Single Source Shortest Path Algorithms

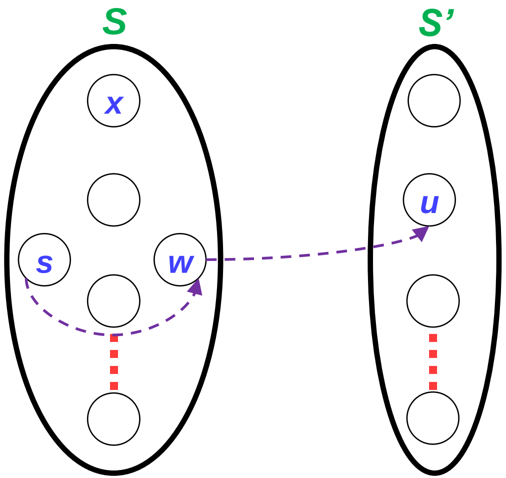
Dr. Shrutilipi Bhattacharjee, Assistant Professor, Dept. of IT, NIT Karnataka, India

Dijkstra's Algorithm

- We know how to compute the cost of the shortest path of any given vertex from the source, when all edges had non-negative costs

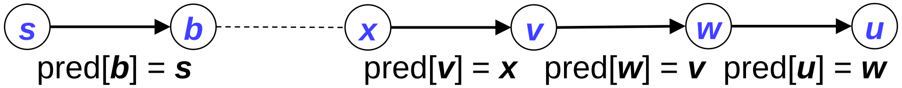
- How to compute single source shortest path?
  - The vertex ***w*** is going to move from ***S'*** to ***S***
  - When this vertex is moving, it causes the distance label of the adjacent vertices to be updated
  - That means the best path from ***s*** to ***u*** includes ***w***
  - If d[u] = d[w] + c(w, u), then the shortest path from ***s*** to ***u*** (using vertices of ***s*** to ***w)*** has ***w*** preceding ***u***
  - This is the information we will maintain with vertex ***u***: what is the vertex preceding it on the shortest path that we have found so far?

if d[u] > d[w] + c(w, u) **then**
d[u] ← d[w] + c(w, u)



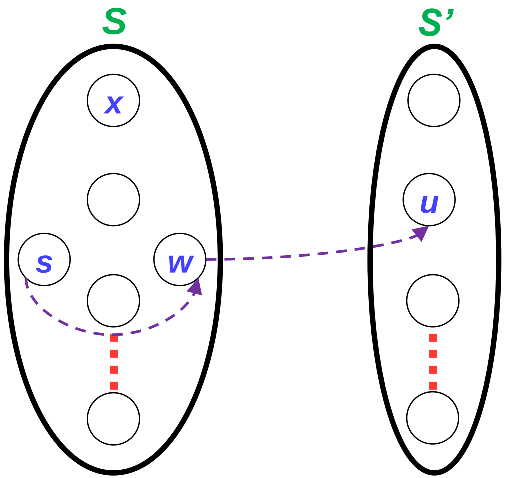
Dijkstra's Algorithm

- How to compute single source shortest path?
  - With each vertex, we maintain predecessor information
  - If ***w*** = pred[***u***], then ***w*** is the vertex preceding ***u*** on the best path from ***s*** to ***u***



  - We have to keep doing this till we reach the source vertex
  - This will give us the shortest path

if d[u] > d[w] + c(w, u) **then**
  d[u] ← d[w] + c(w, u)



if d[u] > d[w] + c(w, u) **then**
  d[u] ← d[w] + c(w, u)
  pred[u] = w

Dijkstra's Algorithm

- Suppose we wanted to find the shortest path from ***s*** to a vertex ***v***?

x = ***v***

while pred[x] != NULL **then** //predecessor of my source to null

    print(x)

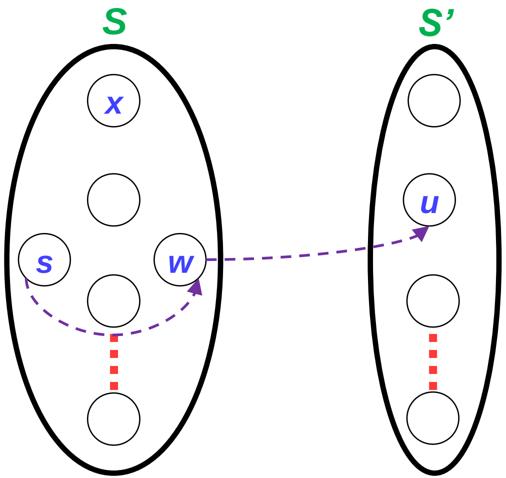
    pred[x] = x

print(s)

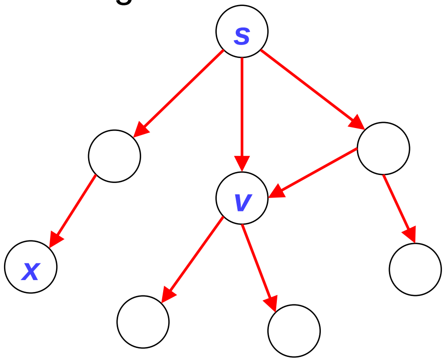
if d[u] > d[w] + c(w, u) **then**

    d[u] ← d[w] + c(w, u)

    pred[u] = w



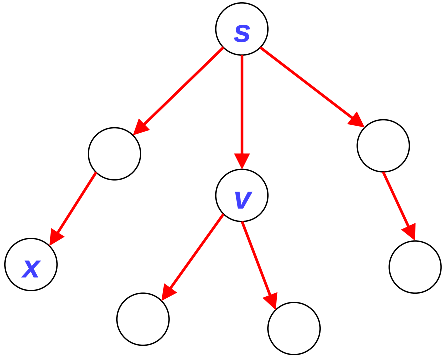
- In Dijkstra's algorithm, can this situation occur?
- How many predecessor edges can be there for each vertex?



Dijkstra's Algorithm

- Predecessor edge?
  - The edge connecting the predecessor vertex of a given vertex to itself
  - The $V - 1$ predecessor edges
  - Using these predecessor edges, we can reach to any vertex from ***s***

- Then what is this subgraph?
  - If we ignore the directions of the edges, then it is a connected subgraph with $V - 1$ predecessor edges
  - It will be a tree that is a ***spanning tree***
  - But if we bring back the direction, (note that in a directed graph, there is no notion of the ***spanning tree***), it is called a ***branching***

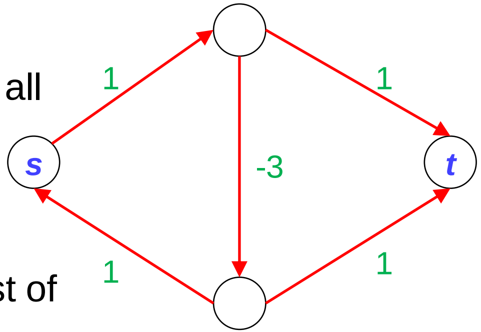


- ***Branching:*** It is basically a sub graph with $V - 1$ edges such that we can reach from a specific root vertex to every other vertex

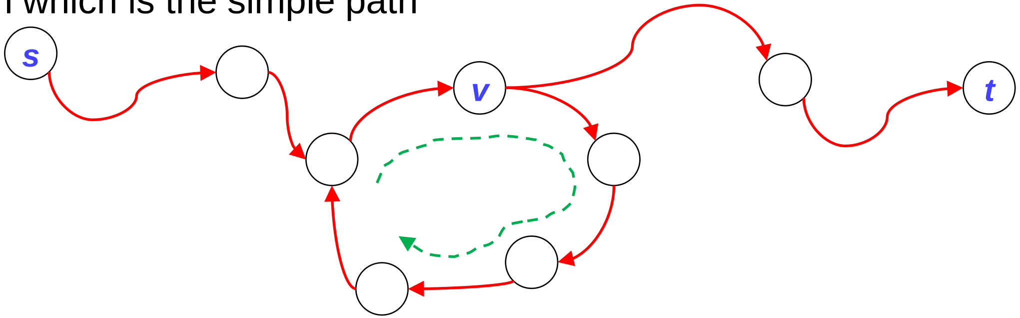
- In the context of shortest path, this is called the ***shortest path tree***
  - We can compute this tree in the same time as required by Dijkstra's algorithm

Graph with Negative Costs

- Edges can have positive and negative costs
- Given a source, we are trying to find out the shortest path from this source to all the vertices of the graph
- ***Assumption:*** There is no negative cycle
  - If the graph has a negative cycle then the shortest path is not defined, cost of the shortest path can be -∞

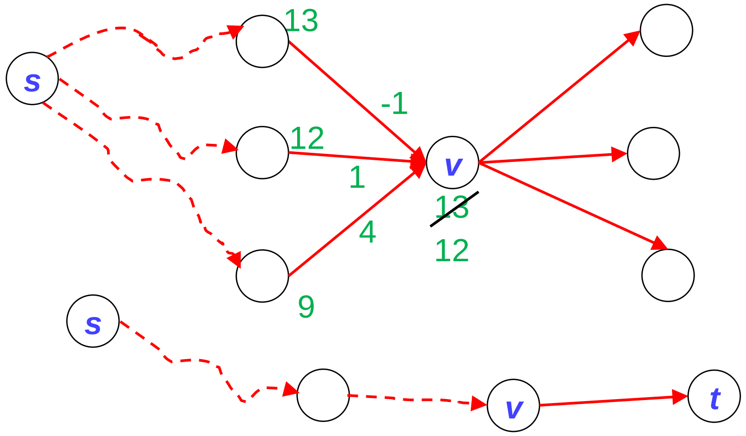


- What can we say about the shortest path from ***s*** to ***v***? Can it be a non simple path (no cycle on the path)?
- There always exists a shortest path which is the simple path



Algorithm

- In one round, every vertex will do the following:
  - Propagate the information to all its neighbors
  - Receive information from its in-neighbors
  - Update its own information
- Information is the current distance label
- How much time one vertex is spending? What is the total time?
  - One node is spending time proportional to its degree, thus total time is proportional to the number of edges in graph ***G***
- What are in the initial diatance labels of each vertex?
  - d[***s***] = 0, ∀ v ∈ V, d[***v***] = ∞
- How many rounds will be there?
  - *V – 1*

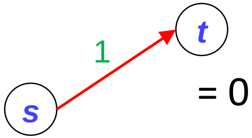


- Suppose this the shortest path from ***s*** to ***t***
- What is the maximum number of edges in the shortest path of any vertex ***v*** from ***s***
  - Shortest path is simple
  - All the shortest path tree forms a spanning tree
  - The shortest path of any vertex ***v*** from ***s*** also consists of the shortest path of the intermediate vertices from ***s***

Algorithm

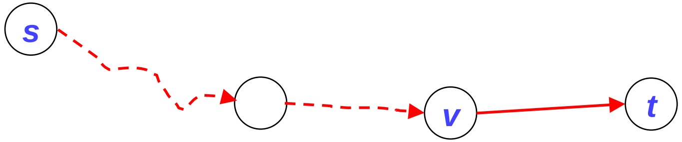
- **Claim:** If the shortest path from ***s*** to vertex ***t*** has ***l*** edges on it, then d[***t***] gets the value of the shortest path after ***l*** rounds

- **Proof:** By induction on ***l***
  - Base case:



= 0 + 1 = 1

  - Induction case: Suppose vertex ***t*** is at ***l*** hop distance from ***s*** and uptil (***l* – *1***) round, everything holds correctly
  - Then, at ***l***th round, d[***t***] will get updated as: min(d[***t***], d[***t***] + c(***v***, ***t***))
  - Thus, the shortest path cost gets updated at ***l***th round

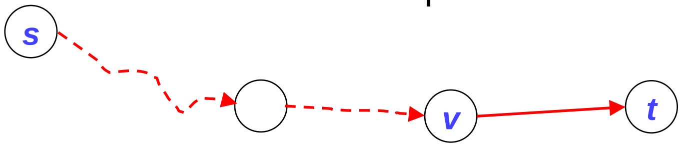


- Suppose this the shortest path from ***s*** to ***t***
- What is the maximum number of edges in the shortest path of any vertex ***v*** from ***s***
  - Shortest path is simple
  - All the shortest path forms a spanning tree
  - The shortest path of any vertex ***v*** from ***s*** also consists of the shortest path of the intermediate vertices from ***s***

Algorithm

- What is the running time?
  - *O(VE)*

- Why it is correct algorithm?
  - **Claim:** If a vertex ***t*** gets a certain distance label ***d*** in round ***r***, then we have found a path of length ***d*** from **s** to ***t***
    - Suppose this claim is true uptil round ***r – 1***
    - For any ***t***, if its distance label is coming from ***v***, d[***v***] is finalized at label ***r – 1***
    - If this not the shortest path from ***s*** to ***v***, there is a contradiction on the assumption



d[s] = 0

∀ ***v*** ∈ V, d[***v***] = ∞

for i = 1 to *V – 1* ***do***

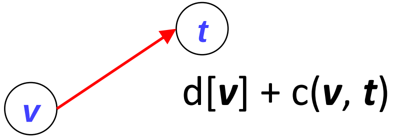
  for all ***v*** **do**

    d'[***v***] = d[***v***]

    for each ***w*** ∈ Adj[***v***] **do**

      d'[***v***] ← min(d'[***v***], d[***w***] + c(***v***, ***w***))

  d[] ← d'[]



All Pairs Shortest Path (APSP)

- If edge costs are +*ve*, what is the time complexity?
  - $O(V(E \log V))$
- If edge costs are –*ve*, what is the time complexity?
  - $O(V(VE))$

- **Best known algorithms:**

	Edge costs	What we have learnt	Best known algorithms
SSSP	+ve	$O(E \log V)$	$O(E + V \log V)$
	-ve	$O(VE)$	$O(VE)$
APSP	+ve	$O(V E \log V)$	$O(V(E + V \log V))$
	-ve	$O(V^2E)$	$O(VE + V^2 \log V)$

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6^{th} Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in, shrutilipi.bhattacharjee@tum.de