CS 400

Graphs: BFS Traversal

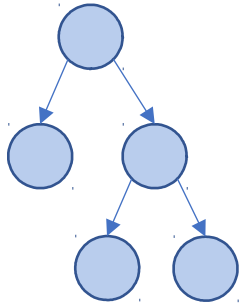
ID: 13-01

Traversal:

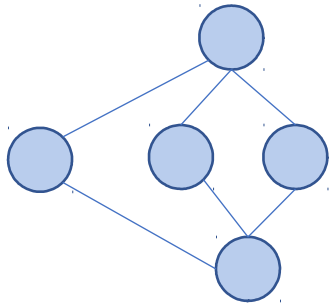
Objective: Visit every vertex and every edge in the graph.

Purpose: Search for interesting sub-structures in the graph.

We've seen traversal before ….but it's different:



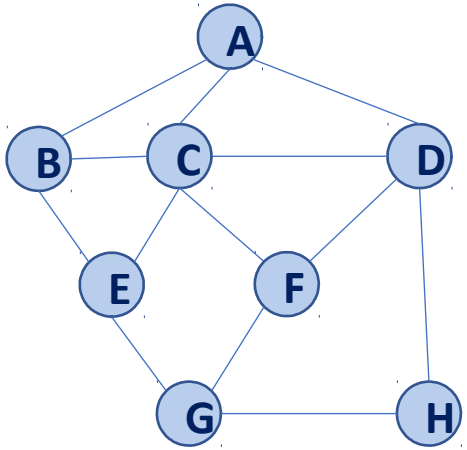
- Ordered
- Obvious Start
- Notion of Completion



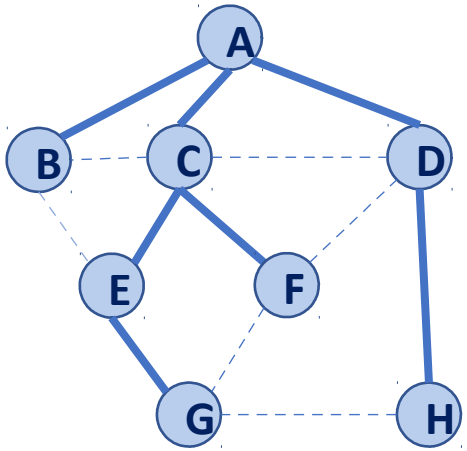
- Not ordered
- Not obvious Start
- No Notion of Completion

Traversal: BFS

Visit all children nodes before visit grandchild nodes, so visit all adjacent nodes before their adjacent nodes



Traversal: BFS



d	p		Adjacent Edges
0	A	**A**	**C B D**
1	A	**B**	**A C E**
1	A	**C**	**B A D E F**
1	A	**D**	**A C F H**
2	C	**E**	**B C G**
2	C	**F**	**C D G**
3	E	**G**	**E F H**
2	D	**H**	**D G**

queue pop out visited node and enqueue its adjacent nodes

```
1  BFS(G):
2    Input: Graph, G
3    Output: A labeling of the edges on
4        G as discovery and cross edges
5
6    foreach (Vertex v : G.vertices()):
7      setLabel(v, UNEXPLORED)
8    foreach (Edge e : G.edges()):
9      setLabel(e, UNEXPLORED)
10   foreach (Vertex v : G.vertices()):
11     if getLabel(v) == UNEXPLORED:
12       BFS(G, v)
```

```
14 BFS(G, v):
15   Queue q
16   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20     v = q.dequeue()
21     foreach (Vertex w : G.adjacent(v)):
22       if getLabel(w) == UNEXPLORED:
23         setLabel(v, w, DISCOVERY)
24         setLabel(w, VISITED)
25         q.enqueue(w)
26       elseif getLabel(v, w) == UNEXPLORED:
27         setLabel(v, w, CROSS)
```

CS 400

Graphs: BFS Analysis

ID: 13-02

BFS Analysis

Q: Does our implementation handle disjoint graphs? If so, what code handles this?

- ***How do we use this to count components?***

Add new line at 13: Comp++;

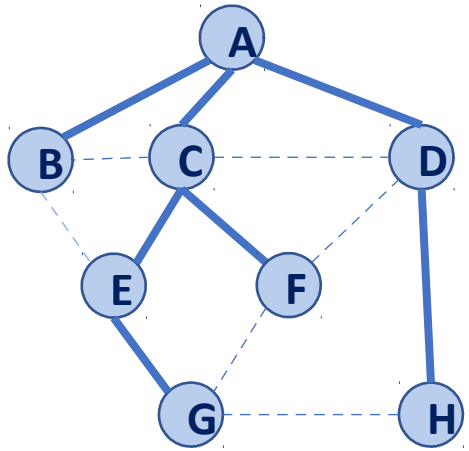
Q: Does our implementation detect a cycle?

- ***How do we update our code to detect a cycle?***

Add new line at 28: cycle = true;

Q: What is the running time?

Running time of BFS



d	p	v	Adjacent
0	A	A	C B D
1	A	B	A C E
1	A	C	B A D E F
1	A	D	A C F H
2	C	E	B C G
2	C	F	C D G
3	E	G	E F H
2	D	H	D G

While-loop at **:19**? n

O(n+m)

For-loop at **:21**? 2m

```
1  BFS(G):
2    Input: Graph, G
3    Output: A labeling of the edges on
4        G as discovery and cross edges
5
6    foreach (Vertex v : G.vertices()):
7      setLabel(v, UNEXPLORED)
8    foreach (Edge e : G.edges()):
9      setLabel(e, UNEXPLORED)
10   foreach (Vertex v : G.vertices()):
11     if getLabel(v) == UNEXPLORED:
12       BFS(G, v)
```

```
14 BFS(G, v):
15   Queue q
16   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20     v = q.dequeue()
21     foreach (Vertex w : G.adjacent(v)):
22       if getLabel(w) == UNEXPLORED:
23         setLabel(v, w, DISCOVERY)
24         setLabel(w, VISITED)
25         q.enqueue(w)
26       elseif getLabel(v, w) == UNEXPLORED:
27         setLabel(v, w, CROSS)
```

BFS Observations

Q: What is a shortest path from **A** to **H**?

A-D-H

Q: What is a shortest path from **E** to **H**?

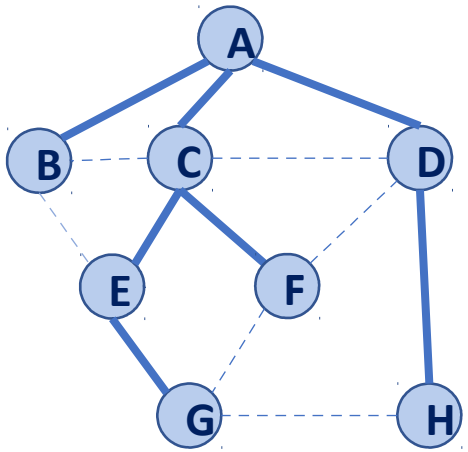
Not possible w/ BFS

Q: How does a cross edge relate to **d**?

Q: What structure is made from discovery edges?

d	p	v	Adjacent
0	A	A	C B D
1	A	B	A C E
1	A	C	B A D E F
1	A	D	A C F H
2	C	E	B C G
2	C	F	C D G
3	E	G	E F H
2	D	H	D G

predecessor vertex



BFS Observations

Obs. 1: Traversals can be used to count components.

Obs. 2: Traversals can be used to detect cycles.

Obs. 3: In BFS, **d** provides the shortest distance to every vertex.

Obs. 4: In BFS, the endpoints of a cross edge never differ in distance, **d**, by more than 1:

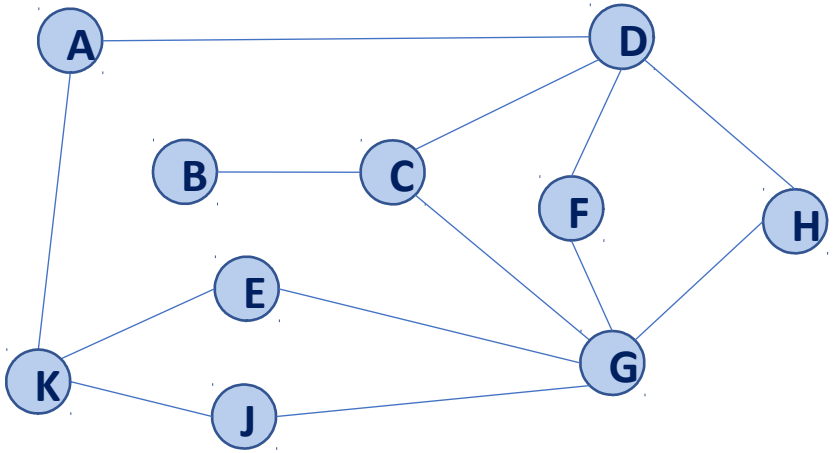
$$|d(u) - d(v)| = 1$$

CS 400

Graphs: DFS Traversal

ID: 13-03

Traversal: DFS



DFS go very deep quickly and use stack structure

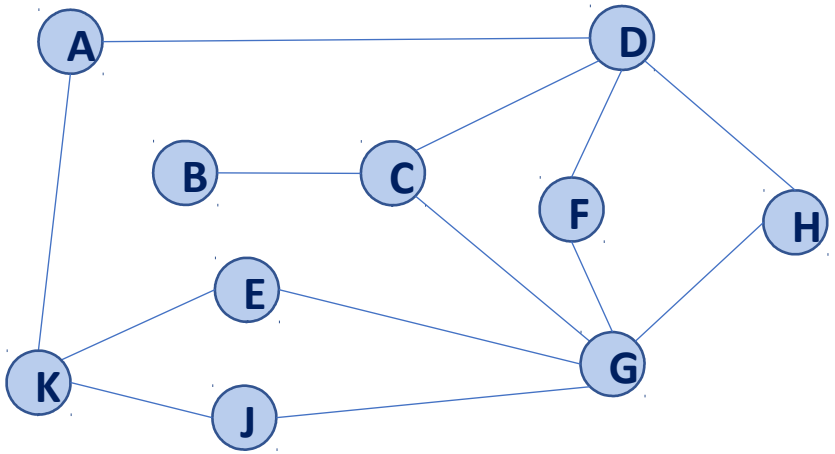
```
1  BFS(G):
2    Input: Graph, G
3    Output: A labeling of the edges on
4        G as discovery and cross edges
5
6    foreach (Vertex v : G.vertices()):
7      setLabel(v, UNEXPLORED)
8    foreach (Edge e : G.edges()):
9      setLabel(e, UNEXPLORED)
10   foreach (Vertex v : G.vertices()):
11     if getLabel(v) == UNEXPLORED:
12       BFS(G, v)
```

```
14 BFS(G, v):
15   Queue q
16   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20     v = q.dequeue()
21     foreach (Vertex w : G.adjacent(v)):
22       if getLabel(w) == UNEXPLORED:
23         setLabel(v, w, DISCOVERY)
24         setLabel(w, VISITED)
25         q.enqueue(w)
26       elseif getLabel(v, w) == UNEXPLORED:
27         setLabel(v, w, CROSS)
```

```
1  DFS(G):
2    Input: Graph, G
3    Output: A labeling of the edges on
4        G as discovery and back edges
5
6    foreach (Vertex v : G.vertices()):
7      setLabel(v, UNEXPLORED)
8    foreach (Edge e : G.edges()):
9      setLabel(e, UNEXPLORED)
10   foreach (Vertex v : G.vertices()):
11     if getLabel(v) == UNEXPLORED:
12       DFS(G, v)
```

```
14 DFS(G, v):
15   Queue q
16   setLabel(v, VISITED)
17   q.enqueue(v)
18
19   while !q.empty():
20     v = q.dequeue()
21     foreach (Vertex w : G.adjacent(v)):
22       if getLabel(w) == UNEXPLORED:
23         setLabel(v, w, DISCOVERY)
24         setLabel(w, VISITED)
25         DFS(G, w)
26       elseif getLabel(v, w) == UNEXPLORED:
27         setLabel(v, w, BACK)
```

Running time of DFS



Labeling:

- Vertex: 2n -> O(n)
- Edge: 2*m -> O(m)

Queries:

- Vertex: n -> O(n)
- Edge: sum(dev(v)) = 2m -> O(m)

O(n+m)

Reference:
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

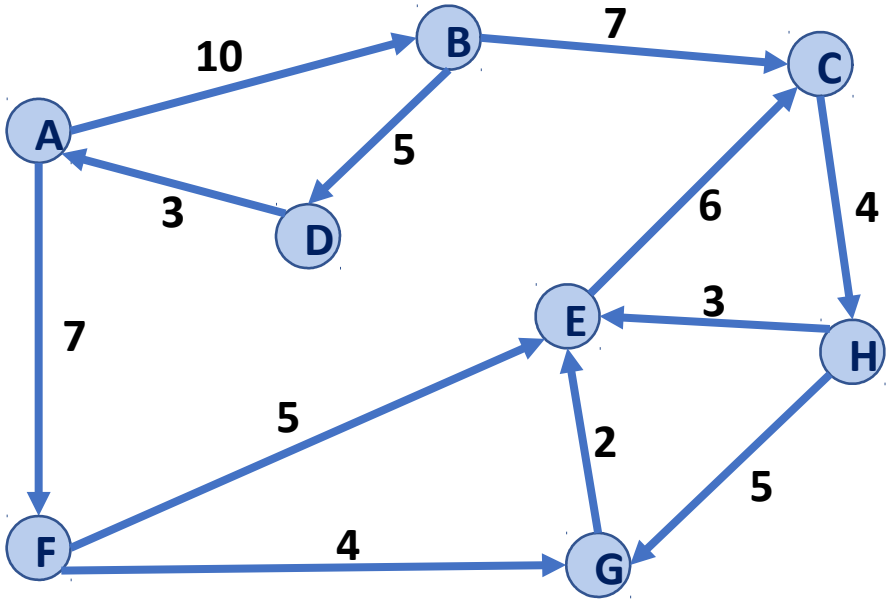
CS 400

Graphs: Dijkstra's Algorithm

ID: 15-01

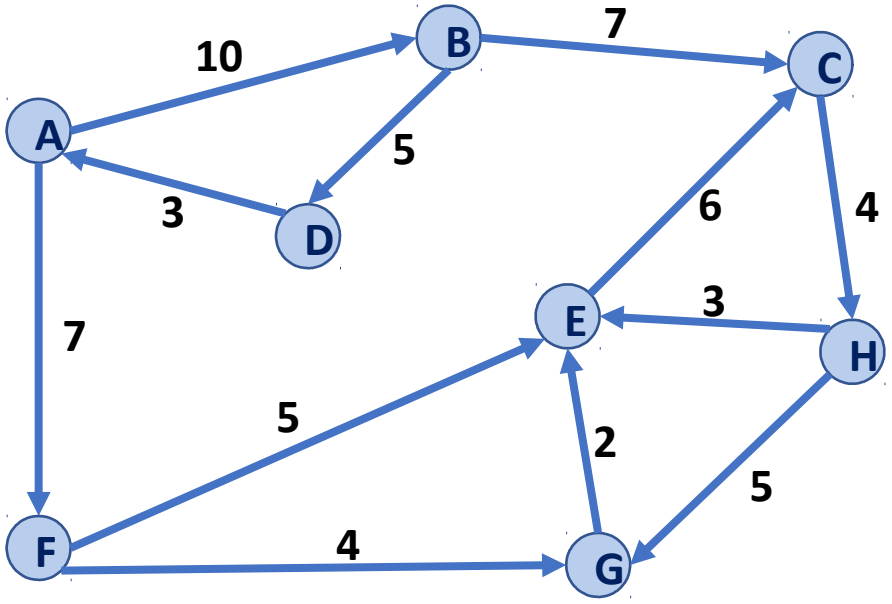
Dijkstra's Algorithm (SSSP)

find shortest path in the graph




```
    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)

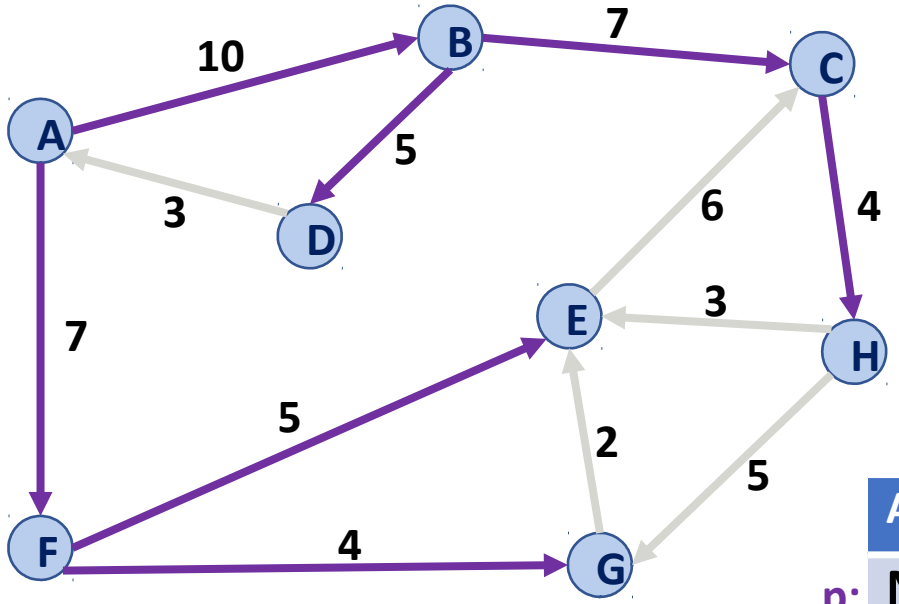



```
    DijkstraSSSP(G, s):
 6    foreach (Vertex v : G):
 7      d[v] = +inf
 8      p[v] = NULL
 9    d[s] = 0
10
11    PriorityQueue Q // min distance, defined by d[v]
12    Q.buildHeap(G.vertices())
13    Graph T          // "labeled set"
14
15    repeat n times:
16      Vertex u = Q.removeMin()
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
```

$O(m + n * \log(n))$

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:	NULL	A	B	B	F	A	F	C
d:	0	10	17	15	12	7	11	21

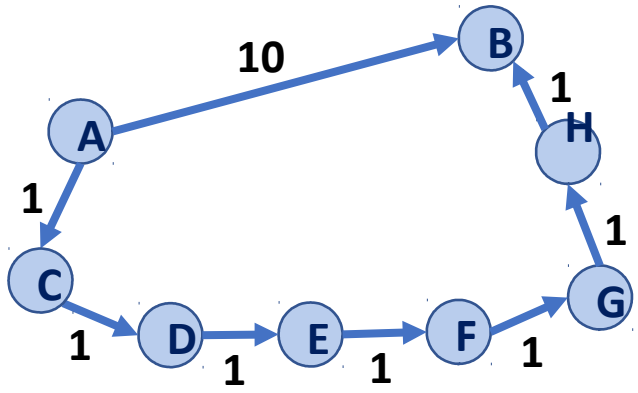
CS 400

Graphs: Dijkstra’s Edge Cases

ID: 15-02

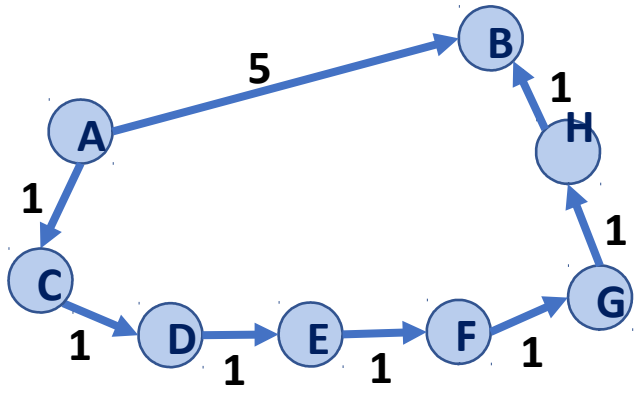
Dijkstra's Algorithm (SSSP)

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



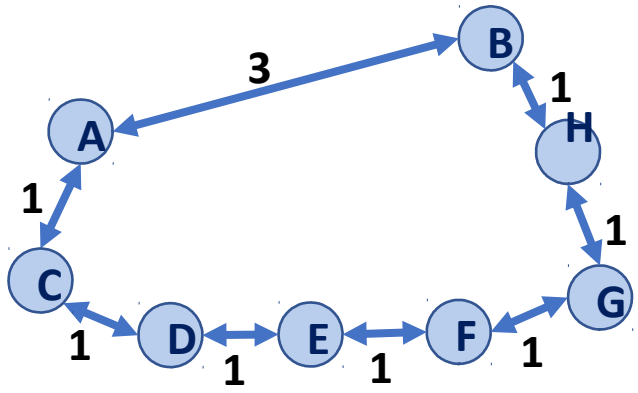
Dijkstra's Algorithm (SSSP)

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



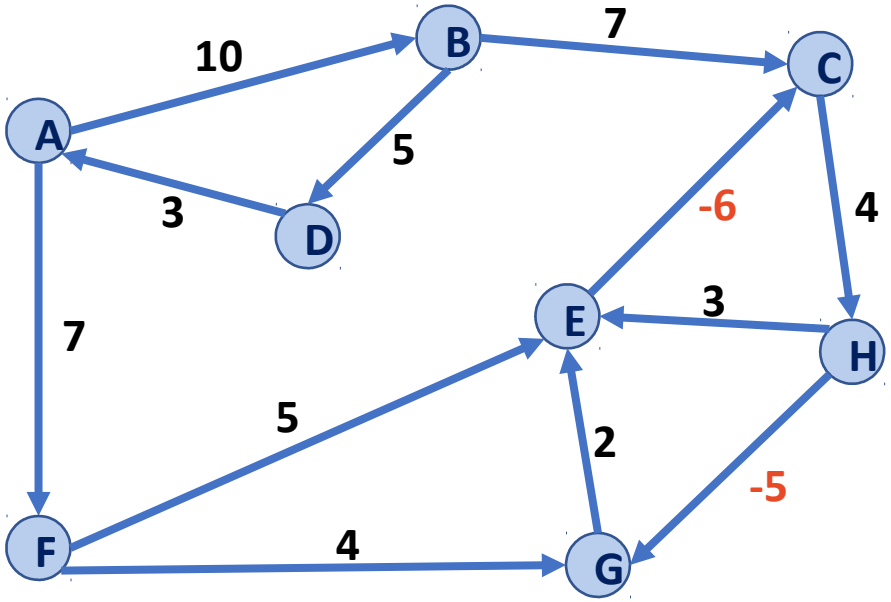
Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle undirected graphs?



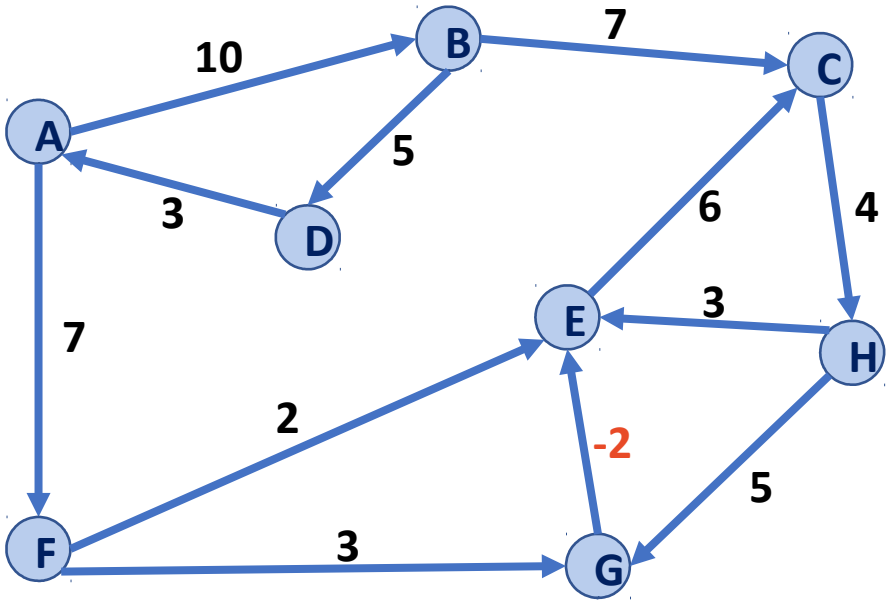
Dijkstra's Algorithm (SSSP)

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?



CS 400

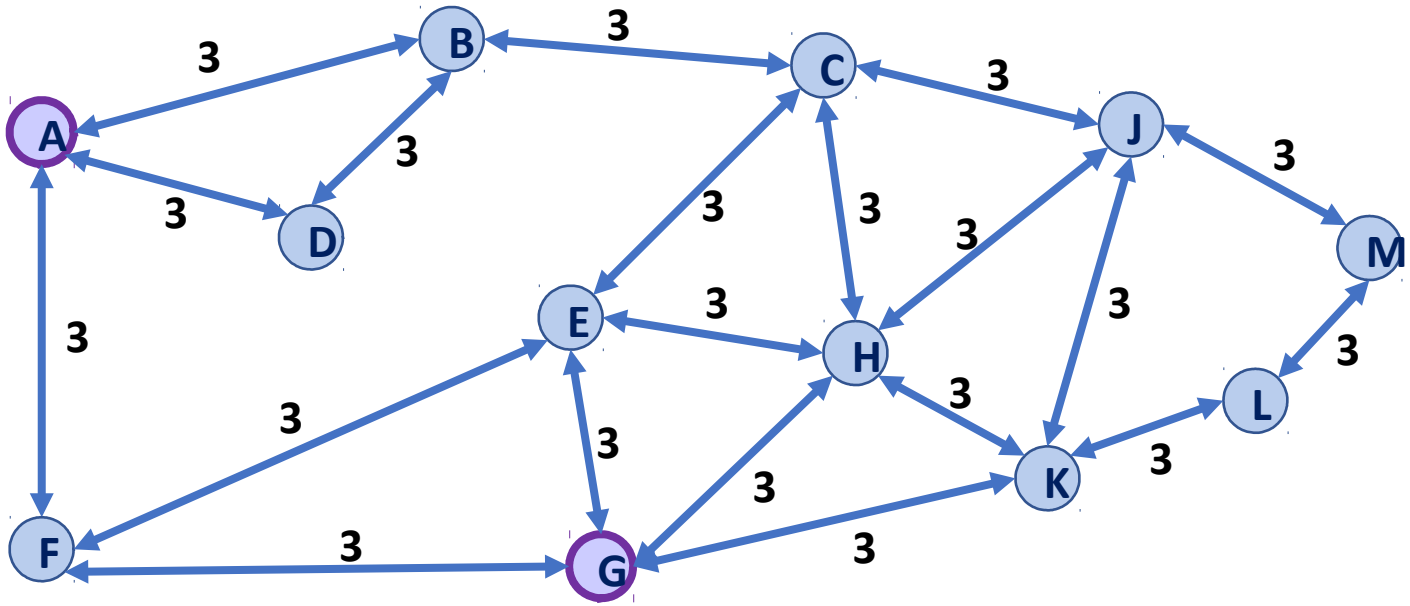
Graphs: Landmark Path Problem

ID: 15-03

Landmark Path Problem

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

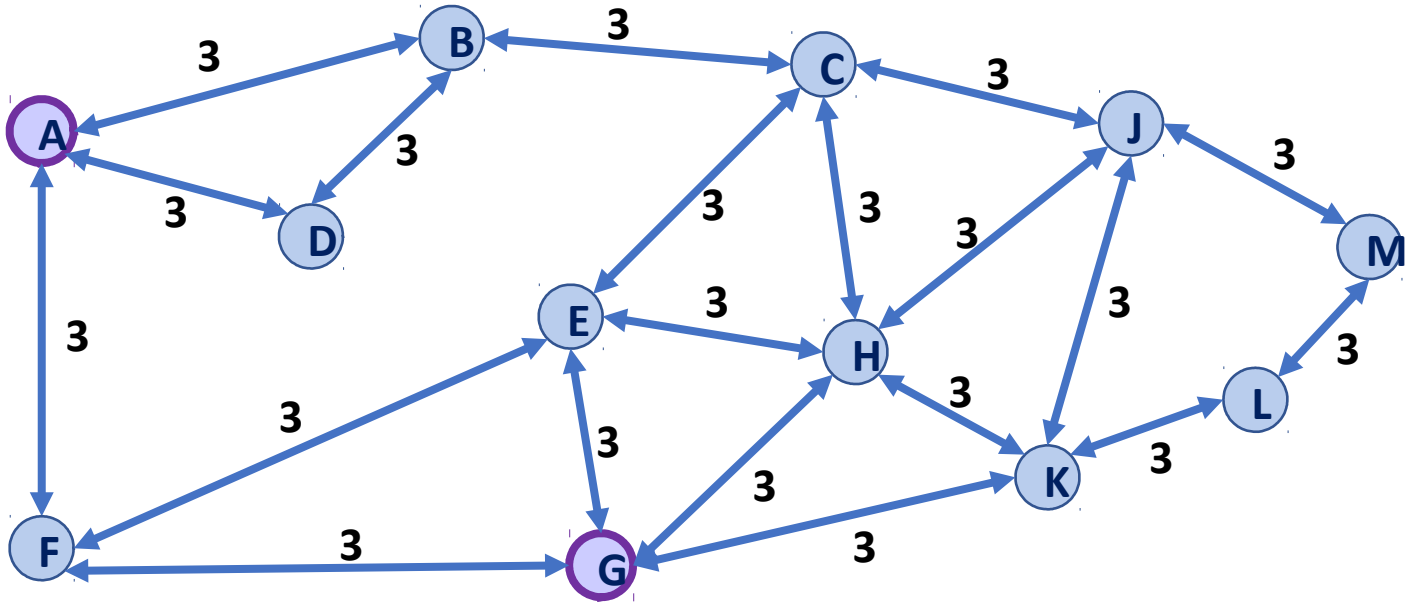
Q1: What is the shortest path from **A** to **G**?



Landmark Path Problem

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

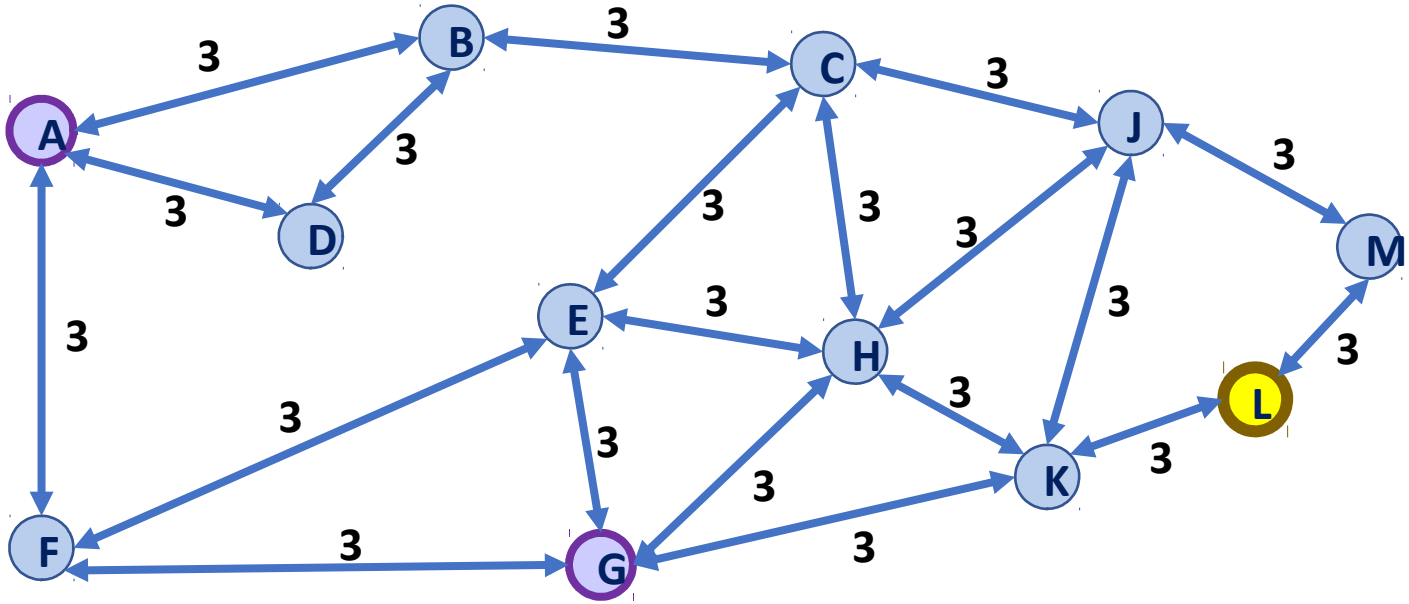
Q2: What is the fastest algorithm to use to find the shortest path?



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?