Algorithmes et structures de données : TD 4 Corrigé

Types - Enregistrements - Temps d'un algorithme T(n)

Exercice 4.1 *Types*

Déclarer des types qui permettent de stocker :
1. Un joueur de basket characterisé par son nom, sa date de naissance, sa nationalité, et son sexe

```
type t_sexe = (masculin, feminin);
type t_joueur = RECORD
                        nom     :       string;
                        nation  :       string;
                        sexe    :       t_sexe;
                END;
```

2. Une association de joueurs de basket

```
type t_assoc = array[1..N] of t_joueur;
```

Remarque : Pour permettre à l'association de croitre, c'est-à-dire avoir plus de N joueurs, il faudra allouer un tableau dynamique ou une liste...
3. Une equipe de basket avec son nom, ses joueurs, ainsi que les points gagnés et les basket marqués et encaissés dans la saison courante.

```
type t_equipe = RECORD
                        nom             :       string;
                        joueurs         :       array[1..N] of t_joueur;
                        points          :       integer;
                        basketmarque    :       integer;
                        basketencaisse  :       integer;
                END;
```

Remarque : Pour les joueurs, il vaudrait mieux allouer un tableau dynamique ou une liste...
4. Un tableau de 18 equipes de basket

```
type t_tableau = array[1..18] of t_equipe;
```

Exercice 4.2 *Je constate que la somme des n premiers nombres impairs est égale à n^2, c'est à dire que $1 + 3 + 5 + ... + (2n - 1) = n^2$.*

1. Ecrivez la `function sommeImpairs(n : integer) : integer;` qui calcule la somme des n nombres impairs.

```
function sommeImpairs(n : integer) : integer;
var i, somme : integer;
begin
  somme := 0;
  for i := 1 to n do
  begin
      somme := somme + (2*i)-1;
  end;
  result := somme;
end;
```

2. Ecrivez la `function carre(n : integer) : integer;` qui calcule n^2

```
function carre(n : integer) : integer;
begin
  result := n*n;
end;
```

3. Faites tourner l'appel de fonction `resultat := sommeImpairs(3);`

n	somme	i	result
3			
	0		
		1	
	1		
		2	
	4		
		3	
	9		

4. Faites tourner l'appel de fonction `resultat := carre(3);`

n	result
3	
	9

5. Faites tourner l'appel de fonction `resultat := sommeImpairs(6);`

n	somme	i	result
6			
	0		
		1	
	1		
		2	
	4		
		3	
	9		
		4	
	16		
		5	
	25		
		6	
	36		

6. Faites tourner l'appel de fonction `resultat := carre(6);`

n	result
6	
	36

7. Quelle algorithme est plus efficace? (Rappel: Si on double la valeur d'entrée, comment va évoluer le temps d'éxecution de l'algorithme).

Bien evidemment c'est ce dernier algorithme qui est plus efficace. On dit qu'il est de complexité constante - aussi appelé O(1) - car le temps d'éxécution n'est pas influencé par la valeur d'entrée de la fonction. Le premier algorithme est de complexité linéaire - aussi appelé O(N) car le temps d'éxécution est linéaire en fonction de la la valeur d'entrée de la fonction.

Exercice 4.3 *Complexité*

1. Considérer le tableau suivant :

```
type t_tableau = array[1..15] of integer;
```

2. Ecrire la fonction

```
function dedans(quoi : integer) : integer;
```

qui renvoie la position dans le tableau de l'élement `quoi` s'il est dedans, et 0 sinon.

```
function dedans(quoi : integer) : integer;
var position : integer;
var i : integer;
var entree : integer;
début
        position := 0;
        i := 1;
        tant que (i<=15 ET position = 0) faire
```

```
                entree = tab[i];
                si (quoi = entree) alors
                        position := i;
fin si
                i := i + 1;
        fin tant que
        result := position;
fin
```

3. Faites tourner l'algorithme `resultat := dedans(31)` avec le vecteur `1,3,3,7,8,12,17,18,18,26,29,31,40,44,46`.

quoi	position	i	entree	result
31				
	0			
		1		
			1	
		2		
			3	
		3		
			3	
		4		
			7	
		5		
			8	
		6		
			12	
		7		
			17	
		8		
			18	
		9		
			18	
		10		
			26	
		11		
			29	
		12		
			31	
	12			
		13		
				12

4. Déterminer la fonction de temps maximale ("worst case") T(n) pour un tableau de taille n.

T(n) = 6n + 3

Remarque : Pour etre démonstrative, j'avais introduit la variable `entrée`, elle n'est pas forcément nécessaire. De plus, la comparaison `position = 0` dans la condition de la boucle `tant que` est optionnel. Si on la met pas, le meilleur cas est aussi le pire de cas, c'est-à-dire

dans tout les cas les 15 itérations de la boucle `tant que` seront effectué.

Exercice 4.4 *Complexité*

1. Vous pouvez maitenant supposer que le `vecteur` est trié d'ordre croissant, c'est à dire que `vecteur[i]` $\leq$ `vecteur[j]` $\forall$ i $\leq$ j.

Considérer la fonction suivant :

```
function enigme(quoi : integer, n : integer) : integer;
begin
    var inf, sup, milieu : integer;
    var trouve : boolean;
    inf := 1;
    sup := n;
    trouve := FAUX;
    tant que (sup >=inf ET trouve = FAUX)
        milieu := (inf + sup) DIV 2;
        si (quoi = tab[milieu]) alors
            trouve := VRAI;
        sinon
            si (quoi < tab[milieu])
                sup := milieu -1;
            sinon
                inf := milieu + 1;
            fin si
        fin si
    fin tant que
    if (trouve = FAUX)
        result := 0;
    sinon
        result := milieu;
    fin si
end
```

2. Faites tourner cette fonction `resultat := enigme(31,15);` dans un tableau avec les valeur de `1,3,3,7,8,12,17,18,18,26,29,31,40,44,46.` .

quoi	n	inf	sup	milieu	trouve	result
31						
	15					
		1				
			15			
					FAUX	
				8		
		9				
				12		
					VRAI	
						12

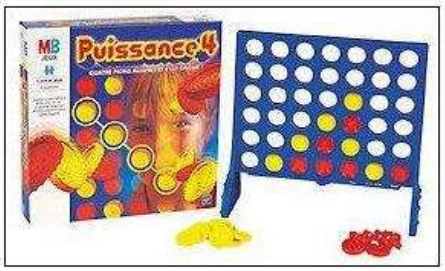
3. Que fait cet algorithme? Est-ce qu'il est mieux que celui dessus? Pourquoi? Avez-vous une idée de sa complexité asymptotique?

C'est un algorithme de recherche dichotomique. En algorithmique, la dichotomie (du grec << couper en deux >>) est un processus itératif de recherche où à chaque étape l'espace de recherche est restreint à l'une de deux parties. Pour des grands valeur de longuer de données n, cet algorithme est mieux que celui dessus. Son temps maximal d'exécution est $T(n) = 7 \log_2 n + 5 \in O(\log n)$, sa complexité asymptotique est donc $O(\log n)$.

Exercice 4.5 *BONUS*

Une solution compréhensible se trouve par exemple sur
http : //www.bmlo.ca/ ∼ gdube/preuves/preuves_eleves_g/lecon8/lecon.html

Exercice 4.6 *Puissance4*



1. Déclarer un type qui permet de stocker un damier du jeu "Puissance4" du type `type t_champ = (vide, jaune, rouge);`.

```
type t_puissance4 = array[1..6] of array[1..7] of t_champ;
var puissance4 : t_puissance4;
```

2. Déclarer un type qui permet de stocker 2 joueurs avec leurs noms et leurs couleurs de pion `t_champ` respectives.

```
type t_joueur = record
                   nom : string;
                   couleur : t_champ;
                end;
var joueur : array[1..2] of t_joueur;
```

3. Ecrire la procédure

```
procedure initialiser;
```

qui initialise le jeu de Puissance4 (tous les champs sont vides).

```
procedure initialiser;
var i,j : integer;
début
    pour i de 1 à 6 faire
        pour j de 1 à 7 faire
            puissance4[i][j] := vide;
        fin pour
    fin pour
fin
```

4. Ecrire la fonction

```
function possible(var c : integer) : boolean;
```

qui teste si un jeueur peut mettre un pion la colonne `c`.

5. Ecrire la procédure

```
function possible(var c : integer) : boolean;
début
    si puissance4[1][c] := vide alors    { si 1 est la ligne la plus haute, sinon [6][c] }
        result := VRAI;
    sinon
        result := FAUX;
fin

procedure poserPion(var c : integer, var couleur : t_champ);
var i : integer;
var ligne : integer;
début
    ligne := 0;
    i := 0;
    tant que i<=6 ET ligne :=0 faire { si 1 est la ligne la plus haute}
        si puissance4[i][c] NOT = vide alors
            ligne := i;
        i:=i + 1;
    fin tant que
fin
```

qui pose un pion de couleur `couleur` dans la colonne `c`.

6. Considérer les fonctions

```
function maxSuiteLigne(var l : integer, couleur : t_champ) : integer;
begin
    max_longueur := 0;
    longueur := 0;
    i := 1;
    tant que i<=7 faire
        si puissance4[l][i] = couleur alors
            longueur := longueur + 1;
            si longueur > max_longueur alors
                max_longueur := longueur;
            fin si
        sinon
             longueur := 0;
        fin si
        i := i +1;
    fin tant que
    result := max_longueur;
end
```

```
function maxSuiteColonne(var c : integer, couleur : t_champ) : integer;
begin
    max_longueur := 0;
    longueur := 0;
    i := 1;
    tant que i<=6 faire
        si puissance4[i][c] = couleur alors
            longueur := longueur + 1;
            si longueur > max_longueur alors
                max_longueur := longueur;
            fin si
        sinon
             longueur := 0;
        fin si
        i := i +1;
    fin tant que
    result := max_longueur;
end
```

qui renvoie la longueur maximale d'une suite d'entrées de `couleur` consécutive dans une ligne `l` (resp. colonnes `c`).

Ecrire la fonction

```
function maxSuite(var couleur : t_champ) : integer;
```

qui utilise ces deux fonctions et qui renvoie la longueur maximale d'une suite d'entrées de `couleur` consécutive dans toutes les lignes et colonnes.

```
function maxSuite(var couleur : t_champ) : integer;
var longueur : integer
var max_longueur : integer;
début
    max_longueur := 0;
    pour i de 1 à 7 faire
        longueur := maxSuiteColonne(i,couleur);
        si longueur>max_longueur alors
            max_longueur := longueur;
        fin si
    fin pour
    pour i de 1 à 6 faire
        longueur := maxSuiteLigne(i,couleur);
        si longueur>max_longueur alors
            max_longueur := longueur;
        fin si
    fin pour
fin
```

7. Le jeu de puissance4 se résume désormais à l'algorithme suivant.

```
joueur := 1; fini := FAUX;
tant que fini = FAUX faire
    afficher le damier
    faire
        afficher Demander au joueur quelle choix;
        lire la colonne du clavier dans la variable colonne;
    tant que possible(colonne);
    poserPion(colonne,joueur[i].couleur);
    si maxSuite(joueur[i].couleur = 4 ou maxSuiteDiagonales(joueur[i].couleur = 4)
        afficher que le joueur a gagné;
        fini := VRAI;
    fin si
    si joueur = 1 alors
        joueur = 2
    sinon
        joueur = 1
    fin si
fin tant que
```

Exercice 4.7 *BONUS*

```
function maxSuiteDiagonales(var couleur : t_champ) : integer;
```

qui renvoie la longueur maximale d'une suite d'entrées de `couleur` consécutive dans les diagonales.

```
function maxSuiteDiagonales(var couleur : t_champ) : integer;

var longueur1, longueur2 : integer
var max_longueur : integer;
var ligne,colonne     : integer;

début

max_longueur := 0;
pour ligne de 1 à 12 faire
    colonne := 1;
    si ligne > 6 alors
        colonne := colonne + (i - 6);
        ligne := 6;
    fin si

    longueur1 := 0;  { Compteur pour la diagonale "du bas à gauche vers le haut à droite" }
    longueur2 := 0;  { Compteur pour l'autre diagonale }
    tant que colonne <= 7 et ligne <= 6 faire
        si damier[ligne][colonnne] = couleur alors
```

```
            longueur1 := longueur1 + 1;
        sinon
            longueur1 := 0;
        fin si;
        si damier[ligne][8-colonnne] = couleur alors
            longueur2 := longueur2 + 1;
        sinon
            longueur2 := 0;
        fin si;
        colonne := colonne + 1;
        ligne := ligne + 1;
    fin tant que
    si longueur1 > max_longueur alors
        max_longueur := longueur1;
    fin si
    si longueur2 > max_longueur alors
        max_longueur := longueur2;
    fin si

fin
```