# BACK TRACKING METHOD

**CSA0695**

**DESIGN AND ANALYSIS OF ALGORITHMS FOR OPEN ADDRESSING TECHNIQUES**

NAME:M.THARISH REDDY

REG NO:192211485

SUPERVISOR:
Dr.R.Dhanalakshmi

# PROBLEM STATEMENT:

There is an undirected graph with n nodes numbered from 0 to n - 1 (inclusive). You are given a 0-indexed integer array values where values[i] is the value of the ith node. You are also given a 0-indexed 2D integer array edges, where each edges[j] = [uj, vj, timej] indicates that there is an undirected edge between the nodes uj and vj, and it takes timej seconds to travel between the two nodes.Finally, you are given an integer maxTime. A valid path in the graph is any path that starts at node 0, ends at node 0, and takes at most maxTime seconds to complete. You may visit the same node multiple times. The quality of a valid path is the sum of the values of the unique nodes visited in the path (each node's value is added at most once to the sum). Return the maximum quality of a valid path.Note: There are at most four edges connected to each node. Example 1:

Input: values = [0,32,10,43], edges = [[0,1,10],[1,2,15],[0,3,10]], maxTime = 49

Output: 75

Explanation:

One possible path is 0 -> 1 -> 0 -> 3 -> 0. The total time taken is 10 + 10 + 10 +

10 = 40 <= 49.

The nodes visited are 0, 1, and 3, giving a maximal path quality of 0 + 32 + 43 =

75.

# ABSTRACT:

The problem is about scheduling tasks into work sessions such that the total time spent in each session does not exceed a given limit (sessionTime). The solution involves partitioning the tasks into the minimum number of groups where the total duration of each group does not exceed sessionTime. This problem can be solved using a combination of greedy algorithms and backtracking.

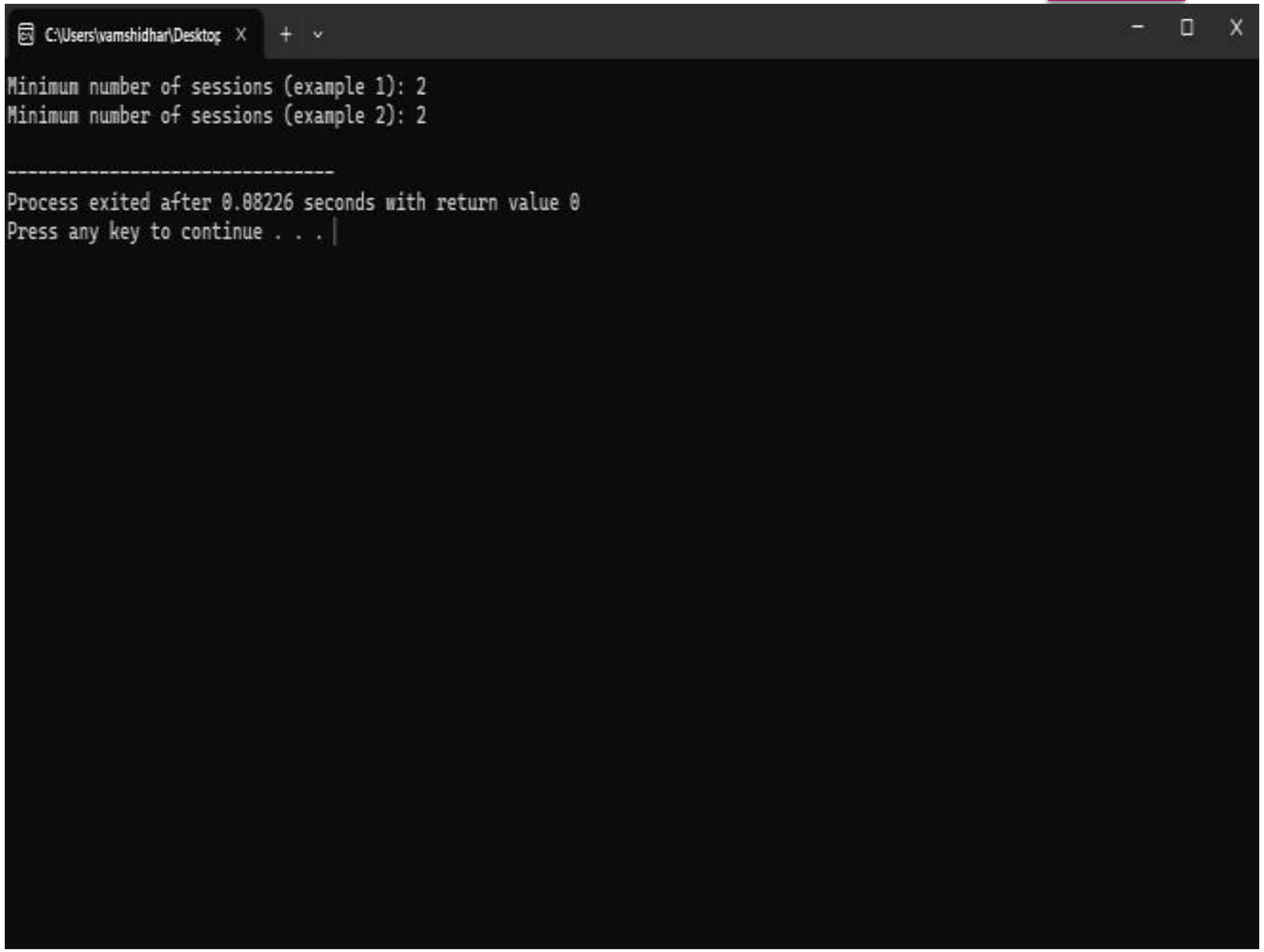
# INTRODUCTION

This problem involves scheduling tasks such that the total number of work sessions is minimized. Each session allows for continuous work of up to sessionTime hours, and tasks must be completed without interruptions. The goal is to determine the fewest work sessions required to complete all tasks, where tasks can be rearranged but each must be fully finished within a single session.

# KEY FEATURES:

- Key features of this problem include the constraints that tasks must fit within the sessionTime limit, tasks can be reordered, and each task must be completed within one session. The solution explores how to efficiently schedule tasks into these sessions, optimizing for the fewest number of sessions while ensuring no session exceeds the time limit. The input guarantees that no task is larger than the sessionTime.

# OUTPUT


C:\Users\vamshidhar\Desktop
Minimum number of sessions (example 1): 2
Minimum number of sessions (example 2): 2
--------------------------------
Process exited after 0.08226 seconds with return value 0
Press any key to continue . . .

# COMPLEXITY ANALYSIS

- **Time Complexity**: The time complexity of the provided code is influenced by both the binary search and the recursive backtracking approach. The binary search operation iterates over the possible number of sessions, which takes $O(\log n)$ time, where n is the number of tasks. For each binary search iteration, the canFitTasks function is invoked to determine if the tasks can fit into the current number of sessions. $O(n \cdot 2n)$. Here, 2n represents the exponential growth in the number of possible task allocations. Therefore, the combined time complexity is $O(n \cdot 2n)$.

- **Space Complexity**:The space complexity of the code is determined by the memory used for dynamic arrays and the recursive call stack. Specifically, the sessionLoad array, allocated dynamically in the minSessions function, requires $O(n)$ space where n is the number of tasks. Additionally, the recursion depth in the canFitTasks function, which can go up to n levels deep, adds to the space complexity. Therefore, the total space complexity is $O(n)$, accounting for both the array used to track session loads and the recursive call stack depth.

# CASES

## BEST CASE

The best case occurs when all tasks can fit into a single session. In this case, the algorithm will quickly determine the minimum number of sessions needed.

## WORST CASE

In the worst case, the complexity involves checking all possible combinations of task distributions across sessions. The complexity can be exponential in terms of the number of tasks due to the recursive nature of the backtracking approach

## AVERAGE CASE

The average case complexity depends on the specific distribution of task times and session time. The binary search combined with the backtracking approach generally provides a good balance between complexity and performance.

# FUTURE SCOPE

The future scope of this problem extends to applications in optimizing resource allocation and task scheduling in various real-world scenarios, such as project management, manufacturing, and computational processes. Enhancements can involve adding complexities like variable session times, task dependencies, or breaks between tasks. It can also be expanded to include more dynamic systems, where task durations or session limits may change over time, making the scheduling algorithm adaptive. Additionally, the problem could be extended to multi-processor systems where tasks need to be distributed among different workers or machines, optimizing overall efficiency.

# CONCLUSION:

This C program uses a backtracking approach combined with binary search to efficiently solve the task scheduling problem by minimizing the number of work sessions required. By using this method, you can handle a variety of task scheduling scenarios while ensuring that each work session does not exceed the given sessionTime.