Classes Préparatoires Ingénieurs - Première Année

Informatique - T.D. N^o 7

9 février 2004

Calcul de complexité

Exercice 1 Écrire l'algorithme qui recherche un élément dans un vecteur de taille n. Calculer la complexité temporelle en fonction du nombre de comparaisons dans le pire et dans le meilleur des cas. Refaire les calculs en fonction du nombre d'accès au vecteur.

FONCTION RechElem(V : VECTEUR de T, n : ENTIER, elem : T) : ENTIER
VAR i,p : ENTIER
  $i \leftarrow 1$
  $p \leftarrow 0$
  TANTQUE $i \leq n$ ET (p=0) **FAIRE**
    SI $V[i] = elem$ **ALORS**
      $p \leftarrow i$
    SINON
      $i \leftarrow i + 1$
    FIN SI
  FIN TANTQUE
RETOURNER p

Corrige 1 on compte le nombre de comparaison avec les éléments de vecteurs. Dans le meilleur de cas où on trouve l'élément à la position i, on effectue 1 comparaison. Dans le pire des cas où on trouve l'élément à la fin du vecteur ou on le trouve pas, on effectue n comparaisons. Une comparison coûte un accès au vecteur.

Exercice 2 Écrire l'algorithme de tri par selection. Calculer la complexité temporelle en fonction de nombre de comparaisons et de permutations dans le pire et dans le meilleur des cas. Refaire les calculs en fonction du nombre d'accès au vecteur.

La procédure *permut* effectue 4 accès au vecteur pour une permutation. La fonction *indmax* effectue toujours $n-1$ comparaisons, et par conséquent $2*(n-1)$ accès au vecteur. Dans la procédure *tripermutation* le nombre de comparaisons est donné par $nc = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2}$. Par contre le nombre de permutations est égale à 0 dans le cas favorable et à $n-1$ dans le cas défavorable.

```
PROCEDURE tripselection( ES V : VECTEUR de T, E n : ENTIER)
VAR i, ind : ENTIER
  i ← n
  TANTQUE i > 1 FAIRE
    ind ← indmax(V, i)
    SI ind ≠ i ALORS
      permut(V, i, ind)
    FIN SI
    i ← i − 1
  FIN TANTQUE
```

```
FONCTION indmax ( V : VECTEUR de T, n : ENTIER) : ENTIER
VAR i, p : ENTIER
  p ← 1
  i ← 2
  TANTQUE i ≤ n FAIRE
    SI V[i] ≥ V[p] ALORS
      p ← i

    FIN SI
    i ← i + 1
  FIN TANTQUE
RETOURNER p
```

```
PROCEDURE permut (ES V : VECTEUR de T, E i,j : ENTIER)
VAR tmp : T
  tmp ← V[i]
  V[i] ← V[j]
  V[j] ← tmp
```

Exercice 3 Écrire l'algorithme de tri à Bulles. Calculer la complexité temporelle en fonction de nombre de comparaisons et de permutations dans le pire et dans le meilleur des cas. Refaire les calculs en fonction du nombre d'accès au vecteur. Comparer avec le tri par selection.

PROCEDURE tribbulle1 (ES V : VECTEUR de T, E n :ENTIER)
VAR i,j : ENTIER
  $i \leftarrow n$
  TANTQUE $i > 1$ **FAIRE**
    $j \leftarrow 1$
    TANTQUE $j < i$ **FAIRE**
      SI $V[j+1] < V[j]$ **ALORS**
        $permut(V, j, j+1)$
      FIN SI
      $j \leftarrow j + 1$
    FIN TANTQUE
    $i \leftarrow i - 1$
  FIN TANTQUE

Le nombre de comparaisons est le suivant : $nc = (n-1)+(n-2)+..1 = \dfrac{n(n-1)}{2}$. Le nombre de permutations est égal à 0 dans le cas favorable et à $\dfrac{n(n-1)}{2}$dans le pire des cas.

Nous pouvons optimiser le tri à bulles en testant si nous avons effectué une ou plusieurs permutations ou non courant un itération donnée $(n-i)$. Si aucune permutation n'est effectuée à l'itération $n-i$, nous pouvons déduire que le vecteur V[1..i] est trié. Nous remarquons que cette optimisation peut influencer sur le nombre de comparisons qui peut être égal à $n-1$ dans le cas favorable.

Exercice 4 Calculer la complexité temporelle de l'algorithme de recherche dichotomique en fonction du nombre de comparaisons dans le pire et dans le meilleur des cas. Refaire les calculs en fonction du nombre d'accès au vecteur. Comparer avec l'algorithme de recherche séquentiel.

Corrigé 4 Dans le meilleur de cas, quand l'élément recherché se trouve au milieu du vecteur, l'algorithme coûte 1 comparaison (1 accès au vecteur). Dans le pire des cas, quand l'élément recherché n'est pas trouvé, l'algorithme effectue 2 comparaison chaque fois et ceci pour un nombre de fois qui est de l'ordre de $log_2(n)$, n étant la taille de vecteur.

```
PROCEDURE tribbulle2 (ES V : VECTEUR de T, E n :ENTIER)
VAR i,j : ENTIER
  permut : BOOLEEN
  i ← n
  permut ← VRAI
  TANTQUE permut FAIRE
    j ← 1
    permut ← FAUX
    TANTQUE j < i FAIRE
      SI V[j + 1] < V[j] ALORS
        permut(V,j,j+1)
        permut ← VRAI
      FIN SI
      j ← j + 1
    FIN TANTQUE
    i ← i − 1
  FIN TANTQUE
```

```
FONCTION dico( V : VECTEUR de T, n : ENTIER, elem : T) : BOOLEEN
VAR inf,sup,m : ENTIER
  trouve : BOOLEEN
  trouve ← FAUX
  inf ← 1
  sup ← n
  TANTQUE inf ≤ sup ET non trouve FAIRE
    m ← (inf + sup)/div2
    SI V[m] = elem ALORS
      trouve ← VRAI
    SINON
      SI V[m] < elem ALORS
        inf ← m + 1
      SINON
        sup ← m − 1
      FIN SI
    FIN SI
  FIN TANTQUE
RETOURNER trouve
```