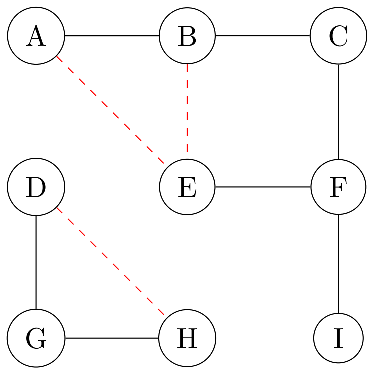
Homework 4

Name:	Rohan Karamel
NetID:	rak218
Course:	Algorithms Section 2
Instructor:	Professor Mario Szegedy
Date:	April 9, 2024

Problem 3.1. Perform a depth-first search on the following graph; whenever there's a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge or back edge, and give the pre and post number of each vertex.

Solution. The graph is shown below with the pre and post numbers of each vertex and the classification of each edge.

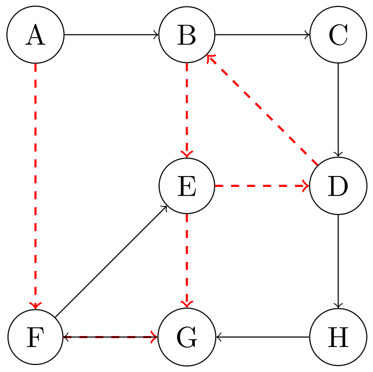


Vertex	Pre	Post
A	1	12
B	2	11
C	3	10
D	13	18
E	5	8
F	4	9
G	14	17
H	15	16
I	6	7

Edge	Classification
AB	Tree
AE	Back
BC	Tree
BE	Back
CF	Tree
DH	Back
DG	Tree
FE	Tree
FI	Tree
GH	Tree

Problem 3.2. Perform a depth-first search on the following graph; whenever there's a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge or back edge, and give the pre and post number of each vertex.

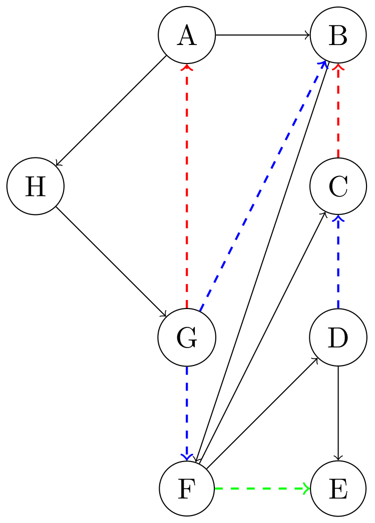
Solution. (a) The graph is shown below with the pre and post numbers of each vertex and the classification of each edge.



Vertex	Pre	Post
A	1	16
B	2	15
C	3	14
D	4	13
E	8	9
F	7	10
G	6	11
H	5	12

Edge	Classification
AB	Tree
AF	Back
BC	Tree
BE	Back
CD	Tree
DB	Back
DH	Tree
ED	Back
EG	Back
FE	Tree
FG	Back
GF	Tree
HG	Tree

Solution. (b) The graph is shown below with the pre and post numbers of each vertex and the classification of each edge.

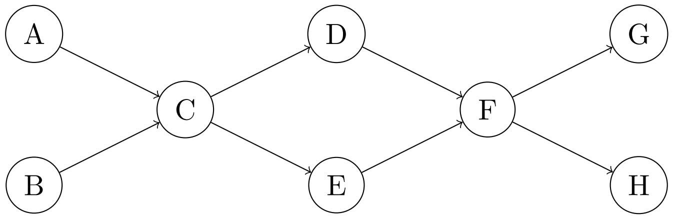


Vertex	Pre	Post
A	1	16
B	2	11
C	4	9
D	5	8
E	6	7
F	3	10
G	13	14
H	12	15

Edge	Classification
AB	Tree
AH	Tree
HG	Tree
BF	Tree
FC	Tree
FD	Tree
DE	Tree
GA	Back
CB	Back
GB	Cross
GF	Cross
DC	Cross
FE	Forward

Problem 3.3. Run the DFS-based topological ordering algorithm on the following graph. Whenever you have a choice of vertices to explore, always pick the one that is alphabetically first.

Solution. (a) Indicate the pre and post numbers of the nodes. The graph is shown below with the pre and post numbers of each vertex.



Vertex	Pre	Post
A	1	14
B	15	16
C	2	13
D	3	10
E	11	12
F	4	9
G	5	6
H	7	8

Solution. (b) What are the sources and sinks of this graph? The sources of the graph are the vertices with no incoming edges. In this case, the sources are vertices A and B. The sinks of the graph are the vertices with no outgoing edges. In this case, the sinks are vertices G and H.

Solution. (c) Give the topological ordering of the vertices. The topological ordering of the vertices is A, B, C, D, E, F, G, H.

Solution. (d) How many topological ordering does this graph have? The graph has only $2^3 = 8$ topological orders. This is because the sources A and B can be placed in any order, as well as D and E, and the same goes for the sinks G and H.

Problem 3.5. The reverse of a directed graph G = (V,E) is another directed graph $G^R = (V, E^R)$ on the same vertex set, but with all edges reversed; that is, $E^R = \{(v,u) : (u,v) \in E\}$. Give a linear-time algorithm for computing the reverse of a graph in adjacency list format.

Solution. The algorithm for computing the reverse of a graph in adjacency list format is given below.

Algorithm 1 Reverse Graph

```
function ReverseGraph($G$)
    $G^R \leftarrow$ Empty Adjacency List
    for $u \in G.V$ do
        $G^R[u] \leftarrow$ Empty List
    end for
    for $u \in G.V$ do
        for $v \in G.Adj[u]$ do
            $G^R[v]$.append($u$)
        end for
    end for
    return $G^R$
end function
```

The algorithm first initializes an empty adjacency list G^R for the reverse graph. Then, for each vertex u in the original graph, it initializes an empty list in the adjacency list of G^R. Finally, for each edge (u, v) in the original graph, it appends u to the adjacency list of v in G^R. The algorithm runs in $O(V + E)$ time, where V is the number of vertices and E is the number of edges in the graph. The correctness of the algorithm follows from the fact that for each edge (u, v) in the original graph, the edge (v, u) is added into the reverse graph. The time complexity stems from the two loops, one for each vertex in the graph, and the other for each edge in the graph. Don't be frightened from the double for loop as it is only getting the adjacent vertices of each vertex in the graph.

Problem 3.9. For each node u in an undirected graph, let twodegree[u] be the sum of the degrees of u's neighbors. Show how to compute the entire array of twodegree[·] values in linear time, given a graph in adjacency list format.

Solution. The algorithm for computing the entire array of twodegree values in linear time is given below.

The algorithm first initializes an empty array twodegree for the twodegree values. Then, for each vertex u in the graph, it initializes the twodegree value of u to 0. Finally, for each edge (u, v) in the graph, it adds the length of the adjacency list of v to the twodegree value of u. The algorithm runs in $O(V + E)$ time, where V is the number of vertices and E is the number of edges in the graph. The correctness of the algorithm follows from the fact that for each edge (u, v) in the graph, the degree of v is added to the twodegree value of u. The time complexity stems from the two loops, one for each vertex in the graph, and the other for each edge in the graph. The algorithm is linear in time because it only traverses the adjacency list of each vertex once.

Algorithm 2 Compute Two Degree

```
1:  function ComputeTwoDegree($G$)
2:      twodegree ← Empty Array
3:      for $u \in G.V$ do
4:          twodegree[$u$] ← 0
5:      end for
6:      for $u \in G.V$ do
7:          for $v \in G.Adj[u]$ do
8:              twodegree[$u$] ← twodegree[$u$] + len($G.Adj[v]$)
9:          end for
10:     end for
11:     return twodegree
12: end function
```

The correctness of the algorithm follows from the fact that for each edge (u, v) in the graph, the degree of v (length of linked list) is added to the twodegree value of u. The time complexity stems from the two loops, one for each vertex in the graph, and the other for each edge in the graph. The algorithm is linear in time because it only traverses the adjacency list of each vertex once.