Module 6: Portfolio Milestone

Brady Chin

Colorado State University Global

CSC506-1: Design and Analysis of Algorithms

Dr. Dong Nguyen

December 22nd, 2024

Portfolio Milestone

This milestone involved conducting an empirical analysis of the Dijkstra and Bellman-Ford algorithms to evaluate their time and space complexities across varying input sizes. Additionally, their performances were compared, highlighting key strengths and weaknesses.

Dijkstra's Time Complexity

Given the original code, the time complexity of Dijkstra's algorithm is **$O(V^2 + E)$** where **V** is the number of vertices (nodes) and **E** is the number of Edges. The following is a breakdown of the time complexity:

1. The initialization of the dictionaries will take **O(V).**
2. The `while` loop runs over the unvisited nodes a single time: **O(V).**
3. To find the minimum-distance node over all the neighbors of the current node, we will have to iterate over all nodes again. This will be **O(V)** for **V** iterations and is therefore **$O(V^2)$**.
4. The inner `for` loop iterates over all neighbors of the current node to find edges. This will be done for all nodes in the weighted graph and will result in **O(E)**.
5. At the end of the while loop, the nodes will have to be removed from the `unvisited` list. This will be a time complexity of **$O(V^2)$**.

Dijkstra's Space Complexity

Similar to the time complexity, the space complexity of this algorithm is **$O(V^2 + E)$**. To calculate the space complexity, there are four elements that are considered:

1. The input graph: **O(V +E).**
2. The distances dictionary: **O(V).**
3. The paths dictionary: **$O(V^2)$.**
4. The unvisited list: **O(V).**

Based on this, the paths dictionary is the primary contributor to the space complexity. In the worst case scenario, every path can store all the nodes in the graph.

Bellman-Ford's Time Complexity

The Bellman-Ford algorithm has a time complexity of **O(V * E)** where **V** is the number of vertices (nodes) and **E** is the number of edges. Here is a breakdown of the time complexity:

1. The initialization of the dictionaries will take **O(V).**
2. The first `for` loop will be executed V - 1 times. This is the relaxation step. An inner `for` loop will iterate over all nodes **O(V)** and all edges **O(E)**. Therefore, the time complexity will be **O((V - 1) * E)** which results in **O(V * E)**.
3. The negative cycle check will iterate over all edges for **O(E)**.

Bellman-Ford Space Complexity

The space complexity of the Bellman-Ford algorithm will be **$O(V^2)$**. Like the Dijkstra algorithm, this is largely due to the paths dictionary.

1. The input graph: **O(V +E).**
2. The distances dictionary: **O(V).**
3. The paths dictionary: **$O(V^2)$.**

Strengths and Weaknesses

Both algorithms have their strengths and weaknesses and are more suitable for certain scenarios.

The Dijkstra algorithm is more efficient than the Bellman-Ford algorithm in cases that does not use negative weights. Also, by using a min-heap priority queue (discussed at a later section) to improve the time complexity, it is more suitable for large graphs. A weakness of this algorithm is that it is not as flexible as it cannot handle negative weights.

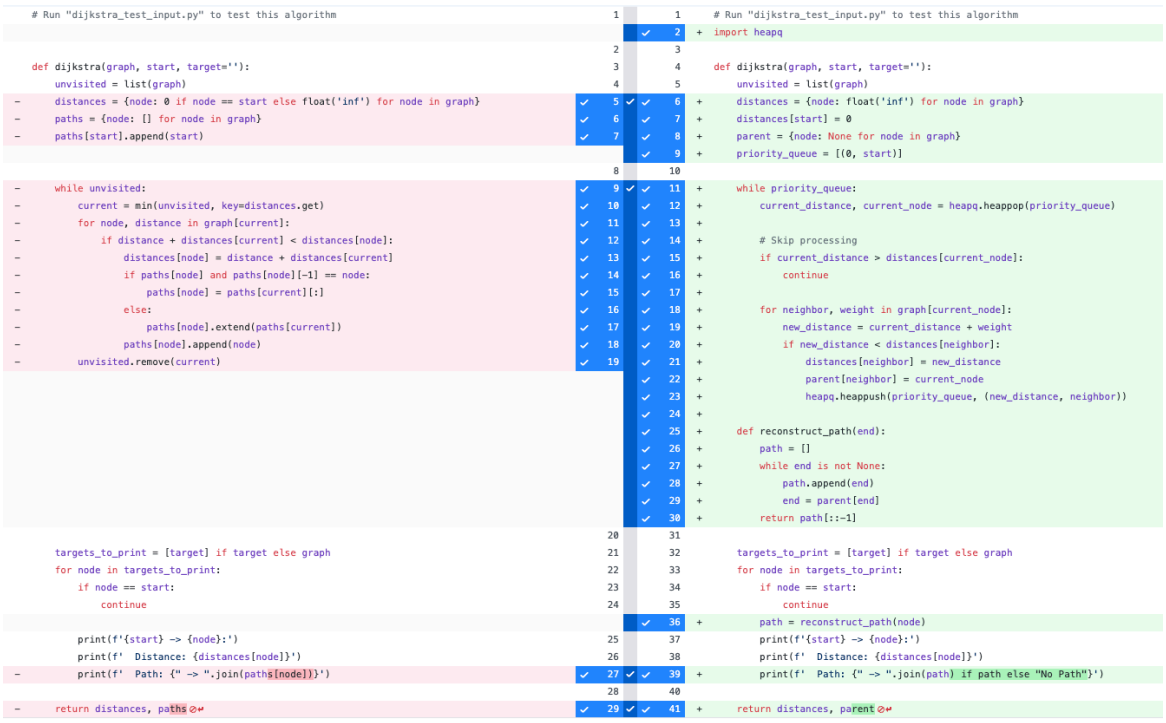
The Bellman-Ford algorithm is helpful in cases that include negative weights. Furthermore, it can also detect negative weight cycles and will end the program if it is detected. Compared to Dijkstra's algorithm using min-heap, it is more inefficient which makes it slower for larger graphs. It also performs unnecessary edge relaxations even if the shortest path is already found.

Optimizations

There were many changes made to the Dijkstra algorithm. Now, uses a heap based priority queue to select the nodes rather than the original linear search method (Pizzo, B., 2024), (geeksforgeeks.org. 2024, February 9). The path construction also is created after the algorithm completes rather than being created dynamically.

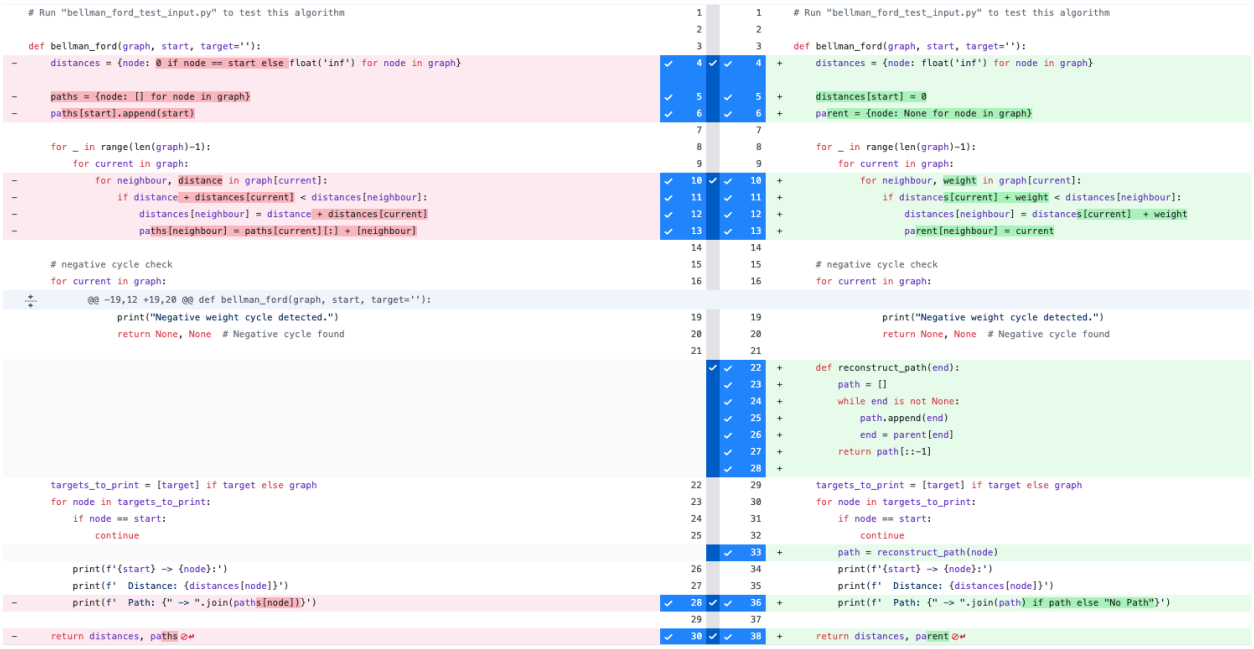
With the use of the min-heap technique, the time complexity is now **O((V + E) log V)**, making it more efficient. In addition, the space complexity is now **O(V)** due to the parent dictionary. These changes can be seen in Figure 1.

Figure 1: Changes made to the Dijkstra algorithm.



Similar changes were made to the Bellman-Ford algorithm. Just like the Dijkstra algorithm the use of a parent for each node is used rather than storing the entire path. This reduces the space complexity to **O(V)** (geeksforgeeks.org. 2024, February 9). In addition, the same path reconstruction technique was used that creates the path after the algorithm completes. The changes are shown in Figure 2.

Figure 2: Change made to the Bellman-Ford algorithm.



Conclusion

The Dijkstra algorithm was optimized by incorporating a heap data structure and with the use of a “parent” to improve its time complexity from **O(V² + E)** to **O((V + E) log V)** and its space complexity from **O(V² + E)** to **O(V)**. The Bellman-Ford algorithm also used a “parent” to reduce it’s space complexity from **O(V²)** to **O(V)**.

References

geeksforgeeks.org (2024, February 9) *Time and Space Complexity of Bellman-Ford's Algorithm.* https://www.geeksforgeeks.org/time-and-space-complexity-of-bellman-ford-algorithm/

geeksforgeeks.org (2024, February 9) *Time and Space Complexity of Dijkstra's Algorithm.* https://www.geeksforgeeks.org/time-and-space-complexity-of-dijkstras-algorithm/

Pizzo, B., (2024) *CSC506: Design and Analysis of Algorithms. 5.11 Heaps.* zyBooks. https://learn.zybooks.com/zybook/CSC506-1_5/chapter/5/section/11