Esiee-Paris - unité d'algorithmique - Feuille d'exercices numéro 3

Février 2021 – R. Natowicz, I. Alame, A. Çela, D. Courivaud, Driss Garrigue

`Squelette de programme et exemple d'exécution en pages 3 et 4.`

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

1. Écrire une fonction "puissance séquentielle", `ps(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 dichotomique", `pd(int a, int n)`, qui retourne la valeur a^n par un calcul dichotomique. Complexité : $\Theta(\log_2 n)$ multiplications entières. Donner une version itérative et une version récursive de cette fonction.

Pour bien comprendre l'intérêt de l'approche dichotomique : avec $n = 2^{20} \approx 10^6$, la puissance séquentielle exécutera 2^{20} multiplications entières alors que la puissance dichotomique n'en exécutera que 20.

Rappels sur le calcul du produit de deux entiers :

1) L'algorithme de calcul de la multiplication séquentielle des deux entiers positifs a et b dans la variable m repose sur la propriété $I(m, b') : a \times b = m + a \times b'$.

Calcul séquentiel, versions itérative et récursive :

```
int ms(int a, int b){ // retourne le produit de a et b, positifs ou nuls. Calcul séquentiel.
   // I(m,b') : ab = m + ab'
   int m = 0, bprime = b;  // I(m,b')
   while (bprime != 0) { // I(m,b') et b'!=0 ==> I(m+a, b'-1)
      m = m+a; // I(m, b'-1)
      bprime = bprime-1; // I(m,b')
    } // I(m,0) donc m = ab.
   return m;
}
int ms(int a, int b){ // retourne le produit de a et b, positifs ou nuls. Calcul séquentiel.
   if (b == 0) return 0;
   return a + ms(a,b-1);
}
```

2) L'algorithme de calcul de la multiplication dichotomique des deux entiers positifs a et b dans la variable m repose sur la propriété $I(m, a', b') : a \times b = m + a' \times b'$.

Calcul dichotomique, versions itérative et récursive :

```
int md(int a, int b){ // retourne le produit de a et b, positifs ou nuls. Calcul dichotomique.
   // I(m,a',b') : ab = m + a'b'
   int m = 0, aprime = a, bprime = b;  // I(m,a',b')
   while (bprime != 0) // I(m,a',b') et b'!=0
      if (bprime % 2 == 0){ // I(m, 2a',b'/2)
         aprime = aprime << 1; // I(m, a', b'/2)
         bprime = bprime >> 1; // I(m,a',b')
      }
      else { // I(m+a',2a',b'/2)
         m = m + aprime; // I(m, 2a',b'/2)
         aprime = aprime << 1; // I(m,a',b'/2)
         bprime = bprime >> 1; // I(m,a',b')

      }
   // I(m,a',0) donc m = ab.
   return m;
}
int md(int a, int b){ // retourne le produit de a et b, positifs ou nuls. Calcul dichotomique.
   if (b == 0) return 0;
   if (b % 2 == 0) return md(a << 1, b >> 1);
   return a + md(a << 1, b >> 1);
}
```

Autres écritures du calcul dichotomique (question de goût...)

```
int md(int a, int b){
   int m = 0, aprime = a, bprime = b;
   while (bprime != 0) {
      if (b%2 == 1) m = m + aprime;
      aprime = aprime << 1;
      bprime = bprime >> 1;
   }
   // I(m,a',0) donc m = ab.
   return m;
}
int md(int a, int b){
   if (b == 0) return 0;
   int m = md(a << 1, b >> 1);
   if (b%2 == 1) m = m + a;
   return m;
}
```

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 : $(1, 1), (1, 2), (1, 3), (2, 2), (2, 4), (3, 3)$.

Le premier algorithme auquel on pense "touche" tous les couples (t_i, t_j), $0 \leq i \leq j < n$, et 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 obtiendrez 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 nous calculons le nombre de couples (t_i, t_j), $0 \leq i \leq j < n$, de valeurs de T dont la somme est égale à s, où s est fixé. Exemple : avec $T = [1, 2, 3, 4, 5, 7]$ et $s = 6$, ces couples sont au nombre de trois : $(1, 5), (2, 4), (3, 3)$.

   Remarque : le premier programme auquel on pense est en $\Theta(n^2)$: "pour tout couple (t_i, t_j), $0 \leq i \leq j < n$, si $t_i + t_j = s$ incrémenter un *compteur*".

   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 $\Theta(n)$. Vous obtiendrez ce beau résultat par une recherche arrière de la valeur s dans un tableau fictif $T'[0:n][0:n]$ de terme général $T'[i][j] = t_i + t_j$.

2. En déduire une 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), $0 \leq i \leq j < n$, de valeurs de T dont la somme est dans T. Ce programme "touche" chaque valeur t de T et applique la recherche arrière de la question précédente avec $s = t$. Il est construit sur la propriété $I(c, k)$:

   nombre de couples de valeurs de T dont la somme est dans $T[0:n]$ =
   c
   +
   nombre de couples de valeurs de T dont la somme est dans $T[k:n]$

Rappel sur la recherche arrière : T est un tableau d'entiers à m lignes strictement croissantes et n colonnes strictement croissantes. La recherche arrière (vue en cours) retourne le nombre d'occurrences de x dans T. Sa complexité est $\Theta(m + n)$[1]. Elle est construite sur la propriété

$I(c, p, q)$: nombre d'occurrences de x dans $T[0:m][0:n]$ = c + nombre d'occurrences de x dans $T[p:m][0:q]$

D'où la fonction :

```
int ra(int x, int[][] T{ int m = T.length, n.= T[0].length;
// I(c,p,q) : nb d'occ. de x dans T[0:m][0:n] = c + nb d'occ. de x dans T[p:m][0:q]
   int c = 0, p = 0, q = n;  // I(c,p,q)
   while (! (p==m || q==0)) // I(c,p,q) et p < m et q > 0
      if (T[p][q-1] == x) // I(c+1,p+1,q-1)
      {c++; p++; q--;} // I(c,p,q)
      else if (x < T[p][q-1]) // I(c,p,q-1)
         q--; // I(c,p,q)
      else // I(c,p+1,q)
         p++; // I(c,p,q)
   // I(c,p,q) et (p==m ou q==0), donc c = nb d'occ. de x dans T + 0 = nb d'occ. de x dans T
return c;
}
```

[1] $\Theta(m + n)$ versus $\Theta(m \times n)$ pour des recherches séquentielles de x dans les lignes de T et $\Theta(m \times \log_2 n)$ pour des recherches dichotomiques de x dans les lignes de T.

```java
public class TD3{
// Exercice 1
   static int ps(int a, int n){ // puissance séquentielle, version itérative.
      ...
      return p;
   }
   static int psr(int a, int n){ // version récursive
      ...
   }
   static int pd(int a, int n){ // puissance dichotomique, version itérative.
      ...
      return p;
   }
   static int pdr(int a, int n) { // version récursive
      ...
   }
   // Exercice 2
   static int ncss(int[] T, int s){ // ligne et colonnes de T strictement croissantes.
      int n = T.length;
      // R.A. de s dans T'[0:n][0:n] (fictif) de terme général T'[i][j]=T[i]+T[j]
      // I(c,p,q) : nb d'occ. de s dans T' = c + nb d'occ. de s dans T'[p:n][0:q]
      int c = 0, p = 0, q = n; // I(c,p,q)
      // Condition d'arrêt p = n ou q = 0 ou p > q
      // Remarque concernant p > q : on veut les couples (ti,tj), 0 <= i <= j < n,
      // et la R.A. compare s et T[p]+T[q-1]. D'où p > q
      // (la somme T[p]+T[p] sera  examinée lorsque q sera égal à p + 1)
      while (p < n && q > 0 && p < q) // I(c,p,q) et non arrêt
         ...
      return c;
   }
   static int ncst(int[] T){ int n = T.length;
   // I(c,k) : nombre de couples de valeurs de T dont la somme est dans T [0 : n] =
   //         c + nombre de couples de valeurs de T dont la somme est dans T[k : n]
      ...
      return c;
   }
   static int rd(int x, int[] T){ int n = T.length;
   // recherche dichotomique de x dans T. La fonction ncts() appelle la fonction ncss()
   // pour tout s valeur de T. La recherche dichotomique permet l'affichage de l'indice
   // de cette valeur s.
   // Remarque : cet affichage n'est pas demandé dans l'énoncé.
   // La recherche dichotomique retourne i tel que T[i] <= x < T[i+1]
   // I(i,j) : T[i] <= x < T[j-1]
   // Init : i = 0, j = n
   // Arrêt : j = i+2. L'invariant étant toujours vérifié,
   // on aura T[i] <= x < T[(i+2)-1], autrement dit : T[i] <= x < T[i+1]
      if (x<T[0]) return -1;
      if (x==T[n-1]) return n-1;
      if (x>T[n-1]) return n;
      // Cas général : T[0] <= x < T[n-1]
      int i = 0, j = n; // I(i,j)
      while (j != i+2){int k = (i+j)/2;
         if (T[k] <= x) // I(k,j)
            i = k; // I(i,j)
         else // T[i] <= x < T[k], c'est-à-dire T[i] <= x < T[(k+1) -1], donc I(i,k+1)
            j = k+1;
      } // I(i,j) et j = i+1
      return i;
   }

   static void afficher(int[] T){int n = T.length;
      System.out.print("[");
      for (int i = 0; i < n-1 ; i++) System.out.print(T[i] + ", ");
      System.out.print(T[n-1]);
      System.out.println("]");
   }
   public static void main(String[] args){
      int a = 2, nmax = 6;
      System.out.println("puissance séquentielle itérative");
      for (int n = 0; n < nmax; n++)
         System.out.printf("%d^%d = %d\n",a,n,ps(a,n));
      System.out.println("puissance séquentielle récursive");
      for (int n = 0; n < nmax; n++)
         System.out.printf("%d^%d = %d\n",a,n,psr(a,n));
      System.out.println("puissance dichotomique itérative");
      for (int n = 0; n < nmax; n++)
         System.out.printf("%d^%d = %d\n",a,n,pd(a,n));
      System.out.println("puissance dichotomique récursive");
      for (int n = 0; n < nmax; n++)
         System.out.printf("%d^%d = %d\n",a,n,pdr(a,n));
      System.out.println("couples de T à somme dans T");
      int[] T = new int[] {1,2,3,4,6};
      System.out.print("T = "); afficher(T);
      System.out.printf("Il y a %d couples de T à somme dans T\n", ncst(T));
   }
}
```

```
/* Exécution :
puissance séquentielle itérative
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
puissance séquentielle récursive
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
puissance dichotomique itérative
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
puissance dichotomique récursive
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
couples de T à somme dans T
T = [1, 2, 3, 4, 6]
(1,1) car 1 + 1 = 2 = T[1]
(1,2) car 1 + 2 = 3 = T[2]
(1,3) car 1 + 3 = 4 = T[3]
(2,2) car 2 + 2 = 4 = T[3]
(2,4) car 2 + 4 = 6 = T[4]
(3,3) car 3 + 3 = 6 = T[4]
Il y a 6 couples de T à somme dans T
*/
```