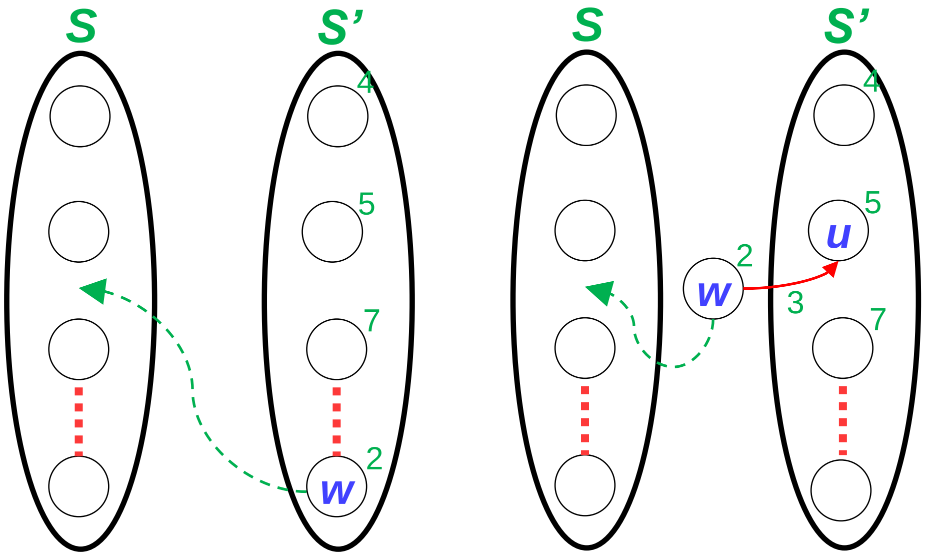
Dijkstra’s Algorithm: Correctness and Analysis

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

Dijkstra’s Algorithm

- The vertex in ***S’*** for which ***d*** is minimum is moved to ***S***
- Now, what happens when this move is done?
  - We can go to ***w*** from ***s*** with cost **2**
  - And we can reach vertex ***u*** from ***s*** with cost **5**



Dijkstra(G, c, s)

For each v ∈ V

  do d[v] ← ∞

d[s] ← 0

S ← Φ //set of discovered vertices

S’ ← V

while S’ ≠ Φ **do**

  w ← Extract-Min(S’)

  S ← S U {w}

  for each u ∈ Adj[w] **do**

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

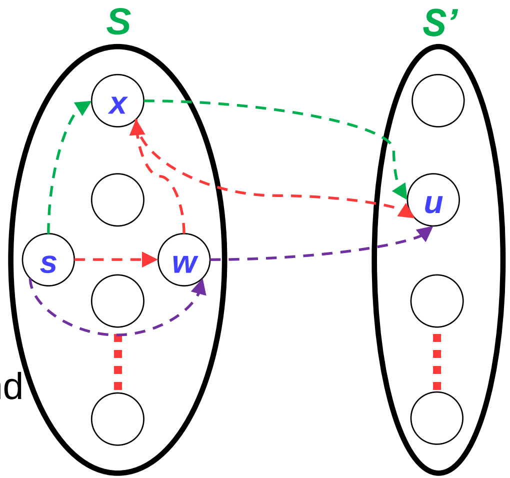
      d[u] ← d[w] + c(w, u)

Dijkstra's Algorithm Correctness

- For all ***u*** ∈ **S**, ***d[u]*** = Length of shortest path from ***s*** to ***u***
- For all ***u*** ∈ **S'**, ***d[u]*** = Length of shortest path from ***s*** to ***u*** that includes only vertices from the set ***S*** (except ***u***)
- Two crucial steps in the algorithm:

while S' ≠ Φ **do**
    w ← Extract-Min(S')
    S ← S U {w}

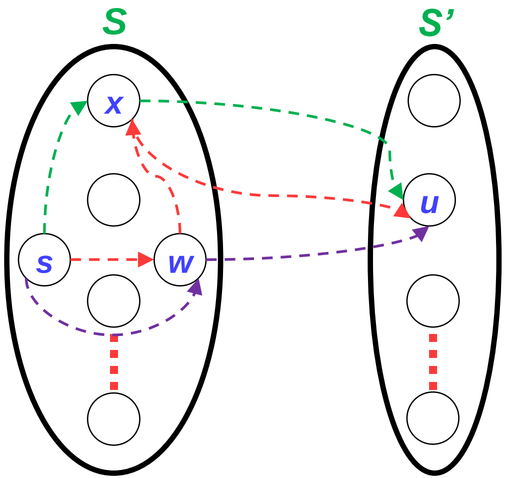
if d[u] > d[w] + c(w, u) **then**
    d[u] ← d[w] + c(w, u)



- We will prove:
  – When vertex ***w*** is added from ***S'*** to ***S***, the weight updation is correct
  – Whenever ***w*** is added to ***S***, ***d[w]*** = ***c(s, w)***, i.e., that ***d[w]*** is minimum, and that equality is maintained thereafter

Dijkstra's Algorithm Correctness

- Three possible paths can occur when vertex ***w*** is added from ***S'*** to ***S*** and we are trying to find the shortest path from ***s*** to another vertex ***u*** in ***S'***:
  - ***s*** to ***w*** to ***u***
  - ***s*** to some other vertex ***x*** in ***S*** to ***u***
  - ***s*** to ***w*** to some other vertex ***x*** to ***u*** → ***Possible?***
    - Only possible when ***c(s, x)*** = ***c(s, w) + c(w, x)***
    - Then both are shortest path from ***s*** to ***u***
    - That is the cost of the path from ***s*** → ***w*** → ***x*** is no less than the length of the path from ***s*** → ***x*** → ***u*** (shortest path from ***s*** → ***x*** doesn't include ***w***, only includes vertices from ***S***)
    - Then this cost is already included in ***d[u]***



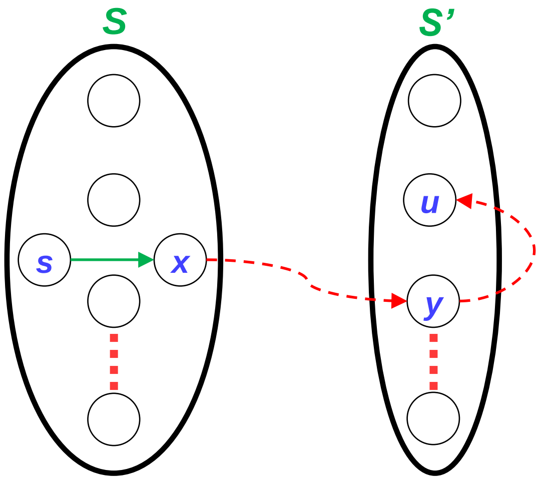
Dijkstra's Algorithm Correctness

- ***u*** = Vertex with the smallest ***d*** value in ***S'***
  - Whenever ***u*** is added to ***S***, ***d[u]*** = ***c(s, u)***, i.e., that ***d[u]*** is minimum, and that equality is maintained thereafter
  - Claim is: ***d[u]*** is the cost of shortest path from ***s*** to ***u***
- Why this is true?
- Proof: **By contradiction !!!**
  - Note that for all ***v*** ∈ ***S'***, ***d[v]*** ≥ ***c(s, v)***
  - Let ***u*** be the first vertex picked such that there is a shorter path than ***d[u]***, i.e., that ***d[u]*** > ***c(s, u)***
  - We will show that this assumption leads to a contradiction
  - Let ***y*** be the first vertex in ***S'*** on the actual shortest path from ***s*** to ***u***
  - Then it must be that ***d[y]*** = ***c(s, y)*** because…

Dijkstra's Algorithm Correctness

- ***d[x]*** is set correctly for ***y***'s predecessor ***x*** in ***S*** on the shortest path (by choice of ***u*** as the first vertex for which ***d*** is set incorrectly)
- when the algorithm inserted ***x*** into ***S***, it relaxed the edge ***(x, y)***, assigning ***d[y]*** the correct value

- ***d[u]*** > ***c(s, u)*** // initial assumption
  - > ***c(s, y)*** + ***c(y, u)*** // optimal substructure
  - > ***d[y]*** + ***c(y, u)*** // correctness of ***d[y]***
  - ≥ ***d[y]*** // no negative weights
- But if ***d[u]*** > ***d[y]***, the algorithm would have chosen ***y*** (from the ***S'***) to process next, not ***u*** → Contradiction!!!
- Thus ***d[u]*** = ***c(s, u)*** at time of insertion of ***u*** into ***S***, and Dijkstra's algorithm is correct



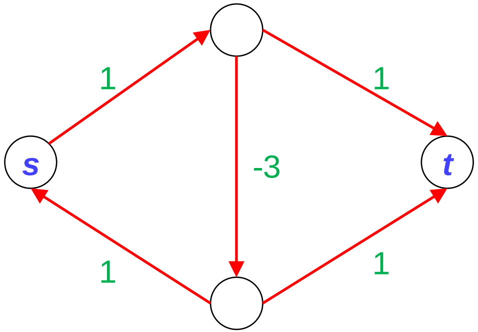
Dijkstra's Algorithm Time Complexity

- The time complexity looks ***O(V²)*** as there are two loops, the while loop and the nested for loop
- The statements in inner loop are executed ***O(V + E)*** times (similar to **BFS**)
- The inner loop has **decreasePriority()** operation which takes ***O(log V)*** time
- So overall time complexity is ***O(V + E)*** * ***O(log V)*** which is ***O((V + E) log V)*** = ***O(E log V)***
- The above code uses *Binary Heap* for *Priority Queue* implementation
- Time complexity can be further reduced to ***O(E + V log V)*** using *Fibonacci Heap*, as *Fibonacci Heap* takes ***O(1)*** time for **decreasePriority()** operation while *Binary Heap* takes ***O(log V)*** time

```
Dijkstra(G, w, s)

For each v ∈ V do
    d[v] ← ∞
    Heap.decresePriority(v, d[v])
d[s] ← 0
Heap.insert(s, 0)
while S' ≠ Φ do
    w = Heap.deleteMin()
    S ← S U {w}
    for each u ∈ Adj[w] do
        d[u] = min(d[u], d[w] + c(w, u))
        Heap.decreasePriority(u, d[u])
```

Dijkstra's Algorithm for Graph with –ve Cost



- Graph with negative cost in the edges
  - If the graph has a negative cycle then the shortest path is not defined, cost of the shortest path can be -∞
  - Negative cycle
- When negative costs are important?
  - The weights can be used to represent the heat produced during a chemical reaction
    - Vertices are the compounds
    - Edge ***(u, v)*** represents that the compound ***v*** can be obtained (*chemically reduced*) from ***u***
    - Negative cost ***–c*** represents that ***c*** cost is needed to get back ***u*** from ***v***
  - The traffic conditions in a map where more negative represents more congestion

- Will Dijkstra's algorithm work for the graphs with negative cost in the edges?

Next Lecture

Minimum Spanning Trees

Thank you for your attention...

Any question?

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