Calculabilité, Complexité et Algorithmique

Lhouari Nourine
Université Blaise Pascal, CNRS, LIMOS

Janvier, 2013 – Fès maroc

C'est quoi?

Quels sont les problèmes qu'une machine peut résoudre?

- Trier un tableau ayant une taille fixée
- Colorier un graphe
- Vérifier si un programme en C++ est syntaxiquement correct
- Vérifier si un programme en C++ est correct !
- Vérifier si un programme C++ s'arrête indépendamment de l'entrée!

➪ Plusieurs problèmes importants en informatique ne sont résolvables ou traitables sur des machines mécaniques

Quelques obstacles

1. Taille des ensembles considérés
   - ⇨ $\mathbb{N}$
   - ⇨ Le nombre de programmes C++ est infini

2. Peut-on les parcourir?
   - ⇨ Afficher $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$.
   - ⇨ Afficher $\mathbb{R}$
   - ⇨ Afficher tous les programmes C++

Dans quels domaines se trouvent ces difficultés

⇨ Génie Logiciel (Vérification, génération de tests,...)

⇨ Systèmes à transitions (Composition de web services, Artifacts, Modèle orienté données)

⇨ Logique

⇨ Optimisation combinatoire

Plan du cours

- ⇨ Ensembles dénombrables (Récursivement Enumérables)
- ⇨ Modèles du calcul
- ⇨ Notion d'algorithme (décidabilité)
- ⇨ Complexité d'un algorithme
- ⇨ Complexité d'un problème

Ensembles dénombrables

➪ Deux ensembles A et B ont la même cardinalité, notée $A \backsimeq B$ s'il y a une bijection entre A et B.

Quelques exemples :

- $\mathbb{N}_{pair} \backsimeq \mathbb{N}_{impair}$. La bijection est :

$$f(n) = \frac{n}{2} \tag{1}$$

- $\mathbb{N} \backsimeq \mathbb{N}_{pair}$. La bijection est :

$$f(n) = 2 * n \tag{2}$$

- $\mathbb{N} \backsimeq \mathbb{Z}$. La bijection est :

$$f(n) = \begin{cases} \frac{n}{2} & \text{si } n \text{ est pair} \\ -\frac{n+1}{2} & \text{si } n \text{ est impair} \end{cases} \tag{3}$$

Ensembles dénombrables

Definition

Un ensemble A est dénombrable s'il est fini ou $A \preceq \mathbb{N}$; sinon il est dit indénombrable.

- $\mathbb{N}$ est dénombrable (puisque $\mathbb{N} \preceq \mathbb{N}$).
- $\mathbb{Z}$ est dénombrable (puisque $\mathbb{N} \preceq \mathbb{Z}$).

Algorithme 1 : Afficher les entiers

---

début

  $i = 0$;

  tant que *1* **faire**

    Afficher (i);

    $i = i + 1$;

fin

Ensembles dénombrables

Theorem

Un ensemble A et l'ensemble de ses parties $2^A = \{B \mid B \subseteq A\}$ n'ont pas la même cardinalité, i.e. $A \not\simeq 2^A$.

Proof.

Supposons qu'il y une bijection $f : A \to 2^A$.
On définit un ensemble C tel que $C = f(a)$ est violée. Soit $C = \{b \in A \mid b \notin f(b)\}$.
C est une image de f. Supposons que $C = f(a)$. Alors $a \in C$ ssi (par definition de C) $a \notin f(a)$ ssi (par $C = f(a)$) $a \notin C$. □

➫ Diagonalisation

Ensembles dénombrables

- $2^{\mathbb{N}}$ n'est pas dénombrable. Puisque $2^{\mathbb{N}}$ n'est pas fini et $\mathbb{N} \not\simeq 2^{\mathbb{N}}$.
- $\mathbb{R} \simeq 2^{\mathbb{N}}$
- $\mathbb{R} \simeq [0, 1]$

Questions

- Pouquoi la preuve par récurrence marche!!

- Peut-on toujours faire une preuve par induction?

Construction des ensembles

⇨ Comment construire des ensembles? Construction inductive
Eléments de base + règles de construction

⇨ Exemple de la construction de $\mathbb{N}$:

- Base : $0 \in \mathbb{N}$
- Règle : Si $n \in \mathbb{N}$ alors $n + 1 \in \mathbb{N}$

Construction inductive

Soit $\mathcal{L}$ l'ensemble des langages réguliers sur un alphabet Σ.

1. $\epsilon \in \mathcal{L}$
2. $\emptyset \in \mathcal{L}$
3. Si $a \in \Sigma$ alors $a \in \mathcal{L}$
4. Si $L_1, L_2 \in \mathcal{L}$ alors $L_1.L_2 \in \mathcal{L}$
5. Si $L_1, L_2 \in \mathcal{L}$ alors $L_1 \cup L_2 \in \mathcal{L}$
6. Si $L \in \mathcal{L}$ alors $L^* \in \mathcal{L}$
7. Tout langage de $\mathcal{L}$ est obtenu par un nombre fini d'applications des règles précédentes.

Construction inductive

Soit $\mathcal{A}$ l'ensemble des arbres binaires.

1. NIL$\in \mathcal{A}$
2. Si $A_1, A_2 \in \mathcal{A}$ et x un nouveau sommet alors l'arbre obtenu en mettant A_1 et A_2 comme fils de la racine x appartient à $\mathcal{A}$.
3. Tout arbre de $\mathcal{A}$ est obtenu par un nombre fini d'applications des règles 1. et 2.

Construction inductive

Preuve d'une propriété sur un ensemble construit inductivement :

1. Prouver la propriété pour les éléments de base.
2. Prouver que les règles de construction préservent la propriété.

Questions

- C'est quoi calculer?

- C'est quoi programmer?

Modèles du calcul

Un modèle du calcul est un ensemble de fonctions de base et des règles de construction pour définir d'autres fonctions plus complexes.

Par exemple :

1. Automate d'états fini (Pile, file, arbre,...)
2. Petri nets
3. Machine de Turing (mémoire infini)
4. RAM
5. Fonctions récursives
6. .....

Modèles du calcul : Fonctions récursives

Definition

L'ensemble des fonctions primitives recursives est défini par :

- Les fonctions de base sont:
  - $zero : \mathbb{N}^0 \to \mathbb{N}$, avec $zero() = 0$.
  - $succ : \mathbb{N} \to \mathbb{N}$, avec $succ(n) = n + 1$.
  - $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$, avec $1 \leq i \leq k$ et $\pi_i^k(a_1, ..., a_k) = a_i$, $(a_1, ..., a_k) \in \mathbb{N}^k$.
- Règles de Construction.
  - Composition : Soient les fonctions primitives récursives $g : \mathbb{N}^m \to \mathbb{N}^n$ et $f_i : \mathbb{N}^k \to \mathbb{N}$, pour $i \in [1, m]$. Alors la fonction $g(f_1(x_1, ..., x_k), ..., f_m(x_1, ..., x_k))$ est récursive primitive.
  - Récursion primitive : Soient les fonctions récursives primitives $g : \mathbb{N}^k \to \mathbb{N}^m$ et $h : \mathbb{N}^{k+m+1} \to \mathbb{N}^m$, $k, n, m \in \mathbb{N}$. Alors la fonction $f : \mathbb{N}^{k+1} \to \mathbb{N}^m$ définie par :

$$f(\overrightarrow{x}, 0) = g(\overrightarrow{x})$$

$$f(\overrightarrow{x}, y + 1) = h(\overrightarrow{x}, y, f(\overrightarrow{x}, y))$$

Modéles du calcul : Fonctions récursives

Soit $add : \mathbb{N}^2 \to \mathbb{N}$ avec $add(x, y) = x + y$.

$$add(x, 0) = x = \pi_1^2(x, 0)$$

$$add(x, y + 1) = (x + y) + 1 = succ\ o\ \pi_3^3(x, y, add(x, y))$$

Algorithme 2 : $add(x, y)$

début
  $r = x$;
  pour $i = 0$ *à* $y - 1$ **faire**
    $r = succ(r)$;
fin

Modéles du calcul : Fonctions récursives

Soit $moins : \mathbb{N}^2 \to \mathbb{N}$ avec $moins(x, y) = x - y$ si $x > y$ et 0 sinon.

$$moins(x, 0) = x = \pi_1^2(x, 0)$$

$$moins(x, y + 1) = moins(x, y) - 1 =$$

$$pred\ o\ \pi_3^3(x, y, moins(x, y))$$

Algorithme 3 : $moins(x, y)$

début
  $r = x$;
  pour $i = 0$ *à* $y - 1$ **faire**
    $r = pred(r)$;
fin

Modéles du calcul : Fonctions récursives

Est-ce-que toutes les fonction calculables sont primitives récursives?

Modéles du calcul : Fonctions récursives

Une autre règle de construction.

- minimisation : Soit $g : \mathbb{N}^{n+1} \to \mathbb{N}$ une fonction récursive. Alors la fonction $f : \mathbb{N}^n \to \mathbb{N}$ définie par :

$$f(\overrightarrow{x}) = \mu y[g(\overrightarrow{x}, y) = 0]$$

est récursive.

⇨ $f(\overrightarrow{x})$ est le plus petit y pour lequel $g(\overrightarrow{x}, y) = 0$ et $g(\overrightarrow{x}, z)$ est définie pour tout $z < y$.

Algorithme 4 : $f(x_1, ..., x_k)$

---

début

  $y = 0$;

  tant que $g(x_1, ..., x_k, y) \neq 0$ **faire**

    $y = y + 1$;

  Retourner(y);

fin

Modéles du calcul : Fonctions récursives

1. $racine(x) = \mu y[(y+1)^2 > x)]$
2. $$\mu y[y^2 = x) = \begin{cases} \sqrt{(x)} & \text{si } x \text{ est un carré parfait} \\ indefini & \text{sinon} \end{cases} \quad (4)$$
3. $$\mu y[2y = x] = \begin{cases} \frac{x}{2} & \text{si } x \text{ est pair} \\ indefini & \text{sinon} \end{cases} \quad (5)$$

Modéles du calcul : Question

➮ Que peut-on programmer si on enlève Tant que et Répéter de C++?

Modèles du calcul : Machine de Turing

Une machine de Turing déterministe (MT) est un 7-tuplé $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- Q un ensemble fini d'états.
- Γ est un ensemble fini de symboles, appelé *alphabet* du travail.
- $B \in \Gamma$, un symbole spécial blanc associé à une case vide.
- $\Sigma \subseteq \Gamma - \{B\}$ est l'ensemble des symboles avec lesquels les entrées sont exprimées.
- $q_0 \in S$, l'état initial.
- $F \subseteq Q$ est l'ensemble des états finaux.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ est la fonction de transition.

Modèles du calcul : Machine de Turing

Opération de base d'un MT. Soit la transition $\delta(q, a) = (q', a', D)$:

- Si la MT est dans l'état q, et le symbole au dessous de la tête de lecture est a, alors
  - L'état est changé par q',
  - le symbole en cette position est changé par a',
  - Si $D = \rightarrow$, la tête se déplace à droite d'une position,
  - Si $D = \leftarrow$, la tête se déplace à gauche d'une position,

- Une machine de Turing M accepte un mot w écrit dans son alphabet d'entrée Σ ssi M s'arrête dans un état final.

Modéles du calcul : Machine de Turing

➮ Une configuration de la MT peut être décrite par une chaîne de caractères :

$$X_1X_2....X_{i-1}qX_i....Xn$$

q est l'état actuel et la tête lit le ième caractère.

➮ Supposons que $\delta(q, X_i) = (p, Y, \leftarrow)$ alors la nouvelle configuration est

$$X_1X_2....X_{i-1}qX_i....Xn \vdash X_1X_2....X_{i-2}qX_{i-1}YX_{i+1}....Xn$$

➮ Il ya deux exceptions pour le déplacement à gauche:

1. Si $i = 1$: $qX_1X_2....Xn \vdash qBYX_2....Xn$
2. Si $i = n$ et $Y = B$: $X_1X_2....X_{n-1}qXn \vdash X_1X_2....Xn - 1qX_{n-1}$

Modèles du calcul : Machine de Turing

Soit la machine de Turing

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$

$Q \times \Gamma$	0	1	X	Y	B
q_0	$(q_1, X, \rightarrow)$	-	-	$(q_3, Y, \rightarrow)$	-
q_1	$(q_1, 0, \rightarrow)$	$(q_2, Y, \leftarrow)$	-	$(q_1, Y, \rightarrow)$	-
q_2	$(q_2, 0, \leftarrow)$	-	$(q_0, X, \rightarrow)$	$(q_2, Y, \leftarrow)$	-
q_3	-	-	-	$(q_3, Y, \rightarrow)$	$(q_4, B, \rightarrow)$
q_4	-	-	-	-	-

La séquence complète de déplacements de M pour le mot 0011 est :

$q_0 0011 \vdash X q_1 011 \vdash X 0 q_1 11 \vdash X q_2 0 Y 1 \vdash q_2 X 0 Y 1 \vdash X q_0 0 Y 1 \vdash$
$X X q_1 Y 1 \vdash X X Y q_1 1 \vdash X X q_2 Y Y \vdash X q_2 X Y Y \vdash X X q_0 Y Y \vdash$
$X X Y q_3 Y \vdash X X Y Y q_3 B \vdash X X Y Y B q_4 B$

Modèles du calcul : Machine de Turing

➫ Langage accepté par une MT

Soit $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ une MT.

On note par $L(M) = \{w \in \Sigma^* \mid q_0 w \vdash^* \alpha p \beta, p \in F, \alpha, \beta \in \Gamma^*\}$.

L'ensemble des langages qu'on peut accepter par une MT est appelé les langages récursivement énumérables (RE).

Modèles du calcul : Machine de Turing

- Un langage $L \subseteq \Sigma^*$ est semi-calculable s'il existe une machine de Turing M telle que l'ensemble des mots sur Σ acceptés est exactement L.

  ⇨ Semi-algorithme ou Semi-décidable ou Récursivement énumérable

- L est calculable si de plus M s'arrête sur tous les mots $w \in \Sigma^*$.

  ⇨ Algorithme ou Décidable ou Récursif

Modèles du calcul : Machine de Turing non déterministe

- La différence entre une MT non-déterministe (NMT) et une MT déterministe est la fonction de transition : $\delta(q, X) = \{(q_1, Y_1, D_1), ..., (q_k, Y_k, D_k)\}$.

- Une NMT accepte une donnée w s'il existe une séquence de choix de déplacements qui mène d'un état initial à un état final.

- Si M_N est une NMT alors il existe une MT M_D tel que $L(M_N) = L(M_D)$.

Complexité d'un algorithme

Definition

Un algorithme est une séquence d'opérations élémentaires fini, s'arrête en un temps fini et qui fournit un résultat répondant à un problème donné.

⇨ Une opération élémentaire est

- une transition dans la MT.
- une opération arithmitique ou logique,... dans le modèle RAM.

Complexité d'un algorithme

➪ La notation O permet une analyse sans tenir compte de facteurs constants.
➪ Les fonctions considérées sont du type : $g(n) : N \rightarrow N$.

- $g(n) = O(f(n))$, s'il existe deux constantes strictement positives c et n_0 telles que $cf(n)$ est une borne supérieure de $g(n)$ pour tout $n > n_0$ (ie. $g(n) \leq cf(n), \forall\ n > n_0$).
- $g(n) = \Omega(f(n))$, s'il existe deux constantes strictement positives c et n_0 telles que $cf(n)$ est une borne inférieure de $g(n)$pour tout $n > n_0$ (ie. $g(n) \geq cf(n), \forall\ n > n_0$).
- $g(n) = \Theta(f(n))$ si $g(n) = O(f(n))$ et $g(n) = \Omega(f(n))$.

Complexité d'un algorithme

Soient A un algorithme ayant une entrée de taille n et $g(n)$ son temps d'exécution. ALors A est dit

- logarithmique si $g(n) \in O(logn)$.
- linéaire si $g(n) \in O(n)$.
- quadratique si $g(n) \in O(n^2)$.
- plynomial si $g(n) \in O(n^k)$, k une constante
- exponentiel si $g(n) \in O(2^n)$..

Complexité d'un problème

La théorie de complexité considère que les problèmes de décision.

Probleme (CHEMIN)

Instance : *Soient $G = (X, E)$ un graphe orienté, $x, y \in X$, k un entier;*
Question : *Existe-t-il un chemin de x à y de longueur au plus k?*

Probleme (HAM)

Instance : *Soit $G = (X, E)$ un graphe non orienté;*
Question : *G est-il un hamiltonien?*

Complexité d'un problème

Definition

Un problème de décision Q est une fonction de $\mathcal{I}_Q$ vers $\{0,1\}$, avec $\mathcal{I}_Q$ l'ensemble des instances de Q.

- Une instance pour le problème CHEMIN est $< G, x, y, k >$.
- Langage CHEMIN
  $L(CHEMIN) = \{< G, x, y, k > \mid G = (X, E)$ est un graphe orienté, $x, y \in X$, k est un entier, et il existe un chemin de x à y de longueur au plus $k\}$.

Complexité d'un problème

⇨ Pourquoi les problèmes de décision alors qu'en pratique sont des problèmes d'optimisation?

- Si un problème d'optimisation est facile alors le problème de décision associé est aussi facile.
- Si un problème de décision est difficile alors le problème d'optimisation associé est difficile.

⇨ La complexité étudie la difficulté des problèmes

Complexité d'un problème

Definition

La classe P est l'ensemble des problèmes de décision qu'on peut résoudre par un algorithme déterministe en temps polynomial.

Complexité d'un problème

Definition

La classe NP est l'ensemble des problèmes de décision qu'on peut résoudre par un algorithme non-déterministe en temps polynomial.

Definition

La classe NP est la classe des problèmes de décision qui peuvent être vérifiés par un algorithme déterministe polynomial.

Complexité d'un problème

⇨ $P \subseteq NP$.

⇨ $NP \subseteq P$ est toujours ouvert.

⇨ Tout problème de décision dans NP, peut être résolu par un algorithme déterministe exponentiel.

⇨ Hypothèse : $P \neq NP$

Complexité d'un problème

Une transformation polynomiale d'un problème de décision Q_1 en un problème de décision Q_2, notée $Q_1 <<_p Q_2$, est une fonction $f : \mathcal{I}_{Q_1} \to \mathcal{I}_{Q_2}$, vérifiant les deux propriétés suivantes :

1. f est calculable en un temps polynomial.
2. Pour tout $i \in \mathcal{I}_{Q_1}$, $i \in L(Q_1)$ ssi $f(i) \in L(Q_2)$.

⇨ La relation $<<_p$ est transitive.

⇨ Quelle est l'intérêt de la transformation polynomial?

Complexité d'un problème

- Si $Q_1 <<_p Q_2$ alors $Q_2 \in P$ implique $Q_1 \in P$.
- Si $Q_1 <<_p Q_2$ alors $Q_1 \notin P$ implique $Q_2 \notin P$.

Algorithme 5 : $A_{Q_1}(i \in \mathcal{I}_{Q_1})$

début

$i' = f(i)$;

$A_{Q_2}(i')$;

fin

Complexité d'un problème

⇨ Deux problèmes de décision Q et Q' sont dits équivalents si $Q <<_p Q'$ et $Q' <<_p Q$.

Probleme (STABLE-MAX)

Instance : *Soient $G = (X, E)$ un graphe non orienté et k un entier;*
Question : *G contient-il un stable de taille au moins k ?*

Probleme (CLIQUE-MAX)

Instance : *Soient $G = (X, E)$ un graphe non orienté et k un entier;*
Question : *G contient-il une clique de taille au moins k ?*

⇨ STABLE-MAX et CLIQUE-MAX sont équivalents.

Complexité d'un problème

Definition

Un problème Q est dit NP-complet si :

1. $Q \in NP$.
2. Pour tout $Q' \in NP$, $Q' <<_p Q$.

Q est dit NP-difficile s'il satisfait la seconde condition.

➪ La classe NP-complets est donc composée des problèmes difficiles de la classe NP.

Complexité d'un problème

Property

Un problème NP-complet Q est dit NP-complet si :

1. $Q \in NP$.
2. $Q' << Q$ *avec* Q' *un problème NP-complet.*

⇨ Quel est le premier problème NP-complet?

Theorem

SAT et 3-SAT sont NP-complets.

Complexité d'un problème

Theorem

STABLE − MAX et CLIQUE − MAX sont NP-complets.

Complexité d'un problème

Probleme (ISO-SOUS-GRAPHE)

Instance : *Soient $G = (X, E)$ et $H = (Y, F)$ deux graphes non orientés;*
Question : *Existe-t-il une application $\phi : Y \to X$ telle que $(y, y') \in F$ ssi $(\phi(y), \phi(y')) \in E$;*

Probleme (LONG-CHEMIN)

Instance : *Soient $G = (X, E)$ et $k \leq |V|$ un entier positif;*
Question : *G contient-il un chemin élémentaire ayant au moins k arêtes. ?*

Complexité d'un problème

⇨ Hypothèse : P=NP, P versus NP.

- On dispose d'un algorithme déterministe (oracle) polynomial pour tout problème de NP.

⇨ Peut-on trouver un algorithme déterministe polynomial pour la version optimisation?

Complexité d'un problème

⇨ Exemples.

- CLIQUE-MAX.
- LONG-CHEMIN
- SAT

Ce qu'il faut retenir

➪ La calculabilité est de savoir s'il existe un algorithme pour rédoudre un problème.

➪ La complexité est classer les problèmes suivant la difficulté de résolution

➪ Pourqoui on s'intéresse aux problèmes de décision?

Dans quels domaines se trouvent ces problèmes

- Génie Logiciel (Vérification, Tests,...)
- Web services
- Logique.
- Optimisation

Y-a-t-il encore des problèmes intéressants et ouverts

⇨ La plupart des instances en pratique ne sont pas difficiles (Solveur SAT, Cplex,...)
⇨ Intégration, data-exchange, Privacy, Base de données incomplètes
⇨ .....