Algorithmique et Complexité
UE 4M017
notes partielles de cours

◇ année 2017 ◇

Michel Pocchiola (michel.pocchiola@imj-prg.fr)

Résumé

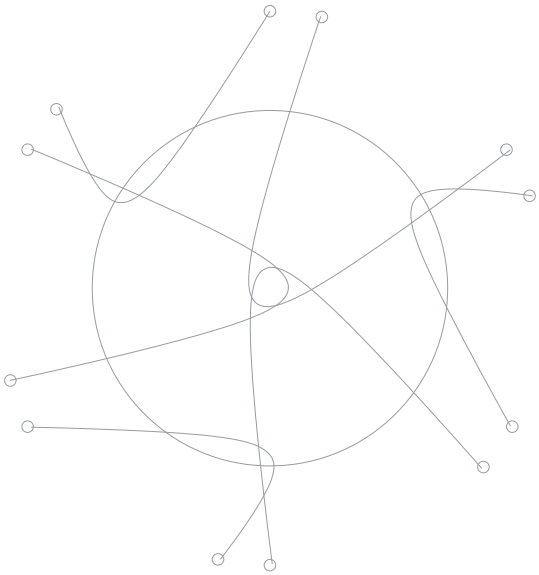


Table des matières

1 Tri et sélection par comparaisons

Notations et rappels (permutations - partie entière supérieure et partie entière inférieure d'un réel - ordre total - ordre partiel - extension linéaire d'un ordre partiel) - arbres binaires - algorithmes de tri par comparaisons et arbres de décision - borne inférieure sur le nombre de comparaisons - tri par insertion linéaire - tri par insertion dichotomique - tri fusion - tri fusion insertion - tri partition - analyse de complexité : nombre de comparaisons dans le pire des cas, nombre de comparaisons en moyenne - sélection par comparaisons

---

A TOURNAMENT PROBLEM

LESTER R. FORD, JR. AND SELMER M. JOHNSON

The American Mathematical Monthly Vol. 66, No. 5 (May, 1959), pp. 387-389

Introduction. In his book [a] Steinhaus discusses the problem of ranking n objects according to some transitive characteristic, by means of successive pairwise comparisons. In this paper we shall adopt the terminology of a tennis tournament by n players. The problem may be briefly stated : "What is the smallest number of matches which will always suffice to rank all n players ?"

Steinhaus proposes an inductive method whereby, the first k players having been ranked, the $(k+1)$-st player is matched against the median player in the first k, and by a "halving" process is finally ranked into this chain. Then the $(k+2)$-nd player is ranked into the new chain of $k+1$ players in the same manner. Using this process, a player can be ranked into a chain of k others in $S(k) = 1 + [\log_2 k]$ matches. Steinhaus thus shows that $M(n)$ matches always suffice for n players where

$$M(n) = 1 + nS(n) - 2^{S(n)}$$

He then states, "It has not been proved that there is no shorter proceeding possible, but we rather think it to be true."

The purpose of this note is to present an improved procedure, compare it with Steinhaus' $M(n)$ as an upper bound and with a lower bound $L(n)$ derived from information theory, and to discuss the asymptotic behavior of these three functions for large n.

A lower bound, $L(n)$, is easily seen to be $L(n) = 1 + [\log_2(n!)]$, since each pairing can do no more than divide the remaining possibilities into two complementary sets ; the results of the comparison then selects one or the other of these. Observing $n!$ possibilities initially, with halving the best we can do at each stage, we are led directly to the above formula for $L(n)$.

The improved procedure. .....

a. H. Steinhaus, Mathematical Snapshots, New York, 1950, pp. 37-40

(**1-**) $[\![n]\!] = \{1, 2, \ldots, n\}$, en particulier $[\![0]\!] = \varnothing$.

(**2-**) $\mathfrak{S}_n$ est l'ensemble des permutations de $[\![n]\!]$. Je rappelle que le cardinal de $\mathfrak{S}_n$ est $n!$ et que

$$n! \sim \sqrt{2\pi n}(\frac{n}{e})^n.$$

(**3-**) La **partie entière inférieure** de $x \in \mathbb{R}$, notée $\lfloor x \rfloor$, est l'unique entier relatif n tel que $n \leq x < n+1$. La **partie entière supérieure** de $x \in \mathbb{R}$, notée $\lceil x \rceil$, est l'unique entier relatif n tel que $n-1 < x \leq n$. Notons que $-\lfloor x \rfloor = \lceil -x \rceil$ (✍).

(**4-**) Les fonctions partie entière supérieure et partie entière inférieure définies sur $\mathbb{R}$ et à valeurs dans $\mathbb{Z}$ sont exactement les fonctions $f : \mathbb{R} \to \mathbb{Z}$ vérifiant les deux propriétés suivantes : pour tout $x \in \mathbb{R}$, tout $c \in \mathbb{Z}$ et tout $n \in \mathbb{N}$

$$\begin{aligned} f(x+c) &= f(x)+c \\ f(f(x)/n) &= f(x/n). \end{aligned}$$

Le lecteur démontrera à titre d'exercice la partie directe (✍) et pourra se reporter pour la réciproque à P. Eisele and K. P. Hadeler. Game of cards, dynamical systems, and a characterization of the floor and ceiling functions. *Amer. Math. Monthly*, 97(6) :466–477, June-July 1990.

(**5-**) Le logarithme à base deux est noté lg.
Notons que pour tout réel $x > 0$ les entiers $\lfloor \lg x \rfloor$ et $\lceil \lg x \rceil$ sont définis par les relations

$$\begin{aligned} 2^{\lfloor \lg x \rfloor} &\leq x < 2^{\lfloor \lg x \rfloor + 1} \\ 2^{\lceil \lg x \rceil - 1} &< x \leq 2^{\lceil \lg x \rceil}. \end{aligned}$$

Par suite pour tout entier $n \geq 1$ (✍)

$$\lfloor \lg n \rfloor + 1 = \lceil \lg(n+1) \rceil$$

et

$$\lceil \lg(n+1) \rceil - \lceil \lg n \rceil = \begin{cases} 1, & \text{si } n \text{ est une puissance de deux,} \\ 0, & \text{sinon.} \end{cases}$$

(**6-**) *ordre partiel, ordre total, diagramme de Hasse d'un ordre partiel, extension linéaire d'un ordre partiel.*

(**7-**) Le n-ième nombre harmonique, noté H_n, est défini comme la somme des inverses des entiers de 1 à n :

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

Je rappelle (ou vous apprend) que

$$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + O(\frac{1}{n^4})$$

où $\gamma = 0.577\,215\,664\ldots$ est une constante dite *constante d'Euler*.

(**8-**) Le n-ième nombre de Catalan, noté C_n, est défini par

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

Un simple calcul (✍) montre que $C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$.

Exercice 1.1. Montrer que pour $n \geq 1$ l'entier $u_n = \lceil \lg(n+1) \rceil$ est le nombre de chiffres de l'écriture en base 2 de n et que la suite $(u_n)_{n\geq 1}$ est définie par la relation de récurrence $u_1 = 1$ et $u_n = 1 + u_{\lfloor n/2 \rfloor}$ pour $n \geq 2$. 1.1

Exercice 1.2. Quel est le nombre d'extensions linéaires de la somme disjointe de deux ordres totaux sur des ensembles de cardinalité n et m ? 1.2

Arbres binaires. Un **arbre binaire** d'ordre $2n+1$ ($n \geq 0$) est la donnée d'un ensemble N de $2n+1$ éléments, appelés **nœuds**, d'une bipartition de N formée d'un ensemble N_i de n nœuds dits **internes** et d'un ensemble N_e de $n+1$ nœuds dits **externes** et de deux applications injectives $f_G : N_i \to N$ et $f_D : N_i \to N$, appelées **fonction fils gauche et fonction fils droit** telles que

(**1-**) $N - f_G(N_i) - f_D(N_i)$ se réduit à un élément r, appelé la racine de l'arbre ;

(**2-**) pour tout $x \in N$ il existe une unique suite $X_1, X_2, \ldots, X_m$, $X_i \in \{G, D\}$ telle que

$$x = f_{X_m} \circ f_{X_{m-1}} \circ \cdots \circ f_{X_2} \circ f_{X_1}(r).$$

La suite $X_1, X_2, \ldots, X_m$ est appelée le **chemin de recherche** du nœud x. En particulier le chemin de recherche de la racine est la suite vide.

Les paires $(x, f_G(x))$ et $(x, f_D(x))$ sont appelées les **arêtes gauches et droites** de l'arbre binaire. À **isomorphisme** près un arbre binaire ne dépend que de l'ensemble des

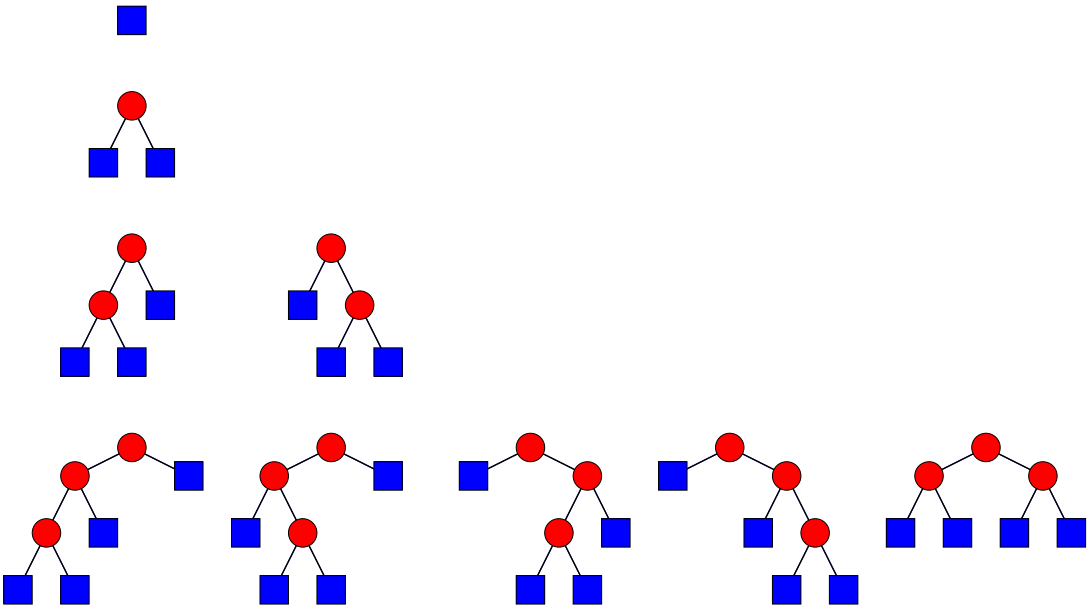


FIGURE 1 – Liste exhaustive à isomorphisme près des arbres binaires d'ordre $1, 3, 5$ et 7 : les disques représentent les nœuds internes, les carrés les nœuds externes, les segments de droites de pente positive les arêtes gauches et les segments de droites de pente négative les arêtes droites.

chemins de recherche de ses nœuds externes (✍).

La **hauteur** d'un arbre binaire est le maximum des longueurs de ses chemins de recherche.

Théorème 1.1. *Un arbre binaire de hauteur h a au plus 2^h nœuds externes. (En d'autres termes un arbre à k nœuds externes est de hauteur $\geq \lceil \lg k \rceil$.)*

Démonstration. Par induction sur h. □

Algorithmes de tri par comparaisons. Une **liste** a de longueur n, $n \in \mathbb{N}$, est un n-uplet $(a_1, a_2, \ldots, a_n)$ d'éléments distincts deux à deux d'un ensemble X muni d'un ordre total $<$. L'unique permutation σ de $\mathfrak{S}_n$ telle que la suite $a \circ \sigma = (a_{\sigma(1)}, a_{\sigma(2)}, \ldots, a_{\sigma(n)})$ soit croissante, i.e., $a_{\sigma(1)} < a_{\sigma(2)} < \cdots < a_{\sigma(n)}$, est notée $\omega(a)$.

Par exemple, si X est l'ensemble des mots sur l'alphabet latin moderne muni de l'ordre lexicographique $<$ et si $a = (aspic, assiette, arbre, artiste, aspirateur)$ alors

$$\omega(a) = (3, 4, 1, 5, 2).$$

En effet $arbre < artiste < aspic < aspirateur < assiette$ et, par suite,

$$\begin{aligned} \omega(a) &= a^{-1} \circ a \circ \omega(a) \\ &= (aspic, assiette, arbre, artiste, aspirateur)^{-1} \\ &\quad \circ (arbre, artiste, aspic, aspirateur, assiette) \\ &= (3, 4, 1, 5, 2) \end{aligned}$$

Un **comparateur** est une paire ordonnée (i, j) d'indices distincts et sa **valeur de vérité** en la liste a est vraie ou fausse selon que $a_i < a_j$ ou $a_j < a_i$. Une suite de comparateurs **tri** la liste a (i.e., détermine $\omega(a)$ ou $a \circ \omega(a)$) si et seulement si l'ordre partiel $\prec$ sur les a_i induit par les valeurs de vérité en la suite a des comparateurs est un ordre total (nécessairement la restriction aux a_i de la relation d'ordre total $<$ sur X).

Par exemple la suite de 7 comparateurs

$$(1,2), (3,4), (2,4), (5,4), (5,2), (1,4), (1,5)$$

tri lexicographiquement la liste de mots $(aspic, assiette, arbre, artiste, aspirateur)$: en effet l'ordre induit par les valeurs de vérité des comparateurs

comparateur		valeur de vérité	ordre induit	
$(1,2)$	$aspic <?assiette$	vrai	$aspic \prec assiette$	
$(3,4)$	$arbre <?artiste$	vrai	$arbre \prec artiste$	1
$(2,4)$	$assiette <?artiste$	faux	$artiste \prec assiette$	
$(5,4)$	$aspirateur <?artiste$	faux	$artiste \prec aspirateur$	
$(5,2)$	$aspirateur <?assiette$	vrai	$aspirateur \prec assiette$	4
$(1,4)$	$aspic <?artiste$	faux	$artiste \prec aspic$	2
$(1,5)$	$aspic <?aspirateur$	vrai	$aspic \prec aspirateur$	3

est un ordre total

$$arbre \underset{1}{\prec} artiste \underset{2}{\prec} aspic \underset{3}{\prec} aspirateur \underset{4}{\prec} assiette.$$

Un **algorithme de tri par comparaisons** est un algorithme de tri dont l'unique opération primitive est la comparaison : un tel algorithme produit pour toute liste a

une suite $C(a)$ de comparateurs qui tri la liste a, suite soumise à la condition suivante : un comparateur de la suite ne dépend que des comparateurs qui le précédent dans la suite et de leurs valeurs de vérité en la suite. En d'autres termes si pour tout $j < i$ les comparateurs de rang j associés aux listes de même longueur a et a' sont bien définis, égaux et ont la même valeur de vérité alors les comparateurs de rang i associés aux listes a et a' sont, sous l'hypothèse d'être bien définis, égaux. En particulier les suites de comparateurs associées aux listes de même longueur commencent toutes par le même comparateur.

Pour un algorithme de tri par comparaisons donné et un entier donné n l'ensemble des suites de comparateurs $C(a) = C_1(a)C_2(a)\dots C_{m(a)}(a)$ associées aux listes a de longueur n peut ainsi être représenté par un arbre binaire, appelé **arbre de décision** pour des données de taille n, dont les chemins de recherche de ses nœuds externes sont les suites

$$E_1(a)E_2(a)\dots E_{m(a)}(a)$$

où $E_j(a) = G$ ou D selon que la valeur de vérité $R_j(a)$ du comparateur $C_j(a)$ en a est vrai ou fausse et dont les nœuds sont étiquetés comme suit : le nœud interne de chemin de recherche $E_1(a)E_2(a)\dots E_k(a)$, $0 \le k < m(a)$, est étiqueté par le comparateur $C_{k+1}(a)$ et le nœud externe de chemin de recherche $E_1(a)E_2(a)\dots E_{m(a)}(a)$ est étiqueté par $\omega(a)$.

Ici je donne l'exemple du tri fusion insertion d'une liste de 5 éléments [du à H. B. Demuth, PhD thesis, 1956] : ✍ — L'algorithme de tri de Demuth de la liste $(a_1, a_2, a_3, a_4, a_5)$ commence par calculer les indices u, u', v, v', w et t tels que $a_u = \max(a_1, a_2)$, $a_{u'} = \min(a_1, a_2)$, $a_v = \max(a_3, a_4)$, $a_{v'} = \min(a_3, a_4)$, $a_w = \max(a_u, a_v)$ et $a_t = \min(a_u, a_v)$ avec la suite de comparateurs $(1,2), (3,4), (u,v)$. Ainsi

$$(u,u') = \begin{cases} (2,1) & \text{si } a_1 < a_2\,; \\ (1,2) & \text{sinon} \end{cases}, \quad (v,v') = \begin{cases} (4,3) & \text{si } a_3 < a_4\,; \\ (3,4) & \text{sinon} \end{cases}$$

et

$$(w,t) = \begin{cases} (v,u) & \text{si } a_u < a_v\,; \\ (u,v) & \text{sinon} \end{cases}$$

A cette étape du déroulement de l'algorithme l'ordre partiel induit par les valeurs de vérités des comparateurs est donné par les relations $a_{t'} < a_t < a_w$ et $a_{w'} < a_w$. L'algorithme poursuit alors en insérant à sa place a_5 dans la chaîne de longueur trois $a_{t'} < a_t < a_w$ en deux comparaisons en commençant avec le comparateur $(t,5)$ puis avec le comparateur $(w,5)$ ou $(t',5)$ selon que $a_t < a_5$ ou $a_5 < a_t$. L'ordre partiel induit à l'issue de cette deuxième étape est alors donné par les relations $a_{t'} < a_t < a_w < a_5$ et $a_{w'} < a_w$, ou par les relations $a_\alpha < a_\beta < a_\gamma < a_w$ et $a_{w'} < a_w$ avec $(\alpha, \beta, \gamma) = (5, t', t)$ ou $(t', 5, t)$ ou $(t', t, 5)$. Dans les deux cas il reste à insérer $a_{w'}$ dans une chaîne de longueur au plus 3, ce qui s'effectue à nouveau en au plus 2 comparaisons avec la méthode de l'étape précédente. Au final l'algorithme de Demuth tri les listes de cinq éléments en 6 ou 7 comparaisons selon les cas. L'arbre de décision associé à l'algorithme de tri de Demuth est décrit dans les trois figures suivantes

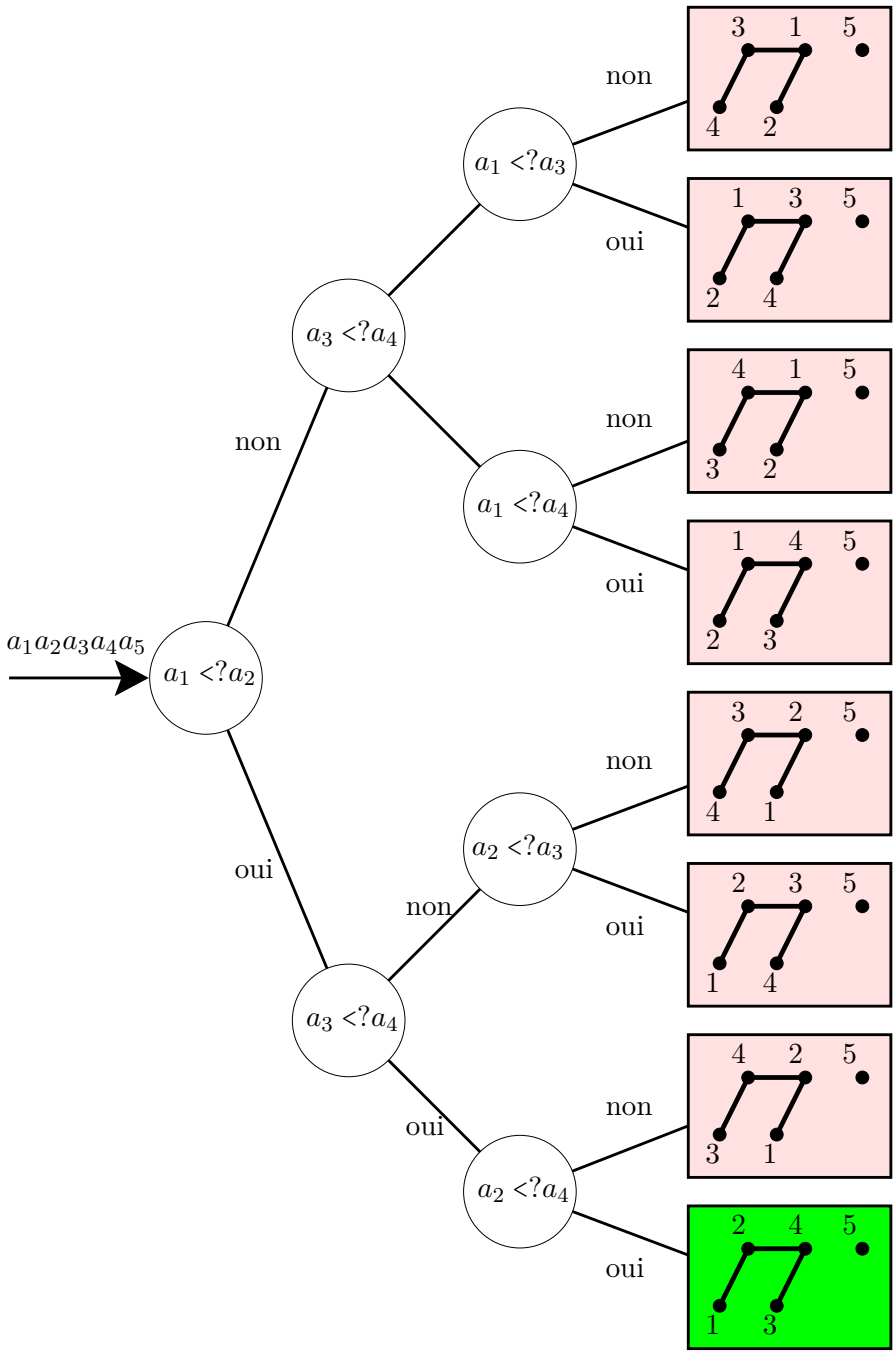


FIGURE 2 – Arbre de décision de l'algorithme de Demuth à l'issue de la première étape, i.e. après application des comparateurs $(1,2),(3,4)$ et (u,v).

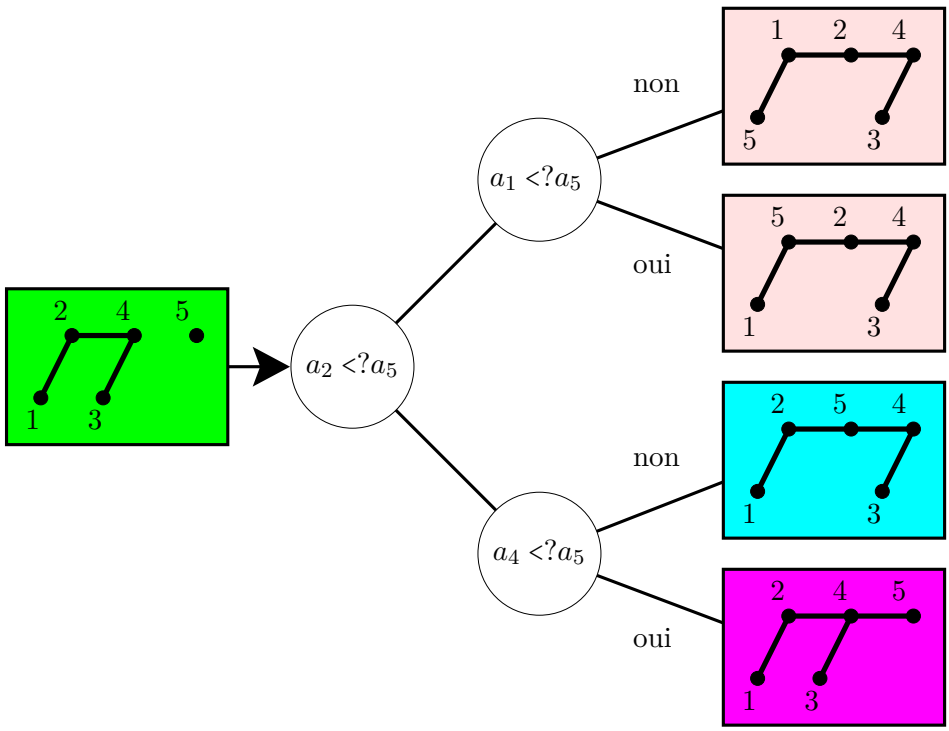


FIGURE 3 – Deuxième étape de l'algorithme de tri Demuth : insertion de a_5 dans la chaîne de longueur trois obtenue lors de la première étape dans le cas $(u, v, w) = (2, 4, 4)$ - les autres cas sont entièrement similaires à reindexation près

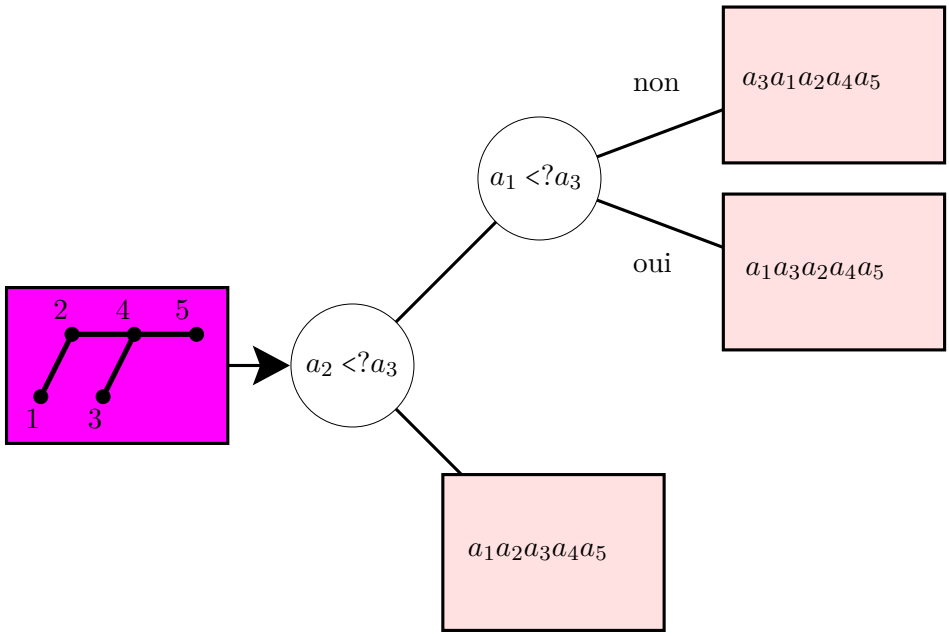


FIGURE 4 – Troisième et dernière étape de l'algorithme de Demuth dans le cas de l'insertion du dernier élément dans une liste de longueur deux : insertion de a_3 dans la chaîne de longueur deux obtenue lors des deux premières étapes dans le cas $(u, v, w) = (2, 4, 4)$ et $a_4 < a_5$ - les autres cas sont similaires à réindexation près

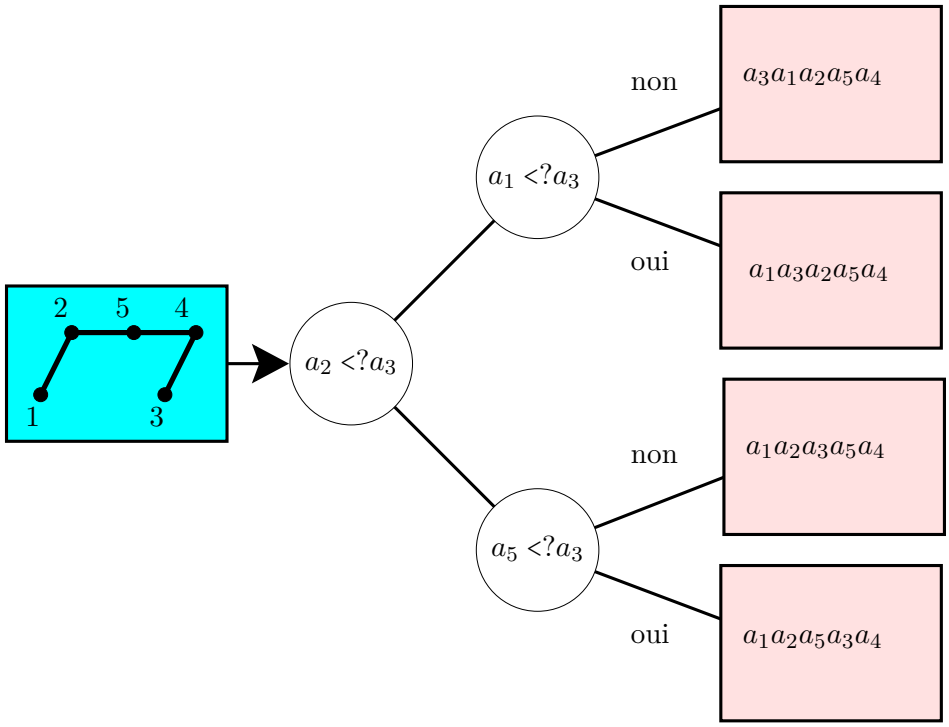


FIGURE 5 – Troisième et dernière étape de l'algorithme de Demuth dans le cas de l'insertion du dernier élément dans une liste de longueur trois : insertion de a_3 la chaîne de longueur trois obtenue lors des deux premières étapes dans le cas $(u, v, w) = (2, 4, 4)$ et $a_2 < a_5 < a_4$ - les autres cas sont similaires à réindexation près

On remarquera que l'ensemble des algorithmes de tri par comparaisons est indépendant du choix de X et que l'on peut se restreindre à considérer pour listes les éléments de $\mathfrak{S}$, auquel cas $\omega(a)$ est l'inverse de la permutation a.

Dans la suite nous étudions quatres algorithmes classiques de tri par comparaisons : le tri par insertion linéaire, le tri par insertion dichotomique, le tri fusion et le tri fusion-insertion. Et pour chacun de ces algorithmes nous étudions sa complexité dans le pire des cas, i.e., le maximum des nombres de comparaisons nécessaires pour trier les listes de longueur n. C'est aussi la hauteur de l'arbre de décision associé. Nous pouvons immédiatement noter une borne inférieure sur cette complexité.

Théorème 1.2. *Le nombre de comparaisons dans le pire cas de tout algorithme de tri par comparaisons sur des données de taille n est minoré par la partie entière supérieure du logarithme en base deux de la factorielle de n :*

$$\lceil \lg n! \rceil.$$

Démonstration. L'arbre de décision associé a $n!$ nœuds externes. Par suite sa hauteur est $\geq \lceil \lg n! \rceil$. La hauteur de l'arbre correspond au nombre de comparaisons dans le pire cas. □

Le tableau des premières valeurs de $\lceil \lg n! \rceil$:

n	$\longmapsto$	$\lceil \lg n! \rceil$
1		0
2		1
3		3
4		5
5		7
6		10
7		13
8		16
9		19
10		22
11		26
12		29
13		33

montre que l'algorithme de tri d'une liste de cinq éléments de Demuth est optimal dans le pire des cas.

Insertion linéaire dans une liste triée. L'**insertion linéaire** d'un élément dans une liste triée selon les valeurs croissantes de ses éléments consiste à insérer l'élément à sa place en le comparant successivement avec l'élément de rang maximal puis, si cela est nécessaire i.e. si l'élément à insérer est strictement inférieur à l'élément de rang maximal, avec l'élément de rang le prédécesseur du rang maximal, etc., etc.

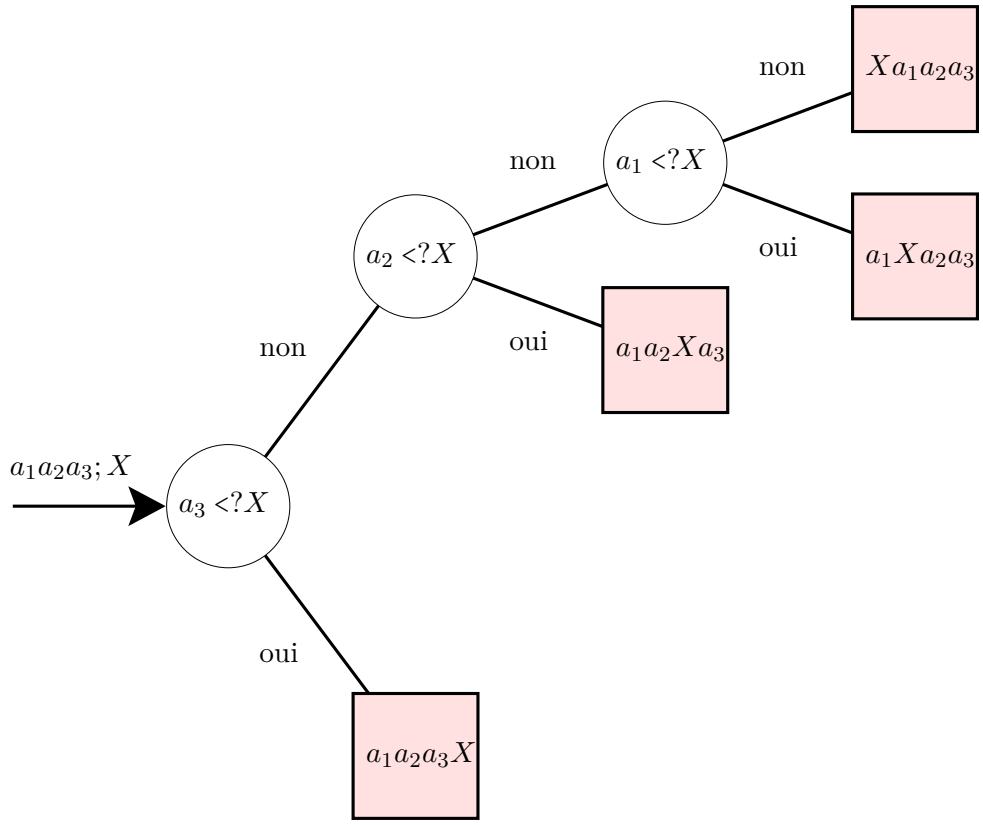


FIGURE 6 – Arbre de décision de l'insertion linéaire d'un élément dans une liste de longueur 3

Le nombre de comparaisons de l'insertion linéaire d'un élément dans une liste triée est donc majoré par la longueur de la liste et cette borne est atteinte si l'élément à insérer est le plus petit élément, au sens strict du terme, ou le deuxième plus petit élément de la liste augmentée de l'élément à insérer.

Tri par insertion linéaire. Le **tri par insertion linéaire** d'une liste de n éléments consiste pour k variant de 1 à $n-1$ à insérer à sa place, selon la méthode de l'insertion linéaire, l'élément de rang $k+1$ de la liste initiale dans la liste triée de ses k premiers éléments.

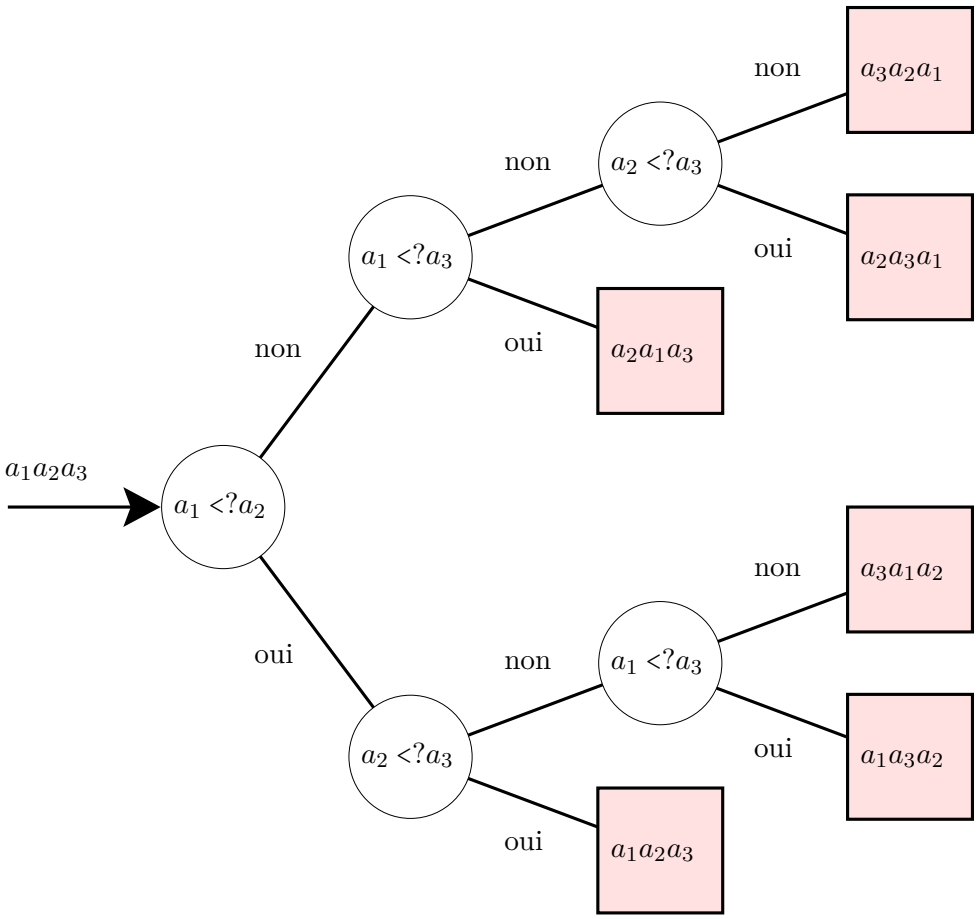


FIGURE 7 – Arbre de décision du tri par insertion linéaire pour les entrées de taille 3

Théorème 1.3. *Le nombre de comparaisons de l'algorithme de tri par insertion linéaire de n éléments est majoré par*

$$\sum_{k=1}^{n-1} k = n(n-1)/2.$$

De plus cette borne est atteinte par exemple si la liste initiale est triée selon les valeurs décroissantes de ses éléments, supposés distincts deux à deux.

Démonstration. Le nombre de comparaisons de l'algorithme de tri par insertion linéaire d'une liste de n éléments est la somme pour k variant de 1 à $n-1$ du nombre C_k de comparaisons de l'insertion linéaire de l'élément de rang $k+1$ dans la liste triée des k

premiers éléments de la liste initiale. Nous avons vu que C_k est majoré par k et que cette borne est atteinte dans le cas où l'élément de rang $k+1$ est le plus petit élément ou le deuxième plus petit élément de la liste des $k+1$ premiers éléments de la liste initiale. Le résultat suit. □

Exercice 1.3. Quel est le nombre de listes de longueur n sur un ensemble totalement ordonné à n éléments dont le tri par insertion linéaire nécessite le nombre maximal de comparaisons ? 1.3

Exercice 1.4. Calculer l'espérance du nombre de comparaisons de l'algorithme de tri par insertion linéaire d'une liste choisie de manière aléatoire selon la loi uniforme dans l'ensemble des listes de longueur n sur un ensemble totalement ordonné de n éléments. 1.4

Exercice 1.5. [**Codage d'une permutation**] Soit τ_{ij} la permutation de $\{1, 2, \ldots, n\}$ qui échange i et j et laisse invariant les autres éléments ; en particulier τ_{ii} est la permutation identité. Montrer que pour toute permutation σ de $\{1, 2, \ldots, n\}$ il existe une unique suite $a_1, a_2, \ldots, a_n$ d'entiers, $1 \le a_i \le i$, telle que σ soit le produit (ou la composée) des τ_{ia_i} pour i variant de n à 1 :

$$\begin{aligned}\sigma &= \prod_{i=n}^{1} \tau_{ia_i} \\ &= \tau_{na_n} \circ \cdots \circ \tau_{3a_3} \circ \tau_{2a_2} \circ \tau_{1a_1}.\end{aligned}$$

1.5

Exercice 1.6. [**Permutation aléatoire**] Quelle est la sortie de l'algorithme Random-Shuffling défini ci-dessous ?

```
procedure Random-Shuffling (var A : array[1..n] of typekey) ;
1. for i := 1 to n do
2.    begin
3.       j := random-number(1, i) ;
4.       swap(A[i], A[j]) ;
5.    end
```

1.6

Insertion dichotomique d'un élément dans une liste triée. L'insertion dichotomique d'un élément dans une liste triée selon les valeurs croissantes de ses éléments consiste à comparer l'élément à insérer avec l'élément de rang médian de la liste dans laquelle on insére et à recommencer récursivement sur la sous-liste des éléments de rang strictement inférieur au rang médian ou sur la sous-liste des éléments de rang strictement supérieur au rang médian selon que l'élément à insérer est strictement inférieur ou strictement supérieur à l'élément de rang médian.

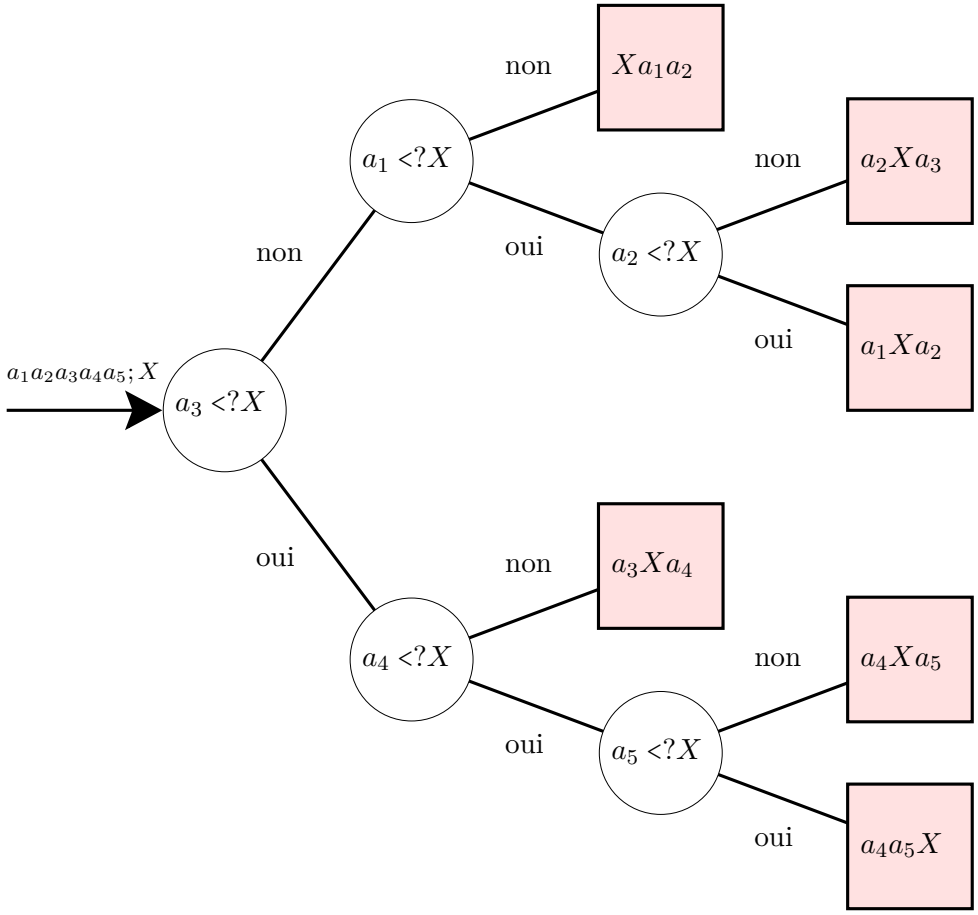


FIGURE 8 – Arbre de décision de l'insertion dichotomique d'un élément dans une liste de longueur 5 (pour des raisons d'espace nous avons simplement indiqué dans les feuilles la position de l'élément inséré)

Le rang médian d'un liste de n éléments est par définition (dans ces notes de cours) le rang $\lceil n/2 \rceil$. Ainsi la sous-liste des éléments de rang strictement inférieur au rang médian est de longueur $\lceil n/2 \rceil - 1 = \lfloor n/2 \rfloor - 1$ ou $\lfloor n/2 \rfloor$ selon que n est pair ou impair ; tandis que la sous-liste des éléments de rang strictement supérieur au rang médian est de longueur $n - \lceil n/2 \rceil = \lfloor n/2 \rfloor$. Ainsi le nombre $C(n)$ de comparaisons dans le pire cas de l'insertion dichotomique d'un élément dans une liste triée de n éléments vérifie la

récurrence $C(n) = 1 + C(\lfloor n/2 \rfloor)$ pour $n \geq 1$ et $C(0) = 0$. Cette récurrence se résoud en $C(n) = \lceil \lg(n+1) \rceil$ pour tout entier n (cf. Exercice 1.1).

Lemme 1.4. *L'insertion dichotomique d'un élément dans une liste ordonnée de $n \geq 0$ éléments nécessite au plus $\lceil \lg(n+1) \rceil$ comparaisons et cette borne est atteinte. En d'autres termes l'insertion dichotomique dans une liste ordonnée de $< 2^k$ éléments nécessite $\leq k$ comparaisons et cette borne est atteinte.* □

Exercice 1.7. Montrer que le nombre de comparaisons dans le pire cas de tout algorithme d'insertion par comparaisons d'un élément dans une liste triée de n éléments est supérieur ou égal à la partie entière supérieure du logarithme à base de deux de $n+1$:

$$\lceil \lg(n+1) \rceil.$$

1.7

Tri par insertion dichotomique. Le **tri par insertion dichotomique** d'une liste de n éléments (mentionné par Steinhaus [10, pages 38-39], [11, Chap. 3]) consiste pour k variant de 1 à $n-1$ à insérer selon la méthode de l'insertion dichotomique l'élément de rang $k+1$ de la liste initiale dans la liste triée de ses k premiers éléments.

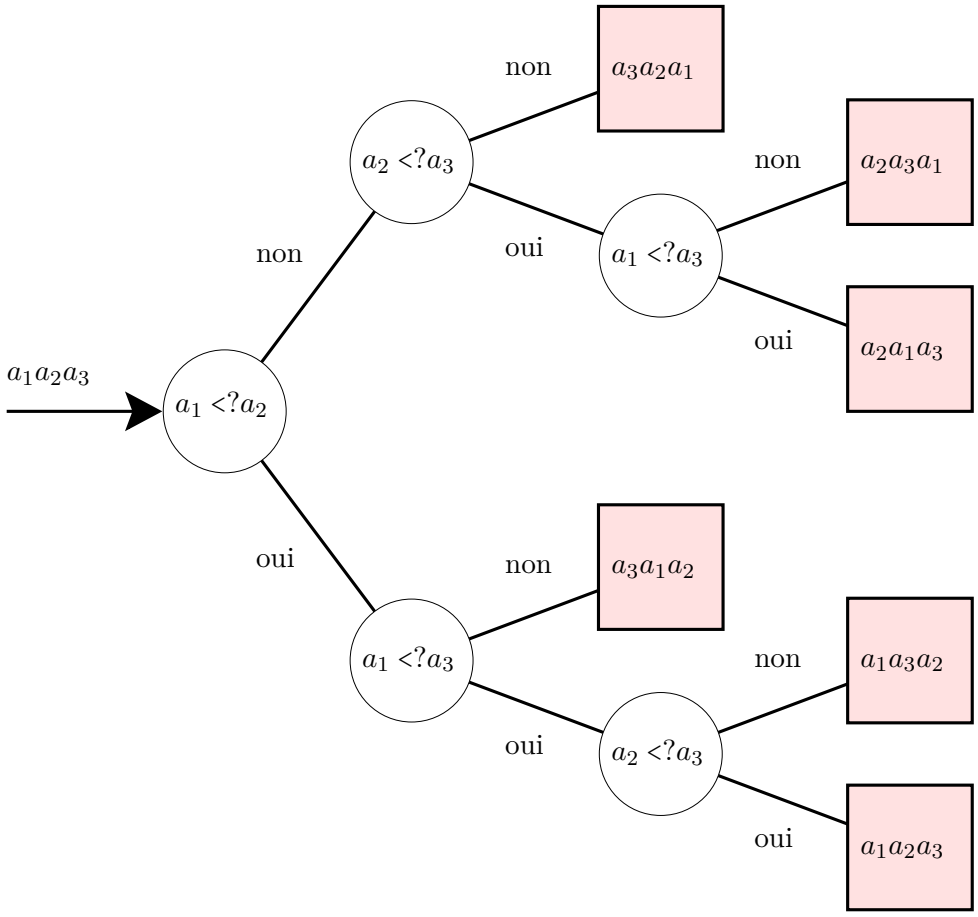


FIGURE 9 – Arbre de décision du tri par insertion dichotomique pour les entrées de taille 3

Théorème 1.5. *Le nombre de comparaisons de l'algorithme de tri par insertion dichotomique de n éléments est majoré par*

$$B(n) = \sum_{k=1}^{n} \lceil \lg k \rceil = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1.$$

De plus cette borne est atteinte.

Démonstration. Le nombre de comparaisons du tri par insertion dichotomique d'une liste de n éléments est égal à la somme pour k variant de 1 à $n-1$ du nombre $C(k)$ de

comparaisons pour insérer selon la méthode dichotomique l'élément de rang $k+1$ dans la liste triée des k premiers éléments de la liste. Nous avons vu que $C(k)$ est majoré par $\lceil \lg(k+1) \rceil$ et que cette borne est atteinte. Le résultat suit (nous laissons au lecteur le soin d' exhiber une ou des listes de longueur n nécessitant ce nombre de comparaisons). Pour la forme close on utilise la transformation d'Abel [1]

$$\begin{aligned}\sum_{1\le k\le n} a_k &= \sum_{1\le k\le n} (k-(k-1))a_k \\ &= na_n + \sum_{1\le k\le n-1} ka_k - \sum_{0\le k\le n-1} ka_{k+1} \\ &= na_n - \sum_{1\le k<n} k(a_{k+1}-a_k)\end{aligned}$$

et

$$\lceil \lg(k+1) \rceil - \lceil \lg k \rceil = \begin{cases} 1, & \text{si } k \text{ est une puissance de 2,} \\ 0, & \text{sinon} \end{cases}$$

pour écrire

$$\begin{aligned} B(n) &= \sum_{1\le k\le n} \lceil \lg k \rceil \\ &= n\lceil \lg n \rceil - \sum_{1\le k<n} k(\lceil \lg(k+1) \rceil - \lceil \lg k \rceil) \\ &= n\lceil \lg n \rceil - \{1+2+2^2+\cdots+2^\ell\} \\ &= n\lceil \lg n \rceil - (2^{\ell+1}-1) \end{aligned}$$

avec $2^\ell < n \le 2^{\ell+1}$ soit $\ell+1 = \lceil \lg n \rceil$. □

1. « Recherches sur la série [binomiale] », p. 219-251 in Niels Abel, Œuvres Complètes, éditées par Ludwig Sylow et Sophus Lie, Christiania 1881, réédition (Gabay) 1992 ; article paru à l'origine dans Journal für die reine und angewandte Mathematik, Berlin, bd. I, 1826. Untersuchungen über die Reihe: 1 + ... u.s. w. ou ici , ou ici , ou encore ici

Les premières valeurs de $B(n)$

n	$\longmapsto$	$\lceil \lg n! \rceil$	$\lceil \lg n \rceil$	$B(n)$
1		0	0	0
2		1	1	1
3		3	2	3
4		5	2	5
5		7	3	8
6		10	3	11
7		13	3	14
8		16	3	17
9		19	4	21
10		22	4	25
11		26	4	29
12		29	4	33
13		33	4	37
14		37	4	41
15		41	4	45
16		45	4	49
17		49	5	54

montrent que le tri par insertion dichotomique est optimal dans le pire des cas pour $n \leq 4$.

Le tri fusion. [A la base des méthodes de tri externe] Le tri fusion d'une liste de n éléments consiste à scinder la liste en deux sous-listes, une sous-liste de taille $\lfloor n/2 \rfloor$ et l'autre de taille $\lceil n/2 \rceil$, à trier récursivement les deux sous-listes selon la méthode du tri fusion puis à fusionner les deux sous-listes triées en commençant par comparer leurs éléments de rang 1 afin de déterminer le plus petit élément de la liste initiale puis à réitérer ce processus de comparaison après suppression du plus petit élément de la sous-liste à laquelle il appartient afin de déterminer le second plus petit élément de la liste initiale puis le troisième plus petit élément de la liste initiale etc. etc. jusqu'à déterminer le grand élément de la liste initiale.

Soit $F(n,m)$ le nombre de comparaisons dans le pire des cas de la fusion d'une liste de taille $n \geq 0$ et d'un liste de taille $m \geq 0$. Alors pour tout $n \geq 0$

$$F(0,n) = F(n,0) = 0$$

et pour tout $n, m \geq 1$

$$F(n,m) = \max\{1 + F(n,m-1), 1 + F(n-1,m)\}.$$

Lemme 1.6. *Pour tout* $n, m \geq 1$

$$F(n,m) = n + m - 1.$$

Démonstration. Procéder par récurrence sur $n+m$. □

Par suite le nombre $C(n)$ de comparaisons dans le pire des cas du tri-fusion vérifie la relation de récurrence pour tout $n \geq 2$:

$$C(n) = n - 1 + C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil)$$

pour tout $n \geq 2$ avec pour valeur intiale $C(1) = 0$. Cette relation de récurrence se résoud en $C(n) = n \lg n - n + 1$ dans le cas particulier où n est une puissance de 2. En effet posons $A(u) = C(2^u)/2^u$. La récurrence s'écrit alors

$$A(u) = A(u-1) + 1 - 1/2^u$$

d'où

$$\begin{aligned} A(u) &= u - (1/2^u + 1/2^{u-1} + \cdots + 1/2) \\ &= u - 1/2 \frac{1 - 1/2^u}{1 - 1/2} \\ &= u - \frac{2^u - 1}{2^u} \end{aligned}$$

Soit

$$C(2^u) = 2^u u - 2^u + 1.$$

Ainsi pour n puissance de deux le coût du tri-fusion dans le pire des cas est le même que celui du tri par insertion dichotomique. Qu'en est-il pour n quelconque ?

Théorème 1.7. *Le coût en nombre de comparaisons du tri-fusion dans le pire cas est le même que celui du tri par insertion dichotomique.*

Démonstration. Soit $B(n)$ le nombre de comparaisons du tri par insertion dichotomique. Montrons par récurrence sur n que $B(n)$ vérifie la même relation de récurrence que $C(n)$.

Supposons $n \geq 3$. Posons $u = \lceil \lg n \rceil$. Ainsi $2 \leq 2^{u-1} < n \leq 2^u$ avec $u \geq 2$.

Si $n = 2m$ alors $1 \leq 2^{u-2} < m \leq 2^{u-1}$ et, par suite, $u - 1 = \lceil \lg m \rceil$.

Ainsi

$$B(\lfloor n/2 \rfloor) = B(\lceil n/2 \rceil) = B(m) = m(u-1) - 2^{u-1} + 1$$

$$\begin{aligned} B(n) &= nu - 2^u + 1 \\ &= 2m(u-1) - 22^{u-1} + 2 + 2m - 1 \\ &= n - 1 + B(\lfloor n/2 \rfloor) + B(\lceil n/2 \rceil) \end{aligned}$$

Si $n = 2m + 1$ alors $1 \leq 2^{u-2} < m + 1/2 \leq 2^{u-1}$ et, par suite, $2^{u-2} < m + 1 \leq 2^{u-1}$, $u - 1 = \lceil \lg(m+1) \rceil$ et

$$B(m+1) = (m+1)(u-1) - 2^{u-1} + 1$$

D'autre part $2^{u-2} \leq m < 2^{u-1}$. Distinguons alors deux cas :

si $2^{u-2} < m$ alors $u - 1 = \lceil \lg m \rceil$ et

$$B(m) = m(u-1) - 2^{u-1} + 1$$

et

$$B(m) + B(m+1) = nu - n - 2^u + 2 = B(n) - n + 1$$

soit

$$B(n) = n - 1 + B(\lfloor n/2 \rfloor) + B(\lceil n/2 \rceil)$$

si $2^{u-2} = m$ alors $u - 2 = \lceil \lg m \rceil$ et

$$B(m) = m(u-2) - 2^{u-2} + 1 = m(u-1) - 2^{u-1} + 1$$

comme précédemment. Ainsi dans ce cas encore on a

$$B(n) = n - 1 + B(\lfloor n/2 \rfloor) + B(\lceil n/2 \rceil)$$

Ainsi $C(n)$ et $B(n)$ vérifient la même récurrence donc $B(n) = C(n)$. □

Tri fusion insertion. Le tri **fusion insertion**, attribué à Lester Ford, Jr et Selmer Johnson [4], opère comme suit

(**1-**) Scinder la liste en $\lfloor n/2 \rfloor$ sous-listes de taille deux plus, si n est impair, une sous-liste de taille 1.

(**2-**) Comparer les éléments de chaque sous-liste de taille deux.

(**3-**) Trier les $\lfloor n/2 \rfloor$ plus grands éléments de l'étape précédente par **fusion insertion**.

On note $a_1 < a_2 < \ldots < a_{\lfloor n/2 \rfloor}$ et $b_i < a_i$ les $2\lfloor n/2 \rfloor$ éléments partiellement ordonnés à la suite des trois premières étapes et on note $b_{\lceil n/2 \rceil}$ l'élément de la sous-liste de taille 1 dans le cas n impair. La suite $b_1, a_1, a_2, \ldots, a_{\lfloor n/2 \rfloor}$ est appelée la chaine principale.

(**4-**) Insérer les b_i dans la chaine principale par insertion dichotomique dans l'ordre suivant

$$
\begin{array}{l}
b_3, b_2; \\
b_5, b_4; \\
b_{11}, b_{10}, b_9, b_8, b_7, b_6; \\
b_{21}, b_{20}, b_{19}, b_{18}, b_{17}, b_{16}, b_{15}, b_{14}, b_{13}, b_{12}; \\
\ldots \\
b_{t_k}, b_{t_k-1}, \ldots, b_{t_{k-1}+1}; \\
\ldots
\end{array}
$$

où $(t_1, t_2, t_3, t_4, t_5 \ldots) = (1, 3, 5, 11, 21, \ldots)$ est définie par la relation de récurrence $t_0 = 1, t_{k-1} + t_k = 2^k$.

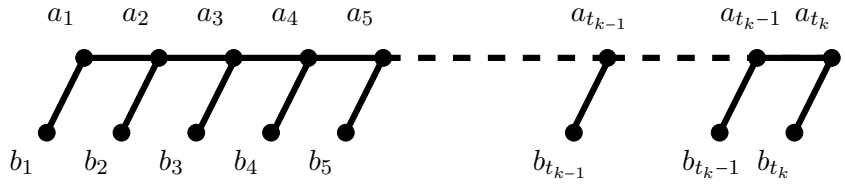


FIGURE 10 –

Exercice 1.8. Montrer que $t_k = \{2^{k+1} + (-1)^k\}/3$. 1.8

Noter que pour $n = 5$ l'algorithme de Ford et Johnson est celui de Demuth.

Lemme 1.8. *Les éléments $b_{t_k}, b_{t_k-1}, \ldots, b_{t_{k-1}+1}$ sont insérés dans la chaîne principale en au plus k comparaisons.*

Démonstration. La chaine principale dans laquelle on insère $b_{t_k}, b_{t_k-1}, \ldots, b_{t_{k-1}+1}$ a pour longueur au plus $t_k + t_{k-1} - 1$, cf. Figure 10 ; cette longueur étant $< 2^k$ le lemma suit par application du Lemme 1.4. □

Soit $F(n)$ le nombre maximal de comparaisons nécessaires pour trier n éléments par fusion insertion.

L'analyse de fusion insertion conduit à la récurrence

$$F(n) = \lfloor n/2 \rfloor + F(\lfloor n/2 \rfloor) + G(\lceil n/2 \rceil) \tag{1}$$

où $G(\lceil n/2 \rceil)$ est le nombre de comparaisons nécessaires à l'étape 4.

Si $t_{k-1} \leq m \leq t_k$ on a

$$\begin{aligned} G(m) &= \sum_{j=1}^{k-1} j(t_j - t_{j-1}) + k(m - t_{k-1}) \\ &= km - (t_0 + t_1 + \cdots + t_{k-1}) \\ &= km - w_k \end{aligned}$$

avec $w_k = t_0 + t_1 + \cdots + t_{k-1}$.

Exercice 1.9. Montrer que $w_k = \lfloor 2^{k+1}/3 \rfloor$, $w_{k-1} = \lfloor w_k/2 \rfloor$ et $t_{k-1} = \lceil w_k/2 \rceil$. 1.9

Théorème 1.9 (A. Hadian, PhD thesis 1969).

$$F(n) = \sum_{1 \leq k \leq n} \lceil \lg(\frac{3}{4}k) \rceil.$$

Démonstration. Nous suivons [7, page 185]. On montre alors que pour $w_k < n \leq w_{k+1}$ on a

$$F(n) - F(n-1) = k$$

en distinguant les cas n pair et n impair et en utilisant $w_{k-1} = \lfloor w_k/2 \rfloor$ et $t_{k-1} = \lceil w_k/2 \rceil$.

Supposons $n = 2u$.

$$\begin{aligned} F(n) - F(n-1) &= \{u + F(u) + G(u)\} - \left\{\lfloor u - \frac{1}{2} \rfloor + F(\lfloor u - \frac{1}{2} \rfloor) + G(\lceil u - \frac{1}{2} \rceil)\right\} \\ &= \{u + F(u) + G(u)\} - \{(u-1) + F(u-1) + G(u)\} \\ &= 1 + F(u) - F(u-1) \end{aligned}$$

De $w_k < u \leq w_{k+1}$ on déduit que $2w_k + 1 < 2u \leq 2w_{k+1}$; puis de $w_{n-1} = \lfloor w_n/2 \rfloor$ on déduit que $2w_{k+1} \leq w_{k+2}$ et $w_{k+1} \leq 2w_k + 1$. Ainsi

$$w_{k+1} < 2u \leq w_{k+2}.$$

D'où

$$F(n) - F(n-1) = 1 + k$$

avec la valeur de k annoncée (induction).

Supposons $n = 2u + 1$.

$$\begin{aligned} F(n) - F(n-1) &= u + F(u) + G(u+1) - u - F(u) - G(u) \\ &= G(u+1) - G(u) \end{aligned}$$

Soit k tel que $t_{k-1} \leq u < t_k$. Alors $t_{k-1} < u + 1 \leq t_k$. Par suite

$$\begin{aligned} G(u+1) - G(u) &= \{k(u+1) - w_k\} - \{ku - w_k\} \\ &= k \end{aligned}$$

Reste donc à montrer que $w_k < 2u + 1 \leq w_{k+1}$. De $t_{k-1} \leq u < t_k$ je déduis que

$$2t_{k-1} < 2u + 1 \leq 2t_k - 1$$

mais $w_k \leq 2\lceil w_k/2 \rceil = 2t_{k-1}$ d'une part et d'autre part

$$2t_k - 1 = 2\lceil w_{k+1}/2 \rceil - 1 = \begin{cases} 2(\alpha+1) - 1 = 2\alpha + 1 = w_{k+1} & \text{si } w_{k+1} = 2\alpha + 1 \text{ est impair} \\ 2\alpha - 1 < w_{k+1} & \text{si } w_{k+1} = 2\alpha \text{ est pair} \end{cases}$$

Ainsi

$$F(n) - F(n-1) = k$$

avec la valeur de k annonçée.

Pour finir il reste à montrer que si $w_k < n \leq w_{k+1}$ alors $k = \lceil \lg 3n/4 \rceil$. De

$$\lfloor 2^{k+1}/3 \rfloor < n \leq \lfloor 2^{k+2}/3 \rfloor$$

je déduis que

$$2^{k+1}/3 < n \leq 2^{k+2}/3$$

car $2^u/3$ n'est pas entier. Ainsi

$$2^{k-1} < \frac{3n}{4} < 2^k$$

d'où

$$k = \lceil \lg 3n/4 \rceil.$$

□

Les premières valeurs de $F(n)$ montrent

n	$\longmapsto$	$\lceil \lg n! \rceil$	$F(n)$	
1		0	0	*
2		1	1	*
3		3	3	*
4		5	5	*
5		7	7	*
6		10	10	*
7		13	13	*
8		16	16	*
9		19	19	*
10		22	22	*
11		26	26	*
12		29	30	**
13		33	34	**
14		37	38	**
15		41	42	
16		45	46	
17		49	50	
18		53	54	
19		57	58	
20		62	62	*
21		66	66	*
22		70	71	**

que la méthode de fusion insertion est optimale pour n variant de 1 à 11 ainsi que $n = 20$ et 21 car dans ce cas $F(n) = \lceil \lg n! \rceil$. Elle est également optimale pour $n = 12$ (prouvé par recherche exhaustive en 1965) ainsi que pour $n = 13, 14$ et 22 (prouvé en 2002) et $n = 15$ (prouvé en 2004 ?). La méthode n'est pas optimale pour $n = 47$ (prouvé en 1981) - on sait également que la méthode n'est pas optimale pour une infinité de valeur de n dont $n = 189$ (prouvé en 1979 dans [8]). Les données ci-dessus sont extraites du Knuth [7] datant de juin 2004 et d'une lecture (très très rapide) de [9] : état de l'art aujoud'hui ? (✍).

Structures de données primitives et notations algorithmiques. *Tableaux - signe d'affectation - boucle* **for** *- boucle* **while** *- etc.* Je renvois par exemple au Beauquier and co [2] ou encore au Cormen and co [3].

Voici du pseudo-code pour le tri par insertion linéaire.

Algorithm Tri_Insertion_Linéaire_En_Place (**var** A : **array**$[1..n]$ **of** typekey) ;
entrée : un tableau A de n éléments
sortie : le tableau trié des éléments de A

```
0.   j := 0                variable auxilliaire entière
1.   for i := 2 to n do
2.     j := i ;
3.     while (j ≠ 1) and (A[j] < A[j − 1]) do
4.       swap(A[j], A[j − 1]) ;
5.       j := j − 1 ;
6.     od
7.   od
```

Exercice 1.10. Expliquer le déroulement de la procédure OPERATEUR suivante :

```
    procedure OPERATEUR(var A : array[1..n] of integer) ;
entrée    : un tableau de n entiers compris entre 0 et n − 1
sortie    :
    var i, j, k : integer ;
        B : array[0..n − 1] of integer ;
        D : array[1..n] of integer ;
    begin
1.    for i := 0 to n − 1 do B[i] := 0 ;
2.    for i := 1 to n do B[A[i]] := B[A[i]] + 1 ;
3.    k := 0 ;
5.    for i := 0 to n − 1 do
      begin
6.      for j := 1 to B[i] do D[k + j] := i ;
7.      k := k + B[i] ;
      end ;
8.    for i := 1 to n do A[i] := D[i] ;
    end ; {OPERATEUR}
```

Quelle est sa complexité asymptotique en nombre d'affectations (:=) ?
1.10

Sélection de la médiane et autres éléments. La **médiane** d'une liste de n éléments est le $\lceil n/2 \rceil$-ième plus petit élément.

L'algorithme SELECT [M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*,7 :448–461,1973.] détermine le k-ième plus petit élément d'une liste L de n éléments comme suit : Si $n \leq 5$ procéder directement en triant la liste et en retournant le k-ième élément de la liste triée ; sinon

- (**1-**) Scinder la liste des n éléments en $\lfloor n/5 \rfloor$ sous-listes de 5 éléments et une sous-liste de 0,1,2,3, ou 4 éléments ;
- (**2-**) Calculer la médiane de chacune des $\lceil n/5 \rceil$ sous-listes ;
- (**3-**) Calculer en utilisant SELECT récursivement la médiane de la liste des médianes des $\lceil n/5 \rceil$ sous-listes, la *médiane des médianes* en abrégé ;
- (**4-**) Scinder la liste L en trois sous-listes : la liste L_1 des n_1 éléments strictement inférieurs à la médiane des médianes, la liste L_2 des n_2 éléments strictement supérieurs à la médiane des médianes et la liste réduite à la médiane des médianes ;
- (**5-**) Si $k = n_1 + 1$ alors retourner la médiane des médianes ; sinon appeler SELECT récursivement pour calculer le k-ième plus petit élément de la liste L_1 ou le $(k - n_1 - 1)$-ième plus petit élément de la liste L_2 selon que $k < n_1 + 1$ ou $k > n_1 + 1$.

Théorème 1.10. *Le nombre de comparaisons de l'algorithme* SELECT *est un* $O(n)$.

Démonstration. Soit $T(n)$ le nombre de comparaisons dans le pire cas de l'algorithme SELECT. Par définition de l'algorithme

$$T(n) \leq An + T(\lceil n/5 \rceil) + T(M)$$

avec $A = O(1)$ et M un majorant de la taille des listes L_1 et L_2.

Soit $b_1, b_2, \ldots, b_{\lceil n/5 \rceil}$ la liste croissante des médianes des $\lceil n/5 \rceil$ sous-listes de taille ≤ 5 de la première étape de l'algorithme.

La médiane des médianes est alors $b_{\lceil \lceil n/5 \rceil / 2 \rceil} = b_{\lceil n/10 \rceil}$ et pour chaque sous-liste de taille ≤ 5 il y a 3 éléments inférieurs ou égaux à cette médiane des médianes et 3 éléments supérieurs ou égaux à cette médiane des médianes sauf pour l'éventuelle sous-liste de taille 1,2,3 ou 4 pour laquelle il y a lieu de remplacer ce chiffre de 3 par 1.

Soit Δ le nombre d'éléments inférieurs ou égaux à la MDM. Clairement

$$\begin{aligned} \Delta &\geq 3\lceil n/10 \rceil - 2 \\ &\geq 3\lfloor n/10 \rfloor - 2 \end{aligned}$$

De même soit ∇ le nombre d'éléments supérieurs ou égaux à la MDM. On a non moins

clairement

$$\begin{aligned}\nabla &\geq 3\{\lceil n/5\rceil - \lceil n/10\rceil + 1\} - 2\\ &= 3\{\lceil n/5\rceil - \lceil n/10\rceil\} + 1\\ &= 3\lfloor\lceil n/5\rceil/2\rfloor + 1\\ &\geq 3\lfloor n/10\rfloor + 1\\ &\geq 3\lfloor n/10\rfloor - 2\end{aligned}$$

Ainsi en notant u le reste de la division de n par 10

$$\begin{aligned}M &\leq n - \min\{\Delta, \nabla\}\\ &\leq n - 3\lfloor n/10\rfloor + 2\\ &\leq 7n/10 + 3u/10 + 2\\ &\leq 7n/10 + 5\end{aligned}$$

Ainsi

$$T(n) \leq An + T(\lceil n/5\rceil) + T(7n/10 + 5)$$

Montrons par récurrence que $T(n) \leq Bn$ pour un certain B. C'est vrai pour $n \leq 60$. Pour $n \geq 61$

$$\begin{aligned}T(n) &\leq B(n/5 + 1) + B(7n/10 + 5) + An\\ &\leq B9n/10 + 6B + An\\ &\leq Bn\end{aligned}$$

pour $6B + An \leq Bn/10$ soit $An \leq B(n/10 - 6)$ soit $B \geq A/(1/10 - 6/n)$ il suffit donc de prendre $B \geq A/(1/10 - 6/61) = 610A$. □

Exercice 1.11. Montrer que le minimum d'une liste de n éléments d'un ensemble totalement ordonné est calculable en $n-1$ comparaisons. Est-ce la meilleure borne possible ? (Justifier votre réponse.) 1.11

Exercice 1.12. Montrer que le second plus petit élément d'une liste de $n \geq 2$ éléments d'un ensemble totalement ordonné est calculable en $n + \lceil \lg_2 n\rceil - 2$ comparaisons. 1.12

Exercice 1.13. Montrer que le maximum et le minimum d'une liste de n éléments d'un ensemble totalement ordonné est calculable en $\lceil 3n/2\rceil - 2$ comparaisons. Montrer que c'est la meilleure borne possible. 1.13

Exercice 1.14. Soit $\varphi : \mathfrak{S}_n \to \mathbb{N}^n$ l'application qui à la permutation σ de $[\![n]\!]$ associe le n-uplet d'entiers $(k_1, k_2, \ldots, k_n)$ où k_j est le cardinal de l'ensemble des $i \in [\![j]\!]$ tel que $\sigma(i) \leq \sigma(j)$. Par exemple $\varphi([27416835]) = (1, 2, 2, 1, 4, 6, 3, 5)$. Montrer que φ est injective. Quelle est son image ? 1.14

Exercice 1.15. Calculer l'espérance du nombre d'exécutions de la ligne 4 de l'algorithme décrit ci-dessous

Algorithm Rand-Min $(A[1..n])$;
entrée : un tableau A de n clés distinctes
sortie : la clé de valeur minimale

```
0.  randomly pick a permutation σ of {1, 2, ..., n}
1.  min := ∞ ;
2.  for i := 1 to n
3.    do if min > A[σ(i)]
4.         then min := A[σ(i)]
5.  return min
```

1.15

Tri partition [6]. [Le tri le plus rapide en mémoire centrale] Le **tri partition** d'une liste a de $n \geq 1$ éléments consiste [C. A. R. Hoare. Quicksort. *Comput. J*, 5 :13–28, 1962.] à choisir un élément τ de la liste que l'on nomme le **pivot**, à scinder la liste initiale en trois sous-listes : la liste a' des éléments inférieurs au pivot, la liste réduite au pivot et la liste a'' des éléments strictement supérieurs au pivot, à trier récursivement les listes a' et a'', et à retourner la concaténation des listes triées a', (τ) et a''.

La complexité en nombre de comparaisons du tri partition dépend de la méthode de choix du pivot. Par exemple si l'on choisit pour pivot la médiane de la liste [avec l'algorithme de Blum and co] alors le nombre maximal $C(n)$ de comparaisons du tri partition d'une liste de n éléments vérifie la récurrence

$$C(n) = O(n) + 2\,C(\lfloor n/2 \rfloor)$$

où le terme $O(n)$ couvre le coût du calcul de la médiane et le coût de la partition de la liste autour du pivot ; récurrence qui se résout en $O(n \lg n)$, ✍. Si l'on choisit pour pivot l'élément de rang 1 alors

$$C(n) = n - 1 + \max_{u+v=n-1} \{C(u) + C(v)\};$$

récurrence qui se résout en $O(n^2)$, ✍. Enfin si l'on choisit pour pivot un élément aléatoire pour la loi uniforme alors l'espérance du nombre de comparaisons du tri partition d'une liste de n éléments est $\sim 2n \ln n$ ($\ln$ est comme d'usage le logarithme népérien ; $\ln n = \ln 2 \lg n = 0.693\ldots \lg n$) ; dans ce dernier nous parlerons du **tri partition randomisé**.

Théorème 1.11. *L'espérance du nombre de comparaisons du tri partition randomisé d'une liste de n éléments est $\sim 2n \ln n$.*

Démonstration. Soit $K_1, K_2, \ldots, K_n$ la suite triée des n clés. Pour $1 \leq i < j \leq n$ notons X_{ij} la variable aléatoire égale à 1 si K_i et K_j sont comparés ; 0 sinon. Nous disons que $X_{ij} = 1$ avec probabilité $2/(j - i + 1)$. En effet

- (**1-**) K_i et K_j sont comparés si et seulement si la clé K_i est choisie comme pivot avant les clés $K_{i+1}, \ldots, K_{j-1}, K_j$ ou la clé K_j est choisie comme pivot avant les clés $K_i, K_{i+1}, \ldots, K_{j-1}$;
- (**2-**) il y a équiprobabilité des événements 'la clé K_l est le premier pivot choisi parmi les K_k, $i \leq k \leq j$'.

Il suit

$$\begin{aligned}
E(\#\text{comp.}) &= E(\sum_{1\le i<j\le n} X_{ij}) \\
&= \sum_{1\le i<j\le n} E(X_{ij}) && \text{(car l'espérance est additive)} \\
&= \sum_{1\le i<j\le n} \frac{2}{j-i+1} \\
&= \sum_{j=2}^{n}\sum_{i=1}^{j-1} \frac{2}{j-i+1} \\
&= \sum_{j=2}^{n}\sum_{k=2}^{j} \frac{2}{k} \\
&= \sum_{j=2}^{n} 2(H_j - 1) \\
&\sim 2n\ln n.
\end{aligned}$$

□

Références

[1] Andersson, Hagerup, Nilsson, and Raman. Sorting in linear time ? *J. Comput. Syst. Sci.*, 57 :74–93, 1998.

[2] D. Beauquier, J. Berstel, and P. Chrétienne. *Eléments d'algorithmique*. Masson, 1992.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[4] L. R. Ford and S. M. Johnson. A tournament problem. *The American Mathematical Monthly*, 66(5) :387–389, 1959.

[5] G. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures : In Pascal and C*. Addison-Wesley, 1991. Second Edition.

[6] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1) :10–16, 1962.

[7] D. E. Knuth. *The Art of Computer Programming : Volume 3 : Sorting and Searching*. Addison-Wesley, 2 edition, 1998.

[8] G. K. Manacher. The Ford-Johnson sorting algorithm is not optimal. *J. ACM*, 26(3) :441–456, July 1979.

[9] M. Peczarski. Towards Optimal Sorting of 16 Elements. *ArXiv e-prints*, Aug. 2011.

[10] H. Steinhaus. *Mathematical Snapshots*. Oxford University Press, 1950.

[11] H. Steinhaus. *Mathématiques en instantanés*. Flammarion, 1964. Chap. 1 Triangles, carrés et jeux - Chap. 2 Rectangle, nombre et accords - Chap. 3 Pesées, mesures et partages équitables - Chap. 4 Pavages, mélanges de liquides, mesures d'aires et de longueurs - Chap. 5 Plus courts chemins, emplacement d'écoles et poursuites de bateaux - Chap. 6 Droites, cercles, symétrie et illusions d'optique - Chap. 7 Cubes, araignées, rayons de miel et briques - Chap. 8 Solides de Platon, cristaux, têtes d'abeilles et savons - Chap. 9 Bulles de savon, terre de lune, cartes de géographie et dates - Chap. 10 Ecureuils, vis, bougies, accords musicaux et ombres portées - Chap. 11 Tissages rectilignes, chainette, charette-jouet et surface minimale - Chap. 12 Encore les solides de Platon, traversées de ponts, confection de nœuds, coloration de cartes et démelage de cheveux - Chap. 13 Grenouilles, étudiants de première année et héliotropes.

2 Gestion de partition : la structure de données union-find

Partitions - logarithme itéré - fonction d'Ackermann - arborescences - partitions pointées - opérations sur les partitions pointées (ajout d'un singleton, addition ensembliste de deux classes, identification de l'élément distingué de la classe d'un élément donné) - représentation d'une partition pointée par une somme d' arborescences - addition ensembliste pondérée et compression des chemins - analyse de la complexité d'une suite d'opérations.

Applications of Path Compression on Balanced Trees
by R. E. Tarjan, in *Journal of ACM* Vol 26, No 4, October 1979, pp 690-715.

Abstract. Several fast algorithms are presented for computing functions defined on paths in trees under various assumptions. The algorithms are based on tree manipulation methods first used to efficiently represent equivalence relations. The algorithms have $O((m+n)\alpha(m+n,n))$ running times, where m and n are measures of the problem size and α is a functional reverse of Ackermann's function.

By using one or more of these algorithms in combination with other techniques, it is possible to solve the followimg graph problems in $O(m\alpha(m,n))$ time, where m is the number of edges and n is the number of vertices of the problem graph

A. Verifying a minimum spanning tree in an undirected graph (Best previously known time bound $O(m \log\log n)$.)

B. Finding dominators in a flow graph (Best previously known time bound $O(n \log n + m)$.)

C. Solving a path problem on a reducible flow graph. (Best previously known time bound. $O(m \log n)$)

Application A is discussed

KEYWORDS AND PHRASES : balanced tree, dominators, equivalence relation, global flow analysis, graph algorithm, minimum spanning tree, path compression, path problem, tree

CR CATEGORIES : 4.12, 4.34, 5.25, 5.32

(**1-**) Une partition d'un ensemble E est un ensemble de parties non vides de E d'intersection deux-à-deux vide et d'union E. Voici la liste des partitions de $\{1, 2, \ldots, n\}$ pour $n = 0, 1, 2, 3$ et 4.

n		
0	$\varnothing$	
1	$\{\{1\}\}$	
2	$\{\{1\},\{2^\circ\}\}$	✓
	$\{\{1,2^\circ\}\}$	
3	$\{\{1\},\{2\},\{3^\circ\}\}$	✓
	$\{\{1\},\{2,3^\circ\}\}$	
	$\{\{1,3^\circ\},\{2\}\}$	
	$\{\{1,2\},\{3^\circ\}\}$	✓
	$\{\{1,2,3^\circ\}\}$	
4	$\{\{1\},\{2\},\{3\},\{4^\circ\}\}$	✓
	$\{\{1\},\{2\},\{3,4^\circ\}\}$	
	$\{\{1\},\{2,4^\circ\},\{3\}\}$	
	$\{\{1,4^\circ\},\{2\},\{3\}\}$	
	$\{\{1\},\{2,3\},\{4^\circ\}\}$	✓
	$\{\{1\},\{2,3,4^\circ\}\}$	
	$\{\{1,4^\circ\},\{2,3\}\}$	
	$\{\{1,3\},\{2\},\{4^\circ\}\}$	✓
	$\{\{1,3\},\{2,4^\circ\}\}$	
	$\{\{1,3,4^\circ\},\{2\}\}$	
	$\{\{1,2\},\{3\},\{4^\circ\}\}$	✓
	$\{\{1,2\},\{3,4^\circ\}\}$	
	$\{\{1,2,4^\circ\},\{3\}\}$	
	$\{\{1,2,3\},\{4^\circ\}\}$	✓
	$\{\{1,2,3,4^\circ\}\}$	

(**2-**) Le **logarithme itéré**, noté $\lg^* n$, est défini par $\lg^{(0)} n = n$, $\lg^{(i)} = \lg^{(i-1)} \lg n$ pour $i \geq 1$, et $\lg^* n = \min\{i \mid \lg^{(i)}(n) \leq 1\}$. (lg est le logarithme à base 2.)

(**3-**) La fonction d'Ackermann et son inverse selon Tarjan [6, Chap. 2] :
For $i, j \geq 1$ we define the **Ackerman's function** $A(i, j)$ by

$$\begin{aligned} A(1,j) &= 2^j \qquad \text{for } j \geq 1, \\ A(i,1) &= A(i-1,2) \qquad \text{for } i \geq 2, \\ A(i,j) &= A(i-1, A(i,j-1)) \qquad \text{for } i,j \geq 2. \end{aligned}$$

We define the inverse function $\alpha(m, n)$ for $m \geq n \geq 1$ by

$$\alpha(m,n) = \min\{\, i \geq 1 \mid A(i, \lfloor m/n \rfloor) \geq \lg n \,\}.$$

The most important property of $A(i,j)$ is its explosive growth. In the usual definition, $A(1,j) = j+1$ and the explosion does not occur quite so soon. However, this change only add a constant to the inverse function α, which grows very slowly. With our definition, $A(3,1) = 16$; thus $\alpha(m,n) \le 3$ for $n < 2^{16} = 65536$. $A(4,1) = A(2,16)$, which is very large. Thus for practical purposes $\alpha(m,n)$ is a constant no larger than four. For fixed n, $\alpha(n,m)$ decreases as m/n increases. In particular, let $a(i,n) = \min\{j \ge 1 \mid A(i,j) > \lg n\}$. Then $\lfloor m/n \rfloor \ge a(i,n)$ implies $\alpha(m,n) \le i$. For instance, $\lfloor m/n \rfloor \ge 1 + \lg\lg n$ implies $\alpha(m,n) \le 1$, and $\lfloor m/n \rfloor \ge \lg^* n$ implies $\alpha(m,n) \le 2$.

(4-) La fonction d'Ackermann et son inverse selon Kozen [4, Lecture 10] (légèrement modifié) :

Ackermann's function is a famous function that is known for its extremely rapid growth. Its inverse $\alpha(n)$ grows extremely slowly. The texts [A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1975], [R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.] give inequivalent definitions of Ackermann's function, and in fact there does not seem to be any general agreement on the definition of "the" Ackermann's function ; but all these functions grow at roughly the same rate. Here is another definition that grows at roughly the same rate :

$$\begin{aligned} A_0(x) &= x+1 \\ A_{k+1}(x) &= \underbrace{A_k \circ A_k \cdots \circ A_k}_{x}(x). \end{aligned}$$

In other words, to compute $A_{k+1}(x)$, start with x and apply A_k x times. It is no hard to show by induction that A_k is monotone in the sense that

$$x \le y \implies A_k(x) \le A_k(y)$$

and that for all x, $x \le A_k(x)$.

As k grows, these functions get extremely huge extremely fast. For $x = 0$ or 1, the numbers $A_k(x)$ are small. For $x \ge 2$, For $x \ge 2$,

$$\begin{aligned} A_0(x) &= x+1 \\ A_1(x) &= 2x \\ A_2(x) &= x2^x && \ge 2^x \\ A_3(x) &\ge \underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{x} && = 2 \uparrow x \\ A_4(x) &\ge \underbrace{2 \uparrow (2 \uparrow \cdots \uparrow (2 \uparrow 2)\cdots)}_{x} && = 2 \uparrow\uparrow x \end{aligned}$$

For $x = 2$, the growth of $A_k(2)$ as a function of k is beyond comprehension. Already for $k = 4$, the value of $A_4(2)$ is larger than the number of atomic particles in the known universe or the number of nanoseconds since the Big Bang.

$$\begin{aligned} A_0(2) &= 3 \\ A_1(2) &= 4 \\ A_2(2) &= 8 \\ A_3(2) &= 2^{11} = 2048 \\ A_4(2) &\ge 2 \uparrow 2048 \end{aligned}$$

We define a unary function that majorizes all the A_k (i.e., grows asymptotically faster than all of them) :

$$A(k) = A_k(2)$$

and call it Ackermann's function. This function grows asymptotically faster than any primitive recursive function, since it can be shown that all primitive recursive functions are bounded almost everywhere by one the functions A_k. The primitive recursive functions are those computed by a simple PASCAL-like programming language over the natural numbers with **for** loops but no **while** loops. The level k corresponds roughly to the depth of nesting of the **for** loops [A. R. Meyer and D. M. Ritchie. The complexity of loop programs. In *Proceedings of the 1967 22nd National Conference*, ACM'67, pages 465–469, New York, NY, USA, 1967. ACM.].
The inverse of Ackermann's function is

$$\alpha(n) = \text{the least } k \text{ such that } A(k) \geq n$$

which for practical purpose is 4. We will ...

Exercice 2.1. La liste des listes des partitions des ensembles $[\![n]\!]$ pour $n = 0, 1, 2, 3$ et 4 donnée page 41 suggère un algorithme pour établir une liste des partitions de $[\![n]\!]$ à partir d'une liste des partitions de $[\![n-1]\!]$. Décrire cet algorithme, démontrer sa correction et analyser sa complexité. Comment modifier cet algorithme pour obtenir une liste des partitions de $[\![n]\!]$ ayant la propriété (dite propriété de Gray) suivante : deux partitions consécutives de la liste sont obtenues l'une à partir de l'autre en déplaçant un élément d'une classe à une autre classe ? Justifier votre réponse. Indication page 55 2.1

Exercice 2.2. Justifier les assertions de Tarjan concernant la fonction d'Ackermann. Idem pour celles de Kozen (en particulier calculer les $A_k(0)$ et $A_k(1)$ et montrer par récurrence sur k que les fonctions A_k sont croissantes et que pour tout x, $x \leq A_k(x)$). 2.2

Arborescences. Une **arborescence** d'ordre $n \geq 1$ est la donnée d'un ensemble X de n éléments, appelés **nœuds**, et d'une application $\mathcal{P} : X \to X$, appelée **fonction parent**, telle que

(**1-**) $\mathcal{P}$ admet un unique point fixe r, appelé la **racine** de l'arborescence ;
(**2-**) pour tout $x \in X$, la suite $x, \mathcal{P}(x), \mathcal{P}^2(x), \ldots$ est constante à partir d'un certain rang de valeur (nécessairement) la racine r de l'arborescence. Le minimum des $k \geq 0$ tels que $\mathcal{P}^k(x) = r$ est alors appelé la **hauteur** de x.

La **hauteur** d'une arborescence est le maximum des hauteurs de ses nœuds.

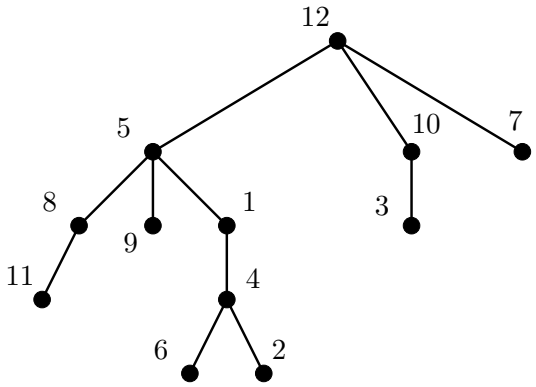
La **sous-arborescence** de racine le nœud x d'une arborescence $\mathcal{P}$ est l'arborescence $\mathcal{P}' : X' \to X'$ définie sur $X' = \{x\} \cup \mathcal{P}^{-1}(x) \cup \mathcal{P}^{-1}(\mathcal{P}^{-1}(x)) \cup \cdots$ par

$$\mathcal{P}'(y) = \begin{cases} \mathcal{P}(y) & \text{si } y \neq x\,; \\ x & \text{sinon.} \end{cases}$$

Par exemple l'application $\mathcal{P}$ de $[\![12]\!]$ dans $[\![12]\!]$ définie par

x	1	2	3	4	5	6	7	8	9	10	11	12
$\mathcal{P}(x)$	5	4	10	1	12	4	12	5	5	12	8	12

est une arborescence que l'on pourra représenter par de diagramme suivant



Exercice 2.3. Dessiner toutes les arborescences à isomorphisme près sur un ensemble à $1, 2, 3$ et 4 éléments. Quelle est la hauteur de chacune de ces arborescences ? 2.3

Exercice 2.4. Une arborescence ordonnée est une arborescence munie pour chaque nœud de l'arborescence d'un ordre total sur l'ensemble des antécédents du nœud par la fonction parent. Quel est le nombre de façons d'ordonner une arborescence $\mathcal{P} : X \to X$ en fonction des cardinaux des $\mathcal{P}^{-1}(x)$, $x \in X$? Dessiner toutes les arborescences ordonnées à isomorphisme près sur un ensemble à $1, 2, 3$ et 4 éléments. 2.4

Gestion de partitions. Une partition **pointée** est une partition d'un ensemble fini dont un élément par classe est distingué. Une partition pointée X est représentée dans la suite par une fonction $\mathcal{P} : \bigcup X \to \bigcup X$ telle que pour tout $x \in \bigcup X$ la suite

$$x, \mathcal{P}(x), \mathcal{P}^2(x), \dots$$

est constante à partir d'un certain rang de valeur l'élément distingué de la classe de x. En particulier les éléments distingués sont les points fixes de l'application $\mathcal{P}$.

Notons que pour tout élément distingué x, la restriction de l'application $\mathcal{P}$ à la classe de x est une arborescence de racine x sur la classe de x.

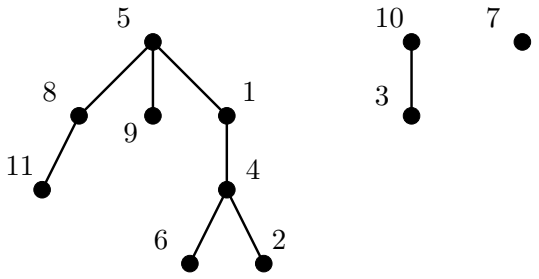
Par exemple l'application $\mathcal{P}$ de $[\![11]\!]$ dans $[\![11]\!]$ définie par

x	1	2	3	4	5	6	7	8	9	10	11
$\mathcal{P}(x)$	5	4	10	1	5	4	7	5	5	10	8

est une représentation de la partition pointée

$$X = \{\{2, 5^\circ, 8, 11, 1, 4, 6, 9\}, \{10^\circ, 3\}, \{7^\circ\}\}$$

Les arborescences associées sont



Trois opérations sont définies sur une partition pointée : l'ajout d'un singleton à la partition, l'addition ensembliste de deux classes données par leurs éléments distingués et l'identification de l'élément distingué de la classe d'un élément donné.

L'ajout d'un singleton $\{x\}$ se réduit à étendre l'application $\mathcal{P}$ à x en posant $\mathcal{P}(x) = x$. Son coût est constant et son implémentation à minima est la suivante

procedure makeset (**element** x) ;
1. $\mathcal{P}(x) := x$;
end makeset ;

L'addition ensembliste des classes des éléments distingués x et y se réduit à redéfinir l'image de x par $\mathcal{P}$ en posant $\mathcal{P}(x) = y$ (ou à redéfinir l'image de y par $\mathcal{P}$ en posant $\mathcal{P}(y) = x$). Son coût est constant et son implémentation à minima est

element function link (**element** x, y) ;
1. $\mathcal{P}(x) := y$;
2. **return** y
end link ;

L'identification de l'élément distingué de la classe de l'élément x se réduit à déterminer le premier élément de la suite

$$x, \mathcal{P}(x), \mathcal{P}^2(x), \ldots,$$

égal à son successeur. Son coût en nombre d'appels à la fonction $\mathcal{P}$ est le rang de cet élément dans la suite, i.e., le plus petit $k \geq 1$ tel que $\mathcal{P}^k(x) = \mathcal{P}^{k-1}(x)$ et son implémentation à minima est (sous forme récursive) la suivante :

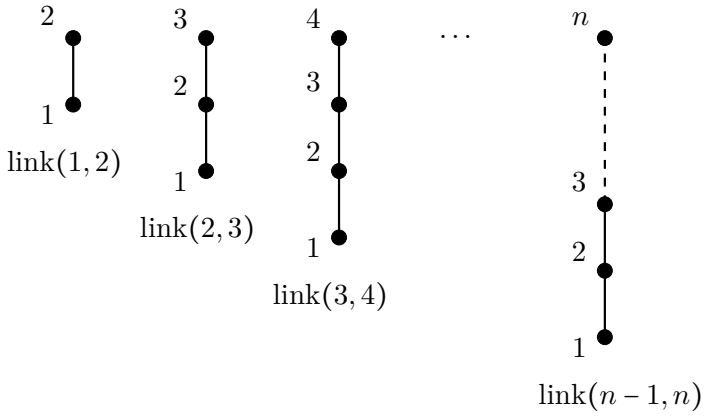
element function find (**element** x) ;
1. **if** $x \neq \mathcal{P}(x)$ **then return** find($\mathcal{P}(x)$) **else return** x **fi** ;
end find ;

Exercice 2.5. Dérécursifier la fonction find. 2.5

Avec les implémentations précédentes de ces trois opérations le coût d'une suite d'opérations peut être quadratique en la longueur de la suite car les arborescences construites peuvent être de hauteur linéaire. Par exemple la suite des $3n$ opérations

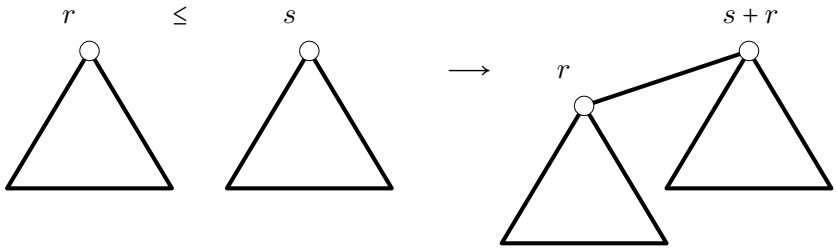
$$\begin{array}{c}\text{makeset}(1)\\ \text{makeset}(2)\\ \vdots\\ \text{makeset}(n)\\ \text{link}(1,2)\\ \text{link}(2,3)\\ \vdots\\ \text{link}(n-1,n)\\ \text{find}(1)\\ \text{find}(2)\\ \vdots\\ \text{find}(n)\end{array}$$

a un coût de l'ordre de n^2 car les arborescences construites sont des chemins



Peut-on faire mieux ? Oui. Voici comment.

Une première heuristique, **l'addition pondérée**, qui consiste à choisir pour élément distingué de l'addition ensembliste de deux classes l'élément distingué de la classe de taille maximale, conduit à des arborescences de hauteur logarithmique et donc à un



coût par opération logarithmique en le nombre d'ajouts. Son implémentation nécessite d'introduire une variable $\mathcal{T}(x)$ pour l'ordre de l'arborescence de racine x. Cette variable est initialisée lors de la création de singletons. La voici : [Pour les besoins de l'analyse nous introduisons également dans le pseud-code une variable $\mathcal{H}(x)$ pour la hauteur de l'arborescence de racine x afin de suivre son évolutions au cours des opérations]

procedure makeset (**element** x) ;
1. $\mathcal{P}(x) := x$; $\mathcal{T}(x) := 1$; $\mathcal{H}(x) := 0$;
end makeset ;

element function link (**element** x, y) ;
1. **if** $\mathcal{T}(x) > \mathcal{T}(y)$ **then return** link(y, x) ;
2. $\mathcal{T}(y) := \mathcal{T}(y) + \mathcal{T}(x)$; $\mathcal{H}(y) := \max\{\mathcal{H}(y), 1 + \mathcal{H}(x)\}$;
4. $\mathcal{P}(x) := y$;
5. **return** y
end link ;

element function find (**element** x) ;
1. **if** $x \neq \mathcal{P}(x)$ **then return** find$(\mathcal{P}(x))$ **else return** x **fi** ;
end find ;

Théorème 2.1. *L'addition pondérée conduit à des arborescences de hauteur logarithmique en le nombre d'ajouts.* □

Soit $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ une suite mixte de m opérations (adition pondérée, identification) appliquées à une partition pointée X de n singletons.

Plaçons nous dans l'espace-temps discret $E \times T$ où $E = \bigcup X$ est l'ensemble des n éléments des singletons de la partition initiale X et où $T = \{0, 1, \ldots, m\}$. Ainsi nous parlerons de l'arborescence de racine le nœud (x, t) de l'espace temps pour l'arborescence de racine x après exécution des t premières opérations de la suite σ.

Pour $u = (x, t)$ un nœud de l'espace-temps on pose $u' = (x, t + 1)$.

Les fonctions parent, taille et hauteur sont maintenant définies sur l'espace-temps.

Lemme 2.2. *Pour tout nœud u de l'espace-temps*

- **(1-)** *si $u \neq \mathcal{P}(u)$ alors $\mathcal{H}(u) < \mathcal{H}(\mathcal{P}(u))$;*
- **(2-)** *si $u \neq \mathcal{P}(u)$ alors $u' \neq \mathcal{P}(u')$ & $\mathcal{P}(u') = \mathcal{P}(u)'$;*
- **(3-)** *$\mathcal{T}(u) \leq \mathcal{T}(u')$ avec inégalité stricte si et seulement si σ_{t+1} est un link avec pour argument la paire de nœuds (v, u) avec $\mathcal{T}(v) \leq \mathcal{T}(u)$ (ou la paire de nœuds (u, v) avec $\mathcal{T}(v) < \mathcal{T}(u)$) ; et dans ce cas $\mathcal{T}(u') = \mathcal{T}(u) + \mathcal{T}(v)$;*
- **(4-)** *$\mathcal{H}(u) \leq \mathcal{H}(u')$ avec inégalité stricte si et seulement si σ_{t+1} est un link avec pour argument la paire de nœuds (v, u) avec $\mathcal{T}(v) \leq \mathcal{T}(u)$ et $\mathcal{H}(u) \leq \mathcal{H}(v)$ (ou la paire de nœuds (u, v) avec $\mathcal{T}(v) < \mathcal{T}(u)$ et $\mathcal{H}(u) \leq \mathcal{H}(v)$) ; et dans ce cas $\mathcal{H}(u') = 1 + \mathcal{H}(v)$;*
- **(5-)** *$\mathcal{T}(u) \geq 2^{\mathcal{H}(u)}$;*

Démonstration. (1-4) ✍
(5) Par récurrence sur le temps. A l'initialisation : pour tout nœud $u = (x, 0)$ on a $\mathcal{T}(u) = 1$ et $\mathcal{H}(u) = 0$. Donc

$$\mathcal{T}(u) = 1 \geq 1 = 2^0 = 2^{\mathcal{H}(u)}.$$

Supposons maintenant qu'à l'instant t, pour tout nœud $u = (x, t)$, on a $\mathcal{T}(u) \geq 2^{\mathcal{H}(u)}$ et montrons qu'il en est de même pour u'. Si $\mathcal{H}(u') = \mathcal{H}(u)$ alors

$$\mathcal{T}(u') \geq \mathcal{T}(u) \geq 2^{\mathcal{H}(u)} = 2^{\mathcal{H}(u')}.$$

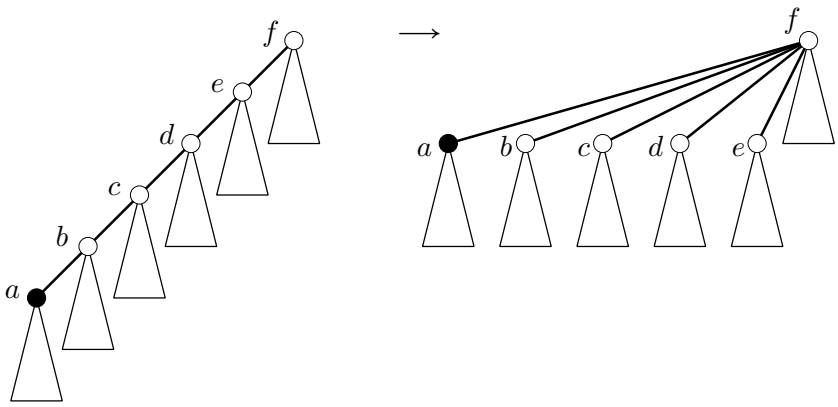
Si $\mathcal{H}(u') > \mathcal{H}(u)$ alors l'opération est un link(v, u) avec $\mathcal{T}(v) \leq \mathcal{T}(u)$ et $\mathcal{H}(v) \geq \mathcal{H}(u)$. Par suite

$$\begin{aligned}
\mathcal{T}(u') &= \mathcal{T}(v) + \mathcal{T}(u) \\
&\geq 2\mathcal{T}(v) \\
&\geq 2 2^{\mathcal{H}(v)} && \text{(par hypothèse de récurrence)} \\
&= 2^{1+\mathcal{H}(v)} \\
&= 2^{\mathcal{H}(u')}
\end{aligned}$$

□

Preuve du Theorem 2.1. évident □

Une deuxième heuristique, l'identification avec **compression des chemins**, qui consiste à faire suivre l'identification de l'élément distingué de la classe d'un élément donné de la compression, comme indiqué dans la figure ci-après, du chemin joignant cet élément



à l'élément distingué de sa classe, conduit à un coût quasi-linéaire en le nombre d'opérations. Voici son implémentation sous forme récursive :

element function find (**element** x) ;
1. **if** $x \neq \mathcal{P}(x)$ **then** $\mathcal{P}(x) := \text{find}(\mathcal{P}(x))$ **fi** ;
2. **return** $\mathcal{P}(x)$

end find ;

Le lecteur prendra garde qu'en raison de la compression des chemins la variable $\mathcal{T}(x)$ ne représente plus nécessairement la taille de l'arborescence de racine x ; cependant c'est bien le cas lorsque x est un élément distingué et c'est ce qui importe pour la correction de la procédure d'addition ensembliste de deux classes. De même la variable $\mathcal{H}(x)$ ne représente plus nécessairement la hauteur de l'arborescence de racine x ; cependant $\mathcal{H}(x)$ est une borne supérieure de la hauteur de l'arborescence de racine x.

Exercice 2.6. Dessiner les arborescences définies par la suite d'opérations

makeset(1)	makeset(2)	makeset(3)	makeset(4)	makeset(5)	makeset(6)
link(1, 2)	link(3, 4)	link(5, 6)	link(2, 4)	link(4, 6)	find(1)

et étiquetter chaque nœud ν par l'entier $\mathcal{T}(\nu)$. 2.6

Exercice 2.7. Montrer que la fonction find fait bien ce qu'on attend d'elle. Dérécursifier la fonction. 2.7

Théorème 2.3 (**Tarjan [5]**). *Le coût d'une suite mixte de $m+n$ opérations dont n ajouts appliquée à une partition pointée initialement vide est un $O(m\alpha(n))$.* □

Nous suivons, en le remaniant, l'exposé du Kozen [4].

Nous reprenons les notations précédentes.

Pour u nœud de l'espace temps posons $u^\infty = (x, m)$ et

$$\mathrm{rang}(u) = \mathcal{H}(u^\infty). \tag{2}$$

Lemme 2.4. *Pour tout nœud u de l'espace-temps*

- (**1-**) *si $u \neq \mathcal{P}(u)$ alors $\mathcal{H}(u) < \mathcal{H}(\mathcal{P}(u))$;*
- (**2-**) *si $u \neq \mathcal{P}(u)$ alors $u' \neq \mathcal{P}(u')$ & ~~$\mathcal{P}(u') = \mathcal{P}(u)'$~~ ;*
- (**3-**) *$\mathcal{T}(u) \leq \mathcal{T}(u')$ avec inégalité stricte si et seulement si σ_{t+1} est un link avec pour argument la paire de nœuds (v, u) avec $\mathcal{T}(v) \leq \mathcal{T}(u)$ (ou la paire de nœuds (u, v) avec $\mathcal{T}(v) < \mathcal{T}(u)$) ; et dans ce cas $\mathcal{T}(u') = \mathcal{T}(u) + \mathcal{T}(v)$;*
- (**4-**) *$\mathcal{H}(u) \leq \mathcal{H}(u')$ avec inégalité stricte si et seulement si σ_{t+1} est un link avec pour argument la paire de nœuds (v, u) avec $\mathcal{T}(v) \leq \mathcal{T}(u)$ et $\mathcal{H}(u) \leq \mathcal{H}(v)$ (ou la paire de nœuds (u, v) avec $\mathcal{T}(v) < \mathcal{T}(u)$ et $\mathcal{H}(u) \leq \mathcal{H}(v)$) ; et dans ce cas $\mathcal{H}(u') = 1 + \mathcal{H}(v)$;*
- (**5-**) *$\mathcal{T}(u) \geq 2^{\mathcal{H}(u)}$;*
- (**6-**) *si $u \neq \mathcal{P}(u)$ alors $\mathrm{rang}(u) < \mathrm{rang}(\mathcal{P}(u))$;*
- (**7-**) *$\mathrm{rang}(u) \leq \lfloor \lg n \rfloor$;*
- (**8-**) *$|\{u = (x, m) \mid \mathrm{rang}(u) = r\}| \leq n/2^r$.*

Démonstration. (1-5) seul le point (1) nécessite d'être adapater à la situation présente.

(6) ce n'est pas équivalent au (1) : il faut montrer que

$$\mathcal{H}(u^\infty) < \mathcal{H}(\mathcal{P}(u)^\infty)$$

(7)

$$\begin{aligned} n &\geq \mathcal{T}(u^\infty) \\ &\geq 2^{\mathcal{H}(u^\infty)} && \text{(assertion (5))} \\ &\geq 2^{\mathrm{rang}(u)} && \text{(définition du rang)} \end{aligned}$$

ainsi

$$\lfloor \lg n \rfloor \geq \mathrm{rang}(u).$$

(8) A temps fixé si u et v ont même rang alors les arborescences de racine u et v sont disjointes. Par suite

$$\begin{aligned} n &\geq \sum_{x:\,\mathrm{rang}(x,m)=r} \mathcal{T}(x,m) \\ &\geq \sum_{x:\,\mathrm{rang}(x,m)=r} 2^r \qquad \text{(assertion (5))} \\ &= |\{(x,m) \mid \mathrm{rang}(x,m) = r\}|\, 2^r \end{aligned}$$

□

Lemme 2.5. *Le coût des opérations d'addition ensembliste est constant par addition.*

Démonstration. ✍ □

Un nœud $u = (x, t)$ de l'espace-temps est dit **rouge** si

(**1-**) l'opération σ_t est une identification portant, disons, sur le nœud v ;
(**2-**) $u \in \text{orbit}(v) = \{v, \mathcal{P}(v), \mathcal{P}^2(v), \ldots\}$.

Ainsi le coût des opérations d'identification est le nombre de nœuds rouges. Pour borner le nombre de nœuds rouges l'idée clé est d'introduire la paire d'entiers

$$\delta(u) = (\eta(u), \vartheta(u))$$

comme étant la paire d'entiers (k, l), $l \neq 0$, maximale pour l'ordre lexicographique, telle que

$$\text{rg}(\mathcal{P}(u)) \geq A_k^l(\text{rg}(u)), \tag{3}$$

où $\text{rg}(u) = \text{rang}(u) + 2$.

Lemme 2.6. *Si $u \neq \mathcal{P}(u)$ alors*

(**1-**) *$\delta(u)$ et $\delta(u')$ sont bien définis ;*
(**2-**) *$\delta(u) \leq \delta(u')$;*
(**3-**) *$\vartheta(u) < \text{rg}(u)$;*
(**4-**) *$\eta(u) < \alpha(n)$ dès que $n \geq 5$.*

Démonstration. (1) Si $u \neq \mathcal{P}(u)$ alors, par application du Lemme 2.4 assertion (1), $\text{rg}(\mathcal{P}(u)) > \text{rg}(u)$. Par suite $\text{rg}(\mathcal{P}(u)) \geq \text{rg}(u) + 1 = A_0(\text{rg}(u))$; ce qui prouve que $\delta(u)$ est bien défini. Il en est alors de même de $\delta(u')$ puisque si $u \neq \mathcal{P}(u)$ alors $u' \neq \mathcal{P}(u')$.

(2) car $\text{rg}(u') = \text{rg}(u)$ et $\text{rg}(\mathcal{P}(u')) \geq \text{rg}(\mathcal{P}(u))$.

(3) car $A_k^{x+l}(x) = A_k^l(A_{k+1}(x)) \geq A_{k+1}(x)$.

(4) Pour $n \geq 5$ la valeur maximale de $\eta(u)$ est au plus $\alpha(n) - 1$, car si $\eta(u) = k$,

$$\begin{aligned}
n &> \lfloor \lg n \rfloor + 2 && \text{(car } n \geq 5\text{)} \\
&\geq \text{rg}(\mathcal{P}(u)) && \text{(assertion (3) Lemma 2.4)} \\
&\geq A_k(\text{rg}(u)) && \text{(par définition de } k\text{)} \\
&\geq A_k(2) = A(k)
\end{aligned}$$

ainsi $\alpha(n) > k$. □

Un nœud de l'espace-temps u est dit **non terminal** si il existe $v \in \text{orbit}(u)$, $v \neq u$, tel que $\eta(u) = \eta(v)$; **terminal** sinon.

Lemme 2.7. *Le nombre de nœuds rouges terminaux est un $O(m\alpha(n))$.*

Démonstration. A t fixé et à valeur de η fixé, par définition d'un nœud terminal, il existe au plus un nœud rouge terminal. Le résultat suit par application du lemme sur la majoration par $\alpha(n)$ du maximum de η. □

Lemme 2.8. *Soit u un nœud rouge non terminal. Alors $\delta(u) < \delta(u')$.*

Démonstration. Soit $v \neq u$ dans l'orbite de u tel que $\eta(v) = \eta(u)$. Posons $\delta(u) = (k, l)$. Ainsi

$$\begin{aligned} \mathrm{rg}(\mathcal{P}(u)) &\geq A_k^l(\mathrm{rg}(u)) \\ \mathrm{rg}(\mathcal{P}(v)) &\geq A_k(\mathrm{rg}(v)). \end{aligned}$$

Soit w l'élément distingué de la classe de u. Alors

$$\begin{aligned} \mathrm{rg}(w) &\geq \mathrm{rg}(\mathcal{P}(v)) \\ &\geq A_k(\mathrm{rg}(v)) \\ &\geq A_k(\mathrm{rg}(\mathcal{P}(u))) \\ &\geq A_k(A_k^l(\mathrm{rg}(u))) \\ &\geq A_k^{l+1}(\mathrm{rg}(u)), \end{aligned}$$

et puisque w' est le parent de u', on a

$$\mathrm{rg}(\mathcal{P}(u')) \geq A_k^{l+1}(\mathrm{rg}(u')).$$

Ceci prouve le lemme car soit $l < \mathrm{rg}(u') - 1$ et, dans ce cas, $\delta(u') \geq (k, l+1) > \delta(u)$ ou $l = \mathrm{rg}(u') - 1$ et, dans ce cas, $\delta(u') \geq (k+1, 1) > \delta(u)$. □

Lemme 2.9. *Le nombre de nœuds rouges non terminaux est un $O(n\alpha(n))$.*

Démonstration. En raison du lemme précédent et des bornes sur δ du lemme 2.6 , pour tout nœud u la suite

$$u, u', u'', \ldots$$

contient au plus $\alpha(n)\,\mathrm{rg}\,u$ nœuds non terminaux rouges. Par suite le nombre de nœuds rouges non terminaux est borné par

$$\begin{aligned} \alpha(n) \sum_{u : u=(x,m)} \mathrm{rg}(u) &= \alpha(n) \sum_{r=0}^{\infty} r |\{u = (x,m) \mid \mathrm{rg}(u) = r\}| \\ &\leq \alpha(n) \sum_{r=0}^{\infty} rn/2^{r-2} \\ &\leq n\alpha(n) \sum_{r=0}^{\infty} r/2^{r-2} \\ &\leq 8n\alpha(n). \end{aligned}$$

□

Preuve du Théorème 2.3. Simple combinaison des lemmes précédents. □

Exercice 2.8. Lire les preuves du théorème de Tarjan ci-dessus données dans le Kozen, le Cormen and co et le Berstel and co.

Indication pour les exercices. Indication pour l'exercice 2.1

n		
0	$\varnothing$	
1	$\{\{1\}\}$	
2	$\{\{1\},\{2^\circ\}\}$	✓
	$\{\{1,2^\circ\}\}$	
3	$\{\{1\},\{2\},\{3^\circ\}\}$	✓
	$\{\{1\},\{2,3^\circ\}\}$	
	$\{\{1,3^\circ\},\{2\}\}$	
	$\{\{1,2,3^\circ\}\}$	✓
	$\{\{1,2\},\{3^\circ\}\}$	
4	$\{\{1\},\{2\},\{3\},\{4^\circ\}\}$	✓
	$\{\{1\},\{2\},\{3,4^\circ\}\}$	
	$\{\{1\},\{2,4^\circ\},\{3\}\}$	
	$\{\{1,4^\circ\},\{2\},\{3\}\}$	
	$\{\{1,4^\circ\},\{2,3\}\}$	✓
	$\{\{1\},\{2,3,4^\circ\}\}$	
	$\{\{1\},\{2,3\},\{4^\circ\}\}$	
	$\{\{1,3\},\{2\},\{4^\circ\}\}$	✓
	$\{\{1,3\},\{2,4^\circ\}\}$	
	$\{\{1,3,4^\circ\},\{2\}\}$	
	$\{\{1,2,3,4^\circ\}\}$	✓
	$\{\{1,2,3\},\{4^\circ\}\}$	
	$\{\{1,2\},\{3\},\{4^\circ\}\}$	✓
	$\{\{1,2\},\{3,4^\circ\}\}$	
	$\{\{1,2,4^\circ\},\{3\}\}$	

Références

[1] D. Beauquier, J. Berstel, and P. Chrétienne. *Eléments d'algorithmique*. Masson, 1992.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.

[3] R. Kaye. A gray code for set partitions. *Information Processing Letters*, 5(6) :171–173, 1976.

[4] D. C. Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, 1992.

[5] R. E. Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26 :690–715, 1979.

[6] R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

3 Arbres binaires de recherche

Compléments sur les arbres binaires : sous-arbre gauche, sous-arbre droit, composition, génération aléatoire, ordre infixe et ordre prefixe sur les nœuds d'un arbre binaire, rotations - définition des arbres binaires de recherche - sous-ensemble canonique associé à un nœud - Chemin de recherche d'une clé : partition induite - insertion d'une clé - arbre binaire de recherche aléatoire - hauteur d'un arbre binaire de recherche aléatoire - dynamisation des arbres binaires de recherche aléatoires - Queue de distribution : borne de Chernoff pour les variables aléatoires harmoniques - notation $\tilde{O}(f(n))$.

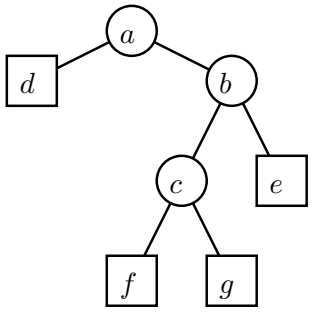
Randomized Search Trees
by R. Seidel and C.R. Aragon, in *Algorithmica* 16, pages 464–497, 1996
(FOCS'89).

Abstract. We present a randomized strategy for maintaining balance in dynamically changing search trees that has optimal expected behavior. In particular, in the expected case a search or an update takes logarithmic time, with the update requiring fewer than two rotations. Moreover, the update time remains logarithmic, even if the cost of a rotation is taken to be proportional to the size of the rotated subtree. Finger searches and splits and joins can be performed in optimal expected time also. We show that these results continue to hold even if very little true randomness is available, i.e., if only a logarithmic number of truely random bits are available. Our approach generalizes naturally to weighted trees, where the expected time bounds for accesses and updates again match the worst-case time bounds of the best deterministic methods. We also discuss ways of implementing our randomized strategy so that no explicit balance information is maintained. Our balancing strategy and our algorithms are exceedingly simple and should be fast in practice.

Sous-arbre gauche et sous-arbre droit Soit A un arbre binaire. Le **sous-arbre** de A de racine un nœud donné de A, appelons-le ν, est l'arbre binaire de fonctions fils gauche et fils droit les restrictions des fonctions fils gauche et fils droit de A à l'ensemble des nœuds de A dont les chemins de recherche commencent par le chemin de recherche de ν dans A. En particulier le sous-arbre de A de racine la racine de A est l'arbre A lui-même et les sous-arbres de racines les nœuds externes sont des arbres vides.

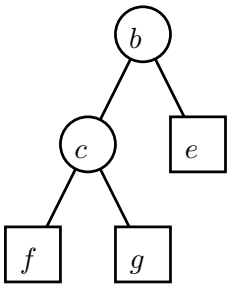
Exercice 3.1. Quel est le nombre de sous-arbres d'un arbre binaire d'ordre $2n+1$? Combien sont des arbres vides ? 3.1

Sous l'hypothèse A non vide, le **sous-arbre gauche** de A est le sous-arbre de A de racine l'image par la fonction fils gauche de la racine de A, et le **sous-arbre droit** de A est le sous-arbre de A de racine l'image par la fonction fils droit de la racine de A. Ainsi les sous-arbres gauche et droit de l'arbre



sont

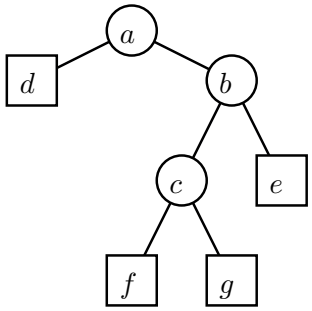
et



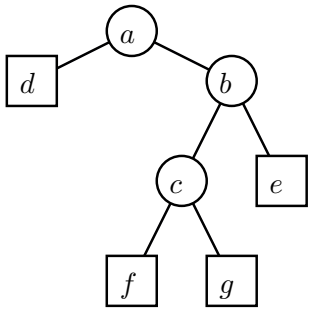
Composition. Le lecteur notera que le singleton formé de la racine, le sous-arbre gauche et le sous-arbre droit d'un arbre binaire non vide sont à supports disjoints (le support d'un arbre binaire est son ensemble de nœuds). Réciproquement, étant donné un singleton $\{\nu\}$, un arbre binaire B et un arbre binaire C à supports disjoints, il existe un unique arbre binaire de racine ν, de sous-arbre gauche B et de sous-arbre droit C. Cet arbre binaire est appelé le composé le ν, B et C et est noté $\Psi_2(\nu, B, C)$. Pour compléter le tableau nous écrirons $\Psi_0(\nu)$ pour l'arbre vide dont le seul nœud est l'élément ν. Par exemple

$$\Psi_2(a, \Psi_0(d), \Psi_2(b, \Psi_2(c, \Psi_0(f), \Psi_0(g), \Psi_0(e))))$$

est l'écriture sous forme de composition des opérateurs Ψ_2 et Ψ_0 de l'arbre binaire



Clairement l'opérateur Ψ_2 est compatible avec la relation d'isomorphisme d'arbres binaires : si B et B' sont isomorphes, si C et C' sont isomorphes alors les arbres $\Psi_2(\nu, B, C)$ et $\Psi_2(\nu', B', C')$ sont isomorphes. Nous noterons $\circ$ l'opérateur (binaire) induit sur les classes d'isomorphisme d'arbres binaires et nous utiliserons la notation $(\circ, B, C)$ en lieu et place de la notation $\circ(B, C)$ pour désigner l'image par $\circ$ des classes B et C. Enfin nous noterons $\square$ la classe d'isomorphisme des arbres vides, classe d'isomorphisme que nous penserons comme un opérateur nullaire. Ainsi toute classe d'isomorphisme d'arbres binaires s'écrit comme un composé des opérateurs $\circ$ et $\square$. Par exemple la classe d'isomorphisme de l'arbre



s'écrit sous la forme

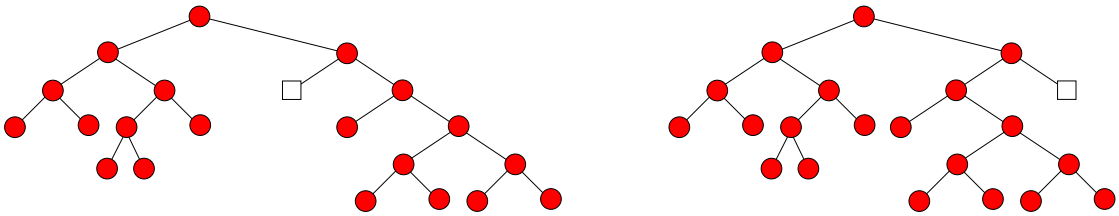
$$(\circ, \square, (\circ, (\circ, \square, \square), \square)).$$

Selon le contexte nous n'hésiterons pas, si cela ne prête pas à confusion, à parler d'arbre binaire pour une classe d'isomorphisme d'arbres binaires. Par exemple l'arbre binaire vide pour la classe des arbres binaires vides.

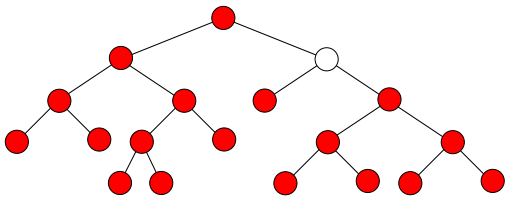
Exercice 3.2. Quels sont les nombres de nœuds externes des arbres binaires (des classes d'arbres binaires) A_n définis par les relations $A_0 = \square$, $A_{n+1} = (\circ, A_n, \square)$. Même question avec les arbres binaires (des classe d'arbres binaires) A_n définis par $A_0 = \square$, $A_{n+1} = (\circ, A_n, A_n)$. 3.2

Génération aléatoire d'un arbre binaire. Etant donné un arbre A et un sous-arbre propre B de A de chemin de recherche $X_1X_2\dots X_{m+1}$, $X_i \in \{G, D\}$, on note $p(B, A)$ le sous-arbre de A de chemin de recherche $X_1X_2\dots X_m$ et $s(B, A)$ le sous-arbre de A de chemin de recherche $X_1X_2\dots \overline{X}_{m+1}$ où $\overline{X} = G$ si $X = D$; D sinon.

Soit $\mathcal{F}_{n+1}$ l'ensemble des classes d'isomorphismes d'arbres binaires de taille $n+1$ dont un nœud externe est distingué, soit $\mathcal{N}_n$ l'ensemble des classes d'isomorphismes d'arbres binaires de taille $n \geq 0$ dont un sous-arbre est distingué et soit $r : \mathcal{F}_{n+1} \to \mathcal{N}_n$ l'application qui associe à l'arbre A, distingué en le nœud externe ν, l'arbre, distingué en $s(\nu, A)$, obtenu à partir de A en substituant au sous-arbre $p(\nu, A)$ l'arbre $s(\nu, A)$. Par exemple l'image par r des deux arbres suivant distingués en la feuille blanche :



est l'arbre suivant distingué en le sous-arbre de racine le nœud blanc :



En d'autres termes en écrivant la classe d'isomorphisme d'un arbre binaire dont un sous-arbre est distingué comme un composé d'opérateurs $\circ$ et $\square$, marqué, en le soulignant par exemple, en le composé correspondant au sous-arbre distingué, l'application r est définie par

$$r\left(X(\circ, \underline{\square}, Y)Z\right) = r(X(\circ, Y, \underline{\square})Z) = X\underline{Y}Z.$$

Théorème 3.1. *L'application $r : \mathcal{F}_{n+1} \to \mathcal{N}_n$ est bien définie, surjective et tout élément de $\mathcal{N}_n$ a deux antécédents par r.*

Démonstration. ✍ □

De cette correspondance nous déduisons (✍) que le nombre a_n de classes d'isomorphismes d'arbres binaires de taille n vérifie la relation de récurrence

$$2(2n+1)a_n = (n+2)a_{n+1}.$$

De cette relation de récurrence et de la condition initiale $a_0 = 1$ on déduit facilement que le nombre de classes d'isomorphismes d'arbres binaires de taille n est le n-ième nombre de Catalan. En effet :

$$\begin{aligned}\frac{a_n}{a_0} &= \prod_{k=0}^{k=n-1} \frac{a_{k+1}}{a_k} \\ &= \prod_{k=0}^{k=n-1} \frac{2(2k+1)}{k+2} \\ &= \prod_{k=0}^{k=n-1} \frac{(2k+1)(2k+2)}{(k+1)(k+2)} \\ &= \frac{(2n)!}{n!n!(n+1)} = \frac{1}{n+1}\binom{2n}{n}\end{aligned}$$

De cette correspondance nous déduisons aussi un algorithme efficace de génération aléatoire selon la loi uniforme d'un arbre binaire de taille donnée : un arbre binaire aléatoire de taille $n+1$ est obtenu à partir d'un arbre binaire aléatoire A de taille n en choisissant de manière aléatoire selon la loi uniforme l'un de ses $2n+1$ sous-arbres Y puis en remplaçant Y dans A par l'un des deux arbres $(\circ, \square, Y)$, $(\circ, Y, \square)$, chacun étant choisi avec probabilité 1/2.

Exercice 3.3. Implémenter cet algorithme de génération aléatoire d'un arbre binaire et vérifier expérimentalement la formule suivante pour la profondeur moyenne d'un nœud interne :

$$p(n) = \frac{4^n - \frac{3n+1}{n+1}\binom{2n}{n}}{\frac{n}{n+1}\binom{2n}{n}} = \sqrt{\pi n}\left(1 + \frac{9}{8n} + \frac{17}{128n^2} + O(n^{-3})\right) - 3 - \frac{1}{n}$$

n	10	20	50	100	200	500	∞
$p(n)/\sqrt{n}$	0.99	1.19	1.38	1.49	1.57	1.64	1.77

3.3

Ordre infixe et ordre préfixe. L'**ordre infixe** $\prec$ sur les nœuds d'un arbre binaire est défini par les relations $\nu' \prec \nu \prec \nu''$ où ν décrit l'ensemble des nœuds internes, ν' l'ensemble des nœuds du sous-arbre gauche du sous-arbre de racine ν et ν'' l'ensemble des nœuds du sous-arbre droit du sous-arbre de racine ν. En d'autres termes l'application Φ définie sur l'univers des arbres binaires récursivement par les relations

$$\Phi(B) = \begin{cases} \nu & \text{si } B = \Psi_0(\nu) \\ \Phi(A)\nu\Phi(A') & \text{si } B = \Psi_2(\nu, A, A') \end{cases}$$

associe à un ab la suite croissante de ses nœuds pour l'ordre infixe.

L'**ordre préfixe** $\prec$ sur les nœuds d'un arbre binaire est défini par les relations $\nu \prec \nu' \prec \nu''$ où ν décrit l'ensemble des nœuds internes, ν' l'ensemble des nœuds du sous-arbre gauche du sous-arbre de racine ν et ν'' l'ensemble des nœuds du sous-arbre droit du sous-arbre de racine ν. En d'autres termes l'application Φ définie sur l'univers des arbres binaires récursivement par les relations

$$\Phi(B) = \begin{cases} \nu & \text{si } B = \Psi_0(\nu) \\ \nu\Phi(A)\Phi(A') & \text{si } B = \Psi_2(\nu, A, A') \end{cases}$$

associe à un ab la suite croissante de ses nœuds pour l'ordre préfixe

Exercice 3.4. Montrer que l'ordre infixe sur les nœuds d'un arbre binaire détermine la bipartition nœuds internes/nœuds externes mais ne détermine pas l'arbre binaire. Quel est le nombre d'arbres binaires à ordre infixe sur les nœuds fixé ? 3.4

Exercice 3.5. Montrer que l'ordre préfixe sur les nœuds d'un arbre binaire ne détermine pas la bipartition nœuds internes/nœuds externes. 3.5

Exercice 3.6. Montrer que l'ordre préfixe sur les nœuds d'un arbre binaire augmenté de la bipartition nœuds internes/nœuds externes détermine l'arbre binaire. Donner un algorithme de complexité linéaire qui prend en entrée la suite croissante pour l'ordre préfixe des nœuds d'un arbre binaire (avec le distingo nœuds internes/nœuds externes) et retourne l'arbre binaire sous la forme d'une composition des opérateurs Ψ_2 et Ψ_0 (que l'on considèrera comme une suite de symboles : Ψ_2, Ψ_0, parenthèse ouvrante "(", parenthèse fermante ")", virgule ",", nœuds externes et nœuds internes). 3.6

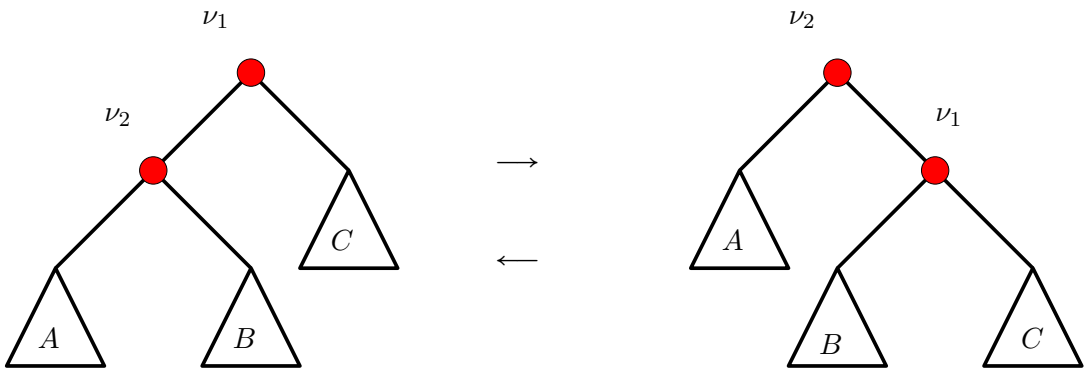


FIGURE 1 – Rotation droite $\longrightarrow$ et rotation gauche $\longleftarrow$

Rotations. Appliquer une **rotation droite** à un arbre binaire consiste à substituer à l'un de ses sous-arbres ayant un sous-arbre gauche non vide :

$$\Psi_2(\nu_1, \Psi_2(\nu_2, A, B), C)$$

l'arbre

$$\Psi_2(\nu_2, A, \Psi_2(\nu_1, B, C)).$$

(Pour spécifier le lieu de la rotation nous dirons que la rotation a lieu en l'arête $\nu_2\nu_1$.) L'opération inverse est appelée **rotation gauche**. L'application d'une rotation à un arbre binaire ne change pas son support, ne change pas l'ordre infixe de ses nœuds mais peut changer sa hauteur, et c'est là tout son intérêt comme nous le verrons dans le paragraphe sur la dynamisation des arbres binaires de recherche.

Exercice 3.7. Lors d'une rotation quels sont les nœuds qui changent de hauteur ? de combien ? 3.7

Exercice 3.8. Quels sont les arbres binaires auxquels on ne peut appliquer de rotations droites ? de rotations gauches ? Que peut-on en déduire sur les graphes de rotations, i.e. les graphes de nœuds les arbres binaires sur un ensemble donné de nœuds et d'arêtes les paires d'arbres binaires reliés par une rotation ? (On dessinera les graphes de rotations sur les arbres binaires de tailles $0, 1, 2$ et 3.) 3.8

Arbre binaire de recherche. Un **univers** est un ensemble totalement ordonné dont les éléments sont appelés des **clés**.

Soit S un ensemble de n clés d'un univers $\mathcal{U}$.

Un **arbre binaire de recherche** sur S est un arbre binaire de taille n dont les nœuds internes sont étiquetés bijectivement par les clés de S de telle sorte que l'ordre infixe sur les nœuds correspond à l'ordre croissant sur les clés. Ainsi en désignant par $K(\nu)$ la clé qui étiquette le nœud interne ν l'application Φ définie récursivement par les relations

$$\Phi(B) = \begin{cases} \epsilon & \text{si } B = \Psi_0(\nu) \\ \Phi(A)K(\nu)\Phi(A') & \text{si } B = \Psi_2(\nu, A, A') \end{cases}$$

associe à un abr la suite croissante des clés des nœuds de l'abr. Notons qu'il n'existe qu'une seule façon d'étiquetter par les clés de S les nœuds internes d'un ab de taille n de telle sorte que l'ab soit un abr sur S.

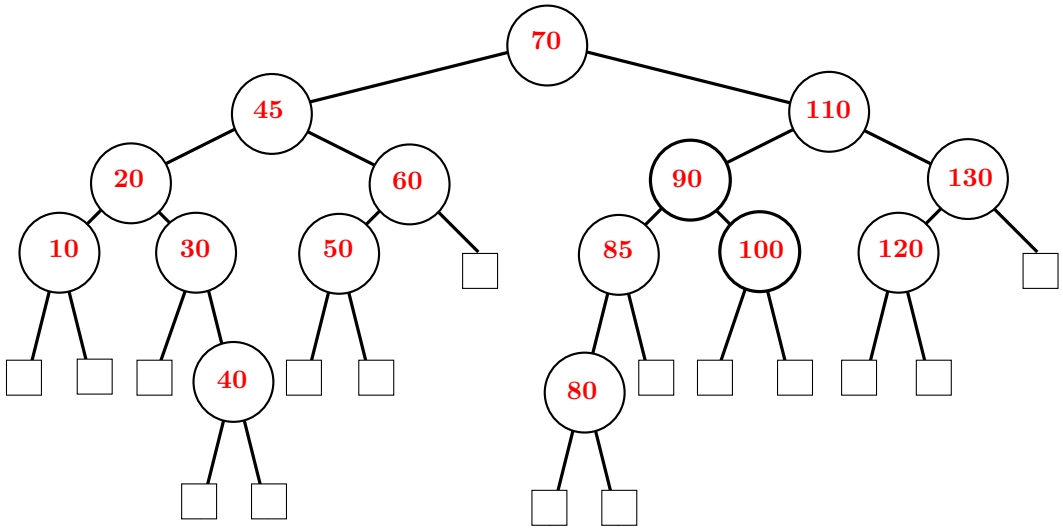


FIGURE 2 – Un arbre binaire de recherche sur l'ensemble des entiers 10, 20, 30, 40, 45, 50, 60, 70, 80, 85, 90, 100, 110, 120 et 130

Le **sous-ensemble canonique** du nœud ν ou encore **le sous-ensemble canonique** du sous-arbre de racine ν est l'ensemble $\mathcal{S}(\nu)$ des clés des nœuds du sous-arbre de racine ν. Par exemple pour l'arbre binaire de recherche de la Figure 2 et en désignant un nœud par son chemin de recherche

$$\begin{aligned} \mathcal{S}(GGD) &= \{30, 40\} \\ \mathcal{S}(GD) &= \{50, 60\} \\ \mathcal{S}(DG) &= \{80, 85, 90, 100\} \\ \mathcal{S}(DDG) &= \{120\} \end{aligned}$$

Proposition 3.2. *Soit A un arbre binaire de recherche sur l'ensemble S et soit a la clé de la racine de A. Alors*

(**1-**) $\mathcal{S}(\epsilon) = S$;
(**2-**) $\mathcal{S}(G) = \{k \in S \mid k < a\}$;
(**3-**) $\mathcal{S}(D) = \{k \in S \mid k > a\}$.

De plus tout sous-arbre de A est un arbre binaire de recherche sur le sous-ensemble canonique associé.

Exercice 3.9. Soit B un abr sur S de taille n et de hauteur h. Montrer que

$$\sum_{\nu} \#\mathcal{S}(\nu) = O(nh)$$

où ν décrit l'ensemble des nœuds internes de B. 3.9

La structure de données arbre binaire. *Pointeurs, structures récursives, etc* Par exemple en Pascal [2, Chap. 10].

```
type tree =↑node ;
  node = record
    k : typekey;
    left, right : tree
    end ;
```

ou en C [3, page 140]

```
struct tnode {
  typekey k ;
  struct tnode *left ;
  struct tnode *right ;
} ;
```

Recherche d'une clé.

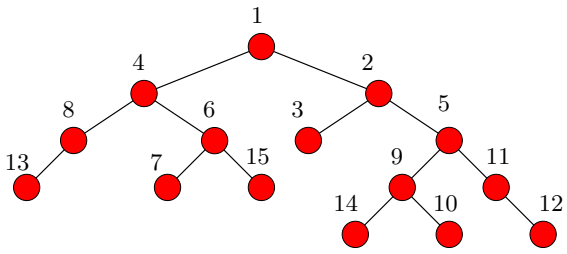
```
 procedure search(key : typekey; var t : tree);
0. begin
1. if t = nil then notfound(key)
2. else if t↑.k = key then found(t↑)
3. else if t↑.k < key then search(key, t↑. right)
4.    else search(key, t↑. left)
5. end ;
```

Insertion d'une clé dans un arbre binaire de recherche. L'insertion s'effectue aux feuilles selon la procédure suivante

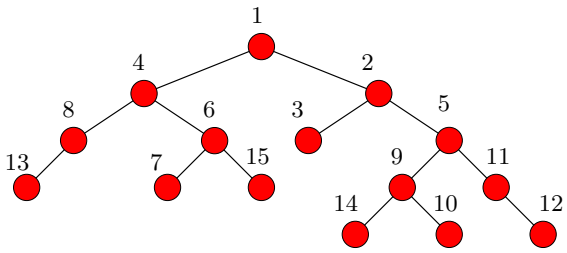
procedure $insert(key : typekey;$ **var** $t : tree);$
0. **begin**
1. **if** $t =$ **nil then** $t := NewNode(key,$ **nil**, **nil**$)$
2. **else if** $t\uparrow.k = key$ **then** $Error$
3. **else if** $t\uparrow.k < key$ **then** $insert(key, t\uparrow.right)$
4. **else** $insert(key, t\uparrow.left)$
5. **end**;

Le coût, en nombre de comparaisons, d'une insertion est donc égal à la longueur du chemin de recherche de la clé insérée, quantité bornée par la hauteur de l'arbre binaire de recherche.

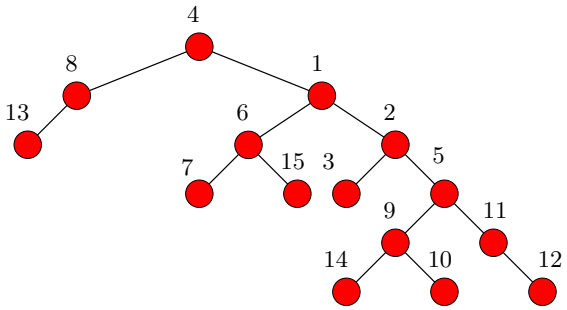
Suppression d'une clé dans un arbre binaire de recherche. Dans le cas particulier où le nœud support, appelons-le ν, de la clé à supprimer a pour fils gauche et fils droit, appelons-les ν' et ν'', des nœuds externes la suppression consiste simplement à remplacer le sous-arbre de racine ν par un nœud externe. Le cas général se réduit au cas particulier au moyen d'une séquence de rotations en l'arête $\nu\nu'$ si ν'' est un nœud externe, en l'arête $\nu\nu''$ si ν' est un nœud externe, en l'arête $\nu\nu'$ ou l'arête $\nu\nu''$ (à notre guise) sinon. Chaque rotation ayant pour effet de diminuer strictement la hauteur du sous-arbre de racine ν (✍) le coût de la suppression d'une clé est borné par la hauteur de l'arbre. Par exemple pour supprimer la racine de l'arbre binaire de recherche (les clés ne sont par indiquées sur le dessin)



on commence ... (TPSVP)



par effectuer une rotation en l'arête 41 (ou 21, nous avons le choix) :



puis en l'arête 21 (ou 61, nous avons encore le choix), etc., etc.

Exercice 3.10. Terminer la suppression du nœud 1. Par ailleurs le paragraphe précédent est fautif : voyez-vous en quoi ? corriger. 3.10

Insertion d'une suite de clés dans un arbre binaire de recherche.

procedure *insert_keys*(*keys* : **array**[1..*n*] **of** *typekey*; **var** *t* : *tree*);
0. **begin**
0.
1. **for** *i* := 1 **to** *n* *insert*(*keys*[*i*], *tree*)
5. **end** ;

Le coût, en nombre de comparaisons, de l'insertion d'une suite d'éléments dans un arbre initialement vide est donc égal à la somme des longueurs des chemins de recherche des éléments dans l'arbre final. Cette quantité est appelée la **longueur de cheminement interne** de l'arbre :

$$\begin{aligned} \mathrm{lci}(A) &= \sum_{\nu} |\,\mathrm{CheRech}(\nu, A)| \\ &= \sum_{k} k n_k \end{aligned}$$

où ν décrit l'ensemble des noeuds internes de A et où n_k est le nombre de nœuds internes de profondeur k.

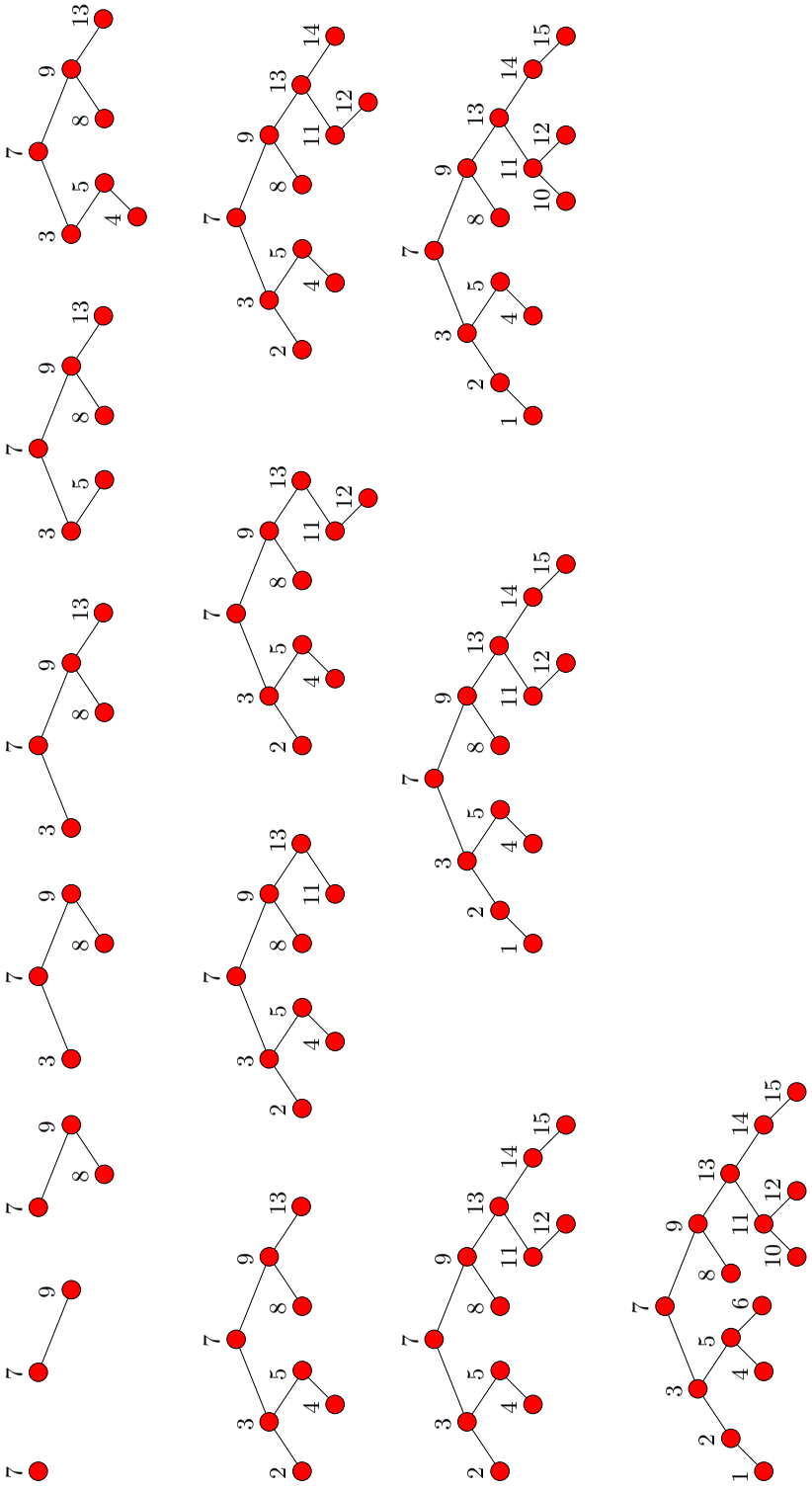


FIGURE 3 – La suite des quinze arbres binaires de recherche successivement produits lors de l'insertion de la suite d'entiers $[7, 9, 8, 3, 13, 5, 4, 2, 11, 12, 14, 15, 1, 10, 6]$ dans un arbre initialement vide. Le coût de construction de l'arbre final, en nombre de comparaisons, est sa longueur de cheminement interne, ici $37 \, (= 2 \times 1 + 4 \times 2 + 5 \times 3 + 3 \times 4)$.

Hauteur d'un arbre binaire de recherche aléatoire. Un arbre binaire de recherche aléatoire sur un ensemble S de n clés est un arbre binaire de recherche sur S choisi de manière aléatoire pour la loi uniforme dans le multi-ensemble des

$$insert\_keys(\sigma, \mathbf{nil}),$$

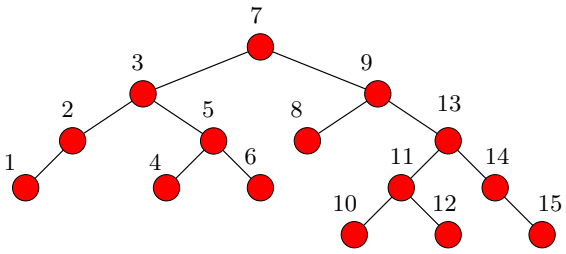
où σ décrit l'ensemble des permutations des clés de S. Un tel arbre est obtenu en insérant successivement aux feuilles d'un arbre binaire de recherche initialement vide les n clés selon un ordre aléatoire pour la loi uniforme :

procedure $abr\_aleatoire(keys : \mathbf{array}[1..n]\,\mathbf{of}\,typekey;\ \mathbf{var}\ t : tree)$;
0. **begin**
0. choose a random permutation σ of $[\![n]\!]$
1. **for** $i := 1$ **to** n $insert(keys[\sigma(i)], tree)$
5. **end**;

Le lecteur vérifiera facilement que les 3 permutations suivantes des entiers de 1 à 15

$$[7,3,2,1,5,4,6,9,8,13,11,10,12,14,15]$$
$$[7,9,13,14,15,11,10,12,8,3,2,1,5,6,4]$$
$$[7,3,2,1,5,6,4,9,13,14,15,11,10,12,8]$$

conduisent au même arbre binaire de recherche :



De fait leur nombre est

$$\frac{15!}{15 \times 6 \times 8 \times 2 \times 3 \times 6 \times 3 \times 2}$$

soit 8 408 400.

Théorème 3.3 (Formule des équerres). *Soit B un arbre binaire de recherche sur un ensemble de n clés. Alors le nombre de permutations des n clés produisant l'arbre binaire de recherche B est égal à*

$$\frac{n!}{\prod_\tau t(\tau)}$$

où τ décrit l'ensemble des nœuds internes de l'arbre B et où $t(\tau)$ est la taille du sous-arbre de B de racine τ.

Démonstration. Soit $N(B)$ le nombre cherché. Alors $N(B)$ vérifie (pourquoi ?) la récurrence $N(B) = 1$ si B est arbre vide ; sinon

$$N(B) = \binom{|B|-1}{|\mathrm{sag}(B)|} N(\mathrm{sag}(B))N(\mathrm{sad}(B)).$$

où $\mathrm{sag}(B)$ et $\mathrm{sad}(B)$ sont les sous-arbres gauche et droit de B et où $|B|$ est la taille de B. Par suite

$$\frac{N(B)}{|B|!} = \frac{1}{|B|} \frac{N(\mathrm{sag}(B))}{|\mathrm{sag}(B)|!} \frac{N(\mathrm{sad}(B))}{|\mathrm{sad}(B)|!}$$

Le théorème suit. □

Exercice 3.11. Quels sont les arbres binaires de recherche sur un ensemble de n clés produit par une unique permutation des clés ? par le plus grand nombre de permutations ?
3.11

Théorème 3.4. *La hauteur d'un arbre binaire de recherche aléatoire sur un ensemble de n clés est un $\widetilde{O}(\log n)$, i.e., que la probabilité que cette hauteur soit $\geq c \log n$ est inférieure à $1/p(n)$ où $p(n)$ est un polynôme en n dont le dégré tend vers l'infini lorsque c tend vers l'infini.*

Démonstration. Soit S un ensemble de n clés, soit $k_1, k_2, \ldots, k_n$ une permutation aléatoire des éléments de S et soit A l'arbre binaire de recherche sur S asssocié. Posons $K_j = \{k_1, k_2, \ldots, k_j\}$. Fixons une clé k n'appartenant pas à S et soit $X_j(k)$ la variable aléatoire égale 1 si, lors de la recherche de k dans l'arbre A, k est comparée avec la clé k_j ; 0 sinon. La longueur du chemin de recherche de k dans l'arbre A est $X(k) = \sum_i X_i(k)$. Soit B une partie à j éléments de l'ensemble des n clés, soit b la clé de B qui précède k dans la liste triée des éléments de $B \cup \{k\}$ et soit b' celle qui suit k. Alors

- **(1-)** $E[X_j(k) \mid K_j = B] \leq 2/j$
  car la probabilité que $X_j(k) = 1$ sachant que $K_j = B$ est la probabilité de l'évènement $k_j = b$ ou b' sachant que $K_j = B$, soit au plus $2/j$.
- **(2-)** $E[X_j(k)] = \sum_B E[X_j(k) \mid K_j = B] \mathrm{Prob}[K_j = B] \leq 2/j$
  par application de la formule des probabilités totales
- **(3-)** $X(k) = \widetilde{O}(\log n)$
  par application de la borne de Chernoff pour les variables harmoniques, cf. Théorème 3.6 en fin de chapitre.

Posons $X = \max_k X(k)$. Soit $(k'_1, k'_2, \ldots, k'_n)$ la liste triée des k_i et soit $c_1, c_2, \ldots, c_{n+1}$ des clés telles que $c_1 < k'_1 < c_2 < k'_2 < \cdots < c_n < k'_n < c_{n+1}$. Alors

$$X = \max\{X(c_1), X(c_2), \ldots, X(c_{n+1})\}.$$

La probabilité de l'évènement $X \geq c \log n$ est majorée par la somme des probabilités des évènements $X(c_j) \geq c \log n$, soit majorée par $n/p(n) = 1/q(n)$ où $p(n)$ est un polynôme

en n dont le degré peut être rendu aussi grand que voulu en choisissant c assez grand. La borne $X(k) = \widetilde{O}(\log n)$ est donc indépendante de la clé requête k. □

Théorème 3.5. *Le problème du tri par comparaisons d'un ensemble totalement ordonné de n éléments est résoluble en $\widetilde{O}(n \log n)$ comparaisons.*

Démonstration. Construire un arbre binaire de recherche aléatoire sur les n clés et effectuer un parcours infixe de l'arbre. Le coût de construction de l'arbre est majoré par n fois sa hauteur. Sa hauteur étant un $\widetilde{O}(\log n)$, n fois sa hauteur est un $\widetilde{O}(n \log n)$ et son coût de construction un $\widetilde{O}(n \log n)$. □

Le tri-rapide. *Description di tri-rapide - arbre binaire de recherche associé au déroulement du tri-rapide - analyse de complexité*

Dynamisation des arbres binaires de recherche aléatoires. Le problème de la dynamisation des arbres binaires de recherche aléatoires est celui de maintenir un arbre binaire de recherche aléatoire initialement vide soumis à une séquence mixte d'insertions et de suppressions de clés. Comment faire ? Imaginons avoir associé des **priorités** $\in [0, \infty]$ aux clés de l'arbre courant et que ces priorités sont croissantes le long des branches (ou chemins de recherche) de l'arbre. Alors l'arbre est exactement l'arbre que l'on aurait obtenu en insérant dans un arbre initialement vide les clés de l'arbre selon l'ordre croissant de leurs priorités. De plus si les priorités ont été choisi de manière aléatoire pour la loi uniforme sur l'intervalle $[0, \infty]$, alors l'arbre est un arbre binaire de recherche aléatoire. Maintenant posons nous la question du rétablissement de l'ordre des priorités lorsque la priorité d'une clé varie ? Les rotations donnent la réponse :

(**1-**) La diminution de la priorité de la clé du nœud ν juste en deçà de la priorité de la clé du nœud parent ν' de ν nécessite pour rétablir l'ordre des priorités une simple rotation droite ou gauche selon que l'arête $\nu\nu'$ est une arête gauche ou droite, cf. Figure 4 ; de même

(**2-**) L'augmentation de la priorité de la clé du nœud ν juste au delà de la plus petite des priorités des clés des nœuds fils gauche et fils droit ν' et ν'' de ν nécessite pour rétablir l'ordre des priorités une rotation droite ou gauche selon que la priorité de la clé du nœud ν' est inférieure ou supérieure à celle de la clé du nœud ν'', cf. Figures 5.

Ces deux règles donnent la solution au problème de la dynamisation des arbres binaires de recherche aléatoires. L'insertion d'une clé consiste à insérer la clé avec une priorité infini puis à effectuer la séquence de rotations nécessaires au rétablissement de l'ordre des priorités lorsque la priorité de la clé passe continûment de l'infini à une valeur attribuée de manière aléatoire selon la loi uniforme sur l'intervalle $[0, \infty]$. (Voir la Figure 6i pour un exemple.) La suppression consiste à effectuer la séquence de rotations nécessaires au rétablissement de l'ordre des priorités lorsque la priorité de la clé à supprimer varie continûment de sa valeur courante (priorité attribuée lors de la dernière insertion de

la clé) à l'infini, puis à substituer un nœud externe au sous-arbre de racine le nœud support de la clé à supprimer. On peut montrer que recherche, insertion et suppression s'effectuent en temps $\widetilde{O}(\log n)$, cf. [6].

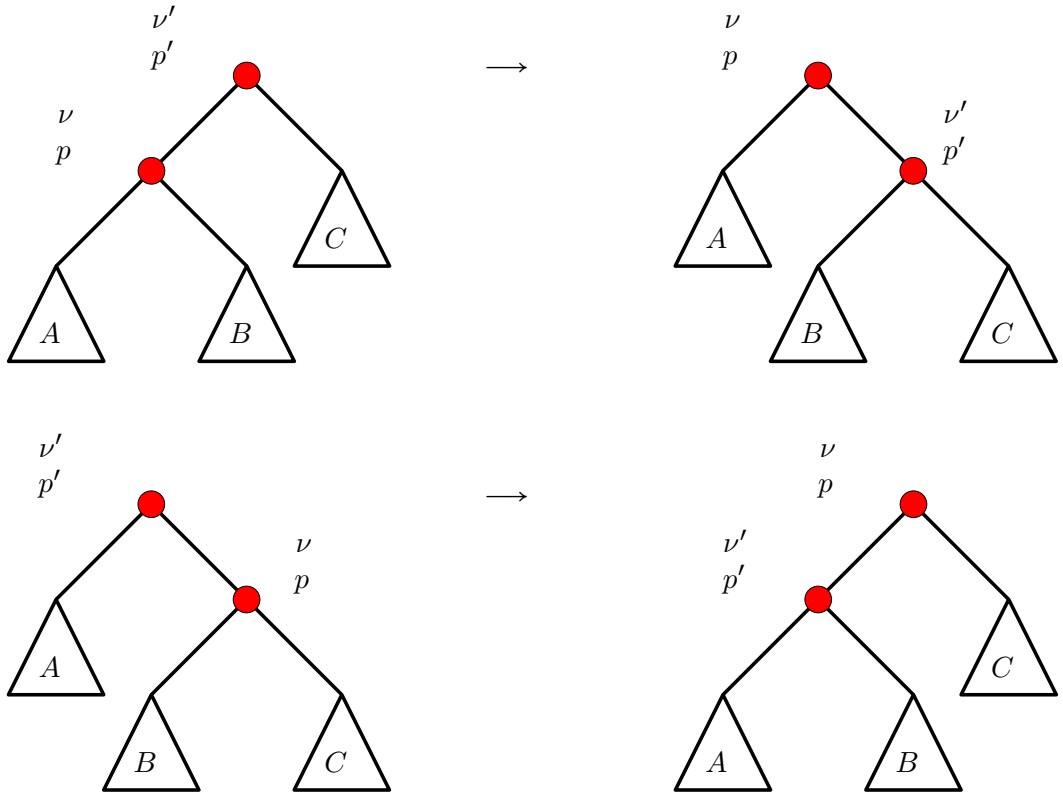


FIGURE 4 – La diminution de la priorité p de la clé du nœud ν en deçà de la priorité p' de la clé du nœud parent ν' se traduit par une rotation droite ou gauche selon que l'arête $\nu\nu'$ est une arête gauche ou droite

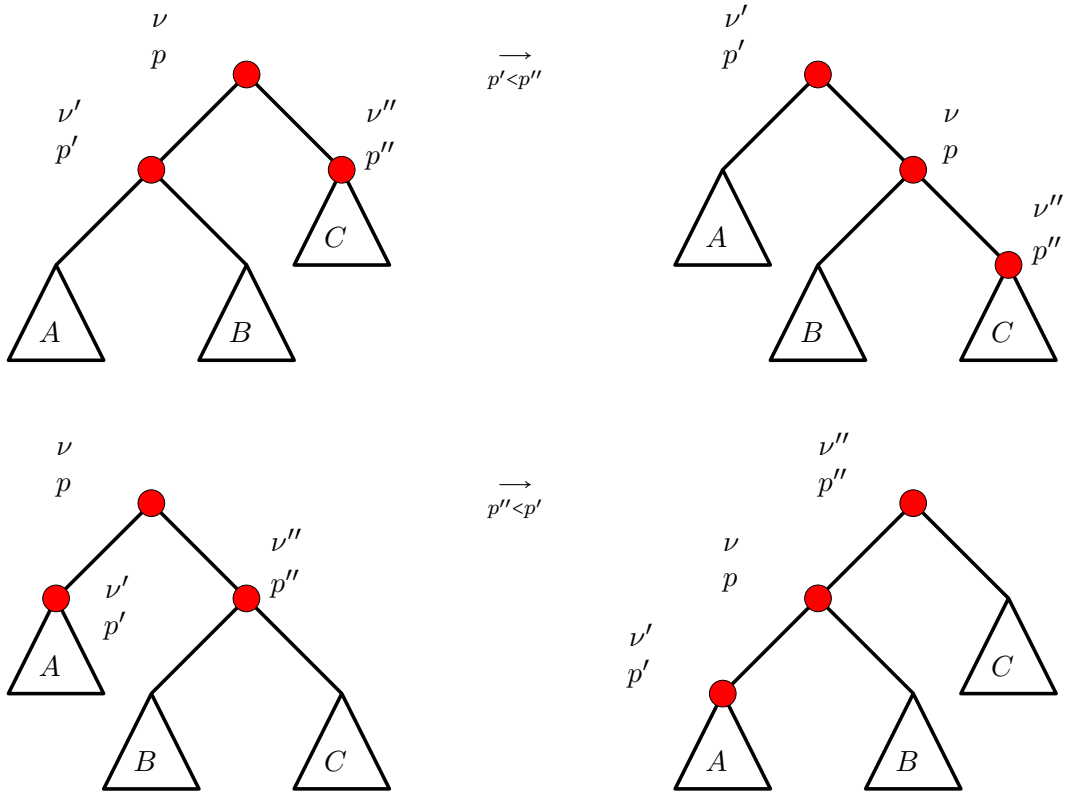


FIGURE 5 – L'augmentation de la priorité p de la clé du nœud ν au delà de la plus petite des priorités des clés des nœuds fils gauche et fils droit ν' et ν'' de ν se traduit par une rotation droite ou gauche selon que la priorité p' de la clé du nœud ν' est inférieure ou supérieure à la priorité p'' de la clé du nœud ν''.

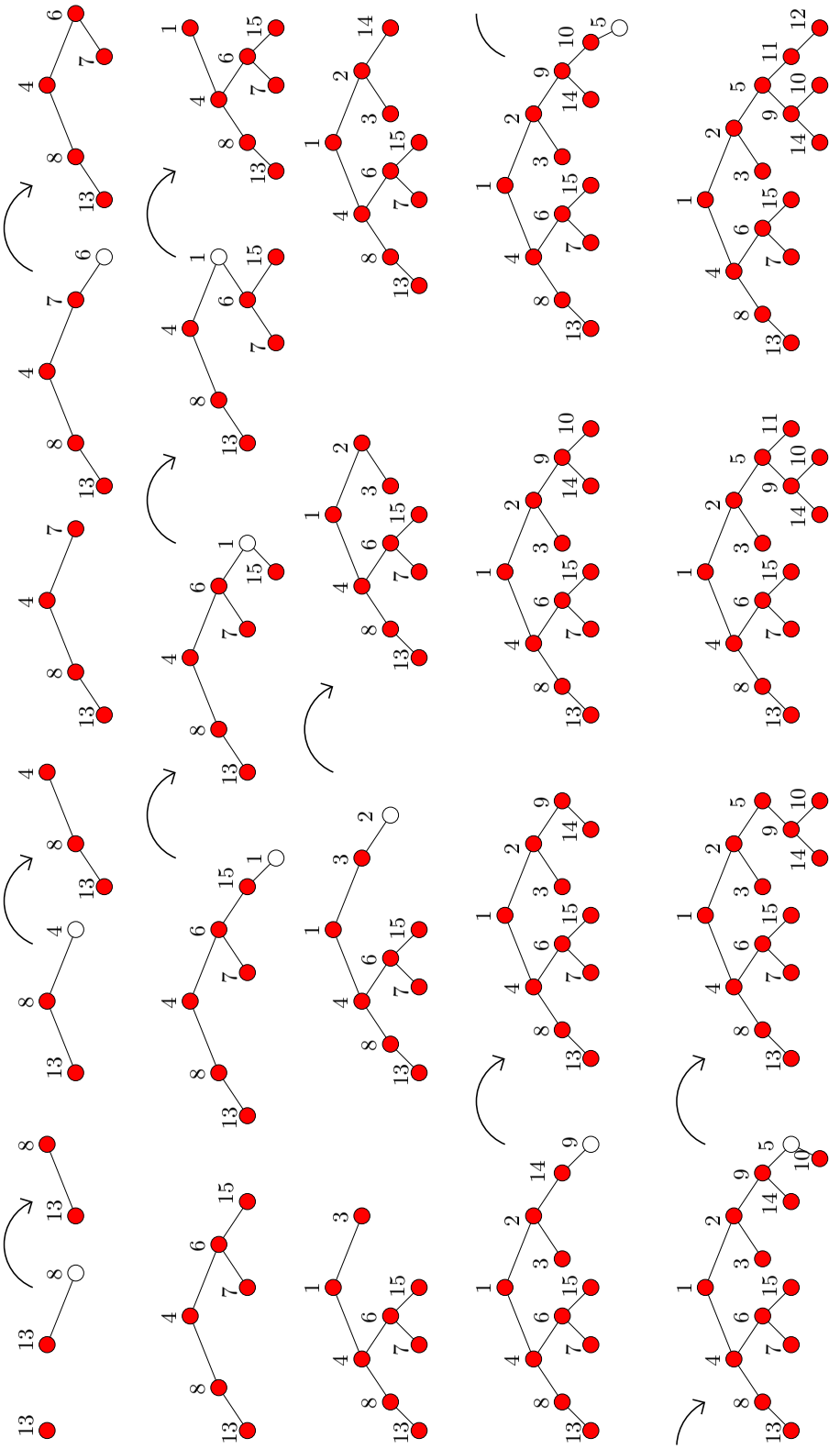
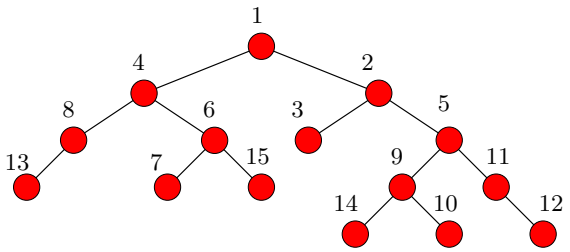


FIGURE 6 – Les arbres binaires de recherche construits successivement en insérant dans un arbre initialement vide les clés $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$ avec les priorités $13, 8, 4, 7, 6, 15, 1, 3, 2, 14, 9, 10, 5, 11, 12$.

Exercice 3.12. Supprimer la racine de l'arbre binaire de recherche ci-dessous. (Les nœuds de l'arbre sont uniquement étiquetés par leurs priorités.) 3.12



Borne de Chernoff relative aux variables aléatoires harmoniques.

Théorème 3.6. *Soit $X_1, X_2, \ldots, X_n$ une suite de n variables aléatoires de Bernoulli mutuellement indépendantes et de paramètres respectifs $1, 1/2, \ldots, 1/n$, soit $X = \sum_{i=1}^{n} X_i$ et $c \geq 1$. Alors*

(1-) $E[X] = H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$, *le n-ième nombre harmonique.*
(2-) $\text{Prob}[X \geq cH_n] \leq e^{-(1+c\ln\frac{c}{e})H_n} = O(\frac{1}{n^{1+c\ln c/e}})$. □

Démonstration. Nous suivons l'exposé de [5, Appendix A].

$$\begin{aligned}
\text{Prob}[X \geq x] &= \text{Prob}[e^{\lambda X} \geq e^{\lambda x}] && \text{(pour } \lambda > 0\text{)} \\
&\leq e^{-\lambda x} E[e^{\lambda X}] && \text{(inégalité de Markov)}
\end{aligned}$$

[Rappel : $\text{Prob}[X \geq \alpha] \leq \frac{E(X)}{\alpha}$ pour $X \geq 0, \alpha > 0$]

$$\begin{aligned}
&= e^{-\lambda x} \prod_{i=1}^{n} E[e^{\lambda X_i}] && \text{(indépendance des } X_i\text{)} \\
&= e^{-\lambda x} \prod_{i=1}^{n} \left\{ \frac{1}{i} e^{\lambda} + (1 - \frac{1}{i}) e^{0} \right\} \\
&= e^{-\lambda x} \prod_{i=1}^{n} \left\{ 1 + \frac{1}{i}(e^{\lambda} - 1) \right\} && (1 + u \leq e^{u}) \\
&\leq e^{-\lambda x} \prod_{i=1}^{n} e^{\frac{1}{i}(e^{\lambda}-1)} \\
&= e^{-\lambda x} e^{(e^{\lambda}-1)H_n}
\end{aligned}$$

Il reste à prendre $x = cH_n$ et $\lambda = \ln c$ pour obtenir la majoration annoncée. □

C'est un cas particulier des inégalités à large déviation inspirées du Théorème central limite [4, 1].

Références

[1] K. M. Ball. An elementary introduction to modern convex geometry. In *Flavors of geometry (S. Levy editor)*, volume 31 of *Mathematical sciences research institute*, pages 1–58. Cambridge University Press, 1997.

[2] K. Jensen and N. Wirth. *Pascal - User Manual and Report*. Springer-Verlag, 1978.

[3] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall Software Series, 1988.

[4] S. Levy, editor. *Flavors of Geometry*. Number 31 in MSRI Publications. Cambridge University Press, 1997.

[5] K. Mulmuley. *Computational Geometry : An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[6] R. Seidel and C. Aragon. Randomized search trees. *Algorithmica*, 16(4-5) :464–497, 1996.

4 Arbres couvrants optimaux

- Terminologie relative aux graphes - dessin de graphes - graphes isomorphes - graphes linéaires, graphes cycliques, graphes complets, graphes bipartis complets - sous-graphes, chemins et cycles d'un graphe - graphe connexe, composantes connexes d'un graphe - arbre - graphe orienté - arbre couvrant de poids minimal - algorithme de Kruskal - la méthode gloutonne - algorithme de Prim - tas de Fibonacci

La terminologie des graphes. Un **graphe** G est un triplet (V, E, φ) où V est un ensemble fini non vide, E un ensemble fini disjoint de V et φ une application de E dans l'ensemble des parties à un ou deux éléments de V.

Nous utiliserons le mot **sommet** ou **nœud** pour un élément de V, le mot **arête** pour un élément de E, l'expression **fonction d'attachement** pour la fonction φ, et nous dirons que G est un **graphe sur** V pour dire que V est l'ensemble des sommets de G.

Une **incidence** est une paire $(v, e) \in V \times E$ telle que $v \in \varphi(e)$; dans ce cas on dit que v est incident à e ou encore que v est une extrémité de e.

Proposition 4.1. *Soit $G = (V, E, \varphi)$ un graphe et soit I l'ensemble des incidences de G. Alors le triplet (V, E, I) détermine de manière unique φ.*

Démonstration. En effet la paire (V, E) détermine le domaine et le co-domaine de φ. Puis pour tout $e \in E$

$$\varphi(e) = \{v \in V \mid (v, e) \in I\}.$$

□

Ainsi on pourra définir un graphe par la donnée de l'ensemble de ses sommets, l'ensemble de ses arêtes et l'ensemble de ses incidences en lieu et place de sa fonction d'attachement.

La **multiplicité** d'une arête est le nombre d'arêtes ayant le même ensemble de sommets incidents. Une arête **simple** est une arête de multiplicité 1. Une arête **multiple** est une arête de multiplicité au moins 2. Une **boucle** est une arête incidente à un seul sommet. Un **graphe simple** est un graphe sans arêtes multiples. Un graphe **d'ordre** n est un graphe sur un ensemble de taille (ou cardinalité) n.

Exercice 4.1. Quel est le nombre de graphes simples sans boucles sur un ensemble à $n \geq 1$ éléments à renommage près des arêtes ? 4.1

Le **degré** d'un sommet est le nombre d'arêtes qui lui sont incidentes, une boucle étant comptée deux fois. Une **feuille** est un sommet de degré 1. Par exemple le graphe de sommets u, v, w, z, d'arêtes a, b, c, d, e, f et de fonction d'attachement $\varphi(a) = \{u\}$, $\varphi(b) = \{u, v\}$, $\varphi(c) = \{u, v\}$, $\varphi(d) = \{v, w\}$, $\varphi(e) = \{w, u\}$, $\varphi(f) = \{w, z\}$ est formé d'une boucle a, de deux arêtes multiples b et c et de trois arêtes simples d, e et f ; la multiplicité des arêtes b et c est 2 ; le degré du sommet u est 5 ; le sommet z est un feuille.

Il est commode pour définir un graphe de le représenter par un dessin dans le plan composé de points pour les sommets du graphe et d'arcs de Jordan pour les arêtes du graphe, arcs soumis, afin de représenter la fonction d'attachement, à la règle suivante : les extrémités des arcs de Jordan sont les points représentatifs des extrémités des arêtes, cf. Figure 1. De plus, afin d'éviter toute ambiguïté, les intersections éventuelles entre des arcs sont en général en nombre fini et transverses.

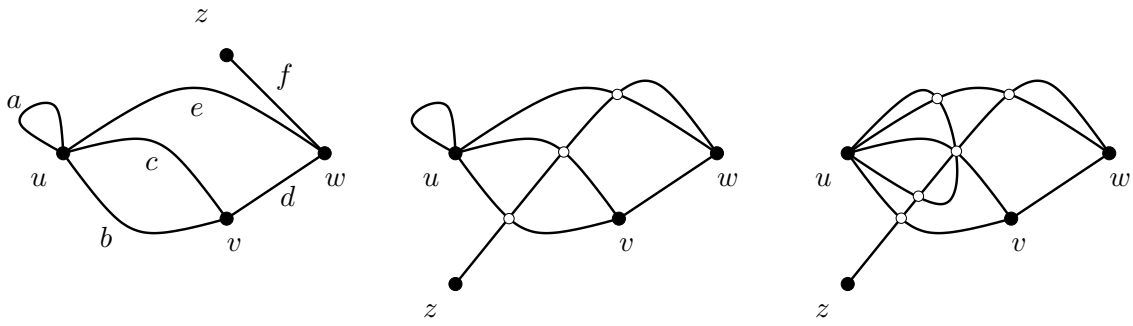


FIGURE 1 – Trois dessins du graphe de sommets u, v, w, z, d'arêtes a, b, c, d, e, f et de fonction d'attachement $\varphi(a) = \{u\}$, $\varphi(b) = \{u, v\}$, $\varphi(c) = \{u, v\}$, $\varphi(d) = \{v, w\}$, $\varphi(e) = \{w, u\}$, $\varphi(f) = \{w, z\}$

Deux graphes sont **isomorphes** si ils sont égaux à renommage près des sommets et des arêtes, i.e., s'il existe une correspondance bijective entre leurs ensembles de sommets et leurs ensembles d'arêtes qui commute avec leurs fonctions d'attachements ou, de manière équivalente, qui conserve les incidences. En d'autres termes, les graphes $G = (V, E, \varphi)$ et $G' = (V', E', \varphi')$ sont isomorphes si il existe des bijections $f : V \to V'$ et $g : E \to E'$ telles que pour tout $(v, e) \in V \times E$ la paire (v, e) est une incidence de G si et seulement si la paire $(f(v), g(e))$ est une incidence de G'. Une telle correspondance bijective entre les ensembles de sommets et d'arêtes est appelé un **isomorphisme** entre les deux graphes ; un **automorphisme** si les deux graphes sont les mêmes.

Exercice 4.2. Montrer que deux graphes $G = (V, E, \varphi)$ et $G' = (V', E', \varphi')$ sont isomorphes si et seulement si il existe une bijection $f : V \to V'$ telle que pour toute paire de sommets (u, v) de G le nombre d'arêtes de G d'extrémités u et v est égal au nombre d'arêtes de G' d'extrémités $f(u)$ et $f(v)$. 4.2

Un **graphe linéaire d'ordre** $n \geq 1$ est un graphe isomorphe au graphe suivant

$$\begin{cases} V & = & \{v_1, v_2, \ldots, v_n\} \\ E & = & \{e_1, e_2, \ldots, e_{n-1}\} \\ \varphi(e_i) & = & \{v_i, v_{i+1}\} \end{cases} \quad (1)$$

Les sommets v_1 et v_n sont ses **extrémités** et le graphe est dit **joindre** v_1 à v_n (ou v_n à v_1).

Un **graphe cyclique d'ordre** $n \geq 1$ est un graphe isomorphe au graphe suivant

$$\begin{cases} V & = & \{v_1, v_2, \ldots, v_n\} \\ E & = & \{e_1, e_2, \ldots, e_n\} \\ \varphi(e_i) & = & \{v_i, v_{i+1}\} \end{cases} \quad (2)$$

avec la convention $v_{n+1} = v_1$.

Exercice 4.3. Quel est le nombre de graphes linéaires/cycliques sur un ensemble de taille $n \geq 1$ à renommage près des arêtes ? 4.3

Exercice 4.4. Quel est le groupe des automorphismes d'un graphe linéaire d'ordre n ? Même question avec un graphe cyclique d'ordre n. 4.4

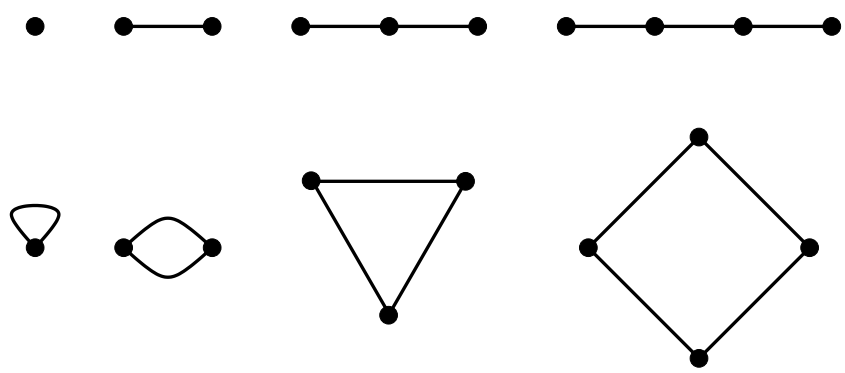


FIGURE 2 – Graphes linéaires et cycliques d'ordres $1, 2, 3$ et 4

Un **sous-graphe** d'un graphe $G = (V, E, \varphi)$ est un graphe (V', E', φ') tel que $V' \subseteq V$, $E' \subseteq E$ et φ' est la restriction de φ au domaine E' et au co-domaine des parties à 1 ou 2 éléments de V'.

Un **chemin/cycle** dans un graphe est un sous-graphe linéaire/cyclique de ce graphe.

Un graphe est **connexe** si il existe un chemin dans le graphe joignant les éléments de toute paire de sommets du graphe. Les **composantes connexes** d'un graphe sont ses sous-graphes connexes maximaux.

Exercice 4.5. Soit G un graphe d'ordre n ayant k arêtes et c composantes connexes. Montrer que $c + k \geq n$. 4.5

Soit V un ensemble fini non vide de cardinalité n. Le **graphe complet** sur V est le graphe simple sans boucle sur V dont les arêtes sont les paires non ordonnées d'éléments de V. Sa classe d'isomorphisme est notée K_n.

Soit A et B deux ensembles finis non-vides, d'intersection vide et de cardinalité n et m, respectivement. Le **graphe biparti** complet de bipartition A, B et le graphe simple sans boucles sur $A \cup B$ d'arêtes les paires non ordonnés formées d'un élément de A et d'un élément de B. Sa classe d'isomorphisme est notée $K_{n,m}$.

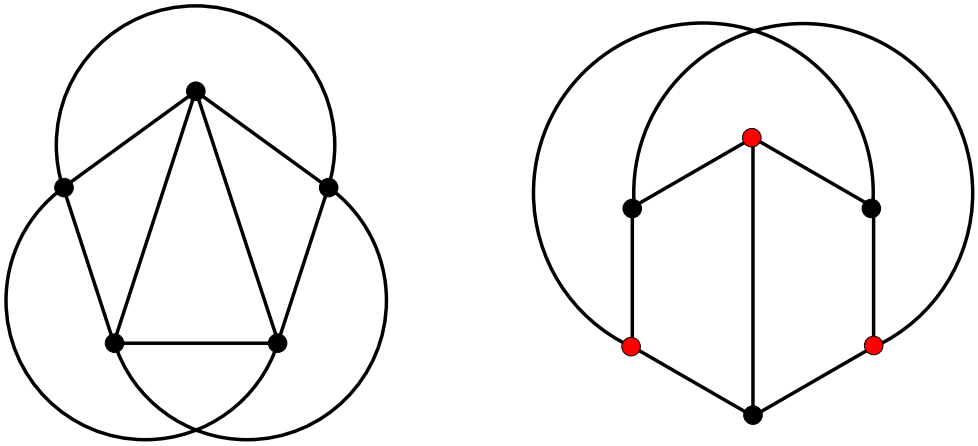


FIGURE 3 – Dessins de K_5 et $K_{3,3}$

Forêts et arbres. Une **forêt** est un graphe sans cycles. Un **arbre** est une forêt connexe.

Proposition 4.2. *Soit $G = (V, E)$ un graphe simple sans boucles connexe d'ordre $n \geq 1$. Alors les assertions suivantes sont équivalentes*

(**1-**) *G est un arbre ;*
(**2-**) *pour tout $x \in V$ et tout $y \in V$ il existe un unique chemin dans G joignant x à y ;*
(**3-**) *pour tout $e \in \binom{V}{2}$ le graphe $G + e$ admet exactement un cycle.*
(**4-**) *pour tout $e \in E$ le graphe $G - e$ a deux composantes connexes ;*
(**5-**) *G a $n - 1$ arêtes ;*

Démonstration. Il y a plusieurs façons de s'y prendre. Par exemple :

(**1-**) $\Longrightarrow$ (**2-**) : on démontre la contraposée en observant que de deux chemins distincts joignant deux sommets d'un graphe on peut extraire un cycle de ce graphe.

(**2-**) $\Longrightarrow$ (**3-**) : Existence : soit P l'unique chemin joignant les extémités x et y de e. Alors $P + e$ est un cycle de $G + e$. Unicité : soit C un cycle de $G + e$. Alors $C - e$ est un chemin joignant les extrémités x et y de e. Par suite $C - e = P$ et $C = P + e$.

(**3-**) $\Longrightarrow$ (**4-**) : on démontre la contraposée : si $G - e$ est connexe alors $P + e$ et $e + e$ sont deux cycles du graphe $G + e$ où P est un chemin joignant les extémités de e dans $G - e$.

(**4-**) $\Longrightarrow$ (**5-**) : par récurrence forte sur n : supprimer une arête et observer que les deux composantes connexes obtenues vérifient (**4-**).

(**5-**) $\Longrightarrow$ (**1-**) : par récurrence sur n en montrant (par exemple à partir de la relation "la somme des degrés des sommets égale deux fois le nombre d'arêtes") qui si G est d'ordre au moins 2 alors G admet au moins une feuille (et même deux feuilles). □

Exercice 4.6. Montrer qu'un arbre d'ordre ≥ 2 possède au moins deux feuilles et exactement deux feuilles si et seulement si l'arbre est un graphe linéaire. 4.6

Forêts et arbres couvrants. Une **forêt couvrante** d'un graphe G est un forêt de G dont l'ensemble des sommets est exactement l'ensemble des sommets de G. En particulier, le graphe discret de sommets les sommets de G est une forêt couvrante de G.

Un **arbre couvrant** d'un graphe connexe G est un arbre de G dont l'ensemble des sommets est exactement l'ensemble des sommets de G, cf. Fig. 4.

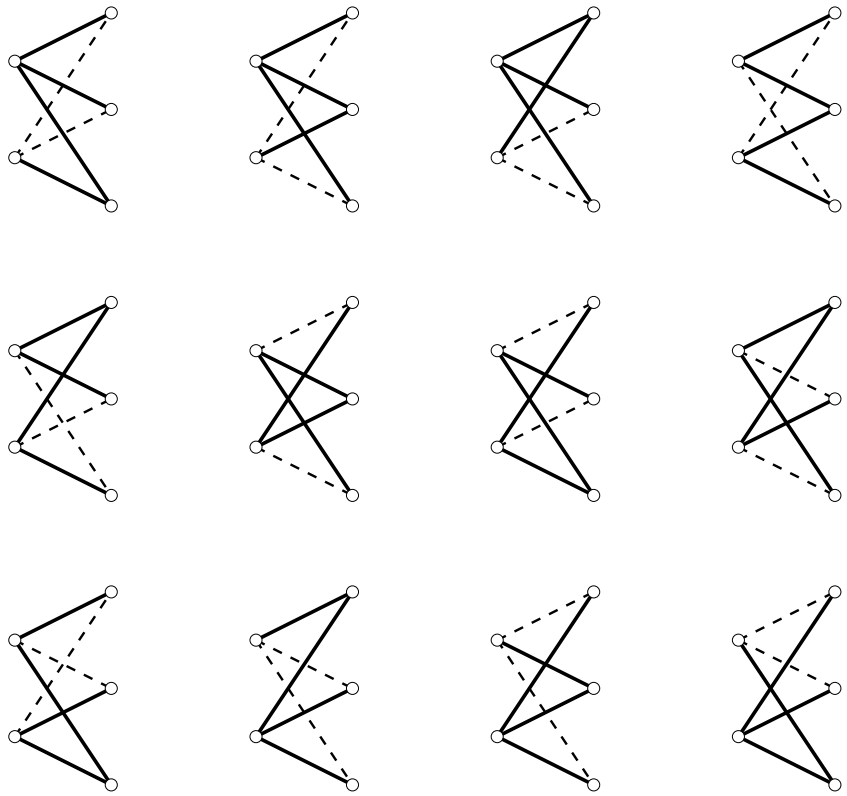


FIGURE 4 – Les arbres couvrants du graphe biparti complet $K_{2,3}$.

Exercice 4.7. Montrer que tout graphe connexe admet un arbre couvrant. 4.7

Exercice 4.8. Soit G un graphe biparti complet de bipartition $A \sqcup B$ où A est de cardinalité 2 et B de cardinalité n. Etablir une bijection entre l'ensemble des arbres couvrants de G et l'ensemble des parties de B dont un élément est distingué. Que peut-on en déduire sur le nombre d'arbres couvrants de $K_{2,n}$? 4.8

Exercice 4.9. Quel est le nombre d'arbres couvrants d'un graphe cyclique d'ordre n ? Un multi-arbre est un graphe connexe sans cycles d'ordre ≥ 3. Quel est le nombre d'arbres couvrants d'un multi-arbre ? 4.9

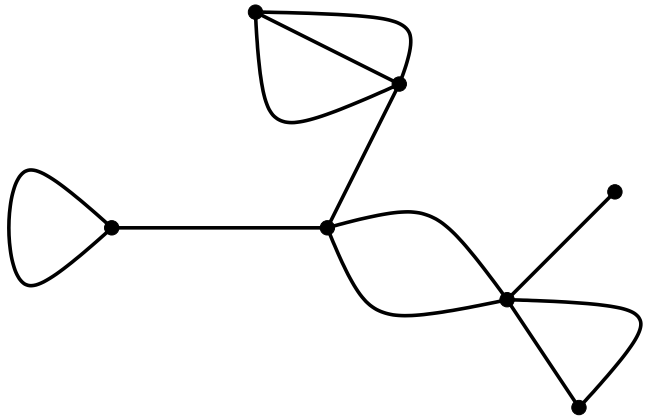
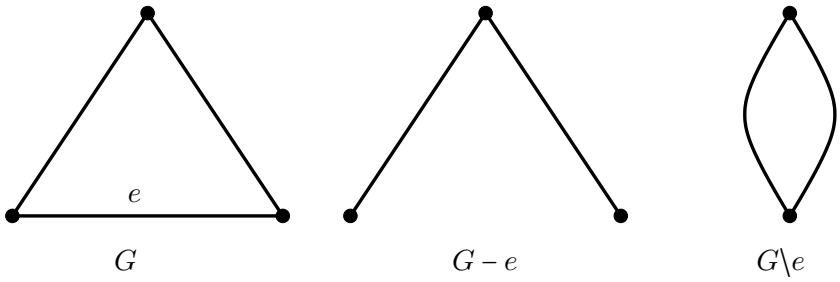


FIGURE 5 – un multi-arbre d'ordre 7

Exercice 4.10. Soit G un graphe connexe sans boucles et soit e une arête de G. Montrer que

$$T(G) = T(G-e) + T(G\backslash e).$$

Ici $G-e$ est le graphe obtenu à partir de G en supprimant l'arête e et $G\backslash e$ est le graphe obtenu à partir de G en identifiant les extrémités de e et en supprimant la ou les boucles ainsi créées. Utiliser ce résultat pour calculer $T(K_{2,3})$. 4.10



Arbres couvrants optimaux Soit $G = (V, E)$ un graphe simple sans boucles connexe d'ordre n muni d'une fonction poids $w : E \to \mathbb{R}$ sur l'ensemble de ses arêtes ; fonction que nous étendons à l'ensemble des parties X de E en posant

$$w(X) = \sum_{e \in X} w(e).$$

Nous examinons le problème de calculer un arbre couvrant dont l'ensemble des arêtes est de poids minimal, **un arbre couvrant (de poids) minimal** en abrégé.

Posons

$$\begin{aligned}
\mathcal{J} &= \{X \subset E \mid \ (V, X) \text{ est une forêt}\} \\
\mathcal{B} &= \{X \in \mathcal{J} \mid \ X \text{ est maximal pour la relation d'inclusion}\} \\
&= \{X \subset E \mid \ (V, X) \text{ est un arbre}\} \\
\mathcal{D} &= \{X \subset E \mid \ (V, X) \text{ n'est pas une forêt}\} \\
&= \mathcal{P}(E) - \mathcal{J} \\
\mathcal{C} &= \{X \in \mathcal{D} \mid \ X \text{ est minimal pour la relation d'inclusion}\} \\
&= \{X \subset E \mid \ (\bigcup X, X) \text{ est un cycle}\}
\end{aligned}$$

Un **circuit** est un élément de $\mathcal{C}$.

Lemme 4.3. *Soit $X \in \mathcal{J}$ et soit $x \in E - X$ tel que $X + x \in \mathcal{D}$. Alors il existe un unique circuit inclus dans $X + x$.*

Démonstration. cf. Proposition 4.2 assertion (**3-**). □

Ce circuit est appelé le **circuit fondamental** de la paire (X, x).

Lemme 4.4. *L'ensemble $\mathcal{J}$ vérifie les trois propriétés suivantes*

(**1-**) $\varnothing \in \mathcal{J}$;
(**2-**) $(\forall X \in \mathcal{J})\,(Y \subset X \implies Y \in \mathcal{J})$;
(**3-**) $(\forall X \in \mathcal{J})(\forall Y \in \mathcal{J})\,(|X| < |Y| \implies$ *il existe* $y \in Y - X$ *tel que* $X + y \in \mathcal{J})$.

Démonstration. Les deux premières assertions sont immédiates. Pour démontrer la troisième assertion nous démontrons sa contraposée

$$(\forall y \in Y - X)(X + y \in \mathcal{D}) \implies |X| \geq |Y|$$

par induction sur le cardinal de $Y - X$

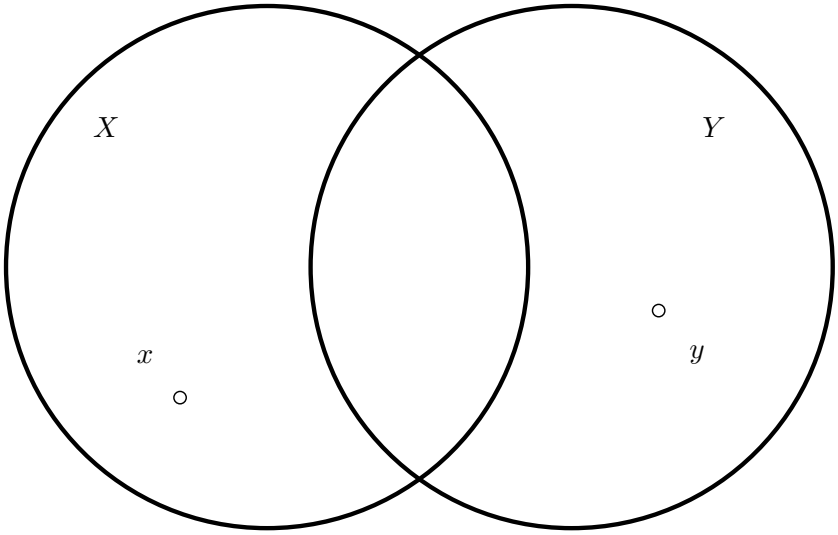


FIGURE 6 –

Soit k le cardinal de $Y - X$.

(**1-**) $k = 0$. Alors $Y \subseteq X$ et, par suite, $|X| \geq |Y|$;
(**2-**) $k \geq 1$. Soit $y \in Y - X$ et soit $x \in X - Y$ appartenant au circuit fondamental C de la paire (X, y) (✍). Posons $Y' = Y$ et $X' = X + y - x$. Alors
  (a) $Y' \in \mathcal{J}$;
  (b) $X' \in \mathcal{J}$;
  (c) $Y' - X' = (Y - X) - y$ et $|Y' - X'| = |Y - X| - 1$;
  (d) pour tout $y' \in Y' - X'$ l'ensemble $X' + y' \in \mathcal{D}$ car du circuit fondamental de la paire (X, y) [qui contient x] et du circuit fondamental de la paire (X, y') [qui contient ou ne contient pas x] on extrait un circuit de $X' + y' = X + y + y' - x$ (✍) ;
(**3-**) par hypothèse de récurrence $|X'| \geq |Y'|$ et par suite $|X| \geq |Y|$.

□

Un **arbre couvrant glouton** de G est une arbre couvrant de G dont une énumération des arêtes $b_1, b_2, \ldots, b_{n-1}$ est défini inductivement par

$$b_i \in \operatorname{argmin} \{w(e) : e \notin \{b_1, b_2, \ldots, b_{i-1}\}, \{b_1, b_2, \ldots, b_{i-1}, e\} \in \mathcal{J}\}.$$

Une telle énumération est dite **gloutonne**.

Théorème 4.5. *Soit $b_1, b_2, \ldots, b_{n-1}$ une énumération gloutonne des arêtes d'un arbre couvrant glouton de G et soit $e_1, e_2, \ldots, e_{n-1}$ une énumération des arêtes d'un arbre couvrant de G suivant les valeurs croissantes de leurs poids, i.e.,*

$$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}).$$

Alors $w(b_i) \leq w(e_i)$ pour tout indice i. En particulier un arbre couvrant glouton est un arbre de poids minimal et réciproquement.

Démonstration. Soit $i \in \{1, \ldots, n-1\}$, $X = \{e_1, e_2, \ldots, e_i\}$ et $Y = \{b_1, b_2, \ldots, b_{i-1}\}$. Alors

(**1-**) $X, Y \in \mathcal{J}$;
(**2-**) $|X| > |Y|$.

Il existe donc $e_j \in X$ tel que $e_j \notin Y$ et $Y + e_j \in \mathcal{J}$. Par construction $w(b_i) \leq w(e_j)$ et par hypothèse $w(e_j) \leq w(e_i)$. Par suite $w(b_i) \leq w(e_i)$. □

Exercice 4.11. Soit G un graphe connexe muni d'une fonction poids $w : E \to \mathbb{R}$ injective sur l'ensemble E de ses arêtes. Montrer que G admet un unique arbre couvrant de valuation minimale. 4.11

L'algorithme de Kruskal. L'algorithme de Kruskal de calcul d'un arbre couvrant minimal soumet chaque arête, dans l'ordre croissant de leurs poids, à la question de son insertion dans une forêt couvrante initialement discrète : en cas de réponse positive, i.e. si les extrémités de l'arête n'appartiennent pas à la même composante connexe de la forêt, l'algorithme augmente la forêt de cette arête ; en cas de réponse négative l'algorithme écarte définitivement cette arête de toute considération ultérieure. Il découle directement des résultats du paragraphe précédent que l'algorithme de Kruskal produit un arbre minimal puisque les arêtes sont examinées selon l'ordre croissant de leurs poids. Une implémentation de l'algorithme de Kruskal utilisant la structure de données union-find est alors la suivante :

```
1. set function minspanningtree (set : vertices, edges) ;
2.     set blue ;
3.     blue : = ∅ ;
4.     edges : = trier edges par poids croissants ;                  O(m lg m)
5.     for v ∈ vertices do makeset(v) ;                                 O(n)
6.     for vw ∈ edges do                                             O(mα(n))
7.       if find(v) ≠ find(w) then
8.       begin
9.         link(find(v), find(w)) ;
10.        blue := blue + {vw} ;
11.      end
12.    return blue
13. end minspanningtree ;
```

Il découle de notre analyse de la structure de données union-find (chapitre 2) que la complexité de cette implémentation est un $O(m\alpha(n) + T)$ où $T = O(m \log m)$ est la complexité du tri des arêtes.

L'algorithme de Kruskal est une première déclinaison d'un méthode plus générale de calcul d'un arbre couvrant minimal, méthode basée sur la simple observation que tout ensemble d'arêtes dont la suppression augmente le nombre de composantes connexes du graphe d'une unité contient nécessairement une arête de tout arbre couvrant. Cette méthode plus générale, appelée **méthode gloutonne**, est l'objet du paragraphe qui suit.

La méthode gloutonne. Un **co-circuit** de G est un ensemble d'arête dont la suppression augmente le nombre de composantes connexes de G d'une unité. La **méthode gloutonne** consiste à colorier les arêtes du graphe par application répétée des deux règles de coloration suivantes :

(**1-**) Colorier en bleu une arête non coloriée de poids minimal d'un co-circuit sans arête bleue, règle bleu en abrégé ;
(**2-**) Colorier en rouge une arête non coloriée de poids maximal d'un circuit sans arête rouge, règle rouge en abrégé.

Théorème 4.6. *La méthode gloutonne colorie toutes les arêtes et maintient la propriété suivante : l'ensemble des arêtes bleues est complétable en un arbre couvrant minimal* □

Démonstration. Voir Tarjan [2, Chapitre 6] □

Exercice 4.12. Montrer que l'algorithme de Kruskal est une déclinaison de la méthode gloutonne. 4.12

Une deuxième déclinaison de la méthode gloutonne : l'algorithme de Prim. L'algorithme de Prim consiste à augmenter un arbre, initialement réduit à un nœud quelconque du graphe, d'un arête de poids minimal dans l'ensemble des arêtes ayant exactement une extrémité dans l'arbre et ce jusqu'à l'obtention d'un arbre couvrant. À cette fin l'algorithme maintient l'arbre sous la forme d'une arborescence

$$p : X \to X$$

sur l'ensemble X de ses nœuds augmentée d'une application $q : Y \to X$ à valeurs dans X définie sur l'ensemble Y des nœuds de $V - X$ adjacents aux nœuds de X par la condition : $uq(u)$ est une arête de poids minimal dans l'ensemble des arêtes de la coupe $X, V - X$ dont u est une extrémité. L'ensemble Y est maintenu sous la forme d'un **tas de Fibonacci** (voir paragraphe suivant) de fonction poids la fonction qui à $u \in Y$ associe le poids de l'arête $uq(u)$.

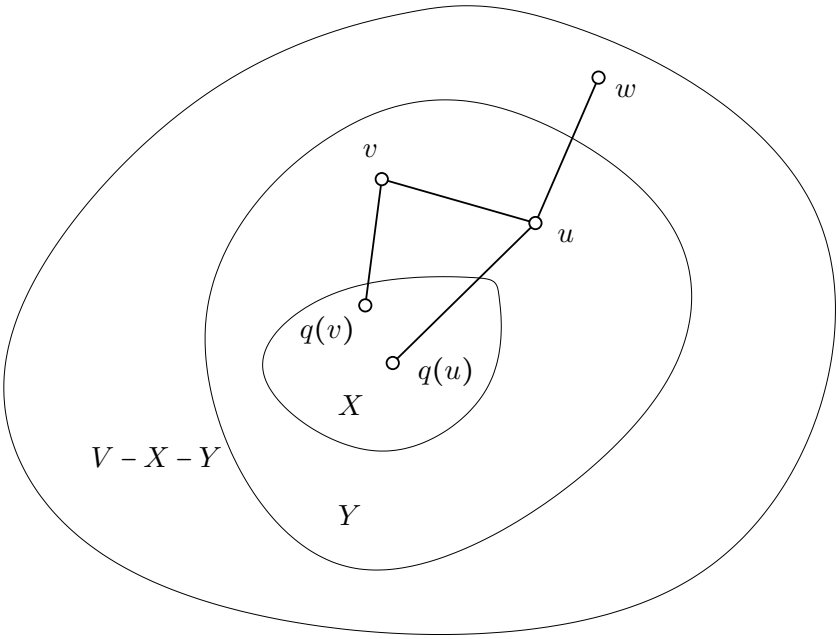


FIGURE 7 –

L'augmentation de l'arbre d'une arête consiste à

- (**1-**) extraire du tas un élément minimal u ;
- (**2-**) augmenter l'arborescence p de la paire $(u, q(u))$;
- (**3-**) augmenter le tas des éléments w de $V - X - Y$ adjacents à u en posant $q(w) = u$ et mettre à jour l'application q en subsituant à la paire $(v, q(v))$ la paire (v, u) pour tout v de l'ensemble des éléments de Y adjacents à u tel que le poids de l'arête vu est strictement inférieur au poids de l'arête $vq(v)$.

La complexité de cette implémentation de l'algorithme de Prim est un $O(m + n \log n)$ (✍).

Exercice 4.13. Dérouler l'algorithme de Prim sur le graphe suivant en choisissant pour nœud initial le nœud d. Quel est sur cet exemple le nombre d'appels à l'opération **decrement** ? 4.13

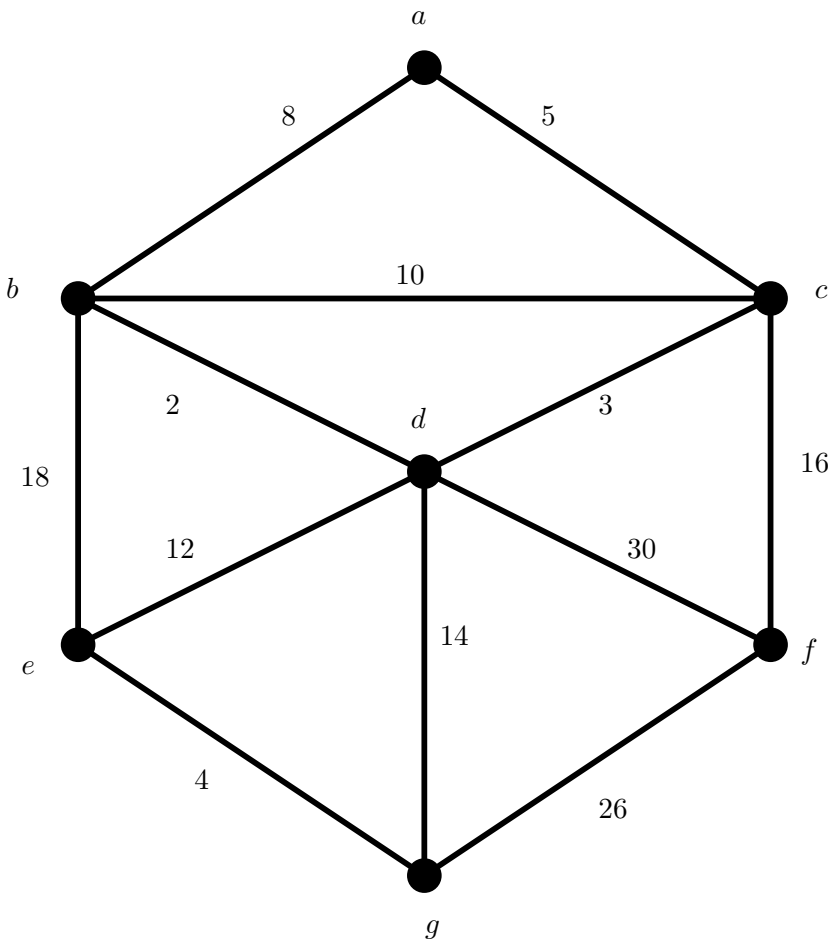


FIGURE 8 –

Les tas de Fibonacci. Un **tas** est une structure de données dynamique définie sur un ensemble d'éléments E muni d'une fonction poids $w : E \to \mathbb{R}$ qui permet principalement d'en extraire un élément de poids minimal. Les tas de Fibonacci sont des extensions des tas binomiaux (que vous avez vu en TD). Extensions qui supportent outre les opérations définies sur les tas binomiaux, à savoir

makeheap(e)	retourne un nouveau tas contenant uniquement l'élément e
findmin(h)	retourne un pointeur vers un élément de h de poids minimal
insert(e, h)	insertion d'un nouvel élément e dans h
deletemin(h)	suppression d'un élément de h de poids minimal
meld(h, h')	addition ensembliste de deux tas disjoints

les deux opérations suivantes

decrement(h, e, Δ)	diminution du poids de l'élément e de Δ
delete(h, e)	suppression del'élément e du tas h.

avec les complexités **amorties sur une séquence d'opérations** suivantes

makeheap	$O(1)$
findmin	$O(1)$
insert	$O(1)$
deletemin	$O(\lg n)$
meld	$O(1)$
	$O(\lg n)$
decrement	$O(1)$
delete	$O(\lg n)$.

où n est le nombre d'éléments du tas.

Références

[1] D. C. Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, 1992.

[2] R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

5 Calculabilité

Fonctions de bases, composition ou substitution, récursion primitive, minimisation, fonctions primitives récursives, fonctions récursives, fonction d'Ackermann, fonctions calculables par machines à registres

---

Computability of Recursive Functions
by J.C. SHEPHERDSON AND H. E. STURGIS
Journal of the ACM (JACM) Volume 10 Issue 2, April 1963 Pages 217-255

1. Introduction. As a result of the work of Turing, Post, Kleene and Church [1, 2, 3, 9, 10, 11, 12, 17, 18] it is now widely accepted that the concept of "computable" as applied to a function of natural numbers is correctly identified with the concept of "partial recursive." One half of this equivalence, that all functions computable by any finite, discrete, deterministic device supplied with unlimited storage are partial recursive, is relatively straightforward once the elements of recursive function theory have been established. All that is necessary is to number the configurations of machine-plus-storage medium, show that the changes of configuration number caused by each "move" are given by partial recursive functions, and then use closure properties of the class of partial recursive functions to deduce that the function computed by the complete sequence of moves is partial recursive. Until recently all proofs [4, 6, 12, 13, 19, 20] of the converse half of the equivalence, namely, that all partial recursive functions are computable, have consisted of proofs that all partial recursive functions can be computed by Turing machines, which are certainly machines in the above sense. Althoughi not difficult, these proofs are complicated and tedious to follow for two reasons : (1) A Turing machine has only one head so that one is obliged to break down the computation into very small steps of operations on a single digit. (2) It has only one tape so that one has to go to some trouble to find the number one wishes to work on and keep it separate from other numbers. The object of this paper is first to obtain, by relaxing these restrictions, a form of idealized computer which is sufficiently flexible for one to be able to convert an intuitive computational procedure with littte change into a program for such a machine. ...

2. Unlimited Register Machine (URM). .....

Fonctions récursives. Soit $\mathcal{F} = \bigcup_{n\geq 1} \mathbb{N}^{\mathbb{N}^n}$ la classe des fonctions (partielles) $f : \mathbb{N}^n \to \mathbb{N}$, $n \in \mathbb{N}^*$.

La classe des fonctions **primitives récursives** est la plus petite sous-classe de $\mathcal{F}$ contenant la classe des **fonctions de base**, close par **composition** (ou **substitution**) et **récursion primitive**.

La classe des fonctions **récursives** est la plus petite sous-classe de $\mathcal{F}$ contenant la classe des fonctions de base, close par composition, récursion primitive et **minimisation**.

Explicitons les termes : fonctions de base, composition, récursion primitive et minimisation.

Fonctions de base : La classe des **fonctions de base** est composée de la fonction nulle $\mathrm{Z} : \mathbb{N} \to \mathbb{N}$, $\mathrm{Z}(x) = 0$, de la fonction successeur $\mathrm{S} : \mathbb{N} \to \mathbb{N}$, $\mathrm{S}(x) = x + 1$, et des projections canoniques $\mathrm{P}_i^n : \mathbb{N}^n \to \mathbb{N}$, $\mathrm{P}_i^n(x) = \mathrm{P}_i^n(x_1, x_2, \ldots, x_n) = x_i$.

Composition ou substitution : Soient f une fonction d'arité k et $g = (g_1, g_2, \ldots, g_k)$ une suite de k fonctions d'arité n. La fonction h d'arité n définie par

$$h(x) = f(g_1(x), g_2(x), \ldots, g_k(x))$$

est dite obtenue par composition (ou substitution) des fonctions f et g. On pourra la noter $f \circ g$ ou $f \circ (g_1, g_2, \ldots, g_k)$.

Exemple : $h(x_1, x_2, x_3) = x_1 + x_2 + x_3 = A(A(x_1, x_2), x_3)$ où $A(x_1, x_2) = x_1 + x_2$ est la composée de la fonction A et de la suite de fonctions $(x_1, x_2, x_3) \mapsto A(x_1, x_2)$, $(x_1, x_2, x_3) \mapsto x_3$.

Récursion primitive : Soient f et g deux fonctions d'arités respectives n et $n + 2$. Alors la fonction h d'arité $n + 1$ définie pour $(x, y) \in \mathbb{N}^n \times \mathbb{N} (= \mathbb{N}^{n+1})$ par

$$\begin{cases} h(x, 0) & = & f(x) \\ h(x, y + 1) & = & g(x, y, h(x, y)) \end{cases}$$

est dite obtenue par récursion primitive à partir des fonctions f et g.

Exemple 1 : l'addition : $x + 0 = x$, $x + (y + 1) = (x + y) + 1$ est définie par récursion primitive à partir des fonctions $f(x) = x$ et $g(x, y, z) = z + 1$.

Exemple 2 : la factorielle : $0! = 1$, $(y + 1)! = y!(y + 1)$ est définie par récursion primitive à partir de la fonction constante $f(x) = 1$ et la fonction $g(x, y, z) = z(y + 1)$.

Théorème 5.1. *Les fonctions suivantes sont primitives récursives :*

(**1-**) $x+y, xy, x^y$;

(**2-**) $x \dot{-} 1 = \begin{cases} 0 & si\ x = 0; \\ x-1 & sinon. \end{cases}$; $x \dot{-} y = \begin{cases} 0 & si\ x \leq y, \\ x-y & sinon. \end{cases}$;

(**3-**) $\mathrm{sg}(x) = \begin{cases} 0 & si\ x = 0; \\ 1 & sinon. \end{cases}$

(**4-**) $\overline{\mathrm{sg}}(x) = \begin{cases} 1 & si\ x = 0; \\ 0 & sinon. \end{cases}$

(**5-**) $|x-y|$;

(**6-**) $x!$;

(**7-**) $\min\{x,y\}, \max\{x,y\}$;

(**8-**) $\mathrm{r}(x,y)$ *le reste de la division euclidienne de* y *par* x *avec la convention* $\mathrm{r}(0,y) = y$

(**9-**) $\mathrm{q}(x,y)$ *le quotient de la division euclidienne de* y *par* x *avec la convention* $\mathrm{q}(0,y) = 0$;

(**10-**) $\mathrm{div}(x,y) = 1$ *si* x *divise* y ; 0 *sinon (avec la convention* 0 *divise* 0 *mais ne divise pas* $y \neq 0$*).*

Démonstration. La fonction obtenue par récursion primitive à partir des fonctions f et g est notée h.

$x+0 = x,\ x+(y+1) = (x+y)+1$ $\qquad$ $f(x) = x,\ g(x,y,z) = z+1, h(x,y) = x+y$

L'addition $\mathrm{A}(x,y) = x+y$ est primitive récursive car A est obtenu par récursion primitive à partir des fonctions $f(x) = x$ et $g(x,y,z) = z+1$, fonctions qui sont elles-mêmes récursives primitives car f est une fonction de base et g est obtenue par composition de fonctions de bases :

$$\begin{aligned} f(x) &= \mathrm{P}_1^1(x) \\ g(x,y,z) &= \mathrm{S}(z) \\ &= \mathrm{S} \circ \mathrm{P}_3^3(x,y,z). \end{aligned}$$

$x0 = 0, x(y+1) = xy + x$ $\qquad$ $f(x) = 0, g(x,y,z) = z+x, h(x,y) = xy$

La multiplication $M(x,y) = xy$ est primitive récursive car M est obtenu par récursion primitive à partir des fonctions $f(x) = 0$ et $g(x,y,z) = z+x$, fonctions qui sont elles-mêmes récursives primitives car f est une fonction de base et g est obtenue par composition de l'addition et de fonctions de bases

$$\begin{aligned} f(x) &= \mathrm{Z}(x) \\ g(x,y,z) &= \mathrm{A}(z,x) \\ &= \mathrm{A}(\mathrm{P}_3^3(x,y,z), \mathrm{P}_1^3(x,y,z)) \\ &= \mathrm{A} \circ (\mathrm{P}_3^3, \mathrm{P}_1^3)(x,y,z). \end{aligned}$$

$x^0 = 1, x^{y+1} = x^y x$ $\qquad$ $f(x) = 1, g(x,y,z) = zx, h(x,y) = x^y$

L'exponentiation $E(x,y) = x^y$ est primitive récursive : par récursion primitive des fonctions

$$\begin{aligned} f(x) &= \mathrm{S} \circ \mathrm{Z}(x) \\ g(x,y,z) &= M(z,x) \\ &= M(\mathrm{P}_3^3(x,y,z), \mathrm{P}_1^3(x,y,z)) \\ &= M \circ (\mathrm{P}_3^3, \mathrm{P}_1^3)(x,y,z). \end{aligned}$$

$0 \dot{-} 1 = 0, (x+1) \dot{-} 1 = x$ $\qquad$ $f(x) = 0, g(x,y,z) = y, x \dot{-} 1 = h(x,x)$

La fonction prédécesseur $\mathrm{P}(x) = x \dot{-} 1$ est primitice récursive : par récursion primitive des fonctions

$$\begin{aligned} f(x) &= \mathrm{Z}(x) \\ g(x,y,z) &= \mathrm{P}_2^3(x,y,z) \\ \mathrm{P}(x) &= h(x,x) \\ &= h(\mathrm{P}_1^1(x), \mathrm{P}_1^1(x)) \\ &= h \circ (\mathrm{P}_1^1, \mathrm{P}_1^1)(x) \end{aligned}$$

$x \dot{-} 0 = x, x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1$ $\qquad$ $f(x) = x, g(x,y,z) = z \dot{-} 1, h(x,y) = x \dot{-} y$

La différence tronquée $\mathrm{DT}(x,y) = x \dot{-} y$ est primitive récursive

$$\begin{aligned} f(x) &= \mathrm{P}_1^1(x) \\ g(x,y,z) &= \mathrm{P}(z) \\ &= \mathrm{P} \circ \mathrm{P}_3^3(x,y,z) \end{aligned}$$

$\mathrm{sg}(0) = 0, \mathrm{sg}(x+1) = 1$ $\qquad$ $f(x) = 0, g(x,y,z) = 1, \mathrm{sg}(x) = h(x,x)$

$$\begin{aligned} f(x) &= \mathrm{Z}(x) \\ g(x,y,z) &= \mathrm{S} \circ \mathrm{Z}(x) \\ &= \mathrm{S} \circ \mathrm{Z} \circ \mathrm{P}_1^3(x,y,z) \\ \mathrm{sg}(x) &= h \circ (\mathrm{P}_1^1, \mathrm{P}_1^1)(x) \end{aligned}$$

$\overline{\mathrm{sg}}(x) = 1 \dot{-} \mathrm{sg}(x) = \mathrm{S} \circ \mathrm{Z}(x) \dot{-} \mathrm{sg}(x) = \mathrm{DT} \circ (\mathrm{S} \circ \mathrm{Z}, \mathrm{sg})(x)$

$|x-y| = (x \dot{-} y) + (y \dot{-} x)$

$$\begin{aligned} |x-y| &= \mathrm{A}(\mathrm{DT}(x,y), \mathrm{DT}(y,x)) \\ &= \mathrm{A}(\mathrm{DT}(x,y), \mathrm{DT}(\mathrm{P}_2^2(x,y), \mathrm{P}_1^2(x,y))) \end{aligned}$$

$\min\{x,y\} = x \dot{-} (x \dot{-} y)$

$$\begin{aligned} \min\{x,y\} &= \mathrm{DT}(x, \mathrm{DT}(x,y)) \\ &= \mathrm{DT}(\mathrm{P}_1^2(x,y), \mathrm{DT}(x,y)) \end{aligned}$$

$\max\{x,y\} = (x \dot{-} y) + y$

$$\begin{aligned} \max\{x,y\} &= \mathrm{A}(\mathrm{DT}(x,y), y) \\ &= \mathrm{A}(\mathrm{DT}(x,y), \mathrm{P}_2^2(x,y)) \end{aligned}$$

$\mathrm{r}(x,y)$: reste de la division euclidienne de y par x avec la convention $\mathrm{r}(0,y) = y$

$$\mathrm{r}(x, y+1) = \begin{cases} \mathrm{r}(x,y)+1 & \text{si } \mathrm{r}(x,y)+1 \neq x; \\ 0 & \text{sinon.} \end{cases}$$

d'où $\mathrm{r}(x,0) = 0, \mathrm{r}(x,y+1) = (\mathrm{r}(x,y)+1)\,\mathrm{sg}(|x-(\mathrm{r}(x,y)+1)|)$ et par suite il suffit de prendre $f(x) = 0, g(x,y,z) = (z+1)\,\mathrm{sg}(|x-(z+1)|)$ pour obtenir par récursion primitive

$$h(x,y) = \mathrm{r}(x,y).$$

$\mathrm{q}(x,y)$: quotient de la division euclidienne de y par x avec la convention $\mathrm{q}(0,y) = 0$

✍ —

$\mathrm{div}(x,y) = \overline{\mathrm{sg}}(\mathrm{r}(x,y))$

□

Exercice 5.1. Montrer que la fonction $\mathrm{q}(x,y)$: quotient de la division euclidienne de y par x avec la convention $\mathrm{q}(0,y) = 0$ est primitive récursive. 5.1

Une prédicat $M(x)$ est dit primitif récursif si sa fonction caractéristique $c_M(x)$ (définie par $C_M(x) = 1$ si $M(x)$ est vrai ; 0 sinon) est primitive récursive.

Corollaire 5.2. *Soit $f(x), g(x)$ deux fonctions primitives récursives et soit $M(x)$ un prédicat primitif récursif. Alors la fonction $h(x)$ définie par*

$$h(x) = \begin{cases} f(x) & \text{si } M(x) \text{ est vrai;} \\ g(x) & \text{sinon;} \end{cases}$$

est primitive récursive.

Démonstration. $g(x) = c_M(x)f(x) + (1 \dot{-} C_M)(x)g(x)$. □

Corollaire 5.3. *Soient $M(x)$ et $N(x)$ deux prédicats primitifs récursifs. Alors il en est de même des prédicats '$\neg M(x)$', '$M(x)$ et $N(x)$' et '$M(x)$ ou (non exclusif) $N(x)$'.*

Démonstration. Leurs fonctions caractéristiques sont, respectivement, $1 \dot{-} C_M(x)$, $C_M(x)C_N(x)$, et $\max\{C_M(x), C_N(x)\}$. □

Exercice 5.2. Montrer que si $M(x)$ et $N(x)$ sont deux prédicats primitifs récursifs alors il en est de même du prédicat $M(x) \Longrightarrow N(x)$. 5.2

Avant de passer à la définition de la mimimisation nous introduisons trois nouveaux opérateurs sur les fonctions, la **somme bornée**, le **produit borné** et la **minimisation bornée**, tous trois basés sur la récursion primitive.

Somme et produit borné : Soit $f(x, z)$ une fonction d'arité $n + 1$ ($x \in \mathbb{N}^n, y \in \mathbb{N}$). Les fonctions $g(x, y)$ et $h(x, y)$ définies par $g(x, 0) = 0$, $g(x, y + 1) = g(x, y) + f(x, y)$ et $h(x, 0) = 1$, $h(x, y+1) = h(x, y) \times f(x, y)$ sont notées $\sum_{z<y} f(x, z)$ et $\prod_{z<y} f(x, z)$. Notons que si f est totale et primitive récursive alors il en est de même de $\sum_{z<y} f(x, z)$ et $\prod_{z<y} f(x, z)$.

Minimisation bornée : Soit $f(x, y)$ une fonction totale d'arité $n + 1$ ($x \in \mathbb{N}^n, y \in \mathbb{N}$). La fonction totale $g(x, y)$ d'arité $n + 1$ définie par "$g(x, y)$ est le plus petit $z < y$ tel que $f(x, z) = 0$ si un tel z existe, y sinon" est notée $\mu z < y(f(x, z) = 0)$ et est dite obtenue par minimisation bornée.

Théorème 5.4. *Avec les notations ci-dessus*

$$\mu z < y(f(x, z) = 0) = \sum_{v<y} \left(\prod_{u \leq v} \mathrm{sg}(f(x, u)) \right).$$

En particulier si $f(x, y)$ est primitive récursive il en est de même de $\mu z < y(f(x, z) = 0)$.

Démonstration. Posons $h(x,v) = \prod_{u\le v} \mathrm{sg}(f(x,u))$ et $z_0 = \mu z < y(f(x,z) = 0)$; on voit facilement que si $v < z_0$ alors $h(x,v) = 1$ et que si $z_0 \le v < y$ alors $h(x,v) = 0$; par suite $z_0 = \#\{v < y : h(x,v) = 1\} = \sum_{v<y} h(x,v)$. □

Corollaire 5.5. *Soit $R(x,y)$ un prédicat primitif récursif. Alors*

(1-) *la fonction $f(x,y)$, définie par 'f(x,y) est le plus petit $z < y$ tel que $R(x,z)$ soit vrai si un tel z existe ; y sinon' est primitive récursive ; sans surprise cette fonction est notée*

$$\mu\, z < y\, (R(x,y))$$

(2-) *les prédicats $M(x,y) = (\forall z < y)(R(x,z))$ et $N(x,y) = (\exists z < y)(R(x,z))$ sont primitifs récursifs.*

Démonstration.

$$\begin{aligned} f(x,y) &= \mu z < y(\overline{\mathrm{sg}}(C_R(x,z)) = 0) \\ C_M(x,y) &= \prod_{z<y} C_R(x,z) \\ N(x,y) &= \neg(\forall z < y)(\neg R(x,z))). \end{aligned}$$

□

Théorème 5.6. *Les fonctions suivantes sont primitives récursives :*

(1-) *$D(x)$ le nombre de diviseurs de x avec la convention $D(0) = 1$;*

(2-) $\mathrm{Pr}(x) = \begin{cases} 1 & \text{si } x \text{ est premier} \\ 0 & \text{sinon} \end{cases}$

(le prédicat être premier est primitif récursif) ;

(3-) *p_x le x-ième nombre premier avec la convention $p_0 = 0$*

(4-) *$(x)_y$: l'exposant de p_y dans la décomposition en facteurs premiers de x avec la convention $(x)_y = 0$ si x ou y est nul.*

Démonstration. ✍ — □

Exercice 5.3. Démontrer le théorème ci-dessus. 5.3

Exercice 5.4. Soit $\pi(x,y) = 2^x(2y+1) - 1$. Montrer que π est bijective, primitive récursive et que les fonctions π_1 et π_2 telles que $\pi(\pi_1(x), \pi_2(x)) = x$ sont primitives récursives. 5.4

Exercice 5.5. Montrer que la suite de Fibonacci est primitive récursive. 5.5

Minimisation : Soit $f(x,y)$ est fonction d'arité $n+1$. La fonction $g(x)$ d'arité n définie par $g(x)$ est le plus petit entier z tel que $f(x,z) = 0$ et $f(x,y)$ bien défini pour tout $y < z$ est dite obtenue par minimisation. Elle est notée $\mu y(f(x,y) = 0)$.

La fonction d'Ackermann. La fonction d'Ackermann $\psi(x,y)$ est définie (dans ce paragraphe) par les relations de récurrences suivantes

$$\begin{cases} \psi(0,y) & = & y+1 \\ \psi(x+1,0) & = & \psi(x,1) \\ \psi(x+1,y+1) & = & \psi(x,\psi(x+1,y)) \end{cases}$$

Théorème 5.7. *La fonction d'Ackermann est récursive.* □

La preuve nécessite un peu de travail : trois lemmes en préliminaires.

Lemme 5.8. *La fonction d'Ackermann est bien définie.*

Démonstration. Par induction sur l'ensemble des paires d'entiers muni de l'ordre lexicographique $(x,y) \leq (x',y')$ si $x < x'$ ou si $x = x'$ et $y \leq y'$.

Si $x = 0$ alors $\psi(x,y) = y+1$ est bien défini.

Si $\psi(u,v)$ est bien défini pour tout $(u,v) < (x+1,y)$ alors soit $y = 0$ et, dans ce cas, $\psi(x+1,y) = \psi(x,1)$ est bien défini car $(x,1) < (x+1,y)$, soit $y \neq 0$ et, dans ce cas, et $\psi(x+1,y) = \psi(x,\psi(x+1,y-1))$ est bien défini par $(x+1,y-1) < (x+1,y)$ et $(x,\psi(x+1,y-1)) < (x+1,y)$.

□

Un ensemble fini S de triplets d'entiers est dit **adéquat** si il satisfait les trois conditions suivantes :

(**1-**) Si $(0,y,z) \in S$ alors $z = y+1$;
(**2-**) Si $(x+1,0,z) \in S$ alors $(x,1,z) \in S$;
(**3-**) Si $(x+1,y+1,z) \in S$ alors il existe un entier u tel que $(x+1,y,u) \in S$ et $(x,u,z) \in S$.

Lemme 5.9. *Soit S un ensemble fini adéquat et soit $(x,y,z) \in S$. Alors $z = \psi(x,y)$.*

Démonstration. Par induction sur l'ensemble des paires d'entiers muni de l'ordre lexicographique $(x,y) \leq (x',y')$ si $x < x'$ ou si $x = x'$ et $y \leq y'$. □

Lemme 5.10. *Pour tout (x,y) il existe un ensemble fini adéquat S tel que $(x,y,\psi(x,y)) \in S$.*

Démonstration. Par exemple $S = S(x, y)$ défini par

$$\begin{array}{lcl} S(0,y) & = & \{(0,y,y+1)\}, \\ S(x+1,0) & = & \{(x+1,0,\psi(x,1))\} \cup S(x,1), \\ S(x+1,y+1) & = & \{(x+1,y+1,\psi(x+1,y+1))\} \cup S(x,\psi(x+1,y)) \cup S(x+1,y). \end{array}$$

est un ensemble fini adéquat. On le montre par induction sur l'ensemble des paires d'entiers muni de l'ordre lexicographique $(x, y) \leq (x', y')$ si $x < x'$ ou si $x = x'$ et $y \leq y'$. □

Preuve du Théorème 5.7. Pour tout triplet (x, y, z) d'entiers naturels on pose $u(x, y, z) = 2^x 3^y 5^z$ et pour tout ensemble de triplets d'entiers $X = \{t_1, t_2, \ldots, t_k\}$ on pose $c(X) = p_{u(t_1)} p_{u(t_2)} \cdots p_{u(t_k)}$, le code de X en abrégé. On note S_v l'ensemble des triplets d'entiers codé par l'entier v, i.e., $c(S_v) = v$. Ainsi si v est le code d'un ensemble de triplets d'entiers alors $(x, y, z) \in S_v \iff p_{2^x 3^y 5^z} | v$.

Le prédicat "v est le code d'un ensemble de triplets d'entiers" est récursif primitif. En effet v est le code d'un ensemble de triplets d'entiers si et seulement si pour tout $x \leq v$, $(v)_x = 0$ ou $(v)_x = 1$ et, dans ce dernier cas, pour tout $3 < y < x$, p_y ne divise pas x.

Le prédicat $R(x, y, v) =$ 'v est le code d'un ensemble adéquat de triplets d'entiers et il existe $z < v$ tel que $(x, y, z) \in S_v$', défini sur l'ensemble des triplets d'entiers, est primitif récursif (✍).

Par suite la fonction $f(x, y) = \mu\, v\, R(x, y, v)$ est récursive et

$$\psi(x, y) = \mu\, z\, ((x, y, z) \in S_{f(x,y)})$$

est également récursive. □

Fonctions calculables. Un programme P pour machine à registres est donné par

(**1-**) une suite infinie $R_1, R_2, R_3 \dots$ de **registres**, chaque registre contenant un entier naturel ;

(**2-**) une suite finie $S_0, S_1, \dots, S_n$ d'**états (de la machine)** et pour chaque état S_i, $i \neq 0$, une **instruction** à appliquer sur les registres de la forme

(a) ajouter 1 aui contenu du registre R_j et passer à l'état S_k ; ou (instruction conditionnelle)

(b) retrancher 1 au contenu du registre R_j et passer à l'état S_k si le contenu du registre $R_j \neq 0$; sinon passer à l'état S_l.

(**3-**) S_0 est l'**état terminal** : la machine s'arrête lorsqu'elle atteint l'état S_0.

FIGURE 9 –

Une fonction (partielle ou non) $f : \mathbb{N}^n \to \mathbb{N}$, $n \in \mathbb{N}^*$, est **calculable** par le programme P si, partant de l'état S_1 avec les contenus des registres $R_1, R_2, \dots, R_n, R_{n+1}, R_{n+2}, \dots$ initialisés à $x_1, x_2, \dots, x_n, 0, 0, \dots$, la machine atteint après application des instructions associées aux états successifs de la machine l'état S_0 avec $f(x_1, x_2, \dots, x_n)$ dans le registre R_1.

Théorème 5.11. *La classe des fonctions calculables est égale à la classe des fonctions primitives.*

Démonstration. ✍ —

□

Références

[1] N. J. Cutland. *Computability : An introduction to recursive function theory.* Cambridge University Press, 1980.

[2] P. T. Johnstone. *Notes on Logic and Set Theory.* Cambridge University Press, 1987. "This short textbook provides a succinct introduction to mathematical logic and set theory, which together form the foundations for the rigourous development of mathematics. It will be suitable for all mathematics undergraduates coming to the subject for the first time. The book is based on lectures given at the University of Cambridge and covers the basic concepts of logic : first-order logic, consistency, and the completeness theorem, before introducing the reader to the fundamentals of axiomatic set theory. There are also chapters on recursive functions, the axiom of choice, ordinal and cardinal arithmetic and the incompleteness theorems. Dr Johnstone has included numerous exercices designed to illustrate the key elements of the theory and to provide applications of basic logical concepts to other areas of mathematics. Consequently the book, while making an attractive first textbook for those who plan to specialise in logic, will be particularly valuable for mathematicians and computer scientists whose primary interest lie elsewhere".

[3] S. C. Kleene. The theory of recursive functions, approaching its centennial. *Bull. Amer. Math. Soc. (N.S.)*, 5(1) :43–61, 07 1981.

[4] J. C. Shepherdson and H. E. Sturgis. Computability of recursive functions. *J. ACM*, 10(2) :217–255, Apr. 1963.

6 A suivre