Méthodes par dichotomie

S1-2 Algorithmique - Partie I

Lycée Carnot - Dijon, 2022 - 2023

Germain Gondor

Sommaire

Objectifs

A la fin de la séquence d'enseignement les élèves doivent :

Objectifs

A la fin de la séquence d'enseignement les élèves doivent :

- pouvoir citer les avantages à adopter une approche par dichotomie en comparaison avec une approche linéaire

Objectifs

A la fin de la séquence d'enseignement les élèves doivent :

- pouvoir citer les avantages à adopter une approche par dichotomie en comparaison avec une approche linéaire
- reconnaitre un cas où une approche par dichotomie est envisageable

Objectifs

A la fin de la séquence d'enseignement les élèves doivent :

- pouvoir citer les avantages à adopter une approche par dichotomie en comparaison avec une approche linéaire
- reconnaitre un cas où une approche par dichotomie est envisageable
- savoir coder en **Python** un algorithme de recherche dichotomique dans un tableau trié

Objectifs

A la fin de la séquence d'enseignement les élèves doivent :

- pouvoir citer les avantages à adopter une approche par dichotomie en comparaison avec une approche linéaire
- reconnaitre un cas où une approche par dichotomie est envisageable
- savoir coder en **Python** un algorithme de recherche dichotomique dans un tableau trié
- avoir quelques notions de coûts linéaire et logarithmique

Sommaire

Principe

Il est des situations où un problème de dimension n se trouve réduit à un problème de dimension $n//2$ à l'itération suivante.

Principe

Il est des situations où un problème de dimension n se trouve réduit à un problème de dimension $n//2$ à l'itération suivante.

Exemple : trouver un nombre entre 0 et 100 choisi par un opérateur (pas forcément humain) qui indique si la proposition est plus grande ou plus petite que la solution.

Principe

Il est des situations où un problème de dimension n se trouve réduit à un problème de dimension $n//2$ à l'itération suivante.

Exemple : trouver un nombre entre 0 et 100 choisi par un opérateur (pas forcément humain) qui indique si la proposition est plus grande ou plus petite que la solution.

Tester les valeurs par ordre croissant peut conduire à 101 propositions.... Avec une méthode par dichotomie, le nombre d'itérations est au maximum 7.

Principe

Il est des situations où un problème de dimension n se trouve réduit à un problème de dimension $n//2$ à l'itération suivante.

Exemple : trouver un nombre entre 0 et 100 choisi par un opérateur (pas forcément humain) qui indique si la proposition est plus grande ou plus petite que la solution.

Tester les valeurs par ordre croissant peut conduire à 101 propositions.... Avec une méthode par dichotomie, le nombre d'itérations est au maximum 7.

Arbre d'appel pour une valeur comprise entre 0 et 10

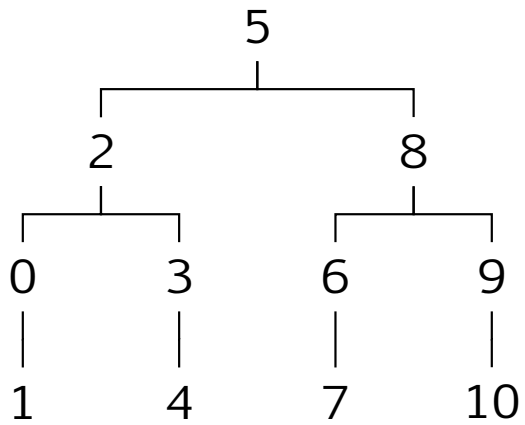
Principe

Il est des situations où un problème de dimension n se trouve réduit à un problème de dimension $n//2$ à l'itération suivante.

Exemple : trouver un nombre entre 0 et 100 choisi par un opérateur (pas forcément humain) qui indique si la proposition est plus grande ou plus petite que la solution.

Tester les valeurs par ordre croissant peut conduire à 101 propositions.... Avec une méthode par dichotomie, le nombre d'itérations est au maximum 7.

Arbre d'appel pour une valeur comprise entre 0 et 10



Méthode par dichotomie

Le principe est donc de diviser le problème en deux sous problèmes. On parle alors de méthode par dichotomie. Elle fait partie des stratégie de type *diviser pour mieux régner*.

L'étude de la complexité au second semestre montrera qu'une telle méthode à un coût en $\log(n)$ quand une approche naïve à un coût en n (coût linéaire).

Sommaire

Recherche dichotomique d'une valeur dans un tableau trié

Énonce : Recherche dichotomique (ou binary seach) d'une valeur dans un tableau de valeurs trié

> *Écrire une fonction* `recherche_dicho(T, x)` *qui prend en argument un tableau trié* `T` *de flottants et* `x` *un flottant. La fonction renvoie un indice de* `x` *si* `x` *est une des valeurs du tableau et* `-1` *sinon.*

Recherche dichotomique d'une valeur dans un tableau trié

Énonce : Recherche dichotomique (ou binary seach) d'une valeur dans un tableau de valeurs trié

Écrire une fonction `recherche_dicho(T, x)` qui prend en argument un tableau trié `T` de flottants et `x` un flottant. La fonction renvoie un indice de `x` si `x` est une des valeurs du tableau et `-1` sinon.

Principe :

Prendre une valeur au milieu du tableau. Si c'est la valeur recherchée, c'est gagné, sinon on la compare à la valeur recherchée et on prendre la moité du tableau où est potentiellement la valeur recherchée.

Algorithm 1 Recherche dichotomique dans un tableau trié

entrée: T un tableau de n valeurs et x un élément (pas forcément du tableau)
résultat: -1 si $x \notin T$, l'indice d'un x dans le tableau T sinon

```
1: recherche_dicho(T, x)
2: n← taille(T )
3: min,max ← 0,n-1
4: mil ← (min+max)//2
5: tant que min<max et T[mil]≠x faire
6:     si T[mil]<x alors
7:         min← mil+1
8:     sinon
9:         max ← mil-1
10:    fin si
11:    mil← (min+max)//2
12: fin tant que
13: si T[mil]≠x alors
14:    rep ← -1
15: sinon
16:    rep ← mil
17: fin si
18: renvoi: rep
```

Exponentiation rapide

Énonce : Exponentiation rapide

Écrire une fonction `puiss(x, n)` qui prend en argument un flottant `x` et un entier `n` et qui renvoie x^n.

Exponentiation rapide

Énonce : Exponentiation rapide

Écrire une fonction `puiss(x, n)` qui prend en argument un flottant `x` et un entier `n` et qui renvoie x^n.

Exemple : pour calculer x^5, on calcule x^2 puis x^4 et finalement x^5.

Exponentiation rapide

Énonce : Exponentiation rapide

Écrire une fonction `puiss(x, n)` qui prend en argument un flottant `x` et un entier `n` et qui renvoie x^n.

Exemple : pour calculer x^5, on calcule x^2 puis x^4 et finalement x^5.

On remarque que $x^9 = x.(x^4)^2 = x.\left(\left(x^2\right)^2\right)^2$.

On calcule x^2 puis x^4 puis x^8. On passe ainsi de n multiplications à $\log_2(n)$ multiplications.

Exponentiation rapide

Énonce : Exponentiation rapide

> *Écrire une fonction* `puiss(x, n)` *qui prend en argument un flottant* `x` *et un entier* `n` *et qui renvoie* x^n.

Exemple : pour calculer x^5, on calcule x^2 puis x^4 et finalement x^5.

On remarque que $x^9 = x.(x^4)^2 = x.\left(\left(x^2\right)^2\right)^2$.

On calcule x^2 puis x^4 puis x^8. On passe ainsi de n multiplications à $\log_2(n)$ multiplications.

Principe :

> *Se servir des résultats intermédiaires pour accélérer le processus de calcul de* x^n *en décomposant n en puissance de 2.*

Exponentiation rapide

Algorithm 2 Exponentiation rapide itérative

entrée: x un nombre réel et n un entier naturel
résultat: un nombre réel $r = x^n$
puiss(x, n)

```
1:  si n==0 alors
2:      renvoi: 1
3:  fin si
4:  r← 1
5:  tant que n > 0 faire
6:      si n modulo 2==1 alors
7:          r←r.x
8:      fin si
9:      x←x.x
10:     n←n//2
11: fin tant que
12: renvoi: r
```

Exponentiation rapide

Algorithm 3 Exponentiation rapide récursive

entrée: n un entier positif et x un nombre réel
résultat: un nombre réel $r = x^n$
puis_rec(x, n)

```
1: si n==0 alors
2:     renvoi: 1
3: sinon
4:     si n modulo 2==0 alors
5:         renvoi: puis_rec(x.x,n/2)
6:     sinon
7:         renvoi: x.puis_rec(x.x,(n-1)/2)
8:     fin si
9: fin si
```