LU3IN003 - PROJET
Un problème de tomographie discrète

Douar Lounes (21113545) et Serigne Fallou Fall (21125890)

November 22, 2023

Sommaire

1 Méthode incomplète de résolution

1.1 Première étape

Q1 : Il suffit de regarder si $T(M-1,k)$ donne vrai.

Q2 :

1. $1 \rightarrow$ Si $l = 0$, alors il n'y a aucun bloc à placer. Par conséquent, colorier toutes les cases en blanches constitue un coloriage possible. Donc T(j,l) donne True.
2. $2a \rightarrow$ Pour que $T(j,\ell)$ soit vrai, il est nécessaire que $j > s_\ell - 1$. Cependant, puisque dans notre cas $j < s_\ell - 1$, cela entraîne que $T(j,\ell)$ est faux.
3. $2b \rightarrow$ Supposons $j = s_\ell - 1$. Alors :
   - Si $l = 1$, nous colorions les $j + 1 = s_\ell$ premières cases, donc $T(j,l)$ est vrai.
   - Si $l \geq 2$, pour que $T(j,l)$ soit vrai, il est nécessaire que $j > s_\ell - 1$, ce qui n'est pas le cas. Donc, $T(j,l)$ est faux.

Q3 :

- Si la case (i,j) est blanche, alors $T(j,\ell)$ est vrai si $T(j-1,\ell)$ est vrai (on ajoute juste une blanche).
- Si la case (i,j) est noire, alors $T(j,\ell)$ est vrai si $T(j - s_\ell - 1, \ell - 1)$ est vrai (car on a que s_ℓ se termine à la case (i,j), et il faut donc juste regarder les cases précédentes avec ce qui reste de la séquence $(\ell - 1)$).
- On a donc la relation suivante :

$$T(j,\ell) = T(j-1,\ell) \,||\, T(j - s_\ell - 1, \ell - 1)$$

- et comme cas de base : $l = 1$ et $j = s_\ell - 1$ donne True

Q4 : il s'agit de la fonction $T1(j,l,i,s)$ dans le fichier *partie1.py*

1.2 Généralisation

Q5 :

- cas de base 1 : si $l = 0$, $T(j,l)$ = True s'il n'y a pas de case noire parmi toutes les cases précédentes.
- cas de base 2a : on a toujours $T(j,l)$ = False.
- cas de base 2b :
  - si $l = 1$ et donc $j = s_1 - 1$, alors $T(j,l)$ = True s'il n'y a pas de case blanche parmi les cases précédentes, puisque les $j + 1$ premières cases contiennent le bloc 1.
  - si $l > 1$, on a toujours $T(j,l)$ = False.
- cas de base 2c : il y a deux cas possibles.
  - la case (i,j) est blanche : il faut regarder si les l premiers blocs rentrent dans les j premières cases, c'est-à-dire qu'il faut faire un appel récursif sur la fonction pour calculer $T(j-1,l)$.

* la case (i, j) est noire : on est sur la dernière case du bloc l, donc $T(j, l) =$ True s'il n'y a pas de case blanche parmi les cases qui constituent le bloc l et si la case d'indice $j - s_l$ n'est pas noire (car si elle l'était, la case $(i, j-1)$ contiendrait la dernière case du bloc l et (i, j) ne pourrait être noire). Si ces deux conditions sont satisfaites, alors il faut vérifier s'il est possible de faire rentrer les $l - 1$ premier²s blocs dans les $j - s_l - 1$ premières cases.

Q6 : La matrice qui contient les valeurs des $T(j, l)$ est de taille $M \times k$. Or, $k \leq M$. Donc il y a au plus $M \times M = M^2$ valeurs à calculer. Pour calculer une valeur de $T(j, l)$, on fait appel aux fonctions *est_noire* et *est_blanche* qui sont de complexité $O(M)$ (dans le pire cas, on cherche une occurrence d'une valeur dans la ligne entière). Conclusion : l'algorithme est de complexité $O(M^3)$.

Q7 : il s'agit de la fonction $T2(j, l, i, s, m, memo)$ dans le fichier *partie1.py*

1.3 Propagation

Q8 La fonction *ColoreLigne* comporte une boucle *for* de M itérations, et chaque itération fait appel à la fonction *T2* de complexité $O(M^3)$, donc elle est de complexité $O(M^4)$. De même, *ColoreColonne* est de complexité $O(N^4)$.
La fonction *Coloration* comporte deux boucles *for*, l'une sur *lignesAVoir* et l'autre sur *colonnesAVoir* de tailles au plus M et N respectivement, qui sont exécutées tant que *lignesAVoir* et *colonnesAVoir* ne sont pas vides, c'est-à-dire tant que l'on peut colorier des nouvelles cases. La boucle *while* fait donc au plus MN itérations, et les boucles *for* au plus N et M itérations.
Pour la boucle sur *lignesAVoir*, on fait appel à la fonction *ColoreLig* qui est de complexité $O(M^4)$.
Pour la boucle sur *colonnesAVoir*, on fait appel à la fonction *ColoreColonne* qui est de complexité $O(N^4)$.
Conclusion : la fonction *Coloration* est de complexité $O(MN(N \times M^4 + M \times N^4)) = \boxed{O(N^2 \times M^5 + M^2 \times N^5)}$. On a donc bien une complexité polynomiale en N et M.

Q9 Il s'agit des fonctions *ColoreLigne, ColoreColonne, Coloration* dans le fichier *partie1.py*

1.4 Tests

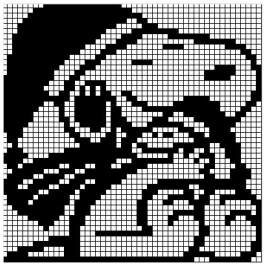
Les tests sont faits sur un Intel i5 cadencé a 1.8 Ghz

Q10 Le tableau suivant donne les temps de résolution de chacune des instances 1 à 10 avec l'algorithme de la partie 1.

instance	1	2	3	4	5
temps de résolution (s)	0.00202	0.13749	0.09646	0.24406	0.21954

instances	6	7	8	9	10
temps de résolution (s)	0.45749	0.30580	0.53451	4.50383	5.45474

La grille obtenue pour l'instance 9.txt est la suivante :



Q11 En appliquant l'algorithme à 11.txt, on constate que la couleur de chaque case de la grille est indéterminée. Ainsi, le programme ne sait pas résoudre cette instance.

2 Méthode de résolution complète

Q12 Pour la fonction *enumerationrec* on fait deux appels récursifs pour chaque case, une en coloriant la case en blanc, et l'autre en noir, et on fait appel à la fonction *ColorierEtPropager* qui est de même complexité que *Coloration* dans le pire cas. Il y a MN cases au total,la fonction est donc de complexité $\boxed{O(2^{MN}(N^2 \times M^5 + M^2 \times N^5))}$.
On retrouve donc une fonction avec une complexité exponentielle

2.1 Implantation et tests

Q13 Il s'agit des fonctions du fichier *partie2.py*. La grille obtenue pour l'instance 11.txt est la suivante :

Q14 Le tableau suivant donne les temps de résolution de chacune des instances 1 à 16 avec l'algorithme de la partie 2.

instance	1	2	3	4	5	6	7	8
temps de résolution (s)	0.00162	0.13925	0.08890	0.23515	0.21622	0.59279	0.29989	0.59080

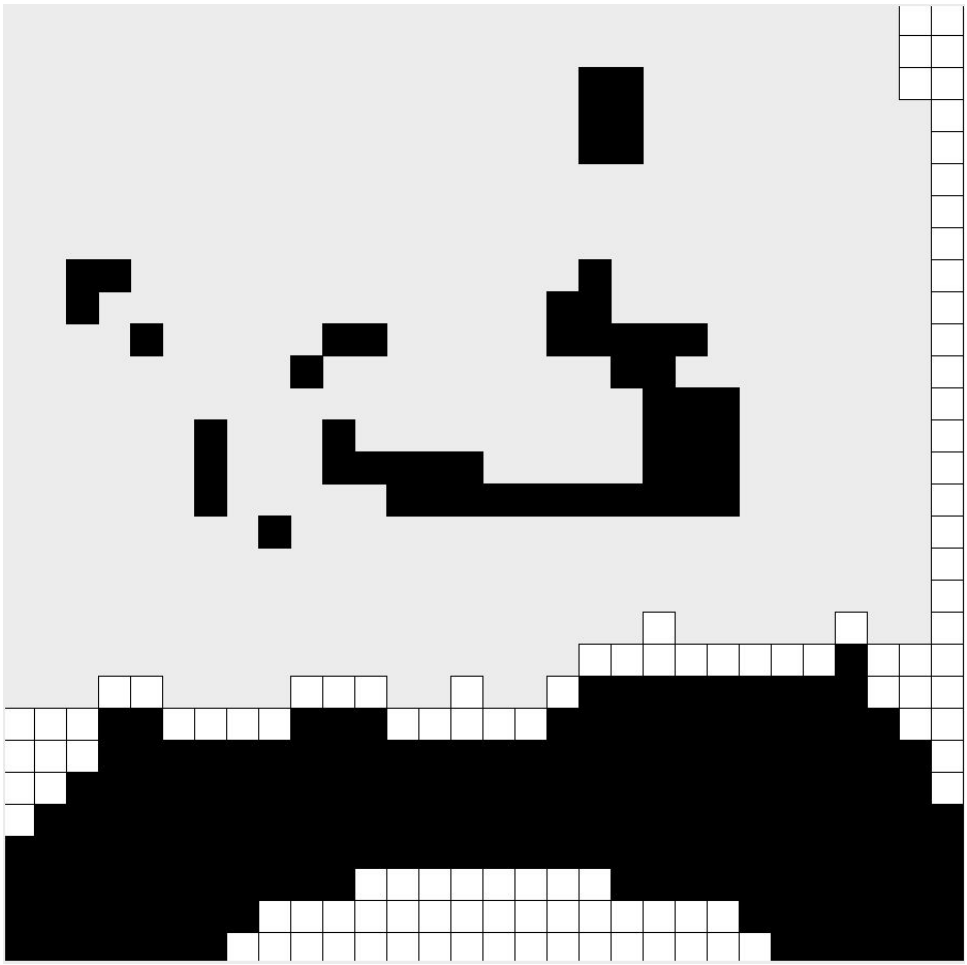
instance	9	10	11	12	13	14	15	16
temps de résolution (s)	4.68198	5.91738	0.00039	0.62217	0.78411	0.66641	0.52060	39.42336

Le tableau suivant donne les temps de résolution de chacune des instances 12 à 16 avec l'algorithme de la partie 1.

instance	12	13	14	15	16
temps de résolution (s)	0.44831	0.65775	0.40245	0.21475	0.75557

1-On remarque que les deux algorithmes résolvent complètement les instances 1 à 10 avec des temps presque pareils.
2-Pour les instances 12 à 16, l'algorithme de coloration ne parvient pas à résoudre la grille de jeu tandis que l'algorithme d'énumération y arrive avec un temps de résolution de meme ordre que les autres instances
3-Pour l'instance 16, ce dernier prend beaucoup de temps, et c'est là que l'on voit la complexité exponentielle de l'algorithme.

La grille obtenue pour l'instance 15.txt avec l'algorithme de la partie 1 est la suivante :



La grille obtenue pour l'instance 15.txt avec l'algorithme de la partie 2 est la suivante :