Introduction to algorithmic Fall 2017

IFT2125-6001 TA: Maëlle Zimmermann

Demonstration 10

1

Question: Give an algorithm to make change with the fewest coins possible siples, assuming that each type of coin exists in unlimited quantities.

Solution: Let $c_1, c_2, ..., c_n > 0$ be the value of the parts available in unlimited quantities. Suppose we want to make money on k units. We are building a painting T of size $n \times (k + 1)$ such that $T[i, j]$ is the minimum number of pieces in order to make the currency on j units using only the first i coins.

Let us first note that $T[i, 0] = 0$ for all $1 \leq i \leq n$ since there is no room to render. Furthermore,

$$T[i, j] = \min \left(\underbrace{T[i - 1, j]}_{\text{do not take part } c_i}, \underbrace{T[i, j - c_i] + 1}_{\text{take part } c_i} \right)$$

assuming that each box outside the table is implicitly $+\infty$. In Indeed, if we do not use the third piece, the number of pieces returned will be identical to the number of pieces needed to make the same amount j using only the $i - 1$ first pieces. If we use the third piece, we will use one more piece than the number of pieces needed to return the remaining amount, of value $j - c_i$. We choose whether or not to use the second piece that both of these options require fewer pieces.

The minimum number of coins needed to make change on k units will be therefore $T[n, k]$. It takes another step to identify the parts to be rendered. To do this, simply start at box $T[n, k]$ of the table and find the path which has been taken since $T[0,0]$. If $T[n, k]$ comes from $T[i-1, j]$, then piece i is never used. If $T[n, k]$ comes from $T[i, j - c_i]$ then the room i was used. This process is repeated iteratively until it reaches $T[0,0]$.

Here is an implementation of this algorithm:

```
def nb_pieces (c, k):
```

```
    T = [[0] * (k + 1) for i in range (len (c))]

    for i in range (len (c)):
        for j in range (1, k + 1):
            a = T [i-1] [j] if i> 0 else float ( "inf" )
            b = T [i] [jc [i]] if j> = c [i] else float ( "inf" )
            T [i] [j] = min (a, b + 1)

    return T

def currency (c, k):
    p = [0] * len (c)
    T = nb_pieces (c, k)
    i, j = len (c) -1, k

    while (i, j)! = (0, 0):
        a = T [i-1] [j] if i> 0 else float ( "inf" )
        b = T [i] [jc [i]] if j> = c [i] else float ( "inf" )

        if T [i] [j] == a:
            i - = 1
        else :
            j - = c [i]
            p [i] + = 1

    return p
```

The exact execution time of money is in θ (nk).

2

Question: Give an algorithm that makes money even when the number of coins available is limited.

Solution: Just create a line for each instance of a room and then use the rule:

$$T[i,j] = \min\Big(\underbrace{T[i-1,j]}_{\text{do not take part } c_i},\ \underbrace{T[i-1,j-p_i]+1}_{\text{take part } c_i}\Big)$$

where $_i$ is the room associated with line i. All boxes are initialized `a + ∞` except from the first columne that is initialized to 0. Here is a possible implementation:

```
def nb_pieces (c, s, k):
    T = [[float ( "inf" )] * (k + 1) for i in range (sum (s) +1)]
```

```
    P = [0] + [p for i in range (len (c)) for p in [c] [i]] * s [i]

    for i in range (len (T)):
        T [i] [0] = 0

    for i in range (1, len (T)):
        for j in range (1, k + 1):
            a = T [i-1] [j] if i> 0 else float ( "inf" )
            b = T [i-1] [jP [i]] if j> = c [i] else float ( "inf" )
            T [i] [j] = min (a, b + 1)

    return T

def currency (c, s, k):
    T = nb_pieces (c, s, k)
    P = [None] + [x for i in range (len (c)) for x in [i] * s [i]
    p = [0] * len (c)
    j = k

    for i in reversed (range (1, len (T)))
        a = T [i-1] [j]
        b = T [i-1] [jP [i]] if j> = P [i] else float ( "inf" )
    if T [i] [j]! = a:
        p [P [i]] + = 1
        j - = c [P [i]]
    return p
```