UFR d'Informatique

Eléments d'algorithmique 1

EA3

2015--2016

F. Laroussinie

L'algorithmique en L2:

EA3: 1,5h de cours + 2h de TD (3 ects)

EA4: 2h de cours + 3h de TD/TP (6 ects)
avec Dominique Poulalhon

et dans tous les cours d'informatique !

plan

- Introduction
  (notions de base, exemples)
- Un peu de tri
- Liste chaînées et arbres
- Backtracking
- Un peu de graphes

objectifs

Apprendre à manipuler les algorithmes.

- lire et comprendre
- écrire
- modifier
- concevoir

Fonctionnement du cours

Cours ~~et~~ + TD

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

Cours: jeudi 15h → 16h30
TD1: Pierre Charbit, mardi de 13h45 à 15h45 en 071E;
TD2: Nicolas de Rugy-Altherre, mardi de 16h à 18h en 056 (bât. Condorcet)
TD3: Thibault Suzanne, jeudi de 16h45 à 18h45 en 476F
TD4: Antonio Bucciarelli, lundi de 10h30 à 12h30, en 480F

francoisl@liafa.univ-paris-diderot.fr

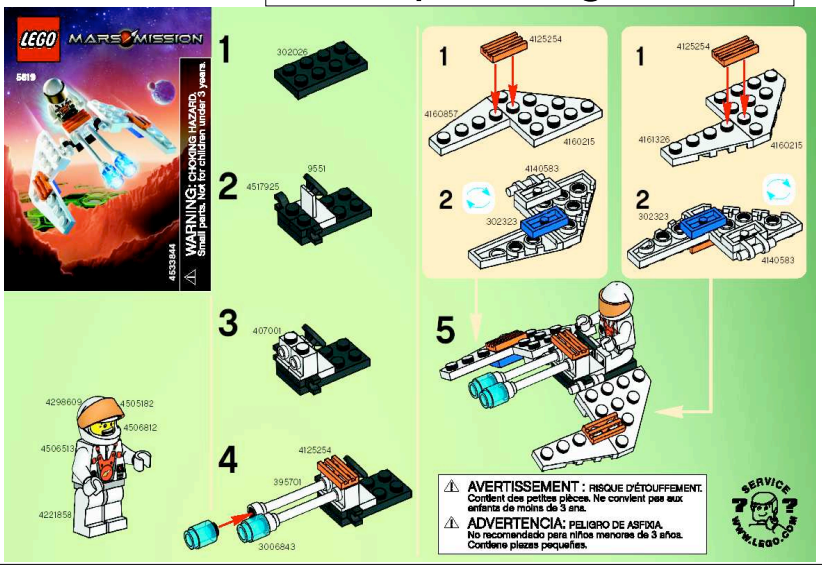
http://www.liafa.univ-paris-diderot.fr/~francoisl/l2algo.html
+ didel

Contrôle des connaissances

Un examen 60% + du contrôle continu (40%)

C'est parti !

Exemple d'algorithme



Exemple d'algorithme

les données

Résultat

484 Émincé de rouget au pistou

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.

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. Rafraîchir. 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 Téfal, 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 saucière.

Un algorithme est une procédure (une "recette") pour obtenir un résultat en un nombre fini d'opérations.

Un algorithme résout un *problème*.

Les problèmes algorithmiques

- Données: deux nombres *a* et *b*
  Résultat: *a+3b+ab*
- Données: un nombre *a*
  Résultat: la somme *1+2+3+...+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*" ?

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

Comment écrire des algorithmes ?

Exemple n° 1

L'addition en colonnes

Additionner deux nombres...

```
  1 1 1 1  1  1 1 1 1
  2314564766879 7654
+      378658243421325
-----------------------
  2352430591221 8979
```

La brique de base : additionner trois nombres de 1 chiffre.

L'addition en colonnes

Problème:

- données: deux nombres entiers a et b.
- résultat: l'entier correspondant à la somme a+b

1. on écrit a et b sur deux lignes en les **alignant à droite**.
2. pour chaque colonne c de **droite à gauche**:
   - 2.a. on ajoute les deux chiffres de la colonne c ainsi que la retenue déjà calculée (initialement à 0),
   - 2.b. on en déduit le chiffre à inscrire dans la colonne c du résultat, et la nouvelle valeur de la retenue.
3. si la retenue est 1, on la reporte dans une nouvelle colonne à gauche du nombre résultat.

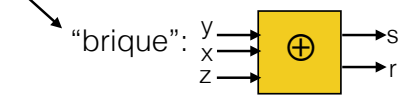
L'addition en colonnes

Problème:

- données: deux nombres entiers a et b à n chiffres
- résultat: l'entier correspondant à la somme a+b

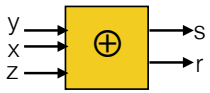
On suppose: $a = a_{n-1}\ a_{n-2} \ldots a_1\ a_0$
et $b = b_{n-1} \ \ldots \ b_1\ b_0$

1. R = 0
2. Pour i=0 à n-1 :
3. s_i , R = $[a_i \oplus b_i \oplus R]$
4. s_n = R
5. Retourner $S=[s_n \ldots s_0]$



L'addition en colonnes

"brique":



en base 2…

x	y	z	s	r
0	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

NB: l'algo marche car additionner trois nombres à 1 chiffre donne toujours un nombre à au plus 2 chiffres (une retenue et une unité).

s: unité du résultat
r: retenue

L'addition en colonnes

Généraliser à des nombres de tailles différentes…

Ecrire un algorithme qui gère des nombres d'une taille maximale et qui détecte les dépassements.

…

Un algorithme n'est pas un programme: il n'est pas destiné à un ordinateur.

On peut l'écrire de manière plus ou moins formelle et précise…

Mais il ne doit pas y avoir d'ambiguité !

En général, on utilisera du "pseudo-code" pour les écrire… plus souple qu'un langage de programmation, plus clair que des phrases.

Comment écrire des algorithmes ?

Les briques de base:

- des variables et des types de données élémentaires (entiers, flottants, fractions, chaînes de caractères,…)
- instructions classiques:
  if then else / while / repeat / for / …
  Si Alors Sinon / Tant que / Répéter … jusqu'à / Pour … / …
- des procédures, des fonctions
- des structures de données "évoluées":
  tableaux, piles, files, listes, arbres,…

Exemple n° 2

La recherche du min

La recherche du min

Trouver le minimum dans une liste de nombres

1. noter le premier nombre de la liste
2. pour tous les nombres suivants, faire:
   3. Si le nombre est inférieur à celui noté, alors remplacer celui-ci par le nouveau
4. renvoyer le nombre noté

12 5 22 34 56 14 78 121 23 4 38

La recherche du min

Problème:

- données: une liste ***non vide*** d'entiers L
- résultat: l'entier le plus petit de L

```
m := L[0]
Pour i=1 à |L|-1 Faire:
    Si L[i] < m Alors m:=L[i]
Retourner m
```

|L| = taille de L

Des instances:

- [3, 10, 5, 24, 2, 5,12]
- [4]
- [ ]

La recherche du min

Problème:

- données: une liste ***non vide*** d'entiers L
- résultat: l'entier le plus petit de L

L[i:j]=[L(i),…,L(j-1)]

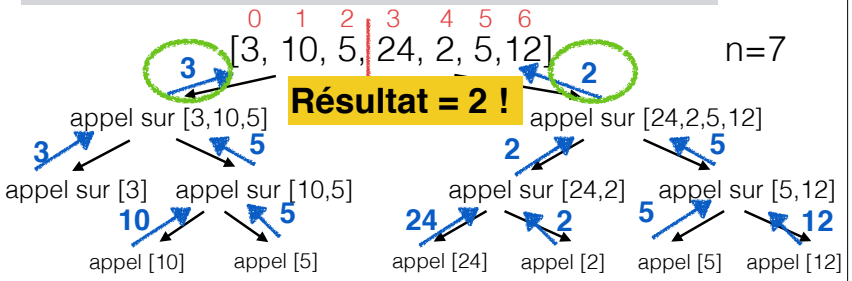
```
Def Recherche-MIN(L) :
n = |L|
Si n == 1 Alors Retourner L[0]
Sinon :
   m1 = Recherche-MIN(L[0 : n/2] )
   m2 = Recherche-MIN(L[n/2 : n] )
 Si m1 < m2 Alors res := m1 Sinon res:=m2
Retourner res
```

⌊n/2⌋

"diviser pour régner"

La recherche du min

```
Def Recherche-MIN(L) :
n = |L|
Si n == 1 Alors Retourner L[0]
Sinon :
   m1 = Recherche-MIN(L[0 : n/2] )
   m2 = Recherche-MIN(L[n/2 : n] )
 Si m1 < m2 Alors res := m1 Sinon res:=m2
Retourner res
```



La recherche du min

1 problème et 2 algorithmes

Lequel choisir ?

???

Quel est le meilleur algorithme ?

Comparer des algorithmes

Plusieurs critères:

- l'efficacité / le coût (en temps, en mémoire)
- la simplicité
- la concision
- l'élégance
- …

L'efficacité d'un algorithme

- On ne veut pas mesurer le temps nécessaire en minutes ou en microsecondes.

  → On veut une notion robuste: indépendante d'un ordinateur donné, d'un compilateur, d'un langage de programmation, *etc.*

- On va évaluer le nombre d'"opérations élémentaires" dans le pire cas (ou en moyenne,…) en fonction de la taille des données.
  (on se contente d'un ordre de grandeur).

On s'intéresse parfois aussi à la quantité de mémoire nécessaire pour l'exécution d'un algorithme.

On parle du coût ou de la complexité de l'algorithme.
Voir le cours EA4…

L'efficacité d'un algorithme
Exemple

Recherche du minimum dans une liste L:

L: 12 5 22 34 56 14 78 121 23 4 38

L contient 11 nombres: il faut **10** comparaisons.

Si L contient n nombres, il faut n-1 comparaisons.

Taille des données: n
Coût de l'algorithme: n-1

$\mathcal{C}(n) = n-1$

"algorithme linéaire"

L'efficacité d'un algorithme

On dit qu'un algorithme est efficace lorsque sa fonction de coût est polynomiale: n, n^2,n^3…

C'est très très différent d'une fonction en 2^n ou n! (algorithme exponentiel)

n	10	50	100	300
n^2	100	2500	10000	90000
n^3	1000	125000	10^6	27.10^6
2^n	1024	…(16c)	…(31c)	…(91c)
n!	$3,6.10^6$	…(65c)	…(161c)	…(623c)

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

Exemple n° 3

La recherche dichotomique

Comment rechercher un nom dans un annuaire ?

Combien fait-on de comparaisons lorsqu'on cherche si il y a un M. Turing dans un annuaire de 1 million de noms ?

Recherche dans un annuaire

Au début, la zone de recherche contient 1 million de noms.
Après la première comparaison, il reste 500000 noms...

1	500000
2	250000
3	125000
4	62500
5	31250
6	15625
7	7812
8	3906
9	1953
10	976

11	488
12	244
13	122
14	61
15	30
16	15
17	7
18	3
19	1
20	0

Il y a au maximum 20 comparaisons à faire:
La recherche dichotomique est un algorithme très efficace !

La recherche dichotomique

Problème:

- données: un tableau ***trié*** d'éléments T, un élément x
- résultat: l'indice de x dans T si x∈T, -1 sinon.

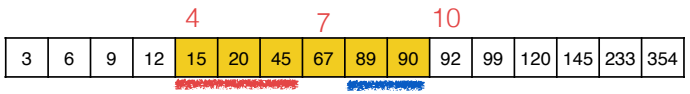
L'algorithme recherche x dans une zone de T:

Initialement la recherche porte sur la zone (0,|T|).

```
Def R-dicho(T,d,f,x) :
Si d≥f Alors Retourner -1
Sinon :
  m = (d+f)/2
  v=T[m]
  Si v==x Alors Retourner m
  Sinon Si x<v Alors Retourner R-dicho(T,d,m,x)
        Sinon Retourner R-dicho(T,m+1,f,x)
```

La recherche dichotomique

```
Def R-dicho(T,d,f,x) :
Si d≥f Alors Retourner -1
Sinon :
  m = (d+f)/2
  v=T[m]
  Si v==x Alors Retourner m
  Sinon Si x<v Alors Retourner R-dicho(T,d,m,x)
        Sinon Retourner R-dicho(T,m+1,f,x)
```



Appel R-dicho(T,4,10,x) m=7

Si x=23, on continue sur (4,7)
Si x=70, on continue sur (8,10)

(Exo: écrire la version itérative)

Premier bilan…

Algorithmes

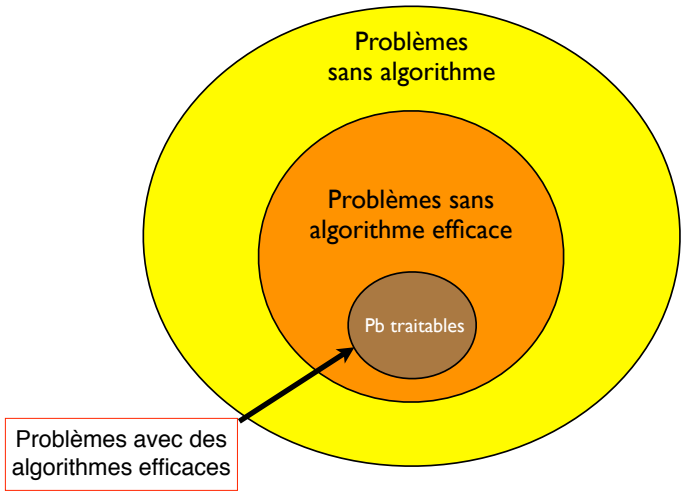
- Il peut exister plusieurs algorithmes pour un problème.
- On cherche des algorithmes corrects et efficaces:
  - ★ Corrects = qu'ils calculent bien ce que l'on veut !
  - ★ Efficaces = ayant un temps de calcul raisonnable !

Trouver le bon algorithme

Etant donné un problème, il est très important de chercher un "bon" algorithme...

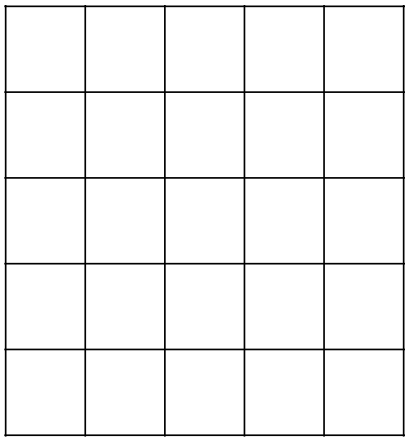
Mais ce n'est pas toujours facile !

Vue globale

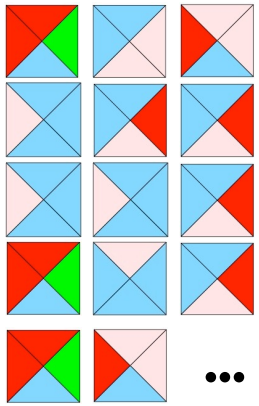


Un puzzle...

Une grille 5x5



25 tuiles



Puzzle

On essaie toutes les possibilités ?

25 tuiles 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...)

Trouver le bon algorithme

Etant donné un problème, il est très important de chercher un "bon" algorithme...

Mais ce n'est pas toujours facile !

Et ce n'est pas toujours possible !!