Complexité

D.E ZEGOUR
École Supérieure d'Informatique
ESI

Complexité

Sommaire

Introduction à la théorie de la complexité

Problèmes de décision

Machine de Turing

Complexité / Introduction

Introduction générale

Comment une solution est complexe, difficile ?

Plusieurs manières de mesurer la complexité d'une solution :

- Facile / difficile à l' exprimer
- Facile / difficile à traiter (exécuter) en temps /espace

Autres mesures importantes : Energie, Communications

L'étude de la complexité a pour but de quantifier ses mesures.

Complexité / Introduction

Introduction générale

L'étude de la complexité a débuté avec l'invention des ordinateurs commerciaux (1950 à 1960)

- processeurs étaient lents et les mémoires beaucoup plus chères qu'aujourd'hui
- il était très important de concevoir des algorithmes efficaces.

De nos jours, reste un problème d'actualité et très ouvert.

Discipline très importante (Domaine en informatique : conception et analyse des algorithmes (proposé par Knuth))

Complexité / Introduction

Introduction générale

Point de départ : formalisation du concept d'algorithme.

Complexité étudiée a travers un modèle de calcul :Machine de Turing
(Autres modèles : Machines à registres, Circuit logique, Fonctions récursives, etc)

Les machines de Turing sont à l'origine de la théorie de la complexité.

Les machines de Turing classent un problème décidable en fonction des ressources utilisées (espace et temps)
(différence essentielle avec les autres modèles)

Complexité / Introduction

Principe de Church-Turing

"Tout ce qui peut être calculé, peut l'être par une machine de Turing"

[Simple affirmation (jamais mise en défaut)]

Complexité / Introduction

Introduction générale

Problèmes centraux étudiés dans la théorie de la complexité :

• Pour un type de ressource (Espace , Temps), quels sont les problèmes qu'on peut résoudre et ceux qu'on ne peut pas résoudre?

• Quelle est le lien entre les différents type de ressources ? Avec plus de temps consommé, peut-on réduire l'espace ou vice versa ?

Complexité / Introduction

Introduction générale

Idée de Turing : *démontrer rigoureusement que plusieurs problèmes fondamentaux de la logique ne peuvent pas être résolu par des algorithmes quand les ressources sont limitées.*

Cependant, beaucoup de problèmes insolubles dans plusieurs domaines peuvent avoir des bénéfices pratiques.

Dans ce cas les heuristiques sont utilisées.

Alan Turing : Mathématicien anglais
* Célèbre pour sa machine (conçue en 1936)
- Célèbre pour avoir décodé les messages des nazis pendant la seconde guerre mondiale
- Logo de la firme Apple

Complexité / Introduction

Problèmes solubles / Problèmes non solubles

Il ya des problèmes pour lesquels on connait des réponses:

- Etant donné un polynôme P quelconque, ayant un nombre quelconque de variables, est-ce que P admet des racines entières ? (Dixième problème de Hilbert)

-> La réponse est NON.

- Etant donné une formule F quelconque de la logique propositionnelle, est-ce que F est satisfiable (Il existe une combinaison de valeurs pour les variables propositionnelles de F pour laquelle F est vraie) ?

-> La réponse est OUI.

- Même question que ci-dessus, mais avec les formules de la logique du premier ordre.

-> La réponse est NON.

Complexité / Introduction

Problèmes solubles / Problèmes non solubles

Il y a des problèmes pour lesquels on ne connait pas de réponses.

Problème de Fermat (il y a 3 siècles) restait toujours d'actualité jusqu'à 1994

En entrée : n

En sortie les triplets (x, y, z) entiers tels que $x^n + y^n = z^n$

Si n=2, on a une solution x= 3, y= 4 , et z=5

Si n>2, on ne sait pas s'il y a des solutions entières.

(Démontré par le mathématicien britannique Andrew Wiles)

Théorème : Il n'existe pas de nombres entiers strictement positifs x, y et z tels que $x^n + y^n = z^n$

Complexité / Introduction

```
int exp(int i, n)
/* computes i to the power n */
{
     int ans, j;
     ans = 1;
     for (j=1; j<=n; j++) ans *= i;
     return(ans);
}

main ()
{
     int n, total, x, y, z;
     scanf("%d", &n);
     total = 3;
     while (1) {
          for (x=1; x<=total-2; x++)
               for (y=1; y<=total-x-1; y++) {
                    z = total - x - y;
                    if (exp(x,n) + exp(y,n) == exp(z,n))
                         printf("hello, world\n");
               }
          total++;
     }
}
```

Problème de Fermat comme un programme

Figure 8.2: Fermat's last theorem expressed as a hello-world program

Complexité / Introduction

Problèmes solubles / Problèmes non solubles

Il ya des problèmes pour lesquels on ne connait pas de réponses.

Fameux problème : version arithmétique de SAT.

Etant donné une équation polynomiale à plusieurs variables, comme

$x^3yz + 2y^4z^2 - 7xy^5z = 6$; Existe-il des valeurs pour x, y et z qui rendent l'équation satisfaite ?

-> Il n y a pas d'algorithme qui résout ce problème (ni polynomial, ni exponentiel, ou pire)

Complexité / Introduction

Problèmes traitables / Problèmes non traitables

Parmi les problèmes solubles

- Quels sont les problèmes qu'un ordinateur peut traiter ?
- Quels sont les problèmes qu'un ordinateur ne peut pas traiter ?

Dans le cas de problèmes traitables :

- Est ce que le temps (espace) pris par l'exécution est acceptable ?
- Est-ce que les méthodes approchées peuvent être utilisées ?

Les machines ont des limites théoriques : elles ne peuvent pas résoudre TOUS les problèmes. Néanmoins, elles en résolvent un grand nombre !

Complexité / Introduction

Classification des programmes

O(1): Temps d'exécution indépendant de la taille du problème n.

O(ln(n)): Décomposition en sous problèmes de taille plus petite.(Logarithmique)

O(n): Traitement simple (Linéaire)

O(n × ln(n)): **D**écomposer en sous problèmes et combiner les solutions(Linéaire logarithmique)

O(n^2): Traitement de toutes les paires d'éléments.(Quadratique)

O(2^n): Temps exponentiel. A éviter. Caractéristiques des méthodes "Essayer toutes les possibilités"

Complexité / Introduction

Classification des programmes

n	ln (n)	n ln(n)	n^2	2^n
10	2,303	23,026	100	**1024**
20	2,996	59,915	400	**1048576**
50	3,912	195,601	2500	**1125899906842620**
100	4,605	460,517	10000	**1267650600228230000000000000**
1000	6,908	6907,755	1000000	**1,0715E+301**

Si chaque étape prend 10^{-3} ms, pour n= 100

0,1 s	4,605 ms	0,46 s	10 s	**401969368413315000000 siècles**

Complexité / Introduction

Technique "Essayer toutes les possibilités"

Exemple 1: (The subset-sum problem (SSP))

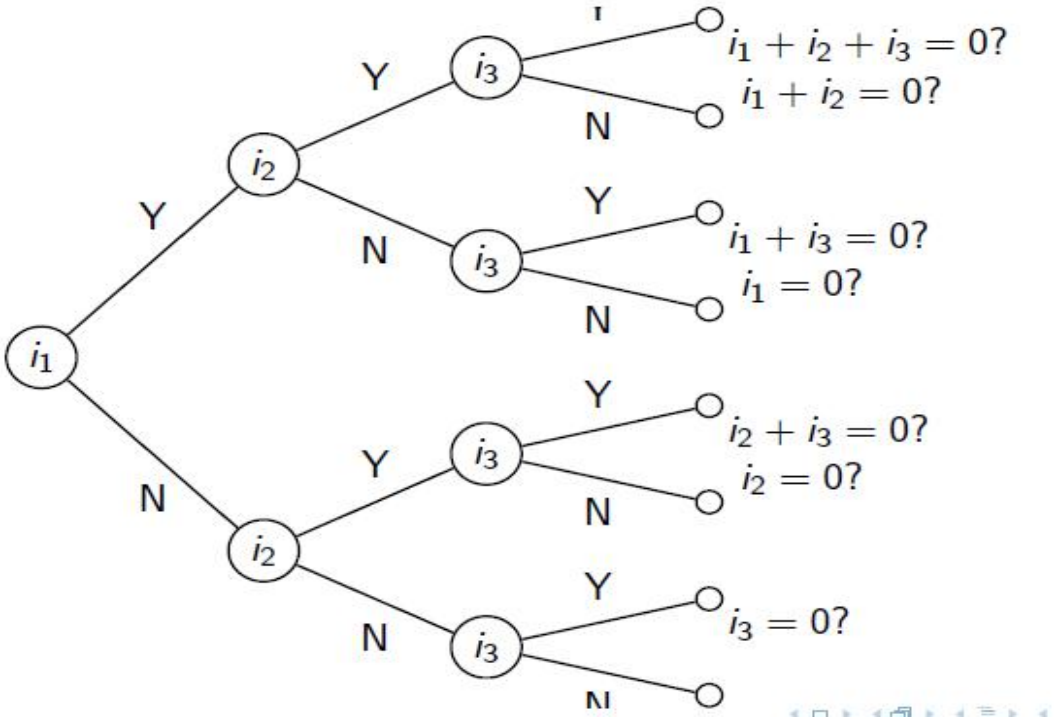
Etant donné un ensemble d'entiers S = {i1, i2, . . . , in}, est-ce qu'il existe un sous ensemble A de S tel que la somme de ses éléments est égale à 0?

Exemple, y a t-il un sous enemble de {−2,−3, 15, 14, 7,−10} avec une somme égale à 0?

Complexité / Introduction

Technique "Essayer toutes les possibilités"

Exemple 1: (The subset-sum problem (SSP))



Cas d'un ensemble avec 3 éléments i1, i2 et i3.

Complexité / Introduction

Technique "Essayer toutes les possibilités"

Exemple 1: (The subset-sum problem (SSP))

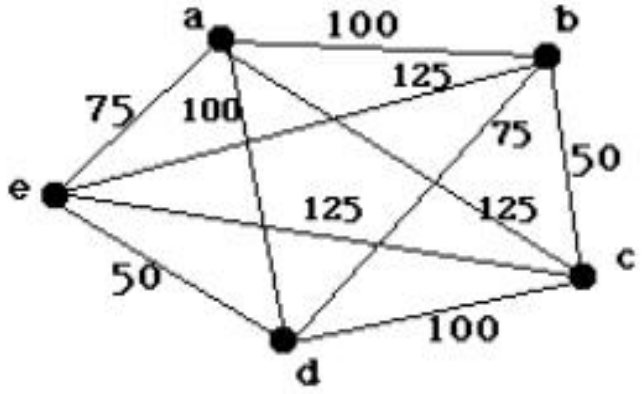
Complexité :

Si S = {i1, i2, . . . , in}

- Pour chaque entier, il y a deux choix.
- Comme il y a n entiers, il y a 2^n branches.
- Pour chaque branche, on fait au plus n additions,

Donc la complexité est $O(n \times 2^n)$.

Complexité / Introduction

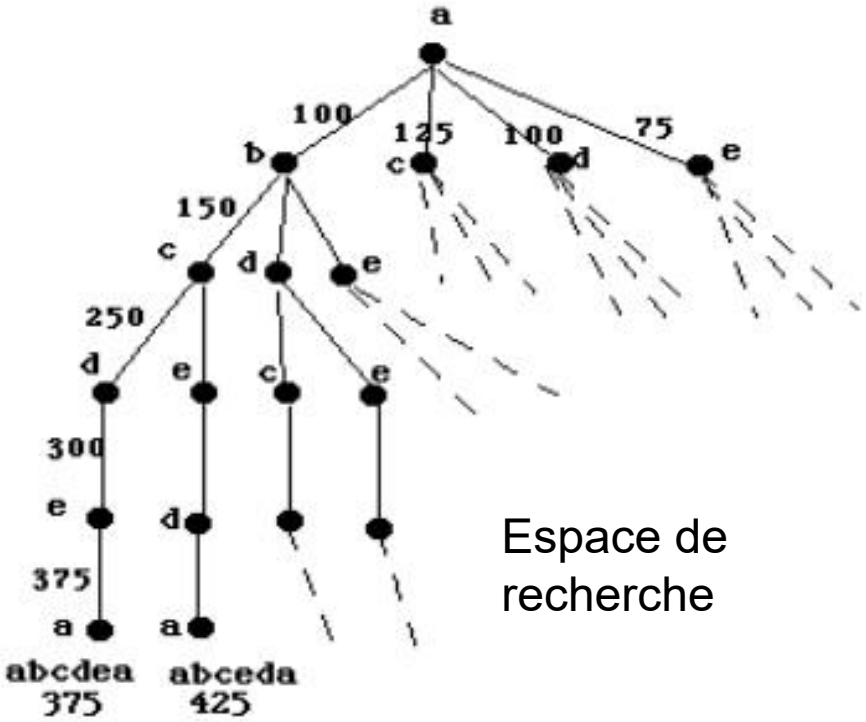


Technique "Essayer toutes les possibilités"

Exemple 2: PVC (Problème du Voyageur de Commerce)

Etant données n cités et les coûts de parcours d'une cité à une autre.

Quel est le tour le plus économique qui visite chaque cité une seule fois et qui retourne à la cité de départ ?



Espace de recherche

Complexité / Introduction

Technique "Essayer toutes les possibilités"

Exemple 2: PVC (problème du voyageur de commerce)

Complexité:
Une branche = une séquence de n cités. Nombre de branches différentes?

A chaque étape k on peut choisir parmi $n - k$ cités.

Donc le nombre de branches est $= (n - 1) \times (n - 2) \cdot \cdot \cdot \times 1 = (n - 1)!$.

Complexité : $O((n - 1)!)$. -> Algorithme (hyper-)exponential [$n! = O((n/2)^n)$)]

Complexité / Problèmes de décision

Langages

Σ^* : ensemble de toutes les chaines que l'on peut former avec l'alphabet Σ

Tout sous ensemble L de Σ^* est appelé langage. $L \subseteq \Sigma^*$

Ne pas confondre langage (tel que défini) et langage généré par une grammaire

Exemple 1 : Langage de toutes les chaines composées de n "0" suivi de n "1"
pour n >= 0 : {$\in$, 01, 0011, 000111, …}

Exemple 2 : Langage de toutes les chaines formées avec le même nombre de 0 et de 1
Exemple 3 : Langage de toutes les chaines dont la valeur est un nombre premier

Complexité / Problèmes de décision

Problème de décision

Question de savoir si une chaine donnée appartient ou non à un langage donné (Théorie des automates)

Si Σ est un alphabet et L un langage dans Σ, alors le problème L est :

Etant donnée une chaine w de Σ^*, décider si w est ou non dans L.

Problème : Test de primalité (tester les nombres premiers)

Langage Lp composé de toutes les chaines de bits dont les valeurs sont des nombres premiers.

En entrée : une chaine de bits,
Problème : répond par oui sou non

Complexité / Problèmes de décision

Décidabilité

P est décidable : Quelquesoit x, P(x) est vrai ou Faux

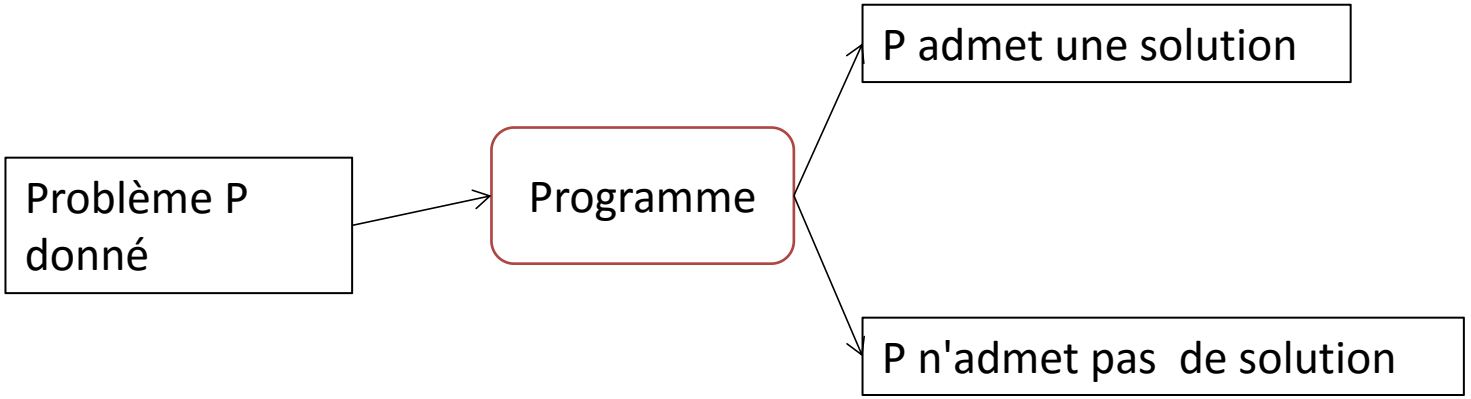
Décidabilité = calculabilité

Selon Turing, un problème est décidable si la machine de Turing s'arrète

Complexité / Problèmes de décision

Décidabilité

Problème P Décidable

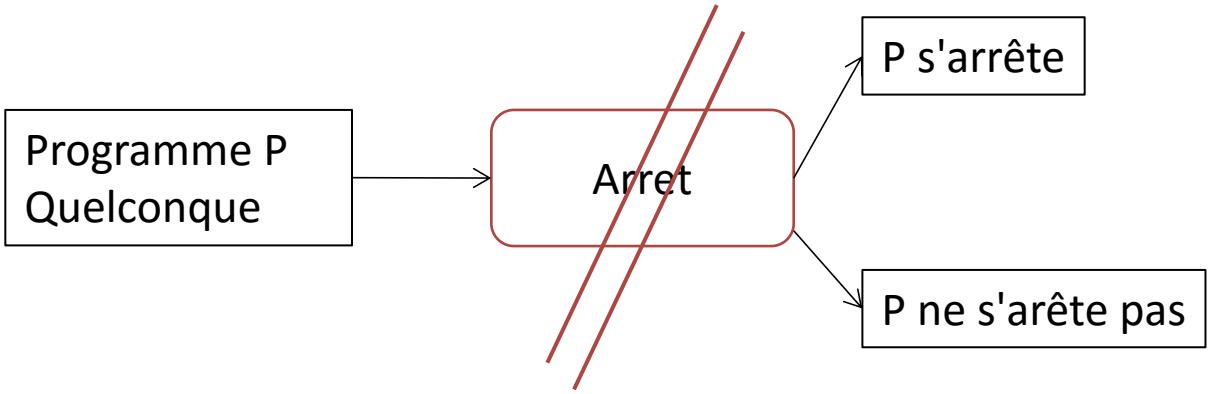


Problème indécidable : ne peut être résolu par un ordinateur.

Complexité / Problèmes de décision

Décidabilité (Résultat 1)

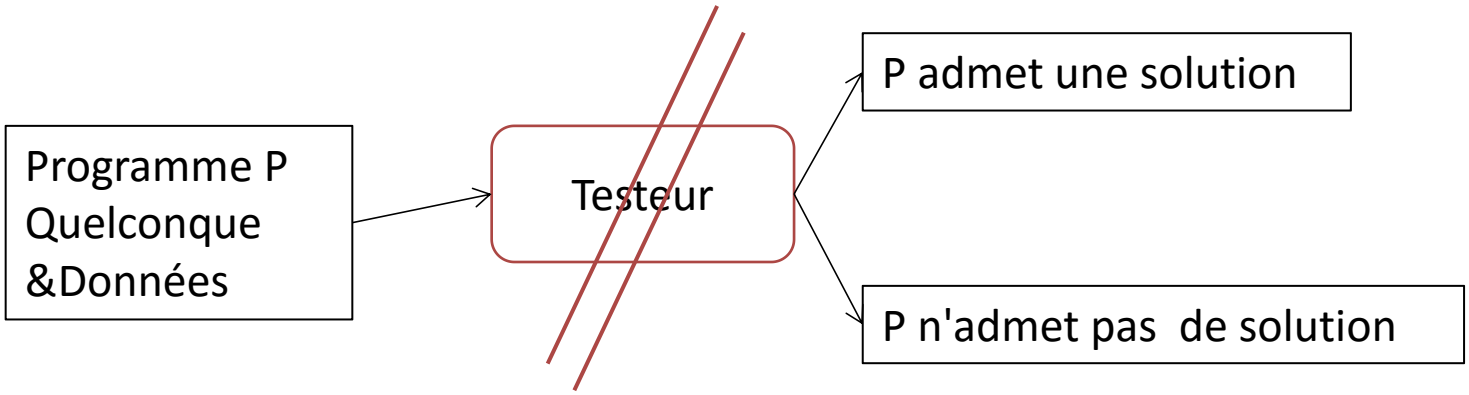
Le problème de l'arrêt d'un programme est indécidable



Complexité / Problèmes de décision

Décidabilité (Résultat 2)

Le problème de solubilité d'un programme est indécidable



Complexité / Types de problème

Décidabilité, Resolution et Optimisation)

Exemples	Problème de décision	Recherche	Optimisation
Tri d'un tableau	est-il trié ?	Trier	trier rapidement
Clique dans un graphe: 2 arêtes quelconque sont reliées par une arête	Existe-il une clique de longueur k?	Trouver une clique de longueur k	Trouver une clique de longueur maximale
Sac à dos Poids total ≤ W Total d'importance ≥ B	Vérifier la proprité	Proposer une solution	maximiser la valeur totale, sans dépasser le poids maximum.

Complexité / Machine de Turing

Introduction (Rappel)

L'étude de la calculabilité (décidabilité) nécessite l'utilisation d'un modèle théorique indépendant du langage et de l'ordinateur

Machine de Turing : le module le plus utilisé

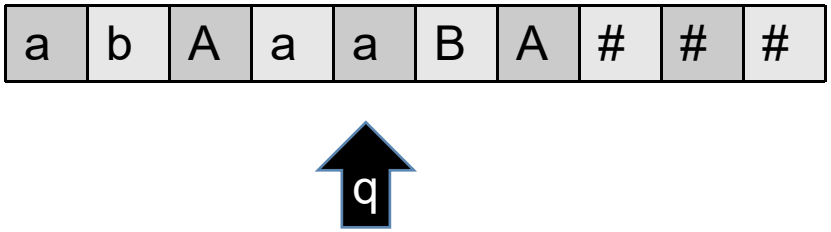
Ce modèle de machine de Turing caractérise bien ce qui est calculable
(Principe de Church-Turing)

Complexité / Machine de Turing

Définition

Une machine de Turing se compose

- une bande infinie de cases défilant devant une tête de lecture/écriture. (Mémoire)
- d'une partie de contrôle (nombre fini d'états possibles et de transitions) (Microprocesseur)



Contrairement à un ordinateur,

- la mémoire est infinie.
- l'accès est séquentiel

(pour un ordinateur la Ram est à accès directe)

Complexité / Machine de Turing

Fonctionnement

La machine est contrôlée par un nombre fini d'états { q_0, …, q_n }.

Au départ, la machine se trouve dans l'état q_0 qu'on appelle *état initial*.

Calcul = une suite d'*étapes de calcul* .

Une étape = (1) changer l'état de contrôle, (2) écrire un symbole sous la tête de lecture et (3) déplacer la tête de lecture.

Les étapes sont décrites par les transitions de la machine.

Complexité / Machine de Turing

Définition formelle

Sept-uplet (Q, Σ, Γ, E, q_0, F, #) où

Q : ensemble fini d'états de contrôle { q_0, …, q_n }.

Σ : alphabet d'entrée sans le symbole blanc #.

Γ : alphabet de bande. Il inclut l'alphabet d'entrée Σ et le symbole blanc #.

E : ensemble fini de transitions de la forme (p, a, q, b, x) où p et q sont des états, a et b sont des symboles de bande et x est un élément de {L, R}.

Une transition (p, a, q, b, x) est aussi notée p, a → q, b, x.

Q_0 : état initial.

F : ensemble des états finaux (états d'acceptation)

est le symbole blanc qui, au départ, remplit toutes les positions de la bande autres que celles contenant la donnée initiale.

Complexité / Machine de Turing

Déterminisme et non déterminisme

Une machine de Turing est *déterministe* si pour tout état p et tout symbole a, il existe au plus une transition de la forme p, a → q, b, x.

Une machine de Turing est non déterministe, si à chaque paire (p, a) est associé un ensemble des triplets (q, b, x) tels que p, a → q, b, x soit une transition.

x = L ou R,

x = ← ou →

Complexité / Machine de Turing

Exemple 1 de machine de Turing

MT :

({0}, {a, b}, {a, b, #}, E, 0, {0})

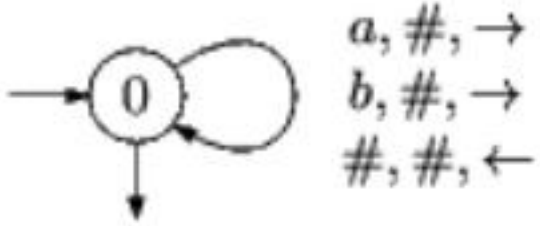
: blanc

.

Matrice

E	a	b	#
0	0, #, R	0, #, R	0, #, L

Diagramme



Complexité / Machine de Turing

Exemple 2 de machine de Turing

Pour le langage X = {$a^n b^n$ | n ≥ 0}

MT :
Q = {0, 1, 2, 3, 4}, Σ = {a, b},
Γ = {a, b, A, B, #}, q_0 = 0,
1 état final : 4

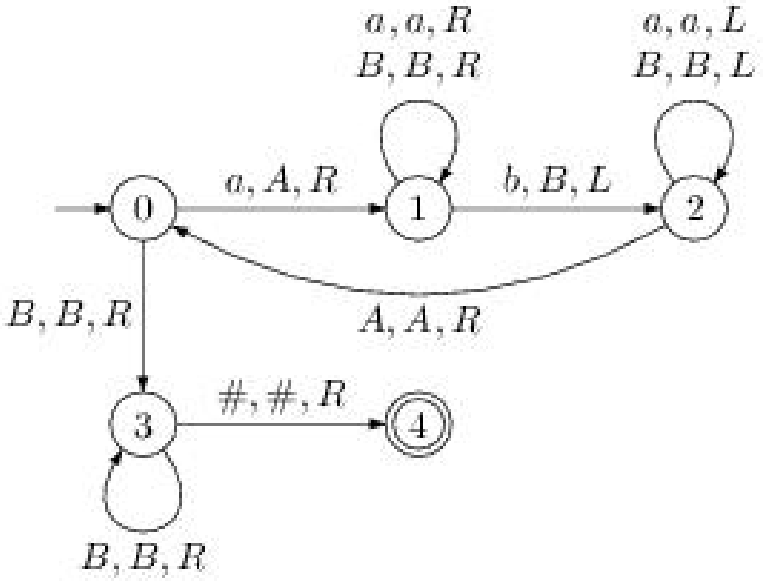
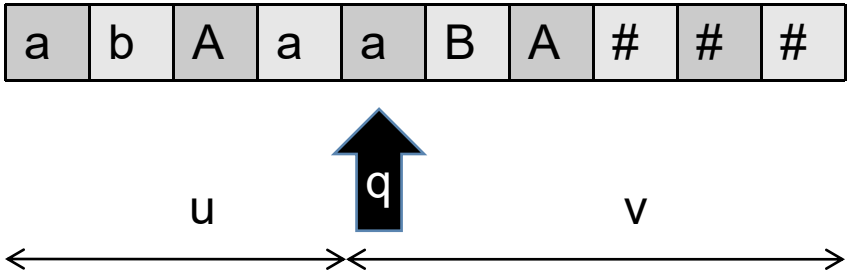


Diagramme des transitions

Complexité / Machine de Turing

Configurations



À l'état q, une configuration = uqv
-u est le contenu de la bande (strictement) à gauche de la tête de lecture
- v est le contenu de la tête à droite de la tête de lecture.

Le symbole sous la tête de lecture est donc le premier symbole de v

Configuration initiale : q_0w (q_0 : état initial et w : donnée initiale)

Complexité / Machine de Turing

Calculs

Un *calcul* : une suite $C_0, \ldots, C_n$ de configurations telles que

$C_0 \Rightarrow C_1 \Rightarrow C_2 \Rightarrow \ldots \Rightarrow C_{n-1} \Rightarrow C_n$.

$C_0 \Rightarrow^* C_n$ (Fermeture réflexive et transitive de la relation $\Rightarrow$).

3 situations peuvent survenir :

Le calcul atteint une configuration acceptante, bloquante, bouclante

Complexité / Machine de Turing

Soit w = aba la donnée fournie à la machine

Calculs

Machine de Turing
({0}, {a, b}, {a, b, #}, E, 0, {0}, #).

E	a	b	#
0	0, #, R	0, #, R	0, #, L

Configurations

C_0 = 0aba###...

C_1 = #0ba###...

C_2 = ##0a###...

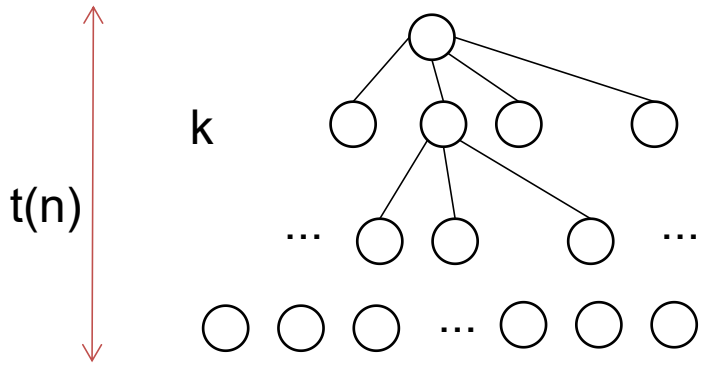
C_3 = ###0###...

C_4 = ##0###...

C_5 = #0###...

C_6 = 0###...

Complexité / Machine de Turing



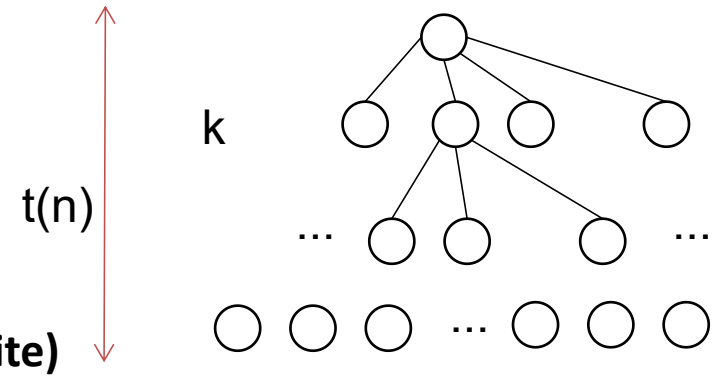
Simulation des machines non déterministes

Toute machine non déterministe peut être simulée par une machine déterministe.

La simulation consiste alors à essayer successivement tous les calculs de la machine non déterministe.

Si la machine non déterministe fonctionne en temps t(n), le nombre de calculs possibles pour une entrée de taille n est borné par $k^{t(n)}$ où k est le nombre maximal de transitions qui peuvent être effectuées à partir d'une configuration.

Complexité / Machine de Turing



Simulation des machines non déterministes (suite)

L'entier k est le cardinal maximal des ensembles $\delta(p, a) = \{ (q, b, x) \mid p, a \rightarrow q, b, x \in E \}$ pour tout p et tout a.

Le temps de simulation d'un calcul est proportionnel à la longueur de ce calcul. Le temps nécessaire pour simuler tous les calculs est donc de l'ordre de $t(n)k^{t(n)}$. Comme on a la relation $t(n)k^{t(n)} = 2^{O(t(n))}$ si $t(n) \geq n$, on obtient le résultat suivant.

Proposition. ***Soit t une fonction de N dans R^+ telle que $t(n) \geq n$ pour tout $n \geq 0$. Une machine non déterministe qui fonctionne en temps t(n) est équivalente à une machine déterministe qui fonctionne en temps $2^{O(t(n))}$.***