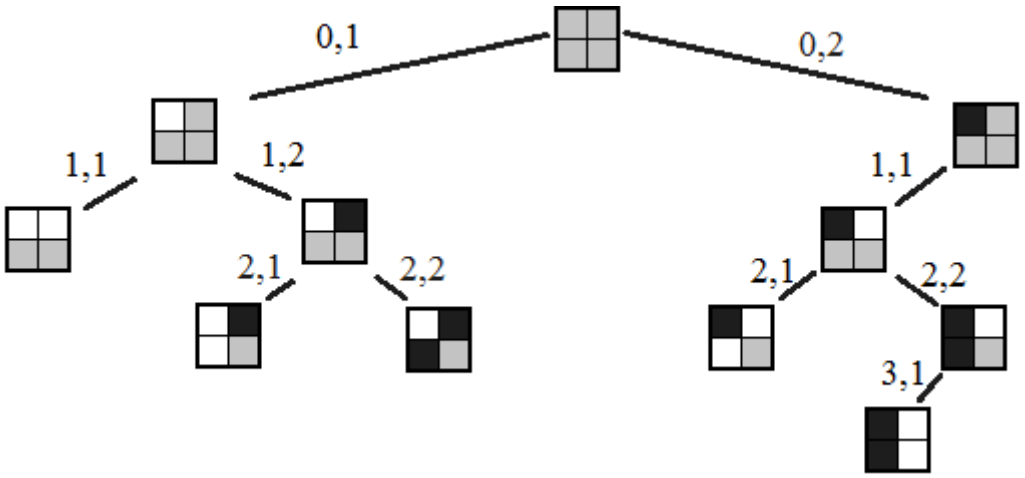
Question 1



L'arbre est exploré par un parcours préfixe.

Question 2
Il y a au plus deux appels par case libre. La complexité pire cas est atteinte pour une matrice dont toutes les cases doivent être coloriées en noir. La complexité est donc en $\Theta(2^p)$.

Question 3
Il suffit de regarder T(m-1, k) car cette valeur indique si il est possible de réaliser un coloriage de V[0 ... m-1] = V en respectant la séquence $(L_1 \dots L_k) = L$.

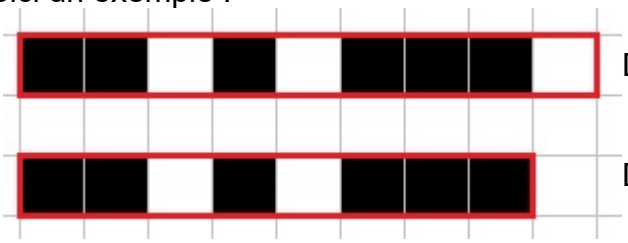
Question 4
T(j, 0) indique si il est possible de réaliser un coloriage de V[0 ... j] sans aucune contrainte, on peut par exemple colorier toutes les cases en blanc. Donc T(j, 0) = vrai pour tout j.

Question 5
$T(L_1\text{-}1, 1)$ considère les cases de 0 à L_1-1, donc considère L_1 cases.
$T(L_1\text{-}1, 1)$ demande à respecter la séquence (L_1), c'est à dire qu'on doit colorier en noir L_1 cases consécutives.
Il est possible de réaliser un coloriage: il consiste à tout colorier en noir. Donc $T(L_1\text{-}1, 1)$ = vrai.

$T(j, l)$ avec $l \geq 1$ et $j < L_l - 1$ considère au plus L_l -1 cases. Il n'est donc pas possible de colorier les L_l cases consécutives correspondant au dernier bloc de la séquence (bloc qui existe car $l \geq 1$).

$T(j, l)$ avec $l \geq 2$ et $j \leq L_l - 1$ considère au plus L_l cases. Colorier les L_l cases correspondant au dernier bloc remplit au minimum toutes les cases de V, il n'est donc pas possible de colorier les cases correspondant à L_1 (L_1 et L_2 existent car $l \geq 2$).

Question 6
Au plus court, on colorie les L1 premières cases en noir puis la case suivante en blanc puis les L2 suivantes en noir. Ainsi de suite jusqu'à Ll.
On a donc besoin de L1 + L2 + ... + Ll cases que l'on coloriera en en noir ainsi que d'une case blanche entre chaque bloc soit l-1 cases.
Finalement, si on fixe V[j] à 1, une case blanche de plus est nécessaire après Ll.
Si on fixe V[j] à 2, la dernière case, c'est à dire V[somme des Li + l-2] étant déjà noire, on n'a pas besoin d'une case supplémentaire.
Voici un exemple :



Dans le premier cas (V[j]=1), jmin = 9

Dans le deuxième cas (V[j]=2), jmin = 8

On obtient donc :
Si V[j] = 1

$$T(j,l) \Leftrightarrow j \geq \sum_{i=1}^{l} L_i \ + l$$

Si V[j] = 2

$$T(j,l) \Leftrightarrow j \geq \sum_{i=1}^{l} L_i \ + l - 1$$

On remarque que si T(j, l) avec V[j] = 1 alors T(j, l) avec V[j] = 2.
Par contraposée, on a :
Si T n'est pas coloriable avec V[j] = 2 alors T n'est pas coloriable non plus avec V[j] = 1.
Pour savoir si T est coloriable avec V[j] = 0, il suffit de regarder si T(j, l) avec V[j] = 2.

Question 7
Pour calculer la valeur de T(j, l), on peut utiliser la formule de récurrence suivante:

$$\begin{cases} \text{vrai} & \text{si } l = 0 \\ \text{faux} & \text{sinon si } j < Ll - 1 \\ T(j - Ll - 1, l - 1) & \text{sinon si V[j] = 0 ou 2} \\ T(j - Ll\ \ , l - 1) & \text{sinon} \end{cases}$$

Question 8
Cette ligne permet de retourner la valeur TT[j][l] si elle a déjà été calculée, c'est le principe de la mémoïsation. Cela permet de réduire la complexité de l'algorithme.

Question 9
De la ligne 1 à la ligne 4, aucun appel récursif ne peut être effectué. Le code à partir de la ligne 5 ne peut être exécuté que si on n'est pas rentré dans le
«Si (TT[j][l] ≠ 'non visité') Alors Retourner TT[j][l] »
Donc les appels récursifs ne se font que si TT[j][l] est égal à 'non visité'.

On peut faire des appels récursifs à TestVecteur_Rec (V, j, l, TT) $\forall\, j, l < m$
Grâce à la mémoïsation, à partir du 2ème appel avec les mêmes arguments, la complexité est en O(1).
On peut donc considérer que le nombre d'appel est en O(m²).

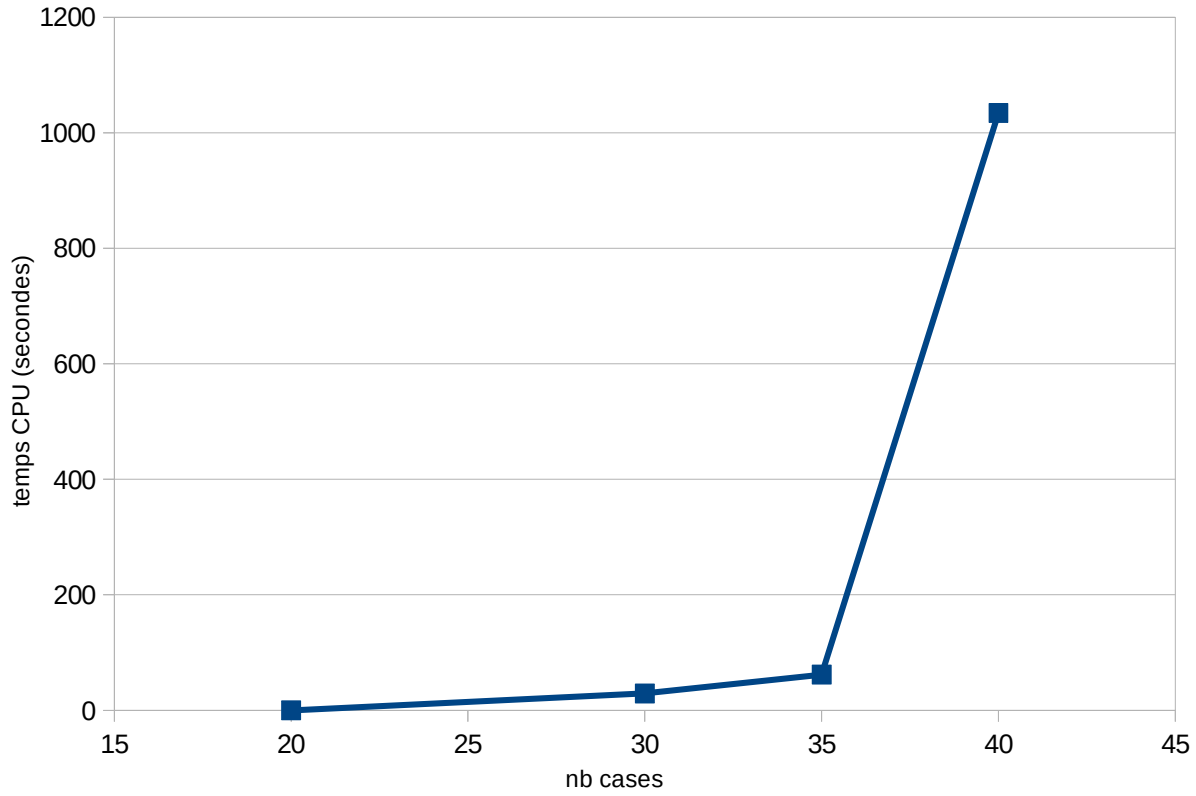
TestSiAucun étant en O(m), la complexité d'un sous-problème est aussi en O(m).
Au final, on a:
complexité = nb_sous_pb * complexité_sans_appel
= O(m²) * O(m)
= $O(m^3)$
Cette complexité est bien polynomiale.

Question 10
On constate que la durée d'exécution est supérieure à 5 minutes sauf pour les instances 0,1 et 11 qui sont très petites. La complexité augmente donc très rapidement.

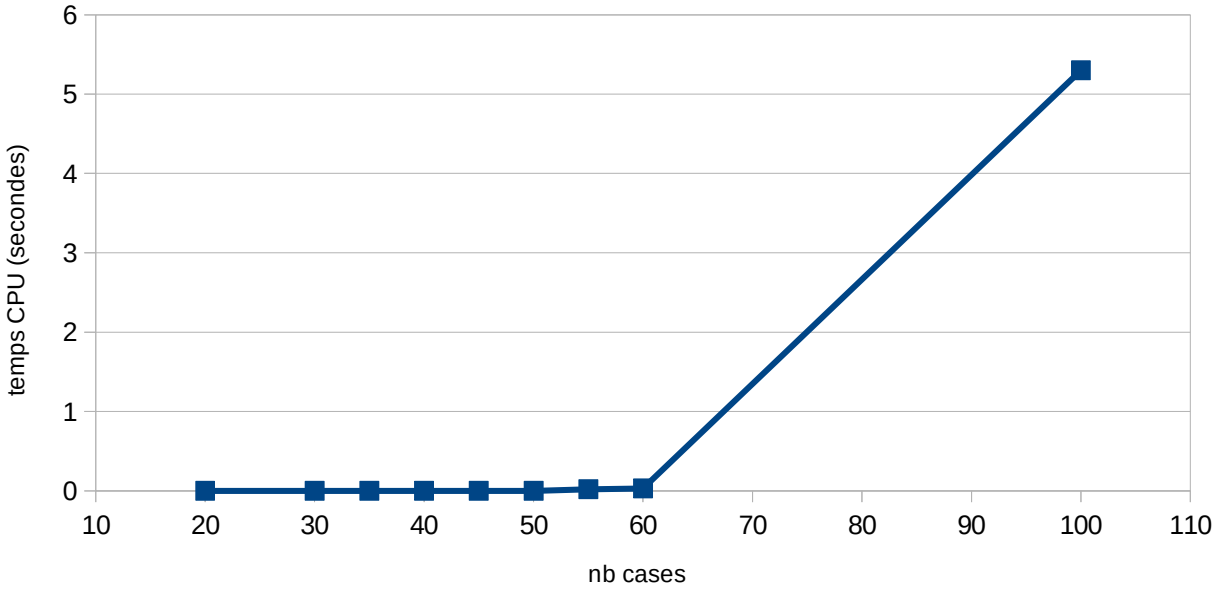
Question 11
Voir code.

Question 12
Voir code.

Question 13



Temps d'exécution de la fonction Enumération en fonction du nombre de case de l'instance



Temps d'exécution de la fonction TestVecteurLigne_Rec en fonction du nombre de case de l'instance

Le temps d'exécution de la fonction Enumération avec une taille de 40 cases est d'environ 1034 secondes (CPU). À partir d'une taille de 45 cases, le temps d'exécution dépasse les 30 minutes (réelles). Cela correspond à la complexité théorique qui est exponentielle.

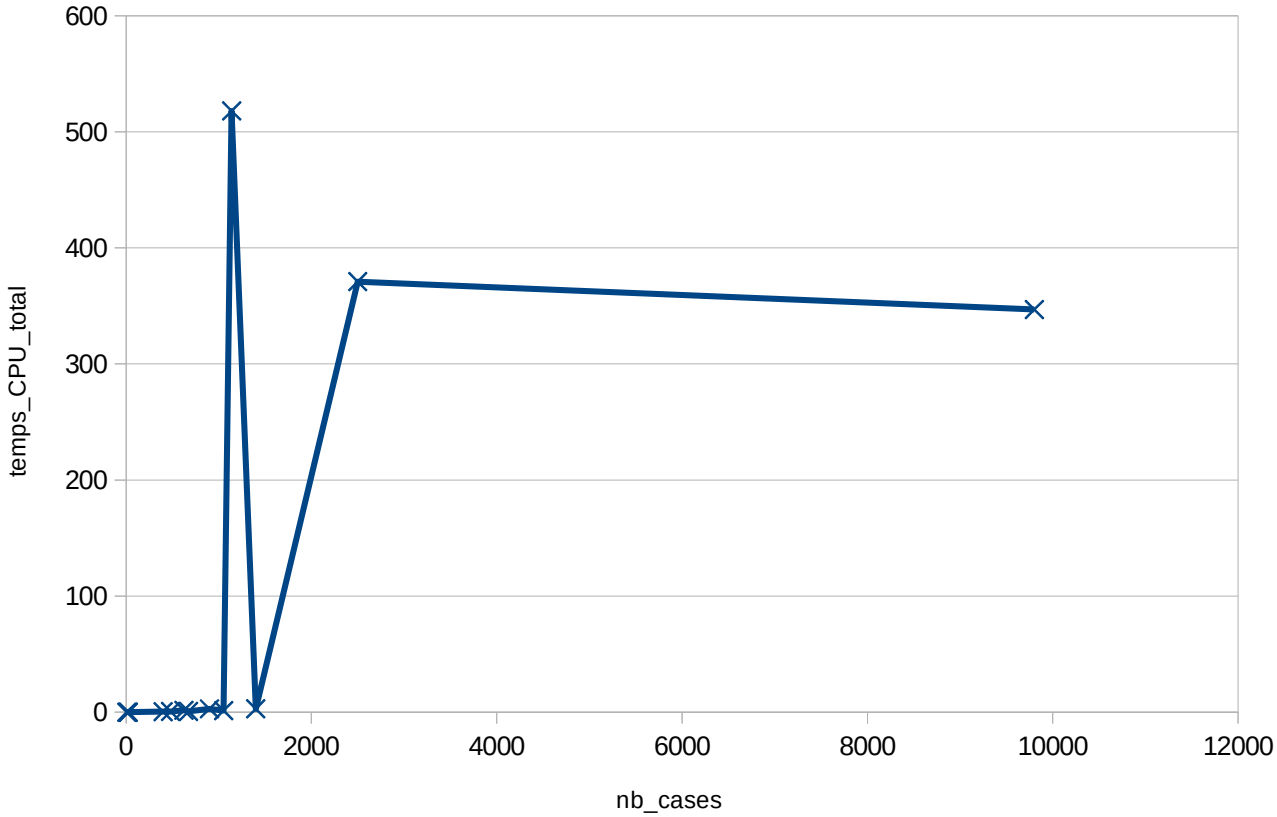
La fonction TestVecteurLigne_Rec est beaucoup plus rapide. Le temps d'exécution pour une instance de 40 cases est à peu près nul. La complexité théorique indique que cette fonction est moins coûteuse asymptotiquement, c'est déjà vrai pour une taille de 40 cases.

Question 14
Voir code.
Question 15

instance	n	m	n*m	ratio	temps propa
0	4	5	20	1	0
1	5	5	25	1	0
2	20	20	400	1	0,47
3	13	37	481	1	0,64
4	25	25	625	1	1,63
5	25	27	675	1	0,66
6	30	30	900	1	2,86
7	31	34	1054	1	1,36
8	40	35	1400	1	2,91
9	50	50	2500	1	370,89
10	99	99	9801	1	346,83
11	2	4	8	0	0
12	33	28	924	0,85	1,63
13	45	45	2025	1	36,42
14	30	38	1140	0,95	1,25
15	30	30	900	0,39	1,19
16	35	50	1750	?	?

D’après les instances fournies, on remarque que la fonction remplit généralement tout le tableau (instances 0 à 10 et instance 13). La fonction peut aussi peut aussi ne rien remplir (instance 11) ou seulement une partie (instances 12, 14 et 15). Finalement, dans le cas de l’instance 16, bien qu’elle ne soit pas particulièrement grande (1750 cases) le temps d’exécution est largement supérieur aux autres.

Question 16
En enchaînant les deux fonctions, le temps d'exécution est fortement réduit comparé à l'exécution de Enumération uniquement. Cela s'explique par le fait que le plus gros du travail est traité en temps polynomial et que seule la fin du problème a une complexité exponentielle.
Par ailleurs, on peut remarquer que la complexité ne dépend pas seulement de la taille de l'instance (la fonction «temps d'exécution en fonction de la taille» n'est pas croissante). Par exemple, pour une instance de 9801 cases nous avons mesuré un temps de 345 secondes CPU alors que pour une instance de taille de 2500 cases nous avons mesuré un temps de 371 secondes CPU.



Temps d'exécution de propagation+Enumération en fonction du nombre de case de l'instance