Project Report

Title: Activity Selection Using Greedy Algorithm in C

Name: **Arshika Noor (232-115-033)**

noorarshika798@gmail.com

Samiha Jannah Najah (232-115-035)

samihajannahnajah01@gmail

Date of Submission: 13-04-2025

1. Introduction

This report presents a C program that solves the Activity Selection Problem using a Greedy Algorithm. The goal is to select the maximum number of non-overlapping activities from a given set, where each activity has a start and a finish time. The program interacts with the user, sorts the activities by finish time, and visually demonstrates the selection process through clear, tabular outputs. This helps learners understand the greedy decision-making process used in task scheduling.

2. Problem Description

The Activity Selection Problem is a classic example of a greedy optimization problem. Given a list of activities with start and finish times, the goal is to choose the largest set of mutually non-overlapping activities.

Optimal Selection: A, B, and D

Applications:

- Job or task scheduling
- CPU scheduling in operating systems
- Resource allocation
- Meeting room or classroom scheduling

3. Solution Approach (Greedy Algorithm)

The greedy strategy is based on always picking the next activity with the earliest finish time that doesn't overlap with the previously selected one.

Algorithm Steps:

1. Sort the activities by finish time (ascending).
2. Select the first activity in the sorted list.
3. For every subsequent activity:

- If its start time ≥ finish time of last selected, then select it.

Code Implementation for Table Formatting:

In the C code, the table formatting is handled by printing horizontal lines using + symbols and aligning columns using printf. The code snippet below shows how this is done:

```
// Print sorted table with lines

printf("\nSorted Activity Table:\n");

printf("+------------+");

for (int i = 0; i < n; i++) printf("--------+");

printf("\n| %-10s |", "Activity");

for (int i = 0; i < n; i++) printf(" %-6s|", activities[i].name);

printf("\n| %-10s |", "Start");

for (int i = 0; i < n; i++) printf(" %-6d|", activities[i].start);

printf("\n| %-10s |", "Finish");

for (int i = 0; i < n; i++) printf(" %-6d|", activities[i].finish);

printf("\n+------------+");

for (int i = 0; i < n; i++) printf("--------+");

printf("\n");
```

In this code:

- Header formatting: The header (Activity, Start, Finish) is printed at the top with appropriate spacing.

- Separator lines: +------------+ and similar lines are used to separate the header, rows, and the end of the table.

- Row formatting: Each row prints activity data with aligned columns using printf and %-6s for string alignment and %-6d for integer alignment.

4. Code Structure and Design

4.1 Input Handling

- User is prompted to enter the number of activities.
- Then, input for:
- Activity name
- Start time
- Finish time

4.2 Sorting by Finish Time

- A bubble sort is used to sort the activities based on finish time.
- This ensures that the activity which finishes earliest is considered first.

4.3 Activity Selection Logic

- First activity is selected by default.
- For each next activity:
- If start time ≥ last selected's finish time → select.
- Else → skip it.
- Each decision is printed with an explanation.

4.4 Output Formatting

- Activities are shown in a tabular format before and after selection.

- This improves readability and helps in understanding the logic.

4.5 Step-by-Step Explanation of the Code

Here's how the code works in a simplified way:

Step 1: Structure Definition

We define a struct to hold activity information:

```
struct Activity {
    char name[20];
    int start;
    int finish;
};
```

Step 2: Input Collection

User inputs:

- Number of activities
- Names of each activity
- Start and finish times

```
scanf("%s %d %d", activity[i].name, &activity[i].start, &activity[i].finish);
```

Step 3: Sort Activities

The activities are sorted by their finish time using Bubble Sort:

```
if (activity[i].finish > activity[j].finish) {
    swap(activity[i], activity[j]);
}
```

Step 4: Select Non-Overlapping Activities

Iterate through the sorted array and select those that don't overlap:

```
if (activity[i].start >= lastFinish) {
   selected[count++] = activity[i];
   lastFinish = activity[i].finish;
}
```

Step 5: Output

The program prints:

- Sorted activity table
- Step-by-step decision (selected or skipped)
- Final table of selected activities

5. Time Complexity

- **Sorting:** O(n²) (due to bubble sort, can be optimized)
- **Selection:** O(n) (single pass after sorting)
- **Overall Time Complexity:** O(n²)
- **Space Complexity:** O(n) for storing activity data and selected activities.

6. Additional Considerations

- Visualization: Intermediate steps and final output are shown in a readable table format.
- User Interaction: Clear prompts and messages make the program interactive.
- Possible Enhancements:
- Replace Bubble Sort with a faster sorting method
- Add graphical visualization like Gantt charts
- Allow input/output from a file

- Add input validation for better user experience

7. Conclusion

This C program effectively demonstrates the Greedy Algorithm for solving the Activity Selection Problem. By sorting activities based on their finish times and selecting those that don’t overlap, the program provides an optimal and efficient solution. It also displays a clear, step-by-step explanation of the greedy selection logic. This makes it a valuable educational tool for understanding greedy strategies and their application in real-world scheduling problems.