Name: Suhail Basalama | Howework: #7 Activity Scheduler | Date: 04/04/2019

Report

Data Structures

1- Activity object containing (index, start, finish, and value as ints)
2- Comparators: (Activity Comparator, SolutionComparator, and SortByFinishTime comparator)
3- Sets and TreeSets to hold activities and sets of activities unique and sorted.
4- Solution object which holds "A": a set of compatible activities, and "v" the max value.
5- Iterators to iterate over the sets and get their contents.
6- Primitive types

Algorithms

First, I created the Activity class to hold an object of an activity with its index, start time, finish time, and value. Then, I tried to solve the problem using greedy algorithms, but that did not work since we have a different criterion, namely max value.

Before implementing the algorithm, I had to make some comparators to keep my data sorted and unique in a certain way. I implemented an ActivityComparator to avoid adding the same activity to a set, and to keep activities sorted by their indices. I implemented a SolutionComparator, which keep a set of sets(solutions) of activities unique and sorted by their starting time. I also implemented a comparator called SortByFinishTime to sort activities of a set by their finish times.

Next, after studying the problem and solving it by hand, I came up with an algorithm that produces the optimal solutions available for any set of activities, given they are sorted increasingly by their finish time. The algorithm starts by creating a set for sets of optimal solutions of compatible activities, called optimalSolutions, using the comparator just mentioned above. Then, I initialized an integer variable "maxValue" to zero to hold the maximum incurred value of compatible activities so far.

After that, I made a loop with "l" variable, where "l" represents a difference between the indices "i" and "j" of a subset of the target set of activities. I made another loop with a variable named "i" which represent the beginning of subset of the target set of activities, and it has another variable named "j" which represents the end of a subset of the target set of activities.

For example, let S_{15} be a set of activities $\{a_1, a_2, a_3, a_4, a_5\}$, then if "l" is 0, the difference between "i" and "j" is 0, so with "l" = 0 the possible subsets of S_{15} are S_{11} … S_{55}, where $S_{11} = \{a_1\}$. Given the "l" (the difference) is increasing between "i" and "j", we will cover all the possible subsets of S_{15} from S_{11}, S_{22} … up to S_{14}, S_{25}, S_{15}. With each of those subsets, I have a function called GetSolution(index,s,f,v, i,j) that operates based on the GREEDY-ACTIVITY-SELECTOR(s,f) algorithm. It finds the maximum number of compatible activities of a set (S_{ij}) and sum up their respective values and returns a solution object. A solution object simply holds a set of activities "A" and a value "v" to represent the value of that solution. Once I get the solution object for every possible subset S_{ij}, I compared the value of that solution to the maxValue variable: if the value of that solution is higher than the current maxValue, I cleared my set of old solutions because they are not optimal, and added the new solution to the set of optimal Solutions and update my maxValue, otherwise, if maxValue is still the same, I add my unique solution to the set of optimal solutions.

My GetSolution method has a run time of $\Theta(n)$, where n is the size of subset passed to it. My two loops for an n size input runs k times according to the following formula:

$$k = (n + n - 1 + n - 2 + \cdots + 1) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$$

Therefore, the run time of my algorithm is $\Theta(n^3)$, which is polynomial.

Results

I had many problems with coming up with a correct solution at first, but then with careful debugging and analysis, my algorithm produced the optimal solutions in a polynomial run time. The output matches the one I calculated by hand, and it matches the format required by the professor. I also added a verbose variable: if set to 0 it outputs the normal output, if set to 1, it outputs all optimal solutions in a set format. The names of files are output1.txt and output2.txt.

Standard output:

First input:

```
3
1 4
IT HAS MULTIPLE SOLUTIONS
```

Second input:

```
1095
1 2
IT HAS A UNIQUE SOLUTION
```

Verbose output:

First input:

```
3
{a1,a4}{a2,a4}{a5}
IT HAS MULTIPLE SOLUTIONS
```

Second input:

```
1095
{a1,a2}
IT HAS A UNIQUE SOLUTION
```