

# Introduction to Algorithm

## CS430 Homework 2

### Question -1

#### Problem-1

Take the following list of functions and arrange them in ascending order of growth rate. That is, if function  $g(n)$  immediately follows function  $f(n)$  in your list, then it should be the case that  $f(n)$  is  $O(g(n))$ .

$$f_1(n) = n^{2.5} \quad // \text{Polynomial}$$

$$f_2(n) = \sqrt{2n} \quad // \text{Linear}$$

$$f_3(n) = n + 10 \quad // \text{Linear}$$

$$f_4(n) = 10^n \quad // \text{Exponential}$$

$$f_5(n) = 100^n \quad // \text{Exponential}$$

$$f_6(n) = n^2 \log n$$

**Answer:**

Based on the growth rate of above six different functions:

Arrangement of function in Ascending order is:  $f_2, f_3, f_6, f_1, f_4, f_5$

**Explanation:**

$f(n) = O(g(n))$  if  $c$  and some initial value  $n_0$  are positive when  $f(n) \leq c * g(n)$  is true.

Linear Function  $f_2(n)$  and  $f_3(n)$ ,

$$\sqrt{2n} < n+10$$

$$\sqrt{2n} < n$$

$$\text{so, } f_2 < f_3$$

Function polynomial growth functions  $f_1(n)$  and  $f_6(n)$

$$f_6(n) < f_1(n) \rightarrow n * n * \log(n) < n * n * \sqrt{n} \rightarrow \log(n) < \sqrt{n}$$

$$f_6(n) < f_1(n)$$

If we compare the function  $f_3(n)$  and  $f_6(n)$

$f_3$  is linear function and  $f_6$  is quadratic function so,

$$f_3(n) < f_6(n)$$

**Partial ordering:  $f_2, f_3, f_6, f_1$**

For function  $f_1$  grow slower than  $f_4$  for large value of  $n$

If we take log for both function f1 and f4

$$\log(n^{2.5}) < \log(10^n) \rightarrow 2.5 \log n < n \log 10$$

$$f1(n) < f4(n)$$

For exponential function f4(n) and f5(n)

$$f4(n) < f5(n)$$

From the equation 1,2 and 3

**Function in ascending order: f2, f3, f6, f1, f4, f5**

$$f2(n) = \sqrt{2}n < f3(n) = n + 10 < f6(n) = n^2 \log n < f1(n) = n^{2.5} < f4(n) = 10^n < f5(n) = 100^n$$

## Problem-2

Take the following list of functions and arrange them in ascending order of growth rate. That is, if function g(n) immediately follows function f(n) in your list, then it should be the case that f(n) is  $O(g(n))$ .

$$g1(n) = 2^{\sqrt{\log n}}$$

$$g2(n) = 2^n$$

$$g4(n) = n^{4/3}$$

$$g3(n) = n (\log n)^3$$

$$g5(n) = n^{\log n}$$

$$g6(n) = 2^{2^n}$$

$$g7(n) = 2^{n^2}$$

**Answer:**

The Ascending order of Function is: g1, g3, g4, g5, g2, g7, g6

g1 is the smallest growing function compare to other functions

For Exponential function g2 and g7,  $g2(n) < g7(n)$

For Exponential function g6 and g7,  $g7(n) < g6(n)$

$$\text{For function } g2(n) < g7(n) \rightarrow 2^n < 2^{n^2}$$

**So, partial ordering of function g1, g2, g7, g6 is :  $g1 < \dots g2 < g7 < g6$**

For polynomial function g3 and g4,

$$n (\log n)^3 < n^{4/3}$$

$$n (\log n)^3 < n * n^{1/3} \Rightarrow (\log n)^9 < n = g3(n) < g4(n)$$

$$\text{so, } g3(n) = n (\log n)^3 < g4(n) = n^{4/3}$$

For function g4 and g5

$$n^{4/3} < n^{\log n} \rightarrow 4/3 \log n < \log n \log n \rightarrow \log n < (\log n)^2$$

For function g3 and g5,

$$g3(n) < g5(n)$$

$$n * (\log n)^3 < n^{\log n} \rightarrow \log(n(\log n)^3) < \log n \cdot \log n \rightarrow \log n + \log(\log n)^3 < (\log n)^2$$

$$\text{so, } g3(n) < g4(n) < g5(n)$$

Function g5 grow slower than g2, g6 and g7

$$\text{So } g5(n) = n^{\log n} < g2(n) = 2^n < g7(n) = 2^{n^2} < g6(n) = 2^{2^n}$$

$$g5 < g2 < g7 < g6$$

**Order sequence is: g3, g4, g5, g2, g7, g6**

As g1 is the smallest growing function compare to other functions

So g1 comes before all other functions

$$g1, g3, g4, g5, g2, g7, g6 \text{ if } g3(n) = n (\log n)^3 < g4(n) = n^{4/3}$$

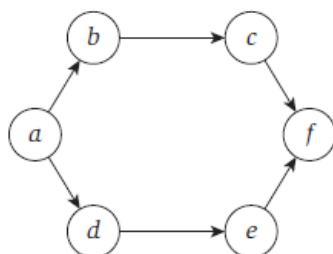
OR

$$g1, g4, g3, g5, g2, g7, g6 \text{ if } g4(n) = n (\log n)^3 < g3(n) = n^{4/3}$$

## Question -2

Consider the directed acyclic graph G in Figure How many topological orderings does it have

**Answer:**



In-degree ( )	
A	0
B	1
C	1
D	1
E	1
F	2

For Simplicity,

$$a = A, b = B, c = C, d = D, e = E$$

Topological ordering:

First node with indegree =0

Output: A

Remove adjacent edge

Indegree of Each node:

B	C	D	E	F
0	1	0	1	2

So next node will be either B or D as their indegree is zero .

Case - I: Next is B

If Select B,

Then indegree of C become zero and next node will be C or D,

Possible topological ordering be : a b c d e

OR a b d c e

OR a b d e c

Case -II: Next is D

If we choose D,

Then indegree of E also become zero and node will be E or B

**Possible topological ordering be** a d e b c

**OR** a d b c e

**OR** a d b e c.

Last node will be f as all nodes are visited.

**There are six different possible Topological ordering are:**

a b c d e f, a b d c e f , a b d e c f, a d e b c f, a d b c e f, a d b e c f

## Question -3

The algorithm described in Section 3.6 for computing a topological ordering of a DAG repeatedly finds a node with no incoming edges and deletes it. This will eventually produce a topological ordering, provided that the input graph really is a DAG.

But suppose that we're given an arbitrary graph that may or may not be a DAG.

Extend the topological ordering algorithm so that, given an input directed graph  $G$ , it outputs one of two things:

- (a) a topological ordering, thus establishing that  $G$  is a DAG; or
- (b) a cycle in  $G$ , thus establishing that  $G$  is not a DAG.

The running time of your algorithm should be  $O(m + n)$  for a directed graph with  $n$  nodes and  $m$  edges.

### Answer:

Let  $S$  is set of All vertices in a Graph

1. Store each vertex's In-Degree in an array IN
2. Initialize a queue  $Q1$  with all in-degree zero vertices
3. if  $Q1$  is Empty and no of vertices in graph  $G > 1$

Pick a vertex  $u$  from set  $S$ ,

L1: set status of  $u$  is visited and store it in Array OUT

For each Edge  $(u, v)$

If node  $v$  is unvisited

Goto L1:

Else

If  $v$  is visited // [already in Array OUT]

Graphs contain Cycle

Display Array OUT

5.else

6. While there are vertices in the queue  $Q1$  with indegree = 0

Dequeue and store a vertex in output Array OUT

Reduce In-Degree of all vertices adjacent to it by 1

Enqueue any of these vertices whose In-Degree became zero

4. return output Array OUT

Running time of Algorithm is  $O(V+E)$

### Question -4

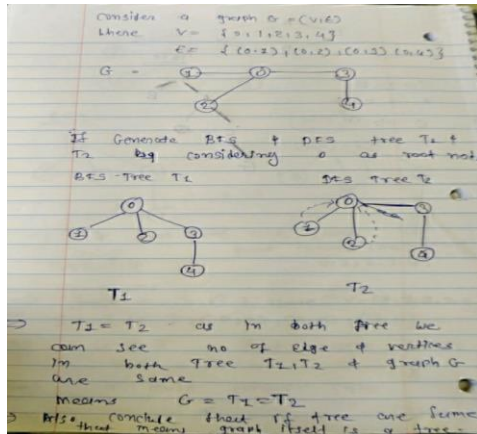
We have a connected graph  $G = (V, E)$ , and a specific vertex  $u \in V$ . Suppose we compute a depth-first search tree rooted at  $u$ , and obtain a tree  $T$  that includes all nodes of  $G$ . Suppose we then compute a breadth-first search tree rooted at  $u$ , and obtain the same tree  $T$ . Prove that  $G = T$ . (In other words, if  $T$  is both a depth-first search tree and a breadth-first search tree rooted at  $u$ , then  $G$  cannot contain any edges that do not belong to  $T$ .)

**Answer:**

A Graph is a connected graph, when a path can be found between every pair of different vertices.

Let  $G = (V, E)$  Where  $V = \{0, 1, 2, 3, 4\}$  AND  $E = \{(1, 0), (0, 2), (0, 3), (3, 4)\}$

Root node = 0



Prove  $G = T$

**Proof by Contradiction:**

Tree generated by DFS and BFS is same tree  $T$ .

Assume

Graph  $G$  not equal to Tree  $T$ ,

Means  $E_1 = (v_1, v_2)$  is part of Graph  $G$  but not belongs to Tree  $T$

If Tree  $T$  is DFS tree,

either  $v_1$  or  $v_2$  must be ancestor of the other or there will be a back edge.

For BFS Tree  $T$ , distance between the two nodes  $v_1$  and  $v_2$  from root  $u_1$  can differ by only level 1.

As both BFS and DFS tree are same,

So, either,  $v_1$  or  $v_2$  should be ancestor of other or both  $v_1$  and  $v_2$  can differ by almost 1 from common root node.

Implies,  $E_1 = (v_1, v_2)$  is an edge in Tree  $T$ .

Which contradict the statement that  $(v_1, v_2)$  does not belong to tree  $T$ .

## Question-5

Claim: Let  $G$  be a graph on  $n$  nodes, where  $n$  is an even number. If every node of  $G$  has degree at least  $n/2$ , then  $G$  is connected.

Decide whether you think the claim is true or false and give a proof of either the claim or its negation.

**Answer:**

### Proof by Contradiction

We can assume Graph is not Connected.

Let node  $v_2$  and  $v_1$  are two nodes in Graph  $G$ .

As each node has degree  $n/2$

Each  $v_1$  and  $v_2$  are connected with  $n/2+1$  other node

Then, Total Number of nodes in set  $V = 2*(n/2+1) \rightarrow n+2$  nodes

Which contradict with statement that we have only  $n$  even nodes in  $G$ .

Simply,

Graph has  $n$  even number of nodes. -----(A)

Every node of Graph has at least  $n/2$  degree. -----(B)

According to equation (A)

We can say that minimum number of nodes are 2 called  $v_1$  and  $v_2$  in set  $V$ .

According to given details (B)

We can say that each node has  $n/2$  degree implies degree of each  $v_1$  and  $v_2$  is 1.

So, if there are only  $n=2$  nodes in graph

Degree of each node is  $n/2 = 1$  that implies that  $v_1$  and  $v_2$  are connected with each other

So graph must be connected, and given statement is true.

## Question-6

Suppose we are given an undirected graph  $G = (V, E)$  and we identify two nodes  $v$  and  $w$  in  $G$ .

Give an algorithm that computes the number of shortest  $v$ - $w$  paths in  $G$ .

**Answer:**

Graph  $G = (V, E)$  where  $V$  is number of nodes and  $E$  is number of edges.

$S$  = source node

1. Set Discovered[ $S$ ] = true
2. Discovered[ $v$ ] = false for all other  $v$
3. Initialize  $L[0]$  to consist the single element  $S$
4. Set the layer counter  $i = 0$

5. Length[S]=0 //distance
6. Spath[S]=1 //shortest path
7. While L[i] is not empty

Initialize an empty list L [i + 1]

For each node  $u \in L[i]$

Consider each edge (u, v) incident to u

If Discovered[v] = false, then

Set Discovered[v] = true

Add v to the list L [i + 1]

End

If length[v] > length [u]+1

Length[v] =length[u]+1

Spath [v] = Spath [u]

End

Else if length[v]=length[u]+1

Spath[v]= Spath[v]+ Spath [u]

End

End for

8. Increment the layer counter i by one

End while

9. Return spath[D]