L3-INFO Algorithmique

Quick 1 – 8 octobre 2021 – durée 1 h

Les documents et les téléphones (incluant smartphone, tablettes,... tout ce qui contient une interface réseau) interdits. Les calculettes sont autorisées.
Seuls les dictionnaires pour les personnes de langue étrangère sont autorisés.
Le barème est indicatif.

Vous êtes vivement encouragés à illustrer vos recherches et vos réponses par des schémas.

Question 1 : Structure de données (5 points)

On rappelle la spécification du type $Pile$:

Opérations

PileVide	:	$void \to Pile$
Empiler	:	$Element \times Pile \to Pile$
EstVide	:	$Pile \to bool$
Dépiler	:	$Pile \to Pile$
Sommet	:	$Pile \to Element$

Préconditions

Dépiler(p)	:	p est non vide
Sommet(p)	:	p est non vide

Axiomes

EstVide(PileVide())	=	vrai
EstVide(Empiler(e, p))	=	faux
Dépiler(Empiler(e, p))	=	p
Sommet(Empiler(e, p))	=	e

1. *(3 pts)* Écrire un algorithme qui prend en argument une pile (non vide) et renvoie une pile constituée des mêmes éléments dans le même ordre, excepté l'un des éléments médians.

   Un élément de la pile est **médian** s'il y a autant d'éléments en-dessous de lui qu'au-dessus de lui, à une unité près quand le nombre d'éléments est pair.

   Exemples :
   - Si la pile donnée en entrée contient dans l'ordre A, B, C, D, E (le sommet est A) on renverra une pile contenant A, B, D, E
   - Si la pile donnée en entrée contient (dans l'ordre) A, B, C, D, E, F (le sommet est toujours A) on renverra une pile contenant A, B, D, E, F ou bien A, B, C, E, F.

   Il est bien entendu souhaitable que votre algorithme soit le plus efficace possible.

2. *(2 pts)* Évaluer précisément le coût de votre algorithme en nombre d'opérations Dépiler et Empiler, en fonction de la taille initiale n de la pile.

Réponse 1 :

1. Un des éléments déterminants ici est de compter combien la pile contient d'éléments, ce qui demande de la parcourir entièrement. On peut réaliser cette opération à part (ce qui est plus coûteux) ou bien la combiner avec la tâche demandée.
   Par ailleurs il faut stocker les éléments temporairement extraits de la pile ; on peut utiliser une pile auxiliaire, ou bien la récursivité (le coût en temps est le même et le coût en mémoire est similaire).
   Dans le cas d'un algorithme récursif il faut un paramètre supplémentaire et une valeur de retour pour compter les éléments et reconnaître la position de l'élément médian.

```
MEDIAN_REC(P, n = 0)
  Données : Une pile P et un accumulateur entier n initialisé à 0 lors du premier appel
  Résultat : Une pile constituée des mêmes éléments que P sans l'élément médian,
  et le nombre d'éléments dans cette pile
si estVide(P) alors
    Renvoyer (P, 0)
sinon
    (P', m) := MEDIAN_REC(Depiler(P), n + 1)
    si m = n ou m = n + 1 alors
        renvoyer (P', m + 1)
    sinon
        renvoyer (Empiler(Sommet(P), P'), m + 1)
```

```
MEDIAN_ITER(P)
  Données : Une pile P
  Résultat : Une pile constituée des mêmes éléments que P sans l'élément médian
n := 0
P' := PileVide()
tant que non estVide(P) faire
    P' := Empiler(Sommet(P), P')
    P := Depiler(P)
    n := n + 1
tant que non estVide(P') faire
    si n ≠ 0 et n ≠ 1 alors
        P := Empiler(Sommet(P'), P)
    P' := Depiler(P')
    n := n − 2
renvoyer P
```

2. Si on s'y prend bien, on n'effectue que n Dépiler et $n-1$ Empiler.
   Les versions naïves en demandent $3n/2$, voire encore plus si on utilise une pile explicite.

Question 2 : Écriture et preuve d'algorithme (15 points)

Soit un tableau T (indexé de 0 à $n-1$) dont les éléments sont des nombres entiers, et une valeur x. L'objet de ce problème est de déterminer un couple d'éléments dans T dont la somme est égale à x.

Par exemple, pour $T = [5, 11, 20, 7, 14, 15]$ et $x = 26$, la solution (ici unique) est le couple $11 + 15 = 26$.

1. *(2 pts)* Écrire un algorithme résolvant ce problème pour un tableau T quelconque.
2. *(2 pts)* Déterminer la complexité de votre algorithme en fonction du nombre n d'éléments dans T.
3. On suppose désormais que le tableau T est **trié** en ordre croissant.

   On propose alors l'algorithme suivant :

```
COUPLE_SOMME(T, x)
  Données : Un tableau T de n entiers trié et un entier x
  Résultat : Un couple (t1, t2) d'éléments de T tels que t1 + t2 = x s'il existe,
  ou la valeur spéciale ECHEC sinon
1 g := 0
2 d := n − 1
3 tant que g < d et T[g] + T[d] ≠ x faire
4     si T[g] + T[d] < x alors
5         g := g + 1
      sinon
6         d := d − 1
7 si g < d alors
8     renvoyer (T[g], T[d])
  sinon
9     renvoyer ECHEC
```

   (1 pt) Illustrer l'exécution de cet algorithme sur un exemple bien choisi.

4. *(2 pts)* Justifier que cet algorithme se termine et calculer sa complexité.
5. Il est clair que cet algorithme répond correctement lorsqu'il renvoie un couple de valeurs. On s'intéresse donc plutôt à démontrer que lorsque le résultat est $ECHEC$, un tel couple n'existe pas.

   (1 pt) Formuler précisément la postcondition correspondante.
6. Pour démontrer cette postcondition, on propose d'étudier l'invariant suivant : *« Pour tout $i < g$, on a $T[i] + T[d] < x$ »*.

   (a) *(1 pt)* Justifier que cette propriété est vérifiée après la ligne 2 de l'algorithme.

   (b) *(2 pts)* Démontrer qu'il s'agit effectivement d'un invariant de boucle.

   (c) *(2 pts)* Que peut-on déduire de cet invariant en sortie de boucle, dans le cas $ECHEC$?

   Attention, il ne s'agit que d'une partie de la postcondition recherchée.
7. *(1 pt)* Formuler (sans le démontrer) un invariant similaire à propos de la partie droite du tableau.
8. *(1 pt)* Que resterait-il à démontrer pour justifier complètement que l'algorithme est correct ?

Réponse 2 :

1.

```
SOMME_NAIF(T, x)
  Données : Un tableau T de n entiers quelconque et un entier x
  Résultat : Un couple (t1, t2) d'éléments de T tels que t1 + t2 = x s'il existe,
  ou la valeur spéciale ECHEC
1 pour i de 0 à n − 1 faire
2     pour j de i + 1 à n − 1 faire
3         si T[i] + T[j] = x alors
4             renvoyer (T[i], T[j])
5 renvoyer ECHEC
```

2. $\sum_{i=0}^{n-1}(n-1-i) = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$ soit $\mathcal{O}(n^2)$
3. Prenons par exemple $T = [5, 7, 11, 14, 15, 20]$ et $x = 21$.
   On a successivement :
   — $g = 0$ et $d = 5$: alors $5 + 20 > 21$, on décrémente d (dans l'espoir de diminuer la somme).
   — $g = 0$ et $d = 4$: alors $5 + 15 < 21$, on incrémente g (dans l'espoir d'augmenter la somme).
   — $g = 1$ et $d = 4$: alors $7 + 15 > 21$, on décrémente d.
   — $g = 1$ et $d = 3$: alors $7 + 14 = 21$, la somme voulue est trouvée.
   Si on avait pris $x = 23$ on aurait abouti à $g = 2$ et $d = 3$ puis $d = 2 = g$ qui dénote un échec.
4. Le coût d'une itération est constant (3 comparaisons et 3 sommes).
   Un variant de boucle est $d - g$ qui diminue de 1 exactement à chaque itération. Comme initialement $d - g = n - 1$ et qu'on s'arrête lorsque $d - g = 0$ ou avant, le coût au pire de l'algorithme est linéaire.
5. Cette postcondition est « Si l'algorithme renvoie $ECHEC$ alors pour tout couple i, j (distincts) entre 0 et $n - 1$ on a $T[i] + T[j] \neq x$ ».
   On peut se contenter des couples $i < j$ sans perte de généralité.
6. (a) À la ligne 2 $g = 0$ donc cette propriété ne concerne aucun i, elle est trivialement vérifiée.
   (b) Supposons qu'en début d'itération, pour tout $i < g$, on a $T[i] + T[d] < x$.
   Deux cas sont à distinguer :
   — Si $T[g] + T[d] < x$ alors on incrémente g. Il faut démontrer que la propriété est vraie pour tout $i < g'$ soit pour tout $i < g + 1$. Comme elle est supposée vraie pour tout $i < g$, il ne « manque » que le cas $i = g$, autrement dit $T[g] + T[d] < x$ ce qui est bien vérifié.
   — Sinon on décrémente d. Alors pour tout $i < g$, on a $T[i] + T[d'] \leqslant T[i] + T[d] < x$ par hypothèse d'ordre du tableau ($T[d'] = T[d-1] \leqslant T[d]$).
   (c) En sortie de boucle, dans le cas $ECHEC$, on a $g = d$ donc : pour tout $i < d$, on a $T[i] + T[d] < x$. A fortiori, pour tout $j < d$, on aura $T[j] \leqslant T[d]$, donc $T[i] + T[j] < x$.
   Autrement dit, aucun couple (i, j) à gauche de $g = d$ ne convient.

7. *« Pour tout $j > d$, on a $T[g] + T[j] > x$ ».*
   Ceci permettra de même de démontrer qu'aucun couple (i, j) à droite de $g = d$ ne convient.
8. Il resterait à démontrer qu'en fin d'algorithme aucun couple (i, j) avec $i < g = d < j$ ne convient.
   Un troisième invariant (non demandé dans le devoir) :
   « Pour tout $i < g$, pour tout $j > d$, on a $T[i] + T[j] \neq x$ »
   qu'on prouve à l'aide des deux précédents, permet de terminer cette démonstration.