**(S2-20\_DSECLZG519) (Data structures and Algorithms Design)**

**Academic Year 2020-2021**

Software Design Documentation

**Assignment 2 PS5 – Assignment - Group 21**

**Overview:**

## This problem is about a maximizing the profit by correctly scheduling the problem using greedy method

**Given Problem Statement:**

A teacher has given a set of questions to the class. Each question takes a day to finish the task. Each problem has a deadline, if finished before the deadline the class gets extra bonus marks. No Bonus,If the problem is not finished before the deadline

For example, the class has been given 3 problems: [A,B,C]. The corresponding deadlines are [1,2,1] and the bonus for finishing the problem before time is [10,20,30]. For maximizing the bonus marks you can get problem C and B. While missing the deadline of problem A, which gives you a maximum 50 bonus marks earned. The order of the tasksis C-B-A

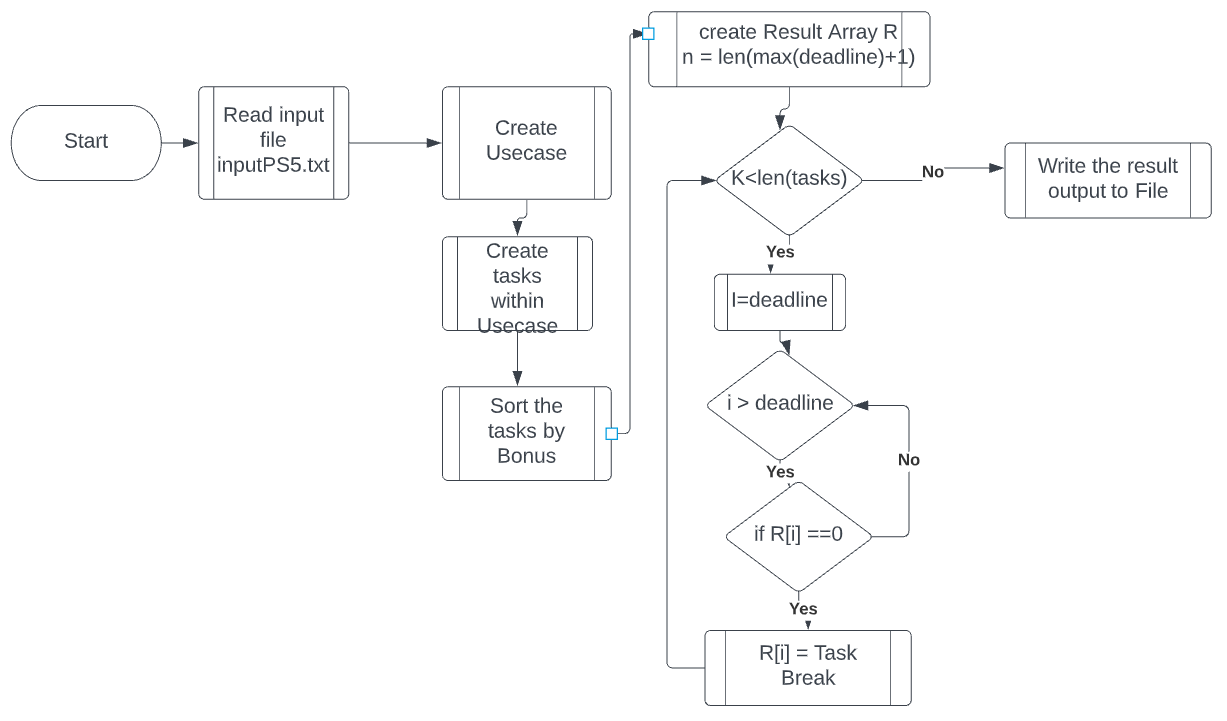


**Goal:**

The goal is to create a python program to read the list of use cases (problems) with various tasks and each task consist deadline and bonus, apply the greedy algorithm to get the maximum profit and then write the profit earned and order in which the tasks are taken to an output file

**Flow Diagram:**

The following diagram shows the overall flow



**Design + Data Structures:**

This program can be solved in various ways. Our primary focus was to reduce the complexity of the program and that is n\*log(n)

Steps for one usecase:

* First order the tasks based on the bonus descending. This takes n\*log(n). This is implemented using Merge sort
* Then allocate the tasks based on the deadline and availbility.
  + Define the assignedTasks array to max of the deadline
  + Loop through all the tasks
    - For each task, Identify the position of the task i.,e where the current task is to be assigned in the assignedTask array.
    - Once position is fount, break the loop and take another task until availability of tasks in assigned tasks

This algorithm uses list and two custom class one to hold attributes of a task and one to hold usecases and encapsulate its functions

* Task: The custom class consists of three attributes
  + Name of the task, generated at runtime and value can A, B, C… Z, A1,B1,C1….Z1, A2,B2…. Z2…
  + Task deadline: positive integer
  + Task bonus : positive integer
* Usecase: Custom class consists of the following attributes and members
  + Attribute: Use case name : Generated at runtime based on the number of use case passed in the file
  + Attribute:
    - Tasks : List of tasks under a usecase
    - Assigned Tasks: Final results after applying the greedy algorithm
  + Method:
    - maximizeBonus: greedy algorithm implemented as part of this method to maximize the bonus
* List: This list holds list of use cases
* There are static methods to implement the merge sort , getAlphabet, readFile and writeFile

This is the simplest data structure meeting all our requirements.

Note: The program didn’t use any python librabries other than lists and custom classes

**Run Time Analysis:**

The max complexity to solve one usecase is **n\*log(n)**

* Step-1 is sort the tasks in ascending order w.r.t deadline. This is implemented using merge sort and it takes O(n\*(log n)) complexity
* Step-2: Loop through max deadline times and this complexity is n
* Overall complexity is = n+n\*log(n)
* In Big O terms, O(n\*logn + n) ~= **O(n\*logn)**

**Alternate Modelling:**

**As an alternate approach, we can use the Disjoint set data structure to solve this problem. Time Complexity using Disjoint Set is going to be O (n\* log n), where n is the size of the jobs array.**

**Please follow the following steps:**

* **All time slots are individual sets and find the maximum deadline for all the tasks.**
* **If Max deadline is d, then create d+1 individual sets, and if the task has a time slot of t where t >=0 then the task is scheduled between t-1 & t slots. So, a set with the value m represents the time slot [m-1, m].**
* **We have to maintain the greatest time slot available and this slot can be allotted to the given task having a deadline. We use the parent array of Disjoint Set Data structures for this purpose. The root of the tree is always the latest available slot and if for any deadline (d), no slot is available then root would be 0.**

**Step 1: Initialize Disjoint Set and Creates initial disjoint sets.**

**Step 2: Find the latest time slot available.**

**Step 3: We assign a time slot ‘t’ to a job and then do the union of ‘t’ & ‘t-1’ in a way that ‘t-1’ becomes the parent of ‘t’. This means that all future queries for time slot t would return the latest time slot available for the set represented by t-1.**

**Algorithm:**

|  |
| --- |
| **Input:** File inputPS5.txt contains list of commands / operations.  E.g.  No of use-cases: 7 Deadlines: 1 2 3 1 4 Bonus: 20 40 10 10 20 Deadlines: 4 1 2 2 Bonus: 20 30 10 40 Deadlines: 4 1 1 1 Bonus: 20 10 40 30 Deadlines: 2 1 2 1 3 Bonus: 100 19 27 25 15 Deadlines: 2 1 1 2 Bonus: 6 8 5 10 Deadlines: 5 3 3 2 4 2 Bonus: 200 180 190 300 120 100 Deadlines: 7 2 5 3 4 5 2 7 3 Bonus: 15 20 30 18 18 10 23 16 25  **Output:** File OutputPS5.txt lists the maximum bonus earned and the sequence of Jobs for every test case.  E.g.  90  90  60  142  18  990  147  Total number of test cases are 7   For the use case Usecase:1, the maximum bonus earned is 90 For the use case Usecase:2, the maximum bonus earned is 90 For the use case Usecase:3, the maximum bonus earned is 60 For the use case Usecase:4, the maximum bonus earned is 142 For the use case Usecase:5, the maximum bonus earned is 18 For the use case Usecase:6, the maximum bonus earned is 990 For the use case Usecase:7, the maximum bonus earned is 147 For the use case Usecase:1, the tasks were scheduled in A-->B-->C-->E For the use case Usecase:2, the tasks were scheduled in B-->D-->A For the use case Usecase:3, the tasks were scheduled in C-->A For the use case Usecase:4, the tasks were scheduled in C-->A-->E For the use case Usecase:5, the tasks were scheduled in B-->D For the use case Usecase:6, the tasks were scheduled in B-->D-->C-->E-->A For the use case Usecase:7, the tasks were scheduled in B-->G-->I-->E-->C-->A-->H  **Function:** The following algorithm gives a high-level overview of the overall program   * **Main**   + Reads the file contents and caches them into uses cases custom datatype   + For each usecase, it performs maximimzation and stores the results   + Finally prints the the results to output file |