Enlai Yii
2064210

Homework 4

Exercise 1

(a)

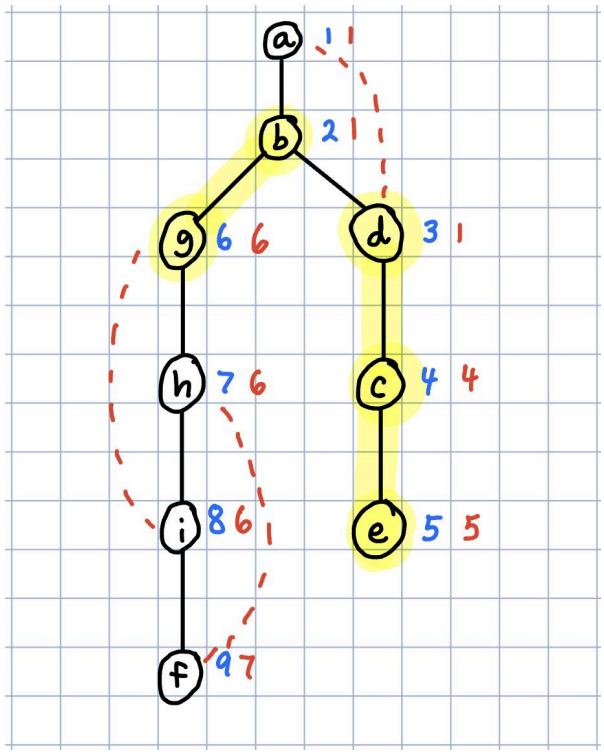
The algorithm is taken from the textbook (algorithm 47). It will take a graph and a starting vertex as parameters. The algorithm will use DFS to find the cut edges. We are going to iterate through all vertices adjacent to s, and all vertices will be visited. After DFS is done, the algorithm will check if the low number of vertex v is greater than the number assigned to s, if true, then a cut edge is found and outputted. The algorithm will keep going until all cut edges are found and outputted.

```
function DFSCutEdges(G, s):
    visited[s] = True
    num[s] = count
    count += 1
    low[s] = num[s]
    for v ∈ Adj[s]:
        if not visited[v]:
            Add edge (s, v) to T
            parent[v] = s
            DFSCutEdges(v)
            if low[v] > num[s]:
                Output s-v is a cut-edge
            low[s] = min(low[s], low[v])
        else if v = parent[s]:
            low[s] = min(low[s], num[v])
```

The condition for an edge to be a cut-edge is:
CE low[Child] > num[Parent]

The runtime for the algorithm is O(|V | + |E|). The algorithm does DFS, and only visits every vertex and edges only once. At first, initializing visited[s], num[s], and low[s] will take O(|V|). The DFS traversal will take O(|V | + |E|).

(b)



Cut edges: b-g, b-d, d-c, c-e

Key:
low

num

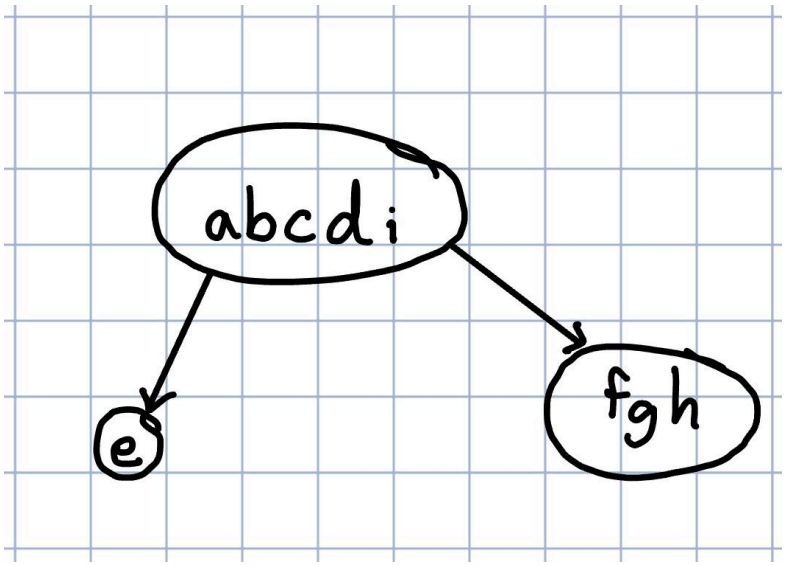
cut edges

Exercise 2

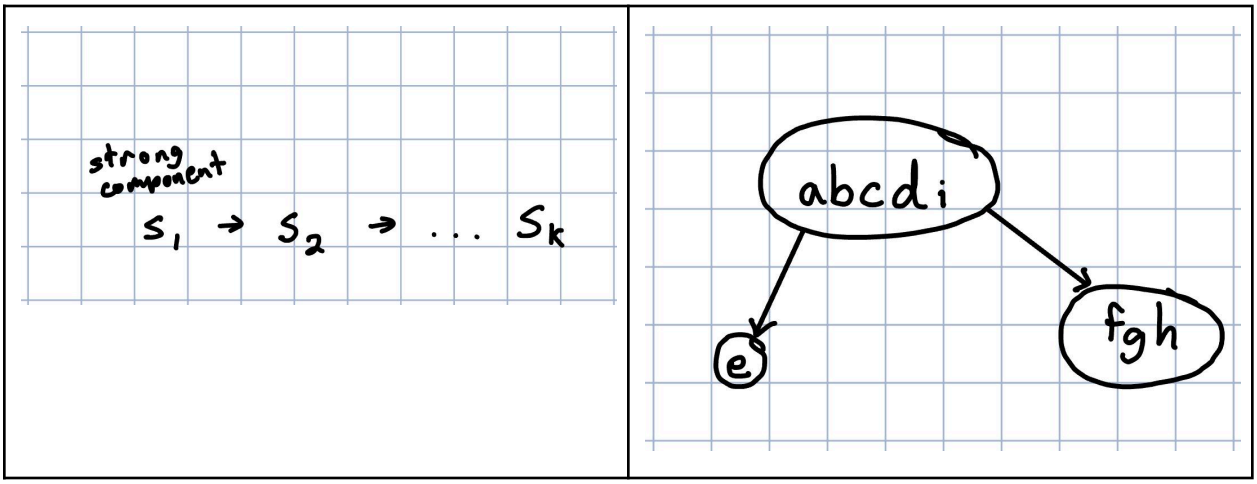
(1)

The strongly connected components of the direct graph are {a,b,c,d,i}, {e}, and {fgh}.

(2)



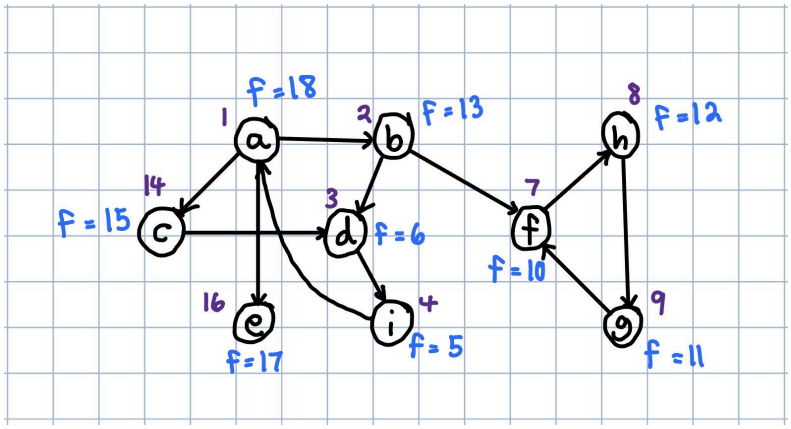
(3)



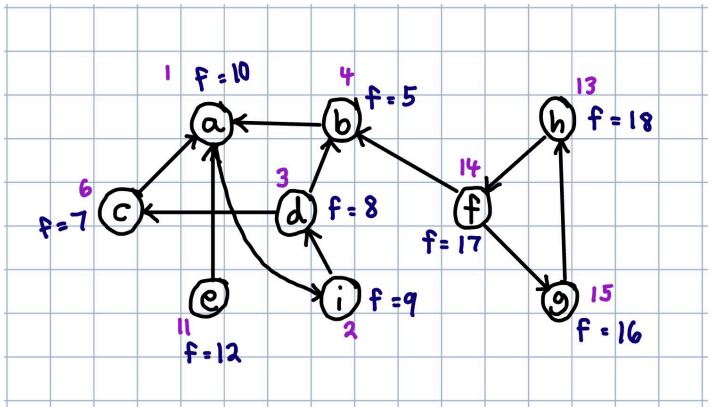
Strong component: maximal set of nodes - can't add more "nodes"
A DAG property is that at least one component finished before the next node.

(4)

1.



2/3.



(5)

Exercise 3

(a)

Kruskal's algorithm

Step 1: add a-b, cut: {a}, {b,c,d,e,f,g}

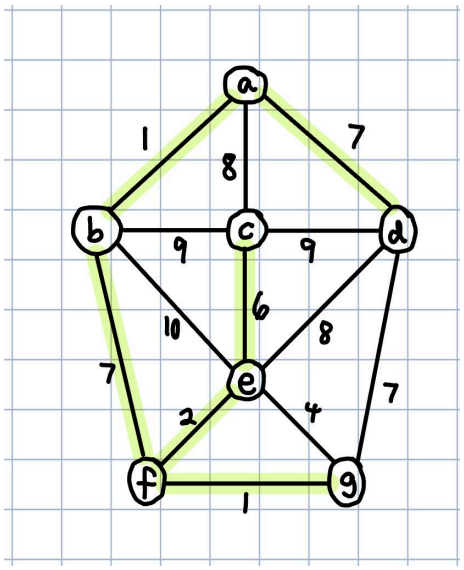
Step 2: add f-g, cut: {f}, {a,b,c,d,e,g}

Step 3: add e-f, cut: {e}, {a,b,c,d,f,g}

Step 4: add c-e, cut: {c}, {a,b,d,e,f,g}

Step 5: add a-d, cut: {d}, {a,b,c,e,f,g}

Step 6: add b-f, cut: {a,b}, {c,d,e,f,g}



(b)

Prim's algorithm

Step 1: add a-b, cut: {a}, {b,c,d,e,f,g}

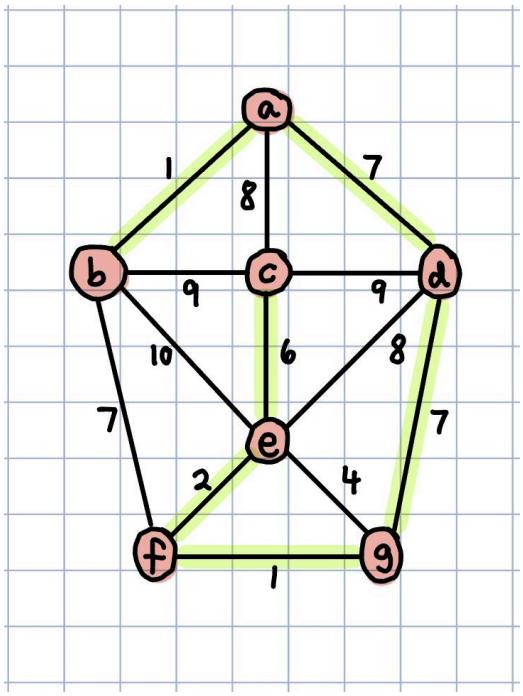
Step 2: add a-d, cut: {d}, {a,b,c,e,f,g}

Step 3: add d-g, cut: {d}, {a,b,c,e,f,g}

Step 4: add g-f, cut: {g}, {a,b,c,d,e,f}

Step 5: add e-f, cut: {e}, {a,b,c,d,f,g}

Step 6: add c-e, cut: {c}, {a,b,d,e,f,g}



(c)

Boruvka's algorithm
Step 1: add a-b, cut: {a}, {b,c,d,e,f,g}

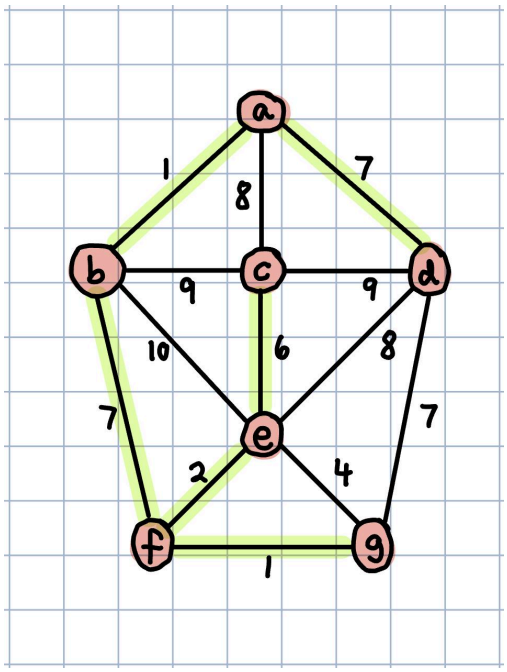
Step 2: add f-g, cut: {f}, {a,b,c,d,e,g}

Step 3: add e-f, cut: {e}, {a,b,c,d,f,g}

Step 4: add c-e, cut: {c}, {a,b,d,e,f,g}

Step 5: add a-d, cut: {d}, {a,b,c,e,f,g}

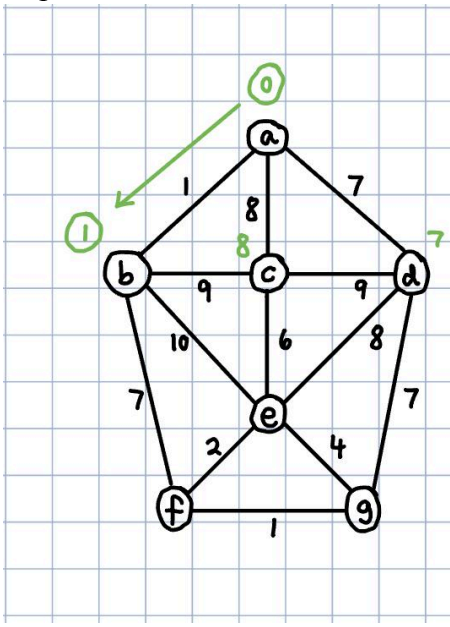
Step 6: add b-f, cut: {a,b}, {c,d,e,f,g}



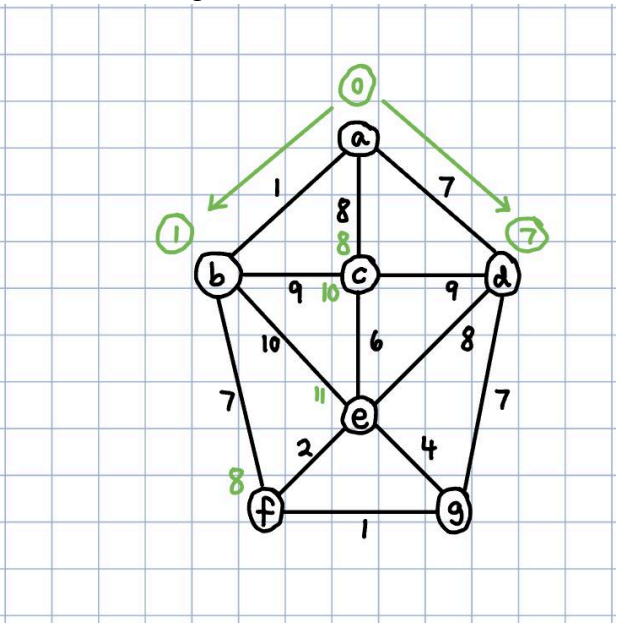
Exercise 4

(a)

Step 1: add a-b

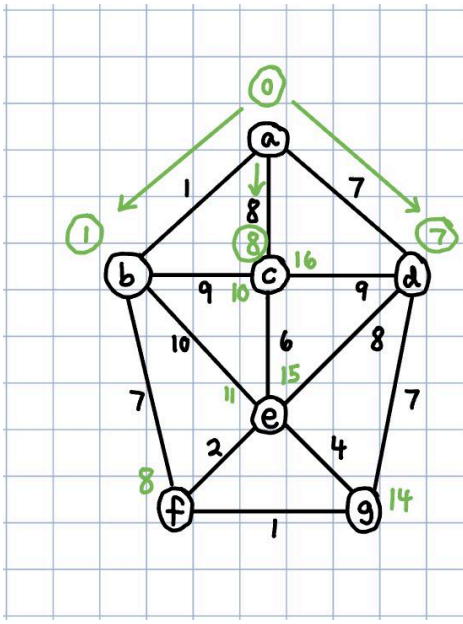


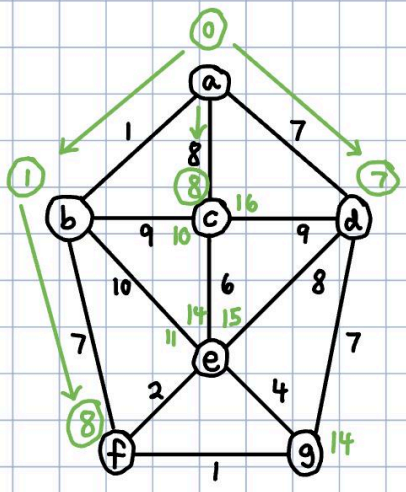
Step 2: add a-d



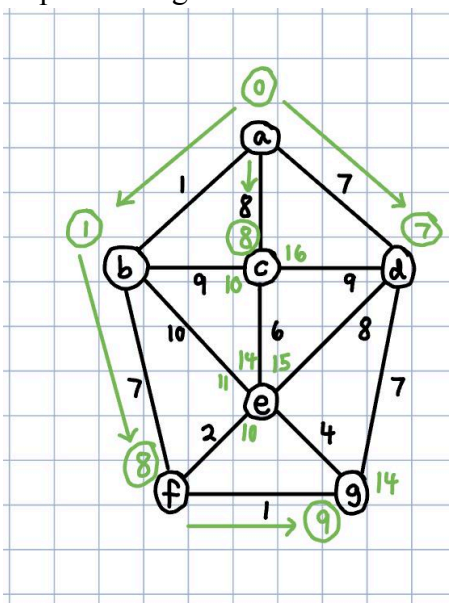
Step 3: add a-c

Step 4: add b-f

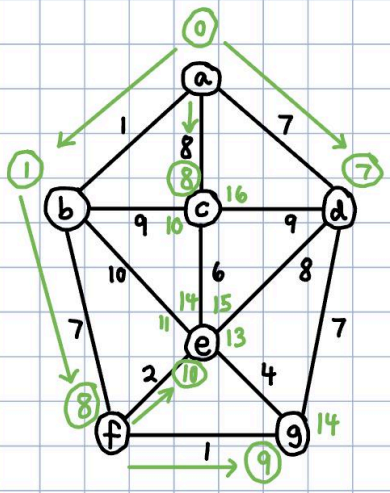




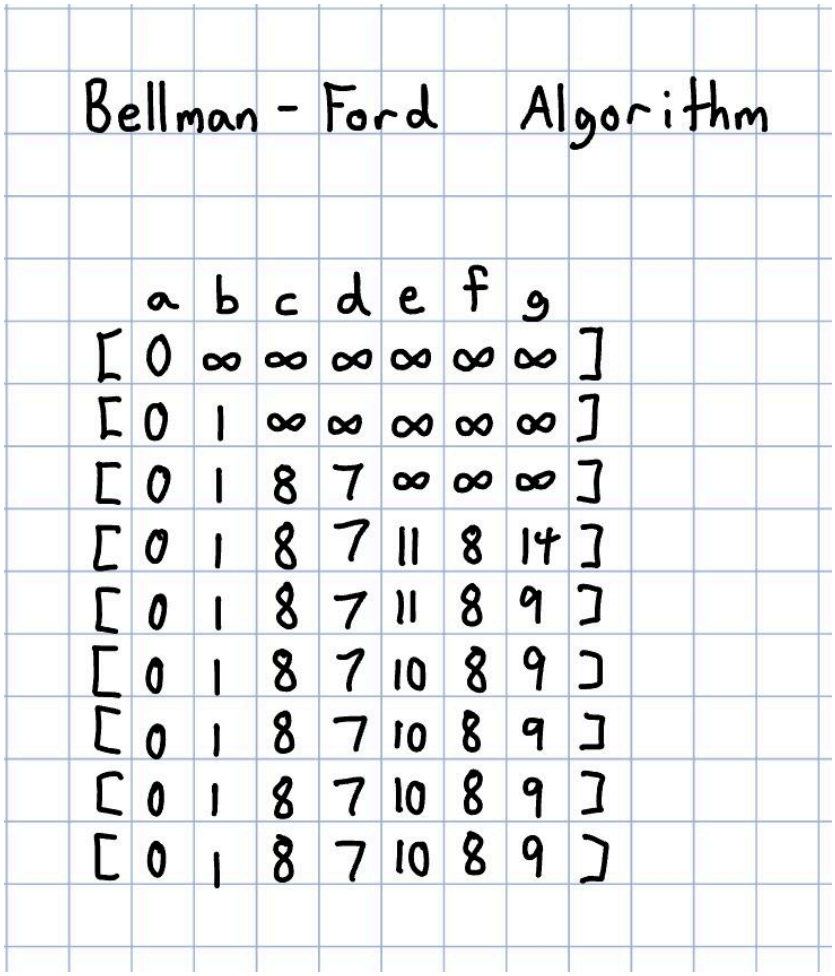
Step 5: add f-g



Step 6: add c-e



(b)



Exercise 5

(1)

Let longestPath(v) denote the longest path from s to v.

(2)

longestPath(v) = $max_{edge(u \to v)}$(longestPath(u) + weight(u, v) if no such nodes output null.

The base cases are:

longestPath(s) = 0

If v has no incoming edges, then we have null path

(3)

```
function Longest-Path(G, S):
    Longest-Path = [ -∞, -∞, -∞ ...] #length n
```

```
    Longest-Path[S] = 0
    Q = Queue(S)
    while len(Q) ≠ 0:
        v = Q.pop()
        for neighbor in neighbors(v):
            if Longest-Path[V] + w(w, neighbor) > Longest-Path(neighbor):
                Longest-Path[neighbor] = Longest-Path[v] + weight(v,
neighbor)
            Q.push(neighbor)
```

(4)
The runtime of the algorithm is O(n + m), where n is the number of the vertices and m is the number of edges in the graph. Every vertex in the graph is going to be visited and each edge is seen only once, which will take O(m). The algorithm is a similar implementation to BFS, so at most, the total runtime for the traversal will be O(n + m).

(5)
This algorithm only works for DAGs because directed acyclic graphs are topological sorted, so the order of the edges will go from left to right. Therefore the graph will have no cycles and the algorithm will work. Meanwhile, arbitrary directed graphs have cycles, so the algorithm will not work.