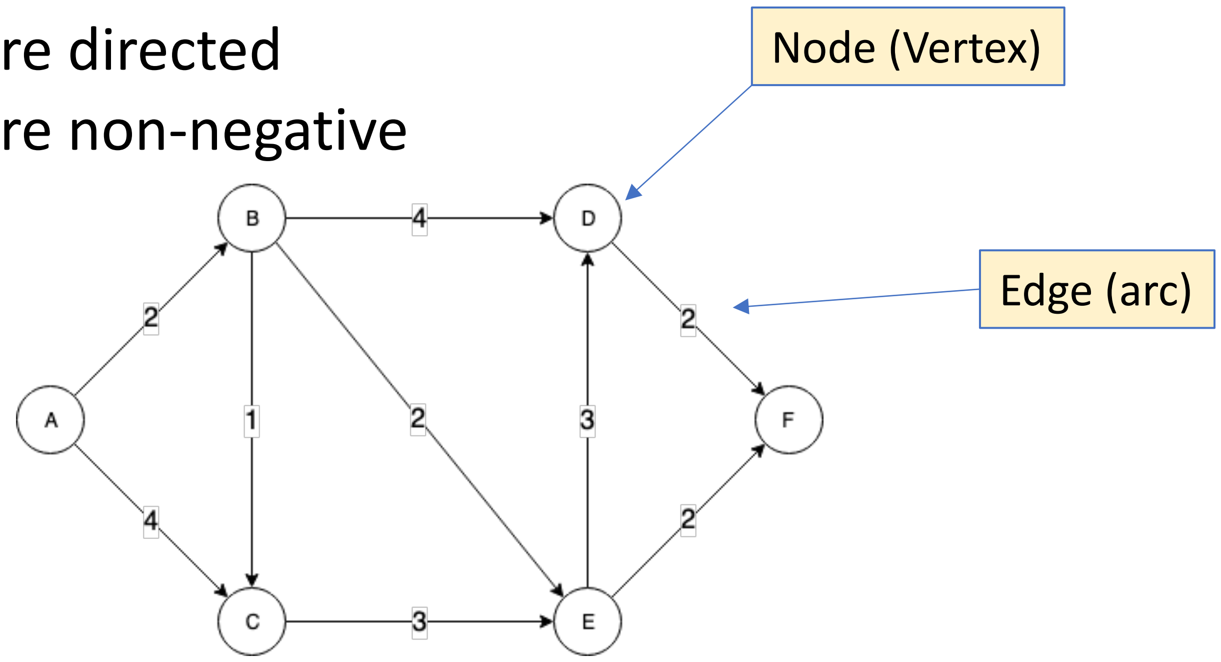
Lecture 17: The Shortest Path Problem

Jianjun Chen (Jianjun.Chen@xjtlu.edu.cn)

Problem Description

- Shortest path problem: Among a set of vertices ($\mathbb{V}$) connected by edges ($\mathbb{E}$), there is a source vertex and a destination vertex. The goal is to find a path with the sum of the weights minimised.
  - All edges are directed
  - All edges are non-negative



Shortest Path

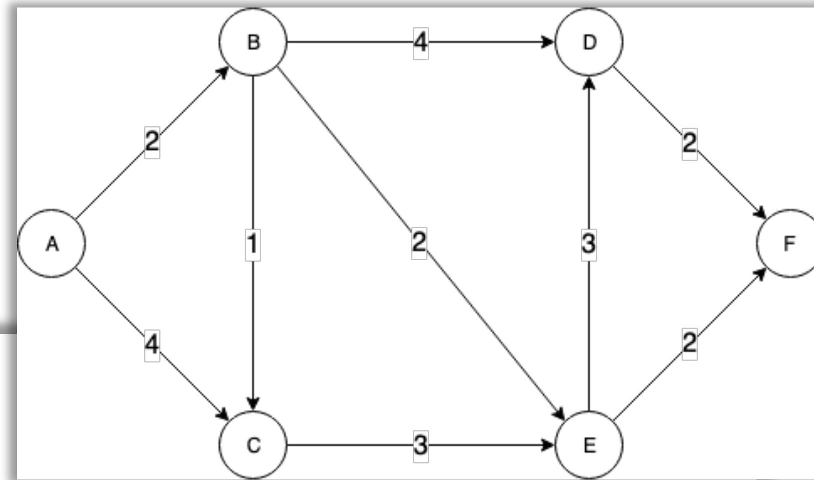
- Search algorithms:
  - **Dijkstra's Algorithm**
  - Bellman-Ford Algorithm
  - Breadth-First Search

- Applications of the shortest path problem:
  - Map searching
  - Networks Routing
  - Computer graphics

Dijkstra’s Algorithm

- Starting from a source `s` in `G(V,E)`
- Each node in `V` has a distance-from-source value.
    - Originally they are infinite. Except for s = 0.
- Maintain two sets, `S` and `Q`
    - `S` contains the vertices whose minimum distance-from-source values have already been determined.
    - `Q = V - S`
- Repeatedly select the vertex `u` from `Q`
    - `u` has the minimum distance-from-source estimate in `Q`
    - Remove `u` from `Q` and add `u` to `S`
    - Update the distances of all neighbouring vertices of `u`
- Results would be a tree

Try three while loops



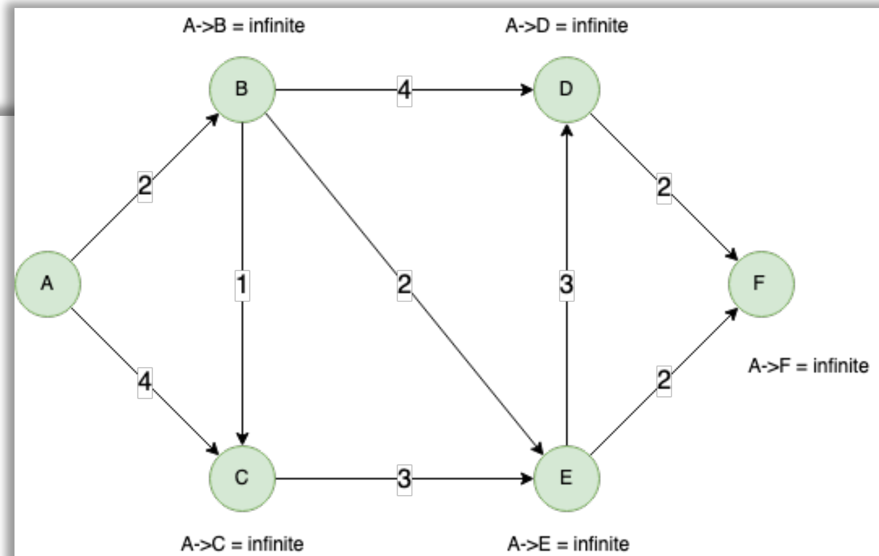

```
Dijkstra(Graph, source):
  create vertex set S and Q
  for each vertex v in Graph: // Initialization
     dist[v] ← INFINITY   // Unknown distance from source to v
     add v to Q           // All nodes initially in Q (unvisited nodes)
  dist[s] ← 0           // Distance from source to source

  while Q is not empty:
     u ← vertex in Q with min dist[u] // Source node will be selected first
     remove u from Q and add u to S
     for each neighbor v of u:        // relax v based on u
        alt ← dist[u] + length(u, v)
        if alt < dist[v]:             // A shorter path to v has been found
           dist[v] ← alt
```

Dijkstra’s Algorithm

```
Dijkstra(Graph, source):
  create vertex set S and Q
  for each vertex v in Graph:
     dist[v] ← INFINITY
     add v to Q
  dist[s] ← 0

  while Q is not empty:
     u ← vertex in Q with min dist[u]
     remove u from Q and add u to S
     for each neighbor v of u:
        alt ← dist[u] + length(u, v)
        if alt < dist[v]:
           dist[v] ← alt
```

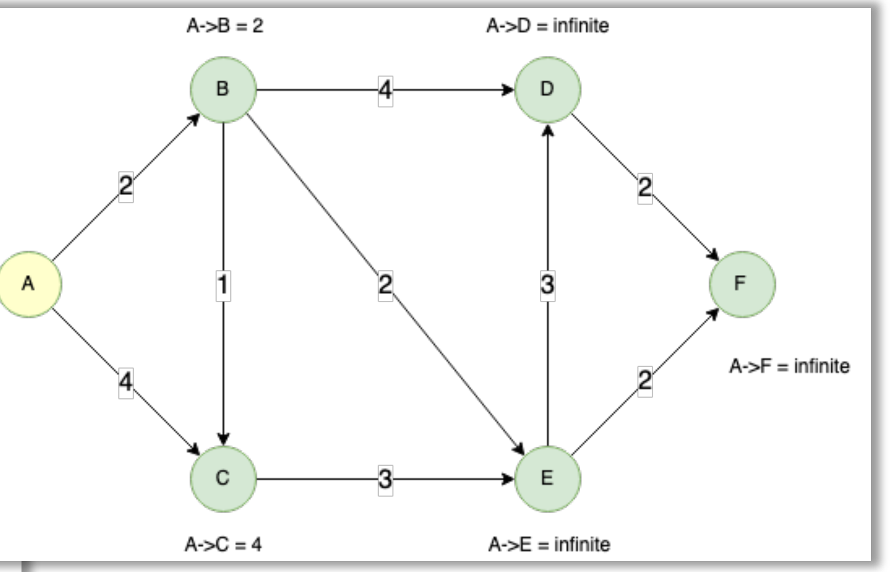


S = {}
Q = {a, b, c, d, e, f}
dist = {0, INF, INF, INF, INF, INF}

Dijkstra's Algorithm

```
Dijkstra(Graph, source):
  create vertex set S and Q
  for each vertex v in Graph:
     dist[v] ← INFINITY
     add v to Q
  dist[s] ← 0

  while Q is not empty:
     u ← vertex in Q with min dist[u]
     remove u from Q and add u to S
     for each neighbor v of u:
        alt ← dist[u] + length(u, v)
        if alt < dist[v]:
           dist[v] ← alt
```

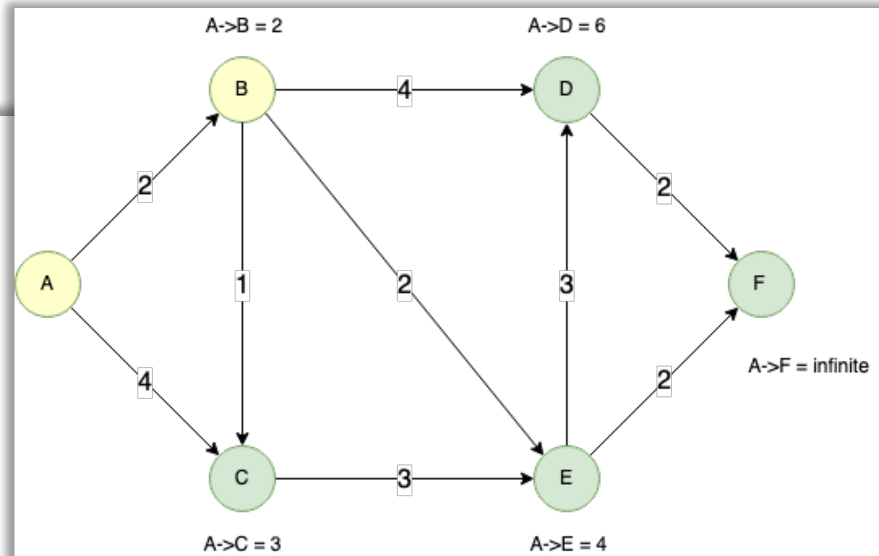


u = a
S = {a}
Q = {b, c, d, e, f}
dist = {0, 2, 4, INF, INF, INF}

Dijkstra’s Algorithm

```
Dijkstra(Graph, source):
  create vertex set S and Q
  for each vertex v in Graph:
     dist[v] ← INFINITY
     add v to Q
  dist[s] ← 0

  while Q is not empty:
     u ← vertex in Q with min dist[u]
     remove u from Q and add u to S
     for each neighbor v of u:
        alt ← dist[u] + length(u, v)
        if alt < dist[v]:
           dist[v] ← alt
```

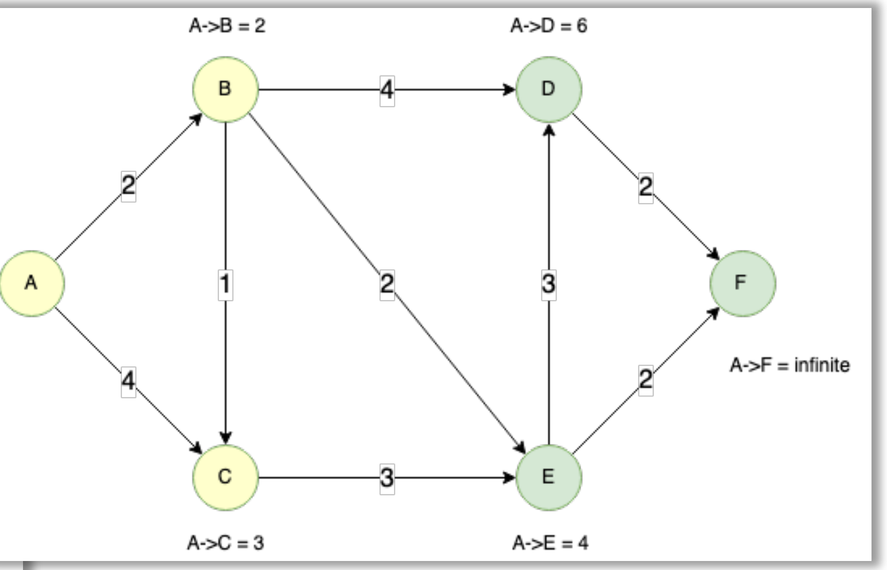


u = b
S = {a, b}
Q = {c, d, e, f}
dist = {0, 2, 3, 6, 4, INF}

Dijkstra’s Algorithm

```
Dijkstra(Graph, source):
  create vertex set S and Q
  for each vertex v in Graph:
     dist[v] ← INFINITY
     add v to Q
  dist[s] ← 0

  while Q is not empty:
     u ← vertex in Q with min dist[u]
     remove u from Q and add u to S
     for each neighbor v of u:
        alt ← dist[u] + length(u, v)
        if alt < dist[v]:
           dist[v] ← alt
```

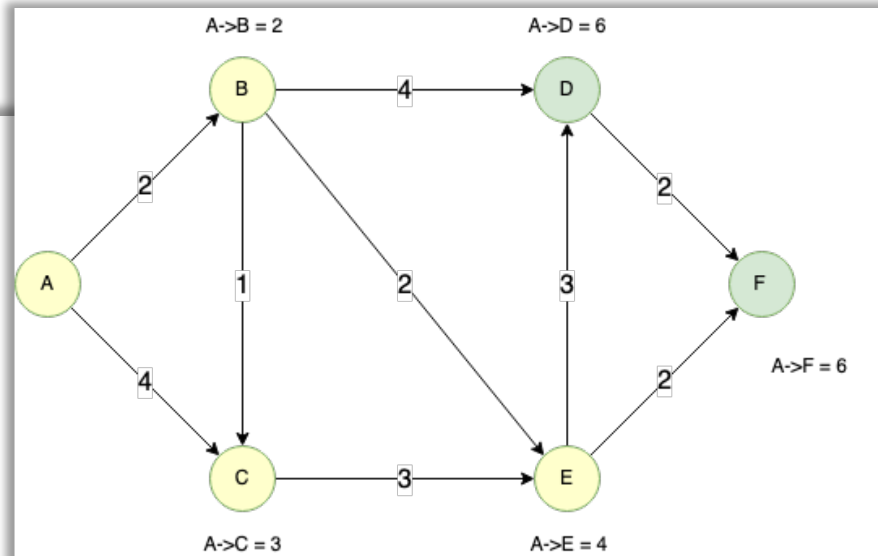


u = c
S = {a, b, c}
Q = {d, e, f}
dist = {0, 2, 3, 6, 4, INF}

Dijkstra’s Algorithm

```
Dijkstra(Graph, source):
  create vertex set S and Q
  for each vertex v in Graph:
     dist[v] ← INFINITY
     add v to Q
  dist[s] ← 0

  while Q is not empty:
     u ← vertex in Q with min dist[u]
     remove u from Q and add u to S
     for each neighbor v of u:
        alt ← dist[u] + length(u, v)
        if alt < dist[v]:
           dist[v] ← alt
```

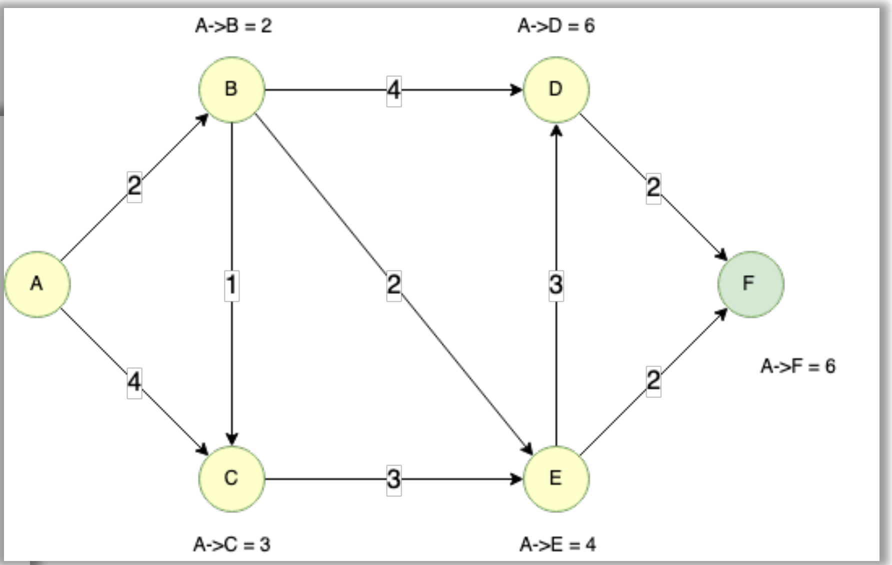


u = e
S = {a, b, c, e}
Q = {d, f}
dist = {0, 2, 3, 6, 4, 6}

Dijkstra’s Algorithm

```
Dijkstra(Graph, source):
  create vertex set S and Q
  for each vertex v in Graph:
     dist[v] ← INFINITY
     add v to Q
  dist[s] ← 0

  while Q is not empty:
     u ← vertex in Q with min dist[u]
     remove u from Q and add u to S
     for each neighbor v of u:
        alt ← dist[u] + length(u, v)
        if alt < dist[v]:
           dist[v] ← alt
```

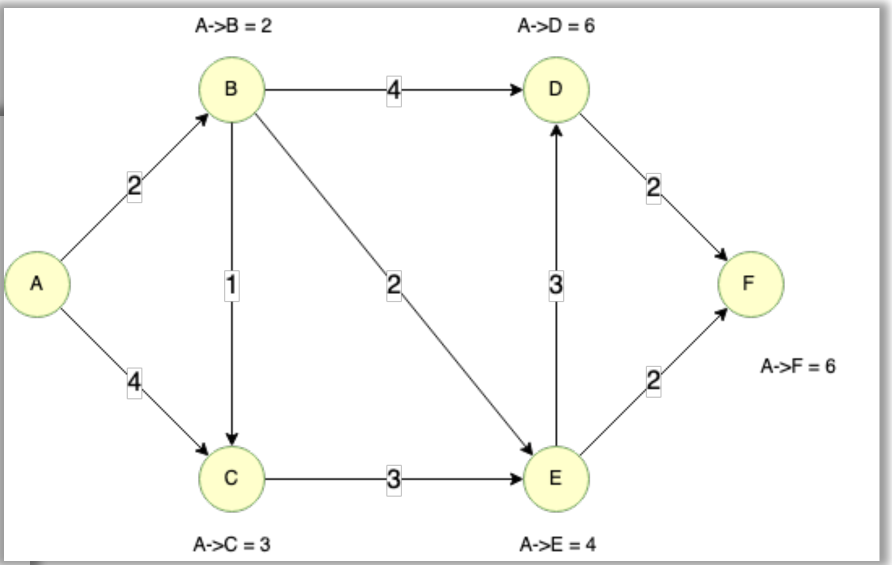


u = d
S = {a, b, c, e, d}
Q = {f}
dist = {0, 2, 3, 6, 4, 6}

Dijkstra's Algorithm

```
Dijkstra(Graph, source):
  create vertex set S and Q
  for each vertex v in Graph:
     dist[v] ← INFINITY
     add v to Q
  dist[s] ← 0

  while Q is not empty:
     u ← vertex in Q with min dist[u]
     remove u from Q and add u to S
     for each neighbor v of u:
        alt ← dist[u] + length(u, v)
        if alt < dist[v]:
           dist[v] ← alt
```

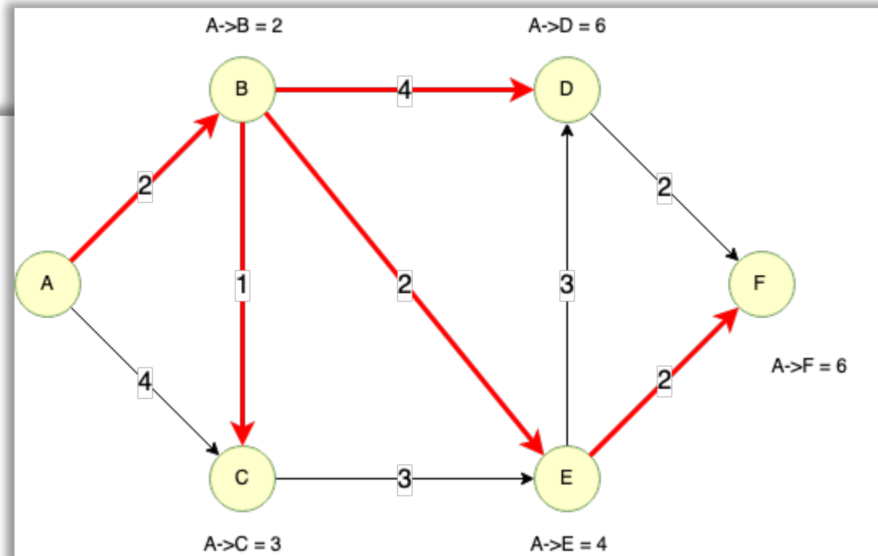


u = f
S = {a, b, c, e, d, f}
Q = { }
dist = {0, 2, 3, 6, 4, 6}

Dijkstra's Algorithm

```
Dijkstra(Graph, source):
  create vertex set S and Q
  for each vertex v in Graph:
     dist[v] ← INFINITY
     add v to Q
  dist[s] ← 0

  while Q is not empty:
     u ← vertex in Q with min dist[u]
     remove u from Q and add u to S
     for each neighbor v of u:
        alt ← dist[u] + length(u, v)
        if alt < dist[v]:
           dist[v] ← alt
```



This is the shortest path tree.
You can use a list to record the traverse order from the source a to any given target vertex

Correctness

- https://web.engr.oregonstate.edu/~glencora/wiki/uploads/dijkstra-proof.pdf

Time Complexity

If you store them in a sorted and balanced tree

- Initialization
  - O(V)
- while loop - O(V)
  - Find the minimum - O(lgV)
  - Access neighbors - O(V)
- Overall:
  - O(V^2)

```
Dijkstra(Graph, source):
  create vertex set S and Q
  for each vertex v in Graph:
     dist[v] ← INFINITY
     add v to Q
  dist[s] ← 0

  while Q is not empty:
     u ← vertex in Q with min dist[u]
     remove u from Q and add u to S
     for each neighbor v of u:
        alt ← dist[u] + length(u, v)
        if alt < dist[v]:
           dist[v] ← alt
```

Shortest Path Problem Further

- Single-source shortest-path problem in a more general settings: edge weight can be negative.

- You may still use Dijkstra’s Algorithm for the shortest path as long as there is not any negative cost circles.

Negative Cost Circles

- The graph cannot contain any negative cost circles
- If some path from s to t contains a negative cost cycle, there does not exist a shortest s-t path.