University of Zurich[UZH]

Department of Informatics
Database Technology Group
Prof. Dr. M. Böhlen

Informatics II
Exercise 11

May 05, 2020

Dynamic programming

Task 1. Imagine several houses built in a circle. Festive illumination needs to be arranged for Christmas and each house has a given number of light bulbs to use for the event. In order to increase the effect of the illuminations, `only houses which are not next to each other (non-neighbor) shall participate`.

Given an array with the number of light bulbs per house, calculate the highest possible number of light bulbs which can be simultaneously lit up in the night. **Example:** Given the array lightBulbs[ ] = $\{17, 5, 10, 18, 22\}$, it would be optimal if the house 0 (with 17 light bulbs) and house 3 (with 18 light bulbs) would participate. Houses 0 and 4 cannot be combined since they are neighbors due to the circular structure.

Solve the illumination problem using (a) recursion, (b) memoization and (c) dynamic programming. Create a C program including the following functions:

- `int maxIlluminationRecursive(int lightBulbs[ ], int n)` which solves the problem recursively.
- `int maxIlluminationMemoized(int lightBulbs[ ], int n, int m[])` which solves the problem using memoization. The array m[n] is used in order to store the intermediate results.
- `int maxIlluminationDynamic(int lightBulbs[ ], int n)` which solves the problem with the help of dynamic programming.

Print the results of your functions for each of the following light bulbs-arrays: `[ 11, 4, 3, 6, 8, 9 ]`, `[ 4, 4, 4, 4, 4 ]` and `[ 13, 15, 31, 21, 9, 12, 44, 32, 12, 43, 22, 9, 11, 32, 26, 22, 21, 3, 4, 29]`.

Task 2. Write in C a function `int lenOfLongestGP(int set[], int n)` that uses dynamic programming to calculate and return **Length of the Longest Geometrix Progression (LLGP)** in a given set of numbers. The common ratio of the Geometric Progression must be an integer. For example in series `[5, 7, 10, 15, 20, 29]`, the LLGP is 3 corresponding to subseries `[5, 10, 20]` with common ratio of 2.

Task 3. Assume a rectangle of size $n \times m$. Determine the minimum number of squares that is prepared to tile the rectangle. The side length of rectangles must be an integer. Figure show as rectangle of 6×5 that requires tiles to tile it with the minimum number of tiles.

The idea for dynamic programming solution is to recursively split the rectangle of size $(n \times m)$ into either two horizontal rectangles $((n \times h)$ and $(n \times (m-h))$ or two vertical rectangles $((v \times m)$ and $((n-v) \times m)$ where $h = 1, 2, \cdots, m-1$ and $v = 1, 2, \cdots, n-1$. Then finding the minimum squares that fill these splits.

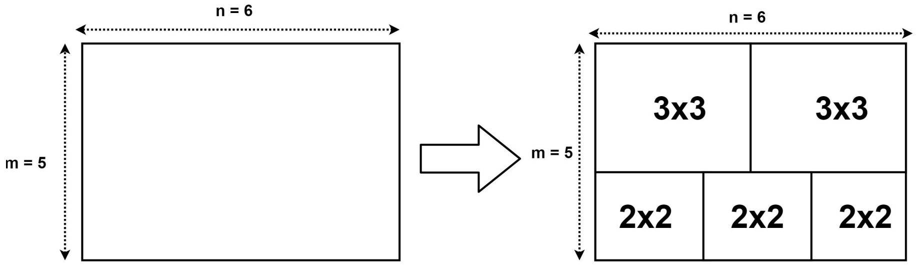

Figure 1: Minimum number of square tiles for a rectangle of size 5×6

- Assume `MinSquare(n, m)` denotes the minimum squares that is required to tile a rectangle of size $n \times m$. State a recursive definition of `MinSquare(n, m)`.
- To efficiently compute the minimum squares that is required to tile a rectangle of $n \times m$, a dynamic programming solution with 2D array dp[$0 \cdots n$][$0 \cdots m$] can be used. Element `dp[i][j]` is the minimum number of squares that is required to tile a rectangle of size $i \times j$. Complete the 2D array below with the minimum squares required to tile rectangles up to size 4×3.

n / m	0	1	2	3	4	
	0					
	1					
	2					
	3					

- For a rectangle of size $n \times m$, write a C program that uses a dynamic programming solution to compute the minimum number of squares that is required to tile the rectangle.