Name: Sakina Ghafoor
COSC 40403 - Analysis of Algorithms: Homework 2
Due: 11:59:59 on September 11

Some of the questions below require you to draw a graph or to trace through a graph algorithm. Please use LaTeXor another computer drawing program (i.e. PowerPoint or similar), create your diagrams.

1. (10 points) Given the adjacency-list representation of a directed graph, explain how long does it take to compute the out-degree of every vertex. Explain how long does it take to compute the in-degree?

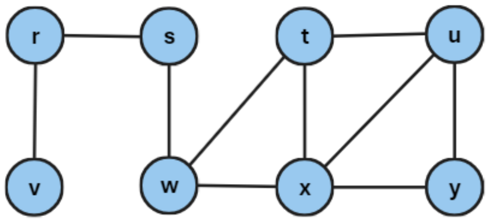
Answer:

The time to compute the out and in degree of every vertex in a graph should be the same. When using an adjacency list representation, an array is used and when accessing an element in the array, it is done in constant time. To find the out or in degree of a vertex, all adjacent edges coming out of or into it are traversed, meaning that the length or number of edges are all entries scanned in the array. It is the same procedure for both in-degree and out-degree. This would be $\Theta(E)$ or $|E|$. We also need to take into account that each every vertex is scanned, which means that for every single vertex we must go through the edge list once to be able to compute the degree or number of edges. So, we would end up with $\Theta(V + E)$ or $\Theta(|V| + |E|)$.

2. (10 points) The BFS algorithm presented in class uses a queue as the data structure for storing discovered vertices. Explain what would happen if you change the data structures to use a stack instead of a queue. Use the BFS graph example presented in lecture to demonstrate your solution.

Answer:

If a stack instead of a queue is used in BFS, then the way a vertex is traversed will change. When visiting a vertex it is enqueued to the end of the queue but with a stack a new visited vertex would be pushed to the top. With dequeue the vertex is removed from the front but with a stack it is removed from the top as well. Queues follow first-in-first-out and stacks follow last-in-first-out.



Using the example from lecture, the BFS queue would start with vertex s. The traversal would go like this: s, r, w, v, t, x, u, y
With a stack the traversal would go like this: s, r, v, w, t, u, x, y
So, the stack transforms this into DFS because of the way each branch or edge is explored. With a stack the whole sub-graph or branch of the child would have to be explored fully before moving onto the other child of the original parent so that the popped vertex correctly corresponds with visitations. This is different from a queue which allows for level by level visitations.

3. (10 points) Give an algorithm to detect whether a given undirected graph contains a cycle. If the graph contains a cycle, then you algorithm should output one. (It should not output all cycles in the graph, just one of them.) The running time of your algorithm should be $O(V+E)$.

Answer:

Algorithm cycleDFS(G, v)

```
setLabel(v, visited)
S.push(v)
for all e $\in$ G.adjacentEdges(v) do
    if getLabel(e) = unexplored then
        w $\leftarrow$ opposite(v, e)
        S.push(e)
        if getLabel(w) = unexplored then
            setLabel(e, discovery)
            cycleDFS(G, w)
            S.pop(e)
        else
            T $\leftarrow$ new empty stack
            repeat
            o $\leftarrow$ S.pop()
            until o = w
            return T.elements()
        end if
    end if
end for
S.pop(v)
```

4. (20 points total) The ***transpose*** of a directed graph $G = (V, E)$ is the graph $G^T = (V, E^T)$, where $E^T = \{(v, u) \in V \times V : (u, v) \in E\}$. That is, G^T is G with all its edges reversed.

   (a) (10 points) Describe an efficient algorithm for computing G^T from G if the graph is represented using an adjacency matrix. Analyze the running time of your algorithm.

   (b) (10 points) Describe an efficient algorithm for computing G^T from G if the graph is represented using an adjacency list. Analyze the running time of your algorithm.

Answer:

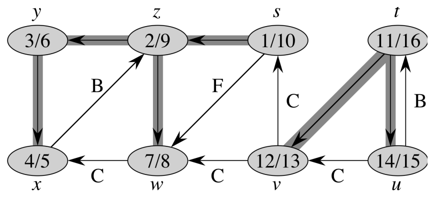
(a) An algorithm that could be used to compute the transpose of a graph using an adjacency matrix is to swap the entries in the matrix itself manually. This would involve switching opposite vertices on either side of the diagonal in the matrix, which represents loops or connections from a vertex to itself. This will reverse the edges by having the diagonal serve as a sort of mirror. The time complexity for this would be $O(V^2)$ or $O(|V|^2)$ because each vertex in the matrix is iterated over and the vertex in the [i][j] position is copied or added it over to the corresponding [j][i] position.

(b) An algorithm that could be used to compute the transpose of a graph using an adjacency list can start with first iterating through all vertices in G and for each vertex v to iterate through its adjacent edges. Then for that vertex v, for each entry in the set or list of adjacent edges, add vertex v. The time complexity would be $O(V + E)$ or $O(|V| + |E|)$. This is because for each vertex, all of its adjacent edges are scanned and the vertex is added.

5. (20 points total) The ***incidence matrix*** of a directed graph $G = (V, E)$ with no self-loops is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that

$$b_{ij} = \begin{cases} -1 & \text{if edge } j \text{ leaves vertex } i, \\ 1 & \text{if edge } j \text{ enters vertex } i, \\ 0 & \text{otherwise} \end{cases}$$

Given the following graph:



(a) (5 points) What is adjacency matrix of the graph?

(b) (5 points) Compute the entries of the incidence matrix B using the function above.

(c) (5 points) Compute BB^T.

(d) (5 points) What do the entries of the matrix product BB^T represent?

Answer:

(a) y, z, s, t, x, w, v, u $\begin{pmatrix} 0&0&0&0&1&0&0&0 \\ 1&0&0&0&0&1&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&1 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&1&0&0 \\ 0&0&0&1&0&0&1&0 \end{pmatrix}$

(b) $\begin{pmatrix} 0&1&0&0&-1&0&0&0 \\ -1&0&1&0&1&-1&0&0 \\ 0&-1&0&0&0&-1&1&0 \\ 0&0&0&0&0&0&-1&0 \\ 1&-1&0&0&0&1&0&0 \\ 0&1&1&0&-1&0&1&0 \\ 0&0&-1&1&0&-1&0&1 \\ 0&0&0&0&0&0&-1&0 \end{pmatrix}$ (c) BB^T is: $\begin{pmatrix} 1&-1&0&0&0&1&0&0 \\ 0&2&0&0&-2&1&0&0 \\ -1&0&1&0&-1&0&0&0 \\ 0&0&-1&0&0&-1&0&0 \\ -1&0&0&0&2&-1&0&0 \\ 1&0&1&0&0&3&0&0 \\ 0&-1&0&1&-1&-1&2&1 \\ 0&0&-1&0&0&-1&0&0 \end{pmatrix}$

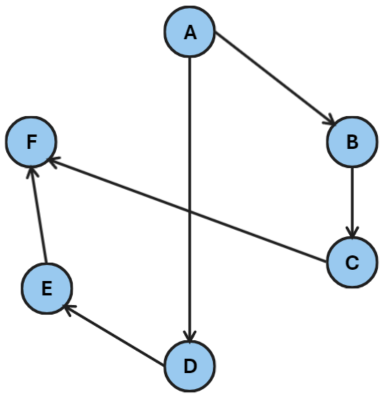
(d) The entries of BB^T represent different things based on the position. Entries in the diagonal represent the in and out-degree or the entire degree. The entries not in the diagonal represent number of edges that are between the vertices.

6. (10 points) Given the following DAG, how many topological orderings does it have? Explain your answer.

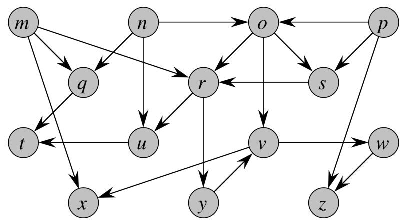
$$G = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Answer:

This DAG has 4 topological orderings. This is because in the graph, starting from vertex A, there are 2 possible paths to go to the next vertex, B or D. From here, there are 2 separate paths that can be taken in partial order. For the first and second path/order we can assign a value of 2 for the number of orders. This is because there are the 2 options, B and D, and then the other options of C or F for B and E or F for D. The sum of the possible orderings would then be 4.

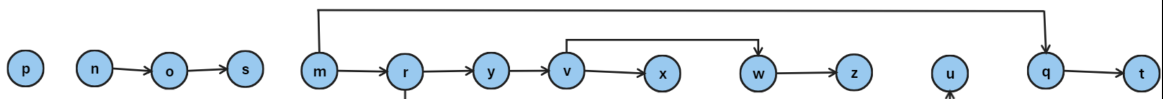


7. (10 points) Trace through the TOPOLOGICAL-SORT algorithm and show the ordering of vertices produced on the following DAG. Assume your DFS algorithm processes the nodes in alphabetical order and the edges for each node in the adjacency list are also listed in alphabetical order.



Answer:

This would be the ordering:



p
n
o
s
m
r
y
v
x
w
z
u
q
t