Assignment #7

Due: March 25, 2022 (11:59 pm)
(Total 40 marks)

CMPUT 204
Department of Computing Science
University of Alberta

Some questions here are based on Problem Set #5 and our lecture notes. Discussions of related problems may also be found online. You may consult, adopt, or revise any of these solutions to compose your answers. But recall that you must understand what you submitted and be able to explain your answers. If your algorithm is the same as the one given in the Problem Set or lecture notes, just indicate so; otherwise please specify.

Problem 1. (15 marks)

a. (5 marks) Consider the 0-1 knapsack problem for items i = 1,2,3,4,5, whose values are $[4, 3, 6, 5, 7]$ and whose weights are $[2, 2, 3, 4, 5]$. Assume the knapsack capacity is 10. Apply the knapsack algorithm discussed in class to the problem by filling the 2-dimensional table $A[i, D]$ $(0 \leq i \leq 5, 0 \leq D \leq 10)$, and apply the Print-Opt-Knapsack procedure to show which items are in your optimal solution.

Solution: The table $A[i, D]$ below is partially filled, and you are to fill the remaining cells.

i \D	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	4	4	4	4	4	4	4	4	4
2	0	0	4	4	7	7	7	7	7	7	7
3	0	0	4	6	7	10	10	13	13	13	13
4	0	0	4	6	7	10	10	13	13	15	15
5	0	0	4	6	7	10	10	13	13	15	17

- Items 1, 3, 5 are in the optimal solution.

b. (5 marks) Find an LCS of strings "ABAECCD" and "ACDEF". You should show how you derived your solution using a table similar to Fig. 15.8 of CLRS p395.

Solution:

		A	B	A	E	C	C	D
	0	0	0	0	0	0	0	0
A	0	↖1	←1	↖1	←1	←1	←1	←1
C	0	↑1	↑1	↑1	↑1	↖2	↖2	←2
D	0	↑1	↑1	↑1	↑1	↑2	↑2	↖3
E	0	↑1	↑1	↑1	↖2	↑2	↑2	↑3
F	0	↑1	↑1	↑1	↑2	↑2	↑2	↑3

- The longest common subsequence is "ACD". Another possible way is to consider the string "ABAECCD" in the rows and the string "ACDEF" in columns. It will give another representation of the table values but the longest common subsequence will be same as "ACD".

c. (5 marks) Matrices A, B, C, D, E have respective dimensions 1×5, 5×2, 2×6, 6×1, 1×4. What is the minimum number of scalar multiplications needed to compute $A \times B \times C \times D \times E$? Give the parenthesization of matrices that yields this number. Show how you arrived at your answer.

Solution: Define $M[i,j](1 \leq i \leq j)$ as the minimum number of scalar multiplications needed to compute product $A_i \times A_{i+1} \times ... \times A_j (i \leq j)$. And matrix S is an array where $S[i,j] = k$ s.t. an optimal parenthesization split the product $A_i \times ... \times A_j$ between k and $k+1$. The matrix M and S are as below:

$$M = \begin{bmatrix} 0 & 10 & 22 & 24 & 28 \\ - & 0 & 60 & 22 & 42 \\ - & - & 0 & 12 & 20 \\ - & - & - & 0 & 24 \\ - & - & - & - & 0 \end{bmatrix} \quad S = \begin{bmatrix} 0 & 1 & 2 & 2 & 4 \\ - & 0 & 2 & 2 & 4 \\ - & - & 0 & 3 & 4 \\ - & - & - & 0 & 4 \\ - & - & - & - & 0 \end{bmatrix}$$

- The minimum number of scalar multiplications is $M[1,5] = 28$.
- The parenthesization of matrices is $(((A \times B) \times (C \times D)) \times E)$.

Problem 2. (8 marks)

In this question, you are asked to show how a problem instance is solved for the following problem given in Problem Set #5.

There are n Hudson's Bay Company posts on the River Koksoak. At any of these posts you can rent a canoe to be returned at any other post downstream (it is next to impossible to paddle against the current.) For each possible departure point i and each possible arrival point j the company's tariff gives the cost of a rental between i and j which is $C[i,j]$. However, it can happen that the cost of renting from i to j is higher than the total cost of a series of shorter rentals, in which case you can return the first canoe at some post k between i and j and continue the journey in a second canoe. There is no extra charge for canoe change. Give an efficient algorithm to determine, given n and costs $C[i,j]$, the sequence of canoe rentals with minimum cost for a trip from post 1 to post n.

- In this question, you are asked to fill the tables $A[i]$ and $S[i]$ $(1 \leq i \leq 6)$ as defined above for the following problem instance:

  $C[1,2] = 7,\ C[1,3] = 8,\ C[1,5] = 21$

  $C[2,3] = 5,\ C[2,5] = 13$

  $C[3,4] = 5,\ C[3,5] = 9,\ C[3,6] = 17$

  $C[4,5] = 8,\ C[4,6] = 15$

  $C[5,6] = 7$

  For all other pairs of $i < j$, $C[i,j] = \infty$, which means renting from i to j is not available. For part marks, we'd like to understand how you did your work. For this purpose, please explain how you filled the cells at $A[5]$ and $A[6]$.

- According to $S[.]$, which rental arrangement yields the minimum cost?

Solution:

i	A[i]	S[i]
1	0	-
2	7	1
3	8	1
4	13	3
5	17	3
6	24	5

- To fill $A[5]$, we compare the values of $A[j] + C[j, 5]$ for $1 \leq j < 5$. The costs will be 21 for $j = 1$, 20 for $j = 2$, 17 for $j = 3$ and 21 for $j = 4$. The minimum is then chosen, which in this case is 17. Similarly for A[6], the cost is ∞ for $j = 1$ and $j = 2$, 25 for $j = 3$, 28 for $j = 4$ and 24 for $j = 5$, so 24 is chosen.
- Renting from 1 to 3, 3 to 5, and 5 to 6 will yield the minimum cost of 24.

Problem 3. (17 marks)

LONGEST SUB-PALINDROME problem: given a sequence $X = \langle x_1, ..., x_n \rangle$ find the longest subsequence $\langle x_{i_1}, ..., x_{i_k} \rangle$ such that $i_j < i_{j+1}$ for any j, and that is a palindrome: k is even and the inverse sequence $\langle x_{i_k}, x_{i_{k-1}}, ..., x_{i_1} \rangle$ is identical to the original sequence $\langle x_{i_1}, ..., x_{i_k} \rangle$. (E.g., "abba" is a sub-palidrome of "tablebrand.") Note that the definition here differs slightly from the standard dictionary definition of palindrome; here we only consider palindromes whose length is an even number.

In this question, you are asked to present a complete solution to demonstrate all the ingredients of solving this problem by DP.

a. Denote the problem by $LSP(x_1, ..., x_n)$. Give a recursive definition to solve the problem. Argue that all of the recursive calls live in a small domain so that the DP approach is possible.

b. Turn your recursive definition to a table definition. Indicate the dimensions of your table, what each cell of your table stores, and which cell stores the value of an optimal solution (the value is the number of characters in a longest sub-palindrome of the given string).

c. Give an algorithm in pseudocode to compute (the value of) an optimal solution. Analyze its running time.

d. Give an algorithm in pseudocode to generate the actual string of an optimal solution. Comment on its running time.

e. Now consider the input string $S = \langle p, e, q, q, d, e \rangle$.

   (i) Draw the table according to your solution, and for each cell fill its value.

   (ii) According to your algorithm, which sub-palindrome is generated? Explain briefly how it is generated.

Solution:

a. The recursion is

$$LSP(x_1, ..., x_n) = \left\{ \begin{array}{cc} LSP(x_2, ..., x_{n-1}) + 2 & \text{if } x_1 = x_n \\ max\{LSP(x_2, ..., x_n), LSP(x_1, ..., x_{n-1})\} & \text{if } x_1 \neq x_n \end{array} \right\}$$

Every recursive call is on a subsequence $x_i ... x_j$ with $1 \leq i \leq j \leq n$. Thus, the total number of distinct recursive calls is $\frac{1}{2}n^2 + \frac{1}{2}n$. Therefore, the dynamic approach is possible.

b. We set a table M of size $\frac{1}{2}n^2 + \frac{1}{2}n$. Each cell $M[i,j]$ will hold the value $LSP(x_i, ..., x_j)$ for each $1 \leq i \leq j \leq n$, and the cell $M[1,n]$ will hold the value $LSP(x_1, ..., x_n)$ which is the value of an optimal solution. The base case is to set $M[i,i] = 0$ for all i. The formula for the i,j-cell is

$$M[i,j] = \left\{ \begin{array}{cc} 0, & \text{if } i = j \\ 2, & \text{if } j = i+1, x_i = x_j \\ 0, & \text{if } j = i+1, x_i \neq x_j \\ M[i+1, j-1] + 2, & \text{if } j > i+1, x_1 = x_n \\ \max\{M[i, j-1], M[i+1, j]\}, & \text{if } j > i+1, x_1 \neq x_n \end{array} \right\}$$

c. Start from filling all the base cases, i.e., cells $M[i,i]$ and $M[i,i+1]$. Then we will fill each cell $M[i,j]$, where $j \geq i+2$, with i from $n-2$ down to 1 and j from $i+2$ to n.

```
Procedure Longest-Sub-Palindrome(X)
   n ⟵ length[X]
   for (i ⟵ 1 to n − 1) do
      M[i, i] ⟵ 0
      j ⟵ i + 1
      if (x_i = x_j) then
         M[i, j] ⟵ 2
      else
         M[i, j] ⟵ 0
   M[n, n] ⟵ 0
   for (i ⟵ n − 2 downto 1) do
      for (j ⟵ i + 2 to n) do
         if (x_i = x_j) then
            M[i, j] ⟵ M[i + 1, j − 1] + 2
         elseif (M[i + 1, j] ≥ M[i, j − 1]) then
            M[i, j] ⟵ M[i + 1, j]
         else
            M[i, j] ⟵ M[i, j − 1]
   return M
```

In Procedure Longest-Sub-Palindrome(X), the first for loop takes $\Theta(n)$ time and the second nested for loop takes $\Theta(n^2)$ time. Thus, in total, the running time is $\Theta(n^2)$.

d. Call Print-LSP($M, X, 1, n, <>$), where $<>$ denotes an empty sequence.

```
Procedure Print-LSP(M, X, i, j, S)
   if (i > j) then
      return S
   elseif i = j then
      return <>
   elseif M[i, j] = M[i + 1, j − 1] + 2 then
      Print x_i
      Print-LSP(M, X, i + 1, j − 1, S)
      Print x_i
   elseif M[i, j] = M[i + 1, j] then
      Print-LSP(M, X, i + 1, j, S)
```

```
else
   Print-LSP(M, X, i, j − 1, S)
```

In Procedure Print-LSP$(M, X, 1, n, <>)$ takes $\Theta(n)$ time.

e. (i)

i / j	1(p)	2(e)	3(q)	4(q)	5(d)	6(e)
1(p)	0	0	0	2	2	4
2(e)	0	0	0	2	2	4
3(q)	0	0	0	2	2	2
4(q)	0	0	0	0	0	0
5(d)	0	0	0	0	0	0
6(e)	0	0	0	0	0	0

(ii) The sub-palindrome generated is "eqqe".