CS 260 – Homework #8

Chris Kasper

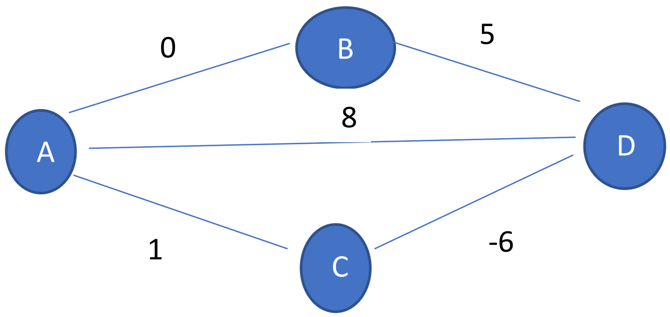
March 2, 2018

Problem 1 (6.6 from text)

Show that the program Dijkstra does not work correctly if arc costs can be negative.

Solution

Consider the following graph:



Running Dijkstra on this graph with A being the starting node to D, the algorithm will result in the shortest path being 5. However, the shortest path to D is in fact -5 (A to C to D).

Problem 2 (7.1 from text)

Describe an algorithm to insert and delete edges in the adjacency list representation for an undirected graph. Remember that an edge (i, j) appears on the adjacency list for both vertex i and vertex j.

Solution

```
Insert_Edge(G,v1,v2){
    New_edge = {}
    If {v1,v2} in adj_list:
        Return
    Else:
        New_edge = Insert(v1)
        New_edge = Insert(v2)
        G.append(New_edge)
}


Delete_Edge(G,v1,v2){
    If {v1,v2} in G:
        For edges in G:
            If edge == {v1,v1}:
                G.remove(edge)
    Else:
        Return
}
```

Problem 3 (7.2 from text)

Modify the adjacency list representation for an undirected graph so that the first edge on the adjacency list for a vertex can be deleted in constant time. Write an algorithm to delete the first edge at a vertex using your new representation. *Hint.* How do you arrange that the two cells representing edge (i, j) can be found quickly from one another?

Solution

Consider if the adjacency list was modified to contain dictionaries (hash map) of edges [values] for each vertex [key]. This would make deleting the first edge of each vertex much easier. You'd just have the index for a specific vertex's edge list, and then delete the first one.

```
Delete_First_Edge(G, v1){
        if not G:
                Return
        else if v1 not in G:
                Return
        else:
                G[v1].remove(0)
                Return G
}
```

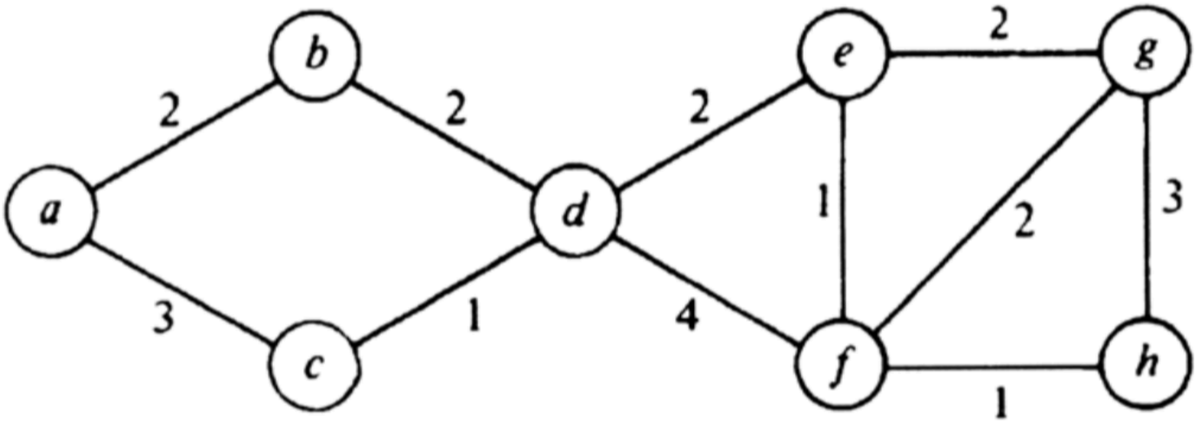
Problem 4 (7.3 a, b, d from text)

Consider the graph of Fig. 7.20.

a. Find a minimum-cost spanning tree by Prim's algorithm.

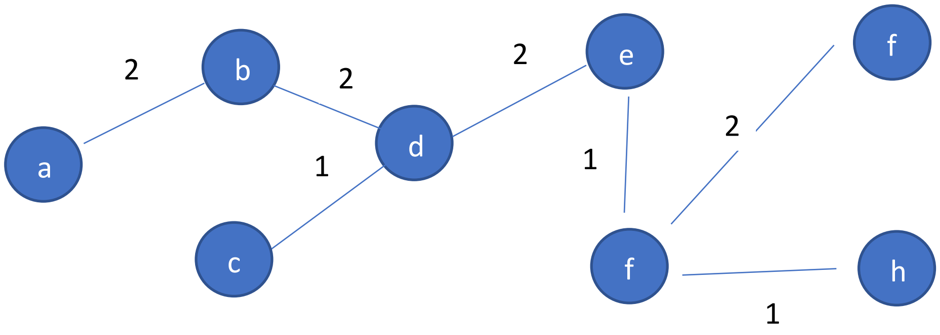
b. Find a minimum-cost spanning tree by Kruskal's algorithm.

d. Find a breadth-first spanning tree starting at *a* and at *d*.

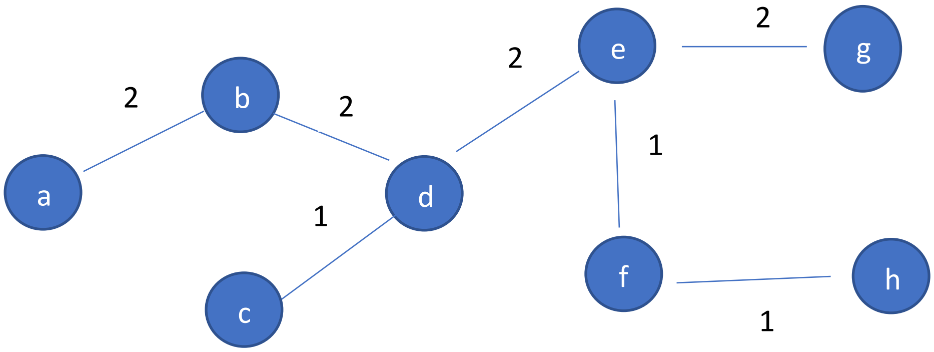


Solution

(a) Find a minimum-cost spanning tree by Prim's algorithm.



(b) Find a minimum-cost spanning tree by Kruskal's algorithm.



(d) Find a breadth-first spanning tree starting at *a* and at *d*.

At a:

Result: a, b, c, d, e, f, g, h

At d:

Result: d, b, c, e, f, a, g, h