Esiee-Paris - unité d'algorithmique

Quelques exercices sur la récursivité – TD 3 bis

Février 2020 – R. Natowicz, I. Alame, A. Çela, D. Courivaud, D. Garrigue

1. Somme des valeurs d'un tableau d'entiers $T[0:n]$. Nous regardons cette somme comme celle des valeurs du 0-suffixe de T.
   La base de la récurrence est obtenue pour le n-suffixe. C'est un sous-tableau vide. Par définition, sa somme est nulle.
   Le cas général est pour $0 \leq k < n$: la somme des valeurs du k-suffixe de T est égale à $T[k]$ + la somme des valeurs du $(k+1)$-suffixe.
   $$\left(\Sigma_{k\leq i<n}\ T[i]\right) = T[k] + \left(\Sigma_{k+1\leq p<n} T[i]\right)$$
   Il s'agit d'une approche séquentielle : la taille du problème restant à résoudre est décrémentée. Complexité $\Theta(n)$

   ```
   int sommeSequentielle(int[] T){ int n = T.length; return ss(T,0,n); }
   int ss(int[] T, int k, int n) { if (k == n) return 0; return ... ; }
   ```

2. Somme des valeurs d'un tableau d'entiers $T[0:n]$. La somme des valeurs du sous-tableau $T[i:j]$ est nulle si ce sous-tableau est vide. Sinon, soit $k = \frac{i+j}{2}$ l'indice médian de l'intervalle $[i:j]$. Alors :
   $$\left(\Sigma_{i\leq p<j}\ T[p]\right) = \left(\Sigma_{i\leq p<k} T[p]\right) + T[k] + \left(\Sigma_{k+1\leq p<j} T[p]\right)$$
   Il s'agit d'une approche "diviser pour régner". Le problème de départ est décomposé en deux sous-problèmes de tailles moitié. Chacun des deux est résolu récursivement. Complexité $\Theta(n)$ car l'équation du temps de calcul est : Temps(0 ou 1) $= \alpha =$ constante , Temps$(n \geq 2) = \Theta(1) + 2 \times$ Temps$(n/2)$. On obtient la complexité $\Theta(n)$ en résolvant cette équation. Remarque : la différence avec l'equation de récurrence du quickSort en $\Theta(n \log n)$ est : Temps$_{qs}(n \geq 2) = \mathbf{\Theta(n)} + 2 \times$ Temps$_{qs}(n \geq 2)$.

   ```
   int sommeDiviserPourRegner(int[] T){ int n = T.length; return sdpr(T,0,n); }
   int sdpr(int[] T, int i, int j) { ... }
   ```

3. Somme de deux entiers a et b, avec $b \geq 0$. Nous savons incrémenter ou décrémenter la valeur d'une variable. Nous voulons calculer la somme $a + b$.
   Base de la récurrence : si $b = 0$, $a + b = a$.
   Cas général, $b > 0$: $a + b = (a + 1) + (b - 1)$.
   Il s'agit d'une approche séquentielle.

   ```
   int somme(int a, int b) { if (b == 0) return a;  ... }
   ```

4. Produit de deux entiers a et b, avec $b \geq 0$. Nous savons calculer la somme de deux entiers et décrémenter la valeur d'une variable. Nous voulons calculer le produit $a \times b$, où $b \geq 0$.
   Base de la récurrence : si $b = 0$, $a \times b = 0$.
   Cas général, $b > 0$: $a \times b = a + a \times (b - 1)$.
   Il s'agit d'une approche séquentielle.

   ```
   int produitSequentiel(int a, int b){ if (b == 0) return 0 ; return ... }
   ```

5. Produit de deux entiers a et b, avec $b \geq 0$. Nous savons calculer la somme de deux entiers, et nous savons en temps constant, diviser et multiplier un entier par 2. Il s'agit d'une division entière : $\frac{2p}{2} = p$, $\frac{2p+1}{2} = p$. La division par deux est le décalage à droite du mot binaire qui représente l'entier considéré. La multiplication entière est son décalage à gauche.
   Exemple : $107 = 1 \times 64 + 1 \times 32 + 0 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$ est représenté par le mot binaire $\boxed{1101011}$.
   Sa division entière par deux est représentée par le mot binaire $\boxed{0110101}$ obtenu par décalage à droite du mot binaire qui le représente. La valeur en base 10 est $0 \times 64 + 1 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 53 = 107/2$.
   Sa multiplication par deux est représentée par le mot binaire $\boxed{11010110}$ obtenu par décalage à gauche. Sa valeur en base 10 est $1 \times 128 + 1 \times 64 + 0 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 = 214 = 107 \times 2$.
   En langage Java (idem en langage C et en langage Python) le décalage à droite d'un mot binaire est réalisé en temps constant par l'opérateur `>>`, opérateur auquel on donne le nombre de bits de ce décalage. Exemples : après l'instruction `int x = 5 >> 1` (décalage à droite d'un bit du mot binaire représentant la valeur 5), la variable x contiendra la valeur 2, et après l'instruction `int x = 5 << 1`, la variable x vaudra 10.

   Base de la récurrence : si $b = 0$, $a \times b = 0$.
   Cas général, $b > 0$: si b est pair, $a \times b = 2a \times b/2$, si b est impair, $a \times b = a + 2a \times b/2$.
   Il s'agit d'une approche dichotomique : le problème à traiter est résolu en résolvant récursivement un sous-problème de taille moitié.

   ```
   int produitDichotomique(int a, int b){ if (b == 0) return 0; return ... }
   ```

6. Affichage à l'endroit des valeurs du tableau $T[0:n]$. Nous regardons ce problème comme l'affichage des valeurs du 0-suffixe de T.
   Base de la récurrence : $k = n$. Le suffixe est vide. Sans rien faire, ce suffixe est affiché à l'endroit.
   Cas général, $0 \leq k < n$: l'affichage à l'endroit du k-suffixe est obtenu en écrivant $T[k]$ puis en affichant à l'endroit le $(k+1)$-suffixe.

```
void endroit(int[] T){ int n = T.length; endroit(T, 0, n); }
void endroit(int[] T, int k, int n) { if (k > 0) ... }
Autre écriture possible : void endroit(int[] T, int k, int n) { if (k == 0) return ;  ... }
```

7. Affichage à l'envers des valeurs du tableau $T[0:n]$. Affichage à l'envers des valeurs du 0-suffixe de T.

   Base de la récurrence : $k = n$. Le suffixe est vide. Sans rien faire, ce suffixe est affiché à l'envers.

   Cas général, $0 \leq k < n$: l'affichage à l'envers du k-suffixe est obtenu en affichant à l'envers le $(k+1)$-suffixe puis en écrivant $T[k]$

   ```
   void envers(int[] T){ int n = T.length; envers(T, 0, n); }
   void envers(int[] T, int k, int n) { if (k > 0) ... }
   Autre écriture possible : void envers(int[] T, int k, int n) { if (k == 0) return ;  ... }
   ```

8. Calculer la renversée d'une chaîne. La renversée d'une chaîne vide est la chaîne vide ; la renversée d'une chaîne non vide est la renversée de son 1-suffixe, "au bout de laquelle" on ajoute son premier élément : renversée("abcd") = renversée("bcd") + "a". Vous aurez besoin de trois méthodes de la classe `String : length()`, `substring(i, j)`, et `+`.

   Exemples :

   1) `String s1 = "abcd", s2 = ""; System.out.println( s1.length() + ", " + s2.length(); ) ;` L'affichage sera `4, 0`.

   2) `System.out.println( "abcd".substring(2,4) + "abcd".substring(0,2) ) ;` L'affichage sera `cdab`.

   ```
   String renversee(String c) { ... }
   ```

9. Vérifier si une chaîne est un palindrome. Une chaîne est un palindrome si elle est égale à sa renversée. Une première façon de résoudre ce problème est :

   `boolean estPalindrome(String c) return c.equals(renversee(c));`

   Une autre façon de faire : une chaîne de longueur 0 ou 1 est un palindrome ; une chaîne de longueur supérieure ou égale à 2 est un palindrome si et seulement si ses premier et dernier caractères sont égaux et si la chaîne privée de son premier et de son dernier caractères est un palindrome. Vous aurez besoin de la méthode `char charAt(int i)` qui retourne le caractère de la chaîne situé à l'indice i. Exemple : `"abcd".charAt(1)"` retourne le caractère `b`.

   `boolean estPalindrome(String c) return (c.length <= 1) || ...  .`

10. Fibonacci, calcul en temps exponentiel (à éviter absolument...). Cette fonction est définie par $f(0) = f(1) = 1$; $f(n \geq 2) = f(n-2) + f(n-1)$. La traduction directe est correcte mais catastrophique en temps de calcul :

    ```
    int f(int n) { if (n == 0 || n == 1) return 1;  return f(n-2) + f(n-1); }
    ```

    Observer son temps de calcul par valeurs n croissantes (et rapprochez vos observations de certains commentaires actuels sur la croissance du nombre de cas de covid : "nous sommes sur un plateau qui augmente lentement, pour l'instant le confinement n'est pas nécessaire".)

    Remarques :

    1) nous savons faire ce calcul en $\Theta(n)$ par une fonction d'invariant $I(k,a,b) : a = f(k-2)$ et $b = f(k-1)$:
    initialisation $k = 2, a = b = 1$,
    condition d'arrêt $k = n+1$. On a alors $I(n+1,a,b)$, autrement dit : $a = f(n-1)$ et $b = f(n)$. On retourne b.
    progression $I(k,a,b)$ et $k < n \Rightarrow I(k+1,b,a+b)$

    2) nous savons le faire en $\Theta(\log n)$ en remarquant que $\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix} \times \begin{bmatrix}f(k-1)\\ f(k-2)\end{bmatrix} = \begin{bmatrix}f(k-1)+f(k-2)\\ f(k-1)\end{bmatrix} = \begin{bmatrix}f(k)\\ f(k-1)\end{bmatrix}$

    donc $\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^{n-1} \times \begin{bmatrix}1\\ 1\end{bmatrix} = \begin{bmatrix}f(n)\\ f(n-1)\end{bmatrix}$.

    Or, nous savons calculer $\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^{n}$ avec une complexité $\Theta(\log n)$: voir l'exercice "Puissance entière" de la feuille de TD numéro 3.

11. Nous avons défini les arbres binaires à valeurs entières par la classe `AB` ci-dessous.

    ```
    class AB{ int v ; g, d : AB ;
       AB(int v, int g, int d) {this.v = v; this.g : g; this.d = d;}
    }
    ```

    Nous avons également défini les fonctions racine, gauche et droit :

    `int racine(AB a) {return a.v;}` qui retourne la valeur à la racine de l'arbre non vide a (la valeur est non définie si a est l'arbre vide.)

    `AB gauche(AB a) {return a.g;}` qui retourne le sous-arbre gauche de l'arbre a (la valeur est non définie si a est l'arbre vide.)

    `AB droit(AB a) {return a.g;}` qui retourne le sous-arbre droit de l'arbre a (la valeur est non définie si a est l'arbre vide.)

    `boolean vide(AB a) {return a == null;}`

    Exemple : l'arbre binaire `a3 = (3, (1, (), (2,(),())), (5, (4,(),()), (7,(),()) ) )` peut être créé par la suite d'instructions suivante.

    ```
    AB a7 = new AB(7, null, null),
          a4 = new AB(4, null, null),
          a5 = new AB(5, a4, a7),
          a2 = new AB(2, null, null),
          a1 = new AB(1, null, a2),
          a3 = new AB(3, a1, a5);
    ```

    Écrire les fonctions suivantes :

- `int hauteur(AB a)` qui retourne la hauteur d'un arbre binaire. Cette hauteur est la longueur maximum des chemins de la racine aux feuilles. Par définition, la hauteur d'un arbre vide est -1.
- `int taille(AB a)` qui retourne le nombre de noeuds de l'arbre. L'arbre $a3$ ci-dessus est de taille 6 (il contient 6 valeurs.)

Un arbre binaire de recherche est un arbre vide ou un arbre non vide dont tout sous-arbre non vide a' vérifie : les valeurs de son sous-arbre gauche sont strictement inférieures à la valeur de sa racine et celles de son sous-arbre droit strictement supérieures.

L'arbre $a3$ de l'exemple ci-dessus est un arbre binaire de recherche.

Écrire les fonctions suivantes.

- `AB inserer(int x, AB a)` où a est un arbre binaire de recherche. Elle retourne l'arbre a dans lequel la valeur x a été insérée. Si la valeur x est déjà présente dans l'arbre, celui-ci est inchangé.

  L'arbre $a3$ de l'exemple ci-dessus aurait pu être créé par la suite d'instruction ci-dessous.

  ```
  AB a3 = null;
  a3 = inserer(3, a3); a3 = inserer(1, a3); a3 = inserer(5, a3);
  a3 = inserer(2, a3);  a3 = inserer(4, a3); a3 = inserer(7; a3);
  ```

- La fonction `AB ahmin(int[] T)`, où $T[0 : n]$ est un tableau d'entiers strictement croissant, retourne un arbre binaire de recherche de hauteur minimum, contenant toutes les valeurs du tableau T. Elle est définie comme suit :

  ```
  AB ahmin(int[] T) {int n = T.length; return ahmin(T, 0, n);}
  ```

  Écrire la fonction `ahmin(int[] T, int i, int j)` qui retourne un arbre binaire de recherche de hauteur minimum contenant toutes les valeurs du sous-tableau $T[i : j]$.

```
class Recursivite{
// quelques exercices pour s'entraîner à la récursivité
   static int sommeSequentielle(int[] T){int n = T.length;
      return ss(T,0,n);
   }
   static int ss(int[] T, int k, int n) {
      ...
   }
   static int sommeDiviserPourRegner(int[] T){ int n = T.length;
      return sdpr(T,0,n);
   }
   static int sdpr(int[] T, int i, int j){
      ...
   }
   static int somme(int a, int b){
      ...
   }
   static int produitSequentiel(int a, int b){
      ...
   }
   static int produitDichotomique(int a, int b){
      ...
   }
   static void endroit(int[] T){ int n = T.length;
      endroit(T,0,n);
      System.out.println();
   }
   static void endroit(int[] T, int k, int n){
      ...
   }
   static void envers(int[] T) { int n = T.length;
      envers(T,0,n);
      System.out.println();
   }
   static void envers(int[] T, int k, int n){
      ...
   }
   static String renversee(String c){
      ...
   }
   static boolean palindrome(String c){
      ...
   }
   static int f(int n){
      ...
   }
   static class AB{int v; AB g, d;
      AB(int v, AB g, AB d){this.v = v; this.g = g; this.d = d;}
   }
   static int racine(AB a){return a.v;}
   static AB gauche(AB a){return a.g;}
   static AB droit(AB a){return a.d;}
   static boolean vide(AB a){return a == null;}
   static int hauteur(AB a){
      ...
   }
   static int max(int x, int y){if (x>=y) return x; return y;}
   static int taille(AB a){
      ...
   }
   static AB inserer(int x, AB a){
      ...
   }
   static AB ahmin(int[] T){ int n = T.length; return ahmin(T,0,n);}
   static AB ahmin(int[] T, int i, int j){
      ...
   }
   static void grd(AB a){
      if (!vide(a)){
         grd(gauche(a));
         System.out.print(racine(a)+ " ");
         grd(droit(a));
      }
   }
   public static void main(String[] Args){
      {  System.out.println("\nExercice 1");
         int[] T = new int[] {0,1,2,3,4};
         System.out.print("T = ");
         endroit(T);
         System.out.printf("sommeSequentielle(T) = % d\n", sommeSequentielle(T));
      }
      {  System.out.println("\nExercice 2");
         int[] T = new int[] {0,1,2,3,4};
         System.out.print("T = ");
         endroit(T);
         System.out.printf("sommeDiviserPourRegner(T) = % d\n", sommeDiviserPourRegner(T));
      }
      {  System.out.println("\nExercice 3");
         System.out.printf("somme(%d,%d) = % d\n", 3,5, somme(3,5));
      }
      {  System.out.println("\nExercice 4");
         System.out.printf("produitSequentiel(%d,%d) = % d\n", 3,5,produitSequentiel(3,5));
      }
      {  System.out.println("\nExercice 5");
         System.out.printf("produitDichotomique(%d,%d) = % d\n", 3,5,produitDichotomique(3,5));
      }
      {  System.out.println("\nExercice 6");
         int[] T = new int[] {0,1,2,3,4};
         System.out.print("T = ");
         endroit(T);
      }
      {  System.out.println("\nExercice 7");
         int[] T = new int[] {0,1,2,3,4};
         System.out.print("T = ");
         envers(T);
      }
```

```
      {  System.out.println("\nExercice 8");
         System.out.printf("renversee(%s) = %s\n", "bonjour", renversee("bonjour"));
      }
      {  System.out.println("\nExercice 9");
         System.out.printf("palindrome(%s) = %b\n", "bonjour", palindrome("bonjour"));
         System.out.printf("palindrome(%s) = %b\n", "elle", palindrome("elle"));
         System.out.printf("palindrome(%s) = %b\n", "kayak", palindrome("kayak"));
      }
      {  System.out.println("\nExercice 10");
         System.out.printf("f(%d) = %d\n", 3, f(3));
      }
      {  System.out.println("\nExercice 11");
         AB a3 = null;
         a3 = inserer(3, a3); a3 = inserer(1, a3); a3 = inserer(5, a3);
         a3 = inserer(2, a3); a3 = inserer(4, a3); a3 = inserer(7, a3);
         System.out.print("Valeurs de l'arbre a3, affichage grd : ");
         grd(a3); System.out.println();
         System.out.printf("hauteur(a3) = %d, taille(a3) = %d\n", hauteur(a3),taille(a3));
         int[] T = new int[] {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14} ; // 15 valeurs,
         // l'arbre de hauteur minimum est de hauteur 4.
         System.out.print("T = "); endroit(T);
         AB a = ahmin(T);
         System.out.print("Parcours GRD de l'ahmin a : "); grd(a); System.out.println();
         System.out.printf("hauteur(a) = %d\n", hauteur(a));
      }

   }
}

/* Compilation et exécution.
% javac Recursivite.java
% java Recursivite

Exercice 1
T = 0 1 2 3 4
sommeSequentielle(T) =  10

Exercice 2
T = 0 1 2 3 4
sommeDiviserPourRegner(T) =  10

Exercice 3
somme(3,5) =  8

Exercice 4
produitSequentiel(3,5) =  15

Exercice 5
produitDichotomique(3,5) =  15

Exercice 6
T = 0 1 2 3 4

Exercice 7
T = 4 3 2 1 0

Exercice 8
renversee(bonjour) = ruojnob

Exercice 9
palindrome(bonjour) = false
palindrome(elle) = true
palindrome(kayak) = true

Exercice 10
f(3) = 3

Exercice 11
Valeurs de l'arbre a3, affichage grd : 1 2 3 4 5 7
hauteur(a3) = 2, taille(a3) = 6
T = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Parcours GRD de l'ahmin a : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
hauteur(a) = 3
%
*/
}
```