Team JPEG

Scrum Master: Jesus Sanchez

Product Owner: Sabrina Dang

Members: Frances Ohide, Jason Zhang, Lee Evans

Data Structures Used

The data structures used for this project include

* Vectors
* Unordered map
* Graph

Big-O Analysis

* **Notes:**
* Overall Big-O highlighted in yellow.
* Big-O of functions commonly used in algorithms:
* findVertex() = O(n)
* distanceBetween() = O(n)
* edgesDiscovered() = O(n)
* distanceFromStart() = O(n)
* smallestEdge() = O(n^2)
* Shortest Paths (Dijkstra’s Algorithm)
* The shortest paths algorithm utilizes Dijkstra’s method to return the shortest path to every vertex from an indicated vertex. If the graph object has previously called any of the traversal functions, the vertices and discovery edges must be marked as unvisited and unused which calls a reset function that runs in O(n2) time. After the starting vertex has been added to T, a while loop is used that runs until all vertices have been marked as visited, adding another O(n) to the runtime of this algorithm. Inside the while loop, the function findClosest() is called for to find the closest vertex from the vertices stored in T. This function runs in O(n^3) time since it contains a while loop that runs for the size of T to find the shortest distance from the start from all of the unvisited edges in T. Inside this while loop, the functions edgesDiscovered(), distanceFromStart(), and distanceBetween() are all called twice and all run in O(n) time adding O(6n) to the runtime of findClosest(). Furthermore, the function smallestEdge() that is also called in the loop twice runs in O(n2) time which leads to an overall runtime of O(6n) + O(2n2). So, because this is encompassed within the while loop, findClosest() runs in O(6n2) + O(2n3) or O(n3). Now, coming back to the shortest paths algorithm, because the findClosest() function is called n times by the while loop, the runtime of this while loop is O(n4). Finally, taking all of this into consideration, the overall runtime of the shortest paths algorithm is findClosest() O(n4)+ O(n2) + O(n) or O(n4).
* Minimum Spanning Tree (Prim-Jarnik’s Algorithm)
* The minimum spanning tree algorithm uses Prim-Jarnik’s Algorithm to find the smallest mileage between all of the vertices in the graph. This algorithm also begins by unmarking all of the vertices and edges in the graph in case another traversal has previously been used which gives this algorithm a O(n2) runtime from the start. Then, the findVertex() function is called to find the graph index of the vertex adding O(n) time to the algorithm’s total Big-O. Once the current vertex is marked as visited, a recursive call is performed on the MST algorithm function by sending the next closest vertex to all of the vertices that have been visited as a parameter. Finding the next closest vertex invokes the smallestEdgeMST() function which runs in O(n3) time. Thus far, the runtime of the MST algorithm is O(n3) + O(n2) + O(n), but since this does not account for the recursive call that occurs “n” times, another “n” is multiplied to each O(n). Ultimately, the Big-O of the MST algorithm is O(n4)+ O(n3) + O(n2) or O(n4).
* Shortest Trip Algorithm (In Selected Order)
* The shortest trip algorithm that recursively visits the stadiums in the selected order runs in O(n5) time. This is because this function calls the shortest paths algorithm to find the shortest path between stadium A and stadium B until all the stadiums have been visited, in the order they were selected. Because a total of “n” stadiums will be visited, the function will perform “n” recursive calls which increases the runtime of the shortest paths algorithm from O(n4) to O(n5).
* Depth-First Search Algorithm
* As every other graph traversal algorithm, all the vertices and edges of the graph are marked as unvisited/unused adding the initial O(n2) to the overall runtime of the algorithm. Once findVertex() returns the graph index of the starting vertex and that has been added to the BFS vector that keeps track of the order that the stadiums were visited in, the smallestEdgeDFS() function is called to find the closest unvisited vertex to the current vertex. This function runs in O(n3) in the worst case because a recursive call is performed with this function to backtrack to the previous vertex in the BFS vector if all the vertices adjacent to the current vertex have already been visited, else finding the closest vertex when there is at least one unvisited adjacent vertex takes O(n2) time. After this, the next closest vertex is found, and the algorithm performs a recursive call using this vertex. Because of this, the O(n3) + O(n2) + O(n) becomes O(n4)+ O(n3) + O(n2), so the overall runtime of this algorithm is O(n4).
* Breadth-First Search Algorithm
* The BFS algorithm begins with the initial O(n2) runtime that clears the visited and marked status of all nodes in the graph. Then, the graph index of the vertex is found and that stadium is added to the BFS vector that keeps track of the order that the stadiums were visited in. After this, the BFSRecur() function that performs the search for the rest of the vertices in the graph is called in the return statement. The BFSRecur() function runs in O(n4) time when the recursive call it performs is considered. This is because of the three nested for loops that check the adjacent vertices of the vertices in the current level to find the closest unvisited vertices to each vertex in the current level. The outermost for loop iterates through the vertices in the current level containing “n” vertices. The next nested for loop iterates through the current vertex’s adjacent edges to find the closest unvisited edge. Finally, the inner most for loop inserts the next closest unvisited vertex of the nth vertex in the current level to the currLevel vector to keep track of the vertices that will be added to the newLevel vector that will be passed into the BFSRecur() function in its next recursive call. This process that begins in the second for loop is repeated until all the current vertex’s unvisited vertices have been added to the currLevel vector in order from closest to furthest. This O(n3) runtime combines with two additional for loops outside the main loop’s body to produce a O(n3) + O(2n) runtime. Now, considering the recursive nature of this function, the runtime becomes O(n4) because the function recurs “n” times for “n” levels. In summation, the BFS algorithm runs in O(n4) + O(n2) + O(n) time or O(n4) time because of the recursive function called in the return statement, the function that resets the graph, and findVertex().