**Advanced Algorithms**

**Exercise for Lecture 11**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name** |  | **Student ID** |  |
| **Problem 1** |  | | |
| **Problem 2** |  | | |
| **Problem 3** |  | | |
| **Total Score** |  | | |
| **Notes** | Deadline: **2023-10-19 24:00**  Submission Format: ‘**Lecture11\_Name\_Student ID.docx**’, and please send to: **[algorithms\_23fall@163.com](mailto:algorithms_23fall@163.com)**.  This assignment is meant to be an evaluation of your **individual** understanding coming into the course and should be completed **without collaboration** or outside help. | | |

**Problem 1. [20 points]** Determine an LCS of  and using the following table.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | 0 |  |  |  |  |  |  |  |  |  |
|  | 0 |  |  |  |  |  |  |  |  |  |
|  | 0 |  |  |  |  |  |  |  |  |  |
|  | 0 |  |  |  |  |  |  |  |  |  |
|  | 0 |  |  |  |  |  |  |  |  |  |
|  | 0 |  |  |  |  |  |  |  |  |  |
|  | 0 |  |  |  |  |  |  |  |  |  |
|  | 0 |  |  |  |  |  |  |  |  |  |

**Solution:**

Solution 1: [10 points]

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | 0 | 1↖ | 1← | 1← | 1← | 1← | 1← | 1↖ | 1← | 1↖ |
|  | 0 | 1↑ | 2↖ | 2↖ | 2↖ | 2← | 2← | 2← | 2↖ | 2← |
|  | 0 | 1↖ | 2↑ | 2↑ | 2↑ | 2↑ | 2↑ | 3↖ | 3← | 3↖ |
|  | 0 | 1↑ | 2↖ | 3↖ | 3↖ | 3← | 3← | 3↑ | 4↖ | 4← |
|  | 0 | 1↑ | 2↑ | 3↑ | 3↑ | 3↑ | 4↖ | 4← | 4↑ | 4↑ |
|  | 0 | 1↑ | 2↑ | 3↑ | 3↑ | 3↑ | 4↖ | 4↑ | 4↑ | 4↑ |
|  | 0 | 1↑ | 2↑ | 3↑ | 3↑ | 4↖ | 4↑ | 4↑ | 4↑ | 4↑ |
|  | 0 | 1↖ | 2↑ | 3↑ | 3↑ | 4↑ | 4↑ | 5↖ | 5← | 5↖ |

Hence, the LCS is . [10 points]

Solution 2: [10 points]

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | 0 | 1↖ | 1← | 1← | 1← | 1← | 1← | 1↖ | 1← | 1↖ |
|  | 0 | 1↑ | 2↖ | 2↖ | 2↖ | 2← | 2← | 2← | 2↖ | 2← |
|  | 0 | 1↖ | 2↑ | 2← | 2← | 2← | 2← | 3↖ | 3← | 3↖ |
|  | 0 | 1↑ | 2↖ | 3↖ | 3↖ | 3← | 3← | 3← | 4↖ | 4← |
|  | 0 | 1↑ | 2↑ | 3↑ | 3← | 3← | 4↖ | 4← | 4← | 4← |
|  | 0 | 1↑ | 2↑ | 3↑ | 3← | 3← | 4↖ | 4← | 4← | 4← |
|  | 0 | 1↑ | 2↑ | 3↑ | 3← | 4↖ | 4← | 4← | 4← | 4← |
|  | 0 | 1↖ | 2↑ | 3↑ | 3← | 4↑ | 4← | 5↖ | 5← | 5↖ |

Hence, the LCS is . [10 points]

**Problem 2. [35 points]** Write an algorithm MAXIMAL-NON-ADJACENT-SEQUENCE-WEIGHT() that, given a sequence of numbers , computes, with worst-case complexity , the maximal weight of any sub-sequence of non-adjacent elements in . A sub-sequence of non-adjacent elements may include or but not both, for all . For example, with , MAXIMAL-NON-ADJACENT-SEQUENCE-WEIGHT() should return .

**Solution:**

We can first start by modeling MAXIMAL-NON-ADJACENT-SEQUENCE-WEIGHT as a classic recursive dynamic programming algorithm. Given a sequence , there are two cases: [10 points]

1. the maximal sequence includes , and therefore does not include and instead includes the maximal sequence for the remaining subsequence ;
2. the maximal sequence does not include and therefore is the same as the maximal sequence for the subsequence starting at .

Thus, the maximal solution is the best of these two. Let denote the maximal weight of non-adjacent elements from a sequence . With this, the algorithm is as follows: [10 points]

Now we just have to write this simple, recursive dynamic programming solution as a single iteration. This can be done by remembering only two values in each iteration, namely the optimal value for the previous two elements in the sequence. We can perform this iteration in either direction, so here we do it in increasing order, left-to-right. Therefore, for each element , we must remember the two previous optimal values and . The full algorithm is as follows:

MAXIMAL-NON-ADJACENT-SEQUENCE-WEIGHT()

1. **for** **to**


5. **return**

This algorithm scans the input sequence once, so the worst-case complexity is . [15 points]

**Problem 3. [45 points]** Santa Claus is packing his sleigh with gifts. His sleigh can hold no more than pounds. He has **different** gifts, and he wants to choose a subset of them to pack in his sleigh. Gift has utility (the amount of happiness gift induces in some child) and weight . We define the weight and utility of a set of gifts as follows:

* The weight of a set of gifts is the **sum** of their weights.
* The utility of a set of gifts is the **product** of their utilities.

For example, if Santa chooses two gifts such that , and , , then the total weight of this set of gifts is pounds and the total utility of this set of gifts is . All numbers mentioned are positive integers and for each gift , . Your job is to devise an algorithm that lets Santa maximize the utility of the set of gifts he packs in his sleigh without exceeding its capacity .

1. Give a recurrence that can be used in a dynamic program to compute the maximum utility of a set of gifts that Santa can pack in his sleigh. Remember to evaluate the base cases for your recurrence.
2. Write pseudo-code for a dynamic program that computes the maximum utility of a set of gifts that Santa can pack in his sleigh. What is the running time of your program?
3. Modify your pseudo-code in part *b* to output the actual set of gifts with maximum utility that Santa packs in his sleigh.

**Solution:**

1. Let be the maximal achievable utility if the gifts are drawn from through (where ) and weigh at most pounds.

For :

If ,

Otherwise (if ), . [10 points]

Base cases: for all integers , . [5 points]

1. We give the following pseudo-code for a dynamic program that takes as input a set of gifts , a maximum weight , and outputs the set of gifts with the maximum utility.

MAXIMUM-UTILITY()

1. **for** **to**
3. **for** **to**
4. **for** **to**
5. **if**
7. **else**
9. **return**

The running time of this algorithm is . [15 points]

1. The following pseudo-code takes as input the table computed in part *b* and variables , and outputs a set of gifts that yield utility . [15 points]

OUTPUT-GIFTS()

1. **if**
2. **output**
3. **elseif**
4. **output**
5. OUTPUT-GIFTS()
6. **elseif**
7. **output** OUTPUT-GIFTS()
8. **elseif**
9. **output**