Lab 7

Let’s look at the Scheduling problem from section 4.3 in the text. We’ll write two Python functions that implement two greedy algorithms to solve this problem.

Say we have jobs that require a certain amount of time to run. We want to minimize the total amount of time the jobs spend in the system. The total time for a given job is its wait time plus its execution time. (This is something operating systems must develop sophisticated algorithms to implement, for example.)

Example: Job 1 takes 5 cycles, Job 2 takes 10 cycles, Job 3 takes 4 cycles.

Ordering these 1 – 2 – 3 leads to 39 total time (triple counting job 1 and double counting job 2 to account for wait time.) Ordering 3–1–2 leads to 32 total time. Make sure you can follow these calculations.

The goal is to design a greedy algorithm that solves this scheduling problem. Decide on a locally optimal selection condition, make the feasibility check, and compute the outcome.

Test on the following data set (job, cycles): (1, 7), (2, 3), (3, 10), (4, 5)

Next, consider a slightly more complex case.

Suppose the jobs must be scheduled by a certain cycle to obtain a reward. If the job is not scheduled, no reward is obtained. So, simply not scheduling a job at all if it cannot be scheduled by its deadline is optimal.

Design a greedy algorithm to maximize the reward. Decide on a locally optimal selection condition, make the feasibility check, and compute the outcome.

The text gives the example set (job, deadline, reward) as (1, 2, 30), (2, 1, 35), (3, 2, 25), (4, 1, 40). The schedule 4, 1 is optimal with reward of 70. Notice that jobs 2 and 3 aren’t scheduled. That’s fine: the goal is to get the maximum reward. In order to do this some jobs aren’t scheduled. Them’s the breaks.

Test your algorithm on the set (1, 2, 40), (2, 4, 15), (3, 3, 60), (4, 2, 20), (5, 3, 10), (6, 1, 45), (7, 1, 55).