CSCI 570 HOMEWORK 2 SOLUTIONS
Spring 2024

Q1. What is the tight bound on worst-case runtime performance of the procedure below? Give an explanation for your answer. (10 points)

```
int c = 0;
for(int k = 0; k <= log₂n; k++)
    for(int j = 1; j <= 2^k; j++)
        c=c+1
return c
```

Solution :
The outer loop runs for log(n) +1 times and the inner loop runs for 2^k times for each k. So the total number of iterations become approximately $\sum_{k=0}^{\log n} 2^k = 2^{\log(n)+1} - 1 = 2n-1$

Therefore time complexity is $\Theta(n)$

Rubrics :
- 4 pts : Correct answer $\Theta(n)$
- 6 pts : Correct explanation

Q2. Given an undirected graph G with n nodes and m edges, design an O(m+n) algorithm to detect whether G contains a cycle. Your algorithm should output a cycle if G contains one. (10 points)

Solution :
Starting from an arbitrary vertex s, run BFS to obtain a BFS tree T, which takes O(m + n) time. If G = T, then G is a tree and has no cycles. Otherwise, G has a cycle and there exists an edge e = (u, v) ∈ G \ T. Let w be the least common ancestor of u and v. There exist a unique path T1 in T from u to w and a unique path T2 in T from w to v. Both T1 and T2 can be found in O(m) time. Output the cycle e by concatenating P1 and P2.

Rubric :
• 5 pts: for detecting whether G contains a cycle
• 3 pts: for finding (the edges in) a cycle if G contains one
• 2 pts: describing that the runtime is O(m + n) in each step (and thus total)

Q3. For each of the following indicate if f = O(g) or f = Θ(g) or f = Ω(g) (10 points)

	f(n)	g(n)
1	$n\log(n)$	$n^2\log(n^2)$
2	$\log(n)$	$\log(\log(5^n))$
3	$n^{1/3}$	$(\log(n))^3$
4	2^n	2^{3n}
5	$n^4/\log(n)$	$n(\log(n))^4$

Solution :

1. f = O(g)
2. f = Θ(g)
3. f = Ω(g)
4. f = O(g)
5. f = Ω(g)

Rubrics : 2 point for each correct answer

Q4. Indicate for each pair of expressions (A,B) in the table below, whether A is O, Ω, or Θ of B (in other words, whether A=O(B), A= Ω(B), or A= Θ (B)). Assume that k and C are positive constants. You can mark each box with Yes or No. No justification needed. (9 points)
(Note: log is base 2)

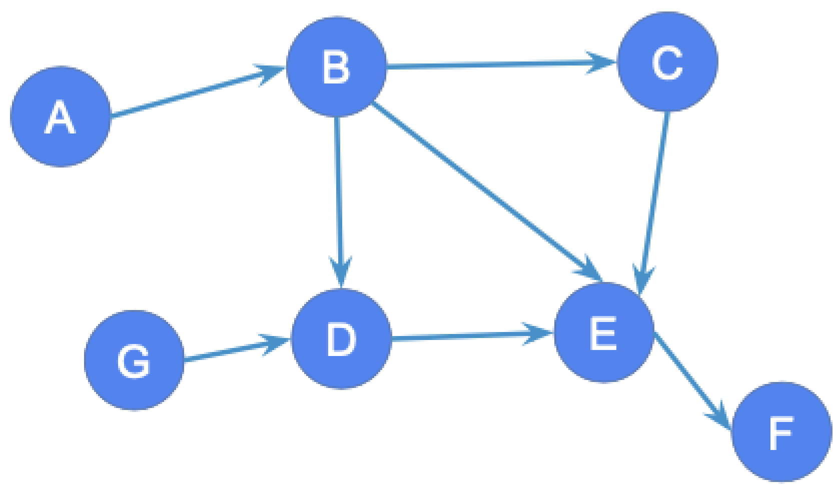
A	B	O	Ω	Θ
$n^3+\log(n)+n^2$	$C*n^3$			
n^2	$C*n*2^{\log(n)}$			
$(2^n)*(2^k)$	n^{2k}			

Solution : 9 points

A	B	O	Ω	Θ
$n^3+\log(n)+n^2$	$C*n^3$	Y	Y	Y
n^2	$C*n*2^{\log(n)}$	Y	Y	Y
$(2^n)*(2^k)$	n^{2k}	N	Y	N

Rubrics : (1 point for each Y/N)

Q5. Find the total number of possible topological orderings in the following graph and list all of them (15 points)



Solution: There are 7 possible Topological ordering

1. G-A-B-D-C-E-F
2. G-A-B-C-D-EF
3. A-G-B-C-D-E-F
4. A-G-B-D-C-E-F
5. A-B-G-C-D-E-F
6. A-B-G-D-C-E-F
7. A-B-C-G-D-E-F

Rubrics :

- 1 pts : Correct no of topological ordering
- 2 pts : for each topological ordering

Q6. Given a directed graph with m edges and n nodes where every edge has weight as either 1 or 2, find the shortest path from a given source vertex 's' to a given destination vertex 't'. Expected time complexity is O(m+n). (8 points)

Solution : We can modify the graph and split all edges of weight 2 into one vertex and two edges of weight 1 each, like edge (u,v) of weight 2 to (u,u') of weight 1 and (u',v) of weight 1. In the modified graph, we can use BFS to find the shortest path. Maximum number of new edges and vertices added is O(m), so complexity of this approach is O(m+n).

Rubrics :

- 4 pts : Correctly utilizing the fact that T is a DFS.
- 4 pts : Correctly utilizing using the fact that T is a BFS

Q7. Given functions f_1, f_2, g_1, g_2 such that $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample. (12 points)

(a) $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

(b) $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$

(c) $f_1(n)^2 = O\, g_1(n)^2$

(d) log2 $f_1(n) = O$ (log2 $g_1(n)$)

Solution:

By definition, there exist c_1, $c_2 > 0$ such that

$$f_1(n) \le c_1 \cdot g_1(n) \text{ and } f_2(n) \le c_2 \cdot g_2(n)$$

for n sufficiently large.

(a) True.

$$f_1(n) \cdot f_2(n) \le c_1 \cdot g_1(n) \cdot c_2 \cdot g_2(n) = (c_1c_2) \cdot (g_1(n) \cdot g_2(n)).$$

(b) True.

$$\begin{aligned} f_1(n) + f_2(n) &\le c_1 \cdot g_1(n) + c_2 \cdot g_2(n) \\ &\le (c_1 + c_2)(g_1(n) + g_2(n)) \\ &\le 2 \cdot (c_1 + c_2) \max(g_1(n), g_2(n)). \end{aligned}$$

(c) True.

$$f_1(n)^2 \le (c_1 \cdot g_1(n))^2 = c_1^2 \cdot g_1(n)^2.$$

(d) False. Consider $f_1(n) = 2$ and $g_1(n) = 1$. Then

$$\log_2 f_1(n) = 1 \neq O(\log_2 g_1(n)) = O(0).$$

Rubric (4 pts for each subproblem):

- 1 pts: Correct T/F claim
- 3 pts: Provides a correct explanation or counterexample

Q8. Design an algorithm which, given a directed graph G = (V, E) and a particular edge e ∈ E, going from node u to node v determines whether G has a cycle containing e. The running time should be bounded by O(|V | + |E|). Explain why your algorithm runs in O(|V | + |E|) time. (8 points)

Solution: Delete e from G to obtain a new graph G', run the DFS or BFS algorithm starting from v, if u can be traversed, then G has a cycle containing e, otherwise G does not have such a cycle.In the worst case, all the edges and nodes are traversed, resulting in O(|V | + |E|) time; or you can say the running time of DFS or BFS is O(|V | + |E|).

Rubrics :

- 6 pts : Correctly utilizing BFS/DFS
- 2 pts : Explanation for run time

Q9. Solve Kleinberg and Tardos, **Chapter 3, Exercise 6.** (8 points)

Solution : Proof by Contradiction: assume there is an edge e = (x, y) in G that does not belong to T. Since T is a DFS tree, one of x or y is the ancestor of the other. On the other hand, since T is a BFS tree, x and y differ by at most 1 layer. Now since one of x and y is the ancestor of the other, x and y should differ by exactly 1 layer. Therefore, the edge e = (x, y) should be in the BFS tree T. This contradicts the assumption. Therefore, G cannot contain any edges that do not belong to T.

Rubrics :

- 4 pts : Correctly utilizing the fact that T is a DFS.
- 4pts : Correctly utilizing using the fact that T is a BFS

Q10. Solve Kleinberg and Tardos, **Chapter 3, Exercise 9.** (10 points)

Solution : We run BFS starting from node s. Let *d* be the layer in which node t is encountered; by assumption, we have *d*>n/2. We claim first that one of the layers *L1, L2*, ..., *Ld*-1 consists of a single node. Indeed, if each of these layers had size at least two, then this would account for at least 2(n/2) = *n* nodes; but *G* has only n nodes, and neither *s* nor t appears in these layers.

Thus, there is some layer *L,* consisting of just the node v. We claim next that deleting v destroys all s-t paths. To see this, consider the set of nodes X = {s} U *Z1* U *Z2* U · · · U *Li*−1 · Node s belongs to X but node *t* does not; and any edge out of X must lie in layer *Li*, by the properties of BFS. Since any path from s to t must leave X at some point, it must contain a node in *Li*; but v is the only node in *L*

Rubrics :

- 3 pts : Correctly using BFS to find the shortest path between s and t
- 5 pts: Proving that removing a node destroys all s-t path
- 2 pts: Proving that the algorithm runs in O(m+n)