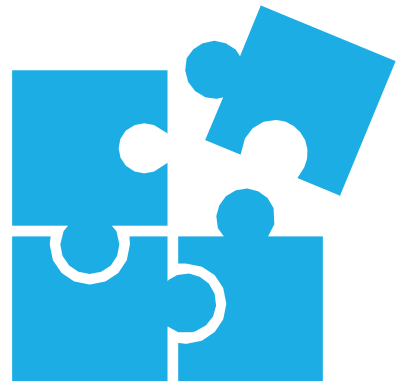




School of
Computer
Science



Dynamic programming

ALGORITHMS

Vicente García Díaz
garciavicente@uniovi.es

The problem of the exchange



- Given a monetary system made up of coins of different values, the **problem of the exchange** is to decompose any given quantity into coins, using the smallest possible number of coins of said monetary system
- Being i the number of different types of coins, j the amount that we want to decompose and $T(1..i)$ a vector with the value of each type of coin in the system
 - We assume that we have enough number of coins of each type
- The final minimum amount of coins must be calculated to return to return the value j

Function to solve the problem



- We have the function **$C(i, j)$** that provides us with the solution to the problem to obtain the quantity **j** restricting itself to the types **T_1, T_2, \dots, T_i**

$$C(i, j) = \begin{cases} \infty & \text{if } (i = 0) \text{ or } (j < 0) \\ 0 & \text{if } (j = 0) \text{ and } (i > 0) \\ \text{Min}(C(i - 1, j), C(i, j - T_i) + 1) & \text{otherwise} \end{cases}$$

Steps to be done



1. Represent the table (unfilled) required to store the intermediate values
 - Knowing that we must return 7 cents with the first five types of coins in the euro system
 - Mark in the table the values we can fill directly
2. Indicate the dependency pattern of any cell, that is, mark the cells that are necessary to calculate a given cell. Also, explain specifically how we would fill cell (1,1)
3. Calculate the table values
4. If we talk about efficiency in runtime, which technique should we choose? dynamic programming? or a greedy algorithm? Reason the answer


Array to store intermediate values



➤ To return 7 cents with the first 5 types of coins

- Coin types $\rightarrow i$
- Amount $\rightarrow j$

Amount (j) \ Coin types (i)	1	2	3	4	5	6	7
1 (1 cent)							
2 (2 cent)							
3 (5 cent)							
4 (10 cent)							
5 (20 cent)							

 $C[i,j]$

Values we can calculate directly



- We have eliminated from the table the values that could be filled directly, therefore all cells must be calculated
- If we include a row 0
 - All the values would be ∞
- If we include a column 0
 - All the values would be 0

Amount Coin Types	1	2	3	4	5	6	7
1 (1 cent)							
2 (2 cent)							
3 (5 cent)							
4 (10 cent)							
5 (20 cent)							

Dependency pattern of any cell



Be careful with values that are can go out of the array

Amount (j) \ Coin types (i)	1	2	3	4	5	6	7
1 (1 cent)							
2 (2 cent)							
3 (5 cent)							
4 (10 cent)							
5 (20 cent)							

Types	1	2	3	4	5
value	1	2	5	10	20

$$C(i,j) = \min(C(i-1,j), C(i, j-T(i))+1)$$

We fill the table following the pattern



Amount (j) Coin types (i)	1	2	3	4	5	6	7
1 (1 cent)	1	2	3	4	5	6	7
2 (2 cent)	1	1	2	2	3	3	4
3 (5 cent)	1	1	2	2	1	2	2
4 (10 cent)	1	1	2	2	1	2	2
5 (20 cent)	1	1	2	2	1	2	2

How to recover the values of the currencies that make up the solution



- Fill cell **origin matrix**
- Origin of the value of cell **i, j**: Do we add a new coin to obtain the value of the cell? -> **TRUE**

Amount (j) Coin types (i)	1	2	3	4	5	6	7
1 (1 cent)	1	2	3	4	5	6	7
2 (2 cent)	1	1	2	2	3	3	4
3 (5 cent)	1	1	2	2	1	2	2
4 (10 cent)	1	1	2	2	1	2	2
5 (20 cent)	1	1	2	2	1	2	2

Amount (j) Coin types (i)	1	2	3	4	5	6	7
1 (1 cent)	T	T	T	T	T	T	T
2 (2 cent)	F	T	T	T	T	T	T
3 (5 cent)	F	F	F	F	T	T	T
4 (10 cent)	F	F	F	F	F	F	F
5 (20 cent)	F	F	F	F	F	F	F

Iterating through the origin matrix



- We go through the **origin matrix** in reverse order, from the solution until there are no more cells
- Depending on the values of the cells:
 - We subtract rows or columns, up to $i \leq 0 \parallel j \leq 0$
- When the value is **TRUE** we write it down

Cantidad (j) Tipos mon (i)	1	2	3	4	5	6	7
1 (1 cent)	T	T	T	T	T	T	T
2 (2 cent)	F	T	T	T	T	T	T
3 (5 cent)	F	F	F	F	T	T	T
4 (10 cent)	F	F	F	F	F	F	F
5 (20 cent)	F	F	F	F	F	F	F

Runtime efficiency



- Efficiency at runtime should not be confused with the possibility that the algorithm returns the optimal solution in all cases
- **Dynamic programming:** Two nested loops that go through each one of the cells of the array. $O(i \cdot j) \rightarrow O(n^2)$
- **Greedy algorithm:** based on the types of coins already sorted $\rightarrow O(n)$. If they are not sorted $\rightarrow O(n \log n)$
 - See the code in the previous topic (the heuristic has no influence on the efficiency of the algorithm)