Exercise 1. $D$-ary Tree

A $d$-ary tree is a rooted tree in which each node has at most $d$ children. Show that any $d$-ary tree with $n$ nodes must have a depth of $\Omega\left(\frac{\log(n)}{\log(d)}\right)$ depth. Can you give a precise formula for the minimum depth that it could possibly have?

Solution:
Exercise 2. Harmonic Series

Unlike a decreasing geometric series, the sum of the harmonic series 1, 1/2, 1/3, 1/4, 1/5 ... diverges; that is,

\[ \sum_{i=1}^{\infty} \frac{1}{i} = \infty. \]

It turns out that for large \( n \), the sum of the first \( n \) terms of the above series can be well approximated as

\[ \sum_{i=1}^{n} \frac{1}{i} \approx \ln(n) + \gamma. \]

where \( \ln \) is the natural logarithm and \( \gamma \) is a particular constant 0.57721... Show that

\[ \sum_{i=1}^{n} \frac{1}{i} = \Theta(\log(n)). \]

(Hint: To show an upper bound, decrease each denominator to the next power of two. For a lower bound, increase each denominator to the next power of two.)

Solution:
Exercise 3. Runtime for Multiplication

(DPV 1.7) How long does the recursive multiplication algorithm (page 25 of DPV) take to multiply an $m$-bit number by an $n$-bit number? Justify your answer.

Solution:
Exercise 4. Modular Arithmetic

(DPV 1.10) Show that if \( a \equiv b \pmod{N} \) and if \( M \) divides \( N \) then \( a \equiv b \pmod{M} \).

Solution:
Exercise 5. Depth-First Search 1

(DPV 3.1) Perform a depth-first search on the following graph; whenever there is a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge or a back edge, and give the pre and post number of each vertex.

Solution:
Exercise 6. Depth-First Search 2

(DPV 3.2) Perform a depth-first search on each of the following graphs; whenever there is a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge, back edge, forward edge, or cross edge and give the $pre$ and $post$ number of each vertex.

Solution:
Problem 1. Asymptotic Notation

i. \( f(n) = n \cdot \log(n) \), \( g(n) = n^2 \)

Solution:

ii. \( f(n) = n^3 + 10 \cdot n^2 + n \), \( g(n) = 100 \cdot n^3 \)

Solution:

iii. \( f(n) = n \cdot \log(n) \), \( g(n) = n \cdot \sqrt{n} \)

Solution:

iv. \( f(n) = \log^2(n) \), \( g(n) = \log^3(n) \)

Solution:

v. \( f(n) = \log_a(n) \), \( g(n) = \log_b(n) \) where \( a, b > 0 \) and \( a, b \neq 1 \)

Solution:

vi. \( f(n) = 2^n \cdot n^2 \), \( g(n) = 2^{2n} \)

Solution:

vii. \( f(n) = e^n \), \( g(n) = n! \)

Solution:

viii. \( f(n) = n^a \), \( g(n) = b^n \) where \( a > 0, b > 1 \)

Solution:
Problem 2. Strongly Connected Components

A— Run the DFS-based topological ordering algorithm on the following graph. Whenever you have a choice of vertices to explore, always pick the one that is alphabetically first.

(a) Indicate the pre and post numbers of the nodes.

Solution:

(b) What are the sources and sinks of the graph?

Solution:

(c) What topological ordering is found by the algorithm?

Solution:

(d) How many topological orderings does this graph have?

Solution:

B— Run the strongly connected components algorithm on the following directed graphs $G$. When doing DFS on $G^R$: whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.

(i)

(ii)

In each case answer the following questions.

(a) In what order are the strongly connected components (SCCs) found?
(b) Which are source SCCs and which are sink SCCs?

Solution:

(c) Draw the “metagraph” (each meta-node is an SCC of $G$).

Solution:

(d) What is the minimum number of edges you must add to this graph to make it strongly connected?

Solution:
Problem 3. Bipartite Graph

A bipartite graph is a graph $G = (V, E)$ whose vertices can be partitioned into two sets ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$) such that there are no edges between vertices in the same set (for instance, if $u, v \in V_1$, then there is no edge between $u$ and $v$).

1. Give a linear-time algorithm to determine whether an undirected graph is bipartite.

   **Solution:**

2. There are many other ways to formulate this property. For instance, an undirected graph is bipartite if and only if it can be colored with just two colors.
   Prove the following formulation: an undirected graph is bipartite if and only if it contains no cycles of odd length.

   **Solution:**

3. At most how many colors are needed to color in an undirected graph with exactly one odd-length cycle?

   **Solution:**
Problem 4. DFS Using Stack

Rewrite the \textit{explore} procedure (DPV Figure 3.3) so that it is non-recursive (that is, explicitly use a stack). The calls to \textit{previsit} and \textit{postvisit} should be positioned so that they have the same effect as in the recursive procedure.

\begin{solution}

\end{solution}
Problem 5. Minimize Cost

You are given a directed graph in which each node \( u \in V \) has an associated price \( p_u \), which is a positive integer. Define the array \( cost \) as follows: for each \( u \in V \),

\[
    cost[u] = \text{price of the cheapest node reachable from } u \text{ (including } u \text{ itself)}
\]

For instance, in the graph below (with the prices shown for each vertex), the \( cost \) value of the nodes \( A, B, C, D, E, F \) are 2, 1, 4, 1, 4, 5 respectively.

Your goal is to design an algorithm that fills the entire \( cost \) array (i.e. for all vertices).

1. Give a linear time algorithm that works for directed acyclic graphs. (Hint: Handle the vertices in a particular order).

Solution:

2. Extend this to a linear time algorithm that works for all directed graphs. (Hint: Recall the “two-tiered” structure of directed graphs).

Solution:
Problem 6. Paths in a DAG

Give an efficient algorithm that takes as input a directed acyclic graph \( G = (V, E) \) and two vertices \( s, t \in V \) and outputs the number of different directed paths from \( s \) to \( t \) in \( G \).

Solution:
Problem 7. A Greedy Heuristic for the Traveling Salesperson Problem

You are given a set of \( n \) points \((x_i, y_i)\) for \( i = 1, 2, \ldots, n \) respectively corresponding to cities \( c_1, c_2, c_3, \ldots, c_n \) in the Euclidean plane. The travel distance between any pair of cities is the Euclidean distance between the corresponding points. City \( c_1 \) located at \((x_1, y_1)\) is your home city.

The Traveling Salesperson Problem (TSP) seeks to find a Hamiltonian cycle, or “tour”, over the \( n \) cities (i.e., a closed path that begins and ends at \( c_1 \), and that visits every other city exactly once) that has minimum total travel distance.

Consider the following “greedy” approach to finding a minimum-cost tour over the \( n \) cities:

- Initialize all cities to “unvisited”.
- Set \( \text{current\_city} = c_1 \); mark \( c_1 \) as “visited”.
- For \( j = 1, \ldots, n - 1 \):
  - Travel from \( \text{current\_city} \) to the nearest city, \( \text{city\_next} \), that is “unvisited”.
  - Mark \( \text{city\_next} \) as “visited”.
  - Set \( \text{current\_city} = \text{city\_next} \)
- Travel from \( \text{current\_city} \) to \( c_1 \).

1. Show that this “greedy” approach to TSP is not optimal by exhibiting a small counterexample.

**Solution:**

2. How badly suboptimal do you think the greedy approach can be, relative to optimal? Please clearly explain your definition of “badly suboptimal”.

**Solution:**