Test de niveau en algorithmique

Christian Rinderknecht

Mardi 4 mars 2003

1 Binôme de Pascal

Soit la fonction `binome` spécifiée ainsi :

```
binome (n,p) {
  if p = 0 or p = n then return 1
  else return binome (n-1,p) + binome (n-1,p-1);
}
```

1. Déterminez le nombre d'appels de la fonction `binome` effectués lors du calcul de `binome (n,p)` en fonction de `n` et `p`.
2. Peut-on augmenter la vitesse de calcul ?

Réponses.

1. Soit $\mathrm{T}(n,p)$ le nombre cherché. On a, en suivant la syntaxe de la définition de `binome` :

$$\begin{cases} \mathrm{T}(n,0) = \mathrm{T}(n,n) = 1 \\ \mathrm{T}(n,p) = 1 + \mathrm{T}(n-1,p) + \mathrm{T}(n-1,p-1) \text{ si } 0 < p < n \end{cases}$$

   Si l'on pose $\mathrm{T}'(n,p) = \mathrm{T}(n,p) + 1$, alors on obtient :

$$\begin{cases} \mathrm{T}'(n,0) = \mathrm{T}'(n,n) = 2 \\ \mathrm{T}'(n,p) = \mathrm{T}'(n-1,p) + \mathrm{T}'(n-1,p-1) \text{ si } 0 < p < n \end{cases}$$

   Il en résulte par récurrence que $\mathrm{T}'(n,p) = 2C_n^p$, et donc $\mathrm{T}(n,p) = 2C_n^p - 1$. Ainsi, la fonction `binome` effectue environ deux fois plus de calculs que le résultat qu'elle fournit !

2. Oui, il faut trouver une moyen de mémoriser les calculs antérieurs, un vecteur par exemple, ou alors une autre formule de récurrence telle que :

$$C_n^p = \frac{n!}{p!(n-p)!} = \frac{n}{p} C_{n-1}^{p-1}$$

   Notez que cette dernière peut provoquer un débordement de capacité lorsque nC_{n-1}^{p-1} est supérieur au plus grand entier représentable en machine. De plus, l'expression `n * (binome(n-1,p-1) / p)` est incorrecte car il est possible que C_{n-1}^{p-1} ne soit pas divisible par p.

2 Complexité et notations de Landau

1. Qu'est-ce que la complexité spatiale et temporelle d'un algorithme ?
2. Définissez les notations O, Ω et Θ et leur convention d'usage.

Réponses.

1. Afin de comparer plusieurs algorithmes résolvant un même problème, on introduit des mesures de ces algorithmes appelées *complexités* :
   — la *complexité temporelle* est le nombre d'opérations, par définition élémentaires, effectuées par uen machine qui exécute l'algorithme ;
   — la *complexité spatiale* est le nombre d'unités de mémoire utilisées par une machine qui exécute l'algorithme (l'unité est caractéristique du modèle de machine).
   Ces deux complexités dépendent de la machine employée mais aussi de la taille des données. Cette dernière doit donc être modélisée explicitement.
2. Soient f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}^+$.

   Definition 2.1 (Majoration asymptotique) $O(g) = \{f : \mathbb{N} \mapsto \mathbb{R}^+ \mid \exists c \in \mathbb{R}^{+*}, \exists n_0 \in \mathbb{N}, \forall n > n_0, f(n) \leqslant c\, g(n)\}$

   Abusivement, on notera $f = O(g)$ ou $f(x) = O(g(x))$ au lieu de $f \in O(g)$. De même $O(f) = O(g)$ au lieu de $O(f) \subset O(g)$.

   Definition 2.2 (Minoration asymptotique) $f \in \Omega(g) \Leftrightarrow g \in O(f)$

   Definition 2.3 (Encadrement asymptotique) $\Theta(g) = O(g) \cap \Omega(g)$

3 Tris

1. Qu'est-ce qu'une relation de comparaison ? Un ordre total ?
2. Qu'est-ce qu'une liste triée ?
3. Qu'est-ce qu'un algorithme de tri par comparaisons ?
4. Qu'est-ce qu'un algorithme de tri interne (ou *sur place*) ?
5. Qu'est qu'un algorithme de tri interne ?
6. Qu'est qu'un algorithme de tri stable ?
7. Présentez deux algorithmes de tri. Montrez leur terminaison et leur correction. Donnez leur complexité temporelle et spatiale dans le pire des cas. Discutez leur stabilité.
8. **Polynômes creux.** On représente les polynômes à une variable à coefficients entiers par des listes chaînées de monômes, un monôme ax^n étant représenté par le couple d'entiers (a, e). Les monômes d'un polynôme sont classés par degré croissant et chaque monôme a un coefficient non nul. Programmez l'addition de deux polynômes pour cette représentation.

Réponses.

1. Une *relation de comparaison* sur un ensemble E est une relation binaire $\leqslant$ sur E réflexive, transitive et telle que pour deux éléments quelconques a et b de E l'une au moins des relations $a \leqslant b$ ou $b \leqslant a$ est vraie. On dit

que a et b sont *équivalents* (ou égaux), ce qui est noté $a = b$, si on a $a \leqslant b$ et $b \leqslant a$. Une relation d'ordre total est une relation de comparaison pour laquelle chaque élément n'est égal qu'à lui-même.

2. Une liste $\mathrm{L} = (a_1, \ldots, a_n)$ d'éléments de E est dite *triée* selon $\leqslant$ si $\forall i \in [1, n-1], a_i \leqslant a_{i+1}$.
3. Un algorithme de tri par comparaisons est un algorithme de tri dans lequel il n'est effectué que des comparaisons entre éléments pour décider quelle permutation de la liste doit être réalisée.
4. Un tri interne est un tri qui n'autorise qu'une mémoire auxiliaire indépendante de la taille de la liste à trier.
5. Un tri stable est un tri qui conserve la position relative des éléments équivalents.
6. **Le tri par sélection.**
   — **Principe.** Il consiste à trouver l'emplacement de l'élément le plus petit du tableau $(a_1, \ldots, a_n)$, c'est-à-dire l'entier m tel que $a_i \geqslant a_m$ pour tout i. Ensuite on échange a_1 et a_m, puis on recommence avec le sous-tableau $(a_2, \ldots, a_n)$:
   — **Algorithme.**

```
tri_selection (a) {
  for i <- 1 to n-1 {
    m <- i
    for j <- i+1 to n {
      if a[j] < a[m] then m <- j
    }
    a[i] <-> a[m]
  }
}
```

   — **Terminaison.** L'algorithme termine car il n'est constitué que d'itérations bornées (boucles `for`).
   — **Correction.** L'invariant de boucle à établir est qu'avant chaque itération les éléments $a_1, \ldots, a_{i-1}$ sont bien placés. En supposant que $a_0 = -\infty$, cette propriété est vraie pour $i = 1$ (i.e. avant la première itération). Supposons que la propriété est vraie avant une itération quelconque. Alors a_m est l'élément minimal du sous-tableau restant $(a_i, \ldots, a_n)$. Par conséquent, après la permutation de a_i et a_m, le nouvel a_i est bien placé. Donc les éléments $a_1, \ldots, a_i$ sont bien placés avant l'itération suivante. Finalement, la boucle s'achève avec $i = n$ (i.e. dépassement de 1 de la borne supérieure). Donc les éléments $a_1, \ldots, a_{n-1}$ sont bien placés. Cela implique que a_n est bien placé, donc que le tableau $(a_1, \ldots, a_n)$ est trié.
   — **Complexité temporelle.** Le nombre de comparaisons est toujours :

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \frac{n(n-1)}{2} = \Theta(n^2)$$

   La complexité temporelle du tri par sélection est quadratique par rapport à la taille du tableau.

— **Complexité spatiale**. Le tri par sélection sur un tableau est un tri interne. La complexité spatiale est donc proportionnelle à la taille du tableau : $\Theta(n)$.
— **Stabilité.** S'il existe plusieurs éléments équivalents minimaux dans le sous-tableau $a_{i+1,n}$, c'est le premier qui sera sélectionné (a_m), et positionné. L'ordre relatif de deux éléments équivalents n'est donc pas modifié : le tri par sélection est stable.

Le tri par insertion.

— **Principe.** Il consiste à insérer un élément dans un sous-tableau déjà trié à gauche (c'est le tri du joueur de cartes) :
— **Algorithme.**

```
tri_insertion (a) {
  for j <- 2 to n {
    key <- a[j]
    i <- j - 1
    while i > 0 and a[i] > key {
      a[i+1] <- a[i]
      i <- i - 1
    }
    a[i+1] <- key
  }
}
```

— **Terminaison.** La terminaison de l'algorithme se ramène à celle de l'itération `while`. La fin de la boucle est assurée si $i = 0$ (dans le pire des cas). Or la variable i est positive et est décrémentée à chaque itération. Donc, l'algorithme termine.
— **Correction.** L'invariant de boucle à établir est qu'avant chaque itération (`for`), le sous-tableau $a_{1,j-1}$ est constitué des éléments originellement dans $a_{1,j-1}$ mais triés. Avant la première itération, $j = 2$, donc le sous-tableau en question est réduit à $a[1]$, qui est le $a[1]$ initial trivialement trié. Supposons la propriété vraie avant une itération quelconque. Avant l'itération suivante, on a $a_1 \leqslant \ldots \leqslant a_i \leqslant a_j \leqslant a_{i+1} \leqslant \ldots \leqslant a_{j-1}$. Donc, le nouveau sous-tableau $a_{1,j}$ est constitué des éléments initiaux mais triés. À la fin de la boucle, on a $j = n+1$, donc le tableau $a_{1,n}$ est trié.
— **Complexité temporelle.** Lorsque le tableau est trié par ordre décroissant, nous sommes dans le pire des cas. Le nombre de comparaisons est alors :

$$\sum_{j=2}^{n} \sum_{i=1}^{j-1} 1 = \frac{n(n-1)}{2} = O(n^2)$$

Lorsque le tableau est déjà trié, nous sommes dans le meilleur des cas. Le nombre de comparaisons est alors :

$$\sum_{j=2}^{n} 1 = n - 1$$

— **Complexité spatiale**. Le tri par insertion sur un tableau est un tri interne. La complexité spatiale est donc proportionnelle à la taille du tableau : $\Theta(n)$.

— **Stabilité.** Un élément a_i n'est déplacé que si $a_i > key$ et $i < j$. Donc si $key = a_i$, alors a_i n'est pas déplacé, et key est inséré juste après a_i, conservant ainsi l'ordre relatif initial de ces deux éléments. Le tri par insertion est donc stable.

7. **Polynômes creux. À FAIRE.**

4 Arbres

4.1 Arbres binaires

1. Définissez les arbres binaires.
2. Qu'est-ce que le parcours en profondeur et en largeur ?
3. Parcours préfixe, postfixe et infixe ?
4. **Dénombrements sur les arbres binaires.** Établir :
   (a) Un arbre binaire à n nœuds possède $n + 1$ branches vides.
   (b) Dans un arbre binaire à n nœuds, le nombre de nœuds sans fils est inférieur ou égal à $(n + 1)/2$. Il y a égalité si et seulement si tous les nœuds ont zéro ou deux fils.
   (c) La hauteur d'un arbre binaire non vide à n nœuds est comprise entre $\lfloor \log_2 n \rfloor$ et $n - 1$.
5. Qu'est-ce qu'un arbre binaire équilibré en hauteur ?

Réponses.

1. Un arbre binaire est un ensemble fini, éventuellement vide, de nœuds liés par une relation de parenté orientée :

$$\begin{cases} x \,\mathcal{F}_d\, y \iff x \text{ est le fils droit de } y \implies y \text{ est le père de } x \\ x' \,\mathcal{F}_g\, y \iff x' \text{ est le fils gauche de } y \implies y \text{ est le père de } x' \end{cases}$$

   avec les propriétés :
   — si l'arbre est non vide alors il existe un unique élément n'ayant pas de père, appelé *racine* de l'arbre ;
   — tout élément à part la racine a un et un seul père ;
   — tout élément a au plus un fils droit et au plus un fils gauche ;
   — tout élément est un descendant de la racine.
2. — Le parcours en profondeur d'abord : partant de la racine, on explore d'abord complètement les sous-arbres ;
   — le parcours en largeur d'abord : partant de la racine, on explore tous les fils avant de passer au niveau suivant.
3. Un parcours préfixe passe par le père puis les sous-arbres des fils. Un parcours postfixe explore les sous-arbres des fils, puis le père. Un parcours infixe passe par un sous-arbre, puis le père, puis l'autre sous-arbre (l'arbre doit être binaire).
4. (a) Un arbre binaire réduit à la racine possède deux sous-arbres vides. Supposons la propriété vraie au rang n. L'ajout d'un nœud revient à remplacer un sous-arbre vide par un sous-arbre non vide, et donc d'ajouter deux sous-arbres vides. Au total on a donc ajouté un sous-arbre vide en ajoutant un nœud.

(b) Soit p le nombre de nœuds ayant un seul fils et q le nombre de nœuds sans fils. On a : $p + 2q = n - 1$ (le nombre de sous-arbres vides), donc $q = (n - 1 - p)/2$.

(c) La hauteur maximale d'un arbre de taille n est au plus $n - 1$ (cas de l'arbre dégénéré en liste [1]). Un arbre binaire non vide de hauteur h a au plus $1 + \ldots + 2^h = 2^{h+1} - 1$ nœuds. Soit a un arbre binaire de taille n et $h = \lfloor \log_2 n \rfloor$: tout arbre binaire non vide de hauteur inférieure ou égale à $h - 1$ a donc au plus $2^h - 1 < n$ nœuds. La hauteur de a est au moins égale à h.

5. Un arbre binaire est dit *équilibré en hauteur* s'il est vide ou si pour tout nœud, les sous-arbres gauches et droits ont même hauteur à une unité près.

4.2 Arbres binaires de recherche

1. Qu'est-ce qu'un arbre binaire de recherche (ABR) ?
2. Écrire une fonction qui teste si un arbre binaire est bien un ABR.
3. Spécifiez l'ajout dans un ABR aux feuilles et à la racine. Comparez leur complexité temporelle et spatiale dans le pire des cas. Quel rapport y a-t-il entre un ABR créé par ajout aux feuilles et un par ajout à la racine, à partir des mêmes éléments ?
4. Programmez la suppression et la recherche dans un ABR. Quels sont leur coûts asymptotiques ?
5. Écrire une fonction réalisant la fusion de deux ABR. Analysez sa complexité temporelle.

Réponses.

1. Un arbre binaire de recherche est un arbre binaire équilibré dont les nœuds sont étiquetés (par des clés, c'est-à-dire des éléments totalement ordonnables) tel que pour tout nœud, son étiquette est plus grande (ou égale) que toutes les étiquettes du sous-arbre gauche, et plus petite que les étiquettes du sous-arbre droit.
2. **À FAIRE.**
3. **À FAIRE.**
4. **À FAIRE.**
5. **À FAIRE.**

5 Tables de hachage

1. Qu'est-ce qu'une table de hachage ? Dans quelles circonstances sont-elles une structure de donnée utile ?
2. Pourquoi recommande-t-on souvent de choisir un nombre premier pour la taille d'une table de hachage ?

1. On parle alors de *peigne*

3. Soient un ensemble E de n éléments et une fonction $h : E \to [1..m]$ uniforme (c-à-d. $\forall e \in E. \forall i \in [1..m]. \mathcal{P}\{h(e) = i\} = 1/m$). Montrez que la probabilité P que h soit injective vaut $m!/(m-n)!m^n$. En particulier, si $m = 356$ et $n = 23$, alors $P < 1/2$. Parieriez-vous que deux élèves dans cette classe soient nées le même jour du même mois ? Qu'en conclure à propos des tables de hachage ?
4. Expliquez les méthodes de résolution des collisions par chaînage interne et externe (*hachage indirect*), et par calcul (*hachage direct*).

Réponses.

1. Dans les méthodes arborescentes (par exemple les arbres binaires de recherche), il y a un ordre sur les clés, et la place d'un élément dépend du rang de sa clé par rapport aux clés des autres éléments. Dans les méthodes de hachage, par contre, la place d'un élément est calculée uniquement à partir de sa clé ; ce calcul est réalisé grâce à une fonction, dite fonction de hachage, qui transforme directement la clé en un indice de tableau. Ces méthodes ont la particularité de permettre les opérations de recherche, d'adjonction et de suppression en un nombre de comparaisons qui est en moyenne constant, c'est-à-dire qui ne dépend pas du nombre d'éléments de la collection.

   Les tables de hachages sont très adaptées pour la modélisation de dictionnaires, où les suppressions sont rares (les suppressions dans les tables de hachage sont plus coûteuses que les insertions), comme par exemple les tables de symboles dans les compilateurs.
2. On prend d'habitude $B = 128$ ou $B = 256$ et N un nombre premier. En effet, le choix d'une puissance de 2 pour la valeur de B est guidé par le fait que la représentation en machine des entiers se fait en base 2, et que les opérations sur eux se font de façon très efficaces. Prendre N premier évite tout interférence entre la mutiplication par B et la division par N. Par exemple, si $B = N = 128$, alors $h(x) = x[n]$, et la fonction h ne dépendrait alors que du dernier caractère de x.
3. **À FAIRE.**
4. (a) Les méthodes de résolution des collisions par chaînage : les éléments dont la clé a la même valeur par la fonction de hachage sont chaînés entre eux, à l'intérieur ou à l'extérieur du tableau. On parle alors de hachage indirect.

   (b) Les méthodes de résolution des collisions par calcul : lorsqu'il y a collision, on calcule un nouvel indice dans le tableau à partir de la clé de l'élément considéré. On parle alors de hachage direct.

6 Indexation

On désire écrire un programme qui, à la lecture d'un programme, repère pour chaque identificateur les lignes où il apparaît. Un identificateur commence par une lettre et n'est composé que de lettres et de chiffres. Certains mots, dits réservés, ne peuvent pas être utilisés comme identificateurs.

Après la lecture, on demande d'imprimer le programme, puis la liste (triée par ordre alphabétique) des identificateurs suivis chacun de la liste des numéros de ligne où il apparaît, ces numéros de ligne étant rangés en ordre croissant.

1. Faites une analyse comparée des différentes structures de données possibles ;
2. programmez une méthode utilisant des arbres binaires de recherche, et une méthode utilisant un tableau de hachage ; analysez les résultats.

7 Graphes

1. Qu'est-ce qu'un graphe non-orienté ? Orienté ?
2. Qu'est-ce qu'un graphe connexe ? Fortement connexe ?
3. Qu'est-ce qu'un parcours en profondeur d'abord ? En largeur d'abord ?
4. Donnez un algorithme de calcul des composantes fortement connexes, et sa complexité temporelle et spatiale dans le pire des cas.
5. Citez des cas où les graphes sont la structure de donnée idoine.

8 Automates et langages réguliers

1. Qu'est-ce qu'un automate fini ?
2. Qu'est-ce qu'un automate fini déterministe et non-déterministe ? Quels sont les liens entre ces deux types d'automates ?
3. Qu'est-ce qu'un automate minimal ? Y a-t-il unicité ? Donnez un algorithme de minimisation et discutez sa complexité.
4. Qu'est-ce qu'un langage régulier ? Une expression régulière ?
5. Quels liens y a-t-il entre les langages réguliers et les automates finis ?
6. Citez des applications intéressantes des automates finis et des expressions régulières.