Paving with tatamis

Description of the problem

If you rent an apartment or a house in Japan, the size of each room is neither measured in square meters or in square feet but in tatamis. Tatamis are mattresses made of *goza*, or rice straw, that are fixed on wooden frames. People use tatamis to cover the floor of rooms in traditional houses.

A basic student room will measure 6 tatamis, i.e. it is possible to place 6 tatamis in the living room (amenities like bathrooms are not measured). A tatami can be modelled as a rectangle of size 2 × 1, though the size of the tatami is not standard: in Tokyo region, tatamis are smaller (88 × 176 cm) than in Kyoto region (91 × 182 cm).

According to the shape of a room, there are several ways to pave it with tatamis. Apart from obvious constraints stating that tatamis cannot overlap or extend outside of the room, there is a Japanese aesthetic standard rule to follow: no four tatamis shall meet at the same point.

Paper submission

Solve this problem for a room of width xmax and length ymax by answering the following questions.

Submit a document in PDF format and the code as a tatami_solve.py file with

1. Define the decision variables of the problem (i.e. placement of the lower left corner and orientation for each tatami).

Hint: to specify the orientation of a tatami, two variables corresponding respectively to its size along the x-axis and its size along the y-axis can be defined, then constrained together.

2. Define two auxiliary variables for each tatami corresponding to the x-coordinate of its right side and to the y-coordinate of its top side.

3. Add constraints to prevent tatamis to extend outside of the room.

4. Add non-overlapping constraints.

5. Break the permutation symmetry among tatamis by lexicographically ordering the coordinates of their lower left corner.

6. Add constraints to prevent four tatamis to meet in the same point.

Hint: a sufficient condition (if the tatamis are lexicographically ordered) is to forbid that, for any two tatamis i < j*, tatami* i *has its upper right corner meets the lower left corner of tatami* j*.*

7. If the room is square, add a constraint to break the first diagonal symmetry.

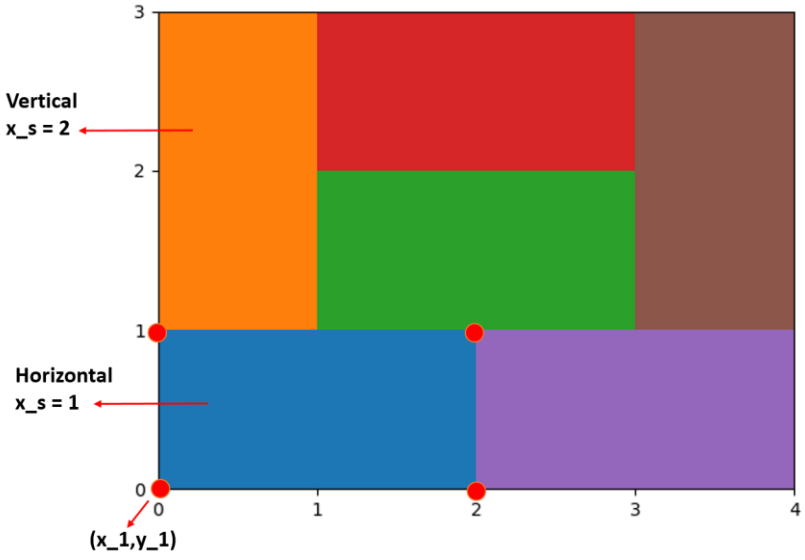
Nos réponses

1. Variables de décisions du problème

On a 3 variables de de décision par tatami :

- x_1: la position du coin inférieur gauche du tatami selon l'axe x, x_1 est compris entre 0 et xmax
- y_1 : la position du coin inférieur gauche du tatami selon l'axe y, y_1 est compris entre 0 et ymax
- xs : le positionnement du tatami, xs =1 ou xs = 2 (1 pour vertical, 2 pour horizontal)

On a donc en tout 3n variables de décision avec n le nombre de tatamis qu'on peut placer dans la pièce.

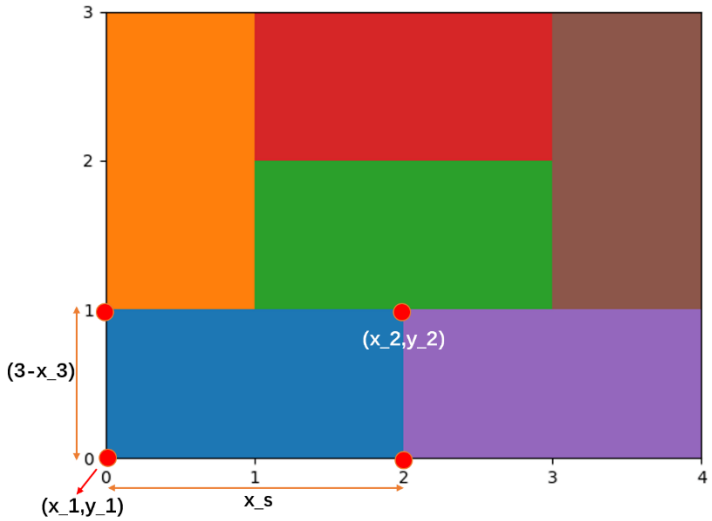


2. *Variables auxiliaires pour chaque tatami*

Pour chaque tatami, on ajoute 3 variables auxiliaires qui correspondent à la position du coin supérieur droit du tatami et qui permettra d'écrire la contrainte « le tatami ne doit pas dépasser de la pièce » :

- x_2 : la position du coin supérieur droit du tatami selon l'axe x
- y_2 : la position du coin supérieur droit du tatami selon l'axe y

Pour chaque tatami, on peut déterminer la valeur de x_2 et y_2 à partir des 3 variables de décisions puisqu'on connaît la position du coin supérieur droit à partir de la position du coin inférieur gauche et les dimensions du tatami selon x et y. Donc pour chaque tatami, on a $x\_2 = x\_1 + xs$ et $y\_2 = y\_1 + (3 - xs)$

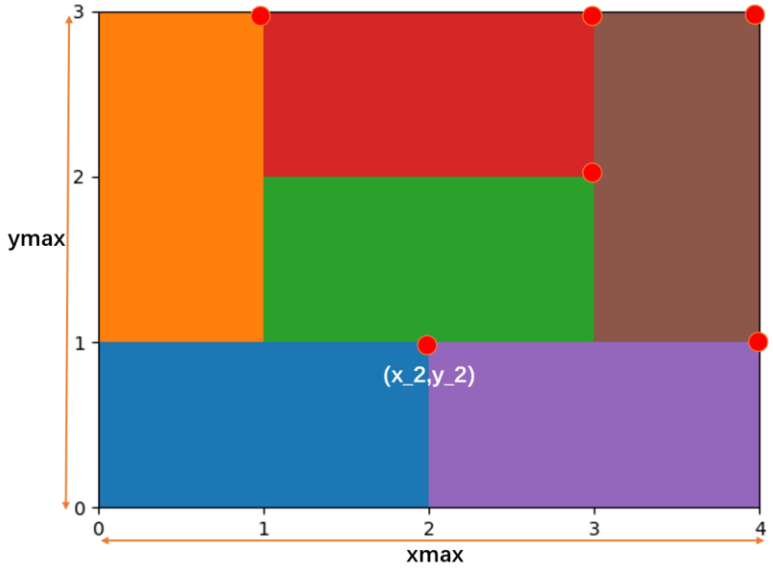


3. *Contraintes sur les limites de la pièce*

On doit contraindre le tatami à être dans la pièce, donc il faut que :

- x_2 soit inférieur ou égal à xmax
- y_2 soit inférieur ou égal à ymax

On n'a pas besoin de contrainte x_1 et y_1 car on a limité les valeurs qu'elles peuvent prendre lors de la définition à (0, xmax) et (0, ymax) respectivement et de plus on a nécessairement x_1 < x_2 et y_1 < y_2 vu qu'un tatami est de taille positive selon x et y.



4. Contraintes du overlapping

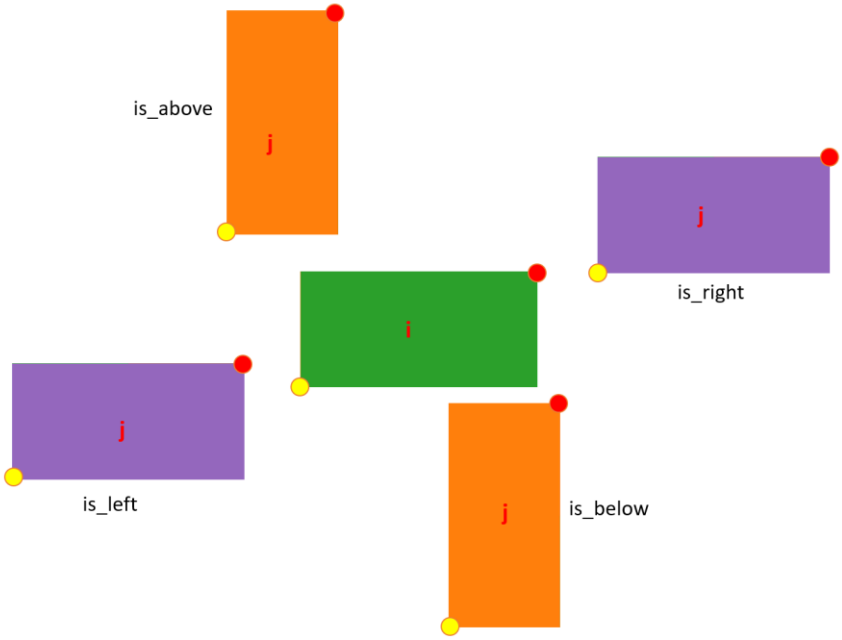
Les tatamis de doivent pas se superposer, on ajoute donc les contraintes suivantes pour chaque paire de tatami possible :

- Un point doit soit se situer strictement au-dessus, soit strictement en-dessous, soit strictement à gauche, soit strictement à droite de l'autre tatami de la paire.

On construit donc 4 variables booléennes pour chaque paire de tatami indexés i et j : is_left, is_right, is_above, is_below qui valent TRUE si le tatami j et respectivement à gauche, à droite, au dessus ou en dessous du tatami i :

- is_left = x_2[j] <= x_1 [i]
- is_right = x_1[j] >= x_2 [i]
- is_ above = y_1[j] >= y_2 [i]
- is_below = y_2[j] <= y_1 [i]

Le tatami j est bien positionné sans overlapping du tatami i lorsqu'on on contraint sur (is_left OU is_right OU is_above OU is_below).



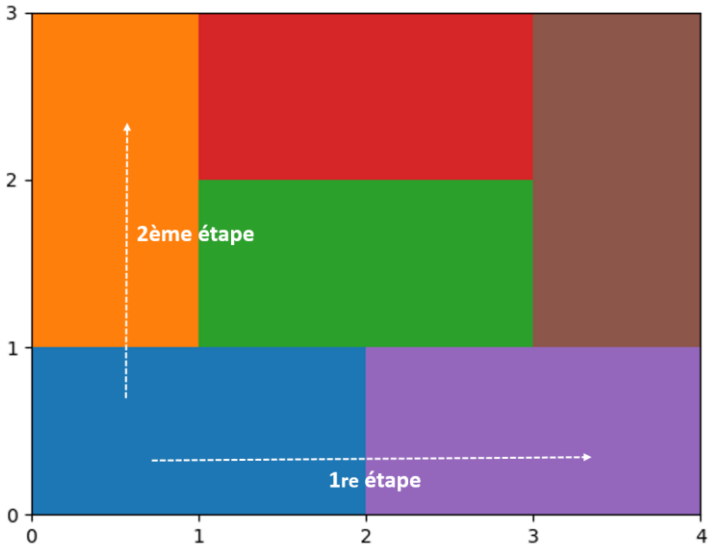
5. Casser la symétrie de permutation

Afin d'éviter d'avoir deux solutions identiques mise à part une permutation de l'ordre des tatami renvoyée par le solveur, nous avons mis en place un ordre des tatamis de la façon suivante : sont indexés en premiers les tatamis de valeur x_1 la plus petite (coin inférieur gauche à gauche sur le placement), puis à valeur x_1 égal, sont indexé en premiers les tatamis de plus petite valeur y_1. Par exemple, le tatami suivant de taille 4x4 est indexé comme dans le schéma suivant :

Pour réaliser cet ordonnancement, nous avons créé deux variables booléennes pour tout i < n:

- x_smaller = x_1 [i] < x_1 [i+1]
- y_smaller = (x_1[i] == x_1[i+1] & y_1[i] < y_1[i+1])

On contraint ensuite sur (x_smaller OU y_smaller). Ainsi, nous avons ordonné sur les x_1 du plus petit au plus grand puis sur les y_1 en cas d'égalité des x_1 et pour des configurations identiques nous avons un seul résultat en sortie.



6. *Empêcher 4 tatamis de partager un sommet*

Ayant indexé les tatamis, cette contrainte se traduit par l'interdiction pour deux points i, j tels que i<j d'avoir le sommet supérieur droit de i qui coïncide avec le sommet inférieur gauche de j. Donc ces deux points ont soit des coordonnées en x différentes, soit des coordonnées en y différentes

On a donc construit les deux variables suivantes :

- x_not_equal = x_2[i] != x_1 [j]
- y_not_equal = y_2[i] != y_1[j]

On constraint ensuite le solution à respecter le condition x_not_equal OU y_not_equal pour toute paire de tatami d'indice i < j.

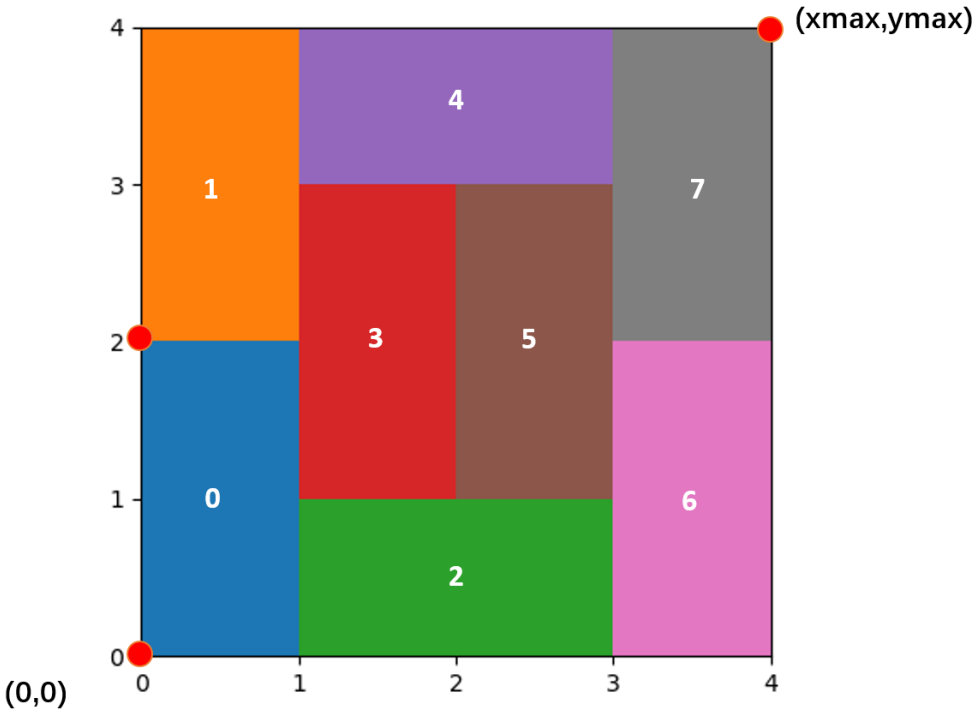
7. *Empêcher la symétrie en cas de pièce carrée*

Considérons l'indexation des tatamis de deux résultats en sortie pour une pièce carrée 4x4 sans contrainte supplémentaire à ce stade.

Nous pouvons voir qu'en imposant au premier tatami d'être en bas à gauche, au dernier tatami d'être en haut à droite, et au deuxième tatami d'être au dessus du premier (on lui interdit donc d'être à sa droite) nous éliminons la duplicité des solutions. Cela se traduit par les contraintes suivantes qui contribuent aussi à améliorer la qualité du solveur puisque nous limitons le champ des solutions possibles à explorer en fixant plus de conditions sur les premiers tatami :

- Si la pièce est carré : on impose que x_1[0] == x_1 [1]
- Et on ajoute les contraintes suivantes :

- x_1[0] == 0
- y_1[0] == 0
- x_2[n-1] == xmax
- y_2[n-1] == ymax

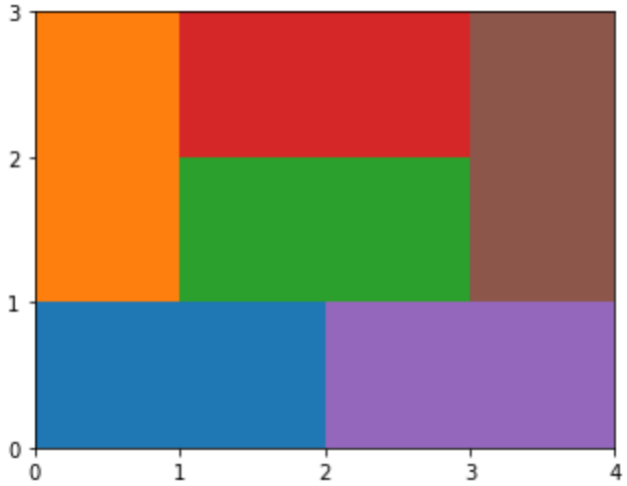


Nos résultats

*1. xmax*ymax=4×3*

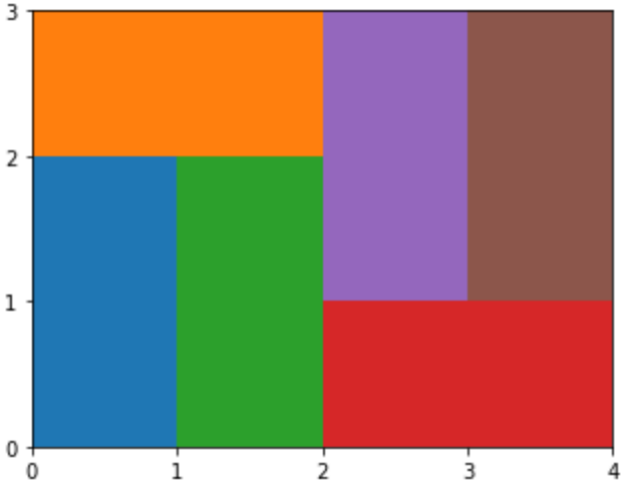
Backtracks : 11
Current solution : [0, 0, 1, 1, 2, 3, ...]
Resolution status : True
Resolution time : 0.00042s

[0, 0, 1, 1, 2, 3]
[0, 1, 1, 2, 0, 1]
[2, 1, 2, 2, 2, 1]



Backtracks : 20
Current solution : [0, 0, 1, 2, 2, 3, ...]
Resolution status : True
Resolution time : 0.00057s

[0, 0, 1, 2, 2, 3]
[0, 2, 0, 0, 1, 1]
[1, 2, 1, 2, 1, 1]

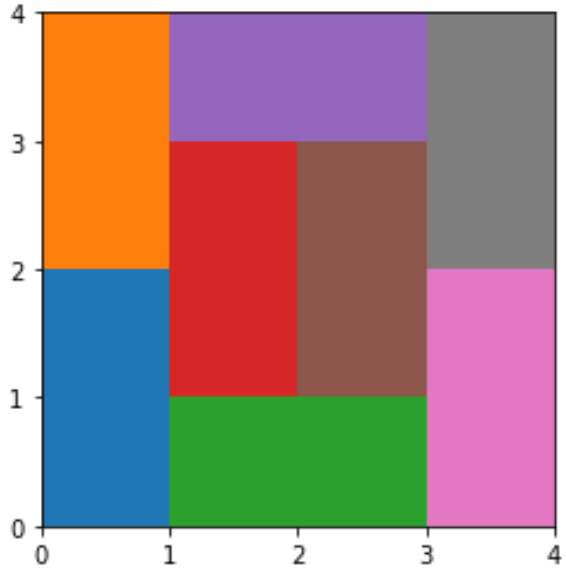


Backtracks : 58
Current solution : None
Resolution status : True
Resolution time : 0.0015s

*2. xmax*ymax=4×4*

Backtracks : 66
Current solution : [0, 0, 1, 1, 1, 2, ...]
Resolution status : True
Resolution time : 0.0022s

[0, 0, 1, 1, 1, 2, 3, 3]
[0, 2, 0, 1, 3, 1, 0, 2]
[1, 1, 2, 1, 2, 1, 1, 1]

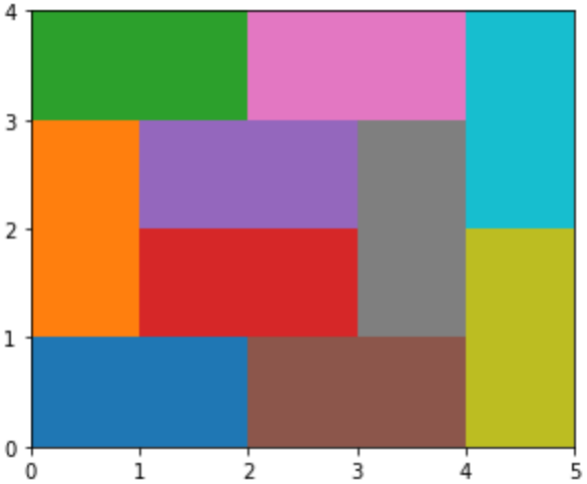


Backtracks : 118
Current solution : None
Resolution status : True
Resolution time : 0.0035s

*3. xmax*ymax=5×4*

Backtracks : 100
Current solution : [0, 0, 0, 1, 1, 2, ...]
Resolution status : True
Resolution time : 0.005s

[0, 0, 0, 1, 1, 2, 2, 3, 4, 4]
[0, 1, 3, 1, 2, 0, 3, 1, 0, 2]
[2, 1, 2, 2, 2, 2, 2, 1, 1, 1]

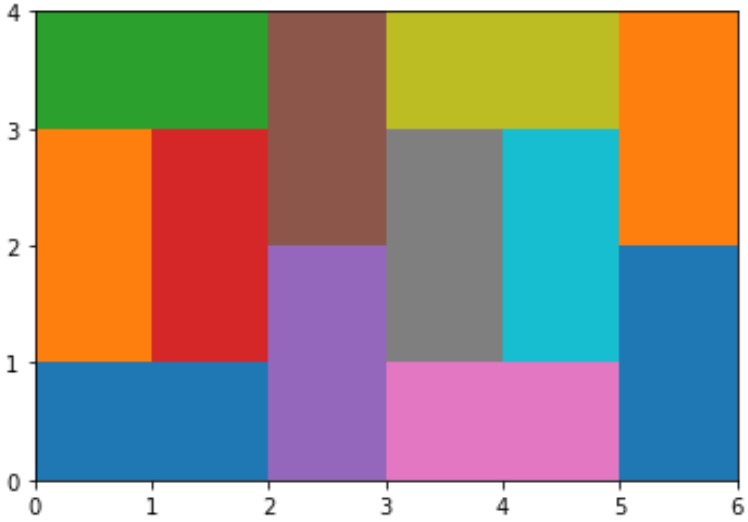


Backtracks : 866
Current solution : None
Resolution status : True
Resolution time : 0.032s

4. xmax*ymax=6×4

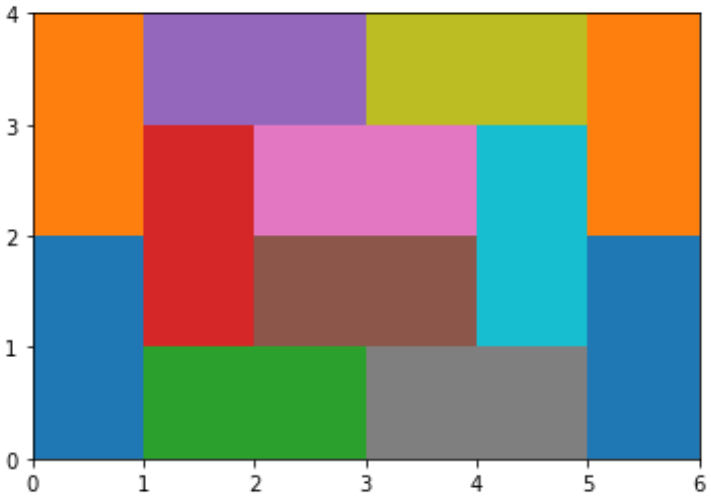
Backtracks : 777
Current solution : [0, 0, 0, 1, 2, 2, ...]
Resolution status : True
Resolution time : 0.03s

[0, 0, 0, 1, 2, 2, 3, 3, 3, 4, 5, 5]
[0, 1, 3, 1, 0, 2, 0, 1, 3, 1, 0, 2]
[2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1]



Backtracks : 1519
Current solution : [0, 0, 1, 1, 1, 2, ...]
Resolution status : True
Resolution time : 0.056s

[0, 0, 1, 1, 1, 2, 2, 3, 3, 4, 5, 5]
[0, 2, 0, 1, 3, 1, 2, 0, 3, 1, 0, 2]
[1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1]

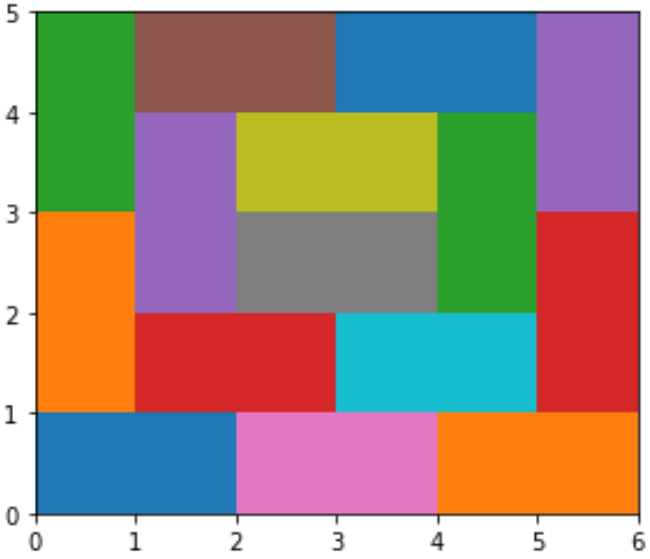


Backtracks : 4352
Current solution : None
Resolution status : True
Resolution time : 0.16s

5. xmax*ymax=6×5

Backtracks : 3855
Current solution : [0, 0, 0, 1, 1, 1, ...]
Resolution status : True
Resolution time : 0.21s

[0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 5]
[0, 1, 3, 1, 2, 4, 0, 2, 3, 1, 4, 0, 2, 1, 3]
[2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1]



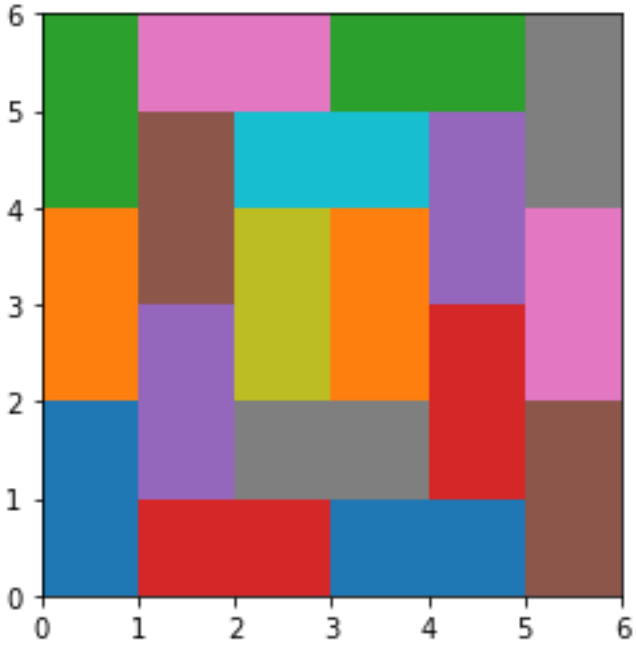
Backtracks : 15670
Current solution : None
Resolution status : True

Resolution time : 0.82s

6. xmax*ymax=6×6

Backtracks : 24293
Current solution : [0, 0, 0, 1, 1, 1, ...]
Resolution status : True
Resolution time : 2.1s

[0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5]
[0, 2, 4, 0, 1, 3, 5, 1, 2, 4, 0, 2, 5, 1, 3, 0, 2, 4]
[1, 1, 1, 2, 1, 1, 2, 2, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1]



Backtracks : 65221
Current solution : None
Resolution status : True
Resolution time : 5.6s