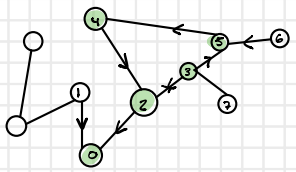
3. Given connected graph $G=(V,E)$ such that $n=|V|$ and $m=|E|$.

a. We note that DFS can be used to efficiently find cycles and so modify DFS to return the first cycle encountered or "no cycle" if no cycle is found.

```
modifiedDFS(s):
  Initialize S to be a stack with one element s
  Initialize Parent to be an empty list
  Initialize Cycle to be an empty list
  While S is not empty
    Take node u from S
    If Explored[u] = false then
      Set Explored[u] = true
      For each edge (u,v) incident to u
        Add v to the stack S
        Add (u,v) to Parent[v] = u
      EndFor
    Else if Explored[u] = true then   //cycle detected
      Add u to Cycle
      p = Parent[u]
      While (p != u)
        Add p to Cycle
        p = Parent[p]
      EndWhile
      Add p to Cycle
      Return Cycle
    EndIf
  EndWhile
  Return "no cycle"
```



Stack

0	2	1	4	3	5	3	3	6	2	7
	1		3	1	3	1	6	3	7	6
			1		1		3	1	6	3
							1		3	1
									1	

Parents

index	0	1	2	3	4	5	6	7	..
value	0	0	0	2	2	4	5	3	..
				3					
				5					

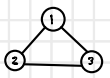
Claim: modifiedDFS terminates in $O(m+n)$.
Proof: Because we are adding and removing from the stack, we let n_v denote the degree of vertex v. Because we break early if v is encountered twice we know the while loop runs at most $\sum_{v=1}^{n} 2n_v = 2m$ times (when there are no cycles). Thus, because we visit every node at most once if there are no cycles, we perform at most $O(mnn)$ work over the outer while and inner for loop. The inner loop adds at most additional n work but, because $O(2n) \Rightarrow O(n)$, the bound $O(m+n)$ stands. □

Claim: modifiedDFS returns the vertices of a cycle if one exists and "no cycle" otherwise.
Proof: Because we're visiting vertices in stack order (as opposed to say traversing a linked list repeatedly) we expect to visit each node only once if there are no cycles. Therefore, if we encounter a node u twice, we know u is part of a cycle. Thus, a path exists from u through u's parents to u again. modifiedDFS simply returns this backwards trace of u's parents which is a cycle. Note that modifiedDFS can also return 'no cycle' if a vertex u is never encountered twice and thus, no cycle exists. □

b. Claim: Any connected graph G that isn't already a spanning tree will have at least 3 spanning trees.
Proof:
Let G be an arbitrary connected graph with $m \geq n$ where m is the number of edges and n is the number of vertices.
Knowing that $m \geq n$ where $m=|E|$ and $n=|V|$, tells us that there is at least one cycle in the given graph G. (Since we know that if G didn't have a cycle, and since G is connected, that G would then be a tree, but trees have $m=n-1$.) Because we have a cycle, there must be a path in that cycle with at least 3 edges where the start and end vertices are the same. By the definition of spanning tree at least one of those edges can be removed to form a spanning tree. Since there are at least 3 edges and at least one of any of them can be removed then it follows that any of them could be removed individually creating at least 3 distinct spanning trees. □

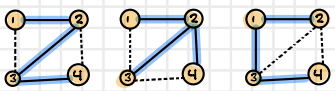
c. We know how to find a cycle from a. So, we can extend that algorithm in a to find a cycle in graph G and from there find multiple spanning trees of G.

```
MinSpanTrees(s):
  Initialize SpanTrees to be a list
  Assign cycle = modifiedDFS(s) of G where modifiedDFS defined above
  If cycle exists
    For each edge e between vertex pairs (u_i, u_i+1) in cycle
      Remove e from G
      Assign BFStree = BFS(s) of G where BFS defined by Kleinberg and Tardos
      Add BFStree to SpanTrees
      Replace e in G
    EndFor
  EndIF
  Return subset of SpanTrees S such that |S|=3 and each minimum spanning tree s∈S is unique
```



Smallest graph w/ spTree. Any 1 edge can be removed leaving 3 possible trees.

No loop ⇒ No spTree can't remove edge



Claim: MinSpanTrees terminates on connected graph G in $O(m+n)$.
Proof: Consider that we know modifiedDFS terminates in $O(m+n)$. And within the for loop, we have exactly m $O(m+n)$ in running BFS such that the for loop runs in $O(m+n)$, we drop the m coefficient because that doesn't affect asymptotic runtime. Thus, because modifiedDFS is $O(m+n)$ and the for loop containing BFS is also $O(m+n)$, and because both are the same order of growth, we have $O(m+n)$ for the entire algorithm. □

Claim: Given a connected graph that is not already a minimum spanning tree, MinSpanTrees returns exactly 3 minimum spanning trees of G.
Proof: We know that we can use BFS to find a minimum spanning tree of arbitrary graph G. However there is no assurance that the resultant spanning tree will be distinct across multiple applications of BFS on the same G. But, we know that cycles always have $3 \leq k \leq m$ edges such that at least one of the k edges k_i can be removed from G for $G'=G-k_i$ such that G' is not disconnected and such that the minimum spanning tree T' of G' will be $T' \subseteq G'$. Because k_i will not, by definition, be in T' and there are at least k distinct spanning trees of G, there can always be found and returned at least 3 distinct minimum spanning trees of G. □