Name: Keaton Whitehead
ID: 104668391

CSCI 3104 **Profs. Grochow & Layer**
Problem Set 8 **Spring 2019, CU-Boulder**

Quick links 1 2 3a 3b 3c 4

1. (10 pts) Ginerva Weasley is playing with the network given below. Help her calculate the number of paths from node 1 to node 14.

   Hint: assume a "path" must have at least one edge in it to be well defined, and use dynamic programming to fill in a table that counts number of paths from each node j to 14, starting from 14 down to 1.

$$\begin{tikzcd}
1 \arrow[r] \arrow[dr] & 2 \arrow[r] \arrow[d] \arrow[dr] & 3 \arrow[r] \arrow[dr] & 4 \arrow[r] \arrow[d] \arrow[dr] & 5 \arrow[r] \arrow[dr] & 6 \arrow[d] & \\
 & 7 \arrow[r] & 8 \arrow[r] & 9 \arrow[r] \arrow[dr] & 10 \arrow[r] \arrow[dr] & 11 \arrow[d] \arrow[dr] & \\
 & & & & 12 \arrow[r] & 13 \arrow[r] & 14
\end{tikzcd}$$

   To calculate the number of possible paths taken we will need to use the recursive Depth-First Search Algorithm below:

```
$pathnum = 0$
$PathCount(node, nList[], targetNode):$
    $if(node == targetNode):$
        $pathnum = pathnum + 1$
    $else\ if(nList[] == Null):$
        $return$
    $else:$
        $for\ neighbor\ in\ nList:$
            $PathCount(neighbor, neighbor.nList, targetNode)$
    $returnpathnum$
```

   The idea of this algorithm is that we are going to do a recursion call for every single node and try to reach the target node as many times as possible without going over the same path twice, and returning how many paths we have found. It works by first checking if the current node is the targetNode, if it is then add 1 to the variable for counting paths. If not, then check to see if this node is a leaf node, by checking if the neighbors list is empty. If it is then just return so that the next recursion can be called on the next neighbor of the parent node. If not, the move on the the for loop where we call the recursive function on all of the neighbors surrounding the current node. Then after this program has run its course it will finally return the number of paths that have reached the target node.

Name: $\boxed{\text{Keaton Whitehead}}$
ID: $\boxed{\text{104668391}}$

CSCI 3104 **Profs. Grochow & Layer**
Problem Set 8 **Spring 2019, CU-Boulder**

2. (10 pts) Ginny Weasley needs your help with her wizardly homework. She's trying to come up with an example of a directed graph $G = (V, E)$, a start vertex $s \in V$ and a set of tree edges $E_T \subseteq E$ such that for each vertex $v \in V$, the unique path in the graph (V, E_T) from s to v is a shortest path in G, yet the set of edges E_T cannot be produced by running a depth-first search on G, no matter how the vertices are ordered in each adjacency list. Include an explanation of why your example satisfies the requirements.

$\boxed{\text{idk}}$

Name: Keaton Whitehead
ID: 104668391

CSCI 3104 **Profs. Grochow & Layer**
Problem Set 8 **Spring 2019, CU-Boulder**

3. Prof. Dumbledore needs your help to compute the in- and out-degrees of all vertices in a directed multigraph G. However, he is not sure how to represent the graph so that the calculation is most efficient. For each of the three possible representations, express your answers in asymptotic notation (the only notation Dumbledore understands), in terms of V and E, and justify your claim.

   (a) (5 pts) An *adjacency matrix* representation. Assume the size of the matrix is known.

```
Dijkstra(G[V][V],S):
    Q = Priority queue
    initialize Parent[V]
    initialize Dist[V]
    for i = 1 to —V—
        Dist[i] = INTMAX
        Parent[i] = null
        Q.push(i)
    Dist[S] = 0
    while —Q— ¿ 0
        N = Q.pop()
        for each neighbor in N:
            alternateDist = Dist[N] + W(C,neighbor)
            if alt > Dist[neighbor]
                Dist[neighbor] = alt
                Parent[neighbor] = N
```

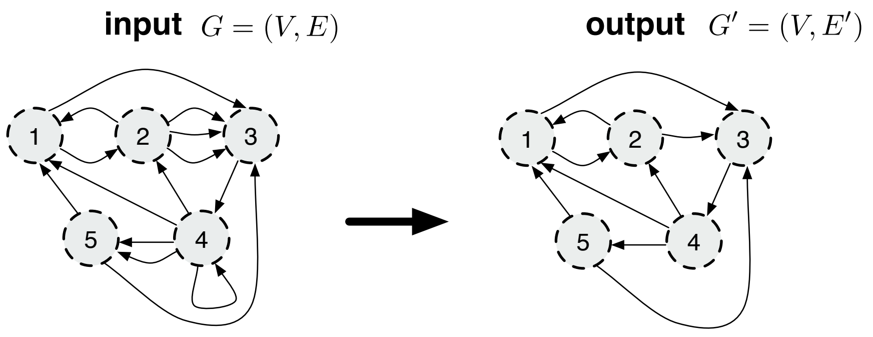
(b) (5 pts) An *edge list* representation. Assume vertices have arbitrary labels. $\boxed{\text{IDK}}$

(c) (5 pts) An *adjacency list* representation. Assume the vector's length is known.
$\boxed{\text{IDK}}$

Name: $\boxed{\text{Keaton Whitehead}}$
ID: $\boxed{104668391}$

CSCI 3104 **Profs. Grochow & Layer**
Problem Set 8 **Spring 2019, CU-Boulder**

4. (25 pts) Deep in the heart of the Hogwarts School of Witchcraft and Wizardry, there lies a magical grey parrot that demands that any challenger efficiently convert directed multigraphs into directed *simple* graphs. If the wizard can correctly solve a series of arbitrary instances of this problem, the parrot will unlock a secret passageway.



1	3	2			
2	1	3	3	3	
3	4				
4	4	5	2	5	1
5	3	1			

1	2	3	
2	1	3	
3	4		
4	1	2	5
5	1	3	

An example of transforming $G \to G'$

Let $G = (E, V)$ denote a directed multigraph. A directed simple graph is a $G' = (V, E')$, such that E' is derived from the edges in E so that (i) every directed multi-edge, e.g., $\{(u, v), (u, v)\}$ or even simply $\{(u, v)\}$, has been replaced by a single directed edge $\{(u, v)\}$ and (ii) all self-loops (u, u) have been removed.

Describe and analyze an algorithm (explain how it works, give pseudocode if necessary, derive its running time and space usage, and prove its correctness) that takes $O(V + E)$ time and space to convert G into G', and thereby will solve any of the parrot's questions. Assume both G and G' are stored as adjacency lists.

Hermione's hints: Don't assume adjacencies `Adj[u]` are ordered in any particular way, and remember that you can add edges to the list and then remove ones you don't need.

$\boxed{\text{IDK}}$