that is possible to construct (*some* shortest because there can be other odd cycles of length 3 in the graph). This contradicts the assumption that the algorithm fails to find some shortest odd cycle, and thus this cannot be the case.

Appendix

```
ModifiedBFS():
    Pick a node s
    Discovered[s] = true
    Discovered[v] = false for all other nodes
    Initialize the BFS tree as an empty set
    Initialize L[0] = {s}
    min_odd_cycle_length = inf
    i = 0

    While L[i] is not empty
        Initialize an empty list L[i+1]
        For each node u in L[i]
            For each edge (u,v) incident to u
                If Discovered[v] is false
                    Discovered[v] = true
                    Add edge (u,v) to the BFS tree
                    Add v to L[i+1]
                Else if Discovered[v] is true and v belongs to L[i]
                    d = Find_shortest_cycle_length(u, v, 3)
                    If d < min_odd_cycle_length
                        min_odd_cycle_length = d
        i += 1
    return min_odd_cycle_length

Find_shortest_cycle_length(u, v, k):
    If parent node of u and parent node of v are not the same
        k = Find_shortest_cycle_length(parent of u, parent of v, k+2)
    Return k
```