CSCE 411 Design and Analysis of Algorithms

HW4: Solutions

Q - 22.4-2

Idea:

We will use additional variable *paths* for each node $u \in V$ to store the number of simple paths from node u to v. For each node u, the number of paths will be given by summing all the number of paths from each node u's neighbors. We will invoke the function as *COUNT_SIMPLE_PATHS(s,t)*

Pseudocode:

Input: Graph $G = (V, E), s, t$

Output: $s.paths$

if $s == t$
    return 1
else if $s.paths \neq NIL$ then
    return $s.pahts$
else
    for each $(s,\ u) \in E$ do
        let $s.paths + = COUNT\_SIMPLE\_PATHS(u, t)$
    end for
    return $s.paths$
end if

Correctness:

The recursive approach will exhaustively search all the nodes starting from s. The recursion will break if it met the node t or have visited all the nodes. Since the search is exhaustive, it ensures that all the path will be included in the paths to be returned.

Time Complexity:

Time Complexity is $O(V + E)$

Q - 22.3

(a)

If G has Euler tour then in-degree(v) = out-degree(v) for each vertex v belongs to V.

As G has Euler tour and the Euler tour in made up of multiple edge-disjoint cycles. For each of these cycles, every vertex has to have an edge coming into it and an edge going out of it ot complete the cycle. The implies that in-degress(v) = out-degree(v) for each vertex v part of the cycle. Now, for the entire graph as we have an Euler tour, each disjoint cycle should be connected such that each cycle has an edge going out of it and edge coming into it. Thus, we can say that for each vertex v in V, in-degree(v) = out-degree(v).

If in-degree(v) = out-degree(v) for each vertex v belong to V, then G has a Euler tour.

Let us start at a vertex u and, via random traversal of edges, create a cycle. We know that once we take any edge entering a vertex $v \neq u$, we can find an edge leaving v that we have not yet taken. Eventually, we get back to vertex u, and if there are still edges leaving u that we have not taken, we can continue the cycle. Eventually, we get back to vertex u ans there are no untaken edges leaving u. If we have visited every edge in the graph G, we are done. Otherwise, since G is connected, there must be some unvisited edge leaving a vertex, say v, on the cycle. We can traverse a new cycle starting at v, visiting only previously unvisited edges, and we can merge this cycle into the cycle we already know. That is, if the original cycle is $u, ..., v, w, ..., u$, and the new cycle is $v, x, ..., v$, then we can create the cycle $u, ..., v, x, ..., v, w, ..., u$. We continue this process of finding a vertex with the unvisited leaving edge on the visited cycle, visiting a cycle starting and ending at this vertex, and merging in the newly visited cycle, until we have visited every edge forming Euler tour. Thus, we can say that G has Euler tour, if in-degree(v) = out-degree(v) for each vertex v belongs to V.

(b)

Idea: See (a) Part II explanation for algorithm idea.

Pseudocode:

Input: $G = (V, E)$

Output: T

for each vertex $v \in V$ do
if in-degree(u) $\neq$ out-degree(v) then
return "NO EULER TOUR"
end if
end for
start from an arbitrary vertex $u \in V$ and find a cycle C without repetitive edges
let $T = C$
remove edges of C from E
while $E \neq \phi$ do
find a vertex $u \in T$ such that there is an edge $(u, v) \in E$
find a cycle C starting with (u, v) without repetitive edges in $G = (V, E)$
remove edges of C from E
let $T = T \cup C$
end while
return T

Correctness:

Traversal in cycle and visiting unvisited edges ensures that all the edges will be visited. In (a), it has been proven that if in-degree(v) = out-degree(v), there exist a Euler tour. With this two conditions, the algorithm will provide a Euler tour.

Time Complexity:

Time complexity of this algorithm will be $O(|E|)$