CS 430 PROJECT

November 30, 2018

Dylan Arguelles and Parker Joncus

Problem Description

Consider m machines and $n \geq m$ jobs, each of which is specified by a start time and a finish time. Each job can be assigned to any machine, but each machine can serve at most one job at any time. The objective is to schedule a largest number of given jobs to the m machines. Develop a polynomial time algorithm and write a program to implement it.

Description of Solution

Our idea for the algorithm is to take a greedy approach. In order to get a maximum number of jobs completed, we need to schedule the maximum number of jobs on each machine individually. If each machine has the maximum number of jobs that it can be performed on it, then there will be no job such that we can schedule it on a machine to increase the total number of jobs completed. We obviously cannot schedule a single job on multiple machines, so once a job is scheduled, it will be removed from the list of jobs that need to be completed. The plan is to run the greedy algorithm on each machine individually, one at a time. Once the greedy algorithm completes for a machine, the jobs that it outputs to schedule will be scheduled on the machine and removed from the list of jobs that need to be completed.

Pseudocode

```
Read txt files and store job start and finish times and number of machines.
Sort jobs by finish times as stack Jobs
For each machine:
    machine.jobs=Jobs[1] give the current machine the first job on the stack
    remove Jobs[1] from Jobs
    for j in Jobs:
        if job j has start time > finish time of machine.jobs[last]:
            machine.jobs=Jobs[j]
            remove Jobs[j] from Jobs
write total jobs performed and the list of jobs on each machine to a text file
```

Test Cases

We tried many different test cases to see if our algorithm would work as expected. They can be found with our code where the input files are called *input.txt* and the results from the algorithm are *draftOutput.txt* with the respectful numbers. When creating these test cases, we wanted to try to give our algorithm both random inputs and the extreme case inputs. The first test *input.txt* is a random example that we created. We also calculated the expected result by hand and our algorithm found the same answer. The next test *input1.txt* is a special case where all of the jobs have the same start and finish times, so each machine will only get one job. Next, with *input2.txt* we used the example give from the TA on blackboard to make sure we got the same answer as the TA. Our algorithm does get the correct answer, the only difference is our algorithm assigns the jobs to different machines than the TA's, but we are assuming the machines are equal so this does not matter. After that we have *input3.txt* which is the case where all jobs can be scheduled on 1 machine, and the jobs are listed in order of how they should be scheduled. Our algorithm works as expected and schedules the jobs on just the first machine and none on the second. We next have *input4.txt* which is the same as *input3.txt*, but the order that we list the jobs are flipped. This is to check that our algorithm will find the jobs in order no matter of what position the algorithm reads the jobs. Our algorithm outputs the same result as *input3.txt* which is what is expected. *input5.txt* and *input6.txt* are random created tests where we are trying to increase the size of the jobs and the machines to see if our algorithm still holds up. Last is *input7.txt* which has jobs that overlap with the adjacent jobs. There are two machines that can be scheduled and therefore the expect output is that one machine gets the even jobs while the other gets the odd jobs. All of our test cases completed as expected.

Proof of Correctness

Proof by Contradiction: Suppose our algorithm returns a list of jobs on each machine and the total number of jobs performed is N. Suppose there is some other combination of jobs on the machines such that $\tilde{N}$ jobs are performed and $\tilde{N} > N$. This means there exists at least 1 machine such that it does not maximize the number of jobs that can be performed. This is a contradiction since the greedy algorithm that is used on each machine selects jobs such that it maximizes the number of jobs that can be performed

on the individual machine. Since each individual machine has the maximum number of jobs that it can perform, then the entire algorithm will maximize the total number of jobs that are being performed. Therefore $\tilde{N} \leq N$ Since our algorithm goes through each machine one at a time and then deletes the jobs that are assigned to it, there will be no job that is scheduled on two machines and each machine will run the maximum number of jobs that are available at the time where jobs are being assigned to it.

Analysis of Running Time

The base of our algorithm is the greedy algorithm. The greedy algorithm has a time complexity of $O(n\ log\ n)$. Our algorithm then performs the greedy algorithm on each machine through a for loop and so in total our algorithms time complexity is $O(mn\ log\ n)$. The reading and writing of the input files will be $O(n)$ and $O(m)$ but since these are performed outside of the loop, we will focus on the higher order time complexity of the algorithm.

References/Appendix

All of our code can be found in the zip file and at:
https://github.com/argueless/machineJobScheduler