

IUT CHARLEMAGNE

DIALLO Mohamed
ZAITIT Abdelaziz
MERET Mickael

SOMMAIRE:

- 1- Environnement de développement ;**
- 2 - Structure de données et explications;**
- 3 - Remplissage des tableaux et graphes;**
- 4- Captures, difficultés rencontrées, conclusion.**



1. Environnement de développement

Nous allons commencer par la description de l'environnement de développement utilisé, ensuite que les étapes de mise en place de l'application.

Description:

- Le système d'exploitation: **Windows 10**, Xubuntu
- Langage de programmation: Python, version 2.7 (Pour l'affichage graphique des pavages)
- Environnement de développement: Notepad++
- Outil d'exécution : cmd

2. Structure de données

Le code source comprend est contenu dans un seul fichier, avec 491 lignes de code.

Dans ce fichier, vous avez :

- 7 imports de bibliothèques nécessaire d'une part au programme et d'autres part au débogage et à l'affichage graphique des pavages.

import pdb : pour le débogage

*(2) import matplotlib.** : pour le graphisme interactif

import numpy: package fondamental pour l'informatique scientifique avec Python

import traceback: imprimer des traces de pile de programmes Python

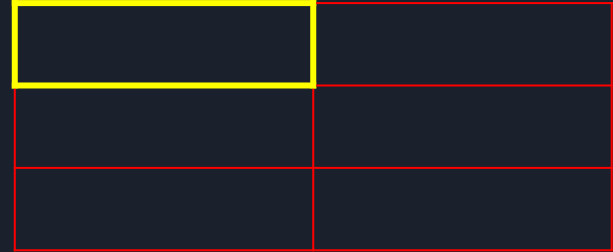
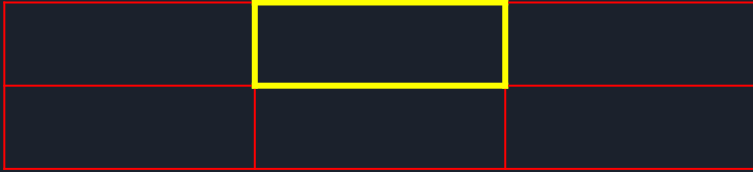
import sys, random: pour l'utilisation des outils systems et la génération aléatoire de valeurs .

Structures de données :

Au tout début on a un rectangle ($n*m$), sur lequel placer des blocs de pavé afin de le couvrir. Chaque façon de le couvrir ou de le "découper" avec une combinaison de blocs de différentes dimension constitue un pavage.

Pour cela le rectangle de départ peut être représenté comme ci dessous :

Exemple : $2 * 3 / 3 * 2$



Dans un rectangle $n*m$ ($2*3 = 6$ pavés).

A l'intérieur, en couleur jaune, vous avez un exemple de pavé ou de bloc ayant une certaine dimension ($1*1$). Nous avons considérez qu'un rectangle donné peut être placé dans un repère graphique ou chaque pavé ou bloc à une coordonné bien précise, donc le rectangle aussi, comme ce qui suit :

Soit ce rectangle (3×3) représenté sous forme de repère, où chaque point a pour coordonnées (y, x) . **Fig gauche** : illustration d'un rectangle;

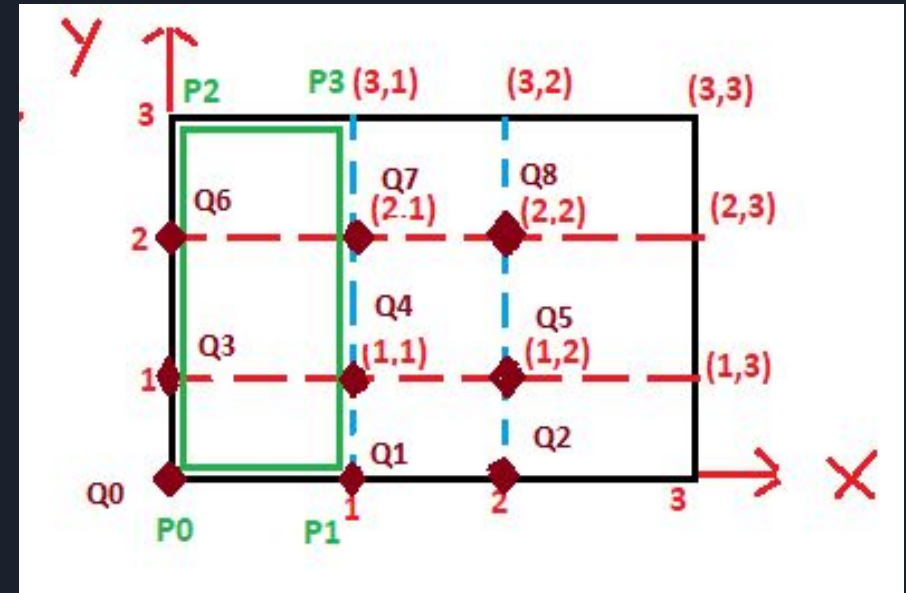
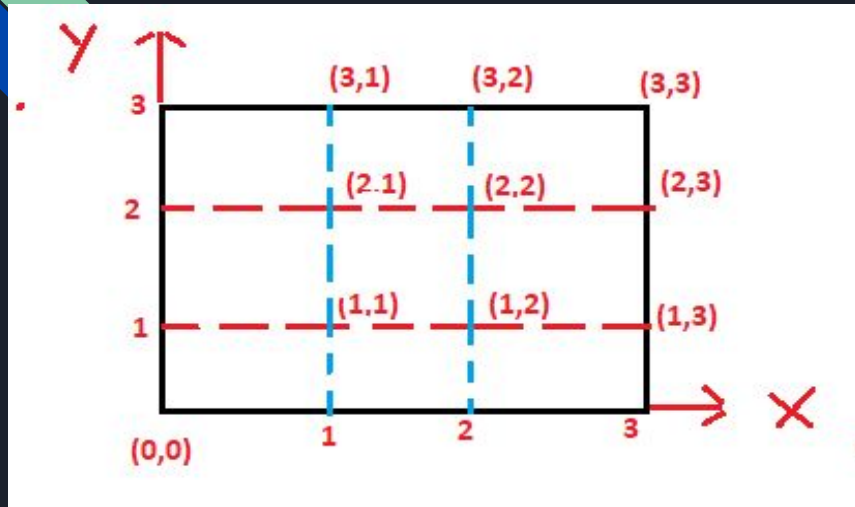
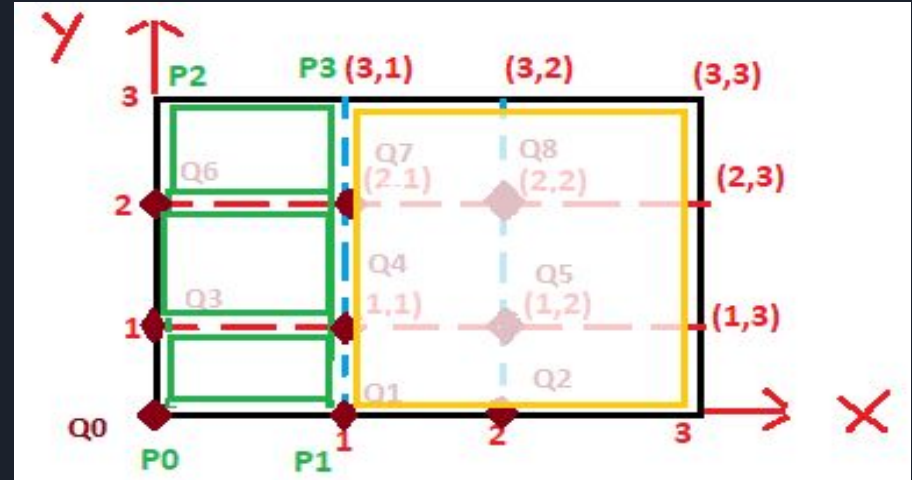
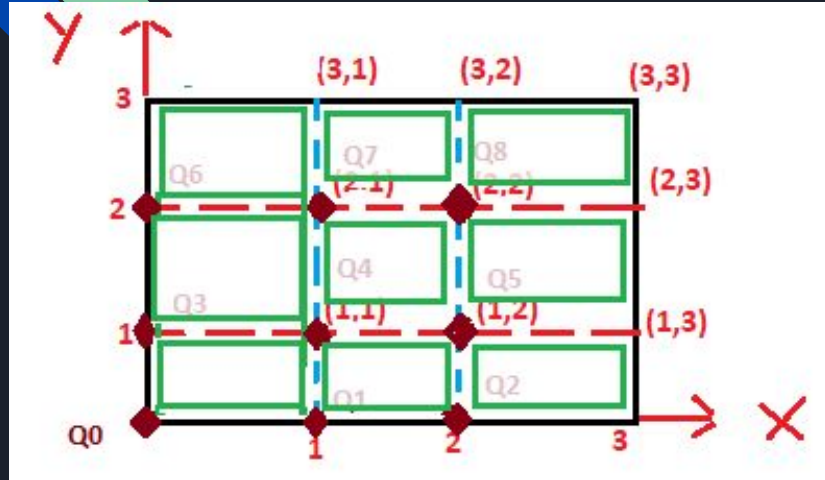


Figure droite :

Soit Q_i , l'ensemble des points intérieurs dans un rectangle donnée: les $Q_i(y, x)$ tel : $0 \leq X_i < m$ et $0 \leq Y_i < m$ et les coord des Q_i : $[Q_0, Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8] = (0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1), (2,2)$.
 Soit P_i , l'ensemble des 4 points qui constituent les bordures d'un pavé ou d'un bloc, si on prends un pavé au hasard comme celui de couleur verte, il a des points sur ces quatres coins de bordure : $[P_0, P_1, P_2, P_3]$ ou les $P_i(y, x)$, pour celui de l'exemple, on a : $(0,0), (0,1), (3,0)$ et $(3,1)$.

Donc un rectangle ($n*m$) est plein, si l'ensemble des points intérieurs Q_i sont occupées (Figure G).
Sachant que si un pavé ou un bloc est grand, il peut occuper plusieurs Q_i à la fois (figure Droite).



Pour le grand pavé (fig G), avant de le poser, comme c'est un rectangle $3*3$, il y a 9 Q_i intérieur dedans soit : $[Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8] = (0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1), (2,2)$, qui sont tous libres. Quand on pose le pavé, son point $P0(0,0)$ se pose sur $Q1$ et comme il est grand, il occupe aussi $Q2, Q4, Q5, Q7, Q8$ en même temps. De même, quand on retire ce pavé ou ce bloc pour poser un autre dans le "backtracking", il libère les points intérieurs qu'il occupait.

Vous avez aussi, comme structures de données, des :

- **Listes** : souvent utilisées pour contenir les coordonnées des points qui représentent les bordures d'un pavé ou bloc (P_i) , comme sur cet exemple :

```
def paves_valides(liste_paves, n, m): #exclus pave nxm
```

```
    new_liste_paves=[]
```

Une liste pour contenir les coordonnées

```
    for (P0, P1, P2, P3) in liste_paves:
        if ((P1[1]-P0[1])<m or (P3[0]-P1[0])<n):
            new_liste_paves.append((P0, P1, P2, P3))
        else:
            #print "pave non valide=> ", (P0, P1, P2, P3)
            pass
```

```
    return new_liste_paves
```

```
def construit_list_points(ensemble1, ensemble2):
    liste_points=[(x,y) for x in ensemble1 for y in ensemble2]
    return liste_points
```

On test pour voir si les coordonnées du pavé sont valides, c'est une fonction de test de la validité des coordonnées d'un pavé

Vous pouvez constater explicitement la construction des listes de points à partir d'ensemble de valeur ..


```
def pave_est_il_placable(pave, dic_points_Qi_occupes):  
    (P0, P1, P2, P3)=pave
```

SDD: listes

```
    largeur = P1[1]-P0[1]  
    hauteur = P3[0]-P1[0] # ou P2[0]-P0[0]
```

Là, on calcule les dimensions
(largeur, et hauteur, soit
n,m du pavé)

```
    placable=True
```

```
    if (dic_points_Qi_occupes[P0]):  
        return False  
    else:  
        for Qi in dic_points_Qi_occupes:  
            if ((Qi[0]>=P0[0]) and (Qi[1]>=P0[1]) and dic_points_Qi_occupes[Qi]):  
                if (((P0[0]+hauteur)>Qi[0]) or ((P0[1]+largeur)>Qi[1])):  
                    placable=False  
                    break  
                else:  
                    pass  
            else:  
                pass
```

A partir des dimensions ci
haut, et les coordonnées
des points intérieurs du
grand rectangle, cette
fonction me renvoie un
booléen, pour signifier si le
pavé est plaçable ou non !
Là aussi très explicite

```
    return placable
```

SDD : listes

Cette fonction prend un pavé et les Qi du rectangle, et essayé de placer le pavé sur à partir d'un Qi

```
def place_pave(pave, dic_points_Qi_occupes):  
    (P0, P1, P2, P3)=pave
```

```
def pavage_est_il_plein(dic_points_Qi_occupes):  
    rep=True
```

En entré les Qi, et après avoir parcouru les Qi (verifie les statuts, renvoi si OUI ou NON, il reste des Qi libres (pavage non plein) ou Tous occupés(pavage plein).

```
def pas_de_lignes_pointilles_completes_y(liste_paves, n):  
    list_y_check=[0]*(n-1)
```

Avec elle, on parcours pour voir si il y a pas de lignes pointillées complètes sur tous les axes Y ou lignes verticales du rectangle.

```
def pas_de_lignes_pointilles_completes_x(liste_paves, m):  
    list_x_check=[0]*(m-1)
```

Même role, sur l'axe des X.

```
def est_ce_une_solution(solution_partielle_a, n, m):
```

Si pas de lignes en pointillées complètes sur les axes X et Y, alors nous avons une solution.

```
def construit_tous_les_paves_possibles_rectangle_NxM(n, m):  
    liste_resultats_combiKnxmM=[]
```

La grande fn, elle construit tous les pavages possibles, appelant plusieurs fns, celle qui génèrent les valeurs aléatoires compris entre n et m, celle qui verifie ces dimensions, celle qui forme si valide à partir des ensembles de points les listes de points qui contitueront les dimensions des pavés.


Vous aurez compris, en grande partie, ce sont des listes et un peu de tableaux qui ont servi comme structure de données à ce projet, les listes étaient dans ce contexte bien placées pour :

- La représentation des points et leur manipulation ;

Les tableaux aussi ont servis, par exemple :

- La récupération en tout début du programme des arguments en lignes de commandes et aussi à l'intérieur du code pour certaines manipulations;

```
if(len(sys.argv)!=4):  
    print_usage(sys.argv[0], "mauvais usage")  
  
var_n=int(sys.argv[1])  
var_m=int(sys.argv[2])
```




des tableaux

Donc, jusque là, nous avons décrit certaines fonctions, expliquer comment le pavé est codé et placé, comment le pavage est effectué, comment nous détectons si les pavages sont valides ou pas !

Le code est très commenté, bien indenté, les noms de fonctions et de variables sont explicites sur leurs rôles respectifs. D'où ces captures d'écran partielles et commentées afin de ne pas trop élargir la documentation .

Les méthodes sont au nombre de 23 + le main() :

Le nom des méthodes est très explicite sur leurs rôles respectifs



```
comput_pavages.py
...combiKparmiN(index, liste_selections, liste_items, results)
...paves_valides(liste_paves, n, m)
...construit_list_points(ensemble1, ensemble2)
...pave_est_il_placable(pave, dic_points_Qi_occupes)
...place_pave(pave, dic_points_Qi_occupes)
...unplace_pave(pave, dic_points_Qi_occupes)
...pavage_est_il_plein(dic_points_Qi_occupes)
...pas_de_lignes_pointilles_completes_y(liste_paves, n)
...pas_de_lignes_pointilles_completes_x(liste_paves, m)
...est_ce_une_solution(solution_partielle_a, n, m)
...construit_liste_de_potentiels_candidats(dic_points_Qi_occupes, liste_paves)
...procede_a_la_solution(solution_partielle_a, liste_de_tous_les_pavages_reduits)
...depose_un_pave(solution_partielle_a, dic_points_Qi_occupes)
...enleve_ce_meme_pave(solution_partielle_a, dic_points_Qi_occupes)
...construit_tous_les_paves_possibles_rectangle_NxM(n, m)
...dessine_un_seul_pave(pave, n, m)
...dessine_pavage(liste_paves, n, m)
...initialisation(n, m, dessine=False)
...backtrack(solution_partielle_a, n, m, dic_points_Qi_occupes, liste_paves, liste_
...clean_pavages_doublons(liste_pavages)
...recup_liste_pavages_compactes(liste_de_tous_les_pavages_reduits)
...affiche_stats_pavages(liste_pavages, text_a_rajouter="")
...print_usage(nom_prog, message)
```



Comparaison entre algos de Backtracking :

Sur le slide suivant, nous allons comparer les fonctions de backtracking, mais nous vous recommandons pour une meilleur lecture de visionner l'image dans le dossier du projet, (on a un peu du mal à l'insérer dans le document)


```
def backtrack(solution_partielle_a, n, m, dic_points_Qi_occupes, liste_paves,
             liste_de_tous_les_pavages_reduits, liste_de_tous_les_pavages):
```

```
    if pavage_est_il_plein(dic_points_Qi_occupes):
        liste_de_tous_les_pavages.append(solution_partielle_a[:])
        if est_ce_une_solution(solution_partielle_a, n, m):
            procede_a_la_solution(solution_partielle_a, liste_de_tous_les_pavages_reduits)

    else:
        pass
```

```
else:
    liste_potentiels_candidats=construit_liste_de_potentiels_candidats(dic_points_Qi_occupes, liste_paves)
```

```
1 for candidat in liste_potentiels_candidats:
    solution_partielle_a.append(candidat) # rajoute un pave a la solution partielle
    depose_un_pave(solution_partielle_a, dic_points_Qi_occupes)
2 backtrack(solution_partielle_a, n, m, dic_points_Qi_occupes, liste_paves,
3 liste_de_tous_les_pavages_reduits, liste_de_tous_les_pavages)
4 enleve_ce_meme_pave(solution_partielle_a, dic_points_Qi_occupes)
    del solution_partielle_a[-1] # supprime dernier pave rajoutee a la solution partielle
```

```
return
```

fonction chercherToutesSolutions (...)
début

si solution complète alors
imprimer la solution

sinon

initialiser la sélection des candidats

1 pour chaque candidat faire
choix du candidat

si acceptable alors

2 enregistrer

3 chercherToutesSolutions (...)

4 effacer enregistrement

fsi

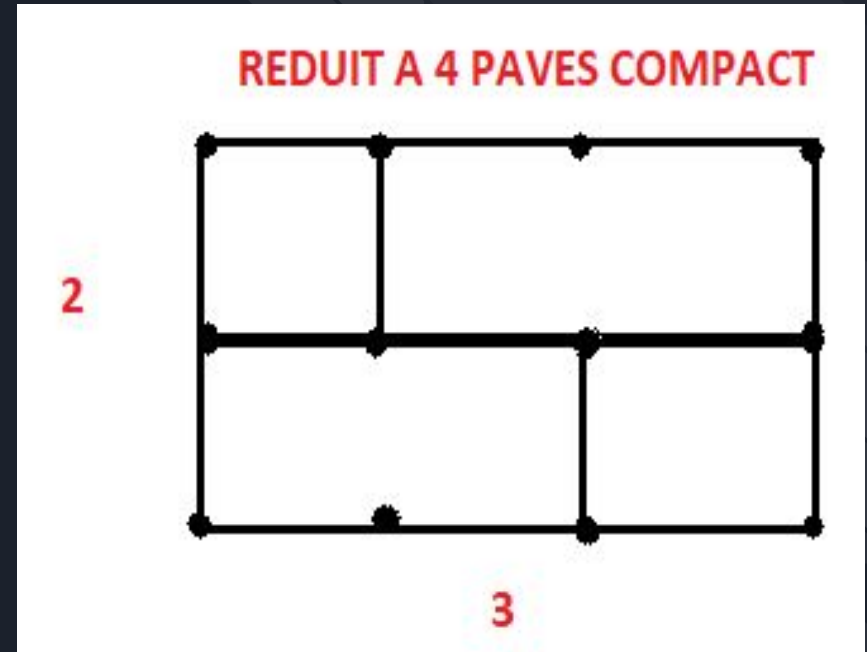
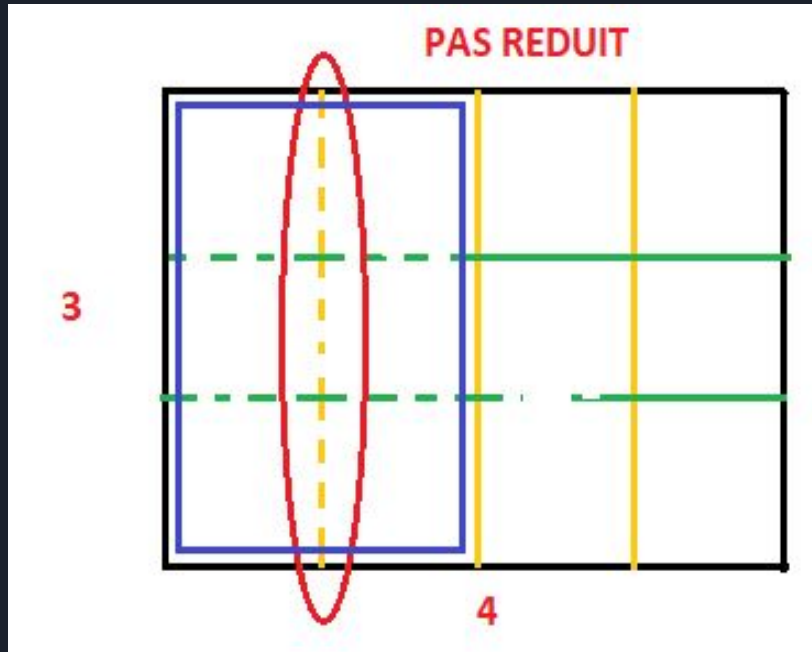
Pour une comparaison, nous ne pouvons conclure que la similitude est parfaite et totale.

- Pavage est réduit ou pas ?
- Pavage est compact ou pas ?

Pour ce qui est d'un pavage réduit ou compact, nous allons illustrer cela par deux exemples :

Ligne pointillée complète détecté sur X

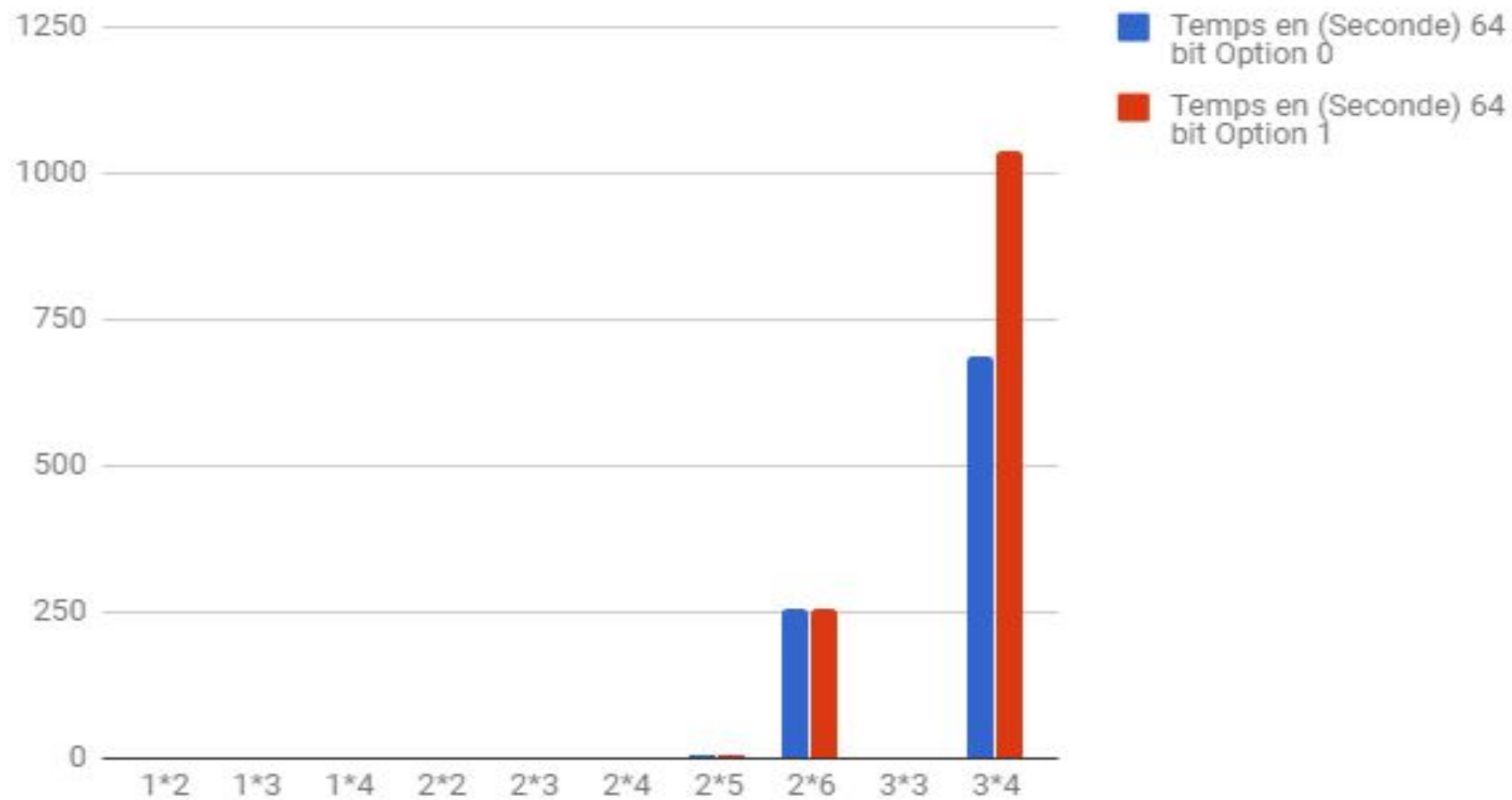
Pas de lignes Point.complète ni X ni Y

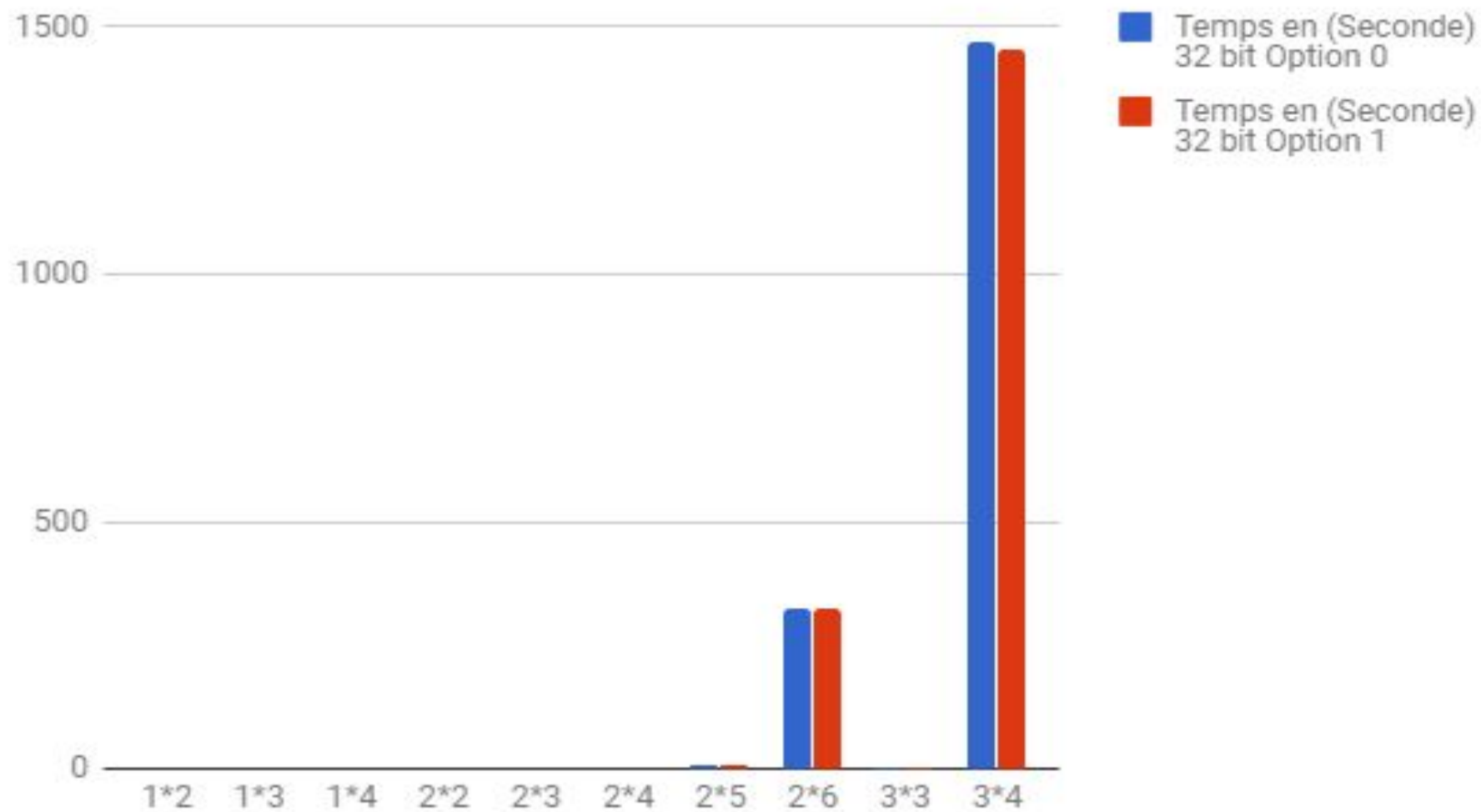


3 - Remplissage des tableaux

(Voir pdf : tableaux (total, execution, réduit et compact **dans le dossier)**

Dimensions	Temps en (Seconde) 32 bit		Temps en (Seconde) 64 bit		Différence des temps d'exécution val(32 bit) - val(64 bits)	
	Option 0	Option 1	Option 0	Option 1	Option 0	Option 1
1*2	0.0005750656128	0.0008819103241	0	0.00200009346	0.0005750656128	-0.001118183136
1*3	0.001043796539	0.001313924789	0	0.01200008392	0.001043796539	-0.01068615913
1*4	0.0008389949799	0.001839876175	0.0009999275208	0.006999969482	-0.0001609325409	-0.005160093307
2*2	0.0008480548859	0.00158905983	0.0009999275208	0.005000114441	-0.0001518726349	-0.003411054611
2*3	0.01315498352	0.01500201225	0.0110001564	0.0150001049	0.002154827118	0.0000019073486
2*4	0.2820608616	0.2517080307	0.1930000782	0.1660001278	0.08906078339	0.08570790291
2*5	6.595897913	5.971367121	4.219000101	4.217000008	2.376897812	1.754367113
2*6	326.573364	324.4923122	254.171	254.1489999	72.40236402	70.34331226
3*3	1.713135958	1.780347109	1.080999851	1.327000141	0.6321361065	0.4533469677
3*4	1470.921625	1454.091045	688.4319999	1039.017	782.489625	415.0740449







- Pour les tableaux de valeurs, dans un premier temps, nous avons essayé de lancer le programme avec des l'option 2:

python comput_pavages n m 2

Pour avoir à la fois : **le nombre de pavages total, réduits et compacts.**
Vu le **temps d'exécution très long**, on a choisi de lancer le programme avec **l'option 0**, pour avoir seulement le **nombre de pavages compact de remplir le tableau "obligatoire"**.

- Pour ce qui est du temps d'exécution, cela dépend des performances de la machine support d'après nos remarques.
- On a essayé de trouver la formule pour la séquence des nombres, mais seul une dans le pdf pavages total à pu être déterminé actuellement;

Formule de la Suite	$U(\text{init}) = 0$				
Pour la	$U(1) = 2 * U(\text{init}) + 1$	1	3	7	15
ligne 1/colonne 1	$U(n) = 2 * U(n-1) + 1$				

4- Enfin, conclusion.

The background features a series of dark gray, three-dimensional rectangular planes that recede into the distance, creating a sense of depth. A light green parallelogram is positioned on one of the upper planes, and a blue parallelogram is on a lower plane, both adding a pop of color to the monochromatic scheme.



• Difficultés rencontrées

En général, seulement lors de la partie explication du projet.

- En premier, on a perdu quelques jours sur la modélisation, la machine à état du projet. La machine à état du projet, nous a beaucoup ouvert les "yeux" sur la façon de faire fonctionner le code;
- En second lieu, une fois qu'on a commencé à coder, on a pas eu de difficultés majeures;
- En conclusion, on a beaucoup appris sur le projet et sur le mécanisme de backtracking.



Détail de lancement du programme

Comment lancer le programme ?

- dézipper le dossier du projet
- ouvrir l'invite de commande cmd dans le dossier
- installer : python 2.7.14 sur **windows** (**cocher l'ajout du path**) ou linux
- Sur cmd : pip install matplotlib (ou nom de biblio.)
- exécuter : **python comput_pavages.py n m u**
ou
- exécuter : **python comput_pavages.py** (sans arguments) = aide de lancement !
et **c'est Ok !** (en cas de souci, faite nous signe !)

Capture d'écran

```
Sélection Windows PowerShell
PS C:\Users\...> python comput_pavages.py 1 2 1

===== DEBUT DU PROGRAMME =====

Total des pavages :
1 avec 2 pieces
(total de 1 pavages )

=====

parmi lesquels on a les pavages reduits suivants :
1 avec 2 pieces
(total de 1 pavages reduits)

=====

Donc un total de (1) pavages compacts
--- Duree - 0.00200009346008 seconds ---

===== FIN DU PROGRAMME =====

appuyez sur <ENTREZ> pour quitter
PS C:\Users\...> python comput_pavages.py 1 2 0

===== DEBUT DU PROGRAMME =====

J'ai trouve (1) pavages compacts
--- Duree - 0.0 seconds ---

===== FIN DU PROGRAMME =====
```


Capture d'écran - python comput_pavages.py 3 3 2 - fig 1

appuyez sur <ENTREZ> pour quitter

PS C:\Users\user\Desktop\etudes\Nancy\cours_archives\algo avance\projet> python comput_pavages.py 3 3 2

===== DEBUT DU PROGRAMME =====

Total des pavages :

4 avec 2 pieces

18 avec 3 pieces

48 avec 4 pieces

92 avec 5 pieces

96 avec 6 pieces

50 avec 7 pieces

12 avec 8 pieces

1 avec 9 pieces

(total de 321 pavages)

=====

parmi lesquels on a les pavages reduits suivants :

12 avec 8 pieces

1 avec 9 pieces

64 avec 5 pieces

92 avec 6 pieces

50 avec 7 pieces

(total de 219 pavages reduits)

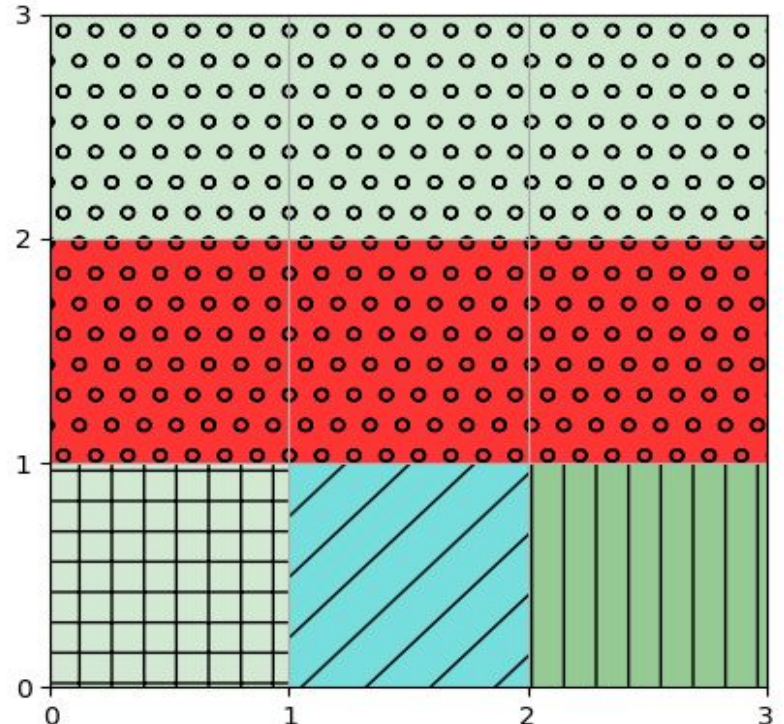
=====

Donc un total de (64) pavages compacts

===== dessin de pavages compacts =====

sol(0)=> [((0, 0), (0, 1), (1, 0), (1, 1)), ((0, 1), (1, 3), (2, 0), (2, 3)), ((2, 0), (2, 3), (3, 0),

Figure 1



Capture d'écran python comput_pavages.py 3 3 2 - fig 2

```
PS C:\Users\user\Desktop\etudes\Nancy\cours_archives\algo_avance\projet> python comput_pavages.py 3 3 2
```

```
===== DEBUT DU PROGRAMME =====
```

```
Total des pavages :  
4 avec 2 pieces  
18 avec 3 pieces  
48 avec 4 pieces  
92 avec 5 pieces  
96 avec 6 pieces  
50 avec 7 pieces  
12 avec 8 pieces  
1 avec 9 pieces  
(total de 321 pavages )
```

```
=====
```

```
parmi lesquels on a les pavages reduits suivants :  
12 avec 8 pieces  
1 avec 9 pieces  
54 avec 5 pieces  
92 avec 6 pieces  
50 avec 7 pieces  
(total de 219 pavages reduits)
```

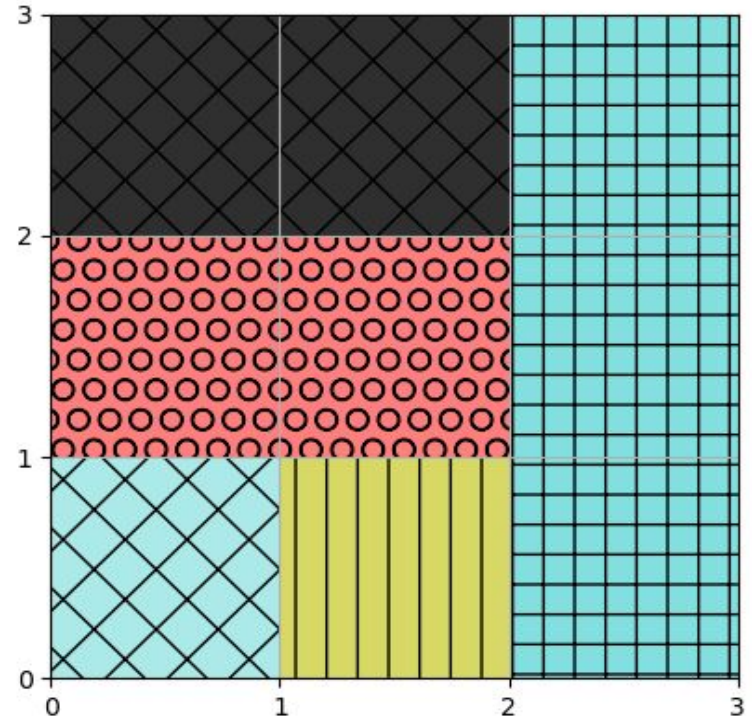
```
=====
```

```
Donc un total de (64) pavages compacts
```

```
===== dessin de pavages compacts =====
```

```
sol(0)=> [((0, 0), (0, 1), (1, 0), (1, 1)), ((0, 1),  
, (1, 3), (2, 0), (2, 3)), ((2, 0), (2, 3), (3, 0), (3, 3))]  
=====  
sol(1)=> [((0, 0), (0, 1), (1, 0), (1, 1)), ((0, 1),  
, (0, 3), (2, 2), (2, 3)), ((2, 0), (2, 3), (3, 0), (3, 3))]
```

Figure 1





Merci !