Instituto Politécnico Nacional

Escuela Superior de Cómputo

Evolutionary Computing

3CM1

1 Dynamic Programing

Santiago Nieves Edgar Augusto

Teacher: Jorge Luis Rosas Trigueros

Date: 01/08/2016

Deadline: 06/08/2016

Theoretical framework:

Dynamic Programming is perhaps the most challenging problem-solving technique among the four paradigms. Dynamic Programming is primarily used to solve optimization problems and counting problems. Dynamic programming algorithms are often used for optimization. A dynamic programming algorithm will examine the previously solved subproblems and will combine their solutions to give the best solution for the given problem. In comparison, a greedy algorithm treats the solution as some sequence of steps and picks the locally optimal choice at each step.

Dynamic programming is both a mathematical optimization method and a computer programming method. In both contexts it refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner. While some decision problems cannot be taken apart this way, decisions that span several points in time do often break apart recursively.

Dynamic programming algorithms are often used for optimization. A dynamic programming algorithm will examine the previously solved subproblems and will combine their solutions to give the best solution for the given problem. In comparison, a greedy algorithm treats the solution as some sequence of steps and picks the locally optimal choice at each step. Using a greedy algorithm does not guarantee an optimal solution, because picking locally optimal choices may result in a bad global solution.

Knapsack problem:

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set 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 than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

Coin change:

Coin Change is the problem of finding the number of ways of making changes for a particular amount of cents, n , using a given set of denominations $d_1 \ldots d_m$. It is a general case of Integer Partition, and can be solved with dynamic programming.

Body:

We have to solve 2 problems, the knapsack and the coin change with the language python, we need to display the table and the solution for each problem, here described the problems and I added the image of the solution in python.

Problem: Given N items, each with its own value V_i and weight W_i, con 1 <= i <= N, and a maximum knapsack size S, compute the maximum value of the items that one can carry, if he can either ignore or take a particular item.

```
aux = map(int, raw_input().split())
n, k = aux[0], aux[1]
weight = map(int, raw_input().split())
benefit = map(int , raw_input().split())

DP = [[0 for j in xrange(k + 1)] for i in xrange(n + 1)]

for i in xrange(n):
    for j in xrange(weight[i]):
        DP[i + 1][j] = DP[i][j]
    for j in xrange(weight[i], k + 1):
        DP[i + 1][j] = max(DP[i][j], DP[i][j - weight[i]] + benefit[i])

print "The table is:"
for i in DP:
    print i
print "The answer is:", DP[n][k]
print "The elements you need to have are: "

elements = []

while n > 0:
    if DP[n][k] != DP[n - 1][k]:
        k -= weight[n - 1];
        elements.append(n)
    n -= 1

elements.reverse()
print elements
```

The code of the solution in python of Knapsack.

```
garo@Garo-PC:~/Documents/Computo Evolutivo$ python mochila.py
5 7
1 3 7 2 1
1 4 2 1 3
The table is:
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 1, 1, 1, 1, 1, 1]
[0, 1, 1, 4, 5, 5, 5, 5]
[0, 1, 1, 4, 5, 5, 5, 5]
[0, 1, 1, 4, 5, 5, 6, 6]
[0, 3, 4, 4, 7, 8, 8, 9]
The answer is: 9
The elements you need to have are:
[1, 2, 4, 5]
```

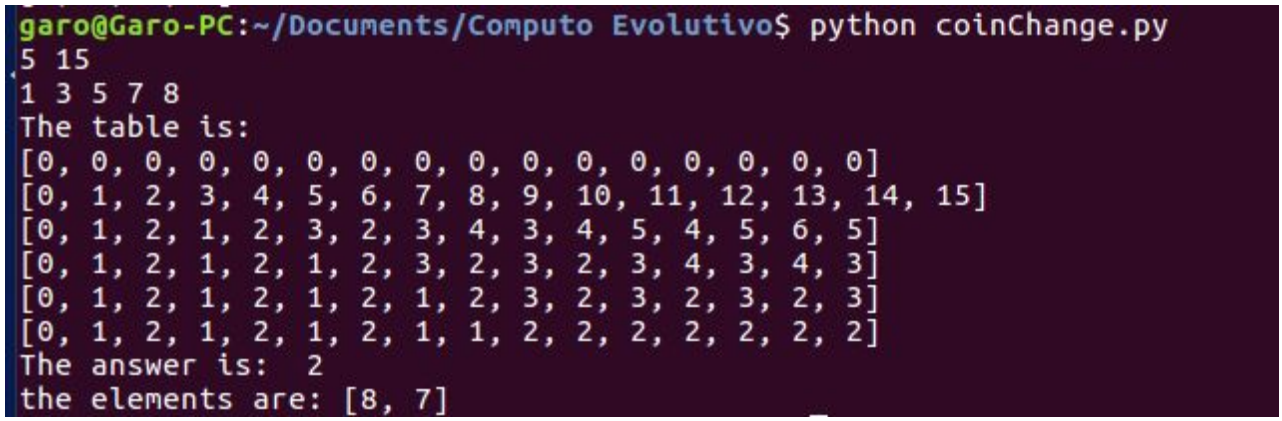
The test in terminal of Knapsack.

Given a value N, if we want to make change for N cents, and we have infinite supply of each of S = { S1, S2, .. , Sm} valued coins, What is the minimum number of coins to have N? The order of coins doesn't matter.

For example, for N = 4 and S = {1,2,3}, there are 2 solutions: {2,2},{1,3}. So output should be 2. For N = 10 and S = {2, 5, 3, 6}, solutions: {5,5}. So the output should be 2.

```
1  aux = map(int, raw_input().split())
2  n, money = aux[0], aux[1]
3  coins = map(int, raw_input().split())
4
5  DP = [[0 for j in xrange(money + 1)] for i in xrange(n + 1)]
6  DP[1] = [i for i in xrange(money + 1)]
7
8  for i in xrange(2, n + 1):
9      for j in xrange(coins[i - 1]):
10         DP[i][j] = DP[i - 1][j]
11
12     for j in xrange(coins[i - 1], money + 1):
13         DP[i][j] = min(DP[i - 1][j], DP[i][j - coins[i - 1]] + 1)
14
15 print "The table is:"
16
17 for i in DP:
18     print i
19
20 print "The answer is: ", DP[n][money]
21
22 elements = []
23
24 while money > 0:
25     if DP[n][money] == DP[n - 1][money]:
26         n -= 1
27     else:
28         elements.append(coins[n - 1])
29         money -= coins[n - 1]
30
31 print "the elements are:", elements
```

The code in python of the solution of Coin Change.



The test in terminal of Coin Change.

Conclusions:

In my opinion in this practice you can see the 2 typical problems in Dynamic Programming, sometimes to understand the process of the solution kind be difficult but if you catch the idea how the program do the solution you can do a lot of solutions with Dynamic Programming and I can see some differences between evolutionary computing and not evolutionary computing, the process of Dynamic Programming is to search all the states that the problem has and choose the best solution of that problem, instead in the evolutionary computing you search a solution with a little information and then you decide what solution can be better you don't need to search all the states possible, I think that the practice let me know the differences between them and what I can do with Dynamic Programming.

Bibliography

Steven Halim, *Competitive Programming 3*, 2013

https://en.wikipedia.org/wiki/Dynamic_programming