Réductions LOGSPACE

Q 2.

> Mq toute MT déterministe qui calcule une réduction en espace logarithmique s'arrête après un nombre d'étapes polynomial.

Si une fonction est logarithmique en espace, le nombre d'étapes est polynomial.

Attention : LOGSPACE n'est pas clos de k rubans à 1 ruban (il y a un passage au carré $\longrightarrow$ on a du $log^2(n)$), donc travailler sur k rubans ou 1 ruban n'est pas équivalent.

Machine à k rubans.

Nb de configurations :

$$N = |Q| \times (|\Sigma|^{\log n})^k \times \lceil \log(n) \rceil^k \times n$$

(états, contenus des k rubans, position des têtes de lecture, taille de l'entrée)

Or : toute exécution de la MT qui s'arrête ne peut pas passer par deux configurations identiques :

donc ce nombre est majoré par N, et est donc polynomial.

Q 3.

> Soient $h_1 = L_1 \longrightarrow L_2$ et $h_2 : L_2 \longrightarrow L_3$ deux réductions calculables en espace logarithmique par des machine déterministes.
>
> Mq la réduction $h_1 \circ h_2$ peut être calculée en espace logarithmique.

Attention : la composition de réductions en LOGSPACE n'est pas trivialement en LOGSPACE !

en effet:

A la MT qui calcule h_1, B celle qui calcule h_2.

la sortie d'une réduction en LOGSPACE n'est pas forcément logarithmique en espace !

ruban A_entrée	**x ⋯ x**
ruban A_travail	LOG
ruban A_sortie	y ⋯ y
ruban B_entrée	y ⋯ y
ruban B_travail	LOG
ruban B_sortie	z ⋯ z

entre A_entrée et B_sortie : on utilise un espace "y ⋯ y" **polynomial** (cf. Q2).

En fait : LOGSPACE est bien stable par composition, mais c'est un théorème.

Idée de la preuve :

Pour lire le i-ème caractère de A_sortie :

on réexécute A depuis le début jusqu'au i-ième caractère, qu'on écrit seul sur le ruban de sortie à chaque fois.

Détails :

- **Init** : $c = 0$
  Puis : on simule M_{h_1} mais on incrémente c si M_{h_1} écrit sur la sortie
- et quand $c = i$, on finit en écrivant ce que M_{h_1} voulait écrire donc $h_1(x)[i]$

- Enfin, $M_{h_2 \circ h_1}$: quand on a besoin d'accéder à $h_1(x)[i]$, on procède comme précédemment.

Q 4

1.

> **Lemme** :
>
> $$ATIME(f(n)) \leq SPACE(f^2(n))$$

f constructible :

Si il existe M_f calculant f en temps $O(f)$

$M \in ATIME(f(n))$:

on procède de même que pour

$$NP \subseteq PSPACE$$

$\longrightarrow$ on simule des branches par "backtracking" avec une pile de taille $f(n)$.

$\longrightarrow$ l'exploration de l'arbre prend un temps exponentiel, mais vaut $f(n)$ en espace.

```
def exploration(config_initiale, Q, delta, lambda_value, f_n):
    tableau = { q:lambda_value(q) for q in Q }
    pile = [config_initiale]
    while pile:
        config_courante = pile.pop()
        for config in delta[config_initiale]:
            pile.append(config)
            if tableau[q] == 'top':
                tableau[config_courante] = tableau[config_courante] lambda_value(config_courante) true
            elif tableau[q] == 'bot':
                tableau[config_courante] = tableau[config_courante] lambda_value(config_courante) false
            else:
                if lambda_value(config_courante) == 'or':
                    tableau[config_courante] = false
                else:
                    tableau[config_courante] = true
    return tableau[config_initiale]
```

2.

Soit M une ATM calculant en ASPACE($f(n)$).

Pour tout x, l'arbre d'exécutions contient au plus $c^{f(n)}$ configurations (pour c constant).

1. On calcule toutes les configurations de taille $f(n)$.
   $\gamma_1, \perp \# \cdots \# \gamma_m, \perp$ ($m \leq c^{f(n)}$)
   Avec $l_i \in \{1, 0, \perp\}$
   On prendra $\gamma_1 = \gamma_i(x)$

```
for i = 0 to m:
    copie gamma_i sur une seconde bande
            gamma_i \# l_i' =
                    0 si lambda(gamma_i) = or
                    1 si lambda(gamma_i) = and
                    0 si lambda(gamma_i) = bot
                    1 si lambda(gamma_i) = top
    for j=0 to m:
        f(gamma_i  -> gamma_j) then
            l_i' <- l_i' lambda(gamma_i) l_j

    Update  la liste l_j <- l_j'
    Retourner l_1
```

   Il y a au plus $|Q| \times (|\Sigma|^{f(n)})^k \times \max(n, f(n))^k$

3.

- $\exists$ un chemin de γ_0 à Accept

Pour $c = 2$:

Soit $m = 2^{f(n)}$ une borne sur la longueur des chemins.

On suppose sans perte de généralité que M calcule en temps exactement m.

$$Reach(2^0, x, y) \iff x \longrightarrow y$$
$$Reach(2^{\alpha+1}, x, y) \iff \exists z(detaille\$f(n)\$); (Reach(2^\alpha, x, z) \wedge Reach(2^\alpha, z, y))$$

On a $f(n)$ appels récursifs, et à chaque appel, on devine une configuration de taille $f(n)$.

Complexité : $O(f(n)^2)$

Pour $c = 2^\beta$:

M accepte $x \iff Reach((2^\beta)^{f(n)}, \gamma_0, Accept)$

NB :

$$AP \subseteq PSPACE$$
$$NPSPACE \subseteq AP$$

donc

$$AP = PSPACE$$

et on a aussi :

$$APSPACE = EXPTIME$$

Réductions

EX 1

Montrer que *INDEPENDENT SET* est NP-complet.

- **Entrée** : un graphe G non orienté.
- **Question** : G a-t-il un ensemble indépendant de cardinal supérieur à m ?

1. Le problème est NP : On vérife en temps polynomial q'un ensemble de sommets est une clique dans le "complémentaire du graphe" G^c (*i.e* : le graphe obtenu à partir de G en remplaçant les 1 de la matrice d'adjacence par des 0, et les 0 par des 1).

```
Algorithme :
    S = ∅

    Pour v ∈ V:
        S = S || S = S ∪ {v}
        # choix non déterministe

    Pour v∈V, v'∈V, v≠v':
        Si (v, v')∈E:
            Echec

    Si |S|<m :
        echec
```

2. Le problème est NP-complet :

Réduction de SAT à ce problème $\mathscr{P}_1$:

$$SAT \preccurlyeq \mathscr{P}_1$$

Idée de la preuve :

Pour une conjonction f de m clauses satsifiable :

$$f \stackrel{\text{def}}{=} \bigwedge_{i=1}^{m} \bigvee_{j=1}^{n} a_{i,j}$$

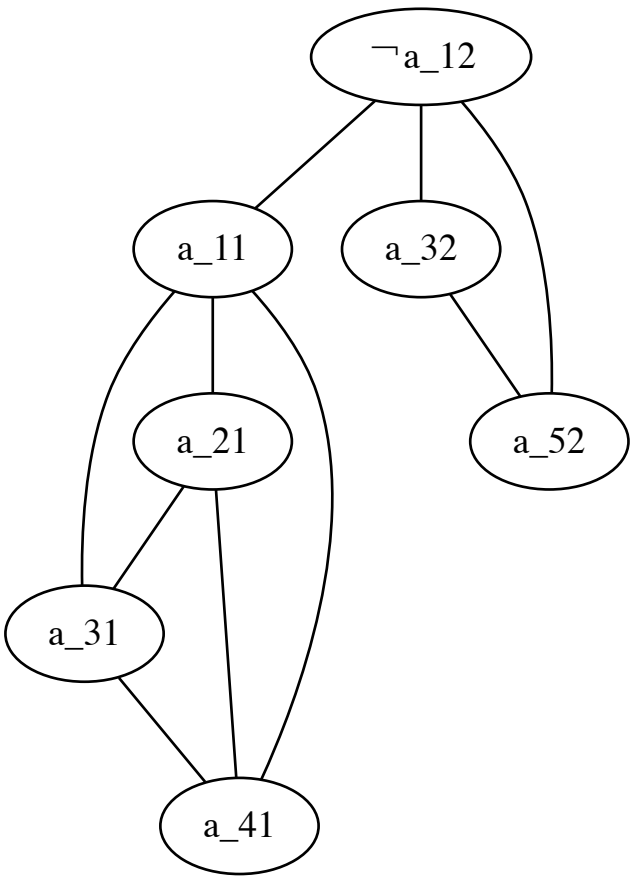
$$C_1 = a_{1,1} \vee a_{1,2} \vee a_{1,3} \vee a_{1,4}$$

$$C_2 = \neg a_{2,1} \vee a_{2,3} \vee a_{2,5}$$

> **Proposition** : f est satisfiable $\iff$ il existe un ensemble indépendant S de card supérieur à m.

L'idée est qu'on va mettre dans une même clique les variables d'une même clause donnée (pour la CS), et qu'on va relier deux sommets s'ils la négation et l'affirmation d'une même variable (pour interdire le choix de x et $\neg x$).

- si la formule est vraie, *alors* chaque clause i contient une variable $a_{i,j}$ vraie : l'ensemble des sommets correspondants est indépendant (et il est de taille m).
- si on a un ensemble indépendant $\{v_1, \cdots, v_n\}$ de taille supérieure à m, alors nécessairement les v_i sont dans des cliques différentes et deux v_i ne sont pas l'affirmation et la négation d'une même variable : chaque v_i appartient à une clause C_i, qui est donc vraie, et f est vraie.



- $\Longleftarrow$:

La réduction est en LOGSPACE :

- construire les sommets un parcours des littéraux de la formule
- Pour les arêtes : deux boucles `for` imbriquées

EX 2

> C est un recouvrement *ssi* C^c est indépendant.

NB : la réduction est encore en LOGSPACE.

EX 3

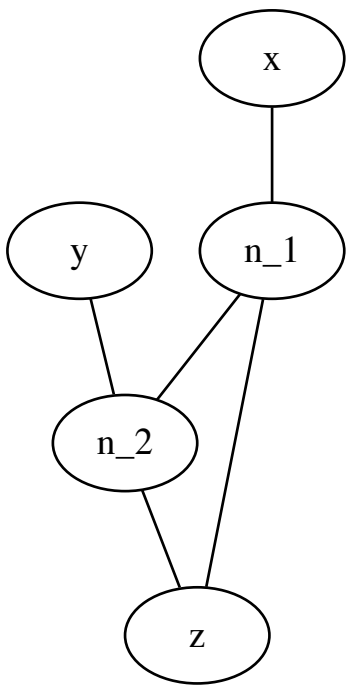
> C est une clique *ssi* C est un ensemble indépendant dans le graphe complémentaire.

EX 4

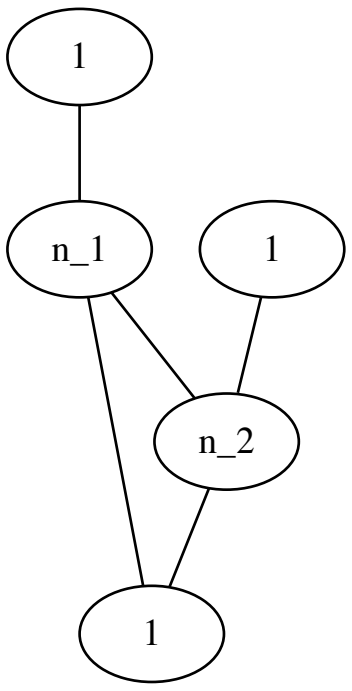
Réduction à partir du problème 3-SAT.

- Vert ⟶ Vrai = 1
- Rouge ⟶ Faux = 0
- Bleu ⟶ ? (degré de liberté)

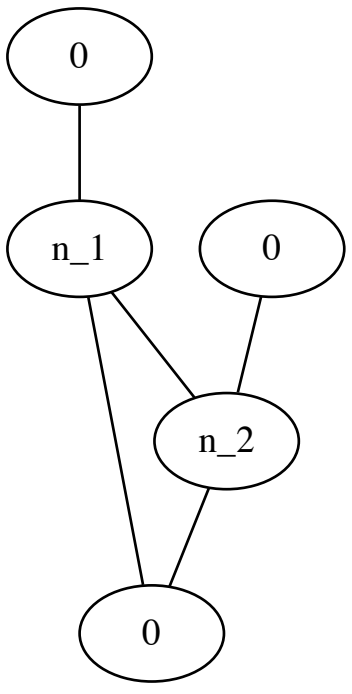
Gadget :



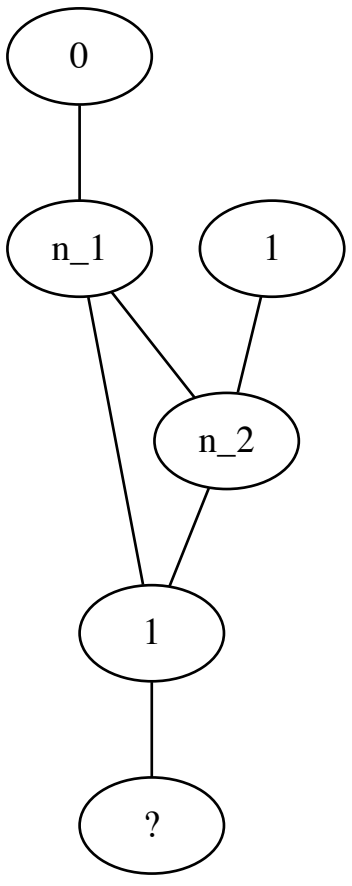
Si $x = y = 1 \longrightarrow$ on peut trouver n_1, n_2 pour avoir $z = 1$



Si $x = y = 0 \longrightarrow$ on peut trouver n_1, n_2 pour avoir $z = 0$



Si $x = 1, y = 0 \longrightarrow$ en reliant la sortie à un sommet colorié "?", on peut forcer $z = 1$



$\Longrightarrow$ On a presque codé une porte “OU” : avedc ce dispositif (“gadget”), on sait que si z vaut 1 en sortie, c’est qu’une des entrées valait 1.

$$f = C_1 \wedge \cdots \wedge C_n$$

où les C_i sont des 3-clauses.

$$\Sigma = \{x_1, \cdots, x_n\}$$

NP-hard :

On devine un coloriage h.

On vérifie que

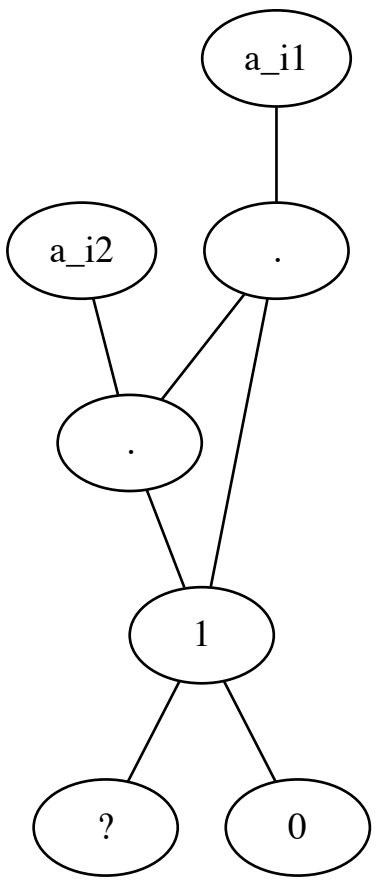
$$\forall (u, v) \in E, h(u) \neq h(v)$$

NP-completeness:

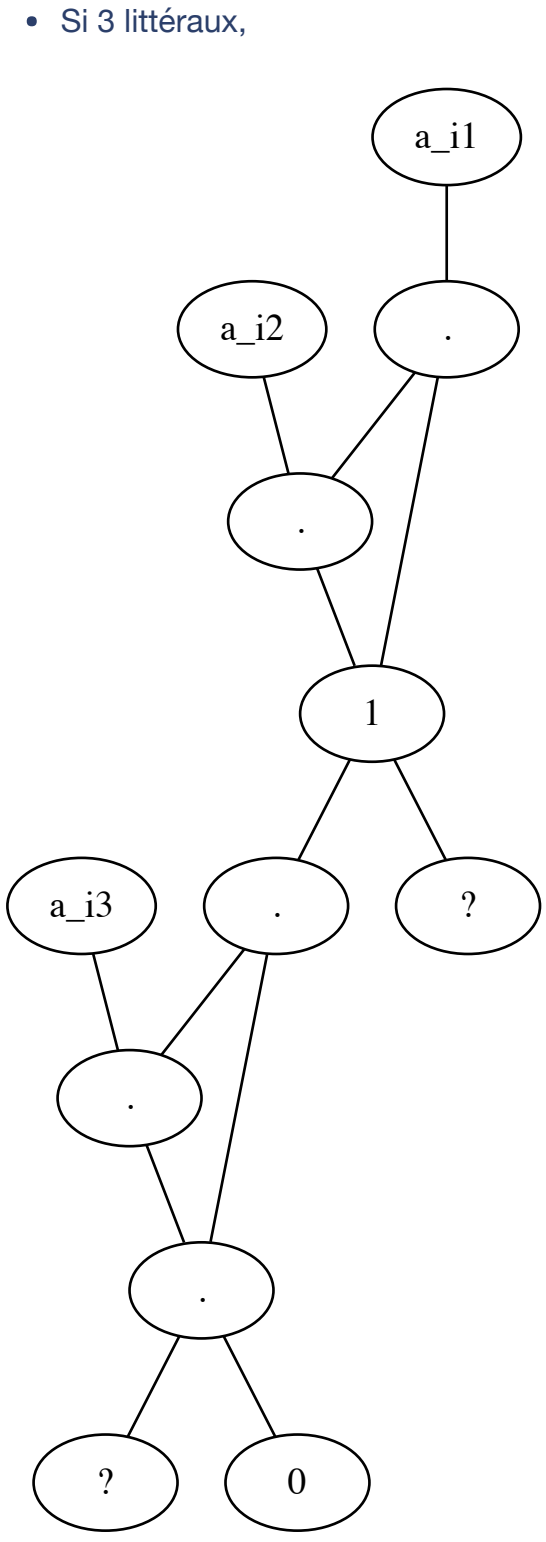
liaison des sommets	x_1	x_2	$\cdots$	x_n
true	?	?	$\cdots$	?
false	?	?	$\cdots$	?

Dans C_i :

- Si 1 littéral, on relie a_i à 0
- Si 2 littéraux,



- Si 3 littéraux,

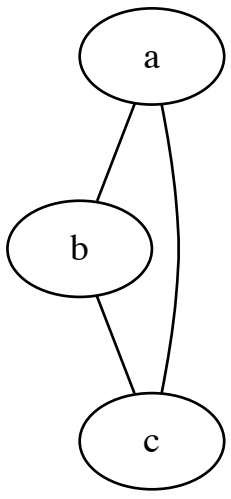


EX 5

Réduction du 3-coloring à ce problème.

Soit h un 3-coloriage de V.

Soit G' le graphe :



Soit $f(v) =$

- a si $h(v) = B$
- b si $h(v) = R$
- c si $h(v) = G$

Donc f est un homomorphisme.

Réciproquement,

soit f un hom., h le coloriage.

$h(v) =$

- B si $f(v) = a$
- R si $f(v) = b$
- G sinon

Si $(u, v) \in E$, alors $(f(u), f(v)) \in E'$.

Donc $f(u) \neq f(v)$

Donc $h(u) \neq h(v)$

Soit $(u, v) \in E$, on sait $h(v) \neq h(u)$.

Donc $f(u) \neq f(v)$

Donc $(f(u), f(v)) \in E'$

```
h = ∅

For v∈V :
    Devine v'∈V'
    h = h ∪ {v ⟶ v'}

For (u,v)∈E:
    Si (h(u), h(v))∉E' :
Echec
```