Hachage direct (par calcul) : les algorithmes

Les algorithmes ci-dessous utilisent :
— La valeur entière m ;
— la fonction de hachage `h(`x`)` qui retourne la valeur de hachage de l'élément x, un entier dans $[0, m-1]$
— la fonction d'essais successifs `essai(`i`, `x`)` qui retourne le résultat du $i^{ème}$ essai ($i \in [1, m]$) pour la valeur x

Sans suppression

Le tableau de hachage `th` (de type `tab_hachage`) indicé de 0 à $m-1$ est ici un simple vecteur d'éléments.

Une case est vide ou occupée. On suppose écrite la fonction `estvide(th, i)` qui indique si la case `i` de `th` est vide.

La fonction de recherche retourne la position de l'élément dans le tableau de hachage, -1 s'il n'est pas présent.

```
fonction recherche_HD(tab_hachage th, element x) : entier
variables
    entier  i, v
debut
    v ← essai(1, x)   /* ou h(x) */
    i ← 1
    tant que non estvide(th, v) et (th[v] <> x) faire
        i ← i + 1
        si i > m alors
            retourne -1   /* m essais */
        fin si
        v ← essai(i, x)
    fin tant que
    si estvide(th, v) alors
        retourne -1
    sinon
        retourne v
    fin si
fin


procedure ajouter_HD(tab_hachage ref th, element x)
variables
    entier  i, v
debut
    v ← essai(1, x)   /* ou h(x) */
    i ← 1
    tant que non estvide(th, v) et th[v] <> x faire
        i ← i + 1
        si i > m alors
            /* erreur : tableau plein */
        fin si
        v ← essai(i, x)
    fin tant que
    si estvide(th, v) alors
        th[v] ← x
    sinon
        /* pas d'ajout, élément déjà présent */
    fin si
fin
```

Hachage direct (par calcul) : les algorithmes

Avec suppressions

Une case est vide, **libre** ou occupée.
Le tableau `th` (de type `tab_hachage`) indicé de 0 à $m-1$ dont chaque case contient ici :
— `th[i].elt` un élément
— `th[i].etat` une valeur dans (*vide*, *libre*, *occupée*).

```
fonction recherche_HD(tab_hachage th, element x) : entier
variables
    entier  i, v
debut
    v ← essai(1, x)   /* ou h(x) */
    i ← 1
    tant que th[v].etat <>  vide et (th[i].etat = libre ou th[v] <> x) faire
        i ← i + 1
        si i > m alors
            retourne -1   /* m essais */
        fin si
        v ← essai(i, x)
    fin tant que
    si th[v].etat =  vide alors
        retourne -1
    sinon
        retourne v (*)
    fin si
fin

procedure ajouter_HD(tab_hachage ref th, element x)
variables
    entier  i, v
debut
    v ← essai(1, x)   /* ou h(x) */
    i ← 1
    lib ← -1
    tant que i <= m et th[v].etat <> vide et (th[v].etat = libre ou th[v] <> x) faire
        si th[v].etat = libre et lib = -1 alors
            lib ← v
        fin si
        i ← i + 1
        si i <= m alors
            v ← essai(i, x)
        fin si
    fin tant que

    si th[v].etat = occupée et th[v].elt = x alors
        /* pas d'ajout, élément déjà présent */ (*)
    sinon
        si lib <> -1 alors
            th[lib].elt ← x
            th[lib].etat ← occupée
        sinon
            si th[v].etat = libre alors
                th[v].elt ← x
                th[v].etat ← occupée
            sinon
                /* erreur : tableau plein */
            fin si
        fin si
    fin si
fin
```

(*) Il pourrait être intéressant de "remonter" la valeur de x à la première place *libre* rencontrée.