Un problème de tomographie discrète

1.Méthode incomplète de résolution:

1.1 Première étape:

Q1. Considérons une ligne li dont la séquence associée est (s1,s2,...sk).
On suppose avoir calculé tous les T(j,l).
Soit la ligne li entière, ce qui signifie que j=M-1, et la séquence entière, autrement dit l=k:
il est possible de colorier la ligne li entière avec la séquence entière si T(M-1,k) retourne vrai.
Il faut que la somme des si (i € {1,..,k}) et des cases blanches entre chaque bloc soit inférieure ou égal au nombre de cases de la ligne.

Q2. Valeur de T(j,l) pour les cas de base:

1. Cas l=0 (pas de bloc), $j \in \{0, \ldots, M-1\}$
   T(j,l) retourne vrai car il est possible de colorier les cases restantes en blanc

2. Cas l≥ 1 (au moins un bloc):

   (a) $j < s_l - 1$:
   T(j,l) retourne faux car il n'est pas possible de placer la séquence sl

   (b) $j = s_l - 1$:
   T(j,l) retourne vrai si l=1 (un seul bloc à placer),
   renvoie faux sinon

Q3. Relation de récurrence permettant de calculer T(j,l) dans le cas 2c:
   Si la case (i, j) est blanche alors:
   T (j, l)=T (j-1, l)
   Si elle est noire :
   T (j, l)=T ($j-S_l$ -1,l-1)

1.2 Généralisation

Q5.
Cas L= 0 :

S'il existe une case parmi les cases indexées de 0 à j coloriée en noir, alors T (j, l) retourne un False: on n'a aucun bloc S_l qui nous pousse à colorier une case en noir, on suppose alors que toutes les cases d'indice k avec k entre {0..j} seront coloriés en blanc. Renvoie True sinon.

Cas L>=1 :

j< S_l-1 :
Dans ce cas T (j, l) retourne directement Faux, nous n'avons même pas besoin de vérifier si les cases ont été pré--coloriées ou pas, car le nombre de cases (les cases indexées de 0 à j) est inférieur à la taille du bloc à colorier. En d'autres termes on nous demande de colorier un certain nombre de cases qui n'existent pas. Impossible!

j=S_L-1 :
Si l=1:
Si toutes les cases indexées de 0 à j ne sont pas coloriées en blanc (elles peuvent être vides ou noires): alors le nombre de cases qu'on peut colorier est de même taille que le bloc S_L T (j, l) retourne donc Vrai. Retourne False sinon.

Si l> 1:

Il y a d'autres blocs à colorier, mais il ne restera aucune case disponible qu'on puisse colorier. La coloration est donc impossible et T (j, l) retourne Faux.

j>S_L-1 :
Si la case est blanche: on fait un appel récursif sur (0,j-1) : est-il toujours possible de placer la séquence sur le reste de la ligne?
Si la case est noire: on vérifie que la case (j-S_L) séparant deux blocs ne soit pas noire, sinon renvoie False. On vérifie également qu'il n'y a aucune case blanche là où on veut placer notre séquence, sinon renvoie False. Si aucun de ces deux cas ne se présente on fait un appel récursif sur j-S_L-1 et l-1: comme si on avait placé notre séquence, on tente de voir s'il est possible de placer les autres séquences sur le reste de la ligne.

Si la case est vide, il suffit de la considérer en blanc (resp en noir) et de faire un appel récursif sur j-1 (resp j). Renvoie False si aucun des cas n'est possible, True sinon.

Q6. Complexité en fonction de M de l'algorithme:

Pour chaque appel de fonction T(j,l) on effectue au plus j*l calculs, j pouvant atteindre au pire cas la valeur M=> complexité en O(M).
Pire cas: par exemple placer [1] et j>Sl-1 (soit une ligne de M cases environ): on peut mettre en blanc toutes les cases mis à part j=0 où il est nécessaire de la considérer noire. On teste alors T(j,l) sur toute la ligne: M cases.
Complexité pire cas en : $O(M^2)$

Q7. Notre fonction dépasse la complexité attendue en raison de l'appel récursif effectué à deux reprises (complexité exponentielle). On calculera dans la suite de ce rapport les complexités en fonction de la question 6 et non pas celle du code qui peut être amélioré.

1.3 Propagation

Q8.
On essaye ici de colorier le maximum de cases à chaque fois, c'est-à-dire lorsqu'on parcourt les lignes i on essaye de colorier un maximum de cases de cette ligne, et on stocke les colonnes j dont on a pu colorier la case (i, j) afin de les reparcourir.
La boucle « pour i dans LignesAVoir » est en 0(N) et à l'intérieur se trouve la fonction ColorLig qui est en est en 0(M3) (car on a une boucle j allant de 0 a M-1 et a l'intérieur la fonction de la partie1.2 qui est en 0(M2) pour une ligne) ce qui implique que la complexité de la boucle est en $0(N.M^3)$

Puis on a la boucle « pour j dans ColonnesAVoir » est en 0(M) et a l'intérieur se trouve la fonction ColorCol(pour nous toujours colorligne) qui est en est en 0(N3) (car on a une boucle j allant de 0 a N-1 et a l'intérieur la fonction de la partie1.2 qui est en 0(N2) pour une colonne) ce qui implique que la complexité de la boucle est en $0(N^3.M)$

Donc à l'intérieur de la boucle tant que, on a une complexité de 0 $(NM^3 + N^3M)$.

La première boucle quand a elle est en 0(NM) car on suppose son arrêt quand toutes les cases ont été coloriées (pire cas).

Donc la complexité de l'algorithme est en $0(N^2M^4 + N^4M^2)$: il s'agit bel et bien d'une complexité polynomiale.

Q10.

Instance	Temps de résolution
0	0.002 secondes
1	0.004 secondes
2	0.8 secondes
3	1.8 secondes
4	5 secondes
5	1.07 secondes
6	6.97 secondes
7	4.13 secondes
8	9.06 secondes
9	Environ 15 minutes
10	Environ 30 minutes

Grille obtenue pour instance 9:

Q.11
On remarque que lors de l'exécution du programme sur l'instance 11.txt l'algorithme de résolution ne retourne pas vrai mais «Je ne sais pas». En effet, la méthode appliquée est incomplète, l'algorithme ne peut pas résoudre ce cas ci.

2. <u>Méthode complète de résolution:</u>

Q12. Dans Enumeration il y a un appel à coloration dont la complexité déduite précédemment est en $0(N^2M^4 + N^4M^2)$. De plus, Enumeration appel Enum_rec à deux reprises : une fois en coloriant la case en Blanc et une autre fois en coloriant la case en noir.
Concernant la complexité de Enum_rec:
On fait appel à color_propage
la boucle while de complexité pire cas O(NM)
A deux reprises : appel récursif sur enum_rec en coloriant en blanc puis en noir.
Concernant la complexité de color_propage: pareil que celle de propagation
$0(N^2M^4 + N^4M^2)$. On a donc pour k=M*N:

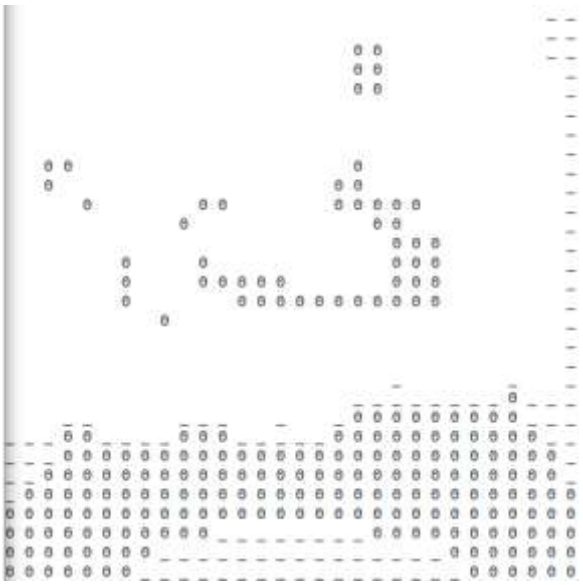
$0((N^2M^4 + N^4M^2)+(N^2M^4 + N^4M^2)*2^k)=>O((N^2M^4 + N^4M^2)*2^k)$

Instances	Temps de résolution
12.txt	2.27 secondes
13.txt	3,11 minutes
14.txt	2.30 secondes
15.txt	5.59 secondes
16.txt	

En raison de la taille de la grille et complexité élevée de notre algorithme, l'instance 16.txt met plus de 2h à s'afficher.

<u>Grille 15.txt avec les deux méthodes:</u>

Méthode 1:



Méthode 2: