Introduction à l'algorithmique et à la programmation en Python

Marc Zeitoun

Licence CHS, 30/9/2021

Objectif

- Introduction à l'algorithmique.
- Un aperçu de quelques algorithmes.
- Un aperçu de quelques techniques algorithmiques.

L'algorithmique avant l'ordinateur

- −300 ? Algorithme d'Euclide.
- 820 Algorithmes d'Al-Khwârîzmî.
  - Résolution des équations du second degré.

L'algorithmique avant l'ordinateur

- −300 ? Algorithme d'Euclide.
- 820 Algorithmes d'Al-Khwârîzmî.
  - Résolution des équations du second degré.
- 1623 Description d'une machine à calculer de Schickard.
- 1645 Machine à calculer de Pascal.
- 1710 Machine à calculer de Leibniz (+ calcul en base 2).
- 1801 Métier à tisser de Jacquart.
- 1842 Programme de Lovelace, Machine de Babbage.

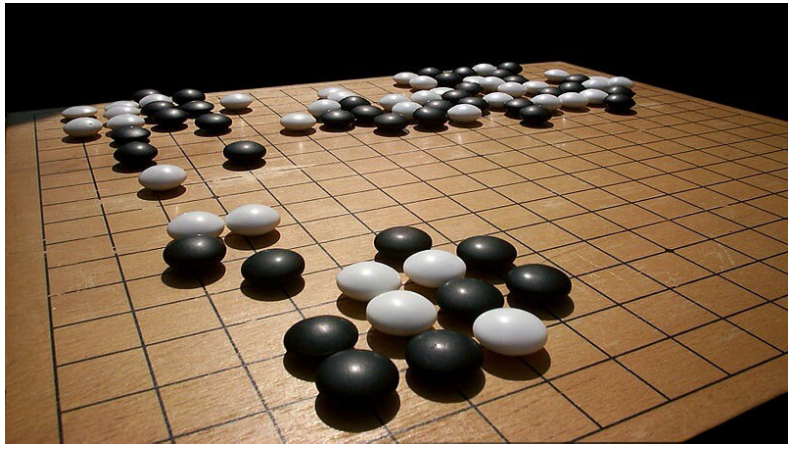
Limites des ordinateurs

- Peut-on résoudre n'importe quel problème avec un ordinateur ?

- Combien de temps faut-il pour résoudre un problème ?
  Combien de mémoire faut-il pour résoudre un problème ?

Des progrès impressionnants

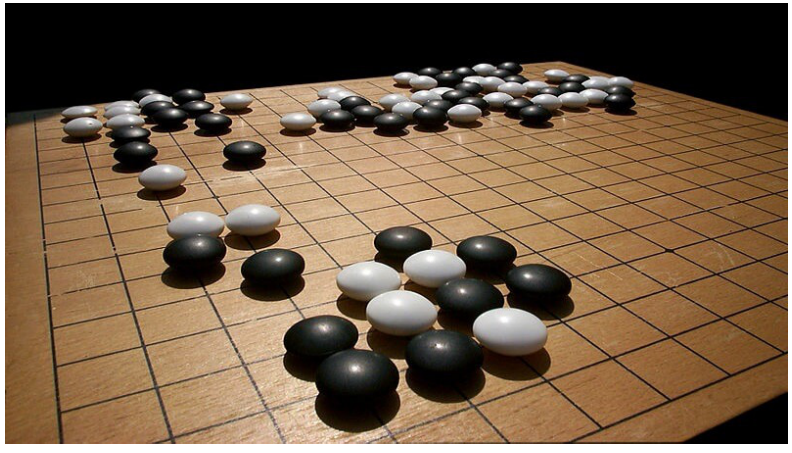
2016 : AlphaGo



Des progrès impressionnants

2016 : AlphaGo

AlphaGo



2015 : Gérard Berry

Gérard Berry : « L'ordinateur est complètement con »

« Fondamentalement, l'ordinateur et l'homme sont les deux opposés les plus intégraux qui existent. » Entretien avec Gérard Berry, informaticien et professeur au Collège de France, médaille d'or 2014 du CNRS.

Algorithmes

Algorithme : procédure de calcul

- définie par un enchaînement d'opérations simples,
- prenant en entrée une valeur,
- calculant en sortie une valeur.

Un algorithme est conçu pour résoudre un problème.

Algorithmes

Algorithme : procédure de calcul

- définie par un enchaînement d'opérations simples,
- prenant en entrée une valeur,
- calculant en sortie une valeur.

Un algorithme est conçu pour résoudre un problème.

Un algorithme provient toujours d'une **solution** au problème.

Éléments des algorithmes

Très peu d'instructions.

Exécution séquentielle.

- Variables et affectation,
- Opérations d'entrée-sortie,
- Structures conditionnelles,
- Structures répétitives.

Qualité des algorithmes

Pour être utilisable, un algorithme doit être

- correct,
- efficace en temps de calcul,
- le moins gourmand possible en mémoire.

la correction d'un algorithme ne peut pas être testée automatiquement.

Compromis temps de calcul / espace mémoire utilisé.

Algorithmique et programmation

- On utilise souvent un pseudo-langage pour présenter les algorithmes.
- Certains langages de programmation ont une syntaxe proche.
- **Exemple** : Python.

Variables et affectation

- Variable = emplacement mémoire
  - où on mémorise une valeur,
  - d'un type donné (entier, chaîne, liste, Booléen).
  - accessible par un nom.

Variables et affectation

- Variable = emplacement mémoire
  - où on mémorise une valeur,
  - d'un type donné (entier, chaîne, liste, Booléen).
  - accessible par un nom.
- Même rôle qu'une mémoire dans une calculatrice.

Variables

- A chaque instant, une variable contient une seule valeur.
- Une variable a un nom qui commence par une lettre.
- Exemple : x, x1, i, resultat.

Variables

- A chaque instant, une variable contient une seule valeur.
- Une variable a un nom qui commence par une lettre.
- Exemple : x, x1, i, resultat.

- Choisir des noms explicites pour les variables importantes.
- Les paramètres d'un algorithme sont aussi vus comme des variables.
- Un algorithme peut utiliser des variables supplémentaires.
- Indiquer clairement les variables utilisées par vos algorithmes.

Types

- Ensemble de valeurs,
- et d'opérations légales sur les éléments du type.

Affectation, syntaxe pseudo-code

- Affectation : mémorisation d'une valeur dans une variable.
- Notation pseudo-code :

  $\langle \texttt{nom\_variable} \rangle \leftarrow \langle \texttt{valeur} \rangle$
- **Exemples**

  $\texttt{resultat} \leftarrow 3$

  $\texttt{resultat} \leftarrow \texttt{resultat} + 1$

  La $2^{\text{ème}}$ affectation suppose que **resultat** contient déjà une valeur.

  Dans ce cas, son effet est d'incrémenter la variable **resultat**.

Affectation, syntaxe Python

- Affecter une valeur à une variable se fait avec le symbole $=$.
- Ce symbole n'a donc pas la même signification qu'en mathématiques.
- `x = 3` signifie : ranger 3 dans la variable x.

  L'ancienne valeur de x est perdue.

Conditionnelles

Pseudo-code

```
si ⟨condition⟩ alors
   ⟨instruction 1⟩
   ⟨instruction 2⟩
   ...
fin si
```

Python

```
if condition:
    instruction1
    instruction2
    ...
```

Note : en python, les blocs sont délimités par l'indentation..

- La condition est évaluée.
- Si elle est vraie, les instructions de la conditionnelle sont exécutées.
- Sinon, on passe à l'instruction suivante.

Conditionnelles

On peut ajouter une partie « sinon ».

Pseudo-code

```
si ⟨condition⟩ alors
    ⟨instruction 1⟩
    ⟨instruction 2⟩
    ...
sinon
    ⟨instruction 3⟩
    ⟨instruction 4⟩
    ...
fin si
```

Python

```
if condition:
    instruction1
    instruction2
    ...
else:
    instruction3
    instruction4
    ...
```

- La condition est évaluée.
- Si elle est vraie, les instructions 1, 2, ... sont exécutées.
- Sinon, , les instructions 3, 4, ... sont exécutées.

Expressions booléennes

Les conditions sont des expressions booléennes.

Pseudo-code	Python
`Vrai`	`True`
`Faux`	`False`

que l'on peut obtenir par des comparaisons

`<`, `>`, `==`, `!=`, `<=`, `>=`

Pour me pas confondre l'affectation et l'égalité, on note l'égalité `==`.

Expressions booléennes

Les booléens se combinent par les opérateurs logiques usuels.

Pseudo-code	Python
`Et`	`and`
`Ou`	`or`
`Non`	`not`

L'évaluation d'une expression Booléenne se fait

- de gauche à droite,
- s'arrête dès le résultat connu.

Pas de risque de division par 0 :

```
if (a != 0) and (b/a != x):
    ...
```

Répétition : boucle « pour tout »

Pseudo-code

pour $n \leftarrow 0$ à k **faire**
  ⟨instruction 1⟩
  ⟨instruction 2⟩
  ...
fin pour

Python

```
for n in range(k+1):
    instruction1
    instruction2
    ...
```

- La variable n prend successivement les valeurs 0, 1, ... k.
- Pour chaque valeur, les instructions sont exécutées

Répétition : boucle « tant que »

Pseudo-code

```
tant que ⟨condition⟩ faire
    ⟨instruction 1⟩
    ⟨instruction 2⟩
    ...
fin tant que
```

Python

```
while condition:
    instruction1
    instruction2
    ...
```

1. La condition est évaluée.
2. Si elle est fausse, on passe à l'instruction qui suit la boucle.
3. Si elle est vraie,
   - Les instructions de la boucle sont exécutées.
   - On revient en 1.

Rupture de boucle

On peut

- sortir d'une boucle depuis le bloc d'instructions :

```
# Sort de la boucle courante
break
```

- Revenir tester la condition sans finir le bloc d'instructions :

```
# Revient tester la condition,
# sans finir le bloc d'instructions
continue
```

Fonctions

En python, les algorithmes sont encapsulés dans des fonctions.

Une fonction est

- d'abord définie,
- puis appelée (utilisée).

Une fonction travaille à partir de paramètres.

```
def delta(a,b,c):
    return b*b - 4*a*c
```

Valeur retournée

Une fonction calcule habituellement une valeur.

On indique la valeur retournée ainsi :

Pseudo-code	Python
`Retourner` ⟨`valeur`⟩	`return resultat`

- L'effet est de faire terminer la fonction.
- Si la fonction est utilisée par un autre algorithme, la valeur retournée peut être utilisée. **Ex : utilisation de la fonction delta :**

```
if delta(a,b,c) < 0:
    ...
```

- Attention : même à l'intérieur d'une boucle, return termine la fonction.