Aix*Marseille université

PORTAIL ET L1EN2ANS DESCARTES

Introduction à l'informatique : Partiel

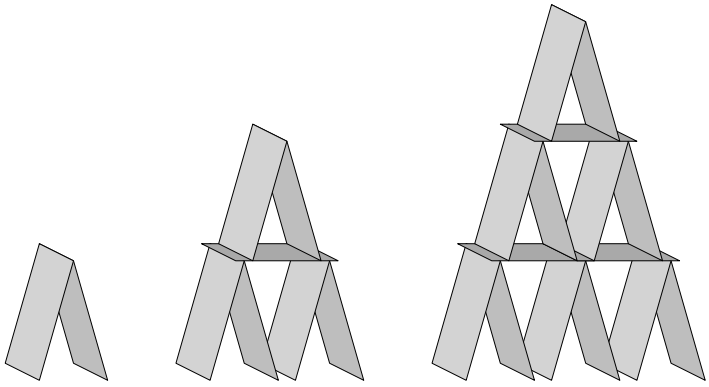
Code UE : SPO1U65

Année 2023-24

Documents et calculatrice interdits

Tous les exercices sont indépendants et peuvent donc être traités dans n'importe quel ordre. Vous pouvez passer une question vous semblant trop difficile, quitte à y revenir ensuite. Par ailleurs, toute réponse non correctement rédigée (avec explication précise et concise) ou non justifiée sera considérée comme fausse.

Exercice 1 (Algorithme informel) Un château de cartes est une structure où des niveaux de paires de cartes sont placés en équilibre en triangle, surmontés par des cartes placées horizontalement :



château à 1 niveau — château à 2 niveaux — château à 3 niveaux

1. Supposons que le château à n niveaux (du niveau 1 au niveau n) soit construit devant vous, pour une valeur quelconque fixée de $n \geq 1$. Expliquez en français comment construire le château de cartes à $n+1$ niveaux en ajoutant des cartes au château devant vous. On supposera avoir toujours à disposition assez de cartes.
2. Soit $c(n)$ le nombre de cartes nécessaires pour construire le château à n niveaux. Par exemple, en observant l'illustration en haut, on peut vérifier que $c(1) = 2$, $c(2) = 7$ et $c(3) = 15$. Exprimez $c(n+1)$ en fonction de $c(n)$, pour tout $n \geq 1$.
3. En justifiant votre réponse mathématiquement, combien de niveaux le château construit avec 40 cartes a-t-il ?

Exercice 2 (Algorithme mystère) On considère dans cet exercice l'algorithme suivant :

```
1  def mystère(t):
2      n = len(t)
3      r = 0
4      for i in range(n):
5          j = i
6          while j < n:
7              if i % 2 == 0:
8                  r = r + t[j]
9              else:
10                 r = r - t[j]
11             j = j + 1
12     return r
```

1. Déroulez l'exécution de cet algorithme avec comme entrée le tableau `t = [2, 1, 3, 6]`. Pour ce faire, vous présenterez l'exécution pas à pas sous la forme d'un tableau d'en-tête :

t	n	r	i	j	t[j]
[2, 1, 3, 6]	4	0	0	0	2
⋮	⋮	⋮	⋮	⋮	⋮

La première ligne vous est donnée et fournit les valeurs de `i`, `j`, `r` et `t[j]` suite à leur initialisation (lignes 3, 4 et 5).

2. Montrez que l'appel de `mystère(t)` termine pour tout tableau `t` donné en entrée.

3. Dénombrez les instructions exécutées par l'appel de `mystère(t)`, avec `t` un tableau de longueur $\texttt{n} \geq 1$. Évaluez la complexité asymptotique de cette fonction en l'exprimant avec la notation « grand O ».
Indication : On appelle « instructions » l'ensemble des lignes d'un algorithme correctement écrit (respectant les conventions de base). On pourra par exemple commencer par calculer le nombre de fois que la ligne 11 est exécutée.

4. Que la fonction `mystère` calcule-t-elle ?

Exercice 3 (Conception d'algorithmes)

1. On a vu en cours et en TD différents tris de tableau. Un autre tri est le *tri par sélection*. L'idée est de trier le tableau en commençant par en extraire l'élément minimum, qu'on place en tête du tableau. On extrait ensuite l'élément minimum du tableau privé de son premier élément, qu'on place à nouveau en tête du tableau, et ainsi de suite jusqu'à la fin. Pour réaliser un tel tri, il faut pouvoir calculer le minimum dans une portion de tableau.

   Proposez une fonction Python `minimum_fin(t, i)` qui prend en argument un tableau `t` de longueur `n` et l'indice `i` d'une case du tableau et qui renvoie un indice j tel que `t[j]` est le plus petit entier parmi les éléments `t[i]`, `t[i+1]`, ..., `t[n-1]`. Par exemple, l'appel de `minimum_fin([1, 8, 5, 6], 1)` renvoie l'indice 2 puisque `5` est l'élément minimal parmi les éléments `8`, `5` et `6`.

2. Pour un entier naturel n, on appelle *racine carrée entière* de n le plus grand entier s tel que $s^2 \leq n$. En particulier, si n est le carré d'un entier, s est égal à $\sqrt{n}$. Proposez une fonction de `racine_carree_entiere` prenant une unique entrée qui est un entier n et qui renvoie -1 si n est strictement négatif, ou la racine carrée entière de n sinon. Par exemple, `racine_carree_entiere(4)` renvoie 2, `racine_carree_entiere(10)` renvoie 3 et `racine_carree_entiere(-50)` renvoie -1.
Indication : Il est interdit d'utiliser la fonction `sqrt` *de la bibliothèque* `math` *de Python.*

Aix*Marseille université

PORTAIL ET L1EN2ANS DESCARTES

Introduction à l'informatique : Partiel

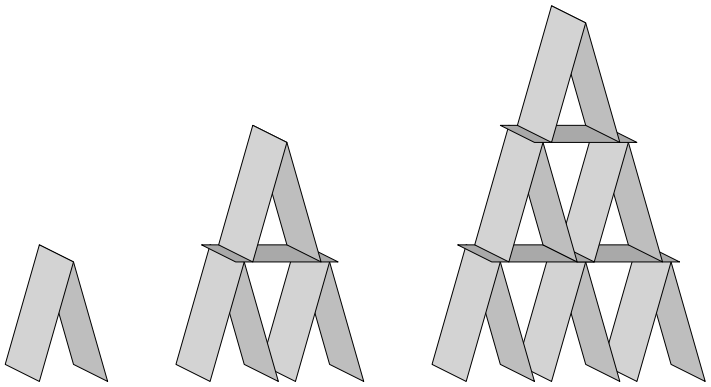
Code UE : SPO1U65

Année 2023-24

Documents et calculatrice interdits

Tous les exercices sont indépendants et peuvent donc être traités dans n'importe quel ordre. Vous pouvez passer une question vous semblant trop difficile, quitte à y revenir ensuite. Par ailleurs, toute réponse non correctement rédigée (avec explication précise et concise) ou non justifiée sera considérée comme fausse.

Exercice 1 (Algorithme informel) Un château de cartes est une structure où des niveaux de paires de cartes sont placés en équilibre en triangle, surmontés par des cartes placées horizontalement :



château à 1 niveau — château à 2 niveaux — château à 3 niveaux

1. Supposons que le château à n niveaux (du niveau 1 au niveau n) soit construit devant vous, pour une valeur quelconque fixée de $n \geq 1$. Expliquez en français comment construire le château de cartes à $n+1$ niveaux en ajoutant des cartes au château devant vous. On supposera avoir toujours à disposition assez de cartes.

   Solution : Il faut ajouter une structure triangulaire au niveau 1 (à droite par exemple) surmontée d'une carte horizontale ; ensuite, on répète la même opération au niveaux 2, 3, ..., n. Enfin, on ajoute une dernière structure triangulaire au dessus du $n+1$-ième niveau, ce qui donne le résultat attendu.

2. Soit $c(n)$ le nombre de cartes nécessaires pour construire le château à n niveaux. Par exemple, en observant l'illustration en haut, on peut vérifier que $c(1) = 2$, $c(2) = 7$ et $c(3) = 15$. Exprimez $c(n+1)$ en fonction de $c(n)$, pour tout $n \geq 1$.

   Solution : On ajoute aux $c(n)$ cartes du château à n niveaux 3 cartes par niveau existant (une structure triangulaire de deux cartes, plus une carte horizontale) et, enfin, on ajoute une structure triangulaire de plus au sommet. Cela donne $c(n+1) = c(n) + 3n + 2$.

3. En justifiant votre réponse mathématiquement, combien de niveaux le château construit avec 40 cartes a-t-il ?

   Solution : Il a 5 niveaux, puisque

$$\begin{aligned} c(3) &= 15 \\ c(4) &= c(3) + 3 \cdot 3 + 2 = 15 + 9 + 2 = 26 \\ c(5) &= c(4) + 3 \cdot 4 + 2 = 26 + 12 + 2 = 40. \end{aligned}$$

Exercice 2 (Algorithme mystère) On considère dans cet exercice l'algorithme suivant :

```
1  def mystère(t):
2      n = len(t)
3      r = 0
4      for i in range(n):
5          j = i
6          while j < n:
7              if i % 2 == 0:
8                  r = r + t[j]
```

```
9               else:
10                  r = r - t[j]
11              j = j + 1
12      return r
```

1. Déroulez l'exécution de cet algorithme avec comme entrée le tableau `t = [2, 1, 3, 6]`. Pour ce faire, vous présenterez l'exécution pas à pas sous la forme d'un tableau d'en-tête :

t	n	r	i	j	t[j]
[2, 1, 3, 6]	4	0	0	0	2
⋮	⋮	⋮	⋮	⋮	⋮

La première ligne vous est donnée et fournit les valeurs de `i`, `j`, `r` et `t[j]` suite à leur initialisation (lignes 3, 4 et 5).

Solution : L'exécution se représente par le tableau suivant :

t	n	r	i	j	t[j]
[2, 1, 3, 6]	4	0	0	0	2
		2		1	1
		3		2	3
		6		3	6
		12		4	
			1	1	1
		11		2	3
		8		3	6
		2		4	
			2	2	3
		5		3	6
		11		4	
		5	3	3	6
				4	

Le résultat de l'appel de `mystère(t)` est donc 5.

2. Montrez que l'appel de `mystère(t)` termine pour tout tableau `t` donné en entrée.

Solution : La fonction `mystère` contient deux boucles, la première sur `i` (boucle `for`), la seconde sur `j` qui est incluse dans la première (boucle `while`). Pour démontrer que `mystère` termine pour tout tableau `t` de longueur $n \geq 1$, il faut démontrer que :

— chaque instance de l'itération sur `j` (pour toute valeur de `i`) termine ; et
— l'itération sur `i` termine aussi.

Commençons par la boucle sur `j`, dans laquelle on rentre à chaque itération de la boucle sur `i`. Quand on entre dans cette boucle sur `j`, `j` est initialisé à `i` (ligne 5) et `j` est incrémenté de 1 à chaque tour (ligne 11). Donc, nécessairement, il existe un tour à la fin duquel `j` vaudra n. Or, quand `j` vaut n, la condition de la boucle (ligne 6) montre que la boucle s'arrête. Par conséquent, la boucle sur `j` s'arrête forcément.

Concernant maintenant la boucle sur `i`, étant donné qu'il s'agit d'une boucle `for`, on sait qu'elle est elle-même bornée par définition. Par ailleurs, comme la boucle sur `j` qu'elle contient termine (cf. paragraphe précédent), on sait qu'elle-même va terminer.

En conclusion, `mystère(t)` termine pour tout tableau `t` passé en entrée.

3. Dénombrez les instructions exécutées par l'appel de `mystère(t)`, avec `t` un tableau de longueur $n \geq 1$. Évaluez la complexité asymptotique de cette fonction en l'exprimant avec la notation « grand O ».
Indication : On appelle « instructions » l'ensemble des lignes d'un algorithme correctement écrit (respectant les conventions de base). On pourra par exemple commencer par calculer le nombre de fois que la ligne 11 est exécutée.

Solution :
L'objectif est de dénombrer chaque instructions en pensant bien à séparer le raisonnement par boucle :

- Commençons par compter le nombre d'instruction en dehors de toute boucle. Il y a 3 instructions de ce type, 2 affectations et 1 `return` (lignes 2, 3 et 12).
- Continuons en nous focalisant sur l'ensemble des instructions portées par la boucle sur `i` :
  - En dehors de la boucle sur `j`, on compte 1 affectation (ligne 5) répétée n fois car `i` varie de 0 à $n-1$ par pas de 1. Soit n instructions.
  - Par ailleurs, le test implicite de la boucle sur `i` (ligne 4) est exécuté $n+1$ fois.
  - ▶ Soit $n+n+1 = 2n+1$ instructions portées par la boucle sur `i`.
- Poursuivons en dénombrant les instructions portées par la boucle sur `j` :
  - Il y a 1 test (ligne 7) et 2 instructions d'affectation (lignes (8 ou 10) et 11), soit 3 instructions au total. Sachant que `j` évolue à chaque instance de boucle de `i` à $n-1$, ces 3 instructions sont répétées $\frac{n\times(n+1)}{2}$ fois, en tout. Soit $3 \cdot \frac{n\times(n+1)}{2}$ instructions.
  - Par ailleurs, le test de la boucle sur `j` (ligne 6) est répété une fois de plus à chaque instance de boucle, soit $\frac{(n+1)\times(n+2)}{2} - 1$ fois.
  - ▶ Soit $3 \cdot \frac{n\times(n+1)}{2} + \frac{(n+1)\times(n+2)}{2} - 1 = \frac{3n^2+3n}{2} + \frac{n^2+3n+3}{2} - \frac{2}{2} = \frac{4n^2+6n+5}{2}$ instructions portées par la boucle sur `j`.
- ▶ On a donc en tout

$$3 + (2n+1) + \frac{4n^2+6n+5}{2} = 2n^2 + 5n + \frac{13}{2}$$

instructions. Ceci est en $O(n^2)$. *On n'attend pas de justification de cela, mais au cas où, en voici une.* Notons qu'il existe $N = 4$ et $c = 3$ tels que

$$\forall n \geq N \quad c \cdot n^2 \geq 2n^2 + 5n + \frac{13}{2}$$

ce qui permet de conclure que la fonction `mystère` est donc en $O(n^2)$.

4. Que la fonction `mystère` calcule-t-elle ?

Solution : La fonction mystère commence par sommer les n entiers du tableau (1ère itération sur `i`). Elle retranche ensuite de cette somme la somme des $n-1$ deniers entiers (2ème itération sur `i`). Elle ajoute ensuite à cette somme la somme des $n-2$ derniers entiers (3ème itération sur `i`). Et ainsi de suite, en déplaçant l'offset d'un cran vers la droite à chaque itération sur `i`.
Par conséquent, elle retourne la somme des entiers d'indice pair du tableau passé en entrée.

Exercice 3 (Conception d'algorithmes)

1. On a vu en cours et en TD différents tris de tableau. Un autre tri est le *tri par sélection*. L'idée est de trier le tableau en commençant par en extraire l'élément minimum, qu'on place en tête du tableau. On extrait ensuite l'élément minimum du tableau privé de son premier élément, qu'on place à nouveau en tête du tableau, et ainsi de suite jusqu'à la fin. Pour réaliser un tel tri, il faut pouvoir calculer le minimum dans une portion de tableau.

   Proposez une fonction Python `minimum_fin(t, i)` qui prend en argument un tableau `t` de longueur `n` et l'indice `i` d'une case du tableau et qui renvoie un indice j tel que `t[j]` est le plus petit entier parmi les éléments `t[i]`, `t[i+1]`, ..., `t[n-1]`. Par exemple, l'appel de `minimum_fin([1, 8, 5, 6], 1)` renvoie l'indice 2 puisque `5` est l'élément minimal parmi les éléments `8`, `5` et `6`.

Solution : On utilise une boucle bornée qui permet de parcourir les éléments d'indice $i+1, i+2, \ldots, n-1$ du tableau, et stocker l'indice m du minimum courant, initialisé avec l'indice i.

```
1  def minimum_fin(t, i):
2      n = len(t)
3      m = i
4      for j in range(i+1, n):
5          if t[j] < t[m]:
6              m = j
7      return m
```

2. Pour un entier naturel n, on appelle *racine carrée entière* de n le plus grand entier s tel que $s^2 \leq n$. En particulier, si n est le carré d'un entier, s est égal à $\sqrt{n}$. Proposez une fonction de `racine_carree_entiere` prenant une unique entrée qui est un entier n et qui renvoie -1 si n est strictement négatif, ou la racine carrée entière de n sinon. Par exemple, `racine_carree_entiere(4)`

renvoie 2, `racine_carree_entiere(10)` renvoie 3 et `racine_carree_entiere(-50)` renvoie -1.
Indication : Il est interdit d'utiliser la fonction `sqrt` *de la bibliothèque* `math` *de Python.*

Solution : On utilise une boucle non bornée *Tant que* en incrémentant petit à petit la valeur de s initialisée à 0. On s'arrête dès lors que $(s+1)^2 > n$.

```
1  def racine_carree_entiere(n):
2      if n < 0:
3          return -1
4      s = 0
5      while (s+1) * (s+1) <= n:
6          s = s + 1
7      return s
```