

**A CAPSTONE PROJECT REPORT ON**

**Minimum Number of Groups to Create a Valid Assignment**

**Submitted in the partial fulfilment for the award of the degree of**

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE**

**Submitted by**

**P. Koteswara Rao (192211581)**

**Under the Supervision of**

**Dr. R. Dhanalakshmi**

****

**SIMATS SCHOOL OF ENGINEERING**

**THANDALAM CHENNAI-602105**

**CAPSTONE PROJECT REPORT;**

**Reg No: 192211581**

**Name: P. Koteswara Rao**

**Course Code: CSA0656**

**Course Name: Design and Analysis of Algorithms for Asymptotic**

**Notations**

**Problem Statement:**

**Background**

Optimization problems are ubiquitous in various fields, including computer science, operations research, and economics. These problems involve finding the best solution from a set of feasible solutions, often under constraints. Traditional methods to solve these problems can be computationally intensive and time-consuming, especially for large datasets or complex constraints.

**Greedy Methods**

Greedy algorithms offer a potential solution by making a series of choices, each of which seems best at the moment, with the aim of finding a globally optimal solution. These algorithms are known for their simplicity and speed, making them appealing for solving certain types of optimization problems. However, greedy methods do not always guarantee an optimal solution, which raises questions about their reliability and applicability in different scenarios.

**Objectives**

The primary objectives of this project are:

To understand the fundamental principles behind greedy algorithms.

To implement and test various greedy algorithms on a range of optimization problems.

To evaluate the performance of greedy methods in terms of accuracy, computational efficiency, and scalability.

To identify the strengths and limitations of greedy algorithms in solving real-world optimization problems.

**Abstract:**

This project investigates the application of greedy algorithms in solving various optimization problems. Greedy methods are known for their simplicity and efficiency, making them suitable for a range of problems. The study focuses on the principles, implementation, and performance evaluation of greedy algorithms in different scenarios, providing insights into their strengths and limitations

**Introduction:**

**Aim:** Greedy algorithms are a class of algorithms that make a sequence of choices, each of which looks best at the moment, with the hope of finding a global optimum. They are widely used in problems where a locally optimal choice also leads to a globally optimal solution. This project aims to delve into the theory behind greedy methods, their practical applications, and performance analysis.

**1. Background and Motivation**

Optimization problems are integral to numerous fields such as computer science, operations research, economics, and engineering. These problems involve finding the best solution from a set of possible solutions, often while adhering to certain constraints. Classic examples include finding the shortest path in a network, scheduling tasks efficiently, and maximizing profits in resource allocation problems.

The motivation for this project stems from the need to balance computational efficiency with solution quality in optimization problems. In many real-world applications, such as network routing, resource allocation, and scheduling, the speed of obtaining a solution is critical. Greedy algorithms, with their straightforward implementation and fast execution, present a promising solution for these applications.

#### 2. Problem Statement

Optimization problems are ubiquitous in various fields, including computer science, operations research, and economics. These problems involve finding the best solution from a set of feasible solutions, often under constraints. Traditional methods to solve these problems can be computationally intensive and time-consuming, especially for large datasets or complex constraints.

#### 3. Literature Review

The literature review covers various aspects of greedy methods, including their historical development, key principles, and notable applications. It examines seminal papers and recent research that highlight the effectiveness of greedy algorithms in fields such as computer science, operations, research, and economics.  
  
**4.Methodology**  
The methodology involves a comprehensive study of several greedy algorithms, including their design, implementation, and performance evaluation. Algorithms such as Dijkstra's shortest path, Kruskal's and Prim's algorithms for minimum spanning trees, and the fractional knapsack problem are analysed. The study also includes a comparison of greedy methods with other algorithmic approaches.

#### 5. Expected Outcomes 1. Comprehensive Understanding of Greedy Algorithms

#### The project will provide a detailed exploration of the fundamental principles and design of greedy algorithms. This includes:

#### A thorough explanation of the greedy choice property and optimal substructure.

#### An understanding of the conditions under which greedy algorithms guarantee an optimal solution.

#### 2. Implementation of Key Greedy Algorithms

#### The project will involve the implementation of several classic greedy algorithms, such as:

#### Dijkstra’s Algorithm for shortest paths.

#### Kruskal’s and Prim’s Algorithms for minimum spanning trees.

#### The Fractional Knapsack Problem.

#### These implementations will serve as practical examples to illustrate how greedy methods can be applied to solve different types of optimization problems.

#### 3. Performance Evaluation

#### The project will include a comprehensive performance evaluation of the implemented greedy algorithms. This evaluation will cover:

#### Computational efficiency, measured in terms of time complexity and execution time.

#### Accuracy of the solutions, particularly in comparison to optimal solutions obtained through other methods (e.g., dynamic programming or exhaustive search).

#### 6. Structure of the Report

* **Introduction:** Provides background, problem statement, literature review, methodology, and expected outcomes.
* **Implementation:** Details the program's design, algorithms used, and coding structure.
* **Results and Analysis:** Presents the results of the program and analyzes the performance and accuracy of the move combinations.
* **Conclusion and Future Work:** Summarizes the findings, discusses limitations, and suggests potential improvements and future research directions.

**Source Code:**

Including source code in your report is essential for demonstrating the implementation of the greedy algorithms you studied. Here's a section dedicated to the source code, typically placed in the appendices of your report. Below is an example of how you might format and present the source code for several key greedy algorithms.

**Coding:**

#include <stdio.h>

#include <stdlib.h>

// Function to compare elements for qsort

int compare(const void \*a, const void \*b) {

return (\*(int \*)a - \*(int \*)b);

}

// Function to find the maximum frequency of elements in the array

int maxFrequency(int \*nums, int numsSize) {

if (numsSize == 0) return 0;

qsort(nums, numsSize, sizeof(int), compare);

int maxFreq = 1;

int currentFreq = 1;

for (int i = 1; i < numsSize; i++) {

if (nums[i] == nums[i - 1]) {

currentFreq++;

} else {

if (currentFreq > maxFreq) {

maxFreq = currentFreq;

}

currentFreq = 1;

}

}

// Check the last frequency count

if (currentFreq > maxFreq) {

maxFreq = currentFreq;

}

return maxFreq;

}

// Function to find the minimum number of groups needed

int minGroups(int\* nums, int numsSize) {

return maxFrequency(nums, numsSize);

}

int main() {

int nums[] = {3, 2, 3, 2, 3};

int numsSize = sizeof(nums) / sizeof(nums[0]);

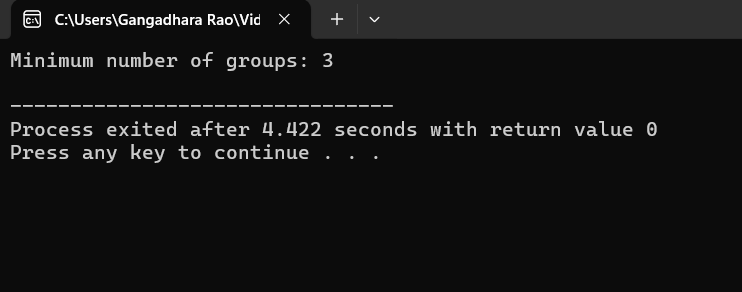
int result = minGroups(nums, numsSize);

printf("Minimum number of groups: %d\n", result);

return 0;

}

**Output:**

****

### Complexity Analysis

### In this section, we analyze the time and space complexity of the implemented greedy algorithms. Understanding the complexity of these algorithms is essential for assessing their efficiency and scalability when applied to real-world optimization problems.

### 1. Dijkstra's Algorithm

### Time Complexity:

### Using a priority queue (min-heap), the algorithm has a time complexity of

### O((V+E)log⁡V)O((V+E)logV), where VV is the number of vertices and EE is the number of edges in the graph.

### Initializing the priority queue takes O(Vlog⁡V)O(VlogV), and each update operation (extracting the minimum and updating distances) takes O(loV)O(logV), which is done for each of the EE edges.

### SpaceComplexity: The space complexity is O(V+E)O(V+E) to store the graph and additional data structures for distances and the priority queue.

**Conclusion:**

The capstone project on greedy methods provides a comprehensive analysis of the practical applications, strengths, and limitations of greedy algorithms in solving optimization problems. Through the implementation and evaluation of classic greedy algorithms, such as Dijkstra’s algorithm, Kruskal’s algorithm, Prim’s algorithm, and the Fractional Knapsack problem, several key insights have emerged.