Introduction à l'algorithmique **Hiver 2018**

IFT2125-6001 TA: Stéphanie Larocque

Démonstration 8

À partir des corrigés de Maelle Zimmermann

1

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

Solution: Let $c_1, c_2, ..., c_n > 0$ the value of the parts available in unlimited quantities. Suppose we want to make change on M units. We build a T array of $n \times (M+1)$ size such that $T[i, j]$ is the minimum number of coins to make change over j units using only the i first coins.

Let us first note that $T[i, 0] = 0$ for all $1 \leq i \leq n$ since there is no coin to return. Furthermore, equation * T [i, j] = min($\underbrace{T[i-1, j]}_{\text{do not take part} c_i}, \underbrace{T[i, j-c_i]+1}_{\text{take piece} c_i}) equation*$

assuming each box outside the table is implicitly $+\infty$. Indeed, if we do not use the i th coin, the number of coins returned will be identical to the number of coins needed to make the same amount j using only the $i-1$ first coins. If we use the i th coin, we will use one more coin than the number of coins needed to make the remaining amount, worth $j - c_i$. We will choose whether or not to use the i th coin according to which of these two options requires the least coins.

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

Here is an implementation of this algorithm:

```python
def nb_pieces(c, M):
  # c = [c1,c2,...,cn] la liste des denominations disponibles
  # M, le montant a retourner
  # retourne le tableau T rempli
  T = [[0]*(M+1) for i in range(len(c))]

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

  return T

def monnaie(c, M):
  # retourne les pieces de monnaie a rendre
  p = [0] * len(c) #nombre de pieces de chaque denomination a rendre
  T = nb_pieces(c, M)
  i, j = len(c)-1, M

  while (i, j) != (0, 0):
    a = T[i-1][j] if i > 0 else float("inf")
    b = T[i][j-c[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 |currency | is in $\Theta(nM)$.

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 part and then use the rule: equation * T [i, j] = min($\underbrace{T[i-1, j]}_{\text{do not take part} c_i}$, $\underbrace{T[i-1, j-p_i] + 1}_{\text{take piece} c_i}$)$equation*$

where p_i is the coin associated with the line i. All cells are initialized to $+\infty$ except for the first column 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][j-c[P[i]]] if j >= c[P[i]] else float("inf")
            T[i][j] = min(a, b+1)

    return T

def monnaie(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][j-c[P[i]]] if j >= c[P[i]] else float("inf")
        if T[i][j] != a:
            p[P[i]] += 1
            j -= c[P[i]]
    return p
```

3

Question: RSA between Alice and Bob

Solution:

Bob :

1. Select 2 ”big” first $p = 19, q = 23$
2. Calculate their product $z = pq = 437$
3. Also calculate $\phi = (p-1)(q-1) = 396$
4. Randomly choose a number $1 \leq n \leq z-1$: $n = 13$
5. Calculates the unique s such that $ns(\mod \phi) = 1$: $s = 61$ (if does not exist or that gcd $(n, z) \neq 1$, then choose a other n as pgcd $(n, z) \neq 1$)
6. Bob sends publicly $z = 437$ and $n = 13$

Alice :

1. Wants to encode the following message $0 \leq m \leq z-1$: $m = 123$
2. Encode with $c = m^n(\mod z)$ thus: $c = 386$
3. Sends the coded message $c = 386$

Bob

1. Get the coded message $c = 386$
2. Decode the message $m = c^s(\mod z)$ thus: $m = 123$

So they managed to exchange a message without a spy discovering it, because finding s would require the factoring of z.