Introduction à l'algorithmique
- première partie -

Vasile-Marian SCUTURICI

Objectif

- Apprendre les bonnes bases pour programmer un ordinateur
  - Concevoir un algorithme
  - Evaluer un algorithme
  - Maîtriser quelques algorithmes utiles

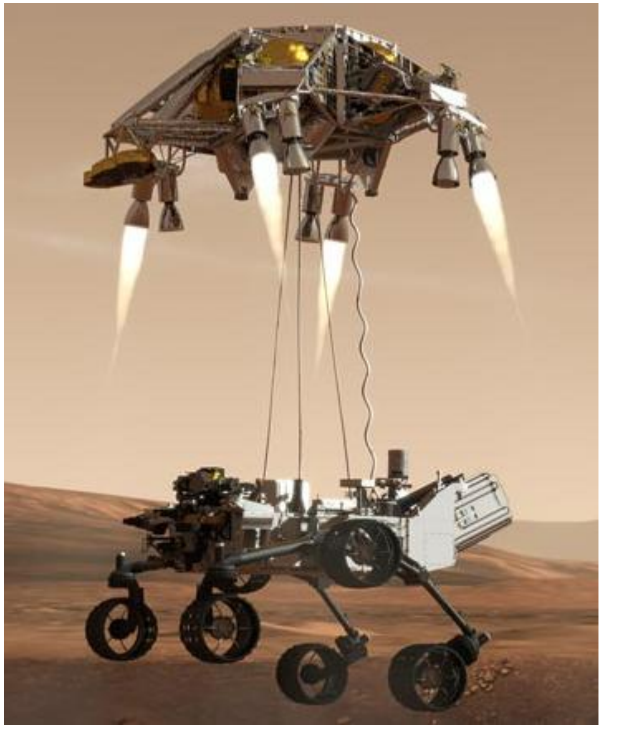
- Structures de données -> 2ème partie (Eric Guerin)

Bibliographie

- **Introduction to algorithms**, Cormen, Leiserson, Rivest, Stein, MIT press, Third edition, 2009.
- **The Algorithm Design Manual**, Steven Skiena, Springer, 2010
- **Algorithms and Data Structures**, Robert Sedgewick, Kevin Wayne, Princeton, 2007
- **Algorithms**, Dasgupta, Papadimitriou, and Vazirani, McGraw-Hill, 2006.
- **Competitive Programming**, Steven Halim, Felix Halim
- ...
- Support de cours (Moodle)
  - http://moodle2.insa-lyon.fr/

Motivation 1/3

- Algorithme = méthode pour résoudre un problème
- Applications :
  - Internet : routage, Web search
  - Robotique spatiale : Curiosity
  - Graphique : jeux vidéo
  - Transports : trajet optimal
  - Sécurité : https
  - Automobile : airbag
  - IA
  - …



Motivation 2/3

Motivation 3/3

Solutions de qualité pour des problèmes complexes

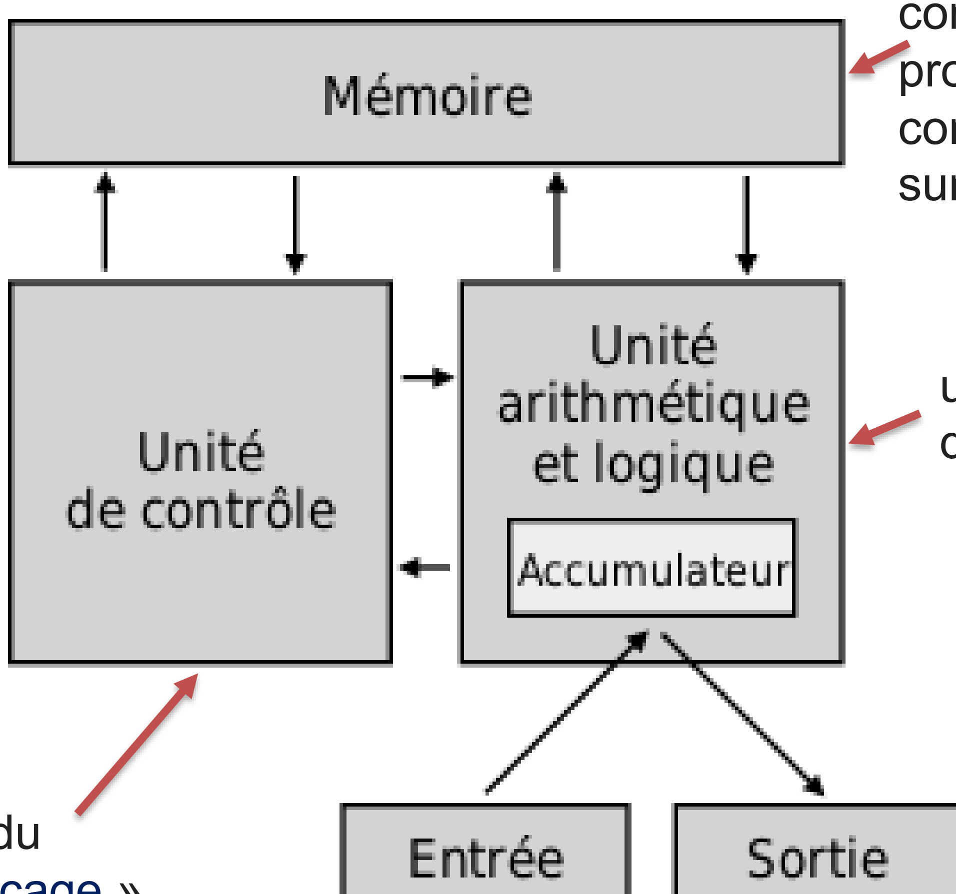
IAGen ?

Court historique

- Algorithme
  - L'origine du mot : Al-Khwarizmi, mathématicien perse du IX$^{\text{éme}}$ siècle + arithmos (= nombre en grec) → algorithmus (en latin médiéval)
  - Premiers algorithmes connus : les anciens grecs (Euclid, Eratosthène, Archimède)
  - Formalisation en 1936-1937 par Alan Turing : la machine Turing

[Larousse] Ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations. Un algorithme peut être traduit, grâce à un langage de programmation, en un programme exécutable par un ordinateur.

Description (très) schématique d'un ordinateur – architecture de von Neumann (1945)



contient à la fois les données et le programme ; indique à l'unité de contrôle quels sont les calculs à faire sur ces données

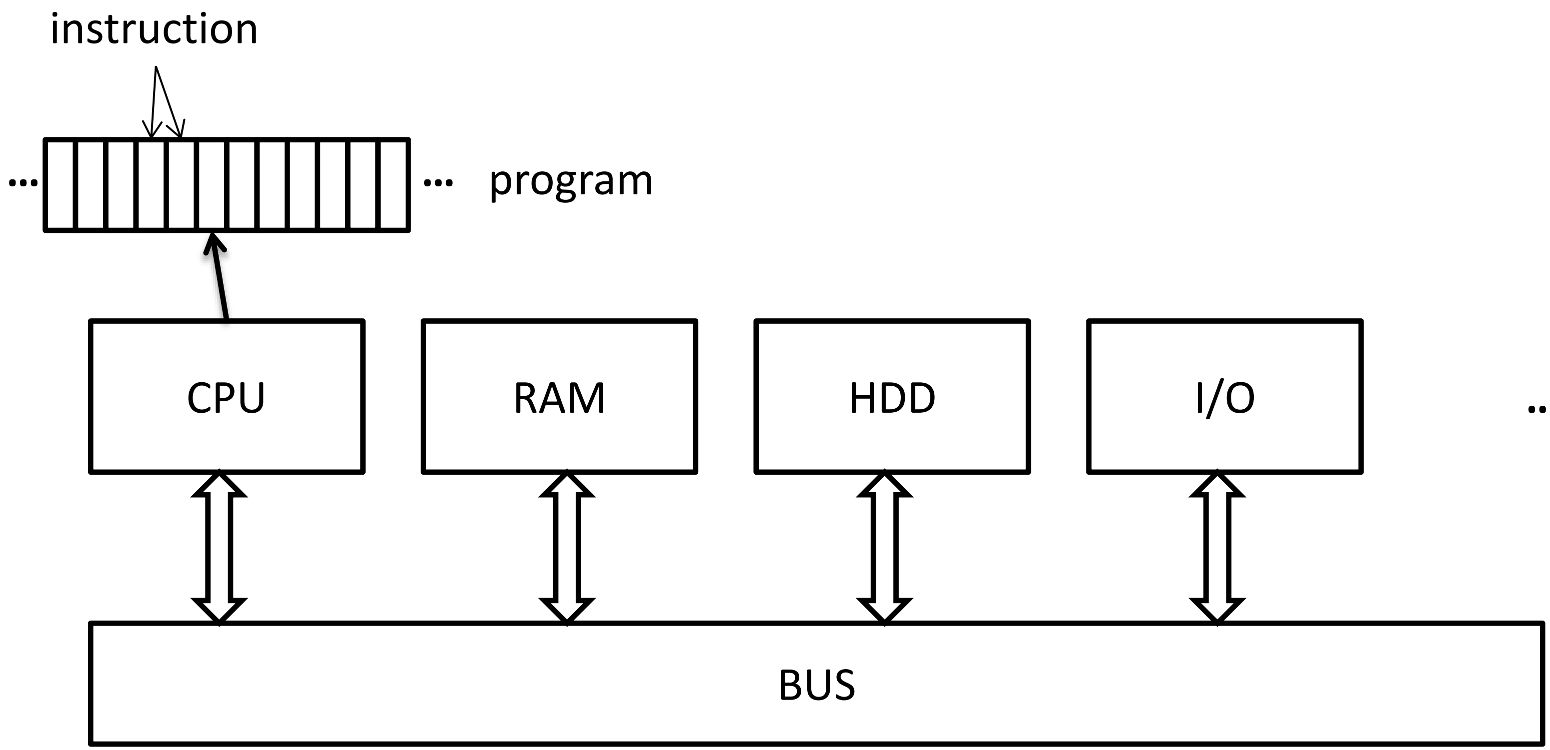
unité de traitement : son rôle est d'effectuer les opérations de base

chargée du « séquençage » des opérations

permettent de communiquer avec le monde extérieur

[Wikipedia]

Description (très) schématique d'un ordinateur

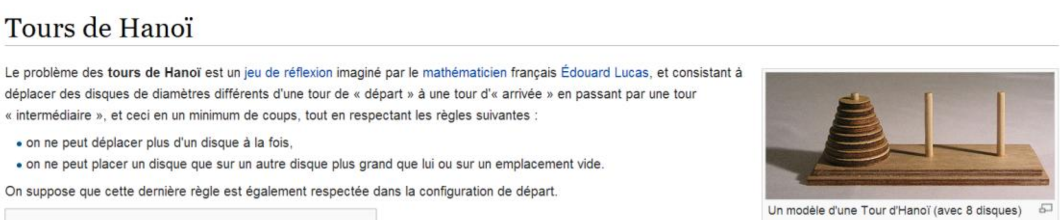


Programme

- Un ordinateur exécute des programmes, décrites via des instructions très simple
- Programme : permet de résoudre un problème
- Programme = algorithmes + structures de données (N. Wirth)

De problème à programme

- Plusieurs niveaux de description (langages) permettant de passer d'un problème décrite en langage naturel vers un code directement exécutable par une machine



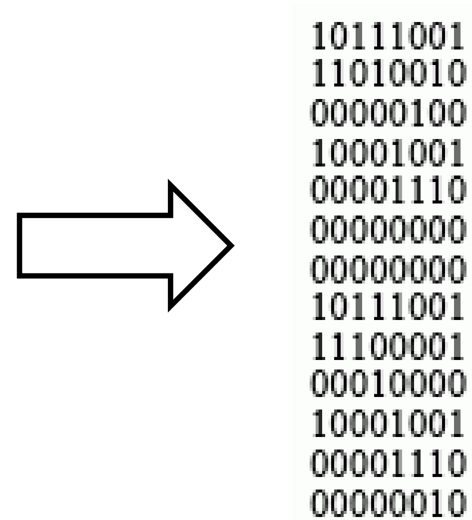
Tours de Hanoï

Le problème des **tours de Hanoï** est un jeu de réflexion imaginé par le mathématicien français Édouard Lucas, et consistant à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :

- on ne peut déplacer plus d'un disque à la fois,
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

On suppose que cette dernière règle est également respectée dans la configuration de départ.

Un modèle d'une Tour d'Hanoï (avec 8 disques)



Description d’un problème

- Problème initiale : langage naturel
  - description informelle, ambigüe et incomplète
- Spécification formelle : ce que doit faire le programme
  - non ambigüe, complète et correcte
  - Spécifie :
    - les paramètres en entrée,
    - les paramètres en sortie,
    - les pré-conditions,
    - la relation entre les paramètres en entrée et les paramètres en sortie,
    - les contraintes à respecter pour la résolution du problème (notamment les contraintes de ressources).

Algorithme

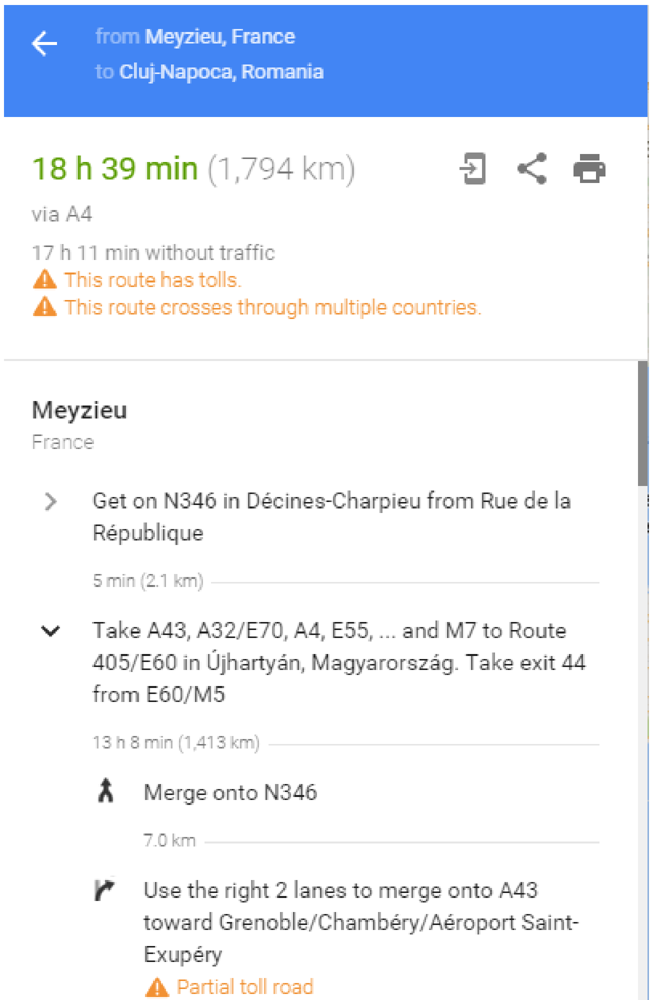
- spécification formelle -> élaboration d’un algorithme
- Algorithme
  – spécifie le “comment”, c’est à dire l’enchaînement d’opérations élémentaires à effectuer pour résoudre le problème
  – *Impérative ("comment")* != *déclarative ("quoi")*
- Algorithme
  – suite finie et non-ambiguë d’opérations ou d'instructions permettant de résoudre un problème

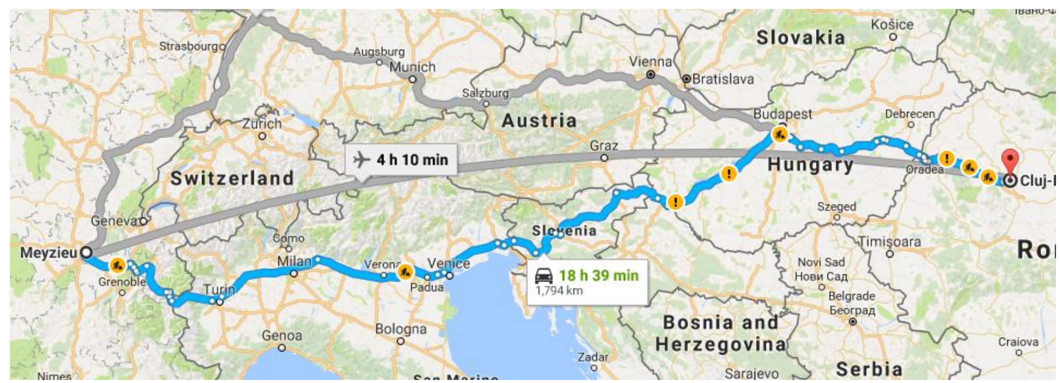
Description d’un problème - exemple

- Aller de Meyzieu à Cluj-Napoca
  - Départ : Meyzieu
  - Arrivée : Cluj-Napoca
  - Contraintes :
    - Voiture
    - Arrêt à Wien
    - Passer par la Suisse

« QUOI »

Algorithme - exemple





« COMMENT »

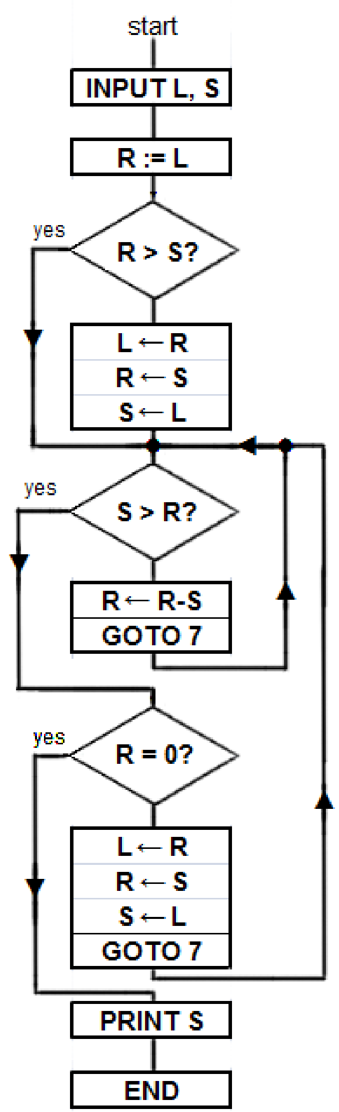
Algorithme - évaluation

- Utilisation de ressources
  - Temps processeur
  - Mémoire
- Terminaison : pour chaque entrée il se termine
- Correctitude : pour chaque entrée il produit la bonne sortie

Description d'un algorithme

- langage naturel
- graphiquement
- pseudo-code
- langage de programmation
- …

```
MAX-HEAPIFY(A, i)
l ← LEFT(i)
r ← RIGHT(i)
if l ≤ heap-size[A] and A[l] > A[i]
   then largest ← l
   else largest ← i
if r ≤ heap-size[A] and A[r] > A[largest]
   then largest ← r
if largest ≠ i
   then exchange A[i] ↔ A[largest]
        MAX-HEAPIFY(A, largest)
```



Pseudo-code

Partie statique - QUOI

Procédure : moyenne$(a, b, resultat)$

Entrée : entier a

entier b

Sortie : réel $resultat$, correspondant à $(a + b)/2$

début

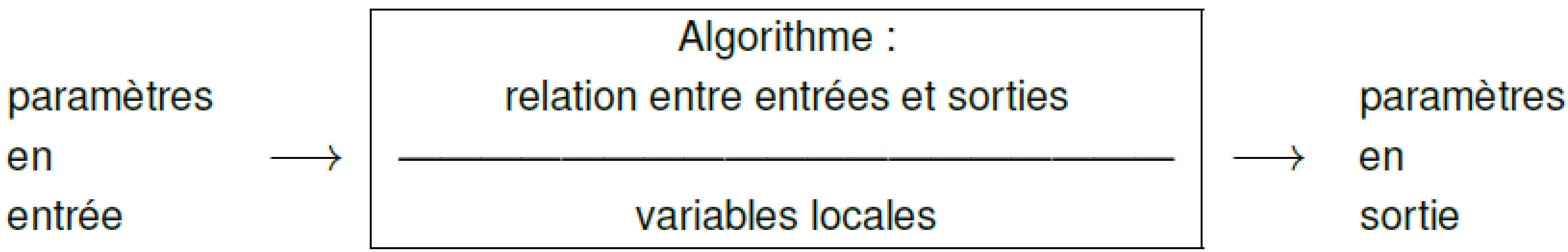
$resultat \leftarrow (a + b);$

$resultat \leftarrow resultat/2;$

Partie dynamique - COMMENT

Variables

- Nom, type, valeur



Variables - nom

- Nom/identificateur : représentation symbolique de l'adresse où est stockée la valeur de la variable en mémoire

- Donner un nom suggestif !

- Exemples :
  - Mot, Resultat, Perimetre, SommeTotale …
  - ~~i23, s12~~

Variables - type

- Permet d’interpréter les octets correspondant à la valeur de la variable

- entier
- réel
- car
- booléen (logique)
- tableau
- …

Algorithme – partie statique

Procédure *nom-de-la-procédure(<noms-des-paramètres>)*

Entrée : pour chaque paramètre en entrée, préciser son type et son nom

Sortie : pour chaque paramètre en sortie, préciser son type et son nom

Précondition : Conditions sur les paramètres en entrée

Postcondition : Relation entre les paramètres en entrée et ceux en sortie

Déclaration : pour chaque variable locale, préciser son type et son nom

const : pour chaque constante, préciser son nom et sa valeur

début

Suite d'opérations élémentaires permettant de calculer les paramètres en sortie en fonction des paramètres en entrée

(partie dynamique de l'algorithme)

Algorithme – partie statique - exemple

Procédure *racines*(a, b, c, r_1, r_2)

Entrée : réel a
réel b
réel c

Sortie : réel r_1
réel r_2

Précondition : $b^2 - 4ac \geq 0$ **et** $a \neq 0$

Postcondition : $ar_1^2 + br_1 + c = ar_2^2 + br_2 + c = 0$ ou autrement dit,
r_1 et r_2 sont les deux solutions de
l'équation $ax^2 + bx + c = 0$

Déclaration : réel *delta*

début

Suite d'opérations élémentaires
permettant de calculer r_1 et r_2 à partir de a, b et c.

Algorithme – partie statique - exemple

- Forme abrégée

Procédure *racines*(a, b, c, r_1, r_2)

Entrée : réel a, b, c

Sortie : réel r_1, r_2

Pseudo-code : instructions

- Affectation
- Expressions
- Instructions conditionnelles
- Boucles
- Entrée/sorties

Affectation

$$\texttt{nom-var} \leftarrow \texttt{expr}$$

- La variable *nom-var* recevra la valeur de l'expression *expr*
- Expression
  - Valeur explicite $a \leftarrow 25$
  - Valeur d'une variable $a \leftarrow b$
  - Résultat d'une opération entre d'autres expressions $r_1 \leftarrow r_1/3.0$

Expressions

entier e_1, e_2,
réel r_1,
logique b_1, b_2

- Operations arithmétiques

$$e_1 \leftarrow (10 \;\; mod \;\; 3) + (5 \;\; div \;\; 2)$$
$$e_2 \leftarrow e_1 * 2$$
$$r_1 \leftarrow 4.5 * 2.0$$

- Operations de comparaison
- Operations logiques

$$b_1 \leftarrow (r_1 > 4.2) \texttt{ ou } (e_1 = e_2)$$
$$b_2 \leftarrow \texttt{non}(b_1) \texttt{ et } r_1 \leq 4.2$$

Enchaînement d’instructions

- Séquentiel
- Alternatif
- Répétitif

Enchaînement séquentiel

Procédure *cercle*(R, D, P, S)

Entrée : réel R

Sortie : réel D, P, S

Précondition : $R \geq 0$

Postcondition : D, P et S contiennent respectivement le diamètre, le périmètre et la surface d'un cercle de rayon R

Déclaration : **const** : pi = 3.14

début

$D \leftarrow 2 * R$

$P \leftarrow D * pi$

$S \leftarrow R * R * pi$

R	3
D	?
P	?
S	?

Mémoire …

Enchaînement séquentiel

Procédure *cercle*(R, D, P, S)

Entrée : réel R

Sortie : réel D, P, S

Précondition : $R \geq 0$

Postcondition : D, P et S contiennent respectivement le diamètre, le périmètre et la surface d'un cercle de rayon R

Déclaration : **const** : pi = 3.14

début

$\boxed{D \leftarrow 2 * R}$

$P \leftarrow D * pi$

$S \leftarrow R * R * pi$

R	3
D	6
P	?
S	?

Enchaînement séquentiel

Procédure *cercle*(R, D, P, S)

Entrée : réel R

Sortie : réel D, P, S

Précondition : $R \geq 0$

Postcondition : D, P et S contiennent respectivement le diamètre, le périmètre et la surface d'un cercle de rayon R

Déclaration : **const** : pi = 3.14

début

$D \leftarrow 2 * R$

$\boxed{P \leftarrow D * pi}$

$S \leftarrow R * R * pi$

R	3
D	6
P	18.84…
S	?

Enchaînement séquentiel

Procédure *cercle*(R, D, P, S)

Entrée : réel R

Sortie : réel D, P, S

Précondition : $R \geq 0$

Postcondition : D, P et S contiennent respectivement le diamètre, le périmètre et la surface d'un cercle de rayon R

Déclaration : **const** : pi = 3.14

début

$D \leftarrow 2 * R$

$P \leftarrow D * pi$

$\boxed{S \leftarrow R * R * pi}$

R	3
D	6
P	18.84…
S	28.26…

Enchaînement séquentiel

Procédure *cercle*(R, D, P, S)

Entrée : réel R

Sortie : réel D, P, S

Précondition : $R \geq 0$

Postcondition : D, P et S contiennent respectivement le diamètre, le périmètre et la surface d'un cercle de rayon R

Déclaration : **const** : pi = 3.14

début

$D \leftarrow 2 * R$

$P \leftarrow D * pi$

$S \leftarrow R * R * pi$

Coût : 3

Instructions : organisation en blocs

- Bloc = instructions avec le même niveau d'indentation
- Un bloc est considéré comme une instruction

```
début
    a ← 1;
    b ← 1;
    début
        a ← a + b;
        b ← a + b;
    début
        a ← b * 10;
        b ← a * 10;
        début
            a ← a * 10;
            b ← b * 10;
```

Instructions : organisation en blocs

```
début
    a ← 1;
    b ← 1;
    début
        a ← a + b;
        b ← a + b;
    début
        a ← b * 10;
        b ← a * 10;
        début
            a ← a * 10;
            b ← b * 10;
```

Coût : 8

Enchaînement alternatif

- L'instruction *si*

```
début
    si condition alors
        bloc du alors
    sinon
        bloc du sinon
```

```
début
    si condition alors
        bloc du alors
```

```
si condition1 alors
    suite-1
sinon si condition2 alors
    suite-2
sinon si condition3 alors
    suite-3
sinon si condition4 alors
    suite-4
sinon
    suite-5
```

Enchaînement alternatif - exemple

Procédure $min(a, b, resultat)$
  Entrée : entier a
  entier b
  Sortie : entier $resultat$
  début
    si $a < b$ **alors**
      $resultat \leftarrow a$
    sinon
      $resultat \leftarrow b$

<a=10, b=12>

a	10
b	12
resultat	?

Enchaînement alternatif - exemple

Procédure $min(a, b, resultat)$
Entrée : entier a
entier b
Sortie : entier $resultat$
début
si $a < b$ **alors**
$resultat \leftarrow a$
sinon
$resultat \leftarrow b$

<a=10, b=12>

a	10
b	12
resultat	?

Enchaînement alternatif - exemple

Procédure $min(a, b, resultat)$
Entrée : entier a
entier b
Sortie : entier $resultat$
début
si $a < b$ **alors**
$\boxed{resultat \leftarrow a}$
sinon
$resultat \leftarrow b$

<a=10, b=12>

a	10
b	12
resultat	10

Enchaînement alternatif - exemple

Procédure $min(a, b, resultat)$

Entrée : entier a
entier b

Sortie : entier $resultat$

début

si $a < b$ **alors**

$resultat \leftarrow a$

sinon

$resultat \leftarrow b$

Coût : 2

```
Procédure nb-solutions(a, b, c, nbSol)
    Entrée         : réel a, b, c
    Sortie         : entier nbSol
    Postcondition  : nbSol = nombre de solutions sur les réels de l'équation ax^2 + bx + c = 0
    Déclaration    : réel delta
    début
        si a = 0 alors
            si b = 0 alors
                si c = 0 alors
                    nbSol ← ∞
                sinon
                    nbSol ← 0
            sinon
                nbSol ← 1
        sinon
            delta ← b * b − 4 * a * c
            si delta < 0 alors
                nbSol← 0
            sinon si delta = 0 alors
                nbSol ← 1
            sinon
                nbSol ← 2
```

Coût : ?

```
Procédure nb-solutions(a, b, c, nbSol)
    Entrée        : réel a, b, c
    Sortie        : entier nbSol
    Postcondition : nbSol = nombre de solutions sur les réels de l'équation ax^2 + bx + c = 0
    Déclaration   : réel delta
    début
        si a = 0 alors
            si b = 0 alors
                si c = 0 alors
                    nbSol ← ∞
                sinon
                    nbSol ← 0
            sinon
                nbSol ← 1
        sinon
            delta ← b * b − 4 * a * c
            si delta < 0 alors
                nbSol ← 0
            sinon si delta = 0 alors
                nbSol ← 1
            sinon
                nbSol ← 2
```

Coût : 3 ou 4 ou 5 $\rightarrow$ 5

Recherche du plus petit de 3 nombres

Procédure *plus-petit*(a, b, c, pp)
...
début
si $a < b$ **alors**
$pp \leftarrow a$
sinon
$pp \leftarrow b$
// $pp =$ plus petit nombre entre a et b
si $c < pp$ **alors**
$pp \leftarrow c$

Coût : ?

Enchaînement répétitif

tant que *(condition)* **faire**
  bloc de la boucle

pour $(init; cond; passage)$ **faire**
  bloc de la boucle

répéter
  bloc de la boucle
jusqu'à *(condition d'arrêt)*;

L’instruction *tant que*

tant que *(condition)* **faire**
  bloc de la boucle

- Répétition d’un bloc d’instruction un nombre de fois

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)
Entrée : entier n
réel x
Sortie : réel p
Précondition : $n \geq 0$
Postcondition : $p = x^n$
Déclaration : entier cpt
début
$p \leftarrow 1$
$cpt \leftarrow 0$
tant que $cpt < n$ **faire**
// invariant : $p = x^{cpt}$
$p \leftarrow p * x$
$cpt \leftarrow cpt + 1$

$<n = 3, x = 2>$

n	3
x	2
p	?
cpt	?

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)

Entrée : entier n

réel x

Sortie : réel p

Précondition : $n \geq 0$

Postcondition : $p = x^n$

Déclaration : entier cpt

début

$\boxed{p \leftarrow 1}$

$cpt \leftarrow 0$

tant que $cpt < n$ **faire**

// invariant : $p = x^{cpt}$

$p \leftarrow p * x$

$cpt \leftarrow cpt + 1$

<$n = 3$, $x = 2$>

n	3
x	2
p	1
cpt	?

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)

Entrée : entier n
réel x

Sortie : réel p

Précondition : $n \geq 0$

Postcondition : $p = x^n$

Déclaration : entier cpt

début

$p \leftarrow 1$

$\boxed{cpt \leftarrow 0}$

tant que $cpt < n$ **faire**

`// invariant :` $p = x^{cpt}$

$p \leftarrow p * x$

$cpt \leftarrow cpt + 1$

$<n = 3, x = 2>$

n	3
x	2
p	1
cpt	0

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)

Entrée : entier n

réel x

Sortie : réel p

Précondition : $n \geq 0$

Postcondition : $p = x^n$

Déclaration : entier cpt

début

$p \leftarrow 1$

$cpt \leftarrow 0$

tant que $cpt < n$ **faire**

// invariant : $p = x^{cpt}$

$p \leftarrow p * x$

$cpt \leftarrow cpt + 1$

$<n = 3,\ x = 2>$

n	3
x	2
p	1
cpt	0

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)

Entrée : entier n

réel x

Sortie : réel p

Précondition : $n \geq 0$

Postcondition : $p = x^n$

Déclaration : entier cpt

début

$p \leftarrow 1$

$cpt \leftarrow 0$

tant que $cpt < n$ **faire**

// invariant : $p = x^{cpt}$

$p \leftarrow p * x$

$cpt \leftarrow cpt + 1$

<$n = 3$, $x = 2$>

n	3
x	2
p	2
cpt	0

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)
  Entrée : entier n
  réel x
  Sortie : réel p
  Précondition : $n \geq 0$
  Postcondition : $p = x^n$
  Déclaration : entier cpt
  début
    $p \leftarrow 1$
    $cpt \leftarrow 0$
    tant que $cpt < n$ **faire**
      // invariant : $p = x^{cpt}$
      $p \leftarrow p * x$
      $cpt \leftarrow cpt + 1$

$<n = 3, x = 2>$

n	3
x	2
p	2
cpt	1

tant que – exemple (x^n)

```
Procédure puissance(n, x, p)
    Entrée         : entier n
                     réel x
    Sortie         : réel p
    Précondition   : n ≥ 0
    Postcondition  : p = x^n
    Déclaration    : entier cpt
    début
        p ← 1
        cpt ← 0
        tant que cpt < n faire
            // invariant : p = x^cpt
            p ← p * x
            cpt ← cpt + 1
```

$<n = 3, x = 2>$

n	3
x	2
p	2
cpt	1

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)

Entrée : entier n

réel x

Sortie : réel p

Précondition : $n \geq 0$

Postcondition : $p = x^n$

Déclaration : entier cpt

début

$p \leftarrow 1$

$cpt \leftarrow 0$

tant que $cpt < n$ **faire**

// invariant : $p = x^{cpt}$

$p \leftarrow p * x$

$cpt \leftarrow cpt + 1$

$<n = 3, x = 2>$

n	3
x	2
p	4
cpt	1

tant que – exemple (x^n)

```
Procédure puissance(n, x, p)
    Entrée         : entier n
                     réel x
    Sortie         : réel p
    Précondition   : n ≥ 0
    Postcondition  : p = x^n
    Déclaration    : entier cpt
    début
        p ← 1
        cpt ← 0
        tant que cpt < n faire
            // invariant : p = x^cpt
            p ← p * x
            cpt ← cpt + 1
```

$<n = 3, x = 2>$

n	3
x	2
p	4
cpt	2

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)

Entrée : entier n

réel x

Sortie : réel p

Précondition : $n \geq 0$

Postcondition : $p = x^n$

Déclaration : entier cpt

début

$p \leftarrow 1$

$cpt \leftarrow 0$

tant que $cpt < n$ **faire**

// invariant : $p = x^{cpt}$

$p \leftarrow p * x$

$cpt \leftarrow cpt + 1$

$<n = 3, x = 2>$

n	3
x	2
p	4
cpt	2

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)

Entrée : entier n
réel x

Sortie : réel p

Précondition : $n \geq 0$

Postcondition : $p = x^n$

Déclaration : entier cpt

début

  $p \leftarrow 1$

  $cpt \leftarrow 0$

  tant que $cpt < n$ **faire**

    // invariant : $p = x^{cpt}$

    $p \leftarrow p * x$

    $cpt \leftarrow cpt + 1$

$<n = 3, x = 2>$

n	3
x	2
p	8
cpt	2

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)

  Entrée : entier n

    réel x

  Sortie : réel p

  Précondition : $n \geq 0$

  Postcondition : $p = x^n$

  Déclaration : entier cpt

  début

    $p \leftarrow 1$

    $cpt \leftarrow 0$

    tant que $cpt < n$ **faire**

      `// invariant :` $p = x^{cpt}$

      $p \leftarrow p * x$

      $\boxed{cpt \leftarrow cpt + 1}$

$<n = 3, x = 2>$

n	3
x	2
p	8
cpt	3

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)

Entrée : entier n

réel x

Sortie : réel p

Précondition : $n \geq 0$

Postcondition : $p = x^n$

Déclaration : entier cpt

début

  $p \leftarrow 1$

  $cpt \leftarrow 0$

  tant que $cpt < n$ **faire**

    // invariant : $p = x^{cpt}$

    $p \leftarrow p * x$

    $cpt \leftarrow cpt + 1$

$<n = 3,\ x = 2>$

n	3
x	2
p	8
cpt	3

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)

- **Entrée** : entier n
  réel x
- **Sortie** : réel p
- **Précondition** : $n \geq 0$
- **Postcondition** : $p = x^n$
- **Déclaration** : entier cpt

début

  $p \leftarrow 1$
  $cpt \leftarrow 0$
  tant que $cpt < n$ **faire**
    `// invariant :` $p = x^{cpt}$
    $p \leftarrow p * x$
    $cpt \leftarrow cpt + 1$

$<n = 3, x = 2>$

Coût : ?

tant que – exemple (x^n)

Procédure *puissance*(n, x, p)

$<n = 3, x = 2>$

- **Entrée** : entier n
  réel x
- **Sortie** : réel p
- **Précondition** : $n \geq 0$
- **Postcondition** : $p = x^n$
- **Déclaration** : entier cpt

début

  $p \leftarrow 1$

  $cpt \leftarrow 0$

  tant que $cpt < n$ **faire**

    `// invariant :` $p = x^{cpt}$

    $p \leftarrow p * x$

    $cpt \leftarrow cpt + 1$

Coût : 1+1+3*3+1 = 12

Coût : 1+1+3*n+1 = 3*(n+1)

tant que - exemple

début
- $p \leftarrow 1$
- $cpt \leftarrow 0$
- **tant que** $cpt < n$ **faire**
  - // invariant : $p = x^{cpt}$
  - $p \leftarrow p * x$
  - $cpt \leftarrow cpt + 1$

début
- $p \leftarrow 1$
- $cpt \leftarrow 1$
- **tant que** $cpt \leq n$ **faire**
  - // invariant : $p = x^{cpt-1}$
  - $p \leftarrow p * x$
  - $cpt \leftarrow cpt + 1$

début
- $p \leftarrow 1$
- $cpt \leftarrow n$
- **tant que** $cpt > 0$ **faire**
  - // invariant : $p = x^{n-cpt}$
  - $p \leftarrow p * x$
  - $cpt \leftarrow cpt - 1$

L'instruction *pour*

début

$p \leftarrow 1$

$cpt \leftarrow 0$ — initialisation

tant que $cpt < n$ **faire** — condition

// `invariant` : $p = x^{cpt}$

$p \leftarrow p * x$ — traitement

$cpt \leftarrow cpt + 1$ — passage

L'instruction *pour*

```
pour (init ; cond ; passage) faire
    traitement
```

```
init
tant que cond faire
    traitement
    passage
```

L'instruction *pour* - exemple

$\langle n = 3, x = 2 \rangle$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(cpt \leftarrow 0\,; cpt < n\,; cpt \leftarrow cpt + 1)$ **faire**

`// invariant :` $p = x^{cpt}$

$p \leftarrow p * x$

n	3
x	2
p	?
cpt	?

L'instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$\boxed{p \leftarrow 1}$

pour $(cpt \leftarrow 0\,; cpt < n\,; cpt \leftarrow cpt + 1)$ **faire**

`// invariant :` $p = x^{cpt}$

$p \leftarrow p * x$

n	3
x	2
p	1
cpt	?

L'instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(\boxed{cpt \leftarrow 0} ; cpt < n ; cpt \leftarrow cpt + 1)$ **faire**

`// invariant :` $p = x^{cpt}$

$p \leftarrow p * x$

n	3
x	2
p	1
cpt	0

L'instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(cpt \leftarrow 0 ; \boxed{cpt < n ;} cpt \leftarrow cpt + 1)$ **faire**

`// invariant :` $p = x^{cpt}$

$p \leftarrow p * x$

n	3
x	2
p	1
cpt	0

L'instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(cpt \leftarrow 0\,;cpt < n\,;cpt \leftarrow cpt + 1)$ **faire**

`// invariant :` $p = x^{cpt}$

$\boxed{p \leftarrow p * x}$

n	3
x	2
p	2
cpt	0

L'instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(cpt \leftarrow 0\,; cpt < n\,; \boxed{cpt \leftarrow cpt + 1})$ **faire**

`// invariant :` $p = x^{cpt}$

$p \leftarrow p * x$

n	3
x	2
p	2
cpt	1

L’instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(cpt \leftarrow 0\ ; \boxed{cpt < n}\ ; cpt \leftarrow cpt + 1)$ **faire**

`// invariant :` $p = x^{cpt}$

$p \leftarrow p * x$

n	3
x	2
p	2
cpt	1

L'instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(cpt \leftarrow 0\,;cpt < n\,;cpt \leftarrow cpt + 1)$ **faire**

`// invariant :` $p = x^{cpt}$

$\boxed{p \leftarrow p * x}$

n	3
x	2
p	4
cpt	1

L'instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(cpt \leftarrow 0\,; cpt < n\,; \boxed{cpt \leftarrow cpt + 1})$ **faire**

`// invariant :` $p = x^{cpt}$

$p \leftarrow p * x$

n	3
x	2
p	4
cpt	2

L'instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(cpt \leftarrow 0 ; \boxed{cpt < n ;} cpt \leftarrow cpt + 1)$ **faire**

`// invariant :` $p = x^{cpt}$

$p \leftarrow p * x$

n	3
x	2
p	4
cpt	2

L'instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(cpt \leftarrow 0\,; cpt < n\,; cpt \leftarrow cpt + 1)$ **faire**

`// invariant :` $p = x^{cpt}$

$\boxed{p \leftarrow p * x}$

n	3
x	2
p	8
cpt	2

L'instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(cpt \leftarrow 0\,; cpt < n\,; \boxed{cpt \leftarrow cpt + 1})$ **faire**

`// invariant :` $p = x^{cpt}$

$p \leftarrow p * x$

n	3
x	2
p	8
cpt	3

L'instruction *pour* - exemple

$<n = 3, x = 2>$

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

pour $(cpt \leftarrow 0 \boxed{; cpt < n ;} cpt \leftarrow cpt + 1)$ **faire**

// invariant : $p = x^{cpt}$

$p \leftarrow p * x$

n	3
x	2
p	8
cpt	3

Coût : ?

L'instruction *pour* - exemple

début
$p \leftarrow 1$
pour $(cpt \leftarrow 0\,; cpt < n\,; cpt \leftarrow cpt + 1)$ **faire**
    `// invariant :` $p = x^{cpt}$
    $p \leftarrow p * x$

début
$p \leftarrow 1$
pour $(cpt \leftarrow 1\,; cpt \leq n\,; cpt \leftarrow cpt + 1)$ **faire**
    `// invariant :` $p = x^{cpt-1}$
    $p \leftarrow p * x$

début
$p \leftarrow 1$
pour $(cpt \leftarrow 1\,; cpt \leq n\,; cpt \leftarrow cpt + 1)$ **faire**
    `// invariant :` $p = x^{cpt-1}$
    $p \leftarrow p * x$

L’instruction *répéter*

répéter
    bloc de la boucle
jusqu’à *(condition d’arrêt)*

bloc de la boucle
tant que *condition d’arrêt* **faire**
    bloc de la boucle

L'instruction *répéter* - exemple

Procédure *puissance*(n, x, p)

...

début

$p \leftarrow 1$

$cpt \leftarrow 0$

répéter

si $(cpt > 0)$ **alors**

$p \leftarrow p * x$

$cpt \leftarrow cpt + 1$

jusqu'à $(cpt \geq n)$

Instructions I/0

- saisir(x)
- afficher(x)

Rappel

- Programme = algorithmes + structures de données
- Algorithme :
  - Pseudo-code
  - Partie statique
    - Variables
  - Partie dynamique
    - Instructions
  - Coût $\rightarrow$ min !

Algorithmes Brute Force

- Apportent une solution facile à implémenter
- La solution est optimale
- Peut nécessiter un temps de calcul trop important

Force brute (BF)

- Brute force / Complete search / Recursive backtracking
- *If all you have is a hammer, everything looks like a nail* (Abraham Maslow, 1962)

- Parcourir tout l’espace de recherche pour trouver la/les solution(s).

Exemple – BF1

- Pour un tableau d'entiers, trouver la plus grande différence entre deux éléments.

Procédure $BF1(tab, k)$

Entrée : entier $tab[1..k]$

Sortie : entier $diffMax$

début

$diffMax \leftarrow 0;$

pour $(i = 1; i < k; i \leftarrow i + 1)$ **faire**

pour $(j = 1; i < k; k \leftarrow k + 1)$ **faire**

si $(abs(tab[i] - tab[j]) > diff)$ **alors**

$diffMax \leftarrow abs(tab[i] - tab[j]);$

Coût ?

Exemple – BF1

- Solution différente possible

Procédure $BF1(tab, k)$

Entrée : entier $tab[1..k]$

Sortie : entier $diffMax$

début

$diffMax \leftarrow max(tab) - min(tab);$

Coût ?

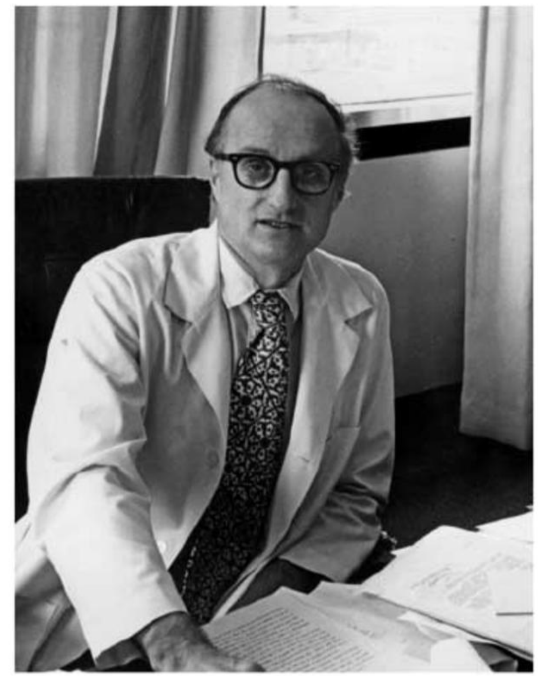
Problème – BF2

Pour deux entiers *A* et *B*, avec $1 \leq A, B \leq 10000$ énumérer tous les entiers *x*, *y* respectant les contraintes suivantes :

$$x * y = A$$
$$x^2 + y^2 = B$$

BF - exemple (et motivation …)

- Emil FREIREICH
- acute lymphoblastic leukemia (ALL)



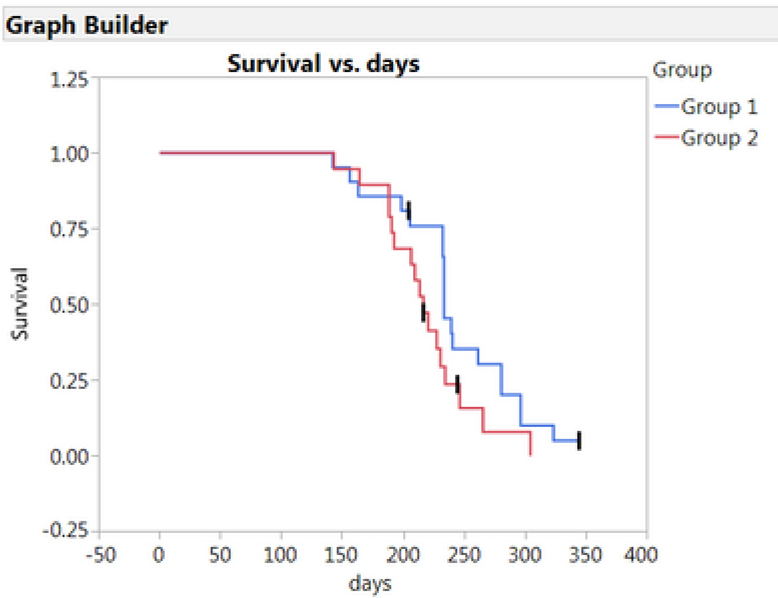
“Progress and Perspective in the Chemotherapy of Acute Leukemia”, Frei E 3rd, Freireich EJ. Advances in Chemotherapy, 1965

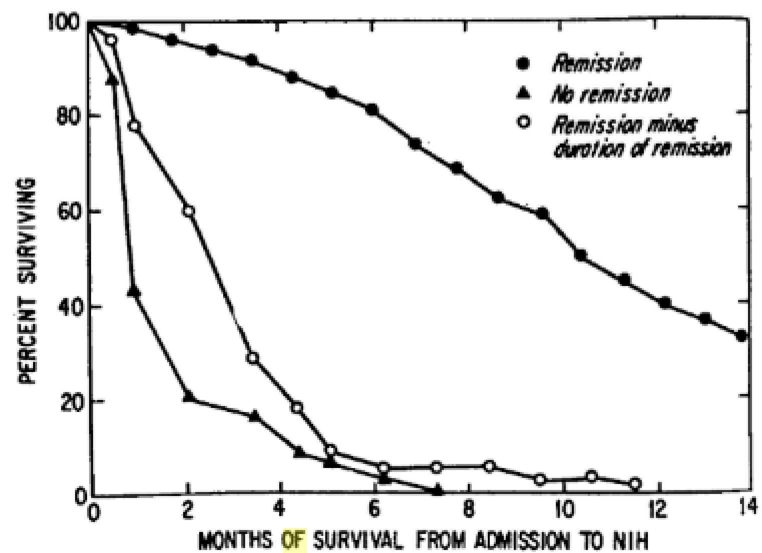
[“The Treatement – Why is so difficult to develop drugs for cancer?”, Malcolm Gladwell, The New Yorker, May 17 2010]

- Monothérapie → vers une approche multiple

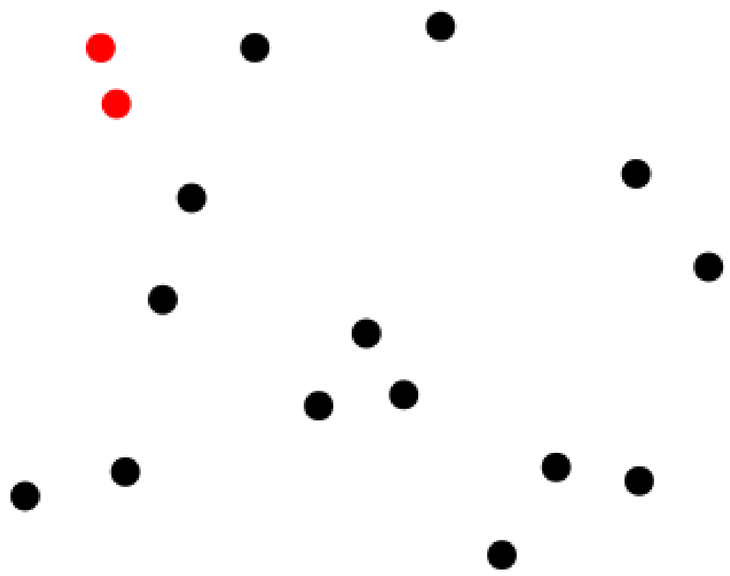
BF - exemple (et motivation …)

- Kaplan-Meyer estimator :





BF - recherche de deux points les plus rapprochés



[Wikipedia]

Problèmes – 1/2

- https://uva.onlinejudge.org/
- http://uhunt.felix-halim.net/
- http://france-ioi.fr
- http://codeforces.com/

- TopCoder, Google CodeJam, Codingame, Prologin, Project Euler …

Problèmes – 2/2

- A. Calculer la somme de *n=4* nombres réels
- B. Calculer la moyenne de *n=4* nombres réels
- C. Même problème que B, mais la moyenne pour les nombres positifs (*>= 0*)

Problème (UVA) : 1585 - Score

There is an objective test result such as ``OOXXOXXOOO''. An `O' means a correct answer of a problem and an `X' means a wrong answer. The score of each problem of this test is calculated by itself and its just previous consecutive `O's only when the answer is correct. For example, the score of the 10th problem is 3 that is obtained by itself and its two previous consecutive `O's.

Therefore, the score of ``OOXXOXXOOO'' is 10 which is calculated by ``1+2+0+0+1+0+0+1+2+3''.

You are to write a program calculating the scores of test results.

?

- http://servifa-algo.insa-lyon.fr/domjudge/