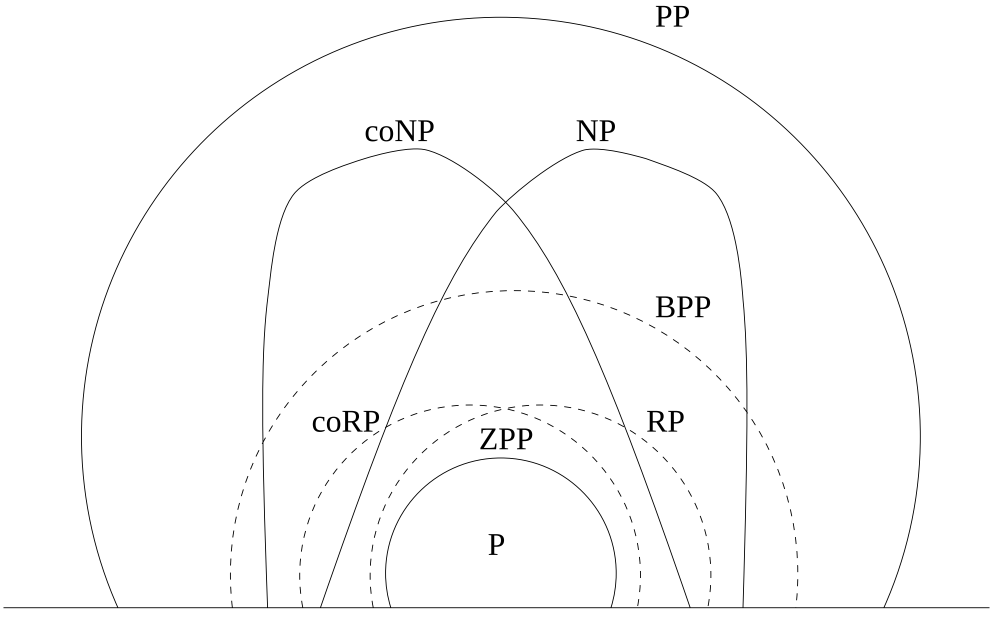
Complexité
Algorithmes probabilistes.

Dominique ROSSIN (d'après le cours de Miklos SANTHA)



September 23, 1998

Contents

1 Introduction aux machines de Turing **3**

1.1 Introduction . . . 3

1.2 Les machines de Turing comme algorithmes . . . 3

1.3 Ressources . . . 4

1.4 Restriction sur les ressources . . . 4

1.5 Accélération . . . 4

1.6 Comparaison de fonctions . . . 5

2 Les classes de complexité standards **6**

2.1 Introduction et théorèmes généraux. . . . 6

2.2 Réductibilité . . . 11

2.3 Etude de la classe $\mathcal{P}$ . . . 12

2.4 Classe $\mathcal{NP}$ . . . 15

2.5 Digression sur la hiérarchie polynomiale . . . 19

2.6 Classe $\mathcal{NL}$ . . . 20

3 Complexité parallèle **23**

3.1 Borne de Chernoff . . . 34

3.2 Test de primalité : SOLOVAY-STRASSEN (77) . . . 35

3.3 Algorithme RNC pour COUPLAGE MAXIMAL . . . 40

3.3.1 Lemme de Schwartz ou Zippel (77) . . . 40

3.3.2 Corollaire . . . 40

3.4 Algorithme RNC pour décider si un graphe contient un couplage parfait . . . 41

3.4.1 Introduction . . . 41

3.4.2 Algorithme RNC pour couplage parfait . . . 42

4 Systèmes de preuves interactifs **47**

4.1 Introduction et définitions . . . 47

4.2 Exemple . . . 49

1 Introduction aux machines de Turing

1.1 Introduction

Définition 1.1 (Machine de Turing)
Une machine de Turing est un quadruplet $M = (K, \Sigma, \delta, s)$. K est un nombre fini d'états,$s \in K$ l'état initial, Σ un nombre fini de symboles (la sortie). Nous supposerons que K et Σ sont disjoints pour la suite. Σ contient toujours les symboles $\sqcup$ et $\triangleright$: le symbole blanc et le premier symbole.δ est une fonction de transition : $\delta : K \times \Sigma \to (K \cup \{h, yes, no\}) \times \Sigma \times \{\leftarrow, \to, -\}$. h est l'état stop, yes l'état acceptant et no l'état refusant, les flèches la direction de la tête de sortie et $-$ la stagnation de la sortie. Ces symboles ne sont pass dans $K \cup \Sigma$. δ est le programme de la machine : $\delta(q, \sigma) = (p, \rho, D)$ $q \in K$ l'état courant, $\sigma \in \Sigma$ le symbole courant et p l'état suivant, ρ le symbole remplacant σ et D la direction de déplacement de la tête.
Une machine à plusieurs bandes travaille dans Σ^k

1.2 Les machines de Turing comme algorithmes

Définition 1.2 (Langages reconnus)
Si pour un langage L il existe une machine de Turing qui répond "yes" pour tout élément du langage et "no" à tout autre élément alors le langage est dit reconnu.

Définition 1.3 (Langages accepté par une machine de Turing)
Un langage L est dit accepté par une machine de Turing si il existe une machine de Turing qui répond "oui" à tout élément du langage et qui ne s'arrête pas pour un autre élément.

Définition 1.4 (Langage récursivement énumérable)
Un langage est dir récursivement énumérable si il existe une machine de Turing qui accepte L

Théorème 1.1 (Reconnu $\Leftrightarrow$ récusivement énuérable)
Si L est reconnu alors L est récursivement énumérable.

Démonstration 1.1
Soit une machine M qui reconnait L. On construit une machine M' comme suit : M' fait exactement comme L sauf que au lieu de répondre no elle va

à droite indéfiniment.

1.3 Ressources

- Temps
- Espace : On exclue la bande d'entrée.

1.4 Restriction sur les ressources

Définition 1.5 (Complexité propre)
Une fonction $f : \mathbb{N} \to \mathbb{N}$ est une fonction de complexité propre si :

- $\forall n, f(n+1) \geq f(n)$
- $\exists MTD$ *qui calcule* $f(n)$ *en temps* $\theta(f(n)+n)$ *ie qui calcule* $0^{f(n)}$*sur une entrée quelconque de longueur* n *en temps* $\theta(f(n)+n)$

Théorème 1.2 ()
Les fonctions standards sont des fonctions de complexité propre comme $P(n), 2^n, \log n, \frac{n}{\log n}$

Dans tout le cours les fonctions considérées qui restreignent les ressources sont des fonctions de complexité propre.

Définition 1.6 (DTIME, NTIME, DSPACE, DTIME)
$DTIME(f) = \{L \subseteq \{0,1\}^*, \exists MTD$ *qui reconnait* L *en temps* $f\}$
$NTIME(f) = \{L \subseteq \{0,1\}^*, \exists MTND$ *qui reconnait* L *en temps* $f\}$
$DSPACE(f) = \{L \subseteq \{0,1\}^*, \exists MTD$ *qui reconnait* L *dans l'espace* $f\}$
$NSPACE(f) = \{L \subseteq \{0,1\}^*, \exists MTND$ *qui reconnait* L *dans l'espace* $f\}$

1.5 Accélération

Théorème 1.3 (Accélération (cf page 32)
Accélération : on peut accélérer en temps et en espace par n'importe quelle constante.
Soit $L \in \mathbf{TIME}(f(n))$. *Alors* $\forall \epsilon > 0, L \in \mathbf{TIME}(f'(n))$ *ou* $f'(n) = \epsilon f(n) + n + 2$.

Démonstration 1.2
Soit $M = (K, \Sigma, \delta, s)$ une machine de Turing à k bandes qui reconnait L

et qui opère en temps $f(n)$. On doit construire une machine de Turing $M' = (K', \Sigma', \delta', s')$ qui reconnait L en temps $f'(n)$ et qui simule M. Le nombre k' de bandes de M' sera égal au nombre k de bandes de M si $k > 1$ et vaudra 2 si $k = 1$. L'idée est que la machine M' code plusieurs symboles de M à la fois.
Soit $m \in \mathbb{N}$, dépendant de M et de ϵ. L'alphabet de M' contiendra tous les symboles de M plus tous les m-uplets de symboles de M. $\Sigma' = \Sigma \cup \Sigma^m$. Au début M' bouge à droite lisant la première chaine de en entrée, c'est à dire l'entrée x. Quand m symboles ont été lus soit $\sigma_1\sigma_2\sigma_3 \cdots \sigma_m$ alors l'unique symbole $(\sigma_1, \sigma_2, \sigma_3, \cdots, \sigma_m) \in \Sigma'$ est écrit dans la seconde chaine. Il est facile de se souvenir de m symboles. En effet, il suffit d'ajouter à K' tous les états de $K \times \Sigma^i, i = 1, \cdots, m-1$. Si la fin de l'entrée est rencontrée avant la fin d'un m-uplet alors on rajoute le nombre de blanc nécessaires à la fin du n-uplet pour le remplir. Donc après $m\lceil \frac{|x|}{m} \rceil + 2$ pas la seconde chaine contient une version de l'entrée mais en m-uplets.
On considère maintenant la seconde bande comme entrée. Si $k > 1$ on considère la première bande comme une bande de travail normale. Toutes les k bandes contienent donc exclusivement des m-uplets de symboles.
Maintenant M' va simuler M en au plus 6 pas de calcul par m-uplet. Dans la suite nous appelleront étape de simulation la simulation de m étapes de calcul de M par M'. Au début de chaque étape, l'état de M' contient $(k+1)$ uplets $(q, j_1, j_2, \cdots, j_k)$. q est l'état de M au début de l'étape de simulation et $j_i < m$ est la position de la i^{eme} tete de lecture dans le m-uplet scanné.
Alors M' bouge sa tete d'un cran à gauche puis de deux crans à droite et enfin d'un cran à gauche. L'état de M' se souvient donc de tous les symboles près des tetes de lecture. Il suffit d'ajouter tous les uplets de symboles nécessaires à cette opération.
M' peut donc prévoir complètement les m prochains mouvements de M. En effet en m mouvements la tete de lecture de M ne peut pas changer de plus d'un n-uplet sur la gauche ou la droite. M' peut donc maintenant utiliser sa fonction δ' pour simuler en au plus deux pas les changements dans les chaines de travail. Il ne faut donc que 6 pas au plus à la machine M'.
Le temps pris par M' est $|x| + 2 + 6\lceil \frac{f(|x|)}{m} \rceil$ au pire. Donc en prenant $m = \lceil \frac{6}{\epsilon} \rceil$ ca marche.

1.6 Comparaison de fonctions

Définition 1.7 ($\mathcal{O}, \Omega, \Theta$)
$\mathcal{O}(f) = \{g, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, g(n) \leq c * f(n)\}$
$\Omega(f) = \{g, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, g(n) \geq c * f(n)\}$
$\Theta(f) = \mathcal{O}(f) \cap \Omega(f)$

2 Les classes de complexité standards

2.1 Introduction et théorèmes généraux.

Définition 2.1 ($\mathcal{L}$,$\mathcal{NL}$,$\mathcal{P}$,$\mathcal{NP}$,$\mathcal{PSPACE}$)
Les classes de complexité standars sont :

- $\mathcal{L} = DSPACE(\log n)$
- $\mathcal{NL} = NSPACE(\log n)$
- $\mathcal{P} = \bigcup_c DTIME(n^c)$
- $\mathcal{NP} = \bigcup_c NTIME(n^c)$
- $\mathcal{PSPACE} = \bigcup_c DSPACE(n^c)$

Théorème 2.1 ()
- $DTIME(f) \subseteq NTIME(f)$
- $DSPACE(f) \subseteq NSPACE(f)$

Démonstration 2.1
Une MTD est une MTND

Théorème 2.2 (Relation temps non déterministe temps déterministe)
$NTIME(f(n)) \subseteq \bigcup_{c>1} DTIME(c^{f(n)})$

Démonstration 2.2
Soit $N = (K, \Sigma, \Delta, s)$. Pour chaque $(q, \sigma) \in K \times \Sigma$ considérons l'ensemble des choix que peut faire la machine. Notons d le nombre maximum global de choix que peut faire la machine. d s'appelle le degré de non-déterminisation. L'idée est qu'une sq́uence de t choix non-déterministes estune séquence de t entiers dans l'intervalle $0 \cdots d-1$. La machine déterministe considère tous ces choix et simule N sur chacun.

Théorème 2.3 ()
$NTIME(f) \subseteq DSPACE(f)$

Démonstration 2.3
On examine tous les chemins de la machine non déterministe chacun à leur tour. Soit L un langage et soit M une MTND en temps $f(n)$ telle que $L(M) = L$. On définit une MTD M' qui en espace f accepte L.
M' sur x (la longueur de x est n)

- Sur une bande de travail M' engendre un par un les mots binaires de longueur $f(n)$ (*guesses*)
- Pour chaque mot, elle simule sur une autre bande le calcul (déterministe) de M avec les choix de ce mot.
- M' accepte ssi pour au moins un choix M accepte

Définition 2.2 (ACCESSIBILITE)
- **INPUT** : *$G = (V, E)$ orienté et $u, v \in V$*
- **QUESTION** : *Existe-t-il un chemin entre u et v ?*

Théorème 2.4 ()
- *On peut résoudre ACCESSIBILITE sur un graphe à n sommets en temps $\theta(n^2)$*
- $NSPACE(f(n)) \subseteq \bigcup_c DTIME(c^{f(n)+\log(n)})$

Démonstration 2.4
- N'importe quel parcours systématique de graphe (parcours en profondeur ou en largeur) est bon.
- Soit $L \in NSPACE(f)$ et soit $M \in MTN$ en espace f qui reconnait L. Soit x une entrée. On définit le *graphe de configuration* $G(M, x)$ comme suit :
  - Les sommets : Toutes les configurations possibles de M sur x en espace f
  - Les arcs (C, C') existent ssi la machine M peut passer en un seul pas de C à C'

  Alors $x \in L \Leftrightarrow (G(M, x), C_{initial}, C_{final-acceptant}) \in ACCESSIBILITE$

  Taille de $G(M, x)$ (Supp que M a k bandes de travail.)

  $$s * n * (t^{f(n)} * f(n))^k = \theta(c^{f(n)+\log n})$$

  - s est le nombre d'états

- n est la taille de la bande d'entrée
- t est le nombre de symboles
- $f(n)$ est le nombre de positions de la tête de lecture

Donc le théorème 1 conclut.

Remarque 1
La machine simulatrice ne doit pas construire explicitement $G(M,x)$. Il suffit de décider cas par cas au cours de l'algorithme si l'arc existe entre deux configurations données.

Théorème 2.5 (SAVITCH)
- *$ACCESSIBILITE \in DSPACE(\log^2 n)$*
- *Si $f(n) \geq \log n$ alors $NSPACE(f) \subseteq DSPACE(f^2)$*

Démonstration 2.5
- $G(V,E)$ est l'entrée pour ACCESSIBILITE. On définit une relation : $CHEMIN(x,y,i) =$
  - VRAI si il existe un chemin de x à y de longueur au plus 2^i
  - FAUX sinon

  Clairement $(G,u,v) \in ACCESSIBILITE \Leftrightarrow CHEMIN(u,v,\log n)$ est vrai. On a $CHEMIN(x,y,i) \Leftrightarrow \exists z, CHEMIN(x,z,i-1)$ et $CHEMIN(z,y,i-1)$. Comme le graphe a n sommets pour représenter x,y,i,z il faut donc $\log n$ bits.
  C'est une récurrence de profondeur $\log n$. A chaque étape de la récurrence on doit examiner un triplet (x,y,i). Comme la longueur d'un tel triplet est $\leq 3 * \log n$ on peut simuler la récurrence en espace $O(\log^2 n)$
- Soit $L \in NSPACE(f)$ et $M \in MTN$ qui en espace f reconnait L. Soit x une entrée. On considère le graphe de configuration $G(M,x)$. Sa taille est $c^{f(n)}$
  D'après la première partie du théorème ACCESSIBILITE pour ce graphe se décide en espace $f^2(n)$.

Remarque 2
Il faut bien voir qu'on ne construit pas explicitement le graphe.

Théorème 2.6 ()
$PSPACE = \bigcup_c NSPACE(n^c)$

Démonstration 2.6
Trivial

> **Théorème 2.7 ()**
> *$ACCESSIBILITE \in \mathcal{NL}$*

Démonstration 2.7
Sur (G, u, v) on devine un chemin entre u et v et on le verifie arc par arc. Comme on garde sur la bande que ce qui est en train d'être vérifié on reste en espace $3 * \log n$. Soit G un graphe orienté $G(V, E)$ et a un sommet. On définit $f(a) = card\{b \in V, b$ est accessible à partir de $a\}$

> **Définition 2.3 ()**
> $L \subseteq \{0,1\}^*$, $coL = \{x \in \{0,1\}^*, x \notin L\}$

> **Définition 2.4 ()**
> *Soit $f : \{0,1\}^* \to \{0,1\}^*$*
> *f est calculable par une MTND si pour tout $x \in \{0,1\}^*$*
>
> - *Chaque branche de l'arbre de calcul de la machine s'arrete par ou bien ECHEC ou bien par $f(x)$*
> - *Il y a au moins une branche qui s'arrête avec $f(x)$.*

> **Théorème 2.8 ()**
> - *$f \in \mathcal{NL}$*
> - *Si $f(n) \geq \log n$ alors $NSPACE(f) = coNSPACE(f)$*
>
> *En fait cela veut dire que le nombre de noeuds atteignables depuis un noeud x dans un graphe G peut etre calculé par une machine non déterministe en espace $\log n$*

Démonstration 2.8
- **IMMERMANN - SZELEPSCENYI**
  On définit $S_k = \{b \in V, \exists$ un chemin de longueur au plus k allant de a à $b\}$

  1. On va calculer successivement $|S_0|, |S_1|, |S_2|, \cdots, |S_n|$

2. Supposons qu'on a $|S_k|$. $l = 0$ Pour chaque sommet u si $u \in S_k$ alors $l = l + 1$
3. Comment décider si $u \in S_k$?
   compteur = 0;
   reponse = non;
   Pour chaque sommet v do

   - Si $v \in S_{k-1}$ then $compteur+ = 1$
   - Si $(v, u) \in E||v = u$ then $reponse = oui$

   Si $compteur \neq |S_{k-1}|$ alors ECHEC sinon decider si $u \in S_k$ selon *reponse*.
4. Comment décider si $v \in S_{k-1}$?
   Par l'algo $\mathcal{NL}$ standard du theoreme 5. (ie on devine au plus $k - 2$ sommets et on vérifie si ils forment un chemin de longueur au plus $k - 1$ entre a et v)

On montre que si une branche a correctement calculé $|S_{k-1}|$ et prétend de calculer $|S_k|$ alors cette deuxième valeur est aussi correcte. Si une branche n'échoue pas alors pour chaque sommet $v \in S_{k-1}$, la branche devine un chemin de longueur au plus $k - 2$. (Cela est assuré par le test compteut contre $|S_{k-1}|$). Comme cela est vrai pour tout u, cette branche calculera correctement $|S_k|$. De plus il est évident qu'il y a une branche qui devine tout correctement.

- $NSPACE(f) = coNSPACE(f)$
  Soit $L \in NSPACE(f)$ on va montrer que coL (notation $\overline{L} = coL$) $\in NSPACE(f)$ en construisant une MTND qui accepte $\overline{L}$ en espace f. Soit M une MTND qui reconnait L en espace f. Soit x une entrée. On considère $G(M, x)$. Soit $l = |G|$ Comme toute à l'heure on calcule $|S_0|, |S_1|, \cdots, |S_{l-1}|$. Puis au lieu de calculer $|S_l|$, on regarde s'il y a au moins une configuration acceptante finale en S_l.
  Comment évalue t on les branches ?

  - Si ECHEC on refuse
  - Si il existe une configuration acceptante en S_l alors refuse
  - Si pas de configuration acceptante en S_l alors OK. Donc $L(M') = \overline{L}$

2.2 Réductibilité

Définition 2.5 ()
Soit $L_1, L_2 \subseteq \{0,1\}^*$. On dit que L_1 est log-space réductible à $L_2 (L_1 \underset{\mathcal{L}}{\leq} L_2)$ s'il existe une MTD qui en espace $\log n$ calcule une fonction $f : \{0,1\}^* \rightarrow \{0,1\}^*$ telle que $\forall x \in \{0,1\}^* x \in L_1 \Leftrightarrow f(x) \in L_2$

Remarque 3
Une définition similaire est possible par exemple avec $\mathcal{P}$ à la place de $\mathcal{L}$.

Théorème 2.9 ()
$L_1 \underset{\mathcal{L}}{\leq} L_2, L_2 \underset{\mathcal{L}}{\leq} L_3 \Rightarrow L_1 \underset{\mathcal{L}}{\leq} L_3$

Démonstration 2.9
Soit M_1 qui réduit L_1 à L_2 via f. Soit M_2 qui réduit L_2 à L_3 via g. On définit M qui calcule $g \circ f$ de facon naturelle. Le problème est que $f(x)$ peut être beaucoup plus long que $\log n$. M_2 a besoin à chaque instant d'un bit au plus de $f(x)$. Supposons par récurrence que M_2 connait le i^{eme} bit de $f(x)$. Si le bit suivant dont elle a besoin est $(i+1)$ on continue la simulation du calcul de M, sur x jusqu'à ce qu'elle sorte le bit suivant de $f(x)$. Si on a besoin du $(i-1)^{eme}$ bit on resimule tout le calcul de $f(x)$ depuis le premier bit.

Définition 2.6 ()
Soit $\mathcal{C}$ une classe de complexité standard. Un langage L est $\mathcal{C}$ complet si :

- $L \in \mathcal{C}($ *est* $\mathcal{C} facile)$
- $\forall L' \in \mathcal{C}, L' \underset{\mathcal{L}}{\leq} L$ *(est $\mathcal{C}$ difficile)*

Théorème 2.10 ()
Si $L \in \mathcal{C} et L' \underset{\mathcal{L}}{\leq} L$ alors $L' \in \mathcal{C}$

Démonstration 2.10
Evident

2.3 Etude de la classe $\mathcal{P}$

Définition 2.7 ()

- *Une fonction booléenne à n variables est une fonction $f : \{0,1\}^n \to \{0,1\}^m$.*
- *Les formules booléennes à n variables par récurrence : $x_1, x_2, \cdots, x_n$.*
  - *Si φ alors $\neg\varphi$*
  - *Si φ et ψ alors $\varphi \wedge \psi$ et $\varphi \vee \psi$*

Remarque 4
Chaque formule booléenne définit de facon naturelle une fonction booléenne.
$\varphi \to f_\varphi$

Définition 2.8 ()
Une formule est en **CNF** *(* **resp. DNF** *(forme normale conjonctive resp. disjonctive) si :*

- $\varphi = C_1 \wedge C_2 \wedge C_3 \wedge \cdots \wedge C_n$
- $C_i = (y_{i_1} \vee y_{i_2} \vee \cdots \vee y_{i_{j_i}})$
- $y_k \in \{X_1, X_2, \cdots, X_n, \overline{X_1}, \overline{X_2}, \cdots, \overline{X_n}\}$

Théorème 2.11 ()
Pour chaque fonction booléenne f il existe une formule booléenne φ telle que $f_\varphi = f$

Démonstration 2.11
$01101 = \overline{x_1} \wedge x_2 \wedge x_3 \wedge \overline{x_4} \wedge x_5$

Définition 2.9 ()
Un circuit booléen à n variables est un graphe orienté sans boucle. Chaque sommet est de degré entrant $0, 1$ ou 2. Les sommets de degré entrant 0 sont étiquetés par les variables, ce sont les entrées.
Les autres noeuds sont étiquetés par :

$$\neg, \wedge, \vee$$

Un certain nombre de sommets de degré sortant 0 sont identifiés comme des sorties.

Chaque circuit C définit naturellement une fonction booléenne à n variables. Une **expression** est un circuit dont le degré sortant est au plus 1.

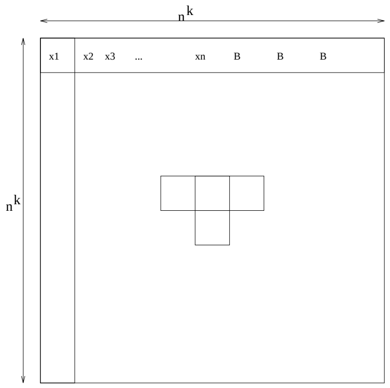
Définition 2.10 (CIRCUIT VALUE)

- *Input : Circuit C à n variables et $a \in \{0,1\}^n$*
- *Output : $C(a)$*

Théorème 2.12 ()
CIRCUIT-VALUE est $\mathcal{P}$-complet.

Démonstration 2.12

- Trivialement circuit-value $\in \mathcal{P}$
- Soit $L \in \mathcal{P}$. On va montrer que $L \underset{\mathcal{L}}{\leq}$circuit-VALUE. Soit M une MTD qui en temps n reconnait L. Soit $x \in \{0,1\}^*$ une entrée. On va construire en $\mathcal{L}$ un circuit $\mathcal{C}$ avec les entrées constantes telle que:
  M accepte $x \Leftrightarrow \mathcal{C} = 1$ (la valeur du circuit est 1).
  On peut décrire le calcul d'une MTD par un tableau.



La première ligne est la bande de travail. Un élément du tableau est référencé par $T_{i,j}, 0 \leq i, j \leq n^k$. B est un blanc sur la bande d'entrée. Le première indice d'un élément représente le temps.$B_{i,j,k}$ sont les bits qui codent $T_{i,j}$. $T_{i,j} \in \sum_a lphabetFini \cup \{\sigma_q : \sigma \in \Sigma, q \in Q$ (états) $\} \cup \{yes, no\}$ Le yes/no indique sile résultat est accepté ou non.

Le contenu de $T_{i,j}$ est déterminé par les contenus de $T_{i-1,j}, T_{i-1,j-1}, T_{i-1,j+1}$.
$\exists n \in \mathbb{N}, |S| \leq 2^m$.
Le changement est une fonction $f : S^3 \rightarrow S$ c'est à dire $f : \{0,1\}^{3m} \rightarrow \{0,1\}^m$.
De plus f est indépendante de x. Elle est déterminée par la fonction de transition de M. On peut construire un circuit C (de taille constante) qui calcule f. La première ligne du tableau est déterminée par le mot x.
Le circuit final $\mathcal{C}$ représentant tout le calcul du tableau sera n^{2k} copies de C mises ensemble de facon naturelle. L'assignation pour $\mathcal{C}$ (les entrées constantes) est donné par la description de la ligne 0 du tableau ie par l'entrée x. Supposons qu'on code yes par $\overbrace{1\cdots}^{n}$ et no par $\overbrace{0\cdots}^{n}$ et mettons que par convention on mette le résultat dans la case 1 alors :
$\mathcal{C} = 1 \Leftrightarrow B_{n^k,1,1} = 1$. On choisit donc $B_{n^k,1,1}$ comme bit de sortie de $\mathcal{C}$.

Définition 2.11 (circuit monotone)
Un circuit monotone est un circuit sans négation.

Définition 2.12 (MONOTONE CIRCUIT VALUE)

- *Entrée : Circuit monotone C et assignation $\overline{a}$*
- *Question : C(a)*

Théorème 2.13 (MCV)
MCV est $\mathcal{P}$-complet

Démonstration 2.13
$CV \underset{\mathcal{L}}{\leq} MCV$
$C \rightarrow D$
On élimine les $\neg$ par les lois de Morgan

$$\neg(E_1 \vee E_2) = (\neg E_1) \wedge (\neg E_2)$$

D contient deux fois plus de portes que C Pour chaque porte G de C on construit aussi une porte $g_\neg$ telle que la fonction calculée par $G_\neg$ est la négation de la fonction calculée par G. De plus, D est monotone. La construction de la partie négative de D se fait par récurrence selon la profondeur de la porte G.

2.4 Classe $\mathcal{NP}$

Définition 2.13 (Relation binaire polynomiale)
Une relation binaire R(x,y) est polynomiale si :

- $R \in \mathcal{P}$
- $\exists$ *polynome* $p(n)$ *tel que* $\forall(x,y), R(x,y) \Rightarrow |y| = p(|x|)$

Théorème 2.14 ()
$L \in \mathcal{NP}$ *ssi* $\exists R$ *relation polynomiale telle que :*

$$\forall x, x \in L \Leftrightarrow \exists y, R(x,y)$$

Démonstration 2.14

- $\Rightarrow$ y est la suite des choix pour x.
- $\Leftarrow$ Supposons que R existe. Alors L est reconnu par la MTND suivante. Pour une entrée x M devine un y de longueur au plus $|x|^k$ et alors utilise l'algorithme polynomial sur x;y pour voir si $(x,y) \in R$. Si oui alors accepte sinon refuse. Il est évident que un chemin acceptant existe si et seulement si $x \in L$.

Définition 2.14 (SAT et circuit SAT)

- *Circuit SAT*
  Entrée : Circuit C.
  Question : Existe-t-il une assignation a telle que $C(a) = 1$ *?*
- *SATISFIABILITE*
  Entrée : Formule booléenne en CNF
  Question : La meme

Théorème 2.15 (complétude SAT)

- *Circuit SAT est $\mathcal{NP}$ complet*
- *SATISFIABILITE est $\mathcal{NP}$ complet*

Démonstration 2.15
Soit $L \in \mathcal{NP}$ et soit M une MTND qui en temps n^k reconnait L et soit $x \in \{0,1\}^*$.On va construire une réduction R en $\mathcal{L}$. $x \xrightarrow{R} \mathcal{C}$. telle que
M accepte x $\Leftrightarrow \exists a, \mathcal{C}(a) = 1$.

Pour chaque suite de choix non déterministe on peut créer un tableau pour le calcul de M sur x.
$T(i,j)$ dépend de $T(i-1,l)$ pour $l = j-1, j, j+1$ et dépend du choix binaire b_{i-1} à l'instant $i-1$
C calcule $T(i,j)$. Il a besoin de $3n+1$ bitspour les trois entiers plus un pour le bit de décision. La construction se fait comme tout à l'heure sauf qu'il y a une entrée supplémentaire pour le choix binaire. $\mathcal{C}$ est construit comme tout à l'heure sauf que les bits $b_0, b_1, \cdots, b_{n^k-1}$ sont les entrées.

Théorème 2.16 (CSAT $\Leftrightarrow$ SAT)
CSAT $\underset{\mathcal{L}}{\leq}$ *SAT*

Démonstration 2.16
Soit C un circuit. On va construire une forme ϕ en CNF telle que :
C est satisfiable $\Leftrightarrow$ ϕ est satisfiable.
Pour chaque porte G de C on introduit une variable ϕ_G et ϕ sera la conjonction de ces formules. ϕ_G décrira la fonction calculée à la porte G.Exemple :

- $x \Leftrightarrow g, (\neg x \vee g) \wedge (\neg g \vee x)$
- $\neg g \Leftrightarrow g, (h \vee g) \wedge (\neg h \vee \neg g)$
- $h_1 \vee h_2 \Leftrightarrow g, (\neg g \vee h_1 \vee h_2) \wedge (\neg h_1 \vee g) \wedge (\neg h_2 \vee g)$

$\phi = \underset{Gporte}{\wedge} \phi_G \wedge (s)$ s pour la sortie.

- $\Rightarrow$ Soit a tel que $C(a) = 1$. Dans ϕ on garde pour les variables initiales x et pour chaque g on donne la valeur $G(a)$.
- $\Leftarrow$ La restriction de l'assignation de ϕ aux variables en C satisfait L. Pour cela on utilise que ϕ_G est vrai $\Leftrightarrow g = G(a)$

Définition 2.15 (kSAT)

- *Entrée : ϕ en CNF ou chaque clause contient exactement k littéraux. $\phi = () \wedge () \wedge$ ou $()$ est un littéral.*
- *Question : Existe-t-il une assignation telle que $\phi(a) = 1$.*

Théorème 2.17 (3-SAT)
3-SAT est $\mathcal{NP}$-complet

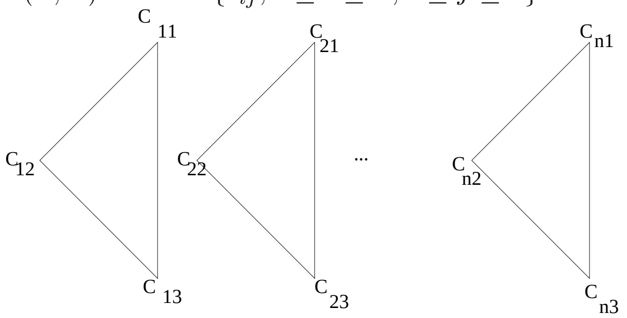
Démonstration 2.17
Dans la construction précédente chaque clause est de longueur au plus 3. Donc clause de longueur 1 et 2 aussi !!! Par exemple si on a $(\alpha \vee \beta)$ on introduit une variable z et on a $(\alpha \vee \beta \vee z) \wedge (\alpha \vee \beta \vee \overline{z})$.

Exemple de problèmes $\mathcal{NP}$ complet

- Three dimensionnal matching
- Cycle hamiltonien. Existe-t-il un cycle qui passe une et une seule fois par chaque sommet d'un graphe ?
- Partition :
  - Entrée : $a_1, a_2, \cdots, a_n$ entiers.
  - Question : Existe-t-il $S \subset \{1, 2, \cdots, n\}, \sum_{i \in S} a_i = \sum_{i \notin S} a_i$
- Independant set
  - Entrée : $G = (V, E), k \in \mathbb{N}$
  - Question : Existe-t-il un ensemble stable de taille supérieure ou égale à k ? (ie $V' \subseteq V, \forall (u, v) \in V', (u, v) \notin E$)
- Vertex cover.
  - Entrée : $G = (V, E)$
  - Question : Existe-t-il une couverture de taille au plus k ? ie $V' \subseteq V, \forall e \in E, V' \cap e \neq \emptyset$
- Clique :
  - Entrée : $G = (V, E), k$
  - Question : Existe-t-il une clique de taille au moins k ?

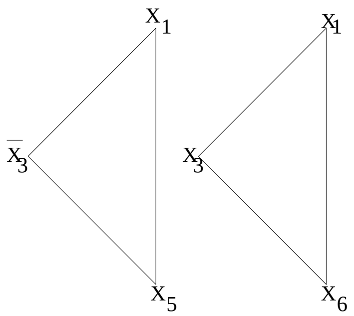
Théorème 2.18 (Independance set)
Independance set est $\mathcal{NP}$ complet

Démonstration 2.18
On montre que $3-SAT \underset{\mathcal{L}}{\leq} IND.SET$.
Soit $\phi = (\alpha_{1,1} \vee \alpha_{1,2} \vee \alpha 1, 3) \wedge (\cdots) \wedge (\alpha_{n,1} \vee \alpha_{n,2} \vee \alpha_{n,3})$.
$\alpha_{i,j} \in \{X_1, \overline{X_1}, \cdots, X_i, \overline{X_i}\}$
$G = (V, E)$ ou $V = \{c_{ij}, 1 \leq i \leq n, 1 \leq j \leq 3\}$



$$E = \{\{c_{i,1}, c_{i,2}\}, \{c_{i,1}, c_{i,3}\}, \{c_{i,2}, c_{i,3}\}\} \cup \{\{c_{i,j}, c_{i',j'}\}, \exists x, \alpha_{ij} = \overline{x}\}$$

Exemple : $\phi = (X_1\ vee\overline{X_3} \vee X_5) \wedge (X_1 \vee X_3 \vee X_6)$



$k = m$

- $\Rightarrow, \phi(a) = 1$
  $\forall i \exists j, \alpha_{i,j}$ est vrai . On prend $c_{i,j}$. Les $c_{i,j}$ forment un ensemble indépendant de taille n.
- $\Leftarrow$ Un ensemble indépendant de taille supérieur à m est en fait exactement de taille m et contient exactement un sommet de chaque triangle. Prenons les littéraux qui correspondent à ces sommets et donnons leur la valeur 1. Pas de contradiction car X est relié à $\overline{X}$. Cela satisfait ϕ car dans chaque triangle il y a un sommet choisi. Donc la clause correspondante est satisfaite.

Théorème 2.19 (Vertex cover)
Vertex cover est $\mathcal{NP}$ complet

Démonstration 2.19
Montrons que IND SET $\underset{\mathcal{L}}{\leq}$ VC.
$G(V,E), k \rightarrow G = (V,E), n-k$
$V' eststable \Leftrightarrow V - V'$ est une couverture.

Théorème 2.20 (CLIQUE)
Clique est $\mathcal{NP}$ complet

Démonstration 2.20
Montrons que IND SET $\underset{\mathcal{L}}{\leq} CLIQUE$.
$G(V,E), k \rightarrow G^c(V,E^c), k$ ou E^c est le complémentaire de E.

2.5 Digression sur la hiérarchie polynomiale

- $\mathcal{P}, \Delta_0, R(X), R$ temps polynomial
- $\mathcal{NP}, \Sigma_1, R(X) \equiv \exists y, S(x,y), S$ polynomial
- $co-\mathcal{NP}, \Pi_1, R(X) \equiv \forall y, S(x,y), S$ polynomiale
- $\Sigma_2, R(X) \equiv \exists y \forall z, S(x,y,z), S$polynomial
- $\Sigma_k, R(x) \equiv \exists y_1 \forall y_2 \exists \cdots \forall \cdots\cdots Q y_k S(x, y_1, y_2, \cdots, y_k), S$ polynomiale
- $\Pi_k = co - \Sigma_k$

$H1 = \{(i, x, 0^n)$, M accepte x en au plus n pas $\}$
H_1 est $\mathcal{NP}-$complet. La hiérarchie polynomiale $HP = \{\Sigma_k, \Pi_k\}$
Conjecture : $\forall k, \Pi_k \neq \Sigma_k \not\subseteq \Sigma_{k-s}$.

Définition 2.16 (QBF : Formule booléenne quantifiée)
- *Entrée :* $\exists x_1, \forall x_2, \exists x_3, \forall x_4, \cdots, \varphi(x_1 \cdots x_n)$ *ou* φ *est sans quantficateurs.*
- *Question : Esct-ce que* φ *est vraie ?*

Théorème 2.21 ()
QBF est PSPACE difficile

Démonstration 2.21

- $QBF \in PSPACE$
  Evaluation récursive. Récurrence de profondeur n. Pour une assignation, $\varphi(a_1, \cdots, a_n)$ s'évalue en espace linéaire.

- QBF est PSPACE difficile.
  Soit $L \in$ PSPACE. Soit M une machine de Turing qui en espace n^k reconnait L et soit $x \in \{0,1\}^*$. Le tableau de calcul de la machine M sur l'entrée x fait n^k cases de large et d^{n^k} cases de haut représentant le temps. Soit $S = \Sigma \cup \{\sigma_g, \sigma \in \Sigma, g \in Q\} \cup \{yes, no\}$. On introduit les variables booléennes $S_{i,X}, 1 \leq i \leq n^k, X \in S$ pour décrire une configuration.
  $S_{i,X} =$
  - 1 si la $i^e me$ cellule contient x
  - 0 sinon

  Définition 2.17 (CHEMIN)
  CHEMIN(I, I', t) = vrai si M passe de I à I' en au plus 2^k pas de temps et faux sinon

  $x \in L \Leftrightarrow$ CHEMIN$(I_{init}, I_{fin}, \log_2(dn^k))$
  I_{init} est la configuration initiale et I_{fin} la configuration finale acceptante. On a envie d'écrire
  CHEMIN$(I, I', t) \Leftrightarrow \exists I_m$,CHEMIN$(I, I_m, t-1) \wedge$CHEMIN$(I_m, I', t-1)$. On écrira plutot :
  CHEMIN$(I, I', t') \Leftrightarrow \exists I_m, \forall (I_A, I_B) \in \{(I, I_m), (I_m, I')\}$ CHEMIN$(I_A, I_B, t-1)$.
  A la fin on obtient une QBF de taille $O(n^{2k})$. Pour mettre sous forme prénexe il suffit de voir que les quantificateurs portent toujours sur les variables qui sont à leur droite.

Remarque 5
QBF reste PSPACE difficile si la partie non quantifiée de la formule est une instance de 3-SAT.

2.6 Classe $\mathcal{NL}$

Théorème 2.22 (ACCESSIBILITE)
ACCESSIBILITE est $\mathcal{NL}$ complet.

Démonstration 2.22

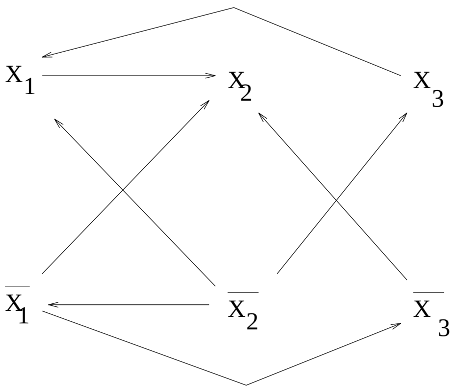
- $\in \mathcal{NL}$ on devine un chemin

- Soit $L \in \mathcal{NL}$ et soit M une MTND qui en espace $\log n$ reconnait L. Soit x une entrée ($\in \{0,1\}^*$). Soit $G(M,x)$ le graphe de configuration la taille de G est polynomiale.
$x \in L \Leftrightarrow (G, I_{init}, I_{final}) \in$ACCESSIBILITE.

Théorème 2.23 (2SAT)
2SAT est $\mathcal{NL}$ complet

Démonstration 2.23

- $\rightarrow$ 2SAT $\in \mathcal{NL}$
En fait on va montrer que 2SAT est dans $co{-}\mathcal{NL}$ mais $\mathcal{NL} = co{-}\mathcal{NL}$.
$\varphi = (\alpha_{1,1} \vee \alpha_{1,2}) \wedge \cdots \wedge (\alpha_{n,1} \vee \alpha_{n,k})$
$(\alpha \vee \beta) \equiv (\neg\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \alpha)$. On définit $G = (V, E)$ orienté ou
$V = \{x_1, \overline{x_1}, \cdots, x_n, \overline{x_n}\}$
$E = \bigcup_{\alpha \vee \beta clause} \{(\neg\alpha, \beta), (\neg\beta, \alpha)\}$



Exemple : $(x_1 \vee x_2) \wedge (x_1 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2) \wedge (x_2 \vee x_3)$

Théorème 2.24 ()
φ n'est pas satisfiable $\Leftrightarrow \exists x, x \rightarrow \overline{x}$ ou $\overline{x} \rightarrow x$

Démonstration 2.24

- $\Leftarrow$ On montre par l'absurde. Supposons que φ est satisfiable et pour une configuration satisfiable supposons sans perte de généralité que $x = 1$.
$x \rightarrow v_1 \rightarrow v_2 \cdots \rightarrow v_k \rightarrow \overline{x}$
$\exists i, v_i = 1 et v_{i+1} = 0$. La clause $v_i \Rightarrow v_{i+1}$ est fausse.
- $\Rightarrow$ Supposons que $\exists x'$ tell que $x \rightarrow \overline{x}$ ou $\overline{x} \rightarrow x$. Alors on va créer une configuration satisfaisable par φ. On prend un sommet quelconqie (Littéralement et on lui donne la valeur vraie 1. On donne aussi 1 à tous les sommets accessibles à partir de celui-ci.

On donne ausi la valeur 0 aux littéraux opposés.) Si il y a encore des variables sans valeur on répète la procédure.
Si $\alpha \to \beta \cdots \neg\gamma$
et $\gamma \to \neg\beta$ alors c'est impossible car alors $\beta \to \neg\gamma$ et donc γ a déja eu une valeur.
De plus l'asignation est satisfaisante : $(\alpha \vee \beta)$
Si α assigné en premier alors OK mais si α recoit 0 c'est que $\neg\alpha$ a recu 1 donc $\exists\gamma$ tel que : $\gamma \to \neg\alpha$ ou $\neg\alpha \to \beta$ donc $\gamma \to \neg\alpha \to \beta$

- En exo montrer que 2SAT est $\mathcal{NL}$ difficile

3 Complexité parallèle

Définition 3.1 ()
Soit$\mathcal{C} = (C_n)_{1 \leq n}$ une famille de circuits. C_n est un circuit à n variables. $\mathcal{C}$ calcule le langage L si $\forall x \in \Sigma^*, |x| = n, x \in L \Leftrightarrow C_n(x) = 1$

Définition 3.2 (Taille et profondeur)
Soit C un circuit. La taille de C est le nombre de portes de C. La profondeur de C est la longueur du plus long chemin entre une entrée et une sortie.

Remarque 6
$\forall L, \exists \mathcal{C} = (C_n), \mathcal{C}$ calcule L

En effet C_n peut etre la forme DNF pour $L^n = \{x \in L, |x| = n\}$

Remarque 7
Cette famille est de taille exponentielle.

Définition 3.3 (Complexité de circuit)
Soit $L \in \{0,1\}^*$ la complexité de circuit de L (en notation $S(L)$) est la taille de la plus petite famille de circuit qui reconnait L. Idem pour la profondeur avec $d(L)$

Théorème 3.1 ($S(L)$)
Si $L \in DTIME(f(n))$ alors $S(L) = O(f^2(n))$

Démonstration 3.1
Comme P complétude de CV

Théorème 3.2 ()
$\mathcal{P}$ admet des circuits polynomiaux.
Question : Est-ce que les circuits de taille polynomiale calcule les langages en $\mathcal{P}$
NON !!!!

Démonstration 3.2
Soit L un langage non récursif tel que $\forall n$ ou bien $\forall x, (|x| = n) x \in L$ ou bien $\forall x, (|x| = n) x \notin L$

Définition 3.4 ()
$\mathcal{C} = (C_n)$ est une famille $\mathcal{L}-$ uniforme de circuits si la fonction f qui à l'entrée n associe la description de C_n est calculable en $\mathcal{L}$.
$f : 0^n \to$ description de C_n.

Description = suite de description de portes (= identificateur, identificateur gauche,identificateur droit, type).

Théorème 3.3 ()
$L \in \mathcal{P} \Leftrightarrow L$ est accepté par une famille uniforme de circuits de taille polynomiaux.

Démonstration 3.3

- $\Rightarrow$ déjà fait
- $\Leftarrow$ On construit C_n et on calcule $C_n(x)$

Définition 3.5 (NC)
$NC^{(i)} = \{L \in \{0,1\}^*, \exists$ une faille uniforme de circuits de taille polynomiale et de profondeur $O((\log n)^i)$ qui accepte L $\}$.

Définition 3.6 (AC)
$AC^{(i)}$ =Comme $NC^{(i)}$ sauf que les porte $\vee$ et $\wedge$ sont non bornées, ie peuvent avoir un nombre quelconque d'entrées.

Théorème 3.4 ()
$\forall i, NC^{(i)} \subseteq AC^{(i)}$
$\forall i, AC^{(i)} \subseteq NC^{(i+1)}$

Démonstration 3.4
Trivial

Définition 3.7 (NC)
$NC = \cup NC^{(i)}$
$NC \subseteq \mathcal{P}$

NC pour Nick's Class (Nick Pippenger).
On avait défini les classes et les inclusions :

$$AC^0 \subseteq NC^1 \subseteq AC^1 \subseteq \cdots \subseteq NC \subseteq \mathcal{P}$$

Théorème 3.5 ()
$NC^1 \subseteq \mathcal{L} \subseteq \mathcal{N}L \subseteq AC^1$

Démonstration 3.5

- On prouve que ACCESSIBILITE $\in AC^1$. Soit $G(V, E)$ un graph orienté et s,t en entrée. M est la matrice d'adjacence de G(matrice booléenne) ie :
  $m_{ij} = 1$ si $(i, j) \in E$ et 0 sinon. On regarde M^k
  $M^k = (m_{ij})$, $m_{ij} = 1$ si il existe un chemin de longueur k de i à j et 0 sinon. $M^k = M^1 \vee M^2 \vee \cdots \vee M^m$ Le circuit AC^1 pour M^* : Hauteur $\log n$ et profondeur constante. Si N est une matrice booléenne alors :
  $(N \times N)_{ij} = \bigvee_{k=1}^{n} (m_{ik} \wedge mkj)$

- $NC^1 \subseteq \mathcal{L}$
  Comment peut on évaluer un circuit de profondeur $\log n$ en $\mathcal{L}$?
  On évalue le circuit par un parcours de graphe en profondeur. La profondeur de la récurrence est $\log n$. Malheureusement, on ne peut pas garder sur la pile de la récurrence les noms des sommets visités sur la bande actuellement évaluée parce que cela ferait espace $\mathcal{O}(\log^2 n)$. C'est pourquoi, à chaque niveau de la récurrence on garde seulement un nombre constant de bits qui indiquent pour les sommets concernés que pqr exemple le sous-arbre gauche a déjà été évalué, il vaut (1 ou 0) et on est en train d'évaluer le sous-arbre droit.
  Bien sur, le problème est de pouvoir retourner dans le parcours. Dans un tel cas on recalcule la même bande, cette fois-ci en gardant aussi les noms des deux derniers sommets visités. Quand on arrive au sommet actuel, l'autre sommet gardé serat son père.

Corollaire : $NC^k \subseteq DSPACE(\log^k n)$
Corollaire : Soit L un langagge $\mathcal{P}$-complet. Si $L \in NC$ alors $\mathcal{P} = NC$

Quelques problèmes en NC :

- ADDITION
  - Input : $a = a_{n-1}a_{n-2}\cdots a_1a_0$
    $b = b_{n-1}b_{n-2}\cdots b_1b_0$
  - Output : $c = c_nc_{n-1}\cdots c_1c_0, c = a + b$

Théorème 3.6 (ADDITION)
$ADDITION \in AC^0$

Démonstration 3.6
$r_0 = a_0 \wedge b_0$
$r_i = (a_i \wedge b_i) \vee (v_{i-1} \wedge (a_i \vee b_i))$
Voilà comment on définit la retenue. $c_i = a_i \oplus b_i \oplus v_{i-1}$ On définit
$g_i = a_i \wedge b_i, p_i = a_i \vee b_i$
$r_i = g_i \vee (v_{i-1} \wedge p_i)$
$r_i = g_i \vee (g_{i-1} \wedge p_0) \vee (g_{i-2} \wedge p_{i-1} \wedge p_i) \cdots \vee (g_0 \wedge p_1 \wedge p_2 \wedge \cdots \wedge p_i)$
$g_i p_i$: profondeur 1
r_i : profondeur 3
c_i : profondeur 5-6

- ITERATED ADDITION
  - Input : $a_1 = a_{n,n-1} \cdots a_{1,1} a_{1_0}$
    $\cdots$
    a_n
  - Output : $\sum a$

Théorème 3.7 (ITERATED ADDITION)
ITERATED ADDITION $\in NC^1$

Démonstration 3.7
Idée 2FOR3 TRICK
On peut réduire la somme de 3 entiers à la somme de 2 entiers en profondeur constante.
$a = a_{n-1} \cdots a_1 a_0, a_i$
$b = b_{n-1} \cdots b_1 b_0, +b_i$
$c = c_{n-1} \cdots c_1 c_0, +c_i$
$a_i + p_i + c_i = p_i g_i$
On définit $p = p_{n-1} \cdots p_1 p_0$
$q = q_{n-1} \cdots q_1 q_0$
$a + b + c = 2p + q$

$$\begin{array}{cccccc}
1 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 \\
\hline
 & & & & 1 & 0 \\
 & & & 1 & 1 & \\
 & & 0 & 1 & & \\
 & 0 & 1 & & & \\
1 & 1 & & & &
\end{array}$$

On répète le TRICK $\log n$ fois.

- INTEGER MULTIPLICATION

– Input : $a = a_{n-1} \cdots a_1 a_0$
  $b = b_{n-1} \cdots b_1 b_0$
– Output : $a * b$

Théorème 3.8 (INTEGER MULTIPLICATION)
INTEGER MULTIPLICATION $\in NC^1$

Démonstration 3.8
Parce que $*$ est en fait ITERATED INT ADD + decalage.

- INTEGER MATRIX MULTIPLICATION
  – Input : A, B deux matrices $n \times n$ avec les entrées à n bits.
  – Output : $C = A \times B$

Théorème 3.9 (INT MAT MULT)
INTEGER MATRIX MULTIPLICATION $\in NC^1$

Démonstration 3.9
$c_{ij} = \sum_k a_{ik} * b_{kj}$
Processeur(i, j, k) calcule $a_{ik} b_{kj}$ en NC^1
Processeur(i, j) calcule c_{ij} en NC^1

- ITERATED INTEGER MATRIX MULTIPLICATION
  – Input : $M_1, m_2, \cdots, M_n$ matrices $n \times n$ avec n entrées à n bits.
  – Output : $M = \prod_{i=1}^{n} M_i$

Théorème 3.10 (IT INT MAT MULT)
ITERATED INTEGER MATRIX MULTIPLICATION $\in NC^2$

Démonstration 3.10
En fait arbre binaire de hauteur $\log n$ dont chaque noeud est de profondeur $\log n$

- POLYNOME CARACTERISTIQUE
  – Input : A matrice $n * n$ avec des entrées à n bits.
  – Output : $\rho(a) = det(A - \lambda I) = \rho_0 \lambda^n + \cdots + \rho_{n-1} \lambda + \rho_n$

Théorème 3.11 (POLY CARAC)
POLY CARAC $\in NC^2$

Démonstration 3.11
CSANKT / BERKOWITZ / CHISTOC

Esquisse : $A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & & & \\ \cdots & & M & \\ a_{n1} & & & \end{pmatrix}$ $R = (a_{12} \cdots a_{1n})$ $S = \begin{pmatrix} a_{11} \\ a_{21} \\ \cdots \\ a_{n1} \end{pmatrix}$

$P(\lambda) = det(A - \lambda I)$ $q(\lambda) = det(M - \lambda I) = q_0\lambda^{n-1} + \cdots + q_{n-2}\lambda + q_{n-1}$
Lemme1 : $p(\lambda) = (a_{11} - \lambda)q(\lambda) - R * adj(M - \lambda I) * S$
Lemme2 : $adj(M - \lambda I) = -\sum_{k=2}^{n}(M^{k-2}q_0 + \cdots + m^1 q_{k-3} + I q_{k-2})\lambda^{n-k}$
Preuve1 : Développer le det selon la première ligne.
Preuve2 : On montre que le coté droit $*(M - \lambda I) = det(M - \lambda I)$. On utilise pour cela Cayley Hamilton.
Fin de la preuve :
Matrice $(n+1) * n$

$$\begin{pmatrix} p_0 \\ p_1 \\ \cdots \\ \cdots \\ \cdots \\ p_{n-1} \\ p_n \end{pmatrix} = \begin{pmatrix} -1 & & & & \\ a_{11} & -1 & & (0) & \\ -RS & a_{11} & -1 & & \\ -RMS & -RS & a_{11} & -1 & \\ \cdots & & & & \\ \cdots & & & & -1 \\ -RM^{n-2}S & -RM^{n-3}S & \cdots & & a_{11} \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ \cdots \\ \cdots \\ \cdots \\ q_{n-2} \\ q_{n-1} \end{pmatrix}$$

On recommence pour $\begin{pmatrix} q_0 \\ \cdots \\ q_{n-1} \end{pmatrix}$ et on a un produit fini de matrices qui donne $\begin{pmatrix} p_0 \\ \cdots \\ p_n \end{pmatrix}$

On a bien ramener le calcul du poly carac à un produit de matrices $\begin{pmatrix} p_0 \\ \cdots \\ p_n \end{pmatrix} = C_n \times C_{n-1} \cdots \times C_1$ ou C_i est une matrice $(i+1) * i$.

Corollaire : Le déterminant $\in NC^2$

- PARITE
  - Input : $X_1, X_2, \cdots, X_n \in \{0, 1\}$
  - Output : $\sum_{i=1}^{n} x_i mod2$

Théorème 3.12 (PARITE)
$PARITE \notin AC^0$

Démonstration 3.12

- Fint, Sah, Sipser
- Yao
- Smolenski
- Hartand

dem de SMOLENSKI
Pour cette boene inférieure on peut prendre AC non uniforme. $F_n = \{f : \{0,1\}^n \to K\}$ ou K est un corps. On va prendre $K = GF(3)$. F_n un espace vectoriel de dimension 2^n. Une base de F_n est la base des monomes. Un monome $\prod_{i \in S} x_i, S \subseteq \{1, 2, \cdots, n\}$
Exemple $X_1, X_2, \cdots, X_n, X_1X_2, \cdots, X_1X_2 \cdots X_n$

Degré d'un monome : degré($\prod_{i \in S} X_i$) $= |S|$ Comme les monomes forment une base une $f \in F_n$ s'écrit comme une somme de monomes.

Définition 3.8 (degré)
$d(f)$ =degré d'un polynome est le degré max qui intervient dans la décomposition dans la base des monomes.

Exemple : f pour $n = 2$
$f(x_1, x_2) = (x_1 - 1) * (x_2 - 1)$

$$\begin{array}{cc|cc} 0 & 0 & 1 & \leftarrow \\ 0 & 1 & 0 & \\ 1 & 0 & 0 & \leftarrow \\ 1 & 1 & 0 & \leftarrow \end{array} \quad d(f) = 2$$

f restreint aux $\leftarrow$ est représenté par $1 + 2x_1 + 2ex_2 + ex_1x_2$ ou $e \in \{0, 1, 2\}$.
Soit $D \subseteq \{0,1\}^n$

Définition 3.9 ()
Un polynome p représente f sur D si $\forall x \in D, si \forall x \in D, f(x) = p(x)$

Définition 3.10 ()
$d_D(f) = min(p)$, p représente f sur D.

Ici $D = \{00, 10, 11\}$ $d_D(f) = 1$ Idée de Smolenski

- Pour chaque fonction calculée par un circuit de longueur constante, il existe un domaine assez grand D tel que $d_D(f)$ est petit.

– Pour D grand, d_D(PARITE) est grand.

Théorème 3.13 (Lemme 1)
Soit $l \in \mathbb{N}$ et soit une fonction qui est calculable par un circuit de taille S et de profondeur k. Alors $\exists D \subset \{0,1\}^n$ tel que :

- $|D| \geq 2 * (1 - S * 3^{-l})$
- $d_D(f) \leq (2l)^k$

Démonstration 3.13
Pour chaque porte du circuit, on va montrer qu'il existe $E \subseteq \{0,1\}^m$ tel que $|E| \leq 3^{-l}2^n$ et la fonction calculée pour la porte restreinte à $\{0,1\}^n - E$ est calculable par un polynome de degré au plus $2l$ en fonction de ses entrées.
Soit E^* l'union de tous ces E, alors $D = \{0,1\}^n - E^*$.
Porte $\neg$: $\neg g(x) = 1 - g(x), d_{\{0,1\}^n}(1-y) = 1$
Porte $\vee$: $\vee_{1 \leq i \leq t} g_i(X_1, \cdots, X_n)$
Cas 2 : $y_1 \vee y_2 = (1-y_1)(1-y_2) = -y_1y_2 + y_1 + y_2$
La fonction $\vee$ est de degré t (si t entrées) sur $\{0,1\}^n$. On va montrer qu'il existe E tel que $|E| \leq 3^{-l}2^n$ tel que $\cdots$
Soit $\overline{C} = (C_{ij})_{1 \leq i \leq l, 1 \leq j \leq t}, l \times t$ eléments de $GF(3)$.
$\forall \overline{C}$ on définit la fonction $f_{\overline{C}}(y_1, \cdots, y_t) = \vee_{i=1}^{l} (\sum_{j=1}^{t} c_{ij}y_j)^2$
Fixons $X \in \{0,1\}^n$, choisissons les $c_{ij} \in \{0,1,2\}$ de facon indépendante et uniforme.
$Pr_E(g_1(X) \vee \cdots \vee g_t(X) \neq f_i(g_1(X), \cdots, g_t(X))) \leq 3^{-l}$

- **Premier cas :** $\forall j, g_j(X) = 0$
- **Deuxième cas :** $\exists j_0, g_{j_0}(X) = 1$
  Pour tout i $(\sum_{j=1}^{t} c_{ij}g_i(X))^2 = (\sum_{j \neq j_0} c_{ij}g_j(X) + c_{ij_0})^2$, $c_{ij_0} \in \{0,1,2\}$ et $\sum_{j \neq j_0}$ est un nombre aléatoire. Donc probabilité de $\frac{1}{3}$ que ce soit nul. Donc $Pr(\forall i (\sum c_{ij}g_j(X))^2 = 0) = 3^{-l}$
  Comme cela est vrai pour tout x :
  $Pr_{\overline{C},X}(\vee_{j=1}^{t} g_j(X) \neq f_i(g_1, \cdots, g_j(X))) \leq 3^{-l}$ ou X est pris aléatoirement de facon uniforme. donc :
  $\exists \overline{C}, Pr_{\overline{X}}(\vee_{j=1}^{t} g_j(X) \neq f_i(g_1(X) \cdots g_t(X))) \leq 3^{-l}$
  Cela veut dire que si on définit $E = \{X, \vee g_j \neq f_i\}$ alors $|E| \leq 2^n 3^{-l}$ et en dehors de e $f_{\overline{C}}$ calcule parfaitement $\vee$.

Théorème 3.14 (Lemme2)
Soit $D \subseteq \{0,1\}^n$, si $d_D(PARITE) = h$ alors $|D| \leq \sum_{k=0}^{\frac{n}{2}+h} \binom{n}{k}$

Démonstration 3.14
On montre que pour chaque fonction $f, d_D(f) \leq \frac{n}{2} + h$.
Parite$(X_1, \cdots, X_n) = \prod_{i=1}^{n}(X_i + 1) - 1$.
On définit $y_i = X_i + 1$. Si $X_i = 0$ ou 1 alors $y_i = 1$ ou 2. Par hypothèse $d_D(\prod_{i=1}^{n} y_i) = h$
On montre que $\forall S \subset \{x_1, \cdots, x_n\}, d(\prod_{i\in S} y_i)_D \leq \frac{n}{2} + h$

- Cas 1 : Si $|S| \leq \frac{n}{2} + h$ alors gagné.
- Cas 2 : Si $|S| > \frac{n}{2} + h$, $\prod_{i\in S} y_i = \prod_1^n y_i \prod_{i\notin S} y_i$ car $y_i^2 = 1$.
  $d_D(\prod i \in Sy_i) \leq h + \frac{n}{2} - h = \frac{n}{2}$
  $F_D = \{f : D \rightarrow K$ espace vectoriel de dimension $|D|\}$.Les monomes de degré $\leq \frac{n}{2} + h$ engendrent cet espace. (En fait plus grand que cet espace.) Donc $|D| \leq \sum_{k=0}^{\frac{n}{2}+h} \binom{n}{k}$

Fin de la démo de Smolensky
On choisit l tel que $(2l)^k = \frac{sqrtn}{2} \Rightarrow l = \frac{1}{2}(\frac{1}{2})^{\frac{1}{k}} n^{\frac{1}{2k}}$ Je suppose que PARITE est calculé par un circuit de taille S et de profondeur k (constante).
D'après le lemme 1 $\exists D$ tel que $d_D(PARITE) = \frac{\sqrt{n}}{2}$, $|D| \geq 2^n(1 - S3^{-l})$
D'après le lemme 2 $|D| \leq \sum_{k=0}^{\frac{1}{2}+\frac{\sqrt{n}}{2}} \binom{n}{k}$
$\Rightarrow 2^n(1 - S3^{-l}) \leq \sum \binom{n}{k}$
$\Rightarrow \binom{n}{\frac{n}{2}} = c\frac{2^n}{\sqrt{n}}$

Théorème 3.15 ()
$\sum_{k=0}^{\frac{1}{2}+\frac{\sqrt{n}}{2}} \binom{n}{k} \leq 0.99 * 2^n$

Démonstration 3.15
$2^{n-1} + \frac{\sqrt{n}2^n}{2\sqrt{n}}$
$(1 - S3^{-l}) \leq 0.01$, $c = \frac{1}{\sqrt{\frac{\Pi}{2}}}$

Définition 3.11 (AC^0-réduction)
Une fonction g est AC^0 -réductible à f s'il existe un circuit de taille polynomiale de profondeur constante avec des portes $\neg, \vee$ et des portes f qui calcule g.

Une porte f est une porte calculant f de ses entrées.

Théorème 3.16 ()
Si $g \underset{AC^0}{\leq} f$ *et* $f \in AC^0$ *alors* $g \in AC^0$

Démonstration 3.16
Evident

Définition 3.12 (Th)
$Th_h(X_1, \cdots, X_n) = \begin{cases} 1 & si \sum_1^n X_i \geq k \\ 0 & sinon \end{cases}$
$\wedge = Th_n$
$\vee = Th_1$

Définition 3.13 (TRI)
$TRI(X_1, \cdots, X_n) = (Th_n(X_1, \cdots X_n), \cdots, Th_1(X_1, \cdots X_n))$

Théorème 3.17 (AJTAI, KOMLOS, SZEMEREDY)
$TRI \in NC^1$
$TRI \notin AC^0$

Démonstration 3.17
On montre que PARITE $\leq_{AC^0}$ TRI.
PARITE$(X_1, \cdots, x_n) = \vee_{i\text{impair}}(\mathrm{Th}_i(X) \wedge \overline{\mathrm{Th}_{i+1}(X)})$ ie au moins i élements mais pas plus de $i+1$.

Définition 3.14 (MAJORITE)
$MAJORITE(X_1, \cdots, X_n) = Th_{n/2}()$

Théorème 3.18 (MAJORITE)
$MAJORITE \notin AC^0$

Démonstration 3.18
$\forall k, \mathrm{Th}_k(X_1, \cdots, X_n) \underset{AC^0}{\leq}$ MAJORITE **Exp**

Définition 3.15 (Random Polynomial)
$RP = \{L \subseteq \{0,1\}^*$, il existe une relation polynomiale$R(x,y)(|y| = P(|x|))$telle que$\forall x, x \in L \Rightarrow \frac{\sharp\{y,(|y|=P(|x|))\wedge R(x,y)\}}{\sharp\{y,|y|=P(|x|)\}} \geq \frac{1}{2}$
Rq : $x \notin L \Rightarrow \forall y, \neg R(x,y)$

Définition 3.16 (Machine de Turing Probabiliste)
Une MTP est une MTND dont les choix sont binaires et pour toute entrée, les calculs possibles de la machine sont de la même longueur. La machine possède une bande spéciale ou il y a une suite de bits aléatoires.
Pr(M accepte x) = $\frac{\sharp\{\text{nombre de branches acceptantes}\}}{\sharp\{\text{nombre total de branches}\}}$

Si $L \in RP \Leftrightarrow \exists MTP$ en temps poly
$\forall x$, si$x \in L \Rightarrow$Pr(M se trompe sur x) $\leq 1/2$
Si $x \notin L \Rightarrow$ Pr(M se trompe sur x) $= 0$
Comment peut on diminuer cette erreur ?
On définit une MTP M^*
Sur x , M^* simule 100 calculs de M avec des tirages aléatoires indépendants et accepte si et seulement si au moins un calcul de M accepte.
$x \in L \Rightarrow$ Pr(M^* refuse x) $\leq (1/2)^{100}$
$x \notin L \Rightarrow \Pr(M^* accepte x) = 0$.

Définition 3.17 (co-RP)
$x \in L \Rightarrow$ Pr(erreur) $= 0$
$x \notin L \Rightarrow$ Pr(erreur) $\leq 1/2$

Définition 3.18 (ZPP)
ZPP = RP $\cap$ co-RP

Soit L dans ZPP. Soit M une MTP qui accepte L selon RP, M' qui accepte L selon co-RP.
On peut définir une machine M^* qui simule de facon alternée M et M' et accepte quand M accepte la première fois et refuse quand M' refuse la première fois.
M^* donne toujours la réponse correcte. Par contre le temps n'est pas forcément polynomial. Ce qu'on peut garantir c'est que l'espérance du temps de M^* est $\mathcal{O}(\text{temps}(M) + \text{temps}(M'))$

RP, co-RP, BPP sont des classes Mont-Carlo $\begin{cases}\text{erreur existe}\\ \text{temps garanti}\end{cases}$

ZPP est une classe LAS-VEGAS $\begin{cases}\text{zero erreur}\\ \text{temps en espérance}\end{cases}$

Théorème 3.19 (PRIMALITE)
PRIMALITE $\in$ *ZPP*

Démonstration 3.19
PRIMALITE $= \{m, m\text{premier}\}$.
On va prouver que PRIMALITE $\in$ co-RP.

Définition 3.19 (Polynomial Probability)
$PP = \{L \subset \{0,1\}^*, \exists MTP, x\ inL \Leftrightarrow Pr(M\ accepte\ x) > 1/2\}$

Définition 3.20 (Bounded Polynomial Probability)
$BPP = \{L \subseteq \{0,1\}, \exists MTP$ *M en temps polynomial telle que*
$$\forall x, \begin{cases} x \in L \Rightarrow Pr(M\ accepte\ x) \geq 3/4(1/2+\epsilon) \\ x \notin L \Rightarrow Pr() \leq 1/4(1/2+\epsilon) \end{cases}$$

3.1 Borne de Chernoff

Théorème 3.20 (Borne de Chernoof)
Soient $X_1, X_2, \cdots, X_n$ *des variables indépendantes, avec* $Pr(X_i = 1) = p_i$
On définit $X = \sum X_i$.
$\mu = E[X] = \sum p_i$
Alors $Pr[X > (1+\delta)\mu] \leq (\frac{e^\delta}{(1+\delta)^{(1+\delta)}})^\mu$ *pour* $\delta > 0$
$Pr[X < (1-\delta)\mu] \leq (\frac{e^{-\delta}}{(1-\delta)^{1-\delta}})^\mu$ *pour* $0 < \delta < 1$.
Corollaire

- $(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}})^\mu \leq e^{\frac{-\delta^2\mu}{2}}$
- $(\frac{e^\delta}{(1+\delta)^{1+\delta}})^\mu \leq \begin{cases} e^{-\frac{\delta^2\mu}{4}} & si\ \delta < 2e-1 \\ 2^{-(1+\delta)\mu} & si\ \delta \geq 2e-1 \end{cases}$

Démonstration 3.20
$\Pr[X > (1+\delta)\mu] = \Pr[(1+\delta)^X > (1+\delta)^{(1+\delta)\mu}]$
$\leq \frac{E[(1+\delta)^X]}{(1+\delta)^{(1+\delta)\mu}}$ (Inégalité de Markov)
Il suffit de montrer que $E[(1+\delta)^X] < e^{\delta\mu}$
$E[(1+\delta)^X] = E[(1+\delta)^{\sum X_i}] = E[\prod(1+\delta)^{X_i}]$

$= \prod E[(1+\delta)^{X_i}]$
$E[(1+\delta)^X] = p_i(1+\delta) + (1-\delta) = 1 + \delta p_i \leq e^{\delta p_i}$
Donc $E[(1+\delta)^X] \leq \prod e^{\delta p_i} = e^{\delta \sum p_i} = e^{\delta\mu}$
Exemple :
BPP $= \{L \subset \{0,1\}^*, \exists MTP$ M en temps polynomial telle que $\forall x$

$\begin{cases} x \in L \Rightarrow Pr[\text{M accepte x}] \geq 3/4 \\ x \notin L \Rightarrow Pr[\text{M accepte x}] \leq 1/4 \end{cases}$ Comment diminuer l'erreur ?

M' sur x fait marcher M sur x k fois et M' accepte ssi M accepte au moins $k/2$ fois.
Pb : Erreur de M' ?
Supposons que $x \in L$

$X_i = \begin{cases} 1 \text{ si le deuxième calcul de M sur x accepte} \\ 0 \text{ sinon} \end{cases}$

$E[X_i] = 3/4$
$X = \sum X_i, E[X] = 3/4k$
$\Pr[M'$ se trompe sur x$] = \Pr[X < k/2] = \Pr[X < (1 - 1/3) * 3/4 * k]$
$\leq e^{-\frac{k}{9}}$

3.2 Test de primalité : SOLOVAY-STRASSEN (77)

Définition 3.21 (PREMIER)
PREMIER $= \{n, n \text{ est premier}\}$

Théorème 3.21 (PREMIER)
PREMIER $\in$ *ZPP* $=$ *RP* $\cap$ *co-RP*

Démonstration 3.21
SS prouve que PREMIER est dans co-RP. L'autre côté est de GOLDWASSER-KILLIAN et ADELMAN-HUANG (84).

Idée de test mais FAUX

- On considère $Z_n = \{0, 1 \cdots n-1\} = Z/nZ$
- On considère le théorème de Fermat $a^{n-1} \equiv 1[n]$
- Si n premier alors ca marche pour tout a
- FAUX : On voudrait que si n composé (non premier) alors pour au moins la moitié des éléments de $Z_n, a^{n-1} \not\equiv 1[n]$

Le problème ici sont les entiers de Carmichael.

Définition 3.22 (entier Carmichael)
n est Carmichael si :

- *n composé*
- $\forall a, a^{n-1} \equiv 1[n]$

Théorème 3.22 (Carmichael (1991))
Il existe un nombre infini de nombres de Carmichael (le premier est 561)

Démonstration 3.22
Non vu

Définition 3.23 (Z_n^*)
$Z_n^* = \{a \in Z, pgcd(a,n) = 1\}$
Si p premier $Z_p^* = \{1?2 \cdots p-1\}$

- Cout des opérations arithmétiques

Operation	Cout en Z	Cout en Z_n
$u+v$	$\log u + \log v$	$\mathcal{O}(\log n)$
uv	$\log u * \log v$	$\mathcal{O}(\log^2 n)$
$u/v(u = qv + r)$	$(\log u + 1) \log v$	$\mathcal{O}(\log^2 n)$
u^m	$\log m(\log u + \log m)^2$	$\mathcal{O}(\log^3 n)$
u^{-1}		$\mathcal{O}(\log^2 n)$

Cout du PGCD via l'algorithme d'Euclide

Théorème 3.23 (Complexité Euclide)
Cout d'Euclide est $\mathcal{O}(\log u_0 \log u_1)$

Démonstration 3.23
A chaque étape le nombre est au moins divisé par 2.

Calcul de l'inverse
Par bezout on a $\alpha a + \beta n =$pgcd$(a,n) = 1$ donc $\alpha = a^{-1}$

- Résiduosité quadratique, symbole de Legendre, symbole de Jacobi

Définition 3.24 (Résidu quadratique)
Soit $p > 2$ un premier. Un entier a est résidu quadratique (mod p) si $x^2 = a[p]$

Définition 3.25 (Symbole de Legendre)
Symbole de Legendre ($a \in Z_p^*$)
$(\frac{a}{p}) = \begin{cases} \text{1 si a résidu quadratique} \\ \text{-1 si a résidu non-quadratique.} \end{cases}$

Théorème 3.24 (Legendre)
$(\frac{a}{p}) = a^{\frac{p-1}{2}}$

Démonstration 3.24

- $(\frac{a}{p}) = 1 \Rightarrow \exists b, b^2 = a[p] \Rightarrow a^{\frac{p-1}{2}} = (b^2)^{\frac{p-1}{2}} = b^{p-1} \equiv 1[p]$(Fermat)
- $a^{\frac{p-1}{2}} = 1, Z_p^*$ est un groupe cycliquedonc il existe un générateur
  Exemple : $p = 7, Z_7^* = \{1, 2, \cdots, 6\}$
  3 est un générateur parce que $3^1 = 3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5, 3^6 = 1$

Définition 3.26 ()
g est générateur si l'ordre de g est $p-1$.
ordre(g) = plus petit entier k tel que $a^k \equiv 1[p]$

Soit g un générateur de Z_p^* donc $a = g^k$ donc $g^{k\frac{p-1}{2}} \equiv 1[p]$
$\Rightarrow p-1|k/2(p-1) \Rightarrow k = 2l \Rightarrow a = (g^l)^2$
Soit g un générateur de Z_p^*. Alors les puissances paires de g sont des restes quadratiques et les puissances impaires des restes non quadratiques.

Théorème 3.25 ()

- $(1) \cdots (\frac{a}{p}) = (\frac{a[p]}{p})$
- $(2) \cdots (\frac{ab}{p}) = (\frac{a}{p})(\frac{b}{p})$
- $(\frac{2}{p}) = (-1)^{\frac{p^2-1}{8}}$,
  $$(\frac{2}{p}) = \begin{cases} 1 \ si \ p \equiv +-1[8] \\ -1 \ si \ p \equiv +-3[8] \end{cases}$$
- $(\frac{p}{q})(\frac{q}{p}) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}$,
  $$(\frac{q}{p}) = \begin{cases} (\frac{p}{q}) \ si \ p \equiv 1[4] \ ou \ q \equiv 1[4] \\ -(\frac{p}{q}) \ si \ p \equiv q \equiv 3[4] \end{cases}$$

Démonstration 3.25
RAS

Définition 3.27 (Jacobi)
Symbole de Jacobi
Soit $b = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ *impair.*
Alors $(\frac{a}{b}) = (\frac{a}{p_1})^{\alpha_1}(\frac{a}{p_2})^{\alpha_2} \cdots (\frac{a}{p_k})^{\alpha_k}$
ATTENTION : $(\frac{a}{b}) = 1$ ne veut pas dire que a est un résidu quadratique modulo b.
$(\frac{2}{3}) = -1, (\frac{2}{5}) = -1 \Rightarrow (\frac{2}{15}) = +1$
Les propriétés du symbole de Legendre restent vraies pour le symbole de Jacobi.

Théorème 3.26 (Jacobi)
Le symbole de Jacobi se calcule en temps $\mathcal{O}(\log n)$

Démonstration 3.26
$(\frac{51}{97}) = (\frac{97}{51}) = (\frac{46}{51}) = (\frac{2}{51})(\frac{23}{51}) = -(\frac{23}{51}) = (\frac{51}{23}) = (\frac{5}{23}) = (\frac{23}{5}) = (\frac{3}{5}) = (\frac{5}{3}) = (\frac{2}{3}) = -1$
En vérité le calcul du symbole de Jacobi est presque la meme chose qu'Euclide avec la seule différence qu'on a pas $pgcd(2a, 2b+1) = pgcd(a, 2b+1)$

$SS(n)$

- Si $n = 2$ dire n *premier*

- Si $n = 1$ ou $n = 2k \neq 2$ dire *n composé*
- Si $n = a^b$ on essaie toutes les valeurs $b = 2, 3, \cdots, \log n$. Pour b fixé on recherche $a, n = a^b$ (Dichotomie pour a entre 1 et n)
  Actuellement à cette étape on a $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ ou $k \geq 2$ et $\exists i$, α_i impair (sinon on aurait un carré.)
- On engendre un entier aléatoire a dans Z_n. Si $(n \wedge a) > 1$ alors dire *n composé*
- Si $(\frac{a}{n}) = a^{\frac{n-1}{2}}[n]$ dire *n premier*
- Si $(\frac{a}{n}) \neq a^{\frac{n-1}{2}}$ dire *n composé*

Théorème 3.27 (SS)
SS(n) est un algorithme co-RP pour PREMIER

Démonstration 3.27

- $SS(n)$ marche en temps polynomial.
- Si n premier, $SS(n)$ ne se trompe pas
- On va montrer que si n est composé alors $\Pr(SS(n)$ se trompe et dise *n premier*$) \leq 1/2$

Après les premières étapes de l'algorithme on se retrouve dans Z_n^*.
On définit $G = \{a \in Z_n^*, (\frac{a}{n}) = a^{\frac{n-1}{2}}[n]\}$.
Pr[erreur]= $\frac{|G|}{|Z_n^*|}$
On va montrer que G est un sous groupe propre de Z_n^* c'est à dire différent de Z_n.
Alors on aura $|G|||Z_n^*|$ et donc $|G| \leq 1/2 * |Z_n^*|$.
Il est facile de montrer que c'est un sous-groupe mais il faut montrer qu'il est propre. On doit montrer qu $\exists a \in Z_n^*, (\frac{a}{n}) \neq a^{\frac{n-1}{2}}[n]$. Soit n sous forme
$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ avec un α_i impair et ≥ 2. Fixons i tel que α_i impair.
Soit g_i un générateur de $Z_{p_i}^*$. Soit a tel que $a \equiv 1[p_j], \forall j \neq i$
$a \equiv g_i[p_i]$ (Théorème du reste chinois)
$(\frac{a}{n}) = (\frac{a}{p_1})^{\alpha_1} (\frac{a}{p_2})^{\alpha_2} \cdots (\frac{a}{p_k})^{\alpha_k} = (\frac{a}{p_i})^{\alpha_i}$ car $\forall j \neq i, (\frac{a}{p_j}) = (\frac{1}{p_j}) = 1$
$= (\frac{g_i}{p_i})^{\alpha_i} = (-1)^{\alpha_i}$ (générateur puissance impaire de lui-meme)
$= -1$ car α_i impair.
On montre que $a^{\frac{n-1}{2}} = 1[n]$ par l'absurde
Supposons que $a^{\frac{n-1}{2}} = -1[n]$
$\Rightarrow n | a^{\frac{n-1}{2}} + 1$
Soit $j \neq i$ (existe car $k \geq 2$)
$p_j | a^{\frac{n-1}{2}} + 1$
$a^{\frac{n-1}{2}} = -1[p_j]$ ABSURDE car $a = 1[p_j]$ (cf plus haut)
Donc deux résidus différeents donc sous groupe propre.

3.3 Algorithme RNC pour COUPLAGE MAXIMAL

3.3.1 Lemme de Schwartz ou Zippel (77)

Théorème 3.28 (Lemme de Schwartz)
Soit F un corps et $P(x_1, \cdots, x_n)$ un polynome à n variables de degré d (degré = degré global c'est à dire somme des degrés des différentes variables dans un monome.) Soit $\Omega \subseteq F, |\Omega| = N$. Alors le nombre des racines de P en Ω^n est $\leq dN^{n-1}$.

Démonstration 3.28
Récurrence sur n.

- Si $n = 1$ OK
- Si $n > 1$ alors soit k le degré maximal de x_1 en P.
  $P(x_1, x_2, \cdots, x_n) = g_0(x_1, x_2, \cdots x_{n-1}) + g_1(x_1, x_2, \cdots, x_{n-1})x_n + \cdots + g_k(x_0, x_1, \cdots, x_{n-1})$ ou $g_k(x_1, \cdots x_{n-1}) \neq 0$
  $\{(a_1, a_2, \cdots, a_n) \in \Omega^n, P(a_1, a_2, \cdots, a_n) = 0\} =$
  $\{(a_1, \cdots a_n) \in \Omega^n, P(a_1, \cdots a_n) = 0$ et $g_k(a_1, \cdots a_{n-1}) \neq 0\} \cup \{\cdots\cdots = 0\}$
  $= A \cup B$
  $\forall (a_1, \cdots, a_{n-1})$ tel que $g_k(a_1, \cdots, a_{n-1}) \neq 0, P(a_1, a_2, \cdots, a_{n-1}, x_n)$ est un polynome à 1 variable de degré k. Donc le nombre de ses racines est au plus k.
  $\Rightarrow |A| \leq |\Omega|^{n-1}k$
  $deg(g_k) \leq d - k \Rightarrow |B| \leq |\Omega|^{n-1}(d - k)$

3.3.2 Corollaire

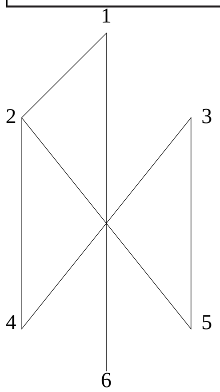
Théorème 3.29 (Corollaire)
F corps, $P(x_1, \cdots, x_n)$ polynome de degré d. $P \neq 0$
$\Omega = \{0, +-1, +-2, \cdots, +-d\}$
Soient $a_1, a_2, \cdots, a_n$ des tirages aléatoires indépendants de Ω, alors $\Pr[P(a_1, \cdots a_n)] < 1/2$

Démonstration 3.29
Evident

3.4 Algorithme RNC pour décider si un graphe contient un couplage parfait

3.4.1 Introduction

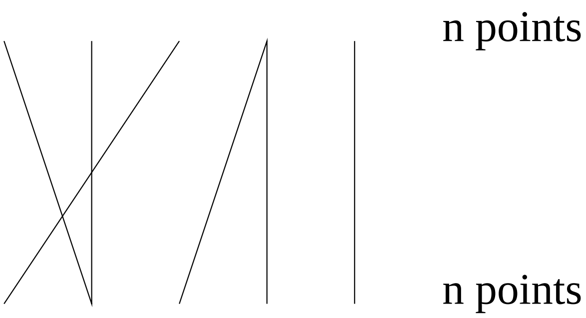
Définition 3.28 (Couplage)
Soit $G = (V, E)$ un graphe. $E' \subseteq E$ est un couplage si
$\forall e_1 \neq e_2 \in E', e_1 \cap e_2 = \emptyset$
Un couplage E' est dit parfait si $2|E'| = |V|$



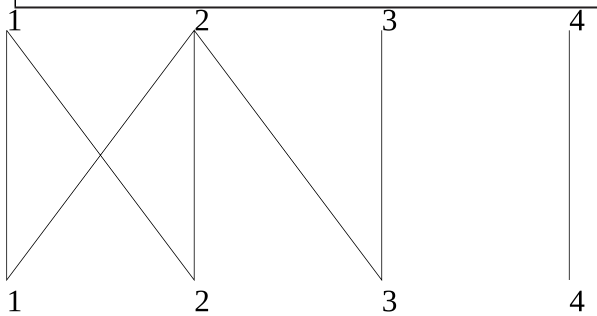
$\{\{1,2\},\{3,5\}\}$ est un couplage mais pas parfait.
$\{\{2,4\},\{1,6\},\{3,5\}\}$ est un couplage parfait.

Théorème 3.30 (Couplage parfait)
CP$\in \mathcal{P}$

Démonstration 3.30
On considère d'abord le cas particulier des graphes bi-partis



Définition 3.29 (Matrice de Tutte)
Soit $G = (V_1, V_2, E)$ un graphe biparti avec $|V_1| = |V_2| = n$. La matrice de TUTTE est définie comme $M = (m_{ij})_{1 \leq i,j \leq n}$ ou
$m_{ij} = \begin{cases} x_{ij} \text{ si } (i,j) \in E \\ 0 \text{ sinon} \end{cases}$



$$M_G = \begin{pmatrix} x_{11} & x_{12} & 0 & 0 \\ x_{21} & x_{22} & x_{23} & 0 \\ 0 & 0 & x_{33} & 0 \\ 0 & 0 & 0 & x_{44} \end{pmatrix}$$

Théorème 3.31 ()
Il existe un couplage parfait en G ssi $\det(G) \neq 0$

Démonstration 3.31
Par exemple $\det(M_G) = + - x_{11}x_{22}x_{33}x_{44} + -x_{12}x_{21}x_{33}x_{44}$
Un couplage parfait définit naturellement la permutation $\Pi \in S_n$
Rappel : $\det = \sum_{\Pi \in S_n} \epsilon(\Pi) \prod x_{i,\Pi(i)}$
Pour un Π défini par un couplage le produit est non nul.

3.4.2 Algorithme RNC pour couplage parfait

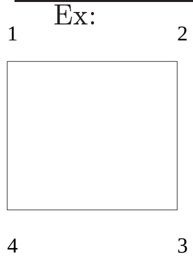
- A partir de G on construit M_G
- On tire aléatoirement les valeurs $\overline{a}$ de l'ensemble $\{1, 2, \cdots, 2n\}$ pour les x_{ij}
- Si $\det(M_G(\overline{a})) = 0$ dire *pas de couplage*
  sinon dire *il existe couplage*

Cas des graphes généraux pour CP

Définition 3.30 (TUTTE)
Soit $G = (V, E), |V| = 2n, V = \{v_1, \cdots, v_{2n}\}$
Lamatrice de TUTTE pour G :
$$M_G = \begin{cases} x_{ij} \text{ si } i < j \text{ et } \{v_i, v_j\} \in E \\ -x_{ji} \text{ si } i > j \text{ et } \{v_i, v_j\} \in E \\ 0 \text{ sinon} \end{cases}$$

Théorème 3.32 ()
Il existe un couplage parfait en $G \Leftrightarrow \det M_G \neq 0$

Ex:



$$\begin{pmatrix} 0 & x_{12} & x_{13} & 0 \\ -x_{12} & 0 & 0 & x_{24} \\ -x_{12} & 0 & 0 & x_{34} \\ 0 & -x_{24} & -x_{34} & 0 \end{pmatrix} = x_{12}^2 x_{34}^2 + x_{13}^2 x_{24}^2 + 2x_{12}x_{24}x_{13}x_{34}$$

Dans certains cas des termes peuvent s'éliminer entre eux donc on n'a pas en général toutes les solutions de représentées mais on est assuré d'en avoir au moins une.

Démonstration 3.32
$\Leftarrow$ Soit le couplage parfait :
$v_{i_1} \longleftrightarrow v_{j_1}$
$v_{i_2} \longleftrightarrow v_{j_2}$
$\cdots\ v_{i_{\frac{n}{2}}} \longleftrightarrow v_{j_{\frac{n}{2}}}$
Quand on considere la permutation Π telle que :
$\Pi(i_k) = j_k, \Pi(j_{i_k}) = i_{i_k}$
La contribution de Π au déterminant est :
$+ - x_{i_1, j_1}^2 * \cdots * x_{i_{\frac{n}{2}}, j_{\frac{n}{2}}}$ et il n'y a pas de moyen d'annuler ce terme.

$\Rightarrow det(M) = \sum_{\sigma \text{permutation}} (-1)^{\text{signe}(\sigma)} \text{valeur}(\sigma)$
signe(σ) = nombre d'inversions de σ
valeur(σ) = $\prod_{i=1}^{n} m_{i,\sigma(i)}$
On regarde la décomposition des permutations en cycle..
Ex: $\sigma = (2, 3, 1, 5, 4)$
(cycle 1 2 3 et cycle 4 5)

Si on a un cycle pair (avec un nombre pair d'élements) sa contribution est non nulle et on peut faire un couplage. Donc il faut trouver une permutation qui se décompose en cycles pairs.

Définition 3.31 (Permutation paire)
Une permutation est paire si dans sa décomposition elle ne comporte que des cycles de longueur paire. Sinon elle est impaire.

Idée de la preuve : On va montrer que la contribution totale des permutations impaires au déterminant est nulle. Comme le déterminant est non nul, il existera donc une permutation paire de valeur non nulle. De cette permutation on peut construire un couplage parfait..

Théorème 3.33 ()
Soit $\Pi_1 = (C_1, C_2, \cdots, C_k)$ et $\Pi_2 = (C_1^{-1}, C_2, \cdots, c_k)$
Alors $signe(\Pi_1) = signe(\Pi_2)$

Démonstration 3.33
Pour toute permutation σ, signe$(\sigma) =$ signe(σ^{-1})
Car signe(σ)signe$(\sigma^{-1}) =$ signe$(\sigma \circ \sigma^{-1}) = 1$

$\Pi_1 = \Pi_{C_1} \circ \Pi_{C_2} \circ \cdots \circ \Pi_{C_k}$ ou Π_{C_j} est le cycle C_j et l'identité ailleurs.

$$\begin{aligned}
\text{signe}(\Pi_1) &= \text{signe}(\Pi_{C_1})\text{signe}(\Pi_{C_2}) \cdots \text{signe}(\Pi_{C_k}) \\
&= \text{signe}(\Pi_{C_1}^{-1}) \cdots \\
&= \text{signe}(\Pi_{C_1^{-1}}) \cdots \\
&= \text{signe}(\Pi_2)
\end{aligned}$$

Théorème 3.34 ()
Π_1 et Π_2 comm en lemme 1.
Alors $valeur(\Pi_2) = (-1)^{|C_1|} valeur(\Pi_1)$

Démonstration 3.34
A chaque arete $x_{i,j}$ dans C_1 correspond par définition l'arête $-x_{i,j}$ de C_1^{-1}.

Théorème 3.35 ()
La contribution totale des permutations impaires au déterminant est nulle

Démonstration 3.35
On va définir une fonction f sur les permutations impaires. Soit Π une permutationimpaire et soit i le plus petit indice qui se trouve en Π dans un cycle impair C. Alors par définition $f(\Pi) = \Pi'$ ou Π' contient C et partout ailleurs est identique a Π.
Clairement, $f(\Pi') = \Pi$ ie c'est un couplage parfait sur les permutations impaires et de plus, la contribution de Π et de $f(\Pi)$ au déterminant est nulle a cause des deux théoremes précédents.

Théorème 3.36 (Couplage parfait)
COUPLAGE PARFAIT $\in$ *RNC*^{2}

Démonstration 3.36
cf graphe bi partite

Théorème 3.37 (Lemme d'isolation (VAZIRANI et id 1985))
Soit $S_1, \cdots, S_t \subseteq S = \{x_1, x_2, \cdots, x_m\}$. *Soient* $w_1, w_2, \cdots, w_m$ *les poids aléatoires pris uniformément de l'ensemble* $\{1, 2, \cdots, n\}$ *ou* $poids(x_i) = w_i$.
$poids(S_j) = \sum_{x_i \in S_j} w_i$
Alors
$Pr[$*Parmi les* $S_1, S_2, \cdots, S_i$*l'ensemble de poids minimal n'est pas unique (E)*$]$
$\leq \frac{m}{n}$

Démonstration 3.37
Intro

Définition 3.32 (singulier)
On dit que x_i *est singulier si il existe* $S_j \neq S_k$ *de poids minimal tels que* $x_i \in S_j$ *et* $x_j \notin S_k$

Donc :

$$\begin{aligned} \Pr(E) &= \Pr(\exists i, x_i \text{ singulier}) \\ &\leq \sum_{i=1}^{m} \Pr(x_i \text{singulier}) \end{aligned}$$

On va montrer que pour un i fixé $\Pr(x_i \text{ est singulier}) \leq \frac{1}{n}$
Fixons $w_1, \cdots w_{i-1}, w_{i+1}, \cdots, w_m$.
On sépare les S_j en 2, ceux pour lesquels $x_i \notin S_j$ et les autres. Pour les premiers on obtient un poids partiel P. Pour les autres on a des poids pour chaque S_k et on veut obtenir le meme poids P il faut donc prendre le plus

petit de ces poids et lui ajouter la valeur ad hoc pour atteindre P. Donc une seule possibilité.
Alors il existe au plus une seule valeur w_i qui va rendre x_i singulier

Algorithme probabiliste pour construire un couplage parfait
$G = (V, E), E = \{e_1 \cdots, e_m\}$
On tire des poids aléatoires pour les aretes de l'ensemble $\{1, 2, \cdots, 2m\}$ alors Pr[Couplage parfait de poids minimal est unique] $\geq \frac{1}{2}$ d'après le lemme d'isolation.
Supposons qu'on a de la chance et que le couplage parfait de poids minimal est unique. Comment décider si une arête fait partie du couplage parfait de poids minimal ?
On reprend la matrice de TUTTE M et on remplace la variabl x_{ij} par $2^{w_{ij}}$
$v_i \underset{\underset{-x_{ij}}{\longleftarrow}}{\overset{x_{ij}}{\longrightarrow}} v_j$ si $v_i \longleftrightarrow v_j$ ou w_{ij} est le poids de l'arête $v_i \longleftrightarrow v_j$
On peut calculer $det(M)$.
On regarde la contribution des permutations a $det(M)$.
Soit W le poids du CP de poids minimal. Alors ka contribution du CP de poids minimal est $+-2^{2W}$
Prenons un autre couplage parfait . Sa contribution est $2^{2W'}$ ou $W' >>$. La contribution totale des permutations impaires esst 0. La contribution d'une permutation paire est $\geq 2^W 2^{W+1} \geq 22^{2W}$ car dans une permutation paire on peut réunir les couplages parfait par deux le couplage et son complémentaire et donc seul un des deux peut etre de poids W l'autre est forcement de poids $> W$

Donc $\mathbf{det}(M) = 2^{2W} *$ impair
On peut donc déterminer W en calculant le déterminant et en cherchant la puissance de 2.

Algorithme parallele pour trouver un couplage parfait dans un graphe qui en contient au moins 1

- On fait des tirages aléatoires w_{ij} pour l'arête $v_i \longleftrightarrow v_j$
- On calcule W à partir de déterminant de matrice de TUTTE instantcié avec $x_{ij} = 2^{w_{ij}}$.
  Pour toute arete $v_i \longleftrightarrow v_j$ en parallele enlève $v_i \longleftrightarrow v_j$ de G et calcule la meme chose (W_{ij}^*) en $G = \{v_i, v_j\}$
  L'arête $v_i \longleftrightarrow v_j$ se colorie rouge si $w_{ij} = W - W_{ij}$
  On vérifie si les aretes rouges forment un couplage parfait sinon on recommence.

4 Systèmes de preuves interactifs

4.1 Introduction et définitions

Définition 4.1 (Protocole interactif)
Soit V (vérifieur) une MTP en temps polynomial.
Soit P (Prouveur) une MT sans restriction.
Un protocole interactif sur un mot $x \in \{0,1\}^*$ (qui est l'entrée commune pour les deux participants) entre V et P est une suite de messages :
$V \to P \quad v_1 = V(x, r_1)$ (r_1 : bits aléatoires)
$P \to V \quad p_1 = P(x, v_1)$
$V \to P \quad v_2 = V(x, p_1, r_2)$
$\cdots$
$P \to V \quad p_k$
V : calcule $V^p(x) = V(x, p_1, p_2, \cdots, p_k, r_1, r_2, \cdots, r_k) \in \{1, 0\}$

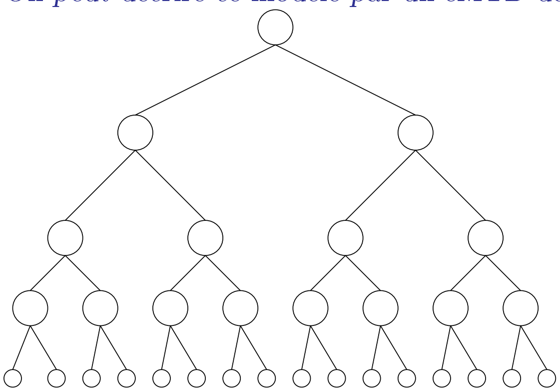
Définition 4.2 (Systeme de preuves interactifs)
$L \subseteq \{0,1\}^*$ possède un système de preuves interactifs s'il existe un vérifieur V tel que :
$\forall x \in \{0,1\}^*, x \in L \Rightarrow \exists P prouveur, Pr[V^p(x) = 1] = 1$
$x \notin L \Rightarrow \forall P prouveur, Pr[V^p(x) = 1] \leq 1/2$

Définition 4.3 (IP)
$IP = \{L \subseteq \{0,1\}^*, L$ *possède une preuve interactive*

Remarque 8

- $\mathcal{NP} \subseteq IP$
- $co - RP \subseteq IP$

Définition 4.4 (BABAI 83 : Jeux Arthur- Merlin)
Un jeu Arthur Merlin est un système de preuves interactifs où les messages du vérifieur sont exactement ses bits aléatoires.
$v_1 = r_1, \cdots, v_i = r_i$
On peut décrire ce modèle par un eMTD de la facon suivante



Définition 4.5 (Graphe non-isomorphisme (GNI))

- *Entrée :* $G_0 = (V_0, E_0), G_1 = (V_1, E_1)$
- *Question : Est-ce que* G_1 *n'est pas isomorphe à* G_2 *?*

Théorème 4.1 (GNI)
$GNI \in IP$

Protocole :
V : Engendre un bit aléatoire b, engendre une copie aléatoire H de G_b. (ie H permutation aléatoire des sommets de G_b sur V)
$V \to P$: H
$P \to V$: $c \in \{0, 1\}$
V accepte ssi $c = b$.

Démonstration 4.1

- $(G_0, G_1) \in$ GNI. $G_0 \neq G_1$
  $H \cong G_b$
  $H \ncong G_{1-b}$
  Prouveur peut déterminer c tel que :
  $H \cong G_c$ et clairement $c = b$.

- $(G_0, G_1) \notin GNI, G_0 \cong G_1$
  $H \cong G_b \cong G_{1-b}$
  P ne peut pas déterminer si H a été engendré de G_0 ou de G_1. Donc

avec probabilité 1/2 c sera différent de b.

4.2 Exemple

PERM = $\{(A, a)$: **A matrice d'entiers, perm(A)=a**$\}$

Théorème 4.2 (NISAN)
PERM $\in$ *IP*

Démonstration 4.2
Soit p un premier assez grand (En fait plus grand que le résultant). On fait tout le protocole en $Z_p = \{0, 1, \cdots, p-1\}$
Du point de vue de V, il y a au cours du protocole à chaque instant une liste d'énoncés $\mathcal{L} = \{(B_1, b_1), (B_2, b_2), \cdots, (B_k, b_k)\}$
ou intuitivement (B_i, b_i) veut dire perm(B_i) = b_i.
dim(B_i) = dim(B_j) $\forall i, j$
Initialement $\mathcal{L} < (A, a) >$
Deux types de pas (pas = échange de message entre V et P)

- Expansion : $\mathcal{L} =< (B, b) >\longrightarrow \mathcal{L}' =< (B_1, b_1) \cdots (B_k, b_k) >$ ou dim(B_i) = dim(B) - 1
- Réduction : $\mathcal{L} =< (B_1, b_1), \cdots (B_k, b_k) >\longrightarrow \mathcal{L}' =< (B_1, b_1) \cdots (B_{k-2}, b_{k-2}), (C, c) >$

A la fin il y a :
$\mathcal{L} =< (B, b) >$ ou dim$(B) = 2$
Alors V calcule perm(B) et accepte si perm(B) = b

- Description de Expansion :
  Développement par rapport a la première ligne
  $P \longrightarrow V, \mathcal{L}' =< (A_1, b_1), \cdots, (A_n, b_n) >$
  ou A_i est le sous-permanent.
  V calcule $\sum_{i=1}^{n} a_{1i} b_i$
  Si $\sum = a$ il continue le protocole sinon il l'arrete et refuse.

  - Si perm(A) = a alors P peut renvoyer b_j = perm(A_j) et le vérifieur va continuer le protocole.
  - Si perm(A) $\neq 0$ alors pour que V n'arrete pas le protocole en le refusant il faut qu'il existe un j tel que $b_j \neq$ perm(A_j)
- Description de réduction :
  $\mathcal{L} =< (B, b), (C, c) >\longrightarrow \mathcal{L}' =< (D, d) >$
  Soit la droite passant par B et C : $xB + (1-x)C = D(x)$

$P \longrightarrow V \;\; g(x) = g_0 + g_1 x + \cdots + g_k x^k$
V teste que g(0)=c et g(1)=b.
Si ce n' est pas vrai il arrete le protocole et refuse.
Sinon V engendre $r \in \{0, 1, \cdots p-1\}$ aléatoire et $\mathcal{L}' =< (D(r), g(r)) >$

- Si $\mathcal{L}$ contient 2 énoncés vrais alors P (prouveur correct) renvoie $g(x) = \text{perm}(D(x))$ et $D(0) = C$
  $g(0) = \text{perm}(D(0)) = \text{perm}(C) = c$
  De plus $\text{perm}(D(r)) = g(r)$
- Si $\mathcal{L}$ contient un énoncé faux perm(B) $\neq$ b.
  P est obligé de renvoyer $g(x) \neq \text{perm}(D(x))$ car sinon on découvre tout de suite que $g(1) \neq b$
  (car $\text{perm}(D(1)) = \text{perm(B)} \neq b$)

Comme deux polynomes distincts de degré k ne peuvent pas avoir la meme valeur en plus de k-1 points.
$\Pr[g(r) = \text{perm}(D(r))] \leq \frac{k-1}{p} \sim \frac{n}{2^{n^2}}$
Donc avec probabilite $1 - \frac{n}{2^{n^2}}$ $\mathcal{L}$ aura un énoncé faux.
A la fin la probabilité d'erreur est $\leq n^2 * \frac{n}{2^{n^2}}$

Théorème 4.3 (Toda)
PH $\subseteq$ PERM
PH : polynomial hierarchy

Démonstration 4.3
Non vu

Théorème 4.4 (Corollaire)
co-$\mathcal{NP} \subseteq$ IP

Démonstration 4.4
Non vu

Théorème 4.5 ()
Il existe A oracle tel que $co - \mathcal{NP}^A \not\subset IP^A$
(Inclusion stricte)

Démonstration 4.5
Idée pour QBF : $\exists x_1 \forall x_2 \exists x_3 (x_1 \vee \overline{x_2}) \wedge (\overline{x_3} \vee x_2)$ (1)
$\sum_{x_1=0}^{1} \prod_{x_2=0}^{1} \sum_{x_3=0}^{1} (x_1 + (1 - x_2))((1 - x_3) + x_2)$

Si $\neq 0$ alors (1) faux sinonon enlève la somme et on obtient un polynome en x1.