Design and Analysis of Algorithms (COMP3001)

Tutorial 9 + 10

Dynamic Programming:

- **0/1 Knapsack Problem**
- **Matrix chain multiplication**
- **LCS**

Question 1.

The 0/1 Knapsack problem is defined as follows. I have a backpack (knapsack) that can hold C kilograms of stuff. I also have n items that I want to put in the backpack. Item i weighs w_i kilograms and it has a profit/usefulness/utility of p_i. I want to put as many items in my backpack as I can so that my profit is maximised.

For example, say $C = 10$, $n = 5$, $w = \{2, 2, 6, 5, 4\}$ and $p = \{6, 3, 5, 4, 6\}$, then I want to put items 1, 2, and 5 in the bag.

a) Accurately define the 0/1 Knapsack problem in algorithmic and mathematical terms.
b) Think of one real life example of where application of the 0/1 Knapsack problem would be beneficial.
c) Why is it called the 0/1 Knapsack problem?

Question 2.

Give a greedy algorithm that might solve the 0/1 Knapsack problem, and show an example list of n items and a value for c where your algorithm will not work.

Question 3

The following recursive algorithm solves the 0/1 Knapsack problem.

KNAPSACK-RECURSE (i, k)

```
if (i = n) then
   if (w_n > k) then
      return 0
   else
```

```
        return p_n

    if (w_i > k) then
        return KNAPSACK-RECURSE (i+1, k)
    else
        x = KNAPSACK-RECURSE (i+1, k)
        y = KNAPSACK-RECURSE (i+1, k-w_i) + p_i
        return max (x, y)
```

a) Give the recurrence function, T(n), of the time complexity of KNAPSACK-RECURSE for the 0/1 knapsack problem with n elements. Explain your answer.

b) Show that the solution of the recurrence function T(n) in part a) is $O(2^n)$.

Question 4.

Consider the following set of 5 items:

$$W = [3, 4, 7, 8, 9]$$
$$P = [4, 5, 10, 11, 13]$$

a) Assuming a knapsack of size 17, use the following dynamic programming approach (discussed in the lecture) to find a way to fill in the knapsack with the highest possible value. [**Hint**: the highest value is 24, and the selected items are X = <0, 0, 0, 1, 1>].

   Knapsack (*S*, *C*)
   Input: Set S of n items with p_i profit and w_i weight, and maximum total weight C
   Output: maximum profit $P[w]$ of a subset S with total weight at most w, for $w = 0, 1, \ldots C$

```
for k = 0 to C do
    P[k] = 0

for i = n downto 1 do
    for k = C downto w_i do
        if P[k - w_i] + p_i > P[k] then
            P[k] = P[k - w_i] + p_i
```

b) The algorithm **Knapsack (*S*, *C*)** produces **only** value of the highest profit, e.g., 24. Suppose you also aim to generate the information about the items selected that produce the highest profit, e.g., X = <0, 0, 0, 1, 1>. Explain an algorithm to achieve the aim.

Question 5.

a) How many scalar multiplications are required to multiply a $p \times q$ and a $q \times r$ matrix?

b) Calculate by hand (using the result from Question 6(a) the number of multiplications that would be required by each of

(i) $(A_1 \times A_2) \times A_3$

(ii) $A_1 \times (A_2 \times A_3)$

where A_1 is a 100×10 matrix, A_2 is a 10×100 matrix, A_3 is a 100×10 matrix.

Which is the preferred bracketing?

Question 6.

a) Analyse the time complexity of the following dynamic programming algorithm for the matrix-chain multiplication problem.

Input: sequence $(p_0, p_1, \ldots p_n)$
Output: an auxiliary table $m[1..n, 1..n]$ with $m[i,j]$ costs and another auxiliary table $s[1..n, 1..n]$ with records of index k which achieves optimal cost in computing $m[i, j]$

```
n = length[p]-1;
for i = 1 to n
      do m[i, i] = 0;
for l = 2 to n
      do for i = 1 to n- l +1
            do j = i + l -1
               m[i, j] = ∞;
               for k = i to j -1
                   do q = m[i, k]+ m[k +1, j]+ p_{i-1} p_k p_j;
                        if q < m[i, j];
                           then m[i, j] = q;
                                s[i, j] = k;
return m and s
```

b) Work through the dynamic programming algorithm to find the optimal parenthesization of a matrix chain product whose sequence of dimensions is <5, 10, 3, 12, 5, 50, 6>

Question 7.

a) Use LCS_length (X, Y) on input X = <1, 0, 0, 1, 0, 1, 0, 1> and Y = <0, 1, 0, 1, 1, 0, 1, 1, 0>.

b) From the obtained table b, construct the LCS.

c) From the obtained table c, X and Y, construct the LCS.

Question 8.

Textbook: Exercise 15.4-5. Give an $O(n^2)$ time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

Example: for n=8 and X = <4,5,2,3,4,7,3,5>, your algorithm produces Z = <2,3,4,7> or <2,3,4,5>.

Question 9.

Which approach, the top-down, or the bottom-up, dynamic programming is better to solve LCS? Why?