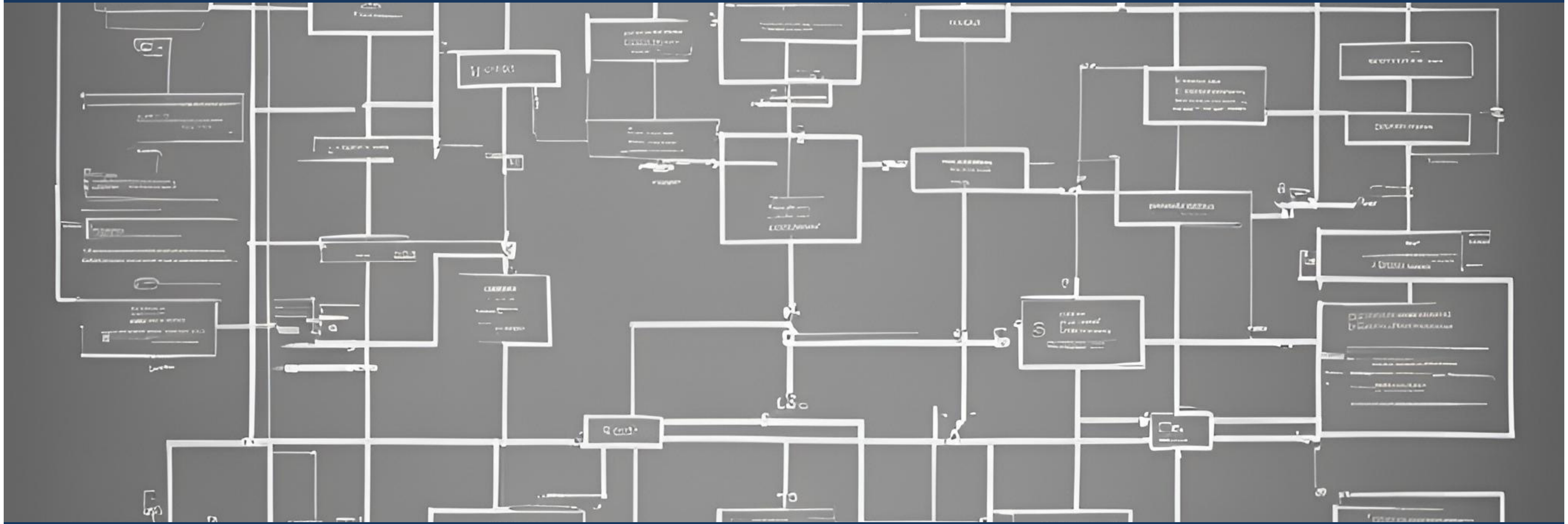
ITM 517 Algorithm

Ja-Hee Kim

Graph



Shortest path

Shortest path problem

- Finding the shortest possible route from Seoul to Pusan.
  - Shortest path

$$\delta(u, v) = \begin{cases} \min \sum_{k=1}^{k} w(v_{i-1}, v_i) & \text{if there is a path} \\ \infty & \text{no path} \end{cases}$$

  - Vertex: intersection
  - Edge: road segment between intersections
  - Weight: distance

Tonghua
通化市
ING
North Korea
Pyongyang
평양
Seoul
서울
Incheon
인천
South Korea
Daegu
대구
Busan
부산
Fukuoka
福岡

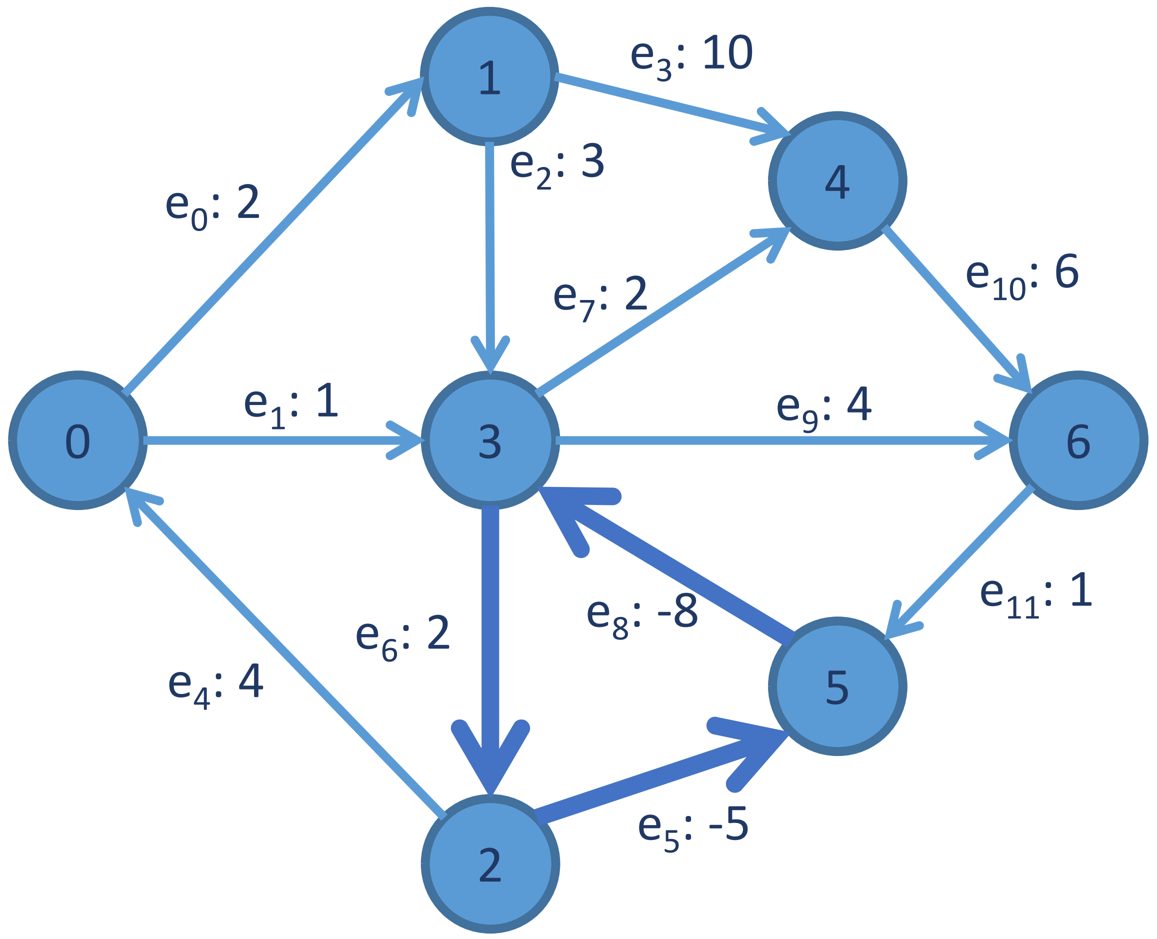
variation

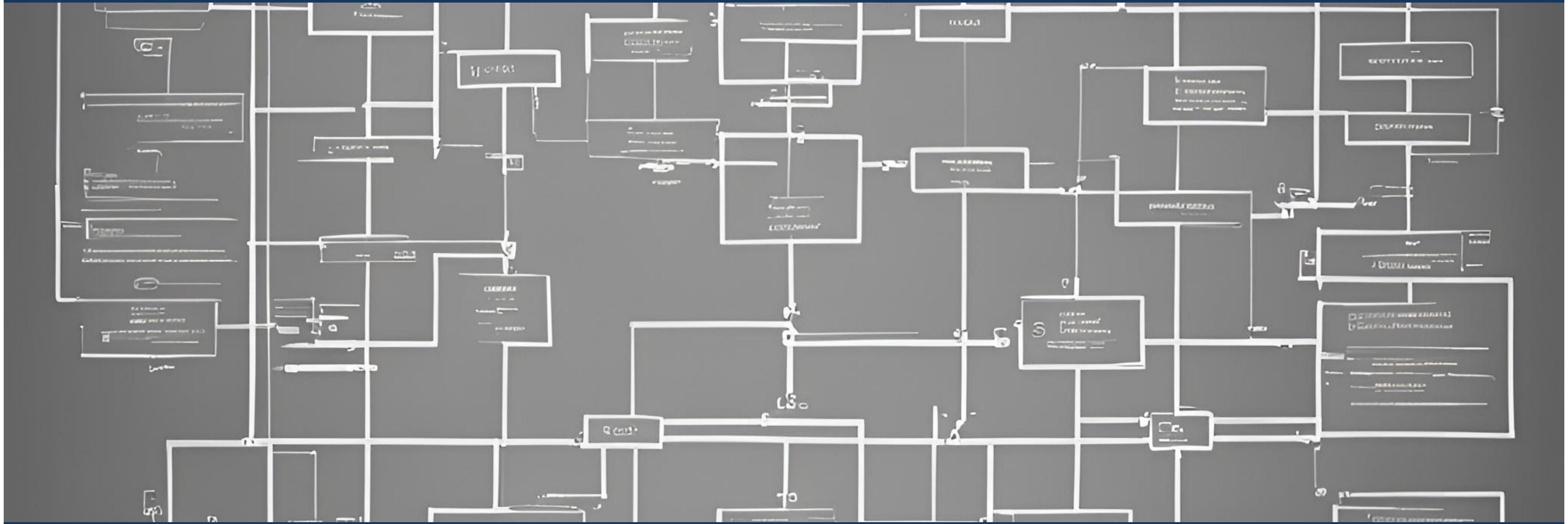
- Single source shortest path
    - No negative weight edges: Dijkstra’s algorithm
    - Negative weight edges: The Bellman-Ford algorithm
    - Directed acyclic graph
- All pairs shortest paths
    - Floyd-Warshall algorithm

- Single destination shortest path algorithm
    - Reverse of single source shortest path
- Single pair shortest path problem
    - A* search algorithm

Negative edge

- What happens if there is a negative cycle?





Dijkstra algorithm

Dijkstra algorithm

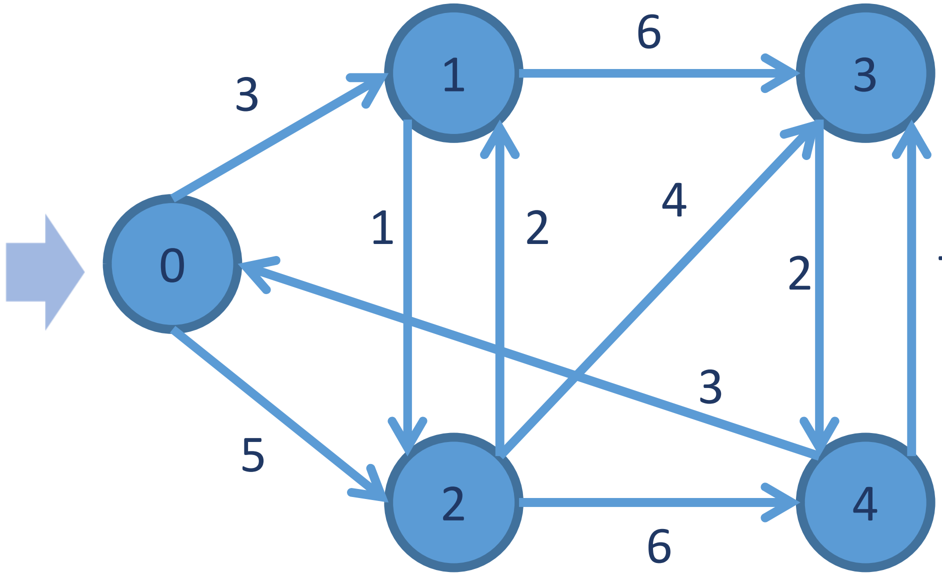
- Condition
  - Dijkstra algorithm works only for **connected graphs**.
  - Dijkstra algorithm works only for those graphs that do **not** contain any **negative weight edge**.
  - Dijkstra algorithm works for directed as well as undirected graphs.

Dijkstra algorithm

```
• dist[S] ← 0 // The distance to source vertex is set to 0
• Π[S] ← NIL // The predecessor of source vertex is set as NIL
• for all v ∈ V - {S} // For all other vertices
          do dist[v] ← ∞ // All other distances are set to ∞
          Π[v] ← NIL // The predecessor of all other vertices is set as NIL
• S ← ∅ // The set of vertices that have been visited 'S' is initially empty
• Q ← V // The queue 'Q' initially contains all the vertices
• while Q ≠ ∅ // While loop executes till the queue is not empty
      do u ← mindistance (Q, dist) // A vertex from Q with the least distance is
      selected
      S ← S ∪ {u} // Vertex 'u' is added to 'S' list of vertices that have been visited
      for all v ∈ neighbors[u] // For all the neighboring vertices of vertex 'u'
          do if dist[v] > dist[u] + w(u,v) // if any new shortest path is discovered
              then dist[v] ← dist[u] + w(u,v) // The new value of the shortest path is selected
• return dist
```

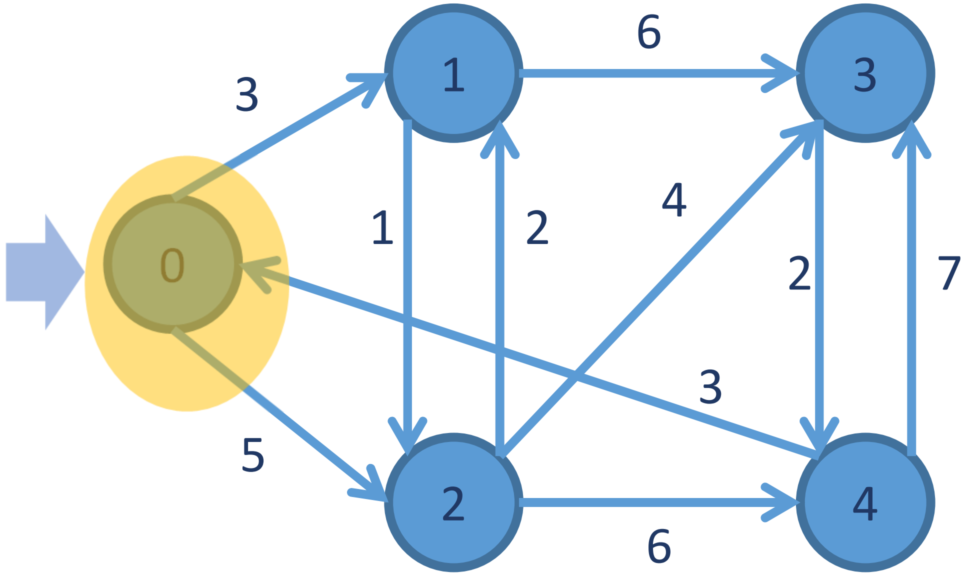
Dijkstra algorithm

```
• dist[S] ← 0
• Π[S] ← NIL
• for all v ∈ V - {S} // For all other vertices
          do dist[v] ← ∞
          Π[v] ← NIL
• S ← Ø
• Q ← V
```



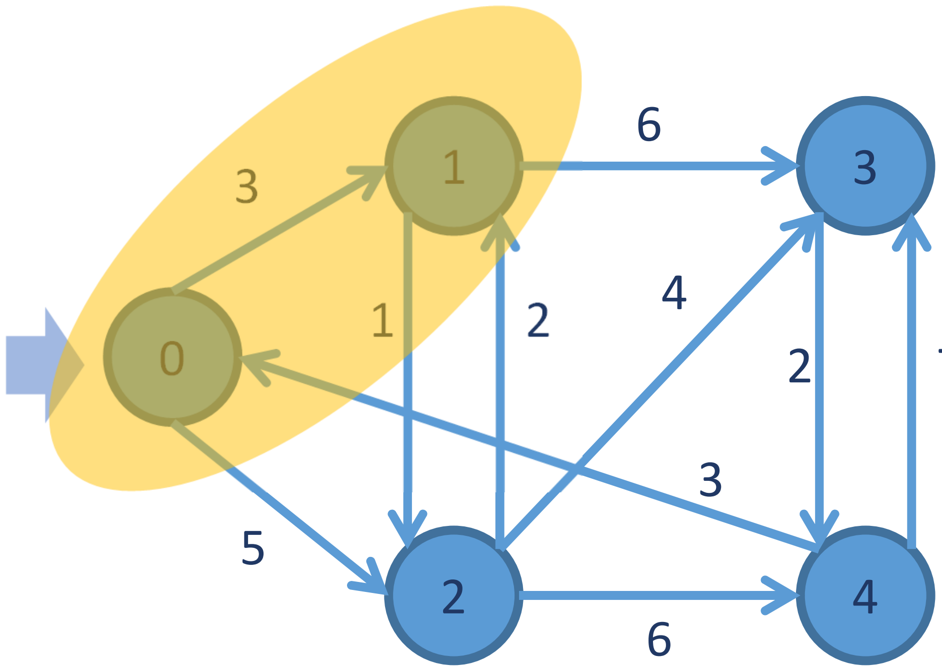
Dijkstra algorithm

```
• while Q ≠ Ø
    do u ← minDistance (Q, dist)
    S ← S ∪ {u}
    for all v ∈ neighbors[u]
        do if dist[v] > dist[u] + w(u,v)
            then dist[v] ← dist[u] + w(u,v)
```



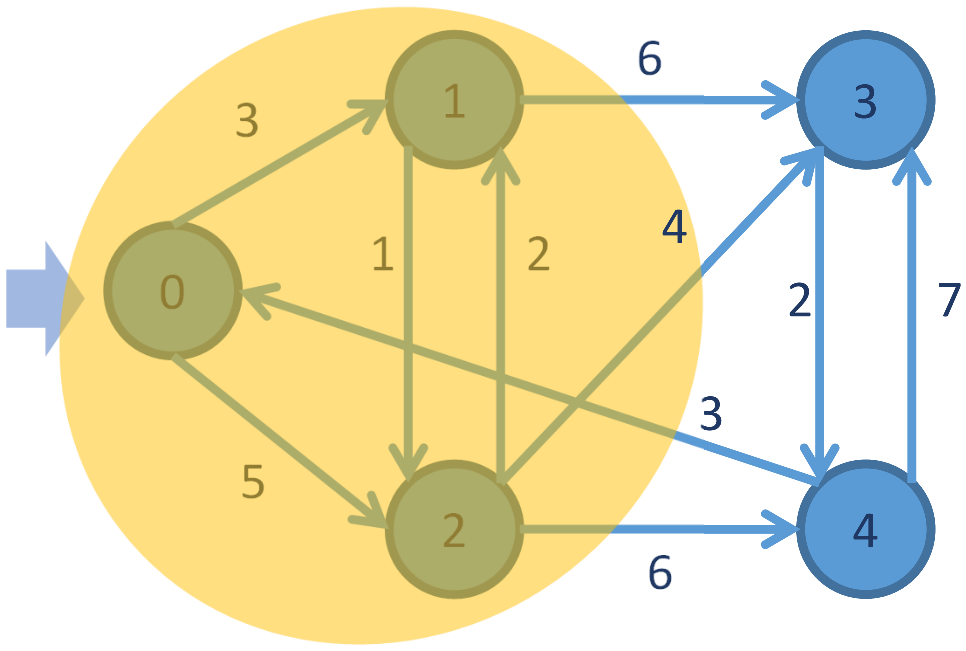
Dijkstra algorithm

```
• while Q ≠ Ø
    do u ← minDistance (Q, dist)
    S ← S U {u}
    for all v ∈ neighbors[u]
        do if dist[v] > dist[u] + w(u,v)
            then dist[v] ← dist[u] + w(u,v)
```



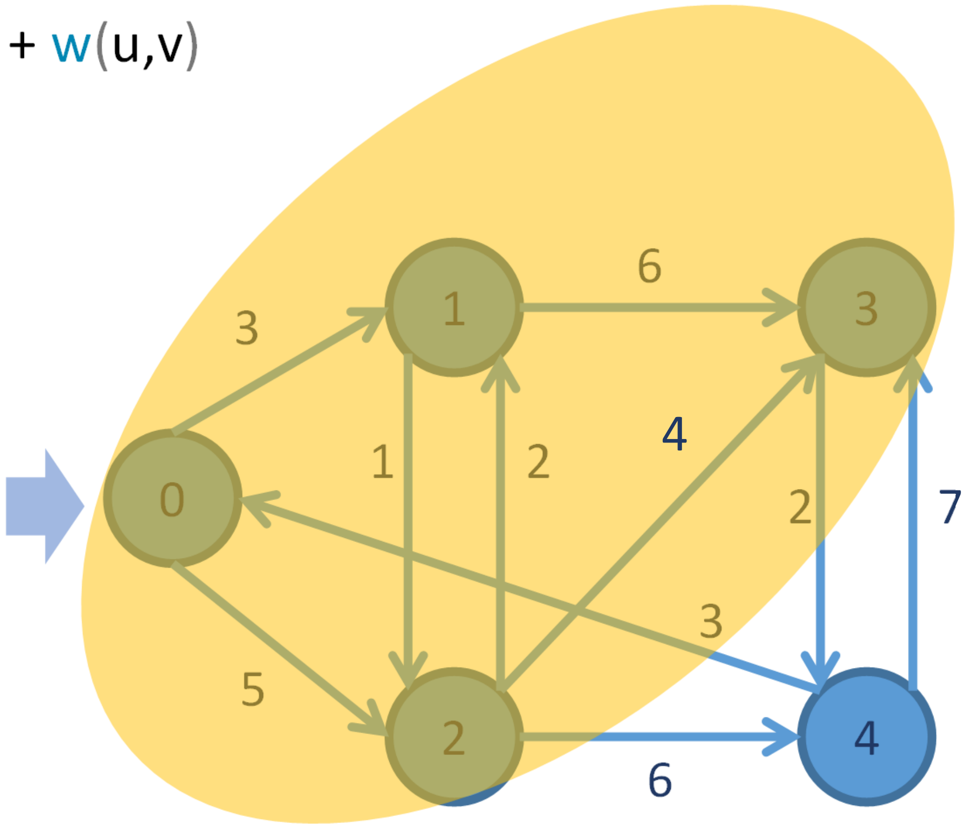
Dijkstra algorithm

```
• while Q ≠ ∅
    do u ← minDistance (Q, dist)
    S ← S ∪ {u}
    for all v ∈ neighbors[u]
        do if dist[v] > dist[u] + w(u,v)
            then dist[v] ← dist[u] + w(u,v)
```



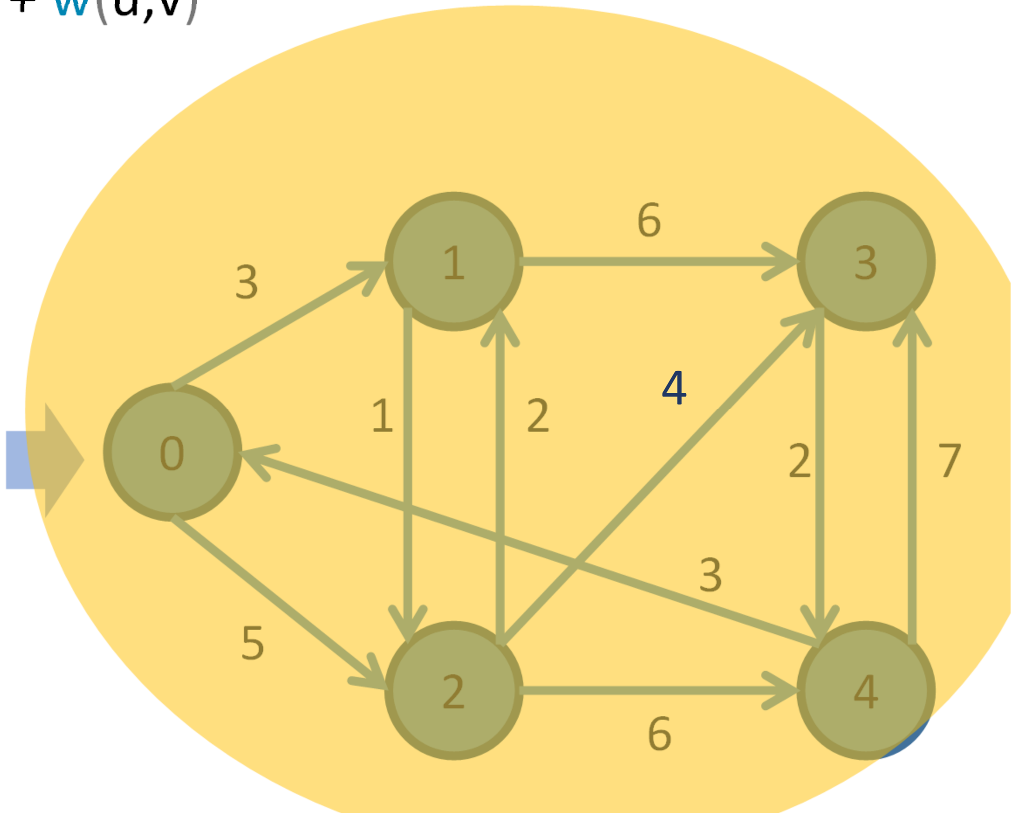
Dijkstra algorithm

```
• while Q ≠ Ø
    do u ← minDistance (Q, dist)
    S ← S ∪ {u}
    for all v ∈ neighbors[u]
        do if dist[v] > dist[u] + w(u,v)
            then dist[v] ← dist[u] + w(u,v)
```



Dijkstra algorithm

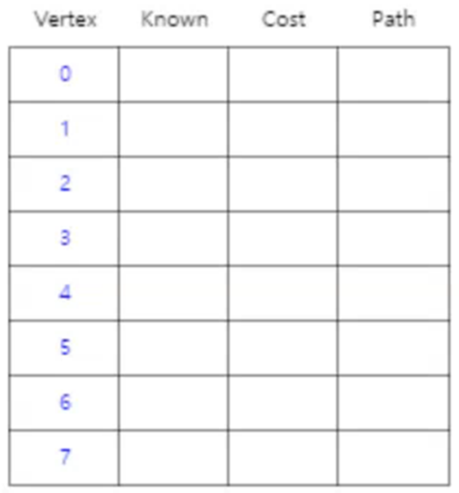
```
• while Q ≠ Ø
    do u ← minDistance (Q, dist)
    S ← S U {u}
    for all v ∈ neighbors[u]
        do if dist[v] > dist[u] + w(u,v
            then dist[v] ← dist[u] + w(u,v)
Return dist
```



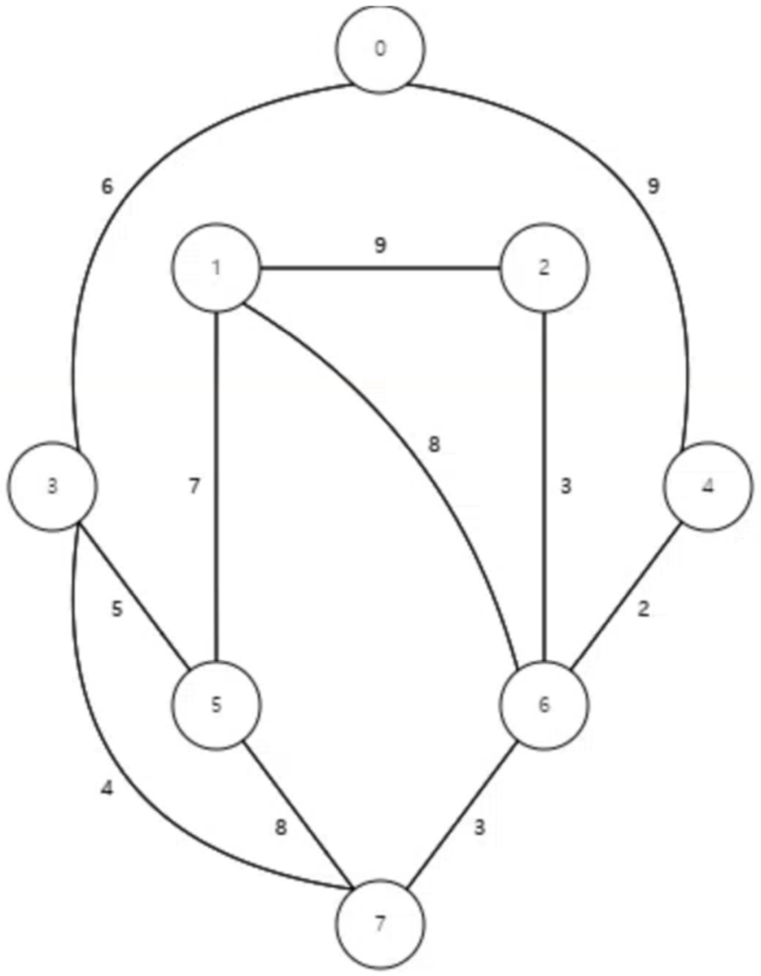
Dijkstra algorithm

- Visualization

  https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html



Vertex	Known	Cost	Path
0			
1			
2			
3			
4			
5			
6			
7			



Time complexity of Dijkstra algorithm

- Time taken for selecting i with the smallest dist is O(V).
- For each neighbor of i, time taken for updating dist[j] is O(1) and there will be maximum V neighbors.
- Time taken for each iteration of the loop is O(V) and one vertex is deleted from Q.
- Thus, total time complexity becomes $O(V^2)$.
- With adjacency list representation, all vertices of the graph can be traversed using BFS in O(V+E) time.
- In min heap, operations like extract-min and decrease-key value takes O(logV) time.

$O(E+V) \times O(\log V) \rightarrow O(E\log V)$

It can be reduced to O(E+VlogV) using Fibonacci heap.

Thanks