Séquence 11 - Arbres

Table des matières

I. Arbres binaires : concepts généraux

I. 1. Définitions, vocabulaire des arbres

Définition 1 (Arbre binaire)

Un arbre binaire (*binary tree* en anglais) est une structure de données construite comme une hiérarchie de cellules de données, appelées nœuds. Un arbre binaire est défini par induction de la manière suivante :

soit la structure est vide : elle ne contient aucun nœud, et on parle d'arbre binaire vide ;

sinon, un arbre binaire est un nœud , relié à **exactement deux autres nœuds** gauche et droit, qui constituent eux-mêmes deux sous-arbres.

Notation 1

On notera $N(\ell, x, r)$ un nœud contenant l'information x et relié au nœud gauche ℓ (*left*) et au nœud droit r (*right*)
On note E l'arbre vide (E comme *empty*).

Remarque. L'arbre lui-même est donc la donnée... d'un premier nœud, appelé racine, qui donnera accès à tous les autres

Définition 2 (Étiquette d'un nœud)

Pour repérer les nœuds d'un arbre, on leur attribue généralement un nombre entier **arbitraire** appelé **étiquette**. On dit que l'on a étiqueté l'arbre.
Cet étiquetage peut être fait de différentes manières, généralement en lien avec les différentes manières de parcourir l'arbre.

Définition 3 (Valeur d'un nœud)

Les données contenues dans les nœuds de l'arbre peuvent être de n'importe quel type. Par contre, elles sont toutes de même type pour un même arbre.
On peut donc faire des arbres d'entiers, de flottants, de chaînes de caractères, de listes, de n-uplets, d'enregistrements, des arbres d'arbres...etc
La donnée stockée dans un nœud est appelée **valeur** du nœud.

Définition 4 (Nœud père, nœuds fils)

Les deux nœuds gauche ℓ et droit r reliés au nœud $N(\ell, x, r)$ sont appelés **nœuds fils** du nœud $N(\ell, x, r)$.
Réciproquement, $N(\ell, x, r)$ est appelé **nœud père** de ℓ et r

Remarques.

- dans un arbre binaire, un nœud a donc exactement deux fils ;
- tout nœud a un nœud père, sauf la racine.

Propriété 1 (Définition inductive du nombre de nœuds)

Le nombre de nœuds d'un arbre binaire est donnée par la définition inductive suivante :

$$\begin{aligned} n(E) &= 0 \\ n(N(\ell, v, r)) &= 1 + n(\ell) + n(r) \end{aligned}$$

Définition 5 (Feuilles d'un arbre binaire)

On appelle feuilles d'un arbre binaire les nœuds dont les deux fils sont vides.

Propriété 2 (Définition inductive du nombre de feuilles)

Le nombre de feuilles d'un arbre binaire est donnée par la définition inductive suivante :

$$\begin{aligned} f(E) &= 0 \\ f(N(E, v, E)) &= 1 \\ f(N(\ell, v, r)) &= f(\ell) + f(r) \end{aligned}$$

Définition 6 (Profondeur d'un nœud)

La profondeur d'un nœud est le nombre d'arêtes le séparant de la racine (distance à la racine).

Définition 7

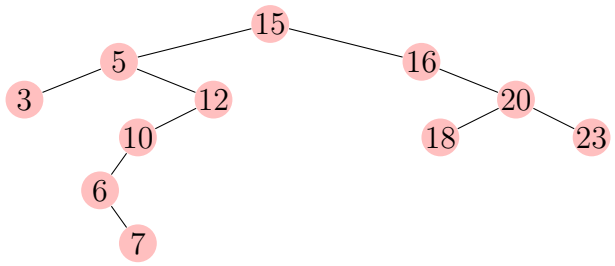
On appelle niveau numéro $k \in \mathbb{N}$ d'un arbre l'ensemble des nœuds de profondeur k dans cet arbre.

Définition 8 (Hauteur d'un arbre binaire)

La hauteur d'un arbre binaire est la profondeur maximale parmi tous les nœuds non vides.
C'est le nombre maximal d'arêtes entre la racine et une feuille.
Par convention, la hauteur d'un arbre vide E est fixée à $h(E) = -1$.

Exemple 1

La hauteur de l'arbre binaire suivant est :



Propriété 3 (Définition inductive de la hauteur)

La hauteur d'un arbre binaire est donnée par la définition inductive suivante :

$$\begin{aligned} h(E) &= -1 \\ h(N(\ell, v, r)) &= 1 + \max\left(h(\ell),\, h(r)\right) \end{aligned}$$

Définition 9 (Quelques arbres binaires particuliers : parfait, complet, peigne)

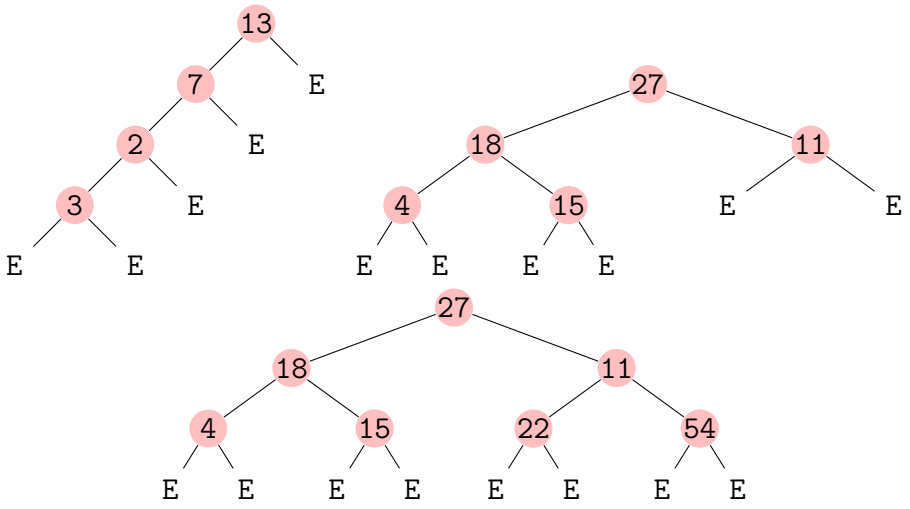
— Un arbre binaire est dit **parfait** lorsque tous ses niveaux sont remplis, y compris le dernier. Les arbres binaires parfaits ont des propriétés de compacité et d'équilibrage très appréciables... mais ils sont rares !

— Un arbre binaire est dit **complet** lorsque tous ses niveaux sont remplis de gauche à droite sans trou, sauf éventuellement le dernier niveau qui peut n'être rempli que partiellement de gauche à droite. Ces arbres sont très intéressants car ils peuvent être stockés sans ambiguïté dans un tableau, nous y reviendrons. Un arbre parfait est, a fortiori, complet.

— Un arbre binaire est appelé **peigne** lorsque tous ses nœuds ont au moins un fils vide.

Exemple 2

Voici, de gauche à droite, un arbre peigne, un arbre binaire complet, un arbre binaire parfait



Propriété 4 (Arbres binaires parfaits)

Chaque niveau k d'un arbre binaire parfait contient 2^k noeuds.
Le nombre de nœuds d'un arbre binaire parfait de hauteur h est donc :

$$n = \sum_{k=0}^{h} 2^k = 2^{h+1} - 1$$

I. 2. Parcours d'arbres

Contrairement au parcours de liste (de la tête vers la queue), il n'existe pas une manière, unique, évidente, naturelle de parcourir un arbre binaire.
Il existe plusieurs manières de parcourir les nœuds d'un arbre :

Parcours préfixe : (*preorder* en anglais), on applique le traitement aux données stockées dans le nœud, puis descend dans le sous-arbre gauche, et enfin dans le sous-arbre droit

Parcours post-fixe : (*postorder* en anglais), on descend dans le sous-arbre gauche, puis dans le sous-arbre droit et ensuite seulement on applique le traitement aux données stockées dans le nœud

Parcours infixe : on descend dans le sous-arbre gauche, on applique le traitement aux données stockées dans le nœud, puis dans le sous-arbre droit et ensuite seulement

Nous avons vu des exemples de parcours en TP.

I. 3. Premiers résultats théoriques (important !)

Théorème 1 (Encadrement du nombre de nœuds en fonction de la hauteur)

On note n le nombre de nœuds non vides d'un arbre binaire et h sa hauteur.
On a l'encadrement suivant :

$$h + 1 \leq n \leq 2^{h+1} - 1$$

Remarque. La minoration correspond au cas particulier d'un arbre peigne, qui peut être vu... comme une liste !
La majoration correspond au cas où l'on a rempli avec un maximum de nœud chaque niveau : c'est le cas pour un arbre binaire parfait

Démonstration

L'inégalité gauche $h+1 \leq n$ est directement issue de la définition de la hauteur d'un arbre binaire.
L'inégalité de droite $n \leq 2^{h+1}-1$ se démontre par induction structurelle.

Cas de base : il n'y a qu'un terme de base, l'arbre vide E. Pour cet arbre, $h(E) = -1$ et $n(E) = 0$ et on vérifie bien l'inégalité $0 \leq 2^{-1+1} - 1 = 0$.

Induction : Supposons l'inégalité vraie pour les générations existantes d'arbres et considérons un nouvel arbre binaire $N(\ell, x, r)$ construit à partir de deux sous-arbres ℓ et r existants.
On note h et n la hauteur et le nombre de nœuds de l'arbre binaire $N(\ell, x, r)$.
On note $h(\ell)$ et $h(r)$ les hauteurs respectives des sous-arbres l et r.
On note $n(\ell)$ et $n(r)$ les nombres de nœuds respectifs des deux sous-arbres ℓ et r.
Par hypothèse d'induction $n(\ell) \leq 2^{h(l)+1} - 1$ et $n(r) \leq 2^{h(r)+1} - 1$.
D'autre part, le nombre de nœuds de l'arbre $N(\ell, x, r)$ est égal à :$h = 1 + n(l) + n(r)$.
Sa hauteur est égale à $h = 1 + \max(h(\ell), h(r))$. On suppose par exemple que le sous-arbre gauche est le plus profond : $h(l) = \max(h(l), h(r))$ et donc $h(l) \geq h(r)$.
Alors :

$$
\begin{aligned}
2^{h+1} - 1 &= 2^{(1+h(\ell))+1} - 1 && \text{car on a supposé que } h = 1 + h(\ell) \\
&= 2^{1+h(\ell)} \times 2 - 1 \\
&= 2^{1+h(\ell)} + 2^{1+h(\ell)} - 1 \\
&\geq 2^{1+h(\ell)} + 2^{1+h(r)} - 1 && \text{car on a supposé que } h(\ell) \leq h(r) \\
&\geq n(\ell) + 1 + n(r) + 1 - 1 && \text{par hypothèse d'induction} \\
&= n \text{ car } n = 1 + n(\ell) + n(r)
\end{aligned}
$$

Nous avons donc montré la propriété pour un arbre issu de la nouvelle génération.

Conclusion : Par principe d'induction structurelle, la propriété est vraie pour tous les arbres binaires.

Propriété 5

La hauteur d'un arbre binaire à n nœuds ($n \geq 1$) est donc telle que :

$$\lceil \log_2(n+1) \rceil - 1 \leq h \leq n - 1$$

Démonstration

$$
\begin{aligned}
& n \leq 2^{h+1} - 1 \\
\Leftrightarrow\ & \log_2(n+1) \leq h + 1 \\
\Leftrightarrow\ & \lceil \log_2(n+1) \rceil \leq h + 1 \text{ car } h + 1 \in \mathbb{N} \\
\Leftrightarrow\ & \lceil \log_2(n+1) \rceil - 1 \leq h
\end{aligned}
$$

Théorème 2 (Nombre de sous-arbres vides d'un arbre binaire)

On note n le nombre de nœuds non vides d'un arbre binaire.
Le nombre de sous-arbres vides est $n_{\text{vides}} = n + 1$

Démonstration

La preuve se fait là encore par induction structurelle.

Cas de base : il n'y a qu'un terme de base, l'arbre vide E. Pour cet arbre, $n(E) = 0$... et il y a bien $n(E) + 1 = 1$ sous arbre vide !

Induction : supposons l'inégalité vraie pour les générations existantes d'arbres et considérons un nouvel arbre binaire $N(\ell, x, r)$ construit à partir de deux sous-arbres ℓ et r existants.
On note n le nombre de nœuds de l'arbre binaire $N(\ell, x, r)$ et n_{vides} le nombre de ses sous-arbres vides.
On note $n(\ell)$ et $n(r)$ les nombres de nœuds respectifs des deux sous-arbres ℓ et r et $n_{\text{vides}}(\ell)$ et $n_{\text{vides}}(r)$ leurs nombres de sous-arbres vides.
Par hypothèse d'induction $n_{\text{vides}}(\ell) = n(\ell) + 1$ et $n_{\text{vides}}(r) = n(r) + 1$.
Par construction inductive :

$$n_{\text{vides}} = n_{\text{vides}}(\ell)+n_{\text{vides}}(r) = (n(\ell)+1)+(n(r)+1) = (n(\ell)+n(r)+1)+1 = n+1$$

On a donc montrer la propriété pour les arbres binaires de la nouvelle génération.

Conclusion : Par principe d'induction structurelle, la propriété est vraie pour tous les arbres binaires.

Remarque. On peut voir les choses de la manière suivante : dès que l'on enracine une nouvelle feuille à gauche ou à droite d'un nœud existant, on enlève un sous-arbre vide, mais on en rajoute deux nouveaux.

$$n_{\text{vides}} = \underbrace{1}_{\text{arbre vide initial}} + \underbrace{(-1+2)+(-1+2)+(-1+2)+\ldots+(-1+2)}_{n \text{ fois car } n \text{ nœuds ajoutés}} = n+1$$

La construction inductive d'un arbre binaire permet ainsi immédiatement de comprendre la formule.

Théorème 3 (Encadrement du nombre de feuilles en fonction de la hauteur)

On considère un arbre binaire **non vide**.
On note f le nombre de feuilles d'un arbre binaire.
On a l'encadrement suivant en fonction de la hauteur h de l'arbre binaire :

$$1 \leq f \leq 2^h$$

On a l'encadrement suivant en fonction du nombre n de nœuds non vides de l'arbre binaire

$$1 \leq f \leq \frac{n+1}{2}$$

Remarque. La minoration correspond au cas des arbres peigne, qui n'ont qu'une seule feuille.
La majoration correspond au cas des arbres binaires parfaits, pour lesquels tous les nœuds du dernier niveau non vide (niveau numéro h contenant 2^h nœuds) sont des feuilles.

Démonstration

La preuve se fait là encore par induction structurelle.

Cas de base : il y a une infinité de cas de bases, constituée de tous les arbres-feuille du type $N(E, x, E)$. Pour ces arbres, $f = 1$ et leur hauteur est bien égale à 0, la propriété est vérifiée

Induction : supposons l'inégalité vraie pour les générations existantes d'arbres et considérons un nouvel arbre binaire $N(\ell, x, r)$ construit à partir de deux sous-arbres ℓ et r existants.
On note h la hauteur de l'arbre binaire $N(\ell, x, r)$ et f le nombre de ses feuilles.
On note $h(\ell)$ et $h(r)$ les hauteurs respectives des deux sous-arbres ℓ et r et $f(\ell)$ et $f(r)$ leurs nombres de feuilles.
Par hypothèse d'induction $1 \leq f(\ell) \leq 2^{h(\ell)}$ et $1 \leq f(r) \leq 2^{h(r)}$.
On a vu que : $f = f(\ell) + f(r)$ et $h = 1 + \max(h(\ell), h(r))$.
Supposons que l'arbre le plus haut est celui de gauche (le cas où c'est celui de droite est traité de manière totalement identique). On a donc $h = 1 + h(\ell)$ et $h(r) \leq h(\ell)$. On utilisant ces éléments, on aboutit à l'encadrement :

$$1 \leq 2 \leq f \leq 2^{h(\ell)} + 2^{h(r)} \leq 2 \times 2^{h(\ell)} = 2^{h(\ell)+1} = 2^h$$

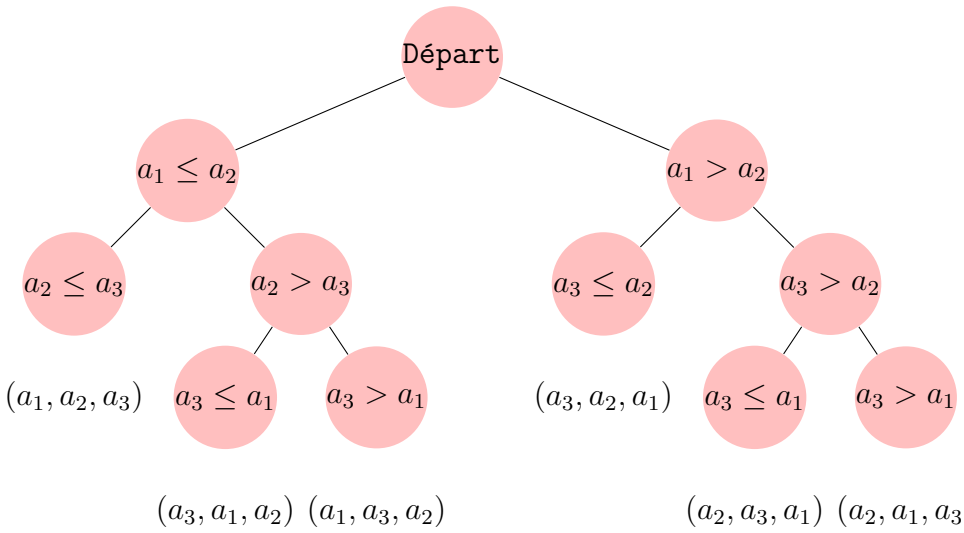
et on a donc montrer la propriété pour les arbres binaires de la nouvelle génération.

Conclusion : Par principe d'induction structurelle, la propriété est vraie pour tous les arbres binaires.

L'autre encadrement s'obtient également en réalisant une preuve par induction structurelle.

I. 4. Application : complexité asymptotique optimale pour les tris par comparaison

Tout algorithme de tri par comparaisons peut être représenté par un arbre binaire appelé **arbre de décision**
Chaque test de comparaison apporte une nouvelle information, qui est utilisée pour faire progresser l'algorithme. Les algorithmes se différencient dans la manière dont ils utilisent ces informations pour avancer, plus ou moins efficacement, et par l'ordre dans lequel ils effectuent ces tests.
Par exemple, un petit algorithme qui trierait simplement 3 valeurs selon la relation d'ordre $\leq$ serait associé à cet arbre de décision.



Chaque feuille de l'arbre de décision est associée à la solution trouvée (représentée sur la figure, en dessous de chaque feuille). Il y a donc autant de feuilles dans l'arbre de décision que de permutations possibles du tableau de taille p donné en entrée.
Par exemple, ici, le tableau d'entrée a 3 éléments, donc le nombre de feuilles est 6.
Tout arbre de décision associé à un algorithme de tri par comparaisons, quelque soit sa structure, doit avoir au moins $p!$ feuilles : en effet il doit pouvoir produire comme verdict chacune des permutations d'un ensemble à p éléments puisqu'il doit pouvoir répondre pour n'importe quel tableau de taille p ayant subi une permutation.
Par ailleurs, le nombre de tests effectués dépendant du chemin depuis la racine de l'arbre jusqu'à atteindre l'une des feuilles : il est égal, dans le pire des cas, à la hauteur h de l'arbre de décision. Si on note $T(p)$ la complexité dans le pire des cas pour un tableau de taille p, on a, d'après la formule du théorème 5 :

$$T(p) = h \geq \lceil \log_2(n+1) \rceil - 1$$

Il nous reste à faire le lien entre le nombre de nœuds n et le nombre $p!$ de feuilles. On sait, d'après la formule du théorème 3, que $f = p! \leq \dfrac{n+1}{2}$ donc $n \geq 2p! - 1$.
On en déduit que :

$$T(p) \geq \left\lceil \underbrace{\log_2(2p!)}_{\substack{=\log_2(2)+\log_2(p!) \\ =1+\log_2(p!)}} \right\rceil - 1 \Rightarrow T(p) \geq \lceil \log_2(p!) \rceil$$

et on a utilisé le fait que l'on a toujours, $\forall a \in \mathbb{N}, y \in \mathbb{R}$, $\lceil a + y \rceil = a + \lceil y \rceil$.
On a donc montré le résultat important suivant, donnant une minoration de la complexité mesurée en nombre de comparaisons pour tout algorithme de tri par comparaison fonctionannt sur un tableau de taille p : :

$$T(p) \geq \lceil \log_2(p!) \rceil$$

Il nous reste à montrer que $\log_2(p!) \underset{p \to +\infty}{\in} \Theta(p \log_2(p))$.
On peut déjà écrire, que le logarithme d'un produit est égal à la somme des logarithmes :

$$\log_2(p!) = \sum_{k=1}^{p} \log_2(k)$$

On a immédiatement $\log_2(p!) \leq p\log_2(p)$ par une majoration grossière.

Pour la minoration, cela peut se démontrer de plusieurs manières

I. 4. a. Ordre de grandeur de $\log_2(p!)$ quand $p \to +\infty$: preuve par comparaison série-intégrale

On pose $f(x) = \log_2(x)$. Soit $k \in \mathbb{N}$, $k \geq 2$. Sur l'intervalle $[k, k+1]$, $f(k) \leq f(x)$ car f est croissante sur $[k, k+1]$.
En intégrant sur cet intervalle, par croissance de l'intégrale :

$$\int_k^{k+1} f(k)\,\mathrm{d}x \leq \int_k^{k+1} f(x)\,\mathrm{d}x$$

$$f(k) \leq \int_k^{k+1} f(x)\,\mathrm{d}x$$

Sur l'intervalle $[k-1, k]$, $f(x) \leq f(k)$ car f est croissante sur $[k-1, k]$.
En intégrant sur cet intervalle, par croissance de l'intégrale :

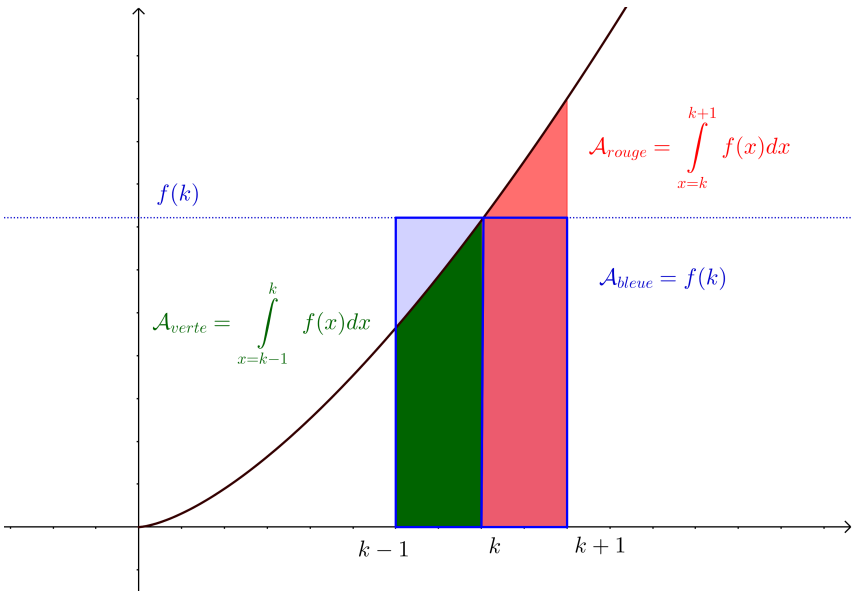
$$\int_{k-1}^{k} f(x)\,\mathrm{d}x \leq \int_{k-1}^{k} f(k)\,\mathrm{d}x$$

$$\int_{k-1}^{k} f(x)\,\mathrm{d}x \leq f(k)$$

On a donc, pour tout $k \geq 2$:

$$\int_{k-1}^{k} f(x)\,\mathrm{d}x \leq f(k) \leq \int_k^{k+1} f(x)\,\mathrm{d}x$$

Cette inégalité peut être visualisée en considérant les aires sur le graphe ci-dessous :



Sommons tous ces encadrements pour k allant de 2 à n : :

$$\sum_{k=2}^{p} \int_{k-1}^{k} f(x)\,\mathrm{d}x \leq \sum_{k=2}^{p} f(k) \leq \sum_{k=2}^{n} \int_{k}^{k+1} f(x)\,\mathrm{d}x$$

$$\Leftrightarrow \int_{1}^{p} f(x)\,\mathrm{d}x \leq \sum_{k=2}^{p} f(k) \leq \int_{2}^{p+1} f(x)\,\mathrm{d}x$$

Et utilisant l'expression algébrique de f, $f(x) = \log_2(x)$, on obtient :

$$\int_{1}^{p} \log_2(x)\,\mathrm{d}x \leq \sum_{k=2}^{p} \log_2(k) \leq \int_{2}^{p+1} \log_2(x)\,\mathrm{d}x$$

Une primitive de $f(x) = \log_2(x)$ est $F(x) = x\log_2(x) - \dfrac{x}{\ln(2)}$, on a donc :

$$\left[x\log_2(x) - \frac{x}{\ln(2)}\right]_1^p \leq \sum_{k=2}^{p} \log_2(k) \leq \left[x\log_2(x) - \frac{x}{\ln(2)}\right]_2^{p+1}$$

ce qui permet de conclure que $\sum_{k=1}^{p} \log_2(k) \underset{p\to+\infty}{\in} \Theta(p\log_2(p))$ Notons que l'on peut faire démarrer la somme à 1 car $\log_2(1) = 0$.

I. 4. b. Ordre de grandeur de $\log_2(p!)$ quand $p \to +\infty$: preuve par inégalité log-somme

Propriété 6 (Inégalité log-somme)

Pour tout choix de nombres $(a_1, \dots a_n) \in \mathbb{R}^+$ et $(b_1, \dots b_n) \in \mathbb{R}^{+,*}$, l'inégalité suivante est vérifiée :

$$\sum_{k=1}^{n} a_k \log(\frac{a_k}{b_k}) \geq a\log(\frac{a}{b})$$

où $a = \sum_{k=1}^{n} a_k$ et $b = \sum_{k=1}^{n} b_k$.

Démonstration

Cette inégalité se montre en utilisant la convexité de la fonction $f(x) = x\log(x)$, où log désigne un logarithme dans une base $\beta > 1$ choisie. Pour toute fonction convexe :

$$f\left(\sum_{k=1}^{n} \lambda_k f(x_k)\right) \leq \sum_{k=1}^{n} \lambda_k f(x_k)$$

où la somme des λ_k vaut 1. Si on applique l'inégalité de convexité en $x_k = \frac{a_k}{b_k}$ pour des coefficient $\lambda_k = \frac{b_k}{\sum_{k=1}^{n} b_k} = \frac{b_k}{b}$, on a :

$$\sum_{k=1}^{n} a_k \log(\frac{a_k}{b_k}) = \sum_{k=1}^{n} b \times \frac{b_k}{b} \times \frac{a_k}{b_k} \log_2(\frac{a_k}{b_k}) = b \sum_{k=1}^{n} \lambda_k f(\frac{a_k}{b_k})$$

$$\geq b \times f(\sum_{k=1}^{n} \lambda_k \frac{a_k}{b_k}) = b \times f(\frac{a}{b}) = a\log(\frac{a}{b})$$

Pour $a_k = 1$ et $b_k = \frac{1}{k}$, cette inégalité donne :

$$\sum_{k=1}^{p} \log_2(k) \geq p\log\left(\frac{p}{\sum_{k=1}^{p} \frac{1}{k}}\right)$$

et, d'après vos connaissances sur la série harmonique, vous savez que :

$$\sum_{k=1}^{p} \frac{1}{k} \underset{p\to+\infty}{\sim} \ln(p)$$

Or, par définition d'un équivalent, cela signifie que $\forall \varepsilon > 0$, $\exists p_0 \in \mathbb{N}$ tel que :

$$\forall p \geq p_0 \quad \sum_{k=1}^{p} \frac{1}{k} - \ln(p)| < \varepsilon|\ln(p)|$$

On peut enlever les valeurs absolues autour de $\ln p$ car $p \geq 1$ et écrire, par définition de la valeur absolue :

$$\forall p \geq p_0 \quad -\varepsilon\ln(p) < \sum_{k=1}^{p} \frac{1}{k} - \ln(p) < \varepsilon\ln(p)$$

En regardant uniquement l'inégalité de droite, cela permet d'écrire :

$$\sum_{k=1}^{p} \frac{1}{k} - \ln(p) < \varepsilon\ln(p) \Rightarrow \sum_{k=1}^{p} \frac{1}{k} < (1+\varepsilon)\ln(p)$$

On en déduit donc, pour $p \geq p_0$:

$$\log_2(p!) \geq p\log_2(p) - p\log_2((1+\varepsilon)\ln(p))$$

ε est maintenant fixé, petit.
Comme $p\log((1+\varepsilon)\ln(p)) \in \underset{p\to+\infty}{o}(p\log(p))$, pour un $\varepsilon' > 0$ arbitrairement petit , $\exists p_1 \in \mathbb{N}$ tel que :

$$\forall p \geq \max(p_1, p_0)\,, \quad p\log((1+\varepsilon)\ln(p)) < \varepsilon' p\log(p)$$

Et donc, en ayant pu fixer ε' arbitrairement petit :

$$\forall p \geq \max(p_1, p_0)\,, \quad \log_2(p!) \geq (1-\varepsilon')p\log(p)$$

Donc, pour p assez grand, il existe une constante $c > 0$ telle que :

$$\log_2(p!) \geq c\, p\log(p)$$

I. 4. c. Ordre de grandeur de $\log_2(p!)$ quand $p \to +\infty$: formule de Stirling

La formule de Stirling nous donne un équivalent asymptotique de $p!$:

$$p! \underset{p\to+\infty}{\sim} \left(\frac{p}{e}\right)^p \sqrt{2\pi p}$$

qui, combiné avec la formule d'Euler Mac-Laurin, nous donne un développement asymptotique très précis de $\ln(n!)$:

$$\ln(p!) = p\ln(p) - p + \frac{1}{2}\ln(2\pi p) + O(\frac{1}{p})$$

et nous permet donc d'affirmer que :

$$\log_2(p!) \underset{p\to+\infty}{\sim} p\log(p)$$

donc c'est a fortiori un grand Θ.
Au final, on a donc montré que, pour p assez grand, il existe $c > 0$ telle que $T(p) \geq c \times p\log(p)$: la complexité dans le pire des cas est au mieux linéarithmétique pour des algorithmes de tri par comparaisons.
On peut montrer un résultat semblable sur la complexité moyenne, mais c'est un peu plus délicat.

Remarque. Il existe des algorithmes de tri qui ne sont pas des tris par comparaison et qui présentent des complexité asymptotiques meilleures que du $p\log(p)$.
On peut citer :

le tri par comptage, qui suppose que l'on trie un tableau d'entiers naturels dont on connaît la plage de valeurs, c'est-à-dire les valeurs maximale et minimale ; Dans ce cas, si b est l'amplitude de la plage de valeurs, le tri se fait en $O(n+b)$

le tri par base (*radix sort*), on trie un tableau de n nombres écrits dans une base donné b. On effectue plusieurs tris par comptage : un premier tri par comptage en prenant comme clé de tri le chiffre des unités, puis un second tri par comptage avec comme clé le chiffre suivant dans l'écriture en base b...etc Si d est le nombre maximum de chiffres parmi tous les nombres à trier, la complexité est en $O(d \times (n+b))$

le tri par paquets (*bucket sort*), qui part de l'hypothèse que les n données à trier sont réparties uniformément sur un intervalle borné $[a,b]$. On découpe cet intervalle en n sous-intervalles, autant qu'il y a de valeurs à trier :

$$I_k = [a + k\frac{b-a}{n},\ a + (k+1)\frac{b-a}{n}], \quad 0 \leq k \leq n-1$$

On détermine, pour chaque valeur x, le numéro k du sous-intervalle auquel elle appartient avec un simple quotient :

$$\frac{k}{n} = \left\lfloor \frac{x-a}{b-a} \right\rfloor \Leftrightarrow k = n \times \left\lfloor \frac{x-a}{b-a} \right\rfloor$$

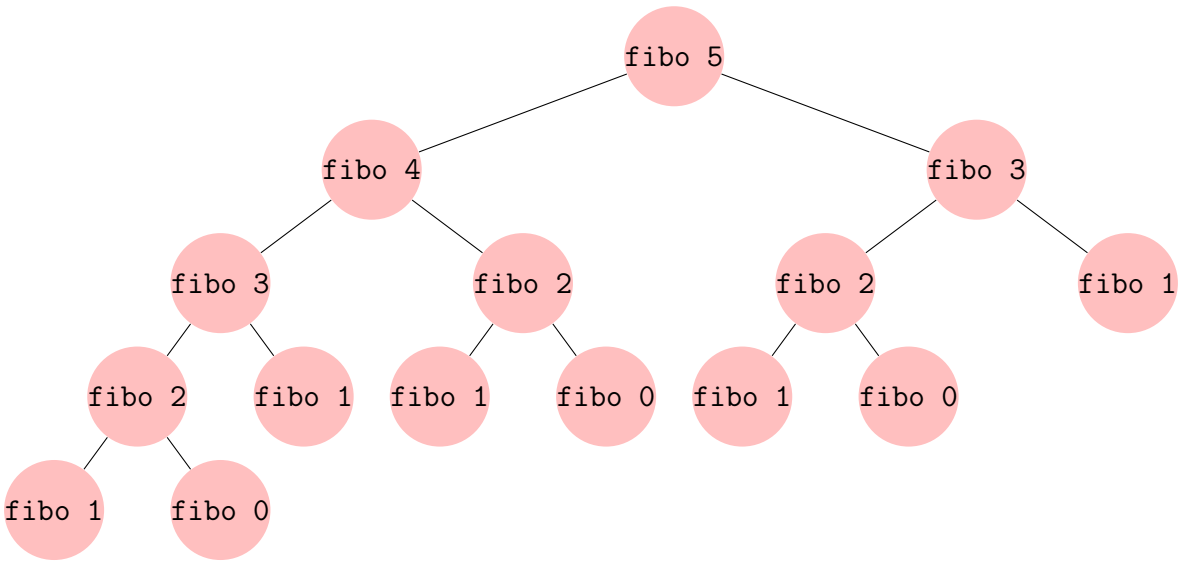
On effectue un tri par comptage, mais avec comme clé de tri le numéro k du sous-intervalle auquel appartient la valeur. Si les valeurs sont réparties uniformément, on a statistiquement une valeur par sous-intervalle. Mais dans la réalité, on peut en avoir plus ou moins, mais on est quasiment certains qu'il n'y aura au plus que quelques valeurs par sous-intervalle. On effectue donc un tri par insertion dans chaque sous-intervalle, qui sera très rapide sur quelques valeurs.

I. 5. Application : arbre des appels pour les algorithmes récursifs

On considère une fonction `fibo` qui calcule le terme de rang n de la suite de Fibonacci :

$$\begin{cases} F(0) = 0 \\ F(1) = 1 \\ F(n) = F(n-1) + F(n-2) \end{cases}$$

Voici l'arbre des appels pour `fibo 5` :



L'arbre des appels nous renseigne sur la complexité de l'algorithme :

Complexité temporelle : elle est en lien avec le **nombre de nœuds de l'arbre**, ici pas forcément évidente mais plus clair sur l'arbre des appels de l'algorithme de Hanoï

Complexité spatiale : elle est en lien avec le **nombre maximal de blocs d'activation empilés** au cours de l'exécution de algorithme, donc avec la **hauteur de l'arbre**

La parcours infixe de l'arbre des appels donne l'ordre de traitement des différents appels récursifs.

I. 6. Implémentations

I. 6. a. Implémentations par maillons chaînés

I. 6. b. Implémentation des arbres binaires complets avec des tableaux

Les arbres binaires complets peuvent être très avantageusement stockés dans des tableaux, le niveau k de l'arbre (niveaux numérotés de 0 à h) dans la convention où $h(E) = -1$ correspondant à l'intervalle d'indice $[\![2^k, 2^k - 1]\!]$. Les nœuds sont numérotés de haut en bas et de gauche à droite Chaque niveau correspond à un intervalle contenant deux fois plus d'indices que le précédent, ce qui est normal avec un arbre binaire complet !
fils 2*i +1, 2*i+2 père $\left\lfloor \frac{i-1}{2} \right\rfloor$ Cette implémentation sera utilisée massivement pour les structures de tas.

II. Cas particulier d'arbres binaires 1 : les arbres binaires de recherche

III. Cas particulier d'arbres binaires 2 : les tas

IV. Arbres généraux