SIMATS SCHOOL OF ENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

CHENNAI-602105

MINIMUM NUMBER OF WORK SESSIONS USING BACKTRACKING

A CAPSTONE PROJECT REPORT

Submitted in the partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE ENGINEERING

Submitted by

S.EASWARI CHARITHA (192210349)

P. DEVIRAMYA (192210261)

C. SRUJANA (192210298)

Under the Supervision of

JOY PRIYANKA.M

TABLE OF CONTENTS

AIM:

The aim of the problem is to determine the minimum number of work sessions required to finish a given set of tasks, considering that each task must be completed within a specified session time. The conditions for completing tasks are: Start a task in a work session and complete it within the same session. Start a new task immediately after finishing the previous one. Tasks can be completed in any order.

ABSTRACT :

This paper addresses the problem of determining the minimum number of work sessions needed to finish a set of tasks, considering constraints on session duration and task completion rules. The problem involves a given array of task times and a session time limit. Tasks must be completed within a single work session and can be started immediately after finishing the previous task. Additionally, tasks can be completed in any order. Our approach utilizes a greedy algorithm to optimize work session allocation while ensuring task completion within session constraints. We explore various strategies for assigning tasks to work sessions to minimize the overall number of sessions required. Experimental results demonstrate the effectiveness of our approach in efficiently managing work sessions, thereby enhancing task completion efficiency and meeting project deadlines.

Keywords: Work Session Management, Task Completion, Greedy Algorithm, Session Time Constraints, Efficiency Optimization.

INTRODUCTION :

Efficient task management is essential in modern work environments to ensure productivity and timely project completion. A key aspect of task management is the allocation of work sessions, where tasks are completed within specified time constraints. This paper addresses the optimization of work session management by minimizing the number of sessions required to complete a given set of tasks, while adhering to session time limits and task completion rules.

The problem at hand involves a scenario where tasks have varying time requirements, and each task must be completed within a single work session. Furthermore, tasks can be started immediately after finishing the previous task, and the order of task completion is flexible. The primary objective is to design an algorithm that efficiently allocates tasks to work sessions, ensuring optimal utilization of session time and minimizing idle time between tasks.

In this paper, we present a greedy algorithm approach to tackle the work session management problem. We explore various strategies for assigning tasks to work sessions, considering session time constraints and task completion rules. Our goal is to develop a solution that optimizes work session allocation, thereby enhancing overall task completion efficiency and meeting project deadlines.

Through experimental analysis and comparison with existing methods, we aim to demonstrate the effectiveness and efficiency of our approach in managing work sessions and improving task completion workflows.

CASE DESCRIPTION:

Scenario Overview: In a software development company, a team of developers is assigned various tasks with different time requirements. The team follows a work session-based approach, where each developer works on tasks within specific session time limits. Tasks must be completed within a single work session, and developers can start a new task immediately after finishing the previous one. The goal is to minimize the number of work sessions required to complete all tasks efficiently.

Task Details:

Task A: 3 hours

Task B: 2 hours

Task C: 1 hour

Task D: 4 hours

Session Time Limit: 5 hours

Task Completion Rules: Tasks must be completed within a single work session. Developers can start a new task immediately after finishing the previous one.

Objective: Design an algorithm to allocate tasks to work sessions optimally, considering the session time limit and task completion rules. The algorithm should minimize the number of work sessions needed to complete all tasks.

Constraints:

Each task must be completed within a single work session.

Session time limit is 5 hours.

Expected Output:

The algorithm should output the minimum number of work sessions required to finish all tasks efficiently. For example, in the given scenario, the optimal solution would be to allocate Task A and Task B in the first work session (3 + 2 = 5 hours), and Task C and Task D in the second work session (1 + 4 = 5 hours), totalling two work sessions.

METHODS:

Input Data Acquisition: Obtain the list of tasks along with their respective time requirements.

Determine the session time limit within which tasks must be completed.

Initialization: Initialize variables to track work sessions, such as current session time and session count. Create data structures to store task completion status and session allocation.

Greedy Algorithm Approach:

Sort the tasks in descending order based on their time requirements. This step ensures that longer tasks are prioritized for allocation.

Iterate through the sorted tasks list:

For each task, check if it can be accommodated within the current work session. If the task fits within the session time limit, allocate it to the current session and update the session time. If the task exceeds the session time limit, start a new work session and allocate the task to it.

Optimal Work Session Allocation:

Continue iterating through the tasks list until all tasks are allocated to work sessions.

Track the number of work sessions required to complete all tasks efficiently.

Output Generation: Output the minimum number of work sessions needed to finish all tasks based on the algorithm's allocation strategy.

Performance Evaluation: Evaluate the algorithm's performance in terms of efficiency and effectiveness in minimizing work sessions while adhering to task completion rules.

Compare the algorithm's results with alternative approaches or baseline methods to assess its optimization capabilities.

RESULTS:

The algorithm successfully minimized the number of work sessions required to complete all tasks efficiently. The output indicated that only two work sessions were needed: Minimum Number of Work Sessions: 2.

This result demonstrates the effectiveness of the algorithm in optimizing work session allocation while meeting session time constraints and task completion rules.

Performance Evaluation: Further evaluation of the algorithm's performance in terms of efficiency and scalability could involve testing with larger task sets and varying session time limits. Comparative analysis with alternative algorithms or strategies could also provide insights into the algorithm's optimization capabilities in different scenarios.

DISCUSSION:

Implications of Results: The achieved result of minimizing the number of work sessions required for task completion has significant implications for workflow efficiency and resource utilization. By optimizing work session allocation, organizations can improve productivity, meet project deadlines, and effectively manage task dependencies.

Algorithm Limitations:

The algorithm's effectiveness may decrease with larger task sets or highly variable task time requirements, as the greedy approach may not always yield globally optimal solutions. It relies on sorting tasks based on time requirements, which may not consider other factors such as task dependencies or critical paths.

Future Research Directions: Exploring advanced optimization techniques and algorithms tailored to specific task allocation scenarios. Conducting empirical studies and simulations to

validate algorithm performance under diverse conditions. Integrating real-time monitoring and adaptive scheduling capabilities to handle dynamic task environments.

PROGRAM:

```
#include <iostream>

#include <vector>

#include <climits>

using namespace std;

void backtrack(vector<int>& tasks, vector<int>& times, int index, int sessions, int&
min_sessions, int sessionTime) {

if (index == tasks. Size()) {

min_sessions = min (min_sessions, sessions);

return;

}

for (int i = 0; i < sessions; i++) {

if (times[i] + tasks[index] <= sessionTime) {

times[i] += tasks[index];

backtrack(tasks, times, index + 1, sessions, min_sessions, sessionTime);

times[i] -= tasks[index];

}

}

times[sessions] = tasks[index];

backtrack(tasks, times, index + 1, sessions + 1, min_sessions, sessionTime);

}

int minSessions(vector<int>& tasks, int sessionTime) {

int min_sessions = INT_MAX;
```

```
vector<int> times (tasks. Size(), 0);

backtrack(tasks, times, 0, 0, min_sessions, sessionTime);

return min_sessions;

}

int main() {

vector<int> tasks1 = {1, 2, 3};

cout << minSessions(tasks1, 3) << endl;  // Output: 2

vector<int> tasks2 = {3, 1, 3, 1, 1};

cout << minSessions(tasks2, 8) << endl;  // Output: 2

return 0;

}
```

OUTPUT:

CONCLUSION

The algorithm effectively minimizes work sessions for task completion, optimizing resource utilization and enhancing workflow efficiency. Despite its strengths in small to moderate task sets, scalability and handling highly variable task requirements remain challenges. Incorporating advanced scheduling techniques and considering additional constraints could improve algorithm performance. The results underscore the importance of efficient work session management in meeting project deadlines and improving productivity. Future research should focus on developing more robust optimization strategies for diverse task allocation scenarios.

REFERENCE

1. https://leetcode.com/problems/minimum-number-of-work-sessions-to-finish-the-tasks/description/

2. https://www.geeksforgeeks.org/minimum-work-to-be-done-per-day-to-finish-given-tasks-within-d-days/

3. https://hsiyinl.medium.com/minimum-number-of-work-sessions-to-finish-the-tasks-leetcode-1986-2a25474d9703

4. https://algo.monster/liteproblems/1723

5. https://www.interviewbit.com/courses/programming/backtracking/