Dijkstra's Shortest Path Algorithm

Finds the *shortest path* (minimum cost) from a source vertex to every other vertex in a weighted graph, where all weights are nonnegative.

Analyzes all possibilities for each neighbor of a vertex, picking the neighbor with the shortest distance from source.

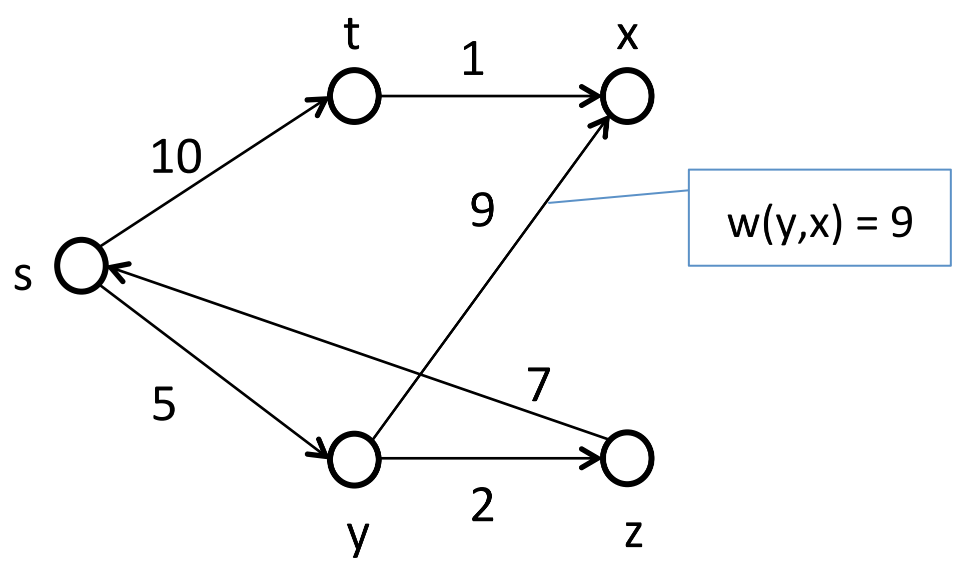
It is a greedy algorithm: solves the problem by making locally optimal decisions at each step with the hope to find the global minimum.

Dijkstra's Shortest Path Algorithm

In a directed graph with *n* vertices and *e* edges

G = (V, E) where V and E are the sets with all vertices and edges respectively

Every edge weight must be nonnegative w(u,v) >= 0



```
DIJKSTRA(G, w, s)
for each vertex v∈G do
  d[v] = ∞
  pred[v] = null
d[s] = 0
S = {}
add s to S
Fringe = {}
for each neighbor t of s do
   add t to Fringe
   d[t] = w(s,t)
   pred[t] = s
while Fringe is not empty do
   u = extract-min(Fringe)
   add u to S
   for each neighbor v of u do
      if d[v] > d[u] + w(u,v) then
          d[v] = d[u] + w(u,v)
          pred[v] = u
      if v not in Fringe, add v to Fringe
```

Shortest distance from **s** to **v** so far

Previous vertex on the optimal path from **s** to **v**

Set of vertices whose final shortest path from **s** have already been determined

Vertices whose final shortest path from **s** have not been determined

Finds the vertex in Fringe with the minimum value of d[v]

Dijkstra's Running Time Analysis

The running time of Dijkstra's algorithm depends on:

1. How many times the distance between two vertices is computed? e times
2. How many times a vertex is selected from the Fringe? n times

The Fringe can be either a *Linked List* or an *Updatable Min Heap*

Dijkstra's Running Time Analysis

```
DIJKSTRA(G, w, s)
for each vertex v ∈ G do
  d[v] = ∞
  pred[v] = null
d[s] = 0
S = {}
add s to S
Fringe = {}
for each neighbor t of s do
   add t to Fringe
   d[t] = w(s,t)
   pred[t] = s
while Fringe is not empty do
   u = extract-min(Fringe)
   add u to S
   for each neighbor v of u do
      if d[v] > d[u] + w(u,v) then
         d[v] = d[u] + w(u,v)
         pred[v] = u
      if v not in Fringe, add v to Fringe
```

The initialization loop is executed **n** times

Executed **n** times

Fringe implementation:

- Linked List
- Min Heap

Only executed **e** times in total

Dijkstra's Running Time Analysis

Fringe: Linked List

To find the minimum distance vertex we must compare all items in the LL.

n iterations:

1. n items: n-1 compares
2. n-1 items: n-2 compares
3. ...
4. 1 item: 0 compares

$= (n-1)+(n-2)+...+2+1$

$= (n-1)(n-1+1)/2 = n(n-1)/2$

$O(n^2)$

Fringe: Min Heap

To find the minimum distance vertex we just remove from the Heap.

n iterations:

1. remove root from a Min Heap with n items
2. remove root from a Min Heap with n-1 items
3. ...
4. remove root from a Min Heap with 1 item

$= \log n + \log n\text{-}1 + ... + \log 1$

$= \log n! \cong n \log n$

$O(n \log n)$

Dijkstra's Running Time Analysis

Fringe: Linked List

$O(n^2 + e) \Rightarrow O(n^2)$

Fringe: Min Heap

$O(n\log n + e\log n) \Rightarrow O(n + e \log n)$

Why **e** is never larger than **n^2**?

The maximum number of edges in a directed graph is:

$n(n-1) = n^2 - n = O(n^2)$