Gretel Rajamoney

February 6, 2021

Homework #4

Problem 1:

a. Verbal Description:

The greedy algorithm starts by first sorting the array containing all of the job penalties (p_i), the penalties are sorted in ascending order so that the jobs with larger penalties are at the end of the array, meanwhile the jobs with smaller penalties are at the beginning of the array. The algorithm will also sort the array containing all of the job deadlines (d_i), the deadlines are sorted in ascending order so that the jobs with longer deadlines are at the end of the array, meanwhile the jobs with shorter deadlines are at the beginning of the array. Next, out of every job listed, the algorithm will select the job containing the shortest deadline as well as the smallest penalty. The algorithm will continue running until all of the jobs are completed with the minimal possible penalty.

Pseudocode:

```
//create necessary arrays
int jobArray[];
int penaltyArray[];
int deadlineArray[];

//sort the job penalties in ascending order
for (int i = 0; i < number of jobs; i++)
{
        if (penaltyArray[i] < penaltyArray[i + 1])
        {
                continue;
        }

        else
        {
                //switch indexes
                int tempHolder;
                tempHolder = penaltyArray[i];
                penaltyArray[i] = penaltyArray[i + 1];
                penaltyArray[i + 1] = tempHolder;
        }
}

//sort the job deadlines in ascending order
for (int i = 0; i < number of jobs; i++)
{
        if (deadlineArray[i] < deadlineArray[i + 1])
        {
                continue;
        }
        else
        {
                //switch indexes
                int tempHolder;
                tempHolder = deadlineArray[i];
                deadlineArray[i] = deadlineArray[i + 1];
                deadlineArray[i + 1] = tempHolder;
        }
}

//selects which jobs to be completed so that
//the minimal possible penalty is achieved
select(penaltyArray[], deadlineArray[])
{
        for (int i = 0; i < number of jobs, i++)
        {
                select job containing smallest penalty
                select job containing shortest deadline
                return the job to the user
        }
}
```

b. Running Time:
Since the algorithm sorts the array based upon penalties because the sum of all penalties must be minimalized, the algorithm must also take into consideration the deadlines of all the jobs that need to be completed resulting in a runtime of $O(n\,log\,n)$ for each iteration. When assigning the job, the runtime of the algorithm with be $O(n)$, therefore the time complexity of the algorithm will be $T(n) = O(n\,log\,n)$.

Problem 2:

By sorting the jobs into the last activity to start instead of selecting the first activity to finish, we can still implement the same greedy algorithm logic. In order to solve this problem using the greedy algorithm, we simply must sort the array containing all of the start times in descending order as oppose to ascending order, so that the larger start times are at the beginning of the array and the smaller start times are at the end of the array. Next, we must also sort the array containing all of the end times so that their indexes are matched up with the start times that were just sorted in descending order. The greedy algorithm itself remains unchanged and will return the jobs as per usual, but this time the activities will be selected in reverse order. Therefore, it can be concluded that it will always yield an optimal solution because the algorithm is considered to be reciprocally compatible and operates reversed as well since all of the activities are disjoint. The time complexity of this approach is the same as the initial approach because the only thing that is changing is the order, but the sorting process remains the same. We can confirm that the greedy approach sorted by last activity to start will yield the most optimal solution similarly to the first activity to finish methodology, this is because the reversed solution contains the same set of activities that are disjoint, therefore it will provide the most optimal solution regardless of how the start and finish times are ordered

Problem 3:

Verbal Description:

In order to yield the most optimal solution while implementing the greedy algorithm using the last-to-start methodology as oppose to the first-to-finish methodology, we must modify the way we order the data. We start by locating the index of the last activity to start, we print the activity number located at this index. Next, we sort the array contain all of the start times into ascending order, the array is sorted and the indexes within the finishing times array are modified to match up with the finishing times array. In order to select which activities will be completed, we parse the array and select the activity if its start time is greater than or equal to its finishing time. The selection function is repeatedly called until all of the data and sets within the data file have been completed.

Pseudocode:

```
// sorts the elements in the array in order
sortFuntion(startTimes[], endTimes[], position[], totalActivities)
{
       use insertion sort syntax from hw1
       sort the startTimes[] in descending order
       save the indexes correctly in endTime[]
       save the indexes correctly in position[]
       returns the sorted function
}

// prints out the activities that are selected
printActivities(startTimes[], endTimes[], totalActivities)
{
       // parse through entire length of activities array
       for (0 to totalActivities)
       {
              // prints out the most optimimal activity selection
              if (startTimes[] >= endTimes[]) {
                     print index of activity
              }
       }

       return 0
}

int main()
{
       read in act.txt file

              // collects all the data in the act.txt and stores it correctly
```

```
        while (there are still sets in the act.txt file)
        {
            for (0 to totalActivities)
            {
                int totalActivities = total number of activities in the set
                    startTime[] = start times of every activity in the set
                    endTime[] = end times of every activity in the set
            }
            print out necessary prompts
            // function calls after the data has been stored
            call sortFuntion(startTime[], endTime[], totalActivities)
            call printActivities(startTimes[], endTimes[], totalActivities)
        }

    close act.txt file
    return 0
}
```

Theoretical Running Time:

The time complexity of the algorithm will be $T(n) = O(n\, log\, n)$ because the algorithm must first sort the data in order prior to determining which activities will be completed.