CSCI 3104 Spring 2018 Problem Set #8
Alison Ostlund (05/22)
Worked with: Kyle Ward, Hannah Haines, Jason Lubrano

---

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.

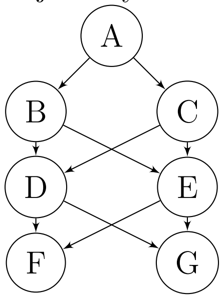
$$\begin{tikzcd}
1 \arrow[r] \arrow[dr] & 2 \arrow[r] \arrow[d] \arrow[dr] & 3 \arrow[r] \arrow[d] & 4 \arrow[r] \arrow[d] \arrow[dr] & 5 \arrow[r] \arrow[d] \arrow[dr] & 6 \arrow[d] & \\
 & 7 \arrow[r] & 8 \arrow[r] & 9 \arrow[r] \arrow[dr] & 10 \arrow[r] \arrow[dr] & 11 \arrow[d] \arrow[dr] & \\
 &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

Using the paths from the previous node can sum up those values to find the value of the current node. This is dynamic because of this fact and you need to count the number of paths of the nodes neighbors and store them in order to get the path value for the current node. For example, to get the 14th node you have

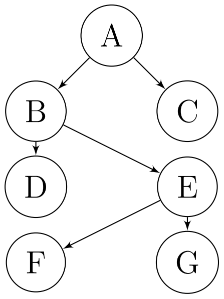
Node	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Prev node paths	13+11	12+11+10	9	10+6+5	9+5+4	8+4	7+3+2	2+1	5	4	3	2	1	0
Paths	30	21	5	9	7	5	4	2	1	1	1	1	1	0

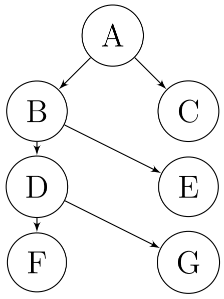
2. (10 pts) Ginny Weasley needs your help with her wizardly homework. Shes trying to come up with an example of a directed graph $G = (V, E)$, a start vertex $v \in V$ and a set of tree edges $E_T \subseteq E$ such that for each vertex $v \in V$, the unique path in the graph

(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.

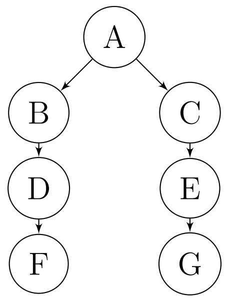


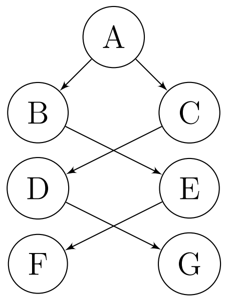
Performing a DFS on graph shown above will give you a variety of different graphs. Such as:





Here are examples of two graphs that DFS will never be able to produce due to its algorithms behavior. This is due to the fact that DFS searches as deep as possible. While going down deep in the graph DFS visits every child on its way back to where it originally came from. On top of that it will never visit something that it has already been marked as visited and it will never check all the children of visited vertex.





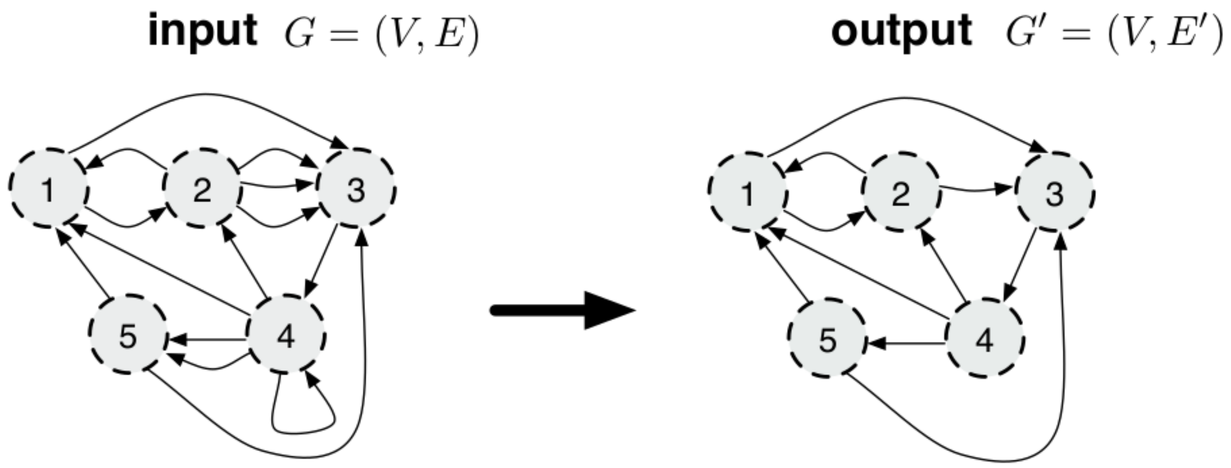
3. (15 pts) Prof. Dumbledore needs your help to compute the in-and out-degrees of all vertices in a directed multigraph G. However, he is not sure how to represent the graph so that the calculation is most efficient. For each of the three possible representations, express your answers in asymptotic notation (the only notation Dumbledore understands), in terms of V and E,and justify your claim.

   (a) *An **adjacency matrix** representation. Assume the size of the matrix is known.*
   $\Theta(V^2)$ - This is because an adjacency matrix will always have V rows and V columns making it a V x V matrix. You will go through the entire matrix to sum up all the ins and outs for each vertex making itV^2. It has to be a theta bound due to the fact that there will never be more or less V's than given.

   (b) *An **edge list** representation. Assume vertices have arbitrary labels.*
   $O(E\log(E))$ - In order to get the edges in the list yoou have to go through the entire edge list which takes E time. Every time you hit a new E in the list you have to store and keep track of the in and out degrees. This is done by keeping trak of the vertexs whom have been visited and to this you need to traverse down a search tree data structure to ensure you do not look at the same vertex twice. This takes an addition of $\log(E)$ time. Due to this the total time is E * $\log(E)$.

   (c) *An **adjacency list** representation. Assume the vectors length is known.*
   O(V+E) - An adjacency list has a linked list of vertices's and edges, which have to be traversed through. For each vertex, there is another linked list of edges, which are that certain vertex edges. This causes a size of E in the worst case because the worst case is that one vertex holds all edges(E). This gives the complexity stated above.

4. (30 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.
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 multiedge, 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.



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 $G \to G'$

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 Sphinxs questions. Assume both G and G' are stored as adjacency lists.
Hermiones hints: Dont 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.

```
newAdj =[]
neighbors = []
for V in verts:
   while E in edges:
        if (E not = V and A[E] not = V) \\ not visited or self loop
            A[E] = V
            newAdj[V].add(E)
        E = next E in list
 return new Adj
```

This algorithm will make a new adjacency list using an array. It then keeps track of the edges it has already visited using the array A[] .It will not add the current edge its on if it has already been visited or if it has a self loop. The array then gets written over as the algorithm iterates over the vertices's in the list.

Proof:
The array A[] is used determine if you've already visited and recored an edge for 2 vertices's and if the edge is a self loop. If either are true the the algorithm will not add the edge to the adjacency list.

Loop invariant:
Any edge in the new adjacency list is not the same or is a self loop.

Loop maintenance:
For any edge in the old adjacency list array, if it is a similar edge or has a self loop, it will not get added it to the new list.

Termination:
When the algorithm goes through the whole adjacency list, it will have not added any self loops or repeated edges to the new adj list.

The runtime for this algorithm is $O(v + E)$ because it has to go through t all the vertices's (V) in the adjacency list. It will look through the vertices's and a E number

of edges.
The space is also $\Theta(V + E)$ because it makes a new list of size V and also a new adjacency list with V vertices's and E edges. This turns to be 2V + E which can be translated to V + E.

5. (15 pts extra credit) Professor McGonagall has provided the young wizard Ron with three magical batteries whose sizes are 42, 27, and 16 morts, respectively. (A mort is a unit of wizard energy.) The 27-mort and 16-mort batteries are fully charged (containing 27 and 16 morts of energy, respectively), while the 42-mort battery is empty, with 0 morts. McGonagall says that Ron is only allowed to use, repeatedly, if necessary, the **mort transfer spell** when working with these batteries. This spell transfers all the morts in one battery to another battery, and it halts the transfer either when the source battery has no morts remaining or when the destination battery is fully charged (whichever comes first). McGonagall challenges Ron to determine whether there exists a sequence of mort-transfer spells that leaves exactly 12 morts either in the 27-mort or in the 16-mort battery.

   (a) Ron knows this is actually a graph problem. Give a precise definition of how to model this problem as a graph, and state the specific question about this graph that must be answered.

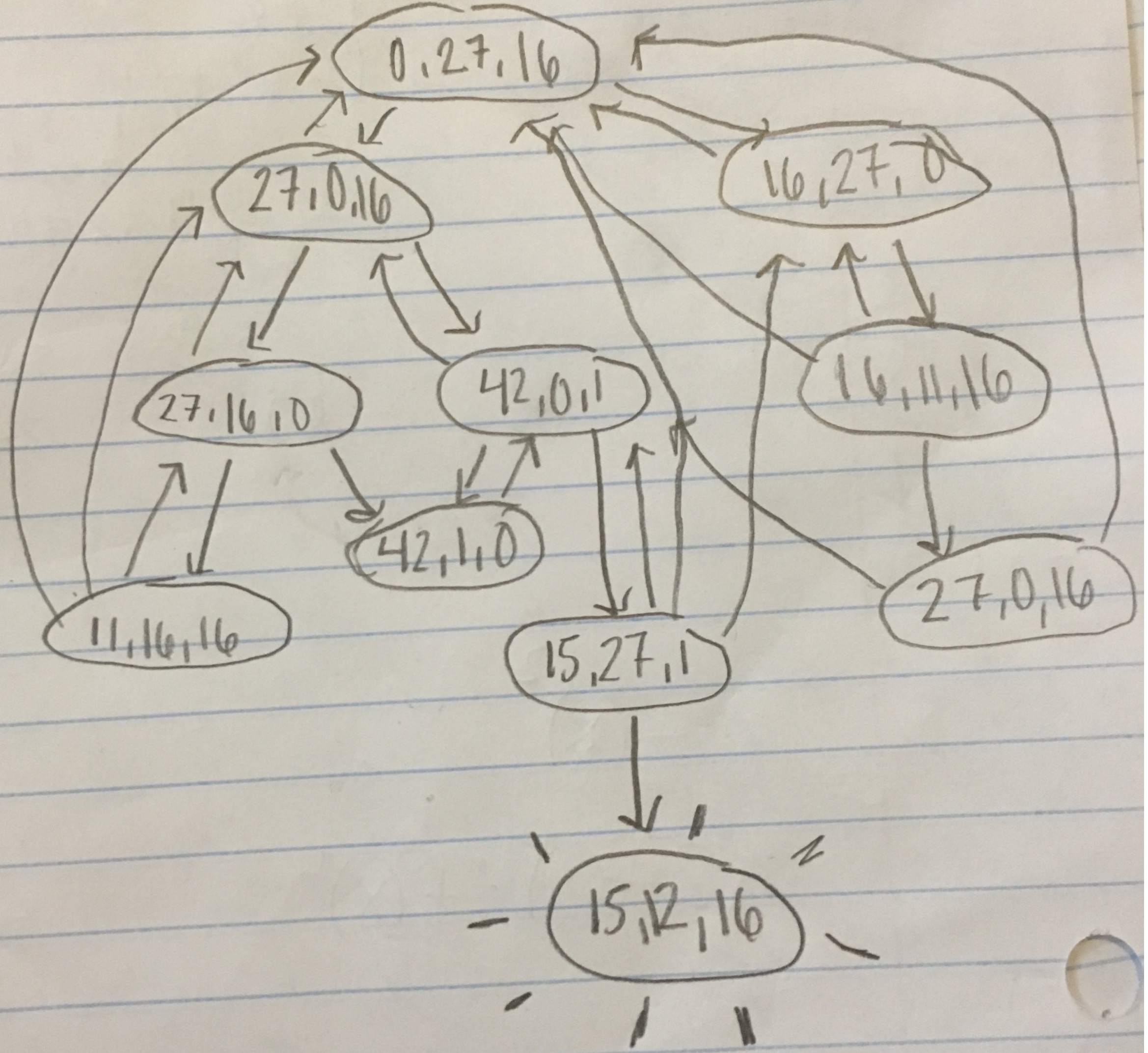
   To model this problem, you will need to use a graph search. First, using the original given mort values and setting that as a vertex. From there, using the mort transfer spell as the edges, can create connecting vertices's, that symbolize another set of possible mort combinations with different mort values that can be created from the original mort batteries. From there you can keep applying the mort transfer spell to each vertex to transfer morts, until eventually getting down to the goal value of 12 morts in the 27 battery or 16 battery.

   (b) What algorithm should Ron apply to solve the graph problem?

   Ron should apply a BFS graph search. This uses layers to see if there is a path along the graph that gives a vertex with the wanted mort values, so therefore would find the values where 12 is in either the 27 or 16 mort battery.

(c) Apply that algorithm to McGonagalls question. Report and justify your answer.

e


0,27,16
27,0,16
16,27,0
27,16,0
42,0,1
16,11,16
42,1,0
11,16,16
15,27,1
27,0,16
15,12,16

First start with the original batteries of 16,27, and 42. We are given that the batteries start with 42:0 morts 27:27 morts and 16: 16 morts. From here we can apply the mort transfer spell on this vertex of batteries. This allows us to get differnt mort values in each battery. From the orignal vertex we can transfer 27 morts into the 42 or transfer 16 into the 42 leaving the 16 battery empty. From here I decided to go down the left side of the graph which is the 42:27, 27:0, 16:16 batteries. From there we can use the spell to come up with more combinations of the batteries. To get 12 morts in the 27 how eve, we have to transfer the 16 morts in the 16 battery into the 42 battery. Which leaves 42 in the 42 , 0 in the 27 and 1 in the 16. Using the mort transfer, we get another set of vertices's, one of which being the combination of transferring the 42 into the 27, leaving 15 morts in the 42, 27 full, and still 1 in the 16. After one final transfer spell we get the wanted batteries. with 15 in the 42, 12 in the 27, and 16 in the 16.