Check in First

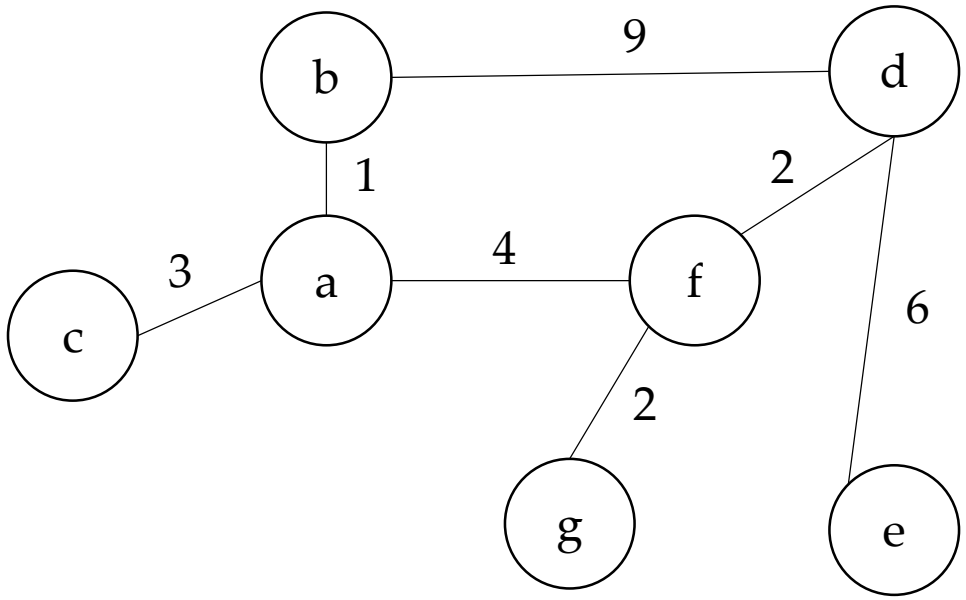
Course Code: JI3NK

Lab 09: Dijkstra's Algorithm and Kruskal's Minimum Spanning Tree

- For this lab, you will write two common graph algorithms: Dijkstra's Shortest path algorithm and Kruskal's Minimum Spanning Tree (MST) algorithm.

- You will read summary map information from a text file, like what you did in lab 05.

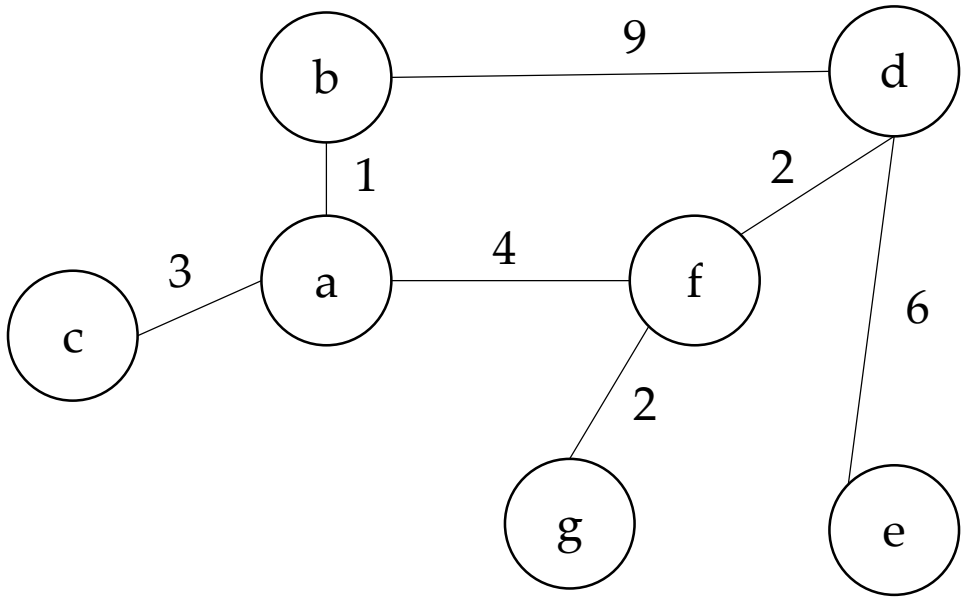
Dijkstra's Shortest Path Algorithm



Distance Table
a
b
c
d
e
f
g

Visited Nodes:
Minimum Heap:

Dijkstra's Shortest Path Algorithm

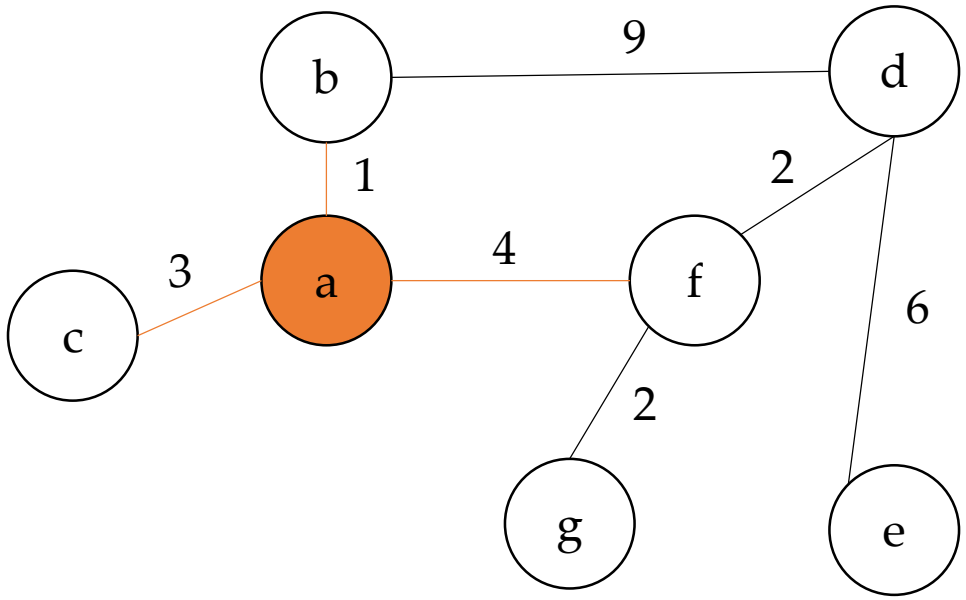


Distance Table	
a	0
b	INF
c	INF
d	INF
e	INF
f	INF
g	INF

Start from a, initialize distance table.

Visited Nodes:
Minimum Heap: (a, 0)

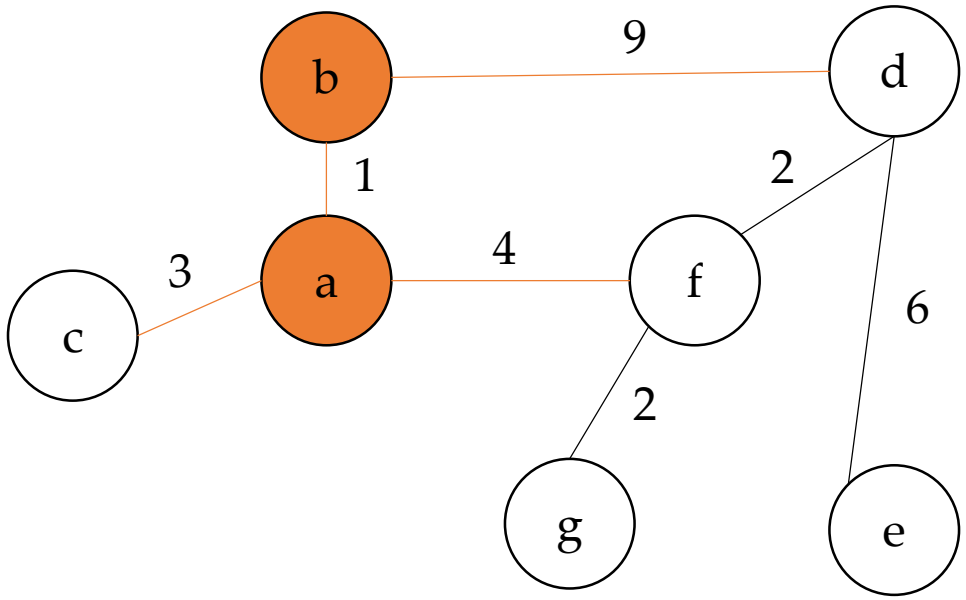
Dijkstra's Shortest Path Algorithm



Distance Table	
a	0
b	0+1 < INF? 1
c	0+3 < INF? 3
d	INF
e	INF
f	0+4 < INF? 4
g	INF

Visited Nodes: a
Minimum Heap: ~~(a, 0)~~ (b, 1), (c, 3), (f, 4)

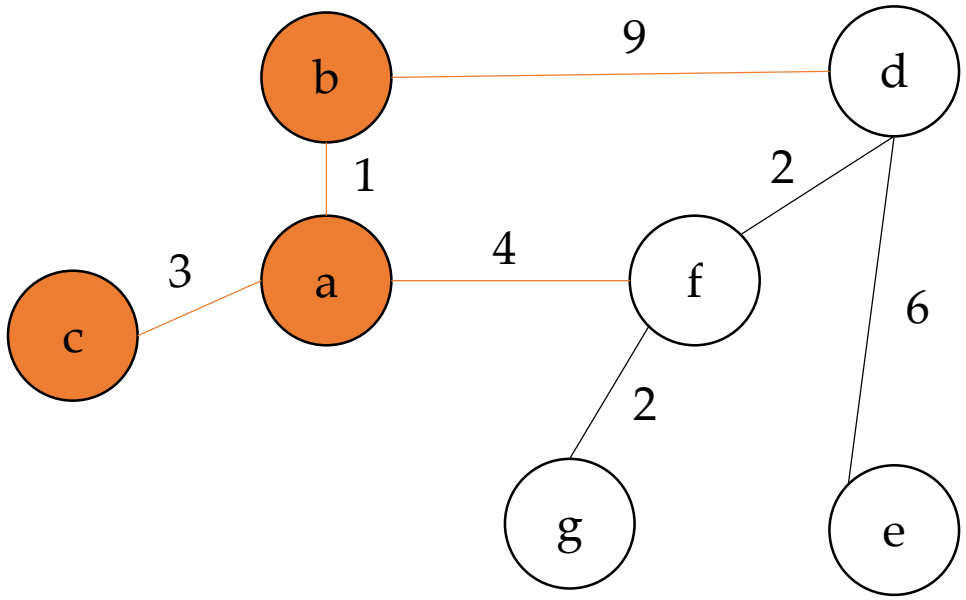
Dijkstra's Shortest Path Algorithm



Distance Table	
a	0
b	1
c	3
d	1+9 < INF? 10
e	INF
f	4
g	INF

Visited Nodes: a, b
Minimum Heap: ~~(b, 1)~~ (c, 3), (f, 4), (d, 10)

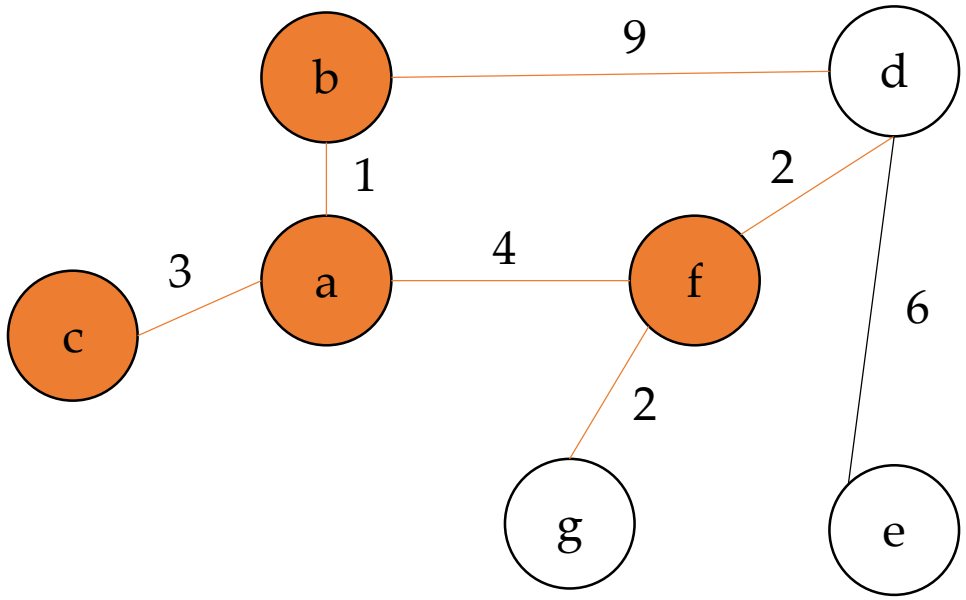
Dijkstra's Shortest Path Algorithm



Distance Table	
a	3+3 < 0? 0
b	1
c	3
d	10
e	INF
f	4
g	INF

Visited Nodes: a, b, c
Minimum Heap: ~~(c, 3)~~ (f, 4), (d, 10)

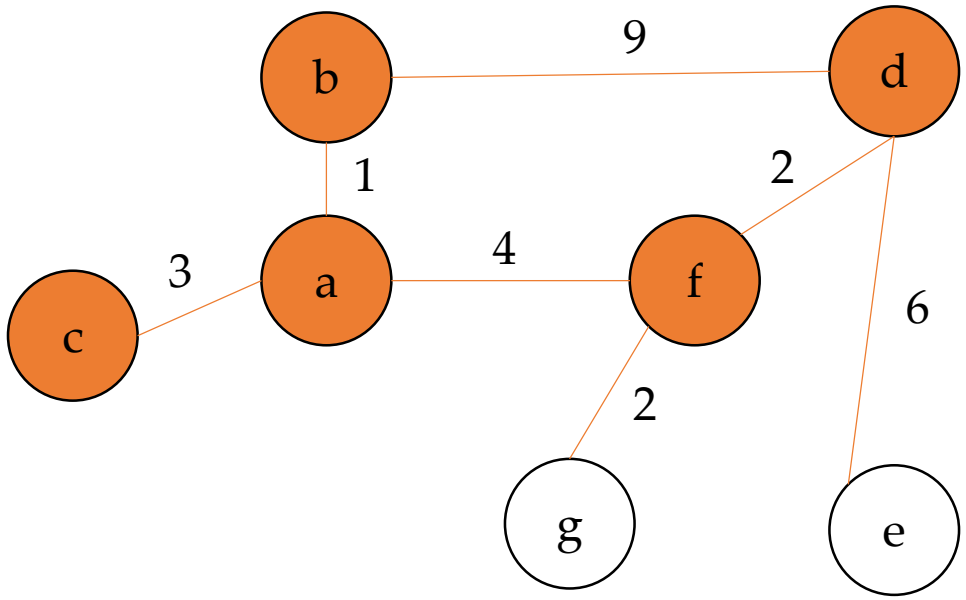
Dijkstra's Shortest Path Algorithm



Distance Table	
a	4+4 < 0? 0
b	1
c	3
d	4+2 < 10? 6
e	INF
f	4
g	4+2 < INF? 6

Visited Nodes: a, b, c, f
Minimum Heap: ~~(f, 4)~~ (d, 6), (g, 6), (d, 10)

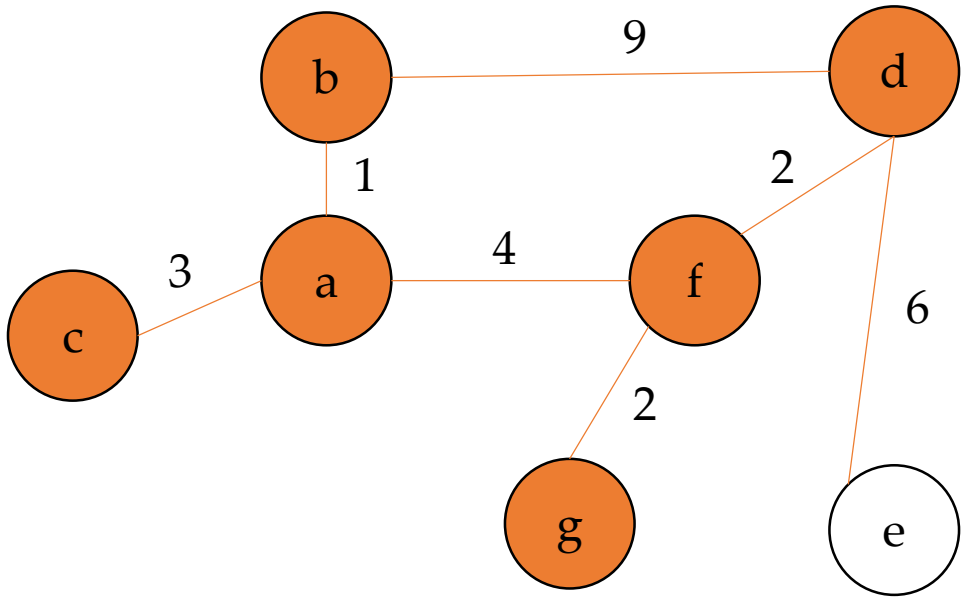
Dijkstra's Shortest Path Algorithm



Distance Table	
a	0
b	6+9 < 1? 1
c	3
d	6
e	6+6 < INF? 12
f	6+2 < 4? 4
g	6

Visited Nodes: a, b, c, f, d
Minimum Heap: ~~(d, 6)~~ (g, 6), (d, 10), (e, 12)

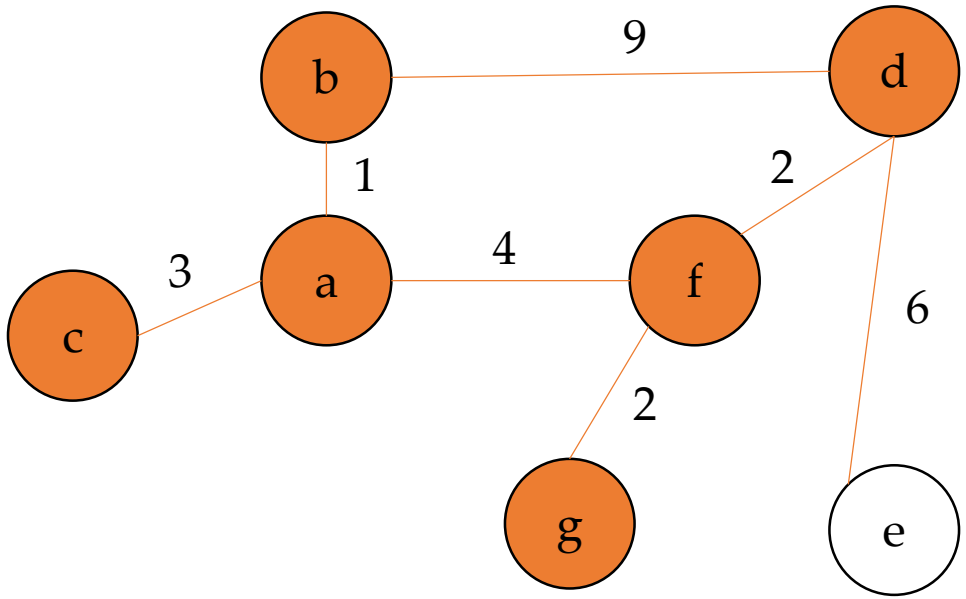
Dijkstra's Shortest Path Algorithm



Distance Table	
a	0
b	1
c	3
d	6
e	12
f	6+2 < 4? 4
g	6

Visited Nodes: a, b, c, f, d, g
Minimum Heap: ~~(g, 6)~~ (d, 10), (e, 12)

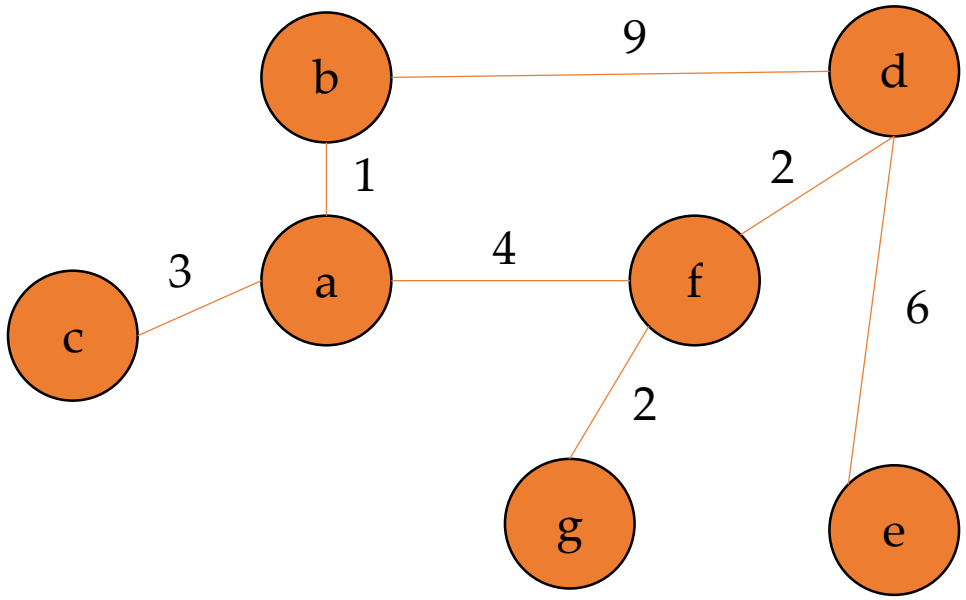
Dijkstra's Shortest Path Algorithm



Distance Table	
a	0
b	1
c	3
d	6
e	12
f	4
g	6

Visited Nodes: a, b, c, f, d, g
Minimum Heap: ~~(d, 10)~~ (e, 12)

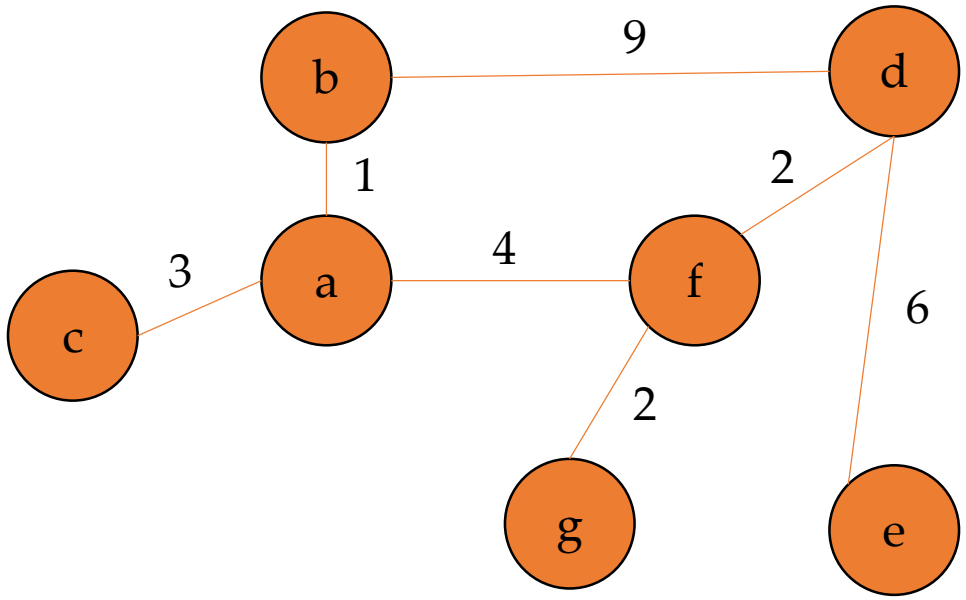
Dijkstra's Shortest Path Algorithm



Distance Table	
a	0
b	1
c	3
d	12+6 < 6? 6
e	12
f	4
g	6

Visited Nodes: a, b, c, f, d, g, e
Minimum Heap: ~~(e, 12)~~

Dijkstra's Shortest Path Algorithm



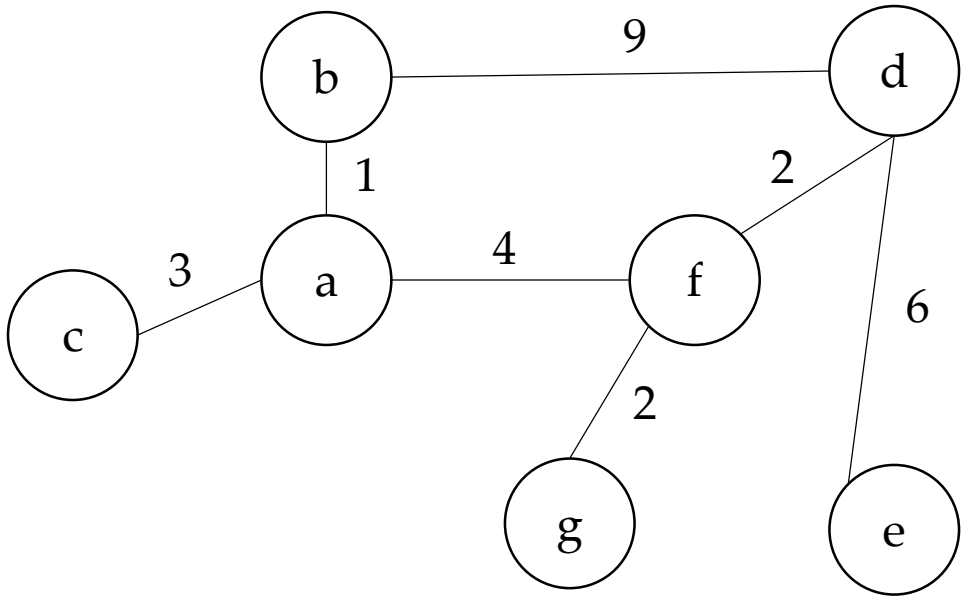
Distance Table	
a	0
b	1
c	3
d	6
e	12
f	4
g	6

Visited Nodes: a, b, c, f, d, g, e
Minimum Heap:

Pseudocode for Dijkstra's Algorithm

```
Create distance table, visited set, and minimum heap
Initialize distance table as slide 5
While the size of visited set != the size of the graph:
    get the front element of the heap
    if we have not visited this node:
        add it to the visited set
        for all its neighbors in the graph:
            new distance = heap distance + distance to its neighbors
            if new distance < distance table[neighbor]:
                update distance table
                push the neighbor into the minimum heap
Print the distance table
```

Kruskal's MST Algorithm



When we create the graph, we also create an edge list, and it should be sorted at the beginning of the algorithm.

Edge list:

(a, b, 1)

(f, g, 2)

(d, f, 2)

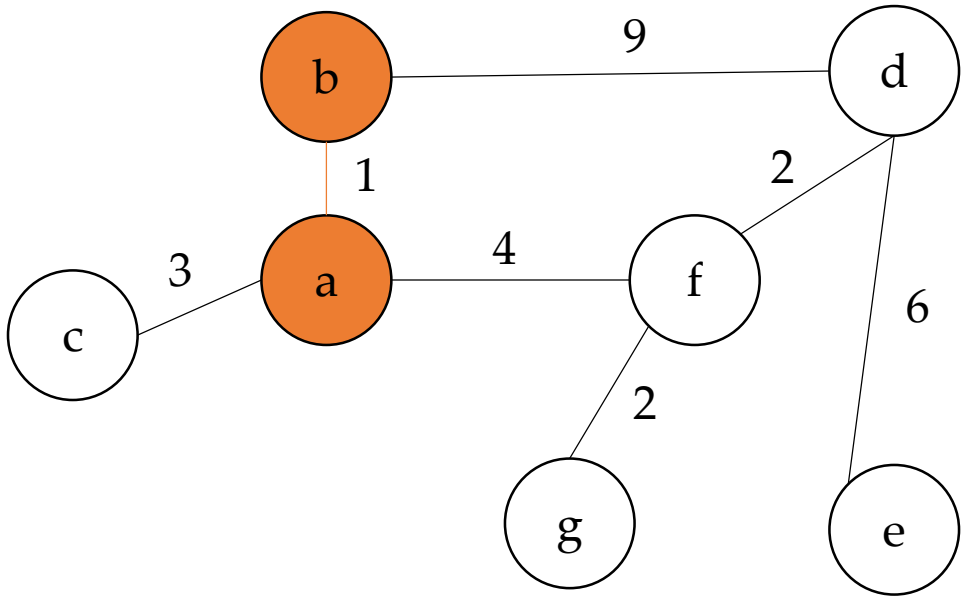
(a, c, 3)

(a, f, 4)

(d, e, 6)

(b, d, 9)

Kruskal's MST Algorithm



Minimum spanning tree:
(a, b, 1)

Edge list:

(a, b, 1)

(f, g, 2)

(d, f, 2)

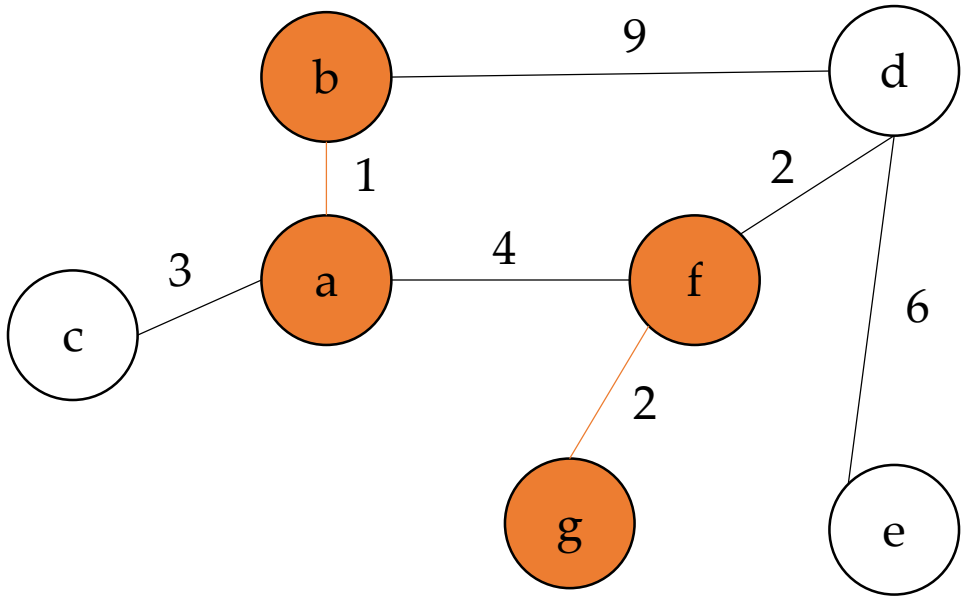
(a, c, 3)

(a, f, 4)

(d, e, 6)

(b, d, 9)

Kruskal's MST Algorithm



Minimum spanning tree:
(a, b, 1), (f, g, 2)

Edge list:

(a, b, 1)

(f, g, 2)

(d, f, 2)

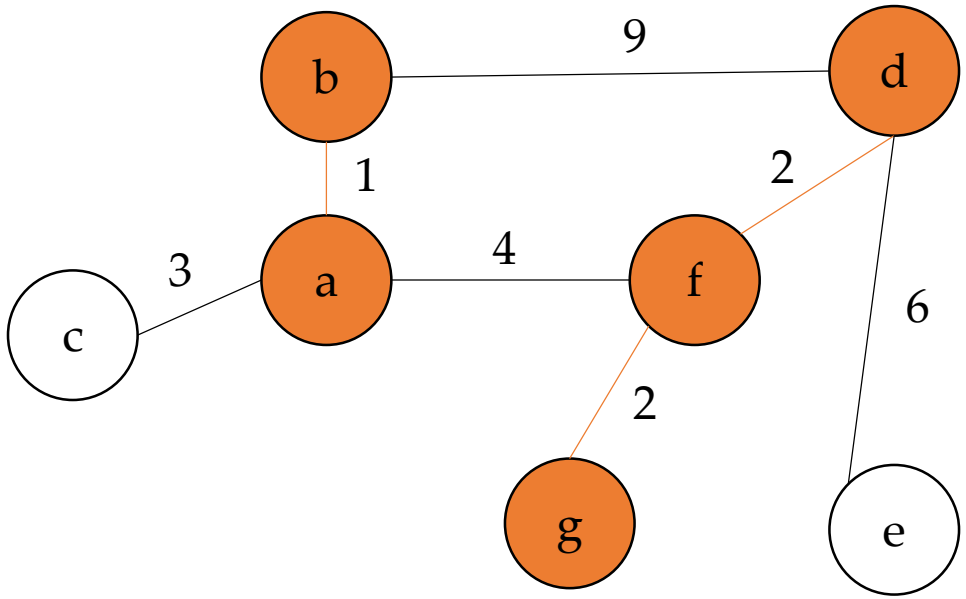
(a, c, 3)

(a, f, 4)

(d, e, 6)

(b, d, 9)

Kruskal's MST Algorithm



Minimum spanning tree:
(a, b, 1), (f, g, 2), (d, f, 2)

Edge list:

(a, b, 1)

(f, g, 2)

(d, f, 2)

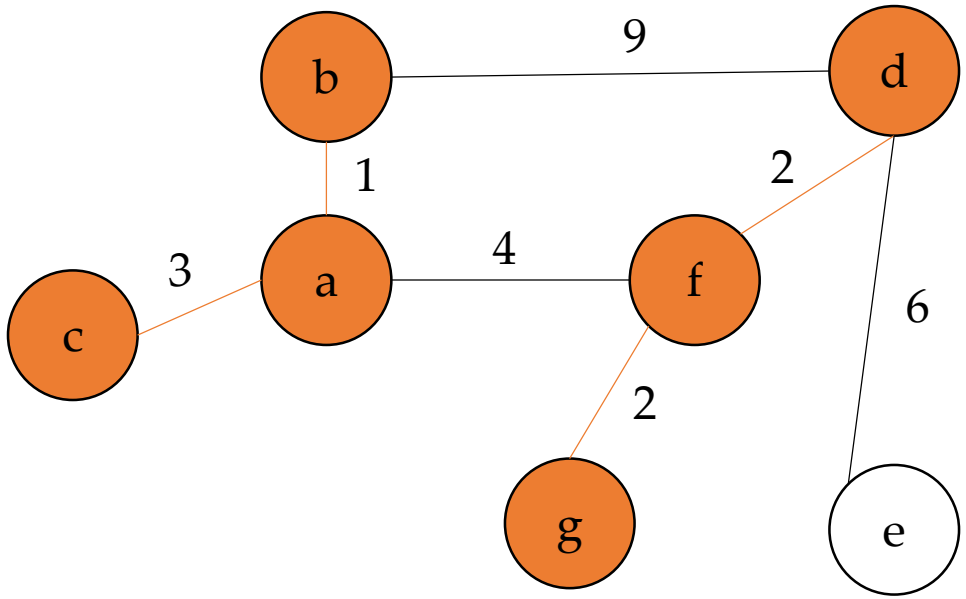
(a, c, 3)

(a, f, 4)

(d, e, 6)

(b, d, 9)

Kruskal's MST Algorithm



Minimum spanning tree:
(a, b, 1), (f, g, 2), (d, f, 2), (a, c, 3)

Edge list:

(a, b, 1)

(f, g, 2)

(d, f, 2)

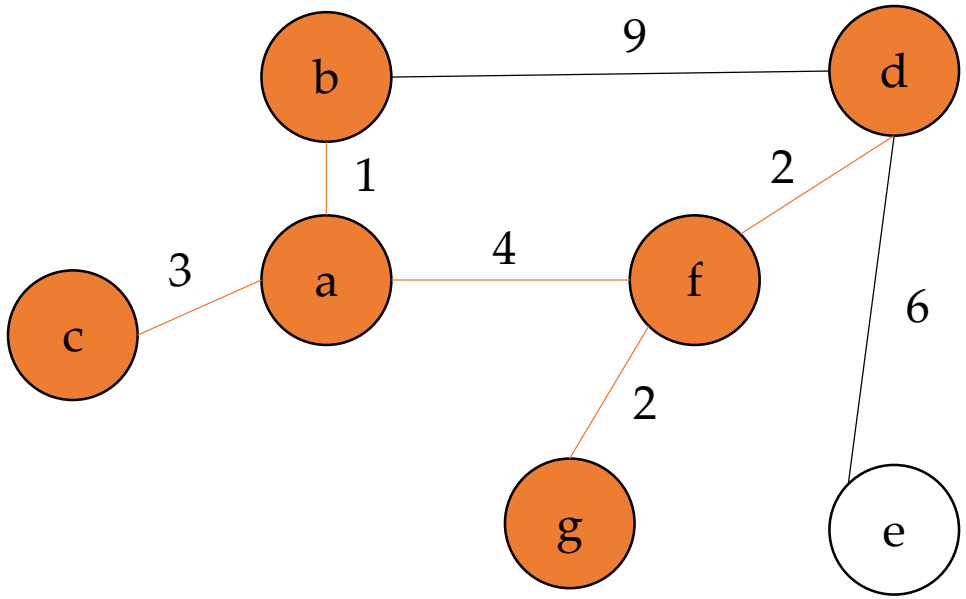
(a, c, 3)

(a, f, 4)

(d, e, 6)

(b, d, 9)

Kruskal's MST Algorithm



Minimum spanning tree:
(a, b, 1), (f, g, 2), (d, f, 2), (a, c, 3), (a, f, 4)

Edge list:

(a, b, 1)

(f, g, 2)

(d, f, 2)

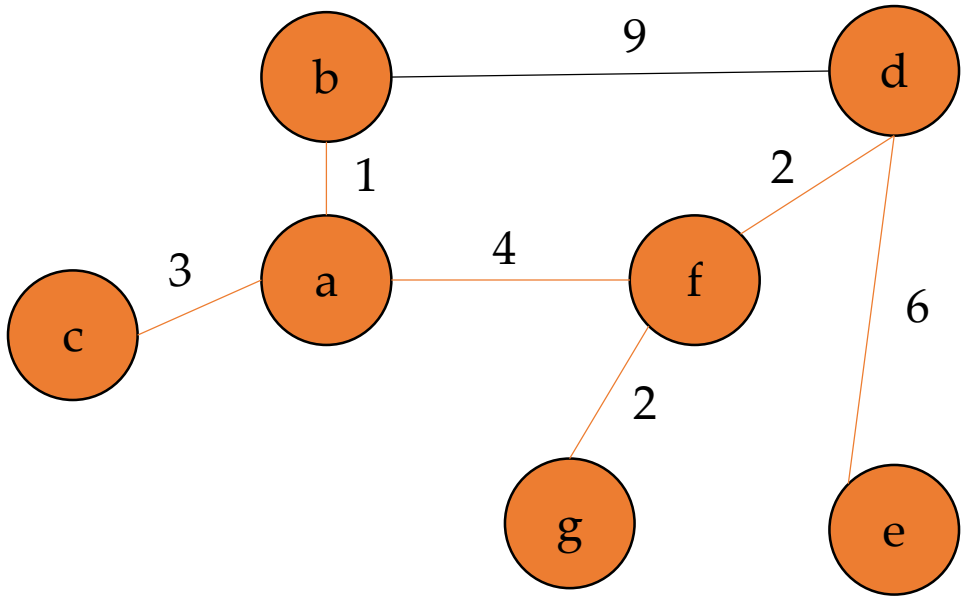
(a, c, 3)

(a, f, 4)

(d, e, 6)

(b, d, 9)

Kruskal's MST Algorithm



Minimum spanning tree:
(a, b, 1), (f, g, 2), (d, f, 2), (a, c, 3), (a, f, 4), (d, e, 6)

Edge list:

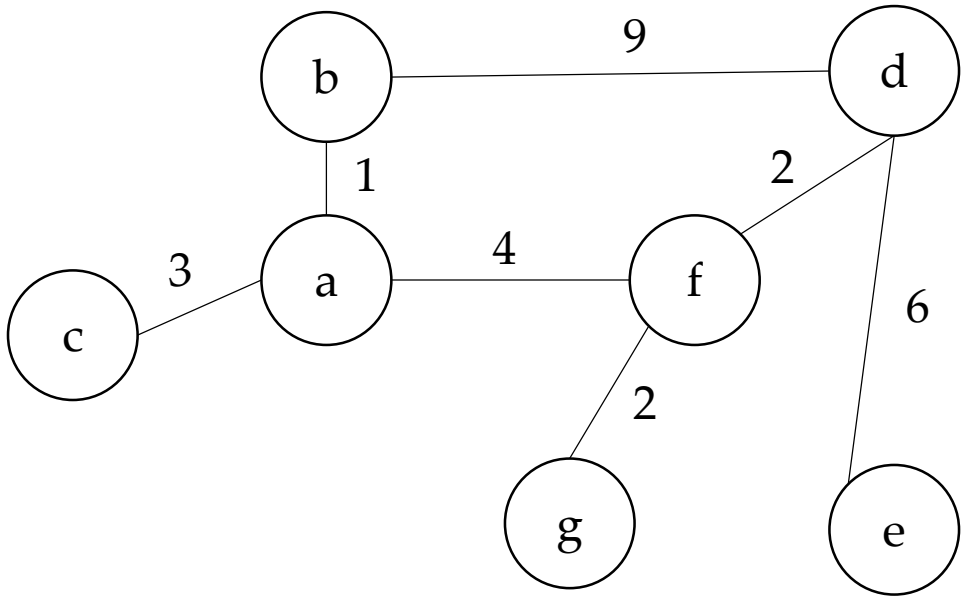
(a, b, 1)
(f, g, 2)
(d, f, 2)
(a, c, 3)
(a, f, 4)
(d, e, 6)
(b, d, 9)

The program should stop here, but if we add b-d edge, we will have a cycle in the graph!

Pseudocode for Kruskal's Algorithm

```
Create minSpanTreeGraph and minSpanTreeEdges
Sort the edge list (add edges in it in add_edge() function)
Initialize i to 0
While the size of minSpanTreeEdges != the size of the graph - 1:
    get the i-th edge of the edge list
    add the edge into minSpanTreeGraph
    if minSpanTreeGraph contains a cycle:
        remove the edges we just added
    else:
        add the edge into the minSpanTreeEdges
Print minSpanTreeEdges
```

Decide a Cycle in an Undirected Graph



- You can use DFS to detect if there is a circle in the graph.
- Useful link: https://www.geeksforgeeks.org/detect-cycle-undirected-graph/

Coding

- Remember we used map<node, vector<pair<node, int> > > in lab 05? However, we may not use it in this lab, because we need sort (node, distance) in the minimum heap. (I suggest that you create a new structure and write the operator function)
- Use <climits> library and INT_MAX to represent the infinite value.
- For Kruskal's algorithm, you need to create a structure (e.g. edge) that contains two nodes and the distance between them, and write operator < function so that you can use sort() in <algorithm> library.

Submission

- Submit both source files (graphalgs.h) and at least three test cases (lab09.txt). Please keep the original test cases!

- Due date: **Next** Friday 11:59 PM.