COL 100M - Lab Exam 1

February 3, 2018

Instructions

- This exam consists of two questions. The second question is a *bonus* question that you may attempt if you have time.
- No notes, phones, local or internet resources are allowed.
- Submit the following files:
  - `test1.ml`
  - `test2.ml` (for the bonus question)
- A password will be announced in class that you can use to submit code on Moodle. You will be allowed to submit / evaluate your code atmost 5 times.

1 Coin Denomination Problem

Submission file: `test1.ml`

Given a set of coin denominations, the coin denomination problem asks to compute the number of ways in which one can make N units, given that we have an infinite supply of each denomination. For example, given denominations of Rs. 1 and Rs. 2, there are 2 ways to make Rs. 3 – (1,1,1) and (1,2). Note that (2,1) is not included in the solution since the order in which the denominations are selected doesn't matter.

1. **[10 marks in total]** Write an OCaml function `coinChanger:int->int->int->int->int->int` that accepts 5 parameters - a target total amount, and 4 coin denominations. `coinChanger n a b c d` returns the *number of unique ways* in which `n` can be made from an unlimited supply of coin denominations in {`a, b, c, d`}.
   (a) **[2 marks]** Input validation : A *valid input* to `coinChanger` consists of positive integers for all `n, a, b, c, d`, where `a, b, c, d` are all unique. If the input is invalid, `coinChanger` should return `-1`.
   (b) **[8 marks]** On valid input, `coinChanger` must return an integer ≥ 0.
   - `coinChanger 10 1 2 3 4` returns 23.
   - `coinChanger 25 1 2 5 10` returns 64.
   - `coinChanger 50 2 5 10 20` returns 34.

   Hint: The following recurrence describes the number of ways in which amount n can be made from coin denominations a, b, c, d.

$$T(n, a, b, c, d) = T(n - a, a, b, c, d) + T(n, b, c, d)$$

$$T(n, b, c, d) = T(n - b, b, c, d) + T(n, c, d)$$

$$\ldots$$

   The above recurrence comes from the observation that at each step, we have the choice of selecting or not selecting the first available denomination. If we choose not to select a particular denomination, it is unavailable for any further selection. This allows to eliminate duplicate combinations.

2. **[5 marks]** (*Submission file:* `test2.ml`) If each coin denomination has an associated cost, then we have a two-fold objective: Coins selected should add up to the target value, and the total cost of the combination (sum of the costs of each coin selected) should be minimized. An example of this scenario is when each coin has a specific weight, and you wish to select the lightest combination of coins to carry. Alternatively, you may wish to minimize the total number of coins required, in which case each coin has an associated cost of 1. In the example in part 1, if each coin has cost 1, then (1,2) has smaller total cost (cost = 2) than (1,1,1) (cost = 3).

   Write an OCaml function `coinChanger_cost` that accepts 6 parameters – a target total amount, 4 coin denominations, and a cost function.

   `coinChanger_cost n a b c d f` returns the *minimum cost* of selecting coins in { `a, b, c, d` } to create target amount `n`, where the cost function is given by `val f : int -> int = <fun>`. As before, a valid input consists of positive integers for all `n, a, b, c, d`, where `a, b, c, d` are all unique. On invalid inputs return `-1`. On valid inputs, `coinChanger_cost` must return an integer *the minimum cost*, if there is atleast one way to achieve the target value. Otherwise if no possible solution exist, return `max_int`.

   If $f(x) = 2^x$, then

   - `coinChanger_cost 10 3 4 5 6 f` returns 32.
   - `coinChanger_cost 25 1 2 5 10 f` returns 50.
   - `coinChanger_cost 50 3 7 11 13 f` returns 352.