Listes creuses - plus d'espace que de temps

Omar AitMous [1] Frédérique Bassino [1] Cyril Nicaud [2]

[1]LIPN UMR 7030
Université Paris 13

[2]LIGM, UMR CNRS 8049
Université Paris Est

11 Janvier 2011

Définition

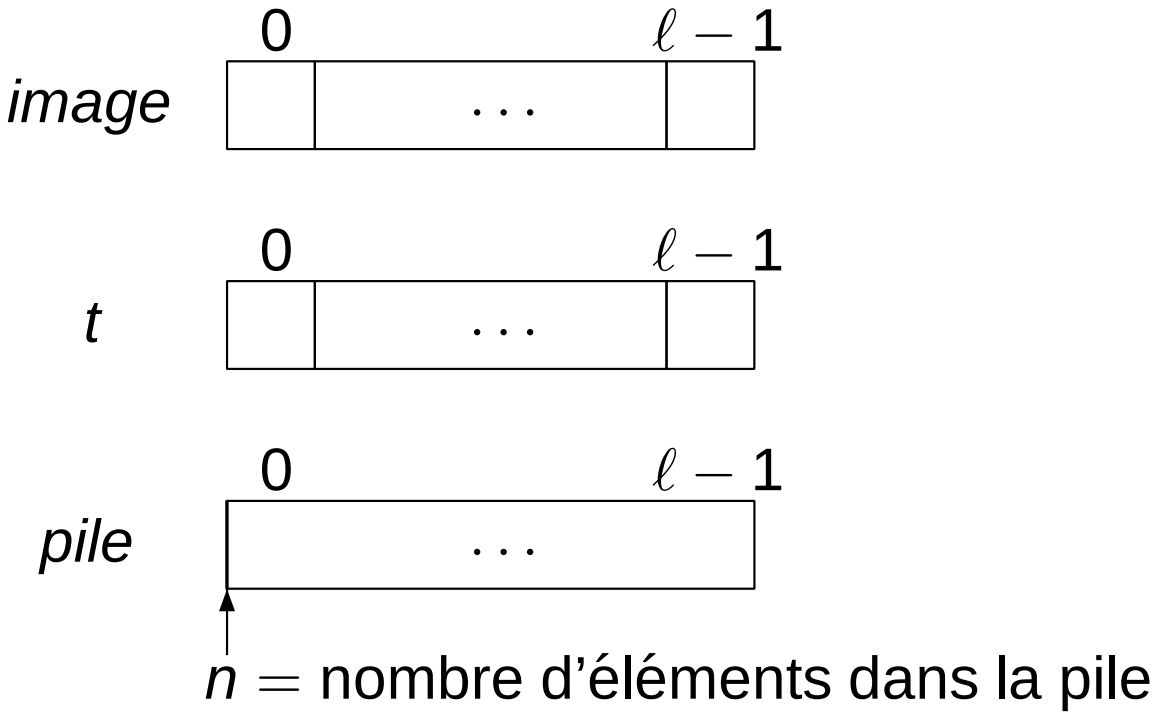
Soit $g : E \to F$ une fonction partielle, où E est un ensemble fini. On note $Dom(g)$ le domaine de g, et $\ell = |E|$.

Une liste creuse est une structure de donnée permettant de représenter g, avec une complexité constante pour les opérations suivantes :

- initialiser $Dom(g) = \emptyset$
- ajouter une valeur $g(x)$ pour $x \in E$
- tester l'existence de $g(x)$
- trouver l'image de x par g si elle est définie
- retirer x de $Dom(g)$

Fonctionnement

Une liste creuse nécessite 3 tableaux de taille ℓ :
Pour simplifier, on considère que $E = \{0, 1, \ldots, \ell - 1\}$.



La complexité en espace est $\mathcal{O}(|E|)$.

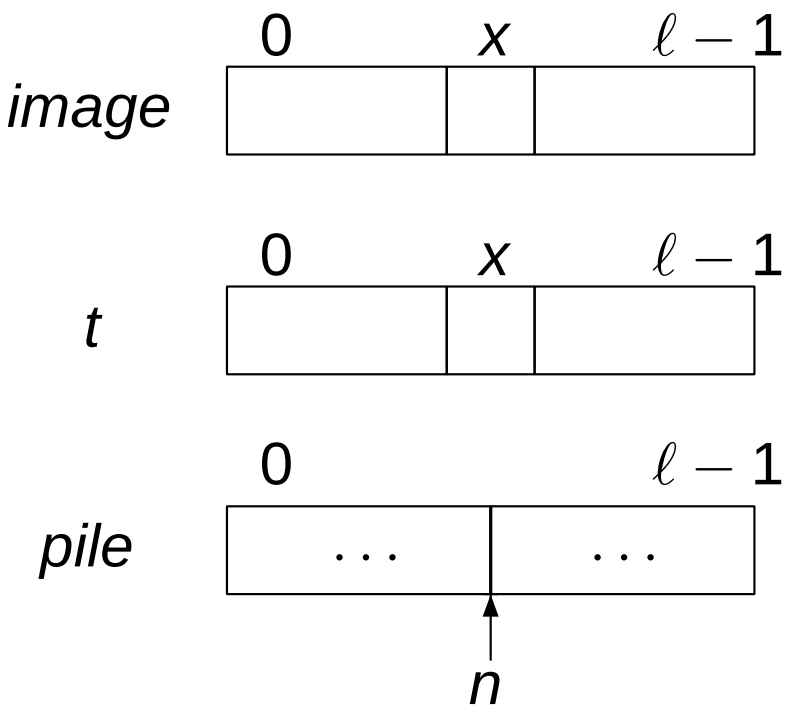
Initialisation

L'initialisation consiste à :

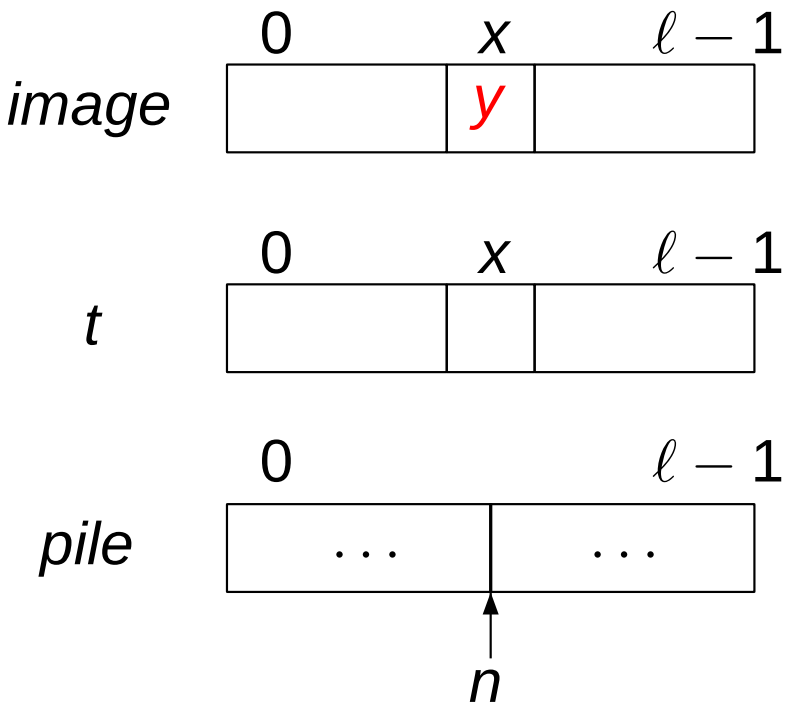
- allouer les 3 tableaux *image*, *t* et *pile* sans les initialiser
- initialiser *n* à 0

Insertion

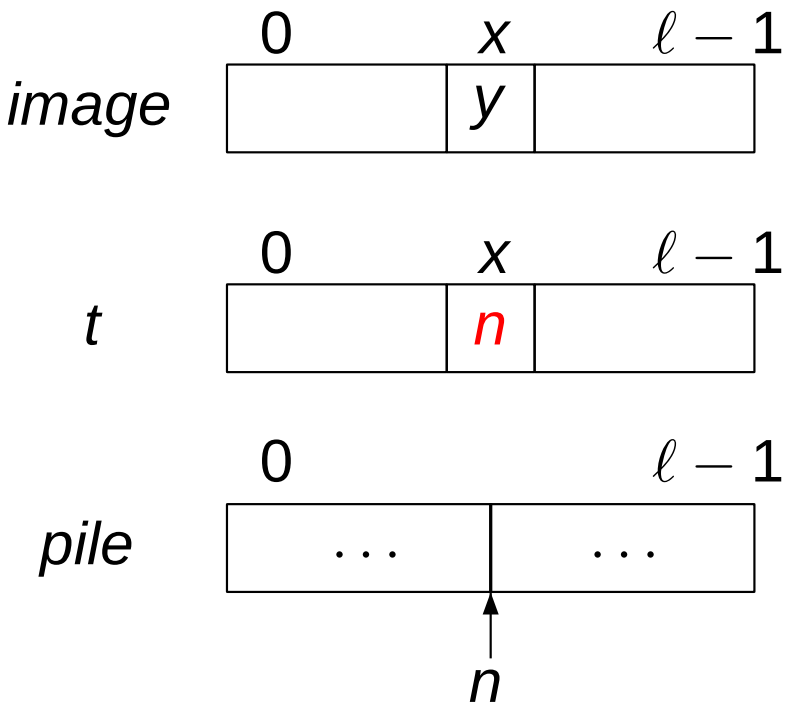
Pour ajouter $g(x) = y$,



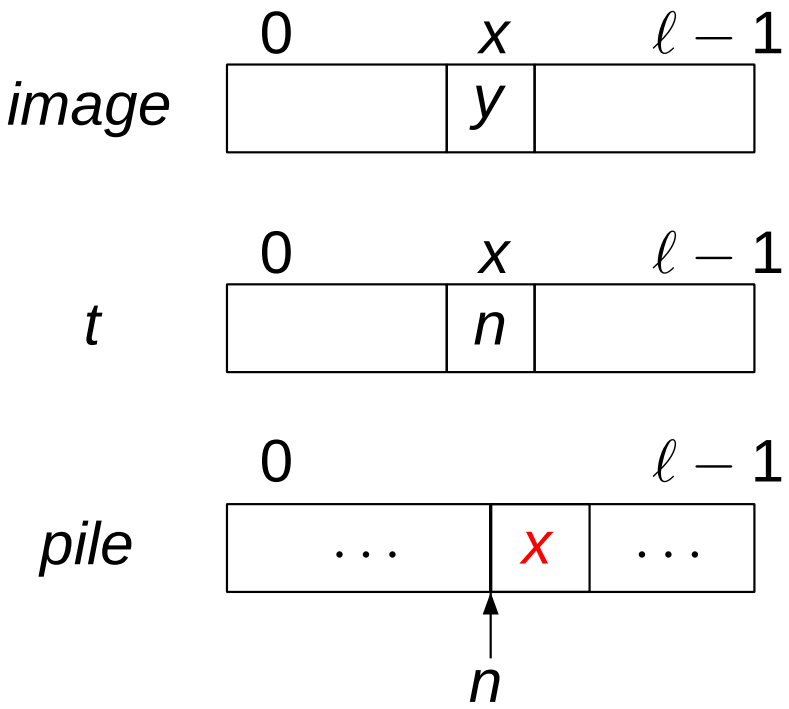
Pour ajouter $g(x) = y$,



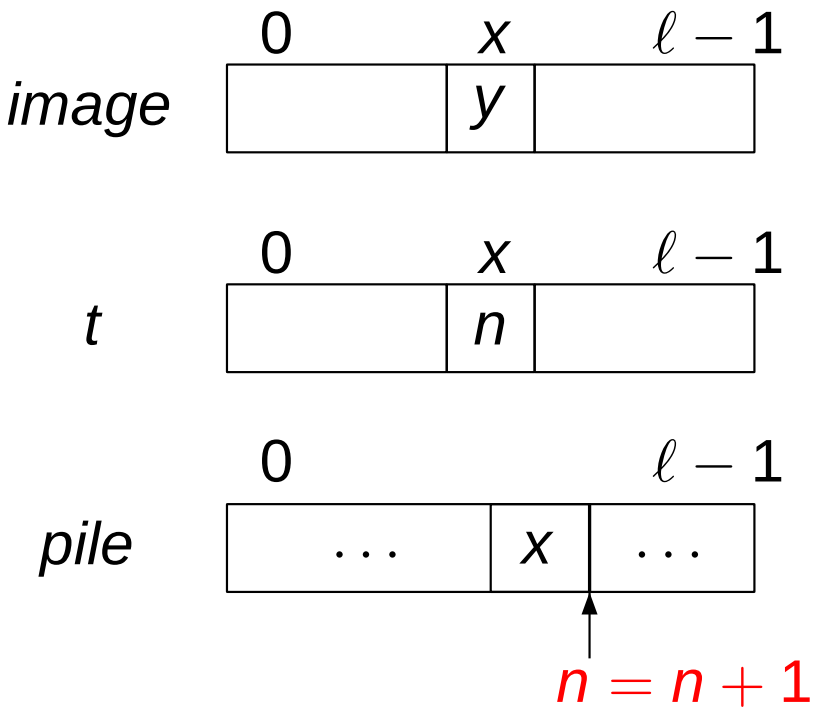
Pour ajouter $g(x) = y$,



Pour ajouter $g(x) = y$,



Pour ajouter $g(x) = y$,



Existence

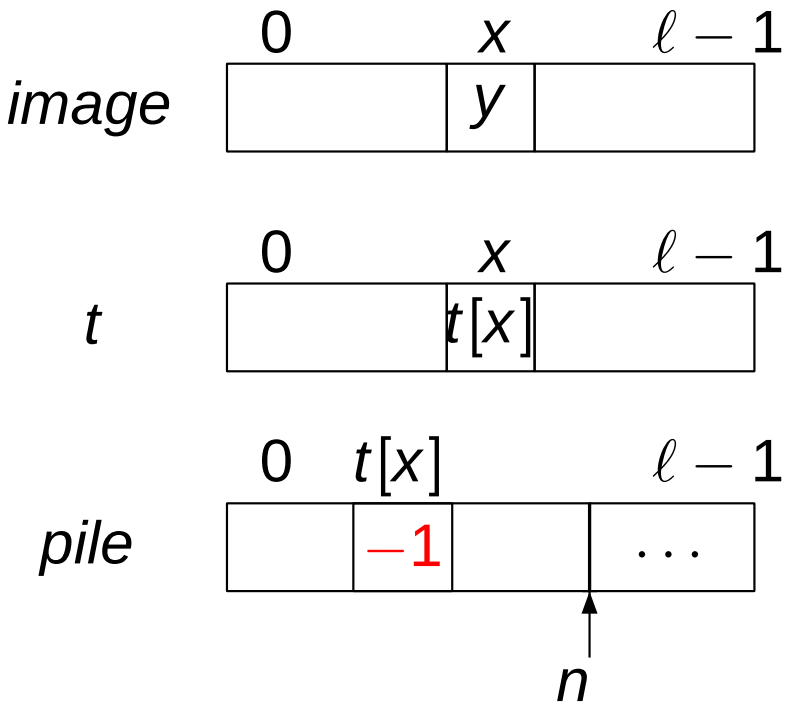
L'image de x par g est définie ssi :

- $0 \leq t[x] \leq n$
- $pile[t[x]] = x$

Dans ce cas, $g(x)$ vaut $image[x]$.

Suppression

Pour retirer x de $Dom(g)$, on remplace $pile[t[x]]$ par -1.



Remarque

On peut définir une liste creuse sous modèle RAM pour les complexités suivantes :

- complexité en espace : $\mathcal{O}(\ell)$
- complexité en temps : $\mathcal{O}(log(\ell))$ (*complexité en bits*)

Récapitulatif

Soit $g : E \to F$ une fonction partielle, où $E = \{0, 1, \ldots, \ell - 1\}$.

Une liste creuse est une structure de donnée permettant de représenter g :

- complexité en espace : $\mathcal{O}(\ell)$
- complexité en temps : $\mathcal{O}(1)$ pour les différentes opérations

Voici quelques situations où les listes creuses permettent :

- améliorer la complexité en temps
- sous réserve que l'on accepte d'utiliser plus d'espace mémoire

Tri lexicographique

Soit X un ensemble de m mots sur un alphabet A de taille k.

Pour regrouper les mots de X selon leur première lettre, on peut utiliser un tableau de taille k (initialisé en temps $\mathcal{O}(k)$).

La case correspondant à la lettre a contiendra les mots de X qui commencent par a.

$\Longrightarrow$ insérer les m mots nécessite un temps de $\mathcal{O}(k+m)$.

On peut améliorer la complexité en temps sous réserve que l'on accepte d'utiliser plus d'espace mémoire.

En remplacant le tableau de taille k par une liste creuse de taille k, l'initialisation se fait en temps $\mathcal{O}(1)$:

$\Longrightarrow$ insérer les m mots nécessite un temps de $\mathcal{O}(m)$.

Automates incomplets (et arbres incomplets)

Manipuler des automates déterministes incomplets sur un grand alphabet A soulève la question de la représentation des états.

On peut représenter un état comme :

- une étiquette
- un tableau de taille $|A|$ pour les transitions sortantes

Problème : initialiser le tableau nécessite un temps $\mathcal{O}(|A|)$.

D'autres structures de données sont possibles.

Exemple : listes chainées, mais ajouter/retirer une transition n'est pas en temps $\mathcal{O}(1)$.

Dans ce cas, on peut utiliser des listes creuses pour :

- initialisation en temps $\mathcal{O}(1)$
- ajout/suppression de transition en temps $\mathcal{O}(1)$

m-uplets

Soit X un ensemble de m mots (où m est fixé), dont la somme des longueurs est n, sur un alphabet A.

On propose un algorithme pour construire l'automate minimal reconnaissant A^*X.

L'approche utilisée (basée sur l'algorithme de Brzozowski) :

- construction directe d'un automate co-déterministe $\mathcal{C}_X$ reconnaissant A^*X ($\mathcal{O}(n)$ en temps et en espace)
- algorithme ad-hoc pour déterminiser $\mathcal{C}_X$

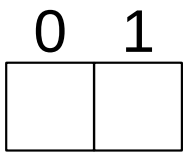
L'automate obtenu est minimal, de taille $\mathcal{O}(n)$, et reconnait A^*X.

Soit $X = \{u_1, u_2, \ldots, u_m\}$.

Durant la déterminisation, on produit des états
$P = (w_1, w_2, \ldots, w_m)$, où w_i est un suffixe non-vide de u_i.

On utilisera un arbre de listes creuses pour la gestion d'états.

Insertion dans l'arbre

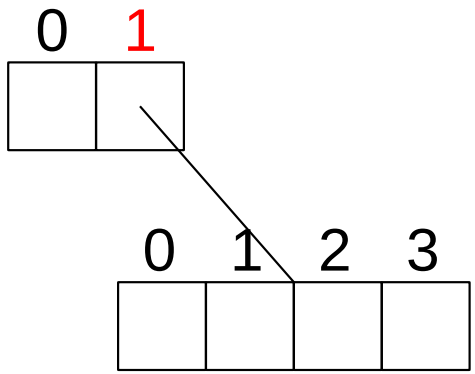


$X = \{aa, aaab, bb\}$.

Soit $P = (a, aab, b)$.

$X = \{aa, aaab, bb\}$.

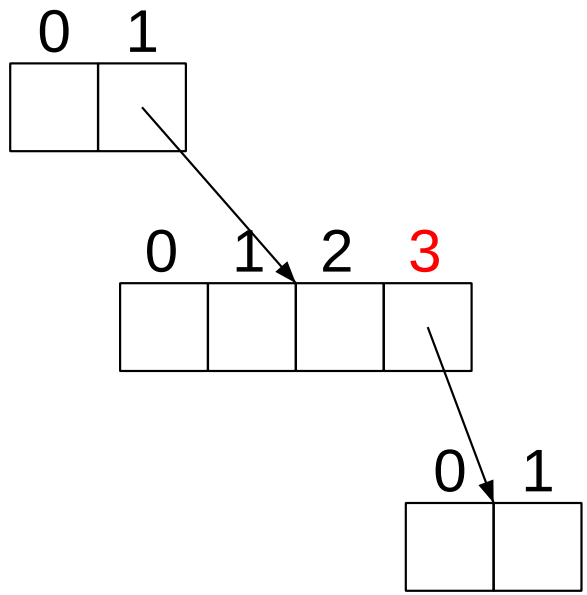
Soit $P = (a, aab, b)$.



m-uplets

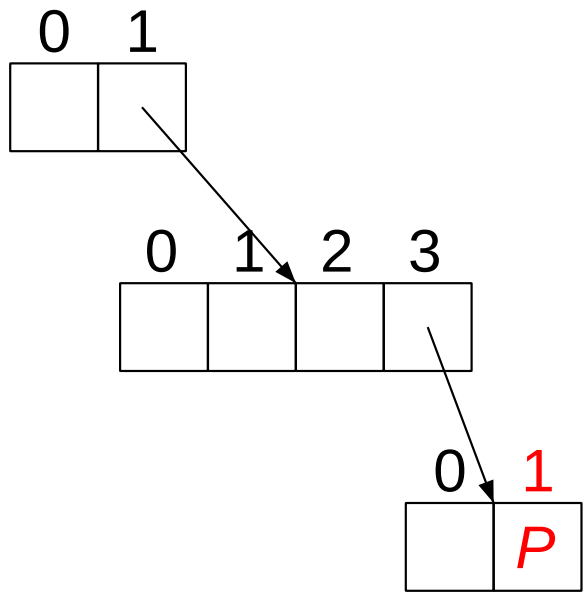
$X = \{aa, aaab, bb\}$.

Soit $P = (a, aab, b)$.



$X = \{aa, aaab, bb\}$.

Soit $P = (a, aab, b)$.



Complexité

Proposition

L'utilisation de listes creuses assure une complexité en temps de $\mathcal{O}(n)$ et une complexité en espace en $\mathcal{O}(n^2)$.

Matrices creuses

Soit $M_{n,m}$ une matrice creuse de taille $n \times m$.

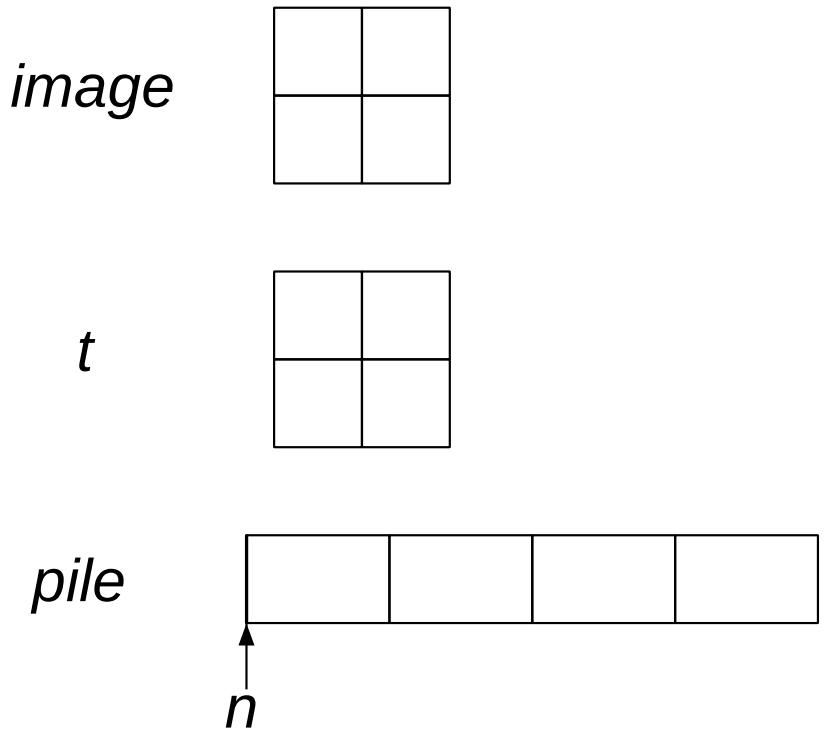
Pour représenter $M_{n,m}$, on peut utiliser une liste creuse de la forme :

- deux tableaux *image* et t, à deux dimensions, et de tailles $n \times m$
- une pile de taille $n * m$ (tableau à une dimension)

Exemple

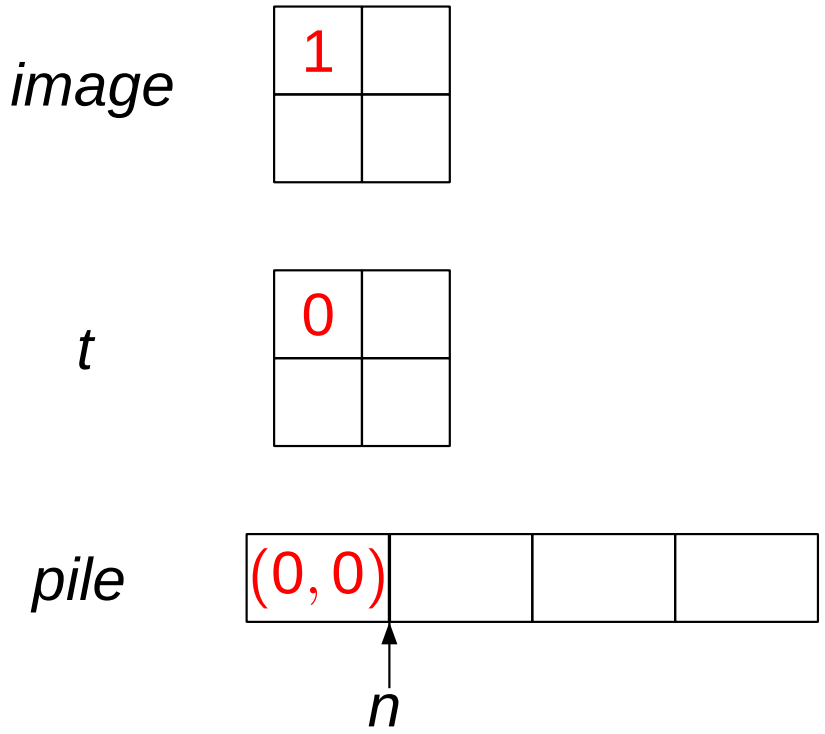
Soit la matrice creuse suivante :

$$M_{m,n} = \begin{pmatrix} 1 & 0 \\ -1 & 3 \end{pmatrix}$$



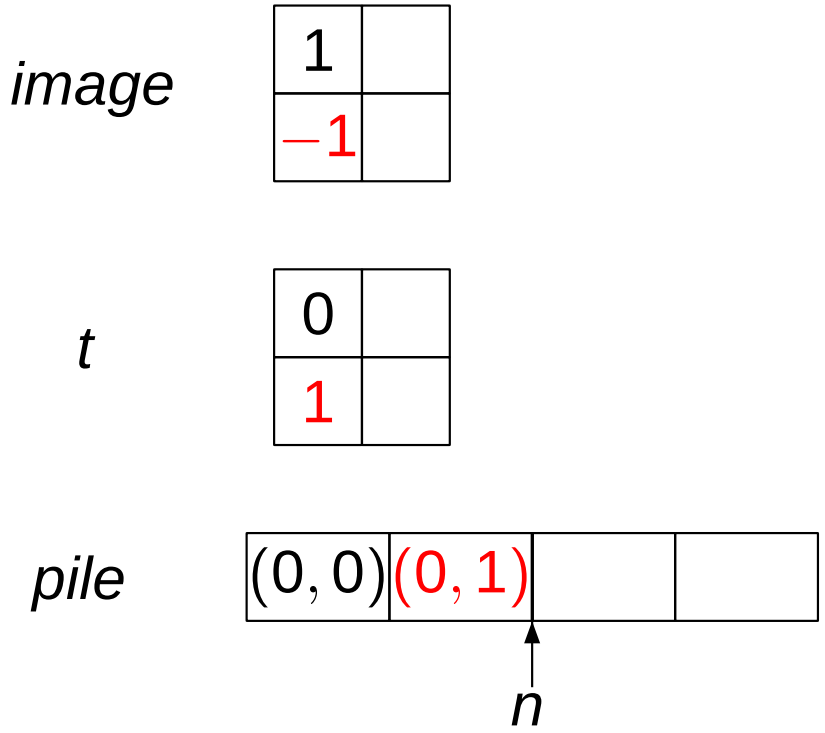
Soit la matrice creuse suivante :

$$M_{m,n} = \begin{pmatrix} 1 & 0 \\ -1 & 3 \end{pmatrix}$$



Soit la matrice creuse suivante :

$$M_{m,n} = \begin{pmatrix} 1 & 0 \\ -1 & 3 \end{pmatrix}$$



Soit la matrice creuse suivante :

$$M_{m,n} = \begin{pmatrix} 1 & 0 \\ -1 & 3 \end{pmatrix}$$

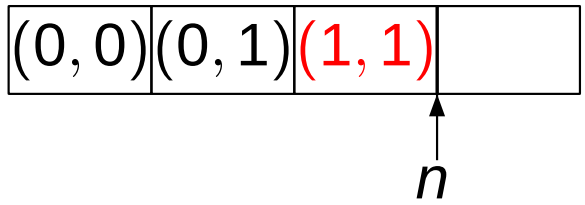
image

1	
−1	3

t

0	
1	2

pile



Soit la matrice creuse suivante :

$$M_{m,n} = \begin{pmatrix} 1 & 0 \\ -1 & 3 \end{pmatrix}$$

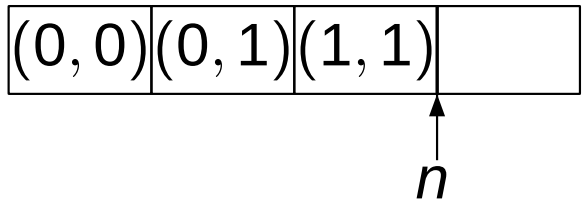
image

1	?
−1	3

t

0	?
1	2

pile



Matrices creuses

En utilisant une telle structure, on obtient, pour des opérations comme l'addition ou la multiplication :

- une complexité en temps équivalente à celle de structures classiques de matrices creuses (listes de listes, . . .)
- nécessite trois fois plus d'espace que la structure de donnée naïve (tableau bidimensionnel)

Conclusion

$g : E \to F$ une fonction partielle, et $\ell = |E|$.

Une liste creuse est une structure de donnée permettant de représenter g, avec :

- Complexité en temps : $\mathcal{O}(1)$ pour les opérations de base
- Complexité en espace : $\mathcal{O}(\ell)$

Utiliser des listes creuses permet :

- améliorer la complexité en temps
- sous réserve de pouvoir utiliser plus d'espace

Merci de votre attention