Chapitre 7

Correction des exercices

Analyse d'un algorithme

Exercice 1 On choisit l'itérateur (n, u_n, u_{n+1}) ou $\big((-1)^n, u_n, u_{n+1}\big)$, qui donnent lieu aux fonctions suivantes :

```
let u n =          (* version impérative *)
  let x = ref (1, 1) in
  for k = 1 to n do
    let (u, v) = !x in
      x := if k mod 2 = 1 then (v, u + v) else (v, v - u) ;
  done ;
  fst !x ;;

let v n =       (* version récursive terminale *)
  let rec aux (s, u, v) = function
    | 0            -> u
    | n when s = 1 -> aux (-1, v, v + u) (n-1)
    | n            -> aux (1, v, v - u) (n-1)
  in aux (1, 1, 1) ;;
```

Exercice 2 Cette fonction réalise l'itération de la suite (x_n, y_n, z_n) définie par la valeur initiale $(0, a, b)$ et les relations :

$$x_{n+1} = x_n + \begin{cases} y_n & \text{si } z_n \text{ est pair} \\ 0 & \text{sinon} \end{cases}, \quad y_{n+1} = 2y_n, \quad z_{n+1} = \left\lfloor \frac{z_n}{2} \right\rfloor.$$

Nous avons clairement $y_n = 2^n a$ et, en considérant la décomposition $(b_p \cdots b_0)_2$ de b en base 2, $z_n = (b_p \cdots b_n)_2$. De ceci il résulte que cette fonction se termine et retourne la valeur de x_{p+1}.

Or on peut constater que $x_{n+1} = x_n + y_n b_n = x_n + a b_n 2^n$, et donc $x_{p+1} = x_0 + a \sum_{k=0}^{p} b_k 2^k = ab$, ce qui prouve que cet algorithme de multiplication est bien correct.

Sa version récursive terminale est :

```
let mult a b =
  let rec aux x y = function
    | 0 -> x
    | z -> aux (x + y * (z mod 2)) (2 * y) (z / 2)
  in aux 0 a b ;;
```

Exercice 3 Cette fonction réalise l'itération de la suite (y_n, z_n, u_n) définie par la valeur initiale $(0, 1, 0)$ et les relations :

$$y_{n+1} = y_n + z_n, \quad z_{n+1} = z_n + 2, \quad u_{n+1} = u_n + 1.$$

Nous avons immédiatement $z_n = 2n+1$, $u_n = n$ et donc $y_{n+1} = y_n + 2n + 1$, ce qui donne : $y_n = y_0 + \sum_{k=0}^{n-1} (2k+1) = n^2$.

La suite $(y_n)_{n\in\mathbb{N}}$ est strictement croissante, ce qui prouve la terminaison de la fonction, qui retourne donc le plus grand entier n pour lequel $n^2 \leqslant a$, à savoir $\lfloor \sqrt{a} \rfloor$ (la racine carrée entière de a).

Exercice 4 Cette fonction réalise l'itération de la suite (x_n, y_n, c_n, d_n) définie par la valeur initiale $(0, u, 1, 1)$ et les relations :

$$x_{n+1} = x_n + \begin{cases} d_n & \text{si } y_n + c_n \bmod 2 = 1 \\ 0 & \text{sinon} \end{cases}, \quad c_{n+1} = \begin{cases} c_n & \text{si } y_n \bmod 2 = 1 \\ 0 & \text{sinon} \end{cases}, \quad d_{n+1} = 2d_n, \quad y_{n+1} = \left\lfloor \frac{y_n}{2} \right\rfloor.$$

Clairement, $d_n = 2^n$ et $y_n = (u_p \cdots u_n)_2$, en ayant noté $u = (u_p \cdots u_0)_2$ la décomposition de u en base 2.
On a $y_n \bmod 2 = u_n$, donc $c_n = u_0 u_1 \cdots u_{n-1}$.
On peut donc affirmer que cette fonction calcule x_{p+1} s'il existe k tel que $u_k = 0$, et x_{p+2} si tous les u_k sont égaux à 1. Traitons alors deux cas.

– s'il existe $k \in [\![0, p]\!]$ tel que $u_k = 0$, considérons la plus petite de ces valeurs ; nous avons $c_n = \begin{cases} 1 & \text{si } n \leqslant k \\ 0 & \text{si } k > n \end{cases}$.

Ainsi, tant que $n < k$ nous avons $x_n = 0$, pour $n = k$ nous avons $x_k = 2^k$ et au delà : $x_n = 2^k + \sum_{j=k+1}^{n-1} u_j 2^j$.

Nous avons donc $u = \sum_{j=0}^{k-1} 2^j + \sum_{j=p}^{n} u_j 2^j$ et $x_{p+1} = 2^k + \sum_{j=k+1}^{p} u_j 2^j = u + 1$.

– Si tous les u_k sont égaux à 1, nous avons $x_n = 0$ pour $n \leqslant p + 1$ et $x_{p+2} = 2^{p+1} = u + 1$.

Dans les deux cas, on peut conclure : la fonction retourne la valeur de $u + 1$. Vous l'aurez sans doute compris, cette fonction singe l'addition en base 2. La référence **c** correspond à la retenue qui se propage tant que les bits de u lus par ordre croissant sont égaux à 1 :

$$\begin{array}{cccccccccccc} & & & & c_k & c_{k-1} & & \cdots\cdots & & c_1 & & \\ & u_p & \cdots\cdots & u_{k+1} & 0 & 1 & \cdots\cdots & & 1 & 1 \\ + & & & & & & & & & & 1 \\ \hline & u_p & \cdots\cdots & u_{k+1} & 1 & 0 & \cdots\cdots & & 0 & 0 \end{array}$$

Exercice 5 Pour la première fonction, on utilise un accumulateur qui mémorise le nombre d'occurrences de x déjà rencontrées :

```
let occurrence x =
  let rec aux acc = function
    | []              -> acc
    | t::q when t = x -> aux (acc + 1) q
    | _::q            -> aux acc q
  in aux 0 ;;
```

Pour la seconde fonction, l'accumulateur va mémoriser la somme des éléments déjà rencontrés, ainsi que leur nombre :

```
let moyenne =
  let rec aux (s, nb) = function
    | [] -> s /. (float_of_int nb)
    | t::q -> aux (s +. t, nb + 1) q
  in aux (0., 0) ;;
```

Pour la troisième fonction, l'accumulateur va transporter la liste $[\![k, q]\!]$ pour $k = q, q-1, \ldots, p$.

```
let intervalle p =
  let rec aux acc = function
    | q when q < p -> acc
    | q -> aux (q::acc) (q-1)
  in aux [] ;;
```

Complexité d'un algorithme

Exercice 6 Une solution possible consiste à effectuer un pas vers la droite, puis deux pas vers la gauche, quatre pas vers la droite, huit pas vers la gauche, etc.

Si on note u_p la position après $p+1$ étapes de la sorte, nous avons : $u_p = \sum_{k=0}^{p} (-1)^k 2^k = \frac{1}{3}\left(1-(-2)^{p+1}\right)$.

Le fait que $\lim u_{2p} = +\infty$ et $\lim u_{2p+1} = -\infty$ prouve la terminaison de l'algorithme.

Dans le même temps, le nombre de pas effectués après $p+1$ étapes est égal à $v_p = \sum_{k=0}^{p} 2^k = 2^{p+1} - 1$.

– Si la porte se trouve sur la droite, il existe un unique entier $p \in \mathbb{N}$ tel que $u_{2p-2} < n \leqslant u_{2p}$. Le nombre de pas p_n effectués avant de trouver la porte est alors égal à $v_{2p} - (u_{2p} - n) = \frac{4}{3}(4^p - 1) + n$.
  Sachant que $\frac{1}{3}(1+2^{2p-1}) < n \leqslant \frac{1}{3}(1+2^{2p+1}) \iff \frac{1}{2}(3n-1) \leqslant 4^p < 2(3n-1)$, on obtient : $3n-2 \leqslant p_n < 9n-4$.
– Si la porte se trouve sur la gauche, il existe un unique entier $p \in \mathbb{N}$ tel que $u_{2p+1} \leqslant -n < u_{2p-1}$. Le nombre de pas p_n effectués avant de trouver la porte est alors égal à $v_{2p+1} + (u_{2p+1} + n) = \frac{2}{3}(4^{p+1} - 1) + n$.
  Sachant que $\frac{1}{3}(1-2^{2p+2}) \leqslant -n < \frac{1}{3}(1-2^{2p}) \iff 3n+1 \leqslant 4^{p+1} < 4(3n+1)$, on obtient : $3n \leqslant p_n < 9n+2$.

Au final, on obtient l'encadrement : $3n-2 \leqslant p_n \leqslant 9n+1$, ce qui prouve que $p_n = \Theta(n)$.

Exercice 7 Puisqu'une star ne connait personne, il ne peut en exister deux dans un groupe. Il y a donc au plus une star.
On peut observer que si on pose à l'individu x la question « connaissez-vous y ? » alors :

– si la réponse est positive, x n'est pas une star ;
– si la réponse est négative, y n'est pas une star.

Chaque question permet ainsi d'éliminer un individu et de garder un candidat potentiel.
L'algorithme consiste alors à interroger ce candidat potentiel au sujet d'un individu non encore éliminé, et à réitérer le procédé. Ainsi, à l'aide de $n-1$ questions il ne restera plus qu'un candidat à la starification.
Il reste à lui poser $n-1$ questions pour vérifier qu'il ne connait personne, et $n-1$ questions pour vérifier que tout le monde le connait, ce qui donne en tout $3n-3$ questions au plus à poser (et au moins n).

Exercice 8

a) Si $k \geqslant \lceil \log n \rceil$, on utilise un algorithme de recherche dichotomique : faire sauter un étudiant de l'étage $p = \left\lceil \frac{n}{2} \right\rceil$, et suivant s'il survit ou pas poursuivre la recherche entre les étages 1 et $p-1$ ou entre les étages p et n.
On sait que le nombre de sauts nécessité par un tel algorithme est un $O(\log n)$, mais il faut vérifier qu'on dispose de suffisamment d'étudiants.
Le nombre maximal d'étudiants tués par cet algorithme vérifie la relation :

$$c_n = \max(1 + c_{p-1},\ c_{n-p+1}) = \max(1 + c_{\lceil n/2 \rceil - 1},\ c_{\lfloor n/2 \rfloor + 1}).$$

Montrons par récurrence que pour tout $n \geqslant 2$ on a : $c_n \leqslant \lceil \log n \rceil$:

– c'est le cas si $n = 2$ car $c_2 = 1$;
– si $n \geqslant 3$, on suppose le résultat acquis jusqu'au rang $n-1$. Alors :

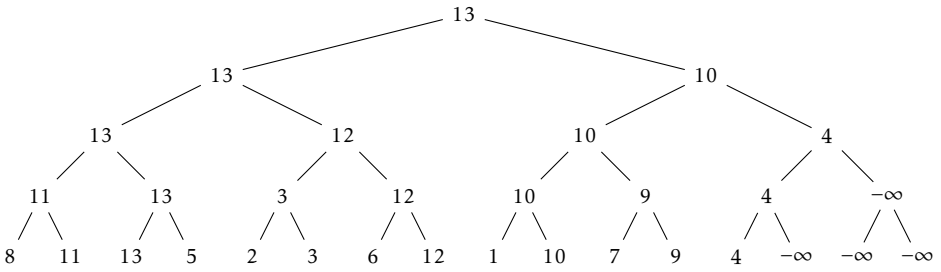
$$1 + c_{p-1} \leqslant 1 + \left\lceil \log\left(\left\lceil \frac{n}{2} \right\rceil - 1\right) \right\rceil = \left\lceil \log\left(2\left\lceil \frac{n}{2} \right\rceil - 2\right) \right\rceil \leqslant \lceil \log n \rceil \qquad \text{et} \qquad c_{n-p+1} \leqslant \left\lceil \log\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right) \right\rceil \leqslant \lceil \log n \rceil$$

donc dans tous les cas, $c_n \leqslant \lceil \log n \rceil$. On dispose bien d'un nombre suffisant d'étudiants.

b) Lorsque $k < \lceil \log n \rceil$, on ne dispose pas d'assez d'étudiants pour appliquer directement la méthode dichotomique. La solution consiste alors à appliquer la méthode dichotomique avec $k-1$ étudiants, puis, le cas échéant, à se servir du dernier étudiant pour une recherche linéaire (de bas en haut) dans l'intervalle restant.
Après la mort de $k-1$ étudiants, il reste un intervalle d'amplitude au plus $\frac{n}{2^{k-1}}$ étages à tester, le nombre de sauts dans le pire des cas est donc un $O\left(k + \frac{n}{2^{k-1}}\right)$.

c) Lorsqu'on ne possède que deux étudiants, la solution consiste à séparer en (approximativement) $\sqrt{n}$ intervalles d'amplitudes $\sqrt{n}$ étages, et à tester chaque 1er étage de chaque intervalle de bas en haut avec le premier étudiant. Lorsque ce dernier meurt, on connait un intervalle d'amplitude $\sqrt{n}$ dans lequel se trouve l'étage recherché. Une simple recherche linéaire avec le second étudiant permet de le trouver. En tout, le nombre maximal de sauts sera de l'ordre de $2\sqrt{n}$; c'est bien un $O(\sqrt{n})$.

Exercice 9 Complétons ce tableau en ajoutant la valeur $-\infty$ de manière à obtenir 2^p éléments, où $p = \lceil \log n \rceil$, et procédons à un tournoi à élimination directe :



On constate que le deuxième plus grand élément fait partie des adversaires malheureux du gagnant (dans l'exemple ci-dessus, 12 a été éliminé en demi-finale). Une fois le tournoi terminé, il suffit de déterminer le plus grand des p adversaires du vainqueur pour trouver le second, ce qui nécessite au plus $p-1$ comparaisons. Sachant que le tournoi a nécessité $n-1$ comparaisons (chaque match élimine un candidat), le nombre total de comparaisons à effectuer est égal à $n + p - 2 = n + \lceil \log n \rceil - 2$.

Exercice 10

a) Il est possible de résoudre le problème à n disques à l'aide de la démarche suivante :

- déplacer $n-k$ disques de la tige 1 à la tige 2 en suivant la démarche optimale ;
- déplacer les k disques restants de la tige 1 à la tige 4 en utilisant l'algorithme classique à trois tiges ;
- déplacer les $n-k$ disques de la tige 2 vers la tige 4 en suivant de nouveau la démarche optimale.

Cette démarche n'est peut-être pas optimale mais prouve la majoration : $t_n \leqslant 2t_{n-k} + 2^k - 1$.

En particulier, pour $k = p$ et $n = \dfrac{p(p+1)}{2}$, on obtient $t_{p(p+1)/2} \leqslant 2t_{p(p-1)/2} + 2^p - 1$.

b) Posons $u_p = \dfrac{t_{p(p+1)/2} - 1}{2^p}$. Alors $u_p \leqslant u_{p-1} + 1$, donc $u_p \leqslant p-1$, et $t_{p(p+1)/2} \leqslant 2^p(p-1) + 1$.

c) Pour résoudre le problème à n disques, on peut utiliser l'algorithme optimal à $n+1$ disques, en omettant le déplacement du plus gros des disques. On obtient ainsi un algorithme résolvant le problème à n disques en au pire $t_{n+1} - 1$ déplacements, ce qui prouve que $t_n \leqslant t_{n+1} - 1$. La suite $(t_n)_{n\in\mathbb{N}}$ est donc strictement croissante.

Pour tout entier $n \in \mathbb{N}$, il existe un unique entier $p \in \mathbb{N}$ tel que : $\dfrac{p(p-1)}{2} < n \leqslant \dfrac{p(p+1)}{2}$. On a alors : $t_n \leqslant t_{p(p+1)/2} \leqslant 2^p(p-1) + 1$. Sachant que $p \sim \sqrt{2n}$, on en déduit que $t_n = O\left(\sqrt{n}2^{\sqrt{2n}}\right)$.