IFT2015 :: hiver 2020

Cours de structures de données, Université de Montréal

FÉVRIER 4, 2020 *by* CSUROSM

Algorithmique: exercices

- EXERCICES
- LAISSER UN COMMENTAIRE

Note de service: *la soumission de Devoir 1 est maintenant ouverte dans Studium. Le devoir sera évalué à 25 points (et non pas 12.5, parce que Studium enforce des valeurs entières).*

Exercices avec arbres

Dans les exercices suivants, chaque noeud interne N possède les variables N.`parent`,N.`left`, N.`right` pour le parent, et les enfants gauche et droit. Les noeuds externes sont dénotés par `null`.

a.1 Exposition

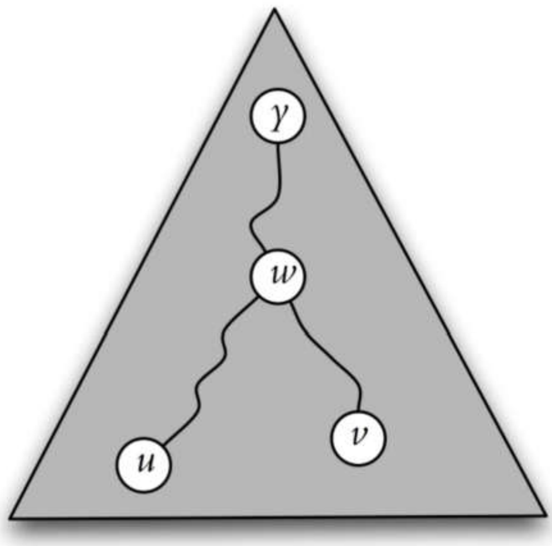
Soit T un arbre binaire. Dans cet exercice, on définit l'**exposition** d'un noeud u comme la distance *la plus courte* à un noeud externe dans le sous-arbre d u. L'exposition d'un noeud externe est 0.

a. Définition récursive. Donner une définition récursive de l'exposition d'un noeud.

b. Algorithme. Montrer un algorithme qui parcourt l'arbre et affiche l'exposition de chaque noeud.

c. Bornes. Donner des bornes inférieures et supérieures sur le maximum de l'exposition dans un arbre à n noeuds internes. (Notez qu'il ne suffit pas de montrer qu'un arbre «extrême» comme l'arbre binaire complet a une telle exposition maximale. Donnez plutôt une preuve par induction.)

a.2 Ancêtre commun



(https://ift2015h19.files.wordpress.com/2019/03/lca.jpg)Soit u et v deux noeuds dans un arbre. Un noeud y est leur **ancêtre commun** si et seulement si u et v appartiennent au sous-arbre enraciné à y. L'**ancêtre commun le plus bas** (ACPB) est l'ancêtre w tel que son sous-arbre ne contient pas d'autres ancêtres communs.
Donner un algorithme `lca`(u,v) qui retourne l'ACPB de deux noeuds internes u,v dans un arbre binaire. Analyser le temps de calcul de l'algorithme.

Indice. Rassemblez les ancêtres de u et v dans une structure de données convéniente, et identifiez les ancêtres en commun en descendant de la racine.

a.3 Longueur du chemin interne

a. Chemin interne. La **longueur du chemin interne** dans un arbre binaire T se définit comme la somme des profondeurs des noeuds internes: $P(T) = \sum_{x \in T} d(x)$ où d(x)$ dénote le niveau du noeud interne x (=0 à la racine).
Donner un algorithme récursif qui calcule P(T) dans un arbre binaire T.

Indice. Développez le code pour `pathlength`(x, d) qui calcule et retourne la somme $\sum_y d(y)$ sur les noeuds internes $y \in T_x$ dans le sous-arbre enraciné à x, dans un parcours récursif. Calculer la profondeur en concomitance, en passant le niveau de x comme l'argument d.

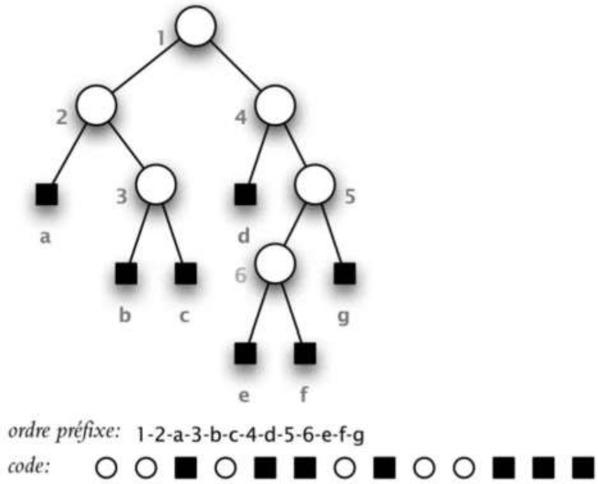
b. Chemin externe. La **longueur du chemin externe** dans un arbre binaire T, dénotée ici par E(T), se définit comme la somme des profondeurs des noeuds externes. Démontrer que

E(T)=P(T)+2n pour tout arbre avec n noeuds internes.

Indice. Procédez par induction: développez des récurrences pour E(T) et P(T).

a.4 Encodage d'arbres

Considérer l'encodage suivant d'un arbre binaire: énumérer les noeuds dans l'ordre préfixe, en affichant ○ pour noeuds internes et ■ pour noeuds externes.



(https://ift2015h19.files.wordpress.com/2019/03/treecodefr.jpg)
a. Décodage. Donner un algorithme qui construit un arbre binaire à partir de son encodage. Vous pouvez assumer que l'entrée, un tableau C[0..m-1], est valide. L'algorithme doit construire l'arbre et retourner sa racine.

Indice. Utiliser une méthode récursive auxiliaire qui prend le tableau C et un indice i, et construit le sous-arbre complète encodé par le segment qui débute avec indice i. La méthode peut retourner une paire (t, N) où N est la racine du sous-arbre et t est sa taille.

b. Nombre d'arbres binaires. Soit t_n le nombre d'arbre binaires à n noeuds internes. Démontrer que $t_n < \binom{2n}{n} < 4^n$ pour tout n>0.

Indice. Utilisez l'encodage des arbres dans l'argument: borner le nombre de codes possibles.

Remarque. On a $t_0 = 1, t_1 = 1, t_2 = 2, t_3 = 5, t_4 = 14$,… On peut compter le nombre d'arbres binaires selon la taille des sous-arbres de la racine. Il y a exactement $t_k \cdot t_{n-1-k}$ arbres binaires avec k noeuds internes dans le sous-arbre gauche de la racine (et (n-1-k) noeuds internes dans le sous-arbre droit). En conséquence, le nombre d'arbre binaires se donne par la récurrence $t_n = \sum_{k=0}^{n-1} t_k \cdot t_{n-1-k}$ pour n>0, et $t_0 = 1$. La solution est $t_n = \frac{1}{n+1}\binom{2n}{n}$. Par l'approximation Stirling, $t_n \sim \frac{4^n}{\sqrt{\pi n^3}}$.

Exercices avec algorithmes

a.5 Crible d'Eratosthène

a.5 Crible d'Eratosthène

L'algorithme d'Eratosthène trouve les nombres premiers inférieurs ou égaux à n par élimination. Il s'agit de marquer tous les multiples 2k, 3k, … ≤ n pour toutes les valeurs entières k=2,3,4,…,n dans l'ordre croissant. Si en arrivant à k, il est non-marqué, alors il est un nombre premier.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

(https://fr.wikipedia.org/wiki/Crible_d%27Ératosthène)

i. Donner l'algorithme `sieve`(n) en pseudocode ou Java pour performer le calcul du crible jusqu'à un n>1. Utilisez un tableau booléen pour marquer les entiers.

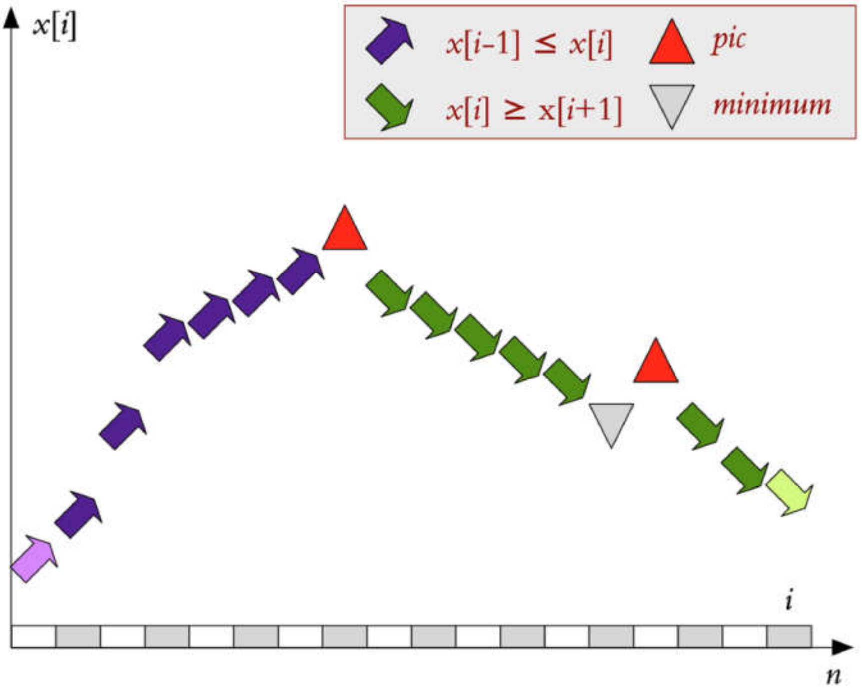
ii. Analyser le temps de calcul de `sieve`.

iii. Le Théorème de nombres premiers (TNP) caractérise la croissance du nombre $\pi(n)$ de nombres premiers inférieurs ou égaux à n: $\pi(n) \sim \frac{n}{\ln n}$. Caractériser la croissance asymptotique du temps amorti (par nombre premier) de `sieve`.

iv. Démontrer qu'il suffit de marquer les multiples de k tandis que $k^2 \le n$. Analysez comment l'accélération impliée (que l'on exécute la boucle pour $k = 2, 3, \ldots, \lfloor\sqrt{n}\rfloor$) change le temps de calcul.

v. Vérifier empiriquement le résultat de **iii.** ou **iv.**

a.6 Chercher le pic



Le *pic* dans un tableau x[0..n-1] est un élément x[i] qui est plus grand que ses voisins: x[i-1] ≤ x[i] et x[i] ≥ x[i+1]. (Afin de permettre le pic même aux extrémités, on imagine x[-1]=x[n]= -∞.)

Donner un algorithme pour trouver un pic. Analyser son temps de calcul. (Temps logarithmique est possible.)

Indice: Diviser-pour-règner avec imbrication (*bracketing*).