LU3IN003- Algorithmique II

Un problème de tomographie discrète

	1 1	1	1 2	1	2
3					
1 1 1					
3					

	1 1	1	1 2	1	2
3	■	■	■		
1 1 1	■		■		■
3			■	■	■

THURAIRAJAH Shaithan 3802984,
ABITBOL Yossef 3804139,
Décembre 2020.

I. Méthode incomplète de résolution

1.1: Première étape)

(Q1)

Soit k tels que S_k est la séquence entière de la ligne li, alors il est possible de colorier la ligne li si :

$\exists\ j$ tq $T(j,k)$est vraie

(Q2)

Pour le 1er cas ($l = 0$) :

On colorie la ligne entière en blanc.

$T(j,l)$ = Vrai

Pour le 2eme cas ($l \geq 1$) :

Pour (2.a) $j < sl - 1$:

Le bloc est plus grand que le nombre de case donc on peut pas colorier les $j + 1$cases avec le bloc sl

$T(j,l)$ = Faux

Pour (2.b) $j = sl - 1$:

$T(j,l)$ = Vrai si sl est l'unique bloc de la séquence, sinon Faux

(Q3)

Pour le dernier cas, on distingue **2 possibilitées** :

1) Le cas où **la case (i,j) est blanche** et donc on décrémente j de 1.

2) Le cas où **la case (i,j) est noire**, on décrémente j du nombre de cases sl et d'une case blanche.

vu que les 2 cas sont possibles, on fait un ou logique et on obtient:

j' = j - sl -1 et l' = l -1

j'' = j -1 et l' = l

$T(j,l) = T(j',l')\ ou\ T(j'',\ l'')$

1.2: Généralisation)

(Q5)
Soit sl le nombre de case du **DERNIER** bloc

Pour le 1er cas ($l = 0$) :

Si il existe une case colorier en noir dans la ligne
Alors on retourne **Faux**
Sinon on retourne **Vrai**

Pour le 2eme cas ($l \geq 1$) :

Pour (2.a) $j < sl - 1$:
Le bloc est plus grand que le nombre de case donc on peut pas colorier les $j + 1$cases avec le bloc sl et même avec des cases déjà colorier la réponse reste :
$T(j, l)$ **= Faux**

Pour (2.b) $j = sl - 1$:
Si sl **n'est pas l'unique bloc** de la séquence
Alors on retourne **Faux**
sinon
Si il existe au moins **une case colorié en blanc dans la ligne**, on ne peut pas poser notre unique bloc
Alors on retourne **Faux**
Sinon on retourne **Vrai**

Pour le dernier cas, on distingue 3 possibilitées :

(1) : Le cas où **la case (i,j) est blanche**,
(2) : Le cas où **la case (i,j) est noire**,
(3) : le cas où l**a case (i, j) est vide**

Pour (1), on fait un fait un appel récursif en **décrémentant j de 1**
Pour (2),
Considérons sld **correspondant aux slderniers cases de la ligne**, dans la suite
si **les sld sont noirs ou vides**, **et que sl est l'unique bloque de s et le reste des cases antécédents sont blanches ou vide**
Alors on retourne **vrai**
OU

Si il **existe une case précédent les sld et qu'elle est vide ou blanche**

On **appelle récursivement T en enlevant sl et les sld privées de la case vide ou blanche**.

Pour **(3)**, les deux cas au dessus sont possibles donc on teste un ou logique entre **(1)** et **(2)**

(Q6)

Nombre d'appel de T(j, l) :

- **En ne prenant pas en compte le tableau dynamique :**

Avec **sl = 1** :

Donc on rentre dans le pire cas ou la case (i,j) est vide :

On calcule le nombre d'appel grâce à la suite u(n)

u(n) = 2 + u(n-1) + u(n-2)
u(n) = 2 + (2 + u(n-2) + u(n-3)) + (2 + u(n-3) + u(n-4))
u(n) = 2 + 2 + 2 + u(n-2) + 2*u(n-3) + u(n-4)
u(n) = 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + u(n-3) + 3*u(n-4) + 3*u(n-5)) + u(n-6)

.....

u(n) = 2(2^n -1)

Du ce fait, on obtient de l'ordre de 2^M nombre d'appel de T(j,l) sans le tableau dynamique

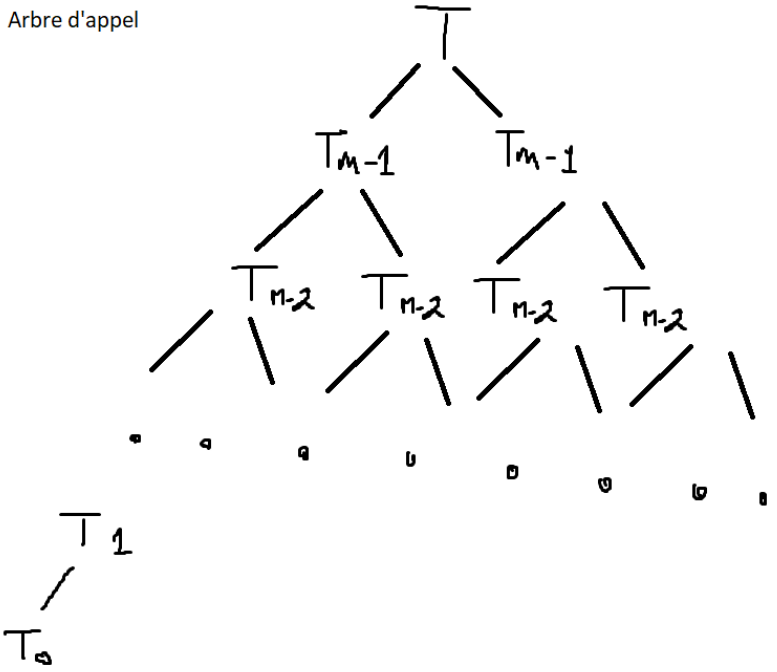
On multiplie ces appels par le temps de calcul d'un T(j,l), qui correspond dans le pire des cas au cas de (i,j) une case vide. Considérons le parcours de la ligne :

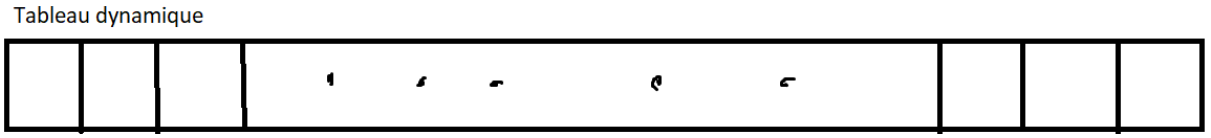
Dans ce cas, on a une complexité en O(M) car on va parcourir toute la ligne pour voir si le bloc est coloriable.

Par conséquent, on a une complexité : $O(2^M * M)$

- **En prenant en compte le tableau dynamique :**

Pour comprendre l'impact du tableau dynamique, prenons un cas plus simple où **T(M) = 2 + 2*T(M-1) qui correspond aussi à une complexité de** $O(2^n)$





Considérons l'arbre d'appel et le tableau dynamique en parallèle:

Lors de la première appelle à T, on appelle d'abord le fils gauche T(M-1) et ensuite T(M-1).

On va ainsi parcourir les branches les plus à gauche de l'arbre.

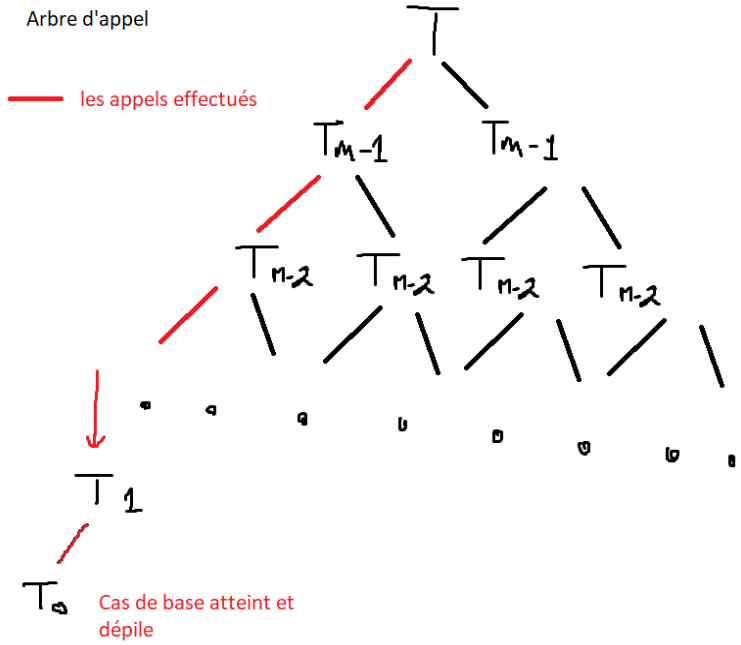


Tableau dynamique

En dépilant, à T(1), on va appeler T(0) une nouvelle fois mais sa valeur est déjà dans le tableau dynamique c'est donc en temps constant et T1 est rajouté au tableau dynamique.

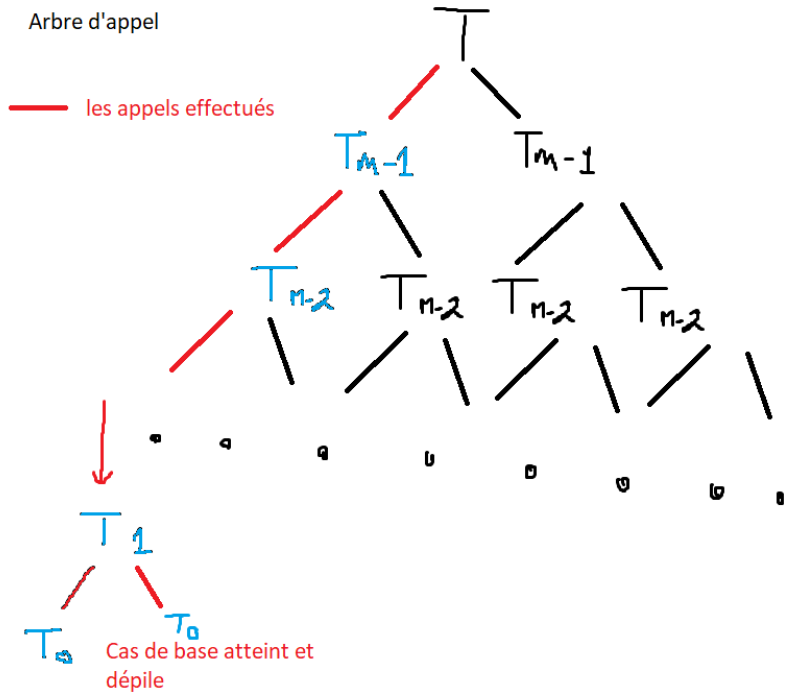


Tableau dynamique

On va effectuer ceci jusqu'à T(M-1)
Ce qui veut dire qu'on va remplir le tableau jusqu'à la M-1 ième case.

Finalement, pour trouver T(M), on a exécuté que les appelles à gauches de l'arbre.

Ceux qui correspondent à une complexité en **O(M),** on remarque ainsi que **la complexité du nombre d'appels de T(j,l) avec le tableau dynamique correspond au nombre d'état unique dans T(j,l) ce qui est équivalent dans le pire cas aux nombres de combinaison unique pris par le couple (j,l).**

Rappelons que **j est la ligne contenant M élements** et **l la séquence de la ligne contenant K éléments.**
Il y a donc **K*M** combinaisons de différentes de (j,l) possibles.

Essayons d'**exprimer K en fonction de M,** dans le pire cas, chaque bloc de la séquence est de taille 1. On veut donc colorier la ligne avec une alternance de cases blanches et noires. On aura donc (M/2 + ou - 1) blocs dans la séquence. On fixe alors **K = M/2.**

Finalement, avec le tableau dynamique, on aura de l'ordre de M^2 appels à T(j,l). On a donc en multipliant par la complexité de T(j,l):

On a donc la complexité de T(j,l) qui est en $O(M^3)$

1.3: Propagation)

(Q8)

Dans coloration, on va parcourir les N lignes et les M colonnes et dans le pire cas, on devra parcourir ses lignes et ses colonnes respectivement M et N fois.

Pour chacune de ses lignes et colonnes, on fait appel à respectivement ColoreLigne et ColoreColonne qui parcourt les cases de ses colonnes et lignes.

Pour chacune de ses cases, on va tester si elle peut être blanche ou noire, ce qui est équivalent à faire 2 appels à T(j, l) dont on avait trouvé la complexité plus haute.

On a donc $O((M * N * 2 * O(N^3)) + (N * M * 2 * O(M^3)))$

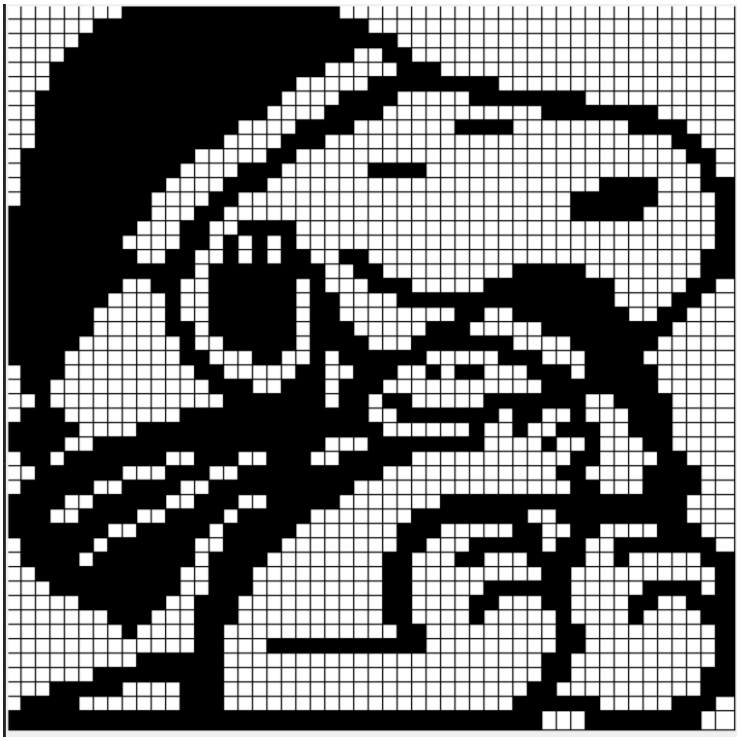
ainsi $O(M * N^4 + N * M^4)$ ou encore $O(max(M * N^4, N * M^4))$

1.4: TESTS)

(Q10)

Numéro de l'instance	temps cpu (s)
1	**0**.001205
2	**0**.17526
3	**0**.13608
4	**0**.25887
5	**0**.3248
6	**1**.079383
7	**0**.4523928
8	**0**.863333
9	**10**.98985
10	**11**.9745347

Instance 9 :



(Q11)

Dans l'instance 11, Colorisation va d'abord essayer de remplir les lignes :

Il va essayer de placer la séquence [2] dans une ligne de 4 cases, il y a plusieurs cas possibles ainsi il ne va rien faire et passer à la ligne suivante,

Il va essayer de placer la séquence [1, 1] dans une ligne de 4 cases. De même, il ne va pas réussir à la placer et passer maintenant aux colonnes.

Pour chaque colonnes aussi, il ne trouvera pas de moyen pour placer une séquence [1] dans une colonne de 2 cases.

On se retrouve alors devant un nonogramme qui ne peut pas être colorié avec Colorisation.

2. Méthode complète de résolution

(Q12)

Etudions d'abord, la complexité de ColorierEtPropager :

Dans le pire cas, ColorierEtPropager à la même complexité que Coloration soit $O(max(M * N^4, N * M^4))$

Etudions maintenant, la complexité de Enum_rec:

Toujours dans le pire cas,
on va parcourir récursivement M*N cases:
pour chaque case, on va appeler ColorierEtPropager,
ensuite, on appelle récursivement la prochaine case vide, ici, k+1 avec la couleur blanche
ensuite avec la couleur noire

On note C = Max(M*n^4, N*M^4)

U(n) = C + 2*(u(n+1))
= C + 2*(C+2*(u(n+2))
= C + 2* C + 4*(C + 2* u(n+3))
= $\sum_{k=0}^{M*N} C * 2^k = C * \sum_{k=0}^{M*N} 2^k = C * (2^{M*N+1} - 1)$

Étudions finalement, la complexité de Enumeration:
Enumeration fait un appel à Coloration, qu'on sait de complexité $O(max(M * N^4, N * M^4))$
Dans le pire cas, on fera appel 2 fois à Enum_rec avec la case 0 initialement coloriée en blanc ou noir.

Ainsi la complexité de Enumération est de :

$O(max(M * N^4, N * M^4)) + O(max(M * N^4, N * M^4)) * (2^{M*N+1} - 1))$
= $O(max(M * N^4, N * M^4)) * (1 + (2^{M*N+1} - 1))$
= $O(max(M * N^4, N * M^4)) * (2^{M*N+1})$

La complexité de Enumeration est donc en $O(max(M * N^4, N * M^4) * 2^{M*N+1})$

2.1: Implantations et tests)

(Q14)

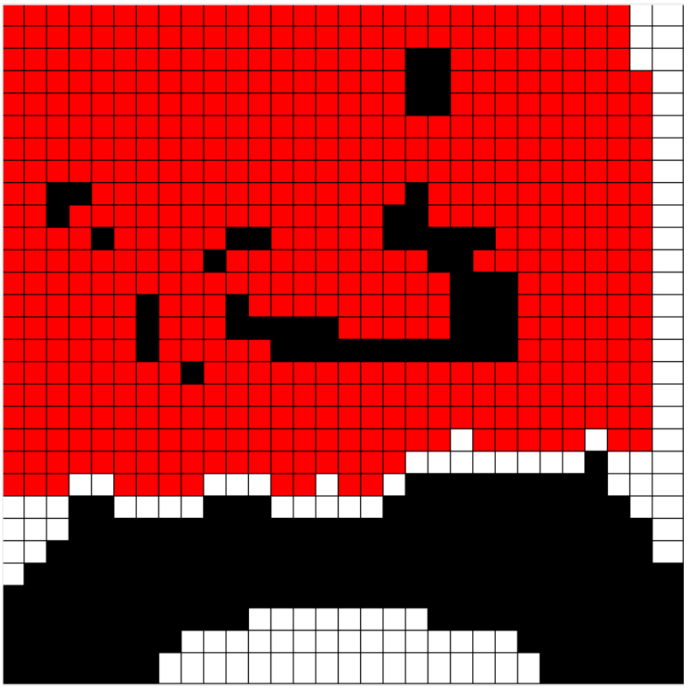
Numéro de l'instance	temps cpu (s)
1	0.0014349
2	0.2214088
3	0.1377765
4	0.2794528
5	0.3279917
6	1.1065541
7	0.4636166
8	0.8630633
9	11.2332032
10	11.9137441
11	0.000547
12	0.749827
13	0.9231958
14	0.8970798
15	0.7247422
16	60.36257

En appliquant la méthode de la section 1 aux instances 12 à 16, l'algorithme retourne des nonogrammes partiellement coloriés et "NeSaitPas" en sortie.

Instance 15 :

Les cases en rouge représentent des cases vides.

Avec coloration (section 1):



Avec Enumeration (section 2) :

Discussion par rapport à la programmation :

Nous avons choisi **Python 3** pour plusieurs raisons :

- Langage connu des deux binômes
- Pas de gestion de mémoire
- Rapidité de programmation
- Débogage facile
- Facile d'installation sur UNIX et Windows
-

Cependant, il y a quelques problèmes :

- Assez lent par rapport à d'autres langages
- utilise plus de mémoire que d'autres langages de prog
- Vitesse significativement différente entre Unix et Windows

Spécificité de Python3 important:

- Booléen : Python utilise l'évaluation "paresseuse" des booléens.

- DeepCopy : En faisant un deepcopy, on ne copie pas seulement la grille de nonogramme mais la séquence et la grille transposée correspondant aux colonnes. C’est donc une opération très coûteuse.