CSE 431/531: Algorithm Analysis and Design Fall 2022

Homework 1

Instructor: Shi Li **Deadline: 9/25/2022**

Your Name: Zheyuan Ma Your Student ID: 50321597

Problems	1	2	3	Total
Max. Score	20	30	30	80
Your Score				

Problem 1. Asymptotic Notations.

(1a) For each pair of functions $f(n)$ and $g(n)$ in the following table, indicate whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$ and $f(n) = \Theta(g(n))$ respectively.

$f(n)$	$g(n)$	O	Ω	Θ
$\log_2(n^3)$	$10\log_2(\sqrt{n})$	✓	✓	✓
$5n^2 + n$	$n \log n$	✗	✓	✗
$10n^2 + n + 10$	n^3	✓	✗	✗
e^n	2^{2n}	✓	✗	✗

(1b) Prove $\lceil 10n\sqrt{n} \rceil = O(n\sqrt{n})$ using the definition of the O-notation.

In the following two problems, we assume every vertex is incident to at least one edge. So we have $n = O(m)$. Then the running time $O(n + m)$ on the slides becomes $O(m)$.

Problem 2: Cycle detection in (undirected) graphs A cycle in an *undirected* graph $G = (V, E)$ is a sequence of $t \geq 3$ *different* vertices $v_1, v_2, \cdots, v_t$ such that $(v_i, v_{i+1}) \in E$ for every $i = 1, 2, \cdots, t-1$ and $(v_t, v_1) \in E$. Given the linked-list representation of an (undirected) graph $G = (V, E)$, design an $O(m)$-time algorithm to decide if G contains a cycle or not; if it contains a cycle, output one (you only need to output one cycle). To output the cycle, you can just output $v_1, v_2, \cdots, v_t$.

If the correctness of the algorithm is easy to see from your pseudo-code, then there is no need to prove the correctness separately. However, you should briefly mention why the algorithm runs in time $O(m)$.

Problem 1

1b) For every $n \geq 1$, we have

$$\lceil 10n\sqrt{n} \rceil \leq 10n\sqrt{n} + 1 \leq 10n\sqrt{n} + n\sqrt{n} = 11n\sqrt{n}$$

So, for every $n \geq 1$, we have $\lceil 10n\sqrt{n} \rceil \leq 11n\sqrt{n}$.

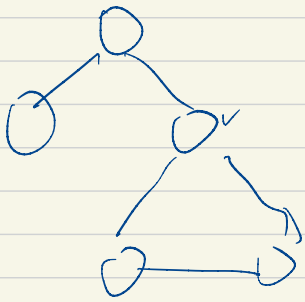
So, $\lceil 10n\sqrt{n} \rceil = O(n\sqrt{n})$.

Problem 2

We can use DFS to detect cycles while maintaining a parent variable, and a list for traversed vertices

mark all vertices as "unvisited"

```
cycle_detect(v)
1: mark v as "visited"
2: append v to traversed[]
3: for all neighbours u of v do
4:     if u is "unvisitd"
5:         parent[u] = v
6:         cycle_detect(u)
7:     else if parent[v] ≠ u
8:         print("Cycle detected!")
9:         print(from traversed[u] to traversed[v]) and exit
```



This algorithm will only visit all the vertices and edges once, so the running time would be $O(n+m) = O(m)$

Problem 3: Cycle detection in directed graphs A cycle in a *directed* graph $G = (V, E)$ is a sequence of $t \geq 2$ *different* vertices $v_1, v_2, \cdots, v_t$ such that $(v_i, v_{i+1}) \in E$ for every $i = 1, 2, \cdots, t-1$ and $(v_t, v_1) \in E$. Given the linked-list representation of a directed graph $G = (V, E)$, design an $O(m)$-time algorithm to decide if G contains a cycle or not; if it contains a cycle, output one (you only need to output one cycle). To output the cycle, you can just output $v_1, v_2, \cdots, v_t$.

If the correctness of the algorithm is easy to see from your pseudo-code, then there is no need to prove the correctness separately. However, you should briefly mention why the algorithm runs in time $O(m)$.

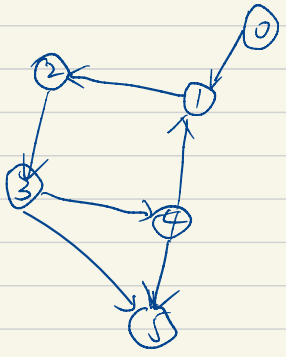
Figure 1: Cycles in undirected and directed graphs are denoted as red edges. (1, 2, 5, 3) is a cycle in the undirected graph. (1, 2, 5, 6, 7, 3) is a cycle in the directed graph. However, (1, 2, 5, 8, 3) is not a cycle in the directed graph.

Remark In a cycle of a directed graph, the directions of the edges have to be consistent. See Figure 1. So, converting a directed graph to a undirected graph and then using algorithm for Problem 2 does not give you a correct algorithm for Problem 3.

Problem 3

We can first use the topological sort algorithm on the graph, and then remove all the vertices that have zero in-degrees. After that, we start from any vertex in the remaining graph, and traverse back along edges until we visit a vertex second time. The vertices in between two visits form a cycle.

```
1:  let d[v]←0 for every v∈V
2:  for every v∈V do
3:      for every u that (u,v)∈E do
4:          d[u] ← d[u]+1
5:  S← {v: d[v]=0}, i←0
6:  while S≠ϕ do
7:      v ← arbitrary vertex in S, S← S\{v}
8:      i ← i+1
9:      for every u that (v,u)∈E do
10:         d[u]← d[u]-1
11:         if d[u]=0 then add u to S
12: if i=n then output "no cycles" and exit
13: let traversed← queue of size n
14: let visited[v]←0 for every v∈V
15: S← {v: d[v]≠0}
16: v← arbitrary vertex in S, i←0
17: while true do
18:     traversed[i]←v, visited[v]←1
19:     for first incoming edge (u,v) of v do
20:         if visited[u] ≠0
21:             visited[u]←visited[u]+1
22:             do print (traversed[i]), i←i-1
23:             until visited[traversed[i]] =2, then exit
24:         else
25:             v=u, i← i+1
```



S	3	2	1	4	3
1	2	1	1	1	

The topological ordering takes time $O(n+m)$. After that, in finding cycles, each vertex will be handled at most once in the while loop (line 17). So those will take $O(n)$. Since $n=O(m)$, the total running time will be $O(m)$.