Introduction à l'algorithmique - première partie -

Vasile-Marian SCUTURICI

Procédures et fonctions

Algorithme

- élaboré par une démarche descendante
- décomposer le problème en sous-problèmes, chaque sous-problème devant être de nouveau spécifié puis résolu
- →réutilisabilité
- →un algorithme peut être "appelé" dans le corps d'un autre algorithme
- →algorithme = procédure

Paramètres effectifs

- appel d'un algorithme :
 - préciser les valeurs des paramètres en entrée,
 - récupérer les valeurs des paramètres en sortie
- → paramètres effectifs.

Exemple $\sum_{i=1}^{n} \cos(i)$

1 Procédure somme-cos(n, sc)

Entrée : entier nSortie : entier scPrécondition : $n \ge 1$ Postcondition : $sc = \sum_{i=1}^{n} \cos(i)$

ou, autrement dit, $sc = cos(1) + cos(2) + cos(3) + \ldots + cos(n)$

1 Procédure cos(x, eps, cosx)

 ${\it Pr\'econdition} \ : \ eps > 0$

Postcondition : cosx = cos(x), avec une précision de eps

(utilise le développement en série : $cos(x) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}$)

Exemple

```
1 Procédure somme-cos(n,sc)
      ...  \begin{array}{ll} \mbox{D\'eclaration} & : \mbox{ r\'eel } i, cosi \\ \mbox{ const } eps = 0.0001 \end{array} 
début
```

Complexité:?

Exemple $\sum_{i=1}^{n} i!$

```
1 Procédure factorielle(n, f)
      Déclaration : entier i
      début

\begin{array}{c}
f \leftarrow 1 \\
i \leftarrow 1
\end{array}

         tant que i \leq n faire
```

Complexité:?

Exemple $\sum_{i=1}^{n} i!$

```
Procédure somme-fact(n, sf)
    Entrée : entier n
    Sortie : entier sf
    Précondition : n \ge 1
    Postcondition : sf = \sum_{i=1}^{n} i!
                    ou, autrement dit, sf = 1! + 2! + 3! + ... + n!
    Déclaration : entier i, fi
    début
     tant que i < n faire
// invariant: sf = \sum_{k=1}^{i} k!
i \leftarrow i+1
factorielle(i,fi)
sf \leftarrow sf + fi
                                                  Complexité:?
```

début

```
egin{aligned} sf \leftarrow 1 \ i \leftarrow 1 \ & tant \ que \ i < n \ faire \ & i \leftarrow i+1 \ & factorielle(i,fi) \ & sf \leftarrow sf + fi \end{aligned}
```

```
1 Procédure factorielle(n,f)
Déclaration : entier i
2 début
3 f \leftarrow 1
4 i \leftarrow 1
5 tant que i \leq n faire
6 f \leftarrow f * i
7 i \leftarrow i + 1
```

somme(3,x)		
n	3	
sf	?	
i	?	
fi	?	

```
1 Procédure factorielle(n,f)
Déclaration : entier i
2 début
3 f \leftarrow 1
4 i \leftarrow 1
5 tant que i \leq n faire
6 f \leftarrow f * i
7 i \leftarrow i + 1
```

somme(3,x)	
n	3
sf	1
i	?
fi	?

```
1 Procédure factorielle(n,f)
Déclaration : entier i
2 début
3 f \leftarrow 1
4 i \leftarrow 1
5 tant que i \leq n faire
6 f \leftarrow f * i
7 i \leftarrow i + 1
```

somme(3,x)		
n	3	
sf	1	
i	1	
fi	?	

```
1 Procédure factorielle(n,f)
Déclaration : entier i
2 début
3 f \leftarrow 1
4 i \leftarrow 1
5 tant que i \leq n faire
6 f \leftarrow f * i
7 i \leftarrow i + 1
```

somm	somme(3,x)	
n	3	
sf	1	
i	1	
fi	?	

```
1 Procédure factorielle(n,f)
Déclaration : entier i
2 début
3 f \leftarrow 1
4 i \leftarrow 1
5 tant que i \leq n faire
6 f \leftarrow f * i
7 i \leftarrow i + 1
```

somm	somme(3,x)	
n	3	
sf	1	
i	2	
fi	?	

1 P	1 Procédure factorielle (n, f)		
7.9	Déclaration : entier i		
2	début		
3	$f \leftarrow 1$		
4	$i \leftarrow 1$		
5	tant que $i \leq n$ faire		
6	$f \leftarrow f * i$		
7	$i \leftarrow i + 1$		

fact(2,fi)		somme(3,x)	
n	2	n	3
f	?	sf	1
i	?	i	2
		fi	?

1 P	1 Procédure factorielle (n, f)			
	Déclaration : entier i			
2	début			
3	$f \leftarrow 1$			
4	$i \leftarrow 1$			
5	tant que $i \leq n$ faire			
6	$f \leftarrow f * i$			
7	$i \leftarrow i + 1$			

fact(2,fi)		somme(3,x)	
n	2	n	3
f	1	sf	1
i	?	i	2
		fi	?

	1 Procédure factorielle (n, f)			
		Déclaration : entier i		
	2	début		
_	3	$f \leftarrow 1$		
	4	$i \leftarrow 1$		
	5	tant que $i \leq n$ faire		
	6	$f \leftarrow f * i$		
	7	$i \leftarrow i + 1$		
	I	_		

fact(2,fi)		somme(3,x)	
n	2	n	3
f	1	sf	1
i	1	i	2
		fi	?

entier i
n faire
$\cdot i$
1

fact(2,fi)		somme(3,x)	
n	2	n	3
f	1	sf	1
i	1	i	2
		fi	?

1 P	1 Procédure factorielle (n, f)			
	Déclaration :	entier i		
2	début			
3	$f \leftarrow 1$			
4	$i \leftarrow 1$			
5	tant que $i \leq$	n faire		
6	$f \leftarrow f *$	i		
7	$i \leftarrow i + i$	1		

fact(2,fi)		somme(3,x)	
n	2	n	3
f	1	sf	1
i	1	i	2
		fi	?

```
1 Procédure factorielle(n,f)
Déclaration : entier i
2 début
3 f \leftarrow 1
4 i \leftarrow 1
5 tant que i \leq n faire
6 f \leftarrow f * i
7 i \leftarrow i + 1
```

fact(2,fi)		somme(3,x)	
n	2	n	3
f	1	sf	1
i	2	i	2
		fi	?

1 P	${f roc\'edure}\ {\it factorielle}(n,f)$
	Déclaration : entier i
2	début
3	$f \leftarrow 1$
4	$i \leftarrow 1$
5	tant que $i \leq n$ faire
6	$f \leftarrow f * i$
7	$i \leftarrow i + 1$

fact(2,fi)		somme(3,x)	
n	2	n	3
f	1	sf	1
i	2	i	2
		fi	?

1 P	rocédure factorielle (n, f))
	Déclaration : entier	i
2	début	
3	$f \leftarrow 1$	
4	$i \leftarrow 1$	
5	tant que $i \leq n$ fai	re
6	$f \leftarrow f * i$	
7	$i \leftarrow i + 1$	

fact(2,fi)		somme(3,x)	
n	2	n	3
f	2	sf	1
i	2	i	2
		fi	?

1 Procédure factorielle (n, f)			
Déclaration : entier i			
2 début			
$f \leftarrow 1$			
4 $i \leftarrow 1$			
tant que $i \leq n$ faire			
$f \leftarrow f * i$			
7 $i \leftarrow i+1$			

fact(2,fi)		somme(3,x)	
n	2	n	3
f	2	sf	1
i	3	i	2
		fi	?

1 P	rocédure factorielle (n, f)
1	Déclaration : entier i
2	début
3	$f \leftarrow 1$
4	$i \leftarrow 1$
5	tant que $i \leq n$ faire
6	$f \leftarrow f * i$
7	$i \leftarrow i + 1$
I	<u> </u>

fact(2,fi)		somme(3,x)	
n	2	n	3
f	2	sf	1
i	3	i	2
		fi	?

1 P	rocédure factorielle(n, f)
2	début
3	$f \leftarrow 1$
4	$i \leftarrow 1$
5	tant que $i \leq n$ faire
6	$f \leftarrow f * i$
7	$i \leftarrow i+1$

fact(2,fi)		somme(3,x)	
n	2	n	3
f	2	sf	1
i	3	i	2
		fi	2

```
1 Procédure factorielle(n,f)
Déclaration : entier i
2 début
3 f \leftarrow 1
4 i \leftarrow 1
5 tant que i \leq n faire
6 f \leftarrow f * i
7 i \leftarrow i + 1
```

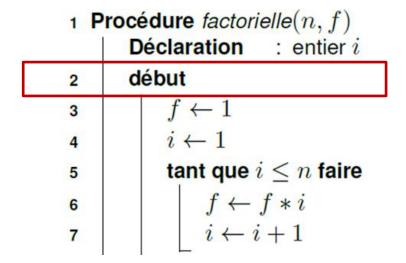
somm	somme(3,x)	
n	3	
sf	3	
i	2	
fi	2	

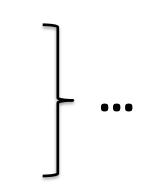
```
1 Procédure factorielle(n,f)
Déclaration : entier i
2 début
3 f \leftarrow 1
4 i \leftarrow 1
5 tant que i \leq n faire
6 f \leftarrow f * i
7 i \leftarrow i + 1
```

somme(3,x)	
n	3
sf	3
i	2
fi	2

```
1 Procédure factorielle(n,f)
Déclaration : entier i
2 début
3 f \leftarrow 1
4 i \leftarrow 1
5 tant que i \leq n faire
6 f \leftarrow f * i
7 i \leftarrow i + 1
```

soı	somme(3,x)	
n		3
sf		3
i		3
fi		2





fact(3,fi)		somme(3,x)	
n	3	n	3
f	?	sf	3
i	?	i	3
		fi	2

1 P	rocédure factorielle(n, f)
2	début
3	$f \leftarrow 1$
4	$i \leftarrow 1$
5	tant que $i \leq n$ faire
6	$f \leftarrow f * i$
7	$i \leftarrow i+1$

fact(3,fi)		somme(3,x)	
n	3	n	3
f	6	sf	3
i	4	i	3
		fi	2

1 P	${f roc\'edure}\ {\it factorielle}(n,f)$
	Déclaration : entier i
2	début
3	$f \leftarrow 1$
4	$i \leftarrow 1$
5	tant que $i \leq n$ faire
6	$f \leftarrow f * i$
7	$i \leftarrow i + 1$

fact(3,fi)		somme(3,x)	
n	3	n	3
f	6	sf	3
i	4	i	3
		fi	6

```
1 Procédure factorielle(n,f)
Déclaration : entier i
2 début
3 f \leftarrow 1
4 i \leftarrow 1
5 tant que i \leq n faire
6 f \leftarrow f * i
7 i \leftarrow i + 1
```

somme(3,x)	
n	3
sf	9
i	3
fi	6

```
1 Procédure factorielle(n,f)
Déclaration : entier i
2 début
3 f \leftarrow 1
4 i \leftarrow 1
5 tant que i \leq n faire
6 f \leftarrow f * i
7 i \leftarrow i + 1
```

somme(3,x)	
n	3
sf	9
i	3
fi	6

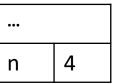
?

$\sum_{i=1}^{n} i!$ — version plus performante

```
Procédure somme-fact(n, sf)
   Entrée : entier n
   Sortie : entier sf
   Précondition : n \ge 1
   Postcondition : sf = \sum_{i=1}^{n} i!
              ou, autrement dit, sf = 1! + 2! + 3! + ... + n!
   Déclaration : entier i, f
   début
Complexité:?
```

Paramètres en entrée et en sortie

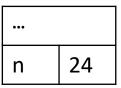
$$n \leftarrow 4$$
$$factorielle(n, n)$$



Paramètres en entrée et en sortie

$$n \leftarrow 4$$

$$factorielle(n, n)$$



```
Procédure nom-algo(var)
\bot Entrée/Sortie : type-de-var var
var^{in}
var^{out}
```

Paramètres en entrée et en sortie

```
Procédure échanger(a,b)

Entrée/Sortie : entier a,b

Postcondition : a^{out} = b^{in}

b^{out} = a^{in}

Déclaration : entier aux

début
aux \leftarrow a
a \leftarrow b
b \leftarrow aux
```

Fonction

 Fonction : procédure dont le but est de calculer une seule valeur.

```
Fonction type-fct nom-fct(< paramètres >)
```

Entrée : liste des paramètres (types et noms)

Précondition : Conditions sur les paramètres en entrée

Postcondition : relation entre la valeur retournée par la fonction et ses paramètres en entrée

 Dans le corps de la fonction, la sortie est spécifiée par :

retourner expr

Fonction - exemple

```
Fonction entier factorielle(n)
    Entrée : entier n
    Précondition : n \ge 0
    Postcondition: retourne n!
    Déclaration : entier i, f
    début
       pour (i \leftarrow 1 ; i \leq n ; i \leftarrow i+1) faire
   // invariant: f = (i-1)!
f \leftarrow f * i
        // Nombre de passages = n; i = n + 1 et f = (i - 1)! = n!
        retourner f
                  x \leftarrow \text{factorielle}(4)
                                                               x = ? y = ?
```

 $y \leftarrow \text{factorielle}(x - 3 * \text{factorielle}(3))$

Passage de paramètres

début

```
sf \leftarrow 1
i \leftarrow 1
tant \ que \ i < n \ faire
i \leftarrow i + 1
fi \leftarrow i
factorielle(fi)
sf \leftarrow sf + fi
```

si factorielle(n)

Procédure factorielle(n)

```
Entrée/Sortie : entier n
Déclaration : entier i, f
début
f \leftarrow 1
\text{pour } (i \leftarrow 1; i \leq n; i \leftarrow i+1) \text{ faire}
| f \leftarrow f * i
```

somme-fact(3, x)

fact(3,fi)		somme(3,x)	
n	_	n	3
f	?	5f	3
i	?	i	3
	-	fi	2

Récursivité

- Stratégie de résolution de problèmes algorithmiques où un problème est divisé en sous-problèmes de même type
- Divide et impera (diviser et conquérir)

Exemple - factorielle

```
n! = 1 * 2 * \dots * (n-1) * n
fact(n) = fact(n-1) * n
                 Fonction Fact(n)
                        Entrée : entier n
                        Sortie : entier factoriel
                       Précondition : n \ge 0
                       Postcondition : factoriel = n!
                       début
                    \begin{array}{|c|c|} & \mathbf{si} \; n = 0 \; \mathbf{alors} \\ & factoriel \leftarrow 1 \\ & \mathbf{sinon} \\ & factoriel \leftarrow n * Fact(n-1) \\ & \mathbf{retourne} \; factoriel \end{array}
```

Exemple - Fibonacci

```
Fonction entier fibo(n)
    Entrée
                    : entier n
    Précondition : n \ge 0
    Postcondition: retourne fibo(n)
                      où fibo est la suite définie récursivement par :
                      fibo(0) = 1
                      fibo(1) = 1
                      fibo(n) = fibo(n-1) + fibo(n-2), \forall n \geq 2
    début
        \mathbf{si} \; n \leq 1 \; \mathbf{alors}
            retourner 1
            retourner fibo_rec(n-1)+fibo_rec(n-2)
```

Complexité: $\Theta(2^n)$

Exemple – x^y

```
Fonction: entier puissance_rec(x, y)
Entrées :
    reel x
    entier y
Précondition :
    y \ge 0
Postcondition:
    retourne x^y
début
   \mathbf{si}\ y = 0\ \mathbf{alors}
   retourner 1
   sinon
    retourner x * puissance rec(x, y - 1)
   finsi
                     Complexité:?
```

Exemple – x^y

```
x^{n+n} = x^n * x^nx^{n+n+1} = x^n * x^n * x
```

Déclarations:

entier y

reel x

Entrées :

entier p

 $\mathbf{si} \ y = 0 \ \mathbf{alors}$

début

```
retourner 1
sinon

p \leftarrow puissance\_rec\_opt(x, y \text{ div } 2)
si p \mod 2 = 0 alors
   | retourner p * p
sinon
   | retourner p * p * x
finsi
finsi
```

Fonction: entire puissance_rec_opt(x, y)

Complexité:?

 $\Theta(\log_2(y))$

fir

Exemple – recherche dichotomique récursive

```
Fonction : entier cherche_dicho_rec(tab, d, f, e)
Entrées :

Telt[?..?] tab
entier d, f
Telt e

Précondition :

\forall i \in [d..f-1], \ tab[i] \leq tab[i+1]

Postcondition :

si \exists i \in d..f tel que tab[i] = e alors retourne i sinon retourne -1
```

Exemple – recherche dichotomique récursive

```
Fonction: entire cherche_dicho_rec(tab, d, f, e)
Déclarations :
    entier milieu
début
                                                 Complexité:?
   \mathbf{si} \ d > f \ \mathbf{alors}
       retourner -1
   sinon
                                                  \Theta(\log_2(f-d+1))
       milieu \leftarrow (d+f)/2
       \mathbf{si} \ tab[milieu] = e \ \mathbf{alors}
           retourner (milieu)
       sinon si tab[milieu] < e alors
           retourner cherche\_dicho(tab, milieu + 1, f, e)
       sinon
           retourner cherche dicho(tab, d, milieu - 1, e)
       finsi
   finsi
```

46

Récursivité

- Avantage
 - Elégance et compréhensibilité du code
- Désavantage
 - Chaque appel récursif peut nécessiter des ressources supplémentaires (mémoire + processeur)

To iterate is human, to recurse, divine.

L. Peter Deutsch

?

Exercice 1

- Proposez un algorithme pour multiplier deux matrices A et B
 - réel A[0..n-1, 0..m-1]
 - réel B[0..m-1, 0..p-1]

Exercice 2

Proposer un algorithme pour multiplier n matrices