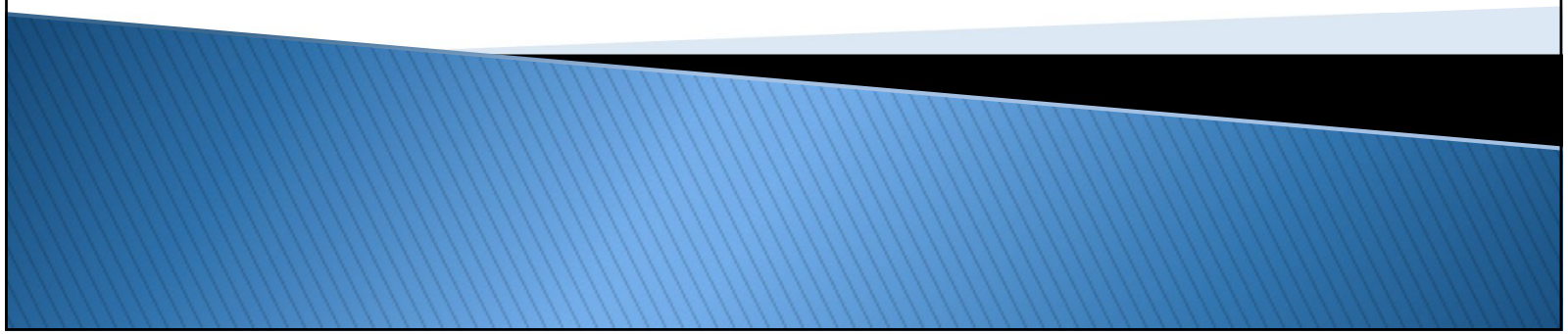
Algorithmique des Tableaux
Licence 1 : maths/Info

2019-20

Carole Blanc carole.blanc@u-bordeaux.fr



Site de l'UE **4TPM205U**

http://dept-info.labri.fr/ENSEIGNEMENT/algotab/

Introduction

- Informations pratiques
- Objectifs du cours

Informations Pratiques

Organisation de l'UE :

- Dans les edt 2 séries A et B
  - 10 séances de cours (1h20)
  - 11 séances de TD (1h20)
  - 11 séances de TD machine (1h20)
- TD 11 groupes :A1…A6, B1B2B4B5 et PI/CMI
  - Organisés selon les options choisies
  - Un groupe spécifique pour le parcours LI/CMI avec TD en anglais.
- TD machine : 16 groupes

  2gpes TD->3gpes Td machine (TM)

  1gpes TD->2gpes TM si effectif important

  1gpe sinon

Informations Pratiques

Organisation de l'UE :

- TD machine : 17 groupes

2gpes TD->3gpes TM

  - Lundi : B1 et B2
  - Mardi : A6 et PI_CMI OPTIM
  - Mardi : A3 et A4
  - Mardi A5 + CMI ISI est autonome sauf effectif croissant
  - Mercredi : Groupes A1 et A2
  - Vendredi : Groupes B4 et B5

IP : Les enseignants

	Enseignant TD	Enseignant TM	Enseignant TM supplémentaire
A1	Zenaïda Tucsnak	Zenaïda Tucsnak	Pierre-Etienne Martin
A2	???	???	
A3	Anne Vialard	Anne Vialard	???
A4	Olivier Delmas	Olivier Delmas	
A5	Driss Nejjari	Driss Nejjari	Driss Nejjari ou ???
A6	Giuliana Bianchi	Giuliana Bianchi	Jason Schoeters
PI-CMI	Thalita Firmo Drumond	Thalita Firmo Drumond	
B1	Anne Vialard	Anne Vialard	Driss Nejjari
B2	Raluca Uricaru	Raluca Uricaru	
B4	Stefka Gueorguieva	Stefka Gueorguieva	Guillaume Blin
B5	Thalita Firmo Drumond	Thalita Firmo Drumond	

IP : l'emploi du temps

- Vous serez répartis sur les 3ème TD machine selon le découpage en demi-groupe.
- Les PI et CMI suivent les précions qui les concernent.
- Lire régulièrement votre boite mail « d'étudiant/e»
- Poser vos questions par mail en utilisant votre adresse étudiante. (carole.blanc@u_bordeaux.fr)

IP : Modalités de contrôle des connaissances

Epreuves	Session 1
2 Tests 20mn en TD	0,1 chacun
TP noté 1h	0,2
DS semaine 1h semaine du 14/04	0,2
Examen	0,4

Pas de Session 2

Objectif du cours

- Ecrire des algorithmes et programmer des fonctions en s'appuyant sur un langage impératif.

- Etudier l'algorithmique des tableaux : tris de tableaux et calcul numérique.

  Seront traitées les notions de:
  - Variables, expressions, affectations.
  - Structures de contrôle : boucles, conditionnelles.
  - Portée des noms et durée de vie des variables.
  - Algorithmes et fonctions récursives.
  - Coût d'une opération et de complexité élémentaire.

- La programmation des algorithmes se fera en Python
- Les supports de cours et de TD/TM seront disponibles sur le site de l'UE au fur et à mesure.

Partie 1 : Rappels

- Algorithme
- Programme
- Syntaxe du langage Python.

Rappels : Algorithmes

- Qu'est-ce qu'un algorithme?
- Notion d'efficacité des algorithmes :
  - structures de données
  - temps de calculs

Qu'est-ce qu'un algorithme?

- Un algorithme est une méthode systématique (comme une recette) pour résoudre un problème donné.

- Il se compose d'une suite d'opérations simples à effectuer pour résoudre un problème.

Qu'est-ce qu'un algorithme?

Exemple 1 : faire ***n*** tasses de café

Qu'est-ce qu'un algorithme?

Exemple 1 : algorithme pour faire *n* tasses de café

```
mettre un filtre
niveau_reservoir =0
Tant que niveau_reservoir < n faire
  mettre une dose d'eau dans le reservoir
  augmenter niveau_reservoir de 1
Fin tant que
nb_doses = 0
Tant que nb_doses < n faire
  mettre une dose de cafe dans le filtre
  augmenter nb_doses de 1
Fin tant que
allumer la cafetiere
```

Vous ne connaissez pas `n`, conséquence ?

Qu'est-ce qu'un algorithme?

Exemple 1 : algorithme pour faire *n* tasses de café

```
mettre un filtre
niveau_reservoir =0
Tant que niveau_reservoir < n faire
  mettre une dose d'eau dans le reservoir
  augmenter niveau_reservoir de 1
Fin tant que
nb_doses = 0
Tant que nb_doses < n faire
  mettre une dose de cafe dans le filtre
  augmenter nb_doses de 1
Fin tant que
allumer la cafetiere
```

Quelle instruction manque-t-il si on veut boire 10 tasses de café ?

- faire 10 tasses de café

Qu'est-ce qu'un algorithme?

- Un algorithme est une méthode systématique (comme une recette) pour résoudre un problème donné.
- Il se compose d'une suite d'opérations simples à effectuer pour résoudre un problème.
- En informatique cette méthode doit être applicable par un ordinateur.

Qu'est-ce qu'un algorithme?

Exemple 2 : afficher les diviseurs de l'entier *n*

```
si n > 0 alors
  pour tout entier i entre 1 et n faire
    si n est divisible par i alors
      afficher(i)
    finsi
  finpour
finsi
```

Remarquez l'usage ici d'un pseudo langage

Importance de l'algorithmique

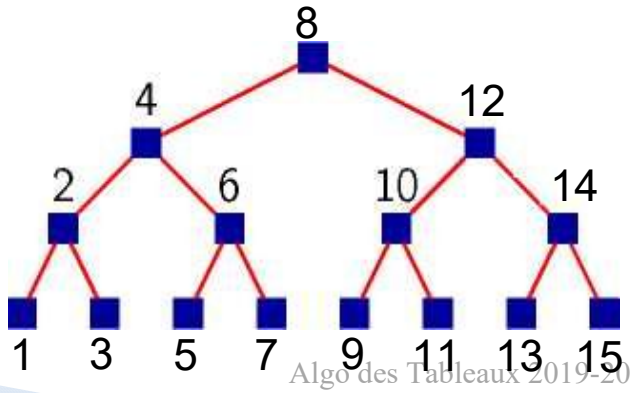
- Pour un problème donné, il y a plusieurs algorithmes.
- Il est facile d'écrire des algorithmes faux ou inefficaces.
- Une erreur peut faire la différence entre plusieurs années et quelques minutes de calculs sur une même machine.
- C'est souvent une question d'utilisation de structures de données ou d'algorithmes connus dans la littérature.
- Une structure de données est une façon particulière d'organiser les données.

Représenter et organiser les données

Exemple : Construire une ville de 15 maisons en évitant aux livreurs de pizzas qui suivent les rues un trajet trop long depuis la pizzeria.

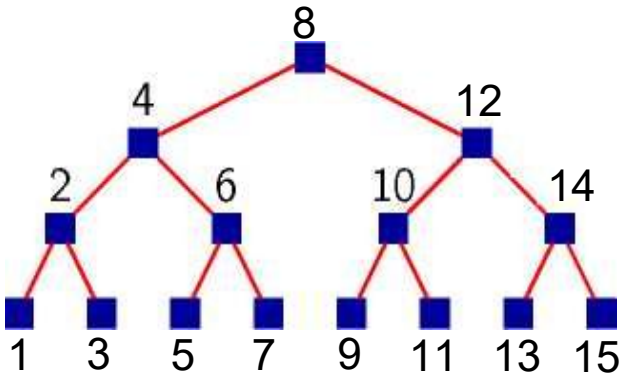
Organisation 1 : Linéaire. Numéros croissants. Pizzeria au numéro 1.

Organisation 2 : Embranchements.
À l'ouest de la maison *k,* n° < *k,* et à l'est, n° > *k.*
La pizzeria est au numéro 8.



Temps de calcul

- Dans les deux organisations, le livreur a une méthode simple pour trouver une maison en partant de la pizzeria.



Temps de calcul

- Dans les deux organisations, le livreur a une méthode simple pour trouver une maison en partant de la pizzeria.
- On suppose qu'il faut une unité de temps pour passer d'une maison à une autre (en suivant une rue).
- Dans **le cas le pire**, quel est le temps mis par un livreur pour aller jusqu'à une maison depuis la pizzeria ?

Temps de calcul

Nombre de maisons	Temps avec organisation 1	Temps avec organisation 2
15	14	3
1023	1022	9
1 073 741 823	1073741822	29
n	n-1	$\sim \log_2(n)$

Rq: une organisation en étoile avec la pizzeria au milieu permet des trajets très courts, mais choisir la bonne rue prend du temps.

Temps de calcul

- Le **temps de calcul** (ou complexité) d'un algorithme est la fonction mathématique qui donne le **nombre maximal d'instructions élémentaires** que l'algorithme effectue en fonction de la taille des données manipulées.

- Par exemple si on travaille **sur des objets de taille *n*,** le temps de calcul est évalué en fonction de ***n*.**

Temps de calcul

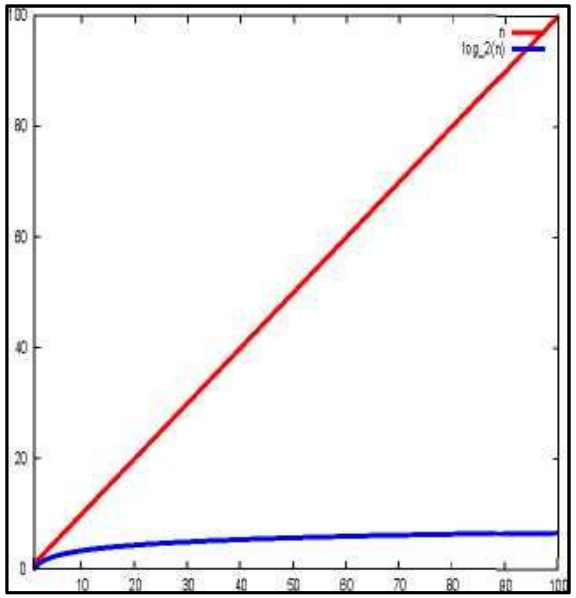
- En pratique, on se contente d'un ordre de grandeur.
- Pour déterminer si un algorithme est efficace, on compte le nombre d'opérations nécessaires à effectuer dans le pire des cas et en fonction de la taille de la donnée.
- Exemples d'opérations élémentaires comptées :
  - additionner, soustraire, multiplier ou diviser deux nombres,
  - tester si une valeur est égale à une autre valeur,
  - affecter une valeur à une variable.

Temps de calcul

- Exemple
  - avec l'organisation 1 pour une ville de **n maisons**, l'algorithme naturel pour trouver une maison a une complexité linéaire qu'on note : **O(n).**
  - avec l'organisation 2 pour une ville de **n maisons**, l'algorithme naturel pour trouver une maison a une complexité logarithmique qu'on note : **$O(\log_2(n))$.**
  - L'organisation 2 est beaucoup plus efficace pour trouver une maison.

Différence entre *n* et log *n*

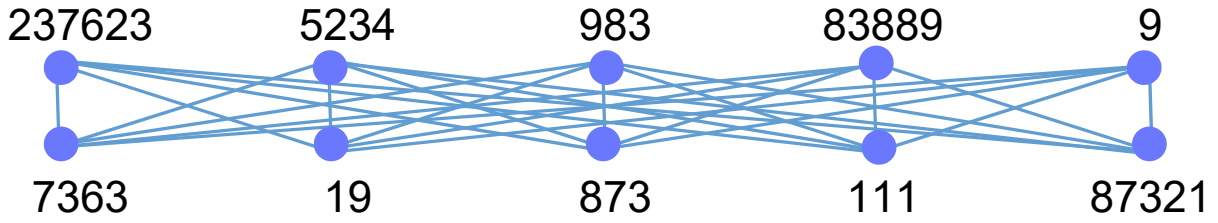
- Pour notre livreur de pizza :
  - Si $n = 10^6$, alors $\log_2(n) \approx 20$
    Il fait 50 000 fois moins de déplacements si les maisons sont organisés par « embranchements ».
  - Si $n = 10^9$, alors $\log_2(n) \approx 30$, il fait alors 30 000 000 fois moins de déplacements.



Temps de calcul : 2ème exemple

- Problème : déterminer si 2 ensembles *E1*, *E2* de *n* entiers ont une valeur commune.

- Algorithme 1 : comparer successivement chaque élément de *E1* avec chaque élément de *E2.* Il faudra environ n^2 comparaisons.



Temps de calcul : 2ème exemple

- Problème : déterminer si 2 ensembles *E1*, *E2* de *n* entiers ont une valeur commune.

- Algorithme 2 : Avec un structure de données adaptées, on peut résoudre le problème avec environ n*log(n) comparaisons .

|E1| = |E2|	Algorithme 1	Algorithme 2
n	n^2	n*log(n)
10	100	10
1000	1000000	3000
100000	10000000000	500000

Différence entre O(n2) et O(n log(n))

Sur un ordinateur exécutant une instruction élémentaire en 10^{-9} s.

- Si les ensembles *E1* et *E2* ont $n = 1\ 000\ 000 = 10^6$ elts
  - Exécuter $n * \log n$ instructions élémentaires nécessite 0,006s.
  - Exécuter n^2 instructions élémentaires nécessite 10^3s soit environ 16mn40s.
- Si les ensembles *E1* et *E2* ont $n = 10\ 000\ 000 = 10^7$ éléments
  - Exécuter $n * \log n$ instructions élémentaires nécessite 0,07s.
  - Exécuter n^2 instructions élémentaires nécessite 10^5s soit plus d'une journée.
- En informatique, on manipule parfois des ensembles énormes
  - Google indexe plusieurs 30 000 milliards de pages web,
  - Google reçoit près de 5,5 milliards de requêtes/jour.