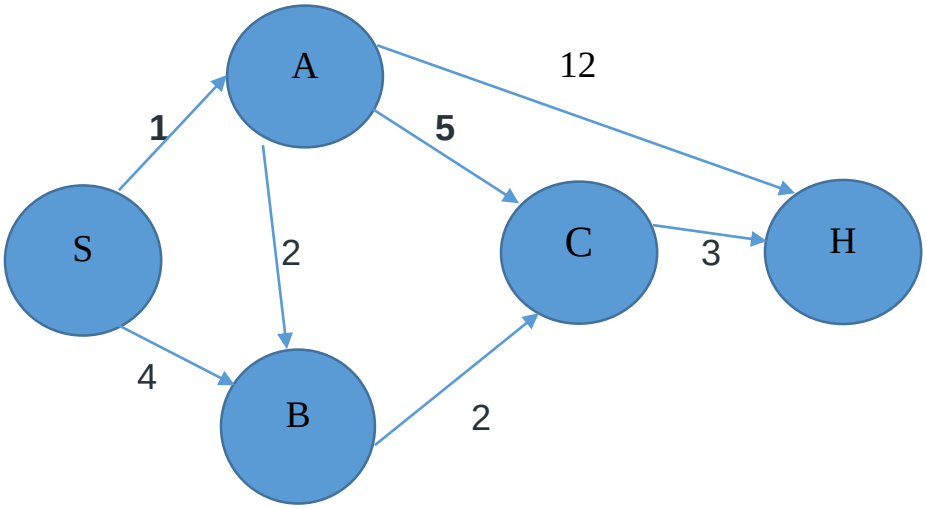
Lecture Six

Graph

3. Dijkstra's algorithm

Dijkstra's Algorithm is a shortest path algorithm used to find the minimum distance from a source node to all other nodes in a weighted graph. It works efficiently with non-negative weights and is commonly used in routing and network optimization problems.

- **Example:** Find shortest path between S and H

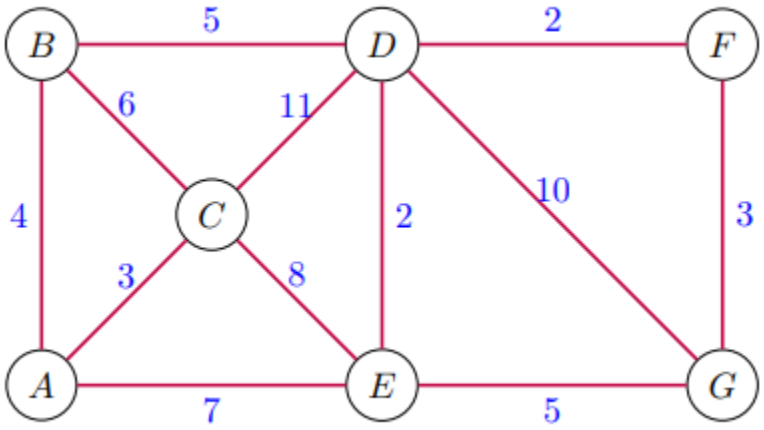


	S	A	C	B	H
S	--	1s	∞	4s	∞
A	--	--	6a	3a	13a
B	--	--	5b	3a	13a
C	--	--	5b	3a	8c
H	--	--	--	--	8c

S → A → B → C → H

- **Example:**

Consider the graph below for instance. Suppose we want to compute the cheapest path from s = A to t = F.



The table below keeps track of the distances computed at every iteration from the source A to every vertex in the graph. Read the table as follows: In the first iteration, the distance from the source to itself is 0, and ∞ to any other vertex. At every iteration, choose the cheapest available vertex and try to build the next cheapest path from said vertex. Repeat the process until all vertices have been visited.

	A	B	C	D	E	F	G	S_i
$i = 0$	0	∞	∞	∞	∞	∞	∞	$\emptyset$
$i = 1$: A	0	4	3	∞	7	∞	∞	$\emptyset$
$i = 2$: C	0	4	3	14	7	∞	∞	$\{AC\}$
$i = 3$: B	0	4	3	9	7	∞	∞	$\{AC, AB\}$
$i = 4$: E	0	4	3	9	7	∞	12	$\{AC, AB, AE\}$
$i = 5$: D	0	4	3	9	7	11	12	$\{AC, AB, AE, BD\}$
$i = 6$: F	0	4	3	9	7	11	12	$\{AC, AB, AE, BD, DF\}$
$i = 7$: G	0	4	3	9	7	11	12	$\{AC, AB, AE, BD, DF, EG\}$

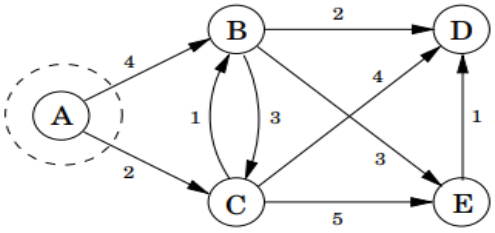
Notice form the table above that at every iteration i, the set Si is the set of edges that could potentially lead to a shortest path. If e = (u, v) is added to Si , with u ∈ Si , v /∈ Si , then the current value of d[v] is a shortest sv path. Formally the algorithm is as follows:

Algorithm 1 Dijkstra's Shortest Path Algorithm

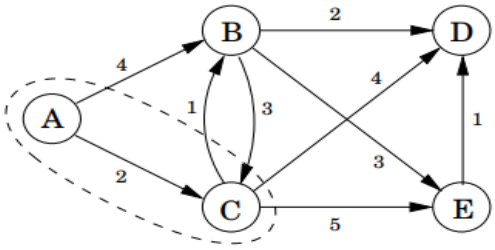
Input: An edge weighted connected graph $G(V, E, w)$ where $w : E \to \mathbb{R}^+$ and two vertices s, t.
Output: A path from s to t with minimum total cost (shortest path)

```
1: S = ∅.
2: Initialize empty priority queue.
3: foreach v ∈ V do
4:     p[v] = NIL                          ▷ predecessor of v n shortest s, v path so far
5:     d[v] = ∞                            ▷ priotity of v = min distance s, v so far.
6:     enqueue(v)                          ▷ With priority d[v] = ∞.
7: end for
8: d[s] = 0
9: Update queue order of s
10: while queue is not empty do            ▷ Main Loop
11:     v = dequeue element with min priority d[]
12:     if p[v]! = NIL then
13:         S = S ∪ {(p[v], v)}
14:     end if
15:     foreach edge (u, v) do
16:         if u is in the queue and d[v] + w(v, u) < d[u] then
17:             p[u] = v
18:             d[u] = d[v] + w(v, u)
19:             Update queue order of u
20:         end if
21:     end for
22: end while
23: return S
```

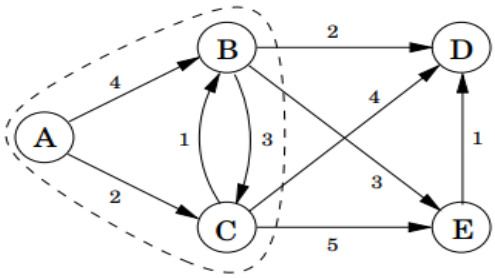
- **Example:** Find shortest path between A and E



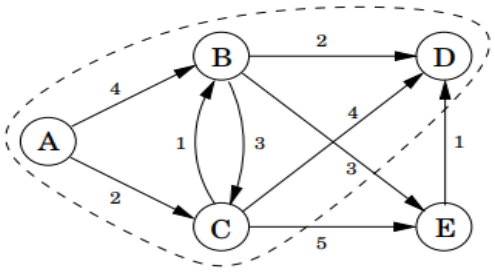
A: 0	D: ∞
B: 4	E: ∞
C: 2	



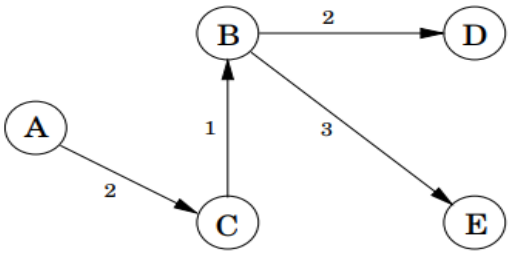
A: 0	D: 6
B: 3	E: 7
C: 2	



A: 0	D: 5
B: 3	E: 6
C: 2	



A: 0	D: 5
B: 3	E: 6
C: 2	



➢ Dijkstra's Algorithm vs BFS vs DFS

Feature	Dijkstra's Algorithm	BFS (Breadth-First Search)	DFS (Depth-First Search)
Purpose	Finds the **shortest path** in a **weighted graph**.	Finds the **shortest path** in an **unweighted graph**.	Explores **all possible paths** in a **graph or tree**.
Graph Type	Works with **weighted graphs** (positive weights only).	Works with **unweighted** graphs.	Works with both **weighted & unweighted** graphs.
Algorithm Type	**Greedy Algorithm** (Selects the shortest path first).	**Graph Traversal Algorithm** (Explores level by level).	**Graph Traversal Algorithm** (Explores depth-first).
Data Structure	Uses a **Priority Queue (Min-Heap)**.	Uses a **Queue (FIFO)**.	Uses a **Stack (LIFO)** (can be implemented using recursion).
Shortest Path Guarantee	✓ Yes (Only for non-negative weights).	✓ Yes (For unweighted graphs).	✗ No (Not guaranteed to find the shortest path).
Time Complexity	**O((V + E) log V)** (with a priority queue).	**O(V + E)** (with a queue).	**O(V + E)** (in general).
Space Complexity	**O(V)** (stores distances).	**O(V)** (stores visited nodes).	**O(V)** (stores visited nodes).
Best Use Case	**Shortest paths in weighted graphs** (e.g., **Google Maps, GPS, AI pathfinding**).	**Shortest paths in unweighted graphs** (e.g., **Friend suggestions in social networks**).	**Exploring all possible paths** (e.g., **Solving mazes, cycle detection, backtracking**).