

Objectifs

À la fin de cette séance, vous devriez être capable de :

- écrire des fonctions de manipulation pour un type abstrait dynamique;
- jongler avec des ABRI ...

Exercice 1 :

On étudie dans ce TD une structure de données qui permet de gérer les réservations d'un équipement pendant certaines périodes de temps. Pour cela, on considère des arbres binaires dont les nœuds sont étiquetés par des intervalles (par exemple d'entiers). Un nœud contenant l'intervalle $[a..b]$ représente alors une réservation de l'instant a à l'instant b .

Pour faciliter l'utilisation de la structure à la manière des arbres binaires de recherche, nous définissons la relation suivante entre intervalles :

$$[a..b] < [c..d] \Leftrightarrow b \leq c$$

Pourquoi cette relation est-elle bien adaptée pour établir un système de réservations et pour construire des arbres binaires de recherche d'intervalles ?

1. Mise en route

- a) Dessinez un ABRI d'au moins 6 nœuds. Est-il possible que des intervalles se chevauchent ? Est-il possible que des intervalles se « touchent » ?
- b) Définissez le type abstrait arbre, et les primitives d'accès et de construction qui vous seront utiles.
- c) Écrivez une fonction qui calcule la somme des longueurs des intervalles d'un ABRI.
- d) À quelle condition est-il possible d'insérer un intervalle $[a..b]$ dans un ABRI ? Écrivez une fonction qui insère un intervalle dans un ABRI. Précisez la convention adoptée pour indiquer si l'insertion a été effectivement possible.

2. Parcours

- a) Écrivez une fonction qui vérifie si un arbre binaire d'intervalles est un ABRI.
- b) Écrivez un fonction qui calcule la longueur du plus grand "trou" entre deux intervalles (d'un ABRI).

3. Transformation

- a) Écrivez une fonction qui transforme un ABRI en "tassant" tous les intervalles à gauche : les intervalles seront alors tous contigus.

Correction de l'exercice 1

1. Mise en route

- a)
- b) Opérations :
 - Vide : $() \rightarrow \text{ABRI}$
 - EstVide : $\text{ABRI} \rightarrow \text{bool}$
 - Inf : $\text{ABRI} \rightarrow \text{entier}$
 - Sup : $\text{ABRI} \rightarrow \text{entier}$
 - G : $\text{ABRI} \rightarrow \text{ABRI}$
 - D : $\text{ABRI} \rightarrow \text{ABRI}$
 - Créer : $\text{ABRI}, \text{entier}, \text{entier}, \text{ABRI} \rightarrow \text{ABRI}$

Axiomes :

- $\text{Inf}(\text{Creer}(g, \text{inf}, \text{sup}, d)) = \text{inf}$
- etc idem pour Sup , G et D
- $\text{EstVide}(\text{Vide}()) = \text{vrai}$
- $\text{EstVide}(\text{Creer}(A, a, b, B)) = \text{faux}$

On utilisera les notations allégées :

- $A.\text{Inf}$ pour $\text{Inf}(A)$ etc

Et la primitive $\text{Creer_feuille}(a, b)$ équivalent à $\text{Creer}(\text{Vide}(), a, b, \text{Vide}())$

- c) Il suffit d'effectuer un parcours quelconque de l'arbre, par exemple en profondeur, et d'additionner la longueur de l'intervalle de la racine avec les valeurs renvoyées par les 2 appels récursifs.
- d) L'insertion est possible si l'intervalle ne chevauche aucun des intervalles présents dans l'ABRI. La fonction *insérer* renvoie un booléen qui vaut vrai ssi l'insertion a été possible.

Données : un ABRI A , deux entiers a et b

Résultat : un booléen indiquant si l'insertion a été possible. l'ABRI est modifié par effet de bord.

Précondition : A est un ABRI, $a \leq b$

Postcondition : si $[a..b]$ ne chevauche aucun intervalle présent initialement dans A , cet intervalle a été inséré dans A .

$\text{Inserer}(A : \text{ABRI}, a, b : \text{entiers})$ renvoie booléen

si $a < A.\text{Inf} < b$ ou $a < A.\text{Sup} < b$ alors $\text{OK} = \text{faux}$

sinon

 si $b \leq A.\text{inf}$ alors

 si $\text{EstVide}(A.G)$ alors

$A.G = \text{Creer_feuille}(a, b)$

$\text{Ok} = \text{vrai}$

 sinon

$\text{OK} = \text{Inserer}(A.G, a, b)$

 sinon (idem à droite)

renvoyer Ok

2. Parcours

- a) Si A n'est pas vide, A est un ABRI ssi A est une feuille ou si
- ses sous-arbres gauche et droit (non vides) sont des ABRI
 - si $A.G$ n'est pas vide, la plus grande borne supérieure d'un intervalle du sous-arbre gauche de A est inférieure à $A.\text{Inf}$,
 - et idem à droite.

En un seul parcours, on effectue cette vérification en calculant par effet de bord les valeurs extrêmes des intervalles de l'ABRI.

$\text{EstABRI}(A : \text{ABRI} : \text{min}, \text{max} : \text{entiers résultats})$ renvoie booléen // précondition A n'est pas vide

si $\text{EstFeuille}(A)$ alors

$\text{min} = A.\text{Inf}$

$\text{max} = A.\text{Sup}$

$\text{Ok} = \text{vrai}$

sinon

 si $\text{EstVide}(A.G)$ alors

$\text{minG} = A.\text{Inf}$

$\text{maxG} = A.\text{Inf}$

$\text{OkG} = \text{vrai}$

 sinon

$\text{OkG} = \text{EstABRI}(A.G, \text{minG}, \text{maxG})$

 idem à droite

$\text{Ok} = \text{OkG}$ et OkD et $\text{maxG} \leq A.\text{Inf}$ et $A.\text{Sup} \leq \text{minD}$

```

    min = minG
    max = maxD
  renvoyer Ok

```

- b) En un seul parcours, on calcule le plus grand trou ainsi que les bornes extrémales de l'arbre.
 Données : un ABRI A non vide
 Résultats : 3 entiers trou, min et max
 Postcondition : trou est la taille du plus grand trou, min est la plus petite borne inf d'un intervalle de A, et max la plus grande borne sup

```

Le plus grand trou entre 2 intervalles de A est le maximum entre
— le plus grand trou dans son sous-arbre gauche
— le plus grand trou dans son sous-arbre droit
— le trou entre l'intervalle de la racine et la plus grande borne sup à gauche,
— le trou entre l'intervalle de la racine et la plus petite borne inf à droite.
PlusGrandTrou(A : ABRI, trou, min, max : entiers résultats)
si EstFeuille(A) alors
    min=A.inf; max=A.sup; trou=0
sinon
    si EstVide(A.G) alors
        minG=A.inf; maxG=A.inf; trouG=0;
    sinon
        PlusGrandTrou(A.G, trouG, minG, maxG)
    idem à droite
    trou=max(trouG, trouD, A.inf-maxG, minD-A.sup)
    min=minG
    max=maxD

```

3. Transformation

- a) Principe : l'intervalle le plus à gauche est conservé, et on tasse les autres intervalles contre lui.
 Données : un ABRI A non vide
 Résultats : ABRI A (transformé), et un entier max
 Postcondition : A est tassé à gauche (son intervalle le plus à gauche est inchangé, et il n'y a plus aucun trou), max est la plus grande borne supérieure d'un intervalle de A.

```

TasserGauche(A : ABRI donnée-résultat; max : entier résultat)
  TasserGaucheContre(A, BorneInf(A), max)
où BorneInf(A) est la borne inférieure de l'intervalle le plus à gauche de A et où TasserGaucheContre(A, min, max) tasse l'arbre A contre la valeur min, et renvoie dans max la borne sup de l'arbre tassé :

```

```

TasserGaucheContre(A : ABRI donnée-résultat; min : entier; max : entier résultat)
si EstVide(A.G) alors
    maxG=min
sinon
    TasserGaucheContre(A.G, min, maxG)
  A.sup=A.sup-A.inf+maxG
  A.inf=maxG
si EstVide(A.D) alors
    max=A.sup
sinon
    TasserGaucheContre(A.D, A.sup, maxD)
  max=maxD

```