CS 477: Homework #7

Due on November 30, 2023

Dr. Nicolescu Section 1001

Christopher Howe

Problem 1

Greedy Strategy

The greedy strategy that was implemented to solve this problem is to sort the customers by their estimated styling times and pick the customers with the longest styling time first.

Implementation

```python
# Chris Howe
# HW 7 Salon Problem

# Greedy algorithm that serves the customers with the longest styling times first
def getLowestCompletionTime(customers):
    sorted_customers = sorted(customers, key=lambda x: x['s'], reverse=True)
    nextStartTime = 0
    completionTime = 0
    for customer in sorted_customers:
        serviceTime = customer['w'] + customer['s']
        jobDone = nextStartTime + serviceTime
        if jobDone >= completionTime:
            completionTime = jobDone
        nextStartTime += customer['w']
    return completionTime

def main():
    # w = wash time, only one customer can have their hair washed at a time
    # s = style time, all customers can be styled at the same time, overlapping here is OK
    # Optimizing earliest/smallest completion time, completion time is the time when all customers are
    customers = [
        {'w': 10, 's': 20},
        {'w': 15, 's': 25},
        {'w': 8, 's': 18},
        {'w': 9, 's': 1},
        {'w': 13, 's': 3},
        {'w': 1, 's': 9},
        {'w': 6, 's': 8},
        {'w': 2, 's': 12},
    ]

    print(f"lowest completion time: {getLowestCompletionTime(customers)}")


if __name__ == "__main__":
    main()
```

Proof

Given some generic optimal solution S and the greedy solution S' which represent their completion times, the greedy solution needs to have a completion time that is equal to or less than the generic optimal solution.

Problem 1 continued on next page. . .

The greedy solution has the customers with longer styling times go first. The generic optimal solution does not necessarily need to have the customers with the longest styling times first.
Since the washing times can only occur sequentially, the time required to complete all the washes is constant. The completion time therefore, is only variable on any styling times that finish after the all the customers have had their hair washed.
In both solutions there are customers i and j with w_i, s_i, w_j and s_j respectively. s_j is greater than s_i. In the greedy solution, customer j would come first. In the generic solution, s_j may not be first. If customers i and j were swapped in the greedy solution, the completion time would either increase or stay the same. If the two elements were near the beginning, then there would be no difference. However, if customer i started at or near the end of the day, swapping them with customer j would increase the completion time since customer j has a longer styling time. They could not decrease the time since the overall washing time is constant.
The optimality of the greedy choice is reflected in the figure 1.

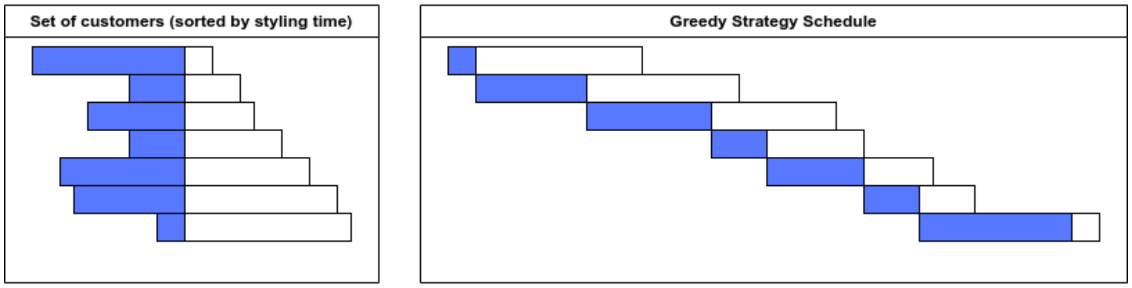

Figure 1: example of greedy algorithm acting on a set of customers.

Problem 2

Description

Given a set of jobs that need to be ran on a server, and a number n of identical servers, determine the lowest possible total time for how long each job spent in the system. This includes how long they wait on the server.

Greedy Strategy

The greedy strategy that was implemented to solve this problem is to sort the jobs by their required time and then pick jobs with the shortest times first, assigning each job to the server with the earliest availability.

Implementation

```
import numpy as np

# Greedy algorithm that serves the quickest jobs first to produce the lowest total time.
def findLeastTime(jobs, s):
    sorted_jobs = sorted(jobs)

    totalTime = 0
    serverOccupiedTimes = np.zeros(s)
    for job in sorted_jobs:
        jobWaited, nextAvailableServer = getNextReadyServer(serverOccupiedTimes)
        serverOccupiedTimes[nextAvailableServer] += job
        totalTime += job + jobWaited
    return totalTime

# Helper function that gets the next ready server
def getNextReadyServer(times):
    nextServer = -1
    nextServerTime = 100000000
    for serverNum, serverOpens in enumerate(times):
        if serverOpens < nextServerTime:
            nextServerTime = serverOpens
            nextServer = serverNum
    if nextServer == -1:
        raise Exception("Failed to get the index of the next ready server")
    return nextServerTime, nextServer



def main():
    # Company has s identical servers
    # There are n customers with jobs to run on the servers.
    # Each customer i has a job that takes ti time
    # Total time a job takes is t + ti where t is the time a job waits to be served.
    # Goal is to minimize the total of the total times for all the jobs.
    numServers = 3
    customerJobs = [5, 18, 12, 7, 14, 2, 9, 16, 1, 10, 8, 3, 19, 6, 11, 17, 4, 13, 20, 15]
```

Problem 2 continued on next page...

```
    print(f"least total time: ", findLeastTime(customerJobs, numServers))

if __name__ == "__main__":
    main()
```

Proof

In this problem, jobs completion times are tallied multiple times. This is because any job that occurs after a different job must also tally the previous job. For example, if job i has a completion time of 2 and job j has a completion of 4, if j occurs after i then the total completion time is 8.
Given some generic optimal solution S and the greedy solution S' which represent the total time for how long each job spent in the system , the greedy solution needs to have a total time that is equal to or less than the generic optimal solution.
The greedy solution runs the fastest jobs first, assigning them to whichever server is available first. The Generic optimal solution does not necessarily do the same.
In both solutions there are jobs i and j. Job i takes longer to complete than j. There are x jobs after i and y jobs after j. If in any solution, i and j are swapped total time would increase by $(i - j) * (x - y)$, the In the greedy solution, x should be greater than or equal to y since the jobs with the longest times are picked first. Therefore, in the greedy solution, If i and j were swapped, then the total time for all the jobs on the server would either increase or remain the same since x is greater than y.
The optimality of the greedy choice is reflected in the figure 2.

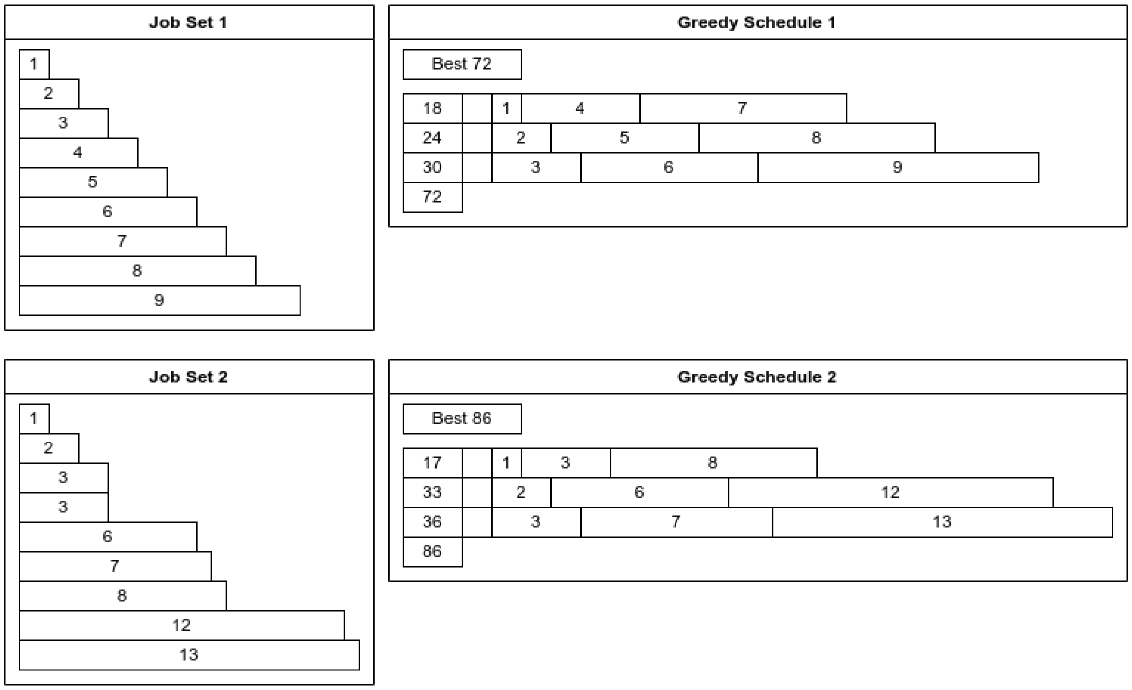

Figure 2: example of greedy algorithm acting on a set of jobs.

Problem 3

Problem

Consider the following jobs and service times. Use the algorithm you developed in problem 2 to minimize the total time spent in the system, if there is only one server available. Indicate what is the total time resulting from your solution.

Job	Service Time
1	7
2	3
3	10
4	5

Table 1: Job Service Times

Solution

Given the jobs above, the first thing my algorithm would do is to sort the jobs by their service time. This would produce the job array $[3, 5, 7, 10]$. The jobs would not needed to be divided among servers since there is only one. My algorithm would put all of the jobs on the same server. The first job would have $t = 3 + 0 = 3$. The second job would have $t = 5 + 3 = 8$. The second job would have $t = 7 + 8 = 15$. The third job would have $t = 10 + 15 = 25$. The total of all these times would be $3 + 8 + 15 + 25 = 41$