INFO I31, partiel, 18 Nov 2014

RÉPONDEZ AUX 23 QUESTIONS DANS L'ORDRE ; MENTIONNEZ LE NUMÉRO DE LA QUESTION DEVANT CHAQUE REPONSE (EVENTUELLEMENT VIDE).

VOUS AVEZ DROIT A TOUS VOS DOCUMENTS PERSONNELS.

TOUT INSTRUMENT ELECTRONIQUE EST INTERDIT.

ECRIVEZ LISIBLEMENT. MERCI !

IL Y A 23 QUESTIONS, NUMEROTEES DE 1 A 23.

Question 1 – Déroulez l'algorithme d'Euclide pour calculer le PGCD de 325 et 240 (en utilisant la division euclidienne, et pas la soustration). Dans la dernière ligne, a ou b est nul, et la case r est vide. Entourez la valeur du pgcd. Le tableau solution a 7 lignes (sans compter la ligne a, b, r).

Réponse :

a	b	r
325	240	85
240	85	70
85	70	15
70	15	10
15	10	5
10	5	0
5	0	_

Donc pgcd$(325, 240) = 5$

Question 2 – Soient $a \geq b > 0$ deux entiers donnés. L'algorithme d'Euclide étendu calcule g le PGCD de a et b ainsi que les plus petits entiers u, v tels que $au + bv = g$. Dans la programmation récursive, la fonction est récursivement appelée sur $a' = b$, et $b' = a \mod b$; elle rend g', u' et v' tels que $a'u' + b'v' = g' =$ PGCD(a', b'). Donnez, sans démonstration, les formules pour calculer g, u, v en fonction de g', u', v'. Vous pouvez utiliser q le quotient (entier) de a par b.

Réponse :

$g = g', u = v', v = u' - q \times v.$

Preuve (non demandée) : soient a et b deux entiers naturels donnés ; on suppose que $a \geq b$. Il faut calculer Bezout$(a, b) = (u, v, g =$ pgcd$(a, b))$ tels que $au + bv = g$. Si $b = 0$, alors $(u, v, g) = (0, 1, b)$ est une solution (n'importe quelle valeur de u convient, puisque $a = 0$). Sinon, soient r, q tels que $r = a \mod b$ et $a = bq + r$ (il est possible de prendre r non négatif dans $[0, b-1]$, ou bien dans $[-b/2, (b+1)/2]$

comme Gauss si bien que $|r| \leq b/2$) ; soit Bezout$(b, r = a \mod b) = (u', v', g')$. Donc

$$bu' + rv' = g' \Rightarrow bu' + (a - bq)v' = g' \Rightarrow av' + b(u' - qv') = g'$$

Une solution possible pour (u, v, g) est donc $(v', u' - qv', g')$. D'autres solutions sont $(u + tb, v - ta, g)$ pour tout t entier.

Question 3 – Déroulez l'algorithme d'Euclide étendu (ou Bézout) pour $a = 325$ et $b = 240$. u et v sont tels que $au + bv = g = PGCD(a, b)$; $r = a \mod b$, et $q = \lfloor \frac{a}{b} \rfloor$ est le quotient entier de la division de a par b Dans la dernière ligne, a ou b est nul, et les cases q et r sont vides.

a	b	r	q	g	u	v
325	240	85	1	5	17	-23
240	85	70	2	5	-6	17
85	70	15	1	5	5	-6
70	15	10	4	5	-1	5
15	10	5	1	5	1	-1
10	5	0	2	5	0	1
5	0	_	_	5	1	0

Question 4 – Citez les noms de 3 algorithmes optimaux de tri

Réponse. Tri par fusion (mergesort), tri par tas (heapsort), tri rapide (quicksort) probablement optimal, tri par insertion dans un arbre binaire équilibré (AVL, 2-3 arbre, b-arbre, etc).

Question 5 – Un étudiant programme une méthode de tri. Son programme met 1 seconde pour trier 100 mille éléments (10^5), et 4 secondes pour trier 200 mille éléments (2×10^5). Notons n le nombre d'éléments à trier. La complexité de son algorithme est en ?

Réponse. Le temps de traitement est multiplié par 4 quand le nombre d'éléments est multiplié par 2. Le programme est donc en $O(n^2)$.

Question 6 – Quel est l'ordre de grandeur du nombre de comparaisons effectuées par un algorithme optimal de tri (qui n'utilise que des comparaisons entre 2 éléments)

Réponse ; $O(n \log n)$. Principe de la preuve : chaque comparaison permet, dans le meilleur des cas, d'éliminer la moitié des ordres possibles a priori. Il y en a $n!$ au départ (factorielle de n). Donc il faut au minimum $\log_2(n!)$ comparaisons. Or $\log(n!) = \log(1) + \log(2) + \ldots \log(n) = O(n \log n)$

Question 7 – Donnez les formules nécessaires pour le calcul récursif de a^n ($a \in \mathbb{R}, n \in \mathbb{N}$). N'oubliez pas le ou les cas terminaux.

Réponse. $a^0 = I$, $a^{2k} = (a^2)^k$, $a^{2k+1} = a \times (a^{2k})$. Donc $\log_2 n$ multiplications sont nécessaires pour calculer a^n.

Question 8 – Citez les noms de 3 algorithmes calculant les plus courts chemins dans un graphe

Réponse. Dijkstra, Ford (ou Ford-Bellman), Dantzig, Warshall.

Question 9 – La difficulté d'un problème NP vient

Réponse. Générer un candidat et tester si un candidat donné est solution se fait en temps polynomial. La difficulté vient de ce que le problème admet une quantité exponentielle de candidats et, contrairement au problème du tri, personne ne sait comment éliminer une fraction significative des candidats en une seule étape (l'équivalent d'une comparaison pour le problème du tri).

Question 10 – Y a t-il des problèmes impossibles à résoudre en informatique ?

Réponse. Oui, certains problèmes sont indécidables, tels que la terminaison d'un algorithme (ou d'une machine de Turing), l'égalité de deux nombres réels (ou complexes), l'égalité de deux fonctions (par exemple de $\mathbb{N}$ dans $\mathbb{N}$).

Question 11 – Un tableau contient n entiers triés par ordre croissant ; combien de comparaisons nécessite la méthode optimale (sans utiliser d'autre structure de données) pour décider si le tableau contient un entier donné

Réponse. Le tableau est trié. Une recherche par dichotomie est donc possible. Elle nécessite $O(\log_2 n)$ comparaisons.

Question 12 – Quelle structure de données utiliser pour répondre en temps constant à la requête précédente ?

Réponse. Une table de hachage (hash table), aussi appelée : dictionnaire.

Question 13 – Un tableau, trié, contient n entiers ; combien de comparaisons nécessite la méthode optimale pour trouver le nombre maximum dans le tableau qui est inférieur (strictement) à un nombre donné, ou pour détecter qu'il n'existe pas

Réponse. Le tableau est trié. Une recherche par dichotomie est donc possible. Elle nécessite $O(\log_2 n)$ comparaisons. Une table de hachage n'est d'aucune aide pour ce problème.

Question 14 – La méthode la plus rapide pour calculer le n ième terme de la suite de Fibonacci, F_n (rappel : $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$ quand $n > 1$) nécessite

Réponse. $O(\log_2 n)$ opérations sur des entiers. Elle utilise la puissance rapide d'une matrice. Voir la question suivante.

Question 15 – Le calcul rapide du membre droit de

$$(F_k, F_{k-1}) = (F_1, F_0) \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k-1}$$

permet de calculer F_k, le k ième terme de la suite de Fibonacci. Pour calculer K_n, où K_0, K_1 sont donnés,

et $K_n = aK_{n-1} + bK_{n-2}$ quand $n > 1$, avec a et b donnés, quelle formule matricielle utiliseriez-vous ?

Réponse.

$$(K_n, K_{n-1}) = (K_{n-1}, K_{n-2}) \begin{pmatrix} a & 1 \\ b & 0 \end{pmatrix} = (K_{n-2}, K_{n-3}) \begin{pmatrix} a & 1 \\ b & 0 \end{pmatrix}^2 = \ldots = (K_1, K_0) \begin{pmatrix} a & 1 \\ b & 0 \end{pmatrix}^{n-1}$$

Question 16 – Pour calculer G_n, où G_0, G_1 sont donnés, et $G_n = g_0 + g_1 G_{n-1} + g_2 G_{n-2}$ quand $n > 1$, avec g_0, g_1 et g_2 donnés, quelle formule matricielle utiliseriez-vous ?

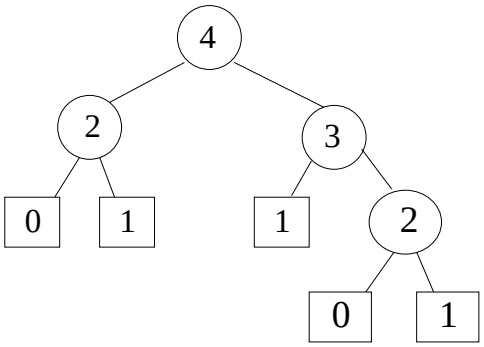
Réponse.

$$(G_n, G_{n-1}, 1) = (G_{n-1}, G_{n-2}, 1) \begin{pmatrix} g_1 & 1 & 0 \\ g_2 & 0 & 0 \\ g_0 & 0 & 1 \end{pmatrix} = (G_1, G_0, 1) \begin{pmatrix} g_1 & 1 & 0 \\ g_2 & 0 & 0 \\ g_0 & 0 & 1 \end{pmatrix}^{n-1}$$

ou bien

$$(G_n, G_{n-1}, g_0) = (G_{n-1}, G_{n-2}, g_0) \begin{pmatrix} g_1 & 1 & 0 \\ g_2 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} = (G_1, G_0, g_0) \begin{pmatrix} g_1 & 1 & 0 \\ g_2 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}^{n-1}$$

On peut aussi prendre la transposée de ces formules, ou changer l'ordre des éléments dans les vecteurs.



Question 17 – L'arbre de Fibonacci T_4 est dessiné ci-dessus. T_0 est une feuille portant l'étiquette 0, T_1 est une feuille portant l'étiquette 1 ; pour $n > 1$, T_n est un noeud binaire, dont le fils gauche est T_{n-2} et dont le fils droit est T_{n-1}. Donnez une formule récursive pour le nombre total de noeuds (feuilles ou noeuds internes), noté t_n, de T_n, pour $n > 1$.

Réponse ; $t_0 = t_1 = 1$ et $t_n = t_{n-1} + t_{n-2} + 1$ pour $n > 1$.

Question 18 – (suite) Donnez une formule récursive pour définir u_n le nombre de feuilles étiquetées 1 de T_n. Que constatez-vous ?

$u_n = u_{n-1} + u_{n-2}$ et $u_0 = 0, u_1 = 1$. La suite u est donc la suite de Fibonacci.

Question 19 – (suite) Donnez une formule récursive pour définir z_n le nombre de feuilles étiquetées 0 de T_n.

$z_n = z_{n-1} + z_{n-2}$ et $z_0 = 1, z_1 = 0$. C'est la suite de Fibonacci à un décalage près. $z_n = u_{n-1}$.

Question 20 – (suite) Remplissez le tableau suivant, où u_n est le nombre de feuilles étiquetées 1 de T_n, z_n le nombre de feuilles étiquetées 0 de T_n, et t_n le nombre de noeuds (feuilles ou non) de T_n.

n	0	1	2	3	4	5	6	7	8	9
t_n	1	1	3	5	9	15	25	41	67	109
u_n	0	1	1	2	3	5	8	13	21	34
z_n	1	0	1	1	2	3	5	8	13	21

Question 21 – (suite) Donnez la formule matricielle pour calculer rapidement t_n.

Réponse.

$$(t_n, t_{n-1}, 1) = (t_{n-1}, t_{n-2}, 1) \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} = (t_1, t_0, 1) \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}^{n-1}$$

Donc la méthode rapide pour la puissance de G^{n-1}, en $(\log n)$ multiplications, donne t_n.

Remarque. $t_n = 2u_{n+1} - 1$, ce qu'on peut facilement prouver par récurence.

Question 22 – Le temps d'exécution d'un algorithme est $T(n)$ pour une donnée de taille n, où $T(1) = 1$ et $T(n) = 3T(n/2) + n$. Prouvez en quelques lignes, par récurrence sur k, que $T(2^k) = 3^{k+1} - 2^{k+1}$. La formule est vraie pour $k = 0$, ce qui permet d'amorcer la récurrence.

Réponse. Supposons la formule vraie jusqu'à $n = 2^k$. Il faut la prouver pour $2n = 2^{k+1}$. Alors

$$T(2^{k+1}) = 3T(2^k) + 2^{k+1} = 3(3^{k+1} - 2^{k+1}) + 2^{k+1} = 3^{k+2} - 2^{k+2}$$

CQFD.

Question 23 – (suite) La complexité d'un algorithme est $T(2^k) = 3^k, n = 2^k$. Déduisez-en la complexité de $T(n)$ en fonction de n ; vous la prouverez en une ligne ; les valeurs numériques de $\log_2 3$, ou autre, ne sont pas demandées.

Réponse. $k = \log_2 n$. La preuve suivante utilise : $3 = 2^{\log_2 3}$ et $2^{\log_2 n} = n$.

$$T(n) = 3^k = 3^{\log_2 n} = \left(2^{\log_2 3}\right)^{\log_2 n} = 2^{(\log_2 3)\times(\log_2 n)} = \left(2^{\log_2 n}\right)^{\log_2 3} = n^{\log_2 3} = n^{1.5849625007211563}$$