Algorithms Example Sheet 2: Core Questions

1 Dynamic programming

Recommended reading/useful links:

- CLRS Chapter 15 (except for 15.5)
- LCS visualisation
- Collection of videos explaining some DP problems
- Various DP visualisations
- Jeff Erickson's Algorithms Chapter 3
- Algorithms' Illuminated: Part 3
- An article on knapsack problems.

Exercise 2.C.1 [Fibonacci Numbers]

(a) Explain how dynamic programming can be used to efficiently compute the Fibonacci numbers.

(b) Write pseudocode and explain the difference between the bottom-up and top-down approach.

(c) What is the time and space complexity of each approach?

(d) (optional) Implement one of the two DP algorithms.

Exercise 2.C.2 [Rod cutting problem]

(a) Read section 15.1 from CLRS and define the *rod cutting problem*.

(b) Design an algorithm to solve it.

(c) (optional) Implement the algorithm. You can test your implementation on **[GeeksForGeeks Rod Cutting]**.

Exercise 2.C.3 [0/1 **Knapsack]**

(a) Define the *0/1-knapsack* problem.

(b) Provide a small counterexample that proves that the greedy strategy of choosing the item with the highest £/kg ratio is not guaranteed to yield the optimal solution.

(c) Describe a DP algorithm to solve 0/1 knapsack.

(d) What is the time and space complexity of the algorithm?

(e) (optional) Implement the DP algorithm. You can test your implementation on **[GeeksForGeeks 0/1 Knapsack]**.

(f) Explain how you could retrieve an optimal solution.

Exercise 2.C.4 [Matrix Chain Multiplication]

(a) Define the *matrix chain multiplication* problem.

(b) Why can we choose the order of the multiplications?

(c) Why may we want to choose the order of the multiplications? Demonstrate this with an example.

(d) Explain how dynamic programming can be used to solve this problem.

(e) What is the time complexity for this approach?

(f) (optional) Attempt **[LeetCode 312]**.

Exercise 2.C.5 [Longest Common Subsequence]

(a) Define the *longest common subsequence* problem.

(b) Formulate the recurrence relation and explain how dynamic programming helps to solve it.

(c) Show the DP table for input sequences `BDCABA` and `ABCBDAB`.

(d) (optional) Implement the LCS algorithm (either bottom up or top-down). (You may want to submit your solution to **[LeetCode 1143]**)

(e) What is the time complexity of your implementation? How does it compare with the brute force approach?

(f) What is the space complexity of your implementation? How can you reduce this?

(g) Explain how you can recover a longest common subsequence. Draw the corresponding table for the example above. What is the time complexity of this algorithm?

(h) Demonstrate a pair of sequences that have more than one LCSs.

2 Greedy algorithms

Recommended reading/useful links:

- CLRS Chapter 16
- Jeff Erickson's Algorithms Chapter 4

Exercise 2.C.6 [Huffman encoding]

(a) What problem does Huffman encoding solve?

(b) Describe Huffman's encoding algorithm.

(c) Show the steps of the algorithm on Figure 3.1 of the lecture notes.

(d) Describe how you would implement this algorithm. What is the time complexity of your implementation? (*Hint:* You may find it beneficial to use min-heap)

(e) (optional) Implement the Huffman encoding algorithm.

(f) (++) Prove that Huffman's algorithm solves the problem you described in (a).

(g) (optional) How can you implement Huffman's encoding algorithm in $\mathcal{O}(n)$ time if you are given the frequencies in sorted order?