ECOLE CENTRALE LYON : 2015-2016

INFO TC1 : ALGORITHMES ET STRUCTURES DE DONNEES

GROUPE TD : A2a

PROBLEME DE MONNAIE

RAPPORT DU BE3

KONAN Jordan N'guessan Ziahi

NDIAYE Serigne Fallou

RAPPORT DU BE3 : MONNAIE

On dispose d'un système de monnaie S=[0,1,2,5,10,20,50,100,200,500,1000,2000,5000,10000]. Les éléments de S sont les valeurs des pièces de monnaie. A partir d'un nombre de pièces disponibles dans S, il faut trouver un moyen de rendre la monnaie. Toutes les pièces sont considérées comme des centimes.

On veut connaitre les éléments de S qui permettent de déterminer M (M est une somme d'argent qui est connue).

1) TECHNIQUE GLOUTONNE

Grâce à cette fonction, nous pouvons déterminer pour une somme donnée, les différentes pièces (élément de S) qui vont permettre de la reconstituer. Notons Q la quantité de pièces utilisées.

```
def monnaie_gloutonne(S,M):
    Mr=M # Monnaie restante
    T=[0]*len(S) # création d'un vecteur de même dimension que S
    for i in range(len(S)-1,-1,-1):
        if S[i]<=Mr and S[i]!=0: #recherche de la valeur la plus grande de S proche de M
            Ti=Mr//S[i]
            T[i]=Ti
            Mr= Mr%S[i]
        else:
            T[i]=0
            if Mr==0:
                break
    return sum(T),T
```

- L'algorithme principal qui utilise la fonction **monnaie_gloutonne** est le suivant :

```
if __name__=="__main__":
    S=[0,1,2,5,10,20,50,100,200,500,1000,2000,5000,10000]
    M=126
    print("Données du problème")
    print("S:",S)
    print("M=",M)
    print("----------------- Gloutonne ----------------")
    print("Q:",monnaie_gloutonne(S,M)[0])
    print("T:",monnaie_gloutonne(S,M)[1])
```

- L'exécution de l'algorithme principal nous donne le résultat suivant :

```
Données du problème
S: [0, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000]
M= 126
----------------- Gloutonne ----------------
Q: 4
T: [0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0]
----------------- Programmation Dynamique ---------------
le nombre de pièces nécessaire:  4
```

Nous obtenons bien le résultat attendu mais rien ne nous garantit qu'il utilise le minimum de pièces possibles dans S.

2) PROGARMMATION DYNAMIQUE

Dans cette partie il s'agit de minimiser le nombre de pièces utilisées pour rendre la monnaie. C'est donc un problème d'optimisation de la quantité de pièces qui sera utilisée pour rendre la monnaie.

La fonction qui traite de ce problème est la suivante :

```
def monnaie_optimale(S,M):
    mat=numpy.zeros([len(S),M+1])
    for i in range(0,len(S)):
        for m in range(0,M+1):
            if m==0:
                mat[i][m]=0
            elif i==0:
                mat[i][m]=float('Infinity')
            else:
                if m-S[i]>=0:
                 val1=1+mat[i][m-S[i]]
                else:
                 val1=float('Infinity')
                if i>=1:
                 val2=mat[i-1][m]
                else:
                 val2=float('Infinity')
                mat[i][m]=min(val1,val2)
    return int(mat[len(S)-1][M])
```

- L'algorithme principal nous donne avec la fonction **monnaie_optimale**

```
if __name__=="__main__":
    S=[0,1,2,5,10,20,50,100,200,500,1000,2000,5000,10000]
    M=756
    print("Données du problème")
    print("S:",S)
    print("M=",M)
    print("----------------- Programmation Dynamique ---------------")
    print("le nombre de pièces nécessaire: ",monnaie_optimale(S,M))
```

- L'exécution du programme principale nous donne comme résultat :

```
Données du problème
S: [0, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000]
M= 756
----------------- Programmation Dynamique ---------------
le nombre de pièces nécessaire:  5
```

3) GLOUTON VS DYNAMIQUE

Pour mieux percevoir la différence entre les fonctions **monnaie_gloutonne** et **monnaie_optimale** nous allons exécuter le programme principal avec un nouveau système de monnaie S=[0,1,4,6].

```
if __name__=="__main__":
    S=[0,1,4,6]
    M=8
    print("Données du problème")
    print("S:",S)
    print("M=",M)
    print("----------------- Gloutonne ----------------")
    print("Q:",monnaie_gloutonne(S,M)[0])
    print("T:",monnaie_gloutonne(S,M)[1])
    print("----------------- Programmation Dynamique ---------------")
    print("le nombre de pièces nécessaire: ",monnaie_optimale(S,M))
```

- Exécution du programme principal

```
Données du problème
S: [0, 1, 4, 6]
M= 8
----------------- Gloutonne ----------------
Q: 3
T: [0, 2, 0, 1]
----------------- Programmation Dynamique ---------------
le nombre de pièces nécessaire:  2
```

Le résultat de l'exécution nous montre bien la différence entre les deux fonctions.

On peut donc conclure que la fonction **monnaie_optimale** minimise le nombre de pièces à rendre.