Dijkstra's Algorithm

It is a greedy algorithm that solves the single-source shortest path problem for a directed graph G = (V, E) with nonnegative edge weights, i.e., w (u, v) ≥ 0 for each edge (u, v) ∈ E.

Dijkstra's Algorithm maintains a set S of vertices whose final shortest - path weights from the source s have already been determined. That's for all vertices v ∈ S; we have d [v] = δ (s, v). The algorithm repeatedly selects the vertex u ∈ V - S with the minimum shortest - path estimate, insert u into S and relaxes all edges leaving u.

Because it always chooses the "lightest" or "closest" vertex in V - S to insert into set S, it is called as the **greedy strategy**.

**Dijkstra's Algorithm (G, w, s)**

1. INITIALIZE - SINGLE - SOURCE (G, s)

2. S←∅

3. Q←V [G]

4. while Q ≠ ∅

5. do u ← EXTRACT - MIN (Q)

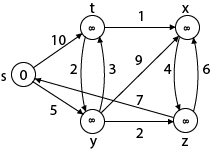
6. S ← S ∪ {u}

7. for each vertex v ∈ Adj [u]

8. do RELAX (u, v, w)

**Analysis:** The running time of Dijkstra's algorithm on a graph with edges E and vertices V can be expressed as a function of |E| and |V| using the Big - O notation. The simplest implementation of the Dijkstra's algorithm stores vertices of set Q in an ordinary linked list or array, and operation Extract - Min (Q) is simply a linear search through all vertices in Q. In this case, the running time is O (|V2 |+|E|=O(V2 ).

**Example:**



**Solution:**

**Step1:** Q =[s, t, x, y, z]

We scanned vertices one by one and find out its adjacent. Calculate the distance of each adjacent to the source vertices.

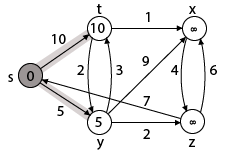
We make a stack, which contains those vertices which are selected after computation of shortest distance.

Firstly we take's' in stack M (which is a source)

M = [S]       Q = [t, x, y, z]

**Step 2:** Now find the adjacent of s that are t and y.

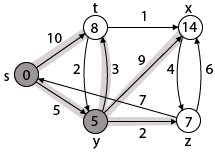
Adj [s] → t, y      [Here s is u and t and y are v]



**Case - (i)**s → t  
                d [v] > d [u] + w [u, v]  
                d [t] > d [s] + w [s, t]  
                ∞ > 0 + 10                [false condition]  
Then       **d [t] ← 10**  
                **π [t] ← 5**  
Adj [s] ← t, y

**Case - (ii)**s→ y  
                d [v] > d [u] + w [u, v]  
                d [y] > d [s] + w [s, y]  
                ∞ > 0 + 5                [false condition]  
                ∞ > 5  
Then       **d [y] ← 5**  
              **π [y] ← 5**

By comparing case (i) and case (ii)  
     Adj [s] → t = 10, y = 5  
     y is shortest  
**y is assigned in 5 = [s, y]**



**Step 3:** Now find the adjacent of y that is t, x, z.

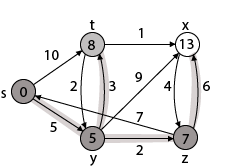
Adj [y] → t, x, z   [Here y is u and t, x, z are v]

**Case - (i)** y →t  
              d [v] > d [u] + w [u, v]  
              d [t] > d [y] + w [y, t]  
              10 > 5 + 3  
              10 > 8  
Then     d [t] ← 8  
              π [t] ← y

**Case - (ii)** y → x  
              d [v] > d [u] + w [u, v]  
              d [x] > d [y] + w [y, x]  
              ∞ > 5 + 9  
              ∞ > 14  
Then      d [x] ← 14  
             π [x] ← 14

**Case - (iii)** y → z  
             d [v] > d [u] + w [u, v]  
             d [z] > d [y] + w [y, z]  
             ∞ > 5 + 2  
             ∞ > 7  
Then      d [z] ← 7  
             π [z] ← y

By comparing case (i), case (ii) and case (iii)  
           Adj [y] → x = 14, t = 8, z =7  
z is shortest  
**z is assigned in 7 = [s, z]**

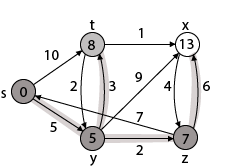


**Step - 4 Now** we will find adj [z] that are s, x

Adj [z] → [x, s]    [Here z is u and s and x are v]

**Case - (i)** z → x  
              d [v] > d [u] + w [u, v]  
              d [x] > d [z] + w [z, x]  
              14 > 7 + 6  
              14 > 13  
Then       d [x] ← 13  
              π [x] ← z

**Case - (ii)** z → s  
              d [v] > d [u] + w [u, v]  
              d [s] > d [z] + w [z, s]  
              0 > 7 + 7  
              0 > 14  
∴ This condition does not satisfy so it will be discarded.  
Now we have x = 13.

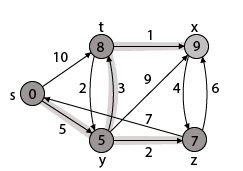


**Step 5:** Now we will find Adj [t]

Adj [t] → [x, y] [Here t is u and x and y are v]

**Case - (i)** t → x  
              d [v] > d [u] + w [u, v]  
              d [x] > d [t] + w [t, x]  
              13 > 8 + 1  
              13 > 9  
**Then       d [x] ← 9**  
              **π [x] ← t**

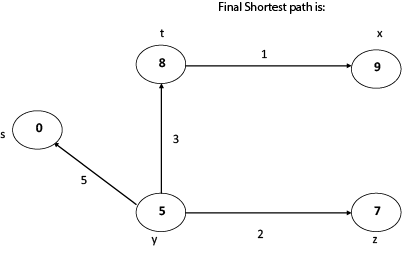
**Case - (ii)** t → y  
              d [v] > d [u] + w [u, v]  
              d [y] > d [t] + w [t, y]  
              5 > 10  
∴ This condition does not satisfy so it will be discarded.



Thus we get all shortest path vertex as

Weight from s to y is 5  
Weight from s to z is 7  
Weight from s to t is 8  
Weight from s to x is 9

These are the shortest distance from the source's' in the given graph.



Disadvantage of Dijkstra's Algorithm:

1. It does a blind search, so wastes a lot of time while processing.
2. It can't handle negative edges.
3. It leads to the acyclic graph and most often cannot obtain the right shortest path.
4. We need to keep track of vertices that have been visited.

**RELEVANT READING MATERIAL AND REFERENCES:**

**Source Notes:**

1. <https://www.javatpoint.com/dijkstras-algorithm>

**Lecture Video:**

1. https://youtu.be/XB4MIexjvY0

**Online Notes:**

1. <http://vssut.ac.in/lecture_notes/lecture1428551222.pdf>

**Text Book Reading:**

1. Cormen, Leiserson, Rivest, Stein, “*Introduction to Algorithms*”, Prentice Hall of India, 3rd edition 2012. problem, Graph coloring.

**In addition: PPT can be also be given.**