Éléments d'Algorithmique
CMTD5: Recherche dichotomique

Christine Tasson
Université de Paris, IRIF

Structures de données

La plupart des bons algorithmes fonctionnent grâce à une bonne organisation des données, amenée par une méthode astucieuse.

Intuitivement, pour retrouver une carte dans un jeu, il est très utile que le jeu soit déjà trié

Algorithme de recherche d’un élément dans un tableau

Entrée : un tableau tab de taille n et un élément e.

Sortie : i tel que tab[i] = e ou NonTrouvé.

```
pour i de 0 à n-1 faire
          si tab[i] = e alors
          renvoyer i
renvoyer NonTrouvé
```

Complexité : O(n).

Sachant que la recherche dans un tableau est une opération de base utilisée dans de nombreux algorithmes, la complexité de cet algorithme est trop élevée.

Recherche d'un élément dans un tableau

Pour aller plus vite, on peut utiliser les tableaux triés et la dichotomie, ou méthode "diviser pour régner".

Idée : si le tableau tab est trié, pour tout indice i,

- les éléments e 6 tab[i] sont d'indice 6 i,
- les éléments e > tab[i] sont d'indice > i.

On essaye avec i au milieu du tableau.

Algorithme de recherche dichotomique

Algorithme RechDichoRec : recherche dans un tableau trié.

- Entrée : un tableau trié tab de taille n, un intervalle [min, max]
- Sortie : i tel que tab[i] = e ou NonTrouvé.

```
si min = max alors
        si tab[min] = e alors renvoyer min
        sinon renvoyer NonTrouvé
mid <- (min + max) / 2
si tab[mid] < e alors
        renvoyer RechDichoRec(tab, mid+1, max, e)
sinon renvoyer RechDichoRec(tab, min, mid, e)
```

Complexité : O(log2(n)).

On obtient une complexité bien meilleure que dans le cas précédent !

Remarque : la recherche dichotomique est récursive terminale.

Recherche dichotomique itérative

Voici la version itérative avec les même convention que précédemment.

Algorithme RechDichoIt : recherche dans un tableau trié.

```
    min <- 0
    max <- n - 1
    tant que min < max faire
        mid <- (min + max) / 2
        si tab[mid] < e alors min <- mid + 1
        Sinon max <- mid
    si tab[min] = e alors renvoyer min
    Sinon renvoyer NonTrouvé
```

Complexité : O(log2(n)).

Pour résumer

Trouver la position la plus centrale du tableau (si le tableau est vide, sortir).

- Comparer la valeur de cette case à l'élément recherché.
- Si la valeur est égale à l'élément, alors retourner la position, sinon reprendre la procédure dans la moitié de tableau pertinente.

Correction de l’algorithme

Récurrence sur la taille de l’intervalle $\beta - \alpha + 1 := m$ d’un tableau trié tab de taille $> m$. On recherche l’élément e.

Propriété à vérifier : pour tout tableau trié tab et pour tout intervalle $[\alpha, \beta]$ de tab, l’exécution se termine en renvoyant "NonTrouvé" si l’élément n’a pas été trouvé entre les indices α et β ou en renvoyant l’indice i dans tab tel que $tab[i] = e$.

Hypothèse de récurrence : la propriété est vraie pour tout tableau trié tab et pour tout intervalle $[\alpha, \beta]$ de tab tel que $\beta - \alpha + 1$ 6 m

Initialisation : si $m = 1$ alors $\alpha = \beta$. Si l’occurrence est trouvé l’algorithme renvoie α, sinon "NonTrouvé". La propriété est donc vraie pour $m = 1$.

Correction de l'algorithme

Hérédité : soit $[\alpha, \beta]$ tel que $\beta - \alpha + 1 := m + 1$ et $\gamma := (\alpha + \beta)/2$.

Alors l'exécution renvoie l'algorithme évalué soit sur $(tab, \gamma + 1, \beta, e)$, soit sur (tab, α, γ, e).

Comme $\beta - (\gamma + 1) + 1 \; 6 \; m$ et $\gamma - \alpha + 1 \; 6 \; m$, l'hypothèse de récurrence est vraie pour ces deux intervalles.

De plus, comme tab est trié, si l'élément e est dans tab alors il est nécessairement soit dans l'intervalle $[\gamma + 1, \beta]$, soit dans l'intervalle $[\alpha, \gamma]$. Par conséquent, la propriété est vraie pour un intervalle de taille $m + 1$.