

Algorithmique M1

2012--2013

jeudi 20 septembre 2012

1

objectifs

Apprendre à **manipuler** les algorithmes.

- concevoir
- analyser
- chercher dans la littérature
- comprendre
- modifier

jeudi 20 septembre 2012

3

plan

- Diviser pour régner
- Gloutons
- Programmation dynamique
- Backtracking
- Algorithmes de flots
- Complexité amortie (Union-Find)

jeudi 20 septembre 2012

2

Fonctionnement du cours

Cours et TD

Cours: jeudi 14h30 → 16h30

TD1: [F. de Montgolfier](#), mercredi 8h30→10h30; 247E

TD2: [S. Laplante](#), mercredi 10h30→12h30; 470E puis **236**

francoisl@liafa.univ-paris-diderot.fr

<http://www.liafa.univ-paris-diderot.fr/~francoisl/m1algo.html>

jeudi 20 septembre 2012

4

Fontionnement du cours

Cours et TD

et : «exprime une addition, une liaison, un rapprochement...»
(le Petit Robert)

Cours: jeudi 14h30 → 16h30

TD1: F. de Montgolfier, mercredi 8h30→10h30; 247E

TD2: S. Laplante, mercredi 10h30→12h30; 470E puis 236

francoisl@liafa.univ-paris-diderot.fr

<http://www.liafa.univ-paris-diderot.fr/~francoisl/m1algo.html>

jeudi 20 septembre 2012

4

Contrôle des connaissances

Un examen + du **contrôle continu**

CC = (1 devoir «maison» +1 partiel)

∨

(2 devoirs «maison»)

jeudi 20 septembre 2012

5

Fontionnement du cours

Cours ~~et~~ TD
+

et : «exprime une addition, une liaison, un rapprochement...»
(le Petit Robert)

Cours: jeudi 14h30 → 16h30

TD1: F. de Montgolfier, mercredi 8h30→10h30; 247E

TD2: S. Laplante, mercredi 10h30→12h30; 470E puis 236

francoisl@liafa.univ-paris-diderot.fr

<http://www.liafa.univ-paris-diderot.fr/~francoisl/m1algo.html>

jeudi 20 septembre 2012

4

Voir J. Bentley
«Pearls of programming»

Exemple

Un problème:

Etant donné un tableau de nombres (positifs ou négatifs) de taille n, calculer la **somme maximale** des éléments **d'un sous-tableau**.

jeudi 20 septembre 2012

6

Exemple

Un problème:

Etant donné un tableau de nombres (positifs ou négatifs) de taille n, calculer la **somme maximale** des éléments d'un sous-tableau.

suite de cases consécutives

Exemple

Un problème:

Etant donné un tableau de nombres (positifs ou négatifs) de taille n, calculer la **somme maximale** des éléments d'un sous-tableau.

suite de cases consécutives

8	-10	10	4	-19	40	0	5	-9	14	2	3	78	7	-24	6	9	-18	7	2
---	-----	----	---	-----	----	---	---	----	----	---	---	----	---	-----	---	---	-----	---	---

somme de tous les éléments = 115

Exemple

Un problème:

Etant donné un tableau de nombres (positifs ou négatifs) de taille n, calculer la **somme maximale** des éléments d'un sous-tableau.

suite de cases consécutives

8	-10	10	4	-19	40	0	5	-9	14	2	3	78	7	-24	6	9	-18	7	2
---	-----	----	---	-----	----	---	---	----	----	---	---	----	---	-----	---	---	-----	---	---

Exemple

Un problème:

Etant donné un tableau de nombres (positifs ou négatifs) de taille n, calculer la **somme maximale** des éléments d'un sous-tableau.

8	-10	10	4	-19	40	0	5	-9	14	2	3	78	7	-24	6	9	-18	7	2
---	-----	----	---	-----	----	---	---	----	----	---	---	----	---	-----	---	---	-----	---	---

somme max:= 140

somme de tous les éléments = 115

Les cas simples...

Les cas simples...

8	0	10	4	19	40	0	5	9	14	2	3	78	7	2	6	9	8	7	2
---	---	----	---	----	----	---	---	---	----	---	---	----	---	---	---	---	---	---	---

solution ? la somme totale... 223

Les cas simples...

8	0	10	4	19	40	0	5	9	14	2	3	78	7	2	6	9	8	7	2
---	---	----	---	----	----	---	---	---	----	---	---	----	---	---	---	---	---	---	---

solution ?

Les cas simples...

8	0	10	4	19	40	0	5	9	14	2	3	78	7	2	6	9	8	7	2
---	---	----	---	----	----	---	---	---	----	---	---	----	---	---	---	---	---	---	---

solution ? la somme totale... 223

-8	-10	-10	-4	-19	-40	-10	-5	-9	-14	-2	-3	-78	-7	-24	-6	-9	-18	-7	-2
----	-----	-----	----	-----	-----	-----	----	----	-----	----	----	-----	----	-----	----	----	-----	----	----

solution ?

Les cas simples...

8	0	10	4	19	40	0	5	9	14	2	3	78	7	2	6	9	8	7	2
---	---	----	---	----	----	---	---	---	----	---	---	----	---	---	---	---	---	---	---

solution ? la somme totale... 223

-8	-10	-10	-4	-19	-40	-10	-5	-9	-14	-2	-3	-78	-7	-24	-6	-9	-18	-7	-2
----	-----	-----	----	-----	-----	-----	----	----	-----	----	----	-----	----	-----	----	----	-----	----	----

solution ? 0 ! (i.e. un sous-tableau de taille 0)

Le problème

Donnée: un tableau $T[0..n-1]$

Résultat: $\text{Max} \{ \text{sum}[i,j] \mid 0 \leq i,j \leq n-1 \}$

$$\text{sum}[i,j] = \sum_{k \in [i,j]} T[k]:$$

intervalle: $i, i+1, \dots, j$

Le problème

Donnée: un tableau $T[0..n-1]$

Résultat: $\text{Max} \{ \text{sum}[i,j] \mid 0 \leq i,j \leq n-1 \}$

$$\text{sum}[i,j] = \sum_{k \in [i,j]} T[k]:$$

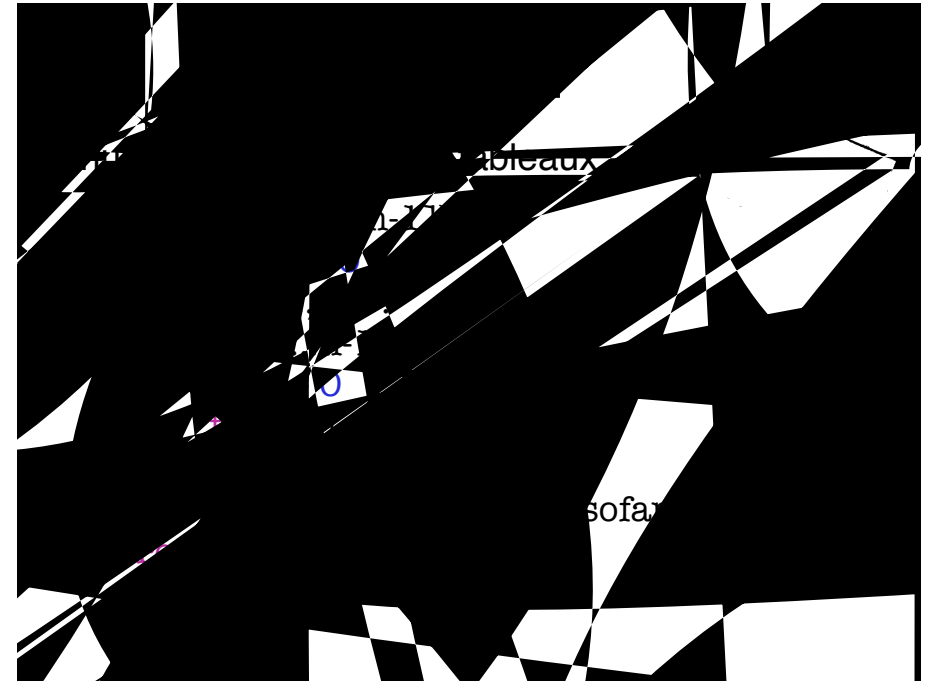
Algo 1

Enumérer tous les sous-tableaux...

Algo 1

Enumérer tous les sous-tableaux...

```
def algo1(T[0...n-1]):  
    maxsofar = 0  
    for i = 0,...,n-1:  
        for j = i...n-1:  
            sum = 0  
            for k = i...j:  
                sum += T[k]  
            maxsofar = max(maxsofar,sum)  
    return maxsofar
```



Algo 1

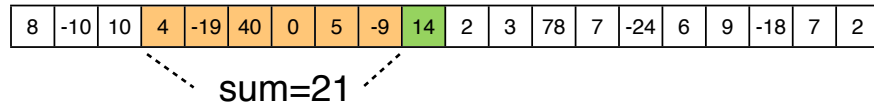
Enumérer tous les sous-tableaux...

```
def algo1(T[0...n-1]):  
    maxsofar = 0  
    for i = 0,...,n-1:  
        for j = i...n-1:  
            sum = 0  
            for k = i...j:  
                sum += T[k]  
            maxsofar = max(maxsofar,sum)  
    return maxsofar
```

Max { sum[i,j] | 0 ≤ i,j ≤ n-1 }
sum[i,j] = $\sum_{k \in [i,j]} T[k]$

Algo 2

idée: beaucoup de calculs inutiles dans algo1...

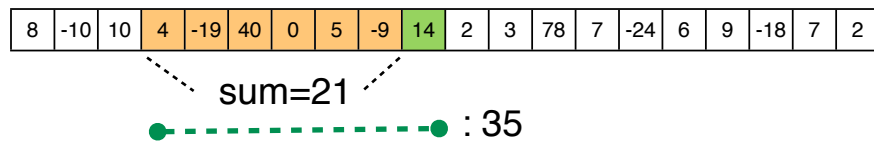


Algo 2

```
def algo2(T[0...n-1]) :
    maxsofar = 0
    for i = 0,...,n-1 :
        sum = 0
        for j = i,...,n-1 :
            sum += T[j]
            maxsofar = max(maxsofar,sum)
    return maxsofar
```

Algo 2

idée: beaucoup de calculs inutiles dans algo1...



Algo 2

```
def algo2(T[0...n-1]) :
    maxsofar = 0
    for i = 0,...,n-1 :
        sum = 0
        for j = i,...,n-1 :
            sum += T[j]
            maxsofar = max(maxsofar,sum)
    return maxsofar
```

itération i:

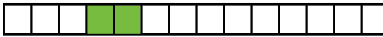
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

i-1

Algo 2

```
def algo2(T[0...n-1]) :  
    maxsofar = 0  
    for i = 0,...,n-1 :  
        sum = 0  
        for j = i,...,n-1 :  
            sum += T[j]  
            maxsofar = max(maxsofar,sum)  
    return maxsofar
```

itération i:




i-1

Algo 2

```
def algo2(T[0...n-1]) :  
    maxsofar = 0  
    for i = 0,...,n-1 :  
        sum = 0  
        for j = i,...,n-1 :  
            sum += T[j]  
            maxsofar = max(maxsofar,sum)  
    return maxsofar
```

itération i:

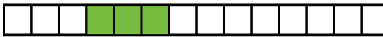


i-1

Algo 2

```
def algo2(T[0...n-1]) :  
    maxsofar = 0  
    for i = 0,...,n-1 :  
        sum = 0  
        for j = i,...,n-1 :  
            sum += T[j]  
            maxsofar = max(maxsofar,sum)  
    return maxsofar
```

itération i:




i-1

Algo 2

```
def algo2(T[0...n-1]) :  
    maxsofar = 0  
    for i = 0,...,n-1 :  
        sum = 0  
        for j = i,...,n-1 :  
            sum += T[j]  
            maxsofar = max(maxsofar,sum)  
    return maxsofar
```

itération i:




i-1

Algo 2

```
def algo2(T[0...n-1]) :
    maxsofar = 0
    for i = 0,...,n-1 :
        sum = 0
        for j = i,...,n-1 :
            sum += T[j]
            maxsofar = max(maxsofar,sum)
    return maxsofar
```

itération i:




i-1

Algo 2

```
def algo2(T[0...n-1]) :
    maxsofar = 0
    for i = 0,...,n-1 :
        sum = 0
        for j = i,...,n-1 :
            sum += T[j]
            maxsofar = max(maxsofar,sum)
    return maxsofar
```

itération i:




i-1

Algo 2

```
def algo2(T[0...n-1]) :
    maxsofar = 0
    for i = 0,...,n-1 :
        sum = 0
        for j = i,...,n-1 :
            sum += T[j]
            maxsofar = max(maxsofar,sum)
    return maxsofar
```

itération i:

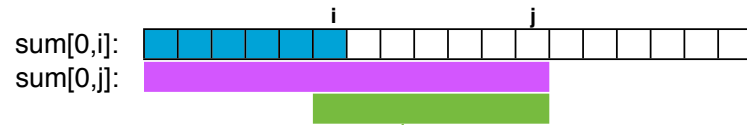


i-1

calcul des sous-tableaux
commençant en i-1

Algo 3

Autre idée:



$$\text{sum}[i,j] = \text{sum}[0,j] - \text{sum}[0,i-1]$$

Algo 3

```
def algo3(T[0..n-1]) :
    sum[i] = 0    ∀ i = -1,0,...,n
    for i = 0..n-1 :
        sum[i] = sum[i-1]+T[i]
    maxsofar = 0
    for i = 0..n-1 :
        for j = i..n-1 :
            sumij = sum[j] - sum[i-1]
            maxsofar = max(maxsofar,sumij)
    return maxsofar
```

ici sum[i] = «sum[0,i]»

Algo 3

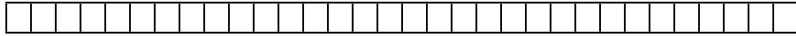
```
def algo3(T[0..n-1]) :
    sum[i] = 0    ∀ i = -1,0,...,n
    for i = 0..n-1 :
        sum[i] = sum[i-1]+T[i]
    maxsofar = 0
    for i = 0..n-1 :
        for j = i..n-1 :
            sumij = sum[j] - sum[i-1]
            maxsofar = max(maxsofar,sumij)
    return maxsofar
```

Algo 3

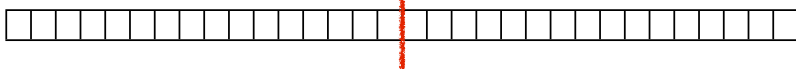
```
def algo3(T[0..n-1]) :
```



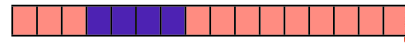
Algo 4



Algo 4



Algo



sous-tab max
m

k



sous-tab max
m

Algo 4



Algo 4

```
def algo4(T,l,u) :  
    if (l>u) : return 0  
    if (l==u) : return max(0,T[l])  
    m = (l+u)/2  
  
    lmax = sum = 0  
    for i = m..l :  
        sum += T[i]  
        lmax = max(lmax,sum)  
    rmax = sum = 0  
    for i = m+1..u :  
        sum += T[i]  
        rmax = max(rmax,sum)  
  
    return max(lmax+rmax, algo4(T,l,m),algo4(T,m+1,u))
```

Algo 4






```
def algo4(T,l,u) :  
    if (l>u) : return 0  
    if (l==u) : return max(0,T[l])  
    m = (l+u)/2  
  
    lmax = sum = 0  
    for i = m..l :  
        sum += T[i]  
        lmax = max(lmax,sum)  
    rmax = sum = 0  
    for i = m+1..u :  
        sum += T[i]  
        rmax = max(rmax,sum)  
  
    return max(lmax+rmax, algo4(T,l,m),algo4(T,m+1,u))
```

Algo 5

Idée: on parcourt le tableau de gauche à droite en gardant:

- le sous-tableau max rencontré dans la partie gauche parcourue
- le sous-tableau «suffixe» max se terminant à la position courante.



Mise à jour: comparer  +  et 
(si  +  positif)

Algo 5

```
def algo5(T[0..n-1]) :
    maxsofar = 0
    maxendinghere = 0

    for i = 0..n-1 :
        maxendinghere = max(maxendinghere + T[i], 0)
        maxsofar = max(maxsofar, maxendinghere)

    return maxsofar
```

Complexité
en $O(n)$

Algo 5

```
def algo5(T[0..n-1]) :
    maxsofar = 0
    maxendinghere = 0

    for i = 0..n-1 :
        maxendinghere = max(maxendinghere + T[i], 0)
        maxsofar = max(maxsofar, maxendinghere)

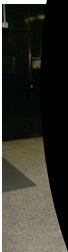
    return maxsofar
```

Bilan

Algo 1	Algo 2 Algo 3	Algo 4	Algo 5
$O(n^3)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n)$
naïf...		diviser-pour-régner	scan

Bilan

Algo 1	Algo 2 Algo 3	Algo 4	Algo 5
$O(n^3)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n)$
naïf...		diviser-pour-régner	scan



1

2

3



4

5

6

7

$n^3, n^2, n \cdot \quad n, n \dots$

Conclusion

⇒ Il y a une vraie différence entre $O(n^3)$, $O(n^2)$, et $O(n)$!

⇒ Il y en a encore plus entre $O(n^k)$ et $O(2^n), \dots$!

Rechercher de meilleurs algorithmes est important.

Un problème peut s'attaquer de plusieurs manières:
les idées sous-jacentes aux algorithmes sont très différentes !

(Higelin a tort... *ce n'est pas toujours la première idée qui est la bonne !*)

$n^3, n^2, n \cdot \quad n, n \dots$

Problèmes traitables ?

⇒ algorithmes en temps polynomial