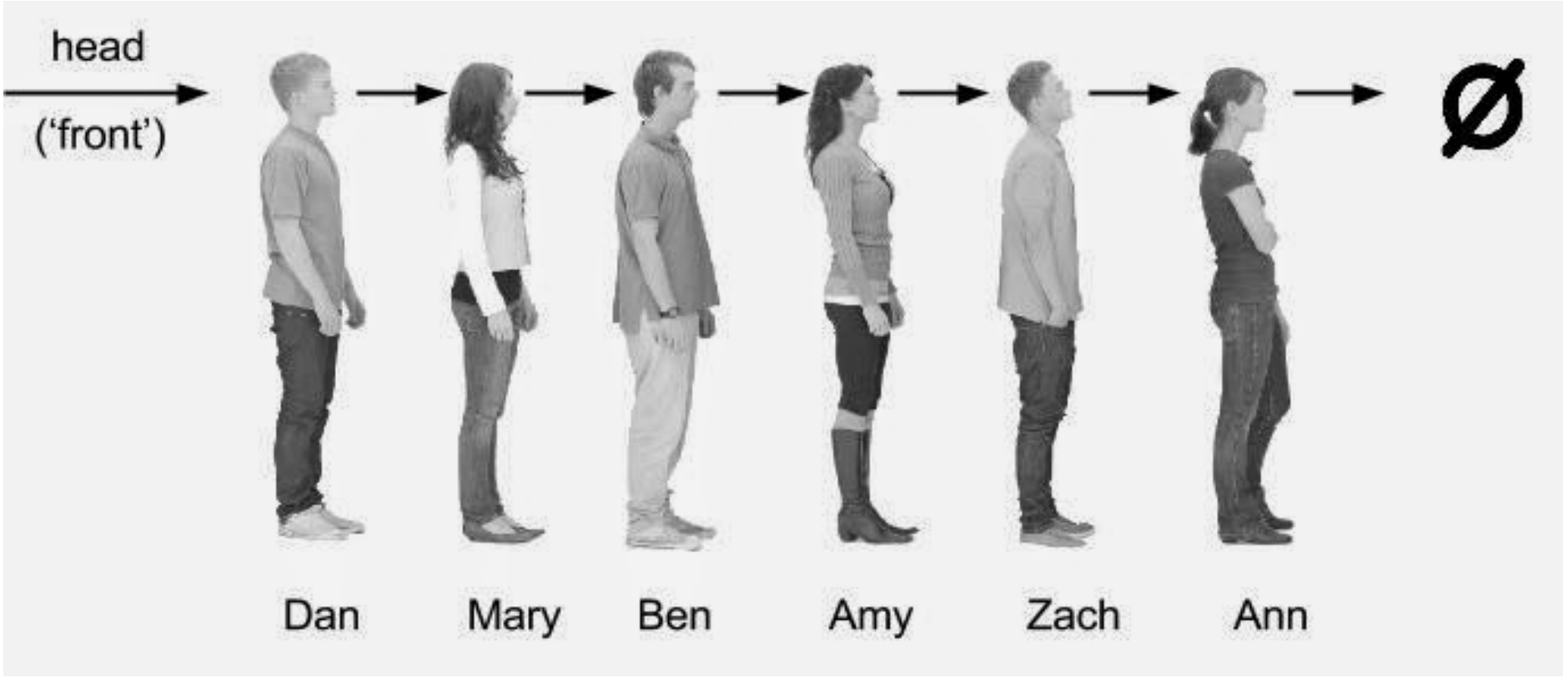


STRUCTURES DE DONNÉES ET ALGORITHMES FONDAMENTAUX

Initiation au développement (DEV)

STRUCTURE DE DONNÉES : LE TABLEAU

Définition

Un tableau est une **structure de données** servant à stocker plusieurs éléments d'un **même type**, sur une **zone contiguë** de la mémoire (=*plage mémoire*).

Notation

nom[taille] : type ou nom : type[taille] *(← utilisée dans ce cours)*

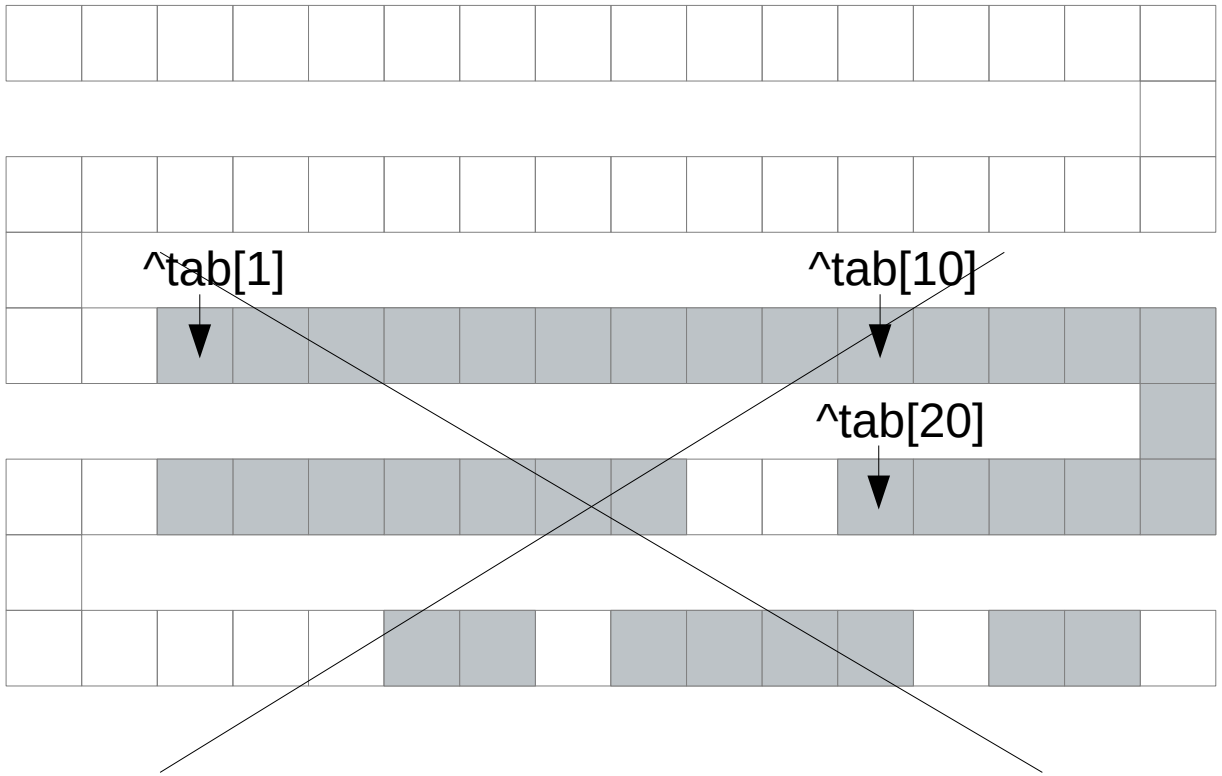
Exemples :

```
tab[10] : réel
data : entier[20]
chaine[15] : caractère // chaîne de 14 caractères en C
Tableau : booléen[] // si la taille n'est pas connue a priori
```

Rappel : nous prenons comme convention d'indexer le premier élément du tableau par l'entier 1. Un tableau à N éléments sera donc indexé de 1 à N inclus.

TABLEAU=PLAGE MÉMOIRE

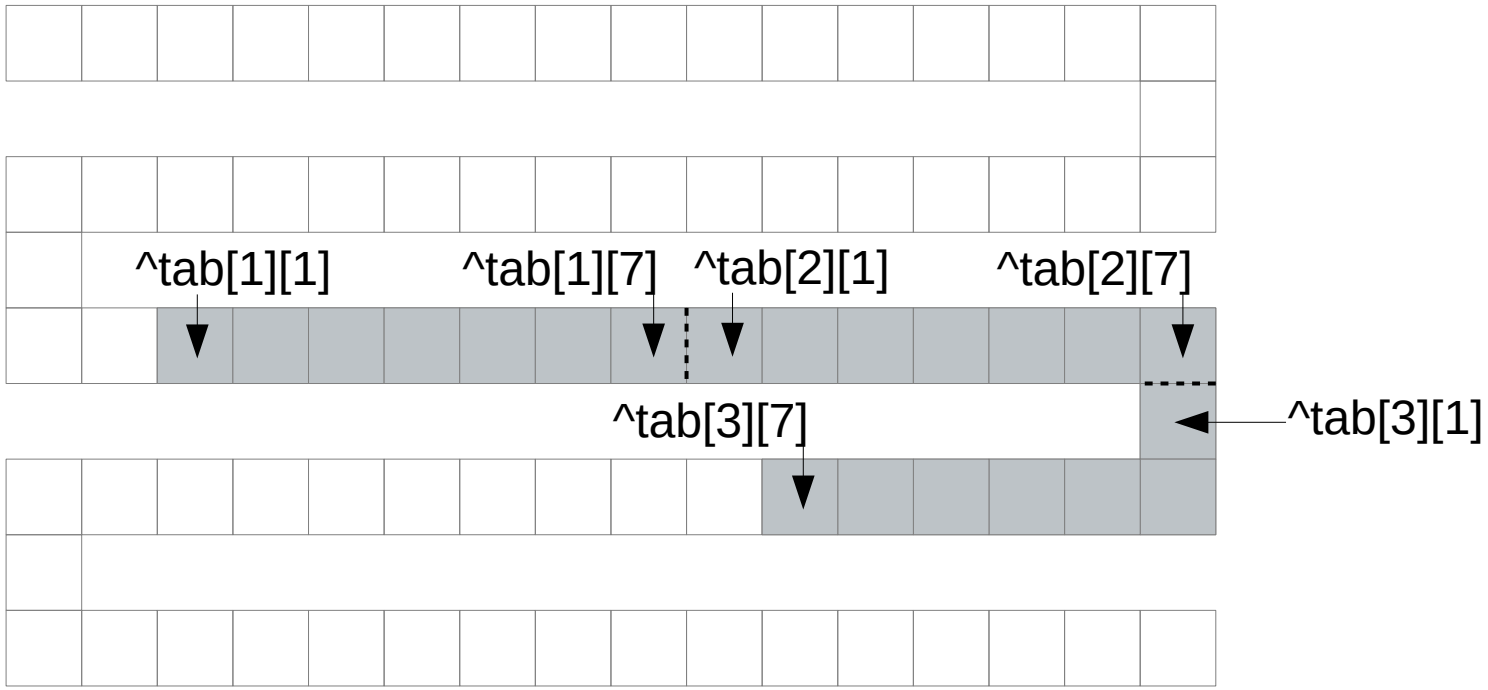
tab : entier[20]



Note : dans ce schéma, une case = un entier = 4 octets
(en règle générale, une case mémoire = 1 octet)
Et ^var signifie « l'adresse de la variable » var

TABLEAU MULTIDIMENSIONNEL

tab : entiers[3][7]



OPÉRATIONS ET COMPLEXITÉ

Accès à un élément (en lecture ou écriture : Contenu)

Coût = calcul de l'adresse
= adresse tableau + indice * taille du type

→ O(1)

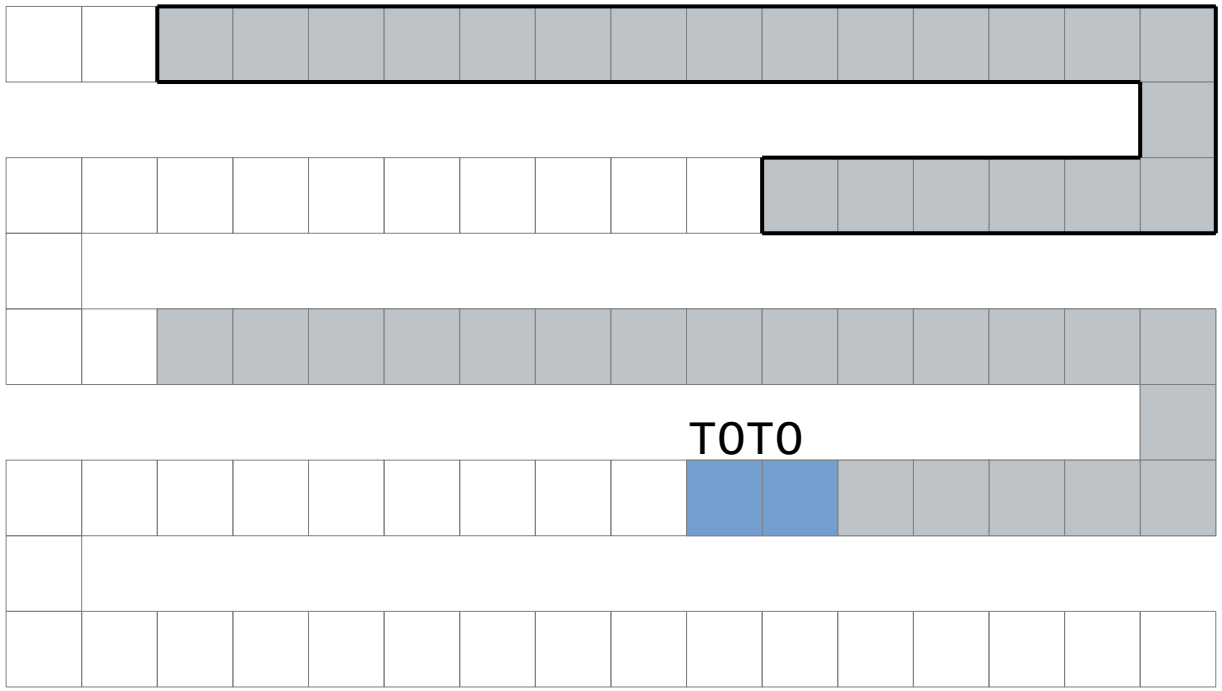
Exemple : ^tab[i] = ^tab[1] + (i-1) * taille(entier)

Note: pas exactement du C !!!

En C, on écrirait : &tab[i] = &tab[0] + i (= tab + i)

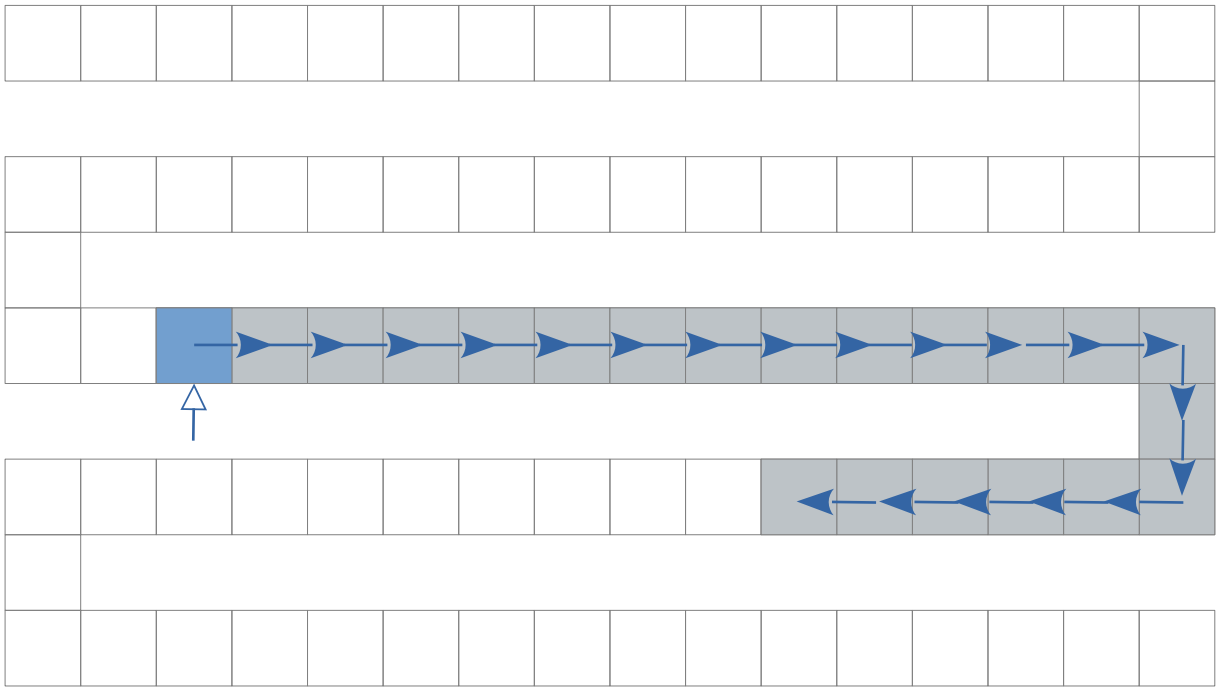
Ajout et suppression d'un élément ?

AJOUT D'UN ÉLÉMENT (AJOUTER)



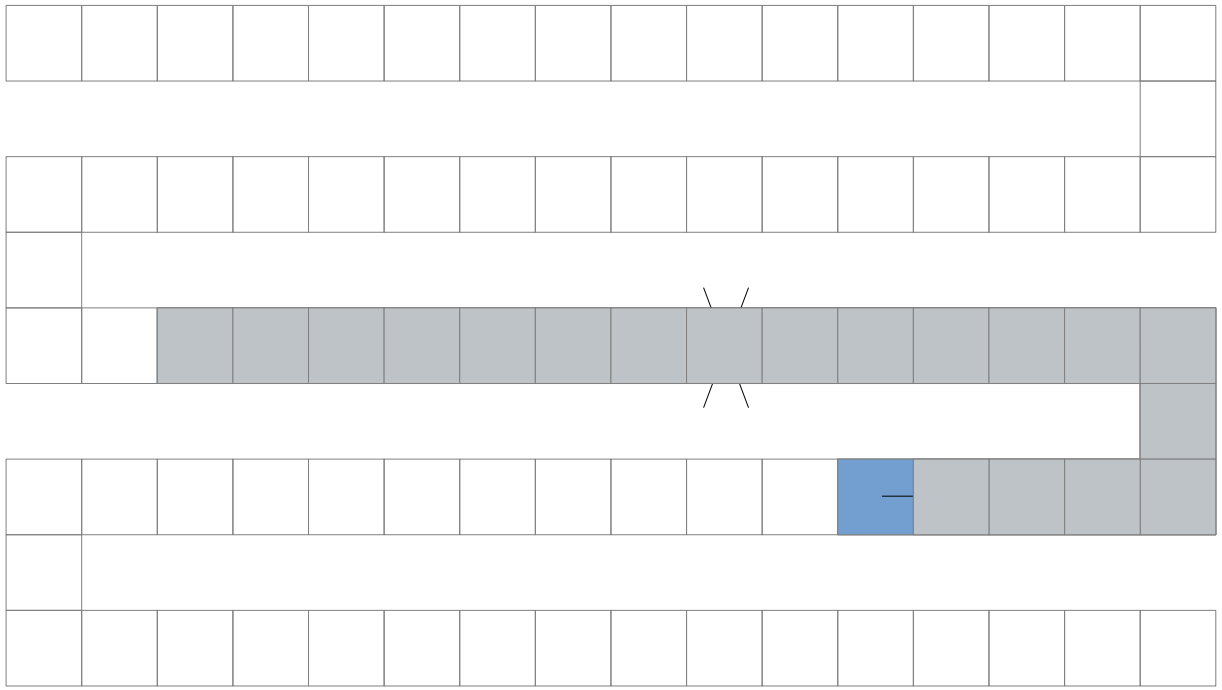
Pire cas : réallocation et recopie des n éléments → O(n)

AJOUT D'UN ÉLÉMENT (AJOUTER V2)



Pire cas : ajout du premier élément = copie de n éléments → O(n)

SUPPRESSION D'UN ÉLÉMENT (SUPPRIMER)



Pire cas : retrait du premier élément = copie de n-1 éléments → O(n)

SYNTHÈSE

Tableaux

- Occupent une zone contiguë en mémoire (même multidimensionnels)
- Accès en O(1)
- ajout/suppression en O(n)

Peut-on faire différemment ? Mieux ?

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**

TAD ENSEMBLE

Signature

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

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

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 (chaque élément a un rang)

Liste itérative : rang=entier → STRUCTURE À ACCÈS DIRECT

Signature

Sorte : ListeIter

Utilise : Élément, Booléen, Entier

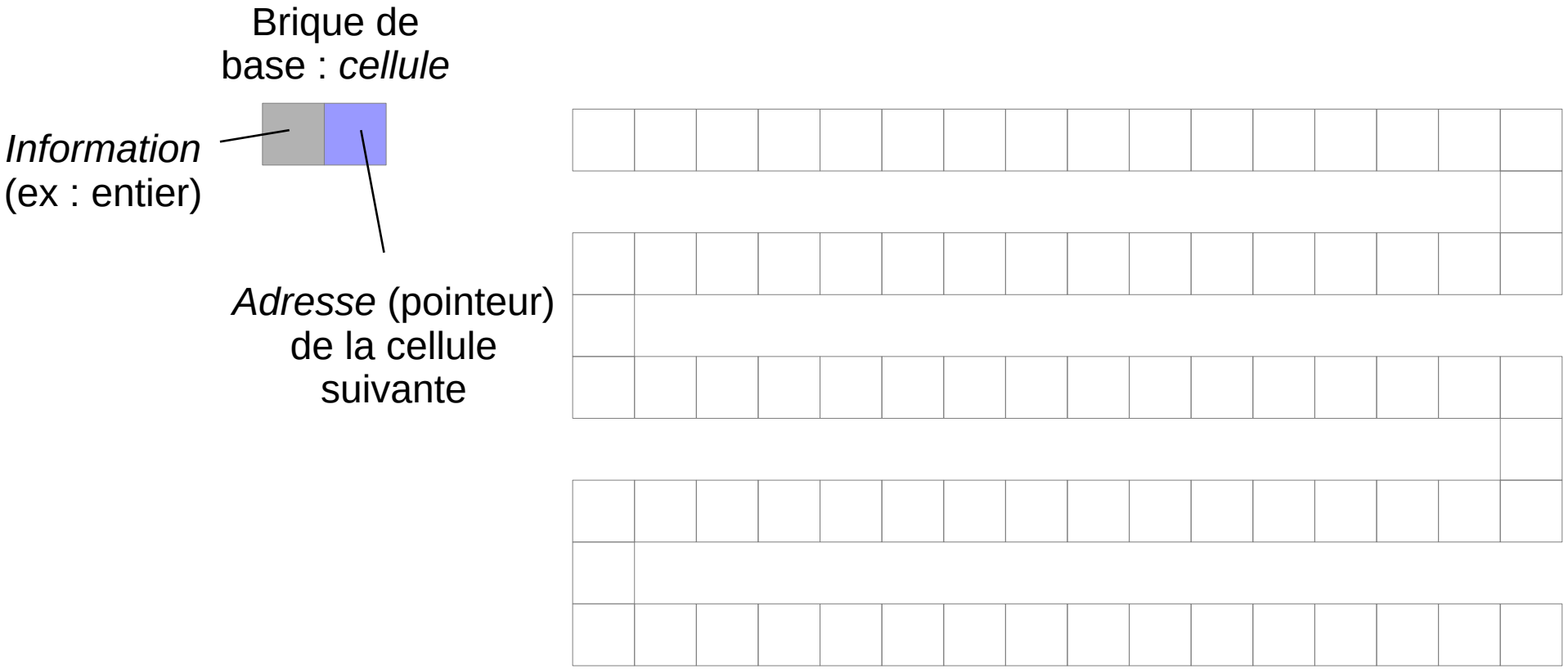
Opérations :

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

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

Implémentation privilégiée : tableau → Contenu(L,i) : L[i]

COMMENT FAIRE AUTREMENT ?



Problème : données contiguës

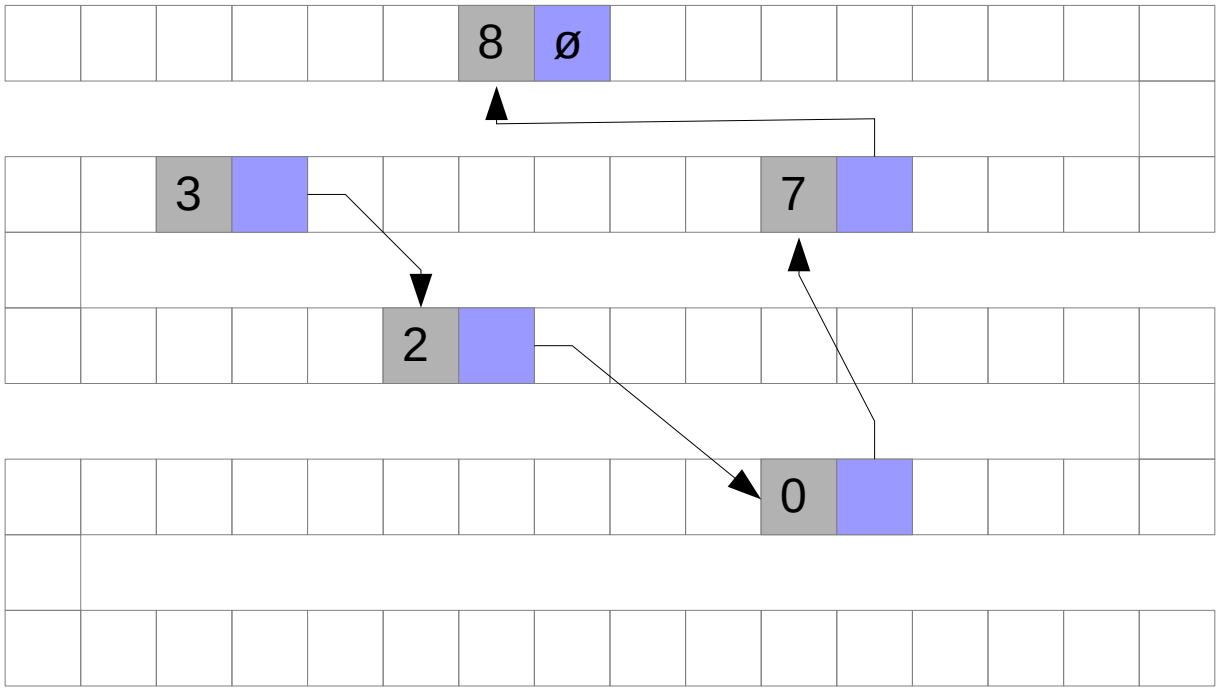
Utiliser des cases dispersées en mémoire

CRÉATION

Ensemble d’entiers : [3,2,0,7,8]

Itérer sur :

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

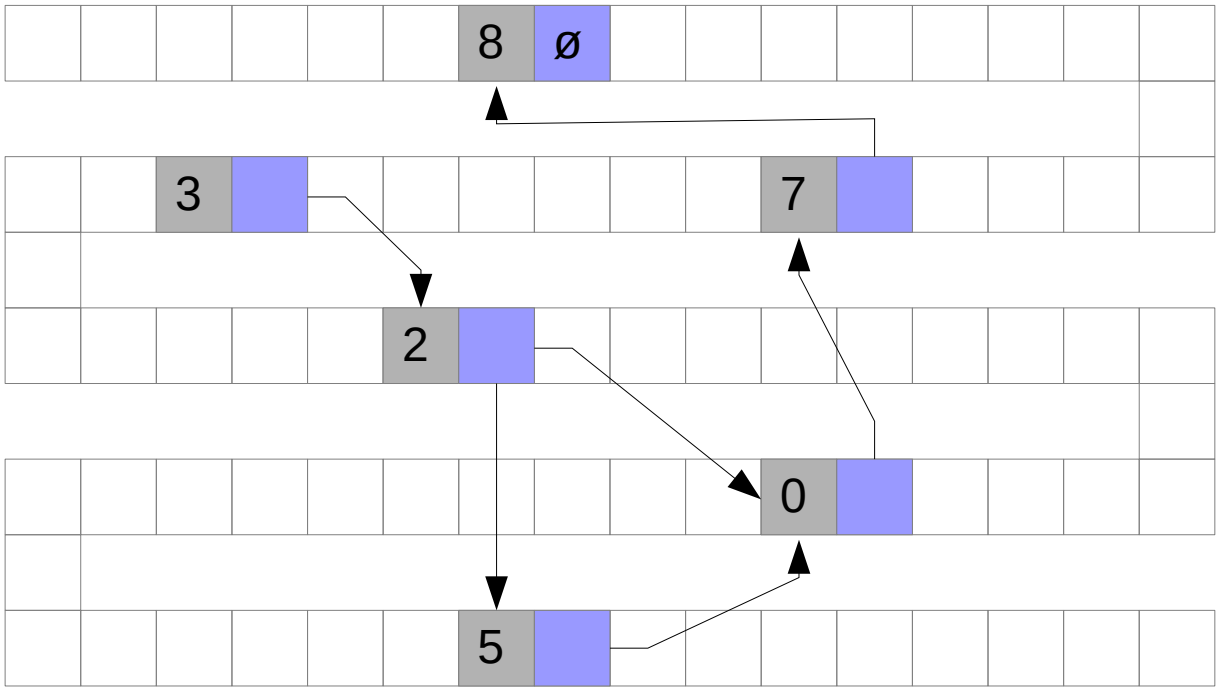


Nouvelle structure de données : liste chaînée

INSERTION

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

Insertion de l'élément 5 en position 3

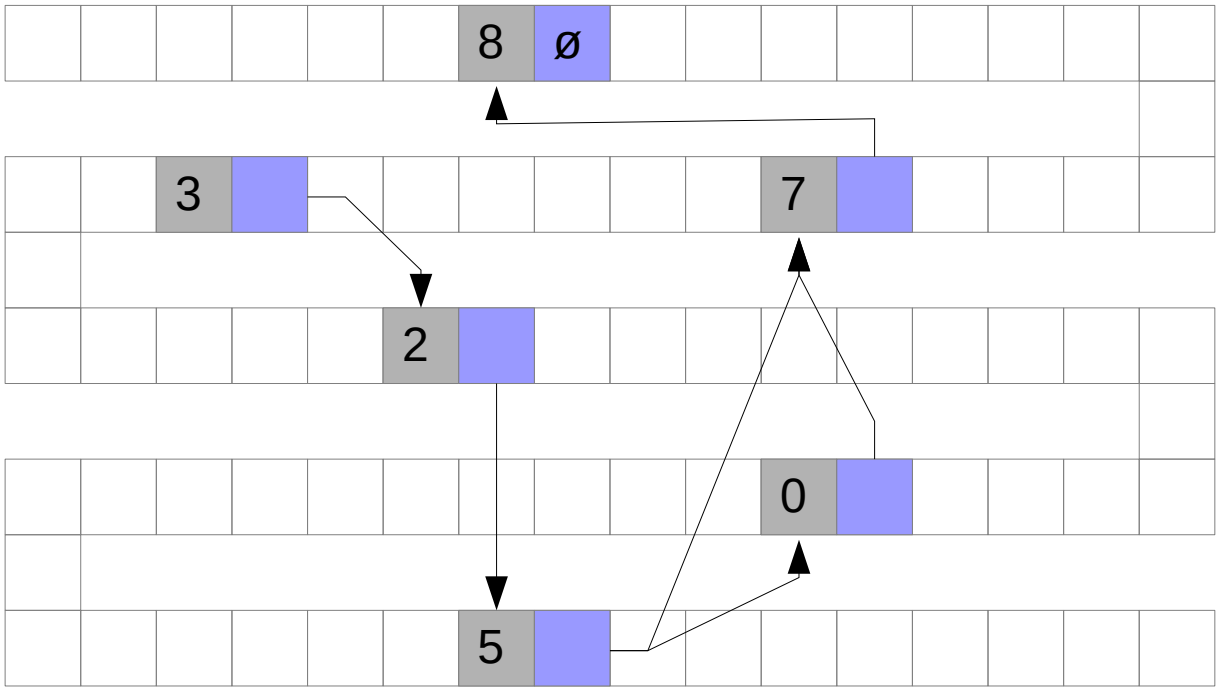


Même coût dans chaque cas → O(1)

SUPPRESSION

Ensemble d’entiers : [3,2,5,0,7,8]

Suppression de l’élément 0

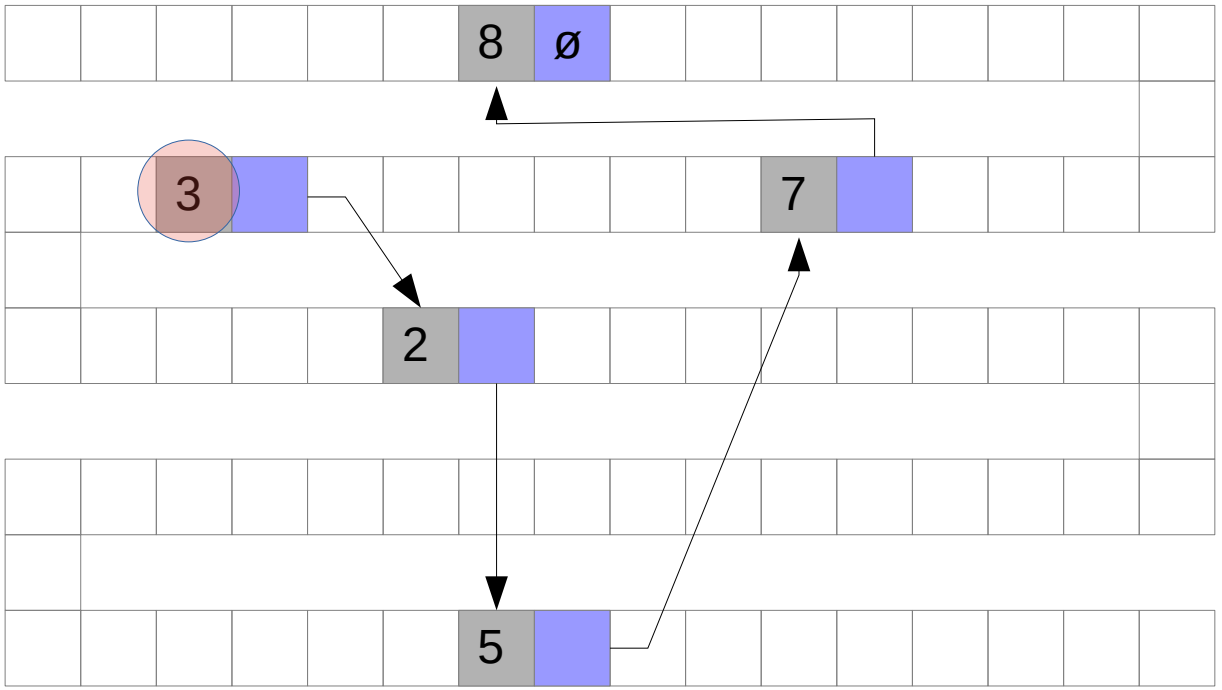


Même coût dans chaque cas → O(1)

ACCÈS

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

Accès au 3e élément (5)

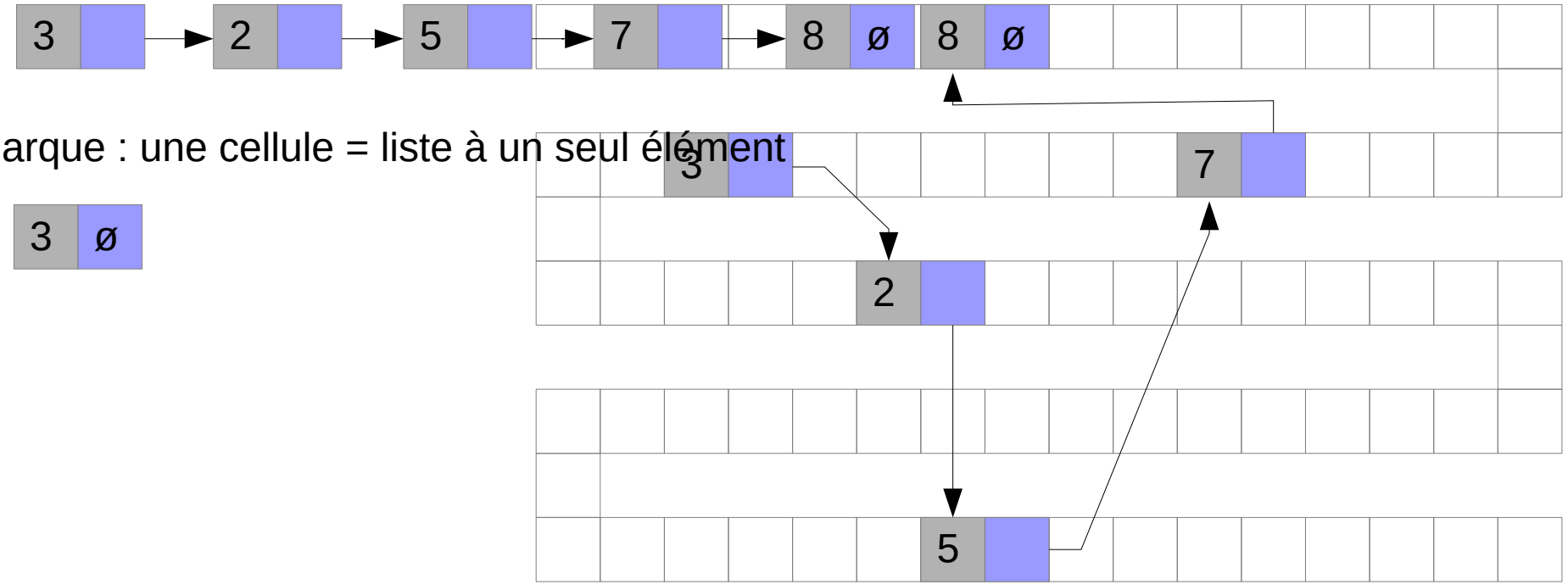


Pire cas : accéder au dernier → O(n)

STRUCTURE À ACCÈS SÉQUENTIEL

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)
EstVide : ListRec → Booléen // *équivalent au test* L=liste_vide
Contenu : ListeRec → Élément // element_invalide *si* EstVide(L)
Succ : ListeRec → ListeRec // *si* EstVide(L), *renvoie* L
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*

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

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

```
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
```

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

Ajouter, Supprimer, EstDans → voir TD9&10

```
Fonction ContenuIter
  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)
```

LISTE CHAÎNÉE

Stockage de données

Avantage : pas de contiguïté, flexibilité

Inconvénient : surplus de mémoire (en O(n))

Opérations

Insertion/suppression : 0(1) (sans compter le temps d'accès, qui peut souvent être évité en pratique)

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, structure à accès direct

Liste Récursive : représentation type comme liste chaînée, structure à accès séquentiel

Mais ce n'est pas obligatoire : voir TP piles, avec représentation comme tableau

BILAN

Stockage de données

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

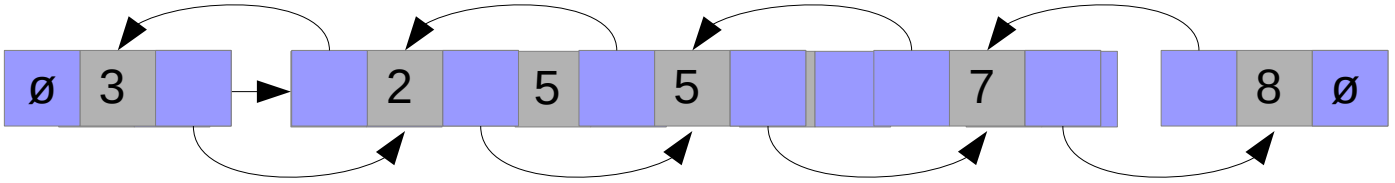
Différence = Complexité des opérations

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

Si peu d'accès aléatoires 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 :

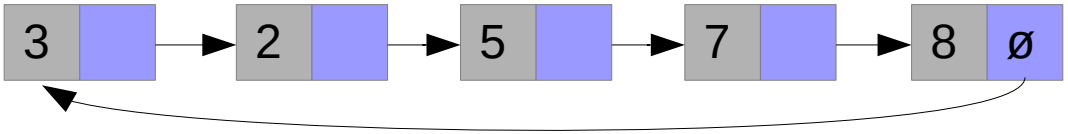
Plus de place mémoire nécessaire

Exemples :

Stockage de structures hiérarchiques
historique d'opérations

Voir TD13&15 pour leur étude

LISTE CIRCULAIRE



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

Avantage :

Succ toujours valide.

Inconvénient :

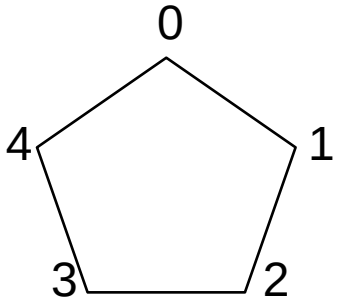
Gestion de la fin de boucle lors de l'énumération

→ fin quand on boucle sur la première cellule

Exemples

Menu circulant

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



STRUCTURE PILE

Usage

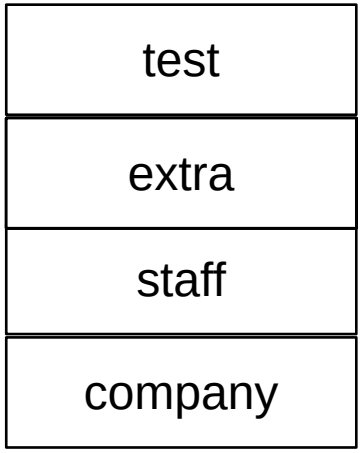
- Opération d'ajout : en tête (ou en fin) de liste
- Opération de lecture de contenu : dernier élément ajouté
- Opération de suppression de contenu : dernier élément ajouté
- Liste LIFO (*Last In First Out*)

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 → Pile	// pop()
Sommet	: Pile → Élément	
EstVide	: Pile → Booléen	

Axiomes

- EstVide(pile_vide) = Vrai
- EstVide(Empiler(P,e)) = Faux
- Sommet(pile_vide) non défini (=élément_invalide)
- Sommet(Empiler(P,e)) = e
- EstVide(Dépiler(pile_vide)) = Vrai
- Dépiler(Empiler(P,e)) = P

Voir TP22&23 pour l'implantation de ces opérations avec une représentation en tableau

STRUCTURE FILE

Usage :

Ajout en fin de liste (pas comme Pile), et suppression en tête de liste (comme Pile)

→ Lecture et suppression de l'élément le plus ancien de la 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

(Note : on peut aussi implémenter une file où on ajoute en tête de liste (comme Pile), mais alors on supprime en fin de liste (pas comme Pile).

DÉFINITION FILE

Signature

Sorte : File
Utilise : Élément, Booléen
Opérations

file_vide	: → File
Enfiler	: File x Élément → File (*enqueue)*
Défiler	: File → File (*dequeue)*
EstVide	: File → Booléen
Premier	: File → Élément

Axiomes

- EstVide(file_vide) = Vrai
- EstVide(Enfiler(F,e)) = Faux
- Premier(file_vide) non défini (=élément_invalide)
- Premier(Enfiler(F,e)) = e si EstVide(F) ;
  Premier(F) sinon
- EstVide(Défiler(file_vide)) = Vrai
- Défiler(Enfiler(F,e)) = file_vide si EstVide(F) ;
  Enfiler(Défiler(F),e) sinon

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

- soit en implémentant sous forme de tableau (comme la Pile dans le TD22&23)
- soit en implémentant comme un enregistrement qui contient la référence vers le début et la fin de la liste chaînée

CONCLUSION

Nouvelle structure de données : Liste Chaînée

- Accès séquentiel en O(n)
- Ajout/suppression en O(1)
- Recherche d'élément en O(n)
- Surtout utilisé dans ses versions pile et file (+ hachage).
- Sinon, une implantation intelligente du tableau est souvent plus efficace.

Type Ensemble

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

LISTE CHAÎNÉE HUMAINE

Expérimentation IRL !!!