Algorithmique
Correction Partiel n° 1

Info-Sup – Epita

20 jan. 2011

Solution 1 **(Un peu de cours... –** ***4 points*****)**

1. Nous utiliserons une pile lorsque des données doivent être mémorisées et traitées en ordre inverse de leur arrivée.

2. Les différentes possibilités d'arbres binaires particuliers sont : les arbres binaires dégénérés ou filiformes, les arbres binaires parfaits, les arbres binaires complets et les arbres binaires localement complets dont une représentation, pour chacun, pourrait être :

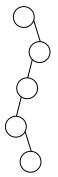

Fig. 1 – Arbre binaire dégénéré ou filiforme.

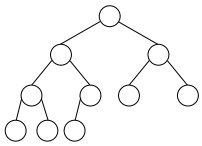

Fig. 2 – Arbre binaire parfait.

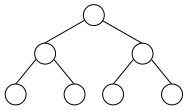

Fig. 3 – Arbre binaire complet.

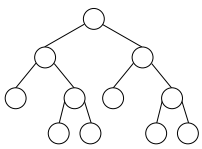

Fig. 4 – Arbre binaire localement complet.

3. Le principe d'un tel algorithme est le suivant : Un palindrome est un mot qui peut se lire de façon identique dans un sens comme dans l'autre (*ex :radar*). Le principe de fonctionnement d'une pile est de restituer les données dans l'ordre inverse de leur arrivée. Dès lors il suffit de lire le mot caractère par caractère et d'empiler ceux-ci au fur et à mesure. En les dépilant ensuite un par un et en les concaténant dans leur ordre de sortie de la pile (inverse de l'ordre d'arrivée dans celle-ci), nous créons un autre mot, inverse de l'original. Il ne nous reste plus qu'à comparer ces deux mots. S'ils sont égaux, nous avons affaire ou non à un palindrome.

   Une optimisation possible, est de ne faire cela que sur la moitié des caractères qui constituent le mot, ce dernier pouvant se lire identiquement dans les deux sens, la comparaison des deux moitiés (dont une inversée) suffira.

Solution 2 **(Cheminons -** ***6 points*****)**

1. **Principe :** On utilise un paramètre local h permettant de connaître la hauteur de chaque nœud. A chaque fois, on vérifie si l'on est sur une feuille ou non. En effet, seuls les nœuds externes (*les feuilles*) nous intéressent. Si l'on se trouve sur une feuille, on augmente la longueur de cheminement externe (*lce*) de la hauteur de celle-ci, sinon on effectue les appels sur les deux fils (éventuellement en vérifiant qu'ils existent) afin d'ajouter à *lce* les hauteurs de leurs feuilles.

   Lorsque l'on revient de l'appel de la procédure, nous sommes en possession de la valeur de la longueur de cheminement externe (*lce*) que nous sommes venus calculer.

2. **Algorithme :**

```
algorithme procedure calcule_lce
parametres locaux
   ArbreBinaire B
   Entier       h
parametres globaux
   Entier       lce

debut
   si B <> arbre-vide alors
      si (g(B) = arbre-vide) et (d(B) = arbre-vide) alors /* feuille */
          lce ←lce + h
      sinon
          si (g(B)<>arbre-vide) alors          /* Test d'accélération */
             calcul_lce(g(B),h+1,lce)
          fin si
          si (d(B)<>arbre-vide) alors          /* Test d'accélération */
              calcul_lce(d(B),h+1,lce)
          fin si
       fin si
   fin si
fin algorithme procedure calcule_lce
```

   Spécification *: Cette procédure ne fonctionne que si l'on effectue correctement son appel (valeurs de paramètres appropriées). Ce qui donne :*

   Appel :

```
lce ←0
parcours_lce(B,0,lce)
```

Solution 3 **(Minimum - *5 points*)**

Spécifications :

La fonction `minimum (t_liste` L`, entier` d`,` f`)` retourne la position de la valeur minimum dans L entre les positions d et f (avec $1 \leq d < f \leq L.longueur$).

Principe :

La valeur en position d est considérée comme la plus petite, on conserve sa position : *pos*.

On parcourt la liste de la position $d+1$ à f, pour chaque élément, s'il est plus petit que la valeur minimale, sa position remplace *pos*.

```
algorithme fonction minimum : entier
    parametres locaux
        t_liste    L
        entier     d, f

    variables
        entier     i, pos
debut
    pos ← d
    pour i ← d+1 jusqu'a f faire
        si L.elts[i] < L.elts[pos] alors
            pos ← i
        fin si
    fin pour
    retourne pos
fin algorithme fonction minimum
```

Solution 4 **(Tri par sélection – *5 points*)**

1. **Spécifications :**
   La procédure `swap (t_element` x`,` y`)` échange le contenu des deux variables x et y.

```
algorithme procedure swap
    parametres globaux
        t_element    x, y
    variables
        t_element    tmp
debut
    tmp ← x
    x ← y
    y ← tmp
fin algorithme procedure swap
```

2. **Spécifications :**
   La procédure `tri_select (t_liste` L`)` trie en ordre croissant la liste L.

   Principe :
   Pour chaque position i de la liste (sauf la dernière) :
   – On recherche la plus petite valeur entre les positions i et n (la longueur de la liste).
   – On échange cette valeur avec celle en position i.

```
algorithme procedure tri_select
    parametres globaux
        t_liste    L

    variables
        entier    i
debut
    pour i ← 1 jusqu'a L.longueur-1 faire
        swap (L.elst[i], L.elts[minimum (L, i, L.longueur)])
    fin pour
fin algorithme procedure tri_select
```