Révisions

→ Méthode du potentiel (pile avec multipop)
→ Programmation dynamique (multiplication d'une suite finie de matrices)

→ Exp. rat. → automates, déterminisation
→ NP-complétude (3SAT, réductions)

1/ Pile (liste simplement chaînée)

$O(1) \leq \beta$ Push / Empiler
$O(1) \leq \beta$ Pop / Dépiler
$O(k) \leq \beta^k$ Multipop(k) / Enlever les k éléments en haut de la pile et les retourner (k fois une opération Dépiler)

Pq tte séquence d'opérations a une complexité amortie par opération en $O(1)$.

Séquence de n opérations de type Push / Pop / Multipop à partir de la pile vide.

On utilise la méthode du potentiel.

$$\begin{cases} \underline{\Phi}(X) \geq 0 \\ \underline{\Phi}(\text{pile vide}) = 0 \\ \underline{\Phi}(X) \text{ doit être grand avant de réaliser une opération coûteuse à amortir, et faible après cette opération.} \end{cases}$$

$\underline{\Phi}(X) = \#\text{éléments sur la pile} \times \alpha$ pour un certain $\alpha \geq 0$

Différences de potentiel

Push	$\underline{\Phi}(\text{Push}(X, v)) - \underline{\Phi}(X) = \alpha$
Pop	$\underline{\Phi}(\text{Pop}(X)) - \underline{\Phi}(X) = -\alpha$
Multipop	$\underline{\Phi}(\text{Multipop}(X, k)) - \underline{\Phi}(X) = -k\alpha$

Complexité amortie

$$\hat{c}(\text{Push}(X,v)) = c(\text{Push}(X,v)) + \alpha \leq \beta + \alpha = 2\alpha$$

$$\hat{c}(\text{Pop}(X)) \leq \beta - \alpha = 0$$

$$\hat{c}(\text{Multipop}(X,k)) \leq \beta k - \alpha k = 0$$

en choisissant $\alpha = \beta$

Complexité amortie $O(1)$.

(Complexité de la séquence

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c}_i - \underbrace{\Phi(X_n)}_{\leq 0} + \underbrace{\Phi(X_0)}_{=0}$$

$$\leq \sum_{i=1}^{n} \hat{c}_i \leq n \times 2\alpha$$

2/ Multiplication d'une suite finie de matrices

$$A_1 \times A_2 \times A_3 \times \dots \times A_n$$

Dimensions : $p_0 \quad p_1 \quad p_2 \quad p_3 \quad p_{n-1} \quad p_n$

n matrices A_1, A_i, A_n de dimension $p_{i-1} \times p_i$

On utilisera la méthode naïve pour multiplier deux matrices : le produit d'une matrice A $k \times l$ par une matrice B $l \times m$ se fait en $O(k \times l \times m)$

```
Pour i = 1 à k
    Pour j = 1 à m
        Resultat[i][j] ← 0
        Pour w = 1 à l
            Resultat[i][j] ← Resultat[i][j] + A[i][w]
                             × B[w][j]
```

$$\left(\text{Resultat}[i][j] = \sum_{w=1}^{l} A[i][w] \times B[w][j] \right)$$

Ex. $A_1 \times A_2 \times A_3$

1×10 $\quad 10\times5$ $\quad 5\times2$

$A_1 \times A_2 \times A_3 = (A_1 \times A_2) \times A_3 = A_1 \times (A_2 \times A_3)$

$1\times5 \quad 5\times2 \qquad 1\times10 \quad 10\times2$

deux fois plus coûteux

Nb de multiplications

$50 = 1\times10\times5$

$100 = 10\times5\times2$

$10 = 1\times5\times2$

$20 = 1\times10\times2$

60

120

pour calculer $A_1 \times A_2$

$A_2 \times A_3$

$(A_1 \times A_2) \times 3$

$A_1 \times (A_2 \times A_3)$

On résout le problème de parenthésage optimal de la multiplication de matrices par programmation dynamique en considérant $c(i,j)$ le coût minimal de multiplier $A_i \times \dots \times A_j$ (avec $j \geq i$).

$$\begin{cases} c(i,i) = 0 \\ c(i,j) = \min\limits_{i \leq k < j} c(i,k) + c(k+1,j) + p_{i-1} \times p_k \times p_j \\ \text{pour } i < j \end{cases}$$

$(A_i \dots A_k) \times (A_{k+1} \dots A_j)$

i \ j	1	2	3	4	5	…
1	0	$p_0p_1p_2$	□ →			
2	—	0	$p_1p_2p_3$			
3	—	—	0			
4	—	—	—	0		
5	—	—	—	—	0	
…						

$\min(c(1,1) + c(2,3) + p_0p_1p_3,$
$c(1,2) + c(3,3) + p_0p_2p_3)$

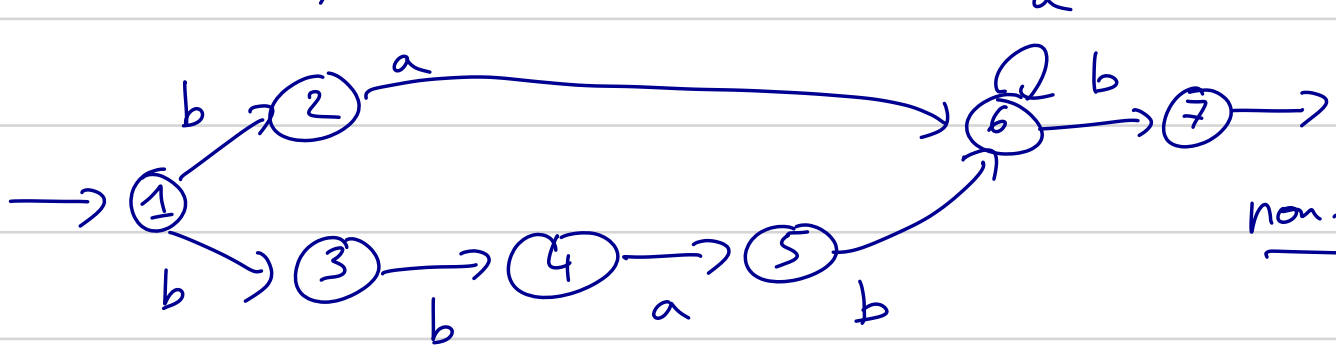
⚠ Il faut parcourir le tableau diagonale par diagonale (par $(j-i)$ croissant)

3/ Exp. rationnelles $\Sigma = \{a, b\}$

$$((a|b)^* aa)^* ba$$

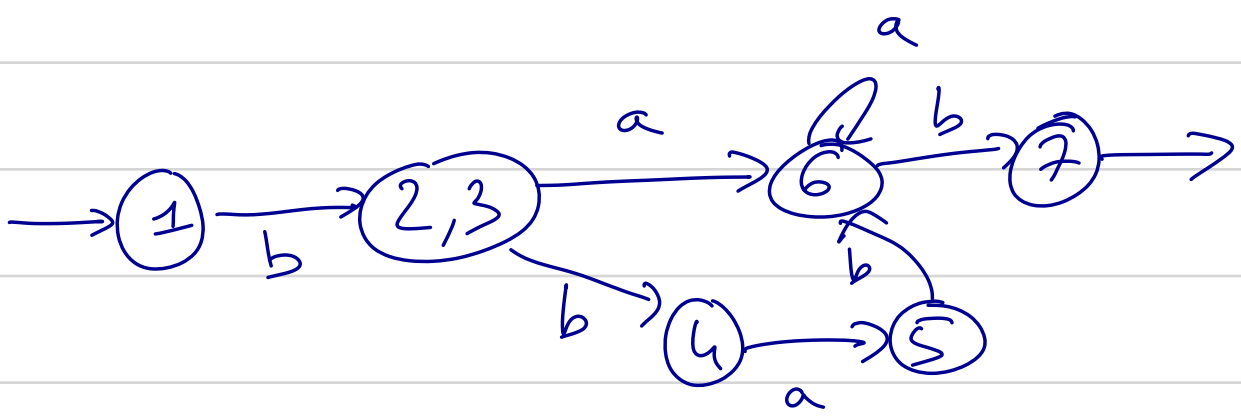
- 0 ou plusieurs fois :
  - 0 ou plusieurs fois :
    - { un a ou un b
  - suivi de un a suivi de un a
- suivi de un b suivi de un a

$$(ba|bbab)a^*b$$



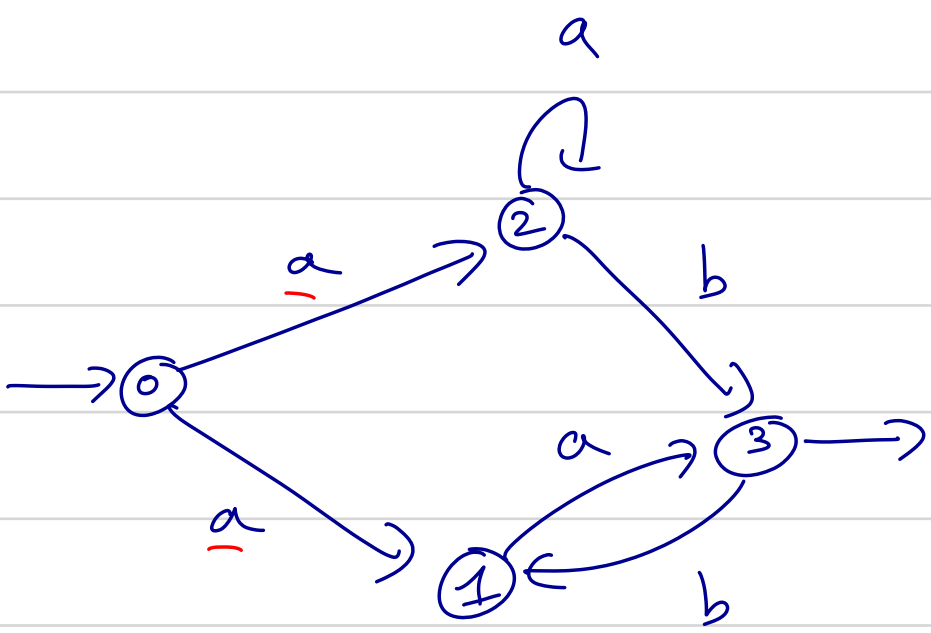
non-déterministe

Déterminisé :



$$(aa^*b|aa)(ba)^*$$
$$\equiv a(a^*b|a)(ba)^*$$

non déterministe

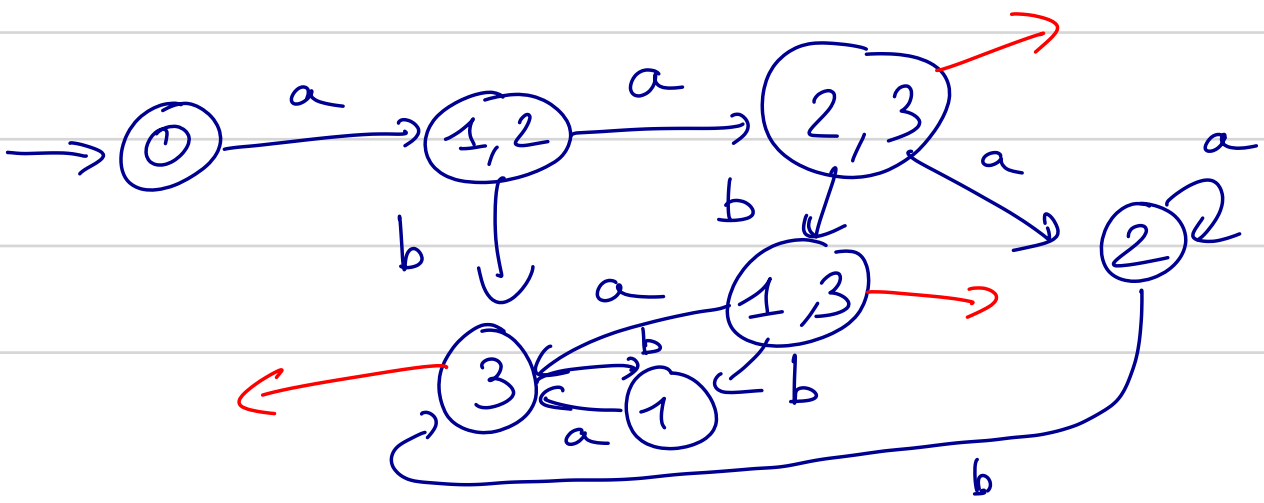


déterminisation

Mots du langage

aababa

aaaaabbabaaba



NP-complet

Rappel Un problème Q est NP s'il existe {une / machine de Turing / algorithme} non déterministe en temps polynomial qui résout Q.

Un problème Q est NP-difficile si pour tout problème $Q' \in NP$, il existe une réduction en temps polynomial de Q' vers Q.

Un problème est NP-complet s'il est dans NP et NP-difficile.

Une réduction en temps polynomial entre un problème Q' et un problème Q est un algorithme A en temps polynomial déterministe qui prend en entrée une entrée de Q' et produit en sortie une entrée de Q t.q. $Q(A(I)) = Q'(I)$ pour tout I.
On écrit $Q' \leq Q$.

Si Q est NP-difficile alors
$\forall Q' \in NP,\ Q' \leq Q$.

Pour montrer que Q'' est NP-difficile, il suffit de montrer que $Q \leq Q''$.