Exercise 2

Introduction

The exercise requires an implementation for a scheduler for a CPU. The scheduler must manage a series of jobs, characterized by a name and a priority, that can increase over time. The scheduler must support these operations:

- Add a new job in the scheduler;
- Assign a job to the CPU;
- Update priorities for each scheduled job

Choice of the ADT

First, we chose the ADT to be used to represent the scheduler. Our choice fell on a heap priority queue because the jobs must be served from the ones with the highest priority to the lower and from the first scheduled to the last in case of same priority. In particular, we chose an *adaptable heap priority queue*, in order to handle the operation of updating priorities for each job in the queue (operation not supported in the basic version of heap priority queue). In addition, considering that the job priority is an integer between -20 (highest priority) and 19 (lowest priority), the adaptable heap priority queue (but also the basic version) natively supports a remove_min operation, which removes the element (i.e. the job in our case) that has the lowest key (i.e. the highest priority).

Class *Job*

The class Job has 3 attributes:

- Name: the name of the job
- Length: the number of time slice that are needed to process this job
- Waiting_slice_time: the number of time slice that he had passed in the scheduler until that moment

It supports the classic operations to get its attributes, and two set methods for the length and waiting_slice_time attributes.

Class *Scheduler*

The class scheduler is based on the ADT heap priority queue. It has two attributes:

- max_waiting_time: the maximum number of time slice that a job can wait before increasing its priority (by default is equal to 4)
- scheduled: the job that is actually scheduled to the CPU.

The class provides these public behaviours:

- **add(job, priority):** it takes in input a *job* and a *priority*. It first checks if the job and the priority are valid, then it eventually calls the default function of the adaptable heap priority queue in order to insert the job in the data structure. It output the name of the job added to the scheduler, with its length and priority
- **assing_job:** if the scheduler is not empty and there is no job currently assigned to the CPU (i.e. the scheduled attributed is *None*), it call the default remove_min function of the adaptable heap priority queue, and assigns the returned job (that is the one with the highest priority among the ones

available in the data structure) to the CPU (assign the job to the attribute *scheduled* of the scheduler). It outputs the job assigned.

- **update_job_priorities**: if the scheduler is not empty, for each job the function increases its waiting_time_slice counter by one. If this number reaches the max_waiting_time threshold, the counter is set to 0 and the priority of the job is increased by one, calling the default update function of the adaptable heap priority queue
- **update_scheduled_job:** if there is a job scheduled, it outputs the name of the job running on the CPU, the it updates the length of the scheduled job, reducing it by one, and if this value becomes equal to zero, the job is completed (and the scheduled attribute is set to *None*). Otherwise, it outputs that there are no job currently running on the CPU.

Main

A simulator is provided in order to test the scheduler. In an infinite loop (each cycle is a time slice), it first updates the scheduled job calling the *update_scheduled_job*. Then, it eventually assigns a new job to the CPU by the *assing_job* function. After that, if a random generated bit is equal to 1, it creates a new job with a random length and priority using the support method **generate_job** and adds it the scheduler by the *add* function. Otherwise, the simulator outputs that there is no new job this slice. Finally, it updates all the priorities of the jobs present into the scheduler, calling the *update_job_priorities* function. At the end of each cycle, the simulator waits 3 seconds in order to make the output clearer and less muddler.

Computational and Space Complexity

As said before, the scheduler methods call inside the default ones of the adaptable heap priority queue, so the time complexities of the functions are unchanged. For a scheduler with **n** jobs:

Space Complexity: O(n)

Function	**Time Complexity**
add(job, priority)	O(log(n))
assign_job()	O(log(n))
update_job_priorities()	O(log(n))
update_scheduled_job()	O(1)