

Introduction à l'algorithmique : Complexité

Serge Haddad

LMF, ENS Paris-Saclay & CNRS & Inria

L3

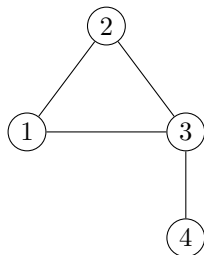
- 1 Equations de récurrence
- 2 Complexité amortie
- 3 Complexité en moyenne
- 4 Bornes inférieures de complexité

Problèmes et instances

Problèmes

- ▶ Connexité d'un graphe
- ▶ Tri d'un tableau

Instances



→ true

$(7 \ 3 \ 5 \ 1) \rightarrow (1 \ 3 \ 5 \ 7)$

Taille d'une instance

Choix d'une représentation.

Un graphe de n sommets et m arcs peut être représenté par :

- ▶ un tableau de n listes d'indices de sommets : $\Theta(n + m \log(n))$;
- ▶ une matrice $n \times n$ de booléens : $\Theta(n^2)$.

$f \in O(g)$ ssi $\exists n_0, C > 0 \forall n \geq n_0 f(n) \leq Cg(n)$,

$f \in \Omega(g)$ ssi $g \in O(f)$, $f \in \Theta(g)$ ssi $f \in O(g) \cap \Omega(g)$.

Selon la nature des problèmes, les entiers sont considérés :

- ▶ de taille constante (borne connue *a priori*)
- ▶ de taille variable,
 1. généralement en représentation binaire d'où $\sim \log(n)$ bits pour n ;
 2. en représentation unaire d'où $\sim n$ bits pour n lorsqu'on désire mesurer l'impact des entiers sur la complexité des problèmes.

Abstraction de la taille des instances.

- ▶ la « taille » du graphe est $m + n$;
- ▶ si analyse multi-paramètres, la « taille » du graphe est (n, m) .

Mesures de complexité

Deux mesures usuelles :

- ▶ le temps d'exécution ;
- ▶ la mémoire occupée.

Quel cas considérer pour une taille d'instance fixée ?

- ▶ le pire cas d'exécution ;
- ▶ le cas moyen qui suppose une distribution sur les instances de même taille ;
- ▶ le meilleur cas (rarement étudié).

Comment s'abstraire de l'évolution des machines ?

- ▶ temps d'exécution symbolique : $X \leftarrow E$, **if** c **then** ... en 1 unité de temps ;
- ▶ opérations significatives : nombre de comparaisons pour un tri.

Complexité asymptotique

On s'intéresse à l'évolution de la complexité en fonction de la taille n :

- ▶ $t_{max}(n) = \max(t(I) \mid I \text{ instance de taille } n)$;
- ▶ $t_{moy}(n) = \mathbf{E}(t(I))$ où I est une instance de taille n aléatoirement choisie selon la distribution spécifiée.

Illustration : Tri par insertion.

```
For  $i$  from 2 to  $n$  do  
   $x \leftarrow T[i]$  ;  $j \leftarrow i - 1$   
  While  $j > 0$  and  $T[j] > x$  do  $T[j+1] \leftarrow T[j]$  ;  $j \leftarrow j - 1$   
   $T[j + 1] \leftarrow x$ 
```

- $T = (n, n - 1, \dots, 1) \Rightarrow t_{max}(n) = \Theta(n^2)$
- Supposons que le contenu de $T[i]$ est choisi dans $\{i, i + 1, \dots\}$ avec $\mathbf{Pr}(T[i] = j) = 2^{-(j-i+1)}$.

$$t_{moy}(n) = \Theta(n)$$

Plan

1 Equations de récurrence

Complexité amortie

Complexité en moyenne

Bornes inférieures de complexité

Intérêt des équations de récurrence (1)

Calcul récursif du maximum d'un tableau ($n = 2^k$).

```
Max( $T, n$ )  
If  $n = 1$  then return( $T[1]$ );  
For  $i$  from 1 to  $\frac{n}{2}$  do  
  If  $T[2 * i - 1] < T[2 * i]$  then  
     $Tbis[i] \leftarrow T[2 * i]$ ;  
  Else  
     $Tbis[i] \leftarrow T[2 * i - 1]$ ;  
return(Max( $Tbis, \frac{n}{2}$ ));
```

$$t(n) = t\left(\frac{n}{2}\right) + \Theta(n)$$

Intérêt des équations de récurrence (2)

Recherche dichotomique dans un tableau trié ($n = 2^k - 1$).

```
Cherche( $x, T, deb, fin$ )  
 $i \leftarrow \lfloor \frac{deb+fin}{2} \rfloor$  ;  
If  $x = T[i]$  then return( $i$ ) ;  
If  $x > T[i]$  then  
  If  $i = fin$  then return(false) ;  
  return(Cherche( $x, T, i + 1, fin$ )) ;  
Else  
  If  $i = deb$  then return(false) ;  
  return(Cherche( $x, T, deb, i - 1$ )) ;
```

$$t(n) = t\left(\frac{n-1}{2}\right) + \Theta(1)$$

Intérêt des équations de récurrence (3)

Produit de matrices par bloc ($n = 2^k$).

$\text{Prod}(A, B, n)$

If $n = 1$ **then return** $(A[1, 1] * B[1, 1])$;

$C[1, \frac{n}{2}][1, \frac{n}{2}] \leftarrow \text{Prod}(A[1, \frac{n}{2}][1, \frac{n}{2}], B[1, \frac{n}{2}][1, \frac{n}{2}], \frac{n}{2})$
+ $\text{Prod}(A[1, \frac{n}{2}][\frac{n}{2} + 1, n], B[\frac{n}{2} + 1, n][1, \frac{n}{2}], \frac{n}{2})$;

$C[1, \frac{n}{2}][\frac{n}{2} + 1, n] \leftarrow \text{Prod}(A[1, \frac{n}{2}][1, \frac{n}{2}], B[1, \frac{n}{2}][\frac{n}{2} + 1, n], \frac{n}{2})$
+ $\text{Prod}(A[1, \frac{n}{2}][\frac{n}{2} + 1, n], B[\frac{n}{2} + 1, n][\frac{n}{2} + 1, n], \frac{n}{2})$;

$C[\frac{n}{2} + 1, n][1, \frac{n}{2}] \leftarrow \text{Prod}(A[\frac{n}{2} + 1, n][1, \frac{n}{2}], B[1, \frac{n}{2}][1, \frac{n}{2}], \frac{n}{2})$
+ $\text{Prod}(A[\frac{n}{2} + 1, n][\frac{n}{2} + 1, n], B[\frac{n}{2} + 1, n][1, \frac{n}{2}], \frac{n}{2})$;

$C[\frac{n}{2} + 1, n][\frac{n}{2} + 1, n] \leftarrow \text{Prod}(A[\frac{n}{2} + 1, n][1, \frac{n}{2}], B[1, \frac{n}{2}][\frac{n}{2} + 1, n], \frac{n}{2})$
+ $\text{Prod}(A[\frac{n}{2} + 1, n][\frac{n}{2} + 1, n], B[\frac{n}{2} + 1, n][\frac{n}{2} + 1, n], \frac{n}{2})$;

return (C) ;

$$t(n) = 8t\left(\frac{n}{2}\right) + \Theta(n^2)$$

Une équation générique

Soit $t : \mathbb{N} \mapsto \mathbb{R}^+$, une fonction croissante à partir d'un certain rang telle qu'il existe des entiers $n_0 \geq 1$, $b \geq 2$, $k \geq 0$, $a > 0$, $c > 0$ et $d > 0$ pour lesquels :

$$t(n_0) = d$$

$$t(n) = at(n/b) + cn^k \text{ pour } n = n_0 b^p \text{ avec } p \geq 1$$

Alors :

$$t(n) = \begin{cases} \Theta(n^k) & \text{si } a < b^k \Leftrightarrow \log_b(a) < k \\ \Theta(n^k \log_b(n)) & \text{si } a = b^k \Leftrightarrow \log_b(a) = k \\ \Theta(n^{\log_b(a)}) & \text{si } a > b^k \Leftrightarrow \log_b(a) > k \end{cases}$$

Interprétation.

t , le (pire) temps d'exécution d'une fonction récursive croît avec la taille.

a , nombre de sous-problèmes à résoudre

b , facteur de réduction de la taille

n^k , ordre de grandeur du temps d'un appel sans les appels récursifs

Preuve de l'équation

Posons $n = n_0 b^p$. Alors par récurrence, on obtient :

$$t(n) = da^p + \sum_{i=0}^{p-1} ca^i (n/b^i)^k = da^p + cn^k \sum_{i=0}^{p-1} (a/b^k)^i$$

Or $da^p = d(b^{\log_b(a)})^p = d(b^p)^{\log_b(a)} = \frac{d}{n_0^{\log_b(a)}} n^{\log_b(a)} = \Theta(n^{\log_b(a)})$.

Posons $\gamma(n) = \sum_{i=0}^{p-1} (a/b^k)^i$. On a :

- ▶ si $a/b^k < 1$ alors $\gamma(n) \sim \frac{1}{1-a/b^k} \Rightarrow \gamma(n) = \Theta(1)$
- ▶ si $a/b^k = 1$ alors $k = \log_b(a)$ et $\gamma(n) = p = \log_b(n) - \log_b(n_0) \sim \log_b(n)$
- ▶ si $a/b^k > 1$ alors $\gamma(n) \sim \alpha \frac{a^p}{b^{kp}}$ avec $\alpha = (a/b^k - 1)^{-1}$
 $\Rightarrow cn^k \gamma(n) \sim \alpha cn_0^k a^p = \Theta(n^{\log_b(a)})$

Soit $n \geq n_0$ et p tel que $n_0 b^p < n \leq n_0 b^{p+1}$.

$t(n_0 b^p) \leq t(n) \leq t(n_0 b^{p+1})$ et $t(n_0 b^p) = \Theta(t(n_0 b^{p+1}))$.

Généralisation de l'équation

Soit $t : \mathbb{N} \mapsto \mathbb{R}^+$, une fonction croissante à partir d'un certain rang telle qu'il existe une fonction $f : \mathbb{N} \mapsto \mathbb{R}^+$, des entiers $n_0 \geq 1$, $b \geq 2$, $k \geq 0$, $a > 0$ et $d > 0$ pour lesquels :

$$t(n_0) = d$$

$$t(n) = at(n/b) + f(n) \text{ pour } n = n_0 b^p \text{ avec } p > 1$$

Alors :

$$t(n) = \begin{cases} \Theta(f(n)) & \text{si } a < b^k \text{ et } f(n) = \Omega(n^k) \\ & \text{et } af(n/b) \leq cf(n) \text{ avec } 0 < c < 1 \\ \Theta(n^{\log_b(a)} \log_b(n)) & \text{si } f(n) = \Theta(n^{\log_b(a)}) \\ \Theta(n^{\log_b(a)}) & \text{si } a > b^k \text{ et } f(n) = O(n^k) \end{cases}$$

Interprétation : $f(n)$ « se comporte » comme n^k .

Applications de l'équation

Calcul récursif du maximum d'un tableau $t(n) = t(\frac{n}{2}) + \Theta(n)$

$$a = 1, b = 2, k = 1 \Rightarrow a < b^k \Rightarrow t(n) = \Theta(n)$$

Recherche dichotomique dans un tableau trié $t(n) = t(\frac{n-1}{2}) + \Theta(1)$

$$a = 1, b = 2, k = 0 \Rightarrow a = b^k \Rightarrow t(n) = \Theta(\log(n))$$

Produit de matrices par bloc $t(n) = 8t(\frac{n}{2}) + \Theta(n^2)$

$$a = 8, b = 2, k = 2 \Rightarrow a > b^k \Rightarrow t(n) = \Theta(n^{\log_2(8)}) = \Theta(n^3)$$

On peut réduire le nombre de multiplications à 7. D'où $t(n) = \Theta(n^{\log_2(7)})$.

Sous-problèmes de taille différente

Soient $\alpha_1, \dots, \alpha_k : \mathbb{N} \mapsto \mathbb{N}$, des fonctions telles qu'il existe un nombre réel $0 < K < 1$ et n_0 un entier avec :

$$\forall n > n_0 \quad \alpha_1(n) + \dots + \alpha_k(n) \leq Kn$$

Si une fonction $t : \mathbb{N} \mapsto \mathbb{R}$ vérifie avec $b > 0$ un réel :

$$\forall n \leq n_0 \quad t(n) \leq a_0$$

$$\forall n > n_0 \quad t(n) \leq t(\alpha_1(n)) + \dots + t(\alpha_k(n)) + bn$$

Alors : $t(n) = O(n)$

Interprétation.

k appels récursifs

$\alpha_i(n)$, taille du i ème sous-problème associé au problème de taille n

K , facteur de réduction de la taille cumulée

n , ordre de grandeur du temps d'un appel sans les appels récursifs

Preuve

Démontrons par récurrence que $t(n) \leq Dn + a_0$ avec :

$$D = \frac{b + (k-1)a_0}{1-K} \text{ d'où } DK + b = D - (k-1)a_0$$

Cette inégalité est vérifiée pour $n \leq n_0$.

Si $n > n_0$ alors :

$$\begin{aligned} t(n) &\leq t(\alpha_1(n)) + \dots + t(\alpha_k(n)) + bn \\ &\leq ka_0 + D(\alpha_1(n) + \dots + \alpha_k(n)) + bn \\ &\leq ka_0 + (DK + b)n \\ &= ka_0 + (D - (k-1)a_0)n \\ &= Dn + a_0 + (k-1)a_0(1-n) \\ &\leq Dn + a_0 \end{aligned}$$

Plan

Equations de récurrence

② Complexité amortie

Complexité en moyenne

Bornes inférieures de complexité

Analyse amortie : illustration

Soit cpt , un compteur de n bits, qu'on peut incrémenter ainsi :

Increment(cpt)

$i \leftarrow 1$; $cpt[i] \leftarrow cpt[i] + 1 \pmod 2$;

While $i < n$ **and** $cpt[i] = 0$ **do** $i \leftarrow i + 1$; $cpt[i] \leftarrow cpt[i] + 1 \pmod 2$;

Quelle est la complexité de la boucle suivante ?

For i **from** 1 **to** k **do** Increment(cpt);

Une analyse superficielle :

- ▶ au plus n modifications de bit par incrémentation;
- ▶ d'où une complexité en $O(kn)$.

Une analyse plus approfondie :

- ▶ Le i ème bit est modifié toutes les 2^{i-1} incrémentations;
- ▶ d'où une complexité en

$$3 \sum_{i=1}^n \left\lceil \frac{k}{2^{i-1}} \right\rceil \leq 3 \sum_{i=1}^n \left(\frac{k}{2^{i-1}} + 1 \right) \leq 6k + 3n = \Theta(k + n).$$

Méthode du potentiel

Un *potentiel* pot est une fonction de l'ensemble des états dans \mathbb{N} .

Notons $s \xrightarrow{\text{op}} s'$ l'exécution de op depuis l'état s conduisant à l'état s' .

Soit $t(\text{op})$, le temps d'exécution de op .

Alors le *temps d'exécution amorti* (par le potentiel pot) est défini par :

$$a(\text{op}) = t(\text{op}) + \text{pot}(s') - \text{pot}(s)$$

Soit une exécution d'un programme $s_0 \xrightarrow{\text{op}_1} s_1 \cdots \xrightarrow{\text{op}_k} s_k$

Alors $\sum_{i=1}^k a(\text{op}_i) = \sum_{i=1}^k t(\text{op}_i) + \text{pot}(s_k) - \text{pot}(s_0)$

D'où :

$$\sum_{i=1}^k t(\text{op}_i) \leq \sum_{i=1}^k a(\text{op}_i) + \text{pot}(s_0)$$

Observations.

- ▶ Le potentiel reflète la complexité « cachée dans l'état ».
- ▶ Le choix d'un potentiel approprié facilite le calcul de la complexité.

Un potentiel pour le compteur

```
Increment(cpt)
```

```
i ← 1; cpt[i] ← cpt[i] + 1 mod 2;
```

```
While i < n and cpt[i] = 0 do i ← i + 1; cpt[i] ← cpt[i] + 1 mod 2;
```

On définit pot comme trois fois le nombre de bits du compteur à 1.

Deux cas d'incréméntation.

- ▶ $cpt < 2^n - 1$: $t(\text{Increment}) = 3 + 3n'$ avec n' tours de boucle et le potentiel décroît de $3(n' - 1)$ d'où $a(\text{Increment}) = 6$;
- ▶ $cpt = 2^n - 1$: $t(\text{Increment}) = 3 + 3(n - 1)$ et le potentiel décroît de $3n$ d'où $a(\text{Increment}) = 0 \leq 6$.

```
For i from 1 to k do Increment(cpt);
```

La complexité amortie de la boucle est inférieure ou égale à $6k$.

La complexité de la boucle est inférieure ou égale à $6k + 3n$.

Gestion d'une pile

PT pointeur sur un tableau ; $n = 1$ taille du tableau ; $top = 0$ sommet de pile ;

Empiler(v) ;

$PT2$ pointeur sur un tableau ; i indice ;

$top \leftarrow top + 1$;

If $top > n$ **then**

$PT2 \leftarrow$ Allouer($f(n)$) ;

For i **from** 1 **to** n **do** $(*PT2)[i] \leftarrow (*PT)[i]$;

 Liberer($*PT$) ; $PT \leftarrow PT2$; $n \leftarrow f(n)$;

$(*PT)[top] \leftarrow v$;

f doit vérifier $f(n) > n$... mais comment choisir $f(n)$?

$f(n) = 2n$ est un bon choix.

Un potentiel pour la pile

Quelle est la complexité de la boucle suivante ?

For i **from** 1 **to** k **do** Empiler(v);

Plus top est proche de n , plus grand est le risque de débordement :

$$Pot = 2top - n + 1$$

Analyse amortie.

- ▶ Il n'y a pas de débordement. $t(\text{Empiler}(v)) = 3$ et Pot croît de 2.
D'où $a(\text{Empiler}(v)) = 5$.
- ▶ Il y a un débordement. $t(\text{Empiler}(v)) = 7 + n$ et Pot décroît de $n + 1$ à 3.
D'où $a(\text{Empiler}(v)) = 9$.

La complexité amortie de la boucle est inférieure ou égale à $9k$.

La complexité de la boucle est inférieure ou égale à $9k$.

Plan

Equations de récurrence

Complexité amortie

3 Complexité en moyenne

Bornes inférieures de complexité

Complexité en moyenne

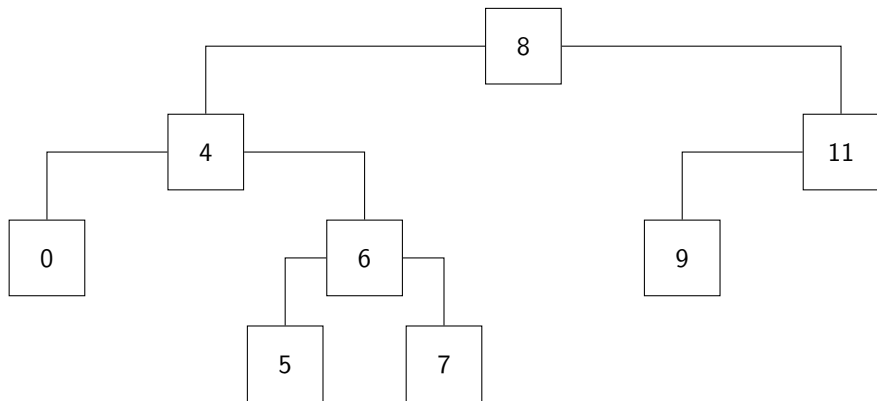
Un arbre binaire de recherche est une structure de données permettant :

- ▶ d'insérer des enregistrements ;
- ▶ de supprimer des enregistrements ;
- ▶ de rechercher des enregistrements par leur clé.

Propriétés.

- ▶ Chaque noeud de l'arbre contient un enregistrement ;
- ▶ Chaque noeud de l'arbre a (éventuellement) un fils gauche et un fils droit ;
- ▶ La clé d'un noeud est supérieure aux clés de son sous-arbre gauche et inférieure aux clés de son sous-arbre droit.

Un arbre binaire de recherche

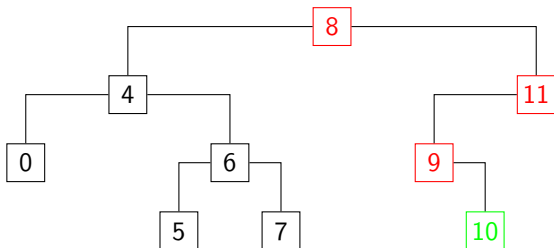


Seules les clés sont présentées.

Recherche et insertion

Recherche par clé. La valeur est comparée à la clé de la racine :

- ▶ S'il y a égalité, on renvoie l'enregistrement ;
- ▶ Si la valeur est plus petite (resp. grande) et la racine a un fils gauche (resp. droit) on itère le procédé à partir du fils gauche (resp. droit) ;
- ▶ Sinon la recherche est infructueuse.



Insertion. On applique la recherche puis on ajoute le fils manquant.

La complexité de ces opérations dépend linéairement de la *hauteur* de l'arbre.

Hypothèse probabiliste

Considérons un arbre binaire obtenu par insertion des clés $\{1, \dots, n\}$ dans un ordre obtenu par une permutation tirée uniformément.

Quelle est la hauteur moyenne de l'arbre aléatoire ainsi formé ?

Soit h_n la hauteur (aléatoire) de cet arbre.

On introduit :

- ▶ $f_n = 2^{h_n}$, la *hauteur exponentielle* de l'arbre.
- ▶ $f_{n,i}$, la *hauteur exponentielle* sachant que i est la première clé insérée.

Raisonnement probabiliste

On observe que si i est la première clé insérée,

- ▶ les clés $\{1, \dots, i-1\}$ sont insérées dans le sous-arbre gauche selon une permutation uniforme ;
- ▶ les clés $\{i+1, \dots, n\}$ sont insérées dans le sous-arbre droit selon une permutation uniforme.

$$\begin{aligned} \text{D'où :} \quad f_{n,i} &= 2^{1+\max(h_{i-1}, h_{n-i})} = 2 \cdot 2^{\max(h_{i-1}, h_{n-i})} \\ &\leq 2(2^{h_{i-1}} + 2^{h_{n-i}}) = 2(f_{i-1} + f_{n-i}) \end{aligned}$$

$$\begin{aligned} \text{D'où :} \quad \mathbf{E}(f_n) &= \frac{1}{n} \sum_{i=1}^n \mathbf{E}(f_{n,i}) \\ &\leq \frac{1}{n} \sum_{i=1}^n 2(\mathbf{E}(f_{i-1}) + \mathbf{E}(f_{n-i})) \\ &= \frac{4}{n} \sum_{i=0}^{n-1} \mathbf{E}(f_i) \end{aligned}$$

Un peu de combinatoire

Pour tout n , $\mathbf{E}(f_n) \leq n^3 + 1$

Preuve.

Elle est satisfaite pour $n = 0$ puisque $f_0 = 1$ et pour $n = 1$ puisque $f_1 = 2$.

Soit $n \geq 2$. Supposons-la satisfaite jusqu'à $n - 1$. Alors :

$$\begin{aligned}\mathbf{E}(f_n) &\leq \frac{4}{n} \sum_{i=0}^{n-1} \mathbf{E}(f_i) \leq \frac{4}{n} \sum_{i=0}^{n-1} (i^3 + 1) \\ &= \frac{4}{n} \left(\frac{1}{4} n^2 (n-1)^2 + n \right) = n(n-1)^2 + 4 \leq n^3\end{aligned}$$

Pour tout n , $\mathbf{E}(h_n) \leq 3 \log_2(n) + 1$

Preuve.

$$\begin{aligned}2^{\mathbf{E}(h_n)} &\leq \mathbf{E}(2^{h_n}) && \text{(par convexité de } 2^x \text{)} \\ &= \mathbf{E}(f_n) \leq n^3 + 1\end{aligned}$$

D'où : $\mathbf{E}(h_n) \leq \log_2(n^3 + 1) = \log_2(n^3(1 + \frac{1}{n^3})) = \log_2(n^3) + \log_2(1 + \frac{1}{n^3}) \leq 3 \log_2(n) + 1$

Plan

Equations de récurrence

Complexité amortie

Complexité en moyenne

4 Bornes inférieures de complexité

Bornes inférieures de complexité

Soit tp un type de données qui permet uniquement :

- ▶ les affectations ;
- ▶ les comparaisons.

On considère un algorithme \mathcal{A} qui trie un tableau T de n données de type tp .

On suppose que T contient initialement une permutation des données $v_1 < v_2 < \dots < v_n$ où les v_i sont toutes différentes des constantes de \mathcal{A} .

Soit σ une permutation de $\{1, \dots, n\}$ et s un état de \mathcal{A} .

L'état $\sigma(s)$ est obtenu en :

- ▶ en conservant les mêmes informations de contrôle (appels en cours, prochaines instructions, etc.) ;
- ▶ en conservant toute donnée $v \notin \{v_1, v_2, \dots, v_n\}$;
- ▶ en substituant à toute donnée v_i la donnée $v_{\sigma(i)}$.

Evolution des états

Observation critique. Considérons les états s et $\sigma(s)$ et op la prochaine instruction à exécuter.

Supposons que $s \xrightarrow{op} s'$. Si :

- ▶ op n'est pas une comparaison de données de tp
- ▶ ou op est une comparaison de données de tp et renvoie le même résultat pour s et $\sigma(s)$

Alors $\sigma(s) \xrightarrow{op} \sigma(s')$.

Dans le cas d'une comparaison sur tp qui ne renvoie pas le même résultat, on parle de comparaison *discriminante*.

Une propriété des arbres binaires

La hauteur moyenne des feuilles d'un arbre binaire comportant n feuilles est supérieure ou égale à $\log_2(n)$.

Preuve par récurrence. (*cas $n = 1$ trivial*)

Soit un arbre binaire \mathcal{A} comportant n feuilles.

Soient \mathcal{A}_g et \mathcal{A}_d ses sous-arbres comportant p et $n - p$ feuilles.

Par induction, la somme des hauteurs des feuilles de \mathcal{A}_g (resp. \mathcal{A}_d) est supérieure ou égale à $p \log_2(p)$ (resp. $(n - p) \log_2(n - p)$).

Par conséquent, la somme des hauteurs des feuilles de \mathcal{A} est supérieure ou égale à $n + p \log_2(p) + (n - p) \log_2(n - p)$.

La fonction $x \log_2(x)$ est convexe : $\frac{1}{2}(p \log_2(p) + (n - p) \log_2(n - p)) \geq \frac{n}{2} \log_2(\frac{n}{2})$.

D'où : $n + p \log_2(p) + (n - p) \log_2(n - p) \geq n + n(\log_2(n) - 1) = n \log_2(n)$.

□

Un arbre binaire

On construit itérativement un arbre binaire de la façon suivante :

- ▶ La racine r contient les $n!$ états initiaux de \mathcal{A} correspondant aux permutations de $\{v_1, v_2, \dots, v_n\}$.

Pour toute paire d'états $s_0 \neq s'_0$

il existe une permutation $\sigma \neq \text{id}$ telle que $s'_0 = \sigma(s_0)$.

- ▶ Etant donné un sommet u de l'arbre contenant un sous-ensemble S d'états identiques à une permutation près, il y a deux possibilités :
 - ▶ soit l'exécution à partir de ces états se termine sans rencontrer une comparaison discriminante.
Alors u est une feuille de l'arbre.
 - ▶ soit l'exécution à partir de ces états rencontre une comparaison discriminante.
Alors u a deux fils contenant S_{\top} et S_{\perp} les états atteints respectivement après un résultat positif ou négatif de la comparaison.

Analyse de l'arbre binaire

Toutes les feuilles de l'arbre contiennent des singletons.

Preuve.

Considérons une feuille de l'arbre.

Supposons que le sous-ensemble d'états associé n'est pas un singleton.

Considérons s et $\sigma(s)$ avec $\sigma \neq \text{id}$, deux états atteints après terminaison de \mathcal{A} .

Alors T n'est pas trié dans l'un des deux états.



Soit une distribution uniforme sur les permutations de $\{1, \dots, n\}$.
Alors le nombre moyen de comparaisons discriminantes de \mathcal{A}
est supérieur ou égal à $\log_2(n!) = \Theta(n \log(n))$.

Preuve.

Il y a $n!$ feuilles dans cet arbre binaire.

