Graphes

Licence 3 MIASHS – parcours info
Université Toulouse Jean Jaurès
2021-2022

Ollivier Haemmerlé
ollivier.haemmerle@univ-tlse2.fr

Plan du cours

1. Introduction à la calculabilité, à la décidabilité, à la complexité
2. Graphes, notions de base
3. Coloration de graphes, Welsh & Powell
4. Arbre couvrant de poids minimal, Kruskal
5. Représenter des connaissances avec des graphes
6. Recherche de plus court chemin, A*
7. Page ranking
8. De l'I.A. dans les jeux

1 Introduction à la calculabilité, à la décidabilité, à la complexité

- Calculabilité
- Machine de Turing
- Décidabilité
- Complexité algorithmique
- Quelques mots sur l'I.A.
- Un peu plus loin en complexité

Calculabilité, décidabilité

- Algorithme
  - description d'un traitement permettant de résoudre un problème
- Peut-on résoudre n'importe quel problème avec un algorithme
  - l'informatique a-t-elle des limites ?

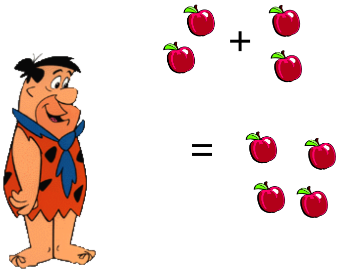
Calcul et calculabilité

- L'homme compte depuis la nuit des temps
  - avec des cailloux (*calculi* en latin)
  - avec ses doigts
    - V et X chez les latins
    - la base 10 chez les arabes

addition

multiplication par additions successives

division par soustractions successives



Cailloutabilité et calculabilité (1)

- Peut-on tout faire avec des cailloux ?

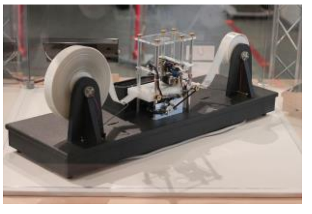
- Quel est l'ensemble des fonctions **f** exprimables en termes mathématiques pour lesquelles il existe une succession de manipulations de symboles déterminant sans ambiguïté l'image de n'importe quelle valeur **x** par **f** ?

- Une fonction est calculable s'il existe une méthode formalisée qui permet de la calculer en temps fini

Cailloutabilité et calculabilité (2)

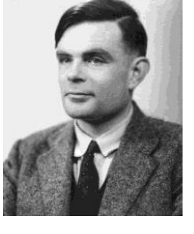
- Ah, ben super… Si je comprends bien ce que tu me dis, une fonction est calculable si elle peut être calculée ? Nous sommes bien avancés !

- Mais si elle peut être calculée par quoi ???



Thèse de Church / Turing (1)

Alonso Church (1903 – 1995)



Alan Turing (1912 – 1954)

- Tout traitement réalisable mécaniquement peut être accompli par une machine de Turing
- La classe des fonctions calculables est la classe des fonctions programmables sur une machine de Turing

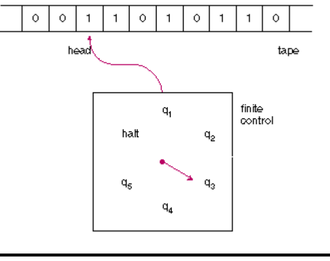
- Modèle de calcul

Thèse de Church / Turing (2)

- Qu'est-ce qui peut être fait par une machine de Turing et à quel coût ?
- **Réponse :**
  - *« Les machines de Turing sont capables de résoudre tout problème algorithmique effectivement solvable ou dit autrement, tout problème algorithmique pour lequel on peut trouver un algorithme qui peut être programmé dans un certain langage de programmation, tout langage, exécutable sur un certain ordinateur, tout ordinateur, même celui qui n'a pas encore été construit, mais qui peut être construit, et même ceux qui nécessitent des durées et des mémoires non limitées, pour des entrées encore plus grandes, est solvable par une machine de Turing. »* [Harel 2004]
- Thèse !!! (pas théorème)
  - solvabilité / calculabilité effective

Machine de Turing

- Un ruban de longueur infinie, divisé en cases (qui peuvent chacune contenir un symbole)
- Une tête de lecture/écriture qui peut
  - lire un symbole
  - écrire un symbole
  - se déplacer d'un cran à droite ou à gauche
  - changer d'état
  - un « programme » ou « table de transition »



Machine de Turing : définition formelle

- MT = (Q, Σ, q_0, δ)
  - Q : ensemble fini d'états
  - Σ : alphabet fini
  - q_0 : élément de Q, état initial
  - δ : programme, ensemble d'instructions
    - (état, symbole lu) → (état, symbole écrit, déplacement)
    - (q, a) → (p, b, G)
    - (q, a) → (p, b, D)
    - (q, a) → (p, b, _)

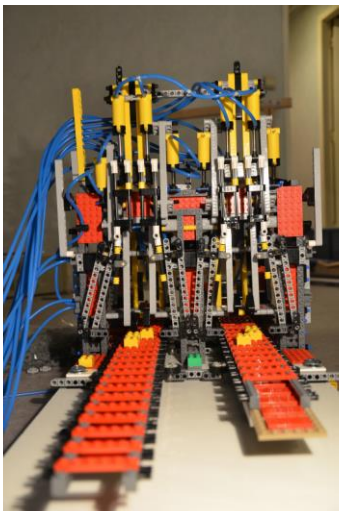
Exemple de machine de Turing

- Réarranger les 0 et les 1
  - sauter tous les '0' jusqu'au premier '1'
    (q0, 0) → (q0, 0, D)
    (q0, 1) → (q1, 1, D)
    (q0, #) → (qm, #, G)
    (qm, 0) → (qSTOP, 0, D)
    (qm, 1) → (qSTOP, 1, D)
    (qm, #) → (qSTOP, #, D)
  - dans l'état q1, parcourir les '1' jusqu'au '0' qui sera changé en '1' (s'il n'y en a pas, succès)
    (q1, 1) → (q1, 1, D)
    (q1, 0) → (q2, 1, G)
    (q1, #) → (qm, #, G)
  - revenir en arrière en sautant les '1' jusqu'au premier '0' pour repartir vers la droite
    (q2, 1) → (q2, 1, G)
    (q2, 0) → (q3, 0, D)
  - changer '1' en '0' puis revenir à l'état initial
    (q3, 1) → (q0, 0, D)

Calculabilité ?

- Jusqu'à nouvel ordre, les fonctions calculables sont les fonctions
  - qui peuvent être calculées par une machine de Turing
  - mais aussi exprimables en lambda-calcul
  - ou représentable par un automate cellulaire…
- Tous ces formalismes sont des « modèles de calcul »
  - ils sont équivalents
  - on peut exprimer les uns en fonction des autres
  - n'importe quelle machine actuelle (et future ?) peut s'y ramener

La machine de Turing réalisée



https://webcast.in2p3.fr/video/la_machine_de_turing_realisee

Qu'est-ce qu'un problème ?

- Problème
  - difficulté à résoudre pour obtenir un résultat
  - difficulté : aucun moyen évident
- Problème de décision
  - être capable de "décider", de répondre oui ou non

Quelques problèmes de décision

- Déterminer si un nombre est pair
- Déterminer si une formule de la logique des propositions est satisfiable
- Déterminer si une expression est syntaxiquement correcte
- Déterminer si un programme va s'arrêter
- Ce sont des classes de problèmes
- Une instance de problème
  - déterminer si 168 est pair

Décidabilité (1)

- Un problème de décision est décidable
  - s'il existe un algorithme qui se termine en un nombre fini d'étapes, qui le décide (répond par oui ou par non)

Décidabilité (2)

- Tous les problèmes sont-ils décidables ?
  - existe-t-il des questions (bien posées) qui ne sont pas solubles algorithmiquement ?
    - si oui, inutile de chercher à les résoudre !
- NON : il existe des problèmes indécidables
  - exemple : problème de la terminaison d'un programme

Terminaison d'un programme (1)

Ne laissez plus boucler vos programmes! Utilisez notre fonction Termine(u). Elle prend comme paramètre le nom de votre procédure et donne pour résultat true si la procédure ne boucle pas indéfiniment et false sinon. En n'utilisant que les procédures pour lesquelles Termine répond true, vous évitez tous les problèmes de non terminaison. D'ailleurs, voici quelques exemples:

```
def F():
        x = 1
        print(x)
```

```
def G():
        x = 1
        while x>0:
                x+=1
```

pour lesquels Termine(F) = True et Termine(G) = False

Terminaison d'un programme (2)

```
def Termine (zeProgramme):
        …
        …
        return bool
```

```
def Absurde():
        while Termine(Absurde):
                pass
```

Suffit-il de savoir qu’un problème est décidable pour être heureux ?

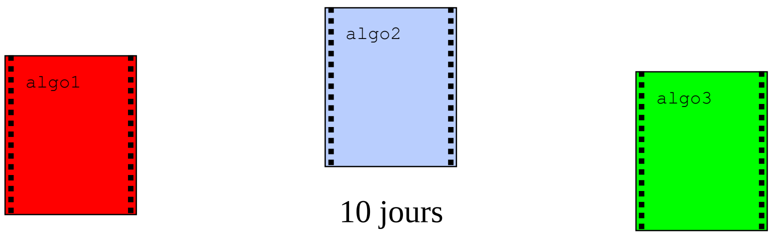
- Un problème soluble qui nécessite 4.10^{16} années de calcul pour traiter 100 données est-il satisfaisant ?

Analyse de la complexité d’un algorithme

- Etude formelle des ressources nécessaires à son fonctionnement
  - espace mémoire utilisé
  - temps d’exécution
- Compromis espace / temps
- Objectifs
  - COMPARER pour comprendre
  - COMPARER pour choisir de manière éclairée
- C’est important dans la vraie vie !

Complexité algorithmique (1)

- Mesurer l'efficacité d'un algorithme
  - comparaison de performances



10 secondes

10 jours

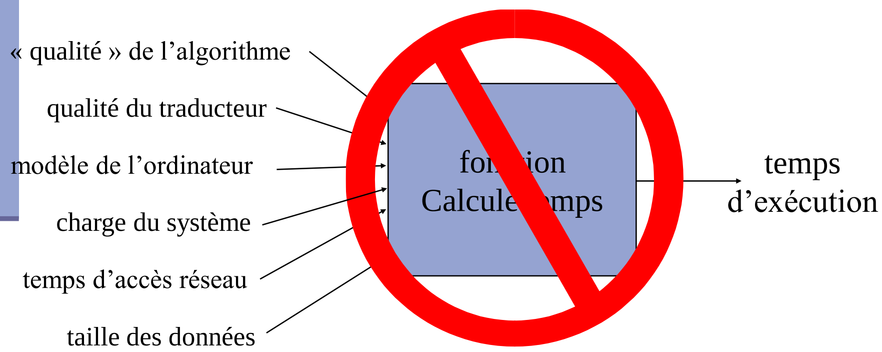
10 siècles

Complexité algorithmique (2)

- Temps d'exécution dépend de
  - qualité intrinsèque (complexité) des algorithmes utilisés
  - qualité du code généré par le compilateur
  - puissance de la machine
  - charge du système
  - temps d'accès réseau
  - taille des données
- Impossible de tout prendre en compte !

Complexité algorithmique (3)

- Une fonction qui calcule l'efficacité d'un algorithme ?



INDECIDABLE

Complexité algorithmique (4)

- Temps d'exécution dépend généralement plus de la taille des données que de leur nature
  - **temps(Σ(1,2,3,4,5)) = temps(Σ(10,20,30,40,50)**
  - **temps(Σ(1,2,3,4,5)) < temps(Σ(1,2,3,4,5,6,7,8,9,10)**
- **T(n)** : temps d'exécution (en secondes) d'un programme en fonction de la taille **n** des données
  - **T(n) = c.n²** avec **c** constante

Complexité algorithmique (5)

- On parlera plutôt de complexité algorithmique
  - notion abstraite, plus simple à manipuler
  - complexité dans le pire des cas
  - complexité en moyenne
- Unité de complexité ???
  - unité de temps impossible à utiliser

« cet algorithme de tri fonctionne en **$0{,}05.n^2$** secondes pour une liste de **n** éléments »

« la complexité de cet algorithme est de l'ordre de **n^2** »

Complexité algorithmique (6)

- Notation asymptotique **O**
  - **T(n)** est en **O(f(n))** si **T(n)** est borné par **c.f(n)** quand **n** tend vers l'infini
  - **$T(n)=4n^3+2n^2$** appartient à **$O(n^3)$**
  - **T(n)** appartient à **$O(n^4)$**
    - peu intéressant en termes de comparaisons
  - **T(n)** n'appartient pas à **$O(n^2)$**
    - **$4n^3+2n^2$** « finira toujours » par être supérieur à **$c.n^2$** quel que soit **c**

Complexité algorithmique (7)

- Complexités principales
  - constante : **O(1)**
    - ajout d'un élément à une liste
  - sub-linéaires / logarithmique : **$O(\log_2(n))$**
    - recherche d'un élément dans un ensemble ordonné
  - linéaires : **O(n)**
    - parcours d'un ensemble de n éléments
  - linéarithmique : **$O(n.\log_2(n))$**
    - tris optimaux
  - polynomiales : **$O(n^2)$** à **$O(n^3)$**
    - multiplications de matrices
  - polynomiales : **$O(n^k)$** avec **$k > 3$**
  - exponentielles : **$O(2^n)$**
    - énumération des sous-ensembles d'un ensemble

Complexité algorithmique (8)

- Temps d'exécution en fonction de la taille des données

	1	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n
$n=10^2$	1 µs	6,6 µs	0,1 ms	0,6 ms	10 ms	1 s	4.10^{16} a
$n=10^3$	1 µs	9,9 µs	1 ms	9,9 ms	1 s	16,6 mn	∞
$n=10^4$	1 µs	13,3 µs	10 ms	0,1 s	100 s	11,5 j	∞
$n=10^5$	1 µs	16,6 µs	0,1 s	1,6 s	2,7 h	31,7 a	∞
$n=10^6$	1 µs	19,9 µs	1 s	19,9 s	11,5 j	$31,7.10^3$ a	∞

- Taille max des données en fonction du temps disponible

	1	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n
1 s	∞	∞	10^6	63.10^3	10^3	100	19
1 mn	∞	∞	6.10^7	28.10^5	77.10^2	390	25
1 h	∞	∞	36.10^8	13.10^7	60.10^3	15.10^2	31
1 j	∞	∞	86.10^9	27.10^8	29.10^4	44.10^2	36

Quelques mots sur l'I.A. (1)

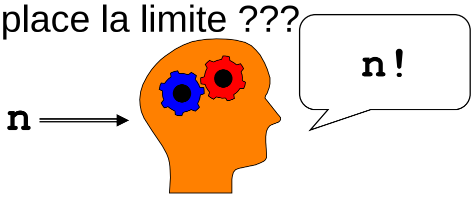
- Intelligence : « faculté de connaître, de comprendre »
- Artificiel : « ce qui est le produit de l'habileté humaine et non celui de la nature »

Quelques mots sur l'I.A. (2)

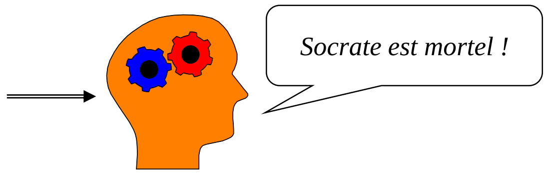
- L'informatique est la science du traitement de l'information
  - représentation de l'information variée
    - types des variables
    - fichiers textes
    - images, sons, films…
  - traitement à l'aide d'algorithmes
    - programmés en différents langages
- Des **0** et des **1** !

Quelques mots sur l'I.A. (3)

- Où se place la limite ???



« Les hommes sont mortels. »
« Socrate est un homme. »

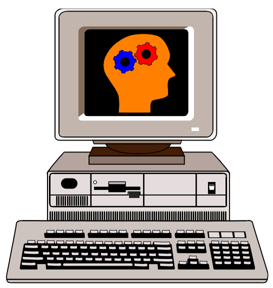


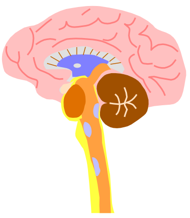
Quelques mots sur l'I.A. (4)

- Jacques Pitrat (1934-2019) : *« l'Intelligence Artificielle est la science dont le but est de faire faire par une machine tout ce que l'homme est capable de faire »*

Quelques mots sur l’I.A. (5)

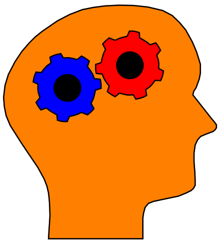
- Eugene Charniak et Drew Mc Dermott :
  « l’étude des facultés mentales à travers l’utilisation de modèles calculables »



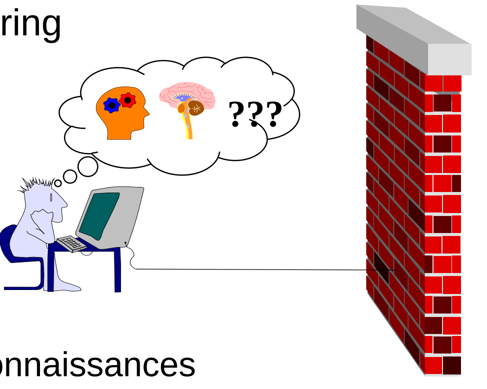
Quelques mots sur l’I.A. (6)

- Jean-Louis Laurière (1945-2005) :
  « tout problème pour lequel aucune solution algorithmique n’est connue relève a priori *de l’Intelligence Artificielle »*

Quelques mots sur l'I.A. (7)

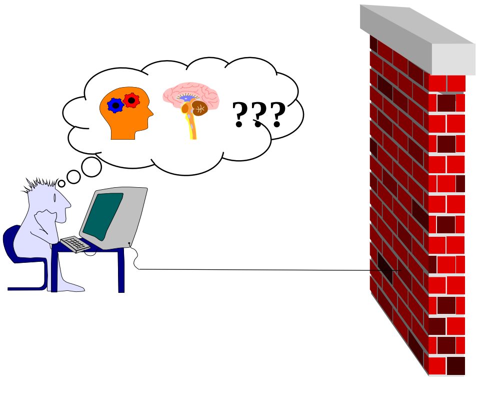
- 1950 : le test de Turing

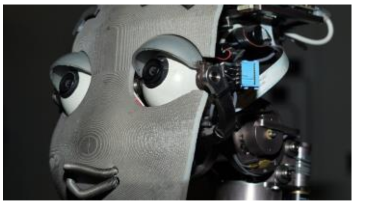


  - langue naturelle
  - représentation de connaissances
  - raisonnement automatique
  - apprentissage automatique

Quelques mots sur l'I.A. (8)

- *« Any AI smart enough to pass a Turing test is smart enough to know to fail it. » [Ian McDonald]*





Un peu plus loin en complexité (1)

- Complexité d'un algorithme
  - insertion en fin de liste en **O(n)**, en **O(1)**
  - tris en **$O(n^2)$**, en **$O(n.\log_2(n))$**
- Complexité d'un problème
  - complexité du meilleur algorithme permettant de résoudre le problème
- Classement des problèmes
  - dans des grandes classes de complexité
  - certains problèmes sont non classés

Un peu plus loin en complexité (2)

- La classe **P**
  - il existe un algorithme de résolution de complexité égale à un polynôme de degré constant
  - la plupart des algorithmes que nous avons rencontrés
    - insertion, suppression, recherche dans liste
    - insertion, suppression, recherche dans arbre binaire
    - tris

Un peu plus loin en complexité (3)

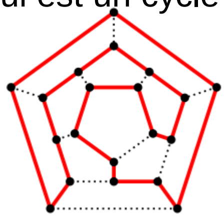
- La classe **E**
  - les problèmes exponentiels par nature
  - énumération d'un nombre exponentiel de sorties
  - exemple : construction de tous les sous-ensembles d'un ensemble
    - **2^n** ensembles à construire
    - au minimum **2^n** itérations
    - temps exponentiel

Un peu plus loin en complexité (4)

- La classe **NP**
  - de nombreux problèmes appartiennent à **NP**
    - la construction d'emplois du temps, respectant des contraintes de salles et de disponibilité d'enseignants, par exemple
    - le chargement d'un conteneur (sac à dos, train, bateau…)
    - la découpe de pièces dans de la matière (découpe de tissus, de métal…)
    - le voyageur de commerce
  - Nombreux problèmes classiques d'I.A. !

Un peu plus loin en complexité (5)

- Recherche d'un cycle hamiltonien
  - un chemin hamiltonien d'un graphe est un chemin qui passe par tous les sommets une fois et une seule
  - un cycle hamiltonien est un chemin hamiltonien qui est un cycle (la boucle est bouclée)



Un peu plus loin en complexité (6)

- Recherche de cycle Hamiltonien

```
fonction hamilton(e : in ensemble ; s : in sommet) : liste
début
      reste ← e   //{s1, s2, … sn}
      courant ← s // courant prend le sommet de départ
      cycle ← (s) // le premier sommet du cycle est s
      tant que (reste ≠ ∅) et (courant.vois ∩ reste ≠ ∅) faire
            courant ← choix(courant.vois ∩ reste)
            cycle ← concat(cycle, courant)
            reste ← reste - courant
      fintantque
      si reste = ∅ et s ∈ courant.vois
      alors
            retourner concat(cycle, s)
      sinon
            retourner échec
      finsi
fin
```

Un peu plus loin en complexité (7)

- **NP** signifie non déterministe polynomial
  - pas d'algorithme résolvant le problème en temps polynomial sur une machine déterministe
  - MAIS algorithme sur machine non déterministe PEUT résoudre le problème en temps polynomial
  - Solution peut être vérifiée en temps polynomial
- Machine non déterministe
  - fonction `choix(E)` choisit une valeur de l'ensemble **E** de manière non déterministe

Un peu plus loin en complexité (8)

- Machine non déterministe, autre vision
  - la fonction `choix` entraîne un `fork()` sur une machine multi-processeurs
  - nombre de processeurs suffisant
  - un des processus résoudra le problème en temps polynomial

Un peu plus loin en complexité (9)

- **P** $\neq$ **NP** ???
- **P** $\subseteq$ **NP** ?
  - naturel
- **NP** $\subseteq$ **P**, **NP** $\not\subset$ **P** ???
  - non prouvé
  - **P** $\neq$ **NP** est une conjecture
- Une des énigmes de l'informatique

NP-complétude

- Un problème NP-complet
  - appartient à la classe **NP**
  - tout problème de la classe **NP** s'y ramène par une réduction polynomiale
- Réduction polynomiale
  - permet de traduire qu'un problème n'est pas « plus dur » qu'un autre
  - si un premier problème se réduit « facilement » en un second, le premier problème est aussi « facile » que le second

Problèmes NP-complets

- SAT (satisfiabilité) et sa variante 3-SAT
  - étant donnée une formule de logique propositionnelle, détermine s'il existe une assignation des variables propositionnelles qui rend la formule vraie
- Recherche d'un cycle hamiltonien
  - et sa variante pondérée, le problème du voyageur de commerce
- Problèmes de coloration de graphe
- Problème du sac à dos [knapsack problem]
  - optimisation combinatoire
  - remplissage d'un sac à dos, en respectant des contraintes
  - avec tout ou partie d'un ensemble donné d'objets ayant chacun un poids et une valeur
  - les objets mis dans le sac à dos doivent maximiser la valeur totale
  - sans dépasser le poids maximum

Le modèle Turing

https://interstices.info/wp-content/uploads/jalios/docs/video/mpeg/2014-11/inria758_modele_turing.mp4