Examen Informatique – CPI 1

20 mai 2010
Durée: 2h
Aucun document ni calculatrice n'est autorisé.

Partie 1. Questions de cours (6 points)

1. Citez un avantage et un inconvénient des listes doublement chainées par rapport aux listes simplement chaînées.
2. Pourquoi est-il pertinent en informatique de définir et d'utiliser des structures linéaires restreintes telles que pile et file, alors que la structure de liste couvre tous les besoins?
3. Citez les différents types de parcours d'un arbre binaire ainsi que toutes les approches algorithmiques que vous connaissez permettant de les réaliser.
4. Présentez en quelques lignes le concept des arbres AVL et justifiez leur intérêt en informatique.
5. Soit un AVL vide, on ajoute les nœuds 25, 60, 35, 10, 5, 20, 65, 70. Ensuite, on supprime le 5. Dessinez l'arbre à chaque étape de ces modifications en explicitant quel type de rotation vous effectuez lorsque c'est nécessaire.

Partie 2. Structures linéaires (7 points)

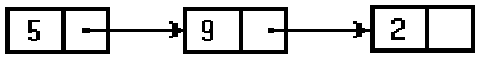
6. Écrivez la fonction « remonterMax » qui reçoit une pile d'entiers à valeurs uniques, et qui renvoie une nouvelle pile contenant les mêmes valeurs, sauf que le maximum est monté au sommet de la pile.
   Exemple : remonterMax (3 → 12 → 7 → 2 → 10) = 3 → 7 → 2 → 10 → 12

```
Fonction getMax(Pile p) : Entier
Variables max, n : Entier
Début
   // p n'est pas vide
   max = sommet(p)
   p = dépiler(p)
   TantQue !estVide(p)
          n = sommet(p)
          Si n > max Alors max = n FinSi
          p = dépiler(p)
   FinTantQue
   Retourner max
Fin

Fonction remonterMax(Pile p) : Pile
Variables
   n, max : Entier
   q : Pile
Début
   Si estVide(p) Alors Retourner pileVide() FinSi
   max = getMax(p)
   TantQue !estVide(p)
          n = sommet(p)
          Si n != max Alors q = empiler(q,n) FinSi
```

```
        p = dépiler(p)
    FinTantQue
    TantQue !estVide(q)
        n = sommet(q)
        q = dépiler(q)
        p = empiler(p,n)
    FinTantQue
    p = empiler(p,max)
    Retourner p
Fin
```

7. Pour représenter les grands nombres entiers qui dépassent la capacité maximale de l'ordinateur, on peut utiliser des listes chaînées. Par exemple, le nombre 295 peut être représenté par la liste suivante (attention au sens inversé pour faciliter les opérations) :



Écrivez l'algorithme qui permet d'additionner 2 entiers représentés par 2 listes chaînées et qui retourne le résultat dans une nouvelle liste. Pour cet exercice, vous pouvez utiliser les opérations d'ajout (ajouterTête, ajouterFin, …) sans les redéfinir.

Fonction addition(l1 : **Pointeur**, l2 : **Pointeur**): **Pointeur**
Variables retenue, v1, v2, v **: Entier**, lres : **Pointeur**
Début
    retenue ← 0
    lres ← listeVide
    Tantque (! estVide(l1)) **et** (!estVide(l2))
        v1 ← valeur(l1)
        v2 ← valeur(l2)
        v ← (v1 + v2 + retenue) mod 10
        retenue ← (v1 + v2 + retenue) div 10
        lres ← ajouterFin(lres, v)
        l1 ← suivant(l1)
        l2 ← suivant(l2)
    FinTantque

    Tantque (! estVide(l1))
        v1 ← valeur(l1)
        v ← (v1 + retenue) mod 10
        retenue ← (v1 + retenue) div 10
        lres ← ajouterFin(lres, v)
        l1 ← suivant(l1)
    FinTantque

    Tantque (! estVide(l2))
        v2 ← valeur(l2)
        v ← (v2 + retenue) mod 10
        retenue ← (v2 + retenue) div 10
        lres ← ajouterFin(lres, v)
        l2 ← suivant(l2)
    FinTantque

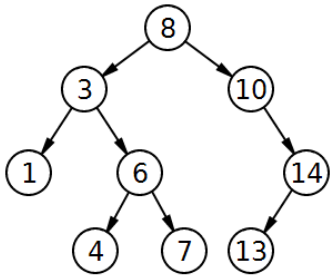
```
    // ajouter la dernière retenue s'il le faut

    Si (retenue > 0) Alors
            lres ← ajouterFin(lres, retenue)
    FinSi

    Retourner lres
  Fin
```

Partie 3. Arbres (7 points)

8. Écrivez une fonction qui calcule la distance entre deux nœuds entiers d'un arbre binaire de recherche. Par définition, la distance entre deux nœuds est le nombre d'arrêtes sur le chemin qui relie ces deux nœuds. Par exemple, pour l'arbre ci-dessous, la distance entre le nœud 7 et le nœud 10 est 4.

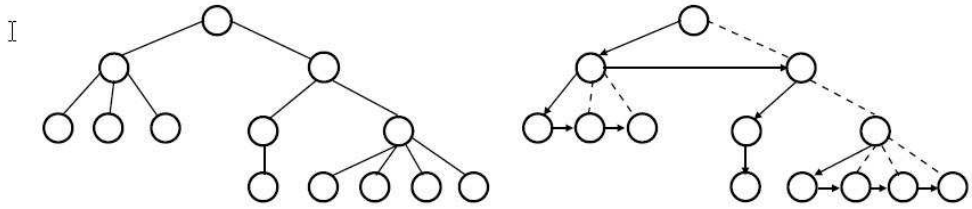


Votre fonction aura 3 paramètres : l'arbre ABR et 2 entiers qui représentent les 2 nœuds. Pour simplifier, on suppose que ces 2 entiers appartiennent à l'arbre.

```
Fonction distance(A : ABR, n1, n2: Entier): Entier
Début
    Si (n1 < racine(A)) et (n2 < racine(A)) Alors
            retourner distance(filsGauche(A), n1, n2)
    Sinon Si (n1 > racine(A)) et (n2 > racine(A))
            retourner distance(filsDroit(A), n1, n2)
    Sinon
            retourner hauteurNoeud(A,n1) + hauteurNoeud(A,n2)
    FinSi
Fin

// Cette fonction calcule la hauteur d'un nœud dans un arbre ABR
Fonction hauteurNoeud(A : ABR, n : Entier): Entier
Début
    Si (n = racine(A)) Alors
            retourner 0
    Sinon Si (n < racine(A))
            retourner 1 + hauteurNoeud(filsGauche(A),n)
    Sinon
            retourner 1+ hauteurNoeud(filsDroit(A),n)
    FinSi
Fin
```

9. Rappel de la structure d'un arbre n-aire :
   - Chaque nœud comporte un pointeur vers son premier fils et
   - Chaque nœud comporte un pointeur vers son frère cadet.



Arbre voulu Implémentation correspondante

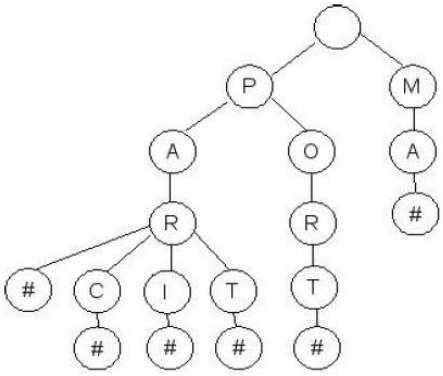
On utilise la constante `arbreVide` et les opérations de base suivantes :

- **estVide:** `Arbre -> Booléen`
- **cons:** `T × Arbre × Arbre -> Arbre`
  `// construction par racine, premier fils, frère cadet`
- **racine:** `Arbre -> T`
- **premierFils:** `Arbre -> Arbre`
- **frereCadet:** `Arbre -> Arbre // renvoie l'arbre vide si pas de frère cadet`

Un arbre n-aire peut représenter une liste de mots, grâce aux spécifications suivantes :

1) Les lettres des mots sont stockées dans les nœuds ;
2) Les mots ayant le même début partagent des nœuds communs ;
3) Le caractère spécial # indique la fin d'un mot ;
4) La racine ne contient rien.

Exemple : L'arbre



contient les mots PAR, PARC, PARI, PART, PORT et MA. Notez que la liste des fils d'un nœud n'est pas triée, et qu'un # n'est pas non plus forcément le premier fils.

Écrivez une procédure qui affiche tous les mots d'un arbre donné.

```
Procédure afficher(a : Arbre, c : Chaîne)
Début
      Si racine(a)='#' Alors
            Ecrire(c)
      FinSi
      Si !estVide(premierFils(a)) Alors
            afficher(premierFils(a) , c & racine(a))
```

```
        FinSi
        Si ! estVide(frereCadet(a)) Alors
                afficher(frereCadet(a), c)
        FinSi

Fin

Procédure afficherMots(a : Arbre)
Début
        Si !estVide(premierFils(a)) Alors
                afficher(premierFils(a) , ‘’)
        FinSi
Fin
```