Map of Tasks

Problem description

The International Informatics Olympiads in Teams (IIOT) pose a unique challenge in the field of programming.

In this specific problem, the goal is to minimize the total time required to complete a set of assigned tasks.

Each task depends on another, forming a tree structure of tasks, with the possibility of using a limited number of "cheats" to reduce the completion time of a task.

Algorithm pseudocode

```
Function cheat(node, remaining_cheats)
    // Base case: If the node is null, return 0
    If node is NULL Then Return 0

    // Check if the result is already computed (Memoization)
    If node.cache[remaining_cheats] is not -1 Then Return node.cache[remaining_cheats]

    // Initialize the lowest time to a very large number
    Set lowest to a very large number

    // Iterate over all possible ways to distribute the cheats
    For cheatsUsed from remaining_cheats down to 0
        // Calculate the cost of completing the child subtree without cheating on the current task
        If cheatsUsed > 0 Then Set costChildNoHead to cheat(node.firstChild, cheatsUsed - 1)
        Else Set costChildNoHead to a very large number

        // Calculate the cost of completing the child subtree including the current task's time
        Set costChild to cheat(node.firstChild, cheatsUsed) + node.value

        // Determine the best cost for the subtree
        // This step compares the cost of completing the subtree including the current node (costChild)
        // versus completing the subtree without including the current node (costChildNoHead)
        // The minimum of these two represents the lowest cost to complete this part of the subtree
        Set bestSubtree to min(costChild, costChildNoHead)

        // Calculate the cost for the sibling nodes with the remaining cheats
        Set siblingCost to cheat(node.sibling, remaining_cheats - cheatsUsed)

        // Determine the highest cost between the current node and its siblings
        // We compare the lowest cost to complete the current subtree (bestSubtree)
        // with the cost to complete the sibling tasks (siblingCost)
        // The maximum of these two is taken because tasks can be performed in parallel,
        // and the longer task set determines the overall time
        Set highestLevelCostSubtask to max(bestSubtree, siblingCost)

        // Update the lowest cost found so far
        lowest = min(lowest, highestLevelCostSubtask)

    // Memoize the computed value
    node.cache[remaining_cheats] = lowest

    Return lowest
End Function
```

Time complexity analysis

- Creating the tree has a complexity of O(N), where N is the number of nodes.
- The `cheat` function has a time complexity of O(N*C) in normal cases, and $O(N*C^2)$ in the worst case: this is because, for each node, it examines all possible combinations of cheats up to the maximum limit (C), and each recursive call can involve up to C iterations.

Test

Status
100 / 100

Known issues

- **Scalability**: The algorithm may become less efficient with a very high number of tasks or cheats.
- **Optimization**: Some parts of the algorithm could be optimized for better performance.