DIVISER POUR RÉGNER

1 Méthode générale

1.1 Principe

Pour résoudre récursivement un problème, on peut procèder en trois étapes :

Division diviser le problème en sous-problèmes du même type

Règne résoudre chacun des sous-problèmes par appel récursif

Fusion fusionner les résultats des sous-problèmes en la solution du problème de départ

En pratique, on veut calculer un objet y à partir d'un objet x :

- on divise l'objet x en b objets plus petits quasiment de même taille : $x_1, \ldots, x_b$
- on calcule des résultats $y_1, \ldots, y_a$ en appliquant a fois le procédé récursif sur les objets $x_1, \ldots, x_b$
- on fusionne les a résultats $y_1, \ldots, y_a$ en le résultat y

1.2 Complexité

Si x est de taille n, on note $C(n)$ la complexité du calcul de y.

Alors si on peut appliquer la stratégie précédente, on obtient la relation de récurrence suivante (en négligeant les variations de taille de plus ou moins une unité) :

$$C(n) = aC(n/b) + D(n) + F(n)$$

où

- a est le nombre d'appels récursifs
- b est le nombre de divisions de l'objet initial
- $D(n)$ est la complexité de l'étape de division
- $F(n)$ est la complexité de l'étape de fusion

Remarque. L'approximation des complexités est valable si la variation de la taille de l'objet de plus ou moins une unité ne change pas l'ordre de grandeur de la complexité : en pratique, cette condition est toujours vérifiée.

Théorème 1 *On suppose que $D(n) + F(n) = O(n^\alpha)$, alors $C(n) = aC(n/b) + O(n^\alpha)$ et on écrit a comme une puissance de b : $a = b^\beta$ (*i.e. *$\beta = \log_b a$). Dans ce cas, on a*

- ▷ *si $\alpha > \beta$, alors $C(n) = O(n^\alpha)$*
- ▷ *si $\alpha < \beta$, alors $C(n) = O(n^\beta)$*
- ▷ *si $\alpha = \beta$, alors $C(n) = O(n^\alpha \log_b n)$*

a) Cas particulier courant

La plupart du temps, $b = 2$ donc la relation de récurrence est du type

$$C(n) = aC(n/2) + O(n^\alpha)$$

Théorème 2 *Si $C(n) = 2^\beta C(n/2) + O(n^\alpha)$, alors*

- ▷ *si $\alpha > \beta$, $C(n) = O(n^\alpha)$*
- ▷ *si $\alpha < \beta$, $C(n) = O(n^\beta)$*
- ▷ *si $\alpha = \beta$, $C(n) = O(n^\alpha \log n)$*

2 Exemples

2.1 Minimum d'un tableau

a) Première méthode brutale

Si t est un tableau de longueur n, on le coupe en deux sous-tableaux

$$u = t[0 \ldots (n/2) - 1], v = t[(n/2) \ldots (n-1)]$$

de longueurs $n/2$ et $n-(n/2)=(n+1)/2$

De manière évidente, on a

$$\min(t)=\min(\min(u),\min(v))$$

D'après le théorème précédent, comme $C(n)=2C(n/2)+O(n)$, alors $C(n)=O(n\log n)$

C'est pire que la méthode classique de parcours du tableau par une boucle (algorithme de complexité linéaire),

b) Deuxième méthode plus subtile

Si t est un tableau de longueur n, alors on note $m(i,j)=\min\limits_{i\leqslant k<j} t[k]$

On a alors $\min(t)=m(0,n)$.

De plus, on vérifie aisément que si $h=(i+j)/2$, alors

$$m(i,j)=\min(m(i,h),m(h,j))$$

D'après le théorème précédent, comme $C(n)=2C(n/2)+O(1)$, alors $C(n)=O(n)$... pas mieux que la méthode classique.

```
let mini t =
  let rec mini_aux i j =
    if j = i + 1 then t.(i)
    else let h = (i + j) / 2 in
      min (mini_aux i h) (min_aux h j)
  in
  mini_aux 0 (vect_length t);;
```

2.2 Dichotomie

a) Généralité

Soit $a<b$ deux entiers, $P(n)$ une proposition dépendant d'un entier n telle que $P(a)$ fausse et $P(b)$ vraie.

On recherche un entier n tel que $a\leqslant n\leqslant b$ tel que $P(n)$ fausse et $P(n+1)$ vraie.

L'idée générale est la suivante :

- si $b=a+1$, alors a convient (cas de base)
- sinon on coupe en deux en $c=(a+b)/2$ et on remplace l'intervalle a,b par a,c ou c,b :
  - si $P(c)$ est vraie, appel récursif avec arguments a,c
  - si $P(c)$ est fausse, appel récursif avec arguments c,b

En ce qui concerne la complexité, en notant $m=b-a$, on a

$$C(m)=C(m/2)+O(m^\alpha)$$

où $O(m^\alpha)$ est le coût du calcul de la valeur booléenne de $P(c)$

Un cas est particulièrement intéressant : si $\alpha=0$, alors $C(m)=O(\log m)$.

b) Racine carrée entière

```
let rac n =
  let rec rac_aux a b =
    if b = a + 1 then a
    else let c = (a + b) / 2 in
      if c * c < n then rac_aux c b
      else rac_aux a c
  in
  rac_aux 0 (n+1);;
```

c) Division euclidienne

```
let div a b =
  let rec div_aux x y =
  if y = x + 1 then (x, a - b * x)
  else let z = (x + y) / 2 in
```

```
    if b * z <= a then div_aux z y
    else div_aux x z
  in
  div_aux 0 a;;
```

d) Recherche dichotomique

Soit t un tableau trié par ordre croissant et x un objet. On recherche s'il existe un indice i tel que $t[i] = x$.

```
let recherche t x =
  let n = vect_length t in
  let rec rech_aux a b =
    if b = a + 1 then begin
        if t.(a) = x then a
        else if t.(b) = x then b
        else (-1) end
    else let c = (a + b) / 2 in
      if t.(c) = x then c
      else if t.(c) > x then rech_aux a c
      else rech_aux c b
    in rech_aux 0 n;;
```

2.3 Multiplication et exponentiation rapide

a) Multiplication récursive

```
let rec multi a b =
  if a > b then multi b a
  else
  if a = 0 then 0
  else let c = a / 2 in
    let r = multi c b in
    r + r + (if (a mod 2 = 0) then 0 else b);;
```

et en version récursive terminale

```
let multi a b =
  let rec multi_aux x y r =
  if x > y then multi_aux y x r
  else
  if x = 0 then r
  else let z = x / 2 in
    let s = r + (if (x mod 2) then 0 else y) in
    multi_aux z (y + y) s
  in
  multi_aux a b 0;;
```

b) Exponentiation récursive

```
let rec expo b a = (* b^a *)
  if a = 0 then 1
  else let c = a / 2 in
    let r = expo b c in
    r * r * (if (a mod 2 = 0) then 1 else b);;
```

et en version récursive terminale

```
let expo b a = (* b^a *)
  let rec expo_aux y x r =
  if x = 0 then r
  else let z = x / 2 in
    let s = r * (if (x mod 2) then 1 else y) in
```

```
    expo_aux (y * y) z s
  in
  expo_aux b a 1;;
```

2.4 Produit de deux polynômes (Karatsuba)

P, Q deux polynômes, $n = \max(\deg P, \deg Q)$. On partage les deux polynômes représentés par des tableaux ou des listes en leurs « milieux »

$$P = P_1 + X^{n/2}P_2, \quad Q = Q_1 + X^{n/2}Q_2$$

a) Calcul classique

Alors si on calcule normalement, on a

$$PQ = (P_1Q_1) + X^{n/2}(P_1Q_2 + P_2Q_1) + X^{2\times n/2}(P_2Q_2)$$

Les opérations coûteuses là-dedans sont les multiplications (on néglige les additions qui sont de complexité linéaire).
D'après le théorème précédent, comme $C(n) = 4C(n/2) + O(n)$, alors $C(n) = O(n^2)$.

Ce n'est pas mieux que la méthode classique de calcul des coefficients : $c_k = \sum_{i=0}^{k} a_i b_{k-i}$.

b) Amélioration imaginée par Karatsuba

Soit $R_1 = P_1Q_1$, $R_3 = P_2Q_2$ et $R_2 = (P_1 + P_2)(Q_1 + Q_2)$. Alors

$$PQ = R_1 + X^{n/2}(R_3 - R_1 - R_2) + X^{2\times n/2}R_3$$

D'après le théorème précédent, comme $C(n) = 3C(n/2) + O(n)$, alors $C(n) = O(n^{\log 3}) = O(n^{1,59})$.
C'est mieux que la méthode classique. En pratique, ce n'est intéressant que si n est grand (pour des valeurs trop faibles de n, on passe plus de temps à découper qu'à multiplier).

2.5 Produit de deux matrices (Strassen)

A, B deux matrices carrées d'ordre n :

$$A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix}, \quad B = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix}$$

a) Calcul classique

$$AB = \begin{pmatrix} A_1B_1 + A_2B_3 & A_1B_2 + A_2B_4 \\ A_3B_1 + A_4B_3 & A_3B_2 + A_4B_4 \end{pmatrix}$$

D'après le théorème précédent, comme $C(n) = 8C(n/2) + O(n^2)$, alors $C(n) = O(n^3)$.

Ce n'est pas mieux que la méthode classique de calcul des coefficients : $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$.

b) Amélioration imaginée par Strassen

On définit les produits de matrices suivants :

$M_1 = (A_1 + A_4)(B_1 + B_4)$ $\quad M_2 = (A_3 + A_4)B_1$
$M_3 = A_1(B_2 - B_4)$ $\quad M_4 = A_4(B_3 - B_1)$
$M_5 = (A_1 + A_2)B_4$ $\quad M_6 = (A_3 - A_1)(B_1 + B_2)$
$M_7 = (A_2 - A_4)(B_3 + B_4)$

Alors $AB = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{pmatrix}$

D'après le théorème précédent, comme $C(n) = 7C(n/2) + O(n^2)$, alors $C(n) = O(n^{\log 7}) = O(n^{2,81})$.