CS 180 Homework 3

Prithvi Kannan

405110096

1. Exercise 10 Page 110

We perform a breadth-first search from vertex v to get the distance of each node to . If the path has layer numbers that increase by one for each node along the path, then is the shortest (aka most direct) path from to .

Now we apply induction to compute the number of shortest paths. For nodes in the first layer, there exists only one path, so if is part of . Consider a node in layer . The shortest path from is composed of a path from to a node in and a path of length 1 from . Therefore, is the sum of all nodes in layer with an edge to .

The BFS from vertex v takes time . Then we compute count for each of the nodes, which will take at most the degree of the node. Using a degree-centric approach, the total sum of degrees in the graph is , so the runtime of the algorithm is .

2. Exercise 6 on page 108

We will prove that must be a tree by contradiction. Assume G is not a tree, meaning there is an arbitrary edge e, from to for , that is not contained in the BFS/DFS tree . Assume that is ’s parent in the tree .

In the BFS tree, since and ’s distance from the root u can differ by at most one. However, in the DFS tree, if is ’s parent, then the distance must be one more than . Therefore, the all edges in will exist in both the BFS and DFS tree, so by contradiction, must be a tree.

3. Exercise 7 on page 108

Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least n/2, then G is connected.

We will prove this is true by indirect proof. Assume there exists arbitrary node and in for which there exists no path, meaning is not connected.

As required, the degree of and is at least . Since we assume that and are not connected, then all the nodes connected to a cannot also be connected to b (if there was a node x that was connected

to and , then there would be a path and the graph would be connected). Therefore

we have nodes in the component containing node and nodes in the component containing node , since and have degree of at least . The comes from the fact that we must also add node or into the count. This would mean we have at least nodes in the graph, but since there are only n nodes in the graph, this is a contradiction.

For this reason, must be connected if each vertex has degree of at least .

4. Exercise 3 on page 189

The goal is to minimize the number of trucks, . A valid solution assigns boxes to trucks such that no truck is carrying more that weight and the order of the boxes is preserved.

The greedy algorithm is as follows:

Assume we have a queue of packages

While truck t's available capacity is less than W

If possible

Add the next wi package from queue

Else

Send truck t off

Increment n

We now show by induction that the greedy algorithm will always be ahead of any other algorithm, defined as having boxes in trucks while another algorithm may have in the same trucks for . For the base case, , both greedy and non-greedy algorithms will fit as many boxes as possible into one truck. Now we assume greedy is ahead for , meaning greedy has fit boxes, and prove that greedy is still ahead for . In truck , the greedy algorithm will pack and the non-greedy algorithm will pack . Since , the greedy solution may still have space for more, making it the optimal solution.

5. Exercise 6 on page 191

Calculate each contestants combined bike and run time

Sort in decreasing order and send out contestants

Consider the optimal solution and assume our algorithm produces a different result. The optimal solution will have candidate and such that the order is and . If we switch and b, the will finish the race earlier than they used to. Candidate will now get out when used to get out, but since ’s combined bike and run time is less than ’s, will finish before ’s old finish time. Performing these swaps, we will only lower the overall race time, and eventually end up at the solution no faster than that produced by the algorithm, meaning the algorithm gives the optimal solution.

This algorithm runs in to calculate the combined bike and run time, and then to sort in decreasing order. This gives an overall runtime of

6. (a) Can you design an algorithm that finds the longest path in a directed graph (DG)? (you can use an edge at most once)? If yes, describe the algorithm and analyze its time complexity.

Keep track of the longest path so far

For each node n in G

Run directed BFS and find the longest path

Compare longest path with longest so far

This algorithm runs , since the inner BFS loop takes steps to run, and it is run on each node in the graph. This approach is brute force, and considers every possible path in the graph.

(b) Can you design an algorithm that finds the longest path in a directed acyclic graph (DAG)? (you can use an edge at most once)? If yes, describe the algorithm and analyze its time complexity.

Run a topological sort algorithm on the graph

Keep track of the longest distances to each vertex

Looping through each vertex v in topological order

Loop through all vertices u connected to v

If the distance of v is less than dist u+1

u+1 is new dist of v

The topological sort takes time complexity The algorithm will go through each vertex in the topological sort and examine all adjacent vertices. The total number of adjacent vertices ina graph is , so this analysis runs in