Médian – LO42
solution

Les documents ne sont pas autorisés (La copie ou les idées du voisin non plus). Le barème est indicatif. Le soin donné à la rédaction sera évalué. Toute réponse devra être claire et justifiée (toute ambiguïté sera mal interprétée). L'élégance de la solution sera jugée.
Sauf indication contraire, dans le cas d'algorithmes les réponses doivent être rédigées en pseudo code.

1) Ah que je ris … ! (3)

Voici un algorithme permettant de créer une image miroir d'un vecteur.

```
Var     i, sauve : entier ;
        V : tableau[1…n] de élément ;
Début
        i ← 1 ;
        Tantque (i<n/2) faire
                sauve ← V[i] ;
                V[i] ← V[n+1-i] ;
                V[n+1- i] ← sauve ;
                i ← i + 1 ;
        Ftq ;
Fin ;
```

et deux conditions :

$\{ 0 < n \wedge (\forall j \in [\ 1 \ldots n\ ] , V[j] = V_0[n+1-j]\ ) \}$
$\{ 0 < n \wedge (\forall j \in [\ 1 \ldots n\ ] , V[j] = V_0[j]) \}$

a) Sachant que V_0 est l'état initial du vecteur, précisez la précondition et la postcondition de la boucle.

b) Définissez l'invariant de la boucle.

Solution :

Precondition (2)
Postcondition (1)

Invariant :

$\{ 0 \leq i \leq n/2 \wedge (\forall j \in [\ 1 \ldots i\ [\ , V[j] = V_0[n+1-j] \wedge V[n+1-j] = V_0[j]\ )$
$\wedge (\forall j \in [\ i \ldots n+1-i\ ] , V[j] = V_0[j]) \}$

La boucle est une manipulation du tableau donc un invariant significatif doit porter sur le tableau.
Cet invariant doit être vrai en début et en fin de chaque exécution de la boucle. Ainsi, il est compatible avec la pré-condition et la post-condition. La condition de boucle intervient sur la définition des différents domaines de réalisation des conditions.

2) Multiplication égyptienne (8)

L'algorithme égyptien qui calcule le produit de deux entiers a_0 et b_0 n'utilise que la somme, le produit par 2 et la division par 2 :

```
A ← a0
B ← b0
S ← 0
Tant que B ≠ 0 faire
        Si B impair alors S ← S+A
        B ← B÷2
        A ← 2*A
ftq
résultat est S
```

Le symbole « division » désigne ici la division entière.

a) Calculez 968 x 21 et 35 x 27 en utilisant cet algorithme.

b) A partir de ces exemples, exprimez une relation entre A, B, S et a_0 , b_0 . On pourra ajouter des indices à A, B et S.

c) Montrez par récurrence que $A_n * B_n + S_n = a_0 * b_0$. Cette relation est un « invariant de boucle ». En déduire que l'algorithme écrit ci-dessus calcule bien le produit de a_0 par b_0.

d) Calculez en fonction de b_0 le nombre d'itérations nécessaires à l'algorithme égyptien pour le calcul d'un produit.

e) Un algorithme simple calculant le produit de 2 entiers est aussi :
S ← 0
Pour I de 1 à b_0 faire S ← S + a_0
résultat est S

Lequel de ces 2 algorithmes consomme le moins de boucles ?

f) Ecrivez une version récursive de l'algorithme de multiplication égyptienne.

Solution :

a) le calcul donne

A	B	S
968	21	0
1936	10	968
3872	5	968
7744	2	4840
15488	1	4840
30976	0	20328

A	B	S
35	27	0
70	13	35
140	6	105
280	3	105
560	1	385
1120	0	945

b) Nous avons

$A_0 = a_0$ $A_i = A_{i-1} * 2$
$B_0 = b_0$ $B_i = B_{i-1}$ Div 2

Donc $A_i = 2^i a_0$ et $B_i = b_0$ div 2^i

En outre les différents calculs nous permettent de dire que

$A_0 * B_0 + S_0 = a_0 * b_0$
$A_1 * B_1 + S_1 = a_0 * b_0$
$A_2 * B_2 + S_2 = a_0 * b_0$
$A_3 * B_3 + S_3 = a_0 * b_0$

c) Démonstration de la relation

Etape initiale : on vérifie pour i = 0

$A_0 = a_0$, $B_0 = b_0$ et $S_0 = 0$ par initialisation donc $A_0 * B_0 + S_0 = a_0 * b_0$

On pose l'hypothèse de récurrence : $A_n * B_n + S_n = a_0 * b_0$

Vérifions à l'étape n+1

$A_{n+1} * B_{n+1} + S_{n+1} = a_0 * b_0$?

$A_{n+1} * B_{n+1} + S_{n+1} = A_n * 2 * B_n$ Div 2 $+ S_{n+1}$

Le mode de détermination des valeurs de B_{n+1} et S_{n+1} dépend de la valeur de B_n, plus précisément de sa parité. Etudions donc les deux cas possibles :

- B_n est paire
  $A_{n+1} = A_n * 2$
  $B_{n+1} = B_n$ Div 2 $= B_n / 2$ il y a égalité entre les deux expressions car B_n est multiple de 2
  $S_{n+1} = S_n$
  Donc $A_{n+1} * B_{n+1} + S_{n+1} = A_n * 2 * B_n / 2 + S_{n+1} = A_n * B_n + S_{n+1} = A_n * B_n + S_n = a_0 * b_0$
- B_n est impaire
  $A_{n+1} = A_n * 2$
  $B_{n+1} = B_n$ Div 2 $= (B_n - 1)/ 2$
  $S_{n+1} = S_n + A_n$
  Donc $A_{n+1} * B_{n+1} + S_{n+1} = A_n * 2 * (B_n - 1)/ 2 + S_{n+1} = A_n * (B_n - 1) + S_{n+1}$
  $= A_n * B_n - A_n + S_{n+1} = A_n * B_n - A_n + S_n + A_n = A_n * B_n + S_n = a_0 * b_0$

Terminaison

Le condition de continuation de la boucle est $B > 0$
La boucle du programme définit une suite de valeurs B_i telles que $B_i = B_{i-1}$ Div 2.
La suite est décroissante et la progression étant la division entière la suite.
La suite est donc finie avec $B_n = 0$

La relation $A_n * B_n + S_n = a_0 * b_0$ étant vraie nous obtenons = $A_n * 0 + S_n = a_0 * b_0$
Donc $S_n = a_0 * b_0$

d) Calcul de la complexité

$T(b_0) = T(B_0) = T(B_1) + 1 = \ldots = T(B_n) + n$
Selon la progression B_i
$T(b_0) = T(B_0) = T(b_0 \text{ Div } 2) + 1 = T(b_0 \text{ Div } 2^n) + n$
D'après la condition de boucle
$T(0) = 0$
Donc si $B_n = 0$ alors $T(b_0) = n$

Soit k tel que $2^k \leq b_0 < 2^{k+1}$
La suite $b_0, \ldots b_0 \text{ Div } 2^i \ldots 1$ des valeurs à l'entrée de la boucle a k + 1 éléments

Le nombre d'itérations est donc $\log_2 (b_0) + 1$

La complexité est $O(\log_2 b_0)$

e) L 'algorithme simple est en complexité $O(b_0)$
$\text{Log}_2 b_0 < b_0$ donc le meilleur est l'algorithme égyptien

f)
```
Fonction mult (A, B : entier) : entier
Var R : entier
Début
        Si B = 0 Alors mult ← 0
        Sinon
                R ← mult(A*2, B ÷ 2) ;
                Si B mod 2 = 0 Alors mult ← R ;
                Sinon mult ← R + A ;
                Fsi
        Fsi
Fin ;
```

3) La puissance soit avec toi ! (5)

Pour calculer la puissance n-ième d'un nombre x on considère les trois méthodes suivantes :

$$x^n = x^{n-1} * x$$

$$x^n = x^{n \text{ div } 2} * x^{n \text{ div } 2} * x^{n \bmod 2}$$

$$x^n = (x^{n \text{ div } 2})^2 * x^{n \bmod 2}$$

a) Ecrivez trois fonctions implémentant ces formules pour calculer la puissance n-ième de x.

b) Quelle est la complexité de chaque méthode ? Si n = 20, puis 200, puis 2000, combien de multiplications effectue-t-on en appliquant la première, la seconde et la troisième méthode ?

c) Quelle méthode est la meilleure ?

Solution :

a) Première version

```
Fonction puis1(r: réel ; n : entier) : réel ;
Début
   Si n > 0 Alors puis1 ← puis1 (r , n-1) * r
   Sinon puis1 ← 1
   Fsi
Fin ;
```

```
Fonction puis1(r: réel ; n : entier) : réel ;
Var R : Réel
Début
   R ← 1
   Pour n de n à 1 Faire
            R ← R * r
   Finpour
```

	Puis1 ← R ; Fin ;

Deuxième version

La formule $x^{n \bmod 2}$ est utilisée pour traiter la parité. On ne fait donc pas appel à la fonction mais on introduit un test.

```
Fonction puis2(r: réel ; n : entier) : réel ;
Début
  Si n > 0 Alors
        Si n = 1 Alors puis2 ← r
        Sinon   Si n pair Alors puis2 ← puis2 (r , n Div 2) * puis2 (r , n Div 2);
                Sinon puis2 ← puis2 (r , n Div 2) * puis2 (r , n Div 2) * r ;
                Finsi ;
        Fsi ;
  Sinon puis2 ← 1 ;
  Fsi
Fin ;

Fonction puis3(r: réel ; n : entier) : réel ;
Début
  Si n > 0 Alors
        Si n = 1 Alors puis3 ← r
        Sinon   Si n pair Alors Puis3 ← carré(puis3 (r , n Div 2));
                Sinon Puis3 ← carré(puis3 (r , n Div 2)) * r ;
                Fsi ;
        Fsi ;
  Sinon puis3 ← 1 ;
  Fsi
Fin ;
```

Si on ne peut pas utiliser la fonction carré, il suffit de stocker le résultat de l'appel à puis3 et de réaliser une multiplication par la suite

```
        Sinon   R ← puis3 (r , n Div 2);
                R ← R * R :
                Si n pair Alors Puis3 ← R;
                Sinon Puis3 ← R * r ;
                Fsi ;
        Fsi ;
```

b) Complexité

	N = 20	N = 200	N = 2000
Puis1	20	200	2000
Puis2	19	199	1999
Puis3	5	9	15

Les calculs du nombre de multiplications pour puis2 et puis3 n'est pas immédiat :

Pour puis2 on peut calculer en développant :

$T(n) = T(n / 2) * 2 + 1$ (si n est paire + 2 si n est impaire)

En sachant que $T(1) = 0$

$T(200) = 2*T(100) + 1 = 2*(2*T(50) + 1) + 1 = \ldots$

Ou

$T(200) = 2*T(100) + 1 = 4*T(50) + 3 = 8*T(25) + 7 = 16 * T(12) + 23 = \ldots$

Pour puis3 on peut calculer en développant de même:

$T(n) = T(n / 2) + 1$ (si n est paire + 2 si n est impaire)

En sachant que $T(1) = 0$

$T(200) = T(100) + 1 = (T(50) +1) + 1 = \ldots$

Ou

$T(200) = T(100) + 1 = T(50) + 2 = T(25) + 3 = T(12) + 5 = T(6) + 6 = T(3) + 7 = T(1) + 9 = 9$

La complexité de la première est O(n)

La seconde fonction est en O(n)

Pour la troisième la complexité est de O(log(n))

C'est donc la troisième qui est la meilleure.

Attention si on retire le test n=1, le nombre de multiplication change principalement dans la deuxième version. En effet on introduit alors des multiplications par 1 inutiles.

```
Fonction puis2(r: réel ; n : entier) : réel ;
Début
  Si n > 0 Alors
        Si n pair Alors puis2 ← puis2 (r , n Div 2) * puis2 (r , n Div 2);
        Sinon puis2 ← puis2 (r , n Div 2) * puis2 (r , n Div 2) * R ;
        Fsi ;
  Sinon puis2 ← 1 ;
  Fsi
Fin ;
```

troisième version

```
Fonction puis3(r: réel ; n : entier) : réel ;
Début
  Si n > 0 Alors
        Si n pair Alors Puis3 ← carré(puis3 (r , n Div 2));
        Sinon Puis3 ← carré(puis3 (r , n Div 2)) * r ;
        Fsi ;
  Sinon puis3 ← 1 ;
  Fsi
Fin ;
```

	N = 20	N = 200	N = 2000
Puis2	51	455	4047
Puis3	6	10	16

4) La vedette américaine (4)

Voici des modèles de procédures récursives

```
Procédure P(x) ;
Début
        Ax ;
        Si Cx Alors
                Ex ;
                P(fx) ;
        Sinon
                Sx ;
        Fsi ;
Fin ;
```

```
Procédure P(x) ;
Début
        Ax ;
        Si Cx Alors
                Ex ;
                P(fx) ;
                Rx ;
        Sinon
                Sx ;
        Fsi ;
Fin ;
```

Donnez les schémas de procédures itératives équivalentes

Solution :

```
Procédure P(x) ;
Début
        Ax ;
        Tantque Cx Faire
                Ex
                x ← fx ;
                Ax ;
        Fait
        Sx ;
Fin ;
```

```
Procédure P(x) ;
Début
        Ax ;
        Tantque Cx Faire
                Ex
                Empiler(x) ;
                x ← fx ;
                Ax ;
        Ftq ;
        Sx ;
        Tantque non pilevide() Faire
                Dépiler(x)
                Rx ;
        Ftq ;
Fin ;
```

Solution possible si C_x n'a pas d'effet de bord

```
Procédure P(x) ;
Début
        Répéter
                A_x ;
                Si C_x Alors
                        E_x
                        x ← f_x ;
                Fsi
        Jusque Non C_x ;
        S_x ;
Fin ;
```

```
Procédure P(x) ;
Début
        Répéter
                A_x ;
                Si C_x Alors
                        E_x
                        Empiler(x) ;
                        x ← f_x ;
                Fsi ;
        Jusque Non C_x ;
        S_x ;
        Tantque non pilevide() Faire
                Dépiler(x)
                Rx ;
        Ftq ;
Fin ;
```