

1

In homework9, the time complexity of Dijkstra's Algorithm is  $O(V^2)$ .  $V$  is the number of vertices and  $E$  is the number of edges. It takes  $O(V)$  time to enqueue all vertices into the queue. And the outer while loop would iterate  $O(V)$  times since all vertices would be removed from the queue. As the queue uses linear search to get the vertex with minimum cost, it takes  $O(V)$  time to dequeue. With adjacency list, iterating all vertices' neighbors is  $O(E)$ . Updating the cost of vertices is  $O(1)$ . Therefore, the complexity is  $O(V^2)$ .

The time complexity of Dijkstra's Algorithm with priority queue is  $O(E + V \log(V))$ . It takes  $O(V)$  time to enqueue all vertices into the priority queue. And the outer while loop would iterate  $O(V)$  times since all vertices would be removed from the queue. With priority queue, it takes  $O(\log(V))$  to dequeue the vertex with the lowest cost. With adjacency list, iterating all vertices' neighbors is  $O(E)$ . By using Fibonacci heap priority queue, updating the cost of vertices in the queue is  $O(1)$ . Thus, the time complexity is  $O(V + V \log(V) + E \cdot 1) = O(E + V \log(V))$ .

2

The space complexity of Dijkstra's algorithm is  $O(V+E)$ .  $V$  denotes the number of vertices in and  $E$  denotes the number of edges. It needs  $O(V)$  memory to store all vertices and  $O(E)$  space to store all edges. Therefore, the space complexity is  $O(V+E)$ .

3

Assume  $G$  is a graph and the source is  $s$ . Assume  $S$  is the set of nodes that Dijkstra's algorithm creates. We want to show each node  $v$  added to the  $S$  has the shortest path from  $s$  to  $v$ .

Base case: The algorithm starts by adding  $s$  to  $S$ . The cost of  $s$  to  $s$  itself is 0, so this is the shortest path.

Inductive Hypothesis: Assume that if the number of nodes in  $S$  is  $k > 1$ , then each node  $v$  added to the  $S$  has the shortest path from  $s$  to  $v$ .

Nodes  $u_1, \dots, u_k$  are the nodes in the set of  $S$ . They store the shortest paths  $d[u]$  from  $s$  to  $u$ . After each iteration it finds the node  $v$  that is not in  $S$  with the shortest path  $d[v] = d[u] + \text{length}(u,v)$  in all unvisited nodes.

Suppose there is another optimal path from  $s$  to  $v$ , the node before  $v$  is  $w$ , which is not in  $S$ .  $d[w]$  is the shortest path from  $s$  to  $w$ .  $d[v] = d[w] + \text{length}(w,v)$ . It is obvious that the  $d[w] > d[u]$  because in each iteration it chooses the one with shortest path. If  $d[w] < d[u]$ ,  $w$  would be added to  $S$  before  $u$ . Therefore,  $d[v] = d[u] + \text{length}(u,v)$  is the shortest path, and in next iteration this remains true. When all nodes added to  $S$ , any node  $u$  has the shortest path  $d[u]$  from  $s$  to  $u$ .