Jho Raven Abalos
CPE21S1

Problem: Activity Selection

Statement of the problem: You are given a list of activities, each with a start and end time. The goal is to choose the fewest number of non-overlapping activities possible.

Discussion: The greedy approach to the Activity Selection Problem entails choosing activities based on their completion times. The algorithm sorts the activities in ascending order of finish time, then iteratively chooses the activity with the earliest finish time that does not overlap with the previously selected activities.

Input Solution:

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

struct Activity {
    int start_time;
    int end_time;

    Activity(int start, int end) : start_time(start), end_time(end) {}
};

bool compareActivities(const Activity& a, const Activity& b) {
    return a.end_time < b.end_time;
}

std::vector<Activity> activitySelection(const std::vector<Activity>& activities) {
    std::vector<Activity> selectedActivities;

    if (activities.empty()) {
        return selectedActivities;
    }

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

    // Greedy approach to select non-overlapping activities
    selectedActivities.push_back(sortedActivities[0]);
    int lastFinishTime = sortedActivities[0].end_time;

    for (size_t i = 1; i < sortedActivities.size(); ++i) {
        if (sortedActivities[i].start_time >= lastFinishTime) {
            selectedActivities.push_back(sortedActivities[i]);
            lastFinishTime = sortedActivities[i].end_time;
        }
    }

    return selectedActivities;
}
```

```
int main() {
    std::vector<Activity> activities = {
        {1, 4},
        {3, 5},
        {0, 6},
        {5, 7},
        {8, 9},
        {5, 9}
    };

    std::vector<Activity> result = activitySelection(activities);

    std::cout << "Selected Activities:" << std::endl;
    for (const Activity& activity : result) {
        std::cout << "Activity: Start Time = " << activity.start_time << ", End Time = " << activity.end_time << std::endl;
    }

    return 0;
}
```

Output:

```cpp
///JHO RAVEN ABALOS -CPE21S1

#include <iostream>
#include <vector>
#include <algorithm>

struct Activity {
    int start_time;
    int end_time;

    Activity(int start, int end) : start_time(start), end_time(end) {}
};

bool compareActivities(const Activity& a, const Activity& b) {
    return a.end_time < b.end_time;
}

std::vector<Activity> activitySelection(const std::vector<Activity>& activities) {
    std::vector<Activity> selectedActivities;

    if (activities.empty()) {
        return selectedActivities;
    }

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

    // Greedy approach to select non-overlapping activities
    selectedActivities.push_back(sortedActivities[0]);
    int lastFinishTime = sortedActivities[0].end_time;

    for (size_t i = 1; i < sortedActivities.size(); ++i) {
        if (sortedActivities[i].start_time >= lastFinishTime) {
            selectedActivities.push_back(sortedActivities[i]);
            lastFinishTime = sortedActivities[i].end_time;
        }
    }

    return selectedActivities;
}

int main() {
    std::vector<Activity> activities = {
        {1, 4},
        {3, 5},
        {0, 6},
        {5, 7},
        {8, 9},
        {5, 9}
    };

    std::vector<Activity> result = activitySelection(activities);

    std::cout << "Selected Activities:" << std::endl;
    for (const Activity& activity : result) {
        std::cout << "Activity: Start Time = " << activity.start_time << ", End Time = " << activity.end_time << std::endl;
    }

    return 0;
}
```

```
Selected Activities:
Activity: Start Time = 1, End Time = 4
Activity: Start Time = 5, End Time = 7
Activity: Start Time = 8, End Time = 9


...Program finished with exit code 0
Press ENTER to exit console.
```

It defines a struct Activity to represent activities and uses the greedy approach to find the greatest number of non-overlapping activities possible. The compareActivities function is used to sort activities based on their completion times. The main function uses a set of example activities to demonstrate how to use the algorithm.