

Algorithmique M1

2015--2016

F. Laroussinie

1

plan

- Diviser pour régner
- Gloutons
- Programmation dynamique
- Backtracking
- Recherche de motifs

2

objectifs

Apprendre à manipuler les algorithmes.

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

3

Fonctionnement du cours

Cours et TD
+

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

Cours: mercredi 10h30 → 12h30

TD1: Fabien de Montgolfier, lundi 13h30→15h30;

TD2: Dominique Poulhalon, lundi 15h30→17h30;

francoisl@liafa.univ-paris-diderot.fr

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

4

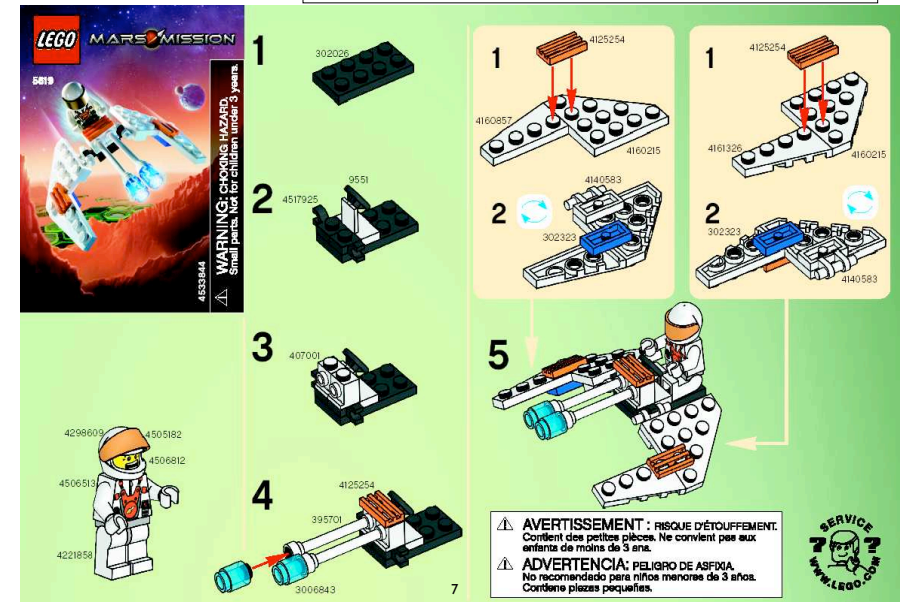
Contrôle des connaissances

Un examen + du contrôle continu

CC = (1 TD noté +1 DM/projet)

5

Exemple d'algorithme



Exemple d'algorithme

les données

Résultat

484 Émincé de rouget au pistou
 24 heures à l'avance
 Préparation : 40 mn - Cuisson : 30 à 35 mn

Écailler les rougets, les laver, puis soulever délicatement les filets et enlever les arêtes qui restent avec une pince à épiler. Mettre les filets dans un plat creux ; arroser avec de l'huile d'olive ; saupoudrer d'herbes de Provence et laisser mariner au frais, pendant 24 heures, le tout couvert par un torchon.

Cuire le fenouil à l'eau bouillante salée, citronnée et parfumée à l'huile d'olive et avec une brindille de thym (15 à 20 mn). Égoutter.

Cuire les pâtes à l'eau bouillante salée, huilée, pendant 12 mn. Égoutter. Rafrâchir. Tenir au chaud.

Pendant ces cuissons préparer le coulis de tomates (41). Assaisonner le fenouil coupé en tranches avec la vinaigrette. Tenir au chaud. Assaisonner les pâtes avec le basilic (à volonté) haché et un mélange d'huile d'olive et de vinaigre de xérès. Disposer les pâtes sur le plat, recouvrir avec la fondue de fenouil.

Rapidement, cuire à la poêle Tefal, à feu très vif, les filets marinés pour qu'ils deviennent dorés et croustillants. Saler. Poivrer et disposer en étoile les filets de rougets, sur le plat de légumes. Servir avec le coulis de tomates en sauce.

1 kg 500 rougets.
400 g pâtes.
500 g fenouil.
2 dl huile olive.
0 dl 5 vinaigre xérès.
Coulis de tomates.
1 citron.
Basilic.
Herbes de Provence.
Thym.
Vinaigrette.
Sel.
Poivre.

C'est parti !

6

8

Un algorithme est une procédure (une “recette”) pour obtenir un résultat en un nombre fini d’opérations.

Un algorithme résout un *problème*.

9

problème / instance

Un problème:

- **Données:** une liste de mots
Résultat: la liste triée dans l’ordre alphabétique

Des instances de ce problème:

- ▶ table, vélo, chaise, pain, chocolat, tram, ciel, pomme
- ▶ Noémie, Pierre, Alain, Célia, Juan, Leila, John
- ▶ pomme, oiseau, vélo, fgsdhgf, hjdshq, sqdkjhd
- ▶
- ▶ bip, bop, bip, tip

11

Les problèmes algorithmiques

- **Données:** deux nombres a et b
Résultat: $a+3b+ab$
- **Données:** un nombre a
Résultat: la somme $1+2+3+\dots+a$
- **Données:** un ensemble de pièces de monnaie, une valeur S
Résultat: un ensemble de pièces de somme S
- **Données:** une liste de mots
Résultat: la liste triée dans l’ordre alphabétique
- **Données:** deux textes
Résultat: la liste des mots communs aux deux textes
- **Données:** une carte, deux villes A et B
Résultat: un chemin le plus court entre A et B
- **Données:** un programme P , une donnée a
Résultat: réponse à “est-il vrai que le résultat de P appliqué à a vaut $8a+5$ ”?

10

problème / instance

Un problème:

- **Données:** une liste de mots
Résultat: la liste triée dans l’ordre alphabétique

Des **Un algorithme doit résoudre toutes les instances d’un problème**

- ▶ table, vélo, chaise, pain, chocolat, tram, ciel, pomme
- ▶ Noémie, Pierre, Alain, Célia, Juan, Leila, John
- ▶ pomme, oiseau, vélo, fgsdhgf, hjdshq, sqdkjhd
- ▶
- ▶ bip, bop, bip, tip

11

Problèmes et algorithmes

Pour un problème, il y a parfois plusieurs algorithmes très différents...

Bien sûr, on cherche des algorithmes
CORRECTS et EFFICACES !

12

Algorithmes *corrects* : y en-a-t-il ?

Algorithmes *efficaces* : notions de complexité
(pire cas, en moyenne, amortie...)

Algorithmes *optimaux* : peut-on faire mieux ?

13

Problèmes et algorithmes

Pour un problème, il y a parfois plusieurs algorithmes très différents...

Ce n'est pas toujours possible !

Bien sûr, on cherche des algorithmes
CORRECTS et EFFICACES !

12

Notions de complexité

Pire cas, en moyenne, amortie...

$C_A(x)$: nombre d'opérations élémentaires nécessaires pour
l'exécution de l'algorithme A sur la donnée x .

Complexité dans le pire cas:

$$C_A(n) \stackrel{\text{def}}{=} \max_{x, |x|=n} C_A(x)$$

distribution de probabilités
sur les données de taille n

Complexité en moyenne:

$$C_A^{\text{moy}}(n) \stackrel{\text{def}}{=} \sum_{x, |x|=n} p(x) \cdot C_A(x)$$

Complexité amortie:

Evaluation du coût cumulé de n opérations (dans le pire cas).

14

Notions de complexité

Obj: avoir un **ordre de grandeur** du nombre d'opérations...

Notations: $O()$, $\Omega()$ et $\Theta()$:

$O(g(n)) = \{f(n) \mid \exists c > 0, n_0 \geq 0 \text{ tq } 0 \leq f(n) \leq c \cdot g(n) \forall n \geq n_0\}$
 -> ensemble des fonctions **majorées** par $c \cdot g(n)$

$\Omega(g(n)) = \{f(n) \mid \exists c > 0, n_0 \geq 0 \text{ tq } 0 \leq c \cdot g(n) \leq f(n) \forall n \geq n_0\}$
 -> ensemble des fonctions **minorées** par $c \cdot g(n)$

$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 > 0, n_0 \geq 0 \text{ tq } 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \forall n \geq n_0\}$
 -> ensemble des fonctions **encadrées** par $c_1 \cdot g(n)$ et $c_2 \cdot g(n)$

15

Notions de complexité

fct \ n	10	50	100	300	1000
$5n$	50	250	500	1500	5000
$n \cdot \log_2(n)$	33	282	665	2469	9966
n^2	100	2500	10000	90000	$10^6(7c)$
n^3	1000	125000	$10^6(7c)$	$27 \cdot 10^6(8c)$	$10^9(10c)$
2^n	1024	... (16c)	... (31c)	... (91c)	... (302c)
$n!$	$3.6 \cdot 10^6(7c)$... (65c)	... (161c)	... (623c)	... !!!
n^n	$10 \cdot 10^9(11c)$... (85c)	... (201c)	... (744c)	... !!!!

notation: (Xc) -> "s'écrit avec X chiffres en base 10"

17

Notions de complexité

On s'intéresse à des grandes familles de fonctions:

- les algorithmes **sous-linéaires**.
Par ex. en $O(\log n)$
- les algorithmes **linéaires**: $O(n)$
ou "**quasi-linéaires**" comme $O(n \cdot \log n)$
- les algorithmes **polynomiaux** $O(n^k)$
- les algorithmes **exponentiels**: $O(2^n)$
- ...

16

Voir J. Bentley
«Pearls of programming»

Exemple

suite de cases **consécutives**

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

18

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)

19

Algo 1

Enumérer tous les sous-tableaux...

Complexité
en $O(n^3)$

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] | $0 \leq i,j \leq n-1$ }

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

21

Le problème

Donnée: un tableau T[0..n-1]

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

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

intervalle: i,i+1,...,j

20

Algo 2

idée: beaucoup de calculs inutiles dans algo1...

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

sum=21 : 35

22

Algo 2

Complexité
en $O(n^2)$

```
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:



calcul de tous les sous-tableaux
commençant en i-1

23

Algo 3

ici $\text{sum}[i] = \text{«sum}[0,i]\text{»}$

```
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
```

Complexité
en $O(n^2)$

25

Algo 3

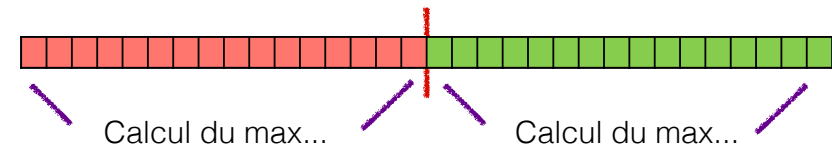
Autre idée:



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

24

Algo 4

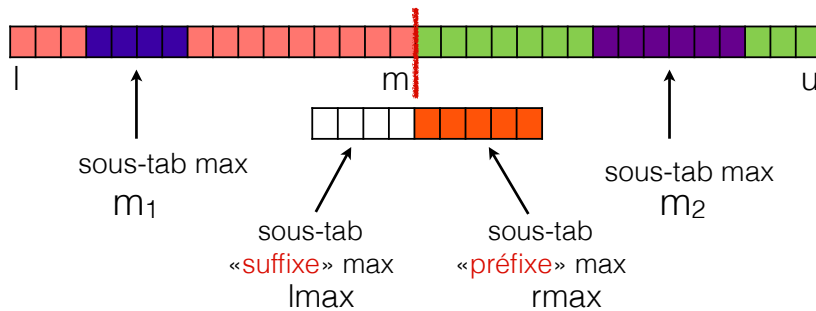


+ «traitement»

Résultat pour T

26

Algo 4



résultat = $\max(m_1, m_2, l_{\max} + r_{\max})$

27

Algo 4

```
def algo4(T,l,u) :
    if (l>u) : return 0
    if (l==u) : return max(0,T[l])
    m = (l+u)/2

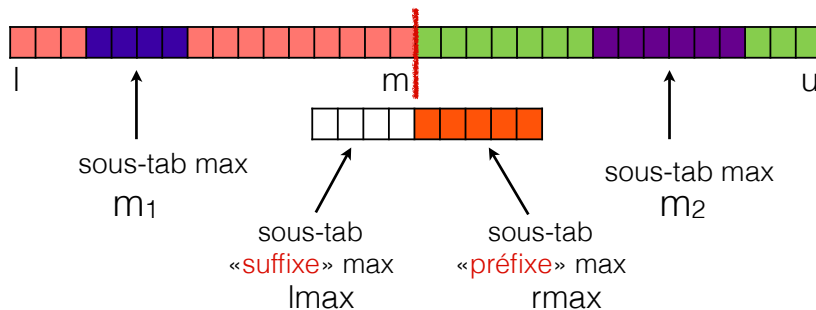
    lmax = sum = 0
    for i = m ... l : // l ≤ m
        sum += T[i]
        lmax = max(lmax,sum)
    rmax = sum = 0
    for i = m+1 ... u : // m ≤ u
        sum += T[i]
        rmax = max(rmax,sum)

    return max(lmax+rmax, algo4(T,l,m),algo4(T,m+1,u))
```

Complexité
en $O(n \cdot \log n)$

28

Algo 4



résultat = $\max(m_1, m_2, l_{\max} + r_{\max})$

Calcul de l_{\max} : tester tous les sous-tab finissant en m.
Calcul de r_{\max} : tester tous les sous-tab commençant en m+1.

27






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:

- 1) comparer  +  et 0 → 
- 2) comparer  et 

29

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 $O(n)$

30

J. Bentley «Pearls of programming», Addison-Wesley (1986)

TABLE I. Summary of the Algorithms

Algorithm	1	2	3 4	4 5
Lines of C Code	8	7	14	7
Run time in microseconds	$3.4N^3$	$13N^2$	$46N \log N$	$33N$
Time to solve problem of size	10^2 10^3 10^4 10^5 10^6	3.4 secs .94 hrs 22 mins 1.5 days 5 mos	130 msec 13 secs 6.1 secs 1.3 min 15 min	3.3 msec 33 msec .33 secs 3.3 secs 33 secs
Max problem solved in one	sec min hr day	67 260 1000 3000	280 2200 17,000 81,000	2000 82,000 3,500,000 73,000,000
If N multiplies by 10, time multiplies by	1000	100	10+	10
If time multiplies by 10, N multiplies by	2.15	3.16	10-	10

1984... machine: VAX-11/750 ³²

Bilan

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

Y a-t-il une différence en pratique ?

31



CRAY-1 vs TR



TABLE II. The Tyranny of Asymptotics

N	Cray-1, FORTRAN, Cubic Algorithm	TRS-80, BASIC, Linear Algorithm
10	3.0 microsecs	200 millisecs
100	3.0 millisecs	2.0 secs
1000	3.0 secs	20 secs
10,000	49 mins	3.2 mins
100,000	35 days	32 mins
1,000,000	95 yrs	5.4 hrs

J. Bentley «Pearls of programming», Addison-Wesley (1986)

33

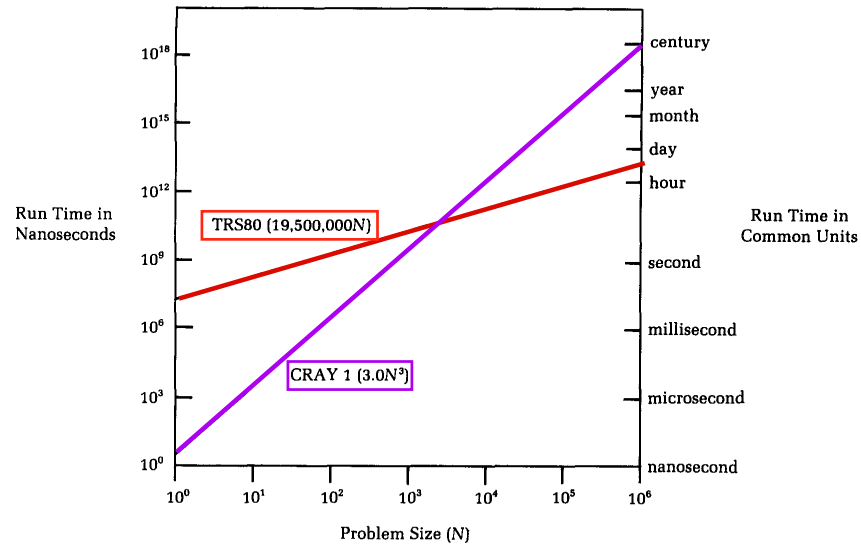


FIGURE 1. The Run Times of Two Programs

J. Bentley «Pearls of programming», Addison-Wesley (1986)

2012: Macbook Air, i7...

(en secondes)

	Algo 1	Algo 2	Algo 3	Algo 4	Algo 5
100	0,0213	0,0018	0,0023	0,0006	0,0001
500	2,0193	0,0423	0,0533	0,0031	0,0003
1 000	16,2401	0,1736	0,2239	0,007	0,0005
2 000	125,9673	0,7059	0,883	0,0154	0,0011
10 000		17,5162	21,31	0,0788	0,0052
30 000		158,801	192,8002	0,2664	0,0174
100 000				1,0107	0,0657
1 000 000				10,1928	0,5407

rappel 1984: 1h 22min 36 15min 33sec

Et aujourd'hui ?

Remarque n°1

La notion de complexité (ou de coût, d'efficacité...) d'un algorithme est robuste...

Remarque n°2

Il y a de fortes différences en pratique entre ces algorithmes !

Et pourtant... ce sont tous des algorithmes dits «*efficaces*» !

il y a beaucoup de problèmes pour lesquels

- ▶ il n'y a que des algorithmes très inefficaces... ou
- ▶ il n'y a même pas d'algorithme !

38

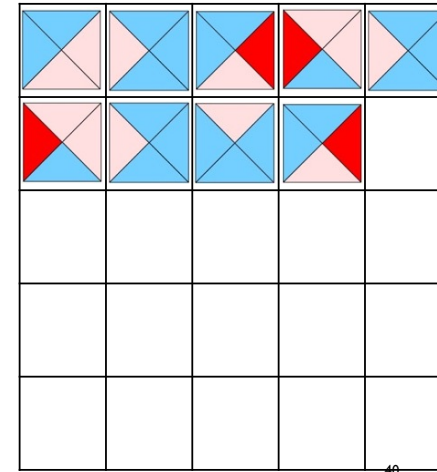
Des problèmes difficiles...

39

Un puzzle...

Une grille 5x5

25 dominos



...

Puzzle

On essaie toutes les possibilités ?

25 cartes à placer sur 25 cases:

- 25 possibilités pour la première,
- 24 pour la seconde,
- 23 pour la troisième,
- ...

25x24x23x...x2 possibilités !

Avec un ordinateur qui évalue 1 milliard de cas par seconde, il faudrait ...

490 millions d'années !

(25! s'écrit avec 26 chiffres...)

41

Questions/objections

Sur le problème du puzzle:

- ▶ les ordinateurs sont de plus en plus rapides...
- ▶ les informaticiens sont incompetents... et incapables de trouver un bon algorithme.
- ▶ nous n'avons pas montré qu'il n'existe pas d'algorithme efficace.
- ▶ le Puzzle est très très spécial... et sans intérêt !

ne suffit pas / non 😏 / pas encore / non !

42

Encore un exemple...

Imaginons un véhicule dont la consommation d'énergie est une fonction linéaire (n) ou exponentielle (2^n) de la distance à parcourir.

Si l'on peut faire 50 kilomètres avec un réservoir V , alors un réservoir 1000 fois plus gros, permettra de parcourir...

- 50000 km si la consommation est linéaire, et
- 60 km si la consommation est exponentielle !

44

Attendre un ordinateur plus rapide ?

En 10 ans, la vitesse des ordinateurs a été multipliée par 50...

Supposons qu'aujourd'hui, on puisse résoudre un problème de taille K en une heure...

Si l'algorithme a une complexité n , alors...

- un ordinateur 100 fois plus rapide, résoudra des pb de taille $100 \times K$.
- un ordinateur 1000 fois plus rapide, résoudra des pb de taille $1000 \times K$.

Si l'algorithme a une complexité n^2 , alors...

- un ordinateur 100 fois plus rapide, résoudra des pb de taille $10 \times K$.
- un ordinateur 1000 fois plus rapide, résoudra des pb de taille $32 \times K$.

Si l'algorithme a une complexité 2^n , alors...

- un ordinateur 100 fois plus rapide, résoudra des pb de taille $7 + K$.
- un ordinateur 1000 fois plus rapide, résoudra des pb de taille $10 + K$.

La course est perdue d'avance !

Le puzzle n'est pas isolé !

Beaucoup de problèmes lui ressemblent...

- ▶ Recherche de chemins hamiltoniens (qui passent une et une seule fois par chaque sommet) dans un graphe
- ▶ Problème du voyageur de commerce
- ▶ Planification (emploi du temps)
- ▶ Colorier une carte avec 3 couleurs
- ▶ (2: facile, 4: toujours possible)
- ▶ ...

45

Les problèmes NP-complets

Tous ces problèmes sont **aussi difficiles** les uns que les autres.

Si on a un algorithme efficace pour un, on peut l'adapter pour les autres.

Aujourd'hui, **on ne sait pas** si il existe des algorithmes efficaces pour tous ces problèmes !

Mais on **sait** que d'autres problèmes sont durs...

46

De “vrais” problèmes difficiles

Question: est-ce que le joueur 1 (ou 2) a une stratégie gagnante ?

Ce “problème” est exponentiel !

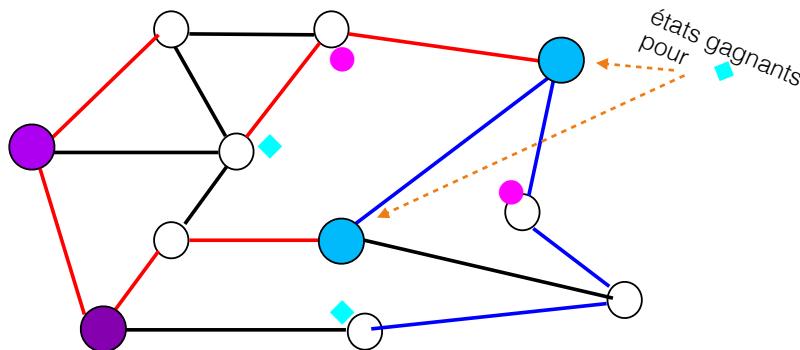
(Il existe un $K > 1$ tel que tout algorithme pour ce problème prend un temps qui augmente au moins aussi vite que K^n où n est le nombre d'intersections.)

Et au delà: doublement exponentiel (2^{2^n}), triplement...

48

De “vrais” problèmes difficiles

2 joueurs avec 2 pions chacun (◆ et ●) cherchent à atteindre des états gagnants en déplaçant leurs pions en suivant les couleurs et sans sauter les autres pions...



47

Toujours plus dur...

On a vu des problèmes que l'on peut résoudre efficacement...

Et d'autres, beaucoup plus difficiles.

Il y a même des problèmes **sans algorithme**.

49

Problème de correspondance de Post

Un ensemble de type de cartes:

abb	a	bab	baba	aba
bbab	aa	ab	aa	a

Peut-on écrire le même mot en haut et en bas ?

a	abb	abb	baba	abb	aba
aa	bbab	bbab	aa	bbab	a

Oui !

50

Problème de correspondance de Post

aa	abab	ba	b	?
a	aaaa	aaab	bab	

Oui !

Mais **la plus petite** solution utilise **781** cartes !!
(voir [PCP@home contest](#))

Le problème PCP est **indécidable**...

52

Problème de correspondance de Post

bab	bb	a
a	ba	abb

?

Il n'y a pas de solution !!

(sur aucune carte, les deux mots ne terminent par la même lettre !)

51

Problème de l'arrêt

Est-ce que ce programme termine ?

1. Tant que $x \neq 1$ Faire $x := x - 2$
2. Stop

Termine si x est impair.

53

Problème de l'arrêt

1. Tant que $x \neq 1$ Faire :
2. Si x est pair Alors $x := x/2$
3. Sinon $x := 3x + 1$
4. Stop

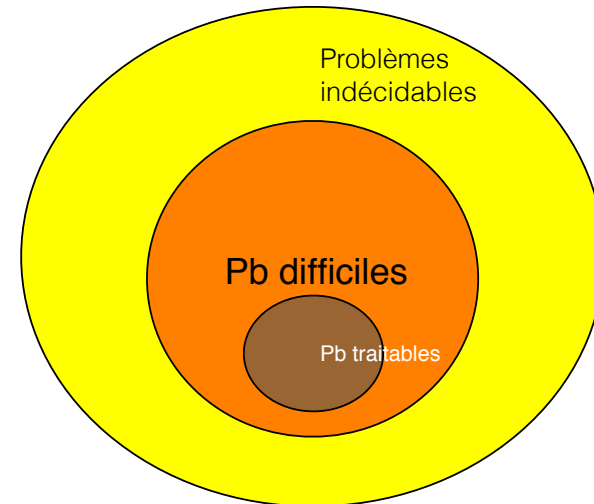
Essai avec 9:

9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

On ne sait pas si cela termine pour tout entier !!

54

Vue globale



56

Problème de l'arrêt

Donnée: un programme P , une entrée x

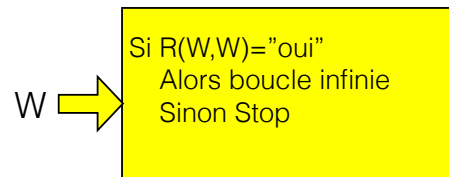
Question: est-ce $P(x)$ s'arrête ?

Ce problème est indécidable.

Soit R une procédure qui résout le problème.

($R(P,x)$ = "oui" ssi $P(x)$ s'arrête)

Soit P le programme suivant:



Que répond $P(P)$? R n'existe pas !!!

55

Classes de complexité

- Dans chaque groupe, il y a de très nombreuses **classes de complexité** contenant des problèmes "aussi difficiles les uns que les autres".
- Etant donné un problème, on cherche:
 - des algorithmes (!)
(leurs coûts donnent une borne supérieure de la complexité du problème)
 - sa complexité exacte
(ou une borne inférieure...)

57

Classes de complexité

- Dans chaque groupe, il y a de très nombreuses **classes de complexité** contenant des problèmes “aussi difficiles les uns que les autres”.

- Etant donné un problème, on cherche:

- des algorithmes (leurs coûts de la complexité du problème)
- sa complexité (ou une borne inférieure)

PUB !

**Allez au cours de
calculabilité et
complexité !**

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 $O(2^n)$,... !

Rechercher de meilleurs algorithmes est important.

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

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