Université Paris Cité

GÉOMÉTRIE ALGORITHMIQUE

Master Informatique
Parcours Vision et Machine Intelligente

F. CLOPPET

2022-2023

SOMMAIRE

- Informations pratiques
- Introduction
- Notions Algorithmiques
- Méthodes Algorithmiques pour la géométrie
- Modéliser *le monde*
- Méthodes géométriques
  - Notions de base en géométrie
  - Méthodes applicables aux modèles discrets
  - Méthodes applicables aux modèles continus

Informations Pratiques

- **Support de Cours**
  Sur moodle
  http://www.math-info.univ-paris5.fr/~cloppet/GeometrieAlgorithmique/

- **Contact**

  florence.cloppet@u-paris.fr

- **Bureau 804 I**

  - Pavillon Sappey 8ème étage

Planning

- Cours
  - 09h00-10h30 - salle Fourier F543

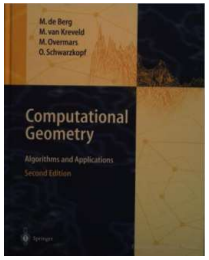
- TD
  - 10h45-12h15 – salle Fourier F 543

Modalités de contrôle des Connaissances

- Examen *E*
  - *CC1 (1h) 40% de la note finale*
  - *épreuve écrite terminale de (1h30) 60% de la note finale*

- **Aucun document autorisé lors des épreuves**

Bibliographie



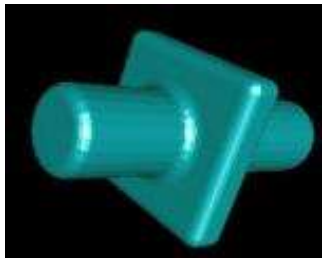
- Géométrie Algorithmique, J. D. Boissonnat, Mariette Yvinec, Ediscience International, 1995
- Computational Geometry: Algorithms and Applications, M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Springer Verlag 2nd Edition, 2000.
- Computational Geometry in C, Joseph O'Rourke,Cambridge University Press, 2nd edition, 1998.
- Géométrie discrète et images numériques, D. Coeurjolly, A. Montanvert, J.M. Chassery, Hermès- Lavoisier, 2007.
- Geometric Tools for Computer Graphics, P. J. Scheider, D. H. Eberly, Morgan Kaufmann Publishers, Elsevier Science, 2003
- Algorithmes pour la synthèse d'images et l'animation 3D, Remy Malgouyres, Dunod Edition

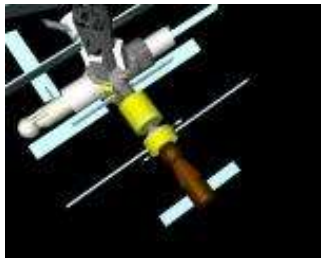
SOMMAIRE

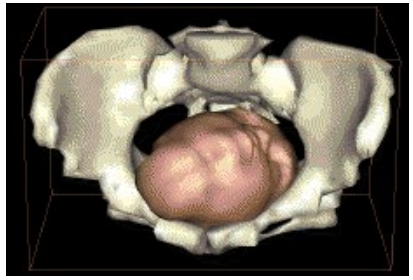
- Informations pratiques
- Introduction
- Notions Algorithmiques
- Méthodes Algorithmiques pour la géométrie
- Modéliser *le monde*
- Méthodes géométriques
  - Notions de base en géométrie
  - Méthodes applicables aux modèles discrets
  - Méthodes applicables aux modèles continus

Introduction (1)

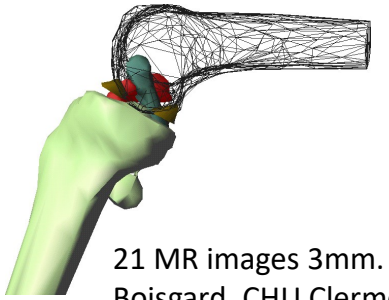
- Nombreux domaines d'application où il faut savoir construire et traiter efficacement des objets de nature géométrique
  - La robotique
  - La vision par ordinateur
  - L'informatique graphique
  - La réalité virtuelle
  - CAO
  - Imagerie médicale







23 MR images 8mm. Courtesy of Frank Muennemann, Resonex Inc. CA. : 6600 triangles and 2900 vertices



21 MR images 3mm. Courtesy of Dr. Stephane Boisgard, CHU Clermont Ferrand

Introduction (2)

- 1ers résultats de nature constructive en géométrie => Euclide
- Développements remarquables au 19ème S
- Mais pas de conception et d'analyse systématique des algorithmes géométriques

Introduction (3)

- Naissance de la géométrie algorithmique (autour des années 1975)
- Discipline à la frontière de la géométrie et de l'algorithmie
  - Construire et traiter de manière efficace des objets de nature géométrique

  ⇔ Conception et analyse d'algorithmes géométriques

Introduction (4)

- Apport majeur de la géométrie algorithmique
  - Rôle central joué par
    - Petit nombre de structures géométriques fondamentales
      polytopes, triangulations, diagrammes de Voronoï
    - Leur lien avec de très nombreux problèmes
  - Ajout de techniques proprement géométriques aux grands paradigmes de l'algorithmique générale
    - Technique de Balayage (Algorithme de Bentley et Ottman) pour calculer les intersections d'un ensemble de segments du plan

Introduction (5)

- Des algorithmes aux programmes
  - Deux points majeurs
    - Précision finie des calculateurs
      - algorithmes sont conçus et analysés dans le cadre d'un modèle abstrait de calculateur (données sont des nbs réels et opérations sont effectuées exactement)
      - Pas le cas de la réalité : implantation naïve d'un algo utilisant la représentation flottante des nombres réels peut => erreurs fatales
    - Traitement des cas particuliers (cas dégénérés ne sont généralement pas exposés dans l'algorithme général)

SOMMAIRE

- Informations pratiques
- Introduction
- Notions Algorithmiques
- Méthodes Algorithmiques pour la géométrie
- Modéliser *le monde*
- Méthodes géométriques
  - Notions de base en géométrie
  - Méthodes applicables aux modèles discrets
  - Méthodes applicables aux modèles continus

Rappels (1)

- Définition d'un critère pour
  - Mesurer l'efficacité d'un algorithme
  - Comparer plusieurs algorithmes entre eux

- Performances évaluées en termes de
  - Temps de calcul
  - Place mémoire nécessaire

☞Dépendent de la machine utilisée, du langage de programmation

Rappels (2)

- Évaluer les performances d'un algo par rapport à une machine abstraite idéale
  - Modèle de calculateur (machine abstraite idéale)
    - Définition d'une unité de temps
      - Opérations élémentaires exécutées en une unité de temps

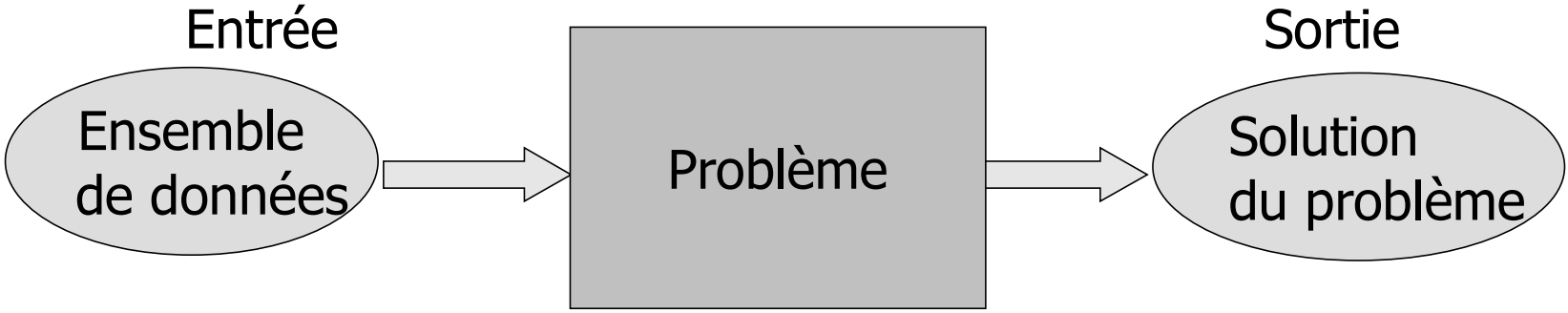
      ☞ *Complexité en temps = nb d'opérations élémentaires nécessaires à l'exécution du programme qui code l'algo*

    - Définition d'une unité mémoire
      - Variables dites élémentaires qui peuvent être représentées dans une unité mémoire

      ☞ *Complexité en espace mémoire = nb d'unités mémoires requises pour l'exécution du programme*

Rappels (3)

- Complexité(s)



  - Taille de l'entrée = *nb d'unités mémoires nécessaires pour représenter cette* entrée
    - Si chaque donnée est élémentaire => taille de l'entrée est proportionnelle au cardinal de l'ensemble des données
  - Nb d'opérations élémentaires dépend surtout de la taille de l'entrée mais également de l'ensemble de données lui-même

Rappels (4)

- Complexités
  - Complexité au pire (*complexité en temps dans le cas le pire*)
    - ☞ Fonction f(n) qui donne la borne supérieure du nombre d'opérations élémentaires effectuées par l'algorithme lorsque taille(entrée) = n
    - ☞ Mesure pessimiste de l'efficacité d'un algo
    - ☞ Borne sup rarement atteinte (ens de données très particuliers dont l'occurrence est marginale ou peut être évitée par un pré-traitement)
  - Complexité en moyenne
    - ☞ Fonction g(n) qui donne la moyenne du nombre d'opérations élémentaires effectuées si on suppose un loi de probabilités sur les ensembles de données de taille n
    - ☞ Plus difficile à évaluer que la complexité au pire
    - ☞ Dépend de la mesure de probabilités choisie sur l'espace des entrées de taille n (doit refléter de façon réaliste l'usage qui est fait de l'algo)

Rappels (5)

- Définitions analogues pour complexité en espace mémoire au pire ou en moyenne

- La plupart du temps : complexité au pire d'un algo qui est évaluée

- Complexité fonction de la taille de la sortie
  - nb d'unités mémoires pour représenter le résultat
  - Taille de la sortie fonction de la taille de l'entrée et de l'ensemble de données lui-même
  - Taille de la sortie au pire s(n)= le max de la taille de la sortie pour toutes les entrées de taille n

Rappels (6)

- Complexité fonction de la taille de la sortie

  - Algorithmes adaptatifs : complexité est fonction de la taille de la sortie correspondant à l'entrée traitée et non de la taille de la sortie dans le cas le pire

  - Analyse d'un algo adaptatif est fonction de 2 variables n (taille de l'entrée) et s (taille de la sortie)

  - Complexité au pire d'un algo adaptatif = f(n,s) qui donne la borne sup du nb d'opérations élémentaires effectuées pour tous les ensembles de données correspondant à une entrée de taille n et une sortie de taille s

Rappels (7)

- Pré-traitement sur l'ensemble de données
  - Parfois si plusieurs requêtes du même type concernant un même ensemble de données
    - ☞ Pré-traitement qui construit une structure de données permettant de répondre efficacement aux requêtes
    - ☞ Analyse séparée de la complexité du pré-traitement et du traitement des requêtes
    - ☞ Si structure de données semi-dynamique (ajout), ou dynamique(ajout + suppression) => opérations (insertion, suppression, requête) ont un coût

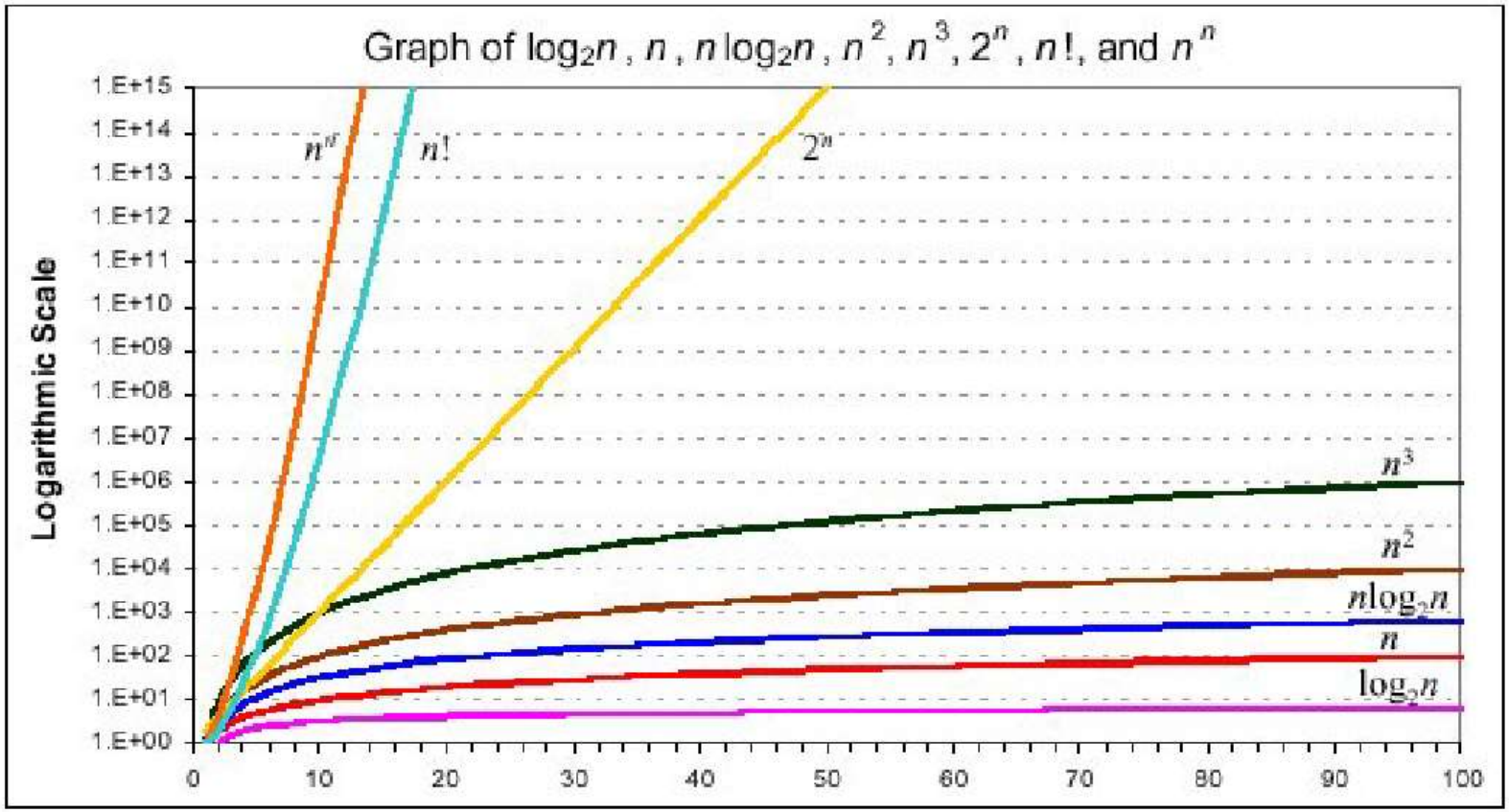
      complexité amortie : (complexité de n opérations)/n opérations

Rappels (8)

- Comportement asymptotique
  - Manière dont la complexité croît en fonction de la taille de l'entrée ($n \Rightarrow \infty$)
  - Ordre de grandeur du comportement asymptotique
    - On borne le terme dominant de la complexité en temps de calcul et en espace mémoire et on néglige les constantes numériques
    - $1, \log(n), n, n \log(n), n^2, n^3, \ldots 2^n$

Rappels (9)

- Comportement asymptotique



Rappels (10)

- Comportement asymptotique

temps / instruction
1,00E-07 seconde

n	log(n)	n	n*log(n)		n^2		n^2 * log(n)		n^3		2^n	
10	0,0000	0,0000	0,0000		0,0000		0,0000		0,0001		0,0001	
20	0,0000	0,0000	0,0000		0,0000		0,0002		0,0008		0,1049	
30	0,0000	0,0000	0,0000		0,0001		0,0004		0,0027		2	mn
40	0,0000	0,0000	0,0000		0,0002		0,0009		0,0064		1	jour
50	0,0000	0,0000	0,0000		0,0003		0,0014		0,0125		4	années
60	0,0000	0,0000	0,0000		0,0004		0,0021		0,0216		37	siècles
70	0,0000	0,0000	0,0000		0,0005		0,0030		0,0343		4	millions d'années
100	0,0000	0,0000	0,0001		0,0010		0,0066		0,1000			
1000	0,0000	0,0001	0,0010		0,1000		0,9966		2	mn		
10 000	0,0000	0,0010	0,01		10,0000		2	mn	1	jour		
100 000	0,0000	0,0100	0,17		17	mn	5	heures	3	années		
1 000 000	0,0000	0,1000	1,99		1	j	23	jours	32	siècles		
10 000 000	0,0000	1,0000	23,25		4	mois	7	années				
100 000 000	0,0000	10,0000	4	mn	32	années	9	siècles				
1 000 000 000	0,0000	100,0000	50	mn	32	siècles						

Rappels (11)

- Comparaison des ordres de grandeur asymptotique

  f et g : 2 fonctions de n (variable à valeurs réelles)

  - $f(n) = O(g(n))$ ssi $\forall n \geq n_0$, $f(n) \leq cg(n)$ avec c: cste réelle

    $\Rightarrow$ complexité de f(n) majorée pour une fonction connue g(n)

  - $f(n) = \Omega(g(n))$ ssi $\forall n \geq n_0$, $f(n) \geq cg(n)$ avec c: cste réelle

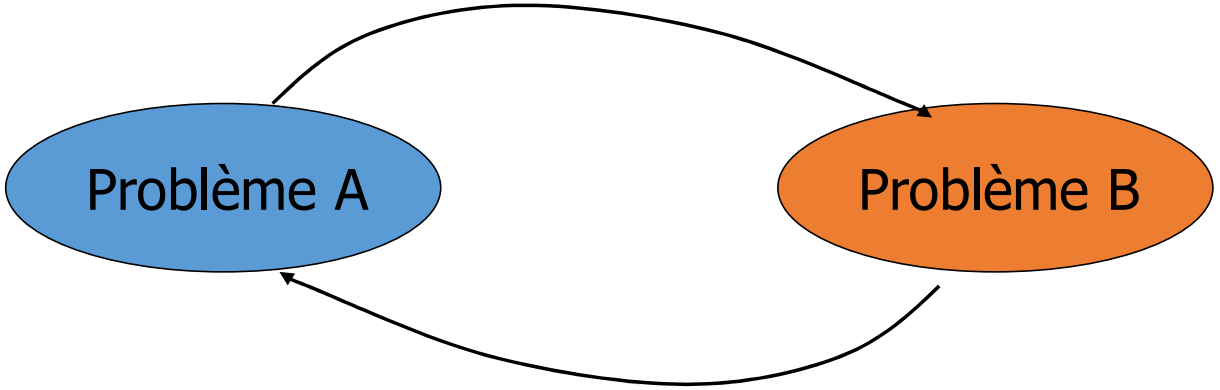
    $\Rightarrow$ complexité de f(n) minorée pour une fonction connue g(n)

  - $f(n) = \theta(g(n))$ ssi $\forall n \geq n_0$, $c_1 g(n) \leq f(n) \leq c_2 g(n)$ avec c_1, c_2 : cstes réelles

    $\Rightarrow$ complexité de f(n) connue pour une fonction connue g(n)

Rappels (12)

- Méthode de transformation
  - Ramener 1 problème à un autre problème dont la complexité est connue



- A est transformé en temps $\tau(n)$ si
  - entrée de A peut être convertie en 1 entrée pour B en $\tau_1(n)$ opérations élément.
  - solution de B peut être transformée en solution de A en $\tau_2(n)$ opérations élément.
  - $\tau_1(n) + \tau_2(n) = O(\tau(n))$

Rappels (13)

- Méthode de transformation (suite)
  - Si 1 pb A de complexité f(n) est transformable en temps $\tau(n)$ en un problème B de complexité g(n)

    Alors
    - $f(n) = O(g(n) + \tau(n))$

      $\Rightarrow$ complexité de B fournit un majorant pour celle de A
    - $g(n) = \Omega(f(n) - \tau(n))$

      $\Rightarrow$ complexité de A fournit un minorant pour celle de B

Structures de données fondamentales

- Structure de données = brique avec laquelle est construite l'édifice algorithmique

- Définition de structures de base supportant des fonctions précises

- Structures de données courantes peuvent être combinées ou assemblées pour former des structures de données géométriques

Structures de données / Listes

- Définition
  - une liste linéaire l est une suite finie éventuellement vide d ’éléments repérés selon leur rang dans la liste

- Remarques
  - ordre sur les places des éléments et non sur les éléments

  - il existe une fonction de succession ***succ*** telle que toute place soit accessible en appliquant ***succ*** de manière répétée à partir de la première place de la liste

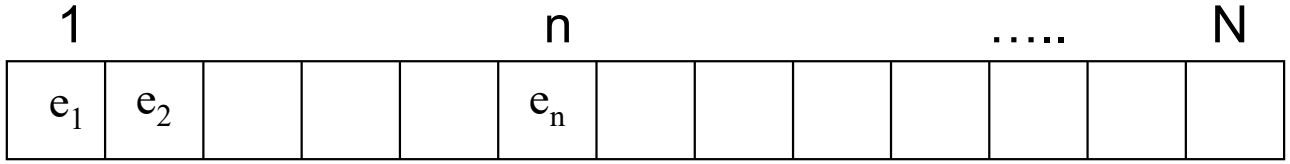
- Opérations de base effectuées sur les listes
  - accéder au k ième élément
  - insérer un nouvel élément après la k ième place
  - supprimer le k ième élément

Structures de données / Listes / Implantation

- Représentation contiguë en mémoire
  - liste représentée par un tableau dont la $i^{ème}$ case est la $i^{ème}$ place de la liste
  - taille du tableau doit être très supérieure à la longueur de la liste pour pouvoir insérer des éléments

    => surdimensionnement du tableau

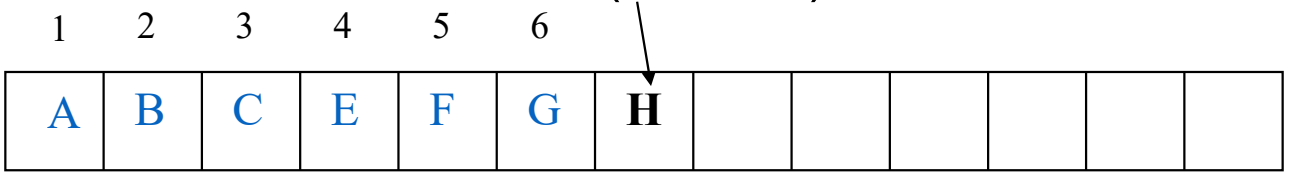
    => pour prendre en compte les éléments de la liste et pas toutes les cases du tableau, il faut connaître la longueur de la liste



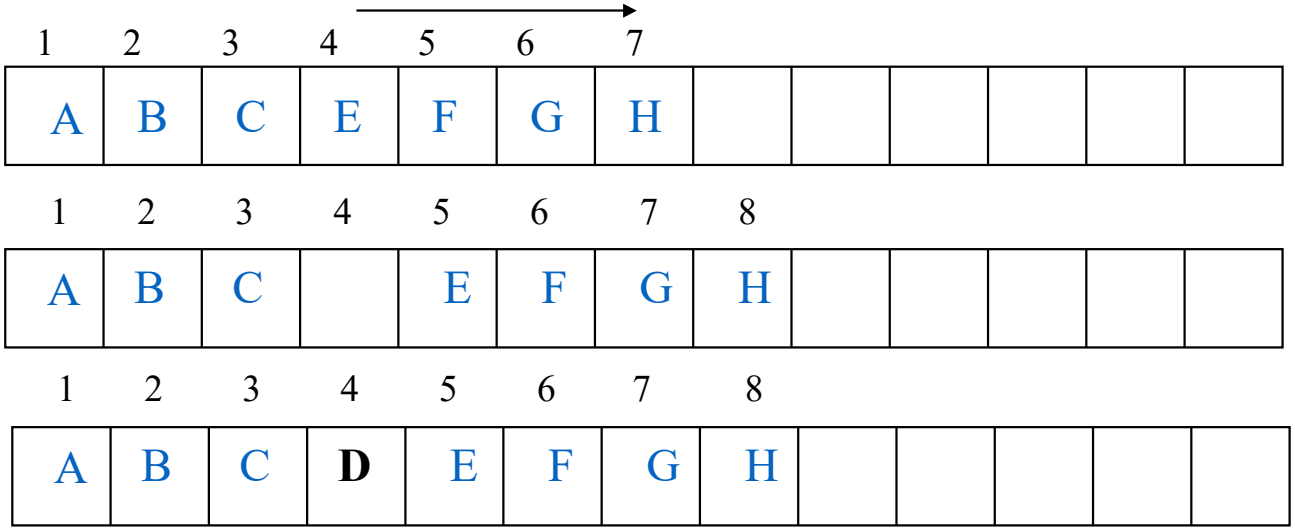
  - l 'opération succ est représentée par la succession des cases du tableau en mémoire

Structures de données / Listes / Implantation

- Opération d ’insertion
  - en fin de liste : ex: inserer (l, 7, ‘ H’)



  - en début ou milieu de liste : ex: inserer (l, 4, ‘ D’)

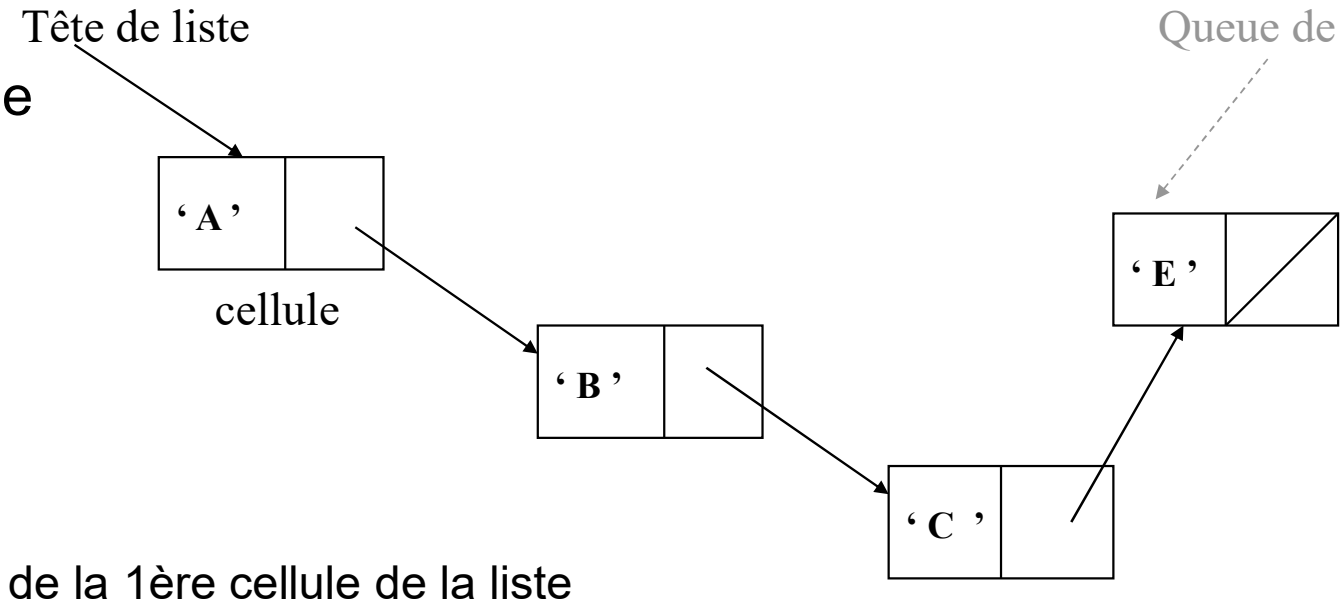


Structures de données / Listes / Implantation

- Avantages Représentation contiguë
  - accès direct
  - parcours séquentiel de la liste facile
  - insertion et suppression sur le dernier élément est simple
- Inconvénients Représentation contiguë
  - il faut majorer la taille des listes
  - insertion et suppression ailleurs qu'en fin de liste sont coûteuses

Structures de données / Listes / Implantation

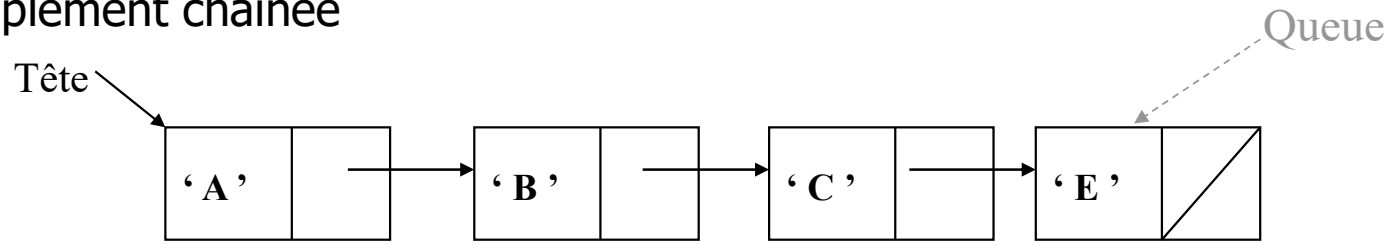
- Représentation chaînée



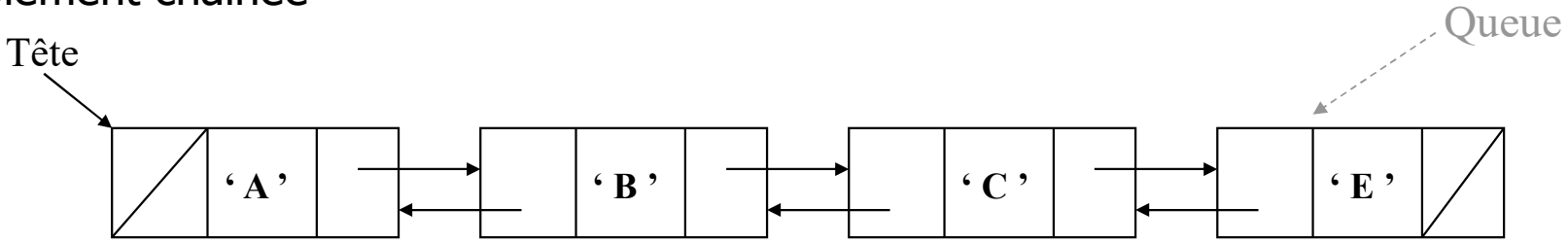
  - Tête de liste = adresse de la 1ère cellule de la liste
  - une cellule contient au minimum 2 champs
    - un champ **info** qui est l ’élément stocké dans la liste
    - un champ **pointeur** qui contient l ’adresse de la prochaine cellule
    - fin de liste est matérialisée par champ **pointeur = NULL**

Structures de données / Listes / Implantation

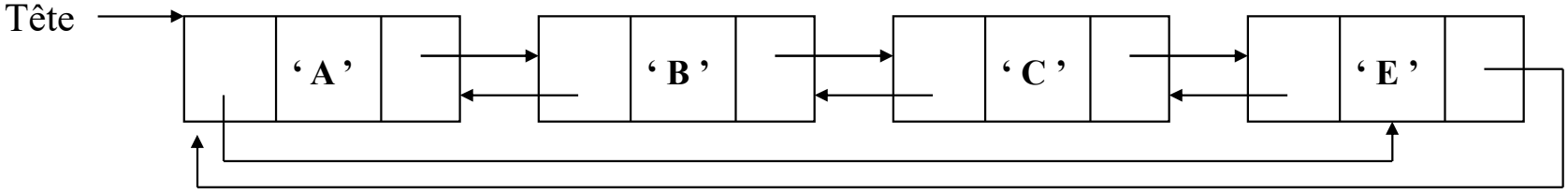
- Il existe plusieurs types de listes chaînées
  - simplement chaînée



  - doublement chaînée

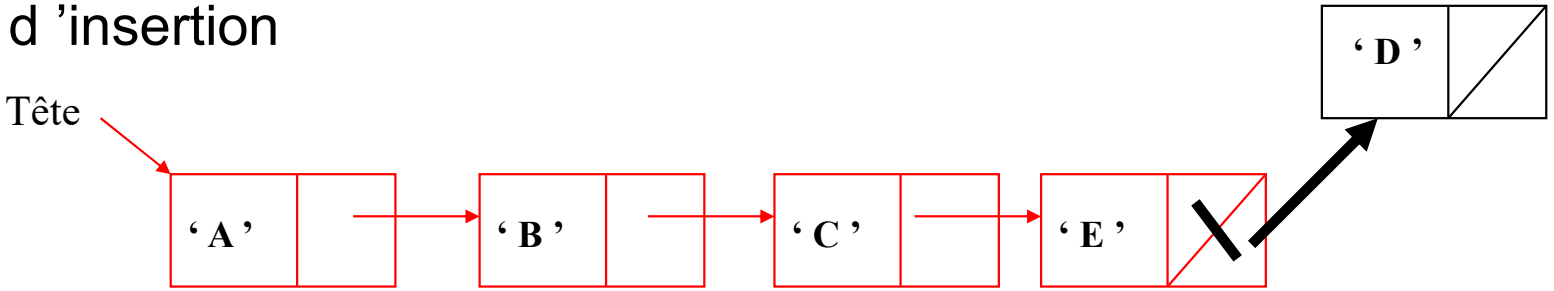


  - circulaire (simplement ou doublement chaînée)

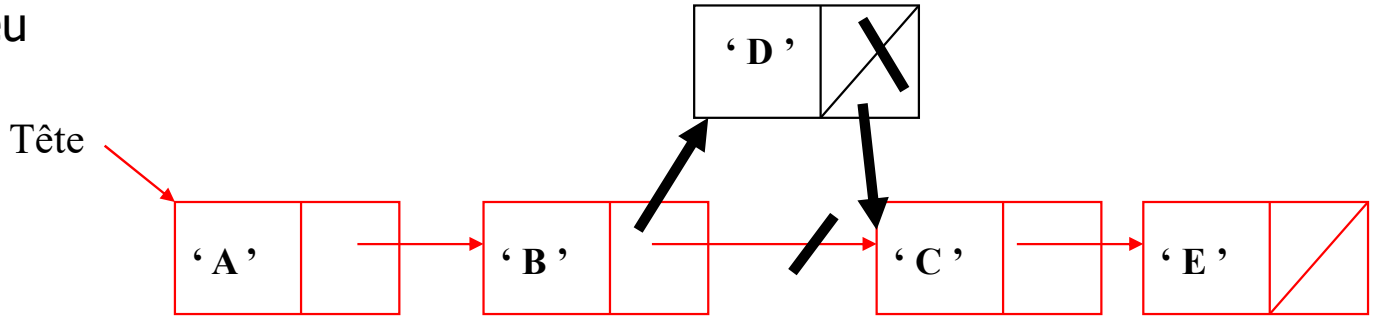


Structures de données / Listes / Implantation

- Opération d ’insertion
  - en fin

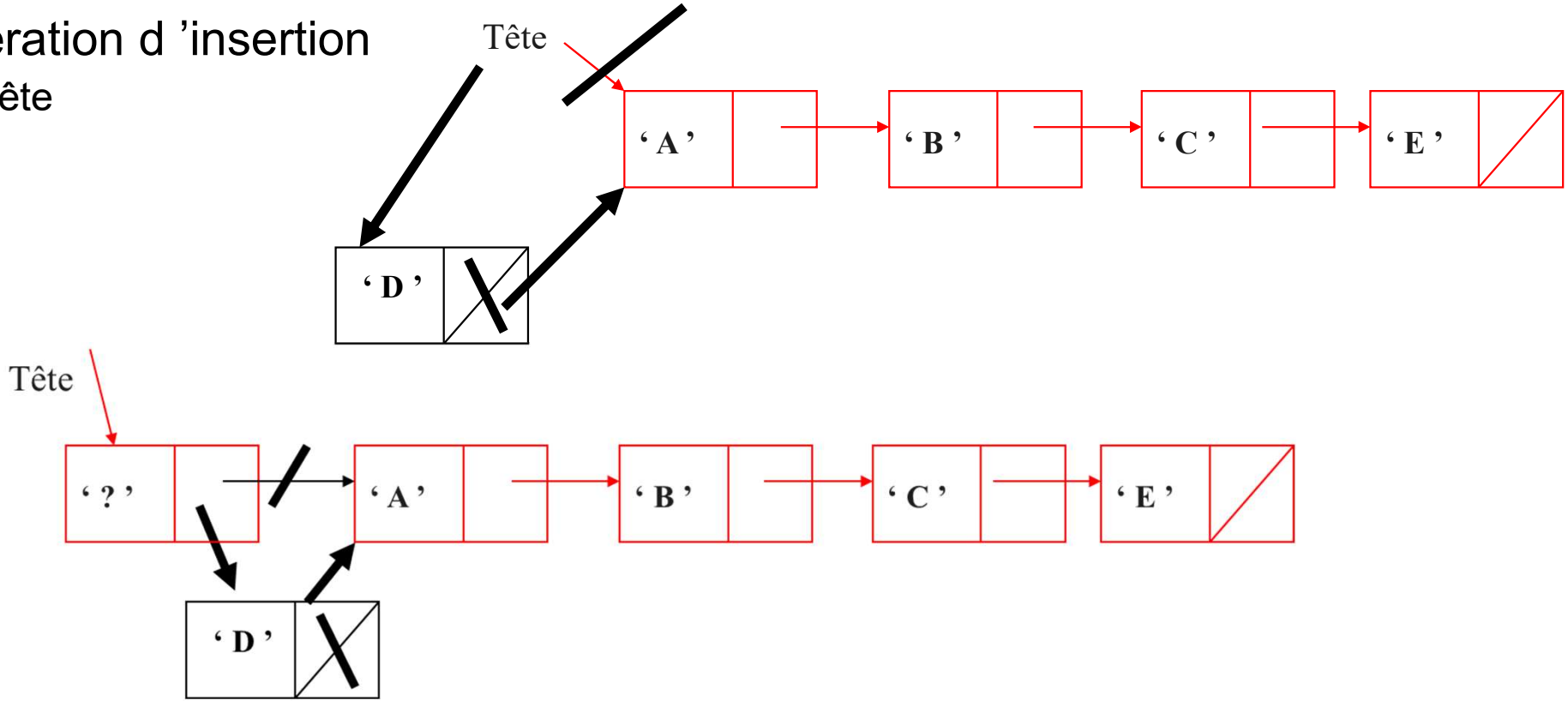


  - en milieu



Structures de données / Listes / Implantation

- Opération d ’insertion
  - tête



Structures de données / Listes / Implantation

- Avantages Représentation chaînée
  - permet de faire évoluer la liste en fonction des besoins de l ’application
    => pas de surdimensionnement

  - insertion ou suppression sont peu coûteuses quelle que soit la place où elles ont lieu

- Inconvénients Représentation chaînée
  - pas d ’accès direct => parcours peut être relativement coûteux

- Remarque:
  - faire attention de ne jamais perdre le point d ’entrée dans la liste

Structures de données / Listes / Implantation

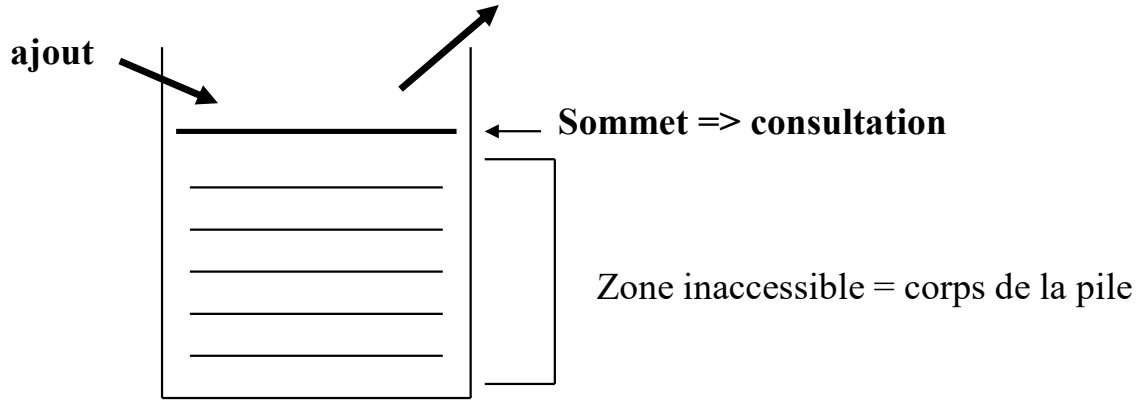
- Complexité : Représentation chaînée
  - Espace mémoire occupé O(n) si liste a n élts
  - Opérations successeur, prédécesseur (si liste doublement chaînée), insertion, suppression sont effectuées en temps constant

☞ 1 liste peut être créée et parcourue en O(n) si n est la longueur de la liste

  - Si pointeur sur 1er et dernier éléments, les opérations de concaténation et partition sont effectuées en temps constant

Struct. de données/ Listes/Cas particuliers

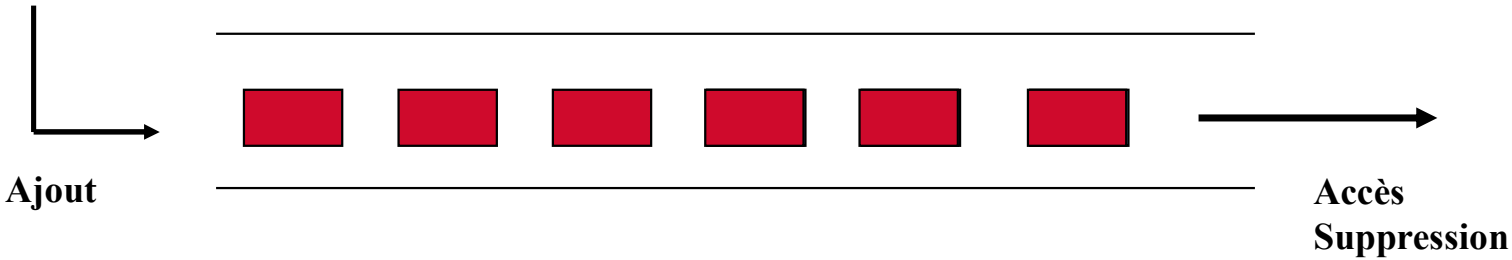
- Pile = structure de type LIFO (Last In First Out)



- Insertions et suppressions ne se font qu'à une seule extrémité de la liste
  => sommet de la pile

Struct. de données/ Listes/Cas particuliers

- File = structure de type FIFO (First In First Out)



- Les insertions se font à une extrémité
- Les accès et les suppressions se font à l ’autre extrémité de la liste

Struct. de données/ Dictionnaires - Queues de priorité

- Représente un ensemble S sous ensemble d'un Univers U
- Opérations minimales
  - Recherche : *étant donné x de U, est ce que x appartient à S ?*
  - Ajout: *ajouter un élément x de U à l'ensemble S*
  - Suppression: *supprimer un élément x de l'ensemble S*

Struct. de données/ Dictionnaires - Queues de priorité

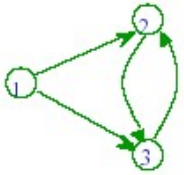
- Si U est totalement ordonné et S sous-ens fini de U
  => Opérations supplémentaires
  - Localisation: étant donné x de U, rechercher le plus petit élément y de S tel que $x \leq y$
  - Minimum: retrouver le plus petit élément de S
  - Maximum: retrouver le plus grand élément de S
  - Prédécesseur : rechercher l'elt de S précédant un elt x de S donné
  - Successeur : rechercher l'elt de S suivant un elt x de S donné

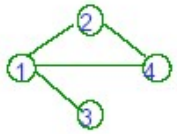
Struct. de données/ Dictionnaires Queues de priorité

- Dictionnaire
  - Structure qui permet les recherches, les insertions et les suppressions
- Queue de priorité
  - Structure qui permet les recherches, les insertions et les suppressions + opération minimum
- Dictionnaire augmenté
  - Structure qui permet toutes les opérations (*Recherche, insertion, suppression, localisation, minimum, maximum, prédécesseur, successeur*)

Struct. de données/ Graphes

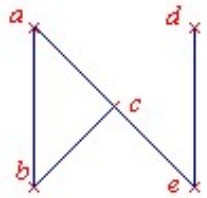
- Graphes
  - Couple (X,A) où X est un ens fini d'élts (nœuds)
    A est un ens de paires de nœuds (arcs)
  - Orienté

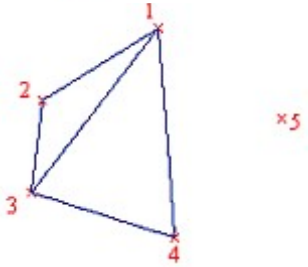




Graphe non orienté :
relation symétrique
entre les nœuds

  - Connexe



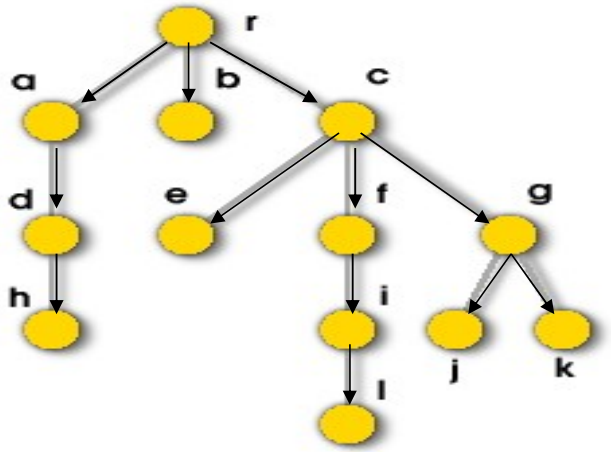


Graphe non connexe :
il n'existe aucun chemin
de 1 à 5, par exemple.

  - Cycle
    - Chemin = liste de sommets telle qu'il existe dans le graphe une arête entre chaque paire de sommets successifs
    - Cycle = chemin finissant à son point de départ

Struct. de données/Arbres

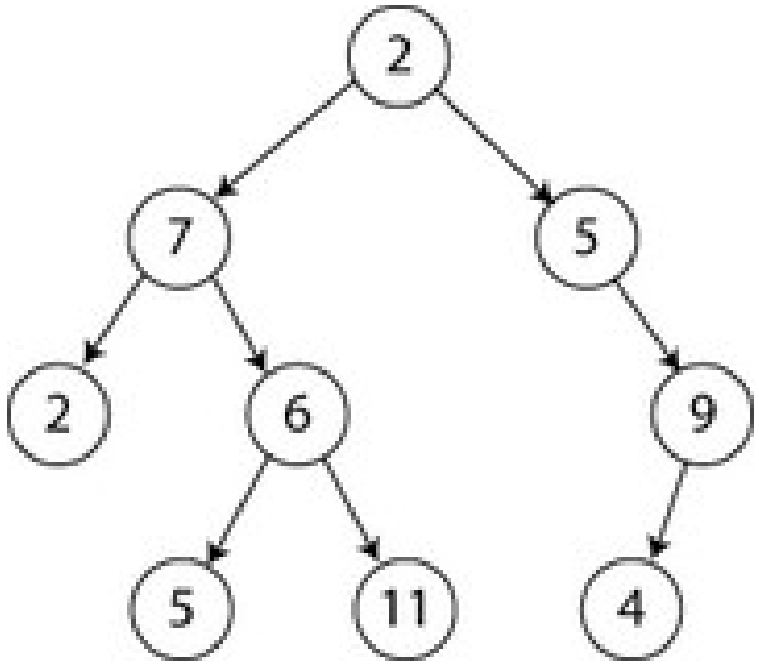
- Arbre
  - Graphe orienté connexe et sans cycle



  - Cas particuliers : arbres équilibrés
    - Implantation efficace des dictionnaires et queues de priorité

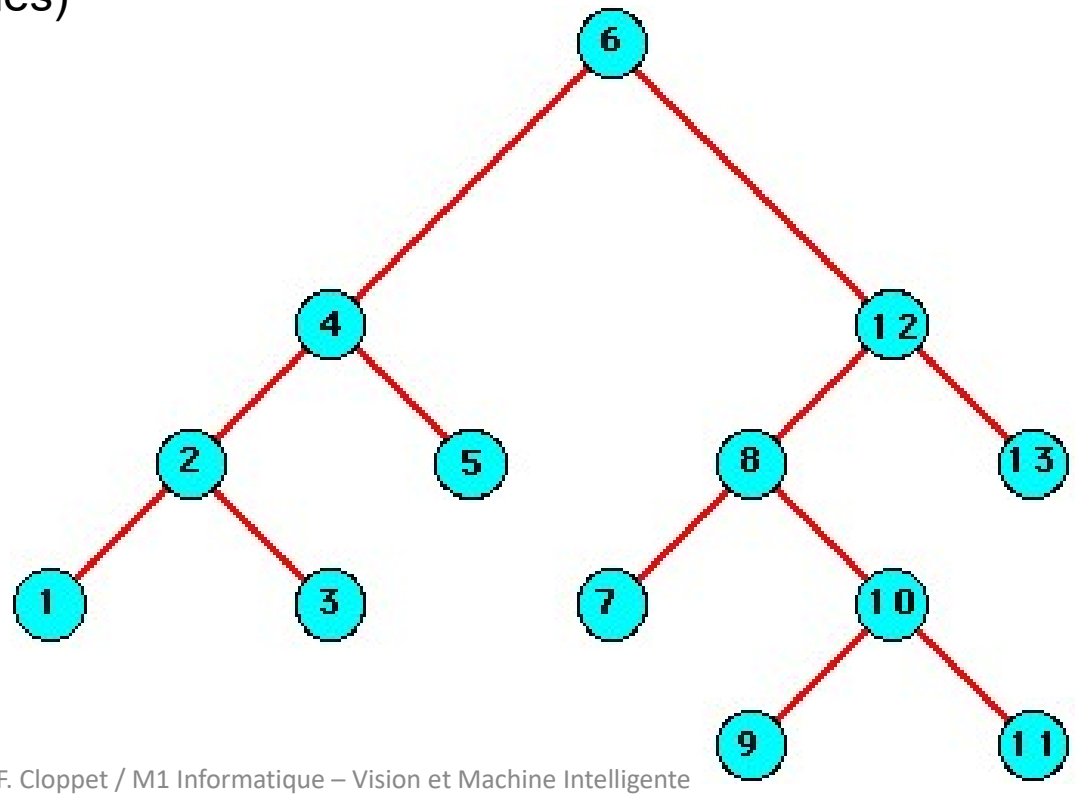
Struct. de données/Arbres équilibrés

- Arbre binaire
  - Arbre dans lequel chaque nœud au plus 2 fils



Struct. de données/Arbres équilibrés

- Arbre binaire de recherche
  - Sous arbre gauche (resp. droit) de tout nœud ne contient que des valeurs strictement plus petites (resp. grandes)



Struct. de données/Arbres équilibrés

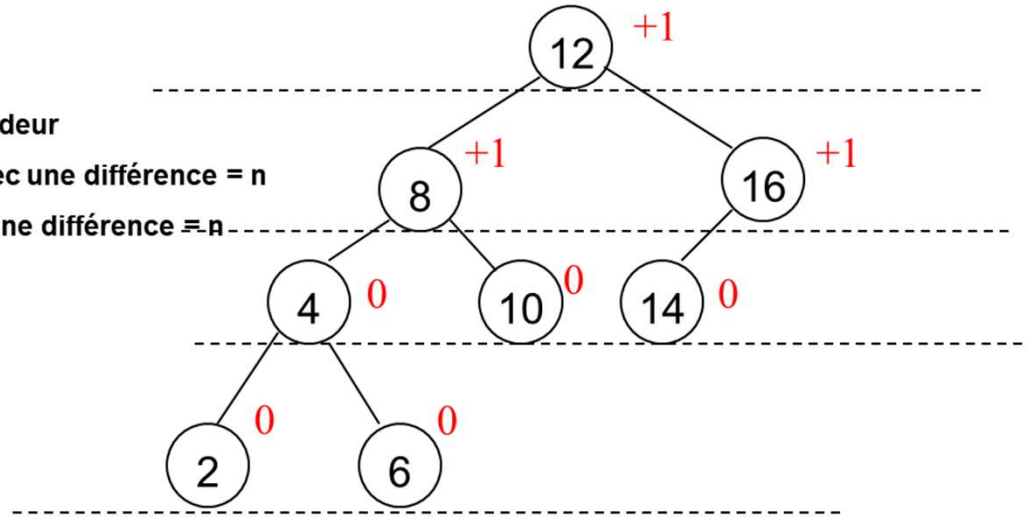
- AVL en 1962 par Adelson-Velskii et Landis
  - Différence entre les hauteurs des fils gauche et droit de tout nœud ne peut excéder 1

Pour chaque noeud on mettra

0 si ses deux sous-arbres ont la même profondeur

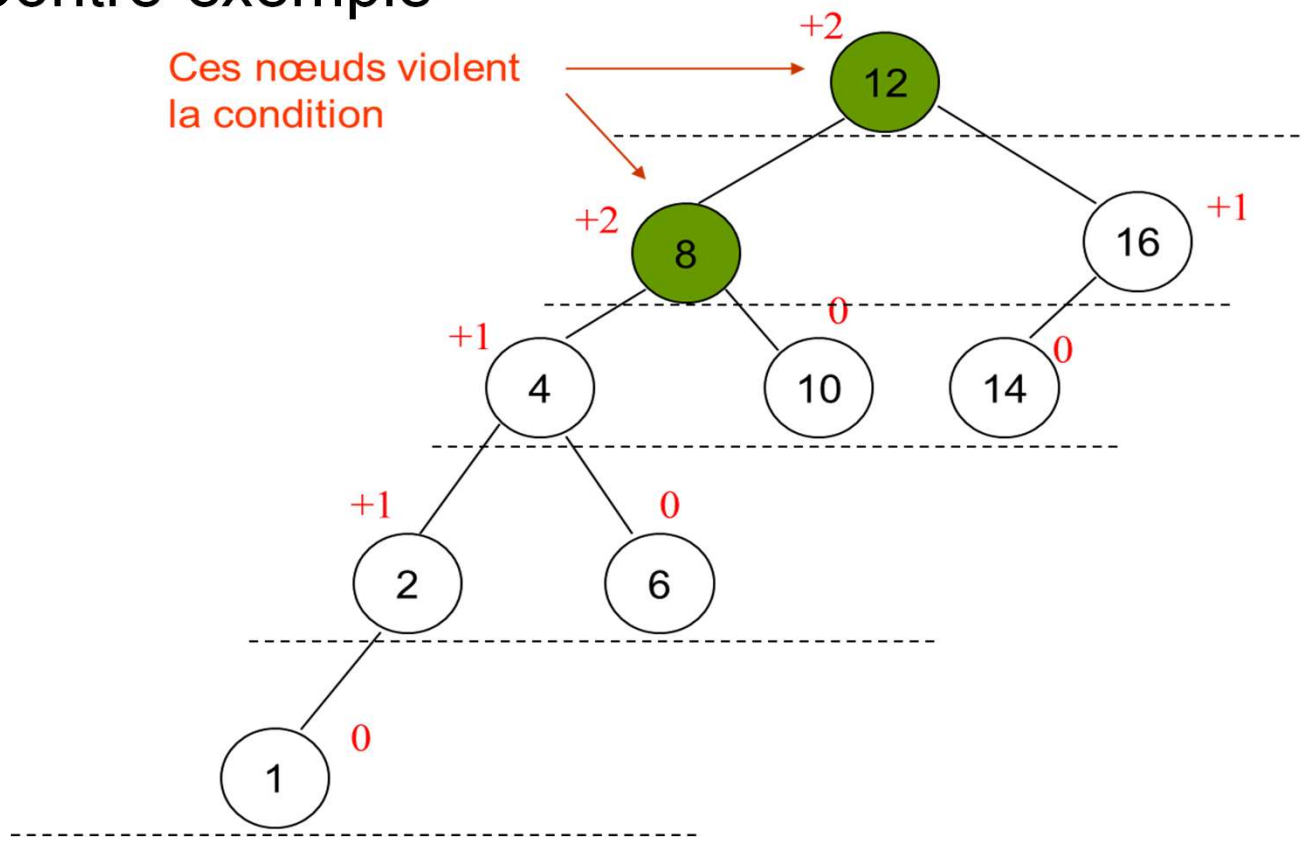
+n si le sous-arbre gauche est plus profond avec une différence = n

-n si le sous-arbre droit est plus profond avec une différence = n



Struct. de données/Arbres équilibrés

- AVL : Contre-exemple

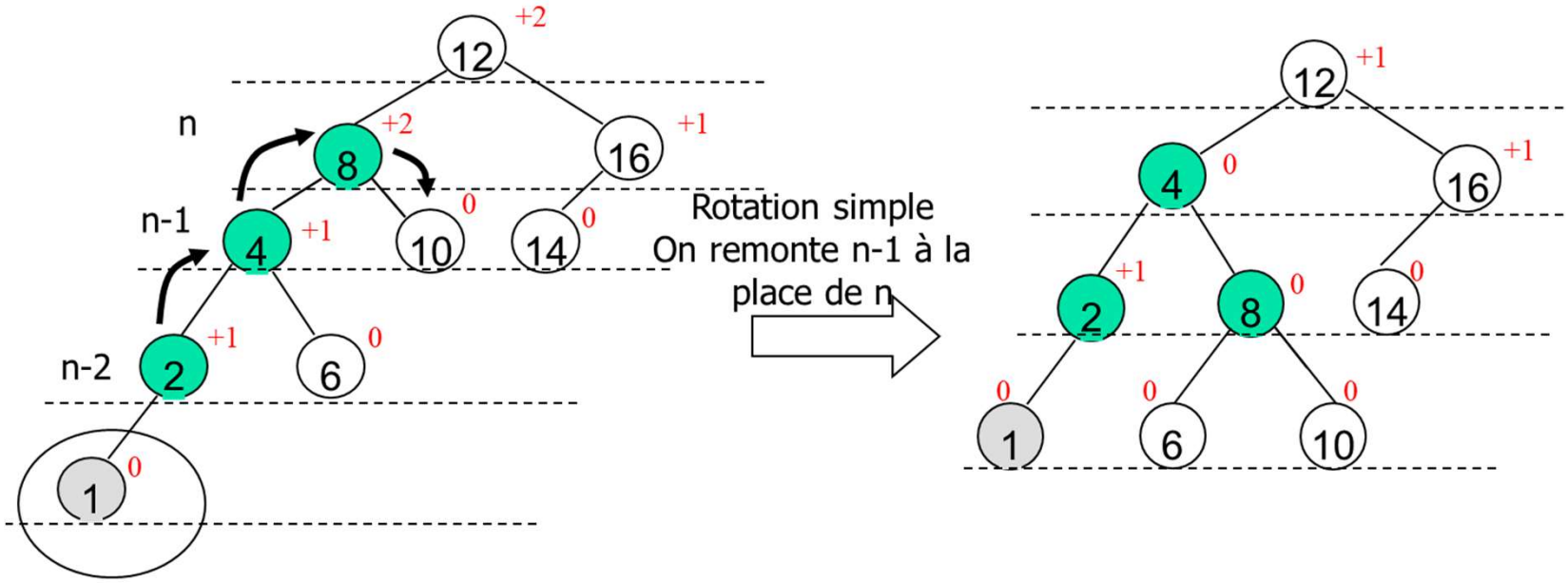


Struct. de données/Arbres équilibrés

- Il faut, après chaque insertion ou retrait, rétablir l'équilibre s'il a été rompu par l'opération
- Observation importante: après une insertion, seuls les nœuds qui sont sur le chemin du point d'insertion à la racine sont susceptibles d'être déséquilibrés
- Deux cas: nœud n où déséquilibre observé
  - insertion dans le sous-arbre de gauche du fils gauche du nœud n ou dans le sous-arbre de droite du fils droit de n
    ⇒ Simple rotation
  - insertion dans le sous-arbre de droite du fils gauche de n ou dans le sous-arbre de gauche du fils droit de n
    ⇒ Double rotation

Struct. de données/Arbres équilibrés

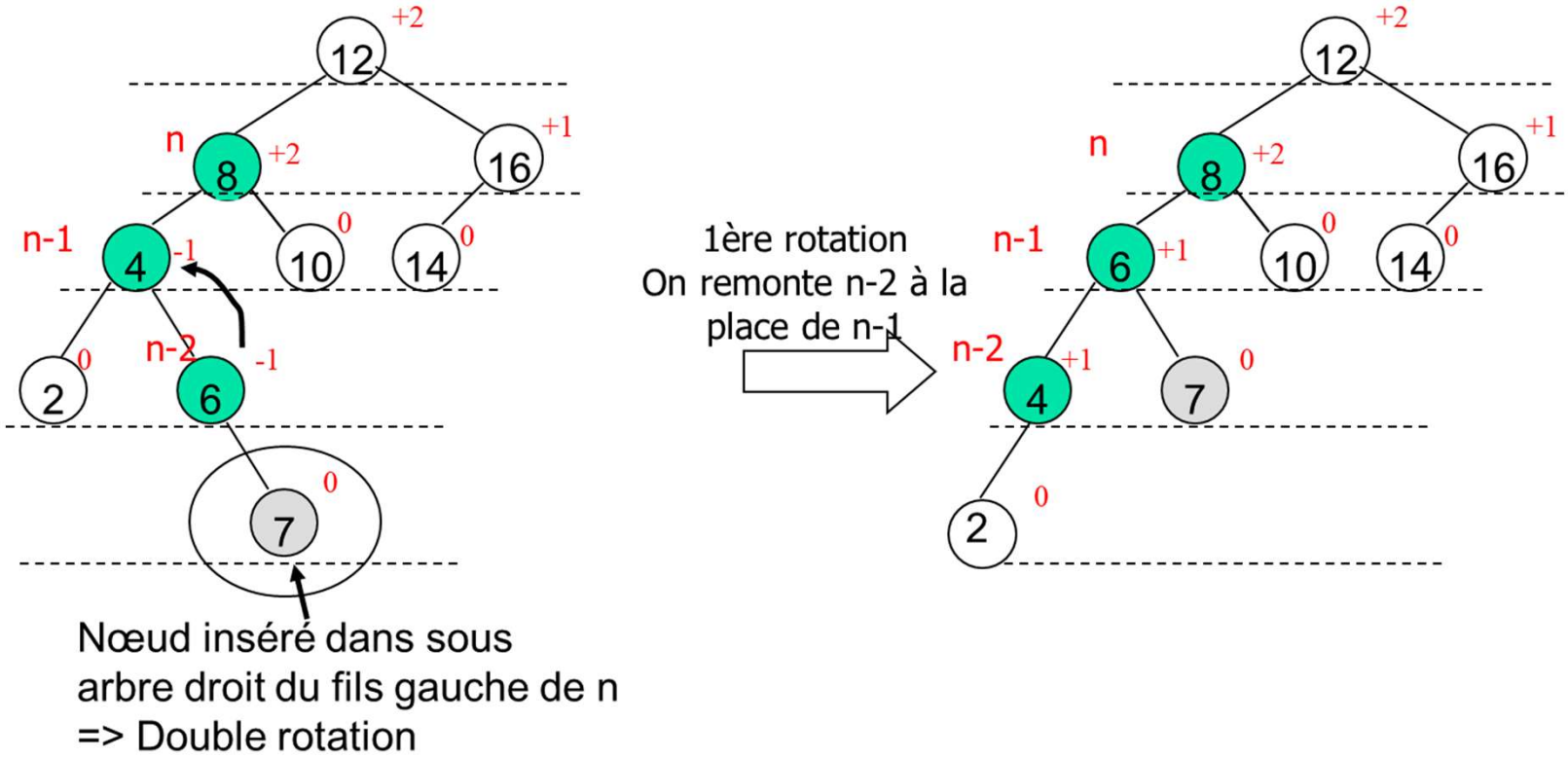
- Rotation simple



Nœud inséré dans sous arbre gauche du fils gauche de n
=> Déséquilibre corrigé par une rotation simple

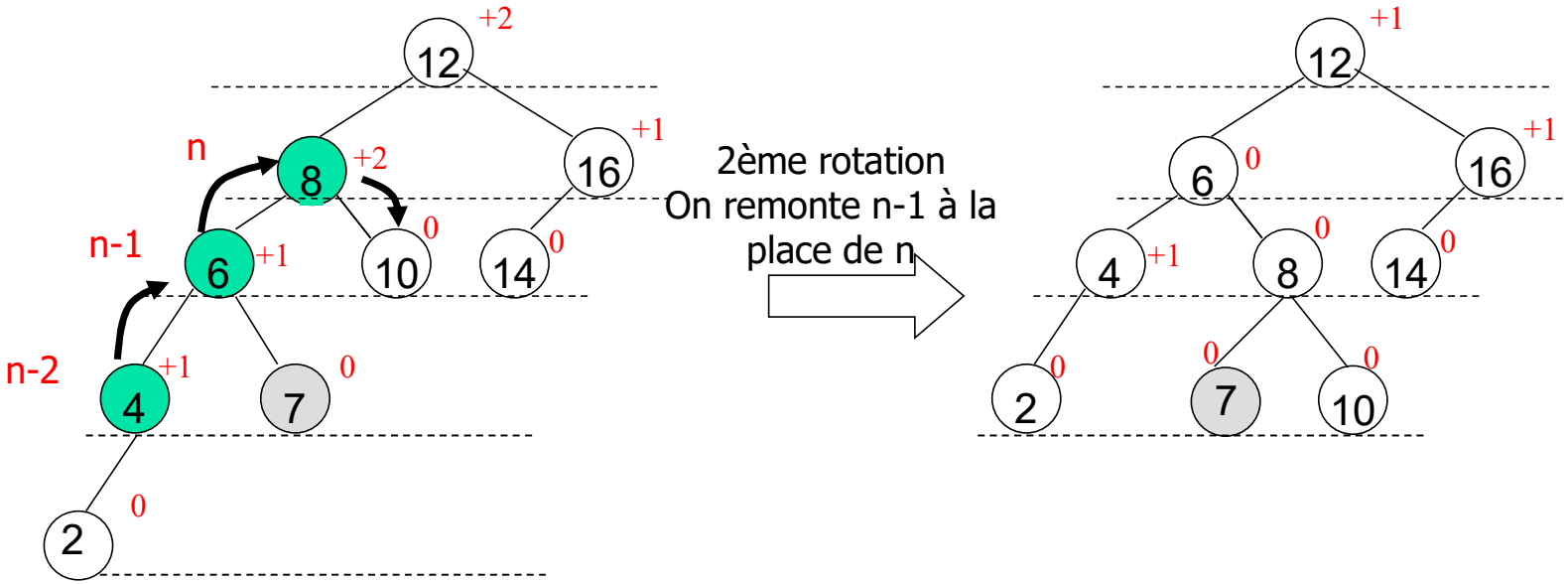
Struct. de données/Arbres équilibrés

- Rotation double



Struct. de données/Arbres équilibrés

- Rotation double (suite)



Struct. de données/Arbres équilibrés

- Arbre binaire de recherche – Analyse
  - Le prix d'une opération (recherche, insertion, retrait) est proportionnel au nombre de nœuds visités
  - Cas le pire : arbre binaire où chaque nœud n'a qu'1 seul successeur droit ou gauche (revient à 1 liste)

    => complexité linéaire O(n) (n: nb de nœuds)

  - Cas le meilleur : arbre binaire équilibré
    - On peut montrer que la hauteur maximale h d'un arbre équilibré comprenant n nœuds

    $\Rightarrow h \leq 1{,}44 * \log_2(n+1) - 0{,}328$

    $\Rightarrow$ Complexité au pire cas des AVL : O(log(n))

n	log (n+1)
1	1
3	2
7	3
1 023	10
1 048 575	20

  - Donc, le coût est O(log n) dans le meilleur cas et O(n) dans le pire cas
    - Si nb de données important => utilisation des arbres

Struct. de données/Arbres équilibrés

- Pour info
  - Il y a d'autres types d'arbres équilibrés plus facile à implémenter (pour éviter rotations)
    - Arbres 2-3 ou 2-3-4 :
      - implémentation plus simple du même principe que AVL mais avec un d° supplémentaire de liberté (nb de fils : 2 ou 3 ou 2 ou 3 ou 4) => éclatements au lieu de rotation
      - Rotations lors des ré-équilibrages remplacés par des éclatements
    - Arbres rouges et noirs
      - Arbre 2-3-4 implémenté à l'aide d'arbre binaire bicolore

        liens frères sont colorés en rouge