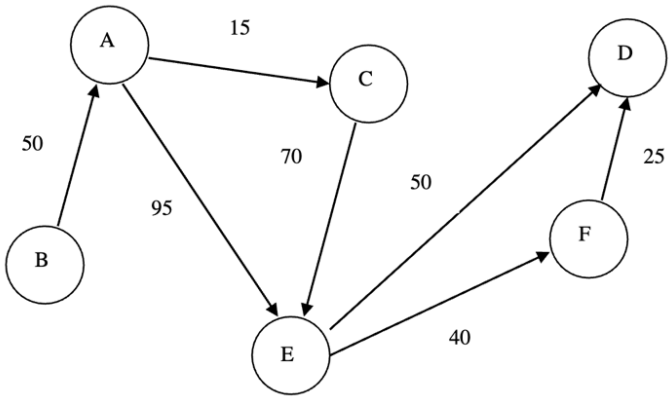
Assignment 12



Assume that the values on each edge of the graph represent the cost of getting from one vertex to another. Show how to determine the cost of the cheapest route from 'B' to 'D'.

To determine the cheapest route from vertex 'B' to 'D', we can use Dijkstra's algorithm.

Steps:

1. Initialize: Set the initial/current vertex cost to 0 and all others to ∞.
2. Update Neighbors: Check all unvisited neighbors of the current vertex and update their costs only if the current route is cheapest.
3. Select Current: Choose the unvisited vertex with the lowest cost as the next "current vertex".
4. Loop: Repeat the update and select steps until all nodes have been visited.
5. End: All vertices have been visited and the shortest path to each has been determined.

Start B (0)		A, C, E, D, F: ∞
B -> A (50)		
Move to A (50)		C, E, D, F: ∞
A -> C (50 + 15 = 65)	A -> E (50 + 95 = 145)	
Move to C (65)		D, F: ∞
C -> E (65 + 70 = 135)	135 < 145 - UPDATE	
Move to E (135)		D, F: ∞
E -> F (135 + 40 = 175)	E -> D (135 + 50 = 185)	
Move to F (175)		
F -> D (175 + 25 = 200)	200 > 185 - NO UPDATE	
Move to D (185)		

B -> A -> C -> E -> D: 185 is the cheapest route

Note: Parent vertices are used (parent pointers/predecessor arrays/dictionaries) to back track cheapest route. This is very important when updates or non-updates are made during the traversal of neighbors.

Show how the vertices of the graph would be traversed under both "depth-first" and "breadth-first" traversal algorithms.

Depth-First Search (DFS)

Note that in a DFS traversal, the actual path can vary depending on the order in which neighbors are chosen.

Here are a few traversal paths that DFS could use:

A	B	C	D	E	F
0	0	0	0	0	0
0	1	0	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	0	1	1
1	1	1	1	1	1

Start at B: From B, one vertex – A.
Move to A: From A, two vertex - C or E.
Move to C: From C, one vertex – E.
Move to E: From E, two vertex - D or F.
Move to F: From F, one vertex – D.
Move to D: All vertices visited.

Traversal: B, A, C, E, F, D

A	B	C	D	E	F
0	0	0	0	0	0
0	1	0	0	0	0
1	1	0	0	0	0
1	1	0	0	1	0
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Start at B: From B, one vertex – A.
Move to A: From A, two vertex - C or E.
Move to E: From E, two vertex – D or F.
Move to F: From F, one vertex – D.
Move to D: From D, no vertex but C is still unvisited.
Move to C with recursion: D -> F -> E -> A -> C

Traversal: B, A, E, F, D, C

A	B	C	D	E	F
0	0	0	0	0	0
0	1	0	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	1	0
1	1	1	1	1	1

Start at B: From B, one vertex – A.
Move to A: From A, two vertex - C or E.
Move to C: From C, one vertex – E.
Move to E: From E, two vertex - D or F.
Move to D: From D, no vertex but F is still unvisited.
Move to F with recursion: D -> E -> F

Traversal: B, A, C, E, D, F

Breadth-First Traversal

Note that in a BFS traversal, the actual path can vary depending on the order in which neighbors are enqueued, typically with an adjacency list.

Here are a few traversal paths that BFS could use:

Start at B:
Enqueue B.

Queue: [B]	Dequeue B and enqueue A.
Queue: [A]	Dequeue A and enqueue A's neighbors - C and E.
Queue: [C, E]	Dequeue C - no new vertices to enqueue (E is already in the queue).
Queue: [E]	Dequeue E and enqueue E's neighbors - D and F.
Queue: [D, F]	Dequeue D - no new vertices to enqueue (F is already in the queue).
Queue: [F]	Dequeue F - no new vertices to enqueue.
Queue: []	

Traversal: B, A, C, E, D, F

Start at B:
Enqueue B.

Queue: [B]	Dequeue B and enqueue A.
Queue: [A]	Dequeue A and enqueue A's neighbors - E and C.
Queue: [E, C]	Dequeue E and enqueue E's neighbors - D and F.
Queue: [C, D, F]	Dequeue C - no new vertices to enqueue (D and F are already in the queue).
Queue: [D, F]	Dequeue D - no new vertices to enqueue (F is already in the queue).
Queue: [F]	Dequeue F - no new vertices to enqueue.
Queue: []	

Traversal: B, A, E, C, D, F

Start at B:
Enqueue B.

Queue: [B]	Dequeue B and enqueue A.
Queue: [A]	Dequeue A and enqueue A's neighbors - E and C.
Queue: [E, C]	Dequeue E and enqueue E's neighbors - F and D.
Queue: [C, F, D]	Dequeue C - no new vertices to enqueue (F and D are already in the queue).
Queue: [F, D]	Dequeue F - no new vertices to enqueue (D is already in the queue).
Queue: [D]	Dequeue D - no new vertices to enqueue.
Queue: []	

Traversal: B, A, E, C, F, D

The BFS traversal path is more straightforward because it simply visits all neighbors of the current vertex before moving on to the next level of neighbors.