German University in Cairo
Department of Computer Science
Dr. Nourhan Ehab

CSEN 703 Analysis and Design of Algorithms, *Winter Term 2025*
Practice Assignment 5

Exercise 5-1

Using the movie selection algorithm discussed in class. Get the maximum number of movies from the following set that can be scheduled.

a_i	1	2	3	4	5	6	7	8	9	10	11
s_i	0	2	3	4	6	8	11	16	1	21	22
f_i	4	5	7	7	8	15	15	20	25	26	28

Solution:

a_i	1	4	6	8	10
s_i	0	4	8	16	21
f_i	4	7	15	20	26

Exercise 5-2

Not just any greedy approach to the activity-selection problem produces a maximum size of mutually compatible activities. Give an example to show that the approach of selecting the activity of least duration from those that are compatible with previously selected activities does not work. Do the same for the approaches of always selecting the compatible activity that overlaps the fewest other remaining activities and always selecting the compatible remaining activity with the earliest start time.

Solution:

Least Duration Approach

a_i	1	2	3
s_i	3	1	4
f_i	5	4	8
d_i	2	3	4

In this case, selecting the least duration activity will select activity 1 only; however, an optimal solution can be achieved by selecting activities 2 and 3. Thus, it does not yield an optimal solution.

Fewest Overlaps Approach

a_i	1	2	3	4	5	6	7	8	9
s_i	1	2	2	3	4	5	6	6	7
f_i	3	4	4	5	6	7	8	8	9
d_i	2	2	2	2	2	2	2	2	2

In this case, selecting the fewest overlaps activity will select activities 1, 5, and 9; however, an optimal solution can be achieved by selecting activities 1, 4, 6, and 9. Thus, it does not yield an optimal solution.

Earliest Start-time Approach

a_i	1	2	3
s_i	1	2	4
f_i	5	3	5
d_i	4	1	1

In this case, selecting the earliest start time activity will select activity 1 only; however, an optimal solution can be achieved by selecting activities 2 and 3. Thus, it does not yield an optimal solution.

Exercise 5-3

Suppose a dentist has several patients waiting to be treated. Every patient i needs time t_i for treatment. The total time spent by all patients, both waiting and being treated, is referred to as the time in the system. A reasonable goal would be to minimize the time in the system.

i. Provide a greedy algorithm to reach this goal.

Solution:

```
1: function GREEDYSCHEDULING(Array t)
2:     Sort t ascendingly
3: return t
4: end function
```

ii. Prove that your algorithm is optimal.

Solution:

Assume that in the optimal schedule, patients are not ordered in increasing order of treatment time, then for at least one i where $1 \leq i \leq n-1$:

$$t_i > t_{i+1}$$

We can arrange our original schedule by interchanging the i^{th} and $(i+1)^{st}$ patients. By doing this, we have taken t_i units off the time the $(i+1)^{st}$ patient (in the original schedule) spends at the clinic, and added t_{i+1} units to the time the i^{th} patient spends at the clinic. Clearly, we have not changed the time any other patient spends at the clinic. Therefore, if T is the total time in the system in our original schedule and T' is the total time in the system in the rearranged schedule,

$$T' = T + t_{i+1} - t_i$$

Because $t_i > t_{i+1}$

$$T' < T$$

which contradicts the optimality of our original schedule. Therefore, the patients need to be ordered in increasing order of treatment time to give an optimal solution.

Exercise 5-4

Consider the problem of scheduling processes with deadlines and penalties for a single processor. In this problem, each process consumes one time unit. You are given the following inputs:

- a set $S = a_1, a_2, \cdots, a_n$ of n unit time tasks
- a set of n integer deadlines $d_1, d_2, \cdots, d_n$ such that each d_i satisfies $1 \leq d_i \leq n$
- a set of n nonnegative weights or penalties $w_1, w_2, \cdots, w_n$, such that we incur a penalty of w_i if task a_i is not finished by time d_i. No penalty is incurred if a task finishes by its deadline (time d_i).

i. Describe a greedy algorithm to minimize the penalties incurred for scheduling processes after their deadline.

Solution:

- Sort tasks with respect to penalties in decreasing order.
- While there are processes to be scheduled, check if there is a free slot before the current process deadline.
- If there exist free slots, place the process in the latest free slot before its deadline; otherwise, place the process in the latest free slot after its deadline.

ii. Prove that your algorithm is optimal.

Solution:

Let us assume a two algorithms A and A', for a subproblem $S_{ij} = \{a_i, a_{i+1}, \cdots, a_j\}$, where A is the result of applying our greedy approach, and A' is from not applying.

Let us consider the following scenarios that differ from the greedy approach for providing an optimal solution:

i. The task scheduled first before its deadline is not the one with the maximum penalty (based on a different heuristic).

If that is the case, then there exists a task with a higher penalty that will be scheduled later, which increases the chance of that task missing the deadline, which will incur a greater penalty than scheduling the maximum penalty first. making that solution suboptimal.

ii. Hence, we have to pick the maximum penalty first to schedule. If we are not using the greedy heuristic to schedule, we have 2 cases:

1. Placing the task at an arbitrary free time slot before its deadline.

This will not yield an optimal solution since the task might occupy the only free time slot for another task not yet scheduled with an earlier deadline.

2. Placing the task at an arbitrary free time slot after its deadline if none are found before it.

This again will not yield an optimal solution since the task may occupy the only free time slot for another task not yet scheduled that has a deadline higher than that of the current task and lower than the latest free slot after the current task deadline.

Exercise 5-5

Company x designed a multi-core processor that has one master m and $S = \{s_1, s_2, \cdots, s_m\}$ slaves. Each slave s_i can handle a process with a maximum complexity of c_i. At any given time t, the master gets a set of processes $P = \{p_1, p_2, \cdots, p_n\}$ and each p_i has a complexity d_i. Assuming that all the processes have the same running time and each slave can process only one task at a given time t, provide a greedy algorithm that helps the master assign the maximum number of processes possible to the available slaves.

Solution:

```
1: function SlaveAssignment(S,P)
2:     sort S and P in ascending order with respect to complexity
3:     i ← S.length
4:     j ← P.length
5:     assigned ← 0
6:     while i > 0 and j > 0 do
7:         if S_i.c ≥ P_j.c then
8:             i − −
9:             assigned + +
10:        end if
11:        j − −
12:    end while
13:    return assigned
14: end function
```