A blue gear with a symbol of medical science

Description automatically generated**SIMATS SCHOOL OF ENGINEERING** 

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**CHENNAI-602105**

**“Difference Between Maximum and Minimum Price Sum Using Dynamic”**

**A CAPSTONE PROJECT REPORT**

**BACHELOR OF ENGINEERING**

**IN COMPUTER SCIENCE AND DESIGN AND ANALYSIS OF ALGORITHMS FOR ROUTING**

**Submitted by: S.SIDDHARTHA REDDY**

**Student RegNo :192210598**

**Under the Supervision of**

**Rajagopal K**

**DECLARATION**

I S.SIDDHARTHA REDDY student of Bachelor of Engineering in Computer Science Engineering and DESIGN AND ANALYSIS OF ALGORITHMS FOR ROUTING at Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the work presented in this Capstone Project Work entitled "Title" is the outcome of my own bonafide work. I affirm that it is correct to the best of my knowledge, and this work has been undertaken with due consideration of Engineering Ethics.

**(Name Reg No)**

**S.SIDDHARTHA REDDY**

**192210598**

**Date: 25-06-2024**

**Place: Saveetha School of Engineering, Thandalam.**

**CERTIFICATE**

This is to certify that the project entitled “Difference Between Maximum and Minimum Price Sum Using Dynamic” submitted by S.SIDDHARTHA REDDY has been carried out under my supervision. The project has been submitted as per the requirements in the current semester of B. E Computer science engineering and DESIGN AND ANALYSIS OF ALGORITHMS FOR ROUTING

Faculty-in-charge

**K. Rajagopal**

**ABSTRACT**

This study addresses the problem of task scheduling under constrained work sessions. Given `n` tasks with specified durations represented by the integer array `tasks`, where each task `i` takes `tasks[i]` hours to complete, the objective is to determine the minimum number of work sessions required to finish all tasks. Each work session allows up to `sessionTime` consecutive working hours before a break is mandated. Tasks must be completed within a single work session once started, and tasks can be undertaken in any sequence. The key requirement is that `sessionTime` must be at least as large as the longest task duration in `tasks`.

We propose an approach to solve this problem, ensuring that the minimum number of work sessions is used. The problem is illustrated with examples for clarity:

- \*\*Example 1\*\*: There exists an undirected and initially unrooted tree with n nodes indexed from 0 to n - 1. You are given the integer n and a 2D integer array edges of length n - 1, where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree. Each node has an associated price. You are given an integer array price, where price[i] is the price of the ith node. The price sum of a given path is the sum of the prices of all nodes lying on that path. The tree can be rooted at any node root of your choice. The incurred cost after choosing root is the difference between the maximum and minimum price sum amongst all paths starting at root. Return the maximum possible cost amongst all possible root choices.

This research provides a clear method for optimizing task completion within limited work periods, offering valuable insights for practical applications in project management and operations research.

**KEYWORDS**

Task Scheduling, Work Sessions, Integer Array, Session Time Constraint, Task Completion, Minimizing Work Sessions, Scheduling Algorithm, Project Management, Operations Research

**INTRODUCTION**

Efficient task scheduling is crucial in various domains, ranging from project management to daily operations, where resources must be allocated optimally to complete tasks within specified constraints. In this context, the problem of scheduling `n` tasks, each requiring a certain amount of time to complete, under the constraint of maximum allowable working hours per session (`sessionTime`), becomes significant.

Given an integer array `tasks` where each element represents the hours required to complete a task, and a constraint that `sessionTime` must be greater than or equal to the maximum task duration in `tasks`, the goal is to determine the minimum number of work sessions needed to complete all tasks. Each work session allows uninterrupted work up to `sessionTime` hours, after which a break is mandatory.

This problem involves strategic decision-making to sequence tasks optimally to minimize the number of sessions, while ensuring that once a task is started within a session, it must be completed within that session. This constraint adds complexity to the task allocation process, necessitating an efficient algorithmic approach to derive the optimal solution.

In this study, we explore various algorithms and methodologies to solve the task scheduling problem, aiming to provide insights into effective task management strategies that balance workload distribution and session constraints.

**CODING**

#include <stdio.h>

#include <limits.h>

int findMaxSum(int prices[], int n) {

int maxSum = 0;

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

if (prices[i] > 0) {

maxSum += prices[i];

}

}

return maxSum;

}

int findMinSum(int prices[], int n) {

int minSum = 0;

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

if (prices[i] < 0) {

minSum += prices[i];

}

}

return minSum;

}

int main() {

int prices[] = {4, -1, 2, 1, -7, 5};

int n = sizeof(prices) / sizeof(prices[0]);

int maxSum = findMaxSum(prices, n);

int minSum = findMinSum(prices, n);

printf("Maximum Sum: %d\n", maxSum);

printf("Minimum Sum: %d\n", minSum);

int difference = maxSum - minSum;

printf("Difference Between Maximum and Minimum Sum: %d\n", difference);

return 0;

}

**OUTPUT**

Maximum Sum: 12

Minimum Sum: -8

Difference Between Maximum and Minimum Sum: 20

**Complexity Analysis**

**Best Case**

 The prices array is such that the maximum and minimum sums are straightforward to compute, possibly when all elements are the same or zero.

 Time Complexity**:** O(n log(S)), where n is the number of tasks and S is the sum of all task durations. This is because the binary search iterates logarithmically with respect to S.

**Worst Case**

 The prices array is such that computing the maximum and minimum sums requires considering all elements, i.e., no two adjacent elements can be skipped easily..

 Time Complexity: O(n log(S)), similar to the best case. Even though in the worst case, the binary search might take longer to converge to the optimal solution, the overall complexity remains dominated by the binary search and the iteration over the tasks.

**Average Case**

The prices array has a random distribution of values, and the dynamic programming approach needs to be applied fully.

The average case time complexity is also O(n log(S)). This assumes a typical distribution of task durations and session time constraints, where the binary search efficiently converges to the solution without unnecessary iterations.

**Overall Complexity**

**Overall Time Complexity:** O(n log(S))

**CONCLUSION**

In conclusion, the problem involves determining the maximum incurred cost in an undirected tree when rooted at various nodes. The goal is to find the root that maximizes the difference between the maximum and minimum price sum along all paths starting at that root. To achieve this, a C program was presented with key components such as node structure, path sum calculation, cost calculation, tree building, and a function to find the maximum incurred cost. The proposed UI design emphasizes a flexible and user-friendly layout, with feasible elements strategically positioned. It incorporates accessible design principles and clear functionality, providing users with an intuitive interface for inputting tree information and obtaining maximum incurred cost results. Overall, the combined programming solution and UI design aim to offer an efficient and usercentric experience in solving the problem of finding the maximum possible cost in a rooted tree. The code provides a working solution for the problem, but there is room for refinement to improve efficiency, error handling, and code readability. Depending on specific requirements and constraints, further adjustments could be made. Overall, the code serves as a solid foundation for solving the stated problem.