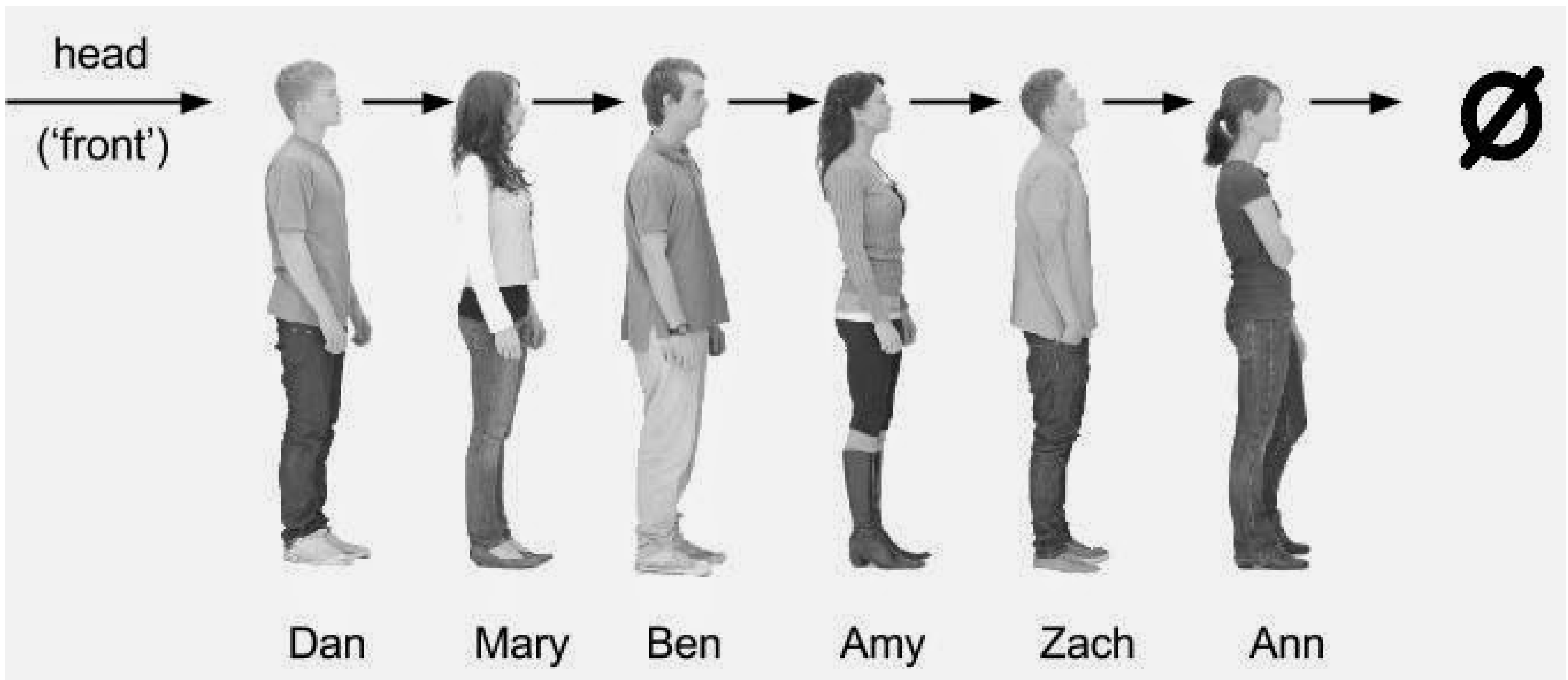


STRUCTURES DE DONNÉES ET ALGORITHMES FONDAMENTAUX

Algorithmique - Programmation – Langages 2 (APL2)

PLAN DU MODULE

CM#1 (rappels +)

Complexité (principe de mesure, ordre polynomial) ; Tableaux (rappels, impact sur la mémoire)

CM#2

Type abstrait de données (ADT, exemple de l'ensemble) ; Listes chaînées (pile, file, liste circulaire, liste doublement chaînée ; algorithmes de base)

CM#3

Arbres binaires ; Tableaux associatifs ; Algorithmes de base

TD x7

Exercices sur ces notions

TP x9

Implantation en C ; Initiation au C

PRENONS UN PEU DE RECUL

Pourquoi une structure de données ?

Stockage et manipulation de nombreuses données

Notion générique d'ensemble

- Accéder à un élément
- Ajouter un élément
- Supprimer un élément
- Tester l'appartenance d'un élément
- Définir l'ensemble vide
- Compter le nombre d'éléments

… (on peut ajouter les opérations d'union, intersection, sous-ensemble...)

→ Analyse centrée sur les opérations : Type Abstrait de Données

TYPE ABSTRAIT DE DONNÉES

Changement de focalisation

- Implantation (représentation) → opérations

Deux éléments de définition

- Signature : nom (sorte), sortes utilisées, ensemble de valeurs valides, opérations
- Préconditions/axiomes

BOOLÉEN COMMME TAD (PROPOSITION)

Signature

Sorte : Booléen

Utilise : (nil)

Opérations

Vrai	: → Booléen
Faux	: → Booléen
¬ (not)	: Booléen → Booléen
∧ (and)	: Booléen x Booléen → Booléen
∨ (or)	: Booléen x Booléen → Booléen

Préconditions/axiomes

$\neg$ Vrai = Faux

$\neg\neg b = b$

(b=Vrai ∨ b=Faux) = Vrai

(b ∧ Faux) = Faux

(b ∨ Vrai) = Vrai

$\neg (a \wedge b) = \neg a \vee \neg b$

Commutativité

$a \wedge b = b \wedge a$

$a \vee b = b \vee a$

Distributivité

$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

Représentation ? Peut être quelconque (0/1, ‘V’/’F’, « VRAI »/« FAUX », 3/-18...)

UTILISATION DU TAD

Rappel des opérations/valeurs

Vrai, Faux, ¬ (not), ∧ (and), ∨ (or)

Définition de nouvelles fonctions (algorithmes)

```
Fonction nand
  Entrées : a, b : Booléen
  Sortie :  Booléen
  Début
    Renvoyer ¬(a ∧ b)
  Fin
```

```
Fonction xor
  Entrées : a, b : Booléen
  Sortie :  Booléen
  Début
    Renvoyer (a ∨ b) ∧ ¬(a ∧ b)
  Fin
```

```
¬(a ∧ b) = not(and(a,b))
```

```
(a ∨ b) ∧ ¬(a ∧ b)=
                    et(ou(a,b),not(et(a,b))
```

INTÉRÊT D’UN TAD

Séparer algorithme et programmation

Algorithme décrit en utilisant les opérations définies pour le TAD

Programmation implante ces opérations, puis l’algorithme

Conception modulaire de programmes

Écrire plein de petites fonctions est bien

Écrire plein de petites fonctions est bien (répétition voulue)

Bon pour

→ conception et réalisation de tests unitaires

→ maintenance du code

→ réutilisation du code

→ identification et correction de bugs

→ structuration de la documentation

→ ...

EXEMPLE IMPLANTATION BOOLÉEN

```
Fonction Vrai
  Sortie : Booléen
  Début
    Renvoyer -18
  Fin
```

```
Fonction Faux
  Sortie : Booléen
  Début
    Renvoyer 3
  Fin
```

```
Fonction not
  Entrée : b : Booléen
  Sortie : Booléen
  Début
    Si b = Faux() alors
      Renvoyer Vrai()
    Sinon
      Renvoyer Faux()
    FinSi
  Fin
```

```
Fonction and
  Entrée : a,b : Booléen
  Sortie : Booléen
  Début
    Si a=Faux() alors
      Renvoyer Faux()
     Sinon
      Si b=Faux() alors
         Renvoyer Faux()
      Sinon
         Renvoyer Vrai()
      Finsi
    Finsi
  Fin
```

```
Fonction or
  Entrée : a,b : Booléen
  Sortie : Booléen
  Début
    Si a=Vrai() alors
      Renvoyer Vrai()
     Sinon
      Si b=Vrai() alors
         Renvoyer Vrai()
       Sinon
         Renvoyer Faux()
      Finsi
    FinSi
  Fin
```

TAD ENSEMBLE

Signature

Sorte : Ensemble
Utilise : Élément, Booléen, Entier
Opérations :

ensemble_vide	: → Ensemble
EstVide	: Ensemble → Booléen
Ajouter	: Ensemble x Élément → Ensemble
Supprimer	: Ensemble x Élément → Ensemble
EstDans	: Ensemble x Élément → Booléen
Taille	: Ensemble → Entier

Axiomes (E:Élément, S:Ensemble)

EstVide(ensemble_vide) = Vrai
Si Taille(S) > 0 alors
    EstVide(S) = Faux
Sinon S=ensemble_vide
EstDans(ensemble_vide, E)=Faux
EstDans(Ajouter(S,E),E) = Vrai
Si EstDans(S,E)=Faux alors
    Taille(Ajouter(S,E)) = Taille(S)+1
Si EstDans(S,E)=Vrai alors
    Taille(Supprimer(S,E)) = Taille(S)-1
Si EstDans(S,E)=Faux alors
    Taille(Supprimer(S,E)) = Taille(S)

Cas sans répétition (cas avec répétition = multi-ensemble)

EstDans(Supprimer(S,E),E) = Faux
Si EstDans(S,E)=Vrai alors
    Taille(Ajouter(S,E)) = Taille(S)

TAD ENSEMBLE

Signature

Sorte : Ensemble
Utilise : Élément, Booléen, Entier
Opérations :

ensemble_vide : → Ensemble
EstVide : Ensemble → Booléen
Ajouter : Ensemble x Élément → Ensemble
Supprimer : Ensemble x Élément → Ensemble
EstDans : Ensemble x Élément → Booléen
Taille : Ensemble → Entier

Problème

Cette définition de permet pas d'accéder à un élément !

Par exemple : comment lister les éléments pour les afficher ? (fonction Afficher(Ensemble))

TAD LISTE ITÉRATIVE

Liste

Ensemble d'éléments rangés

Rangé ≠ trié → Rangé = chaque élément a un rang

Liste itérative : rang=entier

Signature

Sorte : ListeIter

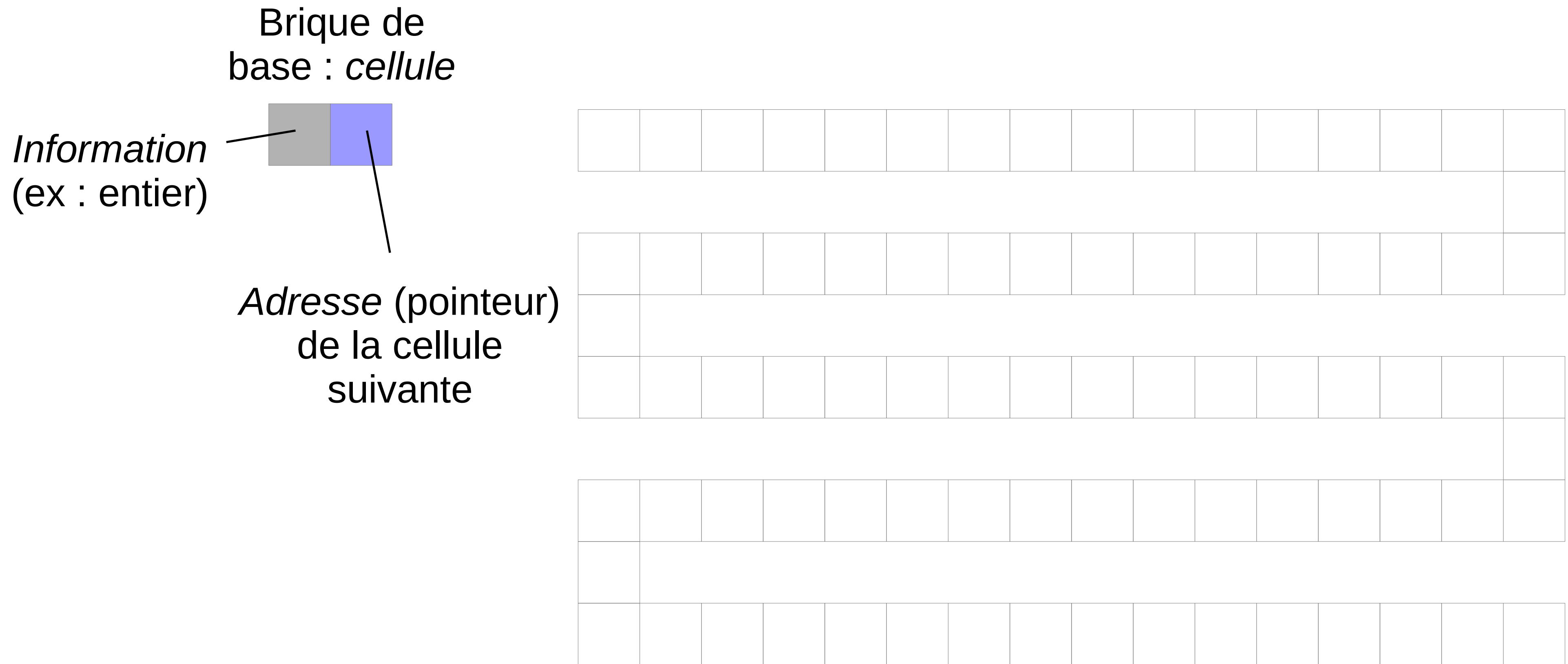
Utilise : Élément, Booléen, Entier

Opérations :

liste_vide	: → ListeIter
EstVide	: ListeIter → Booléen
Contenu	: ListeIter x Entier → Élément
Ajouter	: ListerIter x Entier x Élément → Ensemble
Supprimer	: ListeIter x Entier → ListeIter
EstDans	: ListeIter x Élément → Booléen
Taille	: ListeIter → Entier

```
Procédure Afficher
  Entrée : L : ListeIter
  Variables : i,T : entier
  Début
    T ← Taille(L)
    Pour i allant de 0 à T-1 faire
      Afficher(Contenu(L,i))
    FinPour
  Fin
```

COMMENT FAIRE AUTREMENT ?



Problème : données contiguës

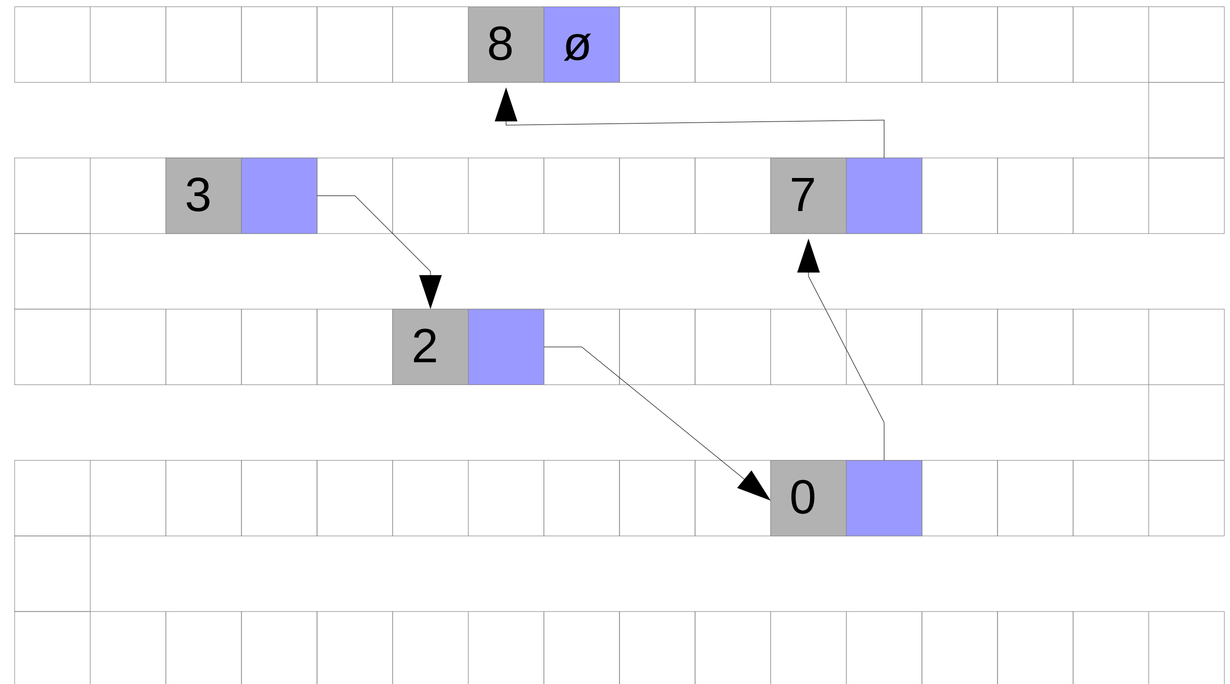
Utiliser des cases dispersées en mémoire

CRÉATION

Ensemble d'entiers : [3,2,0,7,8]

Itérer sur :

- Allocation cellule
- Remplissage cellule
- Mise à jour adresse cellule précédente

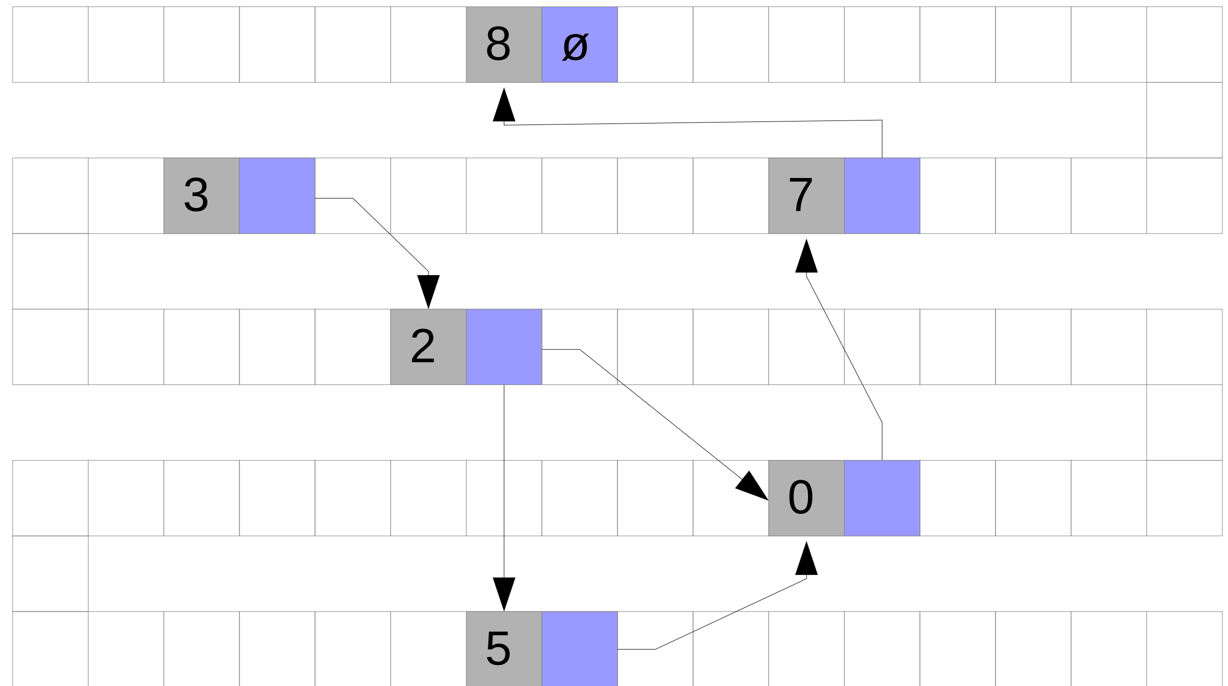


Nouvelle structure de données : liste chaînée

INSERTION

Ensemble d'entiers : [3,2,5,0,7,8]

Insertion de l'élément 5 en position 3

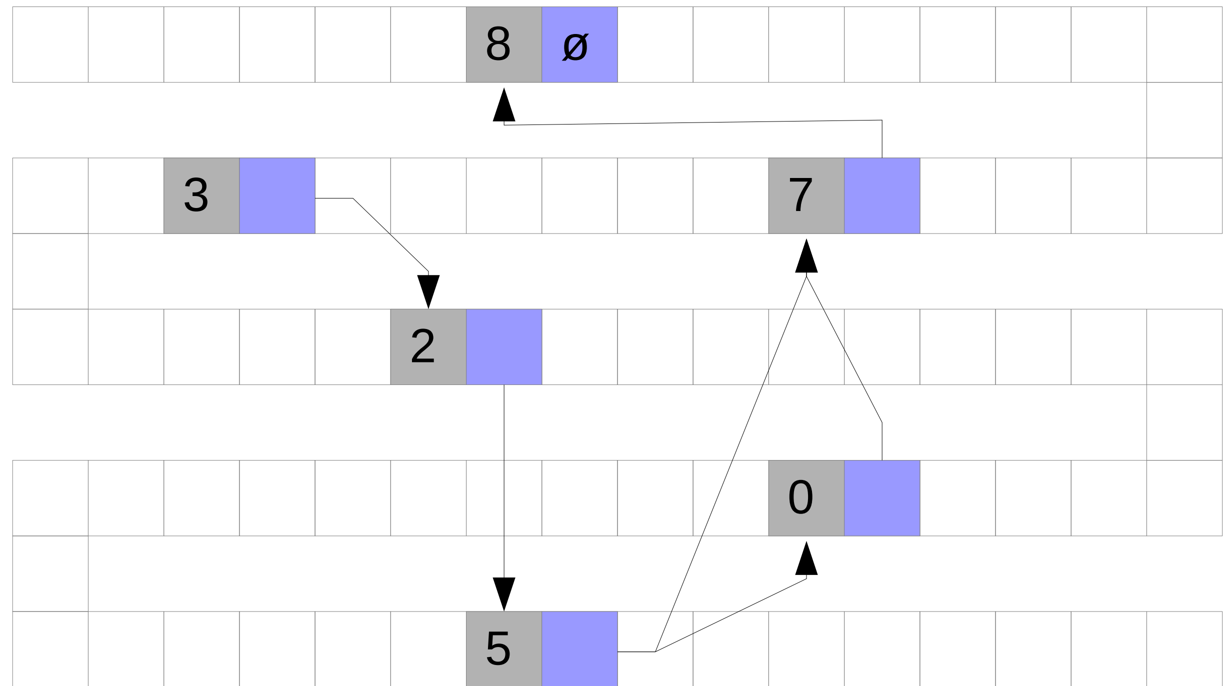


Même coût dans chaque cas → O(1)

SUPPRESSION

Ensemble d’entiers : [3,2,5,0,7,8]

Suppression de l’élément 0

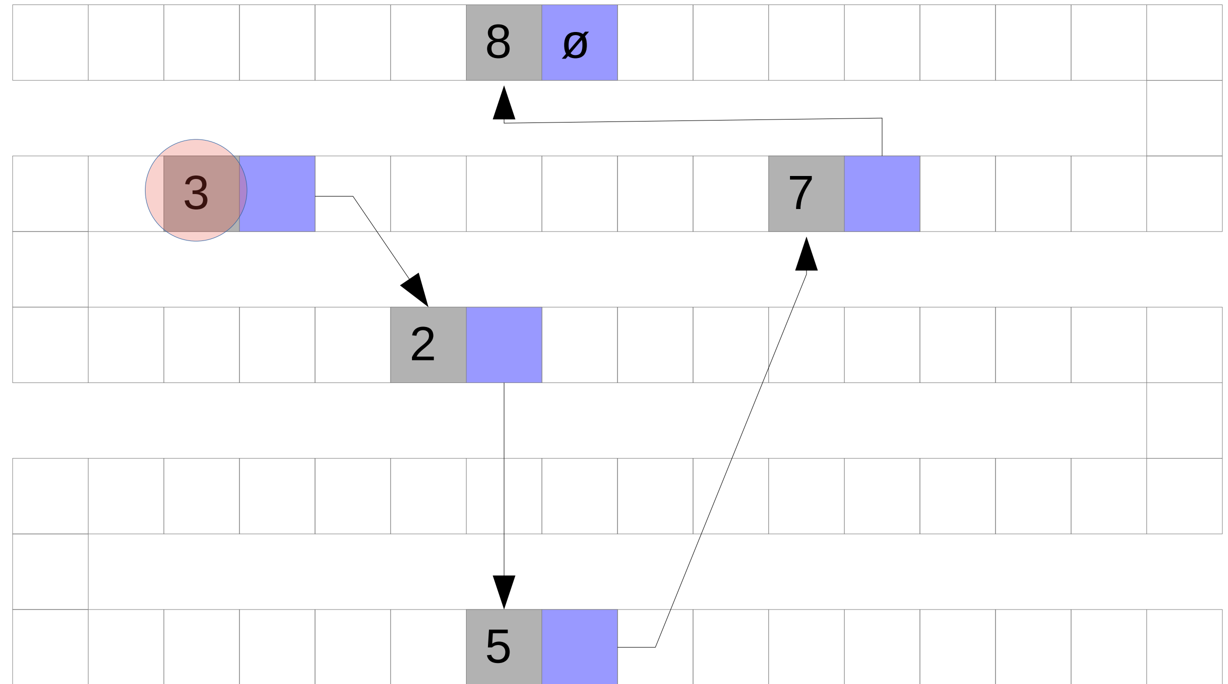


Même coût dans chaque cas → O(1)

ACCÈS

Ensemble d'entiers : [3,2,5,7,8]

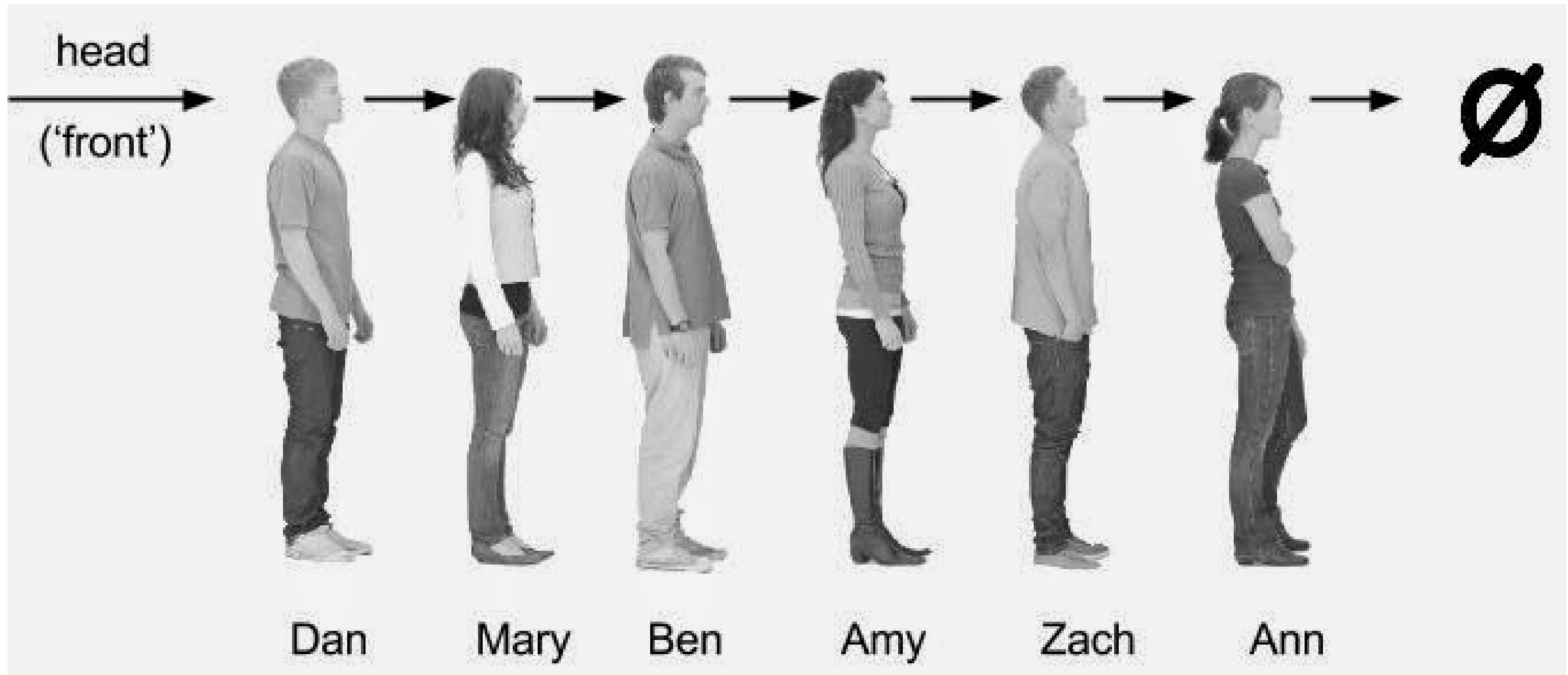
Accès au 3e élément (5)



Pire cas : accéder au dernier → O(n)

LISTE CHAÎNÉE HUMAINE

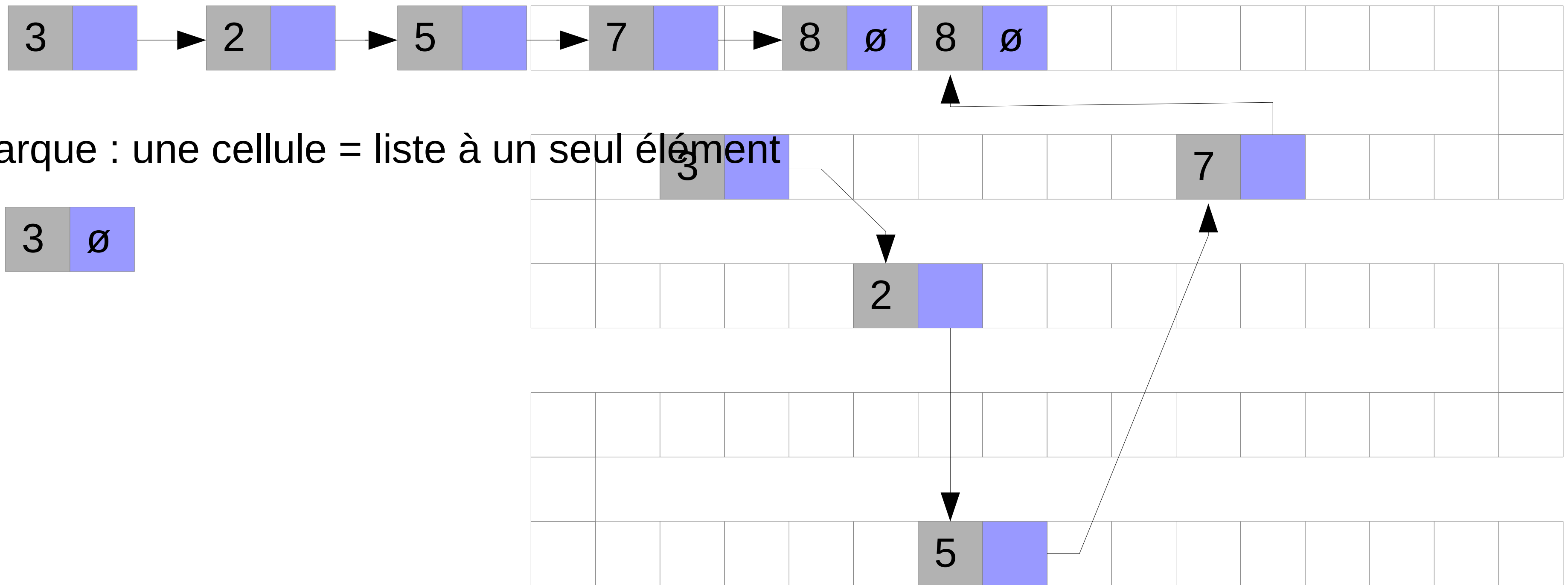
Expérimentation IRL !!!



REPRÉSENTATION D'UNE LISTE CHAÎNÉE

Ensemble d'entiers : [3,2,5,7,8]

Remarque : une cellule = liste à un seul élément



TAD LISTE RÉCURSIVE

Signature

Sorte : ListeRec

Utilise : Élément, Booléen // *Note : Élément définit la valeur* element_invalide

Opérations :

liste_vide : → ListeRec // liste sans aucun élément (ø)
Créer : Élément x ListeRec → ListeRec // *crée une cellule avec un élément et la liste suivante*
Détruire : ListeRec → ListeRec // *détruit la première cellule de la liste et renvoie le reste*
Contenu : ListeRec → Élément // element_invalide *si* EstVide(L)
Succ : ListeRec → ListeRec // *si* EstVide(L), *renvoie* L
EstVide : ListRec → Booléen // *équivalent au test* L=liste_vide

LIEN AVEC LISTE ITÉRATIVE

Rappel : dans ListeRec, sont disponibles

liste_vide, Créer, Détuire, Contenu, Succ, EstVide

Rappel signature ListeIter

Sorte : ListeIter

Utilise : Élément, Booléen, Entier

Opérations :

liste_vide	: → ListeIter
EstVide	: ListeIter → Booléen
Contenul	: ListeIter x Entier → Élément
Ajouter	: ListerIter x Entier x Élément → Ensemble
Supprimer	: ListeIter x Entier → ListeIter
EstDans	: ListeIter x Élément → Booléen
Taille	: ListeIter → Entier

LIENS AVEC LISTE ITÉRATIVE

Rappel : dans ListeRec, sont disponibles

liste_vide, Créer, Détruire, Contenu, Succ, EstVide

Rappel signature ListeIter

Sorte : ListeIter

Utilise : Élément, Booléen, Entier

Opérations :

liste_vide	: → ListeIter
EstVide	: ListeIter → Booléen
ContenuI	: ListeIter x Entier → Élément
Ajouter	: ListerIter x Entier x Élément → Ensemble
Supprimer	: ListeIter x Entier → ListeIter
EstDans	: ListeIter x Élément → Booléen
Taille	: ListeIter → Entier

```
Fonction ContenuI
  Entrée : L : ListeRec, r:entier
  Variables : i:entier, tl:ListeRec
  Sortie : Élément
  Début
    i ← 0
    tl ← L
    TantQue EstVide(tl) = Faux
        et i < r faire
      tl ← Succ(tl)
      i ← i+1
    FinTantQue
    Renvoyer Contenu(tl)
```

LIENS AVEC LISTE ITÉRATIVE

Rappel : dans ListeRec, sont disponibles

liste_vide, Créer, Détruire, Contenu, Succ, EstVide

Rappel signature ListeIter

Sorte : ListeIter

Utilise : Élément, Booléen, Entier

Opérations :

liste_vide	: → ListeIter
EstVide	: ListeIter → Booléen
ContenuI	: ListeIter x Entier → Élément
Ajouter	: ListerIter x Entier x Élément → Ensemble
Supprimer	: ListeIter x Entier → ListeIter
EstDans	: ListeIter x Élément → Booléen
Taille	: ListeIter → Entier

Ajouter, Supprimer, EstDans → voir TD2

```
Fonction Taille
  Entrée : L : ListeRec
  Variables : i:entier, tl:ListeRec
  Sortie : entier
  Début
    i ← 0
    tl ← L
    TantQue EstVide(tl) = Faux faire
      tl ← Succ(tl)
      i ← i+1
    FinTantQue
    Renvoyer i
```

LISTE CHAÎNÉE

Stockage de données

- Avantage : pas de contiguïté, flexibilité
- Inconvénient : surplus de mémoire (en O(n))

Opérations

- Insertion/suppression : 0(1)
- Accès : O(n)

Rappel tableau

- Insertion/suppression : O(n)
- Accès : O(1)

Enumération

- O(n) pour les deux

A noter

- Liste itérative : représentation type comme tableau
- Liste Récursive : représentation type comme liste chaînée
- **Mais ce n'est pas obligatoire** : voir TP piles, avec représentation comme tableau

BILAN

Stockage de données

plusieurs possibilités d'implantation: tableaux, listes chaînées, … ?

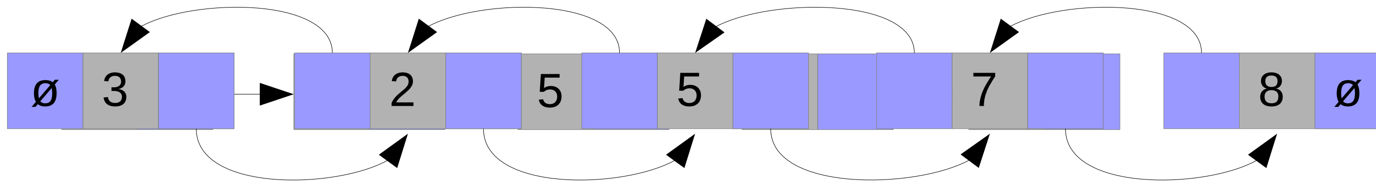
Différence = Complexité des opérations

Si beaucoup d'accès aléatoires et peu d'ajouts/suppressions
→ tableau

Si peu d'accès aléatoire et beaucoup d'ajouts/suppressions
→ liste chaînée

Le choix de la représentation est guidé par les opérations que l'algorithme requiert d'effectuer.

LISTE DOUBLEMENT CHAÎNÉE



Problème : Parcours malaisé car juste successeur et pas prédécesseur

Ajout d'une opération `Préd` (prédecesseur) Avantage :

accès plus aisé à un élément quelconque, parcours dans les deux sens, suppression simplifiée

Inconvénient :

Plus de place mémoire nécessaire

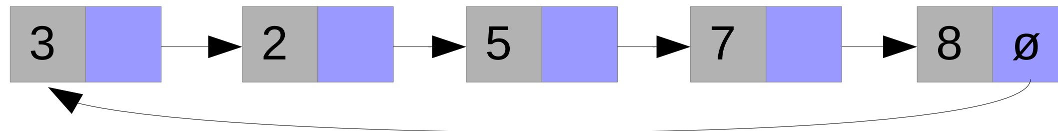
Exemples :

Stockage de structures hiérarchiques

historique d'opérations

Voir TD4 et TP5 pour leur étude et leur implantation

LISTE CIRCULAIRE



Problème : Fin et début de liste à gérer : l'opération `Succ` n'est pas toujours valide

Avantage :

`Succ` toujours valide.

Inconvénient :

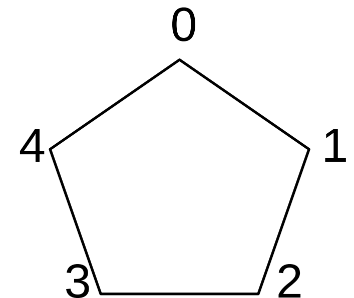
Gestion de la fin de boucle lors de l'énumération

→ ajout d'une opération `Fin` indiquant la dernière cellule de la liste

Exemples

Menu circulant

Polygone fermé (e.g. calcul du milieu de chaque arête)



STRUCTURE PILE

Usage

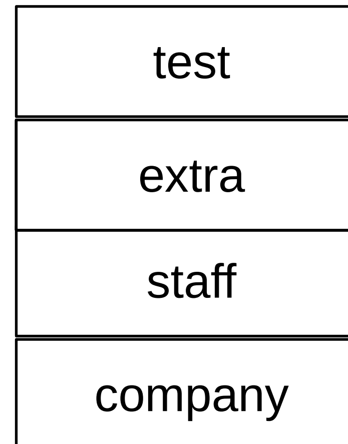
- Opération d'ajout : en tête (ou en fin) de liste
- Opération de lecture de contenu : dernier élément ajouté
- Opération de suppression de contenu : dernier élément ajouté
- Liste LIFO (*Last In First Out*)

Exemple

- Pile d'assiette
- Annulation d'une séquence de commandes dans une interface graphique (Ctrl-Z, « Undo »)
- Pile d'appels d'une fonction récursive
- Parcours d'arbre en profondeur
- Parseur XML

PARSER UN FICHIER XML

```
<company>
        <staff id="1">
                <firstname>
                  yong
                </firstname>
                <lastname>
                  mook kim
                </lastname>
                <salary>
                  1000000
                </salary>
                <age>
                  29
                </age>
            <extra>
                  <test>123</test>
            </extra>
        </staff>
        <staff id="2">
                <firstname>
                  low
                </firstname>
                <lastname>
                  yin fong
                </lastname>
                <salary>
                  500000
                </salary>
        </staff>
</company>
```



DÉFINITION PILE

Signature

Sorte : Pile
Utilise : Élément, Booléen
Opérations

pile_vide	: → Pile	
Empiler	: Pile x Élément → Pile	// push()
Dépiler	: Pile → Élément x Pile	// pop()
EstVide	: Pile → Booléen	

Voir TD3 et TP4 pour la description algorithmique et l’implantation de ces opérations avec une représentation en tableau

STRUCTURE FILE

Usage : similaire à Pile

- Ajout en tête (ou fin) de liste

Mais

- Lecture et suppression de l'élément le plus ancien, en fin (ou tête) de liste
- Liste FIFO (*First In, First Out*)

Exemples

- File d'attente
- Gestion de file d'impression
- Gestion des processus dans une machine (accès processeur)
- Parcours d'arbre en largeur

DÉFINITION FILE

Signature

Sorte : File
Utilise : Élément, Booléen
Opérations

file_vide	: → File
Ajouter	: File x Élément → File
Retirer	: File → Élément x File
EstVide	: File → Booléen

Par rapport à Pile :

Accès fréquent à la fin de liste (soit pour l'ajout, soit pour la suppression)

Accès en O(n)

D'où nécessité de maintenir l'information de position de fin de liste

CONCLUSION

Nouvelle structure de données : Liste Chaînée

Introduction aux Types de Données Abstraits (TAD)

- TAD=Algorithme
- Structure = Implantation
- Plusieurs exemples de TAD (Ensemble, Liste Itérative, Liste Récursive, Pile, File)
- Pour aller plus loin :
  - livre « Types de Données et Algorithmes », par C. Froidevaux, M-C. Gaudel et M. Soria

Implantation modulaire

- TAD=Identifier les fonctionnalités récurrentes dans l'algorithme, conception « haut niveau »
- Structure=Implantation (et TEST!) effective de ces fonctionnalités

Type Ensemble

- Pas très efficace sous forme de tableau ou de liste, si données non rangées
- Prochain cours : arbres (binaires) et tables de hachage