16 Décembre 2013

Complexité et Algorithmique avancée
« Deuxième Contrôle » durée 1h30

Exercice 1 :

Déterminer l'invariant des boucles pour chacun des algorithmes suivants et en déduire l'ordre de complexité. Justifier votre réponse

Algo_A1()

```
Début
i:=1;
Pour j := 1 à N
        Faire  i=2*i;
        Fait;
Pour j:=1 à i
        Faire <opération>;
        Fait ;
Fin.
```

Algo_A2()

```
Début
i:=1;
Tant que ( i <= N )
        Faire
        i=2*i;
        pour j:=1 à N
                Faire <opération>;
                Fait;
        Fait;
Fin.
```

Solution :

Pour l'algorithme A1, nous avons deux boucles qui se suivent, donc l'invariant de boucle de l'algorithme A1 c'est la somme des invariants des deux boucles.

On remarque aussi que la première boucle s'exécute en N itération et la seconde boucle s'exécute en I itération.

La valeur de i évolue dans la première boucle. A la sortie de la boucle $i = 2^n$.

Ainsi l'invariant associé à cet algorithme $F1(N) = N + 2^N = O(2^N)$.

Pour l'algorithme A2. Nous avons deux boucles imbriqués mais indépendante, donc l'avariant de boucles s'exprime au moyen du produit des invariants associés à chaque boucle.

La boucle interne est linéaire et s'exécute donc en N itération.

La boucle externe par contre évolue plus rapidement qu'un ordre linéaire. Cette rapidité est dû à l'instruction d'incrémentation (i := 2*i). Donc l'indice i est initialisé à

i et subi une série de multiplication par 2 au fil des itérations jusqu'à atteindre ou dépassé la borne supérieure de la borne qui est égale à N.

Donc, si on assimile le nombre d'itération à K. la boucle s'arrête lorsque

$i = 2^K$ devient supérieur N. ($i = 2^K >= N$). Ainsi, le nombre d'itération ou l'invariant de cette boucle est égal à Log (N).

De là nous pouvons conclure que l'invariant global F2(N) = Log(N) * N
= O(N.Log N).

Exercice 2 :

On s'intéresse à l'évaluation des expressions arithmétiques sur les opérateurs de base (+,-, *, /) en s'inspirant de l'algorithme de parcours infixé dans un arbre binaire en vue de déterminer l'ordre de complexité associé à cette opération.

- Expliquer clairement les structures de données utilisées
- Illustrer le processus sur l'expression : ((A+B)*C/D+H)*(Z+T)/X
- Proposer un algorithme pour l'évaluation des expressions. En déduire l'ordre de complexité en fonction du nombre d'opérations (+, -, *, /).

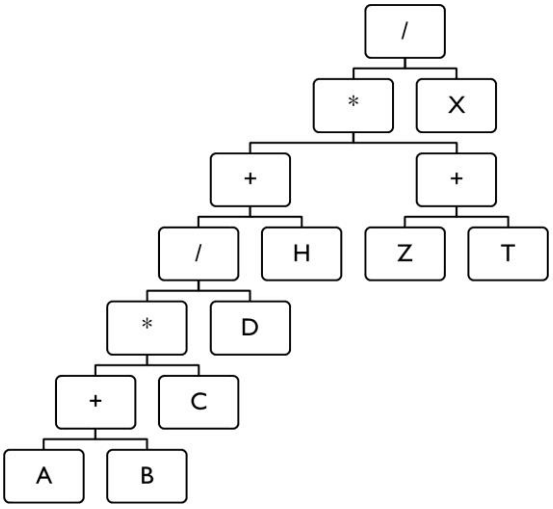
Solution :

La structure de données la plus adéquate est un arbre binaire, tel que les racines des sous arbres correspondent aux opérateurs et les feuilles représentent les opérandes. L'arbre associé à chaque expression arithmétique sera créé au moment de la lecture de cette expression (lors de la compilation) et se fait donc indépendamment de l'évaluation (qui a lieu au moment de l'exécution).

Ainsi l'évaluation d'une expression arithmétique prend en entrée l'arbre associé et retour en sortie le résultat de son exécution.

L'évaluation se fera en s'inspirant du parcours infixé dans un arbre binaire. Un opérateur est évaluer lorsque les opérandes correspondantes sont traitées, à partir de l'algorithme infixé cette évaluation aura lieu au moment du retour arrière. Au lieu d'afficher le contenu du sommet, l'opérateur qui correspond à la racine du sous arbre sera remplacé par le résultat de l'évaluation de la sous expression associé au sous arbre. Donc l'arbre devra évoluer au fil de l'évaluation jusqu'à ce qu'il devient une seule valeur

Illustration :



Algorithme :

```
Début
A:=Racine;  Empiler(A,P);
A:=A->fg;
Tant que (non Vide(P)) faire
Tant que (A ->fg <> null) Faire Empiler(A,P);
                                A:=A->fg;
                        Fait;
B := tete(P)->fd ;
Si (B->fg = NIL) alors R :=Evaluer (A,tete(P),B) ;
                                A :=Depiler(P) ; /* opération traitée */
                                tant que (Operation = vrai)
                                faire
                                Si Operateur (tete(P)) alors B := tete(P) -> fd ;
                                                        Si (B ->fg = Nil) alors
                                                        R :=Evaluer (R,tete(P),tete(P)->fd) ;
                                                        A:= Depiler(P);
                                                        Sinon Empiler(R);
                                                        A:= B; operation := faux ;
                                                        Fsi ;
                                R' := Depiler(P) ; /* opérande 1*/
                                R :=Evaluer (R',tete(P),R) ;
                                A:=Depiler(P);
                                fait;
Si (non_vide(P)) alors A := Tete(P)->fd; /* sommet frères ou
                        bien un encêtre fd*/
```

Bonne Chance !

```
                Sinon A:= null;
Fsi;
Fait;
Fait ;
Sinon Empiler (R, P) ; /* sauvegarder l'opérande 1 en attendant de trouver l'opérande
2*/
        A := B ;
Fin.
```

Bonne Chance !