Instituto Politécnico Nacional
Escuela Superior de Cómputo
Evolutionary Computing Dynamic Programming
Student: Rodríguez Coronado Ricardo Professor: Jorge Luis Rosas Trigueros
Realization: 11/feb/20 Delivery date: 18/feb/20

Theoretical framework

Knapsack problem

Given a ser of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less or than or equal to a given limit and the total value is as large as possible.

In the 0/1 KP the number of each item can only be 0 or 1

Changing-making problem

Also know as minimum coin change problem, addresses the question of finding the minimum number of coins (of certain denominations) that add up to a given amount of money

Dynamic Programming

Dynamic programming is both a mathematical optimization method and a computer programming method. The method was developed by Richard Bellman in the 1950s and has found applications in numerous fields, from aerospace engineering to economics.[1]

The method aims to reduce the execution time of an algorithm using overlapped subproblems and optimal substructures.

A problem has optimal substructure if optimal solutions to subproblems can be reduced to construct the solution of the whole problem

A problem with optimal substructure can be solved as follows:

1. Divide the problem in smaller subproblems
2. Find the optimal solutions to the sub problems applying this process recursively
3. Use the solutions found to construct an optimal solution of the original problem

The subproblems are broken into smaller subproblems until the solutions become easy or even trivial

A problem has overlapping subproblems if a subproblem can be used in the solving of more than one upper level problems

In this case, recalculation of common sub problems can be avoided by storing and reusing the calculated solutions.

This is called memorization.

Ricard Bellman proposed the principle of optionality in 1953: Given a sequence of optimal decisions, any subsequence is optimal as well.

Dynamic programming is a sensible choice when subdivision of a problem yields:

- A large mount if subproblems
- Problems with overlapping in this partial solution
- Problems than can be graphed by increasing complexity

Material and equipment:
Python 3.0+
A text editor
PC or Laptop with: Windows 7+/Linux/MacOS X

Practice development
Write Python scripts to solve the following problems using Dynamic Programming:

- 0/1 Knapsack problem
  The first step to solve the problem is to fill the first row with zeros, after using two cycles 'for', one nested in the other, the first will be responsible for the control of the rows, starting from 1 to the number of objects available; ‘i’ index, while the second will handle the columns, starting from 1 to the capacity of the backpack; ‘j’ index.
  Nested to the specific ones, an 'if' is included, in which the condition is that the weight of the object being evaluated is greater than the index 'j', by this way, guarantees that the object is able to enter the backpack, nested the instruction is registered which copies the value of the cell above. Otherwise, the value that the cell will have will be the maximum value between of:
    - The upper cell
    - The value of the upper cell recorded the number of times equal to the weight; of the object in the row to the left added to the value of the same object.

- Changing-Making Problem
  The algorithm begins by filling the first column with zeros. After a similar way to the previous problem, 2 nested ‘for’ cycles are used, for the control of the rows, index ‘i’; starting from 1 to the number of denominations, and the columns, index ‘j’; starting from 1 to the total number. Within these there is a conditional, verifying that the value of the denomination of the current column is viable, this is done by checking that the denomination is more than the value of the ‘j’ index. If is true, the value of the cell higher than the value of the current cell will be copied. Otherwise, the minimum value is chosen from two options:

    - The value of the upper cell
    - The value of the cell, whose column is the current one minus a number equal to the value of the current denomination plus 1

- Levenshtein Distance
  The algorithm begins by adding Epsilon to the first character of the two strings. Then the first column and row are filled with increments of 1 until the size of each word is reached. The next step is two ‘for’ cycles, one controlling the columns; index ‘i’ and the other, nested, handling the rows; ‘j’ index. Within these cycles, we give value to each of the cells, which will be the minimum value between:
    - The value of the left cell minus 1.
    - The value of the upper cell plus 1.
    - The value of the upper left cell plus 1, in case the characters being analyzed; character number ‘i’ for the string 2 and character number ‘j’ for the string 1, are different.
    - The value of the upper left cell, in case the characters being analyzed; character number ‘i’ for string 2 and character number ‘j’ for string 1, are the same.

Conclusions

The nature of the dynamic algorithm is to reduce a problem, so that it can be solved in a redundant way, and then driving the problem more complex. With this base it is possible to obtain an optimal solution; that maybe are not the fastest way, but effective, since it contemplates different non-optimal solutions, this can be used to obtain solutions to more complex problems without analyzing everything from zero.

Bibliography

[1] Dynamic programming. (2020). In Wikipedia. *Retrieved* 11 February 2020 https://en.wikipedia.org/wiki/Dynamic_programming