Esiee-Paris - unité d'algorithmique - Feuille d'exercices numéro 10

Janvier 2024 – R. Natowicz, I. Alame, A Çela, X. Hilaire, T. Wu, W. Xu

Exercice 18. Puissance entière. On calcule a^n dans p, où a et n sont des valeurs entières positives.

1. Écrire une fonction "puissance entière séquentielle" `int pes(int a, int n)` qui retourne la valeur a^n par un calcul séquentiel. Complexité : $\Theta(n)$ multiplications entières. Donner une version itérative et une version récursive de cette fonction.
2. écrire une fonction "puissance entière dichotomique" `ped(int a, int n)` qui retourne la même valeur par un calcul dichotomique. Complexité : $\Theta(\log_2 n)$ multiplications entières. Donner une version itérative et une version récursive.
   Pour bien comprendre l'intérêt de l'approche dichotomique : avec $n = 2^{20} \approx 10^6$, la puissance séquentielle calculera 2^{20} multiplications entières ; la puissance dichotomique n'en calculera que 20.

Exercice 19. Entiers bi-carrés. L'entier naturel n est bi-carré s'il peut s'écrire $n = x^2 + y^2$ où x et y sont des entiers naturels.

1. Écrire une fonction `int[] bicarrés(int n)` qui retourne : si n est bi-carré alors un couple (x, y) tel que $n = x^2 + y^2$ sinon le couple (-1,-1). La complexité de cette fonction doit être $\Theta(n)$ multiplications entières. Vous obtiendrez ce beau résultat par une recherche arrière dans un tableau fictif $F[0 : n+1][0 : n+1]$ de terme général $F[i][j] = i^2 + j^2$. Ce tableau F est un support pour le raisonnement. Vous ne le construirez pas.
2. Améliorer cette fonction pour obtenir une version qui calcule une et une seule multiplication entière, suivie de $\Theta(n)$ décalages à gauche qui sont des opérations en temps constant (multiplication entière par deux.) Exemple d'exécution du programme :

```
% java Bicarrés 21
21 premiers entiers : bi-carrés et non bi-carrés...
0 = 0^2 + 0^2
1 = 0^2 + 1^2
2 = 1^2 + 1^2
3 n'est pas bi-carré
4 = 0^2 + 2^2
5 = 1^2 + 2^2
6 n'est pas bi-carré
7 n'est pas bi-carré
8 = 2^2 + 2^2
9 = 0^2 + 3^2
10 = 1^2 + 3^2
11 n'est pas bi-carré
12 n'est pas bi-carré
13 = 2^2 + 3^2
14 n'est pas bi-carré
15 n'est pas bi-carré
16 = 0^2 + 4^2
17 = 1^2 + 4^2
18 = 3^2 + 3^2
19 n'est pas bi-carré
20 = 2^2 + 4^2
%
```

Exercice 20. Couples de valeurs de T de somme dans T. Le tableau d'entiers $T[0 : n]$ est strictement croissant. On veut calculer le nombre de couples de valeurs de T, (t_i, t_j), $0 \leq i \leq j < n$, dont la somme $t_i + t_j$ est une valeur de T. Exemple avec $T = [1, 2, 3, 4, 6]$, il y a 6 couples de valeurs dont la somme est dans T. Ce sont les couples $(1, 1), (1, 2), (1, 3), (2, 2), (2, 4), (3, 3)$.

Le premier algorithme auquel on pense "touche" tous les couples (t_i, t_j), $0 \leq i \leq j < n$. Pour chacun d'eux il recherche la valeur $t_i + t_j$ dans T. Si la recherche est séquentielle, l'algorithme est en $\Theta(n^3)$. Si la recherche est dichotomique, l'algorithme est en $\Theta(n^2 \log_2 n)$. L'algorithme que vous étudiez ici est en $\Theta(n^2)$. Ainsi, dans les pires cas respectifs des trois programmes, le vôtre sera respectivement n fois et $\log_2 n$ fois plus rapide. Exemple : avec $n = 2^{20} \approx 10^6$ votre programme sera respectivement 2^{20} fois et 20 fois plus rapide.

Questions :

1. Dans un premier temps, pour une valeur s fixée, nous calculons le nombre n_s de couples (t_i, t_j) de valeurs de T, $0 \leq i \leq j < n$, dont la somme est égale à s. Exemple : avec $T = [1, 2, 3, 4, 5, 7]$ et $s = 6$, il y a 3 couples : $(1, 5), (2, 4), (3, 3)$. Puis dans un second temps la valeur s prendra successivement toutes les valeurs de T.

   Premier programme auquel on pense pour calculer le nombre de couples n_s : "toucher" chaque couple (t_i, t_j) ; pour chacun d'eux comparer la somme $t_i + t_j$ à la valeur s.
   Pour chaque couple (t_i, t_j) les opérations sont en temps constant : calcul de la somme $t_i + t_j$ et comparaison avec la valeur s. La complexité de ce programme *naïf* est $\Theta(n^2)$ car il y a $\frac{n \times (n+1)}{2}$ couples[1]. Nous n'écrirons pas ce programme. Nous écrirons un programme en $\Theta(n)$.

   Fixons les ordres de grandeur : pour $n = 10^3$ et s fixé, le rapport des temps de calcul sera de l'ordre de 10^3 en faveur de notre programme. De plus, rappelons-nous que dans un second temps la valeur s prendra toutes les valeurs de T. Alors le rapport des temps de calcul sera de l'ordre de $10^3 \times 10^3 = 10^6$.

   Écrire une fonction `int ncss(int[] T, int s)`, **n**ombre de **c**ouples de **s**omme **s**, qui retourne ce nombre de couples avec une complexité $\Theta(n)$.
2. Écrire la fonction `int ncst(int[] T)`, nombre de couples de T à somme dans T, qui calcule avec une complexité $\Theta(n^2)$ le nombre de couples (t_i, t_j), $0 \leq i \leq j < n$, dont la somme est dans T. Ce programme "touche" chaque valeur t_i de T et applique la recherche arrière de la question précédente avec $s = t_i$. Il s'agit d'une simple boucle de parcours du tableau T. Inutile de donner l'invariant.

---

[1] n couples $(t_0, t_j), 0 \leq j < n$; $n-1$ couples $(t_1, t_j), 1 \leq j < n$, ..., 1 couple $(t_{n-1}, t_j), n-1 \leq j < n$. Total : $\frac{n \times (n+1)}{2}$.