Project 2: TSP – Search with BFS and DFS

Hunter Ellenberger

CSE-545-02-4248

Speed School of Engineering

University of Louisville, USA

[hlelle01@louisville.edu](mailto:hlelle01@louisville.edu)

**1.**

To start I transferred the functions I would be copying over from the previous project into a new file called “tspUtil.py.” Going forward this file will be used to carry over functions from former projects to latter projects, three functions were transferred into this file in this iteration of the project: “get\_coordinates”, “calc\_distance”, “permutations\_compute”. “get\_coordinates” is the function used to read in the given tsp file and convert those coordinates into a dictionary where the key is the node, and the value is a list containing both the X and the Y coordinate. “calc\_distance” is simply the function used to compute the distance between two nodes using the distance formula. “permutations\_compute” uses the “calc\_distance” function to compute the total distance traveled in every possible path permutation.

Once importing these functions, I then started looking into the makeup of the BFS algorithm. Seeing as I have not taken a class it is covered, I relied on YouTube to inform me what it is, and how it works. Once I learned the algorithm, I then started to implement it, but soon hit a roadblock. I knew how to implement code which would track the first path visited, but after turning the problem over in my head, I could not figure out a way to implement a version which would track all the paths visited. This was a huge roadblock for me and no matter how much time I spent on it I could not figure it out, which was very discouraging for me hence the late submission. This is where I resorted to using the artificial intelligence to rule them all, ChatGPT. A problem I have been turning over for days was solved by ChatGPT in seconds. I then asked ChatGPT to convert my now modified BFS implementation into a DFS one. It once again did it in seconds, with accurate documentation I might add. I have not had to resort to using this tool to help with schoolwork before, but now that I have, I now have a deeper understanding of how powerful this technology can be.

Once implementing both the functions that handle the BFS and DFS search I then simply had to add the time library so that I may record the time it takes for my program to execute with the respective functions.

**2.**

For this project both the Breadth-First Search and the Depth-First Search algorithms are used. They are similar in function as they both are complete search algorithms in the context of a finite graph, but their approaches are different. The Breadth-First Search algorithm searches layer-by-layer, touching the children of every parent node on one layer before moving on to the next layer. This starts at the root node and moves down until the algorithm reaches the “rightmost” leaflet. In our implementation within this project the algorithm goes until it reaches node eleven.

However, Depth-First Search is different. It instead touches the “leftmost” child of every node until it hits a leaflet, once doing so it will then move onto the node “right” of the previous leftmost child (given that there is one to visit) and repeat the process until it has touched every node. In our implementation it will still stop once it reaches node eleven.

**3.**

Each algorithm, seeing as their complete in this implementation, found the optimal path from node one to eleven. However, the BFS algorithm found this path in the shortest theoretical time as it only had to search through eleven paths to find the optimal route. Surprisingly, even though the DFS algorithm had to search through twenty-six paths to find the optimal one, it computationally took less time. The BFS algorithm clocked in at a total time of 10.364ms wheras the DFS algorithm clocked in a time of 8.845ms. This discrepancy could probably be averted by converting both into informed algorithms where they know the distance of the most optimal route and stop whenever they reach it.

In total, there were forty possible paths to be taken, and the optimal route resulted in a distance of 111.272. This optimal route was found by visiting nodes {1, 3, 5, 7, 9, 11} in that order.

**4.**

I find it interesting in this case how no two paths achieved the same distance, whereas in the previous project there were many separate routes to achieve the same distance. I assume this has to be due to the graph lacking completness unlike the previous project. Not only does this result in fewer paths, but it also results in a lot less computational time. In the previous project a graph containing eleven nodes was also used, however, instead of taking just a handful of milliseconds it took five minutes to compute.

I also find it interesting that the BST algorithm managed to be faster in the number of transitions it took to reach the target but was slower computationally compared to the DFS algorithm. This may be caused by the fact that the algorithm must keep appending to a different list each time it reaches a child node whereas the DFS algorithm just adds to the same list, but that is just my best guess at what is happening.

**5.**

I used multiple resources while completing this project and all links can be found below:

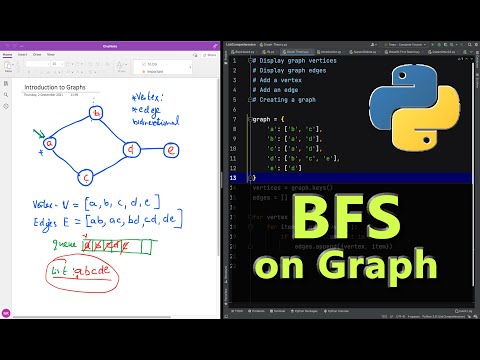
<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

<https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/?ref=header_outind>

<https://www.geeksforgeeks.org/depth-first-search-or-dfs-on-directed-graph/>

<https://chatgpt.com/>

<https://www.youtube.com/watch?v=CnECo0rUbzo>

[](https://www.youtube.com/watch?v=CnECo0rUbzo)