Cours 2. Partie 2: Introduction à l'algorithmique. Dichotomie. Un peu de tris

Olivier Bournez

bournez@lix.polytechnique.fr
LIX, Ecole Polytechnique

Aujourd'hui

Recherche dans un tableau

Dichotomie

Trier

Tri par sélection

Tri à bulles

Tri par insertion

Tri par fusion

Rechercher un élement

- Le problème *RECHERCHE*
  - On se donne une liste d'entiers naturels $T[1], T[2], \cdots, T[n]$.
  - On se donne x un entier.
  - On doit déterminer si $x = T[i]$ pour un certain entier i.

```
int [] T= new int[100];
int x;
int r = Dans(T,x);

...

int Dans(int [] T, int x) {
 for (int j=0; j< T.length; j++) {
      if (T[j] == x)
        return true;}
 return false;
}
```

- Nombre de comparaisons (hors variable de boucle)
  $1 \leq C(n) \leq n$.

  $n =$ nombre d'entiers dans le tableau.

Bornes extrêmes

- $C(n) = 1$ pour un tableau qui débute par x.
- $C(n) = n$ pour un tableau qui ne contient pas x.

- $C(n) \in O(n)$.

- En moyenne:
  - si on suppose que le tableau contient une permutation de $1, 2, \cdots, n$, et que chaque permutation est choisie de façon uniforme, en moyenne $\frac{n+1}{2}$.

Digression: pourquoi? calcul sur les moyennes

- Si on suppose que les entrées sont les listes permutations de $\{1, 2, \cdots, n\}$, et qu'elles sont équiprobables,

$$\begin{aligned} Complexite - Moyenne_{\mathcal{A}}(n) &= Esperance_{d/taille(d)=n}[k] \\ &= \sum_{i=1}^{n} i \times P(k = i) \end{aligned}$$

- Puisque les permutations sont équiprobables, $Proba(k = i) = 1/n$.

$$\begin{aligned} Complexite - Moyenne_{\mathcal{A}}(n) &= \sum_{i=1}^{n} i \frac{1}{n} \\ &= \frac{n(n+1)}{2n} \\ &= \frac{n+1}{2} \end{aligned}$$

- (joli exercice) Pour l'algorithme itératif pour MAX, $Complexite - Moyenne_{\mathcal{A}}(n)$ en affectations entre variables entières est en $\Theta(\log n)$ (de l'ordre de $H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$).

Optimal?

- A tout algorithme qui fonctionne par comparaisons, on peut associer un arbre de décision.
- Si cet arbre possède moins de n sommets, on peut trouver un tableau T sur lequel l'algorithme répond de façon non correcte.
- Il faut donc au moins n comparaisons.... (sans autres hypothèses sur les données).

Aujourd'hui

Recherche dans un tableau

Dichotomie

Trier

Tri par sélection

Tri à bulles

Tri par insertion

Tri par fusion

Rechercher un élement dans un tableau trié

- Le problème *RECHERCHE*
  - On se donne une liste d'entiers naturels **TRIEE**

$$T[0] \leq T[2] \leq \cdots \leq T[n-1].$$

  - On se donne x un entier.
  - On doit déterminer si $x = T[i]$ pour un certain entier i.

Le jeu préféré de Pierrick

- *Papa* choisit un entier $min \leq i < max$ qu'il garde secret. Par exemple, $min = 0, max = 16$.
- *Pierrick* pose des questions du type "est-ce que $i \geq d$" pour un certain entier d.
- *Pierrick* a gagné lorsqu'il a deviné i.

- Il y a t'il une stratégie
  - gagnante?
  - optimale?
    - Combien de question est il nécessaire dans le pire des cas?

$$16 = 2 * 2 * 2 * 2 = 2^4.$$

$$128 = 2 * 2 * 2 * 2 * 2 * 2 * 2 = 2^7.$$

Stratégie de la dichotomie

- Stratégie de la dichotomie:
  - prendre systématiquement $d = (min + max)/2$.
  - actualiser min et max.

Retour sur recherche: arbre des possibilités

- Dessin de l'algorithme naif.
- Dessin de l'arbre de décision pour la dichotomie pour par exemple $n = 16$.
- Remarque:
  - Tout algorithme qui fonctionne par comparaisons peut se représenter par un arbre de décision.

Recherche par dichotomie: version récursive

```
static boolean trouve(int[] T, int v, int min, int max){
    if(min ≥ max) // vide
        return false;
    int mid = (min + max) / 2;
    if (T[mid] == v) return true;
    else if (T[mid] > v) return trouve(T, v, min, mid);
    else return trouve(T, v, mid + 1, max);
}
```

Version non-récursive

- Elimination d'une récursion terminale...
  - aucun traitement à exécuter après chacun des appels qu'elle contient.

```
static boolean trouve(int[] T, int v, int min, int max){
     while (min < max) {
      int mid = (min + max) / 2;
      if (T[mid] == v) return true;
      else if (T[mid] > v) max=mid;
      else min=mid+1;
   }
  }
```

Cet algorithme fonctionne

- Il faut prouver que:

  1. la procédure termine toujours
  2. s'il existe un entier i avec $T[i] = v$, alors l'algorithme répond vrai.
  3. si l'algorithme répond vrai, alors il y a un entier i avec $T[i] = v$.

- Dernier point évident, car quand l'algorithme répond vrai, cette propriété est vraie.
- Pour le premier point, on montre que la suite $max - min$ décroit strictement.
- Pour le second point, preuve par récurrence sur k, le numéro de l'étape.

Analyse

- Complexité:
  - Si $2^{k-1} \leq n < 2^k$, une recherche fructueuse utilise entre 1 et $2k$ comparaisons.
  - Nombre de comparaisons est donc en $O(\lfloor \log_2 n \rfloor) = O(\log_2 n)$.

- Optimal:
  - à tout algorithme par comparaison, on peut associer un arbre:
    - Il doit avoir au moins n sommets.
    - Sa hauteur est donc au moins de l'ordre de $\lfloor \log_2 n \rfloor$.
    - Il a donc au moins une exécution avec $\lfloor \log_2 n \rfloor$ instructions.
    - Il ne peut donc pas être de complexité mieux que $\lfloor \log_2 n \rfloor = O(\log_2 n)$.

Aujourd'hui

Recherche dans un tableau

Dichotomie

Trier

Tri par sélection

Tri à bulles

Tri par insertion

Tri par fusion

Pourquoi trier?

- Trier est une opération naturelle.
- Travailler sur des données triées est parfois plus efficace.
  - Exemple:
    - Rechercher un élément dans un tableau à n éléments: $O(n)$.
    - Rechercher un élément dans un tableau trié à n éléments: $O(\log n)$ (par dichotomie).

Pourquoi trier?

- Trier est une opération naturelle.
- Travailler sur des données triées est parfois plus efficace.
  - Exemple:
    - Rechercher un élément dans un tableau à n éléments: $O(n)$.
    - Rechercher un élément dans un tableau trié à n éléments: $O(\log n)$ (par dichotomie).
    - Trier devient intéressant dès que le nombre de recherches est en $\Omega(Complexite(tri)/n)$.

Aujourd'hui

Recherche dans un tableau

Dichotomie

Trier

Tri par sélection

Tri à bulles

Tri par insertion

Tri par fusion

Principe

- Sur un tableau de n éléments (numérotés de 0 à $n-1$), le principe du tri par sélection est le suivant :
  - rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0;
  - rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
  - continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

```
procédure tri_selection(tableau t, entier n)
    pour i de 0 à n - 2
        min := i
        pour j de i + 1 à n - 1
            si t[j] < t[min], alors min := j
        si min <> i, alors échanger t[i] et t[min]
```

Correct

- Invariant de boucle: “ à la fin de l'étape i,
  1. le tableau est une permutation du tableau initial
  2. et les i premiers éléments du tableau coïncident avec les i premiers éléments du tableau trié.

  ”

Complexité

- Effectue
  $(n-1)+(n-2)+\cdots+1 = 1+2+\cdots+(n-1) = n(n-1)/2$
  comparaisons.
- Soit en $O(n^2)$.
- Non-optimal.
- Intérêt: peu d'échanges :
  - $n-1$ échanges dans le pire cas, qui est atteint par exemple lorsqu'on trie la séquence 2,3,. . . ,n,1 ;
    $n-(1/2+\cdots+1/n) \simeq n-\ln n$ en moyenne, c'est-à-dire si les éléments sont deux à deux distincts et que toutes leurs permutations sont équiprobables (en effet, l'espérance du nombre d'échanges à l'étape i est $(n-i)/(n-i+1)$);
  - 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.

Aujourd'hui

Recherche dans un tableau

Dichotomie

Trier

Tri par sélection

Tri à bulles

Tri par insertion

Tri par fusion

Tri à bulles

1. L'idée consiste à comparer deux éléments adjacents, et à les échanger s'ils ne sont pas dans l'ordre.
2. Lorsqu'on réalise un passage, on range le plus grand élément en fin de tableau.
3. Effectuer des passages jusqu'à ce que le tableau soit entièrement trié.

```
int n=t.length;
for (int i=n-1; i>=0; --i)
for (int j=0; j<i; ++j)
        if (t[j+1] < t[j]) { // comparer deux voisins
    // les echanger si nécessaire
    int tmp = t[j];
    t[j] = t[j+1];
    t[j+1] = tmp;
}
```

Aujourd'hui

Recherche dans un tableau

Dichotomie

Trier

Tri par sélection

Tri à bulles

Tri par insertion

Tri par fusion

Trier par insertion

- Méthode souvent utilisée pour trier un jeu de cartes.
- Etape préliminaire: savoir insérer un élément x dans une liste triée $T[0] \leq T[2] \leq \cdots \leq T[n-1]$.

Trier par insertion: version itérative

- L'objectif d'une étape est d'insérer le i-ème élément à sa place parmi ceux qui précèdent.
  - Il faut pour cela trouver où l'élément doit être inséré en le comparant aux autres, puis décaler les éléments afin de pouvoir effectuer l'insertion.
  - En pratique, ces deux actions sont fréquemment effectuées en une passe, qui consiste à faire "descendre l'élément au fur et à mesure jusqu'à rencontrer un élément plus petit.

```
procédure tri_insertion(tableau T, entier n)
      pour i de 1 à n - 1
          x := T[i]
          j := i-1
          tant que j > 0 et T[j] > x
              T[j+1] := T[j]
              j := j - 1;
          T[j] := x
```

Trier par insertion: version récursive

- Equations récursives:

$$\begin{array}{lcll} \mathit{Insere}(x, \emptyset) & = & (x, \emptyset) & \\ \mathit{Insere}(x, (a, L)) & = & (x, (a, L)) & \text{si } x \leq a \\ \mathit{Insere}(x, (a, L)) & = & (a, \mathit{Insere}(x, L)) & \text{sinon} \end{array}$$

- Correction: par induction, si on suppose (a, L) triée, le résultat est bien trié.

```
static Liste Insere(int x, Liste a) {
  if (a==null) return new Liste(x,null);
  if (x ≤ a.contenu) return new Liste(x,a);
  return new Liste(a.contenu,Insere(x,a.suivant));
}
```

Tri par insertion

```
static Liste Trie(Liste p) {
  Liste r = null;
  for (; p≠ null; p = p.suivant)
    r = Insere(p.contenu,r);
  return r;}
```

- Complexité $I(n)$ de `Insere` en comparaisons : $I(n) = O(n)$.
- Complexité $C(n)$ de `Trie` en comparaisons: $C(n) \leq nI(n) = O(n^2)$.
- Le pire cas est atteint lorsqu'on trie une liste déjà triée, et mène à $\Omega(n^2)$ comparaisons. La complexité du tri par insertion est donc bien en $\Theta(n^2)$.

Tri par insertion en moyenne

- Le meilleur cas est atteint lorsqu'on trie une liste dans l'ordre décroissant, et mène à $O(n)$ comparaisons.
- $O(n) \leq C(n) \leq O(n^2)$.
- En moyenne? Si on se restreint au tri des listes d'entiers distincts deux à deux, et en supposant les permutations équiprobables,

$$\begin{aligned} C(n) &= \tfrac{1}{4}n^2 + \tfrac{3}{4}n - \ln n + O(1). \\ &= \Theta(n^2) \end{aligned}$$

(Supplément) Preuve

Principe:

- Coût moyen de `Insere` avec déjà $k-1$ éléments:

$$I(k) = \frac{1}{k}(1+2+\cdots+k-1+k-1) = \frac{1}{k}(\frac{k(k+1)}{2}-1)$$

(position de l'élément inséré équiprobable)

- Coût moyen de `Trie`:

$$C(n) = 0 + \frac{1}{2}(\frac{2(2+1)}{2}-1) + \cdots \frac{1}{n}(\frac{n(n+1)}{2}-1)$$

(k premiers éléments répartis uniformément)

Aujourd'hui

Recherche dans un tableau

Dichotomie

Trier

Tri par sélection

Tri à bulles

Tri par insertion

Tri par fusion

Tri par fusion

- Fusion:
  - Construire une liste triée qui contienne l'union des éléments de deux listes triées.

- Exemple: $\mathit{Fusion}(1.4.5.7, 2.4.9) = 1.2.4.4.5.7.9$

Tri fusion

- Une liste de 0 ou 1 élément est triée.
- Toute autre liste L
  - peut se découper en deux sous-listes L_1 et L_2 de même taille (à 1 près).
  - Les sous-listes L_1 et L_2 sont triées récursivement,
  - puis fusionnées.

$$TriFusion(L) = Fusion(TriFusion(L_1), TriFusion(L_2)).$$

Paradigme “Diviser pour règner”

- Complexité $C(n)$ en nombre de comparaisons en $O(\mathit{Fusion}) + 2C(n/2))$, soit

$$C(n) = O(n + 2C(n/2)),$$

ce qui mène à

$$C(n) = O(n \log n).$$