Joe Opitz

Professor Pach

4/23/2024

CSCI 332 – Semester Project

**Semester Project Documentation**

My name is Joe Opitz, I am a sophomore from Missoula, Montana. Here at Tech, I am studying Data Science while playing on the golf team. I enjoy spending time outside whether that is golfing, hiking, or just about anything. I was never extremely interested in coding, up until I started getting into the upper-level classes and began doing things that interest me, such as Machine Learning and Artificial Intelligence (AI). So, I am excited to take your AI class next fall.

The Greedy Task Scheduler algorithm is not exactly attributed to one specific person, because it is looked at as a fundamental concept within the Computer Science world. Once Greedy algorithms were discovered around the 1950’s, many researchers began working on finding many ways to apply it. One of the early creators of greedy algorithms was Edsger Dijkstra, where he developed Dijkstra’s shortest path algorithm in the late 1950s.

I decided to implement the Greedy Task Scheduler algorithm for my Graphic User Interface (GUI) project. I chose this, because I believe that I can use for personal use to help boost my productivity while gaining some experience with GUIs. I changed a couple things regarding the logic of the algorithm to better suit my needs for the project. The way that I set up the algorithm for this project, was implementing a GUI that consists of 4 text boxes where the user can input certain tasks that they need to complete.

I have the user supply the GUI with the name of the task, the time that they expect the task to take, how long until they need to have it done, and how much time on tasks today. Once all of those are entered you click “save task” and repeat that process until they have entered all the tasks that they want to complete, which will put the task into a checkbox. The user then clicks “done”, which runs the implemented algorithm and sorts the checklist by what task the user should finish first. The way that it works is by figuring out which task is due first and recommends you complete that first, and does that for the rest of the task. Which is pretty similar to the original algorithm, so the way that I changed it was by taking a segment of a task that will not have time to finish and recommends starting on it for a given time. Once the user completes a given task, they can check the corresponding box, which will then in turn update the progress bar to let the user know how many more tasks they still need to complete.

For the algorithm itself, the time complexity is O(n logn) while the space complexity is O(n) where n is the number of tasks. The time complexity is O(n logn), because to sort the list of tasks is also O(n logn) while the rest is O(n), which results in an overall time complexity of O(n logn). The space complexity is found in a similar way, but where the sort takes O(n) and the rest takes O(1), which results in an overall complexity of O(n). Within the GUI, all methods except for calling the algorithm itself result in O(n) for both space and time complexity.