Asymptotic Analysis for Method findPath() for Both BFS and Dijkstra's Algorithms

Batuhan BASAK

June 2023

1 CSE222BFS findMethod

Algorithm puts identified vertices but not visited vertices to *theQueue* to get the next identified item to examine. The visited vertices are stored in the set *visited*. The parents' of vertices are stored in map *parentMap*. Adding and removing vertices from *theQueue* takes $\theta(1)$. Checking whether *currentVertex* is *end* is $\theta(1)$. Adding a vertex to the set *visited* is $\theta(1)$. Since *buildPath* method traverse each item in *parentMap* from *end* vertex to *start* vertex, traverse only the edges.
Assume the number of edges from *start* to *end* is $|E|$. Then the time complexity of method *buildPath* is $O(E)$.
The outer *while* loop runs until there is no item left in *theQueue*, which is filled, each iteration of inner loop, with immediate neighbor vertices of *currentVertex*. Since the visited items cannot be added to *theQueue* anymore the total number of items in *theQueue* can be at most $|V|$, the number of vertices. Therefore outer loop's time complexity is $O(V)$.

$$T(n) = O(|V| + |E|).$$

2 CSE222Dijkstra findMethod

Initializing the *distanceMap* and *predecessorMap* with initial values takes $O(|V|)$ time, where $|V|$ is the number of vertices in the graph. Adding the start vertex to the priority queue takes $O(log|V|)$ time.
The while loop continues until the priority queue is empty, and in each iteration:

- Removing the vertex with the minimum distance from the priority queue using *poll()* takes $O(log|V|)$ time.

- Adding the current vertex to the *visited* set takes constant time, $O(1)$.

- Checking if the *current* vertex is the *end* vertex (*current.equals(graph.getEnd())*) takes constant time, $O(1)$.

- Obtaining the edges of the *current* vertex using *graph.edgeIterator(current)* may take up to $O(|E|)$ time, where $|E|$ is the number of edges in the graph. This depends on the implementation of the *edgeIterator()* method.

- For each edge, the code checks if the *neighbor* has been visited *(!visited.contains(neighbor))* which takes constant time, $O(1)$.

- Computing the new distance and updating the *distanceMap* and *predecessorMap* takes $O(log|V|)$ time for removing and adding the neighbor to the priority queue.

Building the shortest path from the *predecessorMap* takes $O(|V|)$ time, where $|V|$ is the number of vertices in the graph. This involves traversing the predecessor map and adding vertices to the *shortestPath* list.
Therefore, the overall time complexity of this implementation of Dijkstra's algorithm in the *findPath()* method is $O((|V|+|E|)log|V|)$, where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. The logarithmic factor arises from the operations of adding and removing vertices from the priority queue, which takes $O(log|V|)$ time, and the edge traversal, which takes up to $O(|E|)$ time.

$$T(n) = O((|V| + |E|)log|V|)$$