Algorithmique et structures de données
Examen de 1ère session.

Ensimag 1ère année

28/04/2022

Durée : 3 heures notes de cours autorisées.
La note tiendra compte de la qualité de la rédaction. Le barème est donné à titre indicatif.

1 Croissance des fonctions (2pts)

On considère les fonctions $g_i : \mathbb{Z}_{\geq 1} \to \mathbb{R}_{\geq 0}$, $1 \leq i \leq 6$, définies par :
— $g_1(n) = 2^n$
— $g_2(n) = n \cdot (\log n)^3$
— $g_3(n) = n^{4/3}$
— $g_4(n) = n^{\log n}$
— $g_5(n) = 2^{2^n}$
— $g_6(n) = 2^{n^2}$

1. **0.5pt** Indiquer quelles fonctions parmi $g_1, g_2, \ldots, g_6$ sont d'ordre de croissance polynomiale.
2. **0.5pt** Trier les fonctions $g_1, g_2, \ldots, g_6$ par leur ordre de croissance. Cela veut dire que si $f(n)$ précède $g(n)$ dans votre ordre, alors $f(n) = O(g(n))$.
3. **1.0pt** Soient $f, g : \mathbb{N} \to \mathbb{N}$ telles que $g = O(f)$. Montrer que $f + g = \Theta(f)$.

2 Recherche d'un minimum local (9 pts)

On peut trouver parmi n objets un objet de valeur minimale en temps $\Theta(n)$. Pour n très grand cela n'est souvent pas assez efficace et on se contente alors d'un *minimum local* : un objet de valeur inférieure ou égale au minimum des valeurs de ses objets voisins.

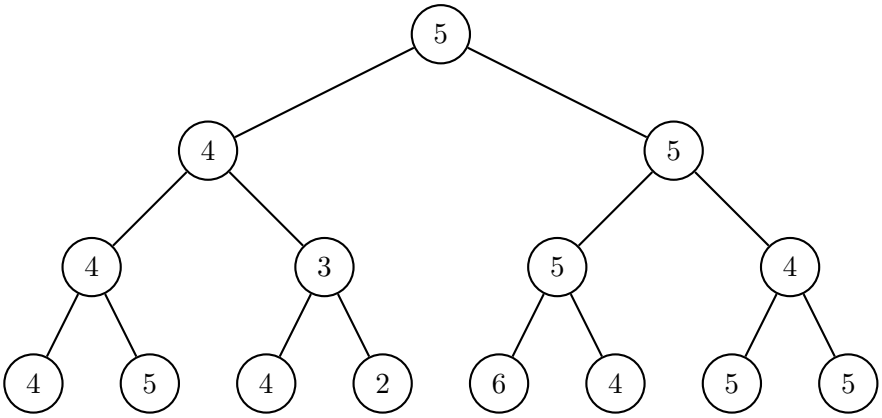
Dans un premier temps on commence par considérer des arbres binaires.

Soit T un arbre binaire non-vide de hauteur h avec racine r (la *hauteur* d'un arbre est la longueur maximale d'un chemin entre la racine et une feuille). Un nœud v de T est un minimum local si la valeur de v est inférieure ou égale au minimum des valeurs de son parent et ses fils (si existant).

Vous pouvez utiliser la classe `Noeud` suivante :

```
class Noeud:
    def __init__(self, valeur):
        self.valeur = valeur
        self.fils = [None, None]
```

1. **0.5pt** Indiquer tous les minimums locaux de l'arbre binaire suivant.



2. **1.0pt** Un *minimum global* de l'arbre binaire T est la valeur minimale des nœuds de T. Montrer qu'un minimum global de T est aussi un minimum local et que le contraire est faux.
3. **1.0pt** Proposer une implémentation de la fonction `is_min_local(v, parent)` en python qui, étant donné un `Noeud` `v` et son parent `parent`, retourne `True` si `v` est un minimum local et `False` sinon. (*remarque :* potentiellement `parent` est égale à `None`)
4. On considère la procédure récursive suivante :

```
def min_local_rec(v, parent):
        if is_min_local(v, parent):
                return v
        if v.fils[0] != None and v.fils[0].valeur <= v.valeur:
                return min_local_rec(v.fils[0], v)
        elif v.fils[1] != None and v.fils[1].valeur <= v.valeur:
                return min_local_rec(v.fils[1], v)
        assert False
```

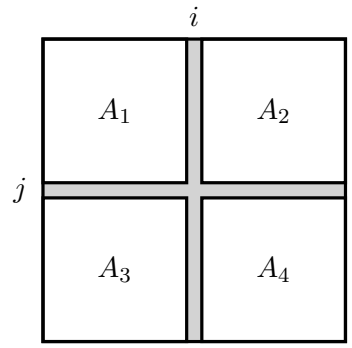
   (a) **1.0pt** Supposons qu'on appelle `min_local_rec(r, None)`, où `r` est un `Noeud` qui correspond à la racine de l'arbre binaire non-vide T. Montrer qu'on n'arrive jamais à la dernière ligne `assert False`.

   (b) **1.5pt** Déduire de (a) que `min_local_rec(r, None)` trouve un minimum local de T en temps $O(h)$, où h est la hauteur de T.

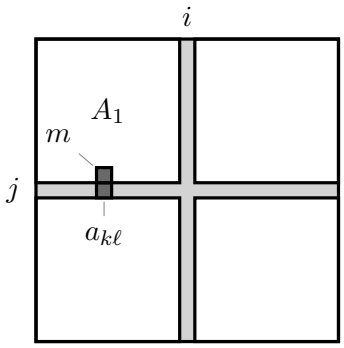
On s'intéresse maintenant à la recherche d'un minimum local d'une matrice. Soit $A \in \mathbb{R}^{n\times n}$ une matrice $n \times n$. On note par a_{ij} l'élément qui se trouve dans la ligne i et la colonne j de la matrice A. L'élément a_{ij} est un *minimum local* si a_{ij} est inférieur ou égal aux éléments voisins au dessus, au dessous, à gauche et à droite de a_{ij}. Par exemple, a_{00} n'a que les deux voisins a_{01} et a_{10}.

5. **1.0pt** Écrire la fonction `search_min(A, i, j)` en python qui, étant donné la matrice `A` et deux indices `i` et `j`, renvoie les indices d'un élément de valeur minimale parmi les éléments de la ligne i et la colonne j.

On partitionne la matrice A en quatre sous-matrices A_1, A_2, A_3, A_4 comme indiqué dans la Figure 1a. Soit $a_{k\ell}$ l'élément renvoié par la fonction `search_min` de la Question 5. On peut vérifier facilement (en temps $O(1)$) si $a_{k\ell}$ est un minimum local. Si oui, on renvoie directement l'élément $a_{k\ell}$. On suppose donc dans ce qui suit que $a_{k\ell}$ n'est pas un minimum local et, par conséquent, il y a un élément voisin m de $a_{k\ell}$ tel que $m < a_{k\ell}$. On suppose sans perte de généralité que m se trouve dans A_1 ; voir la Figure 1b.



(a) Partition de la matrice A en sous-matrices A_1, A_2, A_3, A_4 selon le choix de la ligne i et la colonne j.



(b) L'élément $a_{k\ell}$ qui est le minimum global de la ligne i et la colonne j et son voisin m dans A_1 avec $m < a_{k\ell}$.

6. **1.0pt** Montrer qu'un minimum *global* de la sous-matrice A_1 est un minimum local de A. (*Conseil :* considérer le cas que m' est adjacent à la ligne i ou la colonne j ainsi que le cas contraire.)
7. **1.0pt** Proposer un algorithme récursif efficace (diviser pour régner) en pseudo-code qui trouve un minimum local de A.
8. **1.0pt** Donnez la formule récursive du coût de votre algorithme et en déduire à l'aide du master theorem le coût de l'algorithme. 7

3 Arbres de Fenwick (9pts)

On s'intéresse dans cet exercice à l'optimisation de calculs de préfixes.

Étant donné un tableau T de n entiers, un calcul de préfixe prend en entrée deux indices i et j tels que $i <= j$ et renvoie $pf(T, i, j) = \sum_{k=i}^{j} T[k]$.

Une approche naïve consiste à boucler sur tous les indices entre i et j mais le coût serait alors linéaire et l'on va chercher une approche plus efficace.

On commence par stocker au lieu du tableau T un tableau P de taille n tel que $P[i] = pf(T, 0, i)$. Par exemple si T vaut $[1, 0, 1, 1, 0, 1]$ alors P vaut $[1, 1, 2, 3, 3, 4]$.

1. **1.0pt** Proposer un algorithme efficace pour le calcul de P. Quel est son coût au pire cas ?
2. **1.0pt** Proposer à l'aide de P un algorithme calculant pf. Quel est son coût au pire cas ?

Pour le moment tout semble ok, mais on nous avertit que le tableau T va subir des mises à jour entre les différents calculs de préfixes.

3. **1.5pt** Quel est au pire cas le coût d'une mise à jour de P lors d'un changement d'un unique entier de T ? Au meilleur cas ? En moyenne (en supposant un tirage uniforme de l'entier changé) ?

On se propose maintenant d'écrire une structure efficace pour les calculs de préfixes aussi bien que pour les mises à jour : les arbres de Fenwick.

Un arbre de Fenwick contient n nœuds (v_i) numérotés de 0 à $n-1$. Chaque nœud contient une valeur entière mais la structure de l'arbre ne dépend pas des ces valeurs mais des numéros des nœuds. La racine (v_0) est un nœud spécial et ne contient pas de valeur.

Pour positionner les nœuds dans l'arbre on code leur numéro en binaire puis on utilise les règles suivantes :

— un nœud avec k bits vrais dans le codage de son indice est stocké sur le niveau k (en particulier, les nœuds numérotés par une puissance de 2 sont des fils directs de la racine).

— un nœud avec k bits vrais est donc le fils d'un nœud avec $k-1$ bits vrais : on prend comme père celui dont les $k-1$ bits sont égaux à nos $k-1$ bits de poids forts.
— les fils sont triés par indices.

Par exemple pour $n = 8$ on obtient l'arbre de la figure 2.

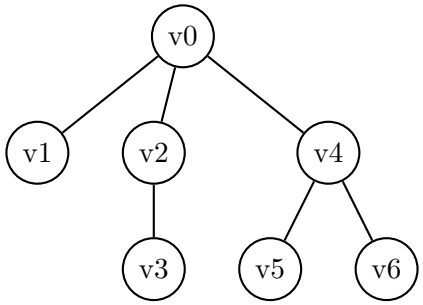


FIGURE 2 : Arbre de Fenwick à 7 nœuds

Pour trouver la position de v_6 par exemple, on note 6 en binaire : 110. Il a deux bits vrais et est donc placé au niveau 2. Pour trouver son père, on regarde l'entier composé de ses $k-1=1$ bits de poids forts soit 100 qui code 4. v_6 est donc un fils de v_4.

4. **1.5pt** Dessiner l'arbre de Fenwick pour $n = 12$.
5. **1.0pt** Quelle est la hauteur de l'arbre en fonction de n ? (Prouvez-le)

On s'occupe maintenant des valeurs stockées dans chaque nœud. Soit v_i un nœud de père v_j. On stocke dans v_i la valeur $pf(j+1, i)$. Si un nœud v_i est un fils direct de la racine, il stocke comme valeur $pf(1, i)$. Note : on suppose ici pour simplifier que le tableau T est indicé à partir de 1.

Par exemple, pour le tableau T contenant $[1, 2, 1, 0, 0, 1]$ on obtient l'arbre de la figure 3 :

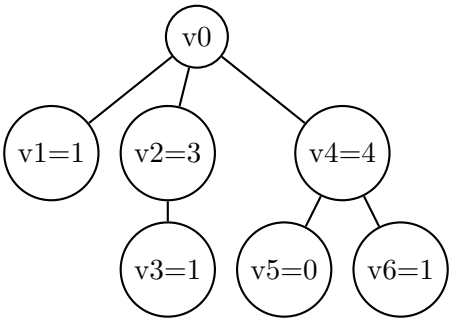


FIGURE 3 : Arbre de Fenwick pour [1,2,1,0,0,1]

6. **1.0pt** Que vaut $pf(1, 6)$? Proposer en pseudo-code un algorithme pour le calcul de $pf(1, i)$ à partir d'un arbre de Fenwick. Vous n'êtes pas obligés de rentrer dans les détails mais l'on doit être en mesure de calculer le coût au pire cas. Quel est le coût au pire cas ?
7. **1.0pt** Proposer en pseudo-code un algorithme pour mettre-à-jour un arbre de Fenwick lors de l'ajout d'une valeur a à $T[i]$. Vous n'êtes pas obligés de rentrer dans les détails mais l'on doit être en mesure de calculer le coût au pire cas. Quel est le coût au pire cas ?
8. **1.0pt** Quelle optimisation peut-on réaliser sur le stockage des arbres pour améliorer les performances (sans toucher au coût asymptotique) ?