Homework 7

Yu Hsiao Yu-Hsuan Wu

Question 1

1. (7.23) Describe an efficient implementation of the algorithm discussed in class (as by-product of an inductive proof) for finding an Eulerian circuit in a graph. The algorithm should run in linear time and space. (Hint: try to interweave the discovery of a cycle and that of the separate Eulerian circuits in the connected components with the cycle removed, in the induction step.)

Question 1

[Theorem] An undirected connected graph has an Eulerian circuit if and only if all of its vertices have even degrees.

$\Rightarrow$ If any degree of $v \in V$ is odd or zero for graph $G(V, E)$, then no Eulerian circuit can be found in G.

We can check if all of the vertices have even degrees first.

Question 1

If all of the vertices have even (non-zero) degrees, we can find an Eulerian circuit by following steps:

1. Initialize a vertex path stack and an edge path stack.
2. Choose a starting point u randomly.
3. Do $\textsc{Euler}(u)$:

   While exist unmarked edge $\{u, v\} \in E$:

   Mark edge $\{u, v\}$.

   Do $\textsc{Euler}(v)$.

   Push edge $\{u, v\}$ into the edge path stack.

   Push vertex u into the vertex path stack.

Question 1

Finally, we can check if there exists any disconnected graph and return the result.

1. Check if there is any disconnected graph by checking if all the edges have been marked.
2. Choose one of them to return:
   - Pop and return all the vertices in the vertex path stack as an Eulerian circuit.
   - Pop and return all the edges in the edge path stack as an Eulerian circuit.

Question 1

(Note: Choose either the red or the blue ones.)

Algorithm FindEulerianCircuit($G(V, E)$)
  if any degree of V is odd or zero **then**
    return "Can't find Eulerian circuit!";
  end if
  initialize a vertex path stack;
  initialize an edge path stack;
  pick a vertex $u \in V$;
  Euler(G, u);
  for all edges $\{u, v\} \in E$ **do**
    if edge $\{u, v\}$ is unmarked **then**
      return "Can't find Eulerian circuit!";
    end if
  end for
  pop and return all the vertices in the vertex path stack;
  pop and return all the edges in the edge path stack;
end Algorithm

Question 1

(Note: Choose either the red or the blue ones.)

Algorithm EULER($G(V, E)$, u)
  while exist unmarked edge $\{u, v\} \in E$ **do**
    mark edge $\{u, v\}$ from G;
    EULER (G, v);
    push edge $\{u, v\}$ into the edge path stack;
  end while
  push vertex u into the vertex path stack;
end Algorithm

Since each edge is gone through once, the time complexity is $O(|E|)$, and since there will be at most $|V|$ vertices in the vertex path stack and at most $|E|$ edges in the edge path stack, the space complexity is $O(|V|)$ or $O(|E|)$.

Question 2

2. (7.28) A **binary de Bruijn sequence** is a (cyclic) sequence of 2^n bits $a_1 a_2 \cdots a_{2^n}$ such that each binary string s of size n is represented somewhere in the sequence; that is, there exists a unique index i such that $s = a_i a_{i+1} \cdots a_{i+n-1}$ (where the indices are taken modulo 2^n). For example, the sequence 11010001 is a binary de Bruijn sequence for $n = 3$. Let $G_n = (V, E)$ be a directed graph defined as follows. The vertex set V corresponds to the set of all binary strings of size $n-1$ ($|V| = 2^{n-1}$). A vertex corresponding to the string $a_1 a_2 \cdots a_{n-1}$ has an edge leading to a vertex corresponding to the string $b_1 b_2 \cdots b_{n-1}$ if and only if $a_2 a_3 \cdots a_{n-1} = b_1 b_2 \cdots b_{n-2}$. Prove that G_n is a directed Eulerian graph, and discuss the implications for de Bruijn sequences.

Question 2

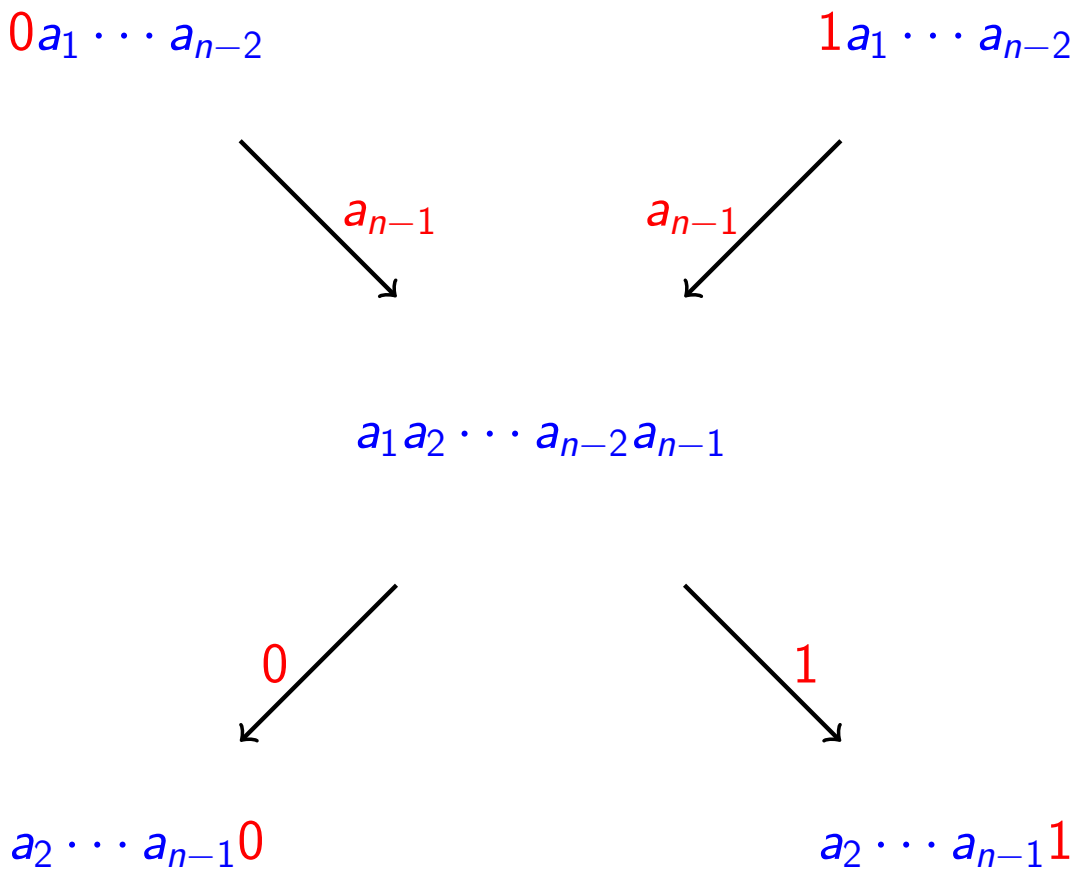


Figure 1: Edge construction of G_n. (There may be self-loops.)

Question 2

A directed Eulerian graph is a directed graph with an Eulerian circuit.
$\Rightarrow$ every vertex has equal indegree and outdegree.

Proof:

1. Indegree:
   For each vertex $v = a_1 a_2 \cdots a_{n-2} a_{n-1}$,
   there are 2 edges pointing to v:
   from vertices $0 a_1 a_2 \cdots a_{n-2}$ and $1 a_1 a_2 \cdots a_{n-2}$ respectively.
   $\Rightarrow$ The indegree is 2.
2. Outdegree:
   For each vertex $v = a_1 a_2 \cdots a_{n-2} a_{n-1}$,
   there are 2 edges pointing from v:
   from $a_2 a_3 \cdots a_{n-1} 0$ and $a_2 a_3 \cdots a_{n-1} 1$ respectively.
   $\Rightarrow$ The outdegree is 2.

Question 2

Implications:

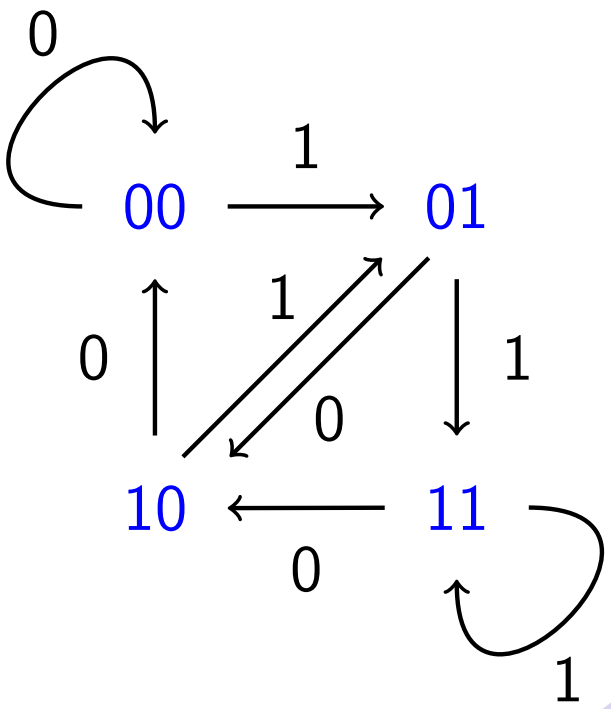
In a **binary de Bruijn sequence** with 2^n bits, all the continuous bit strings with the size of n are actually all the possible combinations of n bits.
For example, when $n = 3$, the sequence 11010001 contains:

$$\{110, 101, 010, 100, 000, 001, 011, 111\}$$

Going through an Eulerian circuit from any vertex of G_n will obtain a possible **binary de Bruijn sequence** with 2^n bits.

Question 2

In the figure (G_n for $n = 3$) below, we can obtain the sequence 11010001 by going through an Eulerian circuit from vertex 01.
All the possible n-bit strings will appear exactly once in Eulerian circuit because an n-bit string is composed of a vertex and an edge come out from it.
For example, the bit string 101 comprises 10 and $\xrightarrow{1}$.



Question 3

3. In the topological sorting algorithm that we discussed in class for directed acyclic graphs, DFS is used to calculate the indegree of each vertex in the input graph. Please give a detailed description of this calculation in adequate pseudocode. You need to define a main routine which invokes the DFS procedure with suitable preWORK and postWORK.

Question 3

```
function DFS(G, v)
    mark v;
    perform preWORK on v;
    for all edges (v, w) do
        if w is unmarked then
            DFS (G, w);
        end if
        perform postWORK on (v, w);
    end for
end function
```

Question 3

```
function DFS(G, v)
    mark v;
    v.indegree := 0;          // initialize in-degree before traversing
    for all edges (v, w) do
        if w is unmarked then
            DFS(G, w);
        end if
        w.indegree := w.indegree + 1;  // increment in-degree by 1
    end for                            // for all v's neighbors
end function
```

Question 3

To handle multiple vertices with in-degree $= 0$, we use a while loop to find the vertices that have not been calculated.

function CALCULATE INDEGREE(G)
    while there is an unmarked vertex v **do**
        DFS(G, v);
    end while
end function

Question 4

4. (7.3) Given as input a connected undirected graph G, a spanning tree T of G, and a vertex v, design an algorithm to determine whether T is a valid DFS tree of G rooted at v. In other words, determine whether T can be the output of DFS under some order of the edges starting with v. The running time of the algorithm should be $O(|V| + |E|)$.

Question 4

All DFS trees T of an undirected graph $G = (V, E)$ satisfy the following property:

$\forall e = \{v, w\} \in E$, one of the following statements must hold:

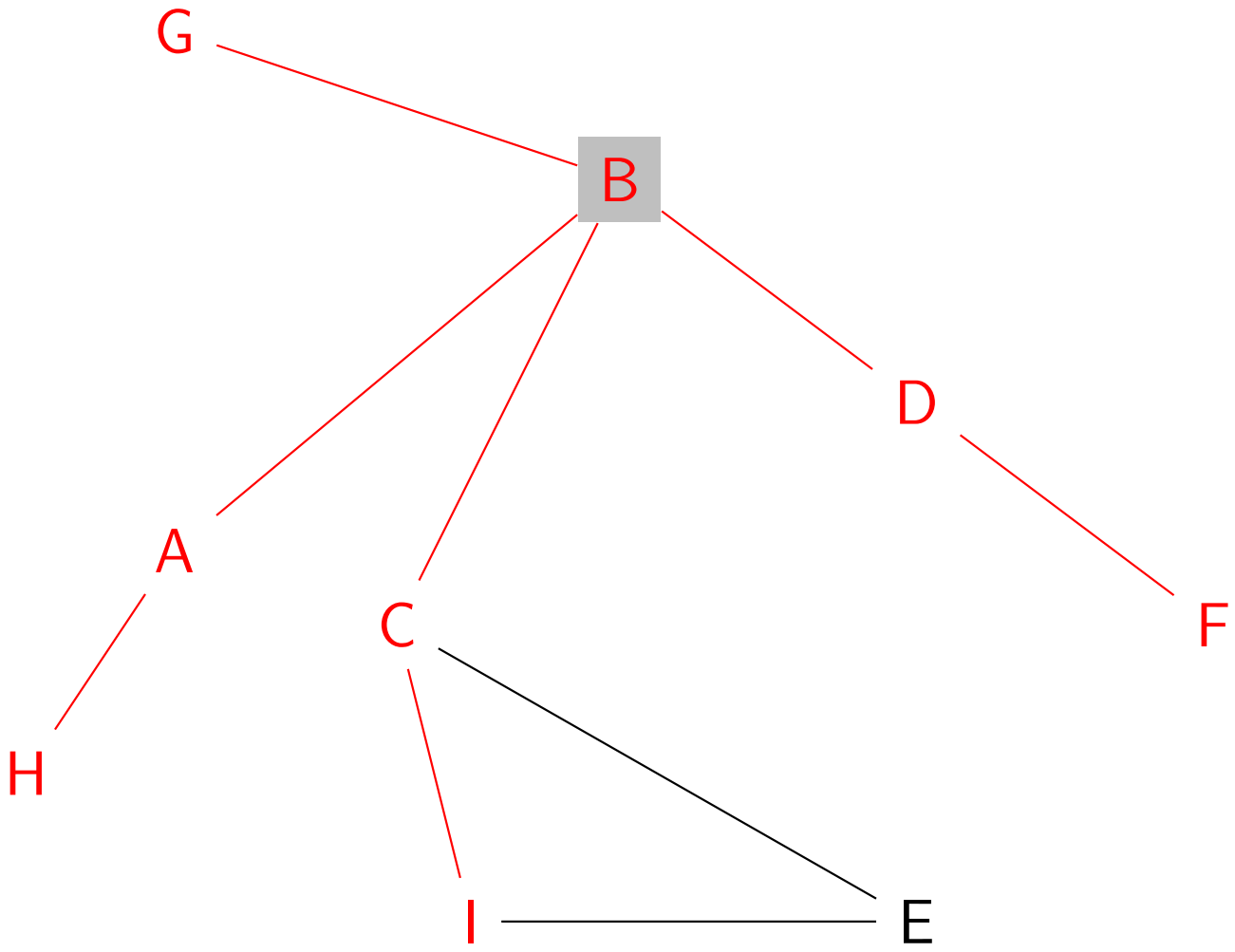
- v is an ancestor of w in T
- w is an ancestor of v in T

In short, DFS tree must cover all vertices in G and avoid cross edges in DFS tree.

The following page shows an example of graph G. Let red part be the input spanning tree T, and the vertex with light gray background color be input vertex v.

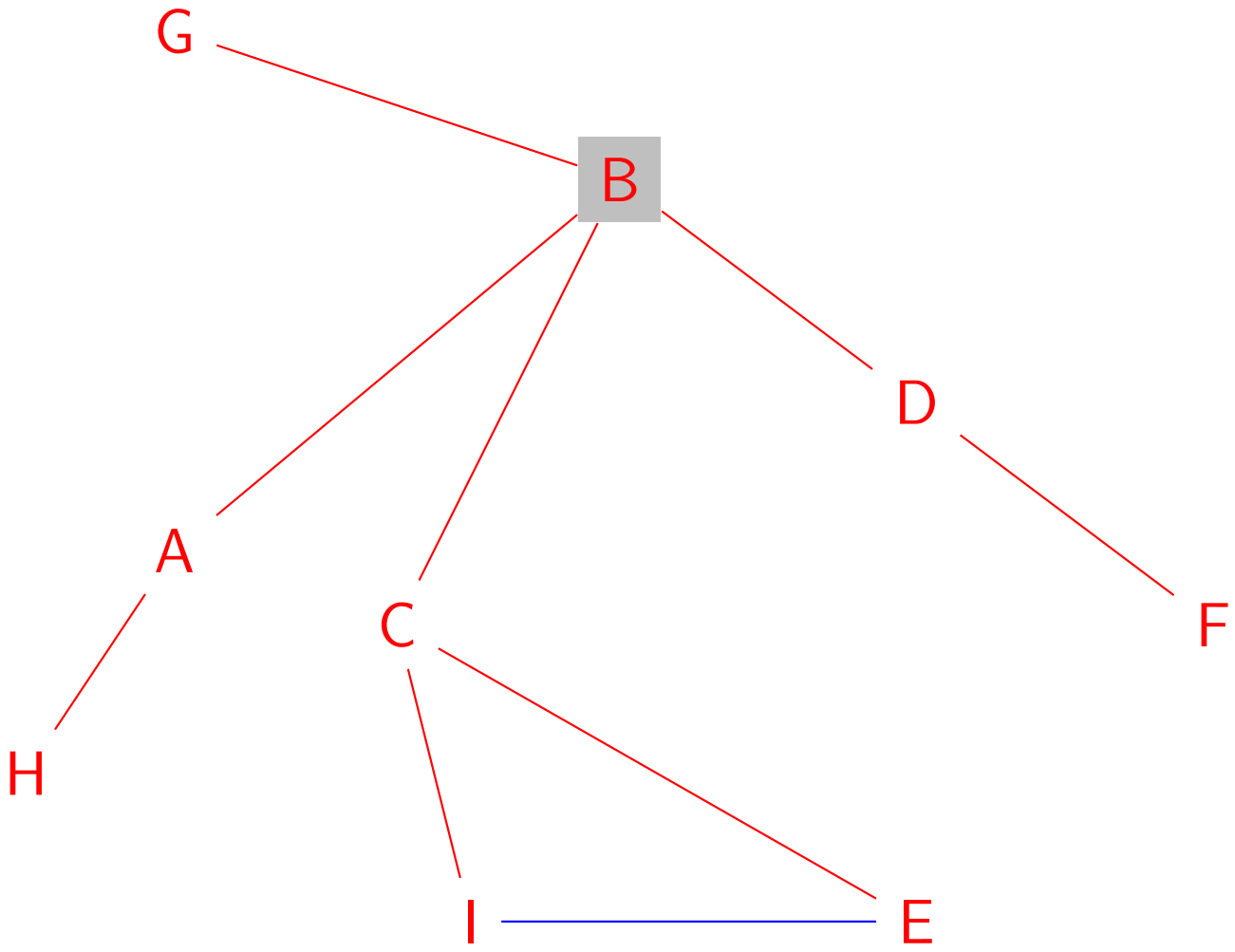
Question 4

Example of invalid DFS tree (cannot reach vertex E):



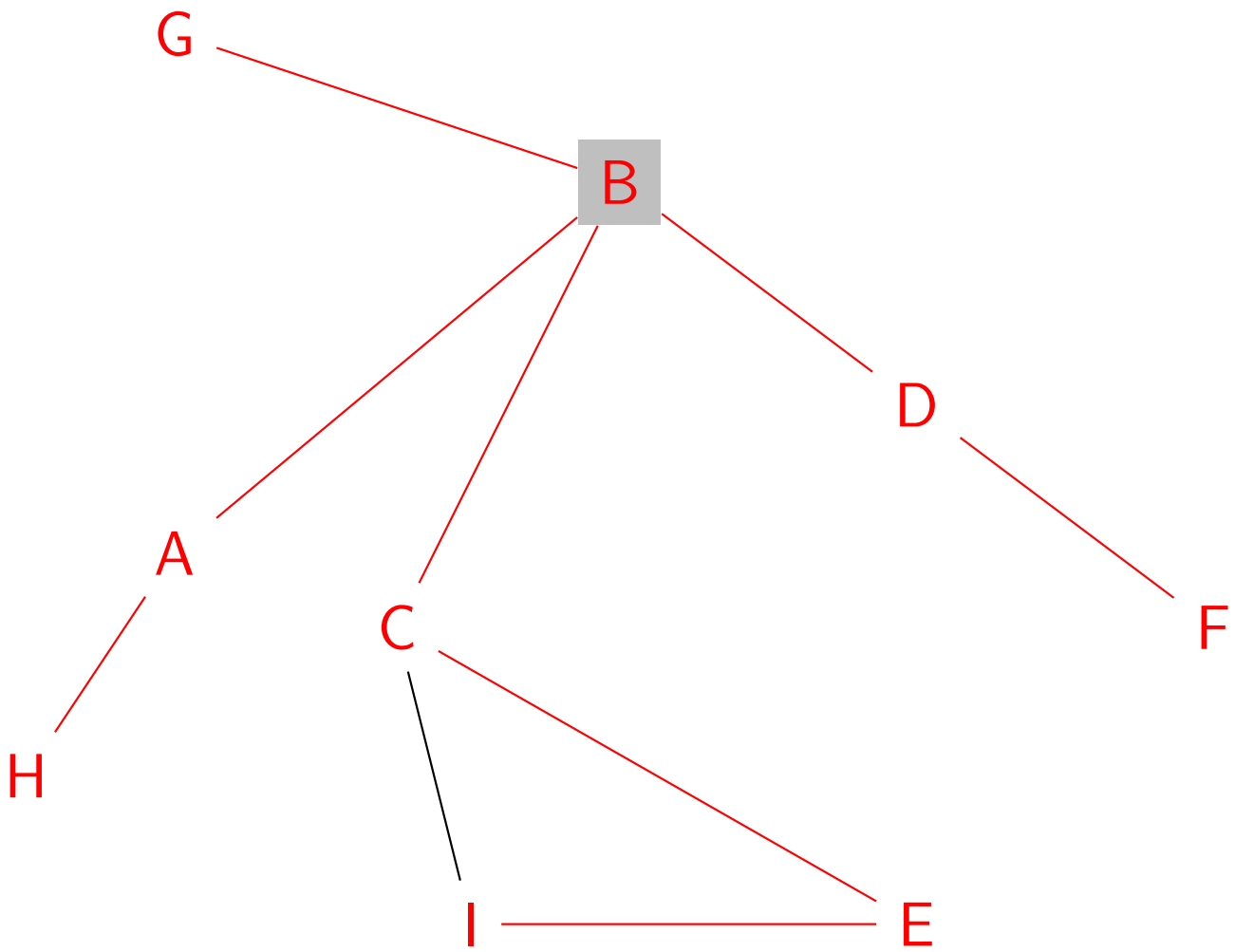
Question 4

Example of invalid DFS tree (cross edge $\overline{IE}$):



Question 4

Example of valid DFS tree:



Question 4

To ensure the spanning Tree T covers all vertices in G and avoid cross edges, we apply Depth-First Search on the spanning tree T.

Every time we finish traversing a vertex, we check whether all neighbors of the current vertex on G are marked.

If not, the tree is not a valid DFS tree.

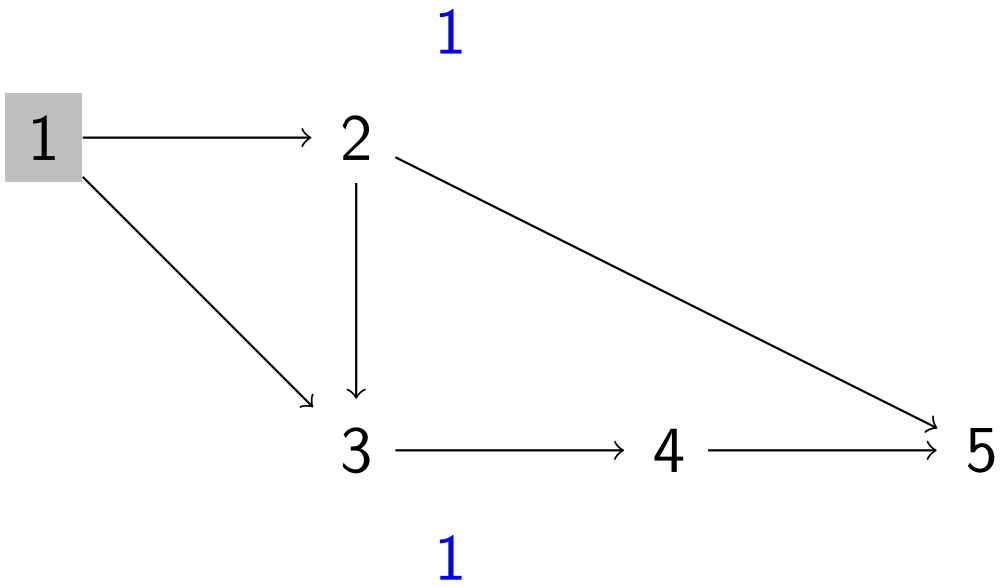
Question 4

```
function isDFSTree(G = (V, E), T = (V', E'), v)
    mark v;
    for all edge (v, w) ∈ E' do
        if w is marked then          // Ignore the path from ancestor
            continue;
        end if
        isDFSTree(G, T, w)
    end for
    for each edge (v, w) ∈ E do
        if w is unmarked then        // w should be visited because
            result := false;         // v is the ancestor of w
        end if
    end for
end function
```

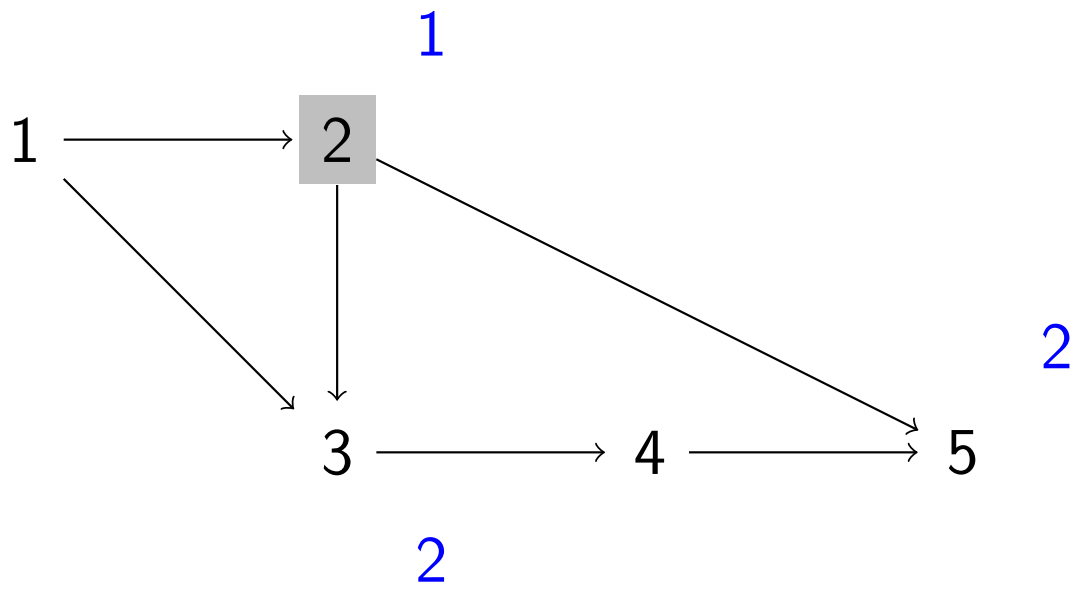
Question 5

5. (7.38) Given a directed acyclic graph $G = (V, E)$, find a simple (directed) path in G that has the maximum number of edges among all simple paths in G. The algorithm should run in linear time.

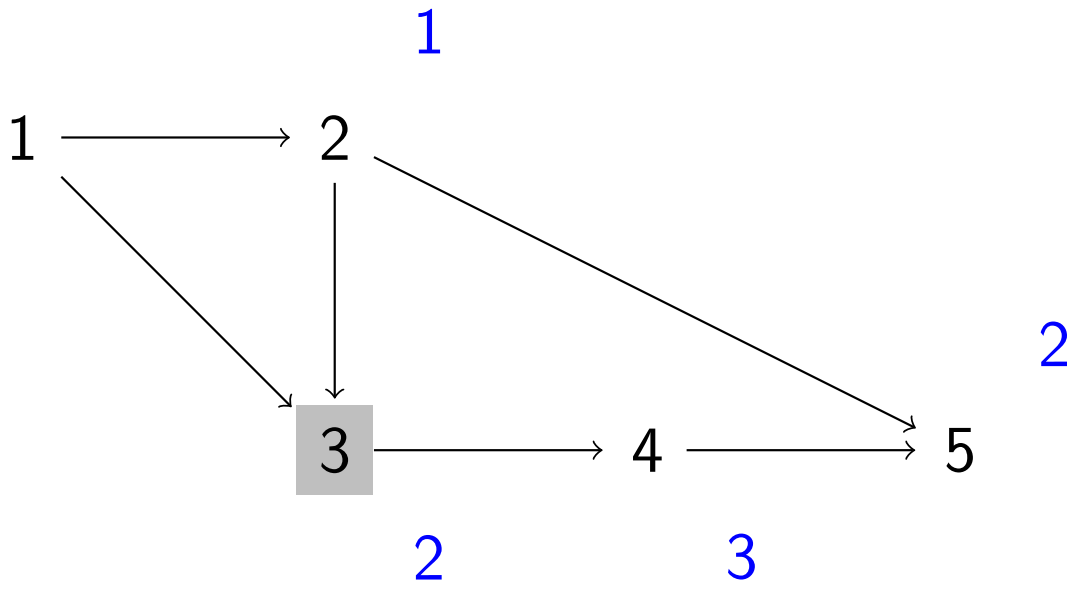
Question 5

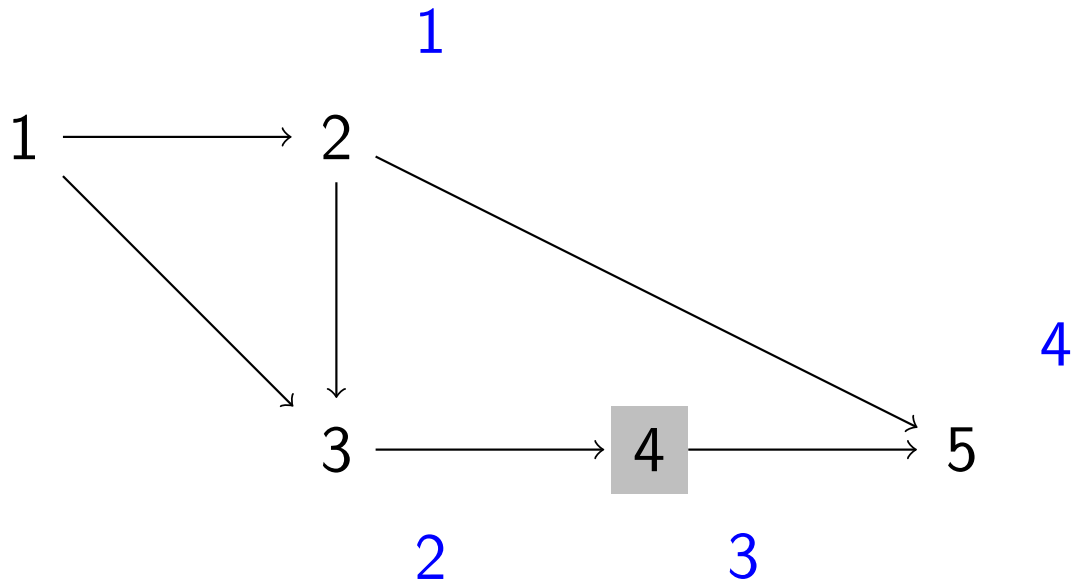


Question 5



Question 5





Question 5

```
function FIND LONGEST PATH(G)
    Initialize all vertex length to 0
    for v in Topological_Sorting(G) do
        for all edges (v, w) do
            if w.length < v.length + 1 then
                w.length := v.length + 1
                w.parent := v          // for path reconstructing
            end if
        end for
    end for
    w := vertex with max length
    LongestPath := [w]
    while w.parent do
        LongestPath.push(w.parent)
    end while
    return LongestPath
end function
```