Devoir 4. Arbres binaires de recherche

Ce devoir vaut 60 points et vous pouvez avoir jusqu'à 10 points boni. Travaillez seul ou en un équipe de deux.

4.1 TAD Partition (30 points)

Une partition de nombres non-négatifs est l'ensemble de segments[1] $\{[0, x_1), [x_1, x_2), \ldots, [x_m, \infty)\}$ avec $m \geq 0$ points de rupture $x_0 = 0 < x_1 < x_2 < \cdots < x_m$. L'interface du type abstrait représentant une partition comprend les opérations suivantes avec un paramètre $x > 0$:

recherche(x)	retourne l'intervalle $[x_{i-1}, x_i)$ qui contient x
fusion(x)	recherche l'intervalle avec $x \in [x_{k-1}, x_k)$, et remplace les intervalles $[x_{k-1}, x_k)$ et $[x_k, x_{k+1})$ par $[x_{k-1}, x_{k+1})$
tranche(x)	recherche $x \in [x_{k-1}, x_k)$, et remplace l'intervalle $[x_{k-1}, x_k)$ par $[x_{k-1}, x)$ et $[x, x_k)$

▶ Proposer une structure efficace pour implanter le type abstrait avec les trois opérations. La structure doit supporter les trois opérations en temps $O(\log m)$ au pire.

[1] Un segment est une intervalle $[a, b)$ avec $a \leq b$: c'est l'ensemble de valeurs réelles

$$\{x : a \leq x < b\}.$$

Il est permis d'avoir $b = \infty$: c'est interpreté comme $[a, \infty) = \{x : a \leq x\}$. Notez que $[a, a)$ est l'ensemble vide.

4.2 *L'égalité avant tout ? (30+10 points)*

On a un arbre binaire de recherche défini par les variables N.key (clé), N.parent, N.left et N.right (enfants gauche et droit) à tout nœud interne N ; null représente un nœud externe. On considère la recherche d'une clé s :

```
ESEARCH(N, s)                          // recherche de s dans le sous-arbre du nœud N
E1  if N = null then return null       // recherche infructueuse
E2  if s = N.key then return N         // recherche fructueuse
E3  if s < N.key then return ESEARCH(N.left, s)
E4  else return ESEARCH(N.right, s)    // {s > N.key}
```

On peut être plus économique en performant les comparaisons dans un autre ordre :

```
LSEARCH(N, s)                          // recherche de s dans le sous-arbre du nœud N
L1  if N = null then return null       // recherche infructueuse
L2  if s < N.key then return LSEARCH(N.left, s)
L3  if s = N.key then return N         // recherche fructueuse
L4  else return LSEARCH(N.right, s)    // {s > N.key}
```

Alors que l'algorithme ESEARCH fait toujours deux comparaisons arithmétiques à chaque nœud N avec $s \neq N$.key (lignes E2 et E3), LSEARCH en fait juste une quand $s < N$.key (ligne L2). On définit alors le coût de la recherche comme le nombre d'opérations de comparaison entre clés (lignes L2, L3) jusqu'à terminaison.

a. Coût à un nœud. Soit $t(N)$ le **coût**, ou le nombre de comparaisons arithmétiques dans LSEARCH(s) finissant avec nœud N, sans compter la dernière comparaison de la recherche fructueuse avec N.key $= s$.

▶ Donner[2] une *définition récursive* de $t(N)$.

b. Calcul du coût moyen. ▶ Donner[3] un algorithme qui calcule le coût moyen de la recherche fructueuse avec LSEARCH.

c. Borne supérieure. Soit $L(t)$ le nombre de nœuds internes trouvés en effectuant exactement t comparaisons dans l'algorithme LSEARCH.

▶ Démontrer[4] que pour tout $t \geq 2$,

$$L(t) \leq L(t-1) + L(t-2), \quad \text{et} \quad L(t) \leq F_t$$

où F_k dénote le k-ème nombre Fibonacci.

d. Arbre optimal. Le coût de recherche fructueuse au pire est le maximum de $t(N)$ dans l'arbre : $t_{\max} = \max\{t\colon L(t) > 0\}$. ▶ Caractériser les arbres avec le maximum nombre de nœuds internes pour un $t_{\max}$ donné.[5]

▶▶ e. (10 points boni) Donner (avec preuve) une borne inférieure sur le coût de recherche fructueuse au pire avec LSEARCH dans un ABR à n nœuds internes.[6]

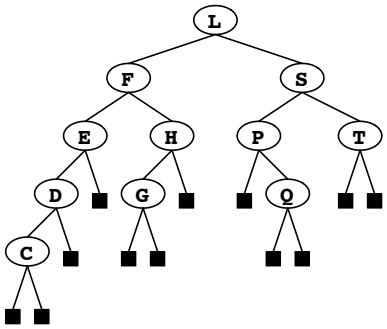

FIG. 1: ABR de l'Exercice 4.2.

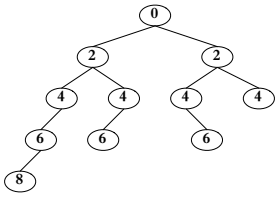

FIG. 2: Recherche fructeuse avec ESEARCH dans l'arbre de Fig. 1 : nombre d'opérations de comparaison sans compter le dernier (succès de $s = N$.key). Coût moyen $= (0+2+2+4+4+4+4+6+6+6+8)/11 = 46/11$.

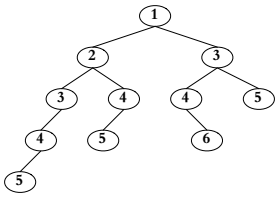

FIG. 3: Recherche fructeuse avec LSEARCH dans l'arbre de Fig. 1 : nombre d'opérations de comparaison sans compter le dernier (succès de $s = N$.key). Coût moyen $= (1+2+3+3+4+4+5+4+5+6+5)/11 = 42/11$.

[2] **Indice :** Il y a une comparaison à compter en Ligne L2 et une autre en Ligne L3. Notez que t croît différemment vers la gauche et vers la droite.

[3] **Indice :** utilisez la récurrence de **a.** pour calculer les coûts, et identifiez le parcours approprié pour sommer les coûts, et compter les nœuds en même temps.

[4] **Indice :** examinez comment la récurrence en **a.** mène à une inégalité pour $L(t)$.

[5] **Indice :** Allant peut-être à l'encontre de l'intution première, l'arbre optimal n'est pas un arbre binaire complet mais il a une structure récursive très bien définie…

[6] **Indice :** écrire $n = \sum_t L(t)$ et utiliser **c.**. Afin de simplifier une somme de termes récursivement définis comme $S(n) = \sum_{k=1}^{n} F_k$, développez une récurrence pour la somme en remplaçant les termes un à un : $S(n) = \sum_k (F_{k-1} + F_{k-2}) = S(n-1) \pm \cdots$