HW 4. Yue Li

No.

Date

1. Ex 22.1-3.

(a) For the adjacency-list representation: (Assume $G=(V,E)$, $G^T=(V,E^T)$.)

Algo:

```
for v in V
    initialize adjL'(v)
for u in V
    for v in adjL(V)
        insert V to adjL'(u)
```

The total running time is $O(V+E)$

(b) For the adjacency-matrix representation of G^T,

Algo:

```
for i from 1 .. |V|
    for j from 1 to |V|
        adjm'[i,j] = adjm[j,i]
```

The total running time is $O(V^2)$

2. Ex. 22.2-7

It can be shown that if we run the BFS twice, once with an arbitrary root and find the most distant leaf from it, and the second time with this most distant leaf as the root and find the most distant vertex from it, then the corresponding distance between the leaf and the root is the diameter of the tree.

The running time is $O(V+E)$, $\because |E|=|V|-1$

$\therefore$ Running time is $O(V)$

No.

Date

3. Ex 22.3-6

```
DFS-VISIT(u)
1.  colour[u] ← GRAY
2.  time ← time + 1
3   d[u] ← time.
4   PUSH(u, S)
5   While S is not empty do (
6   for each v ∈ Adj[u] do
7       if color[v] = White
8           then π[v] ← u
9                PUSH(u, S)
            u = TOP(S).
10      Color[u] ← BLACK
11      f[u] ← time ← time + 1
12      u = pop(S)          )
```

Note that:

$\pi[v]$: is the predecessor of v

color[v]: is the color of v

d[u]: is the distance from root vertex S to vertex u.

S: the stack

TOP(s): returns the top element in the stack

PUSH(u, S): push element u into stack S

POP(S): pop out the top element of the stack S.

f(u): time stamp, records when the search finishes examining v's adjacency list.

No.

Date

4. Ex 22.5-3.

The professor is wrong.

A counter-example:

for the correct algorithm, the result is $\{0, 1\}$, $\{2\}$.

however, for the professor's algorithm, the result is $\{0, 1, 2\}$, which is incorrect.

5. Problem 22-3. next page

CPSC 411

Yue Li(yli@cs.tamu.edu)

Homework 4 solution for Euler tour problem

Problem [redacted] 22-3

This solution is improved from:
http://www.personal.kent.edu/ rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/eulerTour.htm
Definition An Euler tour of a connected, directed graph G = (V, E) is a cycle that traverses each edge of the graph G exactly once, although it may visit a vertex more than once.

In the first part of this section we show that G has an Euler tour if and only if the in-degree of every vertex is equal to the out-degree vertex. In the second part, we describe an algorithm to find an Euler tour of graph if one exists.

Part 1 Show that G has an Euler tour if and only if in-degree(v) = out-degree(v) for each vertex v02V

Proof

$\Rightarrow$ proof.

Suppose G has an Euler tour. If we trace the tour starting at any node, we all that every vertex is entered and left the same number of times.

$\Leftarrow$ proof.

We will call a cycle simple if it visits each vertex no more than once, and complex if can visit a vertex more than once. Suppose we have a connected graph for which the in-degree and out-degree of all vertices are equal. Let C be the longest complex cycle within G[1]. If C is not an Euler tour, then there is a vertex v of G touched by C such that not all edges in and out v of are exhausted by C. We may construct a cycle C' in G-C starting and ending at v by performing a walk in G-C. (The reason is that G-C also has the property that in-degrees and out-degrees are equal.) This means that the complex cycle that starts at v goes along the edges of C' (returning to v) and then goes along the edges of C is a longer complex cycle than C. This contradicts our choice of C as the longest complex cycle which means that C must have been an Euler tour.

Part 2 Find an Euler tour of given graph G if one exists.

ALGORITHM Given a starting vertex, v0, the algorithm will first find a cycle C starting and ending at v0 such that C contains all edges going into and out of v0. This can be performed by a walk in the graph. As we discover vertices in cycle C, we will create a linked list which contains vertices in order and such that the list begins and ends with vertex v0. We set the current pointer to the head of the list. We now traverse the list by moving our pointer "current" to successive vertices until we find a vertex which has an outgoing edge which has not been discovered. (If we reach the end of the list, then we have already found the Euler tour). Suppose we find a vertex, vi, that has an undiscovered outgoing edge. We then take a walk beginning and ending at vi such that all undiscovered edges containing vi are contained in the walk. We insert our new linked list into the old linked list in place of vi and move "current" to the new neighbor pointed to the first node containing vi. We continue this process until we reach the final node of the linked list, and the list will then contains an Euler tour.

[1]This is finite because we assume there are no repeated edges in C

Yue Li(yli@cs.tamu.edu)

Running Time of Euler Tour

The algorithm traverses each edge at most twice, first in a walk and second while traversing the list to find vertices with outgoing edges. Therefore, the total running time of the algorithm is $O(|E|)$.

Reference

- http://www.personal.kent.edu/ rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/eulerTour.htm
- C. L. Liu, Introduction to Combinatorial Mathematics, McGraw Hill, 1968.