CPSC- Fall 2020

Problem Set 6 — Dynamic Programming

Total marks: 85

1. **(45 marks)** Question 2 on page 314-314 of the textbook.

   (a) **(5 marks)** Question 2(a)

   For Question 2(b), answer the following questions:

   (b) **(5 marks)** Give a formal definition (pre- and post-conditions) of the problem described.

   (c) **(5 marks)** Define the optimal substructure relationship by:

   - defining a suitable sub-problem
   - giving an expression for the optimal value for a problem instance in terms of optimal values of sub-problems

   (d) **(5 marks)** Give pseudocode for a recursive algorithm that computes the maximum value that can be earned.

   (e) **(3 marks)** Give a recurrence relation representing an upper bound on the worst-case running time of your recursive algorithm.

   (f) **(7 marks)** State and prove an explicit asymptotic upper bound on the worst-case running time of your recursive algorithm (using the guess-and-prove method).

   (g) **(10 marks**) Give pseudocode for a dynamic programming algorithm that computes the maximal value that can be earned *and* outputs a job plan that obtains it. Your algorithm may be a single algorithm or a combination of two algorithms (one to compute the optimal value and a second to recover the corresponding sequence of moves).f

   (h) **(5 marks)** State and prove a *tight* asymptotic bound on the running time of your dynamic programming algorithm.

2. **(40 marks)** Suppose that you are given an $n \times n$ checkerboard and a checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step you may move the checker to one of three squares:

   - the square immediately above,
   - the square that is one up and one to the left (but only if the checker is not already in the leftmost column),
   - the square that is one up and one to the right (but only if the checker i s not already in the rightmost column).

Each time you move from square x to square y, you receive $p(x, y)$ dollars. You are given $p(x, y)$ for all pairs (x, y) for which a move from x to y is legal. Do not assume that $p(x, y)$ is positive.

Give an algorithm that figures out the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. Your algorithm is free to pick any square along the bottom edge as a starting point and any square along the top edge as a destination in order to maximize the number of dollars gathered along the way. What is the running time of your algorithm?

Your solution to this problem must answer the following questions:

(a) **(5 marks)** Give a formal definition (pre- and post-conditions) of the problem described.

(b) **(5 marks)** A sub-problem in this context is to find the most profitable way to get from some square in row 1 to a particular square (i, j). Describe the optimal substructure relationship for this problem by giving an expression for $d[i, j]$, the maximum profit earned to square (i, j).

(c) **(5 marks)** Give pseudocode for a recursive algorithm that computes the maximum value that can be earned for a particular square with coordinates (i, j), as well as a non-recursive driver program that computes the maximal possible value that can be earned.

(d) **(3 marks)** Give a recurrence relation representing an upper bound on the worst-case running time of your recursive algorithm.

(e) **(7 marks)** State and prove an explicit asymptotic upper bound on the worst-case running time of your recursive algorithm (using the guess-and-prove method).

(f) **(10 marks**) Give pseudocode for a dynamic programming algorithm that computes the maximal possible value that can be earned *and* outputs a sequence of moves that obtains it. Your algorithm may be a single algorithm or a combination of two algorithms (one to compute the optimal value and a second to recover the corresponding sequence of moves).f

(g) **(5 marks)** State and prove a *tight* asymptotic bound on the running time of your dynamic programming algorithm.