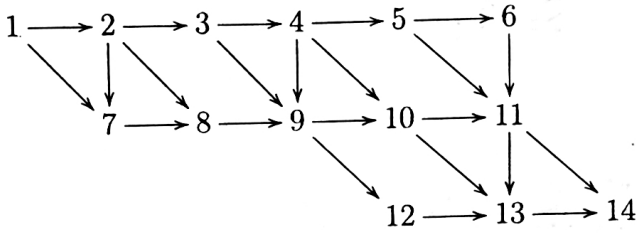
Name: Jon Abrahamson
ID: 107084898

CSCI 3104 **Profs. Grochow & Layer**
Problem Set 7 **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.



(1) 6→14 = 2 paths (6-11-14 & 6-11-13-14)
(2) 5→14 = (1) +5 = 7 paths
(3) 4→14 = (2)+7 = 14 paths
(4) 3→14 = (3)+4 = 18 paths
(5) 2→14 = (4)+8 = 26 paths
(6) 1→14 = (5)+4 = 30 paths

∴ From node 1 to node 14 there are 30 paths

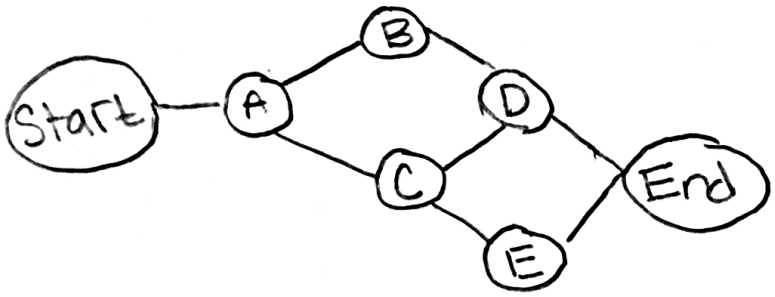
Name: Jan Abrahamson

ID: 107084898

CSCI 3104
Problem Set 7

Profs. Grochow & Layer
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.



For DFS, the algorithm would mark nodes of start → A → B → D → End as visited. After node C the algorithm would see that node d was visited and continue to node e without going on the edge between C → d.

Name: Jon Abramson
ID: 107084898

CSCI 3104 **Profs. Grochow & Layer**
Problem Set 7 **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.

The graph should be a V×V matrix in which you can represent an edge between vertices with either a 1 or 0

$O(V^2)$

(b) (5 pts) An *edge list* representation. Assume vertices have arbitrary labels.

The graph is represented by a list a tuples $[(v_1, v_2), (v_2, v_3), \ldots]$ where each tuple is an edge connecting two vertices

We make a search tree for each edge to represent vertices connected to it

$\therefore O(E \log(V))$

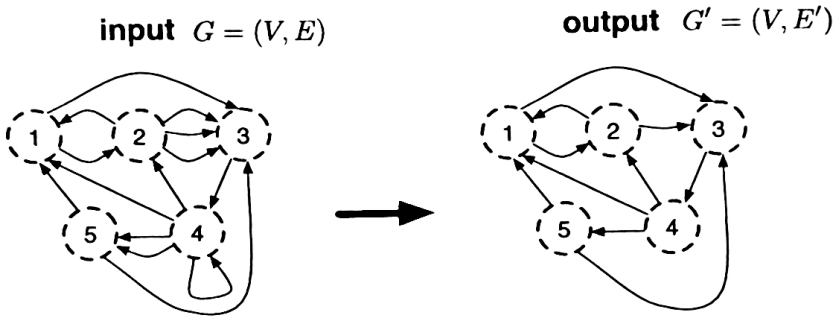
(c) (5 pts) An *adjacency list* representation. Assume the vector's length is known.

The graph is represented by a list where each V in the list has a linked list where other vertices are connected to it by an edge

A search algorithm would run through V number of vertices and E edges in two different arrays

∴ O(V+E)

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.

Name: Jon Abrahamson
ID: 107084898

CSCI 3104
Problem Set 7

Profs. Grochow & Layer
Spring 2019, CU-Boulder

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.

Algorithm Description

(1) Loop through the adj list by i, where variable "head" represents the adjList[i]

(2) For each "head" in list, go through the linked list of edges where "valve" represents the current edge

(3) if the "head" does not equal "valve" and "valve" has not been seen yet, then
- Mark the valve as visited
- Add the valve to the correct position in a new adjacency list

(4) Repeat the process until every "head" has been evaluated and the new adjacency List is built

Psuedo

```
num_v = N

For each vertex i
    bool vis[N] = false

traverse adj[i]
if vertex j in adj[i] && j==i
    remove j
if vis[j]==false
    vis[j]=true
if vis[j]==true
    remove j

return adj
```

Running Time

Since we traverse through vertices V and evaluate E edges, we have:

$$O(V+E)$$

to account for the size of the new adjacency List

Proof of Correctness

The algorithm essentially condenses the original graph into a new transposed graph. Correctness relies on eliminating self-loops & repeats from the original graph.

In the 3rd step of the algorithm description, only edges that are not equal to the head and have not been repeated can be added to the new adjacency list. Since only these operations are performed to change the graph, the new graph retains its validity and is therefore correct.