Un problème de tomographie discrète

Projet réalisé par Sira-Lina Achouri
et Ina Elena Campan

Étudiantes en troisième année de la double licence
Mathématiques-Informatique (Parcours PIMA)
de Sorbonne Université

Groupe 4, Olivier Spanjaard

LU3IN003 - Algorithmique II

Année universitaire 2023 - 2024

Table de matières

1 Introduction

1.1 Présentation du sujet

Le projet a été réalisé dans le cadre de l'unité d'enseignement LU3IN003, en binôme. Le sujet est disponible à l'adresse suivante :
`https://moodle-sciences-23.sorbonne-universite.fr/course/view.php?id=1553`.

Ce rapport reprend les 14 questions du sujet. Il présente les solutions que nous avons proposées, accompagnées de démonstrations mathématiques et de figures qui portent sur l'efficacité des algorithmes proposés.

1.2 Choix du langage de programmation

Nous avons choisi le langage Python [1], afin de manipuler des classes d'objets appropriées à la résolution de notre problème, mais aussi pour un affichage précis des grilles obtenues.

2 Méthode incomplète de résolution

2.1 Première étape

Question n°1

Si l'on a calculé tous les $T(j,l)$, comment savoir s'il est possible de colorier la ligne l_i entière avec la séquence entière ?

On peut colorier la ligne en entier si et seulement si $T(j,k)$ prends la valeur **vrai** pour un certain $j \in \{1,...,M-1\}$ (ici, $l=k$).

Question n°2

Pour chacun des cas de base 1, $2a$ et $2b$, indiquez si $T(j,l)$ prend la valeur vrai ou faux, éventuellement sous condition.

- Pour le premier cas, $T(j,l)$ prend la valeur **vrai** : on colorie tout la ligne en blanc, car la séquence de blocs est vide.
- Pour le deuxième cas, on suppose que $l > 0$.
  Pour le point a, $T(j,l)$ prend la valeur **faux**, car $j+1 < s_l$ implique que la séquence est plus longue que le nombre de cases à colorier.
- Pour le point b, $j+1 = s_l$ implique que $T(j,l)$ prend la valeur **vrai** si $l = 1$: on colorie les j+1 cases en noir. Sinon, $T(j,l)$ prend la valeur **faux**.

Question n°3

Exprimez une relation de récurrence permettant de calculer $T(j,l)$ dans le cas $2c$ en fonction de deux valeurs $T(j',l')$ avec $j' < j$ et $l' \leq l$.

On établie la relation de récurrence suivante :

$$\boxed{T(j,l) = T(j-1,l) \text{ ou } T(j-s_l-1,l-1)} \quad (1)$$

La formule (1) se traduit de la manière suivante :
pour colorier les $j+1$ premières cases où $j+1 > s_l$ il faut :

- soit pouvoir colorier les j première cases avec la sous-séquence $(s_1,...,s_l)$, si la case $j+1$ est blanche ;
- soit pouvoir colorier les $j-s_l-1$ cases avec la sous-séquence $s_1,..,s_{l-1}$, si la case $j+1$ est noire.

La formule (1) prend la valeur **vrai** si l'une de ces deux composantes vaut **vrai**.

Question n°4

Codez l'algorithme, puis testez-le.

La fonction demandée pour calculer les valeurs des $T(j,l)$ se retrouve dans le fichier *methode_incomplete.py*. Il s'agit d'un algorithme de programmation dynamique. Signature de la fonction :

```
valeur_T(T, sequence, j, l)
```

2.2 Généralisation

Question n°5

Modifiez chacun des cas de l'algorithme précédent afin qu'il prenne en compte les cases déjà coloriées.

On modifie l'algorithme de la manière suivante :

— Si $l = 0$:
  1. S'il existe une case noir dans la ligne i, $T(j,l)$ prend la valeur faux.
  2. Sinon, $T(j,l)$ prend la valeur vrai.

— Si $j = s_l - 1$:
  1. Si $l = 1$: s'il existe une case blanche dans la ligne i, prend la valeur faux.
  2. Sinon : $T(j,l)$ prend la valeur vrai.

— Si $j \leq s_l - 1$ et $l \geq 1$: $T(j,l)$ prend la valeur faux.

— Sinon :
  1. Si la case j est coloriée en noir : on peut pas conclure.
  2. Sinon, on effectue l'appel récursif T(j-1,l). Si la valeur retournée est vrai, on la retourne.
  3. Si l'appel retourne faux, on teste si l'on trouve des cases blanches dans la séquence ; si ce n'est pas le cas, on vérifie si la case $j - s_{l-1}$ est coloriée en blanc ou on vérifie la valeur de $T(j - s_{l-1} - 1, l - 1)$, comme dans la question 3.

Question n°6

Analysez la complexité en fonction de M de l'algorithme. Pour ce faire, on déterminera le nombre de valeurs $T(j,l)$ à calculer, que l'on multipliera par la complexité de calcul de chaque valeur $T(j,l)$.

Nombre de valeurs à calculer : $(M-1) \times l = O(\lceil \frac{M}{2} \rceil * (M-1)) = O(M^2)$, car dans une ligne on a un nombre d'au plus $\lceil \frac{M}{2} \rceil$ de séquences (pire cas : séquence du type $[1, 1, ...1]$).

Complexité de calcul de chaque $T(j,l)$: chaque appel est en $O(j) = O(M)$, car, dans le pire des cas, on doit parcourir la liste du coloriage pour tester la présence ou l'absence d'une couleur dans la sous-liste $C[1, ..., j]$.

Complexité totale : $O(M^2 * M) = O(M^3)$.

Question n°7

Codez l'algorithme.

La fonction demandée pour calculer les valeurs de $T(j,l)$ se retrouve dans le fichier *methode_incomplete.py*. Signature de la fonction :

```
valeur_T_couleur(T, sequence, j, l).
```

2.3 Propagation

Question n°8

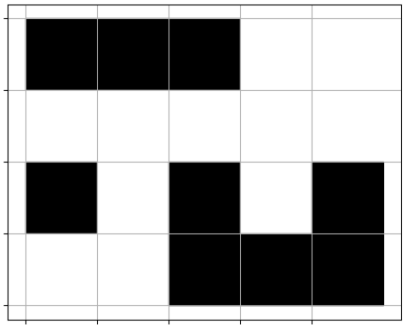
Montrez que cet algorithme est de complexité polynomiale en N et M .

- Complexité *ColoreLig* : M tours de boucle, avec des appels à *valeur_T_couleur*, donc complexité en $O(M * M^3) = O(M^4)$
- Complexité *ColoreCol* : N tours de boucle, avec des appels à *valeur_T_couleur*, donc complexité en $O(N * N^3) = O(N^4)$
- Complexité *ColoreGrille* : $N * M$ tours de boucle, avec $O(N)$ appels à *ColoreLig*, et $O(M)$ appels à *ColoreCol*

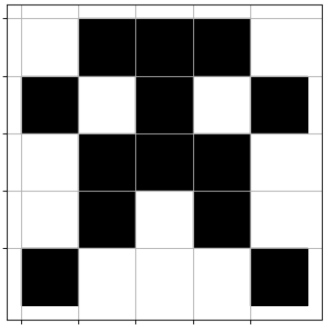
Conclusion : l'algorithme de **propagation** a une complexité totale dans le pire des cas en $O(N * M * (N * M^4 + M * N^4)) = O(N^2 * M^5 + M^2 * N^5)$, il s'agit bien d'une complexité polynomiale en N et M.

Question n°9

Codez l'algorithme de propagation. Votre programme prendra en entrée un fichier texte dont chaque ligne correspond à la séquence associée à une ligne ou une colonne de la grille. Le symbole # indique que l'on passe de la description des lignes à celle des colonnes. Avant le #, il y a autant de lignes dans le fichier que de lignes dans la grille, et à chaque ligne est indiquée la séquence d'entiers représentant les longueurs des blocs. L'algorithme devra permettre une visualisation du coloriage obtenu en sortie, avec des cases de couleur noire, blanche ou indéterminée. Vous vérifierez que votre programme résout correctement l'instance *0.txt*, qui correspond à l'exemple de l'introduction.



(a) Instance 0.txt



(b) Instance 1.txt

FIGURE 1 – Tests sur deux instances simples (Q9) [3]

La fonction demandée pour effectuer le coloriage partiel ou complet de la grille se retrouve dans le fichier *methode_incomplete.py*. Signature de la fonction :

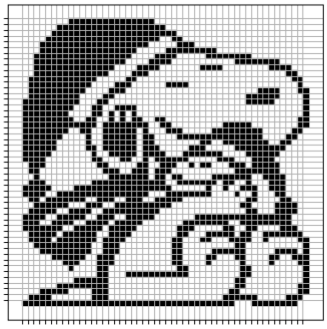
```
ColoreGrille(G, seq_lignes, seq_colonnes)
```

L'algorithme de propagation fait plusieurs appels à ColoreLig (Algorithme n°1 de l'annexe) et ColoreCol (Algorithme n°2 de l'annexe). Il suit le pseudo-code fourni par le sujet. [2] Nous avons créé des variables qui représentent les cas VRAI, FAUX et NESAISPAS, afin de faciliter les tests sur le valeur de la variable ok de l'algorithme.
La lecture d'une instance se fait grâce à la fonction $read\_file$ du fichier *lecture.py*. L'instance est représentée par une classe *Instance* contenant les tableaux de séquences et leurs taille. La grille est représentée par une classe *Grille* contenant la matrice des couleurs et sa taille.

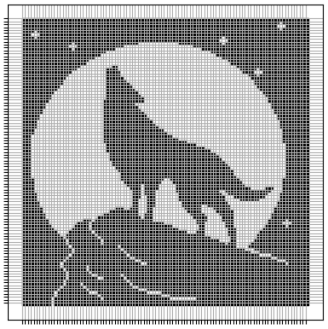
2.4 Tests

Question n°10

Appliquez votre programme sur les instances *1.txt* à *10.txt*. Vous indiquerez dans le rapport les temps de résolution dans un tableau. Vous fournirez dans le rapport la grille obtenue pour l'instance *9.txt*.



(a) Instance 9.txt



(b) Instance 10.txt

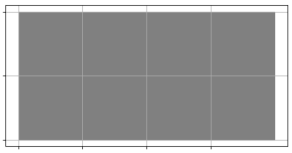
FIGURE 2 – Tests sur deux instances complexes (Q10)

Fichier .txt	**Temps de calcul (en secondes)***
n°0	0.00014
n°1	0.00028
n°2	0.04785
n°3	0.03069
n°4	0.09043
n°5	0.06890
n°6	0.15881
n°7	0.09880
n°8	0.15145
n°9	1.74070
n°10	1.72551

(*) Les tests ont été effectués sur un ordinateur muni d'un processeur Apple M2 (puce basé sur ARM).

Question n°11

Appliquez votre programme sur l'instance *11.txt*. Que remarquez-vous ? Expliquez.



On remarque le fait que l'algorithme n'arrive pas à résoudre l'instance 11.*txt*, car une case grise indique, dans notre code, une incertitude au niveau du choix de la couleur (ie. ok = NESAIPAS). L'algorithme que nous avons codé n'aboutit donc pas à une résolution complète pour toutes les instances.

3 Méthode complète de résolution

Question n°12

Montrez que cet algorithme est de complexité exponentielle en N et M.

D'après le pseudo-code fourni dans les annexes, l'algorithme *Enumeration* est de même complexité que *EnumRec*, soit en $O(2^{M*N}(N^2 * M^5 + M^2 * N^5))$. Cette formule vient du fait que *ColorierEtPropager* est dans le pire des cas en $O(N^2 * M^5 + M^2 * N^5)$. Pour $M * N$ tours de boucles, on fait 2 appels (cas NOIR et cas BLANC) avec *EnumRec*, d'où la complexité exponentielle totale.

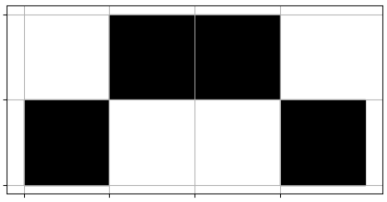
3.1 Implantation et tests

Question n°13

Implantez l'algorithme de résolution complète. Vous vérifierez que votre programme résout correctement l'instance *11.txt*.

La fonction demandée pour effectuer le coloriage complet de la grille se retrouve dans le fichier *methode_ complete.py*. Signature de la fonction :

```
Enumeration(G,seq_lignes,seq_colonnes)
```



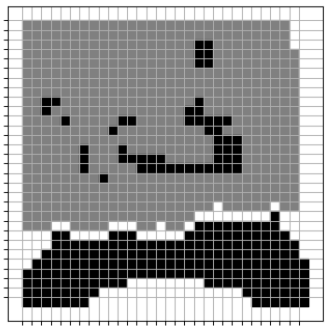
L'algorithme résout bien l'instance 11.*txt*.

Question n°14

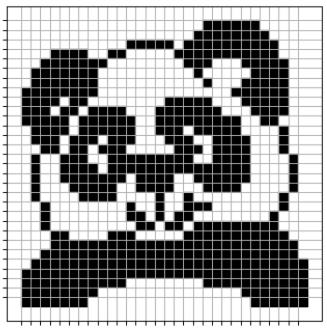
Résolvez les instances *1.txt* à *16.txt* avec un timeout de 2 minutes. Donnez les temps de calcul dans un tableau. Pour les instances *12.txt* à *16.txt*, appliquez également la méthode de la section 1. Commentez. Vous fournirez dans le rapport la grille obtenue avec chacune des deux méthodes pour l'instance *15.txt*.

On remarque une augmentation du temps de calcul pour la résolution des instances : pour la première méthode on a une complexité polynomiale, alors que pour la résolution avec la deuxième méthode on passe à une complexité exponentielle. De plus, pour les instances de 12.txt à 16.txt, avec la méthode incomplète, l'algorithme n'arrive pas à

déterminer la couleur de toutes les cases (on trouve des cases grises dans les images retournées), alors que la deuxième méthode résout intégralement toutes les instances. Remarque : Pour les instances de 0.txt à 10.txt, les temps de calcul sont presque égaux du fait que l'algorithme de résolution complète fait d'abord appel à celui de résolution incomplète avant de tester s'il existe toujours des cases vides dans la grille retrouvée.



(a) Instance 15.txt (avec la méthode n°1)



(b) Instance 15.txt (avec la méthode n°2)

FIGURE 3 – Tests sur l'instance n°15 (Q14)

Fichier .txt	**Temps de calcul (s) pour la méthode incomplète***	**Temps de calcul (s) pour la méthode complète***
n°1	0.00028	0.00102
n°2	0.04785	0.04741
n°3	0.03069	0.02962
n°4	0.09043	0.08654
n°5	0.06890	0.06605
n°6	0.15881	0.15343
n°7	0.09880	0.09510
n°8	0.15145	0.14568
n°9	1.74070	1.67313
n°10	1.72551	1.67609
n°11	0.00005	0.00010
n°12	0.14893	0.15694
n°13	0.19433	0.18637
n°14	0.12141	0.12458
n°15	0.08767	0.16317
n°16	0.33182	18.39279

(*) Les tests ont été effectuées sur un ordinateur portable muni d'un processeur Apple M2 (puce basée sur ARM).

4 Annexe

Algorithme 1 : ColoreLig

Données : G, M, seq_lignes, i (numéro de la ligne), nouveaux_col (à explorer dans la suite)
Sorties : VRAI / FAUX et la grille partiellement coloriée

```
1  C ← copie(G[i]);
2  pour k de 0 à M − 1 faire
3      si C[k] est vide alors
4          T ← tableau de taille (M, longueur + 1) initialisé à −1;
5          C[k] ← BLANC;
6          est_blanc ← valeur_T_couleur(T, seq_lignes[i], M − 1, longueur, C);
7          C[k] ← NOIR;
8          T ← tableau de taille (M, longueur + 1) initialisé à −1;
9          est_noir ← valeur_T_couleur(T, seq_lignes[i], M − 1, longueur, C);
10         si est_noir et est_blanc alors
11             C[k] ← VIDE;
12         fin
13         sinon si non est_blanc et est_noir alors
14             C[k] ← NOIR;
15             G[i][k] ← NOIR;
16             ajouter k à nouveaux_col;
17         fin
18         sinon si est_blanc et non est_noir alors
19             C[k] ← BLANC;
20             G[i][k] ← BLANC;
21             ajouter k à nouveaux_col;
22         fin
23         sinon
24             retourner (FAUX, G);
25         fin
26     fin
27 fin
28 retourner (VRAI, G);
```

Algorithme 2 : ColoreCol

Données : G, M, seq_colonnes, j (numéro de la colonne), nouveaux_lig (à explorer dans la suite)
Sorties : VRAI / FAUX et la grille partiellement coloriée

```
1  longueur ← longueur de seq_colonnes[j];
2  pour ind de 0 à N faire
3  |  C ← C + G[ind][j];
4  fin
5  pour k de 0 à N − 1 faire
6  |  si C[k] est vide alors
7  |  |  T ← tableau de taille (N, longueur + 1) initialisé à −1;
8  |  |  C[k] ← BLANC;
9  |  |  est_blanc ← valeur_T_couleur(T, seq_colonnes[i], N − 1, longueur, C);
10 |  |  C[k] ← NOIR;
11 |  |  T ← tableau de taille (M, longueur + 1) initialisé à −1;
12 |  |  est_noir ← valeur_T_couleur(T, seq_colonnes[i], N − 1, longueur, C);
13 |  |  si est_noir et est_blanc alors
14 |  |  |  C[k] ← VIDE;
15 |  |  fin
16 |  |  sinon si non est_blanc et est_noir alors
17 |  |  |  C[k] ← NOIR;
18 |  |  |  G[i][k] ← NOIR;
19 |  |  |  ajouter k à nouveaux_lig;
20 |  |  fin
21 |  |  sinon si est_blanc et non est_noir alors
22 |  |  |  C[k] ← BLANC;
23 |  |  |  G[i][k] ← BLANC;
24 |  |  |  ajouter k à nouveaux_lig;
25 |  |  fin
26 |  |  sinon
27 |  |  |  retourner (FAUX, G);
28 |  |  fin
29 |  fin
30 fin
31 retourner (VRAI, G);
```

5 Bibliographie

Références

[1] https://docs.python.org/fr/3/.

[2] https://moodle-sciences-23.sorbonne-universite.fr/course/view.php?id=1553

[3] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.grid.html