ALGORITHMIQUE AVANCÉE

Laboratoire Informatique Image Interaction (L3I)
Université de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 France
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

OBJECTIFS DU COURS – ALGORITHMIQUE AVANCÉE

Deux sous objectifs

Algorithmique des graphes

- Cours 1 : Définition et parcours (largeur, profondeur, tri topo)
- Cours 2 : Algos polynomiaux classiques (PCC, connexité, MST, …)
- Cours 3 : Quelques algos plus complexes (communautés)
-

Complexité et structures de données

- Cours 1 : Complexité et structures de données
- **Cours 2 : Calcul de complexité**
- Cours 3 : Classes de complexité et décidabilité
- Cours 4 : Heuristiques et approximations

ALGORITHMIQUE

PART II COMPLEXITÉ
CALCUL DE COMPLEXITÉ

Laboratoire Informatique Image Interaction (L3I)
Université de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 France
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

INTRODUCTION

L'étude de la complexité est une étape essentielle avant toute implémentation :

- Savoir choisir la structure de données appropriée aux données (données statiques / dynamiques – opérations sur les données)
- Savoir choisir un algorithme en fonction de sa complexité (attention aux complexités exponentielles)
  - **Cours 1 :** Complexité et structures de données
- Savoir calculer la complexité d'un algorithme (fonctions récursives)
  - **Cours 2 :** Calcul de complexité

COMPLEXITÉ DU PARCOURS EN LARGEUR

Nom: Parcours en largeur

Entrée : un graphe G de n sommets et m arêtes, un sommet s

Initialiser une file F et y ajouter s

etat (s) = dansfile

tant que F n'est pas vide

t = tête de F

pout tout u ∈ adj(t)

si etat(u) = inexploré

ajouter u dans F

etat(u) = dansfile

défiler F

état(t) = exploré

n itérations au plus
(chaque sommet une fois dans la file)

n itérations au plus
(chaque sommet a au plus n voisins)

Instructions en temps constant

Complexité : O(n^2)

COMPLEXITÉ DU PARCOURS EN LARGEUR

Nom: Parcours en largeur

Entrée : un graphe G de n sommets et m arêtes, un sommet s

Initialiser une file F et y ajouter s

etat (s) = dansfile

tant que F n'est pas vide

t = tête de F

pout tout u ∈ adj(t)

si etat(u) = inexploré

ajouter u dans F

etat(u) = dansfile

défiler F

état(t) = exploré

n itérations au plus
(chaque sommet une fois dans la file)

m itérations en tout
(chaque arête visitée une seule fois)

Instructions en temps constant

Complexité : O(n+m)

RÈGLES DE CALCUL

Boucle for interne:

- n itérations au plus : **$O(n^2)$**
- m itérations en tout : **$O(n+m)$**

La seconde analyse est plus fine et permet d'obtenir une meilleure complexité

Pour calculer finement la complexité d'un algorithme, il est nécessaire d'utiliser des règles de calcul:

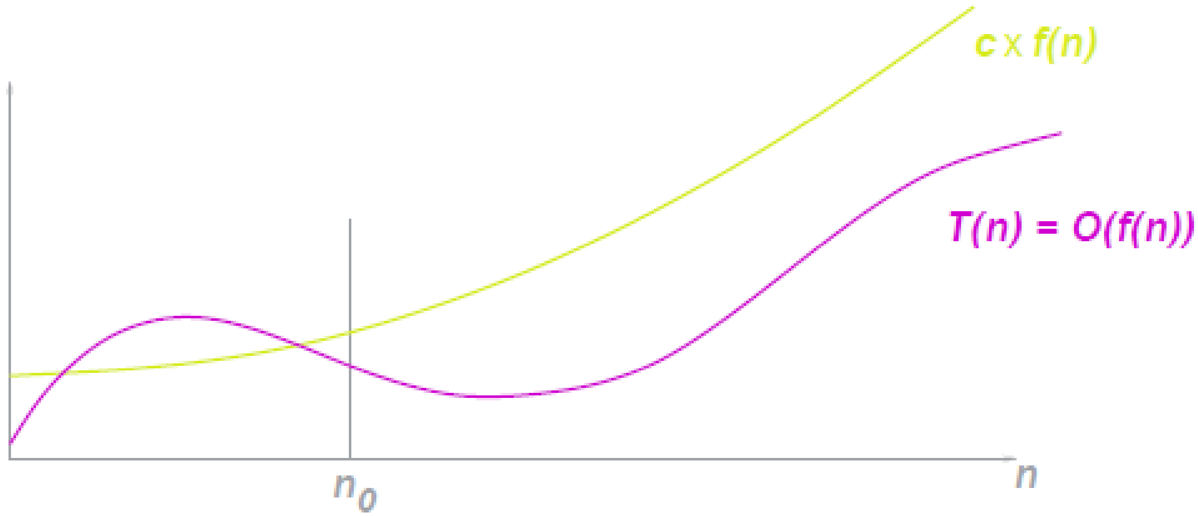
- Sommes
- Equations récursives

PRINCIPE GÉNÉRAL DU CALCUL DE COMPLEXITÉ

- n : taille des données
- T(n) : nombre d'opérations élémentaires
- f(n) : complexité (constante, linéaire, polynomiale, exponentielle)

T(n) = O(f(n))

si il existe des constantes c et no telles que, pour tout $n \geq no$:
$T(n) \leq c\ f(n)$



EXEMPLES

$T(n) = n^3 + 2n^2 + 4n + 2$

$= O(n^3)$ car $T(n) \leq 8n^3$ pour n strictement positif

$T(n) = n \log n + 12n - 10$

$= O(n \log n)$ car $T(n) \leq n \log n$ pour $n > 5/6$

$T(n) = 10n^{10} + 8n^8 + 2^n / 1000$

$= O(2^n)$

PRINCIPE GÉNÉRAL DU CALCUL DE COMPLEXITÉ

Règles pour calculer le nombre T(n) d'opérations élémentaires le plus finement possible :

1. Les instructions en séquence
2. Les instructions conditionnelles
3. Les itérations
4. Les appels de fonction
5. Les fonctions récursives

SÉQUENCE D'INSTRUCTIONS : ADDITION

Nom: Ajout dans une liste chaînée

Entrée : un liste L, un élément x

créer une cellule new	coût c1
Ajouter x dans new	coût c2
new.suivant = L.tete	coût c3
L.tete = new	coût c4

$$T(n) = c1 + c2 + c3 + c4$$
$$= O(1)$$

Car c1, c2, c3 et c4 sont de coût constant

CONDITIONELLE : MAX

<u>Nom:</u> Max de deux éléments

<u>Entrée :</u> deux éléments x et y

<u>Sortie :</u> le max de x et y

Si x > y	coût comp
retourner x	coût constant c1
Sinon	
retourner y	coût constant c2

T(n) = comp + **max** (c1+c2)

= comp + 0(1)

Si x et y sont des entiers:

comp = O(1)

Si x et y sont des points en dimension n :

comp = O(n)

ITÉRATION : SOMME

Nom: Recherche d'un élément x dans un tableau

Entrée : un tableau T de n éléments, un élément x

Sortie : le rang de x dans le tableau, -1 sinon

n fois

Pour i allant de 0 à n-1	coût c1
Si x = T[i] alors	coût c2
retourner i	coût c3
Retourner -1	coût c4

$$T(n) = c4 + \sum_{i=0}^{n-1} (c1 + c2 + c3)$$

$$= c4 + (\mathbf{c1} + \boldsymbol{c2} + \boldsymbol{c3}) * \sum_{i=0}^{n-1} 1$$

$$= c4 + (c1 + c2 + c3) * \mathbf{n}$$

$$= O(n)$$

CE QU'IL FAUT SAVOIR POUR MANIPULER LES SOMMES

- On sort les constantes :

$\sum_{i=1}^{n}(c * f(i)) = c * \sum_{i=0}^{n-1} f(i)$

$\sum_{i=1}^{n}(c + f(i)) = cn + \sum_{i=0}^{n-1} f(i)$

$\sum_{i=1}^{n}(f(i) + g(i)) = \sum_{i=1}^{n} f(i) + \sum_{i=1}^{n} g(i)$

- La somme simple :

$\sum_{i=1}^{n} 1 = \sum_{i=0}^{n-1} 1 = n = O(n)$

- La somme des n premiers entiers :

$\sum_{i=1}^{n} i = \frac{n*(n+1)}{2} = O(n^2)$

$\sum_{i=0}^{n-1} i = \frac{n*(n-1)}{2} = O(n^2)$

- Pour aller plus loin :

$\sum_{i=1}^{n} x^i = \frac{x^{n+1}-1}{x-1} = O(x^n)$

$\sum_{i=1}^{n} 2^i = 2^{n+1} - 1 = O(2^n)$

DOUBLE ITÉRATION

<u>Nom:</u> Tri d'un tableau

<u>Entrée :</u> un tableau T

Pour i allant de 0 à n-1

min = i

Pour j allant de i+1 à n-1

Si T[j] < T[min] alors

Permuter(T[i],T[min])

n fois

n-i fois

Coût perm

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{n-1}\left(\sum_{j=i+1}^{n-1} perm\right) \\
&= \mathbf{perm} * \sum_{i=0}^{n-1}\left(\sum_{j=i+1}^{n-1} 1\right) \\
&= \mathrm{perm} * \sum_{i=0}^{n-1}(\boldsymbol{n} - \boldsymbol{i}) \\
&= \mathrm{perm} * \left(\boldsymbol{n^2} - \sum_{i=0}^{n-1} i\right) \\
&= \mathrm{perm} * \left(\mathrm{n}^2 - \frac{\boldsymbol{n}*(\boldsymbol{n}-\mathbf{1})}{\mathbf{2}}\right) \\
&= \mathrm{perm} * \frac{\mathbf{2}\boldsymbol{n}^2-(\boldsymbol{n}*(\boldsymbol{n}-\mathbf{1}))}{\mathbf{2}} \\
&= O(n^2)
\end{aligned}
$$

APPEL DE FONCTION

Nom: Connexité faible

Entrée : un graphe G=(S,A) de n sommets et m arêtes

Retour : vrai si le graphe est connexe, faux sinon

choisir un sommet s du graphe

parcours (G,s) O(n+m)

pout tout x $\in$ S n fois

si etat(x) = inexploré

retourner faux

retourner vrai

$$
\begin{aligned}
T(n) &= 1 + n+m + \sum_{i=1}^{n} 1 \\
&= 1 + 2n + m \\
&= O(n+m)
\end{aligned}
$$

RETOUR SUR LES GRAPHES

<u>Nom:</u> Parcours en largeur

<u>Entrée :</u> un graphe G=(S,A) de n sommets et m arêtes, un sommet s

Initialiser une file F et y ajouter s

etat (s) = dansfile

tant que F n'est pas vide

|S|=n fois

t = tête de F

pout tout u ∈ adj(t)

|adj(t)| fois

si etat(u) = inexploré

ajouter u dans F

etat(u) = dansfile

défiler F

état(t) = exploré

- Somme classique :

$$T(n) = \sum_{i=1}^{n}\left(1 + \sum_{j=1}^{|adj(x_i)|} 1\right)$$

$$= \sum_{i=1}^{n}\left(1 + |adj(xi)|\right)$$

$$= n + \sum_{i=1}^{n}\left(|adj(xi)|\right)$$

$$= n+m = O\,(n+m)$$

RETOUR SUR LES GRAPHES

Nom: Parcours en largeur

Entrée : un graphe G=(S,A) de n sommets et m arêtes, un sommet s

Initialiser une file F et y ajouter s

etat (s) = dansfile

tant que F n'est pas vide

|S|=n fois

t = tête de F

pout tout u ∈ adj(t)

|adj(t)| fois

si etat(u) = inexploré

ajouter u dans F

etat(u) = dansfile

défiler F

état(t) = exploré

- Somme classique :

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{n}\left(1+\sum_{j=1}^{|adj(x_i)|}1\right)\\
&= \sum_{i=1}^{n}(1+|adj(xi)|)\\
&= n+\sum_{i=1}^{n}(|adj(xi)|)\\
&= n+m\\
&= O(n+m)
\end{aligned}
$$

- Approche ensembliste:

$$
\begin{aligned}
T(n) &= \sum_{t\in S}\left(1+\sum_{u\in adj(t)}1\right)\\
&= \sum_{t\in S}(1+|adj(t)|)\\
&= n+\sum_{t\in S}|adj(t)|\\
&= n+m\\
&= O(n+m)
\end{aligned}
$$

$\sum_{t\in S}|adj(t)| = m$

FONCTION RÉCURSIVE : ÉQUATION RÉCURSIVE

<u>Nom:</u> Recherche dichotomique

<u>Entrée :</u> un tableau T de n éléments, un élément x

<u>Sortie :</u> vrai ou faux

Si (n = 1) alors

renvoyer (x == T[0])

O(1)

Calculer le milieu mil du tableau

Si (x < T[mil])

Rechercher (T[0,mil-1],x)

T(n/2)

Sinon

Rechercher (T[mil,n-1],x)

T(n/2)

- Equation récursive:

$$T(1) = 1$$

$$T(n) = 1 + T\left(\frac{n}{2}\right)$$

$$= 1 + 1 + T\left(\frac{n}{4}\right)$$

$$= 1 + 1 + \cdots + T\left(\frac{n}{2^k}\right)$$

$$= \sum_{i=0}^{k} 1$$

$$= \sum_{i=0}^{\log n} 1$$

$$= \log n$$

car $T\left(\frac{n}{2^k}\right) = T(1) = 1$

et donc k=log n

FONCTION RÉCURSIVE : ÉQUATION RÉCURSIVE

Nom: TriParFusion

Entrée : un tableau S de n éléments

Si ($n \leq 1$) alors

renvoyer S

Décomposer S en S1 et S2 — O(n)

S1 = TriParFusion (S1) — T(n/2)

S2 = TriParFusion (S2) — T(n/2)

S = fusion (S1,S2) — O(n)

renvoyer S

- Equation récursive:

$$T(1) = 1$$

$$T(n) = 1 + n + 2 * T\left(\frac{n}{2}\right) + n$$

$$= (2n + 1) + 2 * T\left(\frac{n}{2}\right)$$

RÉSOLUTION D'ÉQUATION RÉCURSIVE

$T(n) = (2n + 1) + 2\,T\left(\frac{n}{2}\right)$

$= (2n + 1) + (2n + 2) + 4\,T\left(\frac{n}{4}\right)$ (car $T\left(\frac{n}{2}\right) = n + 1 + 2\,T(\frac{n}{4})$)

$= (2n + 1) + (2n + 2) + (2n + 4) + 8\,T\left(\frac{n}{8}\right)$

(car $T\left(\frac{n}{4}\right) = n/2 + 1 + 2\,T(\frac{n}{8})$)

$= (2n + 1) + (2n + 2) + \cdots + (2n + 2^{k-1}) + 2^k\,T\left(\frac{n}{2^k}\right)$

$= \sum_{i=0}^{\log n-1}(2n + 2^i) + 2^{\log n}$ (car $T\left(\frac{n}{2^k}\right) = T(1) = 1$ et donc k=log n)

$= 2n\log n + \sum_{i=0}^{\log n-1} 2^i + n$ (car $2^{log n} = n$)

$= 2n\log n + 2n - 1$ (car $\sum_{i=0}^{log\,n-1} 2^i = 2^{log\,n} - 1 = n - 1$)

= O (n log n)

MÉTHODE GÉNÉRALE

$$T(n) = a\,T\left(\frac{n}{b}\right) + O(n^{d}) \qquad T(1) = c$$

- $a\,T\left(\frac{n}{b}\right)$: décomposition de *T(n)* en *a* sous-problèmes de taille *n/b*
- $O(n^{d})$: reconstruction de la solution pour *T(n)* en fonction des solutions des sous-problèmes
- $T(1) = c$: temps constant pour résoudre le problème de taille 1

<u>**Théorème:**</u>

1. Si $d < log_b a$ alors $T(n) = O\,(n^{\log_b a})$
2. Si $d = log_b a$ alors $T(n) = O\,(\log n * n^{\log_b a})$
3. Si $d > log_b a$ alors $T(n) = O\,(n^{d})$

RETOUR SUR LE TRI PAR FUSION

$$T(n) = 2\,T\left(\frac{n}{2}\right) + 2n + 1 \qquad T(1) = 1$$

On a :

- $2n+1 = O(n)$ donc $O(n^{d}) = O(n)$ et $d = 1$
- $a=2$ et $b=2$ donc $\log_b a = \log_2 2 = 1$

On est donc dans le cas 2 du théorème car $d = log_b a$:

$$\begin{aligned} T(n) &= \log n * n^{\log_b a} \\ &= \log n * n \end{aligned}$$

RÈGLES DE CALCUL DE LA COMPLEXITÉ

Règles pour calculer le nombre T(n) d'opérations élémentaires le plus finement possible:

1. Les instructions en séquence : *addition*
2. Les instructions conditionnelles : *max*
3. Les itérations : *somme*
4. Les appels de fonction
5. Les fonctions récursives : *équations récursives*

AUTRES TYPES DE COMPLEXITÉS

Il existe d'autres types de complexité que la complexité en temps dans le pire des cas:

1. Complexité en espace mémoire
2. Complexité en moyenne
3. Complexité en fonction de la taille du résultat

COMPLEXITÉ EN ESPACE MÉMOIRE

Estimation de l'espace mémoire nécessaire pour stocker des données

- *Cas d'utilisation:*
  1. Comparaison de structures de données stockant le même type de données

     Exemple : Représentation d'un graphe:
     - Par une matrice : $O(n^2)$. Pour des graphes denses
     - Par des listes d'adjacence : $O(n+m)$. Pour des graphes peu denses
     - Par une liste d'arêtes : $O(m)$

  2. Analyse plus fine des algorithmes.

     Certaines algorithmes utilisent un stockage intermédiaire de données qui peut nécessiter beaucoup d'espace mémoire

COMPLEXITÉ EN MOYENNE

Estimation du temps d'exécution pour traiter une entrée tirée selon une *distribution* donnée

(le plus souvent on considère une *distribution uniforme* des données)

- *Cas d'utilisation :*
  1. Lorsque la complexité dans le pire des cas est très élevée, mais les entrées qui correspondent à ces cas sont très rares

     Exemple : recherche dans un arbre binaire de recherche:
     - O (log n) dans le pire des cas (l'élément recherché est une feuille)
     - O (1) dans le meilleur des cas (l'élément recherché est la racine)

  1. Lorsqu'on veut comparer des algorithmes ayant la même complexité dans le pire des cas

     Exemple : choisir le meilleur algorithme de tri en fonction de la distribution des données dans la tableau d'entrée

COMPLEXITÉ EN FONCTION DU RÉSULTAT

Estimation du temps d'exécution en fonction de la taille du résultat
(et non en fonction de la taille de l'entrée)

- *Cas d'utilisation :*
  1. Lorsque l'algorithme est un algorithme de génération d'un résultat dont on ne connait pas la taille

  Exemple : génération d'un graphe aléatoire

INTERLUDE : MACHINE DE TÜRING

Machine de Türing = Machine universelle:

- Modèle abstrait de fonctionnement d'un appareil mécanique de calcul.
- Premier calculateur universel programmable, à l'origine des premiers programmes et des premiers ordinateurs.
- Modèle toujours utilisé en informatique théorique pour résoudre des problèmes de complexité et de calculabilité.

Alan Türing (1912-1954)

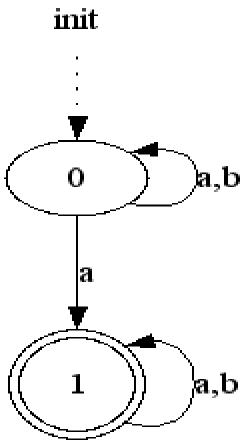
- Mathématicien britannique créateur de la machine de Türing
- A l'origine des concepts de calculabilité et de décidabilité
- S'est distingué par ses travaux sur les codes secrets nazis pendant la second guerre mondiale

RETOUR SUR LES AUTOMATES

Définition formelle:

Un automate est un 5-uplet
(A,E,Δ,e,F)

- A : alphabet
- E : ensemble d'états
- Δ ∈(E x A x E), ensemble de transitions.
- e ∈ E : état initial
- F ⊆ E : état finaux



A = {a,b}
E = {0,1}
e = 0
F = {1}
Δ = { (0,a,0), (0,b,0), (0,a,1), (1,a,1), (1,b,1) }

	(a)	(b)
0	0,1	0
1	1	1

MÉCANISME DE RECONNAISSANCE DE MOTS

Un automate permet de reconnaître un ***ensemble de mots***, donc un ***langage***:

- **Entrée :** un mot
- **Sortie :** vrai ou faux selon que le mot appartient ou non au langage décrit par l'automate

1. Initialiser l'état courant avec l'état initial de l'automate
2. Pour chaque lettre du mot d'entrée lu de la gauche vers la droite, "franchir" une transition de l'automate pour modifier l'état courant:

   $(e,a,e') \in \Delta$

   « si je suis dans l'état e,

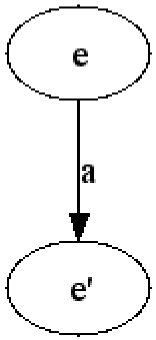
   et si je lis la lettre a,

   alors je vais dans l'état e' »
3. Renvoyer vrai ou faux selon que l'état courant à la fin est un état final ou non

AUTOMATE ET AUTOMATE À SORTIE

Automate classique :

$\Delta \in (E \times A \times E)$



- (e,a,e') $\in \Delta$:

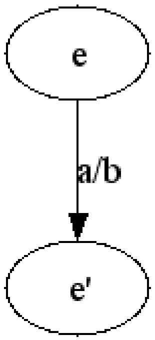
  « si je suis dans l'état e,

  et si je lis a,

  alors je vais dans l'état e' »

Automate à sortie :

$\Delta \in (E \times A \times \mathbf{A} \times E)$



- (e,a,**b**,e') $\in \Delta$:

  «si je suis dans l'état e, et si je lis a,

  *alors **j'écris b***

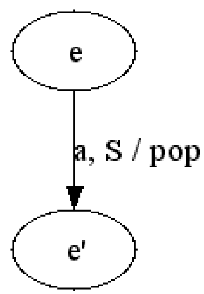
  et je vais dans l'état e' »

AUTOMATE À PILE ET MACHINE DE TÜRING

Automate à pile :

$\Delta \in$(E x A x **P x Op** x E)

pop: dépiler
push(X): empiler X
nop: rien



- (e, a, **S**, **pop**, e') $\in \Delta$:

«si je suis dans l'état e, si **S est en sommet de pile***, et si je lis a, alors je* **dépile la pile** *et je vais dans l'état e'»*

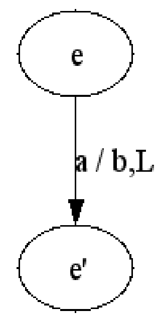
- (e, ε, **S**, **push(X)**, e') $\in \Delta$:

«si je suis dans l'état e et si **S est en sommet de pile***, alors* **je ne lis rien***, j'empile X et je vais dans l'état e'»*

Machine de Türing :

$\Delta \in$(E x A x O x E x **{L,R,S}**)

L: left
R: right
S: stay



- (e,a,e',b,**L**) $\in \Delta$:

«si je suis dans l'état e, et si je lis a, alors j'écris b, ***je déplace le curseur à gauche (L)*** *et je vais dans l'état e'»*

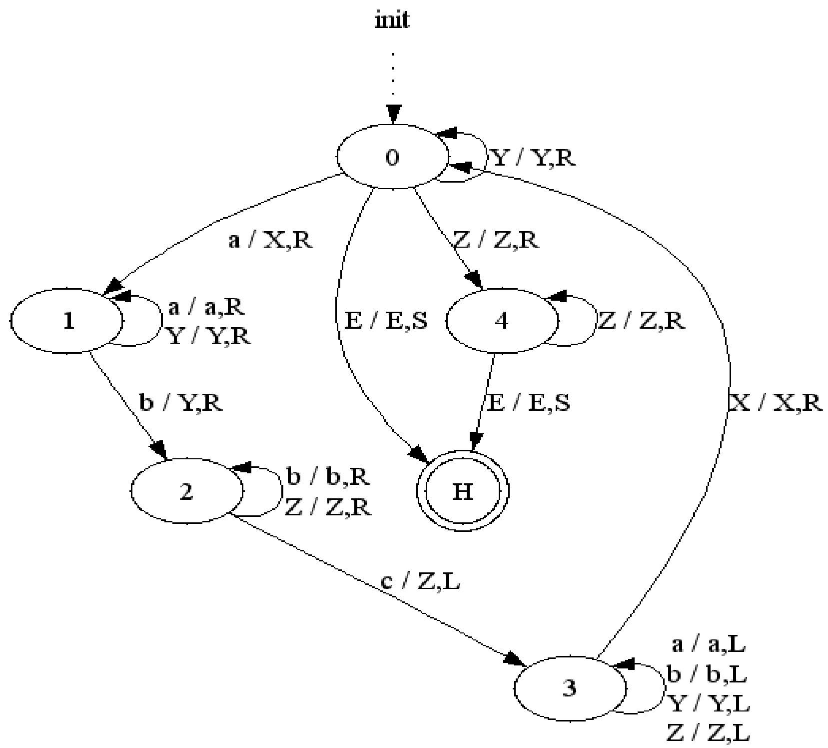
MACHINE DE TÜRING

Langage $a^n b^n c^n$, $n \geq 0$

- Permet de compter:
  - Parent. ouvrantes/fermantes
  - If, then
- Ce langage n'est pas un langage régulier

$\Rightarrow$ Il n'existe pas d'automate pour le reconnaître

Mais il existe une machine de Türing pour le reconnaître



MODÈLES ET MACHINE UNIVERSELLE

Différents modèles de machine de calcul permettant de reconnaitre des mots / langages :

- Automate
- Automate déterministe complet minimal
- Automate à sortie
- Automate à pile
- Machine de Türing
- Machine de Türing déterministe complet
- Machine de Türing multibandes
- ……

Existe-t-il un modèle plus puissant que les autres ?

(puissance = langages reconnus)

MODÈLES ET MACHINE UNIVERSELLE

<u>Théorème:</u>
Pour tout *automate* il existe un automate *déterministe complet minimal* reconnaissant le même langage

⇒ Modèles d'automates **équivalents** (sortie booléenne ou autre)

<u>Théorème:</u>
Pour toute *machine de Türing* il existe une machine de Türing *déterministe complete* reconnaissant le même langage

⇒ Modèles de machine de Türing **équivalents**

<u>Théorème:</u>
Un langage est reconnaissable par un *automate* si et seulement si c'est un *langage régulier*

⇒ Machines de Türing **plus puissantes** que les automates (car reconnaissant $a^n b^n$, $n \geq 0$, langage non régulier)

MODÈLES ET MACHINE UNIVERSELLE

puissance

equivalents
- automate
- automate déterministe complet minimal
- automate à sortie

≈ séquence

- automates à pile

≈ mémoire

equivalents
- Machine de Türing
- Machine de Türing déterministe complet
- Machine de Türing multibandes

≈ itération

Existe-t-il un modèle plus puissant ?

MACHINE DE TÜRING

<u>Thése de Church-Türing (1930) :</u>
Tout ce qui est calculable
peut être calculé par
une machine de Türing

Thèse seulement car non prouvé
mais personne n'a prouvé le contraire
et personne n'a trouvé un modèle plus puissant

$\Rightarrow$ Machines de Türing = machine universelle

CONCLUSION

- **Cours 1 :** Complexité et structures de données
  - Savoir choisir la structure de données - le container appropriée aux données
  - Savoir choisir un algorithme en fonction de sa complexité
- **Cours 2 :** Calcul de complexité
  - Savoir calculer la complexité d'un algorithme (fonctions récursives)

Comment traiter les complexités exponentielles ?

- **Cours 3 :** Classes de complexité et décidabilité
- **Cours 4 :** Heuristiques et approximations