Written Exercise for HW8 Shortest Path

Name: Rong Huang
Date:04/15/2024

1. What is the big-Oh space complexity of Dijkstra's? Justify your answer.

The space complexity of Dijkstra's algorithm is O(V + E). Here's why:

- Vertices and Edges: The algorithm needs to store information about all vertices and edges. Typically, this is done using an adjacency list, which collectively takes up O(V + E) space, where V is the number of vertices and E is the number of edges.
- Additional Arrays: Dijkstra's algorithm also uses a few additional data structures:
  A priority queue to keep track of the next vertex to process. This can contain all vertices in the worst case, taking O(V) space.
  A distance array to store the shortest paths from the source to each vertex, which takes O(V) space.
  A predecessor array to reconstruct paths, also taking O(V) space.

Adding these together, you're primarily looking at O(V + E) space due to the adjacency list and the arrays.

2. What is the big-Oh time complexity for Dijkstra's? Justify your answer.

The time complexity of Dijkstra's algorithm primarily depends on the data structures used for the priority queue.

1) Using a Binary Heap:

- The operations of inserting into the heap and extracting the minimum element from the heap both occur in O(log V) time, where V is the number of vertices in the graph.
- Each vertex is extracted from the heap once, and each of its outgoing edges is relaxed once. Since relaxing an edge and updating the heap both occur in O(log V) time, the total time complexity is O((V + E) log V), where E is the number of edges.

2) Using a Fibonacci Heap:

- The Fibonacci heap improves efficiency, especially for the decrease-key operation which is critical during edge relaxation. Here, the extract-min operation takes O(log V) time, but the decrease-key operation is amortized to O(1).
- This results in a reduced overall time complexity of O(V log V + E), making it particularly suitable for dense graphs where E is much larger than V.

In summary, while the standard implementation using a binary heap has a time complexity of O((V + E) log V), using a more advanced Fibonacci heap can reduce this to O(V log V + E) due to more efficient handling of the decrease-key operations during edge relaxations.

3. Does your implementation operate correctly? How do you know?

To determine if my implementation of Dijkstra's algorithm operates correctly, I have conducted several steps for validation:

- Manual Verification: I manually calculated the shortest paths in the given graph from the start city to the destination and compared these results with those produced by my implementation.
- Consistency Checks: I ran my algorithm on a variety of test cases, including the provided graph, to ensure that the results are consistent and accurate across different scenarios.
- Comparative Analysis: I compared the output of my algorithm with the results from established and tested implementations, ensuring that they match.
- Debugging: During development, I used extensive debugging to trace each step of my algorithm, verifying that the internal logic and state changes are correct.

These steps have given me confidence that my implementation is working correctly. If the shortest path and total distance computed by my algorithm match my manual calculations and those of reliable sources, it is a indicator that the algorithm is functioning as expected.