Comment calculer 3000 termes d'une marche dans le quart de plan

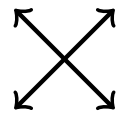
Stephen Melczer

Simon Fraser University &
INRIA Paris–Rocquencourt

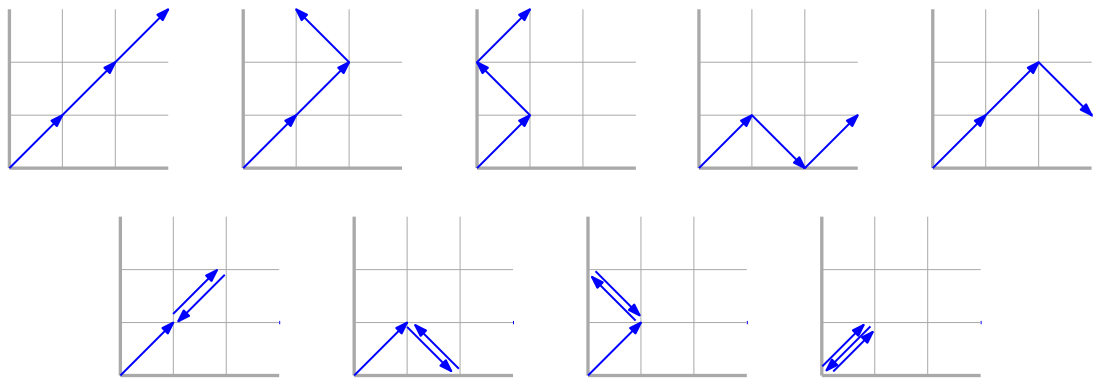
Dénombrement des marches dans le quart de plan

Étant donné : Un ensemble de pas
Compter : Le nombre de marches dans le quart de plan utilisant ces pas.

Par exemple, pour l'ensemble de pas $\mathcal{S} = \{NE, SE, NO, SO\}$



il y a exactement 9 marches de longueur 3 :



Motivations

Reconnaître une suite combinatoire (Catalan, etc.)

Déterminer des estimations asymptotiques

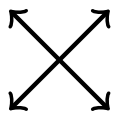
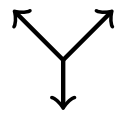
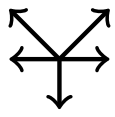
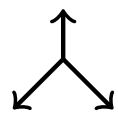
Calculer une équation différentielle pour la série génératrice

Motivations

Reconnaître une suite combinatoire (Catalan, etc.)

Déterminer des estimations asymptotiques

Calculer une équation différentielle pour la série génératrice

S	formule asymptotique	S	formule asymptotique
	$\frac{2}{\pi} \cdot \frac{4^n}{n}$		$\frac{\sqrt{3}}{\sqrt{\pi}} \cdot \frac{3^n}{\sqrt{n}}$
	$\frac{\sqrt{5}}{2\sqrt{2\pi}} \cdot \frac{5^n}{\sqrt{n}}$		$\frac{24\sqrt{2}}{\pi} \cdot \frac{(2\sqrt{2})^n}{n^2}$

Résultats choisis de (Bostan & Kauers, 2009)

Récurrences

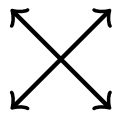
Soit $F(x, y; n)$ le $\#$ de marches à n pas entre $(0, 0)$ et (x, y).

On peut utiliser une récurrence pour déterminer $F(x, y; n)$ à partir de $F(0, 0; 0) = 1$.

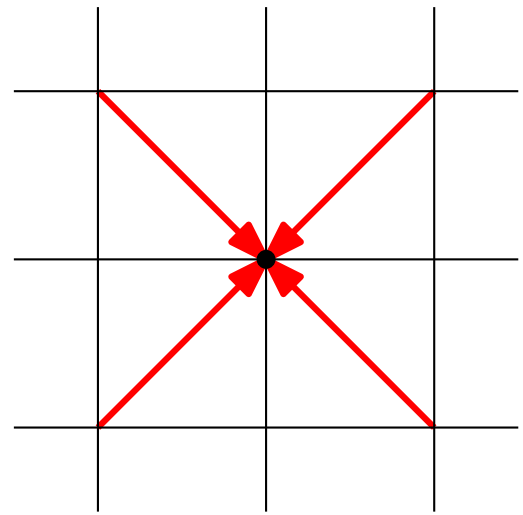
Récurrences

Soit $F(x, y; n)$ le $\#$ de marches à n pas entre $(0, 0)$ et (x, y).

On peut utiliser une récurrence pour déterminer $F(x, y; n)$ à partir de $F(0, 0; 0) = 1$.

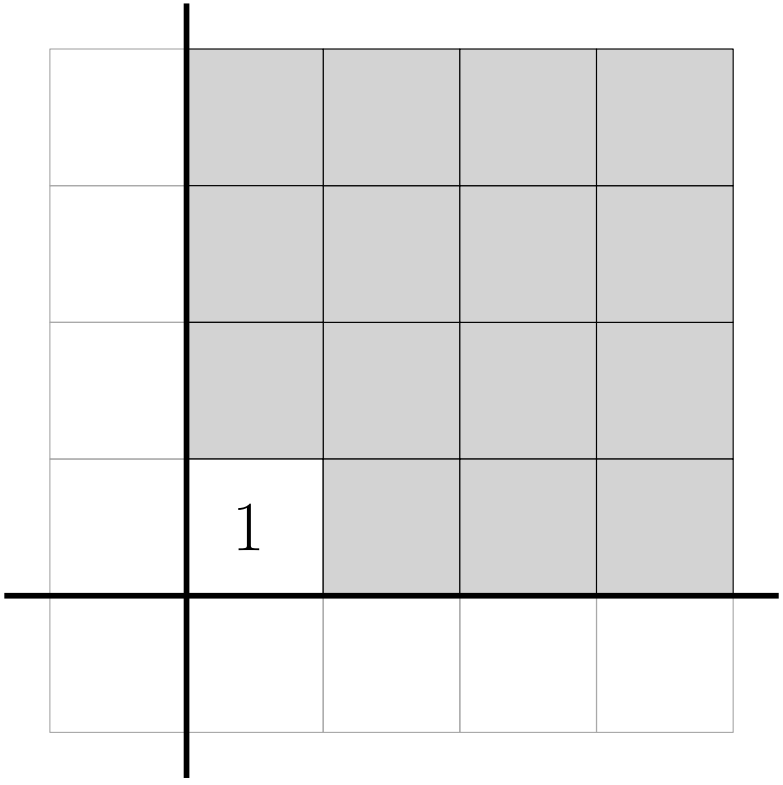
Par exemple, si $\mathcal{S} =$  alors

$$\begin{aligned} F(x, y; n) = & F(x-1, y-1; n-1) \\ & + F(x-1, y+1; n-1) \\ & + F(x+1, y-1; n-1) \\ & + F(x+1, y+1; n-1) \end{aligned}$$

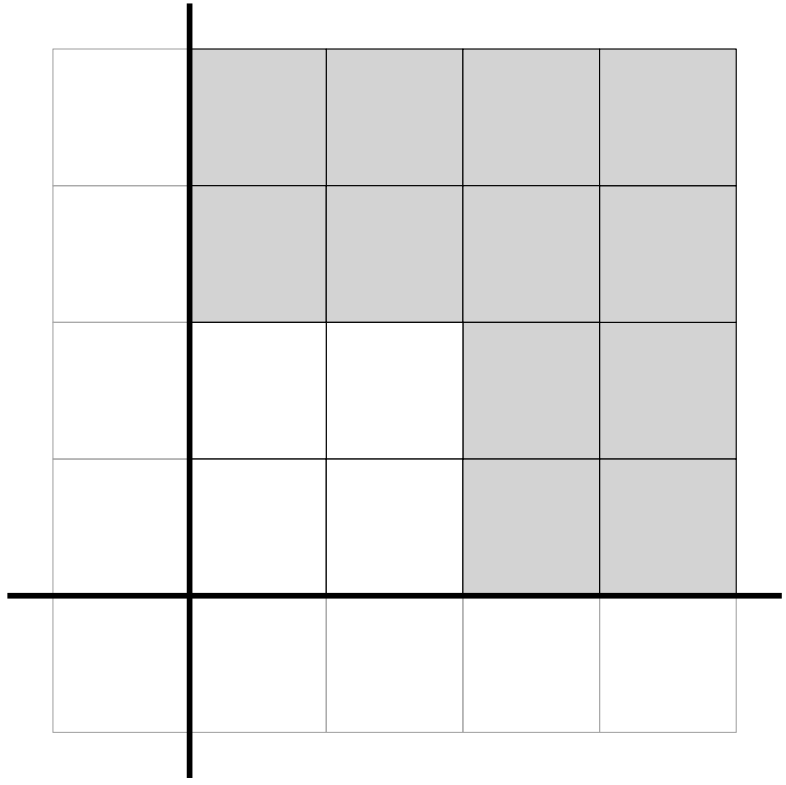


[on additionne seulement des termes d'argument positif]

Illustration de l'algorithme

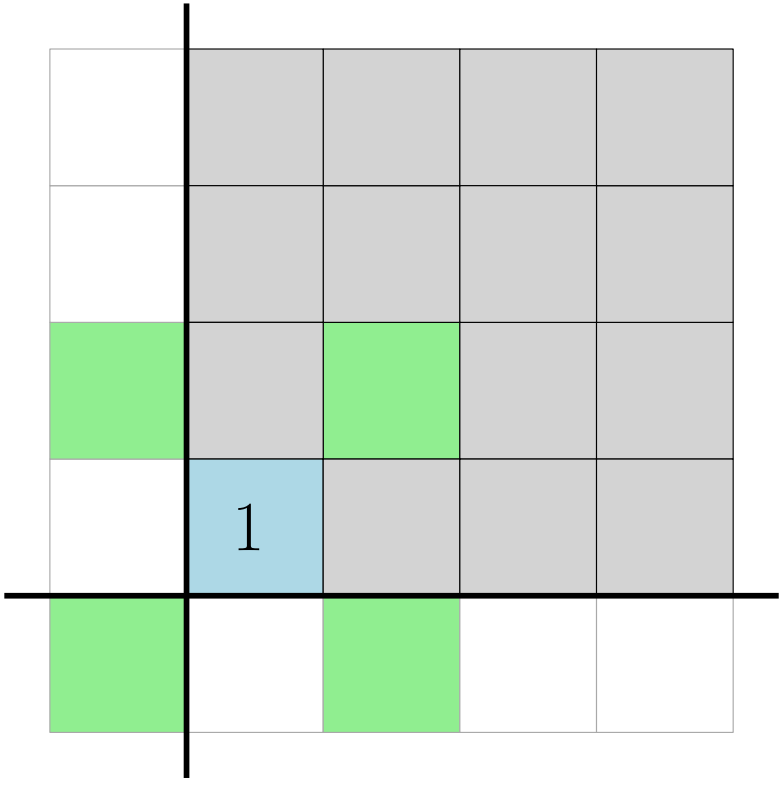


$n = 0$

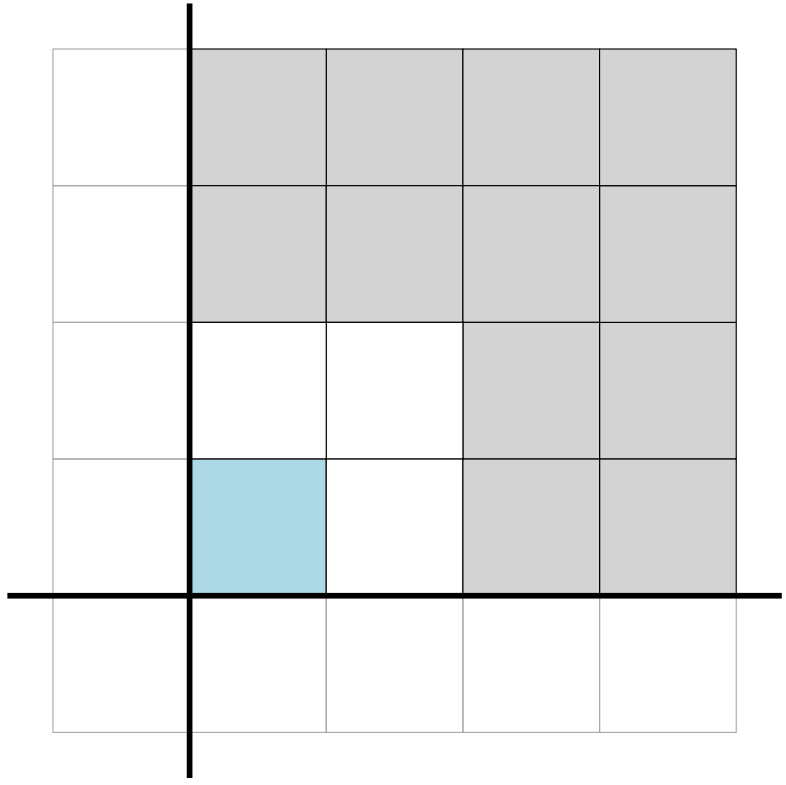


$n = 1$

Illustration de l'algorithme

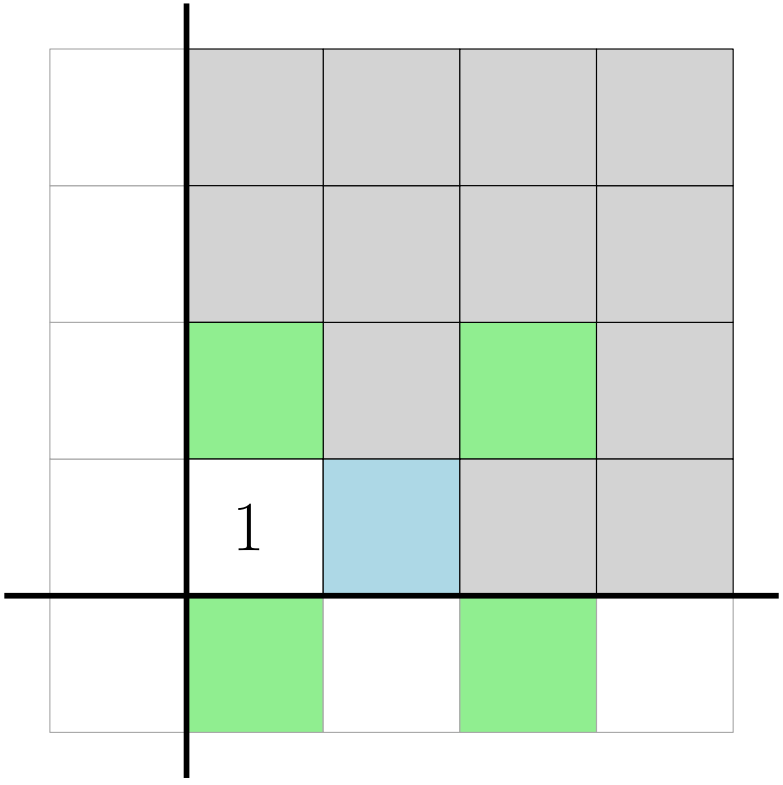


$n = 0$

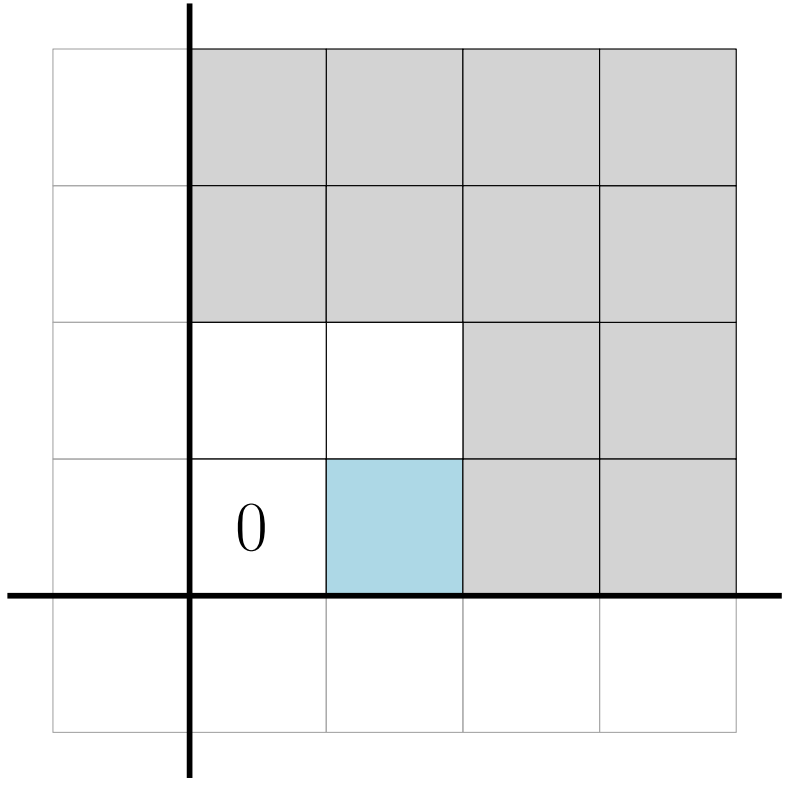


$n = 1$

Illustration de l'algorithme

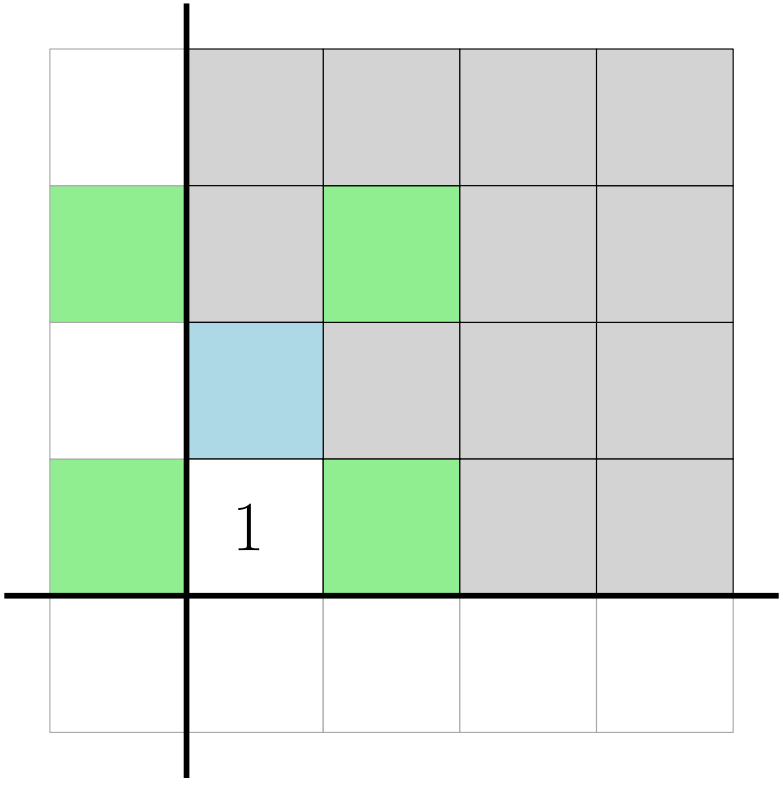


$n = 0$

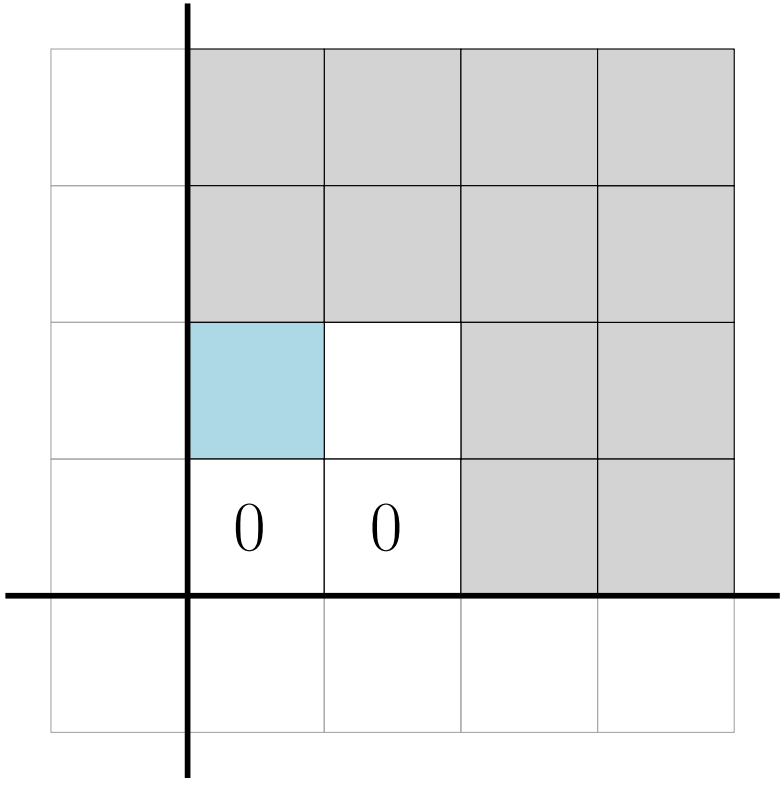


$n = 1$

Illustration de l'algorithme

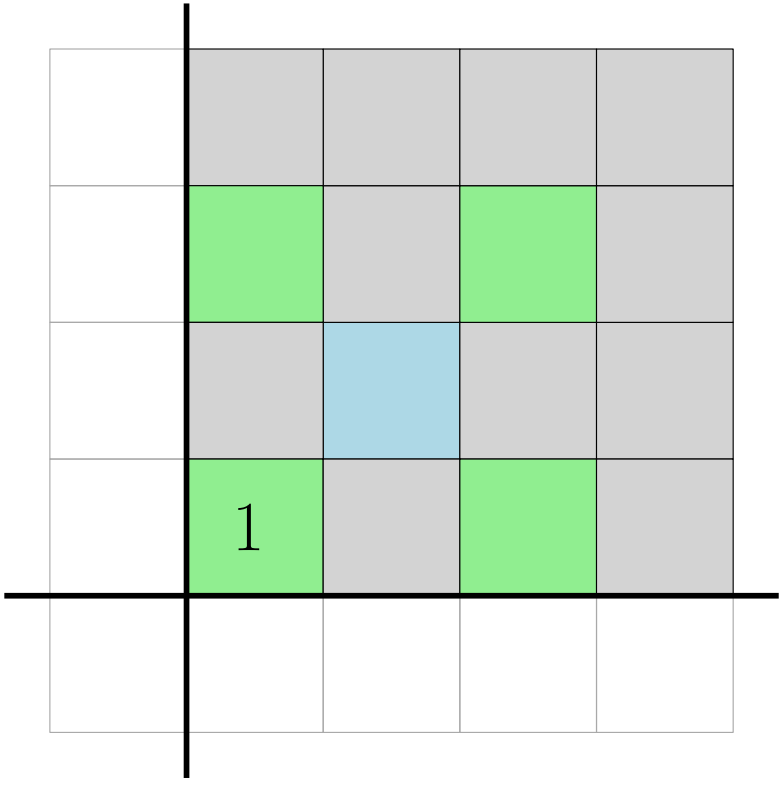


$n = 0$

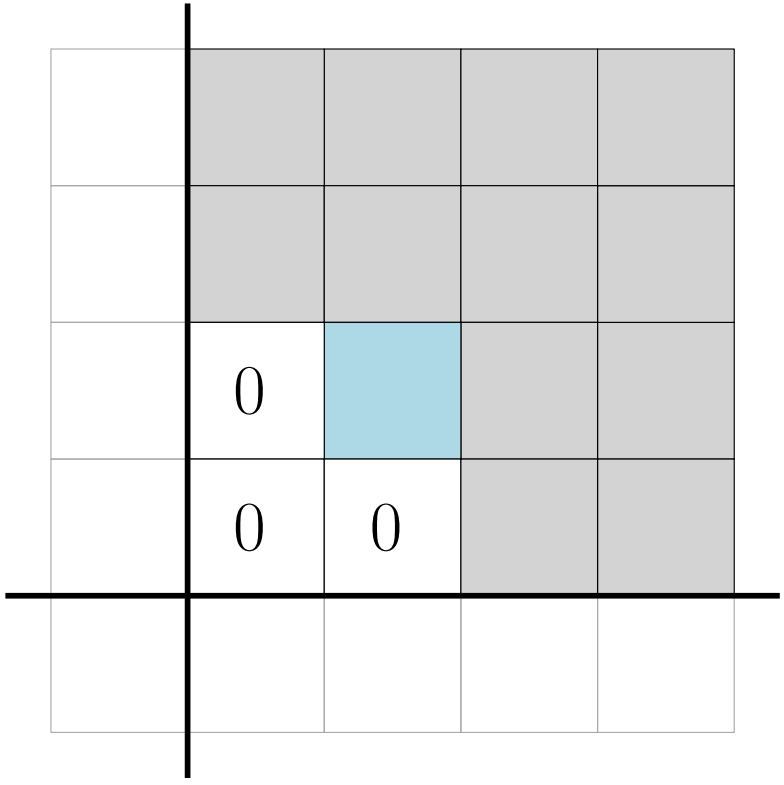


$n = 1$

Illustration de l'algorithme

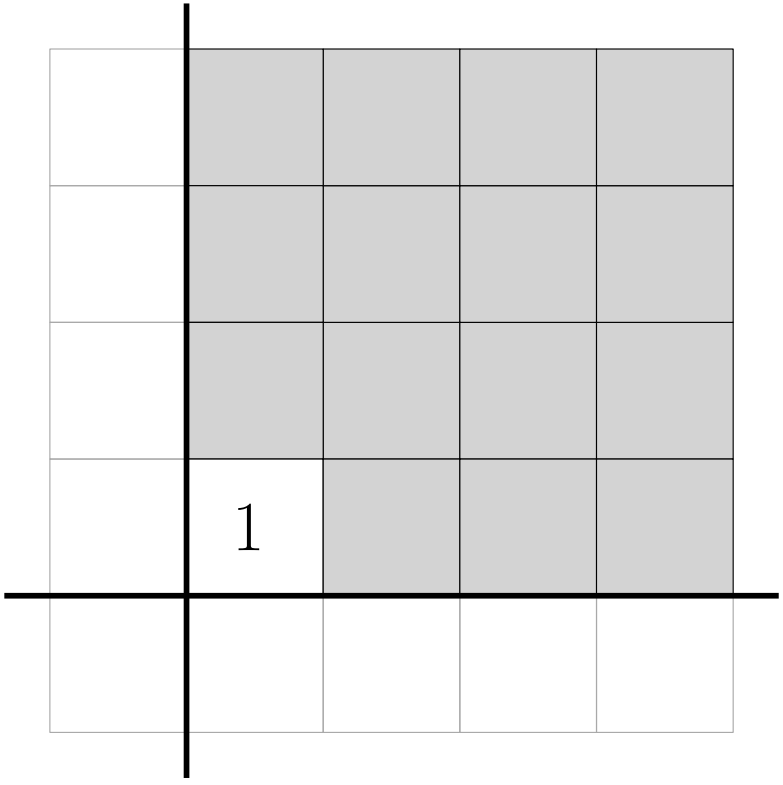


$n = 0$

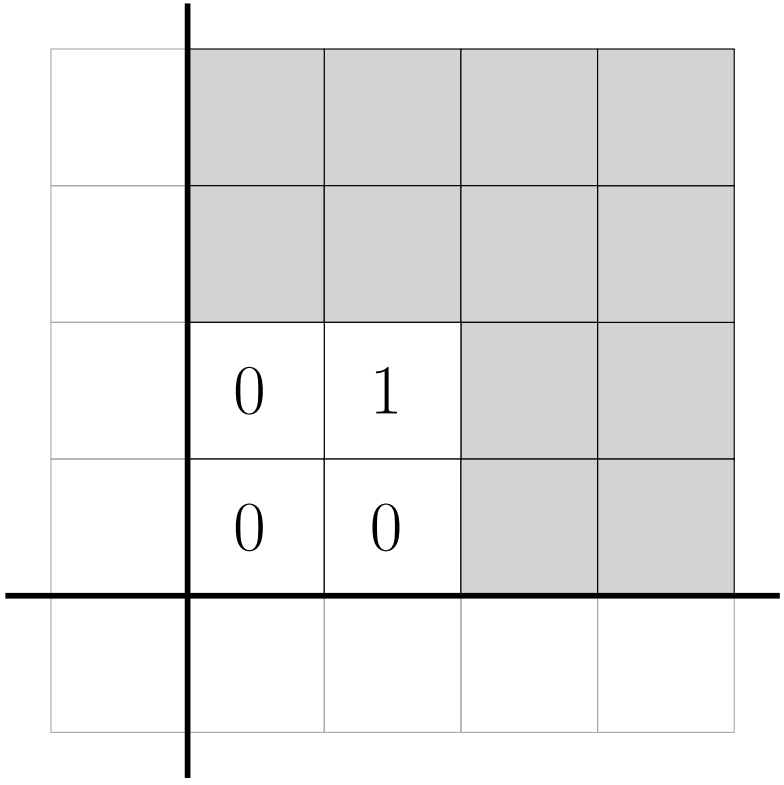


$n = 1$

Illustration de l'algorithme

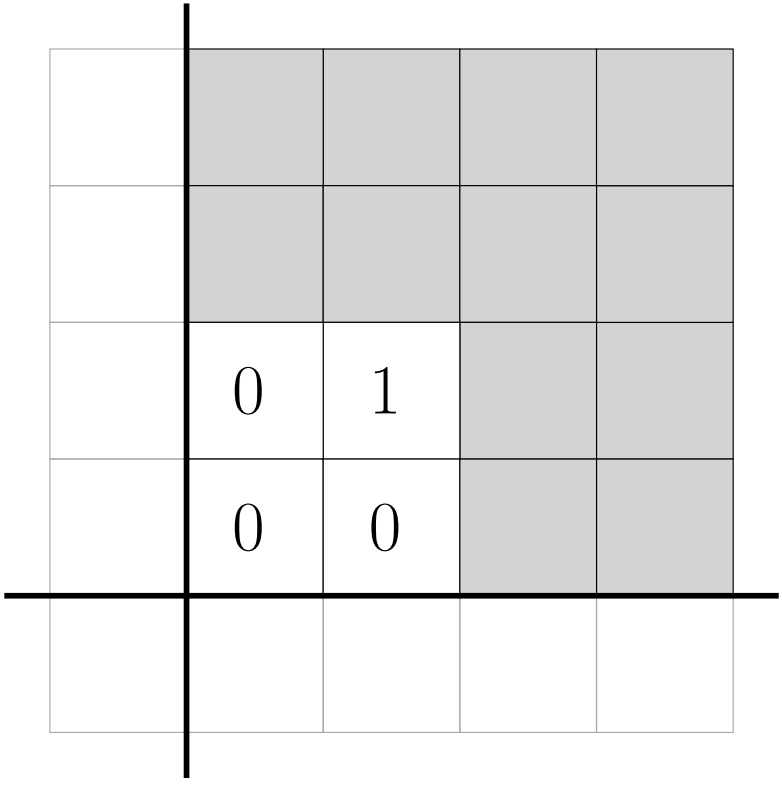


$n = 0$

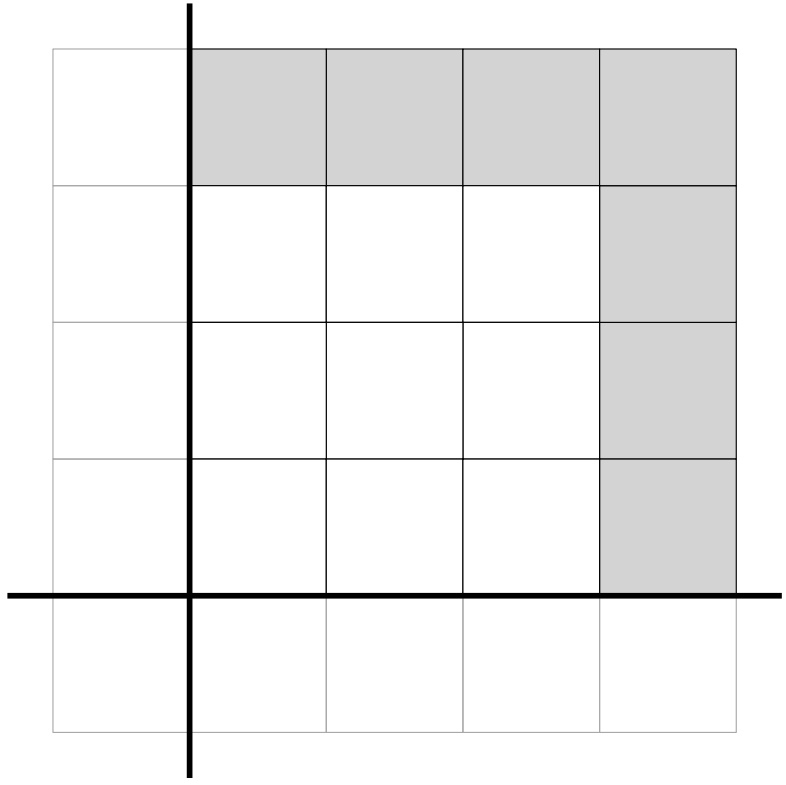


$n = 1$

Illustration de l'algorithme

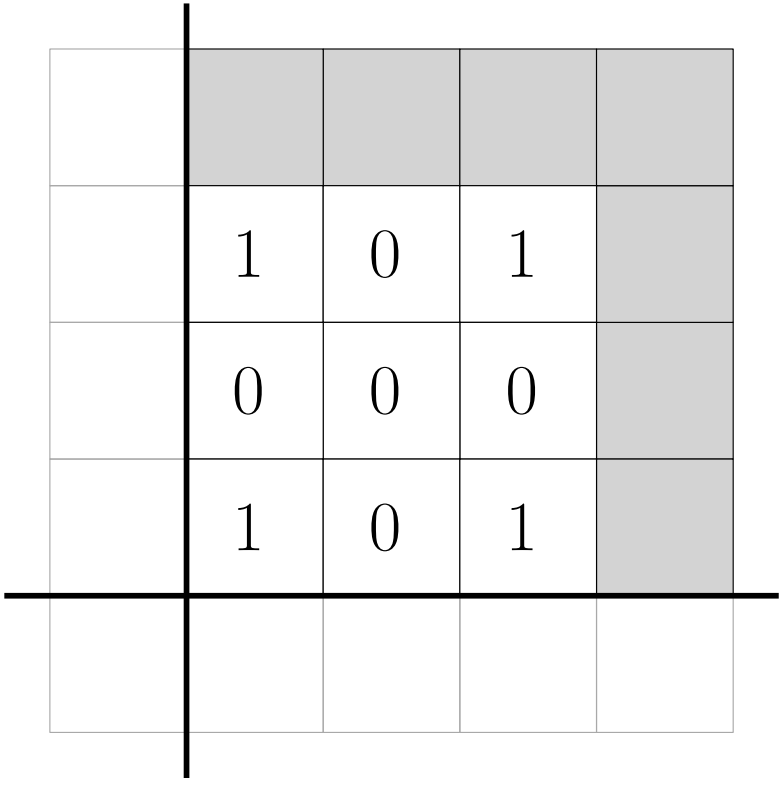


$n = 1$

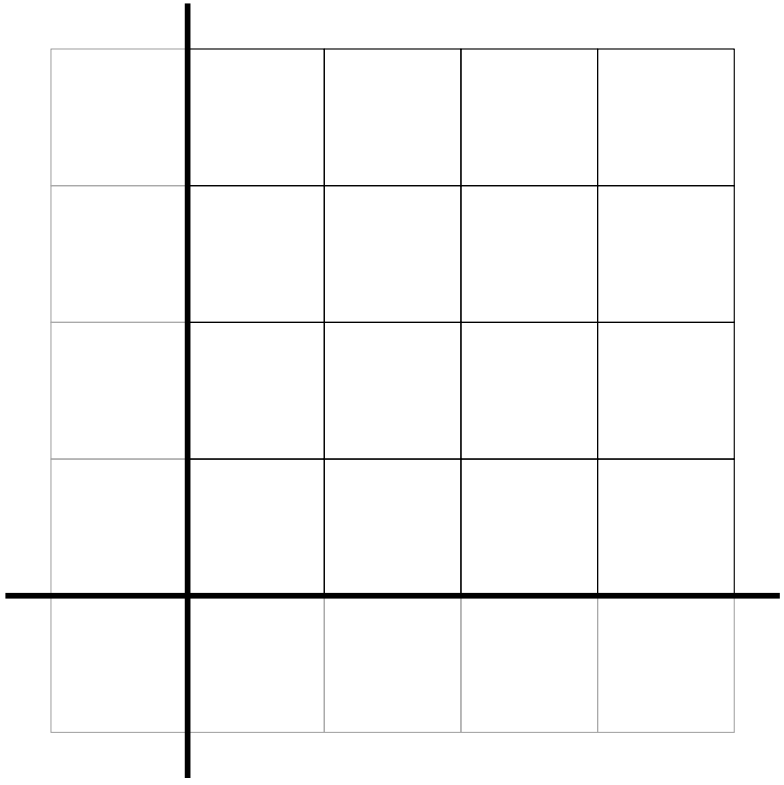


$n = 2$

Illustration de l'algorithme



$n = 2$



$n = 3$

Implémentation naïve

Algorithme 1 Implémentation naïve

Initialiser un tableau F, de taille $N \times N \times N$

pour $k = 1$ to N **faire**

  pour $i, j = 0$ to k **faire**
    Mettre à jour $F[i, j, k]$ en utilisant les valeurs de $F[\cdot, \cdot, k-1]$
  fin pour

fin pour

Une fois F calculé, on peut dénombrer les marches de longueur n en additionnant les $F[i, j, n]$ pour $i, j \leq n$.

Analyse naïve

Espace mémoire :

Le nombre de marches croit comme une exponentielle de base $|\mathcal{S}|$.

$n \log_2 |\mathcal{S}| \in O(n)$ bits sont nécessaires pour stocker le nombre de marches de longueur n.

Donc, l'algorithme utilise une mémoire de $O\left(N^4\right)$.

Analyse naïve

Espace mémoire :

Le nombre de marches croit comme une exponentielle de base $|\mathcal{S}|$.

$n \log_2 |\mathcal{S}| \in O(n)$ bits sont nécessaires pour stocker le nombre de marches de longueur n.

Donc, l'algorithme utilise une mémoire de $O\left(N^4\right)$.

Temps d'exécution :

$1^2 + 2^2 + \cdots + N^2 = O\left(N^3\right)$ opérations arithmétiques.

Cela implique une complexité binaire de $O\left(N^4\right)$.

Le problème

Cela fait trop de mémoire !

On a besoin de garder trace seulement de $F[i, j, n-1]$ et $F[i, j, n]$ à chaque moment.

Cette observation permet de descendre à $O(N^3)$ bits – c'est encore trop !

Pour économiser de l'espace, nous utilisons une **méthode modulaire**.

Implémentation Modulaire

Algorithme 2 Implémentation Modulaire

Choisir une nombre premier p_1 de 63 bits

Exécuter l'algorithme naïf modulo p_1 (en utilisant une gestion efficace de la mémoire)
Stocker le résultat dans un tableau R_{p_1}

Répéter avec plusieurs nombres premiers $p_2, \ldots, p_r$ jusqu'à ce que $\prod p_k > |\mathcal{S}|^N$

Reconstruire la suite de comptage entière à partir des R_{p_k} via le Théorème des restes chinois

Chaque R_{p_k} ayant longueur N, les restes chinois sont peu chers.

Analyse modulaire

Espace mémoire :

On doit exécuter l'algorithme pour $N \cdot \frac{\log |S|}{\log p_1}$ premiers.

À chaque exécution, deux $N \times N$ tableaux d'entiers de taille de 63 bits sont utilisés pour le calcul de R_{p_k}.

Ces tableaux peuvent être réutilisés pour chaque p_k.

Donc, l'algorithme utilise $\mathbf{O}\left(\mathbf{N^2}\right)$ opérations binaires.

Analyse modulaire

Espace mémoire :

On doit exécuter l'algorithme pour $N \cdot \frac{\log |S|}{\log p_1}$ premiers.

À chaque exécution, deux $N \times N$ tableaux d'entiers de taille de 63 bits sont utilisés pour le calcul de R_{p_k}.

Ces tableaux peuvent être réutilisés pour chaque p_k.

Donc, l'algorithme utilise $\mathbf{O}\left(\mathbf{N^2}\right)$ opérations binaires.

Temps d'exécution :

La complexité binaire modulo chaque premier est de $O(N^3)$, car on calcule en taille fixe.

On effectue $O(N)$ exécutions modulaires, pour une complexité binaire totale de $O\left(N^4\right)$.

Comparaison de l'espace mémoire

Algorithme	Naïf* (Maple)	Naïf* (C/GMP)	Modulaire (C/GMP)
500 termes 	150 MB	48 MB	4 MB
1000 termes 	328 MB	270 MB	17 MB
3000 termes 	N/A	5294 MB	142 MB
500 termes 	151 MB	50 MB	4 MB
1000 termes 	334 MB	282 MB	17 MB
3000 termes 	N/A	5635 MB	144 MB

* en utilisant l'amélioration en mémoire de $O(N^4)$ à $O(N^3)$.
L'algorithme de complexité mémoire $O(N^4)$ n'arrive à calculer que 425 termes dans les 2 cas [sur une machine à 8GB de mémoire vive.]

Comparaison des temps d'exécution

Algorithme	Naïf (Maple)	Naïf (C/GMP)	Modulaire (C/GMP)
500 termes 	319 s	10 s	36 s
1000 termes 	4054 s	123 s	671 s
3000 termes 	N/A	7576 s	59509 s
500 termes 	323 s	10 s	59 s
1000 termes 	4541 s	135 s	1029 s
3000 termes 	N/A	8515 s	89600 s

L'algorithme modulaire est plus lent, mais le vrai gain vient du passage de Maple à C.

Conclusions

Conclusions

Les marches croissent vraiment vite.

L'espace mémoire est plus contraignant que le temps de calcul.

Les méthodes modulaires apportent un gain important en mémoire, mais perdent un peu au niveau du temps.

L'implémentation (C vs. Maple) compte vraiment.

Questions ouvertes

Peut-on abaisser la complexité du calcul à $O(N^3)$?

Est-il possible d'améliorer encore la complexité si l'on s'intéresse seulement à des marches particulières ?
(e.g., aux *excusions* = marches qui retournent à l'origine)

Preuves purement combinatoires de l'asymptotique ?
(Johnson & Mishna ont déjà traité 21 cas sur 23)

Références

C. Banderier, P. Flajolet, *Basic analytic combinatorics of directed lattice paths*, 2001.

A. Bostan, M. Kauers, *Automatic classification of restricted lattice walks*, 2009.

M. Bousquet-Mélou, M. Mishna, *Walks with small steps in the quarter plane*, 2010.

I. Kurkova, K. Raschel, *On the functions counting walks with small steps in the quarter plane*, preprint, 2011.

M. Mishna, A. Rechnitzer, *Two non-holonomic lattice walks in the quarter plane*, 2009.

MERCI !