Algorithm and Data Structure

Assignment 2 report

Liu Likun | 2IE19337P

Data of Submission: June 12, 2019

Course: Algorithm and Data Structure

Contents

[Overview 2](#_Toc11234997)

[Principles 2](#_Toc11234998)

[Code details, correctness and execution time 3](#_Toc11234999)

[Execution 4](#_Toc11235000)

[Extended test 4](#_Toc11235001)

# Overview

This program uses two loops to find out the best scheduling solution for a series of tasks with weight parameters.

# Principles

We’ve represented the given tasks as a 2-d Numpy array, in which each row consists of three elements: start time, end time and weight.

We’ve firstly sorted all the tasks by start time in ascending order and initiate 2 arrays with the same length as the tasks: weight array and component array. The weight array is initiated with the weight of corresponding tasks’ weight and component array is initiated with zeros.

For loop variable “i” from the second task, we’ve loop through all the tasks before “i” th task. If the sum of current task[i]’s weight and current weight array[j] is greater than the weight of weight array[i], then update weight array[i] with the sum. Also, the position of j is also logged into component array to help find the solution later. (For programming purpose, we’ve deliberately added 1 to the index so that we could identify the first component of the solution by reaching an index value of 0).

When reaches the end of the task list, we take the weight array and the maximum value in the array is the maximized sum of weight we’re looking for.

The next step we take the index of the maximum value and recursively find the components that form the best solution and print it out.

A screenshot of a cell phone

Description automatically generated

# Code details, correctness and execution time

As we recursively loop through all the possibilities, the algorithm should be the correct one.



The code above is doing some initiation work which involves sorting and initiate some variables and weight sequence



The code above loop through the task array and find the best possible solution. The time complexity of above loop is approximately:

The above equation is under the consumption that we don’t know the condition of the task so that for each “i”, “j” could loop though the entire array in worst case scenario, hence the .



The code above prints out the maximum sum and find the components that form the best solution.

# Execution

The execution result of the program is as follow:

A close up of a screen

Description automatically generated

# Extended test

As always, we’ve implemented a large random data set to test this method, in this case, we’ve designed a job generator with input parameter: time length, maximum weight and the size of the job. In practice, we’ve set the time length to 2 times of loop variable and both maximum weight and job size to just i. The result is shown below:

A close up of a map

Description automatically generated

From the figure above, we could see a clear quadratic growth in time and therefore the .