Algorithms and Data Structures: Homework #10

Due on April 20, 2020 at 23:00

Henri Sota

Problem 10.2

b) Assuming an unsorted sequence of activities, derive a greedy algorithm for the activity-selection problem that selects the activity with the latest starting time. Your solution should not simply sort the activities and then select the activity.

My algorithm of finding the optimal solution to the activity selection maximization problem is implemented based on the iterative approach of the greedy activity selector algorithm given in the Introduction to Algorithms book. Different from the approach taken there, my algorithm doesn't need to preprocess the array of activities by sorting them. My algorithm sets up a loop which will iterate until the given array of activities is empty. It will iteratively choose the latest starting time activity of the current activities in the array, remove it from the array and it will decide to add the activity onto the optimal solution set array depending if the activity is in conflict with the previously added activity of the optimal solution set array. The time complexity of my greedy algorithm is $O(n^2)$, because it requires 2 nested loops inside of each other to create the optimal solution set array, performing worse than the greedy approach given in the book and slides which is $O(n \log(n))$ due to preprocessing the array (time complexity of $O(n)$ omitting sorting).

```
#ifndef ACTIVITY_H_
#define ACTIVITY_H_
#include <vector>

class Activity {
    private:
        int start;
        int end;
    public:
        // Parametrized constructor
        Activity(int start, int end);

        // Friend functions
        // Function to find the latest starting time activity in the list
        friend int findLatest(const std::vector<Activity> &list);
        // Function to output a Activity object properties on the output stream
        friend std::ostream& operator<<(std::ostream &out, const Activity &a);
        friend std::vector<Activity> greedyActivitySelector(
            std::vector<Activity> list
        );
};

#endif /* ACTIVITY_H_ */
```

Listing 1: Activity Class Implementation Declaration in C++

```
Activity::Activity(int start, int end) {
    // Construct activity object with start and end parameter provided
    this->start = start;
    this->end = end;
}

int findLatest(const std::vector<Activity> &list) {
    int position = 0;
    Activity latestStartingActivity = list[0];
    // Search through the array for the interval with the latest starting time
    for (int i = 0; i < (int) list.size(); ++i)
        // Check if the starting time of activity is later than the current
        // we know of
        if (list[i].start > latestStartingActivity.start) {
            latestStartingActivity = list[i];
            position = i;
        }
    // Return the position of the latest starting activity of the list
    return position;
}

std::ostream& operator<<(std::ostream &out, const Activity &a) {
    // Output the properties of a Activity object in format
    // (start,end)
    out << "(" << a.start << "," << a.end << ")";
    return out;
}

std::vector<Activity> greedyActivitySelector(std::vector<Activity> list) {
    std::vector<Activity> optimalSelection;
    std::vector<Activity> activitiesLeft = list;

    // Find the first latest activity of the given list
    int latestActivityPosition = findLatest(list);
    // Remove the latest activity of the list from the activitiesLeft vector
    activitiesLeft.erase(activitiesLeft.begin() + latestActivityPosition);
    // Push the latest activity of the list into the optimal solution vector
    optimalSelection.push_back(list[latestActivityPosition]);

    // Search the activitiesLeft vector until it has no elements left
    while ((int) activitiesLeft.size() != 0) {
        // Find the latest activity of the activitiesLeft vector
        latestActivityPosition = findLatest(activitiesLeft);
        // Save the activity by copying it from the activitiesLeft vector
        Activity latest = activitiesLeft[latestActivityPosition];
        // Remove the last activity from the activitiesLeft vector
        activitiesLeft.erase(activitiesLeft.begin() + latestActivityPosition);

        // Check if the activity that has just been chosen as the latest is
        // compatible with the last activity added on the optimal solution
        if (latest.end <= optimalSelection[optimalSelection.size() - 1].start)
            // Add the activity to the optimal solution if it is non-conflicting
            // with the previously added one
            optimalSelection.push_back(latest);
    }

    return optimalSelection;
}
```

Listing 2: Activity Class Implementation Definition in C++

```cpp
int main() {
    std::vector<Activity> list = {
        Activity(1, 4),
        Activity(3, 5),
        Activity(0, 6),
        Activity(5, 7),
        Activity(3, 9),
        Activity(5, 9),
        Activity(6, 10),
        Activity(8, 11),
        Activity(8, 12),
        Activity(2, 14),
        Activity(12, 16)
    };

    std::vector<Activity> solution = greedyActivitySelector(list);

    std::cout << "Optimal solution set of activities:" << std::endl;
    for (Activity activity: solution)
        std::cout << activity << std::endl;

    return 0;
}
```

Listing 3: Greedy Activity Selector with Latest Starting Time Test in C++