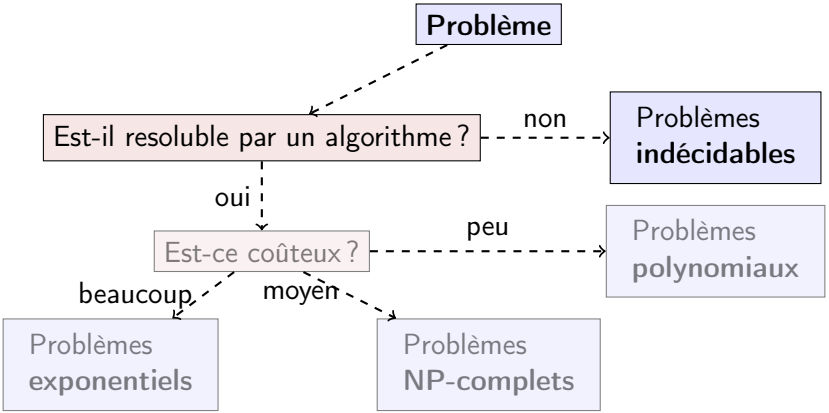
Calculabilité et NP-Complétude

Partie II : Classes de complexité
Master 2

Architecture du cours (schématique)



Métaheuristiques, approximations, etc.

Solveurs SAT

Solveurs IP (VS)

Plan

1. Hiérarchie du temps
2. Hiérarchie en espace
3. Certificats et classe NP
4. Algorithmes non-déterministes
5. Classe complémentaires et complétude

Langage/Problème décidé par une machine de Turing

Langage décidé

Une machine de Turing M décide une langage L si

- Pour tout mot $w \in L$, il existe un calcul maximal fini acceptant w.
- Pour tout mot $w \notin L$, tous les calculs maximaux sont finis et aucun n'est acceptant.

Une machine de Turing est polynomiale s'il existe un entier k tel que pour tout mot w, tout calcul maximal sur w a une longueur dans $O(|w|^k)$.

Langage/Problème décidé par une machine de Turing

Langage décidé

Une machine de Turing M décide une langage L si

- Pour tout mot $w \in L$, il existe un calcul maximal fini acceptant w.
- Pour tout mot $w \notin L$, tous les calculs maximaux sont finis et aucun n'est acceptant.

Une machine de Turing est polynomiale s'il existe un entier k tel que pour tout mot w, tout calcul maximal sur w a une longueur dans $O(|w|^k)$.

Un problème est dans la classe **P** (polynomial) s'il existe une machine de Turing **déterministe** polynomiale le décidant.

Langage/Problème décidé par une machine de Turing

Langage décidé

Une machine de Turing M décide une langage L si

- Pour tout mot $w \in L$, il existe un calcul maximal fini acceptant w.
- Pour tout mot $w \notin L$, tous les calculs maximaux sont finis et aucun n'est acceptant.

Une machine de Turing est polynomiale s'il existe un entier k tel que pour tout mot w, tout calcul maximal sur w a une longueur dans $O(|w|^k)$.

Un problème est dans la classe **P** (polynomial) s'il existe une machine de Turing **déterministe** polynomiale le décidant.
Un problème est dans la classe **P** (polynomial) s'il existe un programme de complexité polynomiale pour le résoudre.

Coût d'exécution sur un mot

Etant donnés une machine de Turing **déterministe** M **décidant** un langage L, et un mot w de longueur $|w|$, on note

$$\mathrm{cost}_M(w)$$

la longueur (nombre de transitions utilisées) du calcul maximal sur w.

Coût d'exécution sur un mot

Etant donnés une machine de Turing **déterministe** M **décidant** un langage L, et un mot w de longueur $|w|$, on note

$$\mathrm{cost}_M(w)$$

la longueur (nombre de transitions utilisées) du calcul maximal sur w.

La complexité en temps de M est la fonction

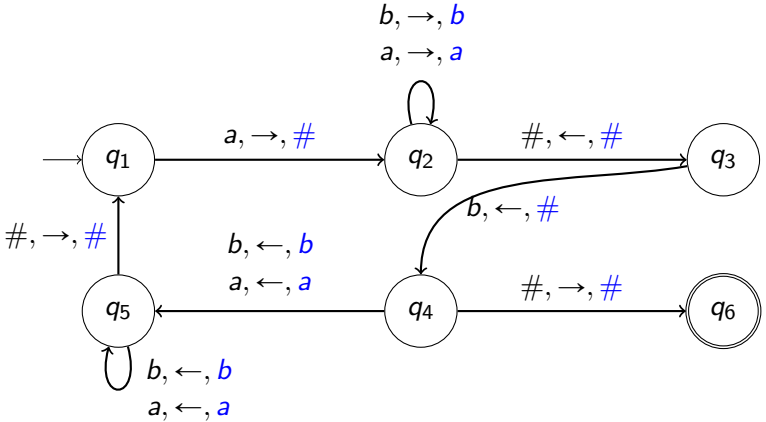
$$C_M(n) = \max_{|w|=n} \mathrm{cost}_M(w)$$

Exemple

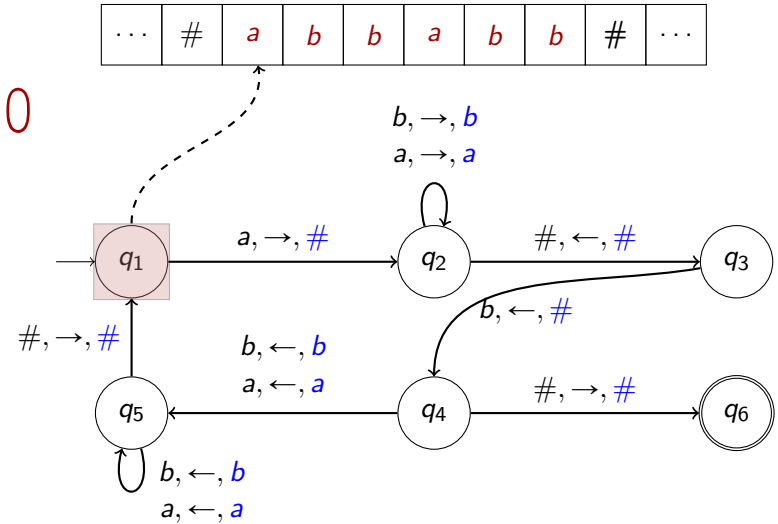
Exemple

Exemple

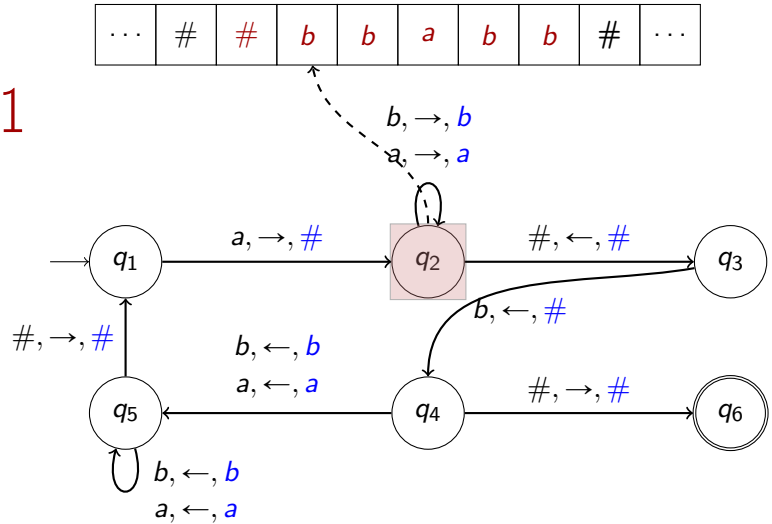
⋯	#	a	b	b	a	b	b	#	⋯



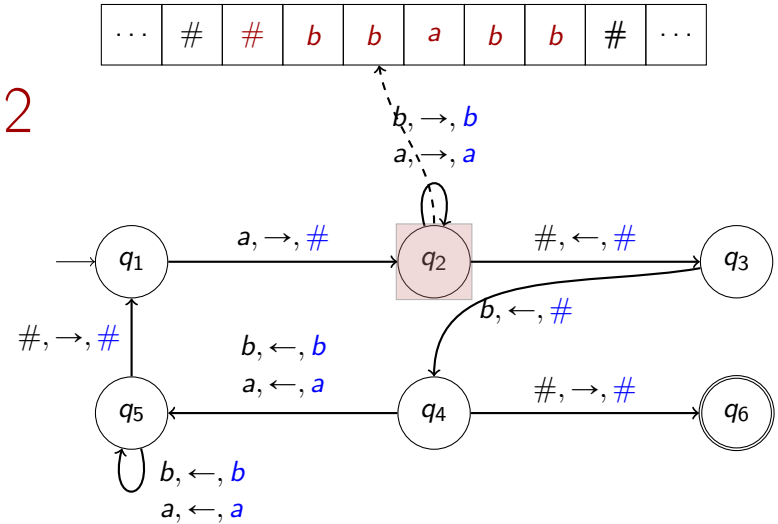
Exemple



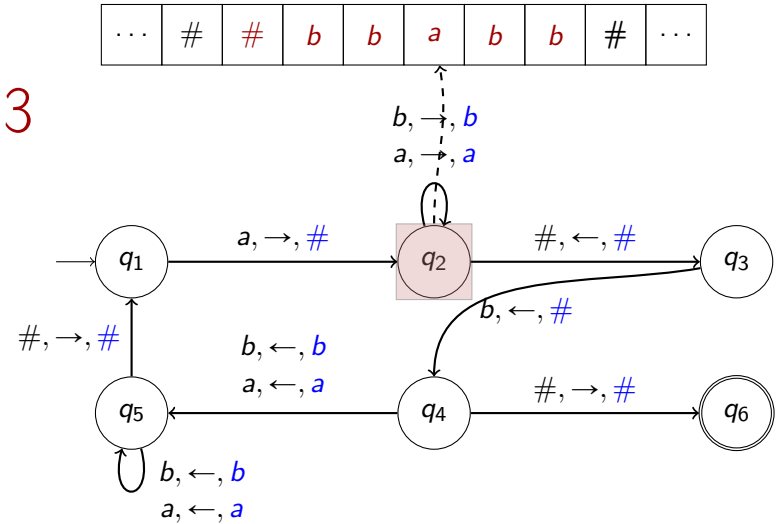
Exemple



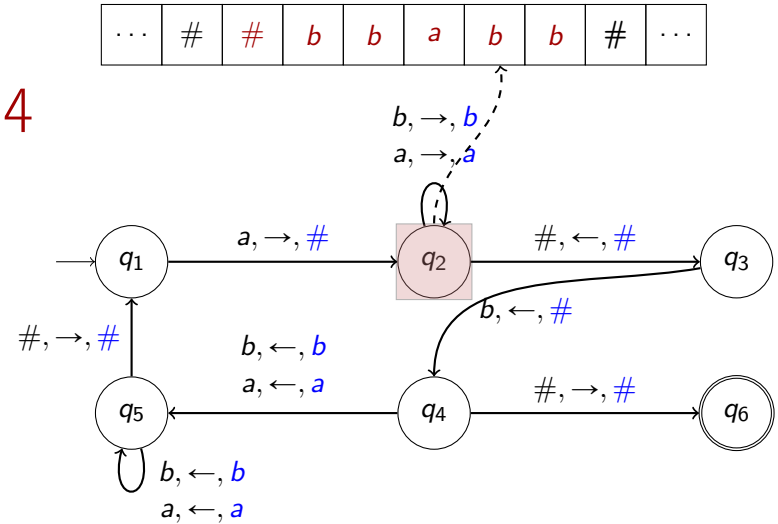
Exemple



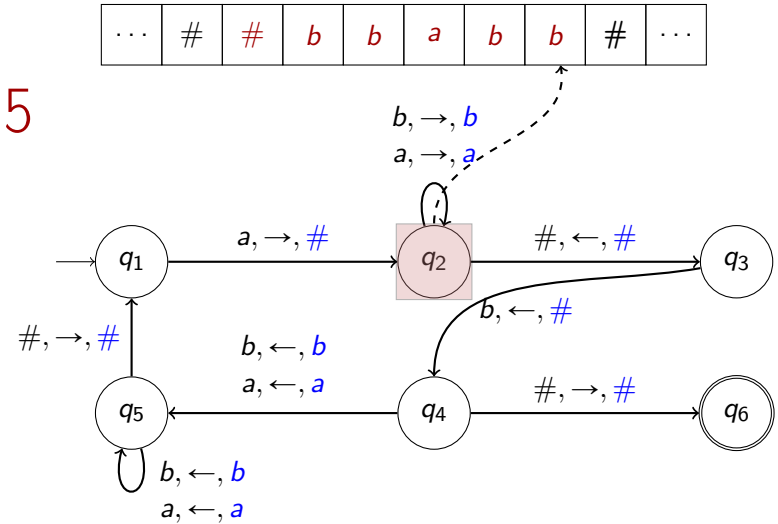
Exemple



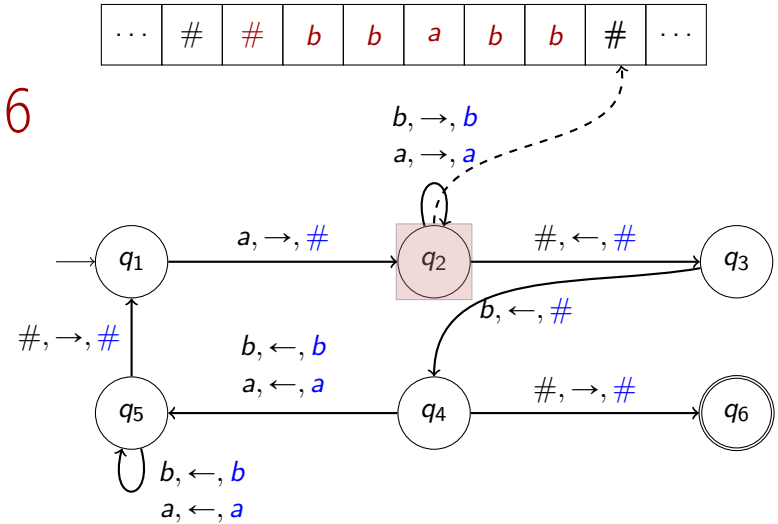
Exemple



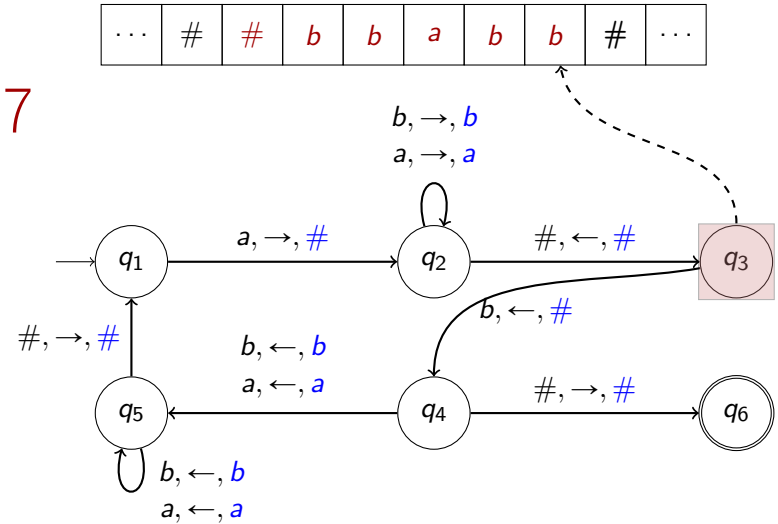
Exemple



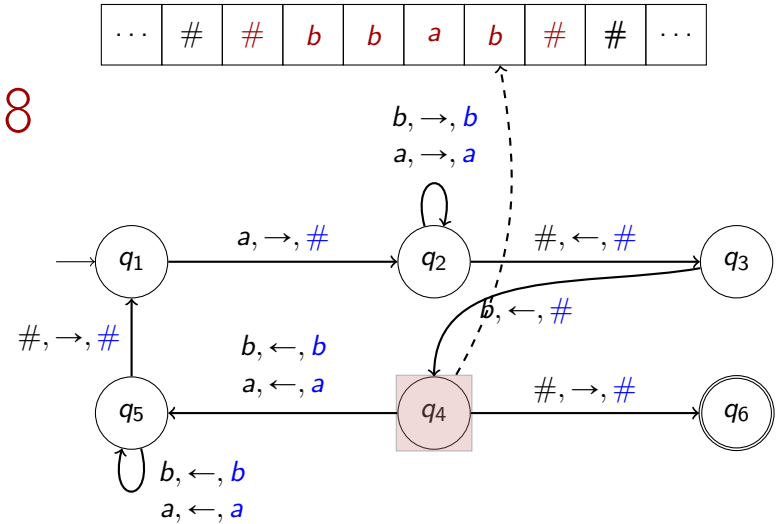
Exemple



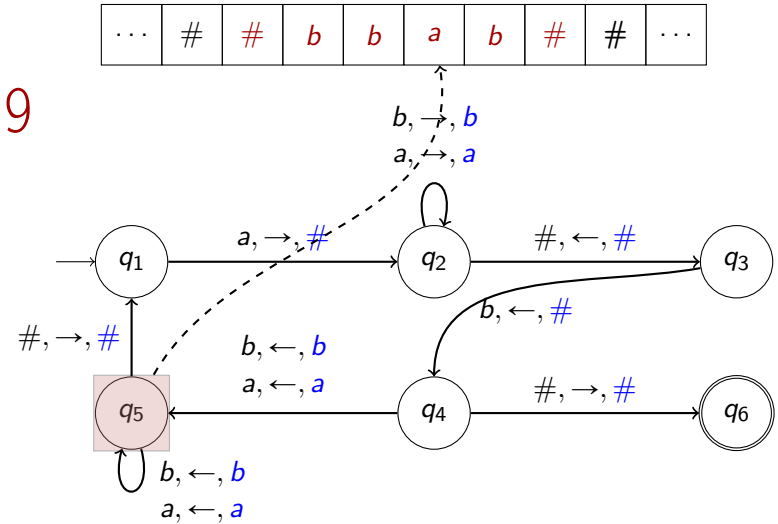
Exemple



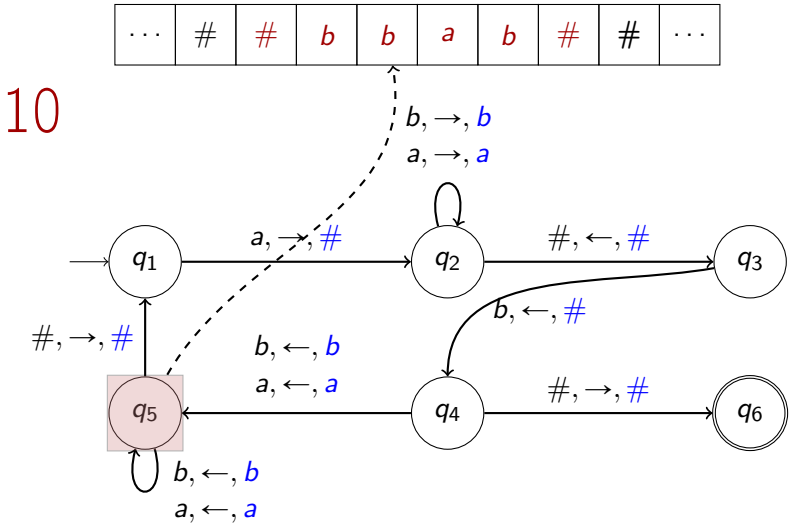
Exemple



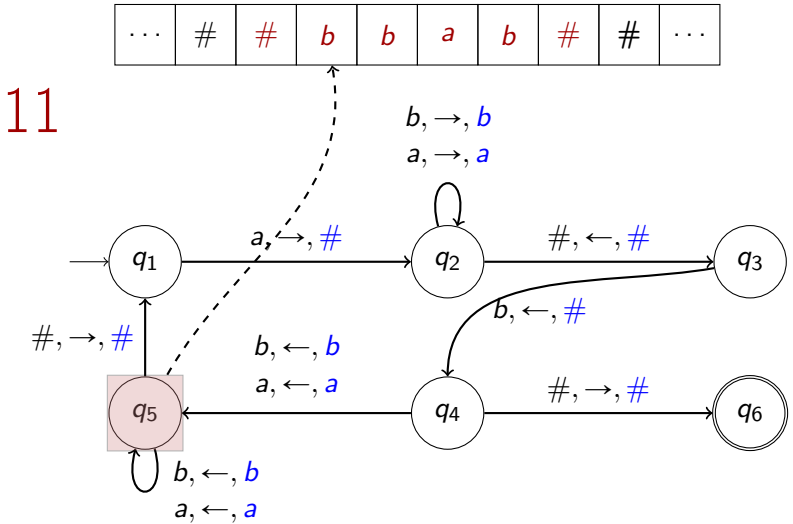
Exemple



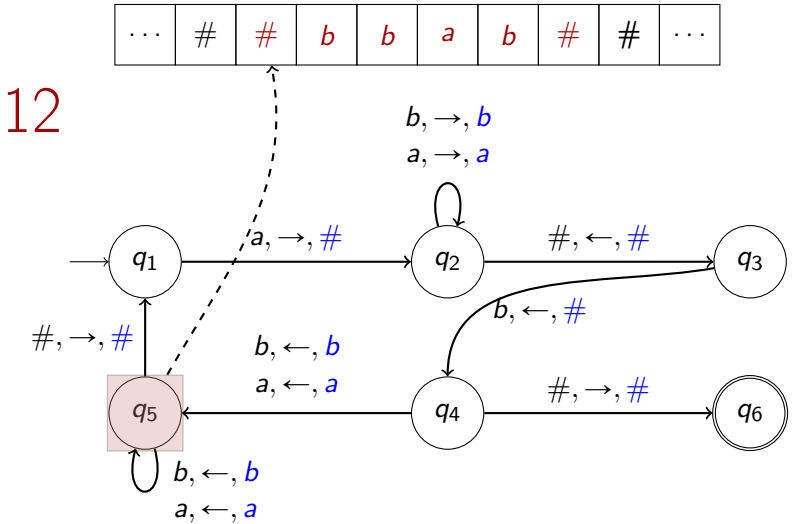
Exemple



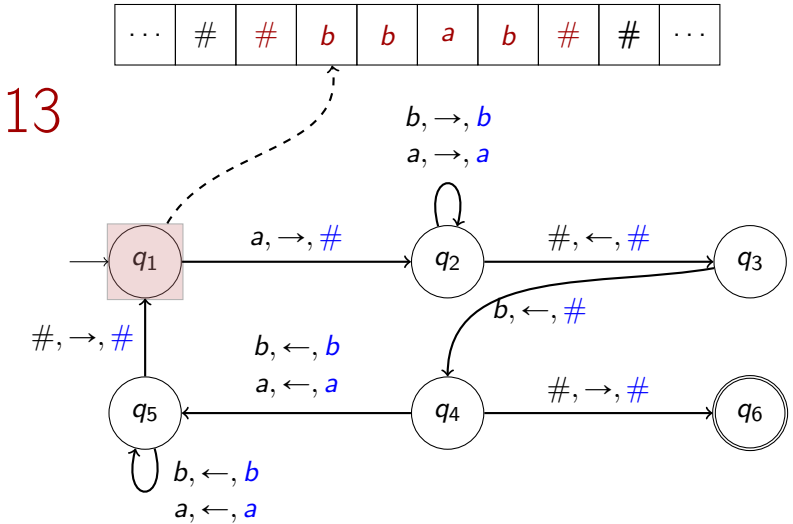
Exemple



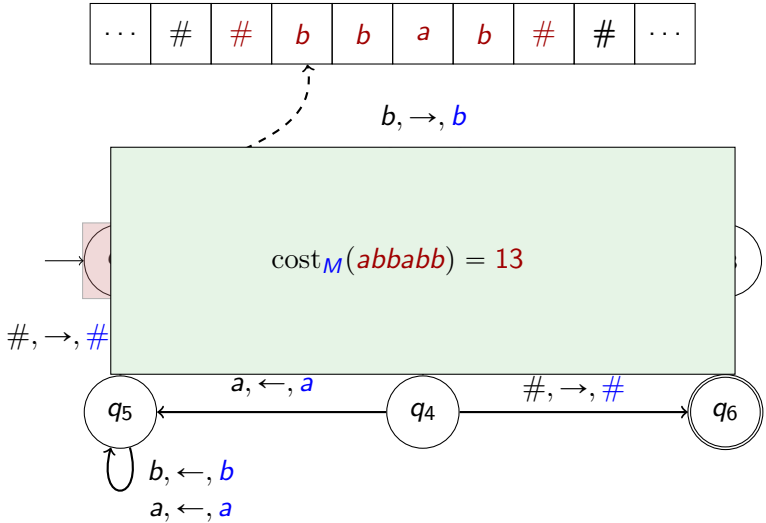
Exemple



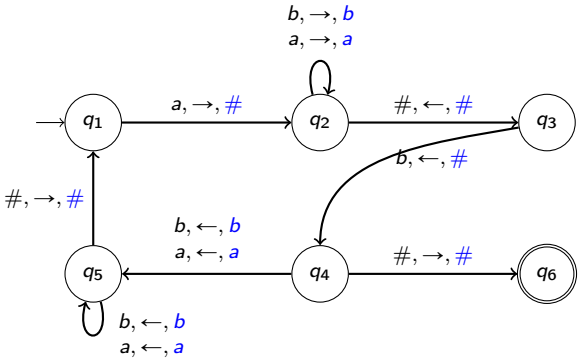
Exemple



Exemple

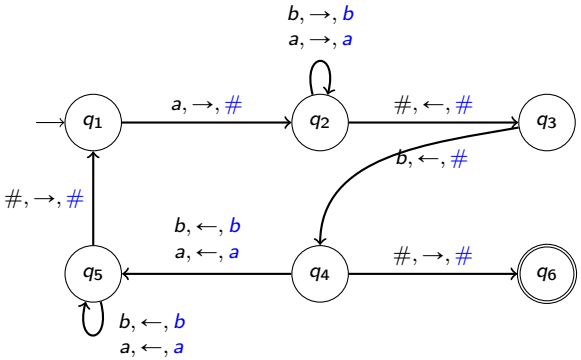


Exemple (suite)



$$C_M(n) = \max_{|w|=n} \text{cost}_M(w)$$

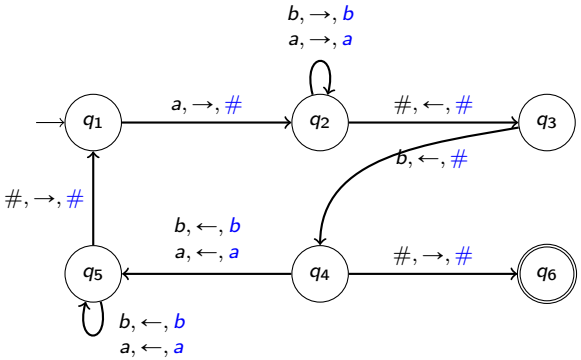
Exemple (suite)



$$C_M(n) = \max_{|w|=n} \text{cost}_M(w)$$

$$\begin{aligned}
\text{cost}_M(aaa) &= 4 \\
\text{cost}_M(aab) &= 11 \\
\text{cost}_M(aba) &= 4 \\
\text{cost}_M(abb) &= 8 \\
\text{cost}_M(baa) &= 0 \\
\text{cost}_M(bab) &= 0 \\
\text{cost}_M(bba) &= 0 \\
\text{cost}_M(bbb) &= 0
\end{aligned}$$

Exemple (suite)



$$C_M(n) = \max_{|w|=n} \text{cost}_M(w)$$

$$\begin{aligned}
\text{cost}_M(aaa) &= 4 \\
\text{cost}_M(aab) &= 11 \\
\text{cost}_M(aba) &= 4 \\
\text{cost}_M(abb) &= 8 \\
\text{cost}_M(baa) &= 0 \\
\text{cost}_M(bab) &= 0 \\
\text{cost}_M(bba) &= 0 \\
\text{cost}_M(bbb) &= 0
\end{aligned}$$

$$C_M(3) = 11$$

Machine de Turing polynomiale (en temps)

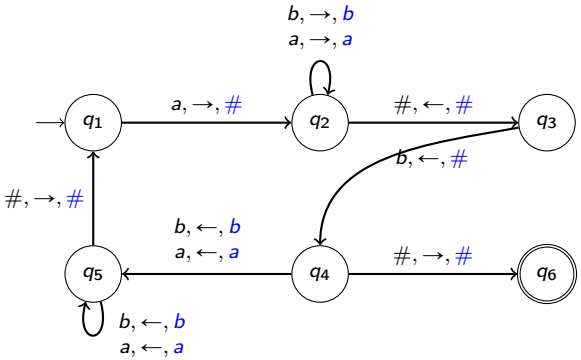
Une machine de Turing déterministe M dont tous les calculs sont finis est dite polynomiale (en temps) s'il existe un polynôme Q tel que pour tout n,

$$C_M(n) \leqslant Q(n).$$

Machine de Turing polynomiale (en temps)

Une machine de Turing déterministe M dont tous les calculs sont finis est dite polynomiale (en temps) s'il existe un polynôme Q tel que pour tout n,

$$C_M(n) \leqslant Q(n).$$



$$C_M(n) \leqslant (n+2)^2$$

La machine est polynomiale.

Exemple 2

On considère sur $\{a, b\}$ le langage des mots ayant autant de a que de b.

```
def NaNb(w): #w est une chaine ne comportant que des a et des b
    compteur=0
    for x in w :
        if x=='a':
            compteur+=1
        else :
            compteur-=1
return (compteur == 0)
```

Le nombre d'opérations est proportionel à la longueur de w. Le problème est dans **P**.

Exemple 3

On s'interesse au problème suivant : *Etant donné un entier en binaire, est-il premier ?*

```
def Prime(n):
    for x in range(2,n-1):
        if n%x == 0 :
            return False
    return True
```

- L'algorithme précédent est-il polynomial ?

Exemple 3

On s'interesse au problème suivant : *Etant donné un entier en binaire, est-il premier ?*

```
def Prime(n):
    for x in range(2,n-1):
        if n%x == 0 :
            return False
    return True
```

- L'algorithme précédent est-il polynomial ?
- Quelle est la taille de l'entrée (en fonction de n) ?
- Combien y a-t-il d'instructions (en fonction de n) ?
- L'algorithme précédent est-il polynomial ?

Classe P

Un problème est dans la classe **P** s'il existe une machine de Turing déterministe polynomiale en temps le décidant.

Classe P

Un problème est dans la classe **P** s'il existe une machine de Turing déterministe polynomiale en temps le décidant.

- Recherche d'un élément dans un tableau, existence d'un chemin de petite taille dans un graphe,...
- Classe des problèmes dits *peu coûteux* à résoudre.

Classe P

Un problème est dans la classe **P** s'il existe une machine de Turing déterministe polynomiale en temps le décidant.

- Recherche d'un élément dans un tableau, existence d'un chemin de petite taille dans un graphe,...
- Classe des problèmes dits *peu coûteux* à résoudre.
- La classe **P** est stable par changement de modèle de calcul equivalent à un programme (sauf quantique) : machine de Turing à plusieurs rubans, faisant du sur place, modèle RAM, réécriture, C, Java, OCaml, etc.

EXPTIME

Définition

Un problème est dans **EXPTIME** s'il existe une machine de Turing déterministe qui le décide en temps $O(2^{P(n)})$ où n est la taille de l'entrée et P un polynôme (qui ne dépend que du problème).

On peut définir des complexités encore plus élevées. On sait que :

$$\text{P} \subsetneq \text{EXPTIME} \subseteq \text{Décidables}$$

.

NP

Langage décidé

Une machine de Turing M décide une langage L si

- Pour tout mot $w \in L$, il existe un calcul maximal fini acceptant w.
- Pour tout mot $w \notin L$, tous les calculs maximaux sont finis et aucun n'est acceptant.

Une machine de Turing est polynomiale s'il existe un entier k tel que pour tout mot w, tout calcul maximal sur w a une longueur dans $O(|w|^k)$.

NP

Langage décidé

Une machine de Turing M décide une langage L si

- Pour tout mot $w \in L$, il existe un calcul maximal fini acceptant w.
- Pour tout mot $w \notin L$, tous les calculs maximaux sont finis et aucun n'est acceptant.

Une machine de Turing est polynomiale s'il existe un entier k tel que pour tout mot w, tout calcul maximal sur w a une longueur dans $O(|w|^k)$.

Un problème est dans la classe **NP** (polynomial non-déterministe) s'il existe une machine de Turing **non déterministe** polynomiale le décidant.

Coût d'exécution sur un mot

Etant donnés une machine de Turing **non déterministe** M **décidant** un langage L, et un mot w de longueur $|w|$ **appartenant à** L, on note

$$\mathrm{cost}_M(w)$$

la longueur (nombre de transitions utilisées) du **plus court** calcul maximal sur w.

1.

Coût d'exécution sur un mot

Etant donnés une machine de Turing **non déterministe** M **décidant** un langage L, et un mot w de longueur $|w|$ **appartenant à** L, on note

$$\mathrm{cost}_M(w)$$

la longueur (nombre de transitions utilisées) du **plus court** calcul maximal sur w.

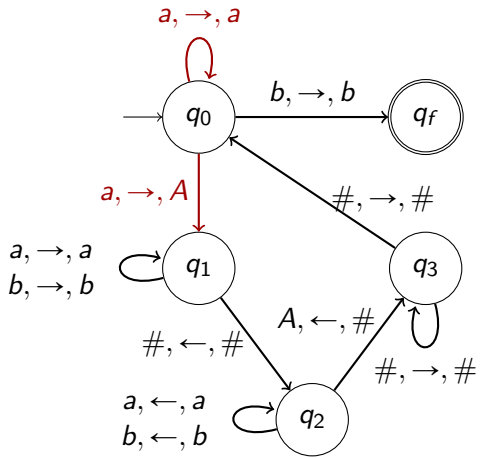
La complexité en temps de M est la fonction [1]

$$C_M(n) = \max_{|w|=n, w \in L} \mathrm{cost}_M(w)$$

1. On impose parfois que $C_M(n) \geqslant n+1$

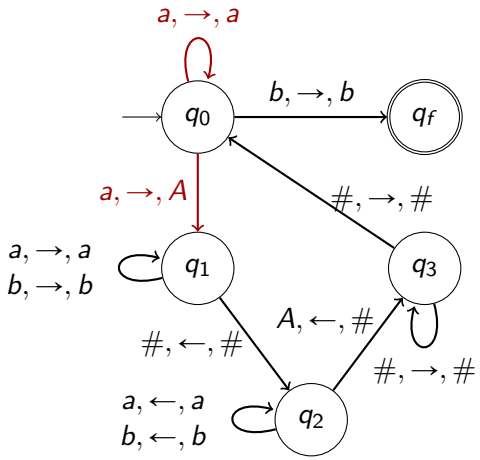
Exemple

···	#	#	#	#	#	#	#	#	···



Exemple

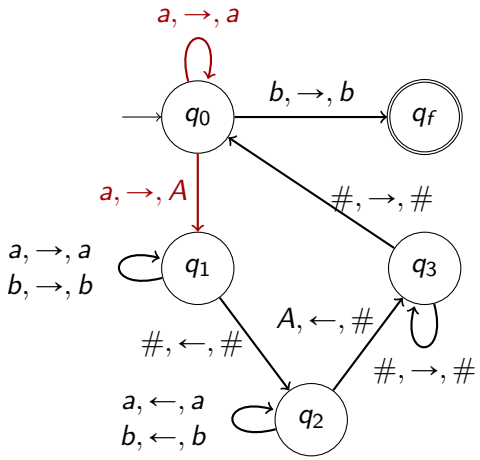
···	#	#	#	#	#	#	#	#	···



- Reconnaît l'ensemble des mots qui contiennent au moins un b.

Exemple

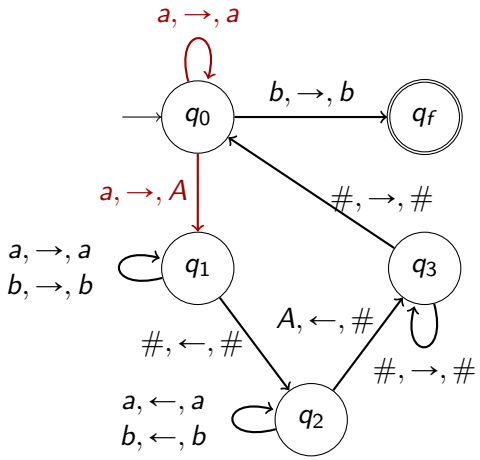
···	#	a	b	#	#	#	#	#	···



- Reconnaît l'ensemble des mots qui contiennent au moins un b.
- Les longueurs des chemins possibles pour ab sont 2 et 6. Donc $\mathrm{cost}_M(ab) = 2$.

Exemple

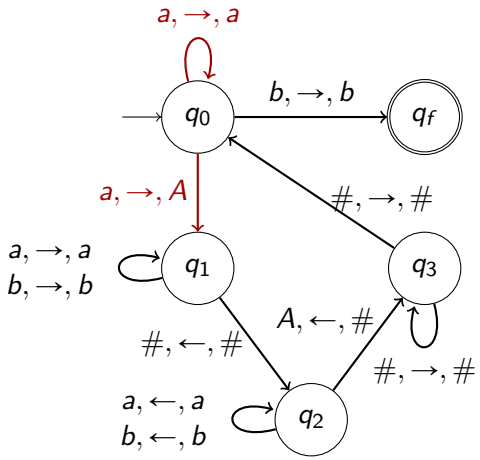
···	#	A	b	#	#	#	#	#	···



- Reconnaît l'ensemble des mots qui contiennent au moins un b.
- Les longueurs des chemins possibles pour ab sont 2 et 6. Donc $\mathrm{cost}_M(ab) = 2$.

Exemple

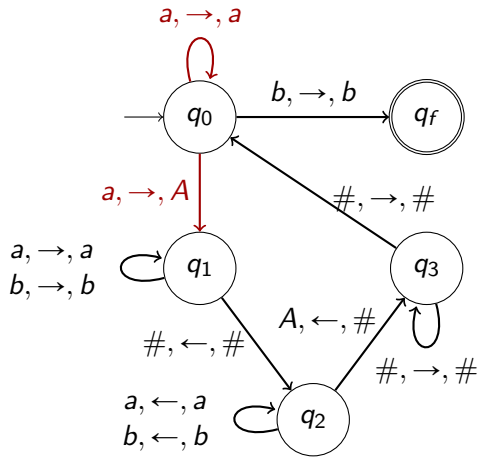
···	#	#	b	#	#	#	#	#	···



- Reconnaît l'ensemble des mots qui contiennent au moins un b.
- Les longueurs des chemins possibles pour ab sont 2 et 6. Donc $\mathrm{cost}_M(ab) = 2$.

Exemple

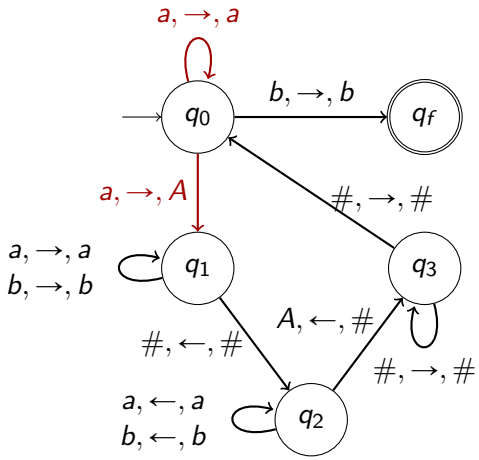
⋯	#	a	a	b	#	#	#	#	⋯



- Reconnaît l'ensemble des mots qui contiennent au moins un b.
- Les longueurs des chemins possibles pour ab sont 2 et 6. Donc $\mathrm{cost}_M(ab) = 2$.
- Les longueurs des chemins possibles pour aab sont 3, 7, 12, 15. Donc $\mathrm{cost}_M(aab) = 3$.

Exemple

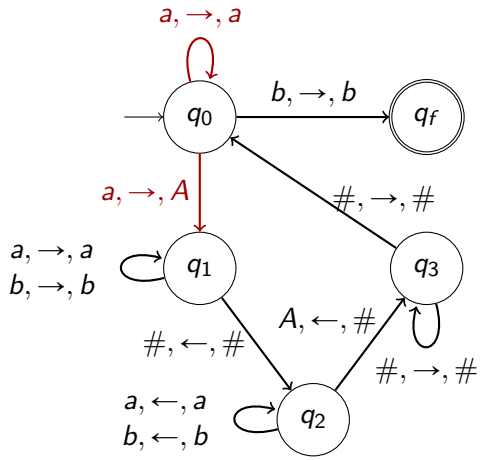
···	#	a	a	b	#	#	#	#	···



- Reconnaît l'ensemble des mots qui contiennent au moins un b.
- Les longueurs des chemins possibles pour ab sont 2 et 6. Donc $\mathrm{cost}_M(ab) = 2$.
- Les longueurs des chemins possibles pour aab sont 3, 7, 12, 15. Donc $\mathrm{cost}_M(aab) = 3$.
- $\mathrm{cost}_M(a^{n-1}bw) = n$.

Exemple

⋯	#	a	a	b	#	#	#	#	⋯



- Reconnaît l'ensemble des mots qui contiennent au moins un b.
- Les longueurs des chemins possibles pour ab sont 2 et 6. Donc $\mathrm{cost}_M(ab) = 2$.
- Les longueurs des chemins possibles pour aab sont 3, 7, 12, 15. Donc $\mathrm{cost}_M(aab) = 3$.
- $\mathrm{cost}_M(a^{n-1}bw) = n$.
- $C_M(n) = n$ (le plus long chemin acceptant $a^{n-1}b$ une longueur de l'ordre de n^2).

Exemple

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$, tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$ tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

Sous-mot

Une mot u est un sous-mot de v si on peut obtenir u à partir de v en effaçant des lettre de v.
Le mot *abbbaaa* est un sous-mot de *abbababababaaa*.

Exemple

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$, tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$ tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

Sous-mot

Une mot u est un sous-mot de v si on peut obtenir u à partir de v en effaçant des lettre de v.
Le mot *abbbaaa* est un sous-mot de *abbababababaaa*.

Réfléchir à un aglorithme (au sens classique) pour ce problème. Quelle est sa complexité ?

Exemple (machine de Turing)

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$ tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$, tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

Trouver une machine non déterminste polynomiale.

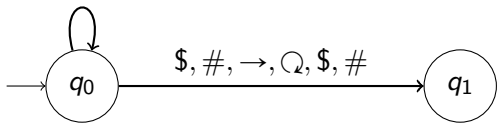
Exemple (machine de Turing)

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$ tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$, tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

On commence par lire w_0 et on écrit de façon non déterministe a ou b sur le second ruban. On obtient un mot x de la taille de w_0.

$a, \#, \rightarrow, \rightarrow, a, a$
$b, \#, \rightarrow, \rightarrow, b, b$
$a, \#, \rightarrow, \rightarrow, a, b$
$b, \#, \rightarrow, \rightarrow, b, a$



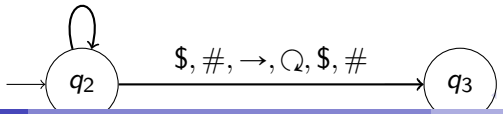
Exemple (machine de Turing)

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$ tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$, tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

On vérifie que le mot x sur le second ruban est sous-mot de tous les autres.

$a, a, \rightarrow, \rightarrow, a, a$
$b, b, \rightarrow, \rightarrow, b, b$
$a, a, \rightarrow, \circlearrowright, a, a$
$a, b, \rightarrow, \circlearrowright, a, b$
$b, b, \rightarrow, \circlearrowright, b, b$
$b, a, \rightarrow, \circlearrowright, b, a$



Exemple (machine de Turing)

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$ tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$, tel que

- $|x| = |w_0|$,
- x est un sous-mot de chaque w_i pour $i > 0$?

La machine est non-déterministe polynomiale, le problème est dans **NP**.

Inclusions

On peut démontrer que les machines de Turing sont déterminisables avec un coût exponentiel.

On sait alors que :

$$P \subseteq NP \subseteq EXPTIME \subseteq \text{Décidables}$$

.
Ces classes sont indépendants du nombre de rubans (et même en pratique du modèle de calcul utilisé).
La grande question est :

$$P=NP\,?$$

Exercice

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$, tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$ tel que

- $|x| = |w_0|$,
- Chaque w_i pour $i > 0$ est un sous mot de x ?

Sous-mot

Une mot u est un sous-mot de v si on peut obtenir u à partir de v en effaçant des lettre de v.
Le mot *abbbaaa* est un sous-mot de *abbababababaaa*.

Exercice

Données $w_0\$w_1\$\ldots\$w_k\$$ un mot sur $\{a, b, \$\}$, tel que $w_i \in \{a, b\}^*$.
Question : Existe-t-il $x \in \{a, b\}^*$ tel que

- $|x| = |w_0|$,
- Chaque w_i pour $i > 0$ est un sous mot de x ?

Sous-mot

Une mot u est un sous-mot de v si on peut obtenir u à partir de v en effaçant des lettre de v.
Le mot *abbbaaa* est un sous-mot de *abb*a*b*a*b*a*b*a*aa*.

Décrire une machine de Turing non-déterministe polynomiale pour répondre à ce problème.

Plan

1. Hiérarchie du temps
2. Hiérarchie en espace
3. Certificats et classe NP
4. Algorithmes non-déterministes
5. Classe complémentaires et complétude

Machine de Turing à entrée/sortie

Machines à entrées/sorties

Une machine de Turing à entrée/sortie est une machine qui possède

- Un ruban d'entrée en lecture seule (on ne peut pas réécrire),
- Un ruban de sortie en écriture seule (on ne peut se déplacer que vers la droite),
- des rubans de travail (éventuellement aucun) sans restriction.

L'**espace** utilisé pour un calcul maximal d'une machine à entrée-sortie est les maximum des somme des longeurs des mots écrits sur les rubans de travail lors de ce calcul (on compte les # si la tête de lecture se déplace sans écrire).

PSPACE, NPSPACE, NLOGSPACE

- **PSPACE** : machine de Turing déterministe à entrée sortie utilisant une mémoire bornée par une fonction polynomiale en la taille de l'entrée.
- **NPSPACE** : même chose sans déterminisme.
- **NLOGSPACE** : machine de Turing non-déterministe à entrée sortie utilisant une mémoire bornée par une fonction polynomiale en le logarithme de la taille de l'entrée.

Exemple de langage **NLOGSPACE** : $\{a^n b^n \mid n > 0\}$

PSPACE, NPSPACE, NLOGSPACE

- **PSPACE** : machine de Turing déterministe à entrée sortie utilisant une mémoire bornée par une fonction polynomiale en la taille de l'entrée.
- **NPSPACE** : même chose sans déterminisme.
- **NLOGSPACE** : machine de Turing non-déterministe à entrée sortie utilisant une mémoire bornée par une fonction polynomiale en le logarithme de la taille de l'entrée.

Exemple de langage **NLOGSPACE** : $\{a^n b^n \mid n > 0\}$

On reverra **NLOGSPACE** dans la section *Algorithmique non-déterministe*.

Inclusions

Il faut du temps pour occuper l'espace :

$$\mathrm{P} \subseteq \mathrm{PSPACE} \quad \text{et} \quad \mathrm{NP} \subseteq \mathrm{NPSPACE}$$

Inclusions

Il faut du temps pour occuper l'espace :

$$\mathrm{P} \subseteq \mathrm{PSPACE} \quad \text{et} \quad \mathrm{NP} \subseteq \mathrm{NPSPACE}$$

Théorème de Savitch

$\mathrm{PSPACE} = \mathrm{NPSPACE}$

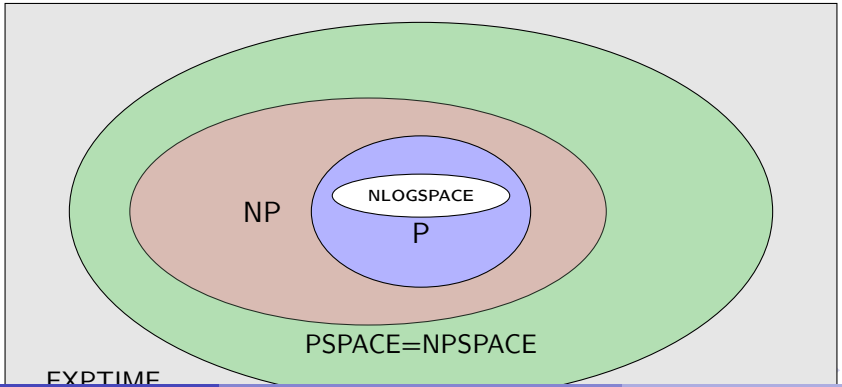
Inclusions

Il faut du temps pour occuper l'espace :

$$P \subseteq PSPACE \quad \text{et} \quad NP \subseteq NPSPACE$$

Théorème de Savitch

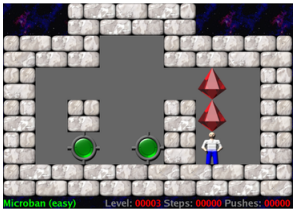
PSPACE = NPSPACE



Exemples

Exemples de problèmes dans **PSPACE** que l'on *pense* ne pas être dans **NP**.

- Etant donnés deux automates $\mathcal{A}_1$ et $\mathcal{A}_2$, reconnaissent-ils le même langage ?
- Etant donnés deux expression régulières, codent-elles le même langage ?
- Sokoban



Plan

1. Hiérarchie du temps
2. Hiérarchie en espace
3. Certificats et classe NP
4. Algorithmes non-déterministes
5. Classe complémentaires et complétude

Certificats

Soit L un langage de Σ^*. Un certificat pour L est une fonction calculable (décidable) C de $\Sigma^* \times \Gamma^*$ dans $\{0, 1\}$ telle que

$$w \in L \quad \text{si et seulement s'il existe } x \in \Gamma^* \text{ tel que } C(w, x) = 1$$

Un langage pour lequel il existe un certificat est dit **semi-décidable**.

Exemple 1

Non Universalité d'un automate

Donnée : Un automate fini $\mathcal{A}$ sur Σ.
Sortie : 1 si $L(\mathcal{A}) \neq \Sigma^*$, 0 sinon.

L est l'ensemble des automates $\mathcal{A}$ sur Σ qui ne reconnaissent pas tous les mots.

Exemple 1

Non Universalité d'un automate

Donnée : Un automate fini $\mathcal{A}$ sur Σ.
Sortie : 1 si $L(\mathcal{A}) \neq \Sigma^*$, 0 sinon.

L est l'ensemble des automates $\mathcal{A}$ sur Σ qui ne reconnaissent pas tous les mots.

Certificat C

Donnée : Un automate fini $\mathcal{A}$ sur Σ et un mot u de Σ^*
Sortie : 1 si $u \notin L(\mathcal{A})$, 0 sinon.

Exemple 1

Non Universalité d'un automate

Donnée : Un automate fini $\mathcal{A}$ sur Σ.
Sortie : 1 si $L(\mathcal{A}) \neq \Sigma^*$, 0 sinon.

L est l'ensemble des automates $\mathcal{A}$ sur Σ qui ne reconnaissent pas tous les mots.

Certificat C

Donnée : Un automate fini $\mathcal{A}$ sur Σ et un mot u de Σ^*
Sortie : 1 si $u \notin L(\mathcal{A})$, 0 sinon.

$L(\mathcal{A}) \neq \Sigma^*$ si et seulement s'il existe $u \in \Sigma^*$ tel que $u \notin L(\mathcal{A})$.
$L(\mathcal{A}) \neq \Sigma^*$ si et seulement s'il existe $u \in \Sigma^*$ tel que $C(\mathcal{A}, u) = 1$.

Exemple 2

Non Totaulogie

Donnée : Une formule booléenne φ.
Sortie : 1 si φ n'est pas toujours vraie, 0 sinon.

Exemple 2

Non Totaulogie

Donnée : Une formule booléenne φ.
Sortie : 1 si φ n'est pas toujours vraie, 0 sinon.

Certificat C

Donnée : Une formule booléenne φ et un mot u de $\{0,1\}^*$
Sortie : 1 si $\neg\varphi(u(1), u(2), \ldots, u(k))$ est vraie, 0 sinon.

$\varphi = \varphi(x_1, \ldots, x_k)$.

Exemple 2

Non Totaulogie

Donnée : Une formule booléenne φ.
Sortie : 1 si φ n'est pas toujours vraie, 0 sinon.

Certificat C

Donnée : Une formule booléenne φ et un mot u de $\{0,1\}^*$
Sortie : 1 si $\neg\varphi(u(1), u(2), \ldots, u(k))$ est vraie, 0 sinon.

$\varphi = \varphi(x_1, \ldots, x_k)$.

φ n'est pas toujours vraie ssi il existe x_1,...,x_k dans $\{0,1\}$ tels que $\varphi(x_1, \ldots, x_k) = 0$

Exemple 2

Non Totaulogie

Donnée : Une formule booléenne φ.
Sortie : 1 si φ n'est pas toujours vraie, 0 sinon.

Certificat C

Donnée : Une formule booléenne φ et un mot u de $\{0,1\}^*$
Sortie : 1 si $\neg\varphi(u(1), u(2), \ldots, u(k))$ est vraie, 0 sinon.

$\varphi = \varphi(x_1, \ldots, x_k)$.

φ n'est pas toujours vraie ssi il existe $x_1,\ldots,x_k$ dans $\{0,1\}$ tels que $\varphi(x_1, \ldots, x_k) = 0$
ssi il existe $x_1,\ldots,x_k$ dans $\{0,1\}$ tels que $\neg\varphi(x_1, \ldots, x_k) = 1$

Exemple 2

Non Totaulogie

Donnée : Une formule booléenne φ.
Sortie : 1 si φ n'est pas toujours vraie, 0 sinon.

Certificat C

Donnée : Une formule booléenne φ et un mot u de $\{0,1\}^*$
Sortie : 1 si $\neg\varphi(u(1), u(2), \ldots, u(k))$ est vraie, 0 sinon.

$\varphi = \varphi(x_1, \ldots, x_k)$.

φ n'est pas toujours vraie ssi il existe $x_1,\ldots,x_k$ dans $\{0,1\}$ tels que $\varphi(x_1, \ldots, x_k) = 0$
ssi il existe $x_1,\ldots,x_k$ dans $\{0,1\}$ tels que $\neg\varphi(x_1, \ldots, x_k) = 1$
ssi il existe $u \in \{0,1\}^*$ tel que $\neg\varphi(u(1), \ldots, u(k)) = 1$

Exemple 2

Non Totaulogie

Donnée : Une formule booléenne φ.
Sortie : 1 si φ n'est pas toujours vraie, 0 sinon.

Certificat C

Donnée : Une formule booléenne φ et un mot u de $\{0,1\}^*$
Sortie : 1 si $\neg\varphi(u(1), u(2), \ldots, u(k))$ est vraie, 0 sinon.

$\varphi = \varphi(x_1, \ldots, x_k)$.

φ n'est pas toujours vraie ssi il existe x_1,...,x_k dans $\{0,1\}$ tels que $\varphi(x_1, \ldots, x_k) = 0$
ssi il existe x_1,...,x_k dans $\{0,1\}$ tels que $\neg\varphi(x_1, \ldots, x_k) = 1$
ssi il existe $u \in \{0,1\}^*$ tel que $\neg\varphi(u(1), \ldots, u(k)) = 1$
ssi il existe $u \in \{0,1\}^*$ tel que $C(\varphi, u) = 1$

Exemple 3

Chemin Hamiltonien

Donnée : Un graphe fini G

Sortie : 1 si G contient un chemin hamiltonien (passant une et une seule fois par chaque sommet), 0 sinon.

Exemple 3

Chemin Hamiltonien

Donnée : Un graphe fini G

Sortie : 1 si G contient un chemin hamiltonien (passant une et une seule fois par chaque sommet), 0 sinon.

Certificat C

Donnée : Un graphe fini G et une séquence de sommets $v_1, \ldots, v_n$ de longueur le nombre de sommets de G.

Sortie : 1 si $v_1, \ldots, v_n$ est un chemin du graphe et les v_i sont deux à deux distincts, 0 sinon.

Certificats et classe NP

Un langage L de Σ^* est dans NP (classe des problèmes non-déterministe polynomiaux) si et seulement s'il existe un certificat C pour L tel que

- C est calculable en temps polynomial,
- il existe un polynome Q tel que pour tout $w \in L$ il existe un $x \in \Gamma^*$ de longueur au plus $Q(|w|)$ tel que $C(w, x) = 1$

Exemple NP

Non Totaulogie

Donnée : Une formule booléenne φ.
Sortie : 1 si φ n'est pas toujours vraie, 0 sinon.

Certificat C

Donnée : Une formule booléenne φ et un mot u de $\{0,1\}^{|\varphi|}$
Sortie : 1 si $\neg\varphi(u(1), u(2), \ldots, u(k))$ est vraie, 0 sinon.

$\varphi = \varphi(x_1, \ldots, x_k)$.

- Evaluer une expression booléenne se fait en temps polynomial.
- La longueur de u est bornée par la longueur φ (ici $Q(n) = n$).

Contre-Exemple NP

Non Universalité d'un automate

Donnée : Un automate fini $\mathcal{A}$ sur Σ.
Sortie : 1 si $L(\mathcal{A}) \neq \Sigma^*$, 0 sinon.

L est l'ensemble des automates $\mathcal{A}$ sur Σ qui ne reconnaissent pas tous les mots.

Certificat C

Donnée : Un automate fini $\mathcal{A}$ sur Σ et un mot u de Σ^*
Sortie : 1 si $u \notin L(\mathcal{A})$, 0 sinon.

- Le certificat C est calculable en temps polynomial.
- Mais on n'a pas de borne polynomiale pour $|u|$ par rapport à la taille de $\mathcal{A}$.

Le problème est connu pour être PSPACE-complet.

Plan

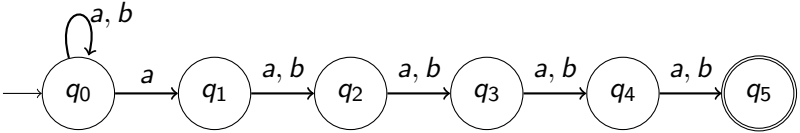
1. Hiérarchie du temps
2. Hiérarchie en espace
3. Certificats et classe NP
4. Algorithmes non-déterministes
5. Classe complémentaires et complétude

Exercice (rappel)

- Dessiner un automate non déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.

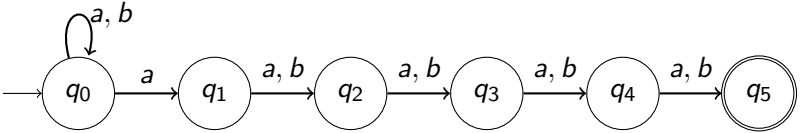
Exercice (rappel)

- Dessiner un automate non déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.



Exercice (rappel)

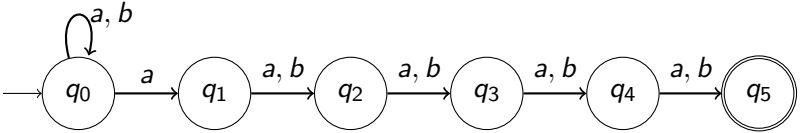
- Dessiner un automate non déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.



- Dessiner un automate déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.

Exercice (rappel)

- Dessiner un automate non déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.



- Dessiner un automate déterministe qui reconnaît l'ensemble des mots sur l'alphabet $\{a, b\}$ dont la cinquième lettre avant la fin est un a.
  Il faut au moins 32 états...

Moral de l'exercice

Grace à l'automate non déterministe de l'exercice on peut savoir sans faire une fastidieuse déterminisation que :

- Le langage est reconnaissable par un automate (déterministe),
- Qu'il existe un tel automate avec au plus 64 états.

L'algorithmique non-déterministe permet d'avoir accès facilement à des information sur la complexité d'un problème, sans avoir à le résoudre explicitement.
Si on a besoin d'un programme effectif, il faudra faire le travail... mais on aura des indications.

Algorithmes non déterministes

Les machines de Turing non-déterministes sont à l'algorithmique non-déterministe ce que les machines de Turing détermnistes sont à l'algorithmique : super pour la théorie, mais peu utilisables en pratique.

Algorithmes non déterministes

Les machines de Turing non-déterministes sont à l'algorithmique non-déterministe ce que les machines de Turing détermnistes sont à l'algorithmique : super pour la théorie, mais peu utilisables en pratique.

- Dans un algorithmes non-deterministes, on a le droit d'utiliser la fontion Choisir(X) où X est un ensemble de caractères fixé. Elle retourne, de façon non déterministe, un élément de X.
- Un algorithme non déterministe décide un problème si
  - Pour toute instance positive du problème, il existe des choix possibles qui font que l'algorithme retourne vrai.
  - Pour toute instance négative du problème, quelque soient les choix, l'algorithme retourne faux.

A mettre en parallèle avec l'existence d'un calcul réussi.

Exemple 1

Donnée : un entier n codé en binaire
Question : n est-il un entier non premier ?

```
def NotPrime(n): #n est une chaine de 0,1
    d=""
    for i in range(len(n)):
        d+=Choisir([0,1])
        if int(d)==0 or int(d)==1 or int(d)==n:
            return False
        if int(n)%int(d) == 0:
            return True
        return False
```

Exemple 1

Donnée : un entier n codé en binaire
Question : n est-il un entier non premier ?

```
def NotPrime(n): #n est une chaine de 0,1
    d=""
    for i in range(len(n)):
        d+=Choisir([0,1])
        if int(d)==0 or int(d)==1 or int(d)==n:
            return False
        if int(n)%int(d) == 0:
            return True
        return False
```

Si n n'est pas premier, en choisissant le codage binaire d'un diviseur strict de n, l'algorithme répondra Vrai.
Si n est premier, l'algorithme répondra toujours Faux.

Exemple 1

Donnée : un entier n codé en binaire
Question : n est-il un entier non premier ?

```
def NotPrime(n): #n est une chaine de 0,1
    d=""
    for i in range(len(n)):
        d+=Choisir([0,1])
        if int(d)==0 or int(d)==1 or int(d)==n:
            return False
        if int(n)%int(d) == 0:
            return True
        return False
```

L'algorithme utilise un temps polynomial : le problème est dans **NP**.

Exemple 1 (suite)

En pratique on s'autorise des raccourcis (en s'assurant qu'on peut toujours revenir aux choix de caractères).

Donnée : un entier n codé en binaire
Question : n est-il un entier non premier ?

```
def NotPrime(n): #n est une chaine de 0,1
        d=Choisir(Entier plus petit que n)
        if d==0 or d==1 or d==n:
            return False
        if n%d == 0:
            return True
        return False
```

Exemple 2

Donnée : un graphe orienté (V, E), p et q deux sommets.
Question : Existe-t-il un chemin de p à q ?

```
def Accessible(V,E,p,q):
    s=p
    for x in range(len(V)):
        if s==q :
            return True
        r=Choisir(V)
        if (s,r) not in E:
            return False
        s=r
    return False
```

Exemple 2

Donnée : un graphe orienté (V, E), p et q deux sommets.
Question : Existe-t-il un chemin de p à q ?

```
def Accessible(V,E,p,q):
    s=p
    for x in range(len(V)):
        if s==q :
            return True
        r=Choisir(V)
        if (s,r) not in E:
            return False
        s=r
    return False
```

On utilise deux variables locales pour les sommet r et s et une pour le compteur x.
Chacune utilise un espace logarithmique en la taille de l'entrée.
Le problème est dans **NLOGSPACE**.

Exercices

1. Montrer que le problème suivant est dans **NP**.

SAT

Donnée : Une formule booléenne $F(x_1, \ldots, x_k)$
Question : Existe-t-il une instanciation des variables qui rende la formule vraie.
(Remarque : la méthode des tables de vérité est exponentielle en k)

2. Montrer que le problème suivant est dans **NP**.

Chemin Hamiltonien

Donnée : Un graphe fini orienté $G = (V, E)$
Question : Existe-t-il un chemin dans G G qui passe une et une seule fois par tous les sommets.

Exercices

1. Montrer que le problème suivant est dans **NLOGSPACE**.

Triangle

Donnée : Un graphe fini non orienté $G = (V, E)$
Question : Existe-t-il trois sommets qui forment un triangle ? (en relationd eux à deux).

2. Montrer que le problème esuivant est dans **NLOGSPACE**.

Circuit

Donnée : Un graphe fini orienté $G = (V, E)$
Question : Existe-t-il un circuit dans G.

Exercice

.

Membership

Données : Un automate fini sur $\{a, b\}$ et un mot $w \in \{a, b\}$.
Question : Le mot w est-il accepté par l'automate.

1. Montrer que si l'on se restreint aux automates déterministes, le sous-problème associé est dans **P**.
2. Donner un algorithme ou une idé d'algorithme pour le problème **Membership**, quand l'automate n'est pas déterministe. Estimer grossièrement la complexité de l'algorithme.

Exercice

.

Membership

Données : Un automate fini sur $\{a, b\}$ et un mot $w \in \{a, b\}$.
Question : Le mot w est-il accepté par l'automate.

1. Montrer que si l'on se restreint aux automates déterministes, le sous-problème associé est dans **P**.
2. Donner un algorithme ou une idé d'algorithme pour le problème **Membership**, quand l'automate n'est pas déterministe. Estimer grossièrement la complexité de l'algorithme.
3. Montrer que **Membership** est dans **NLOGSPACE**. Que peut-t-on en déduire sur la complexité déterministe du problème ?

Exercice

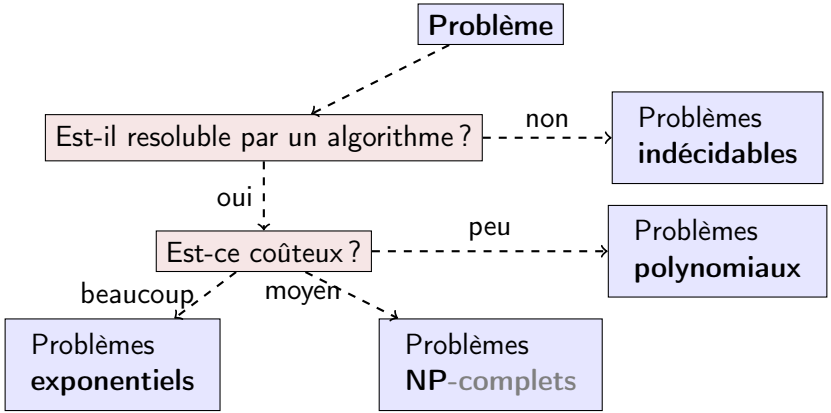
.

Membership

Données : Un automate fini sur $\{a, b\}$ et un mot $w \in \{a, b\}$.
Question : Le mot w est-il accepté par l'automate.

1. Montrer que si l'on se restreint aux automates déterministes, le sous-problème associé est dans **P**.
2. Donner un algorithme ou une idé d'algorithme pour le problème **Membership**, quand l'automate n'est pas déterministe. Estimer grossièrement la complexité de l'algorithme.
3. Montrer que **Membership** est dans **NLOGSPACE**. Que peut-t-on en déduire sur la complexité déterministe du problème ?
4. Donner un algorithme polynomial pour **Membership**.

Architecture du cours (schématique)



Métaheuristiques, approximations, etc.

Solveurs SAT

Solveurs IP (VS)

Plan

1. Hiérarchie du temps
2. Hiérarchie en espace
3. Certificats et classe NP
4. Algorithmes non-déterministes
5. Classe complémentaires et complétude

Problemes complémentaires

Soit Pr un problème. Le problème complémentaire de Pr est le problème associé au complémentaire du langage de Pr.

Problemes complémentaires

Soit Pr un problème. Le problème complémentaire de Pr est le problème associé au complémentaire du langage de Pr.

Problème Pr

Donnée : un mot w sur $\{a, b\}$.
Question : Est-ce que w a autant de a que de b ?

Problème complémentaire de Pr

Donnée : un mot w sur $\{a, b\}$.
Question : Est-ce que w n'a pas autant de a que de b ?

Mais à quoi ça sert ?

Problemes complémentaires et classes déterministes

On note co-$\mathcal{C}$ la classe des langages complémentaires des langages de la classe $\mathcal{C}$.

```
def MonAlgoEnPython(w):
    ....
    return True  # Changer en False
    ....
    return False # Changer en True
```

Pour une classe $\mathcal{C}$ de langages définie par des machines déterministes (sans conditions sur les états finaux), alors co-$\mathcal{C} = \mathcal{C}$.

On a P $=$ co$-$P, PSPACE $=$ co$-$PSPACE, etc.

Problemes complémentaires et classes non-déterministes

Donnée : un entier n codé en binaire
Question : n est-il un entier non premier ?

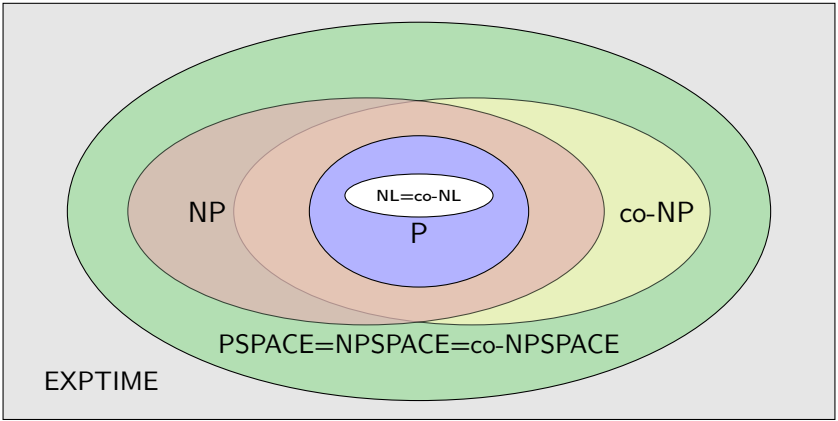
```
def NotPrime(n): #n est une chaine de 0,1
        d=Choisir(Entier plus petit que n)
        if d==0 or d==1 or d==n:
            return False # True
        if n%d == 0:
            return True  # False
        return False
```

Si on change `True` en `False` et réciproquement, ça ne donne pas un algorithme non déterministe pour le problème complémentaire (dans l'exemple, tous les entiers sont acceptés).

Hierarchie

On peut démontrer que :

$\text{NLOGSPACE} = \text{co} - \text{NLOGSPACE}$ et $\text{NPSPACE} = \text{co} - \text{NPSPACE}$



Problème difficile/complet pour une classe

Pour une classe de complexité $\mathcal{C}$, un problème est dit difficile s'il est *plus coûteux à résoudre* que tout problème de $\mathcal{C}$. On dit qu'il est $\mathcal{C}$-difficile.

Un problème est complet pour $\mathcal{C}$ ou $\mathcal{C}$-complet si

- Il est dans $\mathcal{C}$ et
- Il est $\mathcal{C}$-difficile.

Qu'entend-on par *plus coûteux à résoudre* ?

Réduction : introduction

Résultat admis

On ne peut pas construire à la règle et au compas un heptagone régulier.

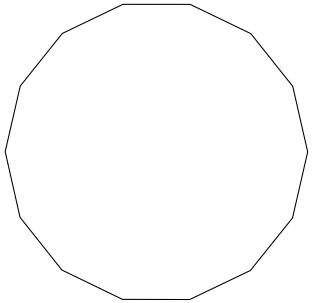
Peut-on construire à la règle et au compas un polygône régulier à 14 côté ?

Réduction : introduction

Résultat admis

On ne peut pas construire à la règle et au compas un heptagone régulier.

Peut-on construire à la règle et au compas un polygône régulier à 14 côté ?



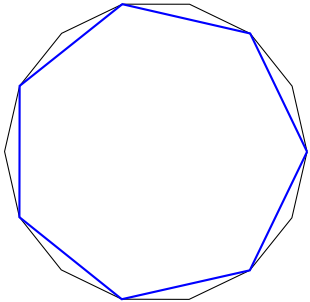
Non. Sinon on trace un 14-gone.

Réduction : introduction

Résultat admis

On ne peut pas construire à la règle et au compas un heptagone régulier.

Peut-on construire à la règle et au compas un polygône régulier à 14 côté ?



Non. Sinon on trace un 14-gone.

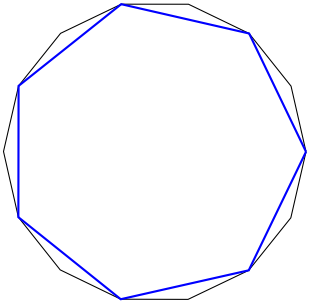
On relie à la règle un point sur deux.

Réduction : introduction

Résultat admis

On ne peut pas construire à la règle et au compas un heptagone régulier.

Peut-on construire à la règle et au compas un polygône régulier à 14 côté ?



Non. Sinon on trace un 14-gone.

On relie à la règle un point sur deux.

On obtientrait une construction d'un heptagone à la règle et au compas. Cela contredit le résultat.

Réduction : un exemple d'indécidabilité

On admet que le problème de l'arrêt est indécidable.

Problème de l'arrêt

Donnée : Un programme M et une entrée w de ce programme.
Question : L'execution de M sur w termine-t-elle ?

Comment en déduire que le problème suivant est indécidable ?

Problème de l'arrêt universel

Donnée : Un programme M.
Question : L'execution de M termine-t-elle pour toute entrée ?

Réduction : un exemple d'indécidabilité

On admet que le problème de l'arrêt est indécidable.

Problème de l'arrêt

Donnée : Un programme M et une entrée w de ce programme.
Question : L'execution de M sur w termine-t-elle ?

Comment en déduire que le problème suivant est indécidable ?

Problème de l'arrêt universel

Donnée : Un programme M.
Question : L'execution de M termine-t-elle pour toute entrée ?

Supposons que l'on a un programme `ArretUniversel` pour le problème AU.

```
def Arret(M,w):
    def NewM(x):    #NewM s'arrete pour toutes les entrees
        if x==w:    #ssi NewM(w) s'arrete ssi M(w) s'arrete
            return M(w)
        return False
    return ArretUniversel(NewM)
```

Réduction définition

Un problème A se réduit à un problème B s'il existe une application φ des entrées de A vers les entrées de B telle que

$$x \in A \quad \text{ssi} \quad \varphi(x) \in B$$

On peut résoudre le problème de l'appartenance à A en sachant calculer φ et en sachant résoudre le problème de l'appartenance à B.

Réduction définition

Un problème A se réduit à un problème B s'il existe une application φ des entrées de A vers les entrées de B telle que

$$x \in A \quad \text{ssi} \quad \varphi(x) \in B$$

On peut résoudre le problème de l'appartenance à A en sachant calculer φ et en sachant résoudre le problème de l'appartenance à B.

1. Si φ et B sont décidables, alors A est décidable.

Réduction définition

Un problème A se réduit à un problème B s'il existe une application φ des entrées de A vers les entrées de B telle que

$$x \in A \quad \text{ssi} \quad \varphi(x) \in B$$

On peut résoudre le problème de l'appartenance à A en sachant calculer φ et en sachant résoudre le problème de l'appartenance à B.

1. Si φ et B sont décidables, alors A est décidable.
2. Si φ et B sont calculable en temps polynomial, alors A est calculable en temps polynomial. On parle alors de réduction polynomiale et on note

$$A <<_{\mathrm{P}} B$$

NP-complet

Un problème B est **NP**-complet si

- Il est dans **NP**,
- Pour tout problème A de **NP**,

$$A <<_{\mathrm{P}} B$$

NP-complet

Un problème B est **NP**-complet si

- Il est dans **NP**,
- Pour tout problème A de **NP**,

$$A <<_P B$$

Le second point est difficile à monter, il y a une infinité de problèmes dans **NP**. En général, on montrer que

- Il est dans **NP**,
- Il existse un problème C **NP**-complet tel que ,

$$C <<_P B$$

(reste à trouver un premier problème NP-Complet, ce qu'on verra au prochain cours.)

Exemple du Sudoku

Coloration de Graphe est plus difficile que **Sudoku**.

Coloration

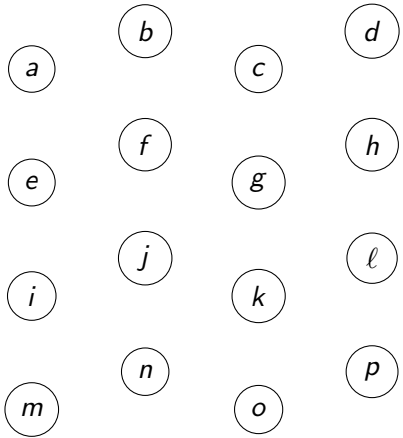
Données : Un graphe fini et k couleurs.
Question : Peut-on colorer chaque sommet du graphe avec une et une seule couleur de telle sorte que deux sommets voisins n'aient pas la même couleur ?

Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

Graphes de contraintes

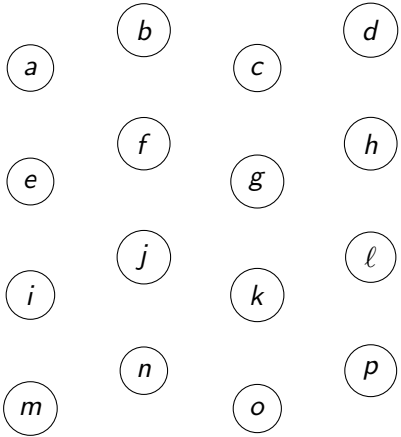
a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

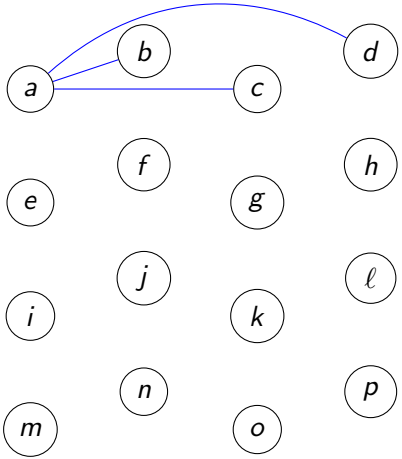
$\alpha - `\beta$ si valeurs différentes



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

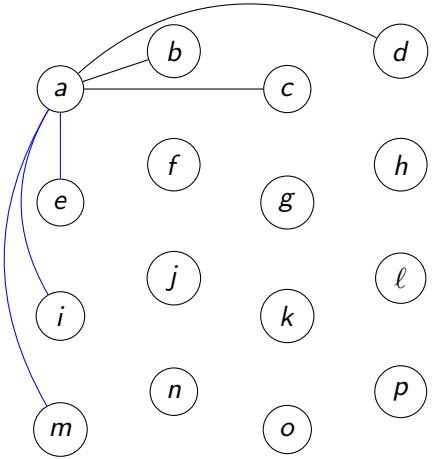
$\alpha - \text{`}\beta$ si valeurs différentes



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

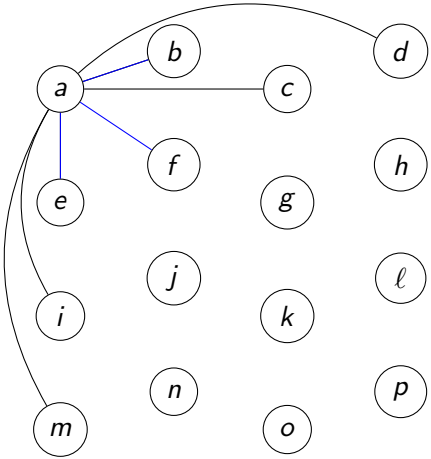
$\alpha - `\beta$ si valeurs différentes



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

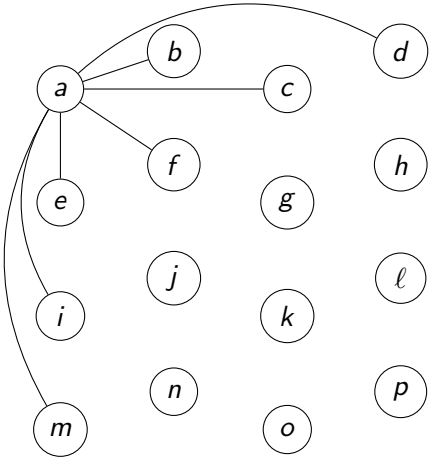
$\alpha - \text{`}\beta$ si valeurs différentes



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

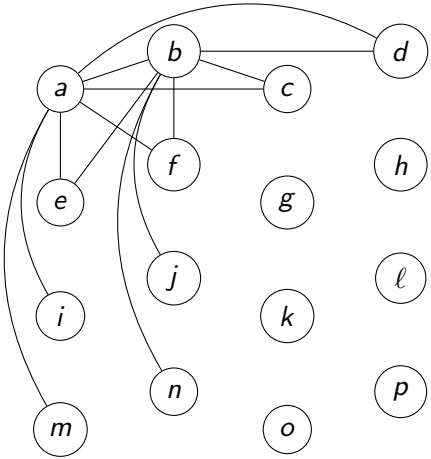
$\alpha - \text{`}\beta$ si valeurs différentes



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

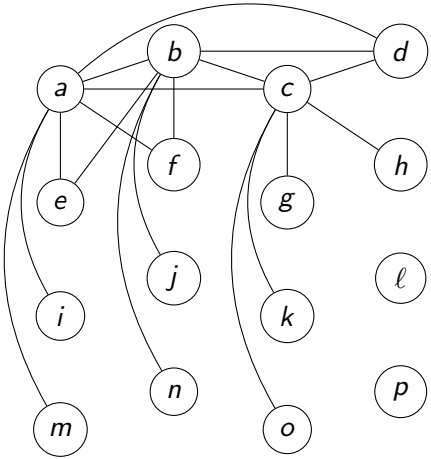
$\alpha - \text{`}\beta$ si valeurs différentes



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

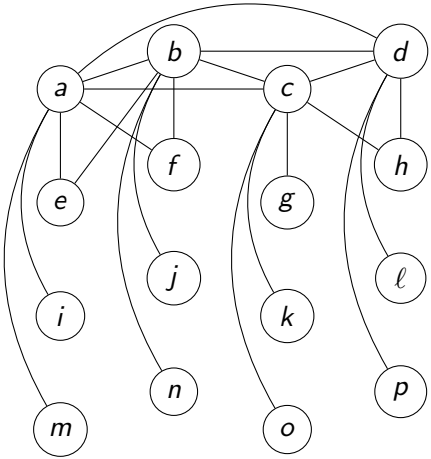
$\alpha - `\beta$ si valeurs différentes



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

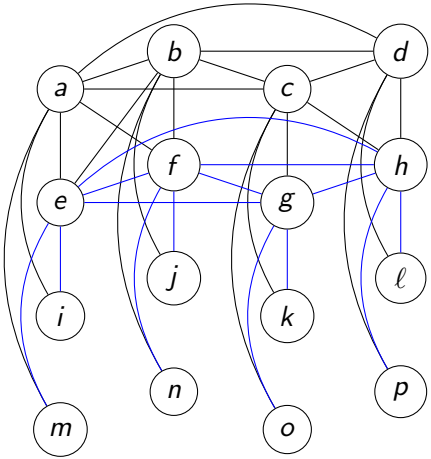
$\alpha - `\beta$ si valeurs différentes



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

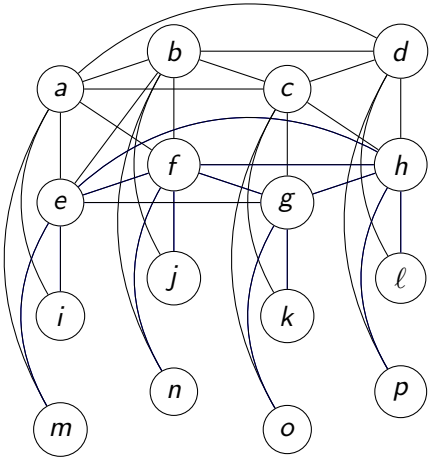
$\alpha - `\beta$ si valeurs différentes



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

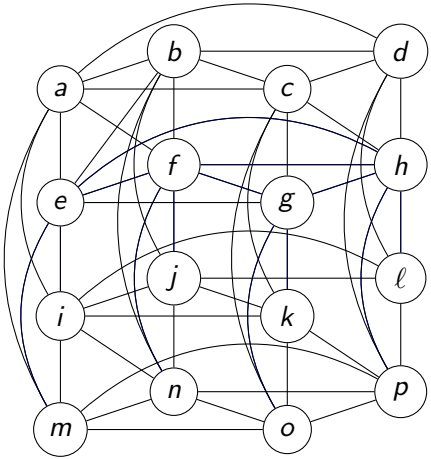
$\alpha - `\beta$ si valeurs différentes



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

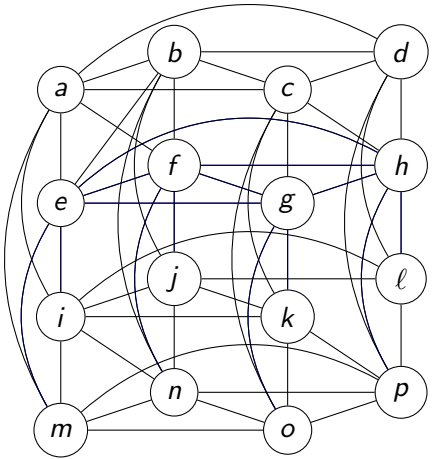
$\alpha - `\beta$ si valeurs différentes



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

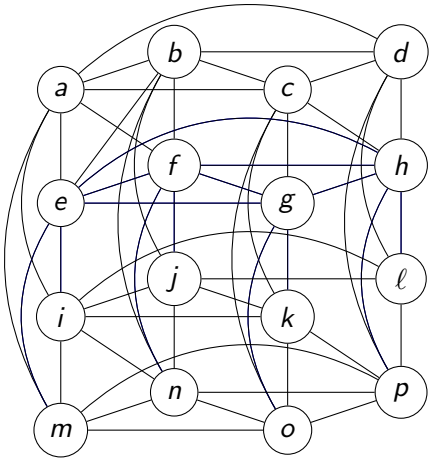
1 : bleu 2 : rouge
3 : vert 4 :orange



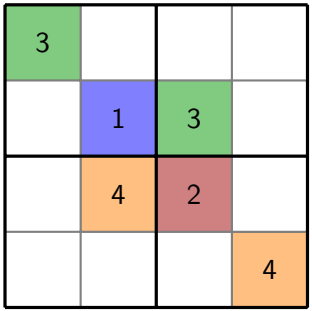
Graphes de contraintes

3			
	1	3	
	4	2	
			4

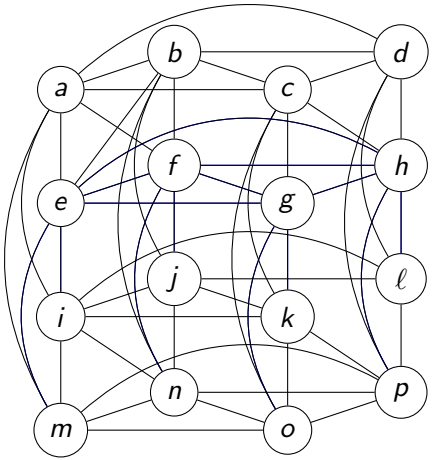
1 : bleu 2 : rouge
3 : vert 4 :orange



Graphes de contraintes



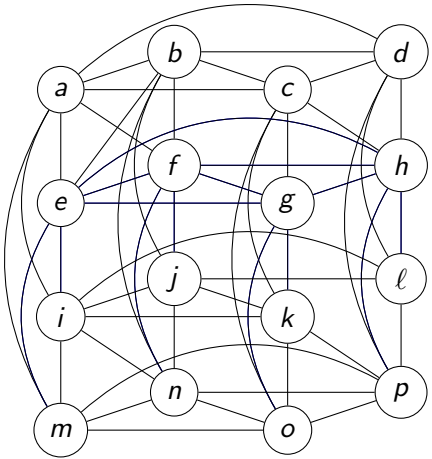
1 : bleu 2 : rouge
3 : vert 4 :orange



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

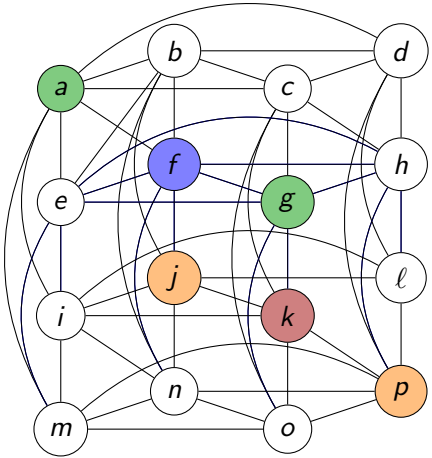
1 : bleu 2 : rouge
3 : vert 4 :orange



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

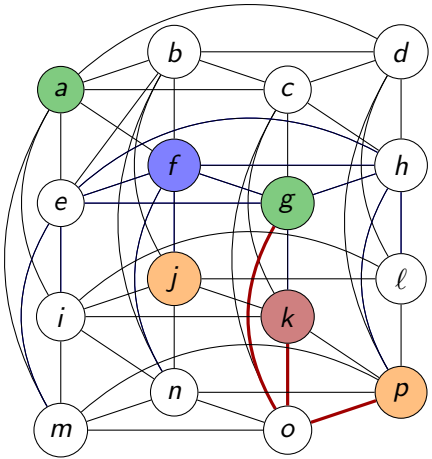
1 : bleu 2 : rouge
3 : vert 4 :orange



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

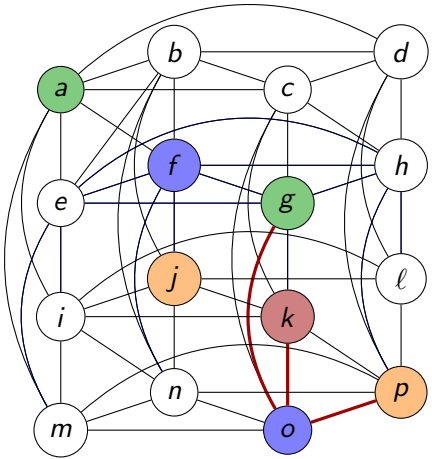
1 : bleu 2 : rouge
3 : vert 4 :orange



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

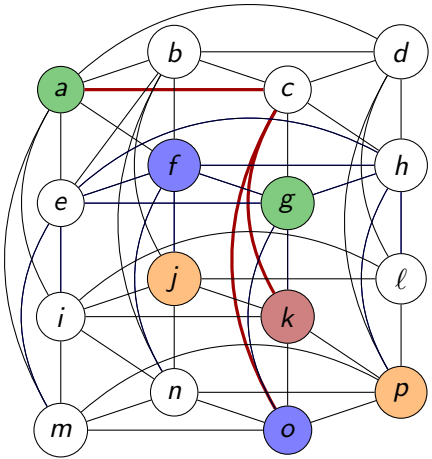
1 : bleu 2 : rouge
3 : vert 4 :orange



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

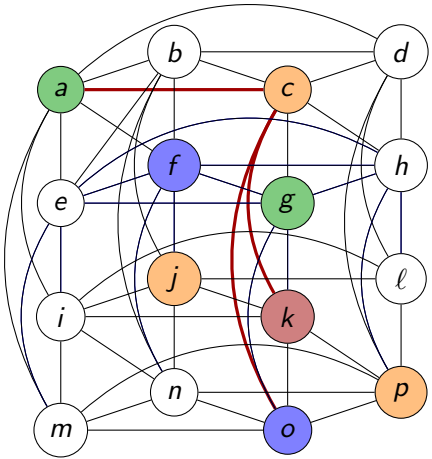
1 : bleu 2 : rouge
3 : vert 4 :orange



Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

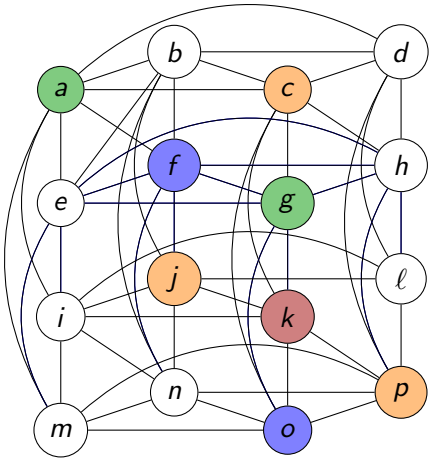
1 : bleu 2 : rouge
3 : vert 4 :orange



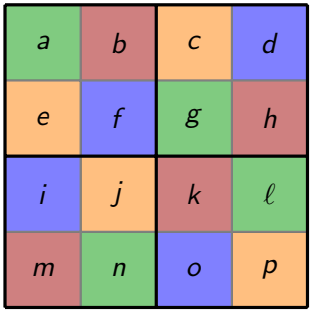
Graphes de contraintes

a	b	c	d
e	f	g	h
i	j	k	ℓ
m	n	o	p

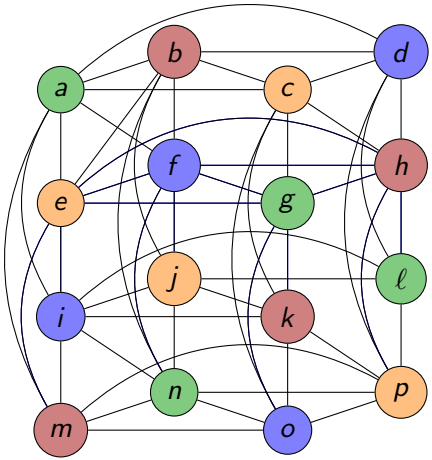
1 : bleu 2 : rouge
3 : vert 4 :orange



Graphes de contraintes

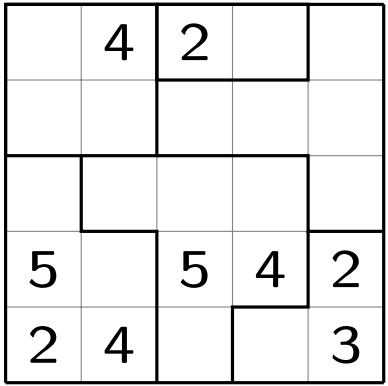


1 : bleu 2 : rouge
3 : vert 4 :orange



Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

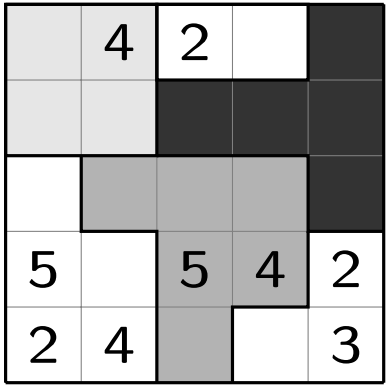


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

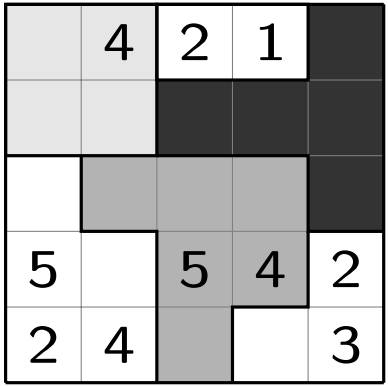


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

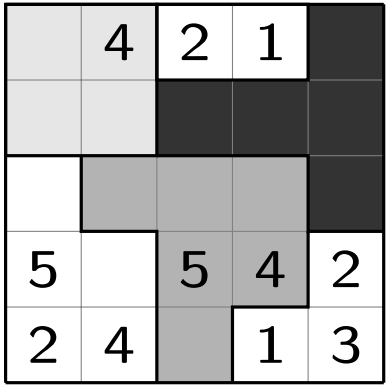


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

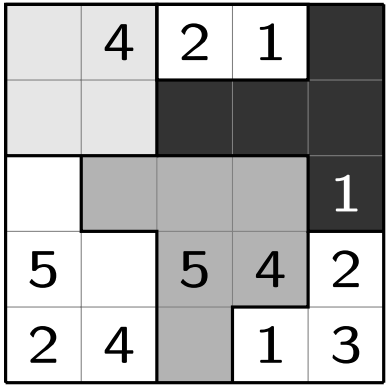


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

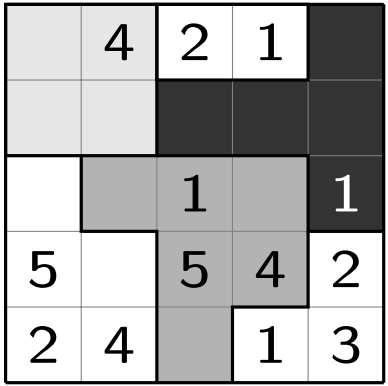


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

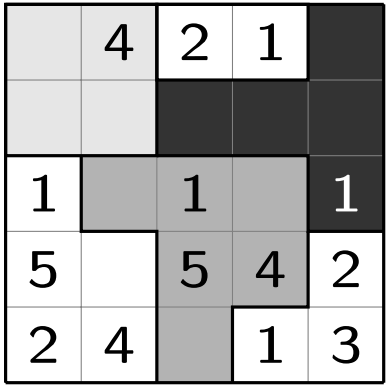


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

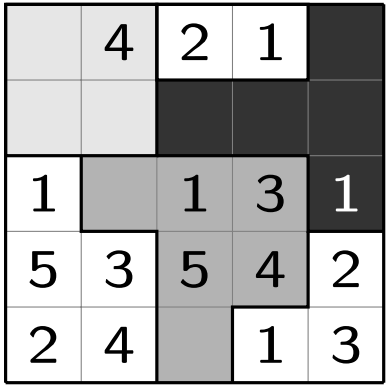


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

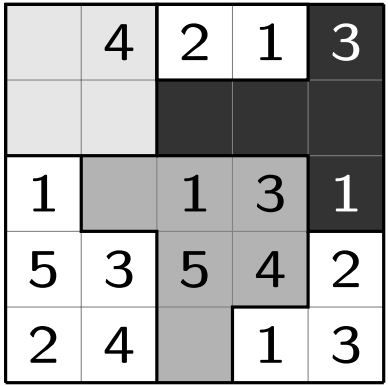


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

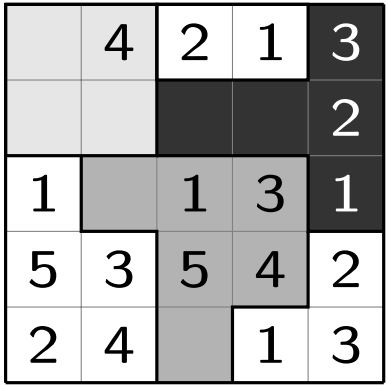


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

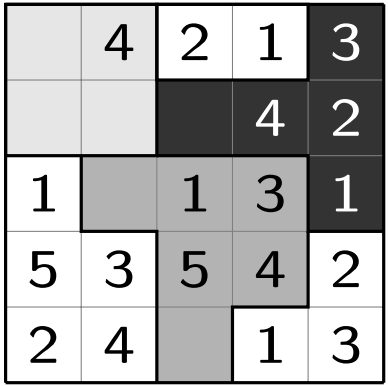


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

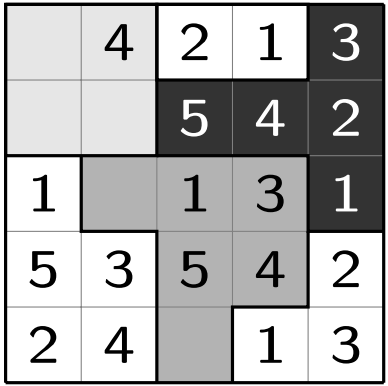


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

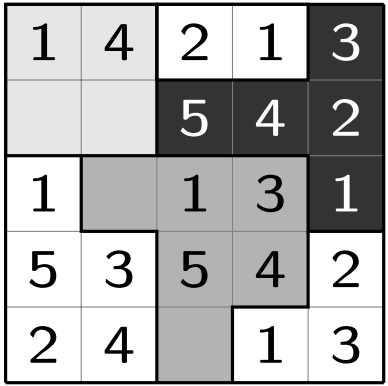


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

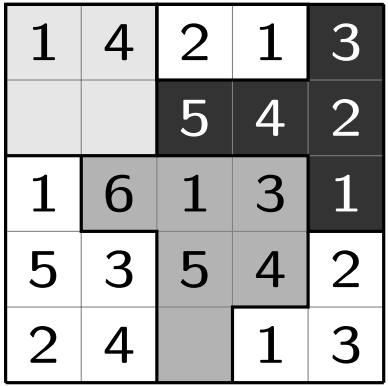


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

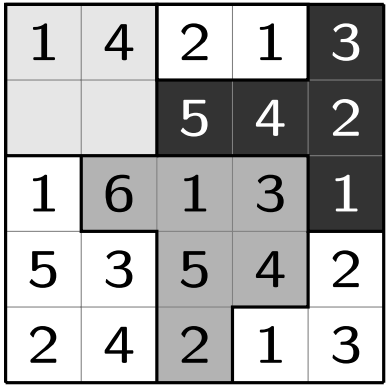


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

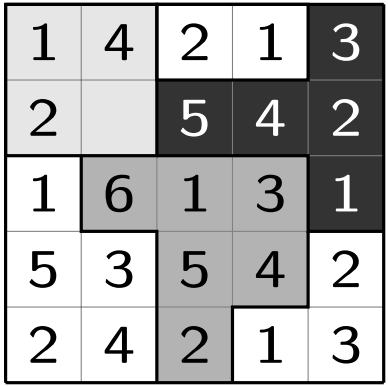


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

Montrer que **COLOR** est plus difficile de **SUGURU**.

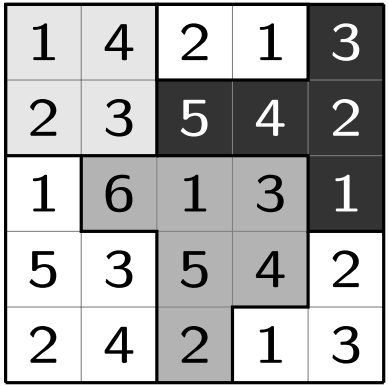


Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

Exercice : jeu du Suguru

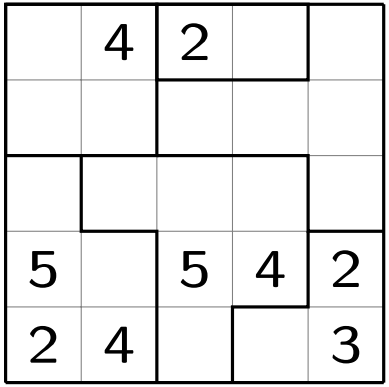
Montrer que **COLOR** est plus difficile de **SUGURU**.



Deux cases consécutives contiennent des entiers différents

Chaque zone contient des entiers consécutifs à partir de 1

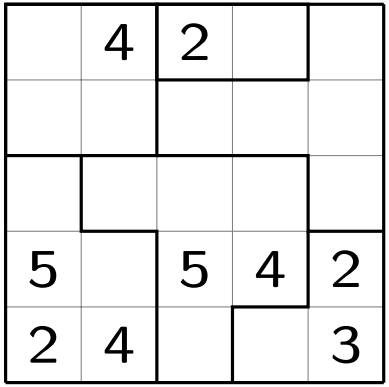
Graphe de contraintes



Deux cases differentes ont des couleurs différentes

Chaque zone contient les couleurs dans l'ordre
bleu (1) rouge (2) vert (3) orange (4) violet (5)
gris (6)

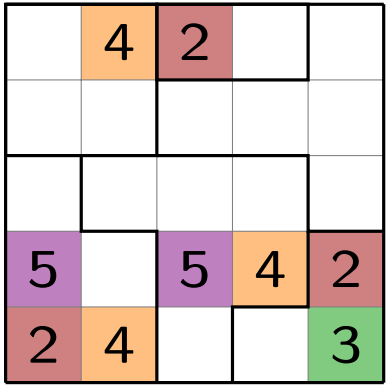
Graphe de contraintes



Deux cases differentes ont des couleurs différentes

Chaque zone contient les couleurs dans l'ordre
bleu (1) rouge (2) vert (3) orange (4) violet (5)
gris (6)

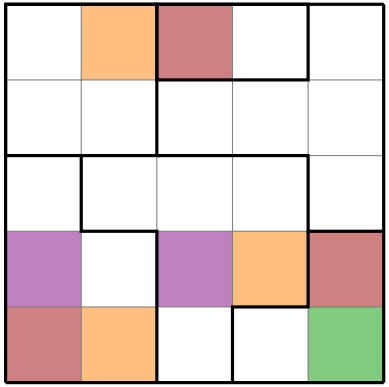
Graphe de contraintes



Deux cases differentes ont des couleurs différentes

Chaque zone contient les couleurs dans l'ordre
bleu (1) rouge (2) vert (3) orange (4) violet (5)
gris (6)

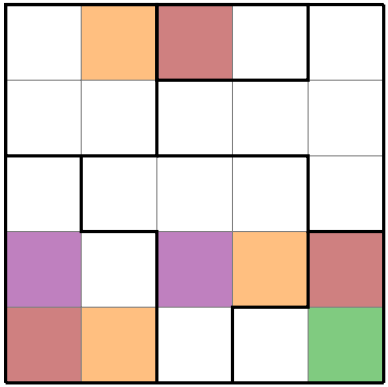
Graphe de contraintes

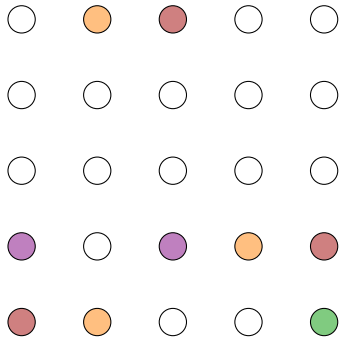


Deux cases differentes ont des couleurs différentes

Chaque zone contient les couleurs dans l'ordre
bleu (1) rouge (2) vert (3) orange (4) violet (5)
gris (6)

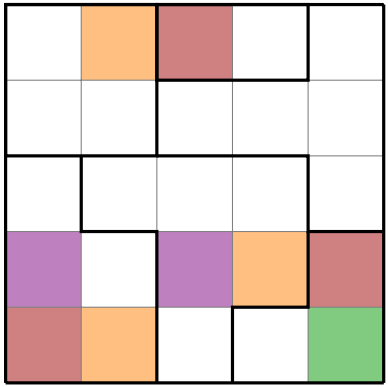
Graphe de contraintes

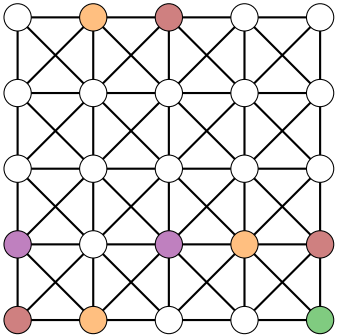




bleu (1) rouge (2) vert (3) orange (4) violet (5) gris (6)

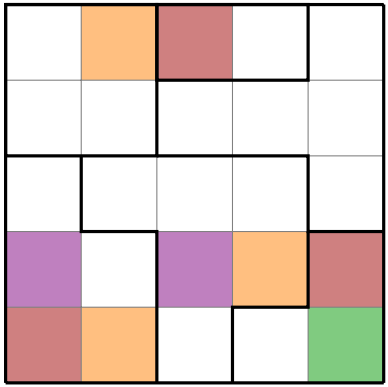
Graphe de contraintes

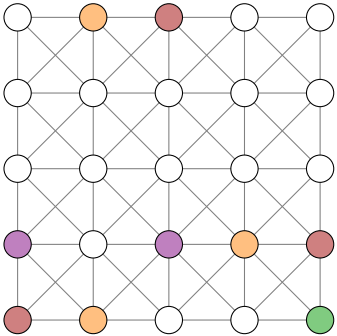




bleu (1) rouge (2) vert (3) orange (4) violet (5) gris (6)

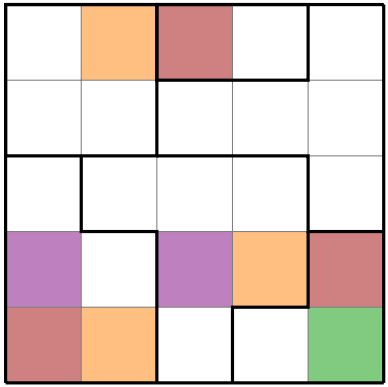
Graphe de contraintes

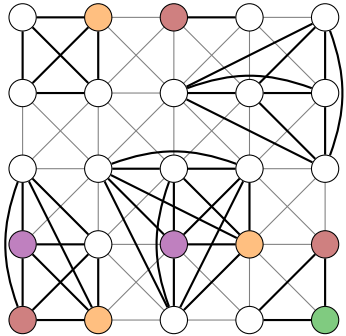




bleu (1) rouge (2) vert (3) orange (4) violet (5) gris (6)

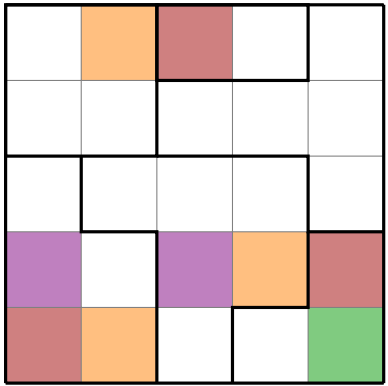
Graphe de contraintes

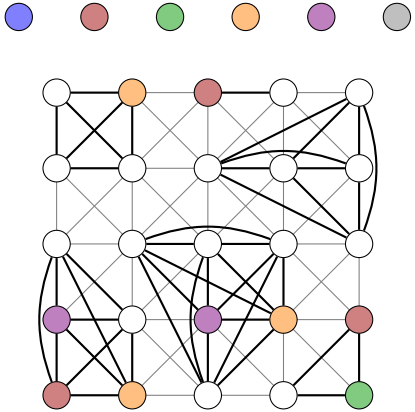




bleu (1) rouge (2) vert (3) orange (4) violet (5) gris (6)

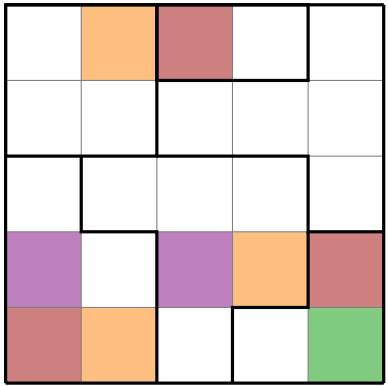
Graphe de contraintes

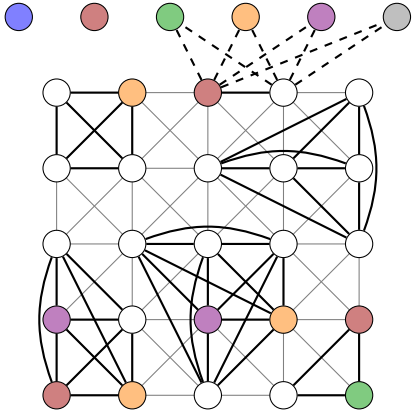




bleu (1) rouge (2) vert (3) orange (4) violet (5) gris (6)

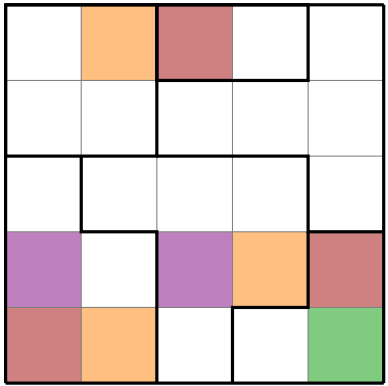
Graphe de contraintes

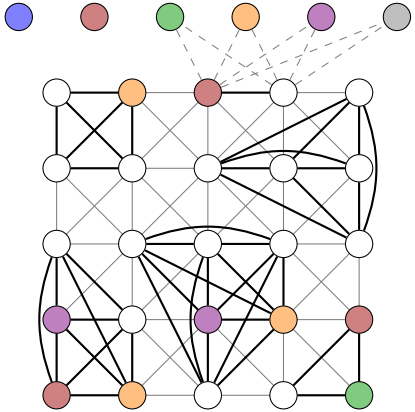




bleu (1) rouge (2) vert (3) orange (4) violet (5) gris (6)

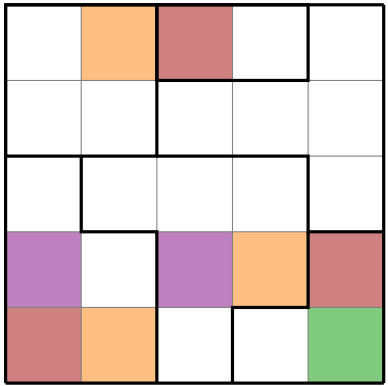
Graphe de contraintes

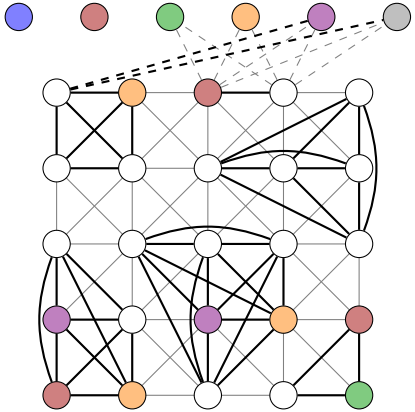




bleu (1) rouge (2) vert (3) orange (4) violet (5) gris (6)

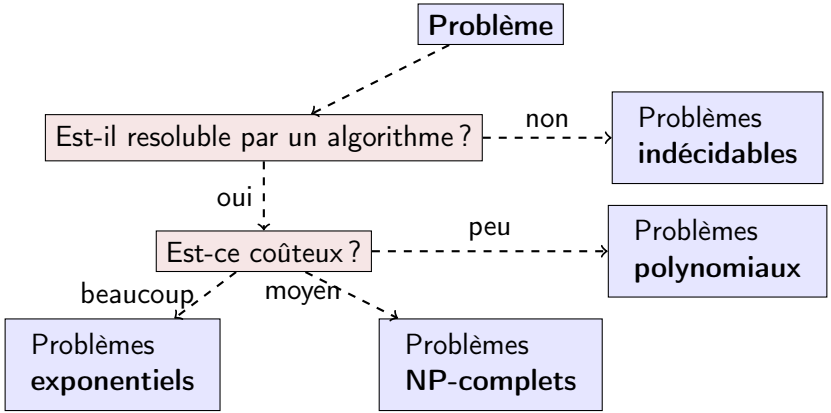
Graphe de contraintes





bleu (1) rouge (2) vert (3) orange (4) violet (5) gris (6)

Architecture du cours (schématique)



Métaheuristiques, approximations, etc.

Solveurs SAT

Solveurs IP (VS)