Homework 4

Foundations of Algorithms

Divesh Badod

1. Electronic Submission
2. Electronic Submission
3. Electronic Submission

4.

a. An adjacency-list is an array representation of the graphs and every vertex connection has a list within that array which shows which other vertices it is connected to, so considering there are V vertices in the graph the size of array would be |V| and with number of edges as E we get to compute the outer-degree of a particular vertex v by counting the number of edges which are outbound from the vertex which is |E| and if we do this for every vertex then it will take a total time of $O(|E| + |V|)$

For in-degree of a vertex we have to scan through the adjacency-list and count how many times that particular vertex has appeared on it, similar to the previous case it will take a total of $O(|E| + |V|)$ time

b.

- If a hash table is used to represent the graph and an efficient hash function is used to store the vertices then the expected time should be $O(1)$
- The disadvantage of this method is the amount of space required because it will take more space than the linked-list of the adjacency-list
- The alternate approach would be to use a binary search tree as the list are ordered and sorted.
- The only disadvantage would the lookup time because in hash table it is expected to be $O(1)$ but here it would be $O(lgV)$ where V is the number of vertices.

5. a.

Vertex	r	s	t	u	v	w	x	y
d	4	3	1	0	5	2	1	1
π	s	w	u	Nil	r	t	u	u

b.

```
BFS(G, s)
    1 for each vertex u ϵG.V − (s)
    2         u.color = WHITE
    3         u.d = INFINITY
    4         u.π = NIL
    5 s.color = GRAY
    6 s.d = 0
    7 s.π = NIL
    8 Q = ϕ
    9 ENQUEUE(Q, s)
   10 while Q ≠ ϕ
   11       u = DEQUEUE(Q)
   12       for each vϵ G.Adj[u]
   13           if v.color == WHITE
   14                v.color = GRAY
   15                v.d = u.d + 1
```

16 $v.\pi = u$
17 $ENQUEUE(Q, v)$

The algorithm will initialize the by colouring all the vertices WHITE, and when enqueued it will turn to GRAY, after the vertex is processed it dequeues from the Q and later colours it BLACK. Here the BLACK colour is just to indicate that the vertex has been processed and when you remove line 18 from the code those vertices will still be processed but only the colour won't change to BLACK, it will remain GRAY and as the wanted output aren't the colours of the vertex but whether they have been visited once or not it doesn't matter at all to remove the 18th line, it will produce the same results.

c. We have $u.d = \delta(s, v)$ at the end of the procedure. Since $u.d = \delta(s, v)$ is a property of the graph, no matter which representation of the graph in terms of adjacency lists that we choose, this value will not change. This is confirmed by the theorem of correctness of the BFS algorithm.

Now, to show that π does depend on the ordering of the adjacency lists. First notice that if we work out the adjacency list of the given diagram t precedes x in the list of w. Hence we have $u.\pi = t$. But if we mess around with the adjacency-list suppose that we had x preceding t in the list of w, this means that we will have $u.\pi = x$ which is different if the order of the list changes.

6.

```
DFS(G)
    count = 0
    Stack = EMPTY
    for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
    for each vertex u ∈ G.V
        if u.color == WHITE
        count = count + 1
        u.d = count
        u.color = GRAY
        PUSH(Stack, u)(Pushes elements in the stack)
        while ! isEmptyStack
            v = TOP(Stack)(Returns top element without popping)
            isNeighbor = true
            if v.color == GRAY
                for each vertex w ∈ G.Adj[v]
                    if w.color == WHITE
                        count = count + 1
                        w.d = count
                        w.color = GRAY
                        PUSH(Stack, w)
                        isNeighbor = false
            if isNeighbor
                count = count + 1
                v.f = count
                POP(Stack)(Returns the top of the element and pops
                    it out of the stack)
```