Homework 9 Solutions

Problem 1:

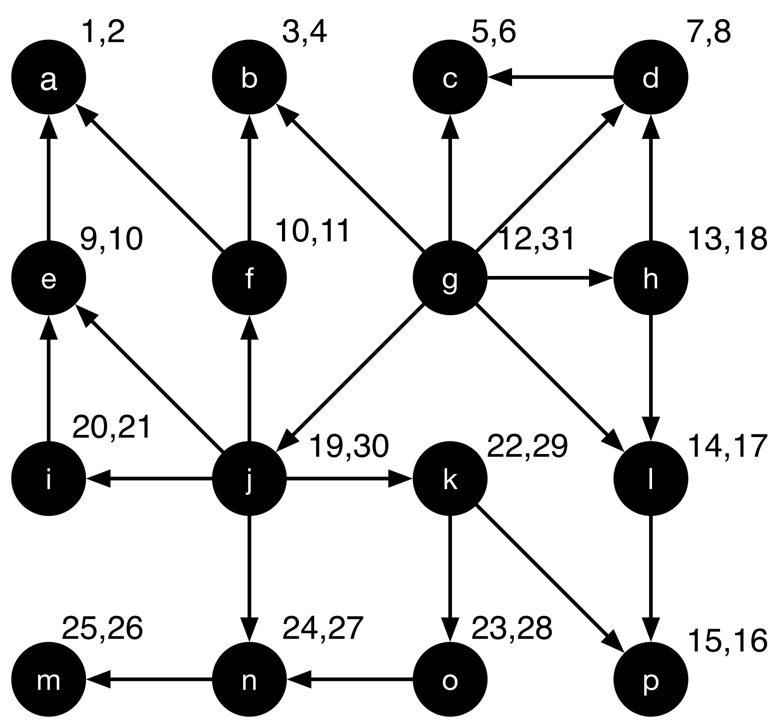
We first create a list or dictionary of empty adjacency lists. This is mandatory for any algorithm and takes time $\Theta(|V|)$. Then we go through the list or dictionary of adjacency lists. The lists are indexed by the source node and are a list of target nodes. For each target node, we add the source node to the adjacency list we just created. Here is a Python implementation which assumes that the adjacency lists are organized as a Python dictionary.

```
def opposite(adjlist):
    result = {node: [] for node in adjlist.keys()}
    for source in adjlist.keys():
        for target in adjlist[source]:
            result[target].append(source)
    return result
```

The second step processes each edge exactly once. This is the minimum work that needs to be done and takes $\Theta(|E|)$. This gives a total runtime for this algorithm and the minimum of any algorithm to $\Theta(|V| + |E|) = \Theta(\max(|E|, |V|)$.

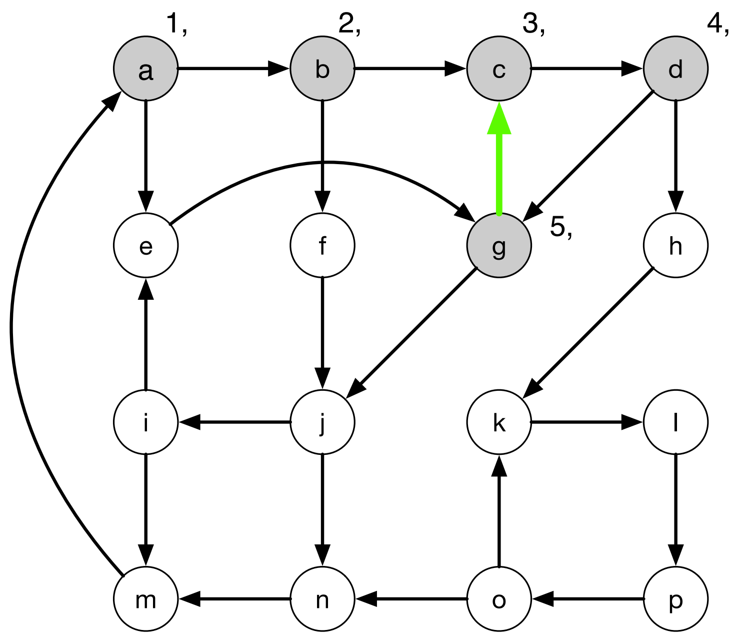
Problem 2:

We first run DFS trying to discover a back-edge. In the first part, there is none. A DFS run (they are not unique, but this one uses alphabetic order) gives



Ordering by reversed finishing time, we get   g, j, k, o, n, m, i, h, l ,p f, e, d, c, b, a for a topological sort.

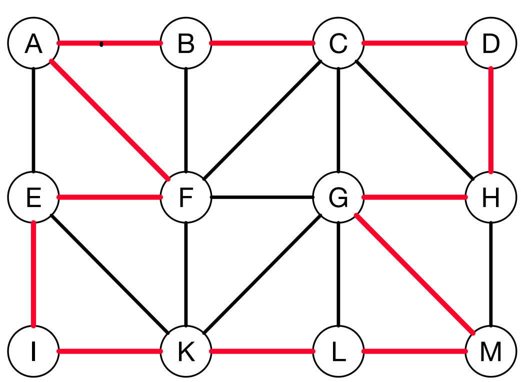
For the second part, we again to a DFS. There are many different ways to do a DFS, again, we use alphabetic order to break ties.



When we reach node g, we find a gray node among the adjacent nodes, so we now have a back-edge, indicated in green in the figure above. This gives us the cycle g-c-d.

Problem 3:

There are a number of Hamiltonian circuits in this graph. Here is one:



Problem 4:

We first count the number of edges between two groups of k-subsets. A single node is connected to k others, so that there are $k \times k$ edges between two groups. There are $\binom{n}{2}$ pairs of k-subsets, for a total for $\frac{k^2 n(n-1)}{2}$ edges.