Pistes de départ :

1) l'algorithme de Bellman-Ford pour graphes acycliques avec l’utilisation du tri topologique : Θ(n+m)

Non applicable car les graphes ne sont pas acycliques.

2) Floyd-Warshall : Θ(n3)

On cherche le plus court entre 2 sommets, Floyd permet de trouver le plus court chemin entre chaque pair de sommet possible.

3) Dijkstra utilisant un monceau pour les noeuds non solutionnées : O((n+m) Log(n))

Ce qui constitue un avantage seulement pour les graphes peu denses. (ce qui est le cas pour les données qu’on a)

Un graphe dense est un graphe dans lequel le nombre d'arêtes (ou d'arcs) est proche du nombre maximal

Pourquoi le temps d’exécution est en O((n+m) Log(n)) ?

**Time Complexity:** The time complexity of the above code/algorithm looks O(V^2) as there are two nested while loops. If we take a closer look, we can observe that the statements in inner loop are executed O(V+E) times (similar to BFS). The inner loop has decreaseKey() operation which takes O(LogV) time. So overall time complexity is O(E+V)\*O(LogV) which is O((E+V)\*LogV) = O(ELogV)

Source :

<http://www.geeksforgeeks.org/greedy-algorithms-set-7-dijkstras-algorithm-for-adjacency-list-representation/>

<https://en.wikipedia.org/wiki/Fibonacci_heap>

Améliorations de la piste de départ Dijkstra avec Fibonacci Heap :

Note that the above code uses Binary Heap for Priority Queue implementation. Time complexity can be reduced to O(E + VLogV) using Fibonacci Heap. The reason is, Fibonacci Heap takes O(1) time for decrease-key operation while Binary Heap takes O(Logn) time.

Ce qui est O(m + n \* Log(n))

Hachage parfait :