Algorithmes et structures de données (avancées)

Cours 5

Patrick Reuter

http://www.labri.fr/~preuter/asd2008

Déroulement

- CM mardi de 8h à 9h (salle 33)
- TD
  - Groupe 1 : mardi, 13h30 – 15h00 (salle 51D)
  - Groupe 2 : mardi, 15h15 – 16h45 (salle 51D)

(**en alternance:** salle TD et salle machine)

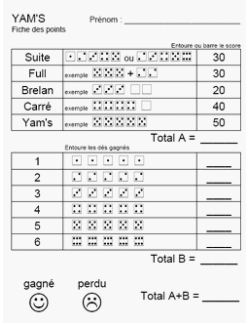
(rendre la feuille à la prochaine séance)

Programme

- Retour sur le TD2 sur machine
- Tableaux numériques
- Théorie de complexité I

En TD sur machine :

- YAM'S


YAM'S
Fiche des points
Prénom : _____

Suite	ou	30
Full	exemple +	30
Brelan	exemple	20
Carré	exemple	40
Yam's	exemple	50

Total A = _____

1	____
2	____
3	____
4	____
5	____
6	____

Total B = _____

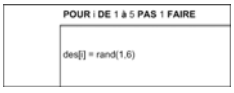
gagné ☺ perdu ☹ Total A+B = _____

Yams

- **Chance** : Ecrire un algorithme qui détermine la somme de tous les 5 dés, et qui affiche à l'écran son résultat

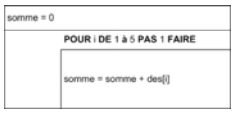
TD 2

- Tableau de 5 dés aléatoires



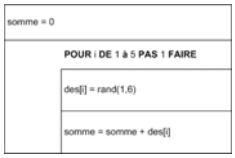
TD 2

- Chance (somme des dés) :



TD 2

- Chance (en une fois) :

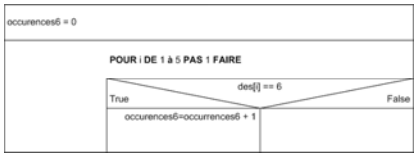


Yams

- **Dès de 6** : Ecrire un algorithme qui compte le nombre de dés de 6, et affiche le résultat.
- **Dès de 5** : Ecrire un algorithme qui compte le nombre de dés de 5, et affiche le résultat.
- **Dès de 4 …**

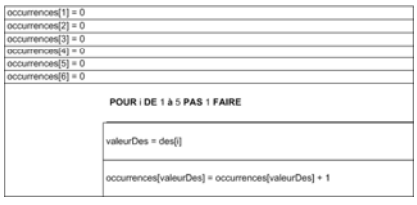
TD 2

- Dés de 6 :



TD 2

- Ou bien :



TD 2

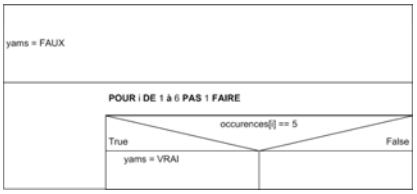
- Ou bien (plus court) :

Yams

- **Yams (50 points)** : Décider si la lancée de dés contient les mêmes 5 dés, et afficher la décision.
- **Brelan :** Décider si la lancée de dés contient au moins un triplet, et afficher la décision ainsi que la somme des 5 dés.
- **Carré** : Décider si la lancée de dés contient au moins un carré, et afficher la décision ainsi que la somme des 5 dés.

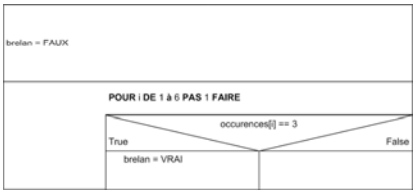
TD 2

- Yams :
  - d'abord calculer le tableau **occurences**



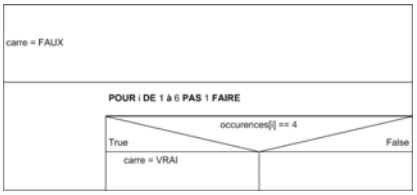
TD 2

- Brelan :
  - d'abord calculer le tableau **occurences**



TD 2

- Carré :
  - d'abord calculer le tableau **occurences**

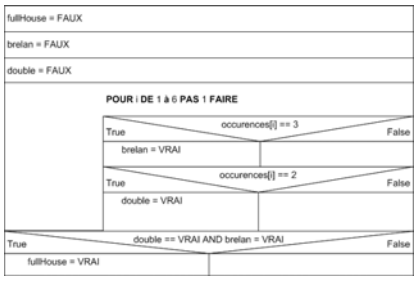


Yams

- **Full House (50 points)** : Décider si la lancée de dés contient un full house (un triplet et un doublet).

TD 2

- Full House : d'abord calculer le tableau **occurences**

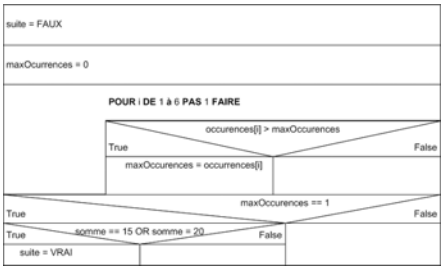


Yams

- **Suite (30 points)** : Décider si la lancée contient la suite 1-2-3-4-5 ou bien 2-3-4-5-6.

TD 2

- Suite : d'abord calculer le tableau **occurences**



Ingrédients d'algorithmes

- Variables
- Affectation
- Condition/Comparaison
- Appel de fonction
- Structure de contrôle
  - Bloc d'instruction
  - Branchements conditionnels
  - Branchements conditionnels (multiples)
  - Boucles

Différence SI et TANT QUE

```
i = 1
SI (i<=5) ALORS
DEBUT
        afficher("Mercredi!")
        i = i + 1
FIN
afficher("Bonne journée");
afficher(i);
```

```
Mercredi
Bonne journée
2
```

```
i = 1
TANT QUE (i<=5) FAIRE
DEBUT
        afficher("Mercredi!")
        i = i + 1
FIN
afficher("Bonne journée");
afficher(i);
```

```
Mercredi
Mercredi
Mercredi
Mercredi
Mercredi
Bonne journée
6
```

Boucles

- Si nombre d'itérations connues :

```
somme =0;
i=1;

TANT QUE i <= 5 FAIRE
DEBUT
    somme = somme + i;
    i=i+1;
FIN
```


```
somme =0;

POUR i de 1 à 5 PAS 1 FAIRE
DEBUT
    somme = somme + i;
FIN
```

1ère exemple: nombre d'itérations connu

- Calculer

$$\sum_{i=1}^{5} i$$

Faire tourner un algorithme

```
somme =0;
i=1;

TANT QUE i <= 5 FAIRE
DEBUT
    somme = somme + i;
    i=i+1;
FIN
```

somme	i
0	1
1	2
3	3
6	4
10	5
15	6

Boucles

- Boucles POUR

```
somme =0;

POUR i de 1 à 5 PAS 1 FAIRE
DEBUT
    somme = somme + i;

FIN
```

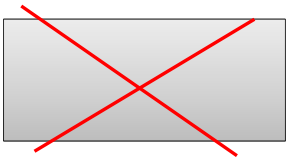
```
$somme = 0;

for ($i=1; $i<=5; $i++)
      $somme=$somme+i;
```

Remarque: `$i++` est equivalent à `$i=$i+1`
`$i--` est equivalent à `$i=$i-1`

Boucles

- Si nombre d'itérations ne sont pas connues :

```
jouer = VRAI

TANT QUE jouer == VRAI FAIRE
DEBUT
    ...
    afficher("Rejouer  (Oui/Non) ?");
    a = input();
    SI (a == "Non") ALORS
          jouer = FALSE
FIN
```



Les tableaux

Les tableaux

- Motivation
- Les tableaux
  - Les tableaux numériques
  - Les tableaux associatifs
  - Les tableaux multidimensionnels

Les tableaux numériques

- **Definition:** Un assemblage d'items qui sont accessible aléatoirement par un nombre entier, leur *index*.

Motivation

en arithmétique :

- **Affectation :**

$a_1 = 1$
$a_2 = 5$
$a_3 = 4$
$a_4 = 8$
…
$a_n = 4$

en algorithmique

```
a[1] = 1;
a[2] = 5;
a[3] = 4;
a[4] = 8;
…
a[n] = 4;
```

en PHP:

```
$a[1] = 1;
$a[2] = 5;
$a[3] = 4;
$a[4] = 8;
…
$a[$n] = 4;
```

Les tableaux

- Motivation
- Les tableaux
  - Les tableaux numériques
  - Les tableaux associatifs
  - Les tableaux multidimensionnels

Les tableaux associatifs

- **Definition:**
  - Une collection d'items qui sont accessible aléatoirement par une clé, souvent une chaîne de caractères

Affectations

- Solution alternative

```
mois['Janvier'] = 31;
mois['Février'] = 28;
mois['Mars] = 31;
mois['Avril'] = 30;
mois['Mai'] = 31;
mois['Juin'] = 30;
mois['Juillet'] = 31;
mois['Août'] = 31;
mois['Septembre'] = 30;
mois['Octobre'] = 31;
mois['Novembre'] = 30;
mois['Décembre'] = 31;
```

```
$mois['Janvier'] = 31;
$mois['Février'] = 28;
$mois['Mars] = 31;
$mois['Avril'] = 30;
$mois['Mai'] = 31;
$mois['Juin'] = 30;
$mois['Juillet'] = 31;
$mois['Août'] = 31;
$mois['Septembre'] = 30;
$mois['Octobre'] = 31;
$mois['Novembre'] = 30;
$mois['Décembre'] = 31;
```

Affectations

- Solution alternative

```
mois['Janvier'] = 31;
mois['Février'] = 28;
mois['Mars] = 31;
mois['Avril'] = 30;
mois['Mai'] = 31;
mois['Juin'] = 30;
mois['Juillet'] = 31;
mois['Août'] = 31;
mois['Septembre'] = 30;
mois['Octobre'] = 31;
mois['Novembre'] = 30;
mois['Décembre'] = 31;
```

```
$mois = array("Janvier" => 31,
              "Février" => 28,
              "Mars" => 31,
              "Avril" => 30,
              "Mai" => 31,
              "Juin" => 30,
              "Juillet" => 31,
              "Août" => 31,
              "Septembre" => 30,
              "Octobre" => 31,
              "Novembre" => 30,
              "Décembre" => 31);
```

Les tableaux associatifs

- aussi appelé
  - **dictionnaire**
  - **table d'association**
  - **map**
  - **hash (cf. ...)**

Le tableau associatif est une strucure de données composé d'un ensemble fini de clefs et d'un ensemble fini de valeurs, où chaque clef est associée à une valeur.

Les tableaux associatifs

- Pourquoi dicitonnaire ?

```
traduction['rouge'] = "red";
traduction['bleu'] = "blue";
traduction['mardi'] = "Tuesday";
…
```

```
$traduction['rouge'] = "red";
$traduction['bleu'] = "blue";
$traduction['mardi'] = "Tuesday";
…
```

Les tableaux associatifs

- Parcourir un tableau associatif

```
POUR CHAQUE <entrée>
   <Bloc d'instruction>
```

Les tableaux associatifs

- Parcourir un tableau associatif

```
POUR CHAQUE (mois as cle => value)
   afficher("Clé ".cle." Valeur ".valeur);
```

```
foreach ($mois as $cle => $valeur)
    echo "Clé :".$cle." Valeur : ".$valeur."<br />";
```

Les tableaux

- Motivation
- Les tableaux
  - Les tableaux numériques
  - Les tableaux associatifs
  - Les tableaux multidimensionnels

Les tableaux multidimensionnels

- Tableau 1D 
- Tableau 2D 
- Tableau 3D 
- Tableau n-D


Tableau 2D

- Matrice de null

$$\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

Tableau 2D

- Affectations :

$$\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

```
a[1][1] = 0;
a[1][2] = 0;
a[1][3] = 0;
…
a[1][colonnes] = 0;
a[2][1] = 0;
a[2][2] = 0;
a[2][3] = 0;
…
a[lignes][colonnes] = 0;
```

```
$a[1][1] = 0;
$a[1][2] = 0;
$a[1][3] = 0;
…
$a[1][$colonnes] = 0;
$a[2][1] = 0;
$a[2][2] = 0;
$a[2][3] = 0;
…
$a[$lignes][$colonnes] =
0;
```

Tableau 2D

- Affectations :

$$\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

```
i = 1
TANT QUE (i<=lignes) FAIR
DEBUT
  j = 1
  TANT QUE (j<=colonnes
  DEBUT
    a[i][j] = 0
    j= j + 1
  FIN
  i = i + 1
FIN
```

```
POUR i de 1
DEBUT
  POUR i d
  DEBUT
      a[i][j] = 0
  FIN
FIN
```

```
POUR i de 1 à 5 PAS 1 FAIRE
    POUR j de 1 à 5 PAS 1 FAIRE
        a[i][j] = 0
```

Tableau 2D

- Affectations :

$$\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

```
POUR i DE 1 à lignes PAS 1 FAIRE
    POUR i DE 1 à colonnes FAIRE
        a[lignes][colonnes] = 0
```

Tableau 2D

- Affectations :

$$\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

```
$i = 1;
while ($i <= $lignes)
{
  $j = 1;
  while ($j <= $ligne
  {
    $a[$i][$j] = 0;
    $j=$j+1;
  }
  $i = $i + 1;
}
```

```
for ($i=1; $
{
  for ($j=1;
  {
    $a[$i][$
  }
}
```

```
for ($i=1; $i<=$lignes; $i++)
  for ($j=1; $j<=$colonnes; $j++)
    $a[$i][$j] = 0;
```

Théorie de la complexité (Partie 1)

Complexité

Exigences à un programme
- Lisibilité
- Extensibilité
- Portabilité
- Réutilisable
- Fiabilité

- Efficacité **(faible complexité)**

Complexité

- Quel est le temps d'éxécution du programme?
  → **Complexité temporelle**

De combien de mémoire le programme a-t-il besoin?
→ **Complexité de mémoire (ou Complexité spatiale)**

Comment déterminer ces complexités ?
→ méthode empirique
→ méthode mathématique

Complexité

Méthode empirique

- Avec une montre et un logiciel d'analyse de mémoire
- **Problème : dépend des facteurs suivants**
  - de la machine utilisée;
  - du jeu d'instructions utilisées
  - de l'habileté du programmeur
  - du jeu de données générées
  - du compilateur choisi
  - …

BIEN SUR : Le temps d'exécution dépend de la longueur de l'entrée (par exemple le nombre de chansons dans la collection).

Complexité

Méthode mathématique
- **Théorie de la complexité**
  - Temporelle
  - Spatiale
- Basée sur une machine abstraite
  - Random-access memory (RAM)
  - Instructions de base (affectation, boucle, appel de fonctions …)
- longueur des données d'entrée **n**
- Ce temps est une fonction **T**(*n*) où *n* est la longueur des données d'entrée.

Complexité : T(n) – Exemple 1

- Calculer $\sum_{i=1}^{n} i$

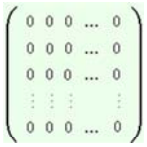
```
somme =0;
i=1;

TANT QUE i <= n FAIRE
DEBUT
    somme = somme + i;
    i=i+1;
FIN
```

Affectations : 2 +2*n
Comparaisons : n+1

→ **T(n) = 3n+3**

Complexité : T(n) – Exemple 2

- Affectations :



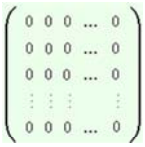
```
i = 1
TANT QUE (i<=n) FAIRE
DEBUT
   j = 1
   TANT QUE (j<=n) FAIRE
   DEBUT
      a[i][j] = 0
      j= j + 1
   FIN
   i = i + 1
FIN
```

Affectations : 2 +2*n
Comparaisons : n+1

3n+3

Complexité : T(n) – Exemple 2

- Affectations :



```
i = 1
TANT QUE (i<=n) FAIRE
DEBUT
   j = 1
   TANT QUE (j<=n) FAIRE
   DEBUT
      a[i][j] = 0
      j= j + 1
   FIN
   i = i + 1
FIN
```

Affectations : 2 +2*n
Comparaisons : n+1

3n+3

Opérations: 1+n*(3n+3)
Comparaisons : n+1

$T(n) = 3n^2+4n+2$

Théorie de la complexité – Diminuer les constantes

- Comment peut-on optimiser T(n) ?
- $T(n) = 4n^2+3n+1$
- 1ère solution : diminuer les constantes a,b,c
- Exemple :
  - $T_1(n) = 4n^2+3n+1$
  - $T_2(n) = 4n^2$
  - $T_3(n) = n^2$

n	1	2	3	10	20	100	1000
$T_1(n)$	8	25	46	431	1661	40301	4003001
$T_2(n)$	4	16	36	400	1600	40000	4000000
$T_3(n)$	1	4	9	100	400	10000	1000000
T_1/T_2	2	1,56	1,28	1,08	1,04	1,01	1
T_1/T_3	8	6,25	5,11	4,31	4,15	4,04	4

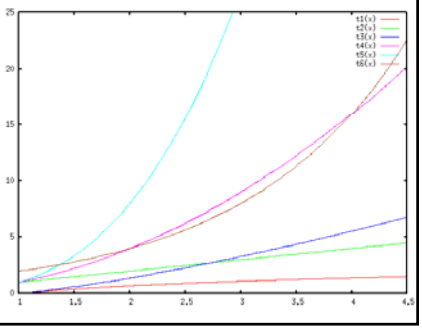
Théorie de la complexité – Diminuer les constantes

- 2 fonctions
  - $T_1(n) = a_1n^2+b_1n+c_1$
  - $T_2(n) = a_2n^2+b_2n+c_2$
- Amélioration :
  - $\lim_{n\to\infty} T_1(n)/T_2(n) = a_1/a_2$
- Pour des grands **n**, seule la constante du plus grand degré est significative

Théorie de la complexité – Changement de la fonction

2ème solution
changer la fonction

- $T_1(n) = \log_2 n$ logarithmique
- $T_2(n) = n$ linéaire
- $T_3(n) = n \log_2 n$ Quasi-linéaire
- $T_4(n) = n^2$ quadratique
- $T_5(n) = n^3$ cubique
- $T_6(n) = 2^n$ exponentiel



Théorie de la complexité – Changement de la fonction

- Quel taille de **n** peut être résolue en 1 seconde, une minute, une heure ?
  - Avec une machine de 1000 instructions par seconde

$T_i(n)$	1 sec	1 min	1 heure
$\log_2 n$	2^{1000}	2^{60000}	$2^{3600000}$
N	1000	60000	36000000
$n \log_2 n$	140	4893	20000
n^2	131	244	1897
n^3	10	39	153
2^n	9	15	21