TAD Collections et conception

Algo – Chapitre 6

Pile<Element> **Gestion LIFO** estVide $\mathcal{P} \rightarrow \mathbb{B}$ empiler $\mathcal{P} \times \mathbb{E} \rightarrow \mathcal{P}$ dépiler $\mathcal{P} \rightarrow \mathcal{P}$ (non(estVide(p))) obtenirElement $\mathcal{P} \rightarrow \mathbb{E}$ (non(estVide(p)))	**Type** *Pile* = *LC*	estVide → estVide empiler → ajouter dépiler → supprimerTete obtenirElement → obtenirElement
File<Element> **Gestion FIFO** estVide $\mathcal{F} \rightarrow \mathbb{B}$ enfiler $\mathcal{F} \times \mathbb{E} \rightarrow \mathcal{F}$ défiler $\mathcal{F} \rightarrow \mathcal{F}$ (non(estVide(f))) obtenirElement $\mathcal{F} \rightarrow \mathbb{E}$ (non(estVide(f)))	**Type** *File* = *LC*	estVide → estVide enfiler → ajouter en fin de liste défiler → supprimerTete obtenirElement → obtenirElement
Liste<Element> **≈ Tableau** estVide $\mathcal{L} \rightarrow \mathbb{B}$ insérer $\mathcal{L} \times \mathbb{N} \times \mathbb{E} \rightarrow \mathcal{L}$ (0 < i ≤ longueur(l) + 1) supprimer $\mathcal{L} \times \mathbb{N} \rightarrow \mathcal{L}$ (0 < i ≤ longueur(l)) obtenirElement $\mathcal{L} \times \mathbb{N} \rightarrow \mathbb{E}$ (0 < i ≤ longueur(l)) longueur $\mathcal{L} \rightarrow \mathbb{N}$	**Type** *Liste* = **Structure** els : *LC* nbEls : *Naturel* **finstructure**	estVide → estVide insérer → insére à la ième place & inc. nbEls supprimer → supprimer ième el & dec. nbEls obtenirElement → obtenir ième élément longueur → champ nbElements
ListeOrdonnee<Element> **≈ Tableau trié** estVide $\mathcal{L} \rightarrow \mathbb{B}$ insérer $\mathcal{L} \times \mathbb{E} \rightarrow \mathcal{L}$ supprimer $\mathcal{L} \times \mathbb{N} \rightarrow \mathcal{L}$ (0 < i ≤ longueur(l)) obtenirElement $\mathcal{L} \times \mathbb{N} \rightarrow \mathbb{E}$ (0 < i ≤ longueur(l)) longueur $\mathcal{L} \rightarrow \mathbb{N}$	**Type** *ListeOrdonnee* = **Structure** els : *LC* nbEls : *Naturel* **finstructure**	estVide → estVide insérer → insérer où il faut & inc. nbEls supprimer → supprimer ième el & dec. nbEls obtenirElement → obtenir ième élément longueur →champ nbElements
Ensemble<Element> ajouter $\mathcal{E} \times \mathbb{E} \rightarrow \mathcal{E}$ retirer $\mathcal{E} \times \mathbb{E} \rightarrow \mathcal{E}$ estPresent $\mathcal{E} \times \mathbb{E} \rightarrow \mathbb{B}$ cardinalite $\mathcal{E} \rightarrow \mathbb{N}$ union $\mathcal{E} \times \mathcal{E} \rightarrow \mathcal{E}$ intersection $\mathcal{E} \times \mathcal{E} \rightarrow \mathcal{E}$ soustraction $\mathcal{E} \times \mathcal{E} \rightarrow \mathcal{E}$	**Type** *Ensemble* = **Structure** els : *LC* nbEls : *Naturel* **finstructure**	ajouter → si non présent, ajouter & inc. nbEls retirer → si présent, supprimer el & dec. nbEls estPrésent cardinalité → champ nbEls union soustraction intersection
Dictionnaire<Clé,Valeur> **≈ Tableau associatif** ajouter $\mathcal{D} \times \mathbb{C} \times \mathbb{V} \rightarrow \mathcal{D}$ retirer $\mathcal{D} \times \mathbb{C} \rightarrow \mathcal{D}$ estPresent $\mathcal{D} \times \mathbb{C} \rightarrow \mathbb{B}$ obtenirValeur $\mathcal{D} \times \mathbb{C} \rightarrow \mathbb{V}$ obtenirClefs $\mathcal{D} \rightarrow \mathcal{L}<\mathbb{C}>$ obtenirValeurs $\mathcal{D} \rightarrow \mathcal{L}<\mathbb{V}>$		
ListeChainee<Element> estVide $\mathcal{L} \rightarrow \mathbb{B}$ ajouter $\mathcal{L} \times \mathbb{E} \rightarrow \mathcal{L}$ obtenirElement $\mathcal{L} \rightarrow \mathbb{E}$ (non(estVide(l))) obtenirListeSuivante $\mathcal{L} \rightarrow \mathcal{L}$ (non(estVide(l)))	**Type** *LC* = ^*Noeud* **Type** *Noeud* = **Structure** el : *Element* listeSuiv : *LC* **finstructure**	**fonction** listeVide () : *LC* **fonction** estVide (liste : *LC*) : *Booleen* **procédure** ajouter (**E/S** liste : *LC*, **E** element : *El*) **fonction** obtenirElement (liste : *LC*) : *El* **fonction** obtenirListeSuivante (liste : *LC*) : *LC* **procédure** fixerListeSuivante (**E** liste : *LC*, **E** nelleSuite : *LC*) **procédure** supprimerTete (**E/S** liste : *LC*) **procédure** supprimer (**E/S** liste : *LC*)
ArbreBinaire<Element> ajouterRacine $\mathbb{E} \times \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$ estVide $\mathcal{A} \rightarrow \mathbb{B}$ obtenirElement $\mathcal{A} \rightarrow \mathbb{E}$ (non(estVide(a))) obtenirFilsGauche $\mathcal{A} \rightarrow \mathcal{A}$ (non(estVide(a))) obtenirFilsDroit $\mathcal{A} \rightarrow \mathcal{A}$ (non(estVide(a)))	**Type** *AB* = ^*Noeud* **Type** *Noeud* = **Structure** element : *Element* filsGauche : *AB* filsDroit : *AB* **finstructure**	**fonction** arbreBinaire () : *AB* **fonction** estVide (a : *AB*) : *Booleen* **fonction** ajouterRacine (fg, fd : *AB*, element : *El*) : *AB* **fonction** obtenirElement (a : *AB*) : *El* **fonction** obtenirFilsG/D (a : *AB*) : *AB* **procédure** fixerFilsG/D (**E** a : *AB*, **E** ag/d : *AB*) **procédure** supprimerRacine (**E/S** a : *AB*, **E** ag, ad : *AB*) **procédure** supprimer (**E/S** a : *AB*)
ABR<Element>	**Type** *ABR* = *AB*	estVide → estVide insérer

TAD Collections et conception

Algo – Chapitre 6

estVide $\mathcal{A} \rightarrow \mathbb{B}$ insérer $\mathcal{A} \times \mathbb{E} \rightarrow \mathcal{A}$ supprimer $\mathcal{A} \times \mathbb{E} \rightarrow \mathcal{A}$ estPresent $\mathcal{A} \times \mathbb{E} \rightarrow \mathbb{B}$ obtenirElement $\mathcal{A} \rightarrow \mathbb{E}$ (non(estVide(a))) obtenirFilsGauche $\mathcal{A} \rightarrow \mathcal{A}$ (non(estVide(a))) obtenirFilsDroit $\mathcal{A} \rightarrow \mathcal{A}$ (non(estVide(a)))	A_g et le A_d sont des ABR les elts de A_g st $\leq e$ les elts de A_d st $> e$	supprimer estPresent obtenirElement → obtenirElement obtenirFilsGauche → obtenirFilsGauche obtenirFilsDroit → obtenirFilsDroit
Arbre<Element> ajouterRacine $\mathbb{E} \times \mathcal{L}<\mathcal{A}> \rightarrow \mathcal{A}$ estVide $\mathcal{A} \rightarrow \mathbb{B}$ obtenirElement $\mathcal{A} \rightarrow \mathbb{E}$ (non(estVide(a))) obtenirFils $\mathcal{A} \rightarrow \mathcal{L}<\mathcal{A}>$ (non(estVide(a)))	**Type** *Arbre = AB* avec : • le fils gauche pour représenter le premier fils d'un nœud de l'arbre • le fils droit pour représenter le prochain frère fils d'un nœud de l'arbre	

I. Représentation possibles des TAD

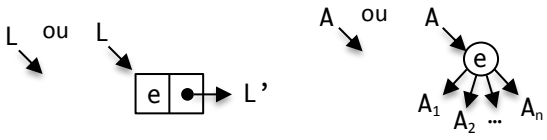
- Structure contenant tableaux et nombre d'éléments
- SDD ou structure contenant une SDD

II. Utilisation des TAD

```
Type File = ListeChainee
Type FileDEntiers = ListeChainee<Entier>
f : File<Entier>
```

III. TAD récursifs

Un(e) liste chainée/arbre est soit :
- un(e) L/A vide **()**,
- un élément suivi d'un(e) L/A **(e, L') / (e, A_1, …, A_n)**.

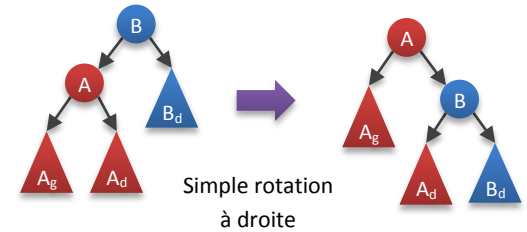


1. Vocabulaire des arbres

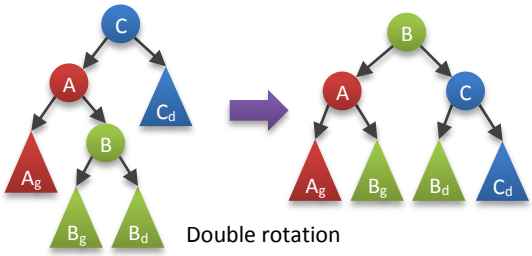
- **Arité d'un nœud :** Nombre de sous-arbres non vides
- **Chemin :** Séquences de nœuds liés (longueur = nombre d'arc)
- **Niveau d'un nœud :** Longueur jusqu'à la racine.
- **Hauteur :** Maximum des niveaux des nœuds.
- **Feuille :** Nœud sans sous-arbre non vide.

2. Les ABR/AVL

- Insérer : Parcours jusqu'à trouver une branche vide
  - Equilibrage par rotations après insertion
- Supprimer : Parcours jusqu'à trouver élément, puis :
  - fg vide et fd vide : supprimer
  - fg vide ou fd vide : supprimer et mettre le non vide a la place
  - aucun vide : plus grand des plus petits nouveau nœud
  - Pour AVL, descendre le nœud à supprimer par rotations sans déséquilibrer l'arbre avant de le supprimer



Simple rotation à droite



Double rotation à droite