Suites récurrentes

1 Récurrer le carrelage

On considère un rectangle $2 \times n$ qu'on veut paver avec des carreaux 2×1 et 1×2.

Question 0 Combien existe-t-il de pavages du rectangle $2 \times n$?

Soit p_n le nombre de pavages d'un rectangle de taille n.
On a $p_0 = 1$ et $p_1 = 1$.
Pour $n \geq 2$: Soit le pavage commence par un carreau vertical, suivi par un pavage $2 \times (n-1)$, soit le pavage commence par deux carreaux horizontaux, suivis par un pavage $2 \times (n-2)$. On a donc $p_n = p_{n-1} + p_{n-2}$.

2 Comptons les lapins

Question 1 Écrire une fonction récursive `pavages` qui calcule le nombre de pavages que vous avez défini.

```
pavages := proc(n)
   if n = 0 then
      1
   elif n = 1 then
      1
   else
      pavages(n - 1) + pavages(n - 2)
   end
end:

seq(pavages(k), k=0..10);
```

Question 2 Quelle est la complexité de votre fonction ?

Notons $c(n)$ le nombre d'appels récursifs pour l'évaluation de `pavages(n)`. Les cas $n = 0$ et $n = 1$ ont un coût unitaire, posons donc $c(0) = 1$ et $c(1) = 1$. Pour $n \geq 2$, $c(n) = c(n-1) + c(n-2)$, donc la complexité croît aussi vite que la suite calculée !
Pour info, un équivalent ce cette suite est $(\frac{1+\sqrt{5}}{2})^n$.

Question 3 Écrivez une fonction `fibonacci` plus efficace que `pavages`.

```
fibonacci := proc(n)
  local u, v, temp, i;
  if n = 0 then
     1
  else
     u := 1:
     v := 1 :
```

```
        for i from 2 to n do
           temp := u + v:
           u := v:
           v := temp:
        od:
        v
     end
  end:

  fibonacciRec := proc(n)
    local fib;
    fib := proc(u, v, n)
       if n = 0 then
          u
       elif n = 1 then
          v
       else
         fib(v, u+v, n-1)
       end:
    end:
    fib(1,1,n)
  end:

  seq(fibonacci(k), k=0..10);
  seq(fibonacciRec(k), k=0..10);
```

3 Votre fidèle compagnon : le calcul matriciel

La suite de Fibonacci est un cas particulier de **suite récurrente linéaire**. Une suite vérifiant une relation de la forme

$$u_{n+p} = a_{p-1}u_{n+p-1} + a_{p-2}u_{n+p-2} + \ldots + a_1 u_{n+1} + a_0 u_n$$

est dite **récurrente linéaire d'ordre p**.

Pour une telle suite, on définit une suite de vecteurs de $\mathbb{R}^p$ comme suit :

$$U_n = \begin{pmatrix} u_{n+p-1} \\ u_{n+p-2} \\ \vdots \\ u_{n+1} \\ u_n \end{pmatrix}$$

✾ **Question 4** Comment calculer U_{n+1} à partir de U_n ?

Il faut décaler les $p-1$ premiers coefficients du vecteur d'une position et calculer le nouveau premier coefficient comme combinaison linéaire des autres coefficients.

On veut l'exprimer sous la forme d'un calcul matriciel. On cherche donc une matrice A telle que $U_{n+1} = AU_n$.

$$A = \begin{pmatrix} a_{p-1} & a_{p-2} & \ldots & a_1 & a_0 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \end{pmatrix}$$

Question 5 Écrire une procédure qui calcule U_{n+k} en fonction de U_n et k.

```
compagnon := [[1 1], [1 0]]:

fiboVect := proc(u, k)
   evalm(compagnon^k &* u)
end:
```

Question 6 En déduire une nouvelle version de la fonction `fibonacci`. Quelle est sa complexité ?

```
fibo3 := n -> fiboVect(array(1..2, [1, 1]), n - 1)[1]:

seq(fibo3(k), k=0..10);
```

La complexité de `fibo3(n)` est linéaire en n.

4 Puissance et rapidité

Tout revient donc à calculer A^n efficacement. Peut-on faire mieux qu'en temps linéaire en n ? On peut remarquer que $A^{2n} = A^n \cdot A^n$.

Question 7 Dans le cas où $n = 2^k$, trouver un algorithme efficace pour calculer A^n.

```
puissanceDeux := proc(A, k)
   local B, i;
   B := evalm(A):
   for i from 1 to k do
       B := evalm(B &* B):
   od:
   evalm(B)
end:

puissanceDeux(compagnon, 5);
```

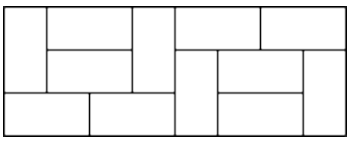
Question 8 Généraliser cette méthode pour un exposant quelconque.

5 Vous êtes donc vraiment rapide...

On peut généraliser la méthode de l'exponentiation rapide à d'autres problèmes.

Question 9 Quelle est la propriété du produit nécessaire pour que cela fonctionne ?

Il faut que le produit soit **associatif**. Sinon A^n n'est même pas défini...
On considère une variante de notre problème de pavage : le rectangle à paver est de taille $3 \times 2n$.



Question 10 Comptez le nombre de pavages pour cette variante.

On introduit des suites mutuellement récursives :
u_n = nombre de pavages qui finissent avec un bord droit, de longueur $2n$

v_n = nombre de pavages qui finissent avec un bord avec un trou 1×1 en bas, de taille $2n$ jusqu'au trou

w_n = nombre de pavages qui finissent avec un bord avec un trou 1×1 en haut, de longueur $2n$ jusqu'au trou

On a $v_n = w_n$ par symétrie.

On trouve :

$$u_0 = 1, v_0 = 0$$

$$u_n = 3u_{n-1} + 2v_{n-1}$$

$$v_n = u_{n-1} + v_{n-1}$$

Question 11 Comment adapter la méthode matricielle à ce cas ?

Introduire le vecteur

$$U_n = \begin{pmatrix} u_n \\ v_n \end{pmatrix}$$

Et la matrice

$$A = \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix}$$

Là encore il suffit de calculer $A^n U_0$.