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 $[0, m-1]$
— le tableau de hachage `th` (de type `tab_hachage`) indicé de 0 à $m-1$ dont chaque case contient :
  — `th[i].elt` un élément
  — `th[i].lien` un entier représentant l'indice dans le tableau du suivant dans la "liste" des collisions, ou à -1 s'il n'y a pas de suivant

Sans suppression

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_HC(tab_hachage th, element x) : entier
variables
    entier  i
debut
    i ← h(x)
    si estvide(th, i) alors
        retourne -1
    fin si
    tant que (i <> -1) et (th[i].elt <> x) faire
        i ← th[i].lien
    fin tant que
    retourne i
fin


procedure ajouter_HC(tab_hachage ref th, element x)
variables
    entier  i, r
debut
    i ← h(x)
    si estvide(th, i) alors
        th[i].elt ← x
        th[i].lien ← -1
    sinon
        tant que (th[i].elt <> x) et (th[i].lien <> -1) faire
            i ← th[i].lien
        fin tant que
        si th[i].elt = x alors
            /* pas d'ajout, élément déjà présent */
        sinon
            r ← m-1
            tant que (r >= 0) et non estvide(th, r) faire
                r ← r-1
            fin tant que
            si r >= 0 alors
                th[i].lien ← r
                th[r].elt ← x
                th[r].lien ← -1
            sinon
                /* erreur : tableau plein */
            fin si
        fin si
    fin si
fin
```

Avec suppressions

Une case est vide, **libre** ou occupée : on ajoute à chaque case du tableau de hachage un champ `etat`.

```
fonction recherche_HC(tab_hachage th, element x) : entier
variables
    entier  i
debut
    i ← h(x)
    si estvide(th, i) alors
        retourne -1
    fin si
    tant que (i <> -1) et (th[i].etat = libre ou (th[i].elt <> x)) faire
        i ← th[i].lien
    fin tant que
    retourne i      (*)
fin


procedure ajouter_HC(tab_hachage ref th, element x)
variables
    entier  i, lib, r
debut
    i ← h(x)
    si th[i].etat = vide alors
        th[i].elt ← x
        th[i].lien ← -1
        th[i].etat ← occupée
    sinon
        lib ← -1
        tant que (th[i].etat = libre ou th[i].elt <> x) et (th[i].lien <> -1) faire
            si th[i].etat = libre et lib = -1 alors
                lib ← i
            fin si
            i ← th[i].lien
        fin tant que
        si th[i].elt = x alors     (*)
            /* pas d'ajout, élément déjà présent */
        sinon
            si lib <> -1 alors
                th[lib].elt ← x
                th[lib].etat ← occupée
            sinon
                r ← m-1
                tant que (r >= 0) et (th[r].etat <> vide) faire
                    r ← r-1
                fin tant que
                si r >= 0 alors
                    th[i].lien ← r
                    th[r].elt ← x
                    th[r].lien ← -1
                    th[r].etat ← occupée
                sinon
                    /* erreur : tableau plein */
                fin si
            fin si
        fin si
    fin si
fin
```

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