3.5

1 Problem

For each pair (f, g) below indicate the relation between them in terms of O, Ω, Θ. For each missing entry, write-down Y (for YES) or N (for NO) to indicate whether the relation holds (no need to justify your answers here). For example, if $f = O(g)$ but not $\Omega(g)$, then you should enter Y in the first box and N in the other two boxes. Similarly, if $f = \Theta(g)$, then you should enter Y in all the boxes. [5 points].

f	g	O	Ω	Θ
$\log_2 n$	$\log_{10} n$	Y	Y	Y
$2^{(\log_2 n)^4}$	n^5	~~Y~~	N	N
$n^3 \cdot 2^n$	3^n	~~N~~	~~Y~~	N
$2\sqrt{\log_2 \log_2 n}$	$\log_2 n$	Y	~~Y~~	~~Y~~
$n!$	n^n	Y	N	N

2 Problem

Answer true or false for the following (no need for explanations) [10 points]: 8

- Consider an instance of the stable matching where a doctor D_1's first choice is H_1 and H_1's first choice is D_1. Is it true that in every stable matching D_1 should be matched to H_1?

  Yes, True

- Consider an instance of the stable matching problem and a candidate perfect matching M where one doctor gets her top choice and one hospital gets its top choice, while every other doctor and hospital get their second choice. Is M necessarily a stable matching?

  True

- If α is an n'th root of unity, then $\sum_{j=0}^{2n-1} \alpha^j = 2n$. $a^0 + a$ a^2 a^{2n-1}

  False

- Dijkstra's algorithm when run on an unweighted undirected graph starting from a vertex s gives us (by looking at the graph formed by *Parent* links) the breadth-first-search tree starting from s.

  True

- Any polynomial $P : \mathbb{R} \to \mathbb{R}$ of degree d is uniquely determined by its evaluations at d distinct points $x_1, \ldots, x_d$.

  False

15

3 Problem

Given a connected undirected graph $G = (V, E)$ as input (in adjacency list representation), give an algorithm to check if G is a tree. You must analyze the time-complexity of your algorithm but don't need to prove correctness. For full credit, your algorithm should be correct and run in time $O(|V| + |E|)$. [15 points]

For simple connected graph G, $|V| = |E| + 1$ iff G is a tree

```
//Given list A, with A[i] = i^th vertice's adjacency list
//Count vertices
① numVertices = A.count()
  numEdges = 0
//Count edges
  for i = 0; i < numVertices; i++ {
②     numEdges += A[i].count()
  }
  numEdges = numEdges / 2;    // takes O(1) to right shift
  return numVertices == numEdges + 1
```

If list.count() takes constant time (list size stored as property of list), then $T(n) = O(|V|)$ from the for loop at ② iterating $|V|$ times and all other operations taking $O(1)$.

If list.count() has to be calculated on the fly by looping through the list, then $T(n) = \underset{①}{\text{count vertices}} + \underset{②}{\text{count all edges}}$ where a single list.count() takes $O(|\text{list}|)$. $\Rightarrow T(n) = O(\underset{①}{|V|} + \underset{②}{|E|})$

4 Problem

Given the coefficients of a polynomial P of degree d and an integer k as input, give an algorithm to compute the coefficients of the polynomial $P(x)^k$. For example, if your input is $(1, 1)$ (to denote the polynomial $1 + x$) and $k = 3$, your output should be $(1, 3, 3, 1)$ to denote the polynomial $(1 + x)^3 = 1 + 3x + 3x^2 + x^3$. Similarly, if the input is $(1, -3)$ (to denote $P = 1 - 3x$), $k = 3$, your output should be $(1, -9, 27, -27)$.

To get full credit, your algorithm should be correct, run in time $O((k \cdot d) \log(k \cdot d))$ and you must analyze the time-complexity of your algorithm (no need to prove correctness). [25 points]

Remark: Here we measure time-complexity as in the fast-polynomial multiplication algorithm, where we count complex additions and multiplications as unit-cost.

~~$T(k \cdot d) \leq T[\ldots] + k \cdot d$~~

~~$P(x)^k = P(x) \cdot P(x)^{k-1} = P(x) \cdot P(x) \cdot \ldots \cdot P(x)$~~

~~Multiply $P(x)$ by itself k times, first taking $O(d \log d)$ and then $O((2d+1) \log(2d+1))$ up to $O(dk \log dk)$, added together to equal $O(2dk \log(2dk))$~~

5 Problem

Let $G = (V, E)$ be a directed graph with nodes $v_1, \ldots, v_n$. We say that G is an *ordered graph* if it has the following properties.

1. Each edge goes from a node with a lower index to a node with a higher index. That is, every directed edge has the form (v_i, v_j) with $i < j$.

2. Each node except v_n has at least one edge leaving it. That is, for every node $v_i, i = 1, 2, \ldots, n-1$, there is at least one edge of the form (v_i, v_j).

Given an ordered graph $G = (V, E)$ (in adjacency list representation), give an algorithm to compute the number of paths that begin at v_1 and end at v_n.

You must analyze the time-complexity of your algorithm (no need to prove correctness). To get full-credit your algorithm must be correct and run in time $O(|V| + |E|)$. [25 points]

Remark: You can assume that adding two numbers takes constant time in your time-complexity calculations.

```
numPaths(i) {
    if exists NumPaths[i] return NumPaths[i]
    np = 0
(1) for each edge on graph (vi, vj) in A[i]
(2)     if (getsToN(j)) np += numPaths(j)
    }
    NumPaths[i] = np
    return np
}
return numPaths(1)
```

25

Where A = given adjacency list

getsToN() described later

6 Problem

10

Decide whether the following statement is true or false. If it is true, give a short explanation (no need for a formal proof - a high-level description is enough). If it is false, give a counter-example.

Suppose we are given an instance of the Minimum Spanning Tree Problem on a graph G, with edge costs that are all positive and distinct. Let T be a minimum spanning tree for this instance. Now suppose we replace each edge cost c_e by its square, c_e^2, thereby creating a new instance of the problem with the same graph but different costs.

True or false? T must still be a minimum spanning tree for this new instance. [10 points]

True

If weight$(T) =$ min and $w(T) = \sum_{e \in T} c_e$

then when you replace c_e with c_e^2 for tree T',

$w(T') = \sum_{e \in T} c_e' = \sum_{e \in T} c_e^2 =$ min

because x^2 = strictly increasing for $x > 0$

so if $a < b$, $a^2 < b^2$ $\forall a, b > 0$

7 Problem

Given an undirected graph $G = (V, E)$, a subset of vertices $I \subseteq V$ is an independent set in G if no two vertices in I are adjacent to each other. Let $\alpha(G) = \max\{|I| : I \text{ an independent set in } G\}$. The goal of the following questions is to give an efficient algorithm for computing an independent set of maximum size in a tree. Recall that a *leaf* in a graph is a vertex of degree at most 1 and that every acyclic graph (graph without any cycles) has at least one leaf.

Let $T = (V, E)$ be an acyclic graph on n vertices.

1. Prove that if u is a leaf in T, then there is a maximum-size independent set in T which contains u. That is, for every leaf u, there is an independent set I such that $u \in I$ and $|I| = \alpha(T)$. [15 points]

2. Give the graph T as input (in adjacency edge representation), give an algorithm to compute an independent-set of maximum size, $\alpha(T)$, in T. To get full credit your algorithm should run in time $O(|V| \cdot |E|)$ (or better) and you must prove correctness of your algorithm. You don't need to analyze the time-complexity of your algorithm and it is sufficient to solve this problem assuming part (1) (if you want) even if you don't solve it. [15 points]

1. Assume $\exists$ max-size independent set I' in T. (there always exists an I, even if $|E|=1$)

If $u \in I'$, then our set $= I'$.

If $u \notin I'$, then we can replace v (such that $v \in I'$ and $\exists$ edge (u,v)) with u to make I.

We can do this because u is only connected to v because it is a leaf, so replacing v with u doesn't add any edges to I' to make I.

15