

UEF 1 : Informatique & Programmation

Faculté des Sciences de Nice
DEUG 2001-2002

Jérôme DURAND-LOSE
Jean-Paul ROY

COURS 11

D D o D c o o e Dichotom e Dichotomie

11-2

Deviner un nombre entre 1 et 127

Se joue à deux

L'un décide d'un chiffre

L'autre doit le deviner le plus rapidement possible

Il propose un chiffre,
on lui dit si c'est le bon, plus grand, ou plus petit

Jouons ensemble !

11-3

Proposition

64

Minimum

1

Maximum

127

11-4

Algorithme

Variables

v : entier
 \min : entier = 1
 \max : entier = 127

Algorithme pour deviner la valeur **répéter**

$v \leftarrow (\min + \max) \text{ divisé par } 2$
si v est la valeur à deviner **alors**
 afficher v et sortir de la boucle
si v est plus petit la valeur à deviner **alors**
 $\min \leftarrow v$
si v est plus grand que la valeur à deviner **alors**
 $\max \leftarrow v$

fin répéter

Seule sortie
de la boucle !
(break, return ?)

11-5

Recherche d'un zéro d'une fonction

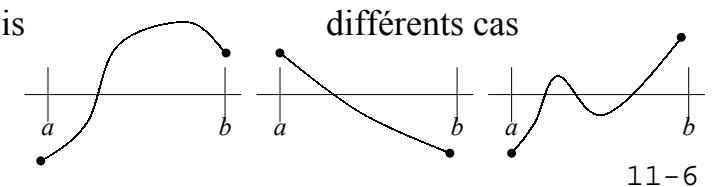
Hypothèses :

- * f continue
- * on connaît a et b tels que $f(a)$ et $f(b)$ sont de signes différents

But :

trouver une bonne approximation d'une racine de f

f doit s'annuler
au moins une fois
entre a et b



Méthode

On calcule $f\left(\frac{a+b}{2}\right)$

S'il est de signe différent de $f(a)$

on recommence avec a et $\frac{a+b}{2}$

sinon

on recommence avec $\frac{a+b}{2}$ et b

Jusqu'à ce que $|a-b|$ soit plus petit que la précision demandée

À chaque fois, la distance entre a et b est divisée par 2

11-7

Algorithme

Données

a : nombre approché
 b : nombre approché
 e : nombre approché

Valeur renvoyée

résultat : nombre approché

Algorithme pour trouver un zéro d'une fonction par dichotomie

tant que $|a-b| < e$ **faire**

si $f(a)$ et $f((a+b)/2)$ sont de signes différents

$b \leftarrow (a + b) / 2$

sinon

$a \leftarrow (a + b) / 2$

fin tant que

 résultat $\leftarrow (a + b) / 2$

11-8

Fonction en Java

```
static double f( double x ) {
    .....
}

static double dichotomieRacine( double a, double b, double e ) {
    while ( a + e < b ) { // e < | a-b |
        double c = ( a + b ) / 2 ;
        if ( f( a ) * f( c ) < 0 ) // f(a) et f(c) sont de signes !=
            b = c;
        else // f(c) et f(b) sont de signes !=
            a = c;
    }
    return ( a + b ) / 2;
}
```

11-9

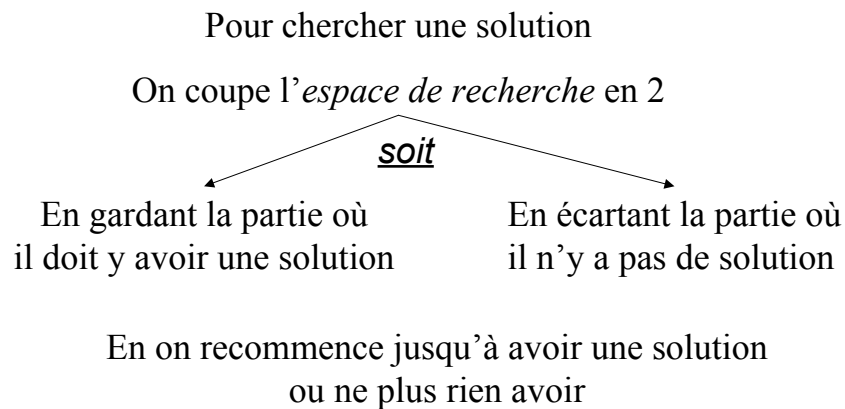
Exemple pour $f(x) = e^x - 2\cos(x)$

Précision	Racine	$f(\text{racine})$
0.1	0.53125	-0.02329165802136024
0.01	0.54296875	0.008751701667358391
0.001	0.53955078125	-6.429358394135498E-4
1.0E-4	0.539764404296875	-5.694561728719627E-5
1.0E-5	0.5397834777832031	-4.617447294963384E-6
1.0E-6	0.5397849082946777	-6.92784223765841E-7
1.0E-7	0.5397851765155792	4.309088375009651E-8
1.0E-8	0.5397851578891277	-8.011562524501414E-9
1.0E-9	0.5397851611487567	9.31365429224229E-10
1.0E-10	0.5397851608286146	5.3042237269096404E-11
1.0E-11	0.5397851608104247	3.1377123121956174E-12
1.0E-12	0.5397851608090605	-6.052935930256353E-13
1.0E-13	0.5397851608092594	-5.950795411990839E-14
1.0E-14	0.5397851608092843	8.659739592076221E-15
1.0E-15	0.5397851608092812	2.220446049250313E-16
1.0E-16	0.5397851608092812	2.220446049250313E-16
1.0E-17	ne termine jamais !	

Problèmes
des
aux
doubles !

11-10

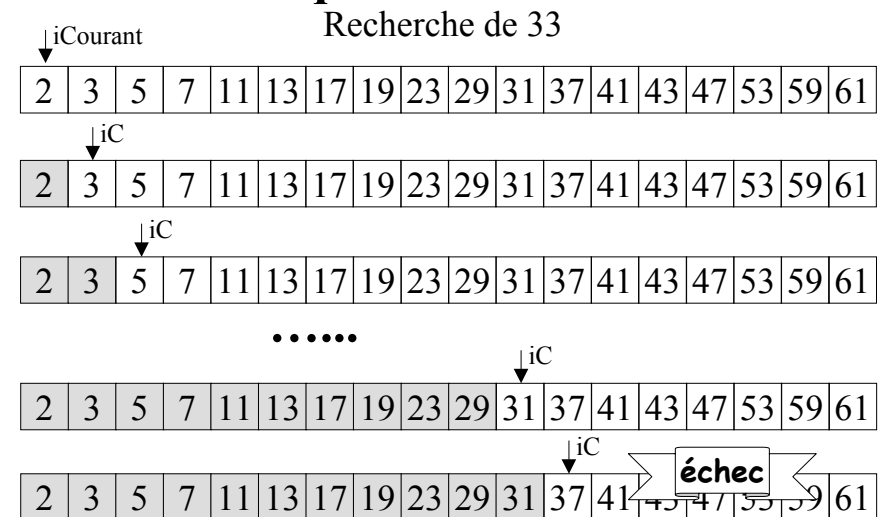
Plus généralement



Exemple type : recherche dans un dictionnaire

11-11

Recherche séquentielle dans un tableau trié



Complexité en $O(\text{nombre de valeurs})$

11-12

Recherche dichotomique

Conditions : *tableau trié* et *recherche compatible avec l'ordre*

Exemples (ordre alphabétique) recherches de :

- « DICHOTOMIE » dans un dictionnaire
- « SUMATRA » dans l'index d'un atlas

Principe : on regarde à chaque fois au milieu de la zone de recherche et selon ce que l'on trouve on recommence dans la première moitié ou la seconde.

A chaque fois on divise par deux la zone de recherche

4 coups pour 2^4 entrées

10 coups pour $2^{10} = 1024$ entrées

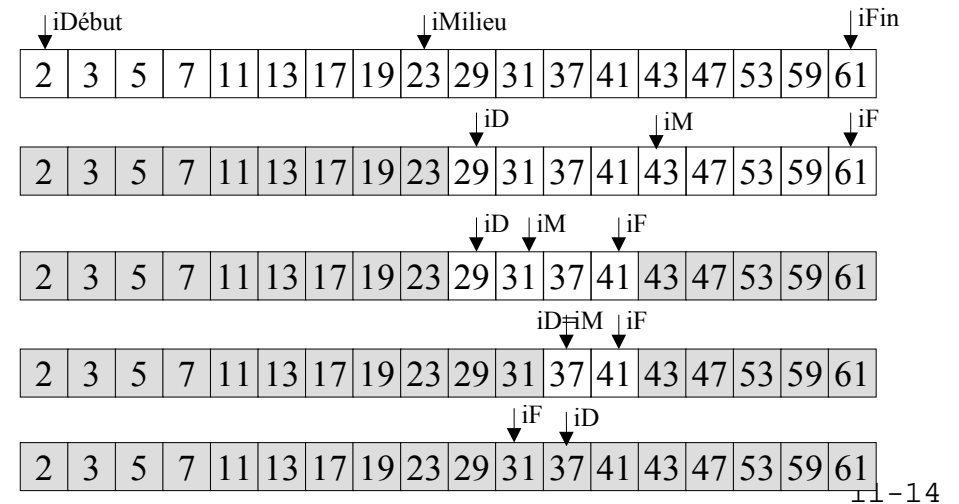
n coups pour 2^n entrées

$\log_2(n)$ coups pour n entrées

11-13

Exemple de recherche dichotomique

Recherche de 33



Recherche dichotomique

Données

un tableau trié (en ordre croissant)

un critère de recherche (compatible avec l'ordre du tableau)

Variables

iDébut, iFin, iMilieu : entiers

Valeur renvoyée

réponse : entier

Algorithme recherche dichotomique

tant que iDébut <= iFin

iMilieu ← (iDébut + iFin) / 2

si la case iMilieu contient la valeur cherchée

fin de l'algorithme, la réponse est iMilieu

sinon si la valeur en iMilieu est avant celle recherchée

iDébut ← iMilieu + 1

sinon

iFin ← iMilieu - 1

Recherche infructueuse

À chaque itération, la différence entre iDébut et iFin est divisée par deux

Fin quand on trouve

Fin quand on ne trouve pas

Complexité en $O(\log(\text{nombre de valeurs}))$

11-15

```
static final int NON_TROUVE = -1;
```

```
static int rechercherDichotomique( double[] t, double v ) {
```

```
    int debut = 0;
```

```
    int fin = t.length - 1;
```

```
    while ( debut <= fin ) {
```

```
        int milieu = ( debut + fin ) / 2;
```

```
        if ( t[ milieu ] == v )
```

```
            return milieu;
```

```
        else if ( t[ milieu ] < v )
```

```
            debut = milieu + 1;
```

```
        else
```

```
            fin = milieu - 1;
```

```
    }
```

```
    return NON_TROUVE;
```

```
}
```

Valeur trouvée
return met fin à la méthode

Tout a été parcouru sans succès

11-16