Dynamic Programming, Graph Algorithms Problems and Their Solutions In Code

D. Gueorguiev, 8/21/2024

Table of Contents

[Problems Solvable by Dynamic Programming 1](#_Toc175170705)

[Rod Cutting Problem 1](#_Toc175170706)

[Longest Common Sequence 2](#_Toc175170707)

[0/1 Knapsack problem 2](#_Toc175170708)

[Problems Solvable By Recursive Backtracking 3](#_Toc175170709)

[N Queens problem 3](#_Toc175170710)

[2 x 2 Sudoku Solver 4](#_Toc175170711)

[References 4](#_Toc175170712)

# Problems Solvable by Dynamic Programming

## The N-th Fibonacci Number

## Rod Cutting Problem

Given a rod of length inches (or centimeters) and a table of prices for find the max obtainable revenue by cutting the rod and selling the pieces.

The problem must exhibit optimal substructure if in order to solve the original problem of size we solve similar problems of the same type but smaller sizes.

(1)

Execution time complexity:

(2)

The solution of (2) is given by

(3)

(3) represents exponential time complexity.

Let us assume we are given an integer n and a list p with len(p) >= n. We want to find the max revenue and a partition of n which achieves it.

in code:

def cut\_rod(n: int, p: list[int]):

if n == 0:

## Longest Common Sequence

Let us consider strings in the form where are strings or characters. We say that the tuple forms a sequence in if appears on the left of for each pair such that . Given two strings and find the length of the largest common sequence .

#### Naïve Implementation

def longest\_common\_sequence(s1: str, s2: str, m: int, n: int):

“““

Determine the length of the longest common sequence.

:param s1: str

:param s2: str

:param m: int

:param n: int

“““

if m == 0 or n == 0:

return 0

elif s1[m-1] == s2[n-1]:

return 1 + longest\_common\_sequence(s1, s2, m-1, n-1)

else:

return max(longest\_common\_sequence(s1, s2, m, n-1),

longest\_common\_sequence(s1, s2, m-1, n))

**Excerpt 7**: naïve implementation for the longest common sequence

Question to the candidate: what is the problem with the Naïve implementation?

Answer: awful time complexity which is exponential

Question to candidate: Why the time complexity is exponential, and can it be eliminated?

Answer: It is exponential because the same string fragments are searched multiple times. In this case using recursion by itself alone does not do us favor. Yes, it can be avoided by using memoization.

Details:

Let us have the following two strings “DIMIT” and “DMTI”. Let us find out how the function in Excerpt 7 will execute. For brevity we will denote the function longest\_common\_sequence on Excerpt 7 with .

#### Dynamic Programming implementation

Optimal Subproblem formulation

Initially,

## 0/1 Knapsack problem

Given are items where each item has some weight and profit associated with it. Also, it is given a bag with capacity - that is, the bag can hold at most weight in it. The task is to put such combination of items in the bag so that the profit is maximized. The constraint is that we can put an item in the bag, or we cannot put it at all, it is not possible to put only a fraction of it.

Input: N = 3, W = 4, profit = [1, 3, 4], weight = [4, 5, 1]

Let us denote with

Naïve recursive algorithm:

# A naive recursive implementation

# of 0-1 Knapsack Problem

def knapSack(W, wt, val, n):

# Base Case

if n == 0 or W == 0:

return 0

# If weight of the nth item is

# more than Knapsack of capacity W,

# then this item cannot be included

# in the optimal solution

if (wt[n-1] > W):

return knapSack(W, wt, val, n-1)

# return the maximum of two cases:

# (1) nth item included

# (2) not included

else:

return max(

val[n-1] + knapSack(

W-wt[n-1], wt, val, n-1),

knapSack(W, wt, val, n-1))

1 ) find the subset of elements , each with a weight less than . That is,

2 )

Matrix Chain Multiplication

# Problems Solvable By Recursive Backtracking

## N Queens problem

Place queens on board so that no pair of Queens attack each other.

For example, when N = 4 we have:

.Q..

...Q

Q...

..Q.

//TODO: Finish the section on the 2 x 2 sudoku solver

## 2 x 2 Sudoku Solver

//TODO: Finish the section on the 2 x 2 sudoku solver

# References

[GeeksForGeeks Dynamic Programming Intro](https://www.geeksforgeeks.org/dynamic-programming/)

[Dynamic Programming: Models and Applications, Eric Denardo, 1982, Yale U.](https://github.com/dimitarpg13/graphs_and_dynamic_programming/blob/master/books/dynamic-programming-models-and-application-denardo.pdf)

[Dynamic Programming, Richard Bellman, Princeton, 1957, Sixth Print 1972](https://github.com/dimitarpg13/graphs_and_dynamic_programming/blob/master/books/1957-bellman-dynamicprogramming.pdf)

[Digraphs Theory, Algorithms, and Applications, Bang-Jensen, and Gutin, 2007](https://github.com/dimitarpg13/graphs_and_dynamic_programming/blob/master/books/DigraphsJensenGutin.pdf)