SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
COLLEGE OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF NETWORKING AND COMMUNICATIONS
DAA, Mini-Project Presentation

ACTIVITY SELECTION PROBLEM

(GREEDY ALGORITHM)

DEGALA LIKHIT AAKASH
RA2211028010237

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Table of Contents

- Introduction
- Problem Statement
- Explanation
- Applications
- Basic Code
- Algorithm
- Pseudo Code
- Implementation
- Conclusion

Introduction

- The Activity Selection Problem is a fundamental challenge in computer science, involving the selection of non-overlapping activities to maximize productivity within a specified time frame.
- Its applications span various domains, including project management, event scheduling, and processor allocation.
- This project aims to explore the problem's principles, solutions, and real-world implications, shedding light on its significance in optimizing resource allocation and task scheduling.

Problem Statement

- **Maximizing Non-Overlapping Activities:**

Given a set of activities with start times and finish times, the objective is to select a maximum number of activities that do not overlap, ensuring each selected activity is completed within its designated time frame.

- **Optimizing Resource Utilization:**

The goal of the Activity Selection Problem is to maximize the utilization of available resources within a given time frame, while preventing scheduling conflicts by selecting activities that do not overlap. This involves strategic selection to efficiently allocate resources and minimize idle time.

Applications of ASF

1. **Meeting Scheduling:** Optimizing meeting times to maximize attendance.
2. **Project Management:** Efficiently allocating resources for project tasks.
3. **Event Planning:** Organizing events to minimize conflicts and ensure smooth execution.
4. **Conference Scheduling:** Maximizing participation and minimizing overlap in conference sessions.
5. **Processor Scheduling:** Allocating CPU time to tasks for efficient processing.
6. **Resource Allocation:** Optimizing resource usage across tasks or projects.
7. **Class Scheduling:** Scheduling classes to accommodate student preferences.
8. **Job Scheduling:** Efficiently assigning tasks to optimize production.
9. **Flight Scheduling:** Optimizing flight routes and crew assignments.
10. **Sporting Events:** Scheduling tournaments and matches to avoid conflicts.

ASP for Maximizing Non-Overlapping Activities

- The Activity Selection Problem entails selecting a subset of activities from a given set, ensuring that no two selected activities overlap in time.
- This optimization problem is fundamental in scheduling tasks, meetings, or events to maximize productivity and minimize conflicts.
- Typically, a greedy algorithm is employed to select activities based on their finish times, resulting in an optimal schedule.
- This approach offers efficient resource utilization and is widely applicable across various domains, including project management, event planning, and processor scheduling.

ASP for Optimizing Resource Utilization

- Optimizing resource utilization is paramount for productivity across various domains. It involves selecting non-overlapping activities efficiently to ensure optimal resource allocation, preventing waste and maximizing output.
- By employing a greedy algorithm based on finish times, activities can be selected effectively, ensuring an optimal balance between productivity and resource utilization.
- This approach has widespread applications in project management, event scheduling, and resource allocation, facilitating streamlined operations and improved efficiency.

Algorithm for ASP

- **Sort activities by finish time:** Arrange the given activities in ascending order based on their finish times.
- **Initialize an empty set of selected activities:** Create an empty set to store the selected activities.
- **Select the first activity:** Add the activity with the earliest finish time to the set of selected activities.
- **Iterate through the remaining activities:**
  - For each activity, check if its start time is after or equal to the finish time of the last selected activity.
  - If the condition is met, add the current activity to the set of selected activities.
- **Return the set of selected activities:** Once all activities have been considered, return the set of selected activities as the optimal solution.

Pseudocode - ASP

```cpp
#include <vector>
#include <algorithm>

// Structure to represent an activity
struct Activity {
    int start;
    int finish;
};

// Function to compare activities based on their finish times
bool compareActivities(const Activity& a, const Activity& b) {
    return a.finish < b.finish;
}

// Function to find the maximum number of non-overlapping activities
std::vector<Activity> activitySelection(const std::vector<Activity>& activities) {
    std::vector<Activity> selectedActivities;
    int lastFinish = std::numeric_limits<int>::min();

    // Sort activities based on finish times
    std::vector<Activity> sortedActivities = activities;
    std::sort(sortedActivities.begin(), sortedActivities.end(), compareActivities);

    // Iterate through sorted activities
    for (const auto& activity : sortedActivities) {
        if (activity.start >= lastFinish) {
            // Select non-overlapping activity
            selectedActivities.push_back(activity);
            lastFinish = activity.finish;
        }
    }

    return selectedActivities;
}

```

Implementation - ASP

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

// Structure to represent an activity
struct Activity {
    int start;
    int finish;
};

// Function to compare activities based on their finish times
bool compareActivities(const Activity& a, const Activity& b) {
    return a.finish < b.finish;
}

// Function to find the maximum number of non-overlapping activities
std::vector<Activity> activitySelection(const std::vector<Activity>& activities) {
    std::vector<Activity> selectedActivities;
    int lastFinish = std::numeric_limits<int>::min();

    // Sort activities based on finish times
    std::vector<Activity> sortedActivities = activities;
    std::sort(sortedActivities.begin(), sortedActivities.end(), compareActivities);

    // Iterate through sorted activities
    for (const auto& activity : sortedActivities) {
        if (activity.start >= lastFinish) {
            // Select non-overlapping activity
            selectedActivities.push_back(activity);
            lastFinish = activity.finish;
        }
    }

    return selectedActivities;
}

```

Implementation - ASP

```
36
37  int main() {
38      // Example usage
39      std::vector<Activity> activities = {{1, 4}, {3, 5}, {0, 6}, {5, 7}, {3, 8}, {5, 9}};
40      std::vector<Activity> selectedActivities = activitySelection(activities);
41
42      // Output selected activities
43      std::cout << "Selected Activities:\n";
44      for (const auto& activity : selectedActivities) {
45          std::cout << "Start: " << activity.start << ", Finish: " << activity.finish << std::endl;
46      }
47
48      return 0;
49  }
50
```

Problem for ASP

```
Activities:
1. Start: 1, Finish: 4
2. Start: 3, Finish: 5
3. Start: 0, Finish: 6
4. Start: 5, Finish: 7
5. Start: 3, Finish: 8
6. Start: 5, Finish: 9
```

- We have a list of activities with their start and finish times.
- The ‘activitySelection’ function is called, which finds the maximum number of non-overlapping activities.
- The function sorts the activities based on their finish times.
- It iterates through the sorted activities and selects non-overlapping activities.
- The selected activities are then printed.

```
Selected Activities:
Start: 1, Finish: 4
Start: 5, Finish: 7
Start: 3, Finish: 8
```

Time Complexity for ASP

- The time complexity analysis of the Activity Selection Problem solution employing the greedy algorithm reflects its efficiency and practicality. Primarily, the dominant factor contributing to the time complexity is the sorting operation performed on the activities based on their finish times. This sorting operation typically exhibits a time complexity of O(n log n), where 'n' denotes the number of activities.
- Following the sorting step, the algorithm iterates through the sorted list once, a process with a time complexity of O(n). Within each iteration, the algorithm conducts constant-time operations to ascertain whether the current activity overlaps with the previously selected one.
- Consequently, the overall time complexity of the greedy algorithm for the Activity Selection Problem is determined to be O(n log n), underscoring its aptitude for efficiently handling substantial activity sets. This succinct analysis underscores the algorithm's efficiency and suitability for real-world applications.

Conclusion for ASP

- The Activity Selection Problem provides a practical framework for efficiently selecting a subset of non-overlapping activities from a given set. Through the application of a greedy algorithm, which selects activities based on their finish times, this problem offers an elegant solution for maximizing productivity and resource utilization while minimizing conflicts.
- In conclusion, the Activity Selection Problem holds significant importance in various domains, including project management, event scheduling, and resource allocation. By effectively managing activities and selecting those that optimize resource utilization, organizations can streamline operations, enhance efficiency, and achieve their objectives in a time-sensitive manner. Moreover, the simplicity and effectiveness of the greedy algorithm make it a valuable tool for solving scheduling and resource allocation challenges in real-world scenarios.

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be **University** *u/s 3 of UGC Act, 1956*

Thank You