COMP3121/9101
Algorithm Design

Tutorial 4
The Greedy Method

Before the Tutorial

Before coming to the tutorial, try and answer these discussion questions yourself to get a firm understanding of how you're pacing in the course. No solutions to these discussion points will be officially released, but feel free to discuss your thoughts on the forum. Your tutor will give you some comments on your understanding.

- In this tutorial sheet, we focus on the second method of proving greedy algorithms – the *exchange argument.*
  - Briefly outline the *exchange argument* method of proof.
- Read through the problem prompts and think about how you would approach the problems.

Tutorial Problems

Problem 1. Recall the *activity selection* problem from lectures. That is, we are given n activities with starting times s_i and finishing times f_i. No two jobs can take place simultaneously and our task is to select the maximum size subset of compatible activities.

We provide six greedy algorithms. For each greedy algorithm, either explain why it works or provide a counterexample to show that it does not work.

(a) Keep choosing the activity x that finishes last and discard all activities which clash with x.

(b) Keep choosing the activity x that starts first and discard all activities which clash with x.

(c) Keep choosing the activity x that starts last and discard all activities which clash with x.

(d) If no activities clash, choose all of them. Otherwise, discard the activity with the longest duration and continue.

(e) Let x be the activity with the *earliest starting* time and y be the activity with the *second earliest starting* time.

- If x and y do not clash, then choose x and continue.
- If x completely contains y, then discard x and continue.
- Otherwise, discard y and continue.

(f) If any activity x completely contains another activity, discard x and continue. Otherwise, choose the activity y that finishes last, discard all activities that clash with y, and continue.

Problem 2. You are given n jobs and each job J_i has a non-negative processing time ρ_i. There is one person available to process each of the jobs. A schedule σ is an ordering of the jobs to be processed by the person. We define the *waiting* *time* of job J_i in schedule σ to be the sum of processing times of all jobs scheduled before J_i in σ. The *total waiting time* is the sum of the waiting times of all jobs in schedule σ.

For example, if we have six jobs in the order $J_1, J_2, J_3, J_4, J_5, J_6$ with processing times

J_1	J_2	J_3	J_4	J_5	J_6
3	4	1	8	2	6

then the total waiting time is

$$0 + 3 + (3 + 4) + (3 + 4 + 1) + (3 + 4 + 1 + 8) + (3 + 4 + 1 + 8 + 2).$$

(a) Design a greedy algorithm to find a schedule (i.e. an ordering of jobs) that minimises the *total waiting time.*

(b) To prove correctness, we proceed with an exchange argument.

- Construct a suitable definition of an inversion.
  - How many inversions does the greedy schedule have?
- Prove that, if a schedule has an inversion, then there exist an inversion between adjacently scheduled jobs.
  - What happens if two non-adjacent jobs form an inversion?
- Show that, by exchanging the position of two adjacent jobs, the new schedule is either just as optimal or strictly more optimal and has one fewer inversions.
- Hence, argue that your greedy algorithm is correct.

(c) Analyse the time complexity of your algorithm.

After the Tutorial

After your allocated tutorial (or after having done the tutorial problems), review the discussion points. Reflect on how your understanding has changed (if at all).

- In your own time, try and attempt some of the practice problems marked [K] for further practice. Attempt the [H] problems once you're comfortable with the [K] problems. All practice problems will contain fully-written solutions.
- If time permits, try and implement one of the algorithms from the tutorial in your preferred language. How would you translate high-level algorithm design to an implementation setting?