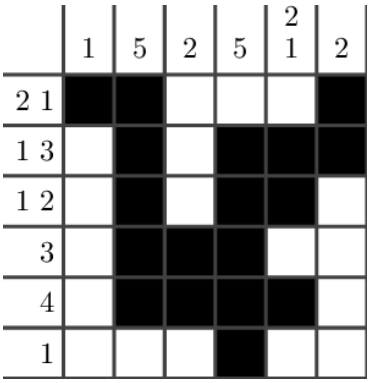
Sorbonne Université
LU3IN003 - Rapport de projet
Un problème de tomographie discrète

SAMAHA Elio, SOUAIBY Christina

Novembre 2023



Sommaire

1 Méthode incomplète de résolution

1.1 Première étape

Question 1

Pour déterminer la possibilité de colorier une ligne entière *li* avec une séquence entière, il suffit de calculer la valeur de `T(M-1,k)` qui, par définition, teste si l'on peut colorier les M premières cases de la ligne (toute la ligne) avec les k blocs de la séquence l.

Question 2

Cas 1. $l = 0$ (pas de bloc), $j \in \{0, \ldots, M-1\}$
Il n'y a aucun bloc à colorier, donc `T(j,l)` retourne `True`.

Cas 2.a. $l \geq 1$ (au moins un bloc), $j < s_l - 1$
Dans ce cas, on a plus de cases à colorier (s_l) pour notre dernier bloc que de cases disponibles ($j+1$), d'où `T(j,l)` retourne `False`.

Cas 2.b. $l \geq 1$ (au moins un bloc), $j = s_l - 1$
Dans ce cas, on a autant de cases à colorier (s_l) pour notre dernier bloc que de cases disponibles ($j+1$), d'où `T(j,l)` retourne `True` si $l = 1$ (un seul bloc, possible à colorier), `False` sinon.

Question 3

Cas 2.c. $l \geq 1$ (au moins un bloc), $j > s_l - 1$
Dans ce cas, on a plus de cases disponibles ($j+1$) que de cases à colorier (s_l) pour notre dernier bloc, donc on a 2 choix : Soit on garde une case blanche et on essaie de colorier le reste (j diminue de 1 case) avec la même séquence (on garde l) en se ramenant aux cas précédents, soit on colorie le dernier bloc de la séquence (on diminue l de 1) , on insère une case blanche et on essaie de colorier les cases restantes (j diminue de sl et 1 cases). Ce qui donne la relation de récurrence suivante : `T(j, l) = T(j - 1, l) | T(j- sl - 1, l - 1)` .

Question 4

Code

1.2 Généralisation

Question 5

En reprenant les cas des questions 2 et 3, on obtient les modifications suivantes :

Cas 1. $l = 0$ (pas de bloc), $j \in \{0, \ldots, M-1\}$
Il n'y a plus aucun bloc à colorier, donc `T(j,l)` retourne `True` s'il n'y a aucune case noire parmi les j+1 premières cases, autrement dit on n'a pas déjà colorié dans une zone où il ne faut rien colorier.

Cas 2.a. $l \geq 1$ (au moins un bloc), $j < s_l - 1$
Dans ce cas, on a plus de cases à colorier (s_l) pour notre dernier bloc que de cases disponibles ($j+1$), d'où `T(j,l)` retourne `False`, quelle que soient les couleurs des cases.

Cas 2.b. $l \geq 1$ (au moins un bloc), $j = s_l - 1$
Dans ce cas, on a autant de cases à colorier (s_l) pour notre dernier bloc que de cases disponibles ($j+1$), d'où `T(j,l)` retourne `False` si $l > 1$ (plus que s_l cases à colorier). Si $l = 1$, on vérifie qu'il n'y ait pas de cases blanches parmi les j+1 premières cases (le long du bloc à colorier). Si on en trouve au moins une, on retourne `False` (car on ne peut plus colorier cette case en noir ce qui nous empêche de former un bloc), sinon on retourne `True`.

Cas 2.c. $l \geq 1$ (au moins un bloc), $j > s_l - 1$
Dans ce cas, on a plus de cases disponibles ($j+1$) que de cases à colorier (s_l) pour notre dernier bloc. On s'intéresse alors à la couleur de la case d'indice j. 3 couleurs sont possibles :

- si la case est incolore, on cherche dans les s_l cases à colorier (indices $j - s_l + 1$ jusqu'à j) si l'une d'elles est blanche (et nous empêche de colorier le bloc).
Si c'est le cas, on décale notre bloc avant la case blanche, donc on appelle la fonction sur `T(b-1,l)` avec b l'indice de la case blanche, à condition de ne pas trouver de cases noires entre b et j (bloc mal placé), dans le cas ou on retournerait directement `False`.
Si on ne trouve pas de cases blanches sur le bloc, il faut alors vérifier si la case qui précède le bloc (indice $j - s_l - 1$) est noire. Dans ce cas, il faut obligatoirement décaler le bloc afin que la case noire en fasse partie. D'où l'appel à `T(j - 1, l)`. Si, en revanche, la case n'est pas noire, on retrouve nos 2 choix de la question 3. (`T(j, l) = T(j - 1, l) | T(j- sl - 1, l - 1)`)

- sinon, si la case est blanche, il faut décaler notre bloc afin que la case blanche n'en fasse pas partie. D'où l'appel à `T(j - 1, l)`.

- sinon, (la case est noire), on cherche dans les s_l cases à colorier si l'une d'elles est blanche (et nous empêche de colorier le bloc). Si c'est le cas, on ne peut plus décaler notre bloc avant la case blanche car on a déjà une case noire placée à la fin. Donc on retourne `False`.
Si l'on ne trouve pas de cases blanches, mais on trouve par contre une case noire avant le bloc (indice $j - s_l - 1$), on retourne `False` également, car notre bloc contiendrait une case noire en trop.
Si l'on ne rencontre aucun de ces cas, on construit notre bloc en partant de notre dernière case noire, d'où l'appel à `T(j- sl - 1, l - 1)`.

Question 6

À priori, il pourrait sembler que la complexité est exponentielle en temps, mais comme on a appliqué la mémorisation à notre algorithme, on se retrouve à juste calculer le nombre d'appels de la fonction, qui n'est autre que l'ensemble des couples (j, l) avec j allant de 0 à M-1 et l allant de 1 à k. On peut grossièrement majorer cela par M/2, donc on aura approximativement M^2 appels de fonctions. De même, pour un appel de fonction, on peut le majorer par M car on parcourt notre liste qui a une longueur de M. Donc au final, on aura une complexité en $O(M^3)$.

Question 7

Code

1.3 Propagation

Question 8

Analysons d'abord les complexités respectives des fonctions `ColoreLig` et `ColoreCol`. Pour `ColoreLig`, on parcourt la ligne, de longueur le nombre de colonnes M (resp. N pour `ColoreCol`). Pour chaque case, on effectue deux tests (blanc et noir) ce qui équivaut concrètement à 2 appels de la fonction `T(j,l)` qui, comme on l'a déjà mentionné, est de complexité en $O(M^3)$ (resp. $O(N^3)$ pour `ColoreCol`). Ce qui nous donne une complexité en $O(M^4)$ pour `ColoreLig` et en $O(N^4)$ pour `ColoreCol`.

Pour la complexité globale de l'algorithme, La boucle principale de l'algorithme est basée sur les ensembles `LignesAVoir` et `ColonnesAVoir`. Comme on a N lignes et M colonnes, la complexité de ce parcours est de l'ordre de N+M.

Dans cette boucle, nous retrouvons deux sous-parcours :
- La boucle `pour i dans LignesAVoir` de complexité $O(N)$ dans laquelle on fait appel à la fonction `ColoreLig` de complexité $O(M^4)$ dans le pire cas.
- La boucle `pour j dans ColonnesAVoir` de complexité $O(M)$ dans laquelle on fait appel à la fonction `ColoreCol` de complexité $O(N^4)$ dans le pire cas.

Dans les 2 sous-parcours, les opération sur l'ensemble `nouveaux` restent d'ordre négligeable par rapport aux autres opérations et n'ont pas de surcoût sur la complexité globale. En sortie de la boucle principale, on teste si toutes les cases sont coloriées, donc on parcourt la grille entière, ce qui se fait dans l'ordre de N*M, qui dans ce cas n'a pas de surcoût sur la complexité globale non plus.

On peut ainsi conclure que la complexité globale de cet algorithme est en $O((N+M)*(N*(M^4)+M*(N^4)))$ (ou $O((N+M)*(max(N*(M^4),M*(N^4))))$. (Une majoration grossière de la boucle principale serait de l'ordre de N*M ce qui aboutirait à une complexité globale pire cas en $O(N^2*(M^5)+M^2*(N^5))$ ou $O(max(N^2*(M^5),M^2*(N^5)))$.

Question 9

Code

1.4 Tests

Question 10

Table 1: Temps de résolution de la méthode incomplète (propagation) pour les instances 1-10

Instance	Temps de résolution (secondes)
1.txt	0.031473
2.txt	0.261886
3.txt	0.204915
4.txt	0.482180
5.txt	0.402088
6.txt	0.957009
7.txt	0.536947
8.txt	0.798724
9.txt	12.454887
10.txt	13.879982

Figure 1: Résultat de l'algorithme de résolution incomplète pour l'instance 9

Question 11

En appliquant l'algorithme à l'instance `11.txt`, on remarque que la matrice résultante est incolore. Normalement, la fonction `coloration` retourne `NESAITPAS` lorsqu'on rencontre une case qui peut être coloriée en noir ou en blanc, ce qui nous empêche de conclure sa coloration. Ici, c'est le cas de toutes les cases de la grille.

2 Méthode complète de résolution

Question 12

Cet algorithme relève de la famille des algorithmes d'énumération, où son objectif est d'explorer tous les cas possibles, qui sont ici de l'ordre exponentiel. Pour déterminer sa

complexité, il faut évaluer le nombre d'appels récursifs que l'on multiplie par la complexité d'un seul appel.

Pour le nombre d'appels récursifs, on constate que dans l'algorithme proposé en annexe, on parcourt notre grille du haut à gauche jusqu'en bas à droite. Pour chaque case de la grille, on tente de la colorier en blanc ou en noir. Lorsque les deux cas sont possibles, on énumère notre arbre des appels récursifs en deux parties. Pour chacune de ces deux parties, on répète la même procédure. Ainsi, pour chaque case, on a 2 choix possibles, et il y a $N * M$ cases, d'où un nombre d'appels de fonctions majoré en $2^{(N*M)}$ (ce cas n'est pas forcément atteint).

À chaque appel de fonction, l'opération dominante est l'appel à `colorierEtPropager`, qui a la même complexité que la fonction de coloration. Ainsi, la complexité d'un appel est en $O((N + M) \times (N * M^4 + M * N^4))$.

En multipliant les deux ensembles, on obtient une majoration en $O((N + M) * (N * M^4 + M * N^4) * 2^{(N*M)})$ pour la complexité globale. Une majoration grossière de la boucle principale serait de l'ordre de N*M ce qui aboutirait à une complexité globale pire cas en $O(2^N * M * (N^2 * (M^5) + M^2 * (N^5)))$.

2.1 Implantation et tests

Question 13

Code

Question 14

Tout d'abord, on constate que la méthode de propagation (complexité polynomiale) prend moins de temps que la méthode d'énumération (complexité exponentielle). De même, on remarque que, contrairement à la fonction de propagation, il n'y a pas d'ambiguïté quant au choix de la couleur de la case. Ainsi, on constate que l'image est bien réalisée et complète avec la méthode d'énumération, car elle énumère tous les cas possibles et choisit celui qui donne l'image voulue par des coups "légaux". En revanche, la fonction de propagation se contente de colorier en noir et blanc si l'on est sûr et certain que cette case sera de la couleur choisie. Elle laisse en gris toutes les cases pour lesquelles on pourrait éventuellement mettre du blanc ou du noir sans gêner la coloration de la ligne ou de la colonne. C'est pourquoi nous obtenons une image incomplète.

Table 2: Temps de résolution de la méthode incomplète (propagation) pour les instances 12-16

Instance	Temps de résolution (secondes)
12.txt	0.859376
13.txt	1.132367
14.txt	0.767649
15.txt	0.554078
16.txt	2.566154

Table 3: Temps de résolution de la méthode complète pour les instances 1-16

Instance	Temps de résolution (secondes)
1.txt	0.032333
2.txt	0.259577
3.txt	0.198913
4.txt	0.426279
5.txt	0.359402
6.txt	0.934311
7.txt	0.537995
8.txt	0.818465
9.txt	12.393283
10.txt	13.549418
11.txt	0.083011
12.txt	0.876161
13.txt	1.044686
14.txt	0.754309
15.txt	0.872855
16.txt	84.722423

Figure 2: Résultat de l'algorithme de résolution incomplète pour l'instance 15

Figure 3: Résultat de l'algorithme de résolution complète pour l'instance 15