There are 2 questions for a total of 10 points.

---

1. (3 points) Consider the adjacency list of a directed graph G given below:

A	: B,	D	
B	: C,	D,	E
C	: F		
D	: H		
E	: A,	H	
F	: I,		
G	: D		
H	: G,	F,	I
I	: H		

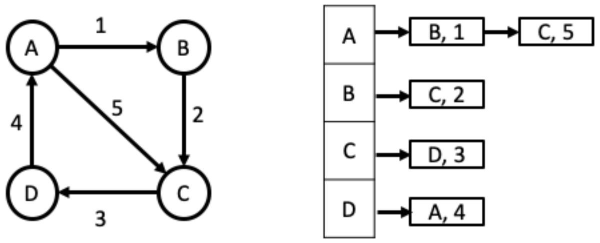
State the order in which the vertices of the graph get marked visited when `explore(`G, A`)` is executed.

Solution: A, B, C, F, I, H, G, D, E.

2. (7 points) (*Network maintenance*) You are given a weighted, directed graph $G = (V, E)$ for a communications network where nodes are computational devices and directed edges represent communication links. The weight of edges are positive integers and denote the cost of maintaining the link. Assume that all edge weights are distinct for this problem. You as a communications engineer are told to examine the possibility of reducing the overall network maintenance cost by throwing out the edge with the largest maintenance cost. However, you need to make sure that removing the heaviest edge does not disrupt the communication between two special nodes s and t. That is, you need to check whether after removing the heaviest edge, there is still a path from s to t and a path from t to s.

Design an algorithm for this problem. You are given as input the adjacency list of a weighted, directed graph $G = (V, E)$ and two nodes $s, t \in V$. Your algorithm should output "yes" if after removing the heaviest edge, there is a path from s to t and t to s, and "no" otherwise. Discuss the running time of your algorithm.

(*We use a weight function $w(.)$ to denote weights of edges. This weight function can be specified within the adjacency list of the graph. For node i with an outgoing neighbor j, the weight $w((i,j))$ of edge (i,j) can be stored in the linked list for i within entry j. See example below.*)



Solution: We first need to find the heaviest edge in the graph. This can be done by making a pass over the adjacency list. Let (i, j) be the heaviest edge. We can construct the graph G' by removing the edge (i, j) from G. Now, we run explore twice on G', once with s as the starting vertex and check if t is reachable and once with t as the starting vertex and check if s is reachable. If both these succeed, then we output "yes", otherwise we output "no". This idea is summarised in the following pseudocode.

```
PossibleConnect($G = (V, E), s, t$)
  - $max = 0$
  - For $i = 1$ to $|V|$:
      - For every neighbor $j$ of $i$:
          - If $(w((i, j)) > max)$
              - $max = w((i, j))$; $e = (i, j)$
  - Let $G' = (V, E - \{e\})$
  - If ($t$ gets marked visited on running explore($G', s$)) and
    ($s$ gets marked visited on running explore($G', t$))
      - return("yes")
  - else
      - return("no")
```

Running time: Finding the heaviest edge involves going over the adjacency list of the graph which is $O(|V| + |E|)$ since the size of the adjacency list is $O(|V| + |E|)$. Then the algorithm runs explore twice on a graph with $|V|$ vertices and at $|E| - 1$ edges. Each of these calls cost $O(|V| + |E|)$ time. So, the overall running time of this algorithm is $O(|V| + |E|)$.