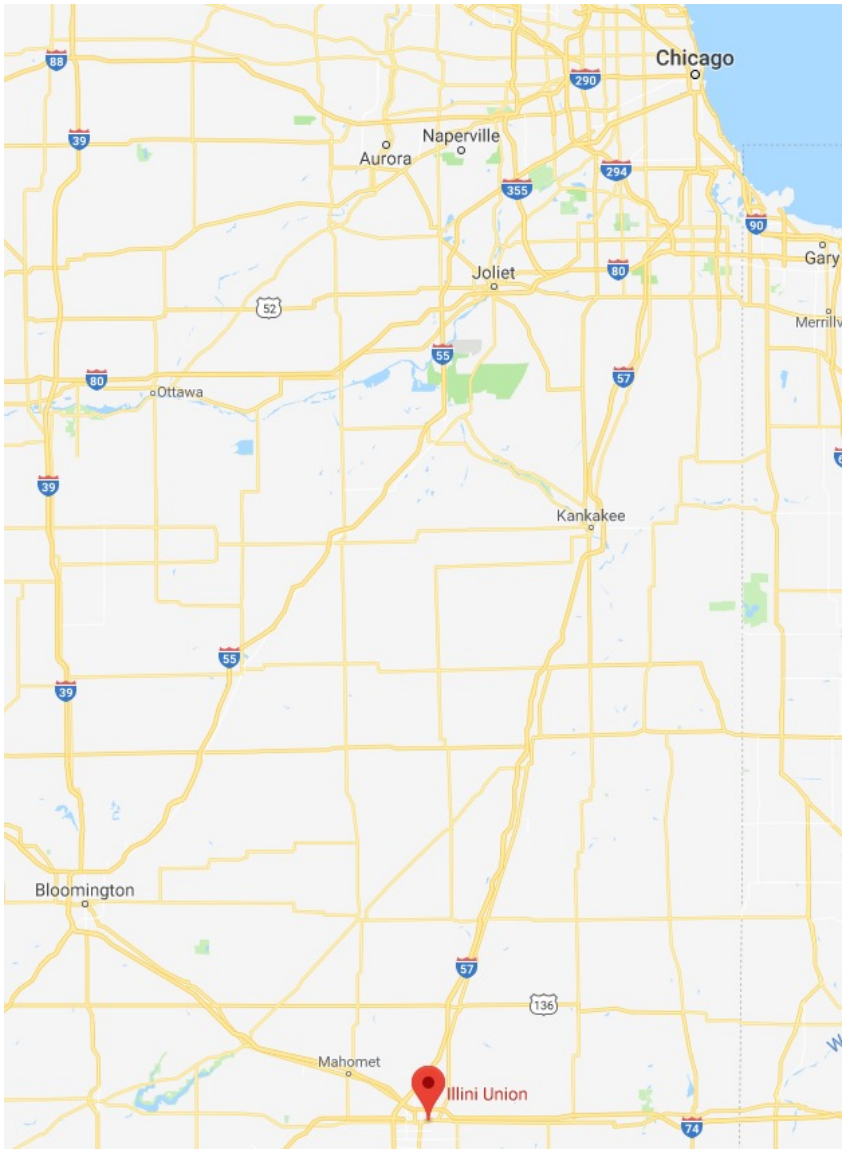
CS 225

Data Structures

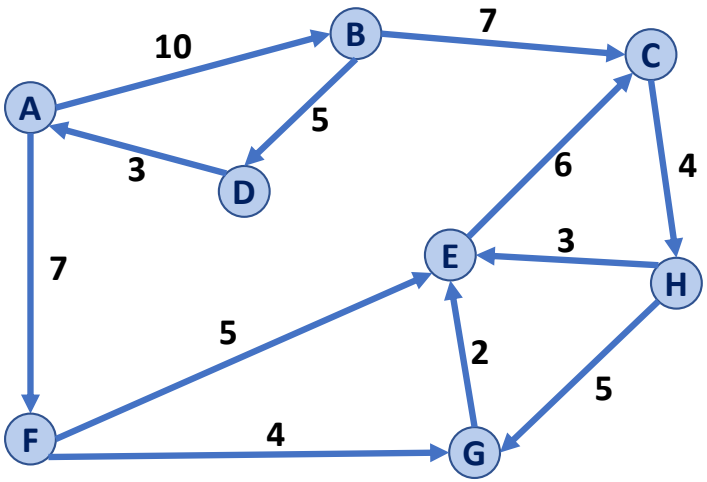
Nov 2 – Dijkstra's Algorithm Analysis

G Carl Evans

Shortest Path

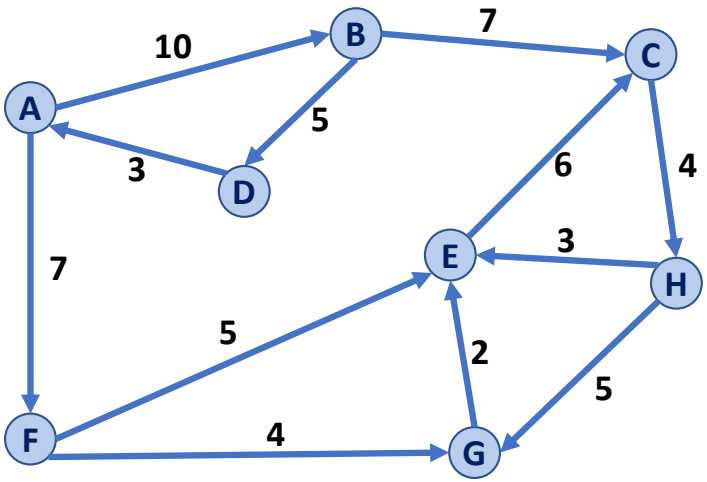


Dijkstra's Algorithm (SSSP)



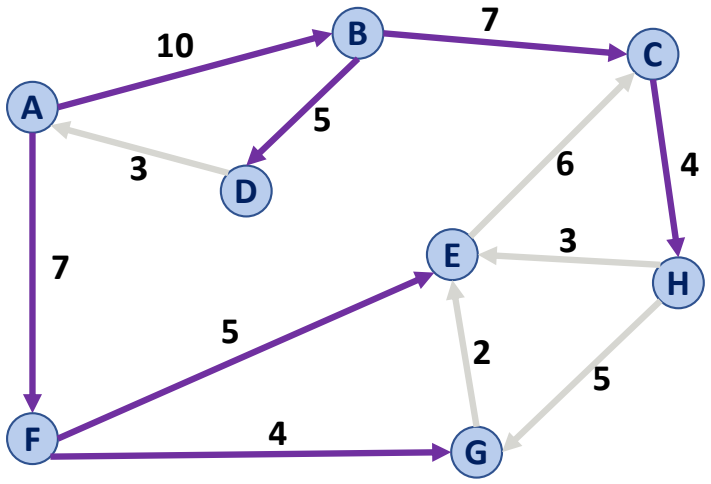

```
    PrimMST(G, s):
 6    foreach (Vertex v : G):
 7      d[v] = +inf
 8      p[v] = NULL
 9    d[s] = 0
10
11    PriorityQueue Q // min distance, defined by d[v]
12    Q.buildHeap(G.vertices())
13    Graph T          // "labeled set"
14
15    repeat n times:
16      Vertex u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19        if cost(v, m) < d[v]:
20          d[v] = cost(v, m)
21          p[v] = m
```

Dijkstra's Algorithm (SSSP)




```
    DijkstraSSSP(G, s):
 6    foreach (Vertex v : G):
 7      d[v] = +inf
 8      p[v] = NULL
 9    d[s] = 0
10
11    PriorityQueue Q // min distance, defined by d[v]
12    Q.buildHeap(G.vertices())
13    Graph T          // "labeled set"
14
15    repeat n times:
16      Vertex u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19        if _________________ < d[v]:
20          d[v] = ____________________
21          p[v] = m
```

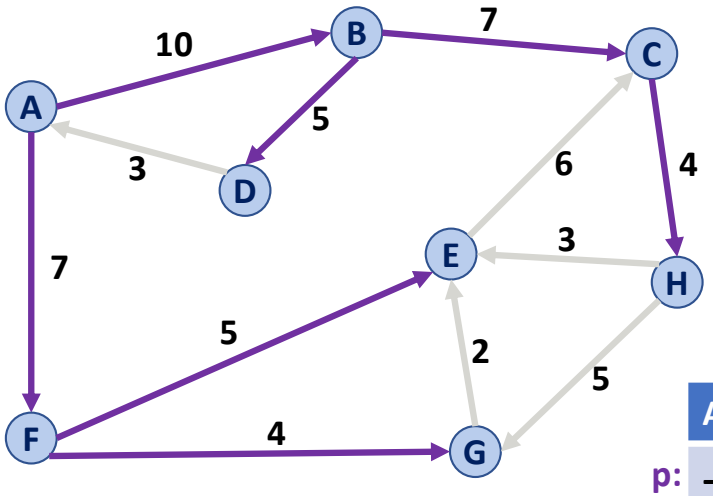
Dijkstra's Algorithm (SSSP)




```
    DijkstraSSSP(G, s):
 6    foreach (Vertex v : G):
 7      d[v] = +inf
 8      p[v] = NULL
 9    d[s] = 0
10
11    PriorityQueue Q // min distance, defined by d[v]
12    Q.buildHeap(G.vertices())
13    Graph T          // "labeled set"
14
15    repeat n times:
16      Vertex u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19        if cost(u, v) + d[u] < d[v]:
20          d[v] = cost(u, v) + d[u]
21          p[v] = m
```

Dijkstra's Algorithm (SSSP)

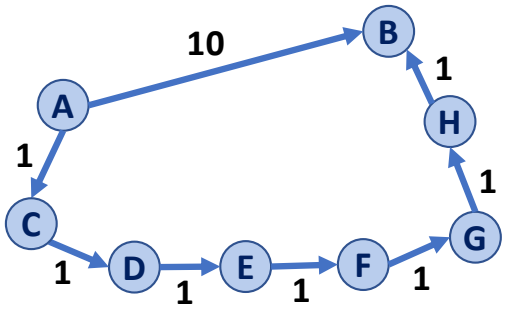
Dijkstra gives us the shortest path from our path (single source) to **every** connected vertex!



	A	B	C	D	E	F	G	H
p:	--	A	B	B	F	A	F	C
d:	0	10	17	15	12	7	11	21

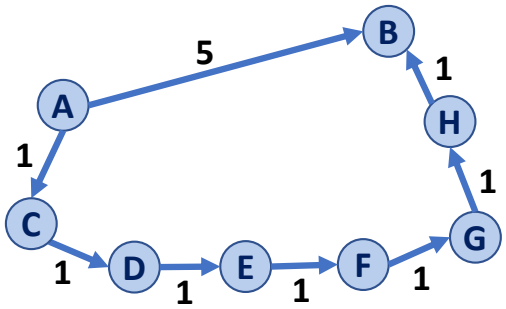
Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle a single heavy-weight path vs. many light-weight paths?



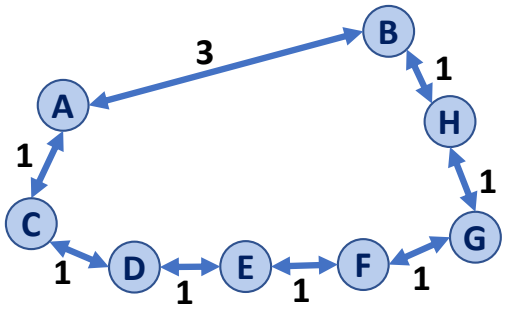
Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle a single heavy-weight path vs. many light-weight paths?



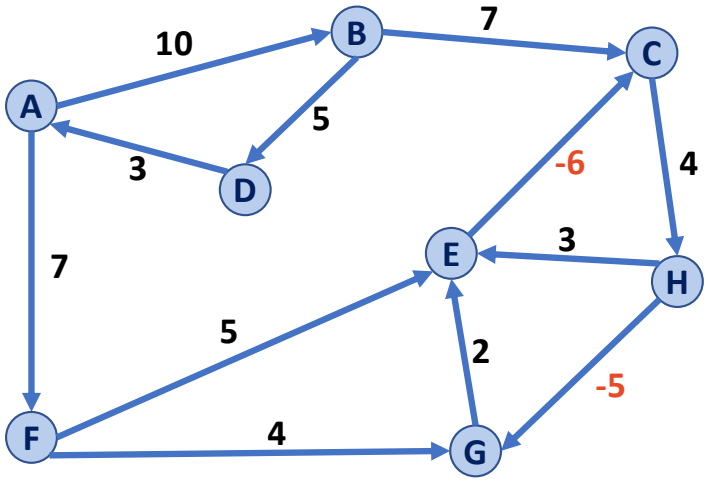
Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle undirected graphs?



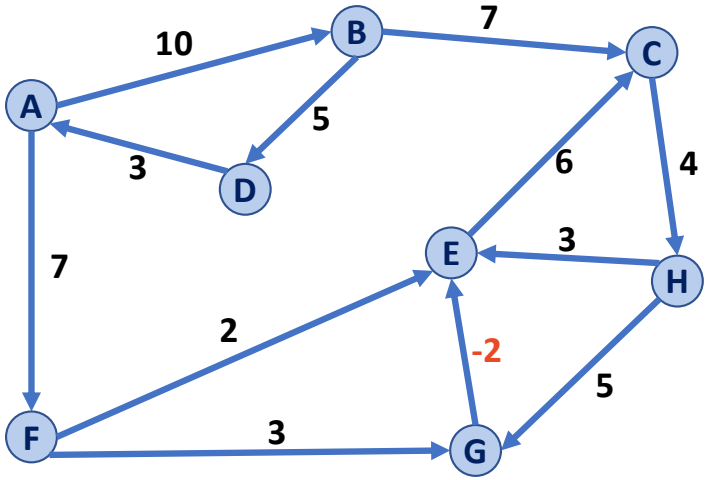
Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle negative weight cycles?



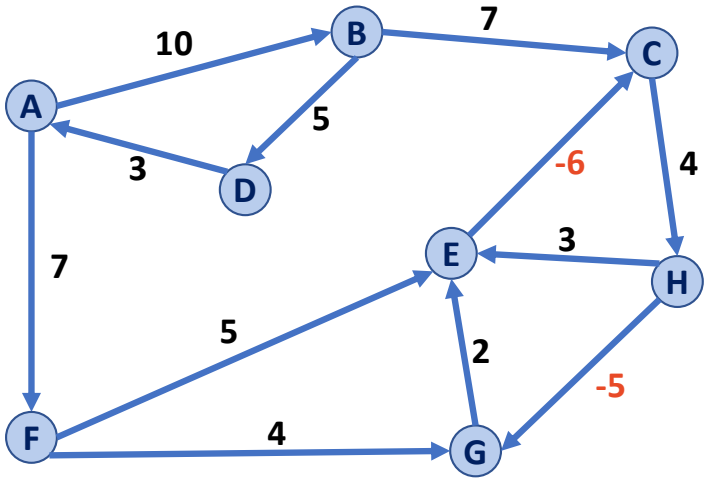
Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle negative weight edges, without a negative weight cycle?



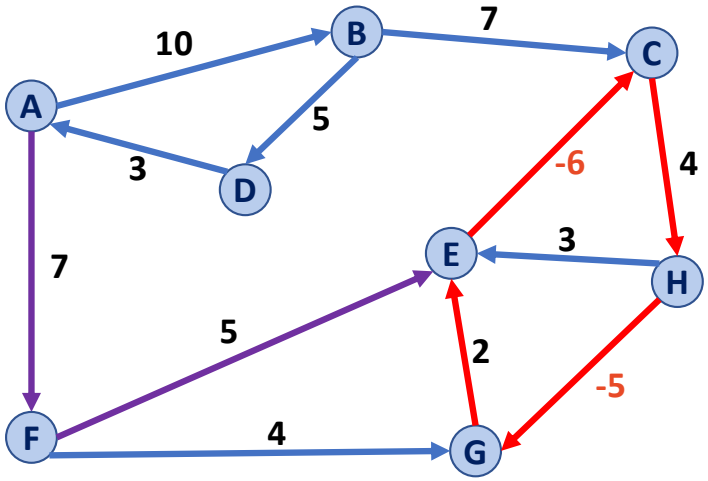
Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle negative weight cycles?



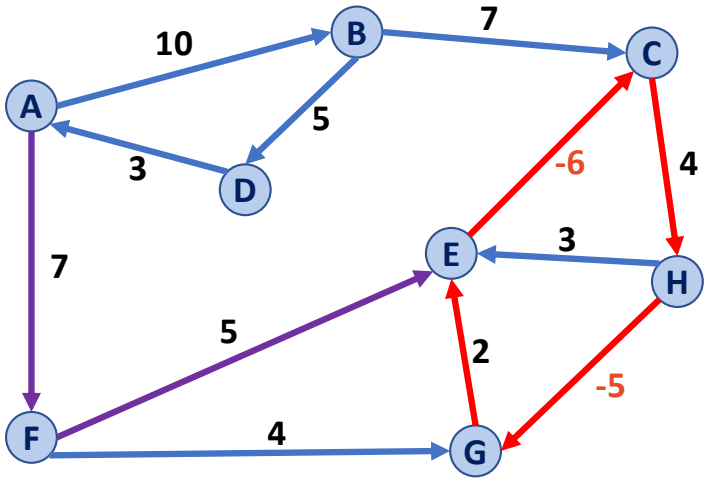
Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle negative weight cycles?



Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle negative weight cycles?

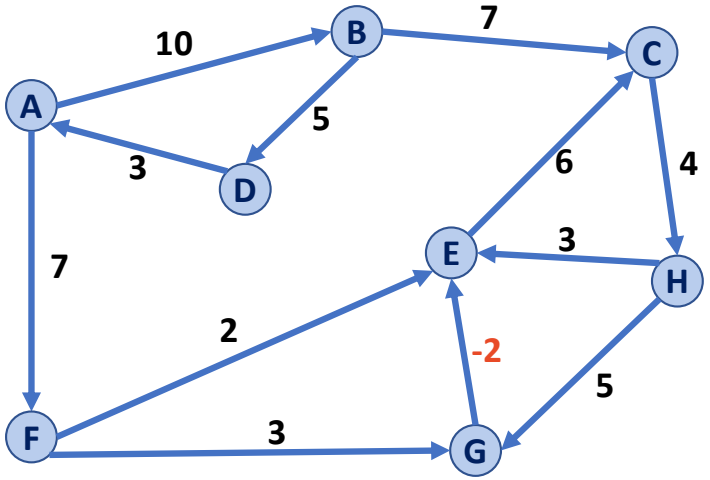


Shortest Path (A ➔ E): A → F → E → (C → H → G → E)*

Length: 12 Length: -5 (repeatable)

Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle negative weight edges, without a negative weight cycle?



Dijkstra's Algorithm (SSSP)

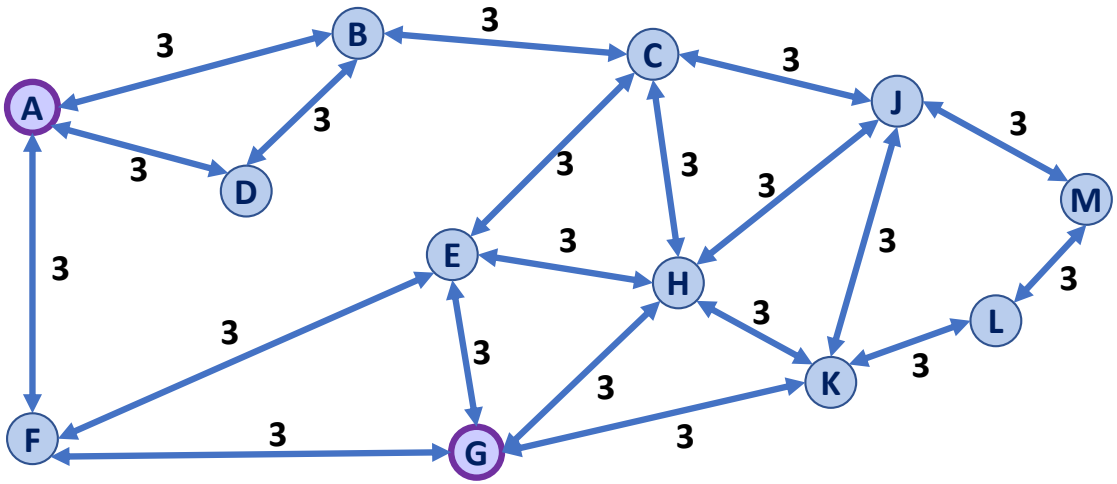
What is Dijkstra's running time?

```
    DijkstraSSSP(G, s):
 6    foreach (Vertex v : G):
 7      d[v] = +inf
 8      p[v] = NULL
 9    d[s] = 0
10
11    PriorityQueue Q // min distance, defined by d[v]
12    Q.buildHeap(G.vertices())
13    Graph T          // "labeled set"
14
15    repeat n times:
16      Vertex u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19        if cost(u, v) + d[u] < d[v]:
20          d[v] = cost(u, v) + d[u]
21          p[v] = m
22
23    return T
```

Landmark Path Problem

Suppose you want to travel from **A** to **G**.

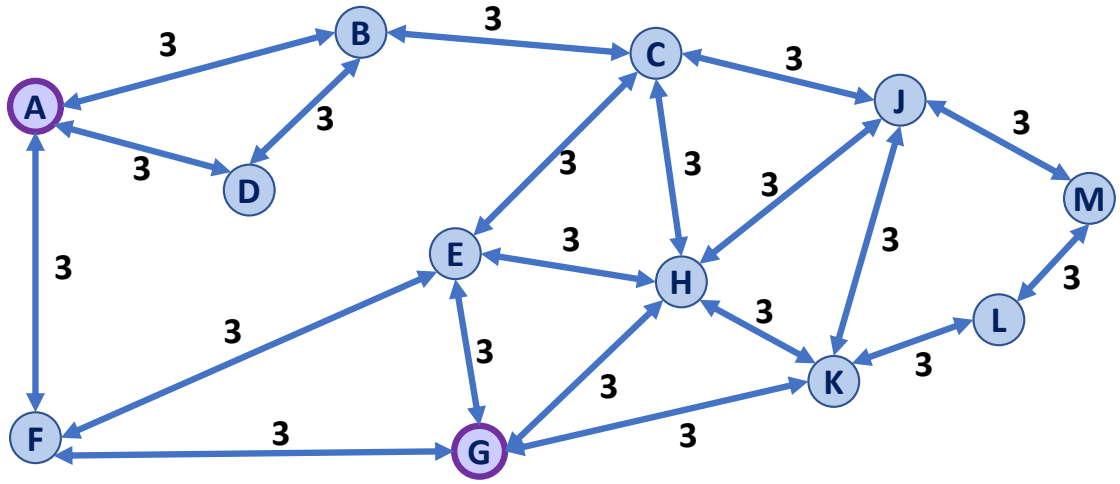
Q1: What is the shortest path from **A** to **G**?



Landmark Path Problem

Suppose you want to travel from **A** to **G**.

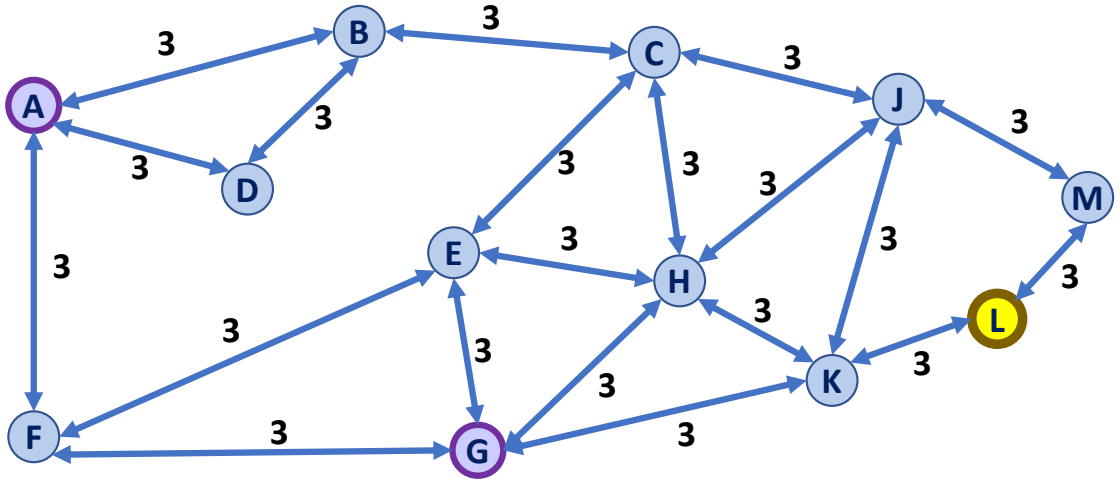
Q2: What is the fastest algorithm to use to find the shortest path?



Landmark Path Problem

In your journey between **A** and **G**, you also want to visit the landmark **L**.

Q3: What is the shortest path from **A** to **G** that visits **L**?



Landmark Path Problem

In your journey between **A** and **G**, you also want to visit the landmark **L**.

Q4: What is the fastest algorithm to find this path?

Q5: What are the specific call(s) to this algorithm?