Exercices dirigés
séance n°4 - corrigé

Tableaux, Méthode de construction par récurrence

Exercice 1 : Le palindrome

On appelle palindrome un mot qui se lit de la même façon de gauche à droite ou de droite à gauche par exemple rotor, elle. On souhaite écrire un programme qui teste si un mot est un palindrome ou non.

Question 1

Ecrire l'algorithme qui détermine si un mot est un palindrome, en utilisant une méthode de construction de boucle par récurrence

Question 2

Traduire cet algorithme en une fonction, puis écrire le programme complet.

Question 3

Écrire un programme java qui initialise un tableau de caractères, détermine si c'est un palindrome et affiche le résultat.

Exercice 2 : Le tri par sélection

L'idée du tri par sélection est de parcourir le tableau et à chaque étape de partitionner le tableau en un sous-tableau trié et un sous-tableau non trié, tel que tout élément du sous-tableau trié soit inférieur ou égal à tout élément du sous-tableau non trié.
Un tableau est trié si tout élément du tableau est inférieur ou égal à l'élément suivant;
Le partitionnement se fait de la manière suivante : on recherche l'élément minimum dans la partie non triée et on l'échange avec le premier élément de la partie non triée.

Question 1

Construire l'algorithme du tri par sélection. On considèrera deux niveaux de raffinement :
Parcours du tableau de gauche à droite :

- Pour toute position dans le tableau
    1- Rechercher l'élément minimum à partir de cette position ;
    2- Placer l'élément minimum à cette position

On utilisera la méthode de construction par récurrence pour construire les boucles à chaque niveau

Question 2

Traduire cet algorithme en une fonction, puis écrire le programme complet.

--- Exercice 1 ------ Solutions ----------

Question 1

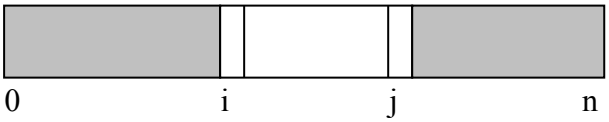
Algorithme :

Idée : parcourir simultanément le tableau à partir de la gauche et à partir de la droite, tester s'il y a égalité des caractères correspondants, si non ce n'est pas un palindrome, si oui poursuivre le parcours, itérer le test jusqu'au parcours complet du tableau.

a) Hypothèse de récurrence

Soit le tableau T, à chaque pas de récurrence, on a :

k [0..i-1], T[k]=T[n-k]



b) Condition d'arrêt

soit `T[i] T[j]` et ce n'est pas un palindrome,
soit `i>=j` et c'est un palindrome.

c) Construire le corps de boucle

```
{  k [0..i-1], T[k]=T[n-k] et i<j et T[i]=T[j]}
// avancer vers la solution et retrouver l'hypothèse de récurrence
i=i+1 ; j=j-1 ;
{  k [0..i-1], T[k]=T[n-k] }
```

d) Initialisation

```
i=0 ; j=n ; // vérifie l'hypothèse qui devient un invariant
{  k [0..i-1], T[k]=T[n-k] }
```

e) Algorithme

```
i=0 ; j=n ;
{  k [0..i-1], T[k]=T[n-k]}
tant que i<j && T[i]=T[j] faire
      i=i+1 ; j=j-1 ;
fait ;
{  k [0..i-1], T[k]=T[n-k] et ((i=j ou T[i]    T[j])}
si T[i]    T[j]alors ce n'est pas un palindrome
sinon c'est un palindrome
fin si ;
```

Programme Java

```
public class Palindrome{
  public static void main( String[] args )
    { String T = "rotor";
    int i=0;
    int j=T.length() - 1;

    //affichage de la suite de caractères saisie
    System.out.print(T);
    //test palindrome
    while ( (i<j) && (T.charAt(i)==T.charAt(j))) {
         i=i+1; j=j-1;
    }
    if (T.charAt(i)!=T.charAt(j))
        System.out.println("  n'est pas un palindrome");
    else
        System.out.println("  est un palindrome");
    }
}
```

Solutions avec fonction

```
/**
 * Cette classe permet de tester si une chaîne de caractères est un
 *  palindrome. Le résultat est affiché
 *
 * @author D.Enselme
 * @version 0.1
 */
public class PalindromeAvecFonctionIterative{
    public static void main( String[] args ) {
        String T = args[0];
        if(T.length()==0) System.out.println(T+" est vide");
        else
            if (palindrome(T))
                System.out.println(T+" est un palindrome");
            else
                System.out.println(T+" n'est pas un palindrome");
    }

    /**
     * test palindrome - solution itérative
     * @param  t   une chaîne de caractères
     * @return     true si la chaîne est un palindrome, false sinon
     *
     * <b>préconditions</b> t n'est pas vide
     */
    public static boolean palindrome(String t){
        int i=0;
        int j=t.length()-1;
        while ( (i<j) && (T.charAt(i)==T.charAt(j))) {
         i=i+1; j=j-1;
        }
        if (T.charAt(i)!=T.charAt(j))
            return false;
        else
            return true;
    }
}
```

```java
/**
 * Cette classe permet de tester si une chaîne de caractères est un
 *  palindrome. Le résultat est affiché
 *
 * @author D.Enselme
 * @version 0.1
 */
public class PalindromeAvecFonctionRecursive{
    public static void main( String[] args ) {
        String T = args[0];
        if (palindromeRec(T,0,T.length()-1))
            System.out.println(T+" est un palindrome");
        else
            System.out.println(T+" n'est pas un palindrome");
    }

    /**
     * test palindrome - solution récursive
     * @param  t   une chaîne de caractères
     * @param  i   premier indice du tableau t
     * @param  j   dernier indice du tableau t
     * @return     true si la chaîne est un palindrome, false sinon
     *
     * <b>préconditions</b> t n'est pas vide
     */
    public static boolean palindromeRec(String t,int i, int j){
       if( t.length()==0 || i==j) return true;
       else if (i<j && t.charAt(i)==t.charAt(j))
                  return palindromeRec(t,i+1,j-1);
            else return false;
    }
}
```

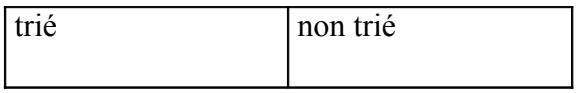
--- Exercice 2 ------ Solutions ----------

L'idée du tri par sélection est de parcourir le tableau et à chaque étape de partitionner le tableau en un sous-tableau trié et un sous-tableau non trié, tel que tout élément du sous-tableau trié soit inférieur ou égal à tout élément du sous-tableau non trié.
Un tableau est trié si tout élément du tableau est inférieur ou égal à l'élément suivant;
Le partitionnement se fait de la manière suivante : on recherche l'élément minimum dans la partie non triée et on l'échange avec le premier élément de la partie non triée.

Soit le tableau tab contenant n entiers :

1er niveau de raffinement : trier le tableau

- hypothèse de récurrence




```
{ k [1..i-1], t[k]>=t[k-1],
  k [i..n], t[k]>=t[i-1]}
```

- Condition d'arrêt

partie non triée est vide => `i-1=n-1`, car le dernier élément est obligatoirement maximal
`la condition d'arrêt est donc : i=n`

```
L'hypothèse devient :
{ k [1..n-1], t[k]>=t[k-1],
  k [n..n], t[k]>=t[i-1]} donc t[n]>=t[n-1]
```

- Construction du corps de boucle

```
{ k [1..i-1], t[k]>=t[k-1],
  k [i..n], t[k]>=t[i-1] et i<n }

x ← indice du plus petit élément de t[i..n];
Echanger t[i] et t[x] ;
// retrouver l'hypothèse de récurrence
i← i+1;

{ k [1..i-1], t[k]>=t[k-1],  k [i..n], t[k]>=t[i-1]}
```

- **Initialisation**

```
i=0 ;
{ k [1..i-1], t[k]>=t[k-1],  k [i..n], t[k]>=t[i-1]} est vraie car
 k [1..-1] est vide et  k [0..n], t[k]>=t[-1] qui n'existe pas }
```

Algorithme

```
i=0 ;
{ k [1..i-1], t[k]>=t[k-1],  k [i..n], t[k]>=t[i-1]}
tant que i<n faire
      x ← indice du plus petit élément de t[i..n];
      Echanger t[i] et t[x] ;
      i←i+1 ;
fait ;
{ k [1..i-1], t[k]>=t[k-1],  k [i..n], t[k]>=t[i-1] et i>=n}
```

=> Le tableau t est trié.

2ième niveau de raffinement

2.1 Rechercher le minimum dans le sous-tableau `t[i..n]`

Idée : parcourir le tableau et retenir la position du minimum dans la partie parcourue

- Hypothèse de récurrence


```
{ x  k [i..j], t[x]<=t[k] }
```

- Condition d'arrêt

la partie non parcourue est vide : `j=n`

- Construction de la boucle

```
{  k [i..j], t[x]<=t[k] et j<n}
si t[j+1]<t[x]
alors
      x=j+1 ;
finsi ;
j←j+1 ;
{  k [i..j], t[x]<=t[k] }
```

- Initialisation

```
x=i;  j=i ;
{  k [i..j], t[x]<=t[k] }  devient
{  k [i..i], t[i]<=t[k] }
```

Algorithme

```
x=i ; j=i ;
{  k [i..j], t[x]<=t[k] }
tant que j<n faire
      si t[j+1]<t[x]
      alors
            x=j+1 ;
      finsi ;
      j←j+1 ;
fait ;
{  k [i..j], t[x]<=t[k] et j>n-1 }
```

2.2 permuter `t[x]` et `tab[j]`

On utilise un élément temporaire `temp` ;

```
temp=t[x];t[x]=t[j];t[i]=temp ;
```

Algorithme final :

```
i=0 ;
tant que i<=n-2 faire
      min=i ; k=i ;
      tant que k<n faire
            si tab[k]<tab[min] alors min=k ; finsi ;
            k←k+1 ;
      fait ;
      temp=tab[min] ;
      tab[min]=tab[i] ;
      tab[i]=temp ;
      i←i+1 ;
fait ;
```

Programme Java du tri par sélection

```
import java.util.Scanner;
import java.util.Random;
public class tri_selection{
    public static void main (String args []){
        Scanner in = new Scanner(System.in);
        System.out.println("tapez le nombre d'entiers a trier ");
        int n = in.nextInt();
        int[] t = new int[n];

        // saisie et affichage de la suite de nombres à trier
        System.out.println ("tableau a trier  ");
        for (int i=0;i<n;i++) t[i] = in.nextInt();
        for (int a=0; a<=n-1;a++) System.out.print("  " +t[a]);
        System.out.println();

        // Partitionnement du tableau
        // i-1 fin du sous tableau trie
        // i debut du sous tableau a trier
        int i = 0;
        int x;
        int j;
        // pour tout k dans [1..i-1], t[k]>t[k-1],
        // pour tout l dans [i..n], t[k]>=t[i-1]
        while( i<n ){
            x = i;
            j = i;
            // recherche de l'indice du plus petit élément de t[i..n]
            // pour tout k dans [i..j], t[x]<=t[j]
            while( j<n ){
                if( t[j]<t[x] ) x = j;
                j = j+1;
            }
            int temp=t[x];t[x]=t[j];t[i]=temp;
            i = i+1;
        }

        //affichage du tableau trie
        System.out.print("le tableau trie est  ");
        for (int m=0; m<=n;m++) System.out.print("  " +t[m]);
        System.out.println();
    }
}
```

Solution avec fonctions et procédures

```
import java.util.Scanner;
public class tri_selection{
    public static void main (String args []){
        Scanner in = new Scanner(System.in);
        System.out.println("tapez le nombre d'entiers a trier ");
        int n = in.nextInt();
        int[] tableau = new int[n];
        // saisie et affichage de la suite de nombres à trier
        System.out.println ("entrer un entier :  ");
        for (int i=0;i<n;i++) {
            tableau[i] = in.nextInt();
        }
        System.out.println();
        // affichage du tableau à trier
        println( tableau );
        // tri par sélection du tableau
        triSelection( tableau );
        // affichage du tableau trié
        println( tableau );
    }

    static void triSelection( int[] t ){
        int i=0 ;
        int x;
        int tmp;
        // pour tout k dans [1..i-1], t[k]>=t[k-1],
        // pour tout k dans [i..n], t[k]>=t[i-1]
        while( i<t.length-1 ){
            // x indice du plus petit élément de t[i..n];
            x = plusPetitIndice( t,i,t.length-1 );
            // Echanger t[i] et t[x] ;
            tmp=t[x];t[x]=t[i];t[i]=tmp ;
            i=i+1 ;
        }
    }

    static int plusPetitIndice( int[] t, int i, int j ){
        int x=i ;
        int k=i ;
        // pour tout k dans [i..j], t[x]<=t[k] }
        while( k<t.length-1 ){
           if( t[k+1]<t[x] ){
                x=k+1 ;
            }
            k=k+1 ;
        }
        return x;
    }

    // affichage du tableau
    static void println( int[] t){
        for (int m=0; m<t.length;m++) {
            System.out.print("  " +t[m]);
        }
        System.out.println();
    }
}
```