Devoir Non Surveillé 1

À rendre le 4 novembre 2024

INFORMATIQUE MP2I

Tournez la page S.V.P.

Analyse d'une fonction

☐ 1 – On donne la fonction suivante :

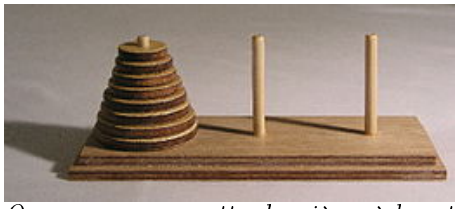
```
let rec fast_fib n =
  if n = 0 || n = 1 then
    1
  else
    fast_fib (n-2) + fast_fib (n-3);;
```

Que fait cette fonction ? étudier sa terminaison et sa correction.
On pourra dessiner un arbre d'appels, ou une partie d'arbre d'appels pour des valeurs bien choisies.

Réponse 1. Terminaison :
Si $n < 0$: `fast_fib n` ne termine pas : on a un chemin infini dans l'arbre des appels récursifs : $n, n-2, n-4, n-6, \ldots$.
Si $n = 0$ ou $n = 1$: `fast_fib n` termine directement et renvoie 1.
Si $n = 3$: `fast_fib n` termine et renvoie 2.
Si $n = 2$: `fast_fib n` appelle `fast_fib (-1)` et boucle donc à l'infinie. Si $n = 2 + 2k$ ou $n = 2 + 2k + 1$ avec $k > 0$: `fast_fib n` ne termine pas : on a un chemin infini dans l'arbre des appels récursifs : $n, n+2(k-1), n+2(k-2), \ldots, 2, -1, \ldots$.
Correction partielle :
`fast_fib 0` renvoie 1, mais dans la suite de Fibonacci, $f_0 = 0$. La fonction n'est donc pas partiellement correcte.

Tours de Hanoi

Les tours de Hanoï (originellement, la tour d'Hanoï) sont un jeu de réflexion imaginé par le mathématicien français Édouard Lucas, et consistant à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :



— *on ne peut déplacer plus d'un disque à la fois ;*
— *on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.*

On suppose que cette dernière règle est également respectée dans la configuration de départ. Wikipédia (Tours de Hanoï)
On dispose de 3 tours que l'on numérotera 0, 1, et 2.

☐ 2 – En remarquant que la somme des numéros des tours fait 3, étant donné 2 tours de numéros a et b, quelle expression désigne la 3e tour ?

Réponse 2. La 3e tour a le numéro $3 - a - b$.

Pour désigner le déplacement d'un disque depuis la tour a vers la tour b, on écrit `a->b`.

☐ 3 – On suppose que la position de départ contient 4 disques dans la tour 0 et que les tours 1 et 2 sont vides. Donner la position après la séquence de coups suivantes :

0–>1
0–>2
1–>0

Réponse 3. Si les disques sont du plus petit au plus grand : `a`, `b`, `c`, `d`. Alors) la fin on a :
```
0 : a c d
1 :
2 : b
```

Si on veut déplacer une pyramide depuis la tour 0 vers la tour 1, on peut remarquer que le plus grand disque (disons, le n-ième) doit être déplacé de la tour 0 vers la tour 2. Pour effectuer cette opération, il faut d'abord que les $n-1$ disques précédents se retrouvent sur la tour 1. On peut alors se ramener au problème de déplacer une pyramide de taille $n-1$, mais en échangeant les rôles des tours.

☐ 4 – Écrire une fonction `hanoi : int -> unit` qui prend en entrée un entier n, et affiche les coups nécessaires pour déplacer une pyramide de taille n de la tour 0 vers la tour 2, les tours 1 et 2 étant initialement vide.

Réponse 4.

```
let hanoi n =
  let rec hanoi_aux n a b =
    if n > 0 then
      let c = 3 - a - b in
        hanoi (n-1) a c;
        print_int a; print_string "->";
        print_int b; print_newline ();
        hanoi (n-1) c b
  in
  hanoi_aux n 0 2
```

En s'inspirant de ce code, nous allons déterminer le nombre de coups nécessaires (notons le x_n) pour déplacer une tour de n disques.

☐ 5 – Trouver une relation de récurrence liant x_{n+1} à x_n.

Réponse 5. En analysant le code, on voit qu'on affiche 1 ligne, et qu'on effectue 2 appels récursifs avec l'ordre $n-1$, on a donc la relation :
$x_{n+1} = 2x_n + 1$.
On a aussi le cas de base $x_0 = 0$.

☐ 6 – Calculer x_n.

Réponse 6. Il s'agit d'une suite arithmético-géométrique.
Le point fixe de $f(x) = 2x + 1$ est -1.
On pose $v_n = x_n + 1$.
On a $v_{n+1} = x_{n+1} + 1 = 2x_n + 2 = 2v_n$.
(v_n) est une suite géométrique de raison 2 et de premier terme $v_0 = x_0 + 1 = 1$.
$\forall n \in \mathbb{N}, v_n = 2^n, x_n = 2^n - 1$.

Rot13

on définit les opérateurs `^+` et `+^` permettant de concaténer un caractère à la fin et au début d'une chaîne de caractères :

Concaténation d'un char à la fin d'une string

```
# let (^+) str c = str ^ String.make 1 c;;
val ( ^+ ) : string -> char -> string = <fun>
# let (+^) c str = String.make 1 c ^ str;;
val ( +^ ) : char -> string -> string = <fun>
# ("Bonjour"^+'n')^+'o';;
- : string = "Bonjourno"
# 'a'+^('b'+^"cd");;
- : string = "abcd"
```

☐ 7 – Écrire une fonction `pos_of_letter : char -> int` qui prend en entrée une lettre minuscule et qui renvoie sa position dans l'alphabet. 0 pour `'a'`, et 25 pour `'z'`. On vérifiera que la précondition (le caractère donné représente une lettre minuscule) est bien vérifiée.

Réponse 7.

```
let pos_of_letter l =
  assert ('a' <= l && l <= 'z');
  int_of_char l - int_of_char 'a'
```

☐ 8 – Écrire une fonction `letter_of_pos : int -> char` qui effectue la transformation inverse. On vérifiera aussi la précondition nécessaire.

Réponse 8.

```
let letter_of_pos p =
  assert (0 <= p && p < 26);
  let code = p + int_of_char 'a' in
  char_of_int code
```

☐ 9 – Écrire une fonction `rot13_char : char -> char` qui prend en entrée une lettre minuscule et qui renvoie la lettre décalée de 13 positions dans l'alphabet, en revenant au début si nécessaire. `rot13 'a'` vaut `'k'` par exemple, et `rot13 'u'` vaut `'h'`.

Réponse 9.

```
let rot13_char c =
  let pos = pos_of_letter c in
  let pos = (pos + 13) mod 26 in
  letter_of_pos pos
```

☐ 10 – Écrire une fonction `rot13_string : string -> string` qui décale toutes les lettres minuscules apparaissant dans une chaîne de caractères de 13 positions.

Réponse 10.

```
let rot13_string str =
  let rec loop i =
    if i = String.length str then ""
    else rot13_char str.[i] +^ rot13_string str
```

On écrit le code suivant dans le fichier `rot13.ml` que l'on compile avec la commande

```
ocamlopt rot13.ml -o rot13
```

rot13.ml

```
(* définition de rot13_string de la question précédente *)

let rec rot13_input () =
  try
    print_endline (rot13_string (read_line ()));
    rot13_input ()
  with
    End_of_file -> ();;

let () = rot13_input ();;
```

☐ 11 – On exécute la commande

```
./rot13 | ./rot13 > montexte.txt
```

Et on saisi au clavier "helloworld".
Que se passe-t-il ?

Réponse 11. Le programme `rot13` est lancé 2 fois. La sortie standard de la première instance est redirigée vers l'entrée standard de la seconde (|), et celle de la seconde est redirigée vers `montexte.txt` (>).
L'entrée "helloworld" subit donc deux rotations successives, qui s'annulent et "helloworld" est écrit dans `montexte.txt`.

Numérotation Shadok

Les shadoks ne possèdent que 4 mots : "GA", "BU", "ZO" et "MEU".
Quand il n'y a pas de Shadoks, on dit "GA". Quand il y a un Shadok de plus, on dit "BU". Quand il y a encore un Shadok on dit "ZO". Et quand il y a encore un autre, on dit "MEU".
Si je met un Shadok en plus, évidemment je n'ai plus assez de mots pour les compter, alors c'est très simple : on les jette dans une poubelle et je dis que j'ai "BU" poubelles, et pour ne pas confondre avec le "BU", je dis qu'il n'y a pas de Shadok à côté de la poubelle et j'écris "BU" "GA".
Arrivé à "MEU" "MEU", si je mets un Shadok en plus il me faut une autre poubelle. Et comme je n'ai plus de mots pour compter les poubelles, je m'en débarasse en les jetant dans une grande poubelle j'écris "BU" grandes poubelles avec pas de poubelles et pas de Shadoks à côté : "BU" "GA" "GA". et on continue.

☐ 12 – Convertir le nombre "BU" "ZO" "GA" "MEU" en décimal.

Réponse 12. On a $\overline{1203}^4 = 3 \times 1 + 0 \times 4 + 2 \times 16 + 1 \times 64 = 99$

☐ 13 – Convertir le nombre 255 en numérotation Shadok.

Réponse 13. On a $256 = \overline{10000}^4$ donc $255 = \overline{3333}^4$, soit "MEU" "MEU" "MEU" "MEU".

☐ 14 – Écrire une fonction `shadok_of_int : int -> string` qui prend en entrée un entier et qui le converti en numérotation Shadok.

Réponse 14.

```
let rec shadok_of_int n =
  if n = 0 then "\"GA\""
  else if n = 1 then "\"BU\""
  else if n = 2 then "\"ZO\""
  else if n = 3 then "\"MEU\""
  else shadok_of_int (n / 4) ^ " " ^ shadok_of_int (n mod 4)
```

Egg throwing

On cherche à savoir à partir de quel étage N d'un certain immeuble les œufs se cassent. On suppose que tous les œufs se cassent à la même hauteur, et qu'un œuf non cassé peut être réutiliser.
Pour l'instant, on suppose que l'immeuble possède 100 étages.

☐ 15 – On dispose d'un seul œuf, quelle est la meilleure stratégie pour déterminer la valeur de N, et combien de lancers doit-on effectuer ?

Réponse 15. On est obligés de lancer l'œuf du premier étage, sinon on ne pourrait pas déterminer N, puis le second, le troisième, et ainsi de suite. Au total, il faut lancer l'oeuf N fois, donc 100 fois dans le pire des cas.

☐ 16 – On dispose d'une infinité d'œufs, quelle est la meilleure stratégie pour déterminer la valeur de N et combien de lancers doit-on effectuer ?

Réponse 16. La meilleure stratégie est la recherche dichotomique : si on cherche N entre a et b (initialement, entre 1 et 100), on lance l'œuf à l'étage $c = \lfloor (a+b)/2 \rfloor$. S'il casse, on cherche entre a et $c-1$, sinon on cherche entre $c+1$ et b. En suivant cette méthode, on peut tester n étages avec de l'ordre de $\log_2(n)$ lancers. Dans ce cas précis, 7 œufs suffisent pour trouver N même dans le pire des cas.

☐ 17 – On dispose de 2 œufs, quelle est la meilleure stratégie pour déterminer la valeur de N et combien de lancers doit-on effectuer ?

Réponse 17. Cette question est un cas particulier de la suivant permettant aux élèves d'y répondre partiellement. En appliquant le résultat de la question suivante, on obtient que 14 oœufs sont nécessaires et suffisants.

☐ 18 – Généraliser à tout immeuble et tout nombre d'œufs, et proposer un algorithme qui résoud le problème. On pourra s'intéresser au nombre maximum d'étages qui peuvent être testés en k lancers avec n œufs.

Réponse 18. Il est plus simple pour raisonner de considérer l'étage le plus haut que l'on puisse tester avec n lancers et k œufs : on a la relation de récurrence suivante :
$T_{n,k} = T_{n-1,k-1} + 1 + T_{n-1,k}$
En effet, si on lance à l'étage $T_{n-1,k-1}+1$ et que l'œuf casse, on pourra tester tous les étages précédents avec les $n-1$ lancers restants et les $k-1$ œufs restants, sinon on pourra aller $T_{n-1,k}$ étages plus loin.
Pour trouver le nombre de lancers optimal, il ne reste plus qu'à trouver le plus petit i tel que $T_{i,k}$ est supérieur ou égal au nombre d'étages à tester.

Petit problème

□ 19 – Une petite fille compte de 1 à 1 000 en utilisant les doigts de sa main gauche de la manière suivante : 1 pour le pouce, 2 pour l'index, 3 pour le majeur, 4 pour l'annulaire, 5 pour l'auriculaire. Puis elle change de direction : 6 pour l'annulaire, 7 pour le majeur, 8 pour l'index, et 9 pour le pouce. Puis 10 pour l'index, et ainsi de suite. Si elle continue ainsi, sur quel doigt va-t-elle s'arrêter ?

Réponse 19. Intéressons nous à la suite des doigts énumérés :
pouce, index, majeur, annulaire, auriculaire, annulaire, majeur, index, pouce, index, majeur, annulaire,...
Cette suite est 8-périodique. Le doigt en position $1\,001 = 125 \times 8 + 1$ est le même qu'en position 1 soit le pouce.
Le doigt en position 1 000 est le précédent, soit l'index.

Fin de l'épreuve