

Benjamin
LOISON
MPSI 1

Option informatique:

07/05/19

18 4e

2. let sumIndice L =

let bestIndice = ref 0 and s = ref 0 and nSum = ref 0 and f = ref (List.hd)

(et aux aux L i =

match L with

a :: b :: c :: d => nSum := (!nSum); f := a

in if (!nSum) > (!s) then (bestIndice := i; s := !nSum)

aux aux (b :: c :: d) (i + 1)

| a :: b :: c => nSum := (!nSum) - (!f) + a; f := a

in if (!nSum) > (!s) then (bestIndice := i; s := !nSum)

aux aux (b :: c) (i + 1)

| a :: b => nSum := (!nSum) - (!f) + a;

in if (!nSum) > (!s) then (i, !nSum) else (bestIndice, !s)

| _ -> (!bestIndice, !s)

in aux L 0;;

Code quelque peu répétitif, on procède par récursivité, le cas d'arrêt est lorsque l'on a une liste vide sinon en fonction de si il reste respectivement au moins 3, 2 et 1 élément dans la liste alors on calculer la somme partielle et on la compare à la plus grande somme et si la somme partielle est plus grande on réaffiche l'indice de la plus grande somme à l'indice actuel.

3. E est un ensemble fini tandis que N est infini, il existe donc deux noms distincts a et b tel que $\llbracket a \rrbracket = \llbracket b \rrbracket$.

A partir de là par réécriture immédiate on observe des valeurs cycliques.

2) Let $\text{mini } f \llbracket u_0 \rrbracket =$

let $i := \text{ref } 0$ and $w := \text{ref}(f \llbracket u_0 \rrbracket)$

while $\llbracket i \rrbracket \leq \llbracket u_0 \rrbracket$ do

 incr i ;

$w := f \llbracket w \rrbracket$;

done;

$(0, i);$ (* La complexité est linéaire *)

4) Attrez numDigits $n =$

if $n \geq 9$ then $(n \bmod 10) + \text{numDigits}(n/10)$
else $n;;$

Consigne non corrigée

let $\text{isParf } n = 2 * n = \text{numDigits } n;;$

let $L = \text{ref } []$ in

for $i := 0$ to 9999 do

 if $\text{isParf } i$ then $L := L @ [i]$ $L := i :: !L$

done;;

! $L;;$

5.1) let rec fibo 1 n a b =

match n with

0 → a

1 → b

l → fibo (n-1) + (fibo (n-2)),;

1/ On a pour tout $n \in \mathbb{N} \setminus \{0, 1\}$, $T(n) = T(n-1) + T(n-2)$

La complexité est en \mathcal{O}^n avec n la taille de l'entrée.

2) let fibo 2 n a b =

let tmp := ref 0 and ad := ref a and d := ref d and nv := ref (a + d)

for i = 0 to (n-3) do

tmp := ad + !d

(* on va de poser dans le for nv avant déjà *)

ad := !d

(* \mathbb{U}_2 , où la borne supérieure du for, on *)

d := nv

(* utilise une variable temporaire qui déclenche *)

nv := tmp

(* projeter à la permutation tant on déclenche *)

| done;
| nv;;

(* le terme suivant *)

il y a une raciale entre temps.

La complexité est linéaire en n .

3. a) On procède par récurrence.

Pour $n=0$, on a bien $\begin{pmatrix} a \\ b \end{pmatrix} = I_2 \begin{pmatrix} a \\ b \end{pmatrix}$

Pour $n=1$, on a bien $\begin{pmatrix} b \\ a+b \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$

On suppose l'égalité vraie pour $n \in \mathbb{N}$, on a alors :

$$\begin{pmatrix} u_{n+2} \\ u_{n+3} \end{pmatrix} = \begin{pmatrix} u_n + u_{n+1} \\ u_{n+1} + u_{n+2} \end{pmatrix} = \begin{pmatrix} u_n \\ u_{n+1} \end{pmatrix} + \begin{pmatrix} u_{n+1} \\ u_{n+2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n+1} \begin{pmatrix} a \\ b \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \left[\begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} b \\ a+b \end{pmatrix} \right] = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} a+b \\ a+2b \end{pmatrix}$$

$$\text{et on a: } \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{h+2} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^h \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

$$\text{ou } \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^h \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$$

$$\text{et } \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} a+b \\ a+2b \end{pmatrix}$$

D'où par récurrence on a la propriété sur N .

1) b) let mul (a, b, c, d) $(e, f, g, h) = (a*e + b*g, a*f + b*h, c*e + d*g, c*f + d*h);;$

c) let rec puis (a, b, c, d) $n =$
 $\text{if } n=0 \text{ then } (1, 0, 0, 1) \text{ else}$
 $\text{let } \begin{pmatrix} e, f \\ g, h \end{pmatrix} = \text{puis } (a, b, c, d) \text{ in } \text{mul } (\text{if } (n \bmod 2)=0 \text{ then } (1, 0, 0, 1) \text{ else}$
 $(a, b, c, d)) \text{ puis } (e, f, g, h) \text{ end}$

d) let fibo 3 n a $b =$

let $(e, f, g, h) = \text{puis } (0, 1, 1, 1) \text{ in } (g*a + h*b);;$
 La complexité est en $\log n$.

1. let h in $N =$

let $T = \text{Array.make } N 0$ in

$T.(0) \leftarrow 1$; let $fdi h = (\text{let } j=\text{ref } 0 \text{ in } h \text{ while not } (\text{found}) \& \& (!j) < N \text{ do}$

$\text{let } i = \text{ref } (j+1) \text{ in }$

$\text{while not } (\text{found}) \& \& (!i) < N \text{ do}$

$\text{if } (T.(j) \bmod h) = 0 \text{ then } \text{found} := \text{true};$

$\text{incr } i;$

$\text{done};$

$\text{found} := \text{not } (\text{found}); \text{ done};$

que c'est
compliqué.

code plus clair

Benjamin
LOISON
MPSI1

for $i = 1$ to N do

$T.(i) \leftarrow \min (\min(\text{min}(f(i), f(i+1)), (3 * T.(f(i-3)))) (2 * T.(f(i-2)))$

done;;

La complexité est polynomial en N^3 .

De manière impérative, on déclare une fonction faisant tourner le plus petit indice du tableau tel qu'il n'y est pas de $T.(i)$ dans celui-ci. La double boucle permet de faire une vérification exhaustive tout en permettant un arrêt brutal si un la multiple est trouvée afin de tester le terme $T.(i+1)$ de la même manière. Puis on affecte à l'indice suivant de T , le minimum des 3 $T.(i)$ précédentes.

2) (et min; i:2 f u o =

(et $L = \text{ref}(u)$) and $i = \text{ref} 0$ and $u = \text{eref}(f u)$

while not ($\text{list.mem } L | L$) do

$L := L @ [i]$; $u := !L$

inck i ;

$u := f[u]$;

and $j = \text{ref} 0$

done; (et $\text{len} = \text{list.length } L$ and $nf = f L.(\text{len}-1)$) in

for $i = 0$ to $(\text{len}-1)$ do

: if $L.(i) = nf$ && $j = 0$ then $j := i$;

done;

(j, len) ;;

Complexité quadratique

On aurait pu utiliser un while pour optimiser un peu du for.

On créer la liste contenant le début puis le premier cycle.