Notes 9-27

Cameron Chalk

Note: Please see Graph Algorithms slides from canvas.

1 Depth-first search/breadth-first search (DFS/BFS)

We want to do the following: we want to return the set of nodes reachable from a starting node u. These are useful modules to use to solve other problems. These searches apply to directed and undirected graphs. We will focus on undirected graphs, but everything discussed is easily generalized to directed graphs.

(See DFS slides). We will analyze the complexity of DFS. To analyze the complexity of graph algorithms, we consider the input size in two parts: the number of vertices n and the number of edges m. Some algorithms will be better/worse depending on the ratio of n to m. For DFS, to keep track of explored vertices, we will create an array *explored* which has $explored[v] = 1$ if v has been explored and $explored[v] = 0$ otherwise.

- initializing *explored* array takes $\Theta(n)$ time
- each edge is examined (to determine if the second node has been explored) at most twice for $\Theta(m)$ examinations; $\Theta(1)$ to find next edge in adjacency list[1]

Thus the total runtime is $\Theta(n + m)$. To see that DFS gives the set of nodes which are reachable from u, consider the DFS tree. Each edge in the DFS tree is an edge in G. However, there can be edges in G which are not in the DFS tree.

Claim 1. *If edge (x, y) is in G but not in the DFS tree, then one of x or y is an ancestor (parent or parent-of-parent, etc.) of the other in the DFS tree.*

Proof. Omitted. Try to prove it for practice! □

Next we'll see breadth-first search. (See BFS slides). It is a little more complicated in pseudocode, but it is easier to understand if we consider the BFS tree generated by the algorithm. If we consider the layers of our BFS tree, what we want is that in layer layer i we have all nodes that aren't in prior layers

[1] In an adjacency matrix, this would be $\Theta(n)$, as we may have to traverse the whole row of u to see if any edges are attached to u.

that are adjacent to some node in layer $i-1$. The runtime of the algorithm is the same as DFS via the same analysis.

BFS is useful for finding the *shortest path distances* from u. Let the shortest path distance from u to v, denoted $d(u,v)$, be the minimum # of edges of any path from u to v (if no path exists, we say $d(u,v)=\infty$).

Claim 2. *If we compute BFS from u, if v is in layer i, then the shortest path distance from u to v is i.*

Proof. Suppose v is in layer i. Then the path from u to v in the BFS tree is of length i, therefore $d(u,v) \leq i$. We now prove $d(u,v) \geq i$. Towards contradiction, assume $d(u,v) < i$. Then there must be an edge in G that skips at least one layer; i.e., an edge (w,z) with w in layer i and z in layer j with $i < j-1$. However, note that adjacent nodes in G are either in the same layer or in adjacent layers. Thus, there are no edges in G which skip more than one layer, contradicting $d(u,v) < i$. □