Informatics II, Spring 2024, Exercise 11

Publication of exercise: May 13, 2024
Publication of solution: May 20, 2024
Exercise classes: May 21 - 24, 2024

Learning Goal

- Learn how to find a recursion relation for DP to reconstruct the solution.
- Gain familiarity with DPs operating on arrays and matrices.
- Learn how to define the subproblems and understand the order of subproblems in DP.

Task 1 [Medium/Hard]

You are given an array of size n. Your task is to find the length of its *longest increasing subsequence*.

A *subsequence* of an array a is a sequence $a[i_1], a[i_2], ..., a[i_k] (k \geq 1)$ where the indexes satisfy the property $1 \leq i_1 < i_2 < ... < i_k \leq n$. You can consider a sequence is derived from the array by deleting some elements without changing the order of the remaining elements. For example, consider the following array a:

1	2	3	4	5	6	7	8
5	10	7	4	8	9	2	10

Figure 1: Array a

For example, valid subsequences are $[5, 7, 10]$, $[10, 7, 2, 10]$, $[9]$, while $[2, 4, 8]$ and $[5, 7, 7, 10]$ are not valid.

An *increasing* subsequence is a subsequence where all the elements are in increasing order, i.e. $a[i_1] \leq a[i_2] \leq ... \leq a[i_k]$. A *longest* increasing subsequence of an array a is an increasing subsequence of the largest length (it has the largest k).

For the array given above, the longest increasing subsequence is $[5, 7, 8, 9, 10]$.

We can solve this problem using dynamic programming:

$$dp_i = \text{the largest length of an increasing subsequence ending in index } i$$

The array dp for the example array above is:

1	2	3	4	5	6	7	8
1	2	2	1	3	4	1	5

Figure 2: Helper array *dp* for array *a*.

a. Given the array *b*

1	2	3	4	5	6	7	8
7	10	4	9	7	10	8	12

Figure 3: Array *b*

fill in its helper array *dp*:

1	2	3	4	5	6	7	8

Figure 4: Helper array *dp* for array *b*.

Hints: Try to find a pattern in how you approach this problem step by step. It often helps to establish an order or "direction" in which you solve the subproblems.

b. Determine the recursive relation of *dp*.

c. Use a bottom-up approach to write the function `int longest_seq(int a[], int n)` which returns the length of the longest increasing subsequence of an array `a` of length `n`.

d. We can also calculate the longest increasing subsequence *itself* (the numbers which make up the subsequence) with the help of a *parent* array p. p_i will store the index of the subproblem of dp_i which gives the optimal solution for dp_i.

For the example of the array a from above

1	2	3	4	5	6	7	8
5	10	7	4	8	9	2	10

Figure 5: Array *a*

and it's helper array *dp*

1	2	3	4	5	6	7	8
1	2	2	1	3	4	1	5

Figure 6: Helper array *dp* for array a.

we have the following parent array p:

1	2	3	4	5	6	7	8
	1	1		3	5		6

Figure 7: Parent array p for array a.

Extend the function `int longest_sequence(int a[], int n)` to also print the elements of the longest increasing subsequence. If there are multiple answers, you can print any.

Task 2 [Medium/Hard]

A path in a matrix M of dimensions $x \times y$ is defined as a sequence of cells $(i_1, j_1), (i_2, j_2), ..., (i_n, j_n)$ in the matrix M $(1 \leq i_k \leq x, 1 \leq j_k \leq y, \forall k \in \{1, 2, ..., n\})$ where each cell (i_{k+1}, j_{k+1}) must be directly below or to the right of the cell (i_k, j_k), but not above or to the left.

An *increasing* path in the matrix M is a path where the values of the cells are in strictly increasing order: $M[i_k][j_k] < M[i_{k+1}][j_{k+1}]$ for all k. The longest increasing path is an increasing path with the maximum number of cells (maximum n).

An example of a matrix M and an increasing path is:

$$\begin{pmatrix} 1 & 7 & 3 & 10 & 6 & 18 \\ 3 & {\color{green}2} & {\color{green}5} & {\color{green}6} & 9 & 16 \\ 6 & 3 & 2 & {\color{green}10} & {\color{green}12} & {\color{green}13} \\ 8 & 7 & 5 & 4 & 8 & {\color{green}15} \end{pmatrix}$$

Figure 8: Example of the longest increasing path problem with the solution of length 7

We can solve this problem using dynamic programming:

$$dp_{i,j} = \text{length of the longest increasing path that ends in cell } (i, j)$$

The *dp* of the matrix M from above is:

$$\begin{pmatrix} 1 & 2 & 1 & 2 & 1 & 2 \\ 2 & 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 1 & 4 & 5 & 6 \\ 4 & 3 & 2 & 1 & 2 & 7 \end{pmatrix}$$

Figure 9: Matrix *dp* of matrix M

a. Write the recursive definition of $dp_{i,j}$.

b. Write the function `int longest_path(int x, int y, int M[x][y])` which returns the length of the longest increasing path in an $x \times y$ matrix M.

c. We extend the definition of a path to allow neighbours from all four directions: from a cell (i_k, j_k) we can now go directly below, above, to the right or to the left. For the matrix above, the longest increasing path now is:

$$\begin{pmatrix} 1 & 7 & 3 & 10 & 6 & 18 \\ 3 & 2 & 5 & 6 & 9 & 16 \\ 6 & 3 & 2 & 10 & 12 & 13 \\ 8 & 7 & 5 & 4 & 8 & 15 \end{pmatrix}$$

Figure 10: New longest path

This new problem can be solved with the same *dp* formulation and a similar recursive definition from above, but the cells have to be visited in a different order.

$$\begin{pmatrix} 1 & 2 & 1 & 4 & 1 & 8 \\ 2 & 1 & 2 & 3 & 4 & 7 \\ 3 & 2 & 1 & 4 & 5 & 6 \\ 4 & 3 & 2 & 1 & 2 & 7 \end{pmatrix}$$

Figure 11: Matrix *dp* showing the new longest path

i. For the given matrix N:

$$\begin{pmatrix} 6 & 5 & 2 & 3 \\ 7 & 14 & 15 & 4 \\ 3 & 12 & 10 & 8 \\ 2 & 7 & 9 & 1 \end{pmatrix}$$

Figure 12: Matrix N

fill in its corresponding *dp* matrix. As a hint, some entries have already been filled in.

$$\begin{pmatrix} 3 & 2 & 1 & 2 \\ & & & 3 \\ 2 & & & \\ 1 & & & 1 \end{pmatrix}$$

Figure 13: *dp* of matrix N

ii. Determine the new recursive definition of $dp_{i,j}$.

iii. Write the function `int longest_path_all_directions(int x, int y, int M[x][y])` which returns the length of the new longest increasing path in an $x \times y$ matrix M.

Task 3 [Medium/Hard]

You are given a set of n items, each with a weight and a value, and a fixed number W. We can represent these weights and values as arrays w and v, where w_i and v_i are the weight and the value of the i^{th} item, respectively.

The task is to determine which items to select such that their total weight is $\leq W$ and the total value is as big as possible.

For example, for $W = 20$ and the following $n = 5$ items:

item	w	v
1	2	3
2	4	5
3	5	8
4	3	4
5	9	10

the largest total value is 26 (choose items 1, 2, 3 and 5).

This problem can be solved using dynamic programming.

a. Define the dynamic programming matrix $dp_{i,j}$ in terms of:

- i - the number of considered items (how many items have we taken into account so far?)
- j - the total weight of the items chosen from those considered items (what is the weight of the items that we chose, out of the items we considered so far?).

What is the purpose of $dp_{i,j}$? What are we trying to maximize?

b. If we know all the entries of $dp_{i,j}$, how do we calculate the final answer for our problem?

c. Write a recursive formulation for $dp_{i,j}$.

d. Write a function `int max_value(int n, int w[], int v[], int W)` which returns the answer for the problem (maximum total value of chosen items).

e. The current solution uses $\Theta(nW)$ memory. Without changing the recursive definition, reduce the space requirements of dp to $\Theta(W)$. Look at the recursive definition to see which subproblems are no longer needed and can be "forgotten".

   Adapt the function `int max_value(int n, int w[], int v[], int W)` to use this memory optimization.

Task 4 [Medium/Hard]

You are given an sequence a of size n: $a_1, a_2, ..., a_n$. We do on this array n erasing operations. An erasing operation remove one of the numbers from the beginning or the end of the sequence. So, for the first operation, we erase either a_1 or a_n. If we are at the i^{th} operation and erase the element a_k, then the cost of that operation is $i \cdot a_k$.

Determine the maximum cost of doing n operations on a sequence a.

Example - for the sequence a below:

1	2	3	4
3	2	4	1

we have the maximum cost of 29, obtained this way:

1. In the first step erase 1 (with cost $1 \cdot 1 = 1$)
2. In the second step erase 3 (cost $2 \cdot 3 = 6$)
3. In the third step erase 2 (cost $3 \cdot 2 = 6$)
4. In the fourth step erase 4 (cost $4 \cdot 4 = 16$)
5. The total cost is $1 + 6 + 6 + 16 = 29$.

a. Because we are always erasing from the beginning or the end of a, at the end of every operation we are left with a contiguous sequence (subarray) of a. In other words, we are left with some subarray $a_i, a_{i+1}, ..., a_j$.

   Define a dynamic programming approach with respect to the subarray $a_i, a_{i+1}, ..., a_j$ that remains after some operations (What defines this subsequence? What are we trying to maximize?).

b. Write the recursive relation of the DP approach. In what order do we have to solve the subproblems? What is the final answer to our problem?

   Hint: The order is similar to the one used for the matrix multiplication problem.

c. Implement a function `max_cost(int n, int a[])` that receives an array a of length n and returns the maximum total cost after doing all n operations.