Exercise 5

Given Information:

Exactly 11 edges

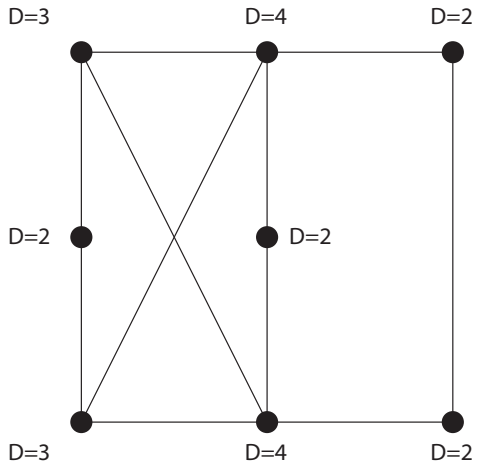
And *at least* 4 vertices with the following properties:

2 have deg of exactly 4
2 of the vertices have degree exactly 3
all other vertices have degree ≤ 2.

Using the handshaking lemma we get:

$$\sum \deg(v) = 2|E| = 2(11) = 22$$

Because 2 vertices have degree 3, there are (3 * 2 = 6) + (2 vertices with degree of 4 (2*4 = 8)). We have 22-(6 + 8) = 22 - 14 = 8 nodes left with a degree of ≤ 2. This requires at least a minimum of ($\left\lceil \frac{8}{2} \right\rceil = 4$) + 2 + 2 = 8 vertices to construct an undirected graph.



$|E| = 11$

$v = 8$

Exercise 6

(a)

Claim: for every unrooted tree T with n ≥ 6 nodes , if T contains $\lceil n/2 \rceil + 2$ leaves then T also contains at least one node of degree 4 or larger.

Proof:

Base:

$\left\lceil \frac{6}{2} \right\rceil + 2 = 5 - 1 = 4$ leaves with at least one internal node that has a degree of 4 and 4 leaves.

Induction Hypothesis, assume this is true:

{IH}= $\#l = \left\lceil \frac{k}{2} \right\rceil + 2 - 1$ for any $n \in \mathbb{N}$ and -1 denotes some internal node with degree >= 4

Step:

$$\#l = \left\lceil \frac{k+1}{2} \right\rceil + 2 - 1$$

$$= \left\lceil \frac{\left\lceil \frac{k}{2} \right\rceil + 2 - 1 + 1}{2} \right\rceil + 2 - 1 \quad \{\text{IH}\}$$

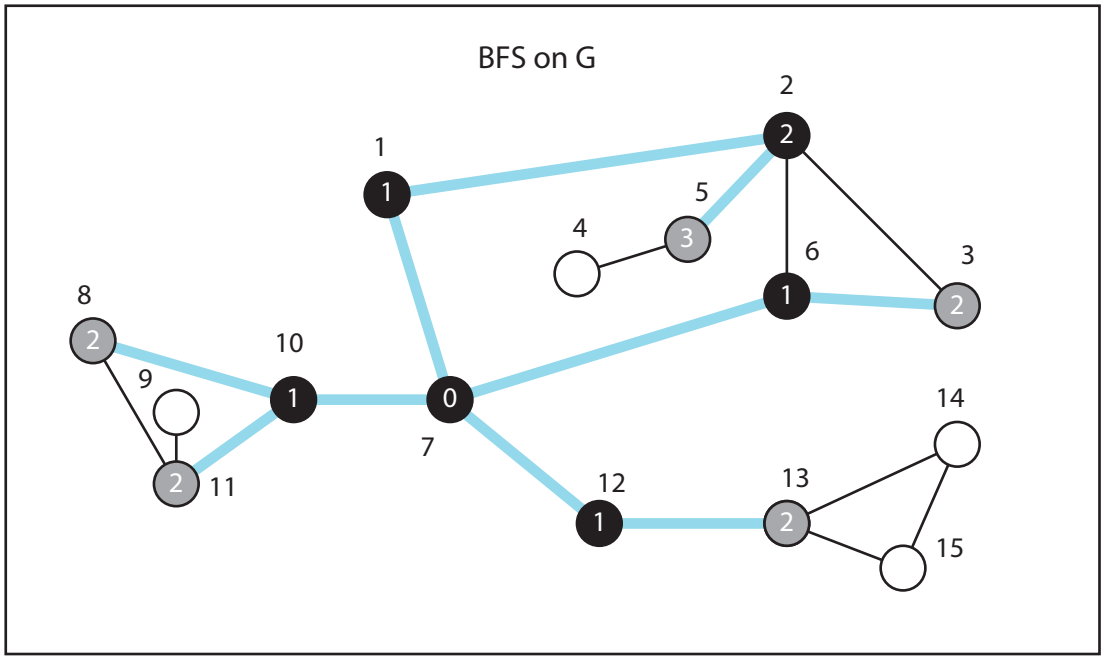
$$= \left\lceil \frac{\left\lceil \frac{k}{2} \right\rceil + 2 - 1 + 1}{2} \right\rceil + 2 - 1$$

$$= \left\lceil \frac{k}{2} \right\rceil + 2 - 1$$

Therefore, by induction we conclude that the claim: for every unrooted tree T with n ≥ 6 nodes for all n , if T contains $\lceil n/2 \rceil + 2$ -1 leaves, then T also contains at least one node of degree 4 or larger is sound.

Exercise 7

(a)

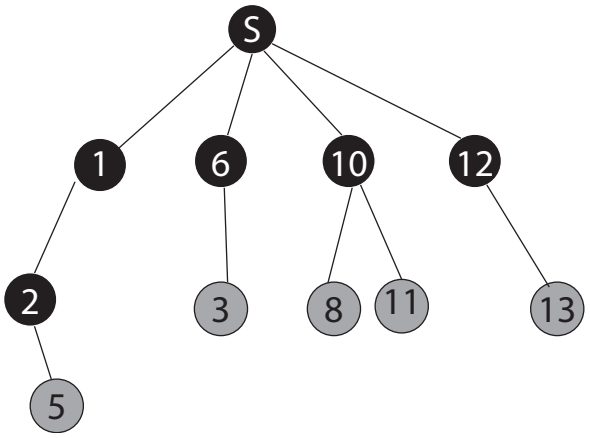


BFS on G

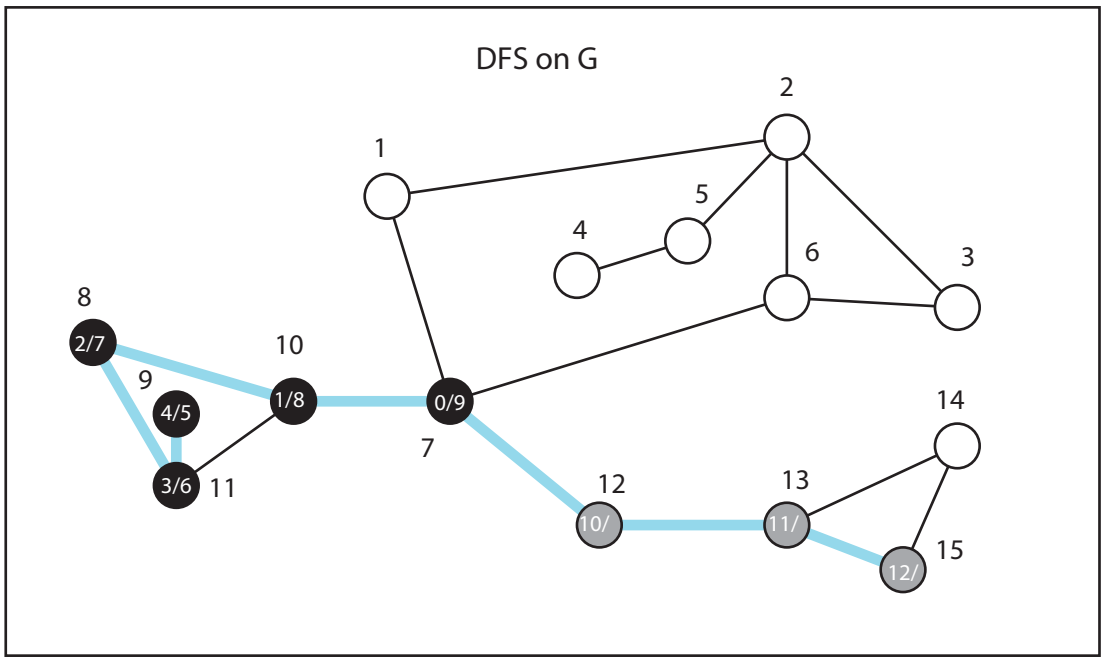
Adj[u] =

1:2,7
2:1,3,6,5
3:2,6
4:5
5:2
6:7,2,3
7:1,6,10,12
8:10,11
9:11
10:8,7,11
11:8,9,10
12:7,13
13:12,14,15
14:13,15
15:13,14

0:Q = S
1:Q = **1**,6,10,12
2:Q = **6**,10,12,2
3:Q = **10**,12,2,3
4:Q = **12**,2,3,8,11
5:Q = **2**,3,8,11,13
6:Q = 3,8,11,13,5



(b)



(b)

Adj[u] =

1:2,7
2:1,3,6,5
3:2,6
4:5
5:2,4
6:7,2,3
7:1,6,10,12
8:10,11
9:11
10:8,7,11
11:8,9,10
12:7,13
13:12,14,15
14:13,15
15:13,14

0:7
1:7,10
2:7,10,8
3:7,10,8,11
NOCALL: S = 7,10,8,11,9
NOCALL: S = 7,10,8,11
NOCALL: S = 7,10,8
NOCALL: S = 7,10
4: S = 7
5: S = 7,12
6: S = 7,12,13
7: S = 7,12,13,15

Exercise 8

(a)
Pseudo code:
Input: an array $S \subseteq V \setminus \{t\}$ and t as starting point
Output: shortest path measured in edges from a starting position to{t}

```
ModifiedBFS(G, S, t)
1:
2://Make every node u white
3: for each u ≠ t
4:      do  color[u] = white; d[u] =∞;π[u] = NIL
5://Mark every node u of the given array S as visited => black
6: for each  u  ∈  S
7:      do color[u] = black;
8:
9:  //Color starting point t gray
10: color[t] = gray; d[t] = 0; π[t] = NIL
11:
12: //Initialise the Q
13: Q ← ∅
14:
15: //Enqueue the Starting point
16: Enqueue(Q, t)
17:
18: //While the queue != empty
19: while Q ≠ ∅
20:
21:        //Dequeue first element and store in u
22:      do u = Dequeue(Q)
23:
24:            //For each v in the adjacency list
25:            for each v ∈ Adj[u]
26:
27:                 //If the colour is white
28:                 do if color[v] == white
29:
30:                                        //If the colour is white then:
31:                                        then
32:                                                  //Make it gray
33:                                                  color[v] = gray;
34:                                                  //Increment the distance of the node
35:                                                  d[v] = d[u]+1;
36:                                                  //Make the current node u a parent and store it in the Q
37:                                                  π[v] = u;
38:                                                  //Put the next v in the Q
39:                                                  Enqueue(Q, v)
40:
41:       /*
42:        * If we have found a node that has already been marked
43:        * as black then that => s. From that node it is the shortest path and thus the node we are looking for
44:        */
45:
46:                 else if color[v] == black
47:                 //Return the distance of the element + itself
48:                 return d[u] + 1
49:
50: //If it is not stored anymore in the Q because it was marked grey mark it as visited = > black
51: color[u] = black
```

Proof of correctness:

LI: for each iteration i the u = dequeue, processes each element of the graph only once.

Init: this is the trivial case. Before the first iteration t = grey. Since the root is enqueued and dequeued in Q and it is already grey, the node will turn black. The invariant is satisfied.

Main: in some iteration i, the loop processes some element and mark it as grey. After the latter iteration and we consider some element with distance n and some descendant with distance n' where n<n', then the n element must be black and is thus processed only once hence, the loop invariant doesn't change.

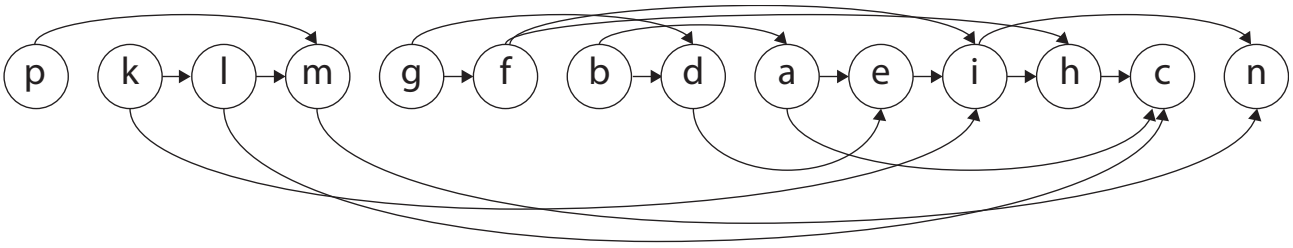
Term: the algorithm terminates when the Q is empty or when the shortest path has been found. If the algorithm in the worst case had to process each element, the algorithm correctly returns all elements which were processed by u only once and returns all the distance from t to the shortest distance. If the algorithm found the shortest path after some iterations that respects the loop invariant, it correctly returns the distance that we are looking for.

Running Time Analysis:

As given in the slides the worst case running time of BFS is $\Theta(1+|Adj[u]|) = O(V + E)$. It takes up to $\Theta(V)$ time to compute the distance and predecessor for each node. Each node is enqueued only once and each edge is visited twice and hence we get O(V+E). The extra operations on line 46 and 48 that are in the while loop are running in constant time and doesn't affect the asymptotic running time. In total we get $\Theta(V+E)$ since best case Ω still needs to do a look up in Adj list and still needs to enqueue a node once.

Exercise 9

(a)
In order to construct a linear ordering a topological sort using DFS starting on node P, has been performed on the given DAG.



(b)

Max distance for each vertex v: