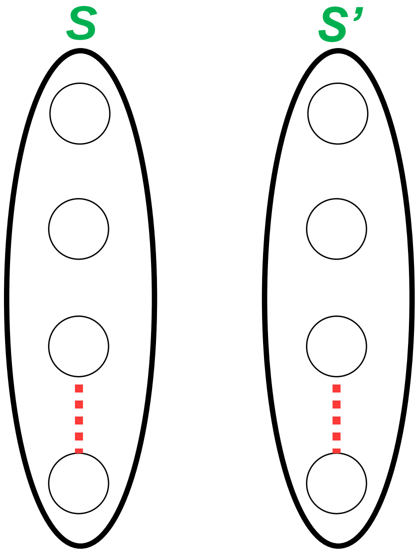
Dijkstra's Shortest Path Algorithm

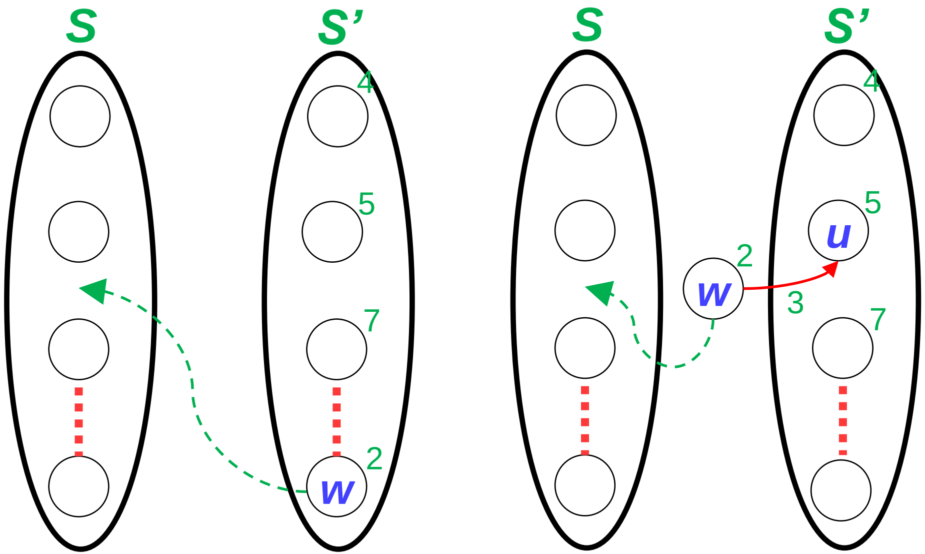
Algorithm

- ***S*** is the set of vertices to which we have found the shortest path from starting vertex ***s***
- ***S'*** is the set of vertices for which we have not yet found the shortest path
  - We just have an upper bound on the value of the shortest path
  - We know that the shortest path less than this number
  - But, I dont know, what is the correct value of shortest path?
- That means, in every step we will include one vertex from ***S'*** to ***S***
- For all ***v*** in ***S'***, ***d***[***v***] is an upper bound on the length of the shortest path from ***s*** to ***v***
- For all ***v*** in ***S***, ***d***[***v***] is the length of the shortest path
- So which is the vertex we will move from ***S'*** to ***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**

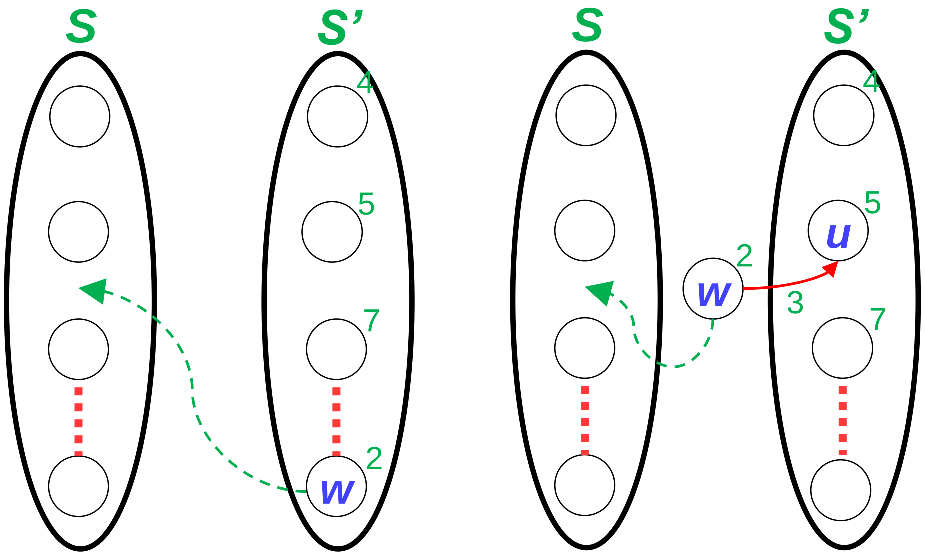



```
Dijkstra(G, c, s)

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)
```

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**

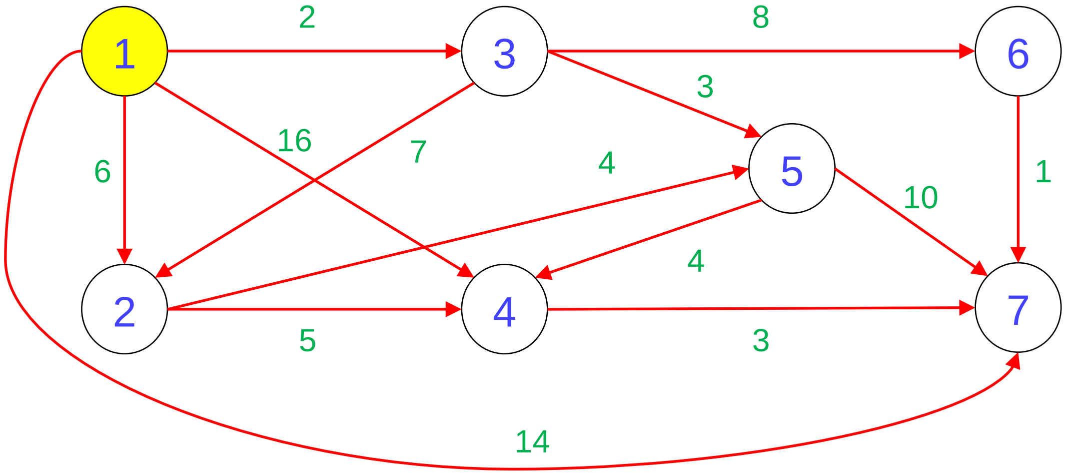



```
Dijkstra(G, w, s)

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

Greedy Single Source All Destinations

- Let d(i) (distanceFromSource(i)) be the length of a shortest one edge extension of an already generated shortest path, the one edge extension ends at vertex i
- The next shortest path is to an as yet unreached vertex for which the d() value is least
- Let p(i) (predecessor(i)) be the vertex just before vertex i on the shortest one edge extension to i

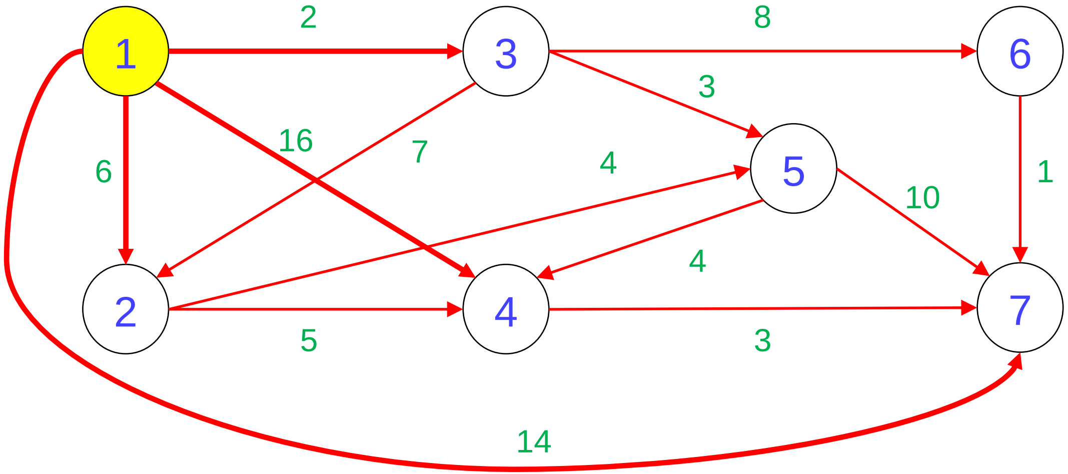


1

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	∞	∞	∞	∞	∞	∞
p	-	-	-	-	-	-	-

Greedy Single Source All Destinations

- Let d(i) (distanceFromSource(i)) be the length of a shortest one edge extension of an already generated shortest path, the one edge extension ends at vertex i
- The next shortest path is to an as yet unreached vertex for which the d() value is least
- Let p(i) (predecessor(i)) be the vertex just before vertex i on the shortest one edge extension to i

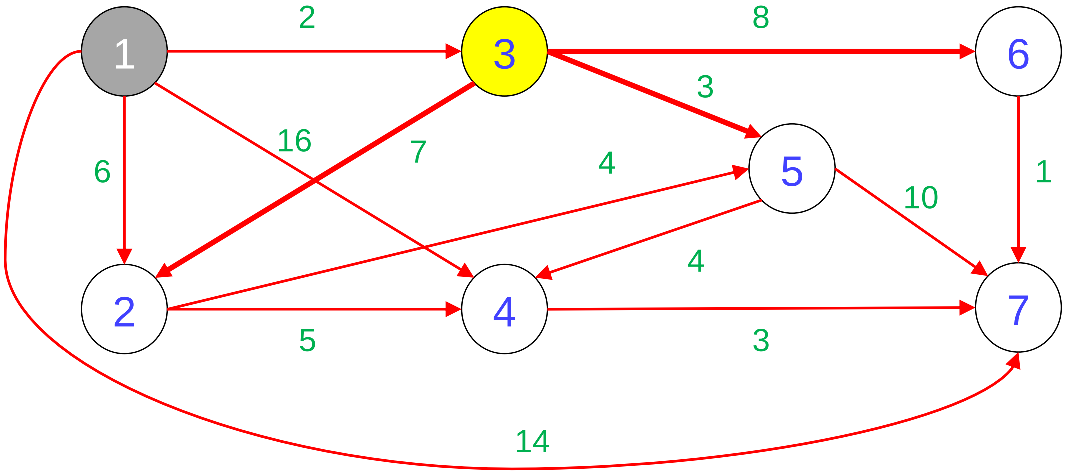


1

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	16	-	-	14
p	-	1	1	1	-	-	1

Greedy Single Source All Destinations

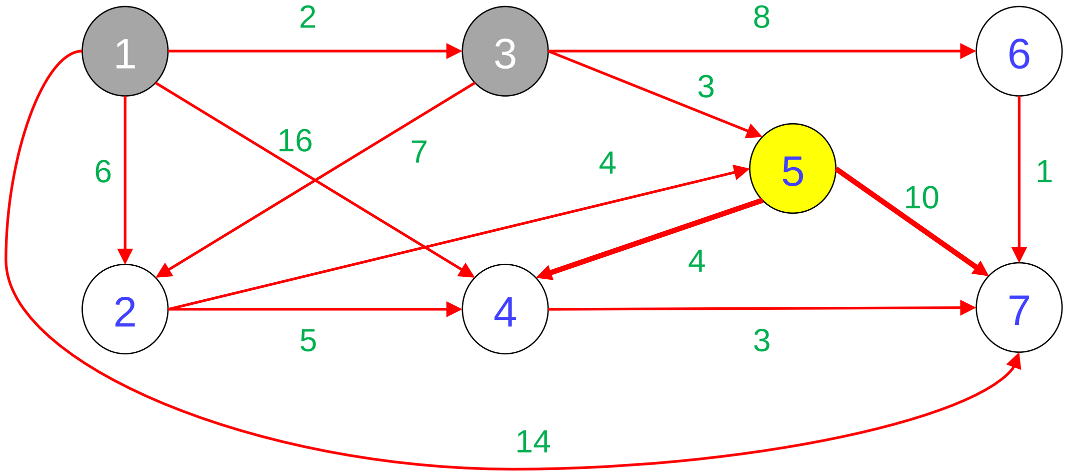
- Let d(i) (distanceFromSource(i)) be the length of a shortest one edge extension of an already generated shortest path, the one edge extension ends at vertex i
- The next shortest path is to an as yet unreached vertex for which the d() value is least
- Let p(i) (predecessor(i)) be the vertex just before vertex i on the shortest one edge extension to i

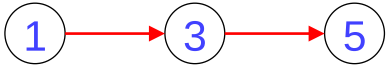


	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	16	5	10	14
p	-	1	1	1	3	3	1

Greedy Single Source All Destinations

- Let d(i) (distanceFromSource(i)) be the length of a shortest one edge extension of an already generated shortest path, the one edge extension ends at vertex i
- The next shortest path is to an as yet unreached vertex for which the d() value is least
- Let p(i) (predecessor(i)) be the vertex just before vertex i on the shortest one edge extension to i

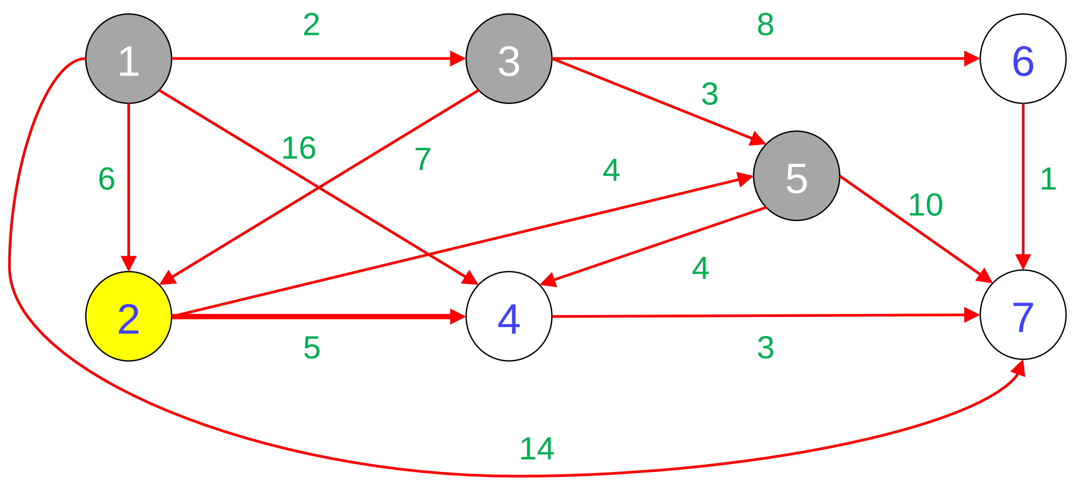


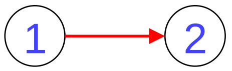


	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	14
p	-	1	1	5	3	3	1

Greedy Single Source All Destinations

- Let d(i) (distanceFromSource(i)) be the length of a shortest one edge extension of an already generated shortest path, the one edge extension ends at vertex i
- The next shortest path is to an as yet unreached vertex for which the d() value is least
- Let p(i) (predecessor(i)) be the vertex just before vertex i on the shortest one edge extension to i

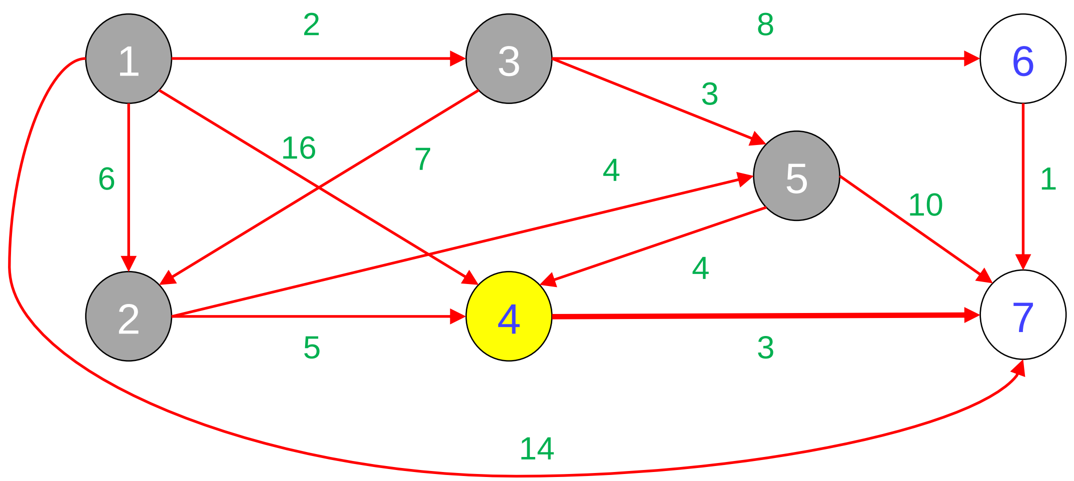




	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	14
p	-	1	1	5	3	3	1

Greedy Single Source All Destinations

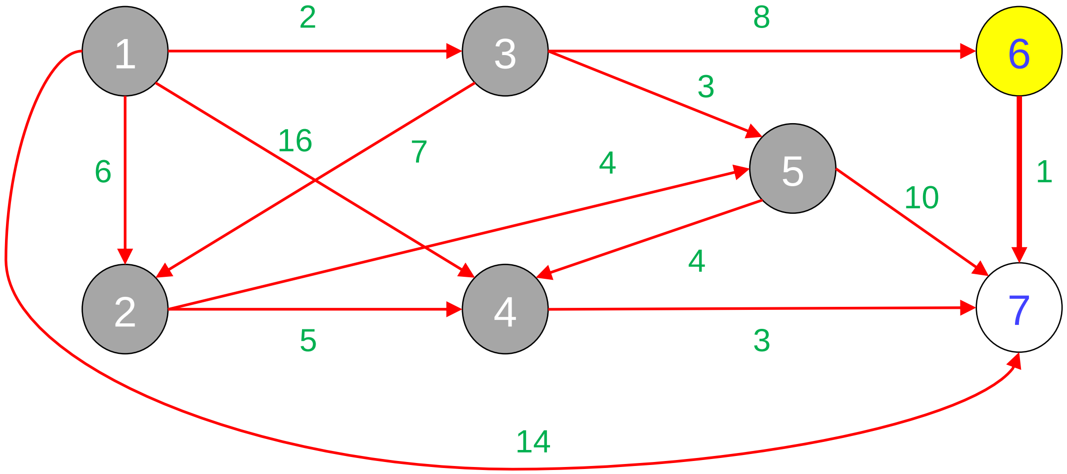
- Let d(i) (distanceFromSource(i)) be the length of a shortest one edge extension of an already generated shortest path, the one edge extension ends at vertex i
- The next shortest path is to an as yet unreached vertex for which the d() value is least
- Let p(i) (predecessor(i)) be the vertex just before vertex i on the shortest one edge extension to i

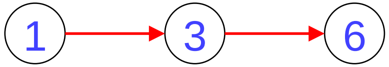


	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	12
p	-	1	1	5	3	3	4

Greedy Single Source All Destinations

- Let d(i) (distanceFromSource(i)) be the length of a shortest one edge extension of an already generated shortest path, the one edge extension ends at vertex i
- The next shortest path is to an as yet unreached vertex for which the d() value is least
- Let p(i) (predecessor(i)) be the vertex just before vertex i on the shortest one edge extension to i

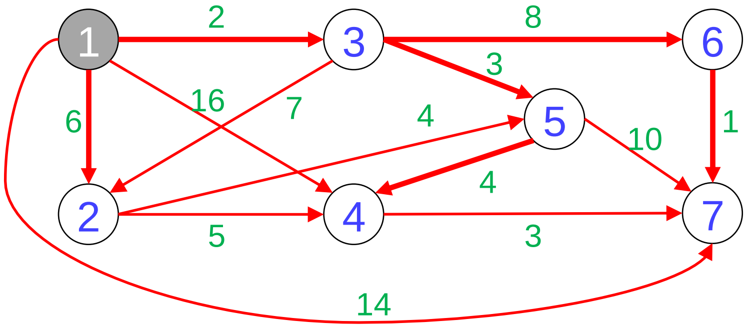




	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	11
p	-	1	1	5	3	3	6

Greedy Single Source All Destinations

Path	Length
1	0
1 → 3	2
1 → 3 → 5	5
1 → 2	6
1 → 3 → 5 → 4	9
1 → 3 → 6	10
1 → 3 → 6 → 7	11



	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	11
p	-	1	1	5	3	3	6

Single Source Single Destination

- Terminate single source all destinations greedy algorithm as soon as shortest path to desired vertex has been generated

Data Structures For Dijkstra's Algorithm

- The greedy single source all destinations algorithm is known as ***Dijkstra's algorithm***
- Implement ***d()*** and ***p()*** as 1D arrays
- Keep a linear list ***S'*** of reachable vertices to which shortest path is yet to be generated
- Select and remove vertex ***v*** in ***S'*** that has smallest ***d()*** value
- Update ***d()*** and ***p()*** values of vertices adjacent to ***v***

Complexity

- ***O(V)*** to select next destination vertex
- ***O(out-degree)*** to update ***d()*** and ***p()*** values when adjacency lists are used
- ***O(V)*** to update ***d()*** and ***p()*** values when adjacency matrix is used
- Selection and update done once for each vertex to which a shortest path is found
- Total time is $O(V^2 + E) = O(V^2)$

Next Lecture

Dijkstra’s Algorithm: Correctness and Analysis

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