5 Récursion et arbres

5.1 Nombres Fibonacci

On définit les nombres Fibonacci par $F(0) = 0, F(1) = 1$ et la récurrence $F(n) = F(n-1)+F(n-2)$ pour $n > 1$. Les racines de l'équation de récurrence homogène sont $\phi = \frac{1+\sqrt{5}}{2}$ et $\phi_- = \phi - 1 = \frac{1-\sqrt{5}}{2}$. La solution spécifique se trouve par la solution des équations W(fr)

$$F(0) = 0 = a\phi^0 + b\phi_-^0; \qquad F(1) = 1 = a\phi^1 + b\phi_-^1.$$

On obtient $a = -b = 5^{-1/2}$, donc

$$F(n) = \frac{\phi^n - \phi_-^n}{\sqrt{5}} = \left(5^{-1/2} + o(1)\right)\left(\frac{1+\sqrt{5}}{2}\right)^n = \Theta(\phi^n). \tag{5.1}$$

Dans d'autres mots, $F(n)$ a une croissance exponentielle. Comme $\phi_-^n/\sqrt{5} < 1/2$ pour tout $n \geq 0$ et $F(n)$ est entier, on voit aussi que $F(n)$ égale à $\phi^n/\sqrt{5}$, arrondi au plus proche entier : $F(n) = \left\lfloor \phi^n/\sqrt{5} + 1/2 \right\rfloor$.

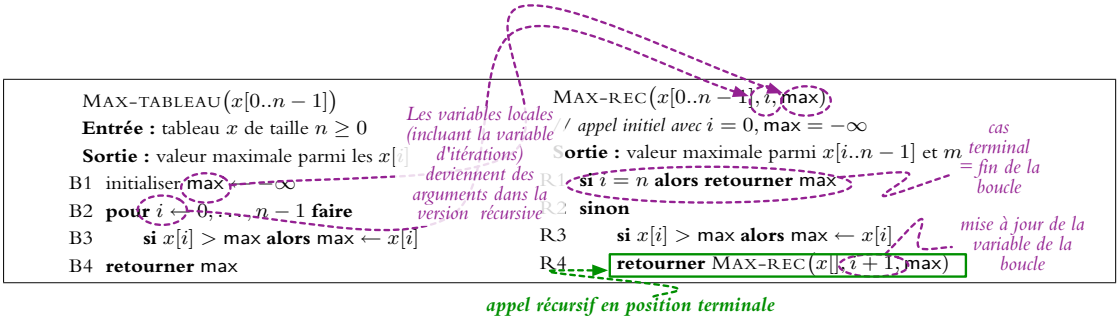
Théorème 5.1. *L'algorithme d'Euclid (v. Exemple 4.2) prend au plus* $\log_\phi(b) + O(1)$ *itérations pour calculer le plus grand commun diviseur entre* a *et* b *avec* $b \leq a$.

Démonstration. On définit n_i pour $i = 1, 2, \ldots$ comme l'indice du nombre Fibonacci pour lequel $F(n_i) \leq a < F(n_i + 1)$ au début d'itération i. Si $b < F(n_i)$, on a immédiatement $n_{i+1} \leq n_i - 1$ par l'affectation $a \leftarrow b$ avant la prochaine itération. Si $b \geq F(n_i)$, alors $a \bmod b \leq a - b < F(n_i - 2)$. Donc, après 2 itérations, on a $a < F(n_i - 2)$ et $n_{i+2} \leq n_i - 2$. En conséquence, le nombre d'itérations est borné par $n_1 - 2$: avec $n_i \leq 3$, on a $0 \leq b \leq a < 3$ et l'algorithme se termine en 1 itération au plus. Pour la borne plus serrée du théorème, on considère le b initiel : $F(n_2) \leq b < F(n_2 + 1)$ (affectation $a \leftarrow b$ dans la première itération), et l'algorithme finit en $n_2 - 1$ itérations au plus. Or, $n_2 = \lceil \log_\phi(b\sqrt{5}) \rceil + o(1)$ par Équation (5.1). ■

REMARQUE. La borne de théorème 5.1 montre le pire cas : c'est avec $a = F(n+1), b = F(n)$. Avec un tel choix, $a \bmod b = F(n-1)$, et l'algorithme doit itérer $n - 2$ fois pour arriver à $a = F(3) = 2, b = F(2) = 1$ et se terminer après une dernière itération de plus où b devient 0 et on retourne la réponse $a = 1$.

5.2 Récursion terminale

On peut toujours transformer une boucle en un algorithme récursif équivalent.



L'appel récursif est en **position terminale** : on n'a pas besoin d'attendre le retour de l'appel, donc on peut simplement transférer le contrôle d'exécution à l'appel executé. Plutôt qu'allouer de nouvelle espace sur la pile d'exécution, on peut remplacer le bloc alloué pour les variables locales et adresse de retour. En conséquence, la pile d'exécution ne déborde pas. Ce remplacement est en fait l'équivalent de transformer la récurrence en une itération, performé automatiquement par des compilateurs modernes. W(fr)

appel récursif n'est pas en position terminale: il faut attendre la valeur retournée par l'appel récursif pour évaluer l'expression max{...}

MAX-REC2$(x[0..n-1], i)$
Sortie : valeur maximale parmi $x[i..n-1]$
R1 **si** $i = n$ **alors retourner** $-\infty$
R2 **sinon retourner** $\max\{x[i], \text{MAX-REC2}(x, i+1)\}$

Si on se sert de la valeur retournée par l'appel récursif dans le calcul, il faut allouer un nouveau bloc d'activation pour chaque appel sur la pile.

5.3 Diviser pour régner

Un principe général dans la conception d'algorithmes récursif est celui de **diviser pour régner** (*divide and conquer*). La démarche générale est (1) couper le problème dans des sous-problèmes similaires, (2) appel récursif pour résoudre les sous-problèmes, (3) combinaison des résultats des sous-problèmes.

MAX-MIN$(x[0..n-1], g, d)$ // *trouve le max et min parmi* $x[g..d-1]$
// *appel initiel avec* $g = 0, d = n$
MM1 **si** $d - g = 0$ **alors retourner** $(-\infty, \infty)$
MM2 **si** $d - g = 1$ **alors retourner** $(x[g], x[g])$
MM3 **si** $d - g = 2$ **alors**
MM4 **si** $x[g] < x[g+1]$ **alors retourner** $(x[g+1], x[g])$ **sinon retourner** $(x[g], x[g+1])$
MM5 mid $\leftarrow \lfloor (g+d)/2 \rfloor$ // *(diviser)*
MM6 $(\text{max}_1, \text{min}_1) \leftarrow$ MAX-MIN(x, g, mid) ; $(\text{max}_2, \text{min}_2) \leftarrow$ MAX-MIN(x, mid, d)
MM7 **si** $\text{max}_1 > \text{max}_2$ **alors** max $\leftarrow \text{max}_1$ **sinon** max $\leftarrow \text{max}_2$
MM8 **si** $\text{min}_1 < \text{min}_2$ **alors** min $\leftarrow \text{min}_1$ **sinon** min $\leftarrow \text{min}_2$
MM9 **retourner** (max, min)

Le temps de calcul de l'algorithme MAX-MIN s'écrit par la récurrence

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(1) \qquad \{n > 2\} \tag{5.2}$$

Théorème 5.2. *La solution de l'Équation* (5.2) *est* $T(n) = \Theta(n)$.

Exercice 5.1. *Démontrer Théorème 5.2.*

Exercice 5.2. *Donner le nombre exact de comparaisons (Lignes MM4, MM7, MM8) dans l'algorithme* MAX-MIN.