Homework Assignment: Implementation Phase

Building on your preparatory work from last week, where you outlined procedures for solving multiple-choice questions (**Week 1-2 shorts**) and devised algorithms for a programming problem (**Week 2-2 Programming task**), this week's assignment shifts focus towards actual implementation and problem-solving. Your task is to apply the strategies and algorithms you've developed to solve the given problems and implement the corresponding solutions.

Task:

Part 1: Multiple-Choice Questions (Week 1-2 Shorts)

1. Revisit your outlined procedures for the multiple-choice questions (Week 1-2 Shorts).
2. Apply these procedures to solve each question.
3. For each question, provide:
    - Your selected answer.
    - A brief explanation detailing why you chose this answer, referencing your initial procedure.

Part 2: Programming Problem (Week 2-2 Programming)

Task: Implement the algorithm you designed for the Week 2-2 Programming problem in your preferred programming language.

Exploring Graph Traversal Algorithms vis Recursion

Objectives

In this assignment, you will explore graph traversal by implementing two fundamental algorithms: Depth-First Search (DFS) and Breadth-First Search (BFS) within a given `Graph` class skeleton.

Tasks

DFS Implementation:

- Complete the `dfs_help` function within the `dfs` method to perform a recursive depth-first traversal.
- Ensure each node is visited exactly once and nodes are printed in the order they are visited.

BFS Implementation (Challenge):

- Implement the `bfs_help` function within the `bfs` method to perform a recursive breadth-first traversal. This is unconventional and will require creative thinking to manage levels of nodes.
- Note: Managing each level's nodes recursively is key. Consider how you can track and process nodes level by level using recursion.

Testing

Provide test cases for your implementations to ensure correctness. Your tests should cover:

- Empty graphs.
- Single-node graphs.
- Linear graphs.
- Graphs with cycles.
- Tree structures.
- Disconnected graphs.

Requirements

- Comment your code thoroughly to explain your logic.
- Adhere to coding standards discussed in class.
- Handle special cases, such as graphs with cycles or disconnected components, appropriately.

Recursion HW:

1. Given **base** and **n** that are both 1 or more, compute **recursively** (no loops) the value of base to the **n power**, so powerN(3, 2) is 9 (3 squared).
    a. Sample Inputs:
        i. powerN(3, 1) → 3
        ii. powerN(3, 2) → 9
        iii. powerN(3, 3) → 27
2. Given a **string**, compute recursively a new string where all the 'x' chars have been removed.
    a. Sample Inputs:
        i. noX("xaxb") → "ab"
        ii. noX("abc") → "abc"
        iii. noX("xx") → ""
3. Given a **string** that contains **a single pair of parenthesis**, compute recursively a new string made of only the parenthesis and their contents, so "xyz(abc)123" yields "(abc)".
    a. Sample Inputs:
        i. parenBit("xyz(abc)123") → "(abc)"
        ii. parenBit("x(hello)") → "(hello)"
        iii. parenBit("(xy)1") → "(xy)"
4. Given a **string** and a non-empty substring **sub**, compute recursively the largest substring which starts and ends with sub and returns its length.
    a. Sample Inputs:
        i. strDist("catcowcat", "cat") → 9
        ii. strDist("catcowcat", "cow") → 3
        iii. strDist("cccatcowcatxx", "cat") → 9
5. Given an **array of ints**, is it **possible** to choose a group of some of the ints, such that the group sums to the given target? Return a boolean determining the possibility.
    a. This is a classic **backtracking** recursion problem. Once you understand the recursive backtracking strategy in this problem, you can use the same pattern for many problems to search a space of choices.
    b. **Hint**: Rather than looking at the whole array, our convention is to consider the part of the array starting at **index start** and continuing to the end of the array. The caller can specify the whole array simply by passing **start** as 0.
    c. Sample Inputs:
        i. groupSum(0, [2, 4, 8], 10) → true
        ii. groupSum(0, [2, 4, 8], 14) → true
        iii. groupSum(0, [2, 4, 8], 9) → false
6. Given an array of ints, is it possible to choose a group of some of the ints, beginning at the start index, such that the group sums to the given target? However, with the additional constraint that all 6's must be chosen. (No loops needed)
    a. Sample Inputs:
        i. groupSum6(0, [5, 6, 2], 8) → true
        ii. groupSum6(0, [5, 6, 2], 9) → false
        iii. groupSum6(0, [5, 6, 2], 7) → false