

NF16 - Arbre Binaire de Recherche

Exercice 1. Rappels sur les ABR

Objectif : Rechercher efficacement.

Définition :

```
pour tout x      cle[x]
                  gauche[x]
                  droit[x]

pour tout x de A  si gauche[x] != NIL
                  alors cle[x] > cle[gauche[x]]
                  si droit[x] != NIL
                  alors cle[x] < cle[droit[x]]
```

Exemples d'arbre et de différentes opérations...

Exercice 2. Final printemps 2003

Implémentation :

```
typedef struct noeud {
    int cle;
    struct noeud* gauche;
    struct noeud* droit;
} noeud;
typedef struct noeud* arbre;

/**
 * renvoie un pointeur sur le noeud qui contient la valeur max
 * ou NULL si A est vide
 */
noeud* max(arbre A) {
    if (A == NULL)
        return NULL;
    while (A->droit != NULL)
        A = A->droit;
    return A;
}

/**
 * renvoie un pointeur sur le noeud qui contient la valeur min
```

```

* ou NULL si A est vide
*/
noeud* min(arbre A) {
    if (A == NULL)
        return NULL;
    while (A->gauche != NULL)
        A = A->gauche;
    return A;
}

/**
 * renvoie un pointeur sur le noeud qui contient la plus petite cle >= v
 * ou NULL si ce noeud n'existe pas
 */
noeud* succ(arbre A, int v) {
    if (A == NULL)
        return NULL;
    if (A->cle == v)
        return A;
    if (A->cle < v)
        return succ(A->droit, v);
    tmp = succ(A->gauche, v);
    if (tmp == NULL)
        return A;
    else
        return tmp;
}

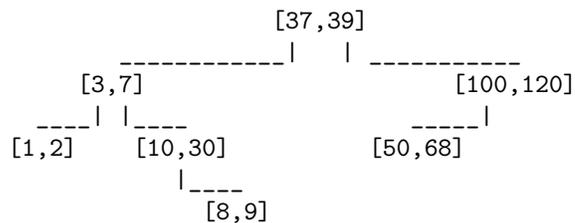
```

Exercice 3.

```

pour tout noeud x    droit[x]
                    gauche[x]
                    u[x]      // 1er jour          <-sert de clé
                    v[x]      // dernier jour

```



```

/**
 * renvoie vrai si [p, d] intersecte avec un interval de A et faux sinon
 * complexité : O(h)
 */
bool intersecte(A: noeud racine, p: jour arrivée, d: jour départ) {
    si (A = NIL) alors
        retourner faux;
    si (d < u[A]) alors
        retourner intersecte(gauche[A], p, d);
    si (d > u[A]) alors
        retourner intersecte(droit[A], p, d);
    // on a d >= u[A] et p <= v[A] : il y a intersection
    retourner vrai;
}

```

Preuve :

L'appel `intersecte(NIL, p, d)` termine en 1 appel en renvoyant faux. Cet appel est donc valide car `[p, d]` n'intersecte avec aucun interval d'un arbre vide. (Notons que la hauteur de NIL est égale à -1)

Supposons que pour tout arbre A de hauteur $h(A) \leq k$, `intersecte(A, p, d)` termine en au plus $h(A) + 1$ appels et est valide.

Soit A un arbre de hauteur $k + 1$ et considérons l'appel `intersecte(A, p, d)`. On a $A \neq \text{NIL}$ si $d < u[A]$, alors `[p, d]` n'intersecte pas avec `[u[A], v[A]]`. De plus si il y a intersection, c'est alors forcément avec un intervalle de `gauche[a]`. On appelle alors `intersecte(gauche[A], p, d)`. Or $h(\text{gauche}[A]) \leq k$ donc cet appel termine en au plus $k + 1$ appels et est valide. De même si $p > v[A]$, on appelle `intersecte(droit[A], p, d)` qui termine en au plus $k + 1$ appels et est valide car $h(\text{droit}[A]) \leq k$.

Ainsi `intersecte(A, p, d)` termine en au plus $k + 2$ et est valide?

On en conclut que pour tout A `intersecte(A, p, d)` termine et est valide. L'algorithme fait au plus $h(A) + 1$ apples en $O(1)$. Il est donc en $O(h(A))$.

	Ajouter	Supprimer	Intersecte
ABR	$O(h)$	$O(h)$	$O(h)$
liste chaînée	$O(1)$ si intersecte, $O(n)$ sinon	$O(1)$ si intersecte, $O(n)$ sinon	$O(n)$
liste chaînée triée	$O(n)$	$O(n)$	$O(n)$