Algorithms
National Taiwan University, Fall 2022

Assignment 7

Yu-Chieh Kuo B07611039†

†Department of Information Management, National Taiwan University

Problem 1

Algorithm 1 Find Eulerian Circuit

```
function EULERIAN(v, G)
    W := all neighbors of v
    for w ∈ W do
        Remove (v, w) from G
        Eulerian(w, G)
        Append (v, w) into the front of the edge path list
    end for
    Append v into the front of the vertex path list
end function
function FINDEULERIANCIRCUIT(G=(V,E))
    if Andy degree of vertices is odd or zero then
        return
    end if
    vertexPathList := 0, edgePathList := 0
    v := node ∈ G                                        ▷ // v can be arbitrary
    Eulerian(v, G)
    return vertexPathList.reverse(), edgePathList.reverse()
end function
```

The time and space complexity of Algorithm 1 is $O(|E|)$.

Problem 2

Remind that one of the property of a directed graph states that a directed graph has an Eulerian cycle *if and only if* every vertex has equal *in degree* and *out degree*. In this case, we discuss *in degree* and *out degree* separately.

In degree: For each vertex, if the first $n-2$ bits of a vertex is $a_1a_2\cdots a_{n-2}$, it has two exact directed edges from $0a_1a_2\cdots a_{n-1}$ and $1a_1a_2\cdots a_{n-1}$. Then the in degree of each vertex is 2.

Out degree: For each vertex, if the last $n-2$ bits of a vertex is $a_2a_3\cdots a_{n-1}$, it has two exact directed edges to $a_2a_3\cdots a_{n-1}0$ and $a_2a_3\cdots a_{n-1}1$. Then the out degree of each vertex is 2.

Consequently, every vertex has equal in degree and out degree, which yeilds that G_n has an Eulerian cycle; that is, G_n is a directed Eulerian graph. Moreover, we can obtain a property that if we start from a vertex in de Bruijn graph and trace by Eulerian Path, we will receive a de Bruijn sequence.

Problem 3

Algorithm 2 Detailed DFS in the topological sorting

```
function MainInvokingProcedure(G=(V,E))
    for v ∈ G do
        if v is unmarked then
            DFS(G, v)
        end if
    end for
end function
function DFS(G,v)
    mark v
    v.inDegree := 0                              ▷ // preWORK
    for (v, w) ∈ G do                            ▷ // All edges in the graph
        if w is unmarked then
            DFS(G, w)
        end if
        w.inDegree + +                           ▷ // postWORK
    end for
end function
```

Problem 4

The time complexity of Algorithm 3 is $\mathcal{O}(|E| + |V|)$.

Problem 5

The problem is equivalent to find the longest path in a graph G. Denote $D(u)$ by the longest valid path starting at node u, and the desired maximum number of edges for all nodes in G is the maximum value of $D(u)$ for all nodes in G, i.e., $\max D(u)\ \forall u \in G$. The algorithm is performed as Algorithm 4.

Note that the statement $D(s) = [s]$ indicates that the longest path ending at node s (i.e., $D(s)$ on the left side of the statement) is the path only containing node s (i.e., $[s]$ on the right side of the statement). The time complexity of Algorithm 4 is $\mathcal{O}(|E|)$. This DP method refers to this post in stackoverflow.

Algorithm 3 From DFS

```
function FromDFS(G=(V,E), T=(V,E'), v)
    result := True
    mark v
    for (v, w) ∈ E' do
        if v.parent = w then
            Continue
        end if
        if w.mark then
            result := False
        end if
        w.parent := v
        FromDFS(G, T, w)
    end for
    for (v, w) ∈ E do
        if !w.marked then
            result := False
        end if
    end for
    return result
end function
```

Algorithm 4 Finding the Longest Path

```
function FindingTheLongestPath(G=(V,E))
    D := HashMap(G)                                  ▷ Keys are nodes and values are path
    D(n − 1) = [n − 1]                               ▷ The base case
    longestPath := D(n − 1)
    for s : n − 2 → 0 do
        D(s) = [s]
        for (s, v) ∈ G do                            ▷ Each edge in G
            if v > s and ([s] + D(v)).length > D(v).length then
                D(s) = [s] + D(v)
            end if
        end for
        if D(s).length > longestPath.length then
            longestPath = D(s)
        end if
    end for
    return longestPath
end function
```