Assignment Three

This is an individual assignment. In this assignment, you will implement a task scheduler in Java.

Background

An embedded system is a computer system performing dedicated functions within a larger mechanical or electrical system. Embedded systems range from portable devices such as Google Glasses, to large stationary installations like traffic lights, factory controllers, and complex systems like hybrid vehicles, and avionic. Typically, the software of an embedded system consists of a set of tasks (threads) with timing constraints. Typical timing constraints are release times and deadlines. A release time specifies the earliest time a task can start, and a deadline is the latest time by which a task needs to finish. One major goal of embedded system design is to find a feasible schedule for the task set such that all the timing constraints are satisfied.

Task scheduler

We assume that the hardware platform of the target embedded systems is a single processor with m identical cores. The task set $V=\{v_1, v_2, \ldots, v_n\}$ consists of n independent tasks. The execution time of each task is one time unit. Each task v_i (i=1, 2, ,,, n) has a release time r_i and a deadline d_i ($d_i>r_i$). All the release times and deadlines are non-negative integers. You need to design an algorithm for the task scheduler and implement it in Java. Your task scheduler uses EDF (Earliest Deadline First) strategy to find a feasible schedule for a task set. A schedule of a task set specifies when each task starts and on which core it is executed. A feasible schedule is a schedule satisfying all the release time and deadline constraints.

The EDF strategy works as follows.

- At any time t, among all the ready tasks, find a task with the smallest deadline, and schedule it on an idle core. Ties are broken arbitrarily. A task v_i (i=1, 2, ,,, n) is ready at a time t if $t \geq r_i$ holds.

It can be shown that the EDF strategy is guaranteed to find a feasible schedule whenever one exists for a set of independent tasks with unit execution times.

An Example

Consider a set of 10 independent tasks whose release times and deadlines are shown in Table 1. The target processor has two identical cores. A feasible schedule of the task set is shown in Figure 1.

Table 1: A set of tasks with individual release times and deadlines.

Task	Release time	Deadline
v_1	0	4
v_2	1	4
v_3	1	4
v_4	1	3
v_5	1	3
v_6	2	3
v_7	4	6
v_8	4	6
v_9	4	7
v_{10}	5	6

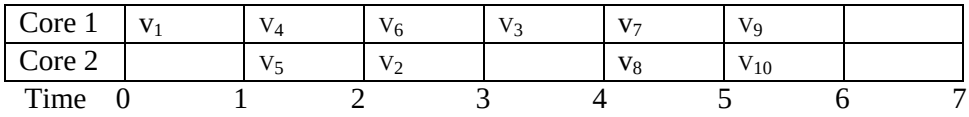

Figure 1: A feasible schedule.

In this example, if the target processor has only one core, no feasible schedule will exist.

The TaskScheduler class

You need to write a task scheduler class named **TaskScheduler**. A template of the **TaskScheduler** class is shown as follows.

```
public class TaskScheduler
{
  …
  static void scheduler(String file1, String file2, int m) {};
  …
}

class ClassX {}  /** Put all the additional classes you need here */
…
```

You can define any fields, constructors and methods within the TaskScheduler class. You can also define additional classes. You must put all the additional classes in the file Taskscheduler,java without class any class modifiers.

The main method scheduler(String file1, String file2, int m) gets a task set from file1, constructs a feasible schedule for the task set on a processor with m identical cores by using the EDF strategy, and write the feasible schedule to file2. If no feasible schedule exists, it displays " No feasible schedule exists" on the screen.

Both file1 and file2 are text files. files1 contains a set of independent tasks each of which has a name, a release time and a deadline in that order . A task name is a string of letters and numbers starting with a letter. All the release times are non-negative integers, and all the deadlines are natural numbers. The format of file1 is as follows.

v_1 r_1 d_1 v_2 r_2 d_2 … v_n r_n d_n

Two adjacent attributes (task name, release time and deadline) are separated by one or more white space characters or a newline character. A sample file1 is shown here.

For simplicity, you may assume that all the task names in file1 are distinct.

This method needs to handle all the possible cases properly when reading from file1 and writing to file2. All the possible cases are as follows.

1. file1 does not exist. In this case, print "file1 does not exist" and the program terminates.
2. file2 already exists. In this case, overwrite the old file2.
3. The task attributes (task name, release time and deadline) of file1 do not follow the formats as shown before. In this case, print "input error when reading the attributes of the task X" and the program terminates, where X is the name of the task with incorrect attributes.

file2 has the following format.

v_1 t_1 v_2 t_2 … v_n t_n

where t_i is the start time of the task v_i (i=1, 2, …, n) in the schedule. In file2, all the tasks must be sorted in non-decreasing start times. A sample file2 is shown here. Notice that you do not need to include the core on which each task is executed.

The maximum deadline of all the tasks can be an arbitrary value. This assumption implies that if you use bucket sort to sort all the tasks in non-decreasing order of their deadlines, the time complexity will be O(n+N) which can be higher than O(nlogn).

Time complexity requirement

You need to include your time complexity analysis as comments in your program. The time complexity of your scheduler is required to be no higher than O(n log n), where n is the number of tasks (**Hints: use heap-based priority queues**). **You need to include the time complexity analysis of your task scheduler in the TaskScheduler class file as comments.** There is no specific requirement on space complexity. However, try your best to make your program space efficient.

Restrictions

You can use net.datastructures-4-0. All the other data structures and algorithms must be implemented in the TaskScheduler class. You are NOT allowed to use any sorting algorithms and priority queues provided by Java.

Bonus marks for an optimal scheduling algorithm

You can get **3 bonus marks** if you propose an O(n) time algorithm for this scheduling problem and implement it correctly assuming that the maximum deadline of all the task is O(n), where n is the number of tasks. You need to perform a detailed time complexity analysis to show your proposed algorithm runs in O(n) time.

Hint: You need to do the following:

1. Use bucket sort together with the disjoint set union-find data structure using both the union by size heuristic and the path compression heuristic. You need to include all the code in the same file, either defining all the additional classes as nested classes or not using any class modifiers for all the additional classes.

2. Read the following reference:
   - H.N. Gabow and R.E. Tarjan . A linear-time algorithm for a special case of disjoint set union. Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 246–251.

   You can download this paper from ACM Digital Library via UNSW library, or download a similar version here. You don't need to fully understand this paper. What you need is to know the following conclusion:
   - If the unions follow a chain structure, the time complexity of a sequence of n union and find operations is O(n).

Please indicate at the beginning of your file that you implement a linear-time scheduling algorithm.

How to submit your code?

Follow this link: https://cgi.cse.unsw.edu.au/~give/Student/give.php. Do the following:

1. Use your z-pass to log in.
2. Select current session, COMP9024 and assn3.
3. Submit TaskScheduler.java.

Marking

Marking is also based on the correctness and efficiency of your code. Your code must be well commented.

The full mark of this assignment is 7 if the time complexity of your scheduler satisfies the above-mentioned requirement. Otherwise, the full mark will be 0. You will get 3 bonus marks for a correct O(n) time scheduling algorithm with a detailed time complexity analysis.

Deadline

The deadline is **11:59:59 pm, 9 October**. No late submission will be accepted.