Algorithmique
TD n° 11 : Analyse amortie et révision programmation dynamique

Exercice 1 : des piles et des files

Dans cet exercice, on considère des piles (structures LIFO) munies des opérations de base classiques : Push(P,x) pour ajouter un élément x dans la pile, Pop(P) pour retirer l'élément du dessus et le renvoyer comme résultat de la fonction, et IsEmpty(P) qui renvoie Vrai si et seulement si la pile est vide. On suppose que toutes ces opérations sur les piles se font en temps constant.
On va implémenter des files (structures FIFO) avec des piles : une file F sera basée sur deux piles, dénotées F.Pin et F.Pout. On construit les fonctions Ajouter, Extraire et EstVide de la manière suivante :

```
def Ajouter(F,x) :
    Push(F.Pin,x)

def Extraire(F) :
    if not IsEmpty(F.Pout) then :
            return Pop(F.Pout)
    else if not IsEmpty(F.Pin) then :
            while not IsEmpty(F.Pin) :
                  Push(F.Pout,Pop(F.Pin))
            return Pop(F.Pout)
    else return erreur

def EstVide(F) :
     return IsEmpty(F.Pin) and IsEmpty(F.Pout)
```

1. Appliquer l'algorithme à la séquence suivante d'instructions (on indiquera à chaque étape le contenu des deux piles) appliquées sur une file F initialement vide : Ajouter(F,2), Ajouter(F,5), Ajouter(F,8), Ajouter(F,6), Extraire(F), Ajouter(F,4), Ajouter(F,5), Extraire(F), Extraire(F), Extraire(F), Extraire(F), Extraire(F).
2. Quelle est la complexité des opérations sur ces files ?
3. Calculer la complexité *amortie* des opérations sur ces files en considérant une suite de n opérations Ajouter(F,-) / Extraire(F) (à partir d'une file vide). On utilisera une des trois méthodes vues en cours (agrégation, comptable, ou potentiel).

Exercice 2 :

Une entreprise est engagée sur plusieurs projets et doit décider d'affecter des ressources (des journées de travail) à chacun de manière à maximiser les bénéfices. Pour cela, elle dispose d'un tableau G à deux dimensions tel que $G[j,p]$ représente le **gain** obtenu par j journées de travail pour le projet numéro p. Dans la suite, on suppose qu'il y n projets numérotés 1,2,...,n et donc que l'on a $1 \leqslant p \leqslant n$. Le nombre de journées est compris entre 0 et un maximum t, on a donc $0 \leqslant j \leqslant t$, on suppose alors que consacrer plus de t journées à un projet ne change pas le bénéfice. On suppose aussi que $G[0,p]$ est nul pour tout p (tout gain nécessite au moins un peu de travail !).

Par exemple, on peut prendre le tableau G suivant où $n = 3$ et $t = 4$:

Gain	projet 1	projet 2	projet 3
0 jour	0	0	0
1 jour	1	1	1
2 jours	1	2	3
3 jours	2	2	3
4 jours	6	4	5

Dans cet exemple, on voit que si l'entreprise ne dispose que d'une journée de travail, tous les projets offrent le même gain (1), mais avec 2 jours le projet 3 est plus intéressant (le gain est de 3) et avec 3 jours on peut obtenir un gain de 4 en affectant 2 jours au projet 3 et 1 journée au projet 1 (ou au projet 2). Pour 6 jours, le mieux est de consacrer 4 jours au projet 1 (gain de 6) et 2 jours au projet 3 (gain de 3), cela donne un gain total de 9.

L'objectif de ce problème est donc de chercher des algorithmes qui étant donné un tableau $G[j,p]$ avec $0 \leqslant j \leqslant t$ et $1 \leqslant p \leqslant n$ et une durée totale D, retournent une manière d'affecter D jours à l'ensemble des projets pour maximiser le gain total. Le résultat de l'algorithme est donc un tableau d'entiers $TA[-]$ de taille n donnant le nombre de jours affectés à chaque projet et tel que $\sum_{i=1\ldots n} TA[i] = D$. Le gain total associé à TA est donc $\sum_{i=1\ldots n} G[TA[i], i]$, c'est cette somme que l'on veut maximiser.

1. Dans l'exemple ci-dessus, quel est le meilleur choix pour TA lorsque D vaut 9 ? Et 10 ? Et 11 ?
2. Est-ce que l'algorithme glouton ci-dessous est optimal ? Justifier votre réponse.

```
Pour i=1..n :
    A[i] := 0
Pour j=1..D:
    Choisir un p tq G[A[p]+1,p]-G[A[p],p] soit maximal
    A[p]:=A[p]+1
retourner A
```

3. Maintenant on note $GM[j,p]$ le gain maximum que l'on peut obtenir en j jours avec les projets $1,\ldots,p$ (avec $0 \leqslant j \leqslant D$ et $1 \leqslant p \leqslant n$). Le gain total pour D jours et les n projets est donc $GM[D,n]$.
   (a) Construire le tableau $GM[-,-]$ pour les trois projets de l'exemple ci-dessus et $D = 6$.
   (b) Exprimer $GM[j,p]$ en fonction de $GM[j',p-1]$ avec $0 \leqslant j' \leqslant j$ et $G[-,-]$
   (c) En déduire un algorithme de programmation dynamique pour calculer $GM[D,n]$. Donner sa complexité.
   (d) Donner un algorithme `Solution` qui retourne un tableau TA correspondant au gain $GM[D,n]$ (pour cet algorithme on dispose du tableau $GM[-,-]$). Donner sa complexité.

Exercice 3 : la vache et la barrière

On a une vache qui cherche à sortir d' un pré avec une clôture comportant une ouverture, mais la vache ne sait pas où est l'ouverture. Le pré est un demi-plan, la vache se trouve au point origine $A = (0,0)$, la clôture est la droite d'équation $y = 1$, et l'ouverture est un point sur cette droite. La vache est myope : elle ne peut voir l'ouverture que si elle est juste dessus. Elle va donc commencer par aller jusqu'à la clôture, au point $(0,1)$, mais alors se pose une question : de quel côté est l'ouverture ? Faut-il aller à droite ou à gauche ? La vache souhaite passer le moins de temps possible à chercher l'ouverture. Elle marche à vitesse constante de 1 mètre par seconde.
Donner un algorithme pour la vache lui garantissant que quelle que soit la position B de l'ouverture, elle la trouvera en temps $O(dist(A,B))$.

Exercice 4 :

On suppose qu'on a une suite d'opérations. La i-ième opération a coût $c(i)$. Dans chacun des cas suivants, est-ce que le coût amorti par opération est constant ?

1. $c(i) = i$ si i est une puissance de 2, et $c(i) = 1$ sinon.
2. $c(i)$ est la plus grande puissance de 2 qui divise i.
3. $c(i)$ est le plus grand k tel que 2^k divise i.