Data structures and Algorithm Designs

Venkataramanan Krishnan
Bala Vijay D
Bala Kavin Pon

05-Sep-2019

AS2_CH1_B1_G6

02ND ASSIGNMENT

DATA STRUCTURES AND ALGORITHM DESIGNS

Created in collaboration of

VENKATARAMANAN KRISHNAN

2018AC04529, Chennai Batch 1

BALA KAVIN PON

2018AC04531, Chennai Batch 1

BALA VIJAY D

2018AC04514, Chennai Batch 1

BITS Pilani
Pilani | Dubai | Goa | Hyderabad
Work Integrated Learning Programmes

Table of Contents

Problem Statement

There are **N** different models of mobiles manufactured at a mobile manufacturing unit.
Each mobile must go through 2 major phases:

1. Parts manufacturing
2. Assembling

Obviously, 'parts manufacturing' must happen before "assembling'. The time for 'parts manufacturing' and 'assembling' (***pmi*** and ***ai*** for ***ith*** mobile) for every mobile may be different. If we have only 1 unit for 'parts manufacturing' and 1 unit for 'assembling', how should we produce n mobiles in a suitable order such that the total production time is minimized?

Requirement

1. Write a Greedy Algorithm to select the mobile parts manufacturing and assembling in such a way that total production time is minimized.
2. Analyse the time complexity of your algorithm.
3. Implement the above problem statement using Python.

Deliverables

1. Word document *designPS1_<group id>.docx* detailing your algorithm design and time complexity of the algorithm.
2. Zipped *AS2_PS1_MM[Group id]* package folder containing all the modules classes and functions for the employee node, binary tree and the main body of the program.
3. *inputPS1.txt* file used for testing
4. *outputPS1.txt* file generated while testing

Algorithm

Greedy Approach

A greedy algorithm approach is a simple, intuitive algorithm that is used in optimization problems. The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem. The greedy method is applied to optimization problems, which involve searching through a set of configurations to find one that minimizes or maximizes an objective function defined on these configurations.

The given problem statement on scheduling the tasks in mobile manufacturing is analogous to the ***Task Scheduling*** problem in Greedy method paradigm. The conventional Task Scheduling problem is to schedule all the tasks in T on the fewest machines possible in a non-conflicting way. Whereas here, the given problem denotes that suppose a mobile has to be manufactured it needs to go through 2 stages of process, one is parts manufacturing(***pmi***) and second is assembling(***ami***) the manufactured parts. The assembly unit will have to wait for the parts manufacturing unit to finish and send it for assembling the parts. This will lead to an Idle time in the assembly unit when there are no mobile parts are finished with manufacturing or the manufacturing process take longer time. The objective of the algorithm is to find the optimal sequence of mobiles to be manufactured that will have minimum idle time of assembly unit.

This document describes the design of algorithm to solve problem of finding the optimal sequence of mobile to manufacture with minimal idle time of assembly units.

Algorithm Design:

Input:

The input is a list of tasks. The tasks will be in the structured form of triplets with a delimiter (**/**). For instance, **1/5/7**, each data in the triples will have to be mapped to ***Task ID, Part Manufacture Time*** and ***Assembling Time***.

Output:

The expected output from the algorithm is to provide an optimal sequence of tasks that will take only minimal idle time in ***Assembly unit***, thus the production time will be minimized.

Functional Elements:

The functional elements of the algorithm indicate the ***functionally significant logics*** that are need to complete the operations of the algorithm. There could be one or more such significant elements in an algorithm which plays a vital role in constructing the algorithm with minimal running time complexity.

There are two functionally significant elements required for constructing the algorithm,

1. Sorting the list of tasks based on parts manufacturing time and assemble time.
2. Selection of tasks in sequence to have minimal idle time
3. Calculating idle time based on the parts manufacturing time and assemble time.

Task Structure

The Input tasks will be in a structure of delimited string of integer. It should be fetched and split based on the delimiter '/' and used for further process in the program. It has to be in a data structure so that it will be handled cohesively as a single unit of element at a time. We used a class for storing this data and the instances of the class had been taken into the list in the further part of the program. The structure of the task is as follows,

```
Class Task{

	TaskId : An unique identity number of the task in the given list
	ManufactureTime : Part manufacturing time of the mobile
	AssembleTime : Mobile assembly time
}
```

Task Merge Sort Algorithm

```
Algorithm MergeSortTasks(TaskList:List of given task objects) {
	Input: List of task object items with manufacture and assemble time
	Output: Sorted list of tasks

	If size(TaskList) > 1{
		Mid := TaskList.size()/2
		Left := TaskList[0 -> Mid]
		Right := TaskList[Mid -> size(TaskList)]

		MergeSortTask(Left) # Recursive Call
		MergeSortTask(Right) # Recursive call

		i := j := k := 0

		# Comparison and swap operation
		While(i < Left.size() && j < Right.size()){
			If (Left[i].manufacture_time < Right[j].manufacture_time) {
				TaskList[k] = Left[i]
				i++
			}
			Else if(Left[i].manufacture_time == Right[j].manufacture_time){
				If(Left[i].assemble_time < Right[j].assemble_time){
					TaskList[k] = right[j]
					j++
				}
				Else{
					TaskList[k] = left[i]
					i++
				}
			}
			Else{
				TaskList[k] = Right[k]
				k++
			}
		}
	}
} Return: Sorted TaskList list.
```

Schedule Algorithm

Schedule function takes is a controller function which takes the list of tasks as an input and orchestrate the required function calls to perform the operation of optimal scheduling sequence with minimal idle time.

```
Algorithm Schedule(TaskList:List of given task objects, OutputFile: File path of output file){
	Input: List of task object items with manufacture and assemble time

	JobList := TaskList # Making a copy of task list

	# Invocation of merge sort in an ascending order of manufacturing time
	MergeSortTasks(JobList)

	# Invoking a method to calculate the Idle time and Total production time
	(JobSequence, TotalTime, IdleTime) := GetRuntimeAssemblyIdleTime(JobList)

	# Invocation of print the results into an output file
	WriteResult(JobSequence, TotalTime, IdleTime)

}
```

GetRunTimeAssemblyIdleTime Algorithm

```
Algorithm GetRunTimeAssemblyIdleTime(TaskList:List of given task objects){
	Input: List of task object items with manufacture and assemble time
	Output: Job sequence list, Total production time and idle time

	JobSequence := List()
	RunTime := AssembleTime := IdleTime := 0

	For i to size(TaskList){
	Begin:
		JobSequence[i] := TaskList[i].TaskId # Adding task id into a sequence
		RunTime += TaskList[i].ManufactureTime
		TimeDiff = Runtime - (IdleTime + AssembleTime)

		If (TimeDiff > 0){ # If the Timediff is +ve add to Idle time
			IdleTime += TimeDiff
			TimeLag := 0
		}
		Else{
			TimeLag := -TimeDiff # If time difference is -ve then it's a forward lag
		}

		AssembleTime += TaskList[i].AssembleTime
	End:
	}

	TotalTime := Runtime + TimeLag + TaskList[n].AssembleTime # Sum of all the time elements

	Return (JobSequence, TotalTime, IdleTime) # Return the values as Tuple

}
```

Program Execution

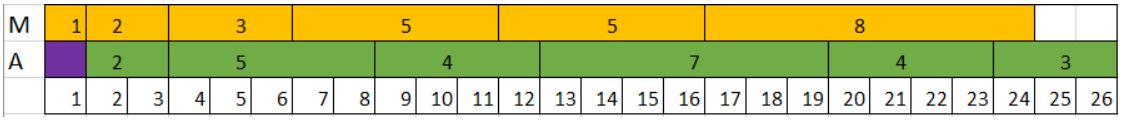
The program is made in python module with 2 class implementation. The program accepts the input as text file with a sequence of triplet numbers delimited with '/' character. The following section describes the program executions for input dataset and the resultant output.

DATASET 1

Input data

```
1/5/7
2/1/2
3/8/3
4/5/4
5/2/6
6/3/5
```

Task Sequencing and wait time



Output result

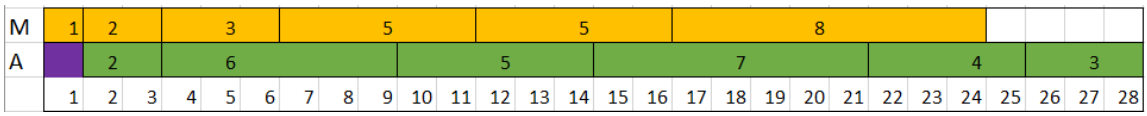
```
Mobiles should be produced in the order: [2, 5, 6, 1, 4, 3]
Total production time for all mobiles is: 28
Idle Time of Assembly unit: 1
```

DATASET 2

Input data

```
1/5/2
2/6/4
3/7/9
4/8/7
5/9/3
6/10/4
```

Task Sequencing and wait time



Output result

```
Mobiles should be produced in the order: [1, 2, 3, 4, 5, 6]
Total production time for all mobiles is: 49
Idle Time of Assembly unit: 20
```

Asymptotic Analysis

Asymptotic analysis in algorithm analysis focuses on the growth rate of the running time as a function of the input size n. Here the program has two important algorithms, which take the input size n. The algorithms are ***MergeSortTasks*** and ***GetRunTimeAssemblyIdleTime*** algorithm

Analysis on MergeSortTasks

INPUT SIZE: n – number of elements in the list

RECURSIVE OPERATION – Yes at **O(logn)**

LOOP OPERATION – Yes at **O(n)**

TIME COMPLEXITY – The algorithm performs the sorting operation by dividing the list by halves recursively. Along with that it also performs the swaps and merge operation in loop. Thus the total time complexity of the algorithm would be **O(n) x O(logn) = O(nlogn)**

Analysis on GetRunTimeAssemblyIdleTime

INPUT SIZE: N – Sorted array elements of object

RECURSIVE OPERATION – No

LOOP OPERATION – Yes at **O(n)**

TIME COMPLEXITY – The algorithm performs just addition operation on the set of numerical variables. The list will be iterated from 0 to N size of the list. The addition operations were performed inside the loop. Hence the time complexity of the algorithm would be **O(n)**

Total Time Complexity of the Program

The entire program has three major functionalities to complete the desired operations. The three major functionalities are,

```
Mobile manufacturing job
scheduling program = (Fetch input from file + Merge sort on tasklist + Idle time calculation)

Time complexity   =  [ O(n) + O(nlogn) + O(n)] => O(nlogn)
```

Therefore, the whole program to find the optimal job sequence with minimal idle time runs in **O(nlogn)** time.

Thank You