Les arbres pour trier

Nour-Eddine Oussous, Éric Wegrzynowski

Licence ST-A, USTL - API2

9 décembre 2009

Objectifs

- Montrer l'utilité des arbres pour trier des données
- Deux tris exposés
  1. tri par construction d'un ABO
  2. tri par construction d'un maximier

Contexte

L'ensemble des données à trier peut être

- un tableau $T[1..N]$ d'éléments de type E
- une liste L d'éléments de type E

Dans la suite le type ENSEMBLE est soit

```
type ENSEMBLE = LISTE;
```

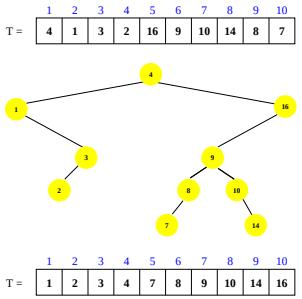
soit

```
type
  INDICE   = 1..N;
  ENSEMBLE = array[INDICE] of ELEMENT;
```

E est supposé totalement ordonné par une relation notée $\leq$.

Les tris considérés produisent un ensemble (liste ou tableau) trié dans l'ordre croissant.

Principe de l'algorithme



Les données à trier (ici un tableau)

1. Construire un ABO
2. Parcourir les nœuds de l'ABO dans l'ordre infixé

Construction de l'ABO

À partir d'un tableau (`ENSEMBLE=array[INDICE] of ELEMENT`)

```
function construitABO(T:ENSEMBLE):ARBRE;
var
  a : ARBRE;
  i : INDICE;
begin
  a := ARBREVIDE;
  for i:=low(INDICE) to high(INDICE) do
     insererABO(T[i],a);
  construitABO := a;
end {construitABO};
```


Construction de l'ABO

À partir d'une liste (`ENSEMBLE=LISTE`), version itérative

```
function construitABO(L:ENSEMBLE):ARBRE;
var
  a : ARBRE;
  L1 : ENSEMBLE;
begin
  a := ARBREVIDE;
  L1 := L;
  while not(estListeVide(L1)) do begin
     insererABO(tete(L1),a);
     L1 := reste(L1)
  end; {while}
  construitABO := a;
end {construitABO};
```


Obtention de l'ensemble trié

À partir d'une liste (`ENSEMBLE=LISTE`)

```
function parcoursABO(a : ARBRE) : ENSEMBLE;
var l1,l2,l3 : ENSEMBLE;
begin
  if estArbreVide(a) then
    parcoursABO := LISTEVIDE
  else begin
    l1 := parcoursABO(gauche(a));
    l2 := parcoursABO(droit(a));
    l3 := ajouteEnTete(racine(a),LISTEVIDE);
    concatener(l1,l3);
    concatener(l1,l2);
    parcoursABO := l1;
  end {if};
end {parcoursABO};
```

Pour obtenir une version triée de la liste L :

```
L1 := parcoursABO(construitABO(L));
```

Tri d'un tableau

À partir d'un tableau (ENSEMBLE=array[INDICE] of ELEMENT)

```
// parcoursABO(a,T,i) range dans l'ordre
// croissant les valeurs situées dans a dans
// le tableau T à partir de l'indice i
// À l'issue i est augmenté de la taille de a
procedure parcoursABO(const a : ARBRE;
                      var T : ENSEMBLE;
                      var i : CARDINAL);
begin
  parcoursABO(gauche(a),T,i);
  T[i] := racine(a);
  i := i+1;
  parcoursABO(droit(a),T,i);
end {parcoursABO};
```

Pour trier T :

```
  i := 0;
  parcoursABO(construitABO(T),T,i);
```


Coût en espace

1. Construction de l'ABO : allocation de N cellules
2. Production de l'ensemble trié à partir de l'ABO
   - aucune dans le cas d'un tableau
   - allocation de N nouvelles cellules dans le cas d'une liste

Coût en temps

1. Coût de la construction de l'ABO :
   - $\Theta(N^2)$ dans le pire des cas (se produit lorsque l'ABO construit est dégénéré)
   - $\Theta(N \log(N))$ dans le meilleur des cas
2. Coût du parcours : en $\Theta(N)$ dans tous les cas

Conclusion

- coût total en $\Theta(N^2)$ dans le pire des cas
- coût total en $\Theta(N \log(N))$ dans le meilleur des cas

Remarque

- Le tri par ABO est très proche d'un tri nommé tri rapide (ou quicksort).
- Le tri rapide des tableaux (bien programmé) ne consomme aucun espace mémoire supplémentaire,
- et a un temps d'exécution en $\Theta(N^2)$ dans le pire des cas, et en $\Theta(N \log(N))$ en moyenne.