Algorithms Homework 7

Liam Dillingham

November 12, 2018

1 Question 16.1-5

Consider a modification to the activity-selection problem in which each activity a_i has, in addition to a start and finish time, a value v_i. The objective is no longer to maximize the number of activities scheduled, but instead to maximize the total value of the activities scheduled. That is, we wish to choose a set A of compatible activities such that $\sum_{a_k \in A} v_k$ is maximized. Give a polynomial-time algorithm for this problem.

My algorithm is a modified version of the book's iterative GREEDY-ACTIVITY-SELECTOR(s, f). Some things I used that may be different is Java's TreeSet and HashMap classes. I built an Activity class to handle all the attributes associated with each activity, and used the TreeSet to contain the output, that is, all the activities that were compatible via their start and end times.

Next, I used a HashMap and keyed the Activities on their key. This makes it easier to fetch activities during the activity selection.

Then, for the actual greedy selection algorithm, I iterated through the activity ids, starting with 1, and going to n. I pushed that id into our result set, and tried to find all compatible activities that came afterward. The result of each iteration of this is a set of compatible activities. From there, I calculate the total value of these activities, and compare to find the max:

```
greedyActivityValueSelector(i)
        n = HashMap.size() // Number of elements in the hashmap
        Set s = new Set
        s.add(HashMap(i))

        a = HashMap(i)
        for j = i to n
                b = HashMap(j) // Get consecutive hash keys
                if b.start >= a.finish
                        s.add(b)
                        a = b
        return s

max = 0
maxSet = null
for activity i to n:
        Set s = greedyActivityValueSelector(i)

        sum = 0
        for each j in s
                sum = sum + j.value

        if sum > max
                max = sum
                maxSet = s

print(maxSet)
```

I used a HashMap in associating the activity ids with an activity. HashMaps can compute in $\Theta(1)$. Also, TreeSets search and store in $\lg(n)$. Note that initially in my "pseudocode" there is a call to greedyActivitySelector(i). If $i = 1$, then the loop will run from 1 to n, giving us initially a worst-case runtime $\mathcal{O}(n \lg n)$. Also, within the function call, we will have to run from $j = i$ to n, which will increase our runtime to $\mathcal{O}(n^2 \lg n)$. However, suppose that every single activity is inserted in the set. Then the size of the set returned from the function is n. Now that we have the function returned, we need to iterate over it and find the maximum sum. If the size of the set is n, then we must iterate over all of the n elements to find the max sum, which gives us a final worst-case runtime of $\mathcal{O}(n^3 \lg n)$.