Algorithmique avancée

Cours 1

## Objectif de la matière :

* Objectif général
  + Apprendre à utiliser des structures de données avancées en algorithmique et en programmation
* Compétences spécifiques
  + Définition et utilisation des tables de hachage
  + Définition, représentation et utilisation des graphes
  + Algorithmes de la théorie des graphes
  + Implémentations en langage C
* Prérequis
  + Info0301 : programmation en C
  + Info0401 : algorithmique

## Structure de la matière

* Partie 1 : introduction et rappels
  + Analyse et conception des algorithmes
* Partie 2 : graphes 1
  + Définition et représentation
  + Algorithmes de base
    - Parcours en largeur
    - Parcours en profondeur
    - Rappels sur les types de données élémentaires nécessaires : Piles, files, listes chainées, tables de hachage
* Partie 3 : graphes 2
  + Algorithmes classiques et/ou avancés
    - Tri topologique
    - Connexité
    - Plus courts chemins
    - Arbres couvrants de poids minimal
    - Flot maximal
    - Optimisation linéaire/combinatoire
    - Rappels sur les types de données élémentaire nécessaires : tas, arbres, ensembles disjoints

Support : T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, « algorithmique », 3e édition, 2010

# Le rôle des algorithmes en informatique

## Algorithme

* Procédure de calcul bien définie qui
  + Prend en entrée une valeur ou un ensemble de valeurs
  + Donne en sortie une valeur ou un ensemble de valeurs
* Séquence d’étapes de calcul qui transforment l’entrée en sortie
* Outil permettant de résoudre un problème de calcul
* Des problèmes complexes à résoudre par des ordinateurs nécessitent des algorithmes perfomants
  + Calcul scientifique : simulation de systèmes
  + Internet : recherche et manipulation de grands volumes de données
  + Commerce électronique : sécurité, encryptage
  + Industrie : allocation de ressource
  + …

## Algorithmes en tant que technologie

* Les ordinateurs
  + Ne sont pas infiniment rapides
  + N’ont pas une mémoire infinie
* Le temps machine et l’espace mémoire sont des ressources limitées
* Une machine moins performante qui exécute de bons algorithmes pourra surpasser une machine plus performante qui exécute de mauvais algorithmes

# Analyse des algorithmes

## Exemple : le problème du tri

* Entrée
  + Une suite de *n* nombres (a1, a2, …, an)
* Sortie
  + Une permutation (réorganisation)

(a1’, a2’, …, an’) de la suite donnée en entrée

* + De sorte que

a1’ < a2’ < … < an’

* La suite (31, 41, 59, 26, 41, 58)
  + Est une instance du problème de tri
* Opération majeure en informatique
  + Employée par une multitude de programmes comme phase intermédiaire
* L’algorithme optimal pour une application donnée dépend
  + Du nombre d’éléments à trier
  + De la façon dont les éléments sont plus ou moins triés initialement
  + Des restrictions sur les valeurs des éléments
  + …

(31, 41, 59, 26, 41, 58) 🡺 (26, 31, 41, 41, 58, 59)

## Un 1er algorithme : le tri par insertion

* Algorithme naturel pour l’être humain
  + Comment triez-vous vos cartes quand vous jouez aux cartes (si vous jouez aux cartes !) ?
  + On prend les cartes une à une et on les place au bon endroit dans sa main
* Efficace quand il s’agit de trier un petit nombre d’éléments
* Exemple 1
  + Application sur l’instance (5,2,4,6,1,3)
* Pseudo-code

|  |
| --- |
| TRI-INSERTION(t)  pour j=2 à t.longeur  clé = t[j]  //insère t[j] dans la séquence triée t[1...j-1]  i=j-1  tant que i>0 et t[i]>clé  t[i+1]=t[i]  i=i-1  t[i+1]=clé |

Fonctionnement d’un algorithme (Schéma 1) (slide 13 : tri par insertion)

## Correction d’un algorithme

* Un algorithme est dit correct si, pour chaque instance est entrée, il se termine en produisant la bonne sortie
  + Un algorithme correct résoud le problème donné
* Un algorithme incorrect peut
  + Ne pas se terminer pour certaines instances
  + Se terminer sur une réponse autre que celle voulue
* On montre qu’un algorithme est correct par les invariants de boucle

## Invariant de boucle

* Propriété qui est vraie à chaque passage dans la boucle
* On doit montrer 3 choses concernant un invariant de boucle
  + Initialisation : il est vrai avant la première itération de la boucle
  + Conservation : s’il est vrai avant une itération de la boucle, il le reste avant l’itération suivante
  + Terminaison : une fois terminée la boucle, l’invariant fournit une propriété utile qui aide à montrer la validité de l’algorithme

## Validité du tri par insertion

* Invariant de boucle

Au début de chaque itération de la boucle pour le sous-tableau t[1… j-1] se compose des éléments qui occupaient initialement les positions t[1…j-1] mais qui sont maintenant triés

(slide 16)