Chapitre 1
Les tableaux

Michaël Krajecki

Université de Reims Champagne-Ardenne
michael.krajecki@univ-reims.fr
http://www.univ-reims.fr/crestic

Graphes et algorithmes

Plan

- Les tableaux
  - les algorithmes de base
  - les algorithmes de tri : tris quadratiques et efficaces
- La récursivité et les listes
- Les arbres
  - terminologie de base
  - arbres binaires
  - arbres équilibrés
- Les graphes
  - les graphes orientés et non orientés
  - parcours de graphes
  - le plus court chemin
  - compostantes fortement connexes
  - arbre recouvrant de poids minimum

Définitions

Le problème principal de l'utilisateur :

- décrire la suite des actions élémentaires
- à partir des données fournies
- afin de produire les résultats attendus.

Cette marche à suivre porte le nom d'algorithme.

Définition (Algorithmique)

c'est l'art de décomposer un problème en évènements simples.

Définitions

Définition (Algorithmique, Encyclopaedia Universalis)

Un algorithme est une suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver, en un nombre fini d'étapes, à un certain résultat, et cela indépendamment des données.

Le mot algorithme provient du nom du mathématicien arabe *Muhammad ibn Musa al Khwarizmi* (ixème siècle).

Algorithme

Le rôle de l'algorithme est fondamental.

Sans algorithme, il n'y aurait pas de programme :

- le programme n'est que sa traduction dans un langage compréhensible par l'ordinateur.

Ils sont indépendants à la fois :

- de l'ordinateur qui les exécute ;
- des langages dans lequel ils sont énoncés et traduits ;

Définition d'un tableau

Un tableau est une suite d'éléments de même type stockés dans des blocs de mémoire contiguë. Il est possible d'accéder aléatoirement (dans un ordre quelconque) à n'importe quel élément d'un tableau.

Définition (tableau à une dimension)

Soient :

- *un ensemble quelconque de valeurs de même type noté E ;*
- *un entier n appelé nombre d'éléments du tableau.*

Un vecteur ou tableau est une application V d'un intervalle I de $\mathbb{N}$. I est appelé ensemble des indices.

Recherche dans un tableau trié

Définition (tableau trié)

un tableau $t[0..n-1]$ *est trié par ordre croissant si :*

$$\forall i \in [0, n-1] \text{ et } i < j < n,\ t[i] \leq t[j].$$

Il est possible d'améliorer la recherche séquentielle d'un élément, si le tableau est trié. $\rightsquigarrow$ si on recherche v dans un tableau trié

$t[0..n-1]$ et $v \notin t[0..i],\ t[i] > v$ alors il est inutile de parcourir le reste du tableau car $t[i+j] \geq t[i] > v$ pour tout j tel que $i+j < n$.

Recherche dichotomique

Le principe de cet algorithme est le suivant :

- l'algorithme considère un élément *pivot* du tableau : p ;
- il compare $t[p]$ à v, 3 cas sont possibles :
  1. $t[p] = v$, l'algorithme termine.
  2. $t[p] > v$, ce qui signifie que $v \notin t[p+1..n-1]$.
  3. $t[p] < v$, l'algorithme en déduit que $v \notin t[0..p-1]$.
- Le principe est appliqué à nouveau sur la partie du tableau qui peut contenir v.

Recherche dichotomique

Fonction *Dichotomie(t : tableau d'entiers, n : entier, v : entier) : entier.*
variable *: p, binf, bsup : entier, trouve : booléen*
Début
  trouve $\leftarrow$ *Faux ; binf* $\leftarrow$ *0 ; bsup* $\leftarrow$ *n-1 ; p* $\leftarrow$ *(binf+bsup)/2*
  Tant que *non trouve et binf* $\leq$ *bsup* **faire**
    Si *t[p]=v* **alors** *trouve* $\leftarrow$ *Vrai*
    Sinon si *t[p]<v* **alors** *binf* $\leftarrow$ *p+1*
    Sinon *bsup* $\leftarrow$ *p-1*
    Fsi
    p $\leftarrow$ *(binf+bsup)/2*
  Ftant
  Si *trouve* **alors retourner** *p*
  Sinon retourner *n* **Fsi**
Fin.

Coût de la recherche dichotomique

Recherche séquentielle : en moyenne on accède à la moitié des éléments $O(n)$.

- on élimine à chaque étape la moitié du tableau ;
- on utilise au mieux les deux informations suivantes :
  1. Le tableau est trié.
  2. L'accès direct offert par les tableaux.
- En moyenne, la recherche d'une valeur dans un tableau de taille n réalise $\log n$ accès à la structure de données : $O(\log n)$.

Pourquoi trier ?

Pour pouvoir réaliser une recherche efficace d'un élément dans un tableau :

- il est nécessaire que ce tableau soit trié ;
- il existe de nombreuses méthodes de tri plus ou moins efficaces ;
- les différents critères à considérer sont le nombre d'accès au tableau et également la place mémoire nécessaire pour mettre en œuvre la méthode analysée.

Plan

1 Introduction

2 Définitions

3 Recherche dichotomique

4 Les tris
- Insertion dans un tableau trié
- Tris quadratiques
- Les tris efficaces

Insertion dans un tableau trié

Quand un tableau est déjà trié, il faut être en mesure d'insérer un nouvel élément dans ce tableau tout en maintenant cette propriété.

Pour réaliser cette opération, il suffit de parcourir le tableau de droite à gauche jusqu'à trouver la place où doit être ajoutée la valeur (en supposant que le tableau est trié par ordre croissant).

$\rightsquigarrow$ Il faut insérer la valeur v après le premier élément du tableau qui lui est inférieur ou égal.

Insertion dans un tableau trié

Fonction *Insertion(t : tableau d'entiers, n : entier, v : entier) : entier.*
variable *: i : entier, trouve : booléen*
Début
  trouve ← *Faux ; i* ← *n-1*
  Tant que *non trouve et i>0* **faire**
    Si *t[i]<v* **alors** *trouve* ← *Vrai*
    Sinon *t[i+1]* ← *t[i] ; i* ← *i-1*
    Fsi
  Ftant
  Si *trouve* **alors** *t[i+1]* ← *v*
  Sinon *t[0]* ← *v* **Fsi**
  retourner *n+1*
Fin.

Plan

1 Introduction

2 Définitions

3 Recherche dichotomique

4 Les tris
- Insertion dans un tableau trié
- Tris quadratiques
- Les tris efficaces

Tri par insertion

Principe :

- un tableau ne comportant qu'une valeur est trié
- nous avons défini une fonction qui insére un élément dans un tableau trié
- Donc, si nous possédons un tableau de n valeurs, dont les $n-1$ premières sont triées, il suffit d'insérer la dernière valeur de façon triée...

Voici une première procédure qui utilise 2 tableaux

Tri par insertion

Procédure *TriInsertion(t : tableau d'entiers, n : entier, tt : tableau d'entiers).*
variable *: i : entier*
Début
    Pour *i* **de** *0* **à** *n-1* **faire**
        Insertion(tt,i, t[i])
    Fpour
Fin.

Il est possible de se passer du tableau *tt*, c'est à dire de trier *sur place*
Il faut alors insérer le deuxième élément du tableau, de façon triée, dans le tableau comportant le premier élement est ainsi de suite..

Tri par insertion

Procédure *TriInsertionSurPlace(t : tableau d'entiers, n : entier.*
variable *: i,j,cle : entier*
Début
  Pour *j* **de** *1* **à** *n-1* **faire**
    cle $\leftarrow$ *t[j]*
    {Insertion de t[j] dans t[0,j-1] }
    i $\leftarrow$ *j-1*
    tant que *i>=0* ***et*** *t[i]>cle* ***faire***
      t[i+1] $\leftarrow$ *t[i]*
      i $\leftarrow$ *i-1*
    Ftant
    t[i+1] $\leftarrow$ *cle*
  Fpour
Fin.

Analyse du tri par insertion

- Compter le nombre d'itérations des 2 boucles
- La boucle externe (pour) : $n-1$ itérations
- La boucle interne (tant que) : $test_j$ itérations, ce nombre est variable pour chaque itérations
- Au total, le nombre d'actions exécutées est de l'ordre de :

$$\sum_{j=1}^{n-1} test_j$$

- Calcul des $test_j$ dans le *le meilleur des cas* et dans le *pire des cas*

Analyse du tri par insertion

- Dans le cas le plus favorable :
  - Le tableau est trié
  - $test_j = 1$
  - $\sum_{j=1}^{n-1} test_j = \sum_{j=1}^{n-1} 1 = n - 1$
  - Le coût d'exécution est *linéaire*
- Dans le cas le plus défavorable :
  - Le tableau est trié par ordre décroissant
  - $test_j = j$
  - $\sum_{j=1}^{n-1} test_j = \sum_{j=1}^{n-1} j$
  - Suite arithmétique de raison 1
  - Le coût d'exécution est alors *quadratique*

Analyse dans le pire des cas

- Pourquoi retenir, en général, l'analyse dans le pire des cas ?

  1. Le temps d'exécution dans le pire des cas constitue *une borne supérieure*
  2. Le pire des cas se présente souvent (exemple : recherche d'une valeur qui n'est pas dans le tableau)
  3. Le cas moyen est souvent proche du pire des cas :
     - coût de l'insertion d'une valeur dans un tableau rempli au hasard
     - en moyenne égale à $\frac{n}{2}$
     - donc, $test_j = \frac{j}{2}$
     - le cas moyen reste ici quadratique

Tri par sélection

Le principe de cette méthode est très simple :

1. l'algorithme recherche le minimum du tableau à trier.
2. Cette valeur est insérée au début d'un nouveau tableau.
3. On remplace cet élément par un majorant du tableau.
4. Puis, ce principe est appliqué pour tous les éléments restants.

Voici un exemple d'exécution du tri par sélection...

Exemple d'exécution du tri par sélection

10	15	2	11
10	15	#	11
#	15	#	11
#	15	#	#
#	#	#	#

2			
2	10		
2	10	11	
2	10	11	15

Tri par sélection

Procédure *Selection(t : tableau d'entiers, n : entier, tt : tableau d'entiers).*
variable *: i, m, majorant : entier*
Début
    majorant ← *Max(t,n) // recherche du maximum*
    Pour *i* **de** *0* **à** *n-1* **faire**
        m ← *IndMin(t,n) // recherche de l'indice du minimum*
        tt[i] ← *t[m]*
        t[m] ← *majorant*
    Fpour
Fin.

Tri par sélection

Remarques :

- Le tableau trié est contenu dans le tableau `tt` ;
- *il est possible d'utiliser qu'un seul tableau*.

Il suffit alors d'inverser deux éléments du tableau :

- la valeur minimum et la valeur contenu à la position $t[i]$ au $i$ème tour ;
- dans ce cas, la recherche du minimum s'effectue sur la partie non triée du tableau (sur $t[i..n-1]$).

Analyse du tri par sélection

Ce tri n'est pas très efficace :

- à l'étape i, le tri par sélection effectue $n - i$ accès au tableau pour calculer le minimum dans le meilleur des cas (en utilisant un seul tableau) ;
- donc pour trier la totalité du tableau, cette méthode accède $1 + 2 + 3 + .. + n = \frac{n(n+1)}{2}$ fois au tableau en lecture.

Tri à bulles

L'état du tableau après $n - i$ itérations est le suivant :

- la partie $t[1..i]$ n'est pas triée ;
- la partie $t[i+1..n]$ est triée ;
- $t[1..i] \leq t[i+1]$.

Le but du tri à bulles est d'amener le maximum de la partie non triée du tableau ($t[1..i]$) en $i$ème position.

Voici la procédure qui applique ce principe...

Tri à bulles

```
Procédure Bulles(t : tableau de réels, n : entier).
variable : i,j : entier, aux : réel
Début
  pour i de 0 à n-1 faire
    pour j de 0 à n-i faire
      Si t[j]>t[j+1] alors
        aux ← t[j+1]
        t[j+1] ← t[j]
        t[j] ← aux
      Fsi
    Fpour
  Fpour
Fin.
```

Plan

1 Introduction

2 Définitions

3 Recherche dichotomique

4 Les tris
- Insertion dans un tableau trié
- Tris quadratiques
- Les tris efficaces

Le tri par fusion

- Le principe mis en œuvre :
  - 1 tableau comportant 1 valeur est trié
  - Il est possible de fusionner deux tableaux triés en un temps linéaire
  - On applique récursivement ce principe jusqu'à obtenir un seul tableau

- Cet algorithme s'appuye sur le principe *diviser pour régner*

Le principe diviser pour régner

- Approche classique en informatique
- Se décline en 3 étapes :
  1. **Diviser** le problème initial en un ensemble de sous problèmes
  2. **Régner** sur chaque sous problème (les résoudre)
  3. **Combiner** les sous problèmes pour résoudre le problème initial

Diviser pour régner

- Application du principe dans le cas du tri par fusion
- **Diviser** le tableau de n valeurs en deux sous tableaux de longueur $\frac{n}{2}$
- **Régner** en triant récursivement les deux tableaux
- **Combiner** les deux tableaux en les fusionnant

La procédure Tri par fusion

```
Procédure TriFusion(t : tableau d'entiers, d,f : entier).
variable : p : entier
Début
    Si d<f alors
        p ← (d+f)/2
        TriFusion(t,d,p)
        TriFusion(t,p+1,f)
        Fusionner(t,d,p,f)
    Fsi
Fin.
```

Au départ, on appelle *TriFusion(t,0,n-1)* avec *n* le nombre de valeurs.

L'utilisation de la récursivité peut être pénalisante

Analyse du tri par fusion

- Il ne reste plus qu'à écrire la procédure *Fusionner()*
- Complexité linéaire
- Les appels récursifs :
  - à chaque appel, la taille des données est divisée par 2
  - de l'ordre de $\log n$ appels
- Complexité du tri par fusion : $O(n \log n)$

Le tris par tas

- Le tri par tas a une complexité en *n log n*, comme le tris par fusion
- Il trie sur place, comme le tri par insertion
- *A priori*, il offre le meilleur des deux solutions précédentes
- Pour cela, il s'appuie sur la notion de *tas*

Le tas

- Un tas est une structure de données qui peut être vue comme :
  - un arbre binaire presque complet
  - un tableau
- L'arbre est complètement rempli, sauf sur le bas
- Chaque nœud est numéroté
- Les trois propriétés suivantes sont toujours vraies :
  1. $Pere(i) = i/2$
  2. $FGauche(i) = 2i$
  3. $FDroit(i) = 2i+1$

La propriété des tas

- La valeur d'un nœud est au plus égale à celle de son père
- $tas[pere(i)] \geq tas[i]$
- Donc, la racine d'un tas est forcément le plus grand élément
- La hauteur d'un nœud : définie par la longueur du plus long chemin jusqu'à une feuille
- La hauteur d'un arbre : définie par la hauteur de la racine
- Comme un tas est un arbre binaire presque complet, sa hauteur est de l'ordre de $log\ n$

Les opérations élémentaires sur les tas

- Nous allons définir 3 procédures sur les tas
- *Entasser()* : permet de rétablir la propriété des tas sur un nœud
- *ConstruireTas()* : construit, en un temps linéaire, un tas à partir d'un tableau non trié
- *TriParTas()* : tri un tableau en $O(n \log n)$

La procédure Entasser()

- Procédure centrale dans la gestion des tris
- À l'appel, on suppose que *FGauche(i)* et *FDroit(i)* sont des tas
- Mais, le nœud *i* peut avoir une valeur plus petite
- Objectif : rétablir la propriété des tas

La procédure Entasser()

```
Procédure Entasser(t : tas, i : noeud).
variable : d,g : tas, aux : entier
Début
    g ← FGauche(i)
    d ← FDroit(i)
    Si g < t.taille et t[g] > t[i] alors
      max ← g
    Sinon max ← i
    Fsi
    Si d < t.taille et t[d] > t[max] alors
      max ← d
    Fsi
    Si max ≠ i alors
      aux ← t[i]
      t[i] ← t[max]
      t[max] ← aux
      Entasser(t,max)
    Fsi
Fin.
```

La procédure *ConstruireTas*

- Objectif : construire un tas à partir d'un tableau non ordonné
- Cette procédure s'appuie sur la procédure précédente
- Considérons les éléments *t[n/2..n-1]* :
- Ce sont des feuilles, donc des tas
- Il suffit de parcourir les autres éléments et d'appeler la procédure *Entasser* pour construire un tas complet
- L'exécution en temps linéaire est possible (à démontrer)...

La procédure ConstruireTas()

Procédure *Construire(t : tas).*
variable *: i : entier*
Début
    Pour *i* **de** *t.taille/2-1* **à** *0* **faire**
      Entasser(t,i)
    Fpour
Fin.

TriParTas()

- Principe du tri par tas :
- Construire un tas à partir du tableau à trier
- t[0] est alors le maximum du tableau
- Il suffit alors de le placer en dernière position (par un échange)
- Et de décrementer la taille du tas
- Ce principe est appliqué à nouveau...

La procédure TriParTas()

Procédure *TriParTas(t : tas).*
variable *: i,aux : entier*
Début
    ConstruireTas(t)
    Pour *i* **de** *t.taille-1* **à** *1* **faire**
      aux $\leftarrow$ *t[0]*
      t[0] $\leftarrow$ *t[i]*
      t[i] $\leftarrow$ *aux*
      t.taille $\leftarrow$ *t.taille - 1*
      Entasser(t,0)
    Fpour
Fin.

Complexité en $O(n \log n)$

Le tri rapide

- Comme le tri par fusion, il met en œuvre le principe *diviser pour régner*
- Sa complexité dans le pire des cas est mauvaise ($O(n^2)$)
- Mais, en moyenne, elle est de l'ordre de $n \log n$
- Le principe
  1. **Diviser** le tableau en deux parties non vides telles que tout élément de *t[0,p]* soit inférieur à tout élément de *t[p+1,n-1]*
  2. **Régner** : les deux sous-tableaux sont triés récursivement
  3. **Combiner** : aucune action à réaliser

Le tri rapide

```
Procédure TriRapide(t : tableau, d,f :entier).
variable : p : entier
Début
    Si d<f alors
      p ← Partitionner(t,d,f)
      TriRapide(t,d,p)
      TriRapide(t,p+1,f)
    Fsi
Fin.
```

Il ne manque plus que la fonction *Partitionner()*...

La fonction *Partitionner()*

```
Procédure Partitionner(t : tableau, d,f : entier).
variable :  aux,x,i,j : entier
Début
    x ← t[d] ;i ← d-1 ; j ← f+1
    Tantque VRAI Faire
      Répéter
        j ← j-1
      Jusqu'à t[j] ≤ x
      Répéter
        i ← i+1
      Jusqu'à t[j] ≥ x
      Si i<j alors
        aux ← t[i]
        t[i] ← t[j]
        t[j] ← aux
      Sinon Partitionner ← j
      Fsi
    Ftant
Fin.
```