

# Programmation avancée

## Listes

Walter Rudametkin

Walter.Rudametkin@polytech-lille.fr  
<https://rudametw.github.io/teaching/>

Bureau F011  
 Polytech'Lille

5 février 2016

1/24

## Structures de Données

Représentation de collections d'informations en fonction de :

- ▶ traitements privilégiés à effectuer
- ▶ contraintes
  - ▶ espace / temps
  - ▶ outils disponibles (selon les différents langages)
- ▶ propriétés
  - ▶ relations d'ordre ?
  - ▶ taux de dynamique
  - ▶ taille des informations

2/24

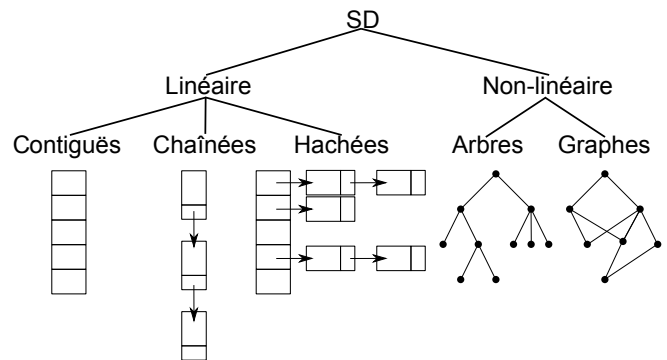
## Structures de Données

### Traitements types

- ▶ Tris
- ▶ Recherche d'informations
  - ▶ Par position: e.g., le  $k$ ème élément
  - ▶ Par valeur (associative) :  $v \in C/P(v)$
- ▶ Mises à jour
  - ▶ Ajout
  - ▶ Suppression
  - ▶ Modification  $\Rightarrow$  recherche

3/24

## Structures de Données: Classification des SD



4/24

## Structures de Données: Analyse des besoins

- ▶ Identification des informations et de leurs caractéristiques
- ▶ Identification des opérations (recherche, ajout, suppression, ...)
- ▶ Opérations à privilégier ?
- ▶ Etude des représentations possibles (structures de données), avec méthodes de résolution et coûts associés
- ▶ Choix de la structure en fonction des coûts

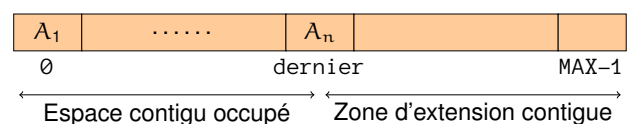
5/24

## Les listes contiguës

### Structures de données contiguës

- ▶ Autorisent accès direct et calculé
  - ▶ Déjà vu : tableaux

### Représentation



6/24

## Les listes contiguës — Définition

### Définition du type liste\_contiguë

```
type Liste_contiguë = structure
    espace : Vecteur[MAX] de <T>
    dernier : Entier
fin
```

### Utilisation

- ▶  $l$  : Liste\_contiguë
- ▶  $l$  vide  $\iff l.dernier = -1$

### Comment estimer MAX ?

7/24

## Les listes contiguës — Exemple

### Exemple : affichage des éléments d'une liste contiguë

```
Action affich(l)
    D : l : Liste_contiguë
    L : i : Entier
    Pour i de 0 à l.dernier Faire
        ecrire(l.espace[i])
    Fpour
Faction
```

8/24

## Les listes contiguës — Recherche

### Par position

- ▶ Accès direct en temps constant : Coût = 1

### Par valeur

- ▶ Non ordonnée  $\Rightarrow$  recherche séquentielle
  - ▶ coût min : 1, coût max : N
- ▶ Ordonnée
  - ▶ Recherche séquentielle ordonnée
    - ▶ coût min : 1, coût max : N
  - ▶ Recherche dichotomique
    - ▶ coût min : 1, coût max :  $\log_2 N$

Soit  $N$  le nombre d'éléments de la liste

9/24

## Les listes contiguës — Ajout

### Non ordonnée

- ▶ Ajout n'importe où  $\Rightarrow$  en queue
- ▶ Coût : 1

### Ordonnée

- ▶ Insertion à l'indice  $p \Rightarrow N - p$  décalages
- ▶ Coût min : 1, coût max : N

10/24

## Les listes contiguës — Suppression

### Non ordonnée

- ▶ Recherche séquentielle de l'élément à supprimer (min : 1, max N)
- ▶ Permutation avec le dernier élément

### Ordonnée — suppression à l'indice $p$

- ▶ Recherche dichotomique de l'élément
  - ▶ Coût min : 1, coût max :  $\log_2 N$
- ▶  $N - p + 1$  décalages (min : 1, max N)
- ▶ Coût min : 1, max :  $N + \log_2 N$

11/24

## Les listes chaînées

### Représentation dispersée

- ▶ Éléments rangés n'importe où en mémoire ... dans des cellules mémoire gérées dynamiquement ... repérées par pointeur

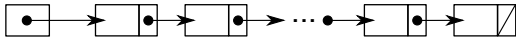
### Chaînées


- ▶ Chaque cellule repère la cellule suivante
- ▶ Pointeur sur la première cellule définit la liste
- ▶ Dernière cellule ne repère aucune cellule : pointeur NULL
- ▶ NULL = valeur de la liste vide

12/24

## Les listes chaînées

### Schématiquement



 Liste vide    l = NULL ou  $\emptyset$

### Déclarations

type Ptcellulle = Pointeur de Cellule

```
type Cellule = structure
  valeur : <T>
  suivant : Ptcellulle
```

fin

type Liste chaînée = Ptcellulle

l : Liste chaînée

13/24

## Les listes chaînées : Notation sur les pointeurs

- ▶  $p \uparrow$  : Cellule  $\Rightarrow$  Pointeur de Cellule
- ▶  $p \uparrow \bullet \text{valeur}$
- ▶  $p \uparrow \bullet \text{suivant}$
- ▶ NULL  $\Rightarrow$  pointeur vide

On trouve souvent la notation  $p \rightarrow \text{valeur}$  à la place de  $p \uparrow \bullet \text{valeur}$

14/24

## Les listes chaînées : Gestion dynamique des cellules

- ▶ Fonction allouer () : Ptcellulle
  - ▶ Fonction qui alloue dynamiquement une cellule
  - ▶ Résultat : pointeur sur la cellule allouée
- ▶ Action libérer(p)
  - ▶ Récupère la cellule mémoire pointée par p
  - ▶ Valeur de p ???

15/24

## Les listes chaînées — Recherche

SD essentiellement séquentielle  $\Rightarrow$  parcours séquentiel

- ▶ Par position
  - ▶ Parcours du chaînage jusqu'au kième (coût = k)
- ▶ Par valeur
  - ▶ Séquentielle (coût min : 1, max : N)
  - ▶ Séquentielle ordonnée (coût min : 1, max : N)
  - ▶ Pas de dichotomie

16/24

## Les listes chaînées — Parcours séquentiel

```
type Ptcellulle = Pointeur de Cellule
type Cellule = structure
  valeur : <T>
  suivant : Ptcellulle
fin
type Liste chaînée = Ptcellulle

1 Action affich (l)
2   D : l : Liste chaînée
3   L : p : Ptcellulle
4   p  $\leftarrow$  l
5   TQ p  $\neq$  NULL Faire
6     ecrire( $p \uparrow \bullet \text{valeur}$ )
7     p  $\leftarrow$   $p \uparrow \bullet \text{suivant}$ 
8   FTQ
9 Faction
```

17/24

## Les listes chaînées — Mises à jour

- ▶ Ajout / Suppression de cellules
  - ▶ Modification locale du chaînage
    - ▶ Pas de décalage de cellules
- ▶ Coût : constant (quelques affectations de pointeurs)
- ▶ Ajout dans une liste non ordonnée
  - ▶ N'importe où, e.g en tête
- ▶ Ajout dans une liste ordonnée
  - ▶ Coût ???

18/24

## Les listes chaînées — Mise à jour

### Ajout dans une liste non-ordonnée

```
1 Action ajout_tête(l, val)
2   D : val : <T>
3   D/R : l : Liste chaînée
4   L : p : Pcellule
5
6   p ← allouer()
7   p↑•valeur ← val
8   p↑•suivant ← l
9   l ← p
10 Faction
```

Cas limite: liste vide

19/24

## Les listes chaînées — Mises à jour

### Ajout dans une liste ordonnée

Rechercher *prec*, pointeur précédent tel que

►  $\text{prec} \uparrow \bullet \text{valeur} < \text{val} \leq \text{prec} \uparrow \bullet \text{suivant} \uparrow \bullet \text{valeur}$

20/24

## Les listes chaînées — Mise à jour

### Ajout dans une liste ordonnée

```
1 Action ajout_après ( prec, val)
2   D : prec : Pcellule, val : <T>
3   L : p : Pcellule
4
5   p ← allouer()
6   p↑•valeur ← val
7   p↑•suivant ← prec↑•suivant
8   prec↑•suivant ← p
9 Faction
```

Cas limites

- ajout en queue : OK
- ajout en tête : pas de *prec* ⇒ algo ajout\_tête
- liste vide : idem

21/24

## Les listes chaînées — Mises à jour

### Suppression

Rechercher la cellule précédant celle à supprimer, soit *prec*

22/24

## Les listes chaînées — Mise à jour

### Suppression

```
1 Action sup_après ( prec )
2   D : prec : Pcellule
3   L : p : Pcellule
4
5   p ← prec↑•suivant
6   prec↑•suivant ← p↑•suivant
7   libérer (p)
8 Faction
```

Cas limites

- en queue : ok
- en tête : pas de *prec* ⇒ action sup\_tête

23/24

## Les listes chaînées — Mise à jour

### Suppression

```
1 Action sup_tête ( l )
2   D/R : l : Liste chaînée
3   L : p : Pcellule
4
5   p ← l
6   l ← l↑•suivant
7   libérer (p)
8 Faction
```

24/24