Derek Johnson, dej3tc
12/4/2019

Complexity Analysis

Prelab: In order to implement the topsort method I created two classes, a Vertex class and a Graph class. The Vertex class will store the class name, the indegree, and a vector containing Vertex pointers for all of the classes this is a prerequisite for. The Graph class will contain a list of all of the Vertex's that are currently added. Adding an edge(two nodes with a direction) to a Graph is done using the insertEdge method. The Graph class is also capable of performing topsort and outputting the classes in the new sorted order.

insertEdge: First check if either of the vertices in the edge are already in the Graph's Vertex list. Next, update the first Vertex's adjacency list by pushing the second Vertex to the back. The big Theta Run time for this will be Teta(n) because you potentially have to check every Vertex in the Graph.

topSort: Push all of the Vertices in the Graph with an indegree of 0 onto a queue. Continue to pop Vertices from the queue, print the class name, and push linked Vertices with an indegree of 1 onto the queue. Do this until the queue is empty. This will print out every linked node in the Graph. It will need to visit each edge once and each Vertex once so the Big Theta run time is Theta(n).

Because a Graph that can be topologically sorted can not contain loops, we know that if there are n Vertices there can be at most (n-1) edges. Additionally, the graph will store pointers to all of the nodes. The space complexity of the pre-lab therefore is Theta(n)

InLab: In order to solve the traveling salesman problem, a computeDistance method is made to find the distance between two cities. From there, the starting point is set and every other node runs through all possible combinations to find the minimum possible distance. This brute force method takes Theta(n!) time. The space complexity for the brute force algorithm is Theta(1) because no new data will need to be generated except for the temporary values. The space complexity for storing the distance grid will be Theta(n^2).

Acceleration Techniques

Held-Karp Algorithm: This optimization of the traveling salesman problem(TSP) relies on the fact that "Every subpath of a path of minimum distance is itself of minimum distance". This means that instead of using a top-down approach such as in the brute force application, we use a bottom-up approach in which all intermediate info is found only once. For a set S we will build a distance matrix dm that is of dimension nxn. First we will pick an arbitrary first node. We will then recursively build sets starting from each other node and traveling to that first node. This will be repeated but each time we scale up the size of the set we are making by one. In building the next layer of sets we will use the solutions to all of the smaller sets we have already computed. For each set we will keep track of the cost and the previous node. This allows us to save time

because all smaller problems have already been solved. The drawback is that we have to store a lot more information. The time complexity is O(2^n*n^2). I am not sure if I am correct in assuming that the processes take similar time, but if so, this method should speed up implementation by about 2 times.

Branch and Bound: In this algorithm, the candidate solutions are thought of as a rooted tree. The starting point node is considered the root. Each branch of the tree represents a possible point to travel to next. The branches alone would amount to brute-force enumeration but in this algorithm, a bound method checks if a candidate solution can be a possible solution. If it can not it is pruned from the tree. The method uses an approximate tsp solution from another method to get the upper time bound. This bound can be updated if we find a better solution while recursing through the function. The lower bound is calculated by adding the two lowest cost edges for each node in the set and dividing this by two. These upper and lower bounds remove many possibilities from the tree and help us same time and space. The worst case is Theta(n^2) because we may never be able to prune a node. The increase in speed would depend on how many nodes can be pruned but it should speed it up by a factor of 3.

Nearest Neighbor Algorithm: This is an Approximation algorithm for the TSP. This algorithm will start at a random city and visit the nearest city until all have been visited. This will yield a quick route but it is not necessarily the shortest one because it is not calculating multiple paths, just the distance between each head node and all the unvisited nodes. The steps for the algorithm as listed by Wikipedia are

1) Initialize all vertices as unvisited
2) Select an arbitrary vertex, set as the head and mark as visited
3) Find the shortest edge connecting the current vertex and an unvisited vertex.
4) That unvisited vertex is the new head and is considered visited
5) Repeat steps 3 and 4 until all vertices are visited.

The worst case performance for this is O(n^2) because every node will have to search at most all remaining nodes minus one. The speed up time for this algorithms is not even worth talking about because it varies so much based on the number of nodes you are searching.