Informatics II, Spring 2023, Exercise 11

Publication of exercise: May 14, 2023
Publication of solution: May 21, 2023
Exercise classes: May 22 - 26, 2023

Dynamic Programming

Task 1

A path in the 2-dimensional matrix M of dimensions $x \times y$ is defined a sequence of squares (i_1, j_1), (i_2, j_2), ..., (i_n, j_n), where $1 \leq i_k \leq x$ and $1 \leq j_k \leq y$ for all $k = 1, 2, \ldots, n$, and the difference in value between consecutive squares in the path, $|M_{i_k,j_k} - M_{i_{k+1},j_{k+1}}| \leq 1$ for all $k = 1, 2, \ldots, n-1$. The path must also satisfy the constraint that (i_{k+1}, j_{k+1}) can only be either directly below (i_k, j_k) or directly to the right of (i_k, j_k), but not directly above or to the left of (i_k, j_k). The longest possible path within the matrix is the one with the maximum value of n.

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

Table 1: Example of the longest path problem with the solution of length 7

Table 2 shows the helper matrix for the given matrix above. The helper matrix denotes in each field the path with the maximum length up to this point. As you can see the longest path from the example of table 1 starts with a size of 1 and ends with a size of 7.

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

Table 2: Helper matrix to the matrix in table 1

a) Fill in the helper matrix in table 4 for the matrix in table 3. Try to already find a pattern on how you approach this problem step by step. Remember that from any location on the path to the next location you can only go down or to the right.

$$\begin{pmatrix} 4 & 6 & 7 & 8 \\ 3 & 2 & 3 & 8 \\ 8 & 6 & 6 & 7 \\ 9 & 5 & 2 & 5 \\ 8 & 4 & 3 & 2 \end{pmatrix}$$

Table 3: Matrix to task 2. a)

Table 4: Helper table to fill in for task 2. a)

b) Determine recursive definition for helper matrix.

c) Use a bottom up approach to write the function *int longestPath(int x, int y, int M[x][y])* which returns the value of the longest path in matrix M of dimension $x \times y$.

Task 2

The goal of this exercise is to determine the minimum number of cuts to divide a given string S into substrings $(s_1, s_2, s_3, ..., s_n)$ such that each substring is a palindrome. One cut splits one string into two nonempty substrings.
An example for this problem would be the string: "ABACDDC" which can be sliced into two halves: "ABA" and "CDDC" in one cut. Those two substrings are both palindromes, so the solution to this example is 1, since only one cut is required.
Another example is the string "ABBCDC" which can be cut into: "A", "BB" and "CDC" thus requiring 2 cuts in total.
Find the amount of cuts needed for the following strings yourself:

a) "ERTTZUTT"

b) "KLOOOLK"

c) "ZTUOOLAEAL"

d) Write a helper function *int isPalindrome(char X[], int i, int j)* which returns 1 if the string X is a palindrome from index i to (and including) index j, and 0 if X[i]...X[j] is not a palindrome.

e) Write a recursive function *int findMinCutsRecusive(char X[], int i, int j)* where X is a string, i is the first index and j is the index of the last character of the string. The return value should be the minimum amount of cuts required to transform the string into subsections which are all palindromes. Example of a call to this function would be: *findMinCutsRecusive("ABBCDC", 0, 5)* and the output should be 2. Hint: Use the helper function from the previous task.

f) Write the C function *int findMinCuts(char X[], int n)* using dynamic programming, where n denotes the amount of characters the string X contains. A valid call to this function is: *findMinCutsRecusive("ABBCDC", 6)* the output should be 2.

Task 3

The goal of this exercise is to find the largest plus-shaped structure in a binary matrix M, which has dimensions of $x \times y$.
The plus-shaped structure is made from 1s and must be symmetrical both horizontally and vertically. There is a central point and the ones are evenly spaced in the north, south, east, and west direction from the center.
The area of the plus can therefore be expressed mathematically as $A = 4h + 1$, where h represents the length of each of the 4 half-arms of the plus and A is the number of 1s in the plus.

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Table 5: Example biggest plus with $h = 2$ and $A = 9$

To calculate the result we want to make use of 4 helper tables: top, bottom, left and right. Each table should at each entry mark the amount of 1s that are directly (in the example of the top helper matrix) above it. In table 6 you can see an example of what the top helper table for the matrix in table 5 looks like.

a) Fill in the helper tables bottom, left and right in the same fashion as the example given in table 6.

0	0	0	1	0	0
0	1	0	2	0	0
1	2	1	3	1	1
2	3	0	4	0	2
3	0	0	5	1	0

Table 6: Helper table top

Table 7: Helper table bottom

Table 8: Helper table left

Table 9: Helper table right

b) Write a C function *void makeHelper(int x, int y, int M[x][y], int top[x][y], int bottom[x][y], int left[x][y], int right[x][y])* this function should transform the 4 initially empty helper matrices according to the input matrix M with dimension x x y into the helper matrices you created in task 4.a).

c) The goal now is to combine the knowledge of those helper matrices created in subtask a) and b) to find the biggest plus in a binary matrix M. Write the C function *int biggestPlus(int x, int y, int M[x][y])* which returns the value of the area of the biggest plus in the given matrix M.