## Dijkstra Algorithm

Dijkstra’s algorithm computes the shortest paths from a starting vertex to all other vertices in a graph. Let G(V, E, w) be an edge weighted graph, where w: E → $R^+$(only **positive weight edge**). Let s, t be two vertices in G (think of s as a source, t as a terminal), and suppose you were asked to compute a shortest (i.e. cheapest) path between s and t.

With adjacency list representation, all vertices of a graph can be traversed in O(V+E) time using BFS. The idea is to traverse all vertices of graph using BFS and use a Min Heap to store the vertices not yet included. Min Heap is used as a priority queue to get the minimum distance vertex from set of not yet included vertices. This is similar like prim’s algorithm.

Following are the detailed steps.

1) Create a Min Heap of size V where V is the number of vertices in the given graph. Every node of min heap contains vertex number and distance value of the vertex.
2) Initialize Min Heap with source vertex as root (the distance value assigned to source vertex is 0). The distance value assigned to all other vertices is INF (infinite).

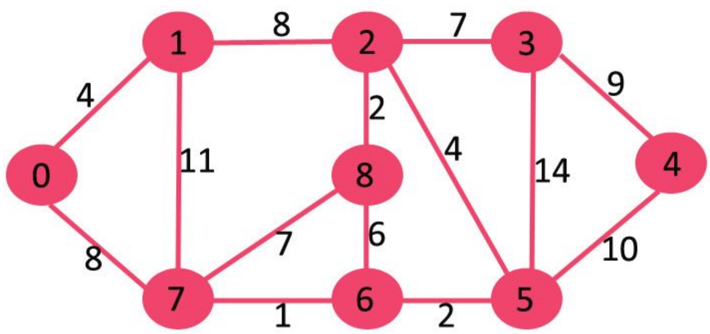
3) While Min Heap is not empty, do following

…..a) Extract the vertex with minimum distance value node from Min Heap. Let the extracted vertex be u.

…..b) For every adjacent vertex v of u, check if v is in Min Heap. If v is in Min Heap and distance value is more than weight of u-v plus distance value of u, then update the distance value of v.

**Example**

Let consider the following graph where source vertex is 0.



Step1:

Source vertex: 0

Current vertex: 0

Path Table

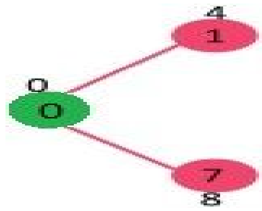
| |
|---|

Distance Table

| V | E |
|---|---|

Heap+ Map

| V | Weight |
|---|---|
| 0 | 0* |
| 1 | ∞ |
| 2 | ∞ |
| 3 | ∞ |
| 4 | ∞ |
| 5 | ∞ |
| 6 | ∞ |
| 7 | ∞ |
| 8 | ∞ |




Step2:

Source vertex: 0

Current vertex: 1

Path Table

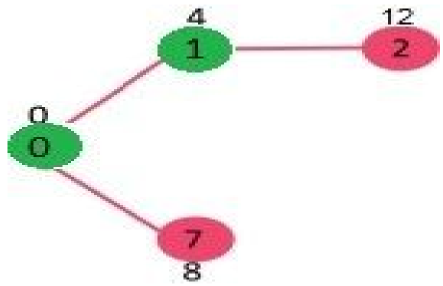
| vertex | parent |
|---|---|
| 0 | null |
| 1 | 0 |
| 7 | 0 |

Distance Table

| vertex | weight |
|---|---|
| 0 | 0 |

Heap+ Map

| vertex | Weight |
|---|---|
| | |
| 1 | 4* |
| 2 | ∞ |
| 3 | ∞ |
| 4 | ∞ |
| 5 | ∞ |
| 6 | ∞ |
| 7 | 8 |
| 8 | ∞ |



Step3:

Source vertex: 0

Current vertex: 7

Path Table

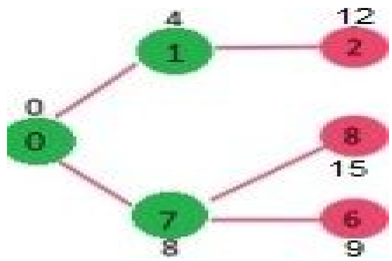
| vertex | parent |
|---|---|
| 0 | null |
| 1 | 0 |
| 7 | 0 |
| 2 | 1 |

Distance Table

| vertex | weight |
|---|---|
| 0 | 0 |
| 1 | 4 |

Heap+ Map

| vertex | Weight |
|---|---|
| | |
| | |
| 2 | 12 |
| 3 | ∞ |
| 4 | ∞ |
| 5 | ∞ |
| 6 | ∞ |
| 7 | 8* |
| 8 | ∞ |




Step4:

Source vertex: 0

Current vertex: 6

Path Table

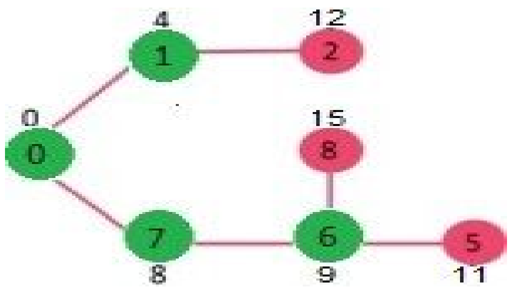
| vertex | parent |
|---|---|
| 0 | null |
| 1 | 0 |
| 7 | 0 |
| 2 | 1 |
| 6 | 7 |
| 8 | 7 |

Distance Table

| vertex | weight |
|---|---|
| 0 | 0 |
| 1 | 4 |
| 7 | 8 |

Heap+ Map

| vertex | Weight |
|---|---|
| | |
| | |
| 2 | 12 |
| 3 | ∞ |
| 4 | ∞ |
| 5 | ∞ |
| 6 | 9* |
| | |
| 8 | 15 |



Step5:

Source vertex: 0

Current vertex: 5

Path Table

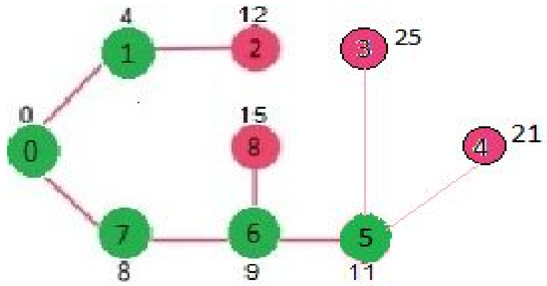
| vertex | parent |
|---|---|
| 0 | null |
| 1 | 0 |
| 7 | 0 |
| 2 | 1 |
| 6 | 7 |
| 8 | 7 |
| 5 | 6 |

Distance Table

| vertex | weight |
|---|---|
| 0 | 0 |
| 1 | 4 |
| 7 | 8 |
| 6 | 9 |

Heap+ Map

| vertex | Weight |
|---|---|
| 2 | 12 |
| 3 | ∞ |
| 4 | ∞ |
| 5 | 11* |
| 8 | 15 |




Step6:

Source vertex: 0

Current vertex: 2

Path Table

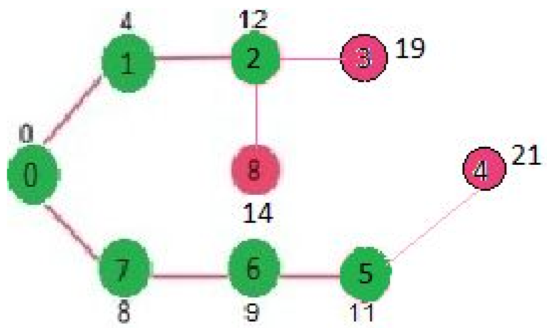
| vertex | parent |
|---|---|
| 0 | null |
| 1 | 0 |
| 7 | 0 |
| 2 | 1 |
| 6 | 7 |
| 8 | 7 |
| 5 | 6 |
| 3 | 5 |
| 4 | 5 |

Distance Table

| vertex | weight |
|---|---|
| 0 | 0 |
| 1 | 4 |
| 7 | 8 |
| 6 | 9 |
| 5 | 11 |

Heap+ Map

| vertex | Weight |
|---|---|
| 2 | 12* |
| 3 | 25 |
| 4 | 21 |
| 8 | 15 |



Step7:

Source vertex: 0

Current vertex: 8

Path Table

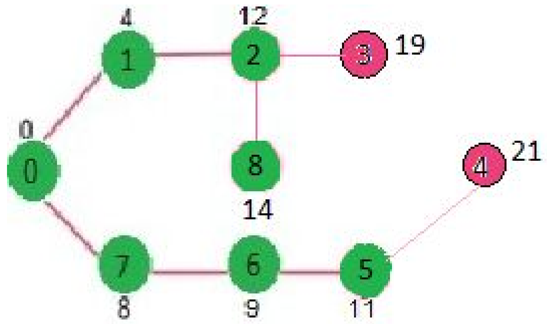
| vertex | parent |
|---|---|
| 0 | null |
| 1 | 0 |
| 7 | 0 |
| 2 | 1 |
| 6 | 7 |
| 8 | 2 |
| 5 | 6 |
| 3 | 2 |
| 4 | 5 |

Distance Table

| vertex | weight |
|---|---|
| 0 | 0 |
| 1 | 4 |
| 7 | 8 |
| 6 | 9 |
| 5 | 11 |
| 2 | 12 |

Heap+ Map

| vertex | Weight |
|---|---|
| | |
| | |
| 3 | 19 |
| 4 | 21 |
| 8 | 14* |




Step8:

Source vertex: 0

Current vertex: 3

Path Table

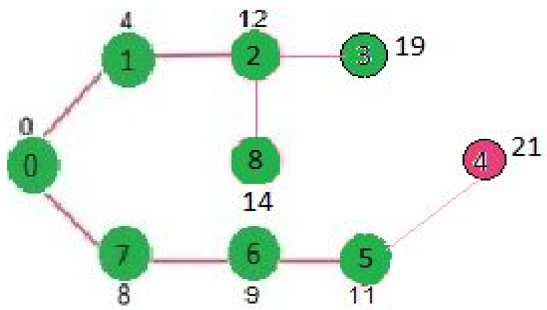
| vertex | parent |
|---|---|
| 0 | null |
| 1 | 0 |
| 7 | 0 |
| 2 | 1 |
| 6 | 7 |
| 8 | 2 |
| 5 | 6 |
| 3 | 2 |
| 4 | 5 |

Distance Table

| vertex | weight |
|---|---|
| 0 | 0 |
| 1 | 4 |
| 7 | 8 |
| 6 | 9 |
| 5 | 11 |
| 2 | 12 |
| 8 | 14 |

Heap+ Map

| vertex | Weight |
|---|---|
| | |
| | |
| 3 | 19* |
| 4 | 21 |



Step9:

Source vertex: 0

Current vertex: 4

Path Table

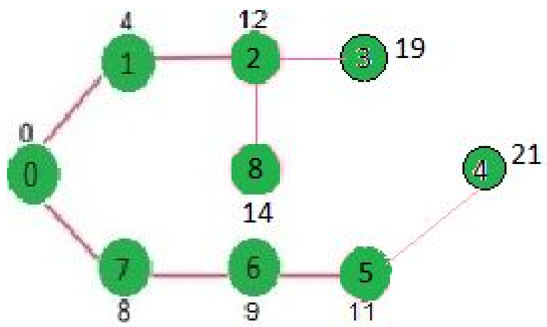
| vertex | parent |
|---|---|
| 0 | null |
| 1 | 0 |
| 7 | 0 |
| 2 | 1 |
| 6 | 7 |
| 8 | 2 |
| 5 | 6 |
| 3 | 2 |
| 4 | 5 |

Distance Table

| vertex | weight |
|---|---|
| 0 | 0 |
| 1 | 4 |
| 7 | 8 |
| 6 | 9 |
| 5 | 11 |
| 2 | 12 |
| 8 | 14 |
| 3 | 19 |

Heap+ Map

| vertex | Weight |
|---|---|
| 4 | 21* |




Step10:

Source vertex: 0

Current vertex: 4

Path Table

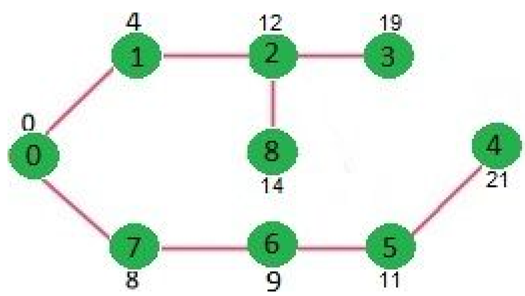
| vertex | parent |
|---|---|
| 0 | null |
| 1 | 0 |
| 7 | 0 |
| 2 | 1 |
| 6 | 7 |
| 8 | 2 |
| 5 | 6 |
| 3 | 2 |
| 4 | 5 |

Distance Table

| vertex | weight |
|---|---|
| 0 | 0 |
| 1 | 4 |
| 7 | 8 |
| 6 | 9 |
| 5 | 11 |
| 2 | 12 |
| 8 | 14 |
| 3 | 19 |
| 4 | 21 |

Heap+ Map

| vertex | Weight |
|---|---|



Step11: Stop(Heap+ Map Empty)

**Time Complexity**

*Case1*: If the graph is represent by adjacency matrix then total time complexity becomes $O(V^2)$ where V is no. of vertices.

*Case2*: The given graph G is represented as an adjacency list.

Priority queue Q is represented as a binary heap.

Here,

- With adjacency list representation, all vertices of the graph can be traversed using BFS in O(V+E) time.
- In min heap, operations like extract-min and decrease-key value takes O(logV) time.
- So, overall time complexity becomes O(E+V) x O(logV) which is O((E + V) x logV) = O(ElogV)