# IFT2125 - Hiver 2016

## Introduction à l'algorithmique

**Sylvie Hamel**

André-Aisenstadt: **3161**
hamelsyl@iro.umontreal.ca

http://www.iro.umontreal.ca/~hamelsyl/IFT2125-H16

## Concomitants:

- IFT 2015: Structures de données
- MAT1978: Probabilités et Statistique

## Horaire et locaux:

| *Activité* | *Jour* | *Local* |
|---|---|---|
| théorie | **Lundi: 13h30 - 15h30** | **Y115** |
| théorie | **Mardi: 12h30-13h30** | **Y117** |
| travaux pratiques | **Mardi: 13h30-15h30** | **Y117** |

# Objectifs:

Le cours IFT2125 permet à l'étudiant(e) d'apprendre à concevoir des algorithmes, d'analyser l'efficacité de ces algorithmes, de se familiariser avec certaines techniques mathématiques et de développer un réflexe essentiel en informatique; celui de ne pas se contenter de la première méthode trouvée mais plutôt de chercher la méthode la plus efficace pour résoudre un problème donné.

# Manuels de références:


1. Gilles Brassard et Paul Bratley, **Fundamentals of Algorithms**, Prentice-Hall, 1996.

2. T.H. Cormen, C.E. Leiserson, R.L. Rivest et C. Stein, **Introduction to algorithms (3rd edition)**, MIT Press, 2009.

3. T.H. Cormen, C.E. Leiserson, R.L. Rivest et C. Stein, **Algorithmique (3ieme édition)**, Dunod, 2010.

4. D. Knuth, **The art of Computer programming**, Addison-Wesley, 1981.

5. A. Aho, J. Hopcroft et J. Ullman, **The design and analysis of computer algorithms**, Addison-Wesley, 1974.

# Contenu:

1. Introduction : Motivations, (Chapitres 1-2 (Brassard-Bratley ou Cormen *et al*.)
2. Outils pour l'analyse d'efficacité: Notation asymptotique, résolution de récurrences (Chapitres 3-4 (Brassard-Bratley ou Cormen *et al*.)
3. Algorithmes gloutons (ou voraces) (Chapitre 6 Brassard-Bradley, Chapitre 16 Cormen *et al*.)
4. Diviser-pour-régner (Chapitre 7 Brassard-Bradley, Chapitre 4 Cormen *et al*.)
5. Programmation Dynamique (Chapitre 8 Brassard-Bradley, Chapitre 15 Cormen *et al*.)
6. Algorithmes probabilistes (Chapitre 10 Brassard-Bradley)
7. Algorithmes pour des graphes (Chapitre 9 Brassard-Bradley)
8. Algorithmes vectoriels ~ si le temps le permet

## Journées Spéciales:

- Premier TP: Mardi 12 janvier, 13h30-15h30 Y-117
- Intra: Mardi 16 février, 12h30-14h30, B2285
- Final: Mardi 19 avril, 12h30-15h30, N-615
- Semaine de lecture: 29 février au 4 mars

## Démonstrateurs:

- Vincent Antaki, antaki.vincent@gmail.com
- Robin Milosz, miloszro@iro.umontreal.ca
- Disponibilité: sur rendez-vous

# Évaluation:

## 2 devoirs:

- Pondération: 15 % chacun
- Discussions permises
- TP1: - réception de l'énoncé: **25 janvier**
  - remise du devoir: **8 février**
- TP2: - réception de l'énoncé: **28 mars**
  - remise du devoir: **11 avril**

Tout retard dans la remise des travaux entraînera une pénalité de 10% par jour (24 heures).

## Intra(Mardi 16 février):

- Pondération: 30 %
- Durée: 2 heures

## Final(Mardi 19 avril):

- Pondération: 40 %
- Durée: 3 heures

Un seuil cumulatif de 50% aux examens sera appliqué (moyenne de 50% pour les deux examens combinés)

# Plagiat:

- Pour vos travaux, vous pouvez utiliser tout ce qui est disponible et accessible publiquement en autant que les références utilisées soient proprement citées.
- Les cas de plagiat seront traités conformément aux règles en vigueur à l'Université de Montréal.

# Algorithmique: Pourquoi?

- Comment développer un algorithme efficace pour résoudre un problème donné?

  - Étude de plusieurs techniques générales de conception d'algorithmes:

    Algorithmes gloutons
    Diviser-pour-régner
    Programmation dynamique
    Algorithmes probabilistes
    Algorithmes vectoriels

- Parmi plusieurs algorithmes résolvant le même problème, lequel choisir?

  - Étude de techniques d'analyse d'efficacité des algorithmes basées sur les notations aymptotiques et la résolution de récurrences.

  - Permet de prédire la quantité de temps ou de mémoire requise à l'exécution d'un algorithme sur des données de grande taille.

# Algorithme

**Historique:** Le terme **algorithme** vient d'**Al Khowarizmi**, mathématicien arabe du IXe siècle

- Le livre d'Al Khowarizmi constitue la base de la notation décimale moderne.
- Au départ, le mot "**algorisme**" désignait les règles nécessaires pour effectuer des calculs arithmétiques en utilisant la notation décimale.
- Le terme **algorithme** apparaît au XVIIIe siècle.

# Algorithme

## Définitions:

1) Le petit Larousse

> Ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations.

2) Encyclopedia Universalis

> Spécification d'un schéma de calcul sous forme d'une suite fini d'opérations élémentaires obéissant à un enchaînement déterminé.

# Algorithme

**Propriétés:**

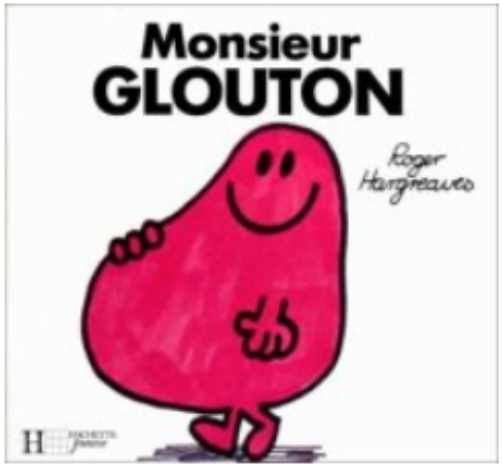
- **Les entrées:** un algorithme prend des valeurs d'entrées à partir d'ensembles définis
- **La sortie:** constitue la solution du problème de départ

⇒ **La finitude:** l'algorithme doit produire la sortie souhaitée en un nombre fini (mais peut-être très grand) d'étapes, quelque soit l'entrée

⇒ **L'efficacité:** Chaque étape de l'algorithme doit pouvoir s'exécuter dans un temps fini

⇒ **La généralité:** l'algorithme s'applique à tous les problèmes d'une forme désirée

# Algorithmes gloutons ou voraces (greedy algorithms)




L'idée ici est de faire, étape après étape, un choix optimum local dans l'espoir d'obtenir à la fin du processus un optimum global pour le problème considéré.

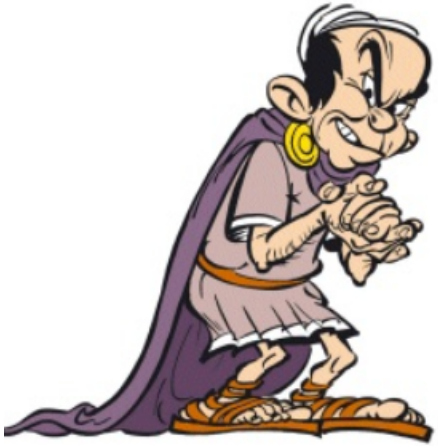
## Problèmes classiques considérés dans le cours:

- Arbre couvrant minimal
- Algorithme du plus court chemin
- Problème du sac à dos
- Problème de la file d'attente

## Problème bio-informatique considéré dans le cours:

- Trier par inversions (génomique comparative)

# Algorithmes diviser-pour-régner (divide-and-conquer)



L'idée ici est de "casser" le problème à résoudre en sous-problèmes. On résout alors les sous-problèmes (en les cassant possiblement de nouveau) et on combine les résultats pour résoudre le problème original.

## Problèmes classiques considérés dans le cours:

- Recherche binaire
- Algorithmes de tri
- Trouver la médiane d'une liste d'entiers

## Problème bio-informatique considéré dans le cours:

- Méthode des 4 Russes pour l'alignement de séquences

# Algorithmes de programmation dynamique

| D | | G | T | C | A | G | G | T |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| C | 1 | 1 | 2 | 2 | 3 | 4 | 5 | 6 |
| A | 2 | 2 | 2 | 3 | 2 | 3 | 4 | 5 |
| T | 3 | 3 | 2 | 3 | 3 | 3 | 4 | 4 |
| A | 4 | 4 | 3 | 3 | 3 | 4 | 4 | 5 |
| G | 5 | 4 | 4 | 4 | 4 | 3 | 4 | 5 |
| T | 6 | 5 | 4 | 5 | 5 | 4 | 4 | 4 |
| G | 7 | 6 | 5 | 5 | 6 | 5 | 4 | 5 |

C'est une approche "bottum-up". L'idée est que pour résoudre un problème, on commence par résoudre les plus petits sous-problèmes et on conserve les solutions dans une table. On se sert alors des solutions calculées pour résoudre des sous-problèmes de plus en plus grands jusqu'à l'obtention de la solution globale.

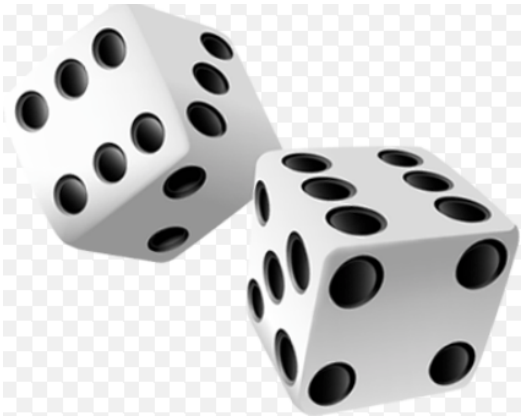
## Problèmes classiques considérés dans le cours:

- Problème du sac à dos
- Algorithme du plus court chemin

## Problème bio-informatique considéré dans le cours:

- Alignements de séquences

# Algorithmes probabilistes



L'idée ici est qu'à chaque étape de l'algorithme on choisit aléatoirement la prochaine étape plutôt que d'essayer d'optimiser. Cela implique que différentes exécutions d'un algorithme probabiliste donnera des solutions différentes au problème.

1) Numérique: Retourne une solution approximative à un problème numérique

plus de temps = plus de précision

2) Monte Carlo: Retourne toujours une réponse mais peut se tromper.

plus de temps = plus grande probabilité que la réponse soit bonne

3) Las Vegas: Ne retourne jamais une réponse inexacte mais parfois ne trouve pas de réponse du tout

plus de temps = plus grande probabilité de succès sur chaque instance de départ

# Algorithmes sur les graphes

L'idée ici est qu'on représente notre recherche de solution pour un problème comme un graphe (arbre) où chaque sommet contient une solution partielle et chaque arête représente une façon d'étendre cette solution.

1) Retour-en-arrière:

On commence dans la racine de l'arbre (solution nulle ou vide) et on explore l'arbre en profondeur en construisant les sommets et solutions partielles au problème au fur et à mesure

2) Branch-and-bound:

L'idée est aussi d'explorer un arbre solutions pour trouver la solution optimale à un problème d'optimisation. À chaque noeud, on calcule une borne pour les valeurs des solutions découlant de ce noeud. Si cette borne nous montre que ces solutions seront nécessairement pire que la meilleure solution trouvée jusqu'à présent alors, on n'explore pas cette partie du graphe.

# Algorithmes vectorielles

Un algorithme vectoriel est un algorithme qui trouve un vecteur de sortie en appliquant un nombre d'opérations sur le vecteur d'entrée, indépendant de la longueur du vecteur.

L'exploitation du parallélisme des opérations sur les vecteurs de bits améliore grandement le temps de calcul de ces algorithmes par rapport aux algorithmes présentés précédemment.