The 2023 Internship Software Challenge

This document is designed to be a test of your imagination. As such, you are free to solve the challenge in any way you see fit, and using the programming language with which you are the most comfortable.

The input data for the various stages of the challenge has been included in the package, and the output schema is loosely defined for each of the four stages.

Your task is to design software that schedules operations on parts in the most efficient way possible. Efficiency, in this context, is measured by how quickly a set of parts is run through all of the intended operations. Some of you may recognize this as the classic **job shop problem**.

There are four stages to this challenge and you may solve how many of them you see fit.

Stage One – Starting Off Slowly

The input file for this part is called "**Input_One.txt**". It contains a list of available job shop machines and their features and a list of parts that need processing.

This first stage is simple: write software that reads the given file and parses it to store all the relevant data in memory. The file is easily human-readable, which usually means that parsing it into a program involves looking out for spaces, blank lines, missing fields and ancillary characters.

Your application should take in the given file and, after processing it, display its contents on screen, with the available machine features and part operations all visible. It should be clear to the user what machines we have at our disposal, what those machines can do and what sort of parts are on our work list. The various operations for each of the parts should also be listed. A console application will suffice for now, no need to invest additional time in a graphical user interface.

Some remarks:

- The code that we write is intended for users, not just for developers. If you come up with an application, it should be as user-friendly as possible. The simple app that you write for this first stage should be able to accept a file path pointing to where the input data is.
- An extension of the previous bullet-point is that the application should also be hardened against unusable inputs. This means that poorly formed file paths or non-standard input data should not crash your application. Instead, the user should be notified about the problem.
- These remarks apply to all of the steps in this challenge.

Stage Two – Simple Scheduling

Using the same input file as before (“**Input_One.txt**”), your application should create a simple processing schedule for all of the required parts. **The goal is to minimize the total time it would take to process all of the required parts**.

This scheduling output should be displayed on screen in a simple form for each part, detailing the start and end times for each of the part’s operations. The total processing time for the whole work list should also be displayed.

Some constraints and remarks:

- **The order of operations on parts is important and it is fixed** in the worklist.
- **Processing a part should be a contiguous affair**. Once the first operation starts on a part, all of the other operations on that part must be run without any pauses in between them, all the way to the end of a part’s process.
- The start time is 00:00:00 (hours:minutes:seconds) and the start and end times for each part operation should also be displayed in this format, counting up from the start time, of course.
- The machines’ capacity ratings are the limiting factors here. Ideally, you’d run all the parts in parallel and save time, but **the machines in this example can only accept one part at a time**. This means that, while a door knob is being processed on Machine Two, no pens can be processed on that machine, for example. Also, parts cannot be processed during cooldown times.
- Part processing does not have to be done in the order in which the parts are listed in the input file. Saving time is our goal, so maybe we could rearrange the work list for greater efficiency.

Stage Three – Flexible Scheduling

The input for this stage is “**Input_Two.txt**”. Machines have realistic names in this file (some with spelling mistakes), and we have two new machines with multi-part capacity. Our work list has also increased.

The goal remains the same: schedule operations for the given parts in order to minimize total processing time.

However, **processing a part no longer needs to be contiguous**. Part operations **must still be run in the order that’s given in the input file**, but the operations may be spread out across the schedule if it optimizes the total processing time. So, for example, if a pen must be processed on a mill after going through the lathe, your software may insert a pause between the two operations if it is more beneficial to have a different part processed on the mill right after the pen is done on the lathe. The pen’s mill operation may be run after the other part is done, if it helps the total time.

The output for this stage should be displayed in the same manner in which it was displayed for the previous stage.

Stage Four

A graphical user interface that displays your application’s scheduling output. The design is up to you. Good luck!