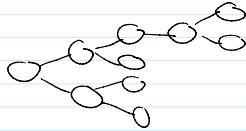


Complexité en espace et PSPACE

27 mars 2023 10:17

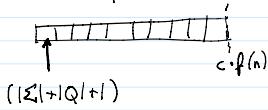
- Une MT dét. est en espace $O(f(n))$ si elle utilise tjs au plus $c \cdot f(n)$ cellules de ruban distinctes. ($c = \text{constante}$)
- Une MT non-dét est en espace $O(f(n))$ si tsg. de transitions possible, la MT accède à $\leq c \cdot f(n)$ cellules



- Espace \leq Temps
- Temps $\leq 2^{\text{Espace}}$

Prop: si une MT M est en espace $O(f(n))$ et que M arrête sur toute entrée, alors M est en temps $O(2^{d \cdot f(n)})$, d=constante

- Si M n'a pas de boucle \varnothing , alors M rerencentre par 2^d la même config.
- Supposons que M utilise un espace $\leq c \cdot f(n)$



Le #de cfg sur cet espace est $\leq (|\Sigma| + |Q| + 1)^{c \cdot f(n)}$

Au pire, la MT visite chaque cfg une fois

$$\begin{aligned} \Rightarrow \text{le temps est } & \leq (|\Sigma| + |Q| + 1)^{c \cdot f(n)} \\ & = (2^{\log_2 (|\Sigma| + |Q| + 1)})^{c \cdot f(n)} \\ & = (2^d)^{c \cdot f(n)} = 2^{d \cdot c \cdot f(n)} \\ & \quad d \cdot c = \text{constante} \end{aligned}$$

$\text{DSPACE}(f(n)) = \{L : \exists \text{MT } M \text{ qui décide } L \text{ en espace } O(f(n))\}$

$\text{NSPACE}(f(n)) = \{L : \exists \text{MT non-dét } M \text{ dont le langage est } L \text{ et qui prend un espace } O(f(n))\}$

$$\text{PSPACE} = \bigcup_{k \geq 0} \text{DSPACE}(n^k)$$

$$\text{NPSPACE} = \bigcup_{k \geq 0} \text{NSPACE}(n^k)$$

Prop: $P \subseteq \text{PSPACE}$

Prop: $NP \subseteq \text{PSPACE}$

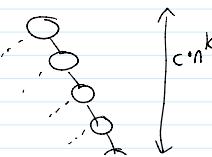
Soit $L \in NP$, $\exists \text{MT } M \text{ non-dét. avec langage } L \text{ en temps } \leq c \cdot n^k$.

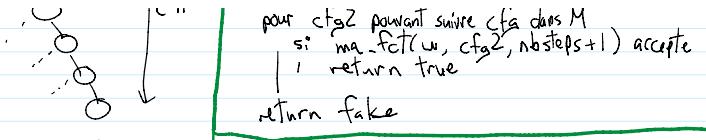
Pour décider L en espace poly:

```

ma-fct(w, cfg, nbsteps)
    si cfg est acceptante
        return true
    si nbsteps > c * n^k (n = |w|)
        return false
    pour cfg2 pouvant suivre cfg dans M
        si ma-fct(w, cfg2, nbsteps+1) accepte
            return true
    end
}
  
```

Cet algo explore un arbre de récursion de profondeur $\leq c \cdot n^k$





De plus, l'espace Appel int: $ma_fct(w, cfg_init, 0)$

pour chaque appel est en espace $O(n^k)$ car
on n'a qu'à stocker cfg et $cfg2$, et chaque cfg est $O(n^k)$
car M prend un temps $O(n^k)$.

Donc l'espace est $O(\text{prof de l'arbre} \times \text{espace par nœud})$
 $= O(n^k \cdot n^k) = O(n^{2k})$ \blacksquare

Autre preuve:

On montre $SAT \in PSPACE$.

$\text{sat}(\varphi, \text{vars } x_1, \dots, x_n)$

//espace $O(n)$ pour chaque assignation A des x :

//espace $O(|\varphi|)$ | si A satisfait φ
| | accepte
| rejette

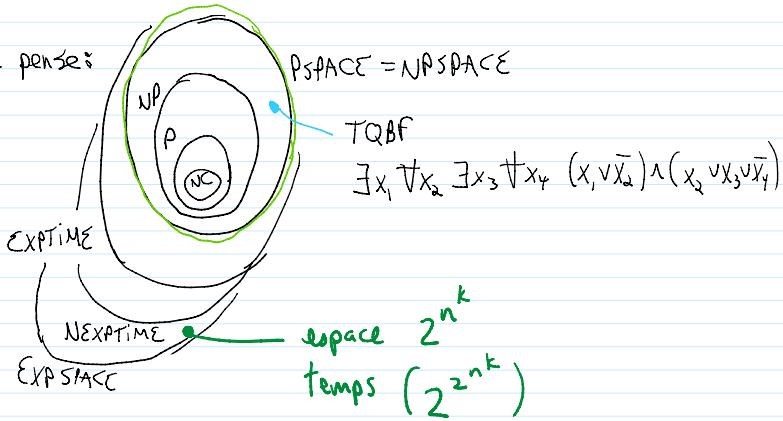
$\Rightarrow SAT \in PSPACE \Rightarrow \exists L \in NP, L \leq_p SAT$

$\Rightarrow \exists f$ en espace + temps poly t_f,

$w \in L \Leftrightarrow f(w) \in SAT$

\Rightarrow Pour décider L , rouler $\text{sat}(f(w))$ \blacksquare

Ce qu'on pense:



$PSPACE \neq EXPSPACE$

Autres classes d'espace

$\text{LOGSPACE} = \text{DSPACE}(\log n) = L$

ex: expression bien parentésée: $(()(())())$

$\overbrace{+/+ -/+/- \dots}^{\text{}}$

$NLOGSPACE = NSPACE(\log n) = NL$

$\Theta: L \stackrel{?}{=} NL$

$$\text{NLLOGSPACE} = \text{NSPACE}(\log n) = \text{NL}$$

$$Q: L \stackrel{?}{=} NL$$

$$\text{PSPACE} = \text{NPSPACE}$$

Théorème de Savitch

$$\bullet \text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f(n)^2)$$

Ceci implique $\text{PSPACE} = \text{NPSPACE}$, car

$$\bullet \text{PSPACE} \subseteq \text{NPSPACE} \quad (\text{trivial})$$

$$\bullet \text{NPSPACE} \subseteq \text{PSPACE}$$

Soit $L \in \text{NPSPACE}$. Alors $\exists k \forall q \ L \in \text{NSPACE}(n^k)$

Par Savitch, $L \in \text{DSPACE}(n^{2k})$

$$\Rightarrow L \in \text{PSPACE}$$

Pour prouver Savitch, on a besoin de:

Lemme: Soit $\text{DPATH} = \{\langle G, s, t, k \rangle : G \text{ est un graphe orienté et } \exists \text{ un chemin de } s \text{ à } t \text{ avec } \leq k \text{ arêtes dans } G\}$

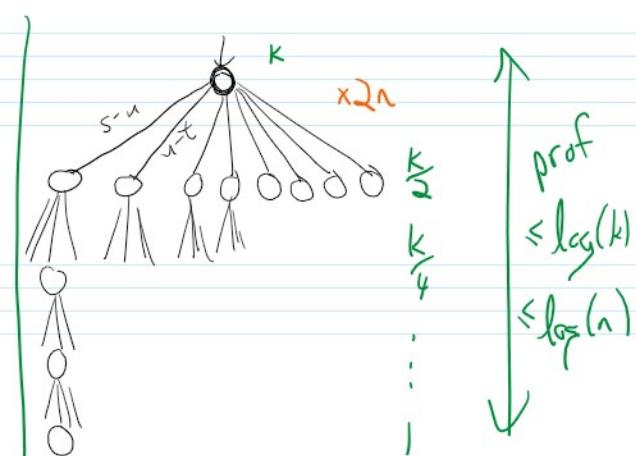


Alors, on peut décider DPATH en utilisant un espace $O((\log n)^2)$, où $n = |V(G)|$.

Preuve: voici un algo:

```

dpath(G, s, t, k)
    si k == 0, return (s == t)
    si k == 1, return ((s, t) ∈ E(G))
    // brancher sur le pt milieu u entre s et t
    // s ————— u ————— t
    pour u ∈ V(G)
        si dpath(G, s, u, ⌈ k/2 ⌉) ∧ dpath(G, u, t, ⌈ k/2 ⌉)
            | return true
        return false
    
```



Cet algo explore un arbre de récursion de profondeur $O(\log n)$. Chaque appel prend un espace $O(\log n)$ pour stocker s, t, u, k si on suppose que les sommets de G sont représentés par $V(G) = \{1, 2, \dots, n\}$.
 \Rightarrow espace total $\in O(\log n \cdot \log n) = O((\log n)^2)$.

sommets de Σ sont représentés par $V(\Sigma) = \{c_1, c_2, \dots, c_n\}$.

\Rightarrow espace total $\in O(\log n \cdot \log n) = O((\log n)^2)$. \square

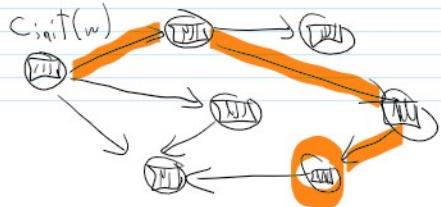
Soit $L \in \text{NSPACE}(f(n))$. $\exists M$ non-déf. avec langage L

en espace $O(f(n))$.

Encore une fois, le #de configs possible est $O(2^{d \cdot f(n)})$

Soit G_M le graphe où

- $V(G_M) = \text{cfgs possibles de } M$
- $(c_i, c_j) \in E(G_M)$ si M permet d'aller de c_i vers c_j .



On note que $w \in L$ si:

Échecin de c_{init} vers un c_i acceptant.

Voici l'algo pour décider L en espace poly:

pour chaque c_i acceptante // $O(f(n))$ espace

| si $\text{dpath}(G_M, c_{init}, c_i, 2^{df(n)})$
| accepter

rejeter

L'espace requis est $O(f(n))$ pour c_i

$$O(\log |V(G_M)|^2 + \log(2^{df(n)}))$$

$$= O(\log(2^{df(n)})^2 + \log(2^{df(n)}))$$

$$= O(\log(2^{df(n)})^2)$$

$$= O((df(n) \cdot \log 2)^2)$$

$$= O(d^2 \cdot (f(n))^2)$$

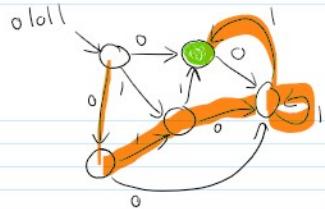
$$= O((f(n))^2) \quad \square$$

stocker l'entier
 $2^{df(n)}$

Corollaire: $\text{PSPACE} = \text{NPSPACE}$

Exemple d'utilité:

AFN = automate fini non-déterm.



$\text{AFN}_\emptyset = \{ A : A \text{ est un AFN qui n'accepte rien} \}$

Prop: $\text{AFN} \in \text{PSPACE}$

Lemme: si un AFN A ne rejette pas tout, $\exists w \in \Sigma^*$ accepté avec $|w| \leq 2^{|A|}$ ($|A| = \# \text{états}$).

idée: pour chaque w de taille $2^{|A|}$ // ne marche pas (espace exponent.)
voir si A accepte w

On montre que $\overline{\text{AFN}}_\emptyset \in \text{NPSPACE}$

$$\overline{\text{AFN}}_\emptyset = \{ A : A \text{ accepte } \geq 1 \text{ mot} \}$$

Sur entrée A
 $S = \text{état init de } A$
pour $i = 1 \dots 2^{|A|}$ espace $O(C \log(2^{|A|})) = O(|A|)$
 |
 | $c \leftarrow$ symbole trouvé par non-déterminisme
 | $S \leftarrow$ état suivant S sur symbole c trouvé par non-déterm.
 | si S est acceptant
 | accepter
 |
 X

$$\begin{aligned} &\Rightarrow \overline{\text{AFN}}_\emptyset \in \text{NPSPACE} \\ &\Rightarrow \overline{\text{AFN}}_\emptyset \in \text{PSPACE} \\ &\Rightarrow \overline{\overline{\text{AFN}}_\emptyset} \in \text{PSPACE} \\ &\Rightarrow \overline{\text{AFN}}_\emptyset \in \text{PSPACE}. \end{aligned}$$

Pour montrer $L \in \text{PSPACE}$

- 1) Force brute (SAT)
 - 2) Algo récursif ($\text{NP} \subseteq \text{PSPACE}$)
fouille en prof.
 - 3) $L \in \text{NPSPACE}$
-

PSPACE -complétude

Un langage L est PSPACE -complet si:

$$1) L \in \text{PSPACE}$$

$$2) \forall L \in \text{PSPACE} \exists c \in \mathbb{N} \text{ tel que } L \leq_p cL$$

1) $L \in \text{PSPACE}$

2) L est PSPACE-difficile, i.e.

$\forall L' \in \text{PSPACE}, L' \leq_p L \quad (\exists f \text{ en temps poly } t_f \quad w \in L' \Leftrightarrow f(w) \in L)$

TQBF : True Quantified Boolean Formula

Formule avec alternance de \exists, \forall

ex: $\exists x_1 \forall x_2 \exists x_3 \forall x_4 (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \bar{x}_3)$

$x_1 \in \{\top, \perp\}$ ex: $x_1 = \top \quad x_1 = \perp \quad \notin \text{TQBF}$

$x_2 \in \{\top, \perp\}$ $x_2 = \perp \quad x_2 = \top$

$x_3 \in \{\top, \perp\}$ $x_3 = \perp$

$x_4 \in \{\top, \perp\}$ $x_4 = \perp$

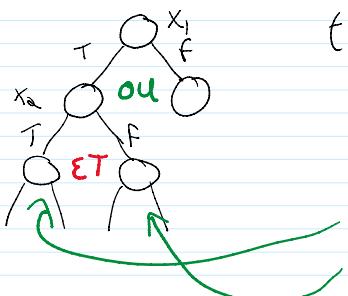
SAT: $\exists x_1 x_2 x_3 \dots x_n \psi$

Thm: TQBF est PSPACE-complet

1) TQBF $\in \text{PSPACE}$

(2) PSPACE-difficile exercice

$\exists x_1 \forall x_2 \exists x_3 \dots \forall x_n \psi$



$t_{qbf}(\psi = \exists x_1 \forall x_2 \dots \exists x_n \forall x_n \psi)$

si: ψ n'a pas de variables
retourner la valeur de ψ

$\varphi_1 = \psi$ avec $x_1 = \top$ et $x_2 = \perp$

$\varphi_2 = \psi$ avec $x_1 = \top$ et $x_2 = \top$

si: $t_{qbf}(\varphi_1) \wedge t_{qbf}(\varphi_2)$
accepter

$\varphi_3 = \psi$ avec $x_1 = \perp$ $x_2 = \top$

$\varphi_4 = \psi$ avec $x_1 = \perp$ $x_2 = \perp$

si: $t_{qbf}(\varphi_3) \wedge t_{qbf}(\varphi_4)$
accepter

rejetter

CG: Dans GG, on reçoit un graphe orienté $G = (V, E)$

GG: Dans GG, on reçoit un graphe orienté $G = (V, E)$ et un sommet de départ s . Les 2 joueurs doivent construire un chemin P .

Au départ $P = (s)$

J1: choisir v_1 tel que $(s, v_1) \in E$. P devient (s, v_1) .

J2: choisir $v_2 \neq v_1$ $(v_1, v_2) \in E$. P devient (s, v_1, v_2)

J1: ...

P ne doit pas contenir le même sommet 2 fois.

Le premier joueur qui ne peut rien ajouter à P a perdu.

$GG = \{ \langle G, s \rangle : \text{joueur 1 peut toujours gagner sur } (G, s) \}$

Thm: GG est PSPACE-complet

1) GG ∈ PSPACE. Faire comme TQBF.

2) GG est PSPACE-difficile.