Chapitre IV

Complexité spatiale

Introduction

On se pose le même problème que pour la complexité temporelle, mais, cette fois-ci, avec la quantité de mémoire (= le nombre de cases utilisées sur la bande) nécessaire à la résolution d'un problème.

Ce type de complexité est appelée la complexité spatiale.

Ceci va nous amener à poser la définition de nouvelles classes de complexité.

Puis, à rechercher les propriétés de ces classes entre-elles, et avec les classes de complexité temporelle.

1 Complexité spatiale

Définition : complexité spatiale pour une MT déterministe.

Soit M une MT déterministe qui s'arrête sur toutes ses entrées.

La complexité spatiale de M est une fonction $f : \mathbb{N} \to \mathbb{N}$ où $f(n)$ est le nombre maximum de cases différentes sur lesquelles M passe sur n'importe quelle entrée de longueur n.

Définition : complexité spatiale pour une MT non déterministe.

Soit M une MT non-déterministe que s'arrête sur toutes ses entrées.

La complexité spatiale de M est la fonction $f(n)$ représentant la complexité spatiale maximale sur l'ensemble des branches de M sur n'importe quelle entrée de longueur n.

Remarques :

- Si la complexité spatiale de M est $f(n)$, on dit M s'exécute en espace $f(n)$.
- Défini ici en nombre de cases utilisées par la MT.
  Défini en octets sur un processeur.

2 SPACE et NSPACE

Soit $f : \mathbb{N} \to \mathbb{R}$ une fonction.

On définit alors : **Définition 40 : SPACE**$(f(n))$

SPACE$(f(n))$ est l'ensemble des langages décidés par une MTD M qui s'exécute en espace $f(n)$.

Définition 41 : NSPACE$(f(n))$

NSPACE$(f(n))$ est l'ensemble des langages décidés par une MTND M qui s'exécute en espace $f(n)$.

Note :

Dans un premier temps, on considérera que $f(n) \geq n$, car la bande d'entrée contient l'entrée qui elle-même est de taille n. Donc, si $f(n) < n$ cela signifierait que l'entrée n'a pas été lue en entier.

Théorème 62 :
$\mathtt{SAT} \in \mathtt{SPACE}(n)$

Démonstration:
Soit Φ une formule booléenne. La MTD M suivant décide $\mathtt{SAT}$.

$M(\langle\Phi\rangle) =$ pour chaque affectation des valeurs de vérité des variables
$x_1, \ldots, x_m$ de Φ
évaluer Φ pour ces valeurs de vérité
si Φ est évalué comme vrai alors accepter
rejeter.

La bande doit seulement stocker la valeur de vérité courante, ce qui s'effectue en $O(m)$ symboles. Passer à la valeur de vérité suivante revient à compter en binaire.

Le nombre de variables m est au plus de n (=longueur de l'entrée).
En conséquence, M s'exécute en espace linéaire. □

On rappelle que $\mathtt{ALL}_{\mathtt{ANF}} = \{\langle M\rangle \mid M$ est un $\mathtt{ANF}$ et $L(M) = \Sigma^*\}$.

En conséquence, $\overline{\mathtt{ALL}_{\mathtt{ANF}}}$ est l'ensemble des ANFs qui rejettent au moins un mot.

Note : nous ne savons pas si $\mathtt{ALL}_{\mathtt{ANF}}$ est dans **NP** ou dans **coNP**.

Théorème 63 :
$\mathtt{ALL}_{\mathtt{ANF}} \in \mathtt{coNSPACE}(n)$
à savoir, $\overline{\mathtt{ALL}_{\mathtt{ANF}}} \in \mathtt{NSPACE}(n)$

Démonstration:
Soit M un ANF reconnaissant le langage Σ. Notons $p = \#\Sigma$ et q le nombre d'état de M. Le chemins le plus long qu'un tel ANF peut avoir avant de repasser par le même état et la même transition est de longueur p^q.

Au-delà de cette longueur, on repasse nécessairement par un état et une transition déjà empruntée (autrement dit, on boucle).

Il suffit donc de tester tous les chemins de longueur p^q, et si aucun n'est n'est accepté, aucune mot ne le sera jamais.

Considérons la MT S suivante qui décide si l'ANF M :

$S(\langle M\rangle) =$ ① Q = état du départ de l'ANF M
② Répéter p^q fois
Choisir de façon non-déterministe le symbole s suivant.
Mettre à jour de $Q = \delta(Q, s)$ (simulation de M).
Si Q n'est pas un état acceptant, alors accepter.
③ Rejeter.

Cette machine génère tous les mots de longueur p^q et rejette si M rejette au moins un mot. En conséquence, cette machine décide $\overline{\mathtt{ALL}_{\mathtt{ANF}}}$.

L'exécution de M nécessite de stocker $\langle M\rangle$ (= n), Q (= $O(1)$) et le compteur de répétition (= $O(q) = O(n)$).

Donc $\overline{\mathtt{ALL}_{\mathtt{ANF}}}$ s'exécute dans un espace non-déterministe $O(n)$.

□

3 Théorème de Savitch

Théorème 64 : Savitch
soit une fonction $f : \mathbb{N} \to \mathbb{R}$ telle que $f(n) \geq n$, alors $\texttt{NSPACE}(f(n)) \subseteq \texttt{SPACE}(f(n)^2)$.

Autrement dit, toute MTND qui s'exécute en espace $f(n)$ peut s'exécuter en espace $\texttt{SPACE}(f(n)^2)$ sur une MTD.

Idée de la démonstration :
- Il faut réaliser sur une MTD la simulation d'une MTND en espace $f(n)$.
- Or si une MTND est en espace $f(n)$, elle est au pire en temps $f(n)$. On a vu qu'elle pouvait se simuler sur une MTD en temps $2^{f(n)}$, qui elle même est donc au pire en espace $2^{f(n)}$. Donc, simuler toutes les branches n'est pas la bonne façon d'aborder le problème.
- **problème d'atteignabilité :** soit deux configurations c_1 et c_2 deux configurations de la MTND, on définit un décideur $T(c_1, c_2, t)$ qui accepte si la configuration c_1 peut conduire à la configuration c_2 en t étapes.
- Si on trouve un algorithme récursif qui permette de réutiliser ou partager la mémoire entre l'exécution de $T(c_1, c_m, t/2)$ et $T(c_m, c_2, t/2)$, on pourra gagner une quantité considérable de mémoire.

Démonstration:
Soit une MTND N qui décide un langage A en espace $f(n)$. Construisons une MTD M qui décide A.

Soit w une chaine à l'entrée de N. Pour des configurations c_1 et c_2 de $N(\langle w \rangle)$, et un entier t puissance de 2, $T(\langle N, c_1, c_2, t\rangle)$ accepte si N peut aller de la configuration c_1 à la configuration c_2 en t étapes (ou moins) sur un chemin non déterministe.

On note $c_1 \xrightarrow{N} c_2$, si c_2 peut être obtenu à partir de c_1 en une étape.

$T(\langle N, c_1, c_2, t\rangle)$ **=** si $t = 1$ alors
 si $c_1 = c_2$ alors accepter.
 si $c_1 \xrightarrow{N} c_2$ alors accepter.
 rejeter.
si $t > 1$ alors pour chaque configuration c_m de $N(\langle w \rangle)$
 exécuter $T(\langle N, c_1, c_m, t/2\rangle)$
 exécuter $T(\langle N, c_m, c_2, t/2\rangle)$
 si les deux acceptent, alors accepter.
rejeter

Évidemment, T fonctionne bien (récursivement) comme attendu.

On construit la MTD M de façon à simuler la machine N de la façon suivante :
- quand N accepte, la bande est effacée, ramène la tête à gauche, et entre dans la configuration acceptante c_{accept}.
- $c_{\text{début}}$ est la configuration de départ de $N(\langle w \rangle)$.
- choisir d tel que N a au plus $2^{d.f(n)}$ configurations (**note :** nombre de configurations = borne supérieure du temps d'exécution de $N(\langle w \rangle)$).

On définit maintenant M comme :

$M(\langle N, w\rangle)$ **=** renvoyer le résultat de $T(\left\langle N, c_{\text{début}}, c_{\text{accept}}, 2^{d.f(n)} \right\rangle)$

Vérifions que M s'exécute en espace $O(f^2(n))$.

A chaque appel récursif de T :
- il stocke c_1, c_2, t et c_m sur une pile afin de pouvoir poursuivre l'exécution au retour de l'appel récursif.

— il utilise un espace $O(f(n))$ supplémentaire.
— t est divisé par deux. Partant de $2^{df(n)}$, la profondeur de récursion est donc de $O(\log 2^{df(n)}) = O(f(n))$.

En conséquence, l'espace total utilisé est $O(f^2(n))$. □

4 PSPACE

Définition 42 : PSPACE
PSPACE est la classe des langages qui sont décidables en espace polynomial par une MTD, à savoir : **PSPACE** $= \cup_k \mathtt{SPACE}(n^k)$

Définition 43 : NPSPACE
NPSPACE est la classe des langages qui sont décidables en espace polynomial par une MTND, à savoir : **NPSPACE** $= \cup_k \mathtt{NSPACE}(n^k)$

Théorème 65 :
PSPACE = **NPSPACE**

Démonstration:
Par le théorème de Savitch : comme $\mathtt{NSPACE}(f(n)) \subseteq \mathtt{SPACE}(f(n)^2)$, pour tout k, $\mathtt{NSPACE}(n^k) \subseteq \mathtt{SPACE}(n^{2k}) \subset$ **PSPACE**. Donc, **NPSPACE** ⊆ **PSPACE**.

Inversement, toute MTD est aussi trivialement une MTND (à une branche). Pour tout k, $\mathtt{SPACE}(n^k) \subseteq \mathtt{NSPACE}(n^k) \subset$ **NPSPACE**. En conséquence, **PSPACE** ⊆ **NPSPACE**

D'où l'égalité. □

4.1 PSPACE et coNPSPACE

Commençons par un premier résultat.

Proposition 66 :
$\forall f(n), \mathtt{SPACE}(f(n)) = \mathtt{coSPACE}(f(n))$

Démonstration:
— $\forall A \in \mathtt{SPACE}(f(n))$, soit M la machine décidant A. Alors la machine M' basée sur M en inversant *accepter* et *rejeter* permet de décider $\overline{A}$. Or, M et M' sont évidemment dans la même classe (changer la réponse renvoyée ne change pas la complexité). Donc $\mathtt{coSPACE}(f(n)) \subseteq \mathtt{SPACE}(f(n))$.
— le même raisonnement en inversant $\mathtt{SPACE}(f(n))$ et $\mathtt{coSPACE}(f(n))$ conduit à $\mathtt{SPACE}(f(n)) \subseteq \mathtt{coSPACE}(f(n))$.

□

Remarques :
— la même démonstration pourrait être faite pour démontrer que **P** = co**P**.
— le situation est très différente pour une MTND, car il suffit qu'une branche accepte afin que la machine accepte. En revanche, une MTND ne rejette que si toutes ses branches ont rejetées.

Théorème 67 :
PSPACE = co**NPSPACE**

Démonstration:
$\forall x \in$ **NPSPACE**, $x \in$ **PSPACE** (car **PSPACE** = **NPSPACE**).

Par définition, $\overline{x} \in$ co**PSPACE**.

Or, co**PSPACE** = **PSPACE** (car $\forall f(n), \mathtt{SPACE}(f(n)) = \mathtt{coSPACE}(f(n))$).

Donc, $\overline{x} \in$ **PSPACE**

Mais, d'après la définition de x, on a aussi $\overline{x} \in$ co**NPSPACE**.

En conséquence, **PSPACE** = co**NPSPACE**.

□

On a donc :

$$\textbf{PSPACE} = \texttt{co}\textbf{PSPACE} = \textbf{NPSPACE} = \texttt{co}\textbf{NPSPACE}$$

4.2 PSPACE et P, NP

Théorème 68 :
P ⊆ **PSPACE**

Démonstration:
Si un langage est décidé par une MTD M en temps $f(n)$, alors M visite au plus $f(n)$ cellules différentes. Donc, $\texttt{TIME}(f(n)) \subseteq \texttt{SPACE}(f(n))$, et par conséquent **P** ⊆ **PSPACE**. □

Théorème 69 :
NP ⊆ **PSPACE**

Démonstration:
De la même façon, $\texttt{NTIME}(f(n)) \subseteq \texttt{NSPACE}(f(n))$. Donc, **NP** ⊆ **NPSPACE**. Or, on a montré que **PSPACE** = **NPSPACE**. En conséquence, **NP** ⊆ **PSPACE**. □

4.3 PSPACE et EXPTIME

Lemme 70 :
une MTD M en espace $f(n)$ a au plus $f(n).2^{O(f(n))}$ configurations différentes.

Démonstration:
Notons $|Q|$ le nombre d'états de M et $|\Gamma|$ le nombre de symboles de son alphabet (de bande).

Si M est en espace $f(n)$,

— le nombre de bandes différentes est $|\Gamma|^{f(n)}$,

— le nombre de couples (position,état) différents sur la bande est $f(n).|Q|$

En conséquence, le nombre de configurations différentes est donc de :

$$f(n).|Q|.|\Gamma|^{f(n)} = f(n).2^{O(f(n))}$$

□

On rappelle que **EXPTIME** = $\bigcup_k \texttt{TIME}(2^{n^k})$.

Théorème 71 :
PSPACE ⊆ **EXPTIME**

Démonstration:
si un langage est décidé par une MTD M en espace $f(n)$ ($f(n) \geq n$), M peut visiter au plus $f(n)2^{O(f(n))}$ configurations (voir lemme ci-avant).

Donc, l'exécution dure au plus un temps $f(n)2^{O(f(n))}$, car si l'on repasse une seule fois par une même configuration, c'est que l'on est en train boucler (et donc que M n'est pas un décideur).

En conséquence, $\texttt{SPACE}(f(n)) \subseteq \texttt{TIME}(2^{O(f(n))})$.

On en déduit que **PSPACE** ⊆ **EXPTIME**. □

Donc, un programme qui utilise une taille de mémoire polynomiale s'exécute au pire en temps exponentiel.

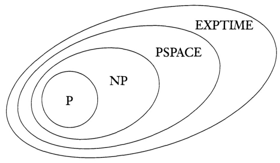
4.4 Résumé

En résumé,

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE} \subseteq \mathbf{EXPTIME}$$

Nous montrerons plus tard que $\mathbf{P} \neq \mathbf{EXPTIME}$.

Donc, au moins une des inclusions est stricte.



Remarques :
- actuellement, on ne sait pas laquelle est stricte.
- beaucoup pensent qu'elles le sont toutes.

5 PSPACE complétude

Définition 44 :
Un langage B est dit **PSPACE**-complet si il satisfait les deux conditions suivantes :
- $B \in \mathbf{PSPACE}$
- $\forall A \in \mathbf{PSPACE}$, $A \leq_{\mathbf{P}} B$, à savoir tout autre langage de **PSPACE** peut être réduit en temps polynomial à B.

Notes :
- on utilise une réduction en temps polynomial, car une réduction doit être facile par rapport au problème polynomial (rappel : $\mathbf{NP} \in \mathbf{PSPACE}$). Si la réduction était elle-même difficile, une solution facile à un problème complet n'entrainerait pas nécessairement une solution facile pour les problèmes qui se réduisent à lui.
- si B vérifie seulement la deuxième condition, alors B est dit **PSPACE**-difficile.

Définition 45 : QBF et TQBF
- **quantificateur** : $\forall$ (pour tout) et $\exists$ (il existe)
- **formule booléenne quantifiée :** (ou `QBF`) est une expression logique dont **tous** les littéraux sont quantifiés.
- **problème TQBF :** il s'agit de déterminer si une `QBF` est vraie, à savoir :
  $\mathtt{TQBF} = \{\langle F\rangle \mid F \text{ est une QBF vraie}\}$

Exemples :
- $\phi_1 = \forall x \exists y \left[(x \vee y) \wedge (\overline{x} \vee \overline{y})\right]$
  ϕ_1 est vrai.
  Pour toute valeur de x, je peux trouver une valeur de y qui rende ϕ_1 vrai : si $x = 0$ (resp. $x = 1$) prendre $y = 1$ (resp. $y = 0$).
- $\phi_2 = \exists y \forall x \left[(x \vee y) \wedge (\overline{x} \vee \overline{y})\right]$
  ϕ_2 est faux.
  Si $y = 0$, ϕ_2 n'est pas vrai pour toute valeur de x (prendre $x = 0$). Idem si $y = 1$ (avec $x = 1$).

Théorème 72 :
`TQBF` est **PSPACE**-complet.

Grandes lignes de la démonstration : en deux étapes,
Etape 1 : montrer que TQBF $\in$ **PSPACE**

Ceci s'effectue avec le décideur récursif suivant :

$U(\langle\phi\rangle) =$ si ϕ contient aucun quantificateur, l'expression contient uniquement des constantes, alors renvoyer le résultat de l'évaluation de ψ (= accepter si elle est vrai et rejeter sinon)
si $\phi = \exists x, \psi$, retourner l'évaluation de $U(\langle\psi_{x=0}\rangle) \vee U(\langle\psi_{x=1}\rangle)$
si $\phi = \forall x, \psi$, retourner l'évaluation de $U(\langle\psi_{x=0}\rangle) \wedge U(\langle\psi_{x=1}\rangle)$

La profondeur de récursion est au plus égal au nombre m de variables dans ϕ. A chaque niveau, on a besoin seulement besoin de stocker la valeur d'une variable.

Donc, l'espace total utilisé est $O(m) \leq O(n)$ et U s'exécute en espace **PSPACE**.

Etape 2 : Montrer que tout langage $A \in$ **PSPACE** se réduit à TQBF en temps polynomial.

Pour simuler une MT M en espace n^k, on utilise la réduction en temps linéaire suivante qui transforme l'exécution $M(< w >)$ en une QBF Φ qui est vraie ssi M accepte w.

— la QBF $\Phi_{c_1,c_2,t}$ est définie comme vraie si M peut aller de la configuration c_1 à la configuration c_2 en t étapes au plus.
— **cas** $t = 1$ **:** $\Phi_{c_1,c_2,1}$ code deux configurations consécutives de manière similaire à la preuve du théorème de Cook-Levin (cf la preuve pour les détails ; s'effectue en temps polynomial).
— **cas** $t > 1$ **:** comme pour le théorème de Savitch, on utilise une formulation récursive : $\Phi_{c_1,c_2,t} = \exists m \forall (u, v) \in \{(c_1, m), (m, c_2)\} [\Phi_{u,v,t/2}]$
— on définit alors $\Phi = \Phi_{c_{\text{start}},c_{\text{end}},h}$, où c_{start}, c_{end} sont respectivement les configurations de départ et acceptante de M, et $h = 2^{df(n)}$ est le nombre maximum de configurations possibles de M pour une entrée de longueur n.

Pour calculer la taille de la formule $\Phi_{c_{\text{start}},c_{\text{end}},2^{df(n)}}$, chaque appel récursif ajoute une portion de la formule en $O(f(n))$, le nombre de niveaux de récursion est $\log(2^{df(n)}) = O(f(n))$. Donc, la taille de la formule est $O(f^2(n))$ assure que la réduction est dans **PSPACE**. □

5.1 FORMULA-GAME

— Un jeu à formule est une paire (X, Ψ) où X est une liste de littéraux (= variables booléennes, *i.e.* $X = [x_1, x_2, \ldots, x_m]$) et Ψ est une formule booléenne sans quantificateurs.
— Le jeu consiste en m tours. Au $i^{\text{ème}}$ tour, un joueur affecte la valeur de vérité du littéral x_i.
— Deux joueurs A et B jouent à tour de rôle.
— Si Ψ est vrai après m tours, alors A gagne, sinon B gagne.
— On dit qu'un joueur a une stratégie gagnante si il peut toujours gagner le jeu, indépendamment de ce que l'autre joueur fait.

Exemple :

$\Psi = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})$
si $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, alors $\Psi = 0$ et B gagne.
si $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, alors $\Psi = 1$ et A gagne.

On définit maintenant le langage suivant :

FORMULA-GAME $= \{\langle\Psi\rangle \mid$ le joueur A a une stratégie gagnante pour le jeu à formule associé à $\Psi\}$

Exemple : dans l'exemple précédent

$\Psi = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})$
si A choisit $x_1 = 1$, alors
si B choisit $x_2 = 0$, alors A choisit $x_3 = 1$ et gagne ($\Psi = 1$)
si B choisit $x_2 = 1$, alors A choisit $x_3 = 0$ et gagne ($\Psi = 1$)
Donc, le choix $x_1 = 1$ et $x_3 = \overline{x_2}$ est une stratégie gagnante pour A.

Théorème 73 :
`FORMULA-GAME` est **PSPACE**-complet.

Démonstration:
On note Ψ la formule booléenne associé à un jeu.
Soit maintenant la QBF $\Phi = \exists x_1 \forall x_2 \exists x_3 \forall x_4 \dots [\Psi]$.
Alors si Φ est vrai, il existe une stratégie gagnante pour Ψ. En effet, l'alternance de $\exists$ et de $\forall$ signifie qu'il existe toujours un choix pour A qui permette quelque soit le choix de B, et ceci pour tous les tours de jeu. Donc, $\Psi \in$ `FORMULA-GAME` $\Rightarrow \Phi \in$ `TQBF`.

Inversement, pour un $\Phi \in$ `TQBF`, il suffit de considérer que le tour d'un joueur consiste à choisir la valeur de plusieurs littéraux (= tous ceux consécutifs associés à un même quantificateur). Ainsi, $\Phi \in$ `TQBF` $\Rightarrow \Psi \in$ `FORMULA-GAME`.

Les deux problèmes sont donc identiques, et la **PSPACE**-complétude de `FORMULA-GAME` est celle de `TQBF`. □

6 Classes L et NL

Pour l'instant nous avons toujours considéré que $f(n) \geq n$

à savoir, la bande devant au moins contenir l'entrée, et l'entrée étant de longueur n.

En revanche, il pourrait être intéressant de savoir ce qu'un décideur utilise comme mémoire **en plus** de n.

- donner une borne $O(n)$ sur la quantité de mémoire utilisée n'est pas un indice sur la performance en espace du décideur (car $O(1000.n) = O(n)$).
- le sub-linéaire devrait être possible.

Remarque : cette approche n'a pas de sens dans le cadre de la complexité temporelle (temps de lecture de la bande + accès séquentiel aux données).
Conséquence : ceci nous amène à modifier légèrement le modèle de MT considéré jusqu'à présent afin de quantifier les décideurs à espace sub-linéaire.

On considère la MT suivante à 2 bandes
- la première bande est la bande d'entrée (contient l'entrée de la MT et est en lecture seule)
- la seconde bande est la bande de travail (en lecture/écriture)

De plus, pour ce type de machine :
- sur la bande d'entrée, le pointeur reste toujours sur la partie de la bande contenant l'entrée.
- sur cette machine, une **configuration** contient l'état de la MT, la bande de travail, les positions des pointeurs sur chaque bande.

Par la suite, langage et espace ayant des bornes en espace sub-linéaire utilisera par défaut ce type de machine.

Définition 46 : classe L
L est la classe des langages qui sont décidables en espace logarithmique sur une MTD.
L = `SPACE`$(\log n)$

Définition 47 : classe NL
NL est la classe des langages qui sont décidables en espace logarithmique sur une MTND.
NL = $\mathtt{NSPACE}(\log n)$

Ces espaces nous intéressent à plusieurs titres :
- ils sont assez gros pour permettre la résolution de problèmes intéressants.
- ils sont robustes (indépendant du modèle de machines ou de la méthode de codage de l'entrée, pour peu qu'elle reste raisonnable).

6.1 Langage 0^k1^k

Soit le langage $A = \left\{0^k1^k \mid k > 0\right\}$
Théorème 74 :
$A \in \mathbf{L}$

Remarque préliminaire :

L'algorithme que l'on avait vu pour décider A sur une MT était en espace linéaire (il utilisait la bande d'entrée pour rayer un symbole sur deux). Ceci n'est plus possible puisque l'on ne peut plus écrire sur la bande d'entrée. Il faut donc trouver une autre méthode.

Démonstration:
On utilise une MT qui décide A de la façon suivante
- vérifier la syntaxe (une suite de 0 suivie d'une suite de 1).
- dans un compteur sur la bande de travail, compter en binaire le nombre de 0.
- dans un deuxième compteur sur la bande de travail, compter en binaire le nombre de 1.
- si les deux compteurs sont égaux, accepter, sinon refuser.

Or la taille de l'entrée étant n, cela signifie que chaque compteur occupent au plus $\log_2 n$ bits sur la bande de travail, soit au total $2\log_2 n = O(\log n)$.
D'où $A \in \mathtt{SPACE}(\log n) = \mathtt{NSPACE}(\log n) = \mathbf{L}$. □

6.2 PATH

On rappelle la définition de PATH :
PATH = $\{\langle G, s, t\rangle \mid$ le graphe orienté G a un chemin orienté de s à $t\}$
Rappel : on a vu que PATH $\in \mathbf{P}$ mais l'algorithme utilisait un espace linéaire.
Théorème 75 :
PATH $\in \mathbf{NL}$

Démonstration:
Une MTND décidant $\langle G, s, t\rangle \in$ PATH en temps **NL** s'écrit de la manière suivante. Soit p le nombre de nœuds dans le graphe.

1. enregistrer le nœud de départ $x = s$, et la longueur du chemin $i = 0$
2. pour le nœud courant x, choisir de façon non déterministe un nœud y tel qu'il existe une arête orientée de x vers y (rejeter s'il n'y en a pas).
3. mise à jour de la bande de travail ($x = y$ et $i = i + 1$).
4. si ($y = t$) alors accepter.
5. si ($i \geq p$) alors rejeter.
6. recommencer en 2.

La bande de travail contient uniquement deux compteurs : le numéro du nœud courant et la taille actuelle du chemin. Ils sont stockés en binaire et utilisent donc au plus une taille $2\log_2 p = O(\log p) = O(\log n)$.
D'où PATH $\in \mathtt{NSPACE}(\log n) = \mathbf{NL}$. □

7 NL-complétude

On a vu que PATH $\in$ **NL**, mais appartient-il à **L** ?

De façon similaire à la question de savoir si **P** = **NP**, il se pose la question de savoir si **L** = **NL**

Nous avons vu qu'une façon de classifier la difficulté d'un problème était d'exhiber des problèmes **NP**-complet. De la même façon, si **L** et **NL** sont différents, tous les langages **NL**-complet n'appartiennent pas à **L**.

Comme nous le verrons, les problèmes de **NL** peuvent être résolu en temps polynomial. En conséquence, une réduction en temps polynomial ne permet pas de différentier la difficulté des problèmes. Nous introduisons donc un nouveau type de réduction : la réduction en espace logarithmique.
Nous définissons maintenant une MT qui découple complètement l'espace d'entrée, de travail et de sortie :

Définition 48 : transducteur
Un transducteur est une MT avec une bande d'entrée en lecture seule, une bande de travail et une bande de sortie en écriture seule.

Définition 49 : fonction calculable en espace logarithmique
Une fonction calculable en espace logarithmique $f(w)$ est :
- un transducteur qui prend en entrée $\langle w\rangle$, avec $n = |\langle w\rangle|$.
- dont la bande de travail contient au plus $O(\log n)$ symboles,
- et qui renvoie la valeur stockée sur sa bande de sortie au moment où le transducteur s'arrête.

Définition 50 : réduction en espace logarithmique
Un langage A est dit réductible en espace logarithmique à un langage B si il existe une réduction de A à B par une fonction calculable en espace logarithmique.
On note : $A \leq_{\mathbf{L}} B$
Rappel : réduction = fonction calculable tq $\forall w, w \in A \Leftrightarrow f(w) \in B$.

Définition 51 : NL-complétude
Un langage B est dit **NL**-complet si :
- $B \in \mathbf{NL}$
- $\forall A \in \mathbf{NL}$, $A \leq_{\mathbf{L}} B$, à savoir tout autre langage de **NL** peut se réduire à B.

Théorème 76 :
Si $A \leq_{\mathbf{L}} B$ et $B \in \mathbf{L}$ alors $A \in \mathbf{L}$.

Démonstration:
Même idée de preuve que pour la version décidable de ce théorème : pour décider $w \in A$, décider $f(w) \in B$. A savoir, calculer $f(w)$, puis utiliser M_B pour décider $f(w)$.

Le problème ici est de simuler à la fois M_B et f en SPACE($\log n$). Comme leurs bandes de travail sont en SPACE($\log n$), il est possible de stocker leurs configurations associées en même temps sur la bande de travail de M_A.

Néanmoins, les entrées/sorties de f et l'entrée de M_B n'ont aucune raison d'être en SPACE($\log n$). Il est donc nécessaire d'effectuer l'exécution symbole par symbole.

La machine M_A fonctionne de la manière suivante :

1. la bande d'entrée contient w, son pointeur de lecture est utilisé pour la lecture de l'entrée de f.
2. initialiser les machines f et M_B (configuration de départ).
3. suite de la simulation de f : lecture symbole par symbole de w sur la bande d'entrée jusqu'à ce que f produise un symbole s.

4. suite de la simulation de M_B avec le symbole s.
5. si M_B accepte (resp. rejette) alors accepter (resp. rejetter)
6. tant que la lecture de w n'est pas finie, recommencer en 3.

Les simulations pas à pas sont possibles grâce au stockage de la configuration (permet de reprendre la simulation dans l'état dans lequel on l'a laissée).

En terme de stockage, la bande de travail de M_A contient :
— la configuration de la bande de travail de f (espace $O(\log n)$).
— la configuration de la bande de travail de M_B (espace $O(\log n)$).
— le symbole s produit par f (espace $O(1)$).
Donc, la machine M_A décide bien A en $\mathtt{SPACE}(\log n)$ et $A \in \mathbf{L}$ □

Corollaire 77:
si n'importe quel langage **NL**-complet est dans **L**, alors $\mathbf{L} = \mathbf{NL}$

Démonstration:
si B est **NL**-complet alors $\forall A \in \mathbf{NL}$, $A \leq_{\mathbf{L}} B$.
L'existence de cette réduction implique que $\forall w, w \in A \Leftrightarrow f(w) \in B$.
Si $B \in \mathbf{L}$, le décideur de B permet donc de décider $f(A)$en espace L.
Donc, $A \in \mathbf{L}$ (et pour tout $A \in \mathbf{NL}$). D'où $\mathbf{L} = \mathbf{NL}$. □

Proposition 78 :
PATH est **NL**-complet.

Démonstration:
Nous avons déjà démontré que $\mathtt{PATH} \in \mathbf{NL}$.
Il reste à démontrer que **PATH** est NL-difficile (à savoir $\forall A \in \mathbf{NL}$, $A \leq_{\mathbf{NL}} B$).
Il s'agit donc de trouver une réduction f de tout $A \in \mathbf{NL}$ dans PATH.

Soit M une MTND qui décide A en espace $O(\log n)$. La réduction f construit un graphe $\langle G, s, t\rangle$ à partir de $M(\langle w\rangle)$ tel que :
— nœuds $\{c_i\}$ = les configurations de M
— arêtes (c_i, c_j) = si M permet d'aller des configurations c_i à c_j.
— le nœud s est la configuration de départ.
— il y a une configuration acceptante unique associée au nœud t.
f réduit A à PATH : si M accepte w, la branche acceptante correspond à un chemin dans G de s vers t.

Considérons le transducteur f en espace logarithmique suivant. On note p = longueur d'encodage d'une configuration, T = symboles codant une configuration (alphabet de bande, états de M), t = code (spécifique) de la configuration acceptante.
— **écriture de la liste des sommets :**
pour tout $u \in T^p$, si u est une configuration, l'écrire.
— **écriture de la liste des arêtes :**
pour chaque configuration $(u, v) \in T^p \times T^p$
si u et v sont des configurations et $u \Rightarrow v$, alors écrire (u, v).
si u est une configuration acceptante, écrire (u, t)
— **écriture de la configuration initiale :** (=bande vide + état de départ).
— **écriture de la configuration acceptante :** écrire t
Trivialement, cet algorithme fonctionne en espace $O(\log n)$ (on stocke au plus deux configurations en même temps sur la bande de travail, chacune étant en espace $O(\log n)$). □

Cette propriété est aussi valide pour le complémentaire de PATH :

Proposition 79 :
$\overline{\texttt{PATH}}$ est **NL**-complet.

Note : $\overline{\texttt{PATH}}$ est l'ensemble des graphes (G, a, b) pour lesquels il n'existe pas de chemin de a à b.

Démonstration:
Il faut donc trouver un MTND capable de déterminer s'il n'existe aucun chemin de a à b dans le graphe G, en espace logarithmique.

Nous sommes confrontés à deux problèmes :

— **Problème avec l'espace logarithmique :** on ne peut pas stocker le chemin (en $O(n)$). Il faut donc un moyen d'énumérer les chemins sans jamais en stocker aucun.

— **Problème d'acceptation :** s'il est facile de rejeter (dès que l'on rencontre un chemin qui relie a à b), une MTND n'accepte que si au moins une de ses branches accepte, et l'on ne peut accepter que s'il n'y a aucun chemin entre a et b.

Soit (G, a, b), une instance de PATH. Soit m le nombre de nœuds de G. Supposons que nous disposions de k = nombre de nœuds de G qui sont atteignables depuis a. Alors, on peut écrire la MNTD suivante :

Reach*$(\langle G, a, b, k\rangle)$ = $s = 0$
$\forall v \in G \setminus \{b\}$
  choix ND d'un chemin de longueur $\leq n$ dans G
  si on atteint v, alors $s = s + 1$
si $s = k$ alors accepter sinon rejeter.

La branche qui accepte a donc pu trouver k chemins qui vont de a à un nœud autre que b, et donc il n'existe aucun chemin entre a et b.

Comment calculer k ?
Notons k_p le nombre de nœuds de G atteignables depuis a en au plus p pas. On propose la MTND suivante qui permet de calculer k_{p+1} à partir de k_p. Évidemment, on sait que $k_0 = 1$. L'idée consiste à générer (de manière non-déterministe) toutes les k_p combinaisons de chemins de longueur p, et de tester tous les prolongements possibles de ces k_p chemins.

Ceci est réalisé par : NextK$(\left\langle G, a, b, p, k_p \right\rangle)$ =

1. $c_u = 0$
2. pour tout $u \in G$
3. $c_v = 0$
4.   pour tout $v \in G$
5.   choix ND d'un chemin de longueur $\leq p$ partant de a
6.   si le chemin se termine par v alors $c_v = c_v + 1$
7.   si $(v, u) \in E$ alors $c_u = c_u + 1$ et passer au u suivant (en 2).
8.   si $c_v \neq k_p$ alors rejeter
9. retourner c_u (= valeur de k_{p+1})

On remarquera que :

— à l'étape 6, on a trouvé un chemin de longueur $\leq p$ qui va jusqu'à v (donc, on incrémente c_v).

— à l'étape 7, c_u n'est incrémenté que si l'on peut poursuive ce chemin jusqu'à u. Donc, dans ce cas, u est atteignable par un chemin de longueur $\leq p + 1$. On peut donc passer au u suivant.

— l'étape 8 n'est là que si l'on a pas pu trouver de chemin vers u depuis a, dans ce cas, il n'est pas possible de poursuivre à partir des sommets atteignables (comptés dans k_p). Si $c_v \neq k_p$, cela signifie que les choix ND n'ont pas permis d'atteindre tous les sommets atteignables.

A la fin de l'exécution, k_{p+1} contient bien le nombre de sommets atteignables depuis a au en plus $p + 1$ pas.

Au final, la MTND M acceptant $\overline{\texttt{PATH}}$ est donc la suivante :

$M(\langle G, a, b\rangle) =$ n =nombre de nœuds de G
// calcul de k
$k = 1$
pour $i = 0$ à $n - 1$
  $k = \texttt{NextK}(\langle G, a, b, i, k\rangle)$
// vérification que b n'est pas atteignable
$\texttt{Reach}^*(\langle G, a, b, k\rangle)$

En terme de stockage, on a besoin par branche de stocker n, p, k, c_u, c_v, i et quelques pointeurs (vers a, u et v). En conséquence, M s'exécute en $\texttt{NSPACE}(\log n)$.

$\forall \overline{A} \in \texttt{co}\mathbf{NL}$, $A \in \mathbf{NL}$ existe et est défini. Soit N une MTND en espace $O(\log n)$ qui accepte A. Alors, $\forall x \in \overline{A}$, on peut construire son graphe de configuration $G_{N,x}$ et décider $N(\langle x\rangle)$ en passant $(G_{N,x}, \texttt{début}, \texttt{accept})$ à $\overline{\texttt{PATH}}$. Donc, $\overline{\texttt{PATH}}$ est **NL**-complet.

Autrement dit, le transcodeur f utilisé pour prouver que $\texttt{PATH}$ était **NL**-complet peut-être réutilisé pour montrer que $\overline{\texttt{PATH}}$ est **NL**-complet. □

Théorème 80 :
NL = co**NL**

Démonstration:

1. $\forall x \in \mathbf{NL}$, nous avons $x \leq_{\mathbf{L}} \texttt{PATH}$ puisque $\texttt{PATH}$ est **NL**-complet.
   Alors, par la même fonction de réduction, nous avons $\overline{x} \leq_{\mathbf{L}} \overline{\texttt{PATH}}$.
   Donc, $\overline{x} \in \mathbf{NL}$ puisque $\overline{\texttt{PATH}} \in \mathbf{NL}$.
   En conséquence, $\mathbf{NL} \subseteq \texttt{co}\mathbf{NL}$.
2. $\forall x \in \texttt{co}\mathbf{NL}$, on a $\overline{x} \in \mathbf{NL}$.
   De façon similaire, $\overline{x} \leq_{\mathbf{L}} \texttt{PATH}$ et $x \leq_{\mathbf{L}} \overline{\texttt{PATH}}$.
   Encore une fois, puisque $\overline{\texttt{PATH}} \in \mathbf{NL}$, on a aussi $x \in \mathbf{NL}$.
   Donc, $\texttt{co}\mathbf{NL} \subseteq \mathbf{NL}$.

D'où le résultat. □

Corollaire 81:
$\mathbf{NL} \subseteq \mathbf{P}$

Démonstration:

Montrons qu'un langage décidable en espace logarithmique est nécessairement en temps polynomial :

Démo 1 : On a vu qu'un MTD M en espace $f(n)$ a au plus $c(n) = f(n).2^{O(f(n))}$ configurations différentes.

Si $M \in \mathbf{NL}$, alors $f(n) = O(\log n)$, $\exists k_1, k_2$ tel que, pour n assez grand : $c(n) \leq k_1 \log(n).e^{k_2 \log(n)} = k_1 \log(n).n^{k_2} \leq n^{k_2+1}$

Donc, le nombre de configurations différentes $c(n)$ est de l'ordre de $O(n^k)$.

Comme M est décidable (donc on ne passe au plus qu'une fois dans chaque configuration), on a nécessairement $M \in TIME(n^k) \subset \mathbf{P}$.

Démo 2 : On a vu que :
- $\texttt{PATH}$ est **NL**-complet.
- $\texttt{PATH} \in \mathbf{P}$ (voir leçon précédente).

En conséquence, tout langage $L \in \mathbf{NL}$ peut être réduit en espace logarithmique à $\texttt{PATH}$, mais comme la réduction en espace logarithmique et $\texttt{PATH}$ s'effectuent en temps polynomial, tout $L \in \mathbf{NL}$ appartient aussi à **P**.

On en déduit que $\mathbf{NL} \subseteq \mathbf{P}$. □

8 Conclusion

Conclusion de ce chapitre :

$$\mathbf{L} \subseteq \mathbf{NL} = \mathsf{co}\mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$$

On peut montrer aussi que $\mathbf{NL} \subsetneq \mathbf{PSPACE}$.

En conséquence,
— soit $\mathsf{co}\mathbf{NL} \subsetneq \mathbf{P}$.
— soit $\mathbf{P} \subsetneq \mathbf{PSPACE}$.

Malheureusement, on ne sait pas laquelle des deux est vraie ou si les deux sont vraies (on pense qu'il s'agit de ce dernier cas).