Cours IUFM
Algorithmique et Programmation

Algorithmique

Objectifs du cours

Objectifs officiels : Programme prédéfini.

→ subira des modifications ...

Objectifs personnels. Les miens :

→ améliorer la communication avec le secondaire

→ former des encadrants. Futurs vacataires ?

=> vpage@univ-ag.fr / 05 90 48 30 75

Objectifs personnels. Les votres :

→ un cours pour les lyceens ?

→ une compétence personnelle ?

Objectifs du cours

Cours en deux parties :

Algorithmique : Science de la mise au point de processus systématiques de résolution d'un problème.

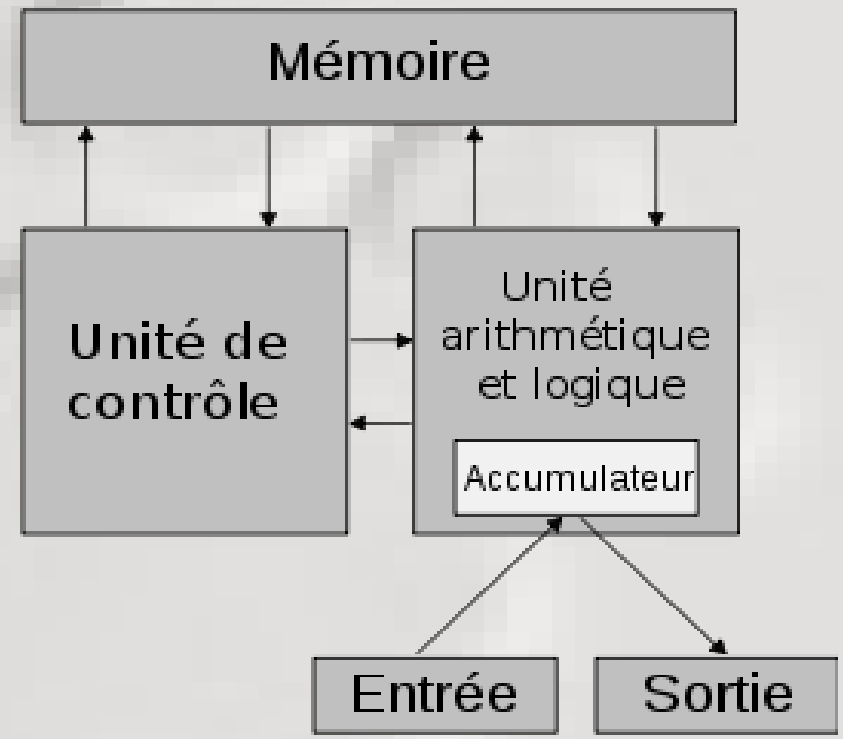
Programmation. Mise en œuvre d'un algorithme dans un langage particulier (C, Pascal, python, Php, Java, Matlab....)

Architecture des ordinateurs (1)

Tres rapide car peu intéressant de mon point de vue pour la programmation.

Architecture de von Neumann :

- L'unité arithmétique et logique (ALU) : Effectue les opérations de base ;
- L'unité de contrôle. Séquençe les opérations (maintient l'ordre)
- La mémoire. Contient à la fois les données et le programme qui dira à l'unité de contrôle quels calculs faire sur ces données.
- Les dispositifs d'entrée-sortie. Communiquent avec le monde extérieur.



Architecture des ordinateurs (2)

Mémoires :

- disque, RAM, swap, cache.

Représentation des données.

Un ordinateur travaille exclusivement sur des 0 et des 1.

→ Pourquoi ? Plus facile technologiquement (transistor)

=> Toute donnée est représentée par un paquet de bits.

Le plus souvent, la taille du paquet est prédéfinie.

=> Trouver un codage :

- Un entier naturel : en base deux.
- Un entier : un bit de signe, le reste est la valeur abs.
- Un réel : signe, mantisse, exposant.
- Un caractère : table de correspondance (ascii par exemple)

Limite de codage : un entier naturel codé sur 4 octets => MaxInt ?

Type de données

Ce codage implique que l'ordinateur sache quel type de donnée il manipule (addition d'entier != addition de réels).

En programmation, les données sont stockées sous formes de variables dont le type est prédéfini.

Types simples :
- Int, float, char

Attention : la chaine '555' n 'est pas l'entier 555
'555'+1 → '5551' (en java, + concatene)

Programme

Un ensemble d'instructions agissant sur des variables.

- Ces instructions sont exécutées séquentiellement avec un début et une fin.
- Les variables sont déclarées au début du programme (peut changer en fonction du langage....

Ici, en langage algorithmique :

```
Begin
    Int A ;    // Declaration de variable de type entier.
    Int B ;    // idem
    A=lireAuClavier() ; // Appel d'une fonction qui change l'affection de A
    A= 5 ;          // affectation directe de A
    B= A/2 ;        // affectation de B
    A= A+1 ;        // ???
    Afficher(B) ;       // appel de fonction utilisant la valeur de B.
End
```

Instructions de base

Affectation :

Variable = résultat de calcul (évaluation)

En Python

```
a = math.sin(b-5)
```

Branchement conditionnel :

Si (condition) alors
{ Instructions 1}
Sinon
{Instructions 2}

```
If a <5 :
        print ' plus petit que 5'
        b = b+8
else :
        Print 'plus grand que 5

print 'la suite...'
```

Conditions : Egalité (==), <, > , NON , ET, OU

```
If non a < b ou c > d :
        print ' cas numéro 1'
```

Instructions de base

En Python

Boucles :

Repeter une instruction plusieurs fois.

Boucle Tant Que :

Tant que (condition) Faire
{ Instructions 1}

```
i = 0
while i<10 :
    print i
    i = i+1
```

Boucle Pour :

Pour i allant de … a … Faire
{ Instructions }

Pas de boucle for sous cette forme en python !
(ci dessus, une equivalence avec un while

Exercice standard de variable

Donner un algorithme demandant 2 entiers (a et b) à l'utilisateur, puis stockant le minimum dans a et le max dans b.

→ certaines variables sont des données « utiles » du programme.
→ certaines variables sont des « tampons » de stockage temporaire.

Tableaux

Une succession de variables de même type.
Comment sont ils gérés en mémoire : Dans la plupart des langages, ces variables sont stockées de façon contigue dans la mémoire.
Mais pas en python, vu que ce sont des listes, que l'on verra plus loin.

Int tab[5] ;// Déclaration d'un tableau de 5 entiers.
Mais en python, on ne déclare pas ! (et les tableaux n'existent pas vraiment, ce sont en fait des listes)

Int tab2D[5][2] ; // Déclaration d'un tableau de 5 lignes, 3 colonnes (contraire ?)
En python, comment ferait-on un tableau 2D ? *=> tableau de tableaux*

B = tab[3]+tab[2] ; // utilisation de variables stockées dans le tableau.
b = tab[3] + tab[2] # bonne nouvelle : en python on pourra les utiliser comme en algorithmique !

Exercices standards de variable

Donner un algorithme permettant de trouver la plus petite valeur dans un tableau.

Exercices standards de variable

Donner un algorithme permettant de calculer la somme des valeurs contenues dans un tableau

→ certaines variables servent « d'accumulateur » que l'on augmente au fur et à mesure.

Exercices standards de variable

Donner un algorithme permettant de chercher si une valeur est présente dans un tableau.

Version 1 : parcours de tout le tableau
Version 2 : avec arrêt en cas de présence.

→ certaines variables sont des «drapeaux » signalant un état ou un changement d'état

Notion de complexité

Le temps de calcul est important. Sa mesure est délicate : dépend du materiel, des opérations en cours sur la machine.... Comment départager les algorithmes ?

=> on compte plutôt le nombre d'opérations à effectuer en fonction de la taille des données. (nombre moyen ou nombre dans le pire cas).

Recherche dans un tableau de n cases :
nombre moyen : n/2 tests
Pire cas : n tests. Dans ces 2 cas, on parle de O(n)

Notion de complexité

Trouver un algorithme efficace de recherche dans un tableau trié et évaluer sa complexité.

Dichotomie
Etape 1 : n cases Complexité T(n)
Etape 2 : n/2 cases Complexité T(n/2) => T(n) = T(n/2)+1

On pose n=2^p => T(2^p) = T(2^(p-1) +1
Dans le pire cas, par recurrence : T(2^p) = T(1) + p

$$T(2^p) = O(p)$$

$$T(n) = \log_2(p)$$

Importance de la complexité

Mettons qu'une recherche sur 10000 entrées coute 10s

Quel est le temps mis pour traiter 20000 entrées pour les complexités suivantes
- O(n) : 20s
- O(log (n) : 10,8s
- O(nlog(n)) : 21,5s
- O(n^2) : 40s

Certains problèmes ont une complexité exponentielle ce qui les rends quasiment insolubles dans la pratique.

Exercices

Décrire les programmes permettant de faire les choses suivantes :

- Calcul de moyenne d'un tableau. (accumulateur)
- Calculer la factorielle d'un nombre entier choisi par l'utilisateur.
- Calculer la valeur du nième terme de la suite de fibonnacci.
- Calculer le produit matrice vecteur suivant

  [1 2] [1]
  [3 4] * [2]
- Calculer un produit matriciel.

fonctions

Une fonction est une sous partie d'un programme relativement indépendante du reste du programme et réalisant une tache précise.
Elle a des arguments en entrées et un (ou plusieurs) arguments en sortie

Intérêt :

1. code propre et bien segmenté => facile a écrire et à maintenir
2. réutilisable => évite les copier coller => facile à debugger.
3. Certains algorithmes sont plus simples sous forme recursive (plus loin)

fonctions

Du point de vue algorithmique :

- Une fonction a des paramètres qu'elle utilise mais ne modifie pas.
- Une fonction a une valeur de retour (eventuellement plusieurs)
- Une fonction a des parametres qu'elle va modifier (plutot a eviter).

Exemples :

Void afficheTableau (int tab[255], int n)
int pgcd (int x, int y)
int echange (MOD int x, MOD int y)

fonctions

Exemple algorithmique :

```
Int max (int a, int b) {
    Int max ;
    if(a<b)
        Max= b ;
    Else
        Max = a ;

    Return max ;
}

Programme principal :
    Afficher(max(3,8)) ;
```

En Python

```
# Définition de la fonction
def max (a,b)
    int max
    if a <b :
        max = b
    else :
        max = a

    return max

# Programme principal
resu = max (10, 12)
print resu

# ou plus court
print max (-5, 27)
```

fonctions

Les paramètres passés à la fonction lors de son appel sont appelés **Paramètres effectifs** :
ils ont une valeur

Les paramètres contribuant à la définition de la fonction sont appelés ***Paramètres formels :***
ils sont, pour le programmeur sans valeur particulière, mais permettent de définir les opérations que l'on fera dans la fonction.

Dans l'exemple ci contre :
a et b sont des paramètres formels (comme en math)
x et y sont des paramètres effectifs

En Python

```
# Définition de la fonction
def max (a,b)
        int max
        if a <b :
                max = b
        else :
                max = a

        return max

# Programme principal
x=10
y= 12
print max (x, y)
```

Portée des variables

Une variable n'est connue que dans la fonction ou elle est déclarée.
On parle alors de variable « locale » à la fonction.

Il existe aussi des variables « globales » connues de toutes les fonctions,
Et que celles ci peuvent modifier. Mais c'est LE MAL ! A eviter autant que possible.

=> Passez les variables utiles à vos fonctions.
Toutes les variables utiles, uniquement elles. (propre)

En python, les variables du programme sont accessibles par les fonctions. Mais n'utilisez pas cette spécificité (car cela vous posera plus de soucis que cela n'en règlera).

Remarque sur les variables

Dans tous les langages, les variables représentent des noms associés à une zone de la mémoire.
Que signifie a = 5 ? Cela dépend.

Dans certains langages (pas python), on met la valeur 5 dans la case mémoire de a.

Dans certains langages, (python), on crée une case mémoire contenant 5 et on dit que a référence cette case mémoire. (techniquement, **a** contient l'adresse de la case mémoire contenant 5)

Ceci a des répercussions assez importantes !

Remarque sur les variables

Exemple en python

Que produira le code suivant :

```
# Programme principal
tab = [1,1]
b = tab

print b
print tab


b[0] = 0
print tab
print b
```

Gardez en mémoire que b et tab référencent le même objet ([1,1] initialement).
La modification sur b[0] modifie donc aussi l'objet que référence tab[0]

A noter : Ici, nous n'avons pas modifié **b** (qui reste une référence sur le même tableau) mais nous avons modifié le contenu d'une case de ce tableau.

Remarque sur les variables

Plus dur :

Que produira le code suivant :

```
# Programme principal
tab = [1,1]
b = tab

print b
print tab


b = [0,0]
print tab
print b
```

Au début, **b** et **tab** référencent le même objet ([1,1] initialement).
L'affectation de **b** implique que maintenant, **b** référence le tableau [0,0]
Mais **tab** référence toujours le tableau [1,1] !

A noter : Contrairement au cas précédent, ici, nous avons modifié **b** lui même.

Remarque sur les variables

En python, certains types sont « immutables » : on ne peut pas les modifier.
C'est le cas des types simples (int, float, string).

Que produira le code suivant :

```
# Programme principal
a = 5
b = a

print b
print a


b = a+1
print a
print b
```

Comme précédemment, l'affectation de b fait que b référence maintenant un objet contenant 6
Mais **a** reste une référence de l'objet contenant 5

Mécanisme des fonctions

Dans certains langages (C, Fortran, Pascal, C++) :
A l'appel de la fonction,

1. le programme appelant s'interrompt (ses variables sont stockées).

2 . Des espaces mémoire sont préparés pour toutes les variables de la fonction.

3. Les valeurs des paramètres effectifs sont copiées dans les paramètres formels correspondants

4. la fonction s'exécute et renvoie sa réponse.

5. Cette réponse remplace l'appel de la fonction dans le programme appelant.

Mécanisme des fonctions

Dans certains langages (python, java) :
A l'appel de la fonction,

1. le programme appelant s'interrompt (ses variables sont stockées).

2 . Des espaces mémoire sont préparés pour toutes les variables de la fonction.

3. Les références des paramètres effectifs sont copiés dans les paramètres formels correspondants

4. la fonction s'exécute et renvoie sa réponse.

5. Cette réponse remplace l'appel de la fonction dans le programme appelant.

recursivité

On peut donc faire des choses comme :

```
Int fact(int n) {
      int resu ;
      if (n<=1)
            return 1 ;
      resu = n*fact(n-1) ;

      return resu ;
}
```

Programme principal :

```
      Afficher (fact(3)) ;
```

Faire une version recursive de fibonacci. Est ce efficace ?

Données plus complexes

Il peut etre interessant de regrouper nos variables en objets ou en structures complexes.

Exemples :

```
Nouveau type Point2D {
    int x ;
    int y ;
}
```

```
Nouveau type contact {
    String Nom ;
    String Prenom ;
    String Tel ;
    String Adresse ;
}
```

Pointeurs et adressage

Les données sont stockées dans la mémoire, et leur valeur est manipulable sous forme de variable... Mais aussi via leur emplacement.

En parle alors d'adressage, ou de pointeur, ou de reference.

Une adresse est un numero, de taille fixe quel que soit le contenu de ce qui est dedans.

Ceci permet de faire des listes chainées, des arbres, des graphes.

Exemple :

```
nouveau type noeud {
    float note ;
    noeud suivant ;    // En fait une reference plus ou moins cachée en fonction des langages....
}
```

Allocation dynamique de mémoire

Avec cette notion de pointeurs ou de référence, vient la notion d'allocation dynamique de mémoire : lorsqu'il faut, pendant l'exécution, que le programme demande un espace de stockage dont la taille peut varier en fonction des actions de l'utilisateur.

Exemple : création du tableau dans l'exercices des moyennes. L'allocation est faite au moment du « tab = new int[n] ; »

En fonction des langages, il faudra gérer cette allocation (demander de l'espace et la libérer quand on ne s'en sert plus) ou ceci est pris en compte automatiquement (libération par ramasse-miettes)

En java, (donc javascool) c'est automatique.

Listes chainées

C'est une collection ordonnée et de taille arbitraire d'éléments de même type. L'accès aux éléments d'une liste se fait de manière séquentielle : chaque élément permet l'accès au suivant (dans un tableau, l'accès se fait de manière absolue, par adressage direct de chaque cellule).

- Une liste chainée est en fait une référence sur le premier element (si la liste est vide, la référence ne « pointe » sur rien...).
- Un élément est constitué d'un objet (ici entier) et d'une référence sur l'élément suivant.
- Pour parcourir la liste, il faut une référence sur l'élément courant qui va se déplacer de suivant en suivant....
- L'ajout d'un élément en fin de liste se fait de la façon suivante : 1. On alloue de la mémoire pour le nouvel élément. 2. on fixe la valeur du suivant du nouvel élément a « rien ». 3. On fixe la valeur du suivant du dernier élément actuel sur le nouvel élément.

Listes chainées : intérêt

Permet de faire des objets algorithmiques intéréssants :

- **Pile** (comme une pile d'assiette : on pose les éléments au dessus, et on retire les éléments par le dessus). Appelée Last In First Out (LIFO)
Exemple d'application : la fonction annuler d'un traitement de texte.

- **File** (comme une file d'attente : on attend a la fin et on traite le début) Appelée FIFO (First In First Out)
Exemple d'application : gestion de travaux d'impression.

- **liste simplement chainée** : idem, mais on insère et on retire ou l'on veut. Nécessite un élément courant. **Liste doublement chainée** (chaque élément connait son précédent et son suivant). **Liste circulaire** (le suivant du dernier est le premier...)

Notez que l'on peut faire des files, des piles avec des tableaux. Mais c'est problématique lors du dépassement de taille et de l'ajout d'un element au milieu...

Listes chainées : exercices

Regarder le code des listes chainées ...

Faire une version récursive de l'affichage d'une liste chainée.

Définir l'algorithme permettant d'insérer un élément à une position donnée dans une liste.

Définir l'algorithme permettant de supprimer un élément à une position donnée dans une liste.

Définir l'algorithme permettant de dupliquer une liste chainée.

Définir l'algorithme permettant créer une version « retournée » d'une liste chainée.

Arbres...

Les listes chainées, c'est bien. Mais comment stocker, par exemple, un arbre généalogique ascendant (vous, vos parents, leurs parents respectifs....) ?

On parle alors d'arbres binaires : un sommet a au maximum 2 sommets.

Un peu de la même façon que les listes chainées... Imaginez une implémentation pour cela.

Ce type d'arbres convient-t-il pour, par exemple, stocker une arborescence de fichiers sur un ordinateurs ?

Imaginez une implémentation pour ce type d'arbres...

Exemple d'application : Intelligence Artificielle pour le jeu du morpion...

Arbres, exercices..

Regardez le code fourni pour les arbres...

- Faites une fonction pour compter les sommets de l'arbre. (en recursif, c'est plus facile...)

- Faites une fonction pour calculer la hauteur de l'arbre...

- faites une fonction qui affiche tous les nœuds de l'arbre (en recursif c'est plus facile)

- Essayez de trouver un algorithme itératif pour afficher ces nœuds...

Graphes

Un arbre, c'est bien, mais...

- Imaginez un réseau routier. Peut on le modéliser avec un arbre ?
- Vous voulez faire de l'IA pour un jeu d'echec. Un arbre est il vraiment adapté ?

Un graphe est : Un ensemble de sommets et un ensemble d'arêtes reliant des nœuds. Ces arêtes peuvent être : valuées (km ou cout par exemple) ou non, orientée (sens unique) ou non.

Ca sert a quoi ? A stocker (et analyser) des relations entre objets...

Qu'est ce qu'on met la dedans ?

- dans les sommets : des objets du monde réel / des configurations.
- dans les arêtes : des relations. Si l'arete est valuée, c'est un cout ou un type de relation...

Comment pourait t'on implémenter tout ca ?

Types de langages

On peut classer les langages de programmation de multiples manières.
La plus importante est sans doute la suivante : leur Mode d'execution.
On distingue :

• Les langages compilés : Le code source est transformé par un compilateur en un exécutable qui est lancé directement par le système d'exploitation.
Exemple de langage : langage C. Exemple de programme : firefox...

• Les langages interprétés : Le code source est interprété par un logiciel (l'interpréteur) qui prend en charge l'exécution des commandes auprès du système d'exploitation.
Exemple de langage : python. Exemple de programme : script dos...

• Le langage semi compilé (seul exemple : java) : Le langage est compilé en un pseudo exécutable qui va s'executer sur une machine virtuelle (la jvm). C'est la jvm qui va interagir avec l'OS.

Précisions sur ces types

- langages compilés :

Exemples : C , C++, Fortran, Pascal....

Avantages / Inconvénients :

+ : Execution + rapide.

+ : On peut distribuer l'exécutable sur toute architecture semblable sans rien installer.

- : Pour changer d'architecture, il faut compiler pour cette architecture.

- langages interprétés :

Exemples : python, matlab / octave , algoBox, php, perl, javascript, ...

Avantages / Inconvénients :

- : plus lent.

- : Pour distribuer un programme, il faut installer sur chaque machine l'interpréteur.

+ : en cas de changement d'architecture, rien de spécifique a faire (l'interpreteur installé est adapté a l'architecture).

Types de langages (2)

Autre type de classement : Paradigme de programmation.

On distingue :

- Programmation impérative : Ce qu'on a fait jusqu'ici.

Exemples : C, Fortran, Pascal, AlgoBox,...

- Programmation objet : On considère le monde du problème a résoudre comme un ensemble d'objets. Chaque objet a des propriétés et des fonctions (méthodes) spécifiques. La résolution du problème se fait par programmation impérative utilisant ces propriétés et ces méthodes.

Un langage supportant la programmation objet peut être utilisé sans créer d'objets. C'est en fait ce que l'on a fait jusqu'ici....
Exemples : C++, java, php , perl.....

- Autres paradigmes : dépassent le cadre de ce cours...

Choix de langages

Tout dépend de l'application. En gros (ce n'est pas parole d'évangile) :

• Apprendre à des débutants l'algorithmique sans les rebuter.

=> eviter la prog objet. Se concentrer sur la notion de méthodes et de typage. Essayer de trouver un langage ou les opérations sont simplifiées.

=> Solutions : Java (sans trop d'objet), AlgoBox, Javascool.

• Faire des programmes pour illustrer vos cours : Essentiellement des calculs mathématiques et de l'affichage de courbes.

=> Un langage ou tout cela est déjà prévu. Eviter les problèmes algorithmiques.

=> Solutions Matlab / Octave.

• Cadre industriel :

=> Gros projet long terme : C++, java (maintenance)

=> Temps réel : C, Fortran.

=> Fiabilité : ADA (Pascal adapté)

=> Développement Web : PHP, j2E

Les Images

Des matrices …

Fichiers (Input/Output)

Détournement des entrées sorties classiques
Un programme qui s'exécute dans un terminal a déjà 3 entrées/sorties :
- l'entrée standard (stdin) : par défaut, vient du clavier.
- la sortie standard (stdout) : par défaut, s'affiche dans le terminal (println)
- la sortie d'erreur standard (stderr) : par défaut, s'affiche dans le terminal aussi.

Méthode 1 pour écrire dans un fichier pour le programme « print.java »

Trouver le fichier « print.class » du programme et lancer
java print.class > monFichier

Ceci redirige la sortie standard vers le fichier monFichier.
Fonctionne pour écrire tout ce qui vient d'un « System.out.println » vers un fichier. Mais le nom du fichier est fixé par la personne qui execute le programme (pas toujours pratique).

Fonctionnement système

Un programme qui veut écrire/lire dans un fichier n'interagit pas directement avec le disque dur. En effet, on veut éviter qu'un programme puisse faire n'importe quoi.

Toute lecture/écriture doit donc demander au système l'autorisation de le faire. C'est un appel système qui va nous renvoyer un « pointeur sur un fichier » qu'on utilisera pour écrire (partiellement masqué en java)

Ceci peut générer des problèmes qu'il va falloir gérer
- pb d'autorisation de lecture / d'écriture
- lecture de fichier qui n'existe pas...)

=> On verra ceci plus loin.

Fichier texte ou binaire

Tout fichier est composé d'octets (représentant des nombres).

Un fichier texte est composé d'octets.
Quand on ouvre un fichier texte avec un éditeur de texte, ces octets sont interprétés comme des caractères a l'aide tables de conversions (ascii, unicode comme utf8).

Quand on veut écrire dans un fichier, on va choisir quelle forme va prendre l'ecriture : caractères ou octets.

Exemple : double x = 2.56789 ;
Ecrit en caractères : 7 caractères => ~7 octets
Ecrit en octets : 4 octets.

Fichier texte

On va utiliser la classe FileWriter (pour ecrire dans un fichier...)

On importe la classe :
Import java.io.FileWriter ;

Puis dans le main :
FileWriter fw = new FileWriter("file1.txt");

fw.write("bonjour a tous") ;

fw.close() ;