1 Définition et premier exemple

Un algorithme récursif est par définition un algorithme qui s'appelle lui-même. L'exemple le plus simple et le plus naturel d'un tel procédé est l'étude des suites définies par une relation de récurrence, par exemple $\begin{cases} u_{n+1} = u_n^2 \\ u_0 \end{cases}$. Le calcul de u_5 appelle le calcul de u_4 qui lui-même appelle le calcul de u_3 et ainsi de suite...pour remonter jusqu'à u_0. Il est donc nécessaire d'avoir une initialisation.
Comment écrit-on de façon récursive une procédure Maple qui calcule u_n ?
Nous avons déjà vu deux **procédures itératives** utilisant une boucle `for` permettant de le faire :

```
> #Version classique
> uiter1:=proc(n)
> local k,u ;
> u:=2;
> for k from 1 to n do
>   u:=evalf(u^2)
> od;
> RETURN(u);
> end:
```

```
> #Version avec des tables
> uiter2:=proc(n)
> local U,k ;
> U[0]:=2;
> for k from 1 to n do
>   U[k]:=evalf(U[k-1]^2)
> od;
> RETURN(U[n]);
> end:
```

Voici la **procédure récursive** :

```
> #Version récursive
> urec:=proc(n);
> if n=0 then RETURN(2)
>        else RETURN(urec(n-1)^2)
> fi;
> end:
```

L'avantage de la procédure récursive est qu'elle est plus naturelle à écrire et à lire. Elle est une fidèle réécriture de la définition itérative de la suite. L'inconvénient par rapport à la première méthode itérative c'est qu'elle nécessite le stockage de toutes les valeurs de u_n intermédiaires tandis que la procédure `uiter1` écrase à chaque tour de boucle la valeur précédemment calculée.

2 Suite récurrente double

Exemple classique : la suite de Fibonacci $\begin{cases} F_{n+2} = F_{n+1} + F_n \\ F_0 = 0 \quad F_1 = 1 \end{cases}$.

Voici pour rappel les deux procédures itératives :

```
> #Version classique
> fiboiter1:=proc(n)
> local f1,f2,temp,k;
> f1:=0;
> f2:=1;
> for k from 2 to n do
>   temp:=f2;  #sauvegarde temporaire de f2
>   f2:=f1+f2;
>   f1:=temp;
> od;
> RETURN(f2);
> end:
```

```
> #Version avec table
> fiboiter2:=proc(n)
> local F,k;
> F[0]:=0;
> F[1]:=1;
> for k from 2 to n do
>   F[k]:=F[k-1]+F[k-2]
> od;
> RETURN(F[n]);
> end:
```

Puis la procédure récursive :

```
> #Version récursive
> fiborec:=proc(n)
> if n=0 then RETURN(0)
>        elif n=1 then RETURN(1)
>        else RETURN(fiborec(n-1)+fiborec(n-2))
> fi;
> end:
```

Les avantages et les inconvénients évoqués dans la première partie sont toujours valables. Observons un autre in-

convénient : le temps de calcul à l'aide de la procédure récursive est plus long que pour les méthodes itératives. On l'observe précisément avec Maple à l'aide de la commande `time` :

```
> t1:=time():fiboiter1(25):time()-t1;
> t2:=time():fiboiter2(25):time()-t2;
> t3:=time():fiborec(25):time()-t3;
                                    0.
                                    0.
                                  1.359
```

Ceci est dû à une mauvaise gestion du stockage des valeurs déjà calculées. On corrige le problème en rajoutant l'instruction optionnelle `option remember` (qui ne s'utilise que pour l'écriture de procédures récursives) :

```
> #Version récursive avec option remember
> fiborec2:=proc(n) option remember;
> if n=0 then RETURN(0)
>        elif n=1 then RETURN(1)
>        else RETURN(fiborec2(n-1)+fiborec2(n-2))
> fi;
> end:
```

```
> t4:=time():fiborec2(25):time()-t4;
                                    0.
```

3 Exercices

1) Écrire une procédure récursive calculant $n!$ pour $n \in \mathbb{N}$.

2) En utilisant la relation $\binom{n}{p} = \binom{n-1}{p} + \binom{n-1}{p-1}$ vraie pour tout $(p,n) \in (\mathbb{N}^*)^2$, $p \leq n$. Écrire une procédure récursive calculant $\binom{n}{p}$ pour tout $(p,n) \in \mathbb{N}^2$.