Caml-TD2 : Quelques jeux

Paul Melotti (paul.melotti@ens.fr) Thomas Bourgeat

3 octobre 2013 - 10 octobre 2013

1 Nombres de Fibonacci et mémoïsation

Soit (F_n) la suite définie par $F_0 = 0$, $F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$

Q.1 Programmer une fonction récursive `fibo : int -> int` qui calcule F_n. Quelle est la complexité de votre fonction ? Est-ce optimal ?

Dans cette première partie nous allons proposer une manière un peu différente de faire de la récursivité : la *mémoïsation*.

Une table de hachage est un tableau de listes : `table : (a'*b') list array` ainsi qu'une fonction de hachage `h : 'a -> int`. Le but est de stocker des éléments de type `'b`, indexés par leurs clefs de type `'a`. Pour stocker (k, e) dans une table,on calcule le hachage $h(k)$, qui donne l'indice du tableau auquel on va insérer (k, e), en tête de liste. De même on peut effectuer une recherche ou une suppression en temps raisonnable. Toutes les manipulations portent sur les clefs et le hachage des clefs. L'idée de la mémoisation consiste à stocker les $(n, f(n))$ d'une fonction récursive au fur et à mesure qu'on les calcule. Ainsi si l'on a besoin plus tard d'un calcul déjà effectué il est inutile de le refaire : on va le chercher dans la table.

Q.2 Quelles sont les complexités de recherche, d'insertion, de suppression ? Quelle propriété faut-il sur h pour que notre table de hachage soit intéressante ?

Q.3 Implémenter une table de hachage `table : (int * int) list vect` de taille 17. On pourra prendre pour fonction de hachage `h : int -> int` la réduction de l'entier modulo 17. On programmera une fonction `ajoute : int * int -> unit` qui ajoute un élément à la table, et une fonction `recherche : int -> int` de recherche à partir de la clef, qui renvoie l'élément cherché s'il est dans la table, lève une exception sinon.

La mémoïsation consiste à faire de la récursivité améliorée : lorsqu'on appelle la fonction sur un élément on vérifie d'abord si on ne l'aurait pas déjà calculé, auquel cas on le renvoie immédiatement. Sinon, on le calcule, on le rajoute dans la table puis on le renvoie. Les éléments déjà calculés sont stockés dans une table de hachage.

Q.5 Implémenter une fonction `fibo_memo : int -> int` qui calcule Fibonacci en utilisant la mémoïsation.

Q.6 Quel est l'avantage par rapport à l'utilisation d'un grand tableau pour stocker directement les valeurs de la suite ?

2 Problème de Josèphe

2.1 Version historique

n personnes se réunissent en cercle dans une salle. Ils sont menacés par des ennemis à l'extérieur et décident de se suicider, pour sauver leur honneur, chacun leur tour. Pour ça ils choisissent un premier, et en partant de là comptent m personnes, et cette personne doit se suicider. Puis on recommence en partant de la personne qui suit la dernière personne morte. Le but est d'être le dernier survivant : certainement pour retourner sa veste et peut-être s'en sortir. Pour plus d'histoire on ira voir la page wikipédia de Flavius Josèphe.

Q.1 Proposer une manière de représenter le groupe de n personnes. Quel que soit le type choisi, on parlera par la suite de « liste circulaire », et ce type sera nommé `lc`.

Q.2 Écrire une fonction `convertit : 'a list -> 'a lc` qui transforme une liste en liste circulaire.

Q.3 Écrire une fonction `estpresent : 'a -> 'a lc -> bool` de recherche d'élément dans une liste circulaire.

Q.4 Écrire une fonction `supprime : int -> 'a lc -> 'a lc` de suppression d'élément dans une liste circulaire.

Q.5 En déduire une fonction `josephe : int -> int -> int` qui prend deux arguments n et m et détermine quelle est la position gagnante lorsqu'il y a n personnes dans le cercles et qu'on en tue une sur m.

2.2 Version de ma cour de récréation

La problématique, non moins sérieuse est légèrement différente : on souhaite désigner le loup [1] parmi un groupe d'enfants. Cette fois-ci on ne reprend pas à compter à partir du survivant qui suit le dernier mort, mais toujours en repartant du premier survivant qui suit le compteur. Si le compteur est éliminé, son premier successeur devient compteur. Et accessoirement on ne tue personne : c'est la version soft. Le dernier restant est le loup.

On numérote les n enfants de 0 à $n-1$, le compteur initial portant le numéro 0. On note $l(n, m)$ le loup élu pour n enfants avec une comptine de longueur m.

Q.1 Montrer que $l(n, m) = \begin{cases} 0 & \text{si } n = 1 \\ l(n-1, m) & \text{si } l(n-1, m) < m \mod n \\ l(n-1, m) + 1 & \text{sinon.} \end{cases}$

Q.2 Écrire une fonction `loup : int -> int -> int` qui désigne le loup par cette méthode. On pourra adapter les fonctions du problème de Josèphe, ou bien utiliser la question 1.

Q.3 Montrer que $l(n-1, n) = 0$ ssi n est premier.

Q.4 Ce jeu vous semble-t-il juste ? Y-a-t-il des positions préférentielles à n fixé ? On pourrait imaginer qu'on tire m aléatoire au début du jeu. Faire des expérience avec votre fonction.

3 Un jeu de Nim

3.1 Décomposition de Zeckendorf

Le mathématicien belge Édouard Zeckendorf a démontré en 1972 le théorème suivant :

Théorème 1 *Tout entier positif s'écrit de manière unique comme somme de termes de la suite de Fibonacci tels que deux termes ne sont jamais consécutifs. Autrement dit, pour tout* $N \geq 0$*, il existe une unique suite d'entiers* $c_0, \ldots, c_k$ *tels que* $c_0 \geq 2$, $c_{i+1} > c_i + 1$ *et*

$$N = \sum_{i=0}^{k} F_{c_i}$$

où F_n *est le* n*-ième terme de la suite de Fibonacci.*

On va écrire la fonction qui à un entier N associe cette décomposition.

Q.1 En admettant que si F est un terme de la suite de Fibonacci strictement positif, son indice est donné par la formule $n(F) = \left\lfloor \log_\phi \left(F\sqrt{5} + \frac{1}{2} \right) \right\rfloor$, écrire une fonction `max_fibo : int -> int` qui à un entier k associe le plus grand terme de la suite de Fibonacci qui lui est inférieur ou égal.

Ici, la mémoïzation est bien utile car on ne sait pas a priori jusqu'à quels termes on devra calculer la suite de Fibonacci.

1. Bien entendu tout le monde sait jouer au loup

Q.2 En déduire une fonction `zeckendorf : int -> int list` qui à un entier $k \geq 0$ associe la liste des termes de Fibonacci qui apparaissent dans sa décomposition.

Q.3 (Question mathématique) (*) Démontrer le théorème de Zeckendorf. Pour l'unicité, on pourra commencer par montrer que : la somme de termes de la suite de Fibonacci, distincts, non consécutifs, dont le plus grand élément est u_j, est strictement inférieure à u_{j+1}.

Q.4 (Question mathématique 2) Démontrer la formule de la question 1. On pourra se contenter de la démontrer pour les F assez grands.

3.2 Jeu de Fibonacci-Nim

Le jeu de Fibonacci-Nim est une variante du célèbre jeu des allumettes, dit jeu de Nim. On dispose de N allumettes. Le premier joueur peut prendre entre 1 et $N - 1$ allumettes. Ensuite, les joueurs pourront prendre entre 1 et $2p$ allumettes, où p est le nombre d'allumettes que vient de prendre l'adversaire. L'objectif est de prendre la dernière allumette.

On peut montrer qu'une stratégie gagnante consiste à prendre systématiquement le plus petit nombre de Fibonacci apparaissant dans la décomposition de Zeckendorf du nombre d'allumettes restantes. Si ce n'est pas possible, alors on est dans une position perdante (et on prend donc un nombre arbitraire d'allumettes, voué à la défaite puisque notre adversaire est un parfait mathématicien). Par contre, si on arrive à jouer ce coup une fois dans la partie, alors on peut montrer que ce coup sera valide pour tous les tours suivants, et nous mènera donc à la victoire.

Q.1 Écrire une fonction `coup_gagnant : int -> int -> int` qui étant donné le coup précédent p et le nombre n d'allumettes restantes, renvoie le nombre d'allumettes qu'il faut prendre pour espérer gagner. Par convention, si p vaut 0 on considèrera qu'on est le premier à jouer.

Q.2 En déduire une fonction `partie : int -> int list` qui étant donné un nombre initial d'allumettes N renvoie la liste des nombres d'allumettes obtenus dans une partie jouée entre deux mathématiciens parfaits, qui ont implémenté la fonction `zeckendorf` dans leur tête.

Q.3 (Question mathématique, le retour) (*) Montrer que cette stratégie est bien une stratégie gagnante. On supposera qu'on est le premier à jouer, et que le nombre initial d'allumettes N n'est pas un nombre de Fibonacci. On montrera que :

1. Si on joue ce coup une fois, alors l'adversaire ne peut pas gagner au tour suivant.
2. Si on joue ce coup une fois, alors soit on vient de gagner (N était un nombre de Fibonacci), soit on pourra rejouer ainsi au tour suivant. Il faudra pour cela distinguer les cas où la décomposition de N contient 2 termes ou plus…

4 Références

Le problème de Josèphe est très bien étudié dans le livre *Concrete Mathematics* de Ronald Graham, Donald Knuth et Oren Patashnik. La récurrence est résolue dans le cas $m = 2$ de plusieurs manières différentes.