1.1

Algorithmique et Structures de Données

Jean-Charles Régin

Licence Informatique 2ème année

1.2

Tris et Tas

Jean-Charles Régin

Licence Informatique 2ème année

Tris : avertissement

- Ce qui était vrai dans les années 70 ou 80 ne l'est plus maintenant car
  - On a beaucoup plus de mémoire
  - On a intérêt à être localisé en mémoire

Plan

- Tri par comptage
- Tri par base
- Tri par insertion
- Tri fusion
- Tri par sélection
- Tri par tas

Tri par comptage

- Quand on a des nombres de 1 à p on peut les trier facilement en comptant le nombre d'occurrence de chaque nombre.
  - On crée un tableau de p valeurs
  - On met toutes ces valeurs à 0
  - On traverse T. On compte le nombre de fois ou T[i] est pris = on incrémente P[T[i]]
  - Ensuite on balaie le tableau P et on copie autant de fois une valeur qu'elle apparait dans P

Tri par comptage


```
Booleen tri(T[],n)
min ← T[1]; max ← T[1]
pour i de 1 à n {
      si (T[i] < min){
            min ← T[i]
      }
      si (T[i] > max) {
            max ← T[i]
      }
}
pour k de 1 à (max-min+1) {
      P[k] ← 0
}
pour i de 1 à n {
      P[T[i]-min+1] ← P[T[i]-min+1] + 1
}
i ← 1
pour k de 1 à (max-min+1) {
      pour j de 1 à P[k]{
            T[i] ← k + min -1
            i ← i + 1
      }
}
```

Tri par comptage

- La complexité de ce tri est de O(n +p)
- Si p est de l'ordre de n alors le tri est linéaire, mais si p >> n ?
  - n=100 et p = 10 000 : c'est quadratique!
  - n=100 et p = 1 000 000 : c'est cubique !

Plan

- Tri par comptage
- **Tri par base**
- Tri par insertion
- Tri fusion
- Tri par sélection
- Tri par tas

Tri par base

- Le **tri par base** (ou tri radix ou radix sort) est un algorithme de tri rapide qui suppose que les éléments à trier sont des nombres ou des chaines de caractères
- Cet algorithme était utilisé pour trier des cartes perforées en plusieurs passages.

Tri par base

- On veut trier des cartes à jouer :
  - On fait 4 tas : un pour les ♣, un pour les ♦, un pour les ♥ et un pour les ♠
  - Puis on trie les cartes de chaque couleur par un tri par comptage

- On veut trier des noms de famille
  - Des idées ?

Tri par base

- L'ordre de tri est typiquement le suivant :
  - les clefs courtes viennent avant les clefs longues,
  - les clefs de même taille sont triées selon un ordre lexical.
- Cette méthode correspond à l'ordre naturel des nombres s'ils sont représentés par des chaînes de chiffres.
- Son mode opératoire est :
  - prend le chiffre (ou groupe de bits) le moins significatif de chaque clef,
  - trie la liste des éléments selon ce chiffre, mais conserve l'ordre des éléments ayant le même chiffre (ce qui est un tri stable).
  - répète le tri avec chaque chiffre plus significatif.

Tri par base

- Trier la liste : 170, 45, 75, 90, 2, 24, 802, 66
- tri par le chiffre le moins significatif (unités) :
- 170, 90, 2, 802, 24, 45, 75, 66
- tri par le chiffre suivant (dizaines) :
- 2, 802, 24, 45, 66, 170, 75, 90
- tri par le chiffre le plus significatif (centaines) :
- 2, 24, 45, 66, 75, 90, 170, 802

Tri par base

- Pour i de 1 à d

  Utiliser un tri stable pour trier le tableau T sur le digit i

- digit i dans la base b du nombre n
  - digit(i,b,n) = [n / $b^{(i-1)}$] modulo b
  - digit(2,10,324) = [324/10] modulo 10 =2

- Complexité ?

Plan

- Tri par comptage
- Tri par base
- **Tri par insertion**
- Tri fusion
- Tri par sélection
- Tri par tas

Tri par insertion

- Dans l'algorithme, on parcourt le tableau à trier du début à la fin. Au moment où on considère le *i*-ème élément, les éléments qui le précèdent sont déjà triés.
- L'objectif d'une étape est d'insérer le *i*-ème élément à sa place parmi ceux qui précèdent.
  - trouver où l'élément doit être inséré en le comparant aux autres,
  - décaler les éléments afin de pouvoir effectuer l'insertion.
- Le tri par insertion est
  - un tri stable (conservant l'ordre d'apparition des éléments égaux)
  - un tri en place (il n'utilise pas de tableau auxiliaire).

Tri par insertion


```
Tri_insertion(T[],n){
pour i de 2 à n {
    x ← T[i]
    j ← i
    tant que (j > 1 ET T[j-1] > x){
        T[j] ← T[j-1]
        j ← j - 1
    }
    T[j] ← x
}
```

Tri par insertion

- Voici les étapes de l'exécution du tri par insertion sur le tableau $T = [9,6,1,4,8]$. Le tableau est représenté au début et à la fin de chaque itération.
- $i = 2$ 96148 69148
- $i = 3$ 69148 16948
- $i = 4$ 16948 14698
- $i = 5$ 14698 14689

Tri par insertion

- Invariant : a l'étape i: Le tableau est une permutation et les i premiers éléments sont triés

Avantages

- Implémentation simple
- Efficace pour les petits ensemble de données
- Efficace pour les ensembles de données qui sont presque déjà triés : complexité $O(n + d)$, où d est le nombre d'inversions
- Le meilleur des cas est linéaire
- Stable
- En place
- Au vol (online) : on peut trier une liste comme on la reçoit

Inconvénients

- Pire des cas quadratique
- Fait beaucoup d'échanges (décalages)

- Complexité : si tableau presque trié alors très rapide
- Si une erreur : O(n)
- Si 2 erreurs ? 2.O(n)

Plan

- Tri par comptage
- Tri par base
- Tri par insertion
- **Tri fusion**
- Tri par sélection
- Tri par tas

Tri fusion

- L'algorithme peut être décrit récursivement :
- On découpe en deux parties à peu près égales les données à trier
- On trie les données de chaque partie
- On fusionne les deux parties
- La récursivité s'arrête car on finit par arriver à des listes composées d'un seul élément et le tri est alors trivial.

Tri fusion

- Complexité : O(n log(n))
  - Chaque niveau procède à des fusions dont le coût total est O(n)
  - La profondeur maximale est log(n)

Avantages

- Très bonne complexité
- Toujours au pire O(n log(n))
- Pénible à écrire pour des tableaux
- On peut utiliser d'autres tris par moment

Plan

- Tri par comptage
- Tri par base
- Tri par insertion
- Tri fusion
- **Tri par sélection**
- Tri par tas

Tri par Sélection

- Sur un tableau de *n* éléments (numérotés de 1 à *n*), le principe du tri par sélection est le suivant :
  - rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
  - rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 2 ;
  - continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Tri par Sélection


```
void tri_selection(T[], n) {
pour i de 1 à n - 1{
    min ← i
    pour j de i + 1 à n {
        si (T[j] < T[min]){
            min ← j
            si (min ≠ i) {
                échanger(T[i],T[min])
            }
        }
    }
}
```

Tri par Sélection

- L'invariant de boucle suivant permet de prouver la correction de l'algorithme :
  - à la fin de l'étape *i*, le tableau est une permutation du tableau initial et les *i* premiers éléments du tableau coïncident avec les *i* premiers éléments du tableau trié.

Tri par Sélection

- Tri sur place (les éléments sont triés dans la structure)
- Complexité
  - Dans tous les cas, pour trier *n* éléments, le tri par sélection effectue n(n-1)/2 comparaisons. Sa complexité est donc $O(n^2)$. De ce point de vue, il est inefficace. Il est même moins bon que le tri par insertion
- Par contre, le tri par sélection n'effectue que peu d'échanges :
  - *n*-1 échanges dans le pire cas, qui est atteint par exemple lorsqu'on trie la séquence 2,3,…,n,1 ;
  - en moyenne
  - aucun si l'entrée est déjà triée.
- Ce tri est donc intéressant lorsque les éléments sont aisément comparables, mais coûteux à déplacer dans la structure.

Plan

- Tri par comptage
- Tri par base
- Tri par insertion
- Tri fusion
- Tri par sélection
- **Tri par tas**

Tri par Sélection et Tri par Tas

- Le tri par tas fonctionne comme le tri par sélection mais utilise une structure de données particulière pour accélérer la recherche : **un tas**

Tas : structure de données

- Un tas descendant est un arbre binaire vérifiant les propriétés suivantes :
  - la différence maximale de profondeur entre deux feuilles est de 1 (i.e. toutes les feuilles se trouvent sur la dernière ou sur l'avant-dernière ligne) ;
  - les feuilles de profondeur maximale sont "tassées" sur la gauche.
  - chaque noeud est de valeur inférieure à celle de ces deux fils.

- Un tas ou un arbre binaire presque complet peut être stocké dans un tableau, en posant que les deux descendants de l'élément d'indice n sont les éléments d'indices 2n et 2n+1 (pour un tableau indicé à partir de 1).
- En d'autres termes, les noeuds de l'arbre sont placés dans le tableau ligne par ligne, chaque ligne étant décrite de gauche à droite.

Tas : structure de données

- Un tas est **descendant** si
  - chaque noeud est de valeur **inférieure** à celle de ces deux fils.
- Propriétés
  - Chaque chemin est croissant
  - Le père est le minimum
- Un tas est **ascendant** si
- chaque noeud est de valeur **supérieure** à celle de ces deux fils.
- Propriétés
  - Chaque chemin est décroissant
  - Le père est le maximum

Tas : insertion

- L'insertion d'un élément dans un tas se fait de la façon suivante :
  - on place l'élément sur la première case libre
  - on échange l'élément et son père tant que ce dernier est supérieur et qu'il existe.

Tas : insertion


```
inserer(T[],n,d,elt){
    T[d] ← elt
    père ← d/2
    fils ← d
    tant que(père > 0 ET T[père]>T[fils]){
        échanger(T[père],T[fils])
        fils ← père
        père ← père/2
    }
}
```

Tas : extraction

- L'opération de **tamisage** consiste à échanger la racine avec le plus petit de ses fils (si elle est plus grande), et ainsi de suite récursivement jusqu'à ce qu'elle soit à sa place.

- L'extraction consiste à
  - Supprimer la racine
  - Mettre à sa place le dernier élément du tas
  - Tamiser le tas

Tas : tamisage et extraction


```
tamiser(T[],n){
    père ←1
    fils ← 2*père
    fini ← faux
    tant que (fils ≤ n ET non fini){
        si (fils<n ET T[fils+1]< T[fils]){
            fils ← fils+1
        }
        si (T[père] > T[fils]){
            échanger(T[père],T[fils])
            père ← fils
            fils ← 2*père
        } sinon {
            fini ← vrai
        }
    }
}
```

Tas : tamisage et extraction


```
entier extraire(T[],n){
    res ← T[1]
    échanger (T[1],T[n])
    tamiser(T,n-1)
    retourner res
}
```

Tri par tas

- On insère tous les éléments dans le tableau
- On extrait successivement toutes les racines

Tas

- Le tri par tas a certains défauts mais la structure de tas est très pratique pour répondre à d'autres problèmes