CAPSTONE PROJECT

Optimizing Task Scheduling

Name: V. Laxmi Nivas

Reg no: 192211694

Subject: CSA0656-Design Analysis and Algorithms for Asymptotic Notations

Guided by: Dr. R. Dhanalakshmi

Proffessor Departement : Department of Machine Learning

This project report explores the problem of minimizing work sessions required to complete a set complete a set of tasks.

Subset Sum Backtracking

The study employs a subset sum backtracking approach to tackle this problem.

Algorithm

Systematically explores different combinations of tasks.

Feasible Partitions

Finds partitions that adhere to the the session time constraint.

Optimal Task Distribution

Identifies the optimal task distribution that results in the fewest fewest number of work sessions. sessions.

Benefits of the Approach

The approach ensures tasks are completed efficiently and provides insights into balancing task distribution and session utilization.

1

Improved Scheduling Efficiency

Offers practical solutions for complex task management scenarios. scenarios.

2

Maximized Productivity

Minimizing the number of work sessions is essential for maximizing productivity.

3

Resource Utilization

Optimizes resource utilization by efficiently allocating tasks.

Algorithm Explanation

A backtracking algorithm solves problems by building a solution piece by piece.

1

Initial Feasibility

Compute the total sum of all tasks and calculate the minimum number of sessions needed.

2

Define Variables

Let n be the number of tasks, tasks be the array of task times, and sessionTime be the maximum allowable session time.

3

Create an Array to Track Session Times

Initialize an array sessions of length k to keep track keep track of the total time used in each session. session.

4

Backtracking Algorithm

Define a function assignTask(taskIndex, currentSessions) that tries to assign the task at at taskIndex to one of the currentSessions.

5

Optimization

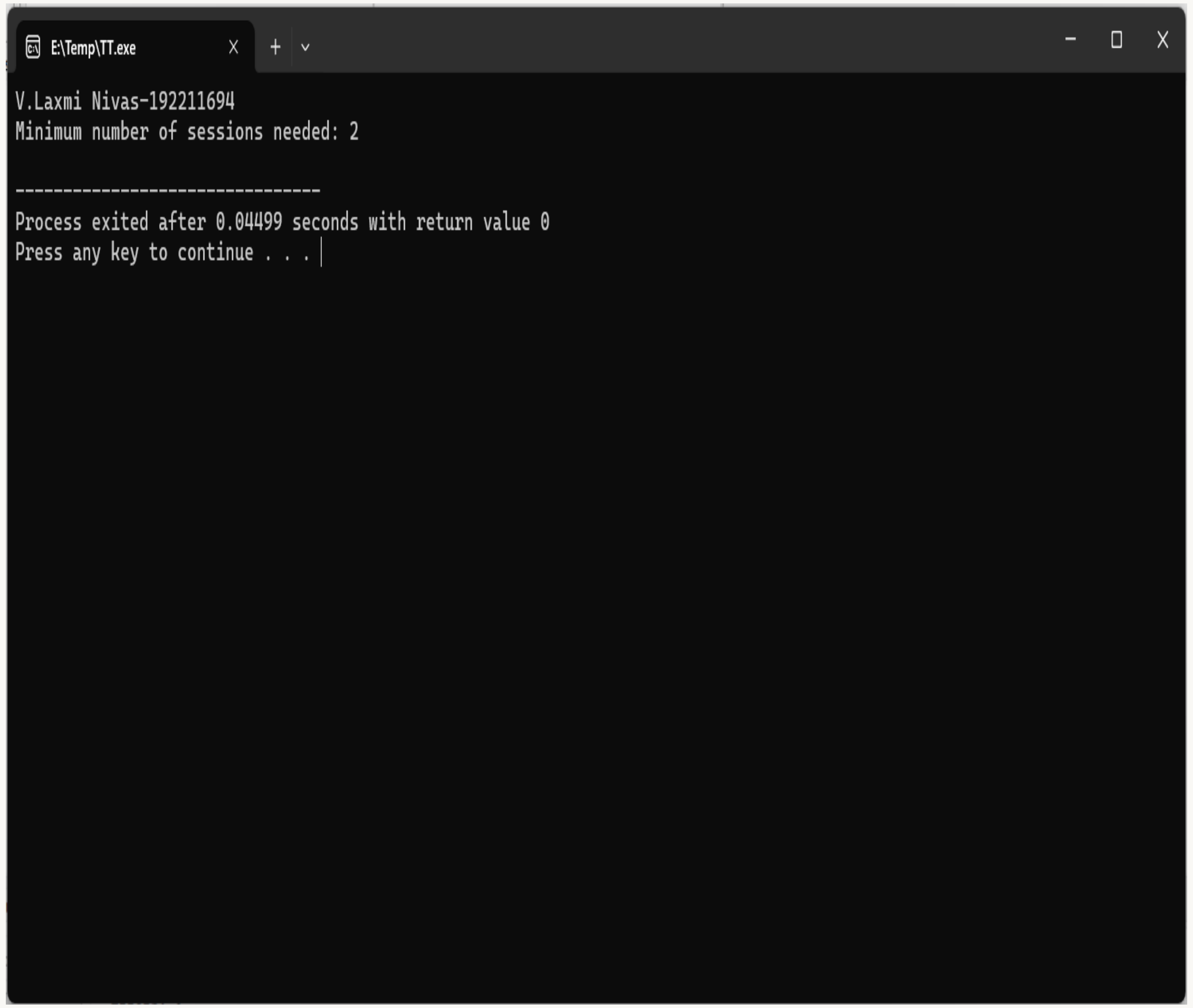
To find the minimum number of sessions, iterate iterate through possible session counts starting starting from the computed lower bound and and check if the tasks can be assigned within that within that number of sessions.

Example Calculation

For tasks = \[3, 1, 3, 1, 1\] and sessionTime = 8.

Step 1	Total time T = 3 + 1 + 3 + 1 + 1 = 9.
Step 2	Minimum number of sessions = ceil(9 / 8) = 2. = 2.
Step 3	Initialize sessions array for 2 sessions: sessions = \[0, 0\].
Step 4	Assign Tasks: Start with the largest task first (3 hours).
Step 5	Try placing 3 in session 1: sessions = \[3, 0\].
Step 6	Next, place 3 in session 2: sessions = \[3, 3\].
Step 7	Place remaining tasks to fill sessions.
Step 8	Check Feasibility: Ensure all tasks fit within the sessionTime of 8 hours and that the total number of sessions used is minimized.

OUTPUT:-



Code Explanation

The provided C code effectively uses a backtracking approach to solve the problem of minimizing the number of work sessions required to complete a set of tasks within a given session time limit.

1

Initialization

Define the maximum number of tasks, session time, and an array to store task times.

2

Recursive Calls

The algorithm explores all possible ways to assign n tasks into sessions.

3

Backtracking Function

Base Case: If all tasks are assigned, update minSessions with the current number of sessions if it is fewer than previously recorded.

4

Main Function

Initialize task array and session time.

Time Complexity

The time complexity of the provided backtracking algorithm is exponential, specifically $O(2^n)$, where n is the where n is the number of tasks.

Exponential Time Complexity

The algorithm explores all possible ways to assign n tasks into sessions.

Practical for Small Task Sets

The approach is practical for small to moderate-sized task sets.

Optimization Techniques

For larger problems, optimization techniques or approximation algorithms might be necessary.

Conclusion

The provided C code effectively uses a backtracking approach to solve the problem of minimizing the number of work sessions required to complete a set of tasks within a given session time limit.

Recursive Exploration

The algorithm recursively explores different task allocations and leverages pruning to eliminate infeasible configurations.

Optimal Number of Sessions

The algorithm aims to find the optimal number of sessions.

Practical for Small Task Sets

The approach is practical for small to moderate-sized task sets.

Optimization Techniques

For larger problems, optimization techniques or approximation algorithms might be necessary.