ECE 143

May 22, 2018

1 ECE 143: Individual Project

1.0.1 All import statements for the project

```
In [33]: # import Python Libraries for the Individual Project
         import numpy as np
         import matplotlib.pyplot as plt
         import matplotlib.patches as patches
         import random
         import warnings
```

1.0.2 All code for the project

1. Object: MyClass instance
   - Object.length - length of map
   - Object.width - width of map
   - Object.map - map
   - Object.tower - number of towers on map
   - Object.trial - number of trials
   - Object.totalArea - total area of map
   - Object.currentArea - current area of map
   - Object.colorList - list of colors for rectangles
2. Methods:
   - init (length, width) - initialize map
   - add (startLength, startWidth, endLength, endWidth) - add rectangle to map
   - displayMap () - display map
   - getNumTowers () - get number of towers on map
   - getNumTrials () - get number of trials
   - getTotalArea () - get total area on map
   - getCurrentArea () - get current area on map
3. Note
   - Relevant assert statements, docstrings for methods, comments, and concise / modular code included

In [34]:
```
class MyClass(object):

    # MyClass object (self) contains several relevant fields as listed below:
    #
    # self.length - length of the desired coverage footprint
    # self.width - width of the desired coverage footprint
    # self.map - desired coverage footprint
    # self.tower - number of towers on the desired coverage footprint
    # self.trial - number of trials on the desired coverage footprint
    # self.totalArea - total area of the desired coverage footprint
    # self.currentArea - currentarea of the desired coverage footprint
    # self.colorList - a list of colors used for the desired coverage footprint

    def __init__(self, length, width):
        '''
        Given `self`, which is a MyClass object, `length`, which is the length of
        the desired coverage footprint, and `width`, which is the width of the
        desired coverage footprint, the init method initializes the desired
        coverage footprint with the given dimension to the MyClass object.

        :param length: length of the desired coverage footprint
        :type length: int
        :param width: width of the desired coverage footprint
        :type length: int
        :returns: none
        '''
        # relevant assert statements
        assert isinstance(length, int), 'Invalid input: parameter length is not type in
        assert isinstance(width, int), 'Invalid input: parameter width is not type int'
        assert length > 0, 'Invalid input: paramater length is not a positive integer'
        assert width > 0, 'Invalid input: paramater width is not a positive integer'
        # set relavant fields for MyClass object
        self.length = length
        self.width = width
        self.map = np.zeros((self.length , self.width)).astype(int)
        self.tower = self.trial = self.currentArea = 0
        self.totalArea = length * width
        colorList = []
        numColors = 100
        for i in range(numColors):
            colorList.append('%06X' % random.randint(0, 0xFFFFFF))
            colorList[i] = '#' + colorList[i]
        self.colorList = colorList

    def add(self, startLength, startWidth, endLength, endWidth):
        pass
        '''
        Given `self`, which is a MyClass object, `startlength`, which is the
```

```
start position of length for the desired coverage footprint,
`startwidth`, which is the start position of width for the desired
coverage footprint, `endlength`, which is the end position of length
for the desired coverage footprint, and `endwidth`, which is the end
position of width for the desired coverage footprint, the add method
adds the rectangle with given specifications to the desired coverage
footprint of the MyClass object

:param startLength: start position of length of the desired coverage footprint
:type length: int
:param startWidth: start position of width of the desired coverage footprint
:type length: int
:param endLength: end position of length of the desired coverage footprint
:type length: int
:param endWidth: end position of width of the desired coverage footprint
:type length: int
:returns:
            -1 if rectangle is not added to the desired coverage footprint
            0  if the desired coverage footprint is fully occupied
            1  if rectangle is added to the desired coverage footprint
'''
# relevant assert statements
assert isinstance(startLength, int), 'Invalid input: parameter startLength is n
assert isinstance(startWidth, int), 'Invalid input: parameter startWidth is not
assert isinstance(endLength, int), 'Invalid input: parameter endLength is not t
assert isinstance(endWidth, int), 'Invalid input: parameter endWidth is not typ
assert startLength in range(0, endLength), 'Invalid input: paramater startLengt
assert startWidth in range(0, endWidth), 'Invalid input: paramater startWidth i
assert endLength in range(startLength + 1, self.length + 1), 'Invalid input: pa
assert endWidth in range(startWidth + 1, self.width + 1), 'Invalid input: param
# check if the desired coverage footprint is fully occupied
if self.currentArea == self.totalArea: return 0
flag = False
# add rectangle to the desired coverage footprint
for i in range(startLength, endLength):
    for j in range(startWidth, endWidth):
        if self.map[i][j] == 0:
            self.map[i][j] = self.tower + 1
            flag = True
            self.currentArea += 1
self.trial += 1
# check if rectangle is not added to the desired coverage footprint
if flag == False: return -1
flag = False
maxcount = bestI = bestJ = bestK = bestL = -1
# find optimal tuple of (startLength, startWidth, endLength, endWidth) that giv
for i in range(startLength, endLength):
    for j in range(startWidth, endWidth):
```

```
                count = 0
                for k in range(i, endLength):
                    for l in range(j, endWidth):
                        if self.map[k][l] == self.tower + 1: count += 1
                        else:
                            flag = True
                            break
                        if count > maxcount:
                            maxcount = count
                            (bestI, bestJ, bestK, bestL) = (i, j, k, l)
                    if flag == True: break
        (xRange, yRange) =  (range(bestI, bestK + 1), range(bestJ, bestL + 1))
        # zero out areas outside of the rectangle to give the correct shape of the desi
        for i in range(startLength, endLength):
            for j in range(startWidth, endWidth):
                flag = i not in xRange or j not in yRange
                if self.map[i][j] == self.tower + 1 and flag == True:
                    self.map[i][j] = 0
                    self.currentArea -= 1
        self.tower += 1
        # return 1 since rectangle is added to the desired coverage footprint
        return 1

    def displayMap(self):
        '''
        Given `self`, which is a MyClass object, the displayMap method
        displays the desired coverage footprint for the MyClass object.

        :returns: none
        '''
        warnings.filterwarnings('ignore')
        figureSize = 5
        # set the figure and axis for the plot
        plt.figure(figsize = (figureSize, figureSize))
        plt.xlim((0, self.length))
        plt.ylim((0, self.width))
        plt.xticks(np.arange(0, self.length + 1, 1.0))
        plt.yticks(np.arange(0, self.width + 1, 1.0))
        ax = plt.subplot()
        # add colors to rectangles one by one and then display the plot
        for i in range(1, self.tower + 1):
            for j in range(0, self.length):
                for k in range(0, self.width):
                    if self.map[j][k] == i:
                        rect = patches.Rectangle((j,k), 1, 1, alpha = 1, color = self.c
                        ax.add_patch(rect)
                        plt.text(j + 0.5, k + 0.5, i,
                        horizontalalignment = 'center',
```

```
                        verticalalignment = 'center',
                        fontsize = 10)
            plt.show()

        def getNumTowers(self):
            '''
            Given `self`, which is a MyClass object, the getNumTowers method
            returns the number of towers on the desired coverage footprint for the MyClass

            :returns: number of towers on the desired coverage footprint for the MyClass ob
            '''
            return self.tower

        def getNumTrials(self):
            '''
            Given `self`, which is a MyClass object, the getNumTrials method
            returns the number of trials on the desired coverage footprint for the MyClass

            :returns: number of trials on the desired coverage footprint for the MyClass ob
            '''
            return self.trial

        def getTotalArea(self):
            '''
            Given `self`, which is a MyClass object, the getTotalArea method
            returns the total area of the desired coverage footprint for the MyClass object

            :returns: total area of the desired coverage footprint for the MyClass object
            '''
            return self.totalArea

        def getCurrentArea(self):
            '''
            Given `self`, which is a MyClass object, the getCurrentArea method
            returns the current area of the desired coverage footprint for the MyClass obje

            :returns: current area of the desired coverage footprint for the MyClass object
            '''
            return self.currentArea
```

1.0.3 Test Cases

Test Case 1 for init: negative inputs

```
In [36]: xLim = -5
         yLim = 5
         obj = MyClass(xLim, yLim)
```

```
---------------------------------------------------------------------------

AssertionError                            Traceback (most recent call last)

<ipython-input-36-2759b202a79d> in <module>()
      1 xLim = -5
      2 yLim = 5
----> 3 obj = MyClass(xLim, yLim)


<ipython-input-34-846d96ca698b> in __init__(self, length, width)
     28         assert isinstance(length, int), 'Invalid input: parameter length is not type
     29         assert isinstance(width, int), 'Invalid input: parameter width is not type i
---> 30         assert length > 0, 'Invalid input: paramater length is not a positive intege
     31         assert width > 0, 'Invalid input: paramater width is not a positive integer'
     32         # set relavant fields for MyClass object


AssertionError: Invalid input: paramater length is not a positive integer
```

Test Case 2 for init: non-integer inputs

```
In [37]: xLim = 1
         yLim = list()
         obj = MyClass(xLim, yLim)
```

```
---------------------------------------------------------------------------

AssertionError                            Traceback (most recent call last)

<ipython-input-37-386cb41a095f> in <module>()
      1 xLim = 1
      2 yLim = list()
----> 3 obj = MyClass(xLim, yLim)


<ipython-input-34-846d96ca698b> in __init__(self, length, width)
     27         # relevant assert statements
     28         assert isinstance(length, int), 'Invalid input: parameter length is not type
---> 29         assert isinstance(width, int), 'Invalid input: parameter width is not type i
     30         assert length > 0, 'Invalid input: paramater length is not a positive intege
     31         assert width > 0, 'Invalid input: paramater width is not a positive integer'


AssertionError: Invalid input: parameter width is not type int
```

Test Case 1 for add: non-integer inputs

```
In [38]: xLim = 5
         yLim = 5
         obj = MyClass(xLim, yLim)
         startLength = range(1,1)
         startWidth = (1,1)
         endLength = 4.0
         endWidth = list()
         value = obj.add(startLength, startWidth, endLength, endWidth)
```

```
        ---------------------------------------------------------------------------

        AssertionError                            Traceback (most recent call last)

        <ipython-input-38-5f7234c043f1> in <module>()
          6 endLength = 4.0
          7 endWidth = list()
    ----> 8 value = obj.add(startLength, startWidth, endLength, endWidth)


        <ipython-input-34-846d96ca698b> in add(self, startLength, startWidth, endLength, endWidt
         69         '''
         70         # relevant assert statements
    ---> 71         assert isinstance(startLength, int), 'Invalid input: parameter startLength i
         72         assert isinstance(startWidth, int), 'Invalid input: parameter startWidth is
         73         assert isinstance(endLength, int), 'Invalid input: parameter endLength is no


        AssertionError: Invalid input: parameter startLength is not type int
```

Test Case 2 for add: startLength >= endLength or startWidth >= endWidth

```
In [40]: xLim = 5
         yLim = 5
         obj = MyClass(xLim, yLim)
         startLength = 4
         startWidth = 3
         endLength = 2
         endWidth = 3
         value = obj.add(startLength, startWidth, endLength, endWidth)
         net.displayMap()
```

```
        ---------------------------------------------------------------------------

        AssertionError                            Traceback (most recent call last)
```

```
<ipython-input-40-c3a679d97868> in <module>()
  6 endLength = 2
  7 endWidth = 3
----> 8 value = obj.add(startLength, startWidth, endLength, endWidth)
  9 net.displayMap()


<ipython-input-34-846d96ca698b> in add(self, startLength, startWidth, endLength, endWidt
 73         assert isinstance(endLength, int), 'Invalid input: parameter endLength is no
 74         assert isinstance(endWidth, int), 'Invalid input: parameter endWidth is not
---> 75         assert startLength in range(0, endLength), 'Invalid input: paramater startLe
 76         assert startWidth in range(0, endWidth), 'Invalid input: paramater startWidt
 77         assert endLength in range(startLength + 1, self.length + 1), 'Invalid input:


AssertionError: Invalid input: paramater startLength is not in range
```

Test Case 3 for add: Out of Bounds

In [41]:
```
xLim = 6
yLim = 6
obj = MyClass(xLim, yLim)
startLength = -1
startWidth = 0
endLength = 5
endWidth = 5
value = obj.add(startLength, startWidth, endLength, endWidth)
net.displayMap()
```

```
---------------------------------------------------------------------------

AssertionError                            Traceback (most recent call last)

<ipython-input-41-fc44631c6a31> in <module>()
  6 endLength = 5
  7 endWidth = 5
----> 8 value = net.add(startLength, startWidth, endLength, endWidth)
  9 net.displayMap()


<ipython-input-34-846d96ca698b> in add(self, startLength, startWidth, endLength, endWidt
 73         assert isinstance(endLength, int), 'Invalid input: parameter endLength is no
 74         assert isinstance(endWidth, int), 'Invalid input: parameter endWidth is not
---> 75         assert startLength in range(0, endLength), 'Invalid input: paramater startLe
 76         assert startWidth in range(0, endWidth), 'Invalid input: paramater startWidt
```

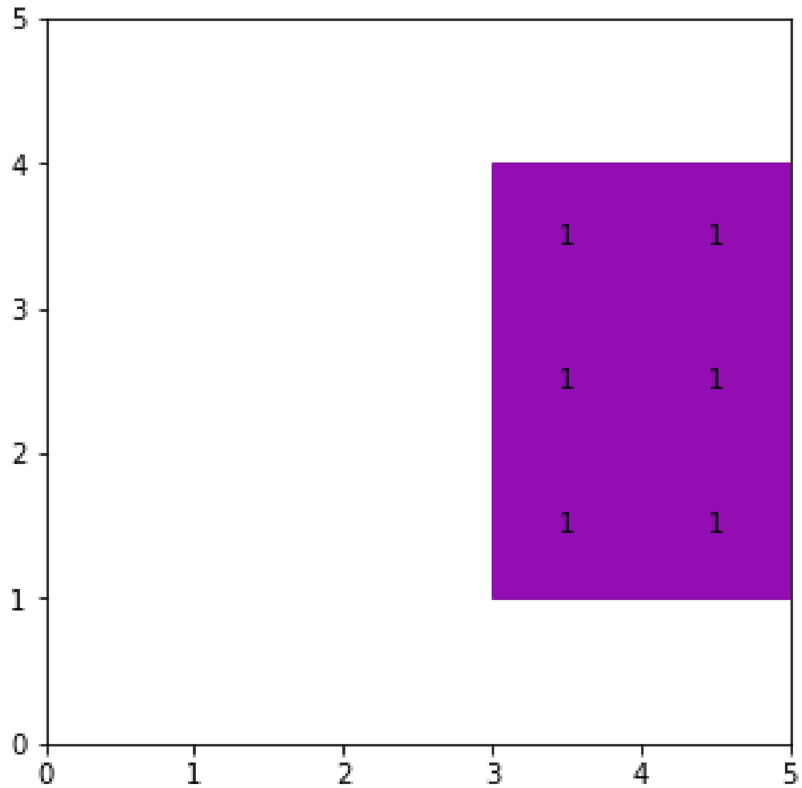
```
77         assert endLength in range(startLength + 1, self.length + 1), 'Invalid input:
```

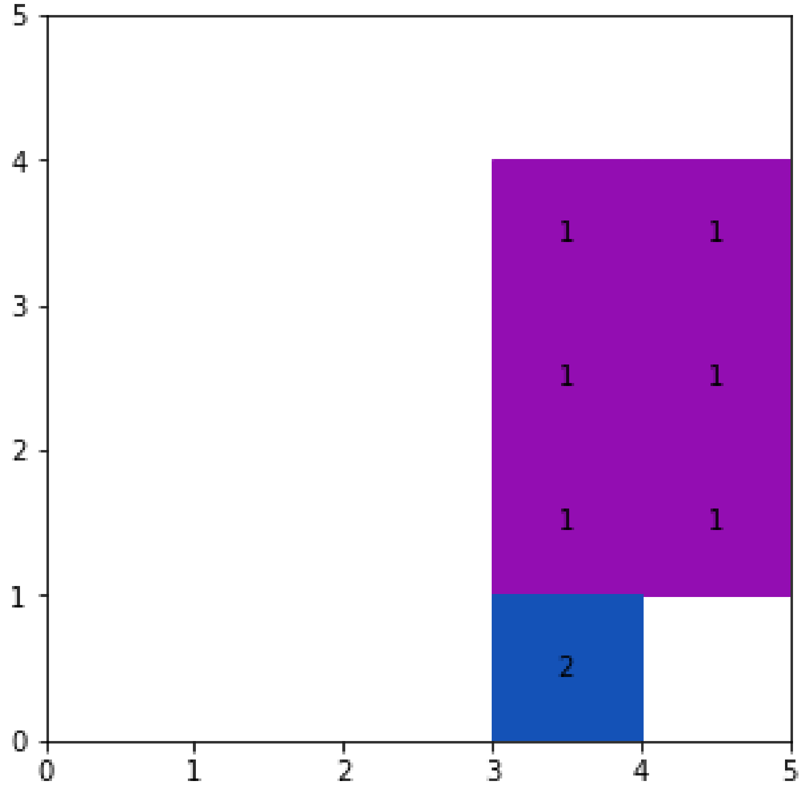
```
AssertionError: Invalid input: paramater startLength is not in range
```

(1) Given an overall desired coverage footprint and a sequence of n communications towers, what is the resulting resolved coverage?

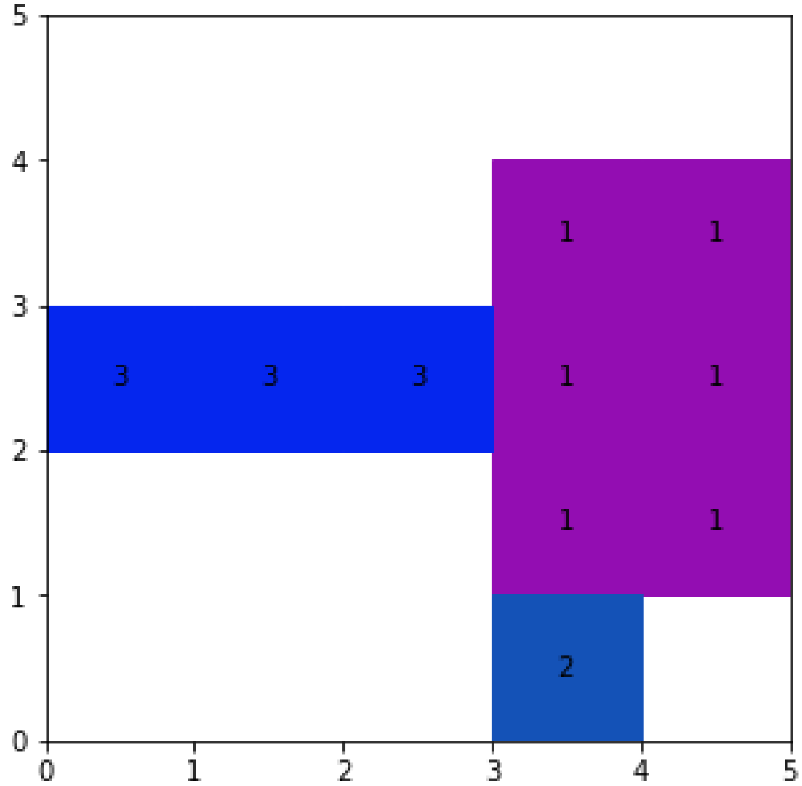
```
As shown, the code shows that the current area of map is 15 for 10 communication towers and
towers are added sequentially, as can be seen in the displayed maps below step by step.
Note: The numbers and colors show the type of color and tower numbers associated with the rectan
```

```
In [42]: n = 10
         xLim = yLim = 5
         obj = MyClass(xLim, yLim)
         for j in range(0, n):
             startLength = random.randint(0, xLim - 2)
             startWidth = random.randint(0, yLim - 2)
             endLength = random.randint(startLength + 1, xLim)
             endWidth = random.randint(startWidth + 1, yLim)
             value = obj.add(startLength, startWidth, endLength, endWidth)
             if value == 0: break
             obj.displayMap()
         print ('Resulting resolved coverage: ' + str(obj.getCurrentArea()) + ' for ' + str(n) +
```

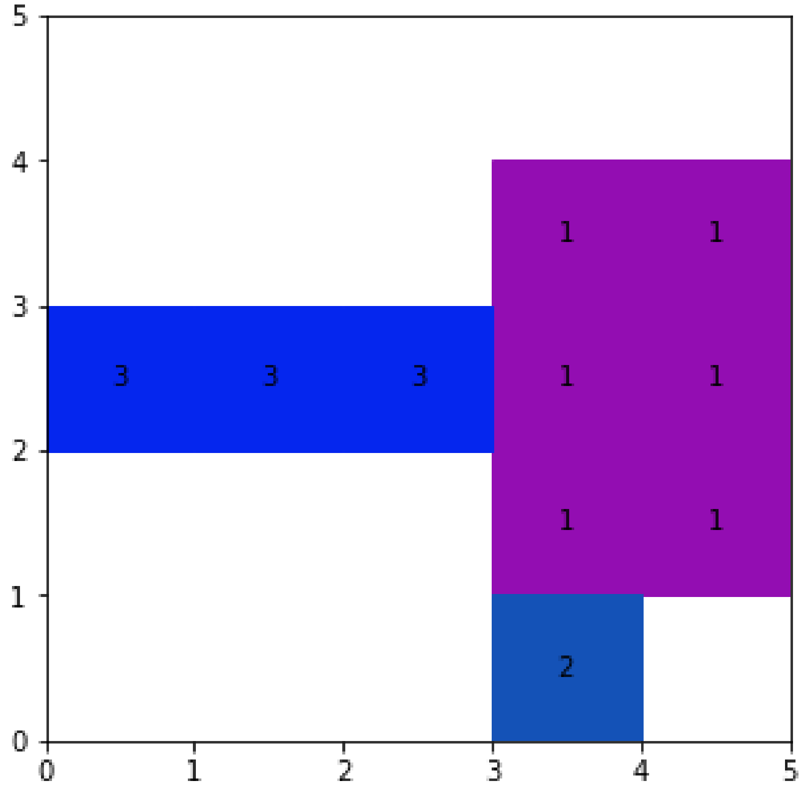




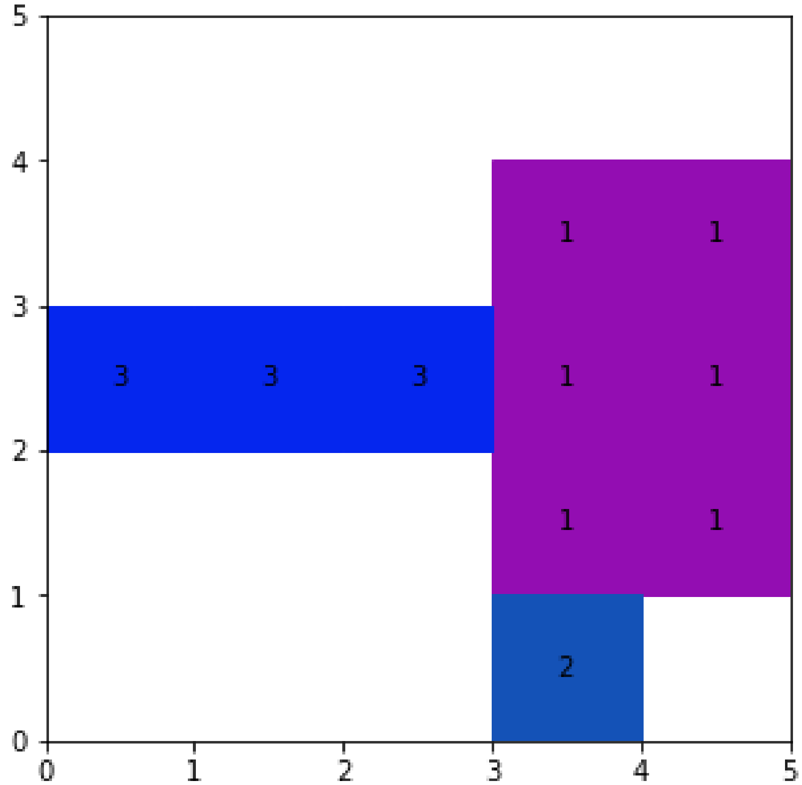
5
4
3
2
1
0
0
1
2
3
4
5
1
1
1
1
1
1
2

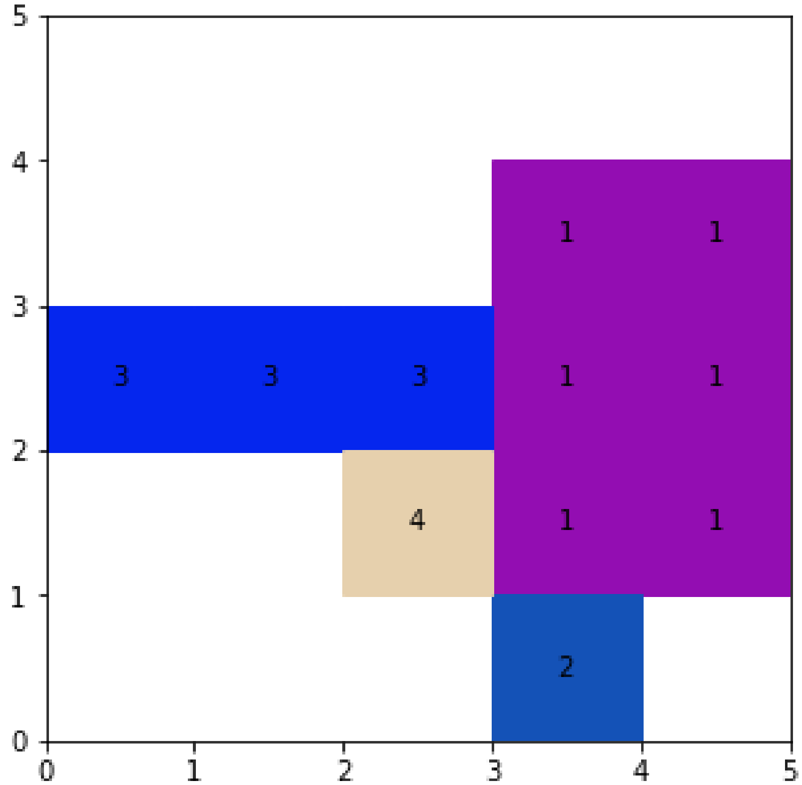


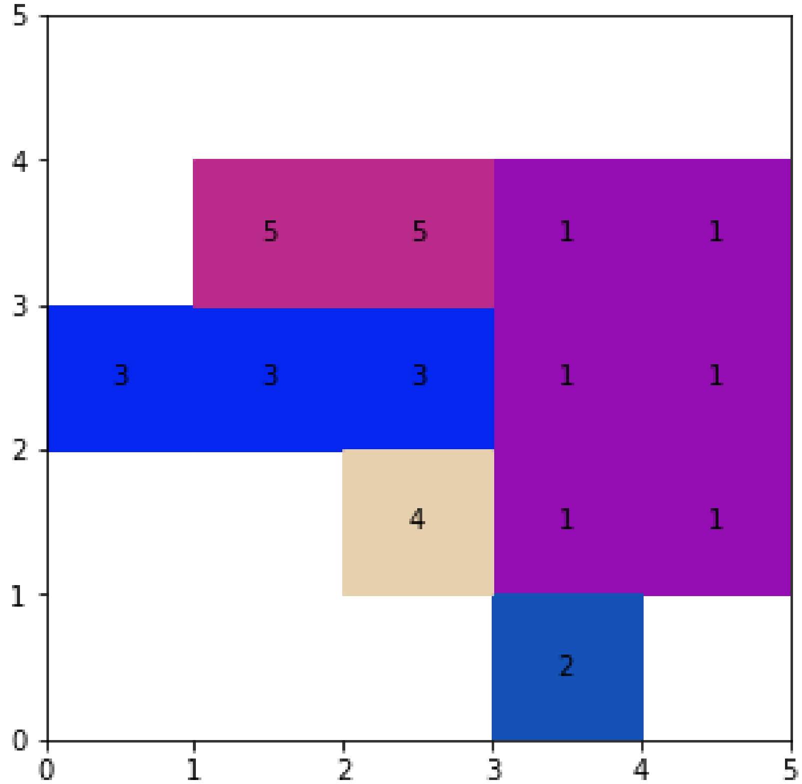
5
4
3
2
1
0
0
1
2
3
4
5
1
1
3
3
3
1
1
1
1
2

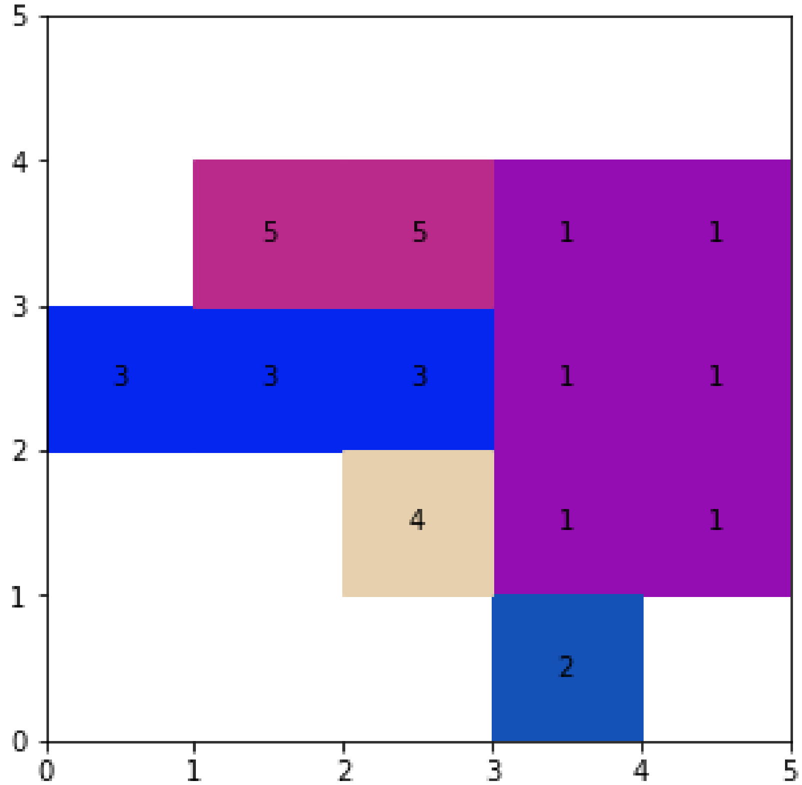


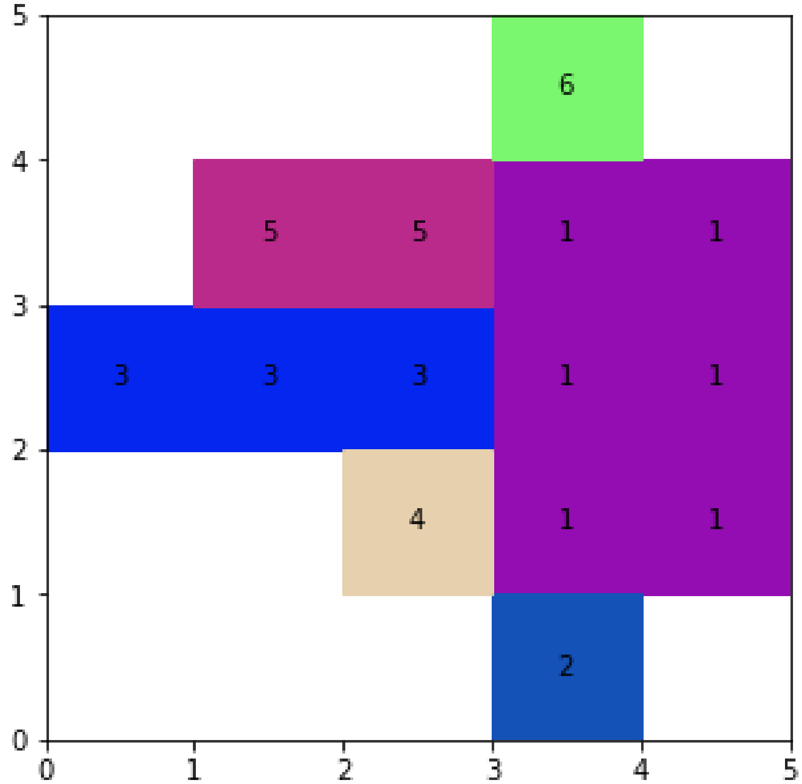
1
1
3
3
3
1
1
1
1
2
0
1
2
3
4
5


5
4
3
2
1
0
0
1
2
3
4
5
1 1
3 3 3 1 1
1 1
2

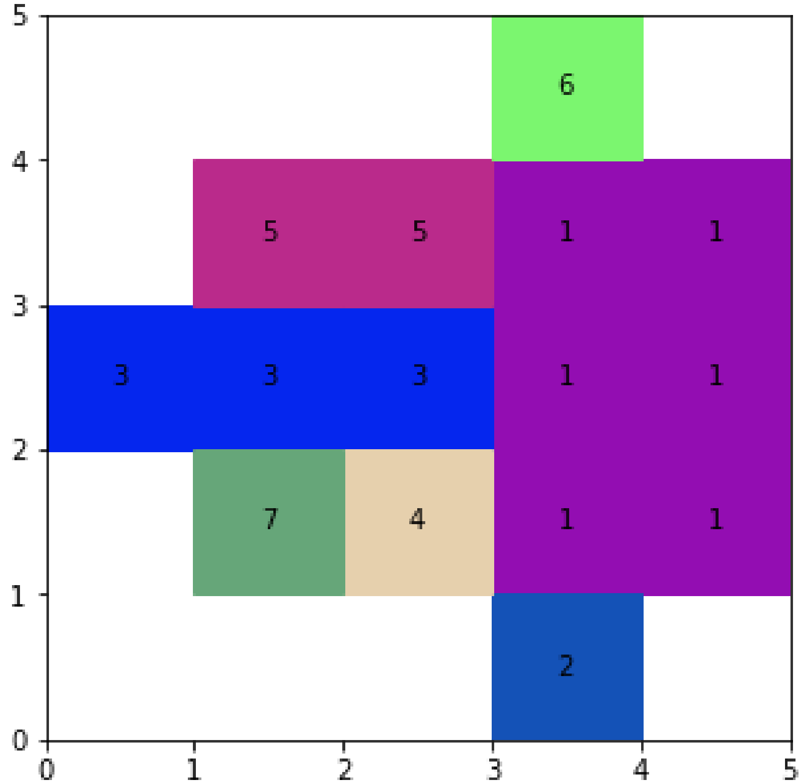

5
4
3
2
1
0
0
1
2
3
4
5
1 1
3 3 3 1 1
4 1 1
2


5
4
3
2
1
0
0
1
2
3
4
5
5
5
1
1
3
3
3
1
1
4
1
1
2


5
4
3
2
1
0
0
1
2
3
4
5
5 5 1 1
3 3 3 1 1
4 1 1
2



6
5
5
1
1
3
3
3
1
1
4
1
1
2
0
1
2
3
4
5



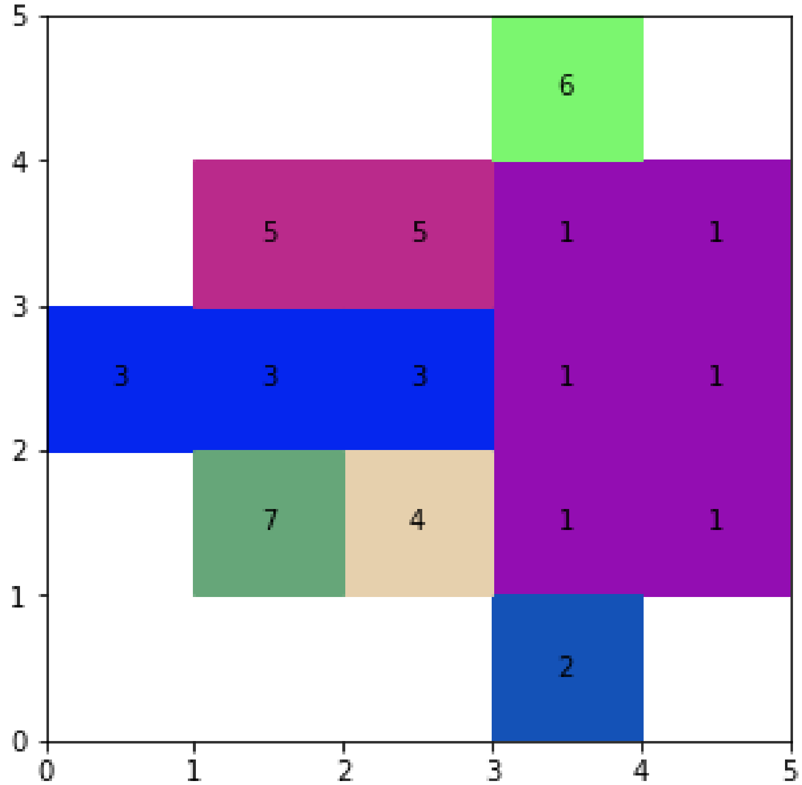

```
Resulting resolved coverage: 15 for 10 communication towers
```

(2) What is the total area of coverage relative to the desired total coverage area of the original footprint? That is, are there any gaps in coverage?

As shown, the code shows that the total area of coverage is 60% for 10 communication towers. Sin
cell already addressed the step-by-step sequentially adding of towers on map, this cell simply j
the final schematic of the map along with the total coverage area on the map.
Note: The numbers and colors show the type of color and tower numbers associated with the rectan

```
In [43]: print ('Total area of coverage relative to desired total coverage area: ' + str(obj.get
         obj.displayMap()
```

```
Total area of coverage relative to desired total coverage area: 60.0%
```

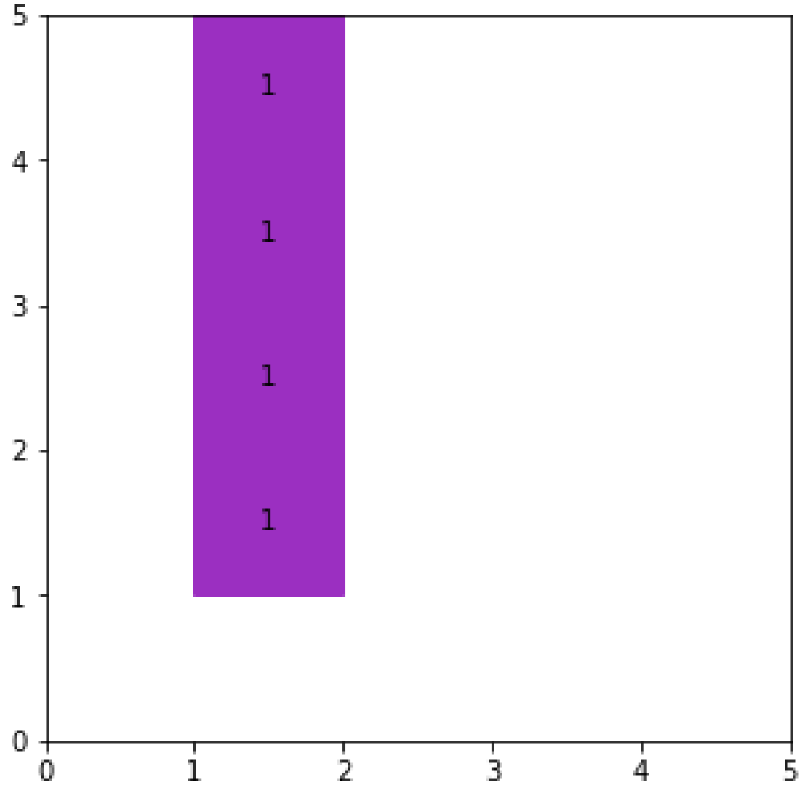


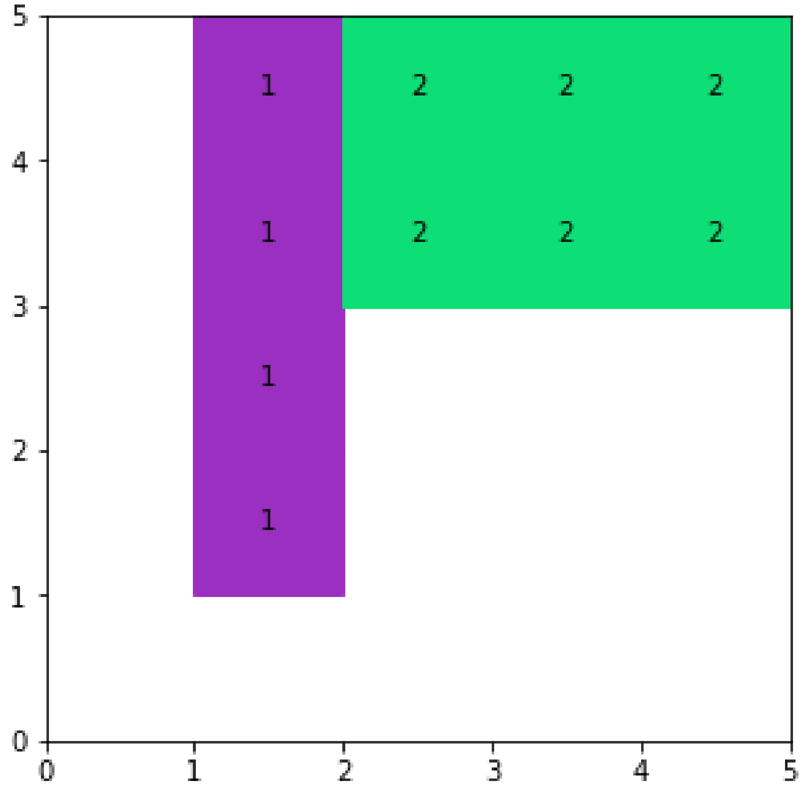
(3) On average, how many communications towers are required before full coverage is obtained?

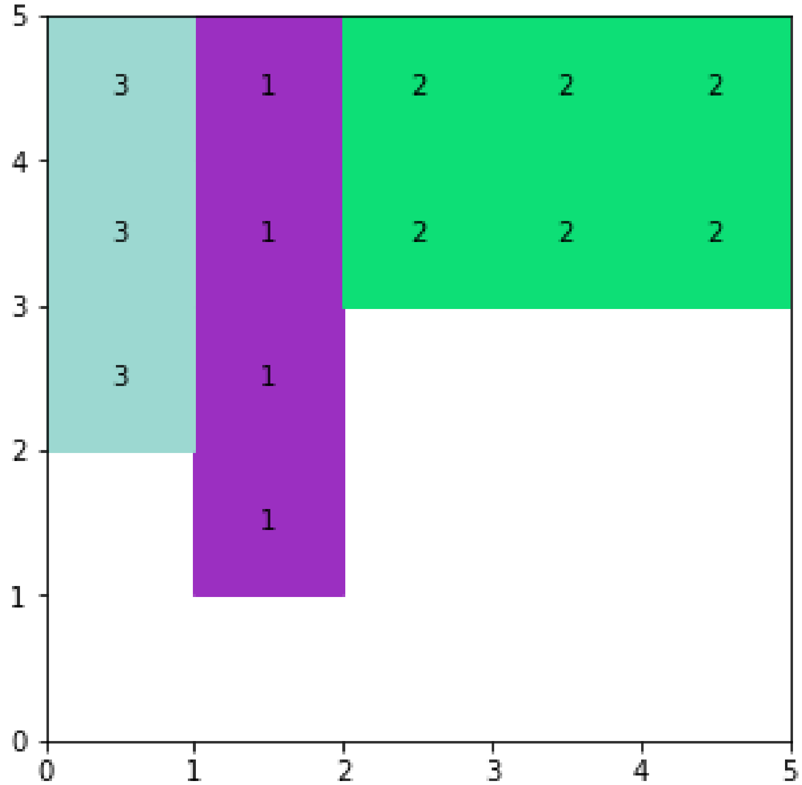
```
As shown, the code shows across 10 experiments, it takes on an average 14 communication towers
until full coverage is reached.
Agaim, the process of adding towers is done sequentially to show the state of the map step-by-st
Note: The numbers and colors show the type of color and tower numbers associated with the rectan
```

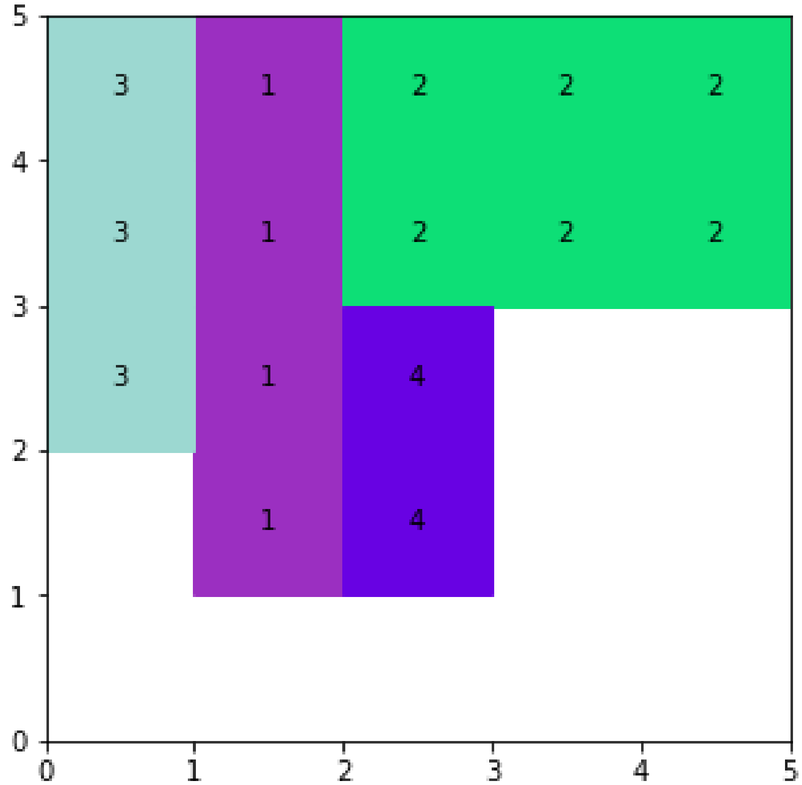
```
In [45]: numExperiment = 10
         maxTrial = 100
         aList = list([0] * (numExperiment))
         currentSum = 0
         # after this loop found numExperiment full maps
         for i in range(0, numExperiment):
             obj = MyClass(xLim, yLim)
             # after this loop we found one full map
             for j in range(0, maxTrial):
                 startLength = random.randint(0, xLim - 2)
                 startWidth = random.randint(0, yLim - 2)
                 endLength = random.randint(startLength + 1, xLim)
                 endWidth = random.randint(startWidth + 1, yLim)
```

```
        value = obj.add(startLength, startWidth, endLength, endWidth)
        if value == 0:
            currentSum += obj.tower
            break
        obj.displayMap()
print ('On average across ' + str(numExperiment) + ' experiments, it takes ' + str(curr
```

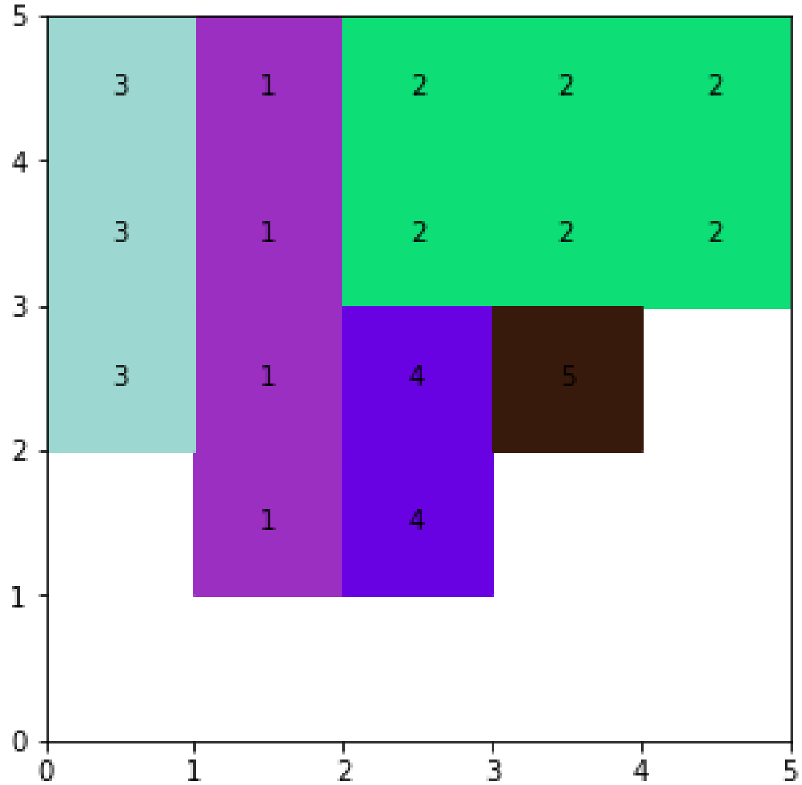


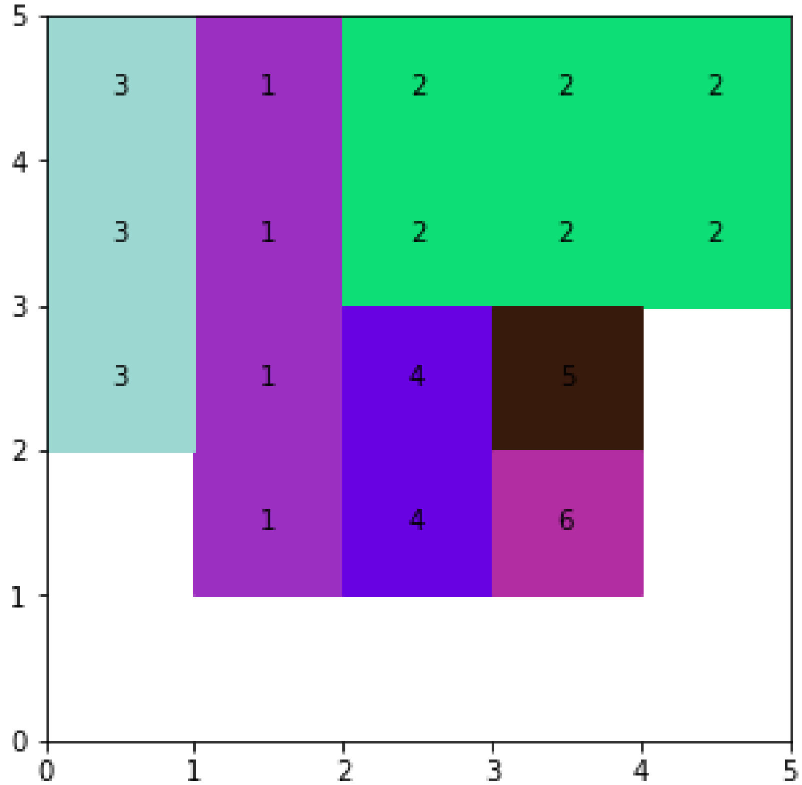

1
2
2
2
1
2
2
2
1
1
0
1
2
3
4
5

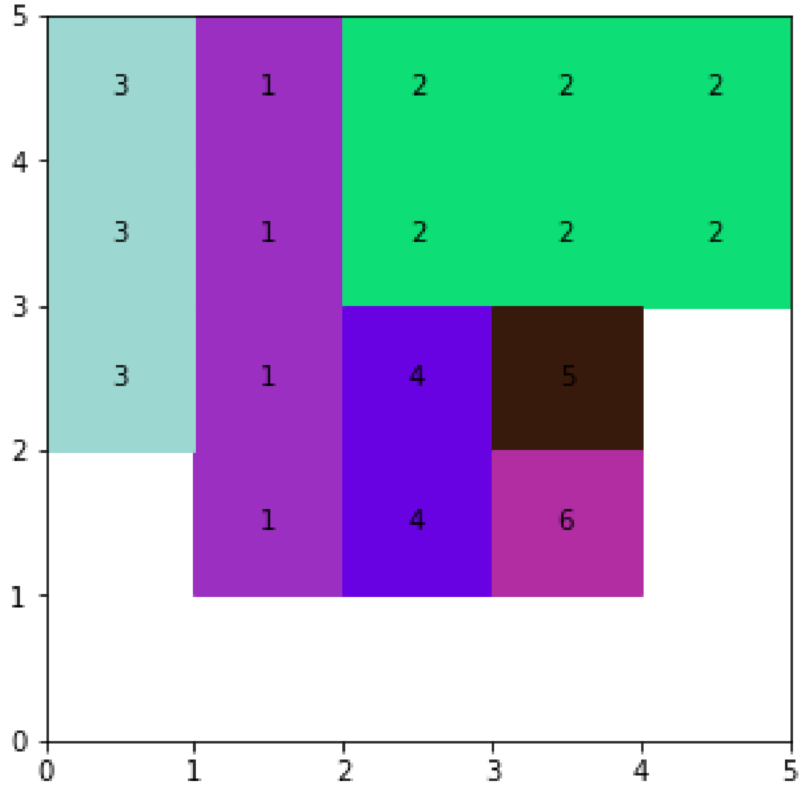

3
1
2
2
2
3
1
2
2
2
3
1
1
0
1
2
3
4
5
0
1
2
3
4
5

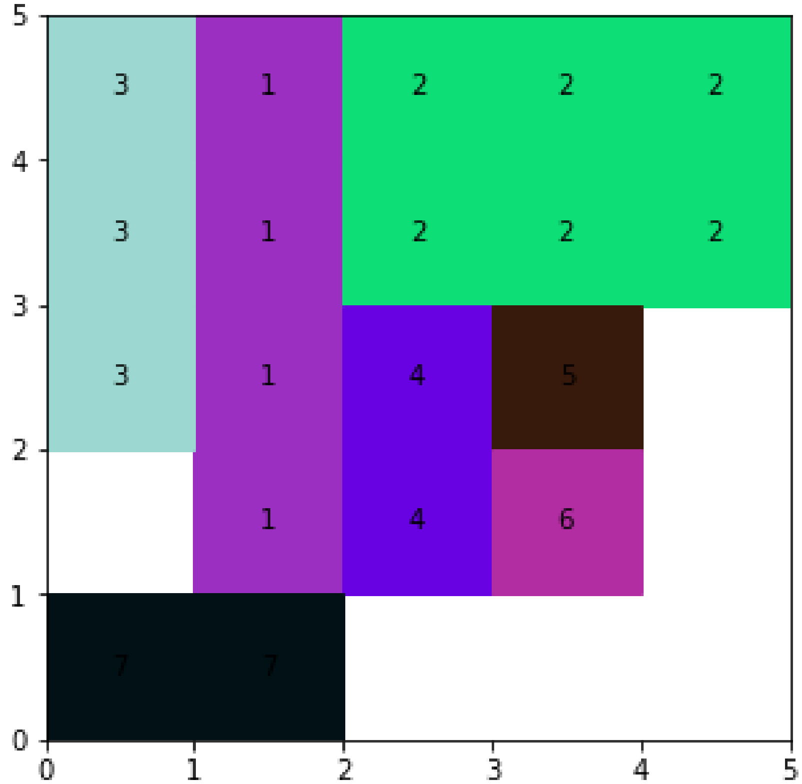


3
1
2
2
2
3
1
2
2
2
3
1
4
1
4
0
1
2
3
4
5
0
1
2
3
4
5

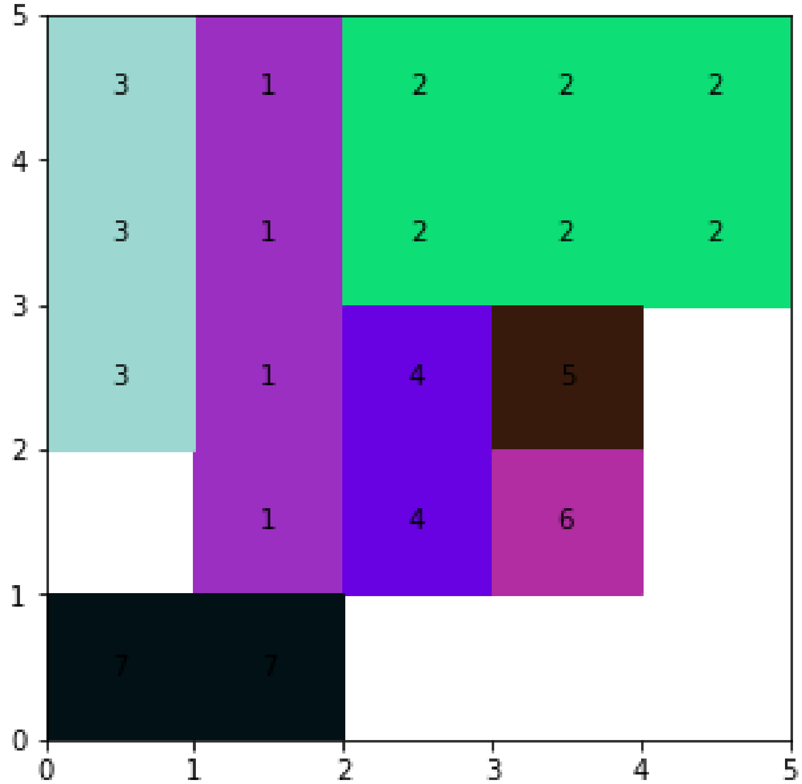

3
1
2
2
2
3
1
2
2
2
3
1
4
5
1
4
0
1
2
3
4
5

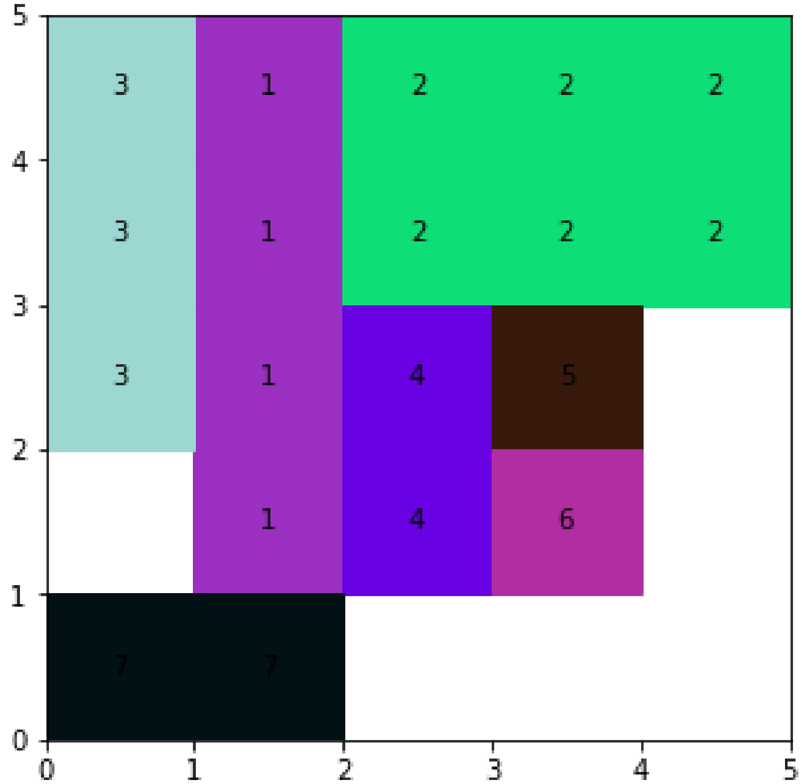

3
1
2
2
2
3
1
2
2
2
3
1
4
5
1
4
6
0
1
2
3
4
5


3
1
2
2
2
3
1
2
2
2
3
1
4
5
1
4
6
0
1
2
3
4
5
0
1
2
3
4
5

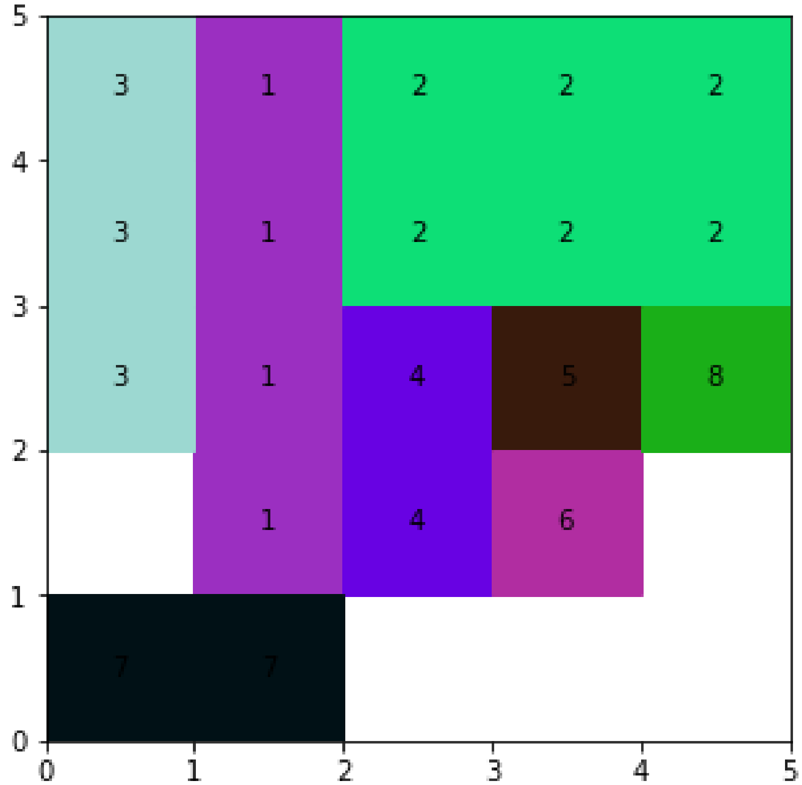


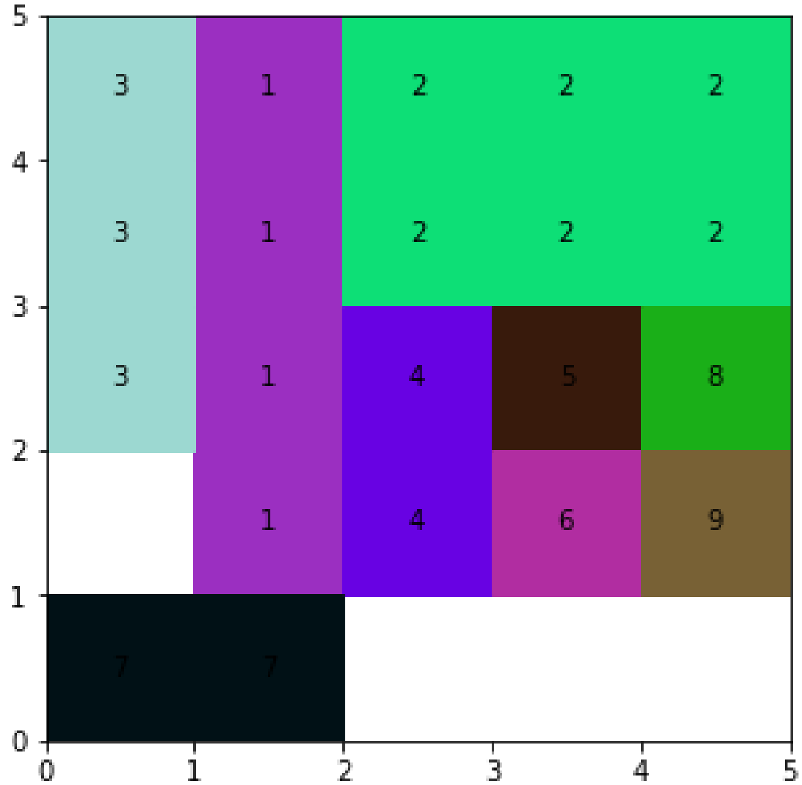
3
1
2
2
2
3
1
2
2
2
3
1
4
5
1
4
6
7
7
0
1
2
3
4
5
0
1
2
3
4
5


5
4
3
2
1
0
0
1
2
3
4
5
3 1 2 2 2
3 1 2 2 2
3 1 4 5
1 4 6
7 7

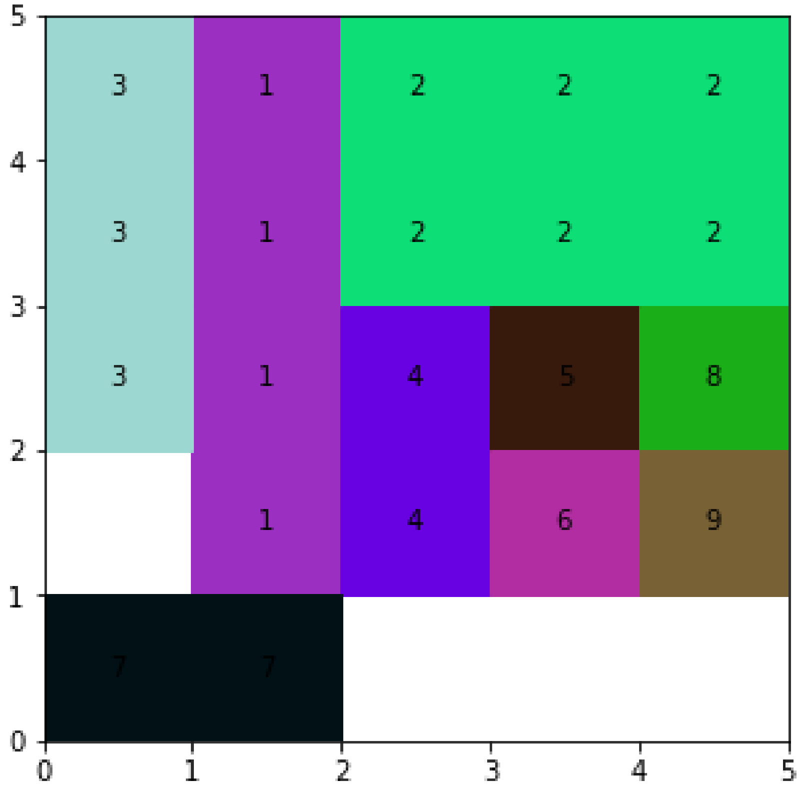


5
4
3
2
1
0
3
1
2
2
2
3
1
2
2
2
3
1
4
5
1
4
6
7
7
0
1
2
3
4
5

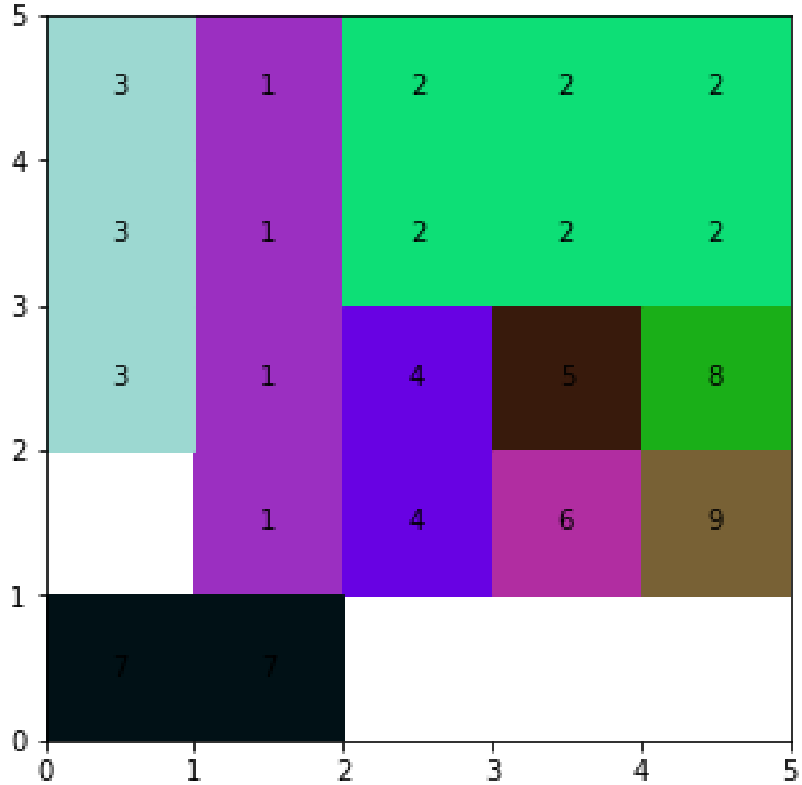

3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
1
4
6
7
7
0
1
2
3
4
5
0
1
2
3
4
5



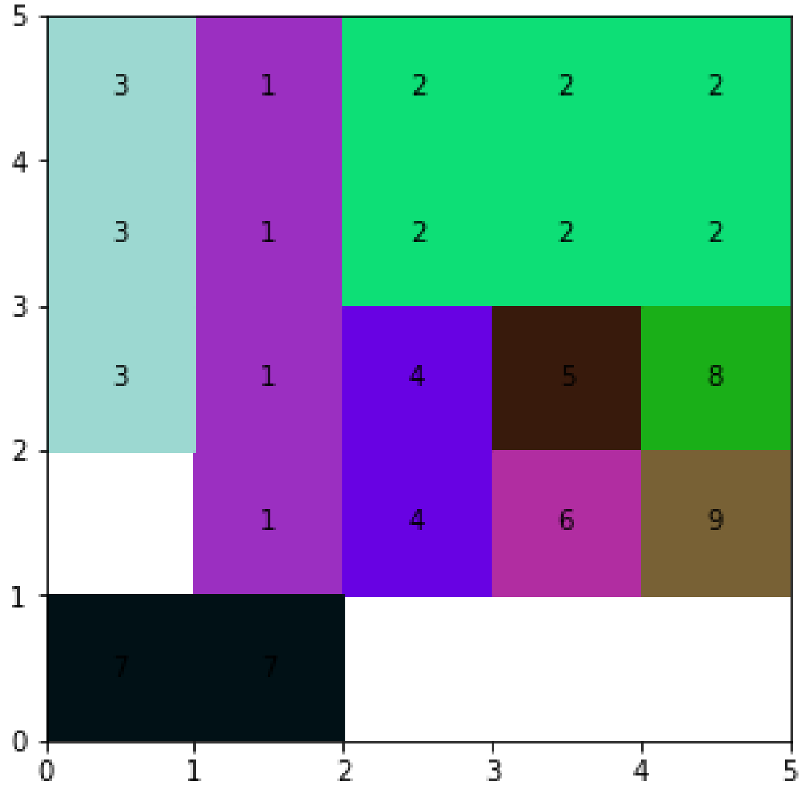
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
1
4
6
9
7
7
0
1
2
3
4
5

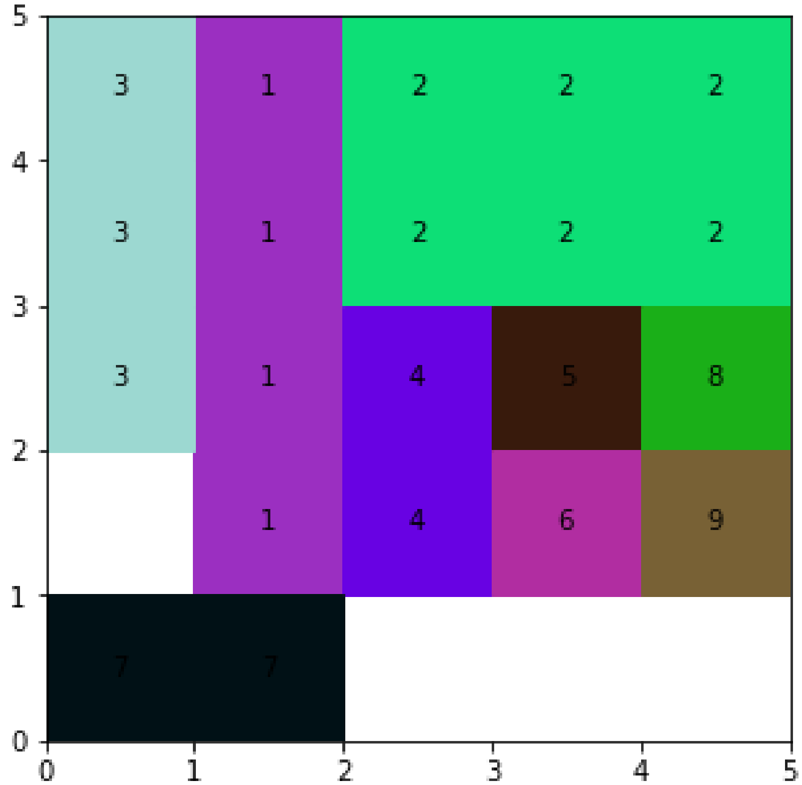


3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
1
4
6
9
7
7
0
1
2
3
4
5

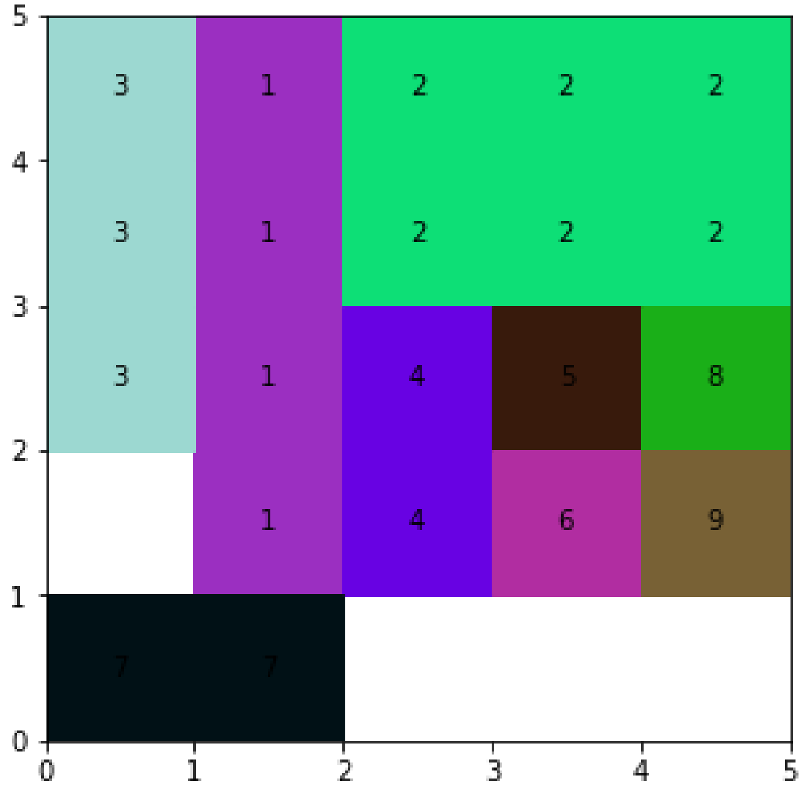


3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
1
4
6
9
7
7
0
1
2
3
4
5

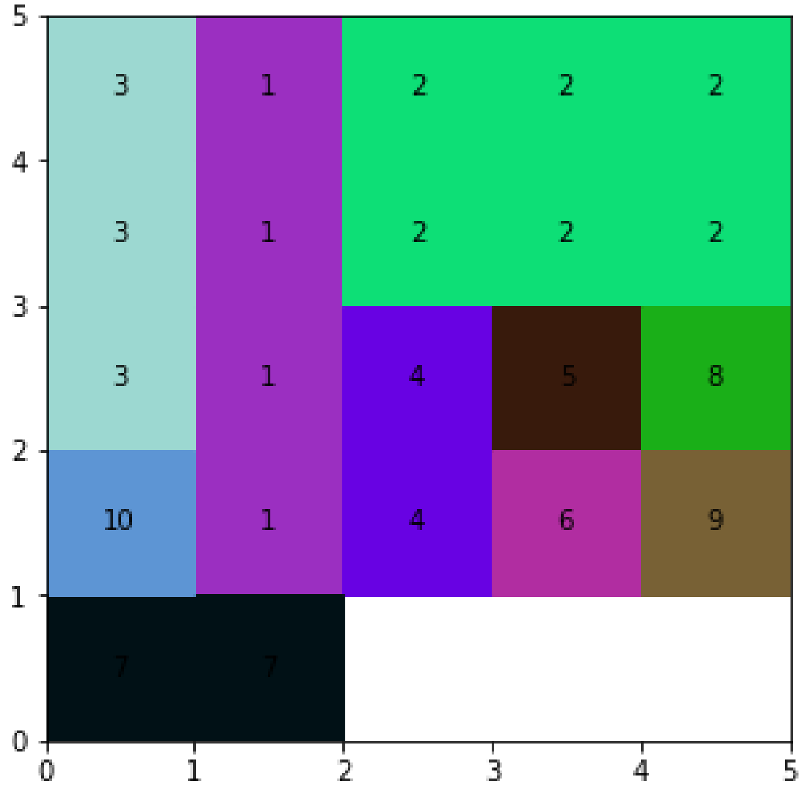

3 1 2 2 2
3 1 2 2 2
3 1 4 5 8
1 4 6 9
7 7
0 1 2 3 4 5
0 1 2 3 4 5



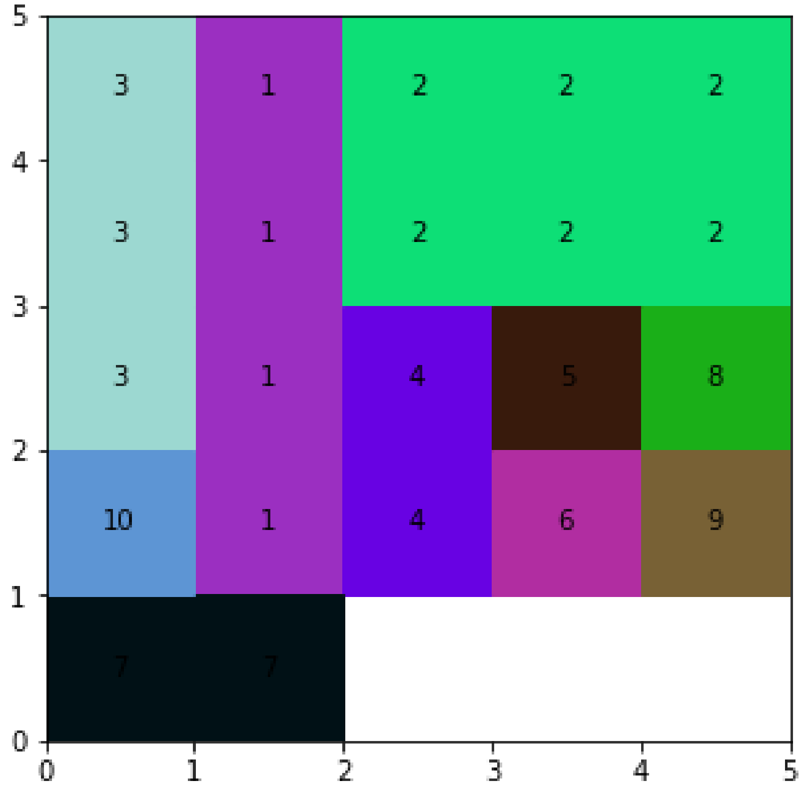
5
4
3
2
1
0
0
1
2
3
4
5
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
1
4
6
9
7
7



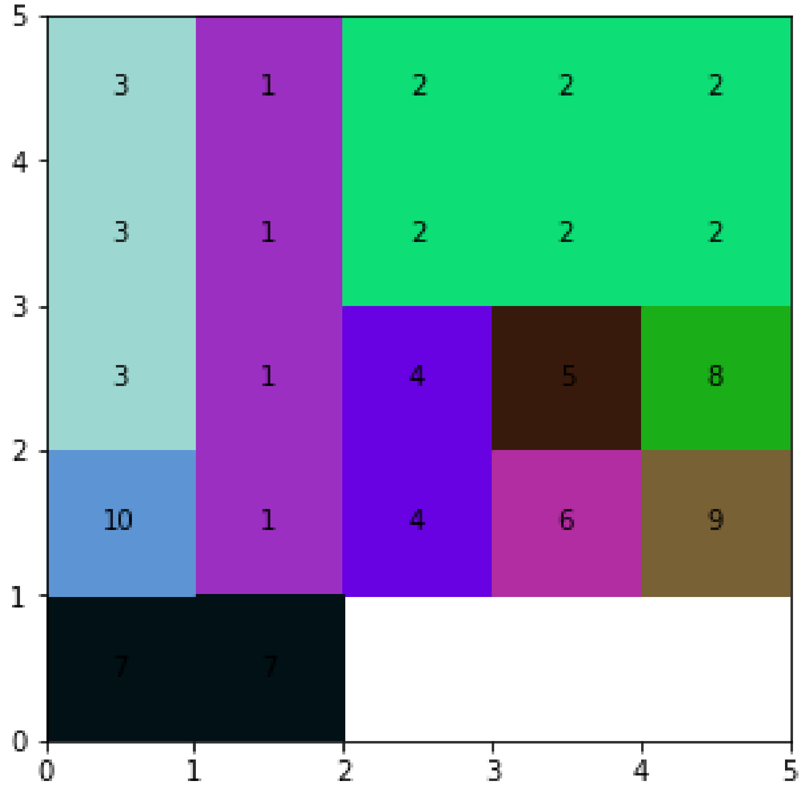
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
1
4
6
9
7
7
0
1
2
3
4
5



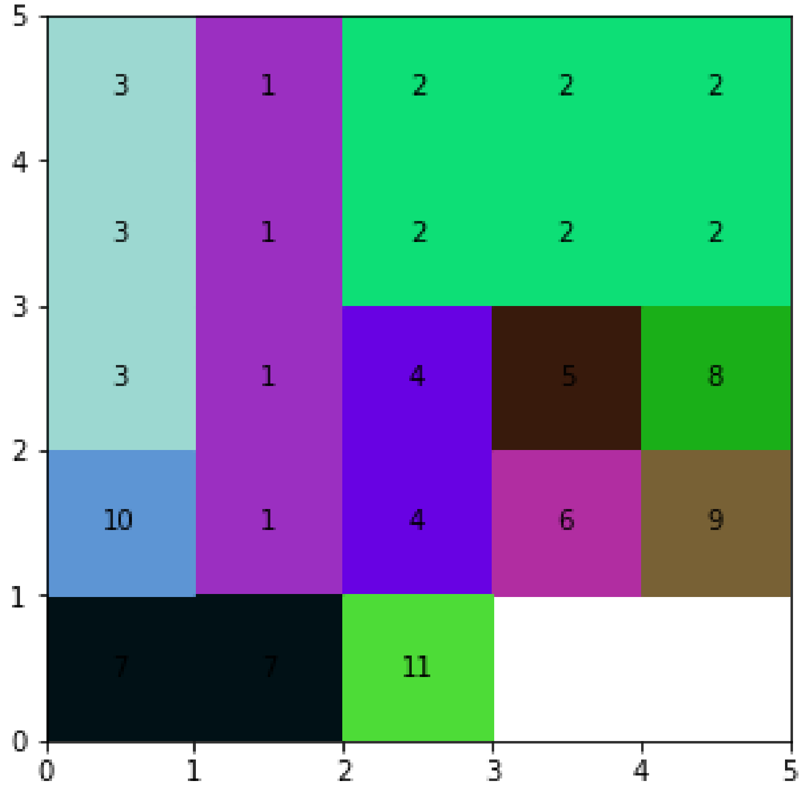
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
0
1
2
3
4
5

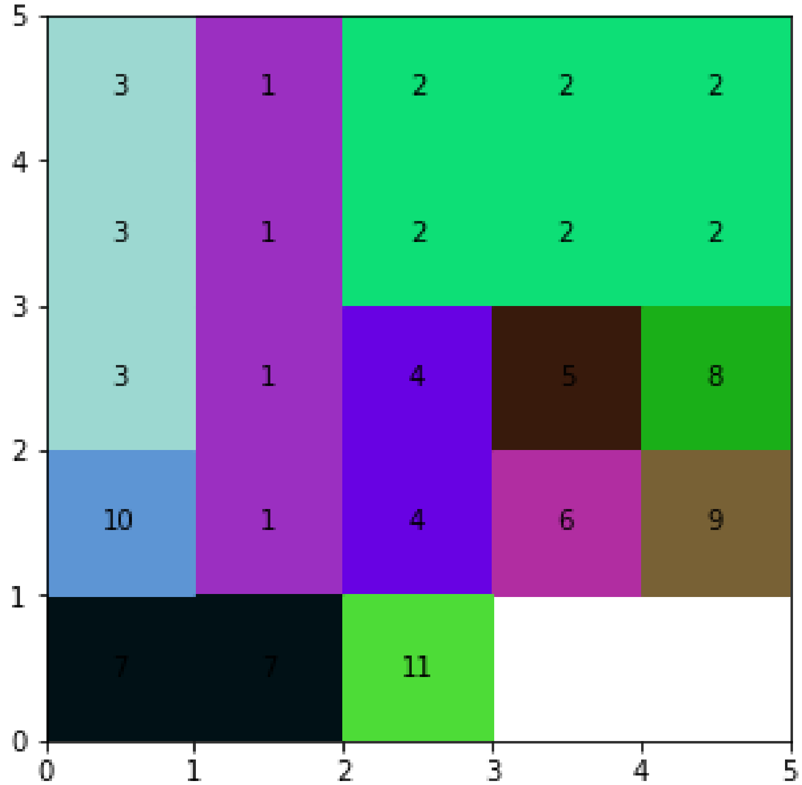


3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
0
1
2
3
4
5

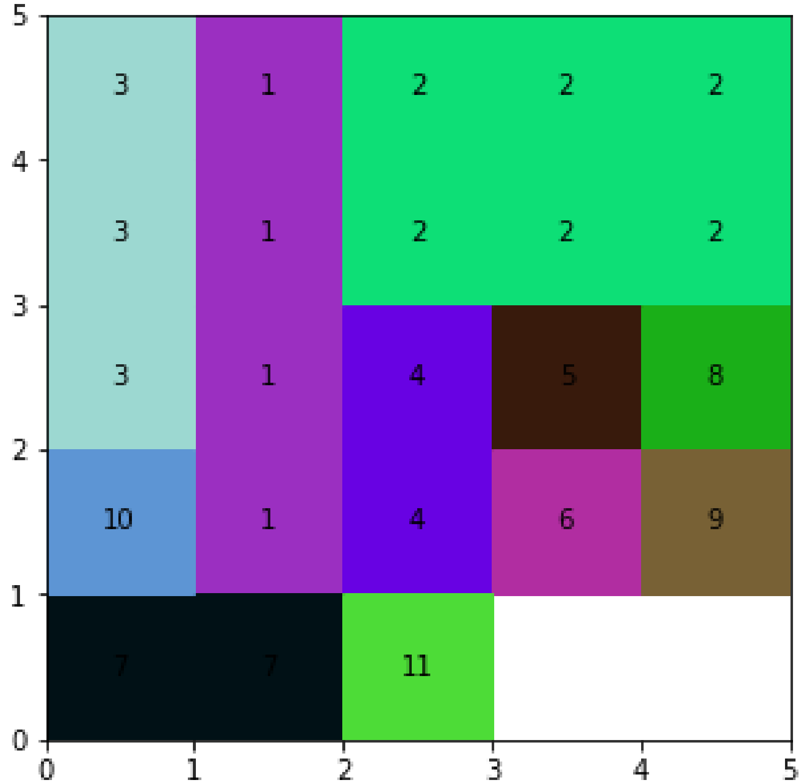


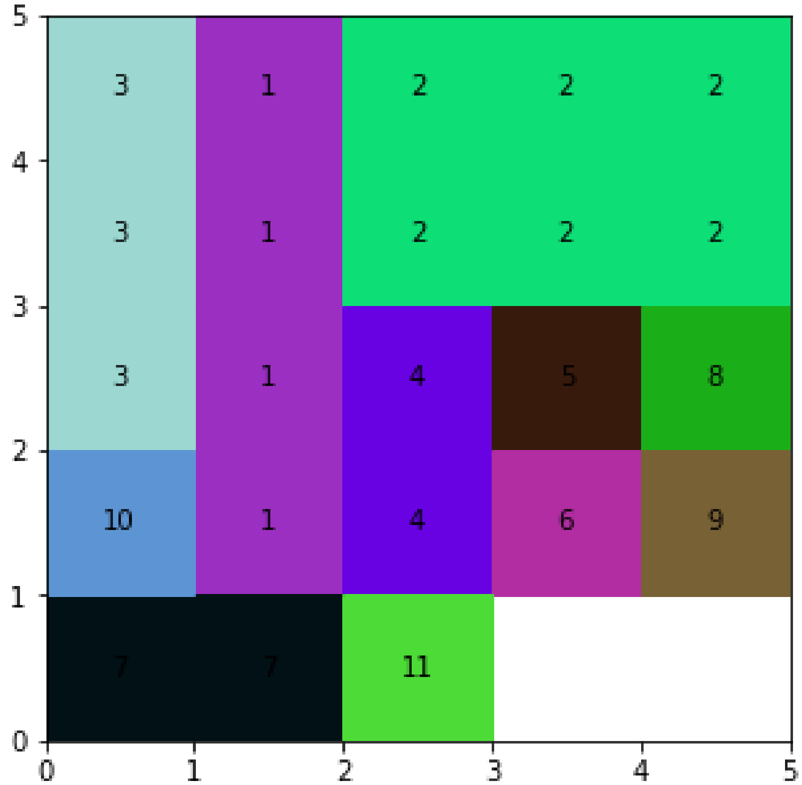
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
0
1
2
3
4
5

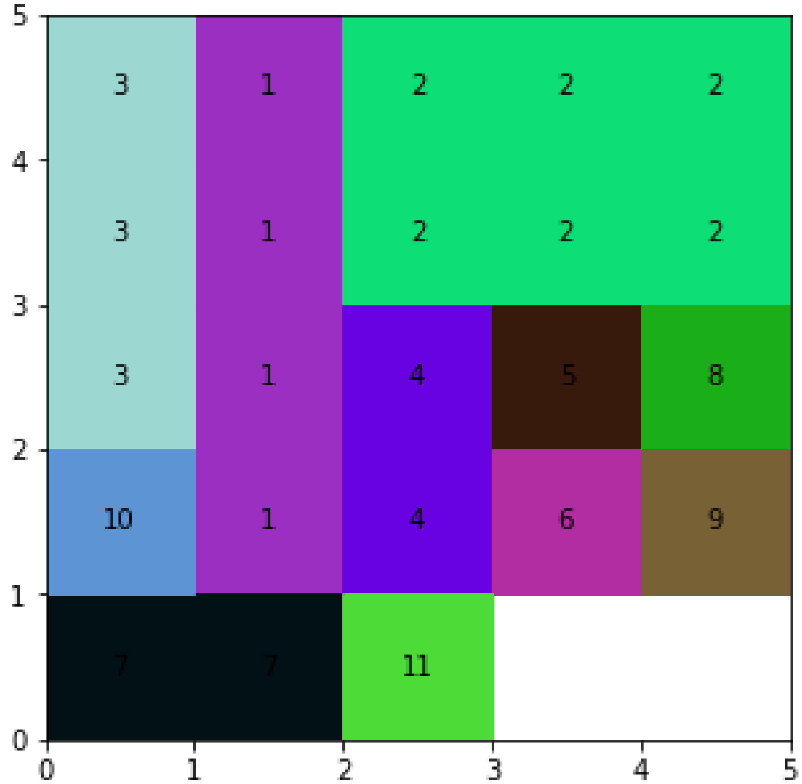

3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
0
1
2
3
4
5



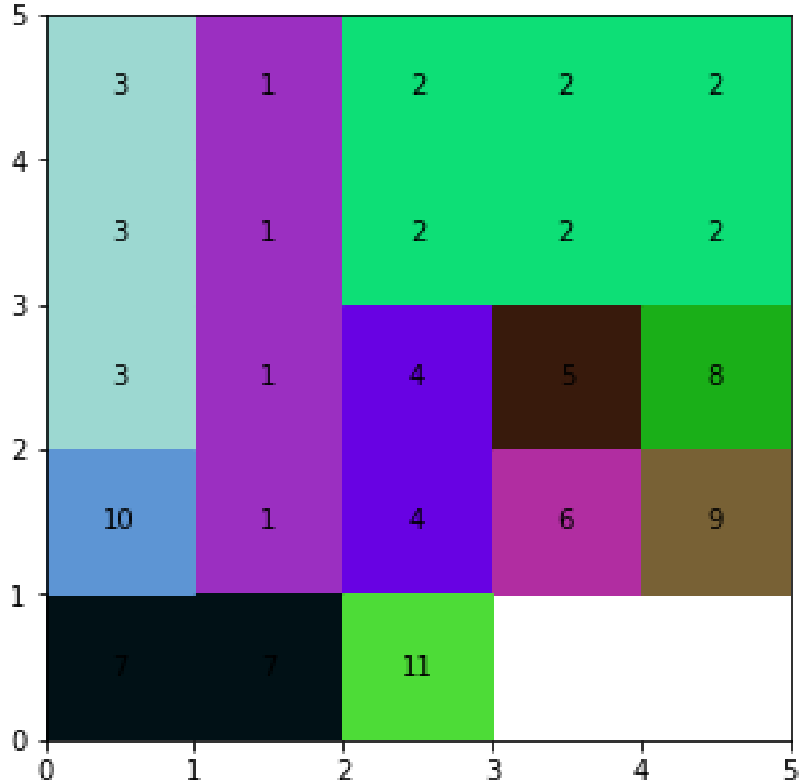
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
0
1
2
3
4
5
0
1
2
3
4
5

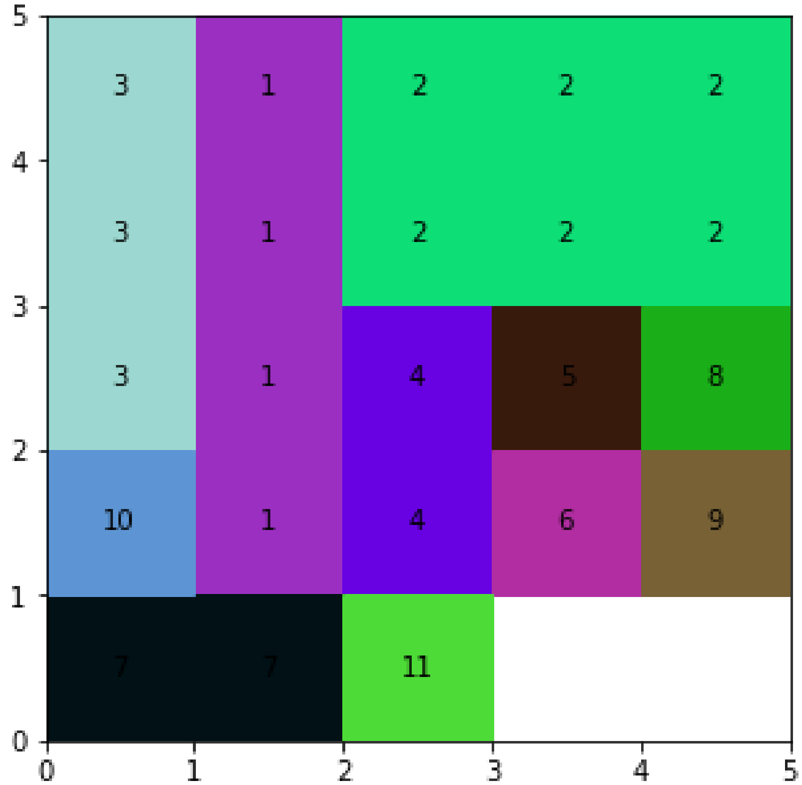

3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
0
1
2
3
4
5
0
1
2
3
4
5


3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
0
1
2
3
4
5

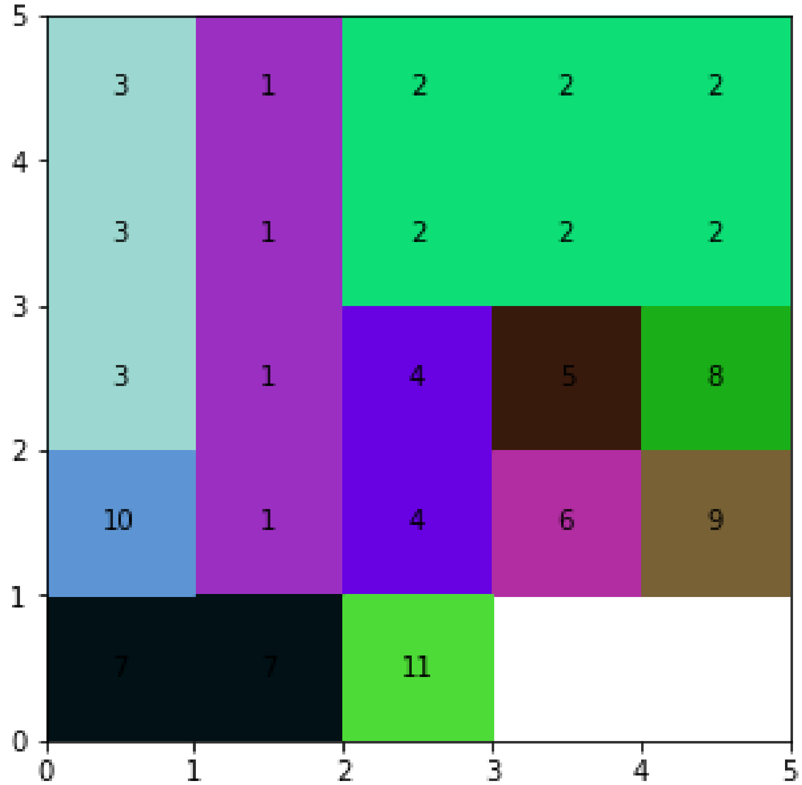


3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
0
1
2
3
4
5
0
1
2
3
4
5

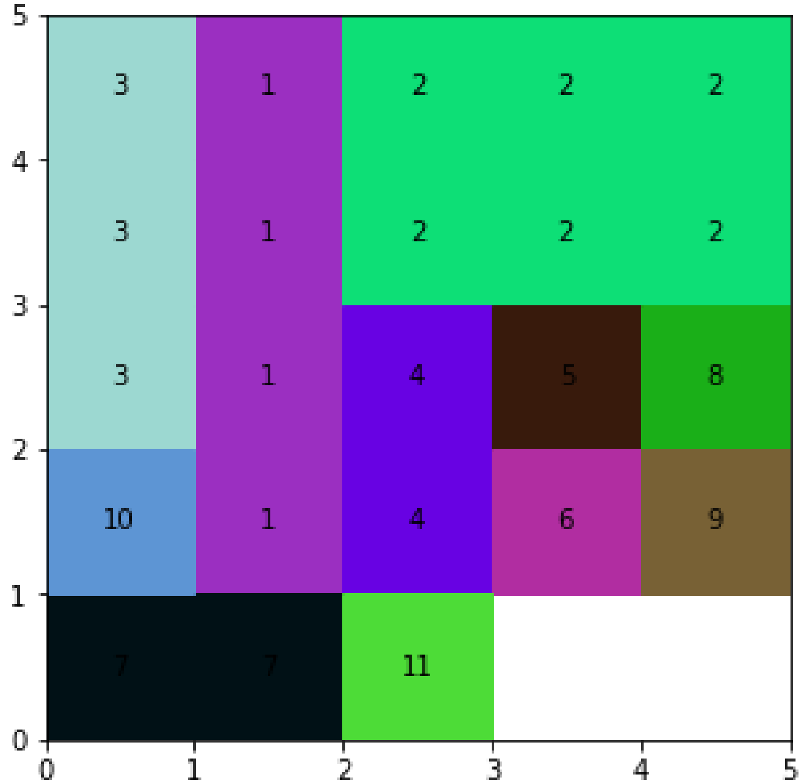

5
4
3
2
1
0
0
1
2
3
4
5
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11



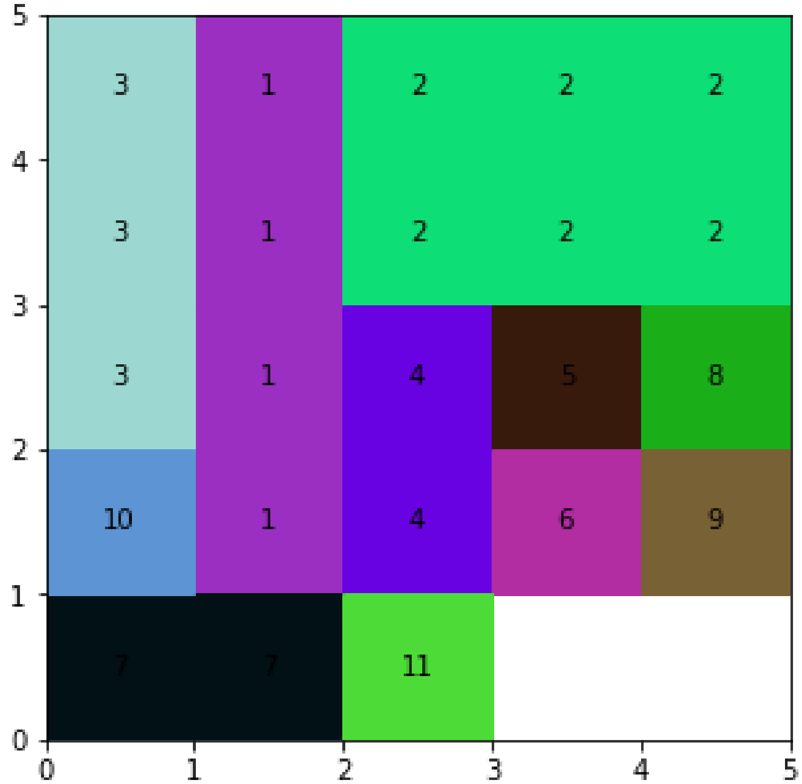
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
0
1
2
3
4
5



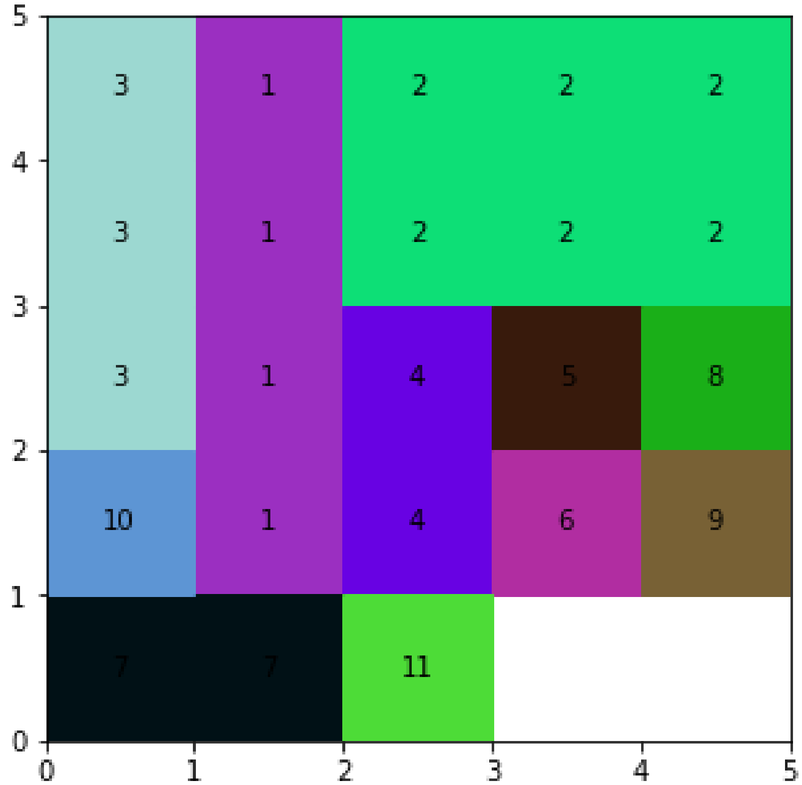
3 1 2 2 2
3 1 2 2 2
3 1 4 5 8
10 1 4 6 9
7 7 11
0 1 2 3 4 5

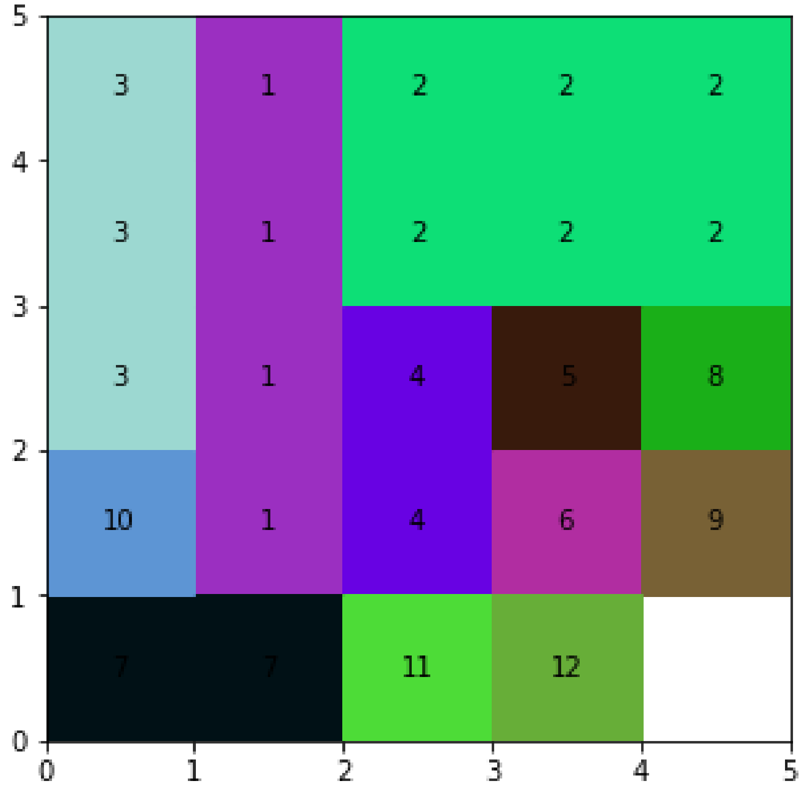


3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
0
1
2
3
4
5

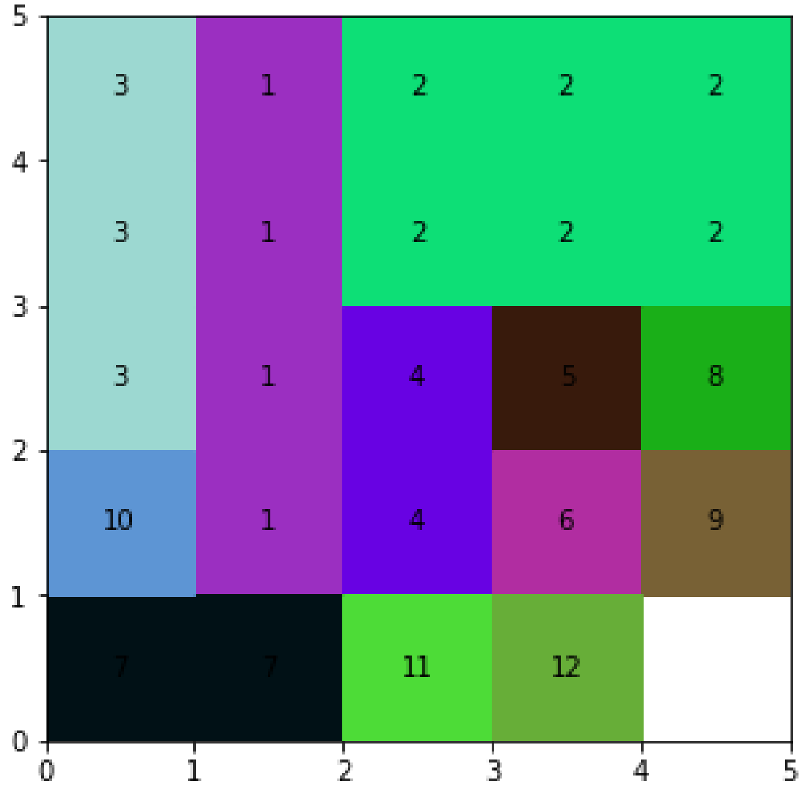


3 1 2 2 2
3 1 2 2 2
3 1 4 5 8
10 1 4 6 9
7 7 11
0 1 2 3 4 5

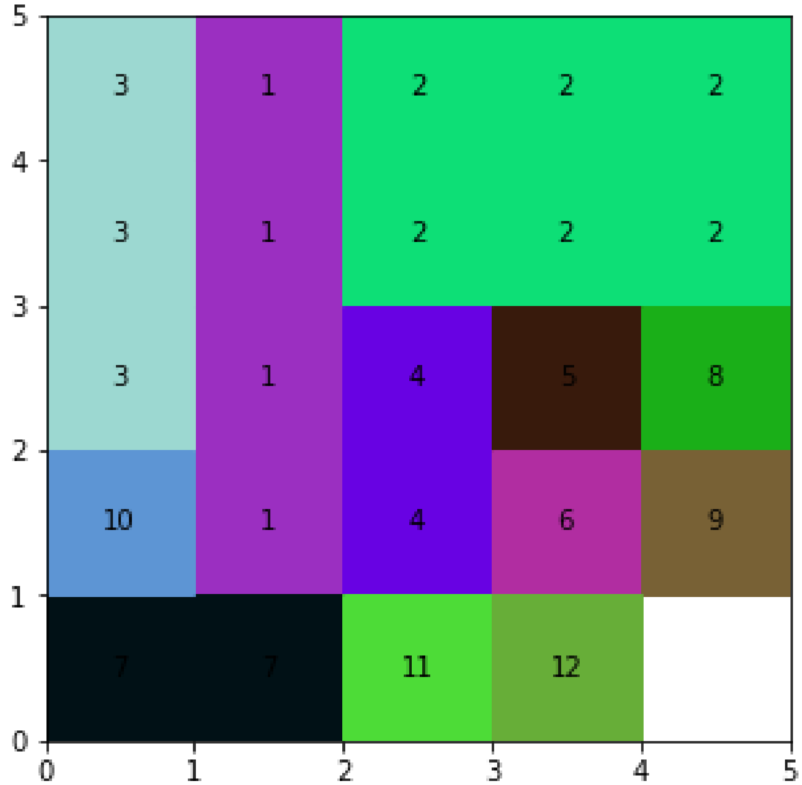

5
4
3
2
1
0
0
1
2
3
4
5
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11

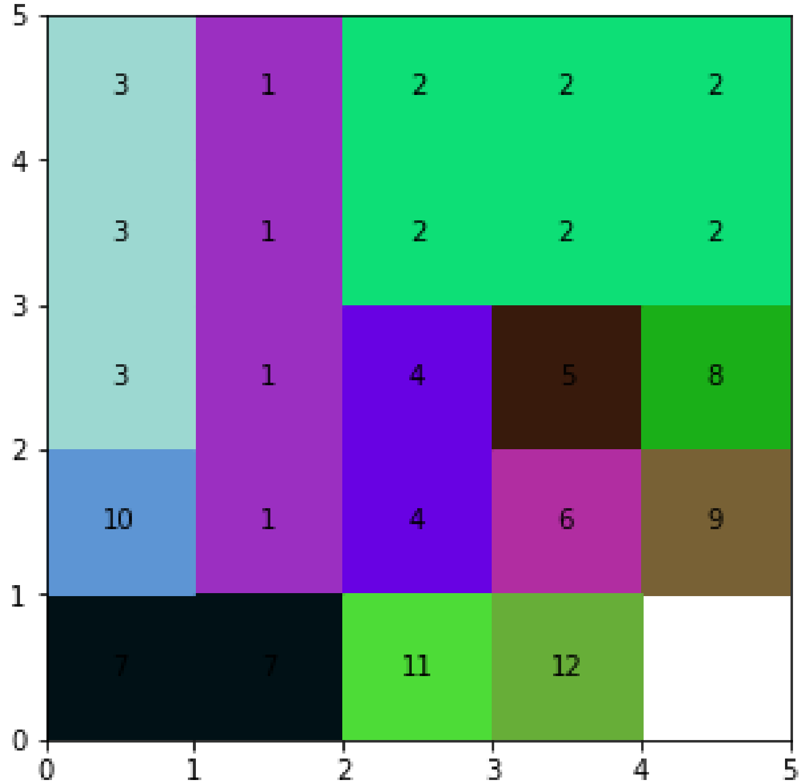


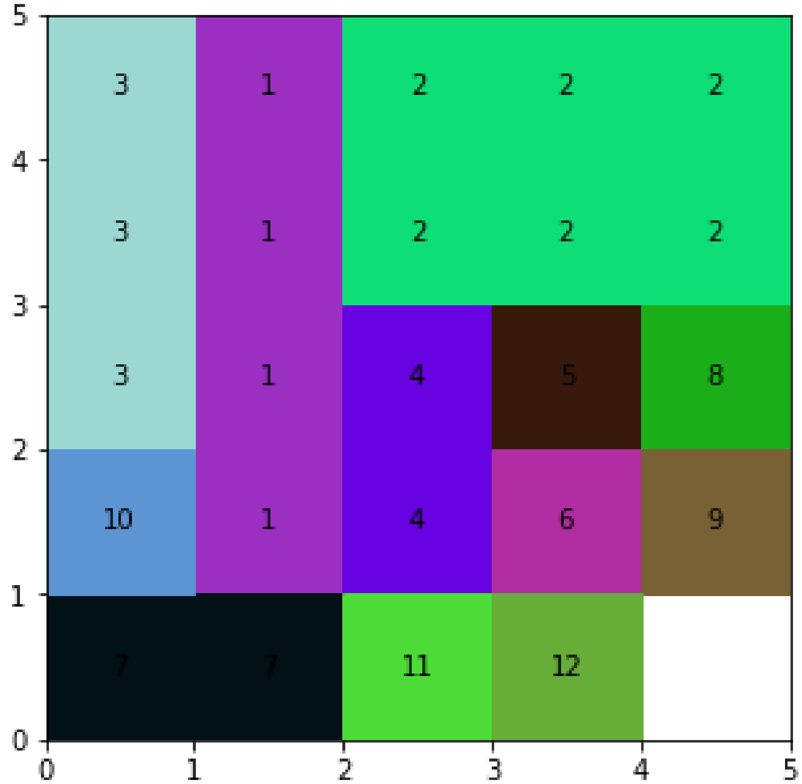
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
0
1
2
3
4
5



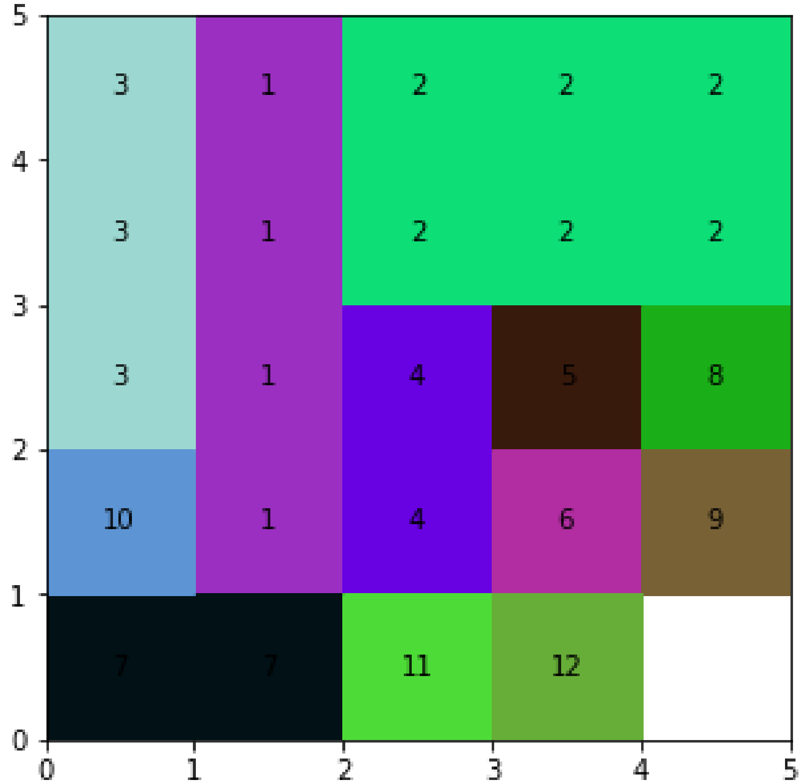
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
0
1
2
3
4
5
0
1
2
3
4
5


3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
0
1
2
3
4
5

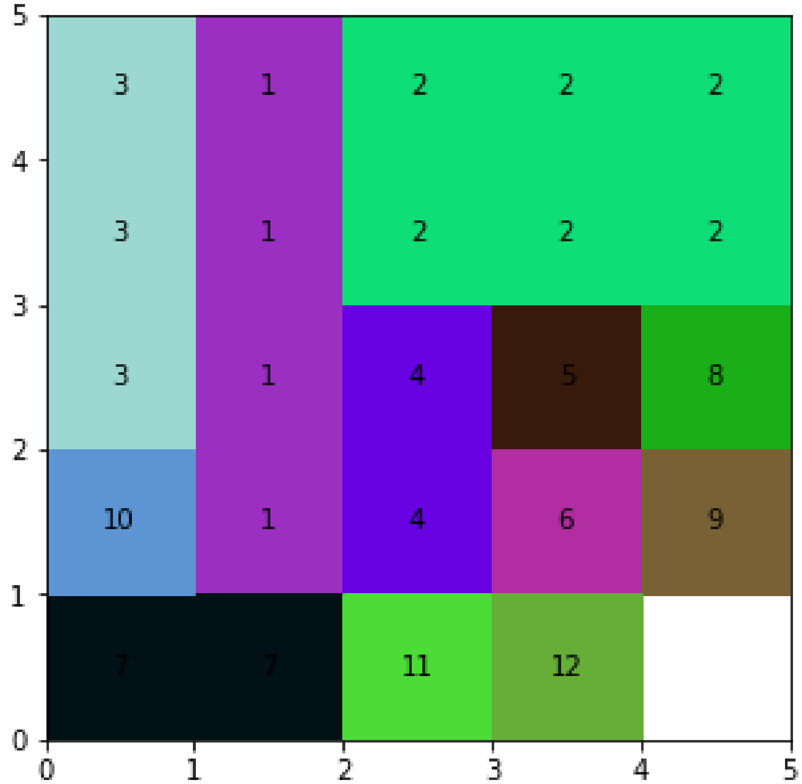

3 1 2 2 2
3 1 2 2 2
3 1 4 5 8
10 1 4 6 9
7 7 11 12
0 1 2 3 4 5



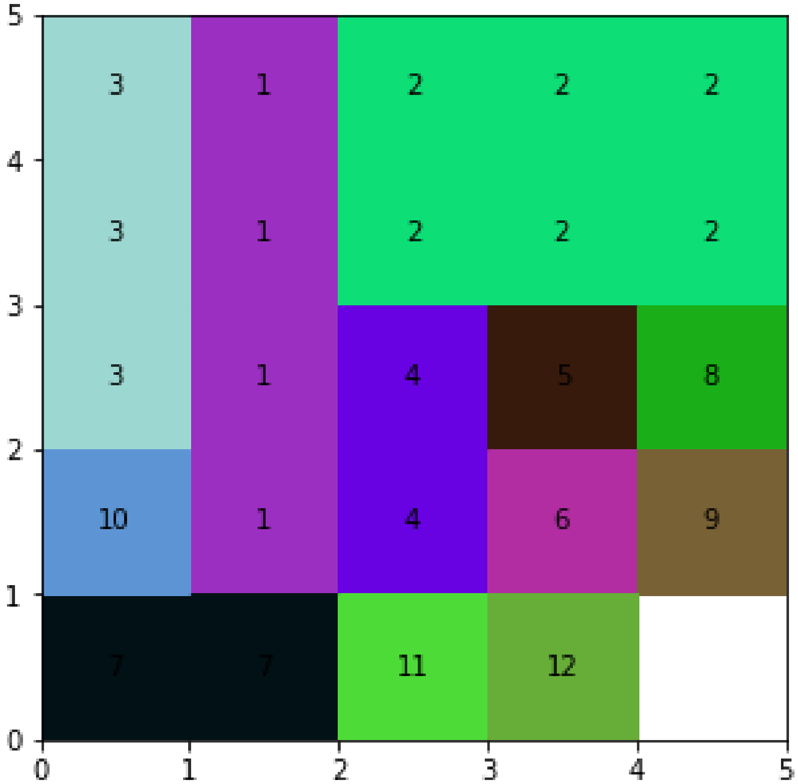
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
0
1
2
3
4
5



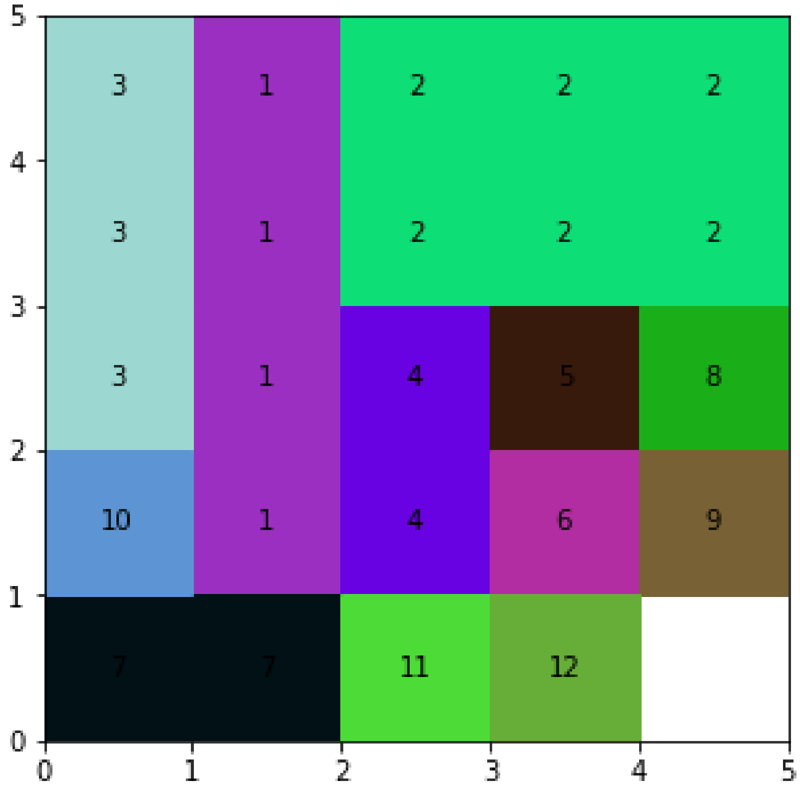
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
0
1
2
3
4
5

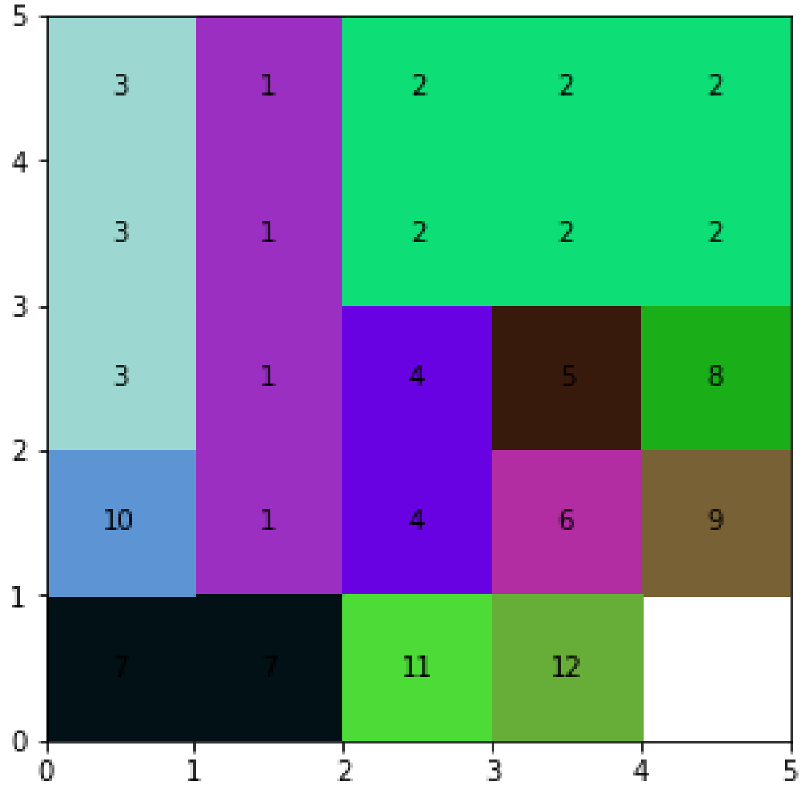


5
4
3
2
1
0
0
1
2
3
4
5
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12

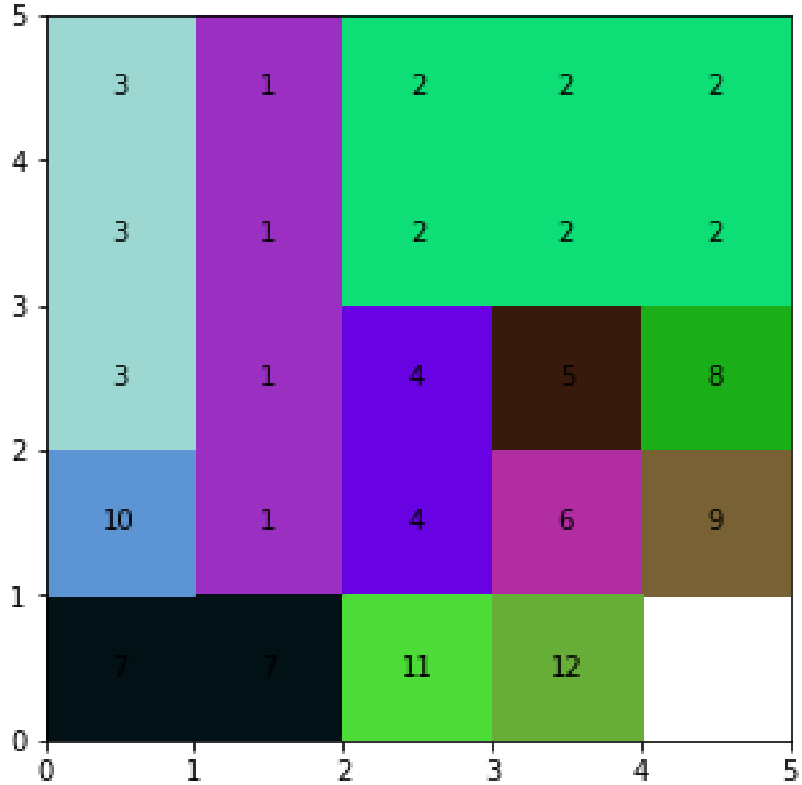


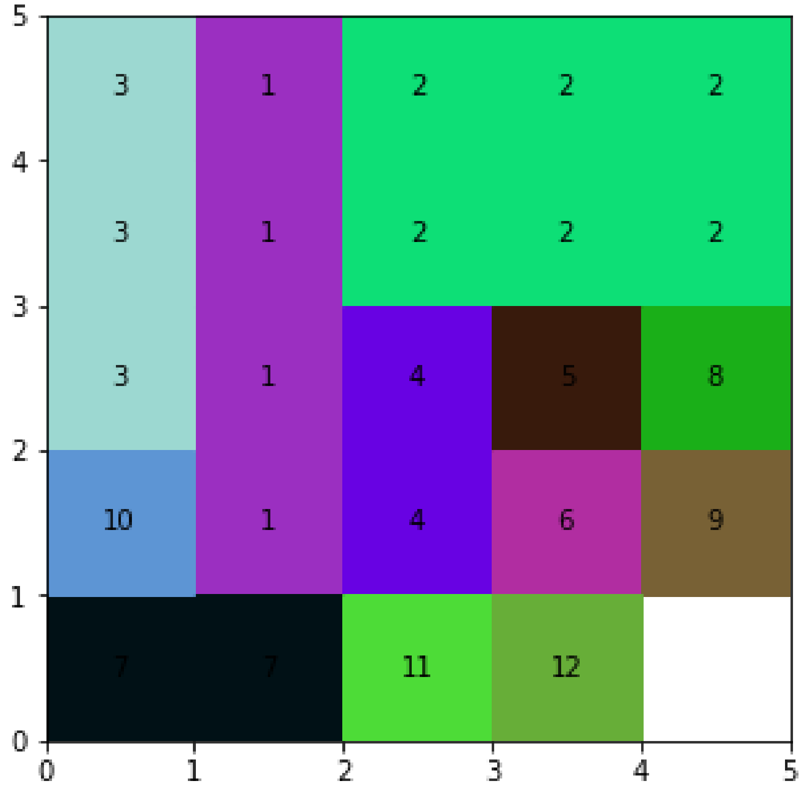
5
4
3
2
1
0
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
0
1
2
3
4
5

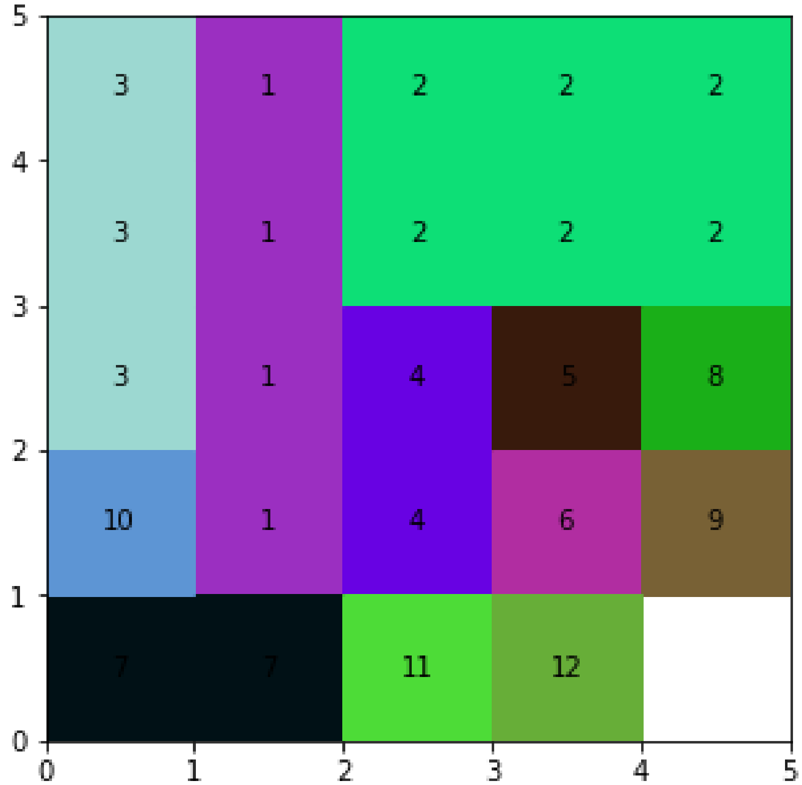

5
4
3
2
1
0
0
1
2
3
4
5
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12

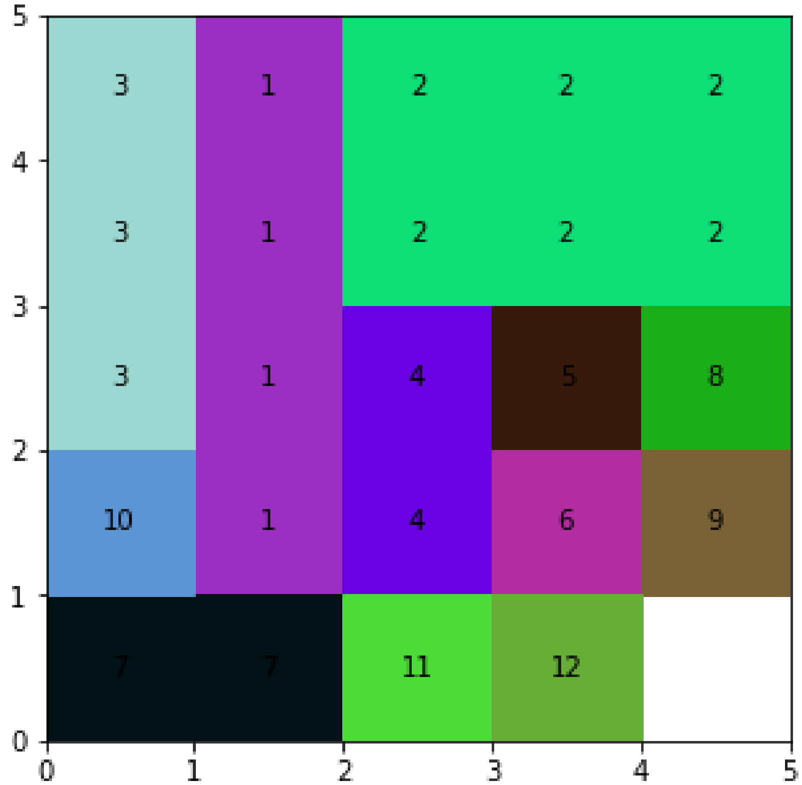


3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
0
1
2
3
4
5

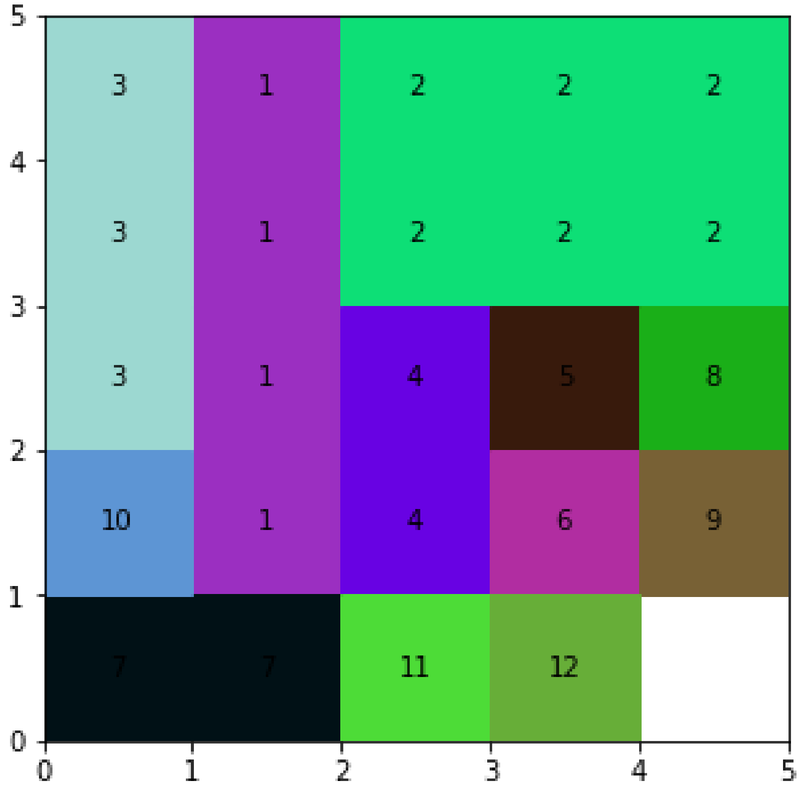

3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
0
1
2
3
4
5


3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
0
1
2
3
4
5

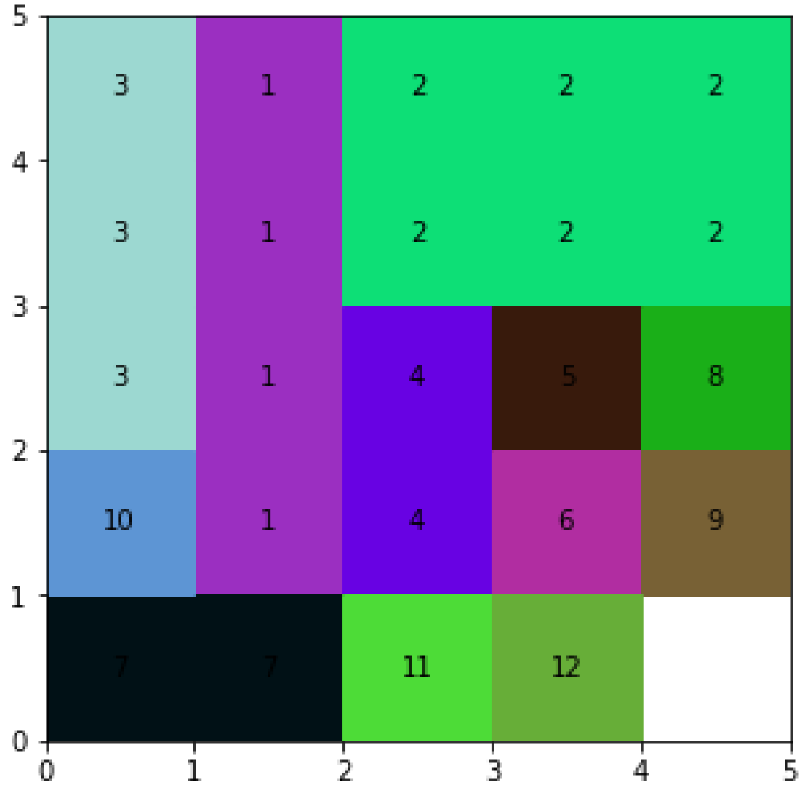

3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
0
1
2
3
4
5



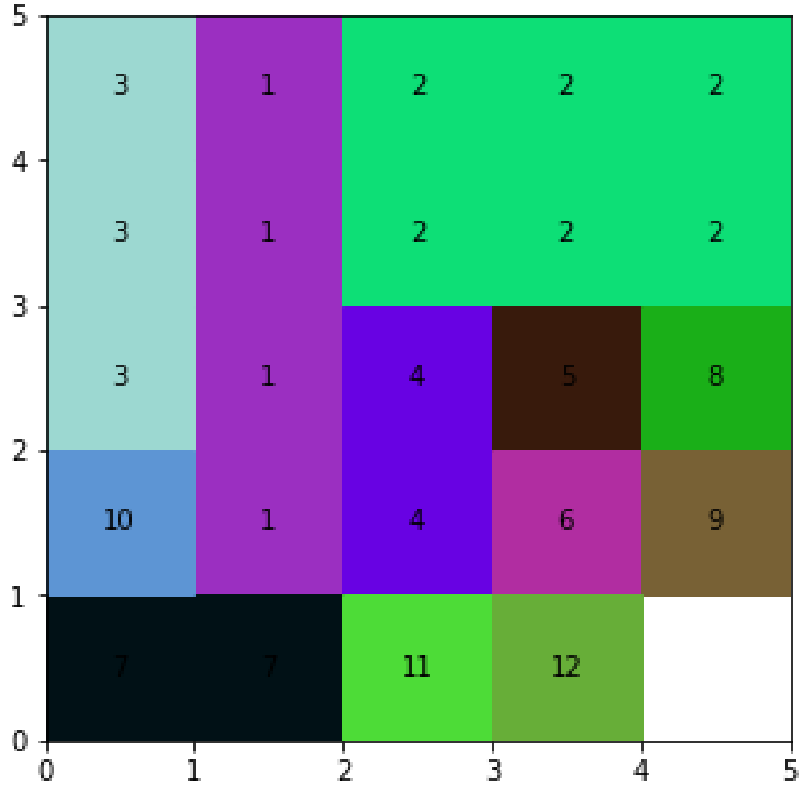
3
1
2
2
2
4
3
1
2
2
2
3
3
1
4
5
8
2
10
1
4
6
9
1
7
7
11
12
0
0
1
2
3
4
5

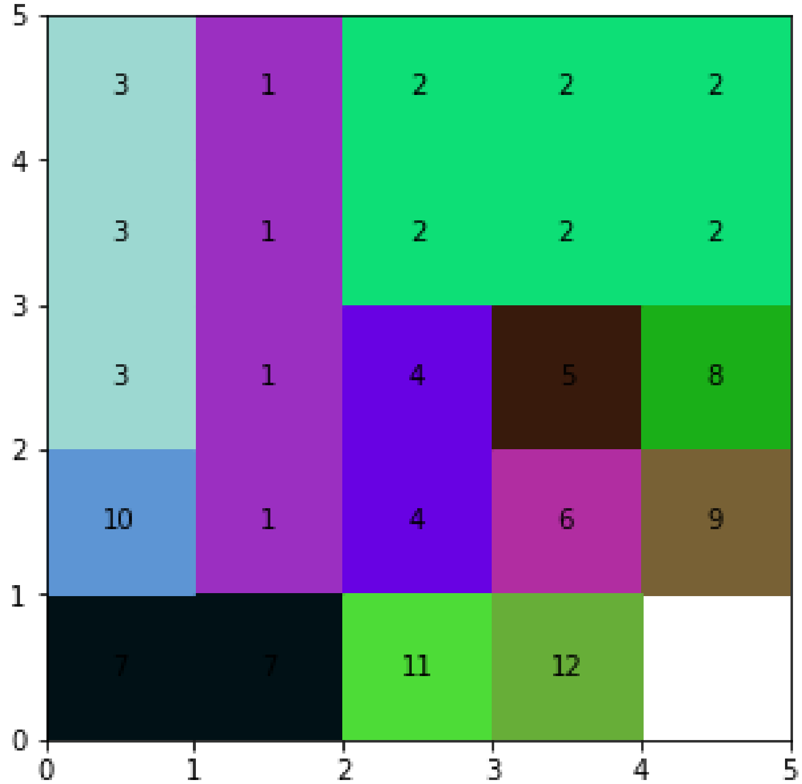


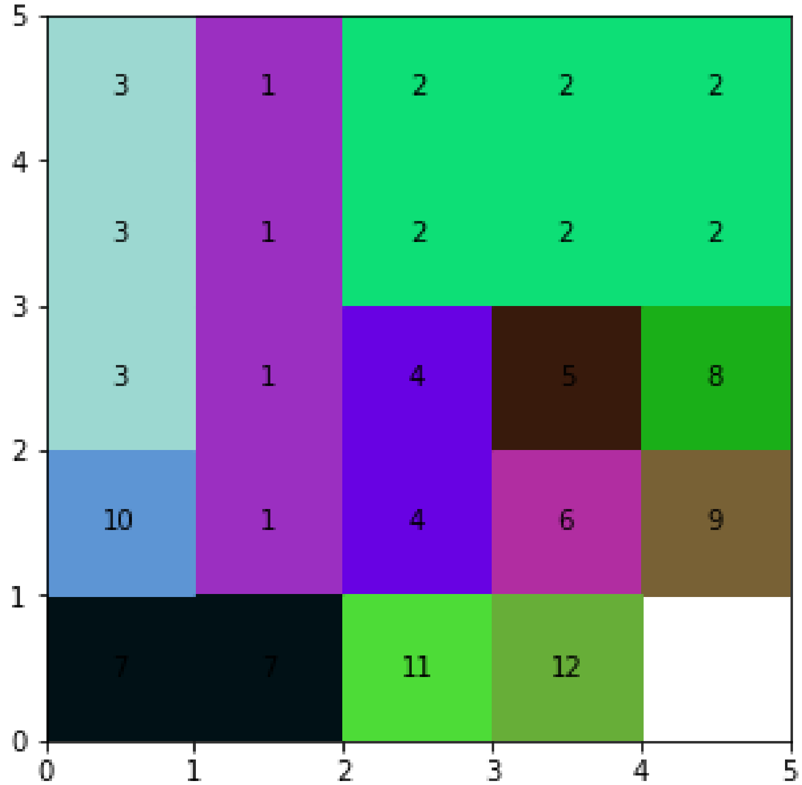
5
4
3
2
1
0
0
1
2
3
4
5
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12

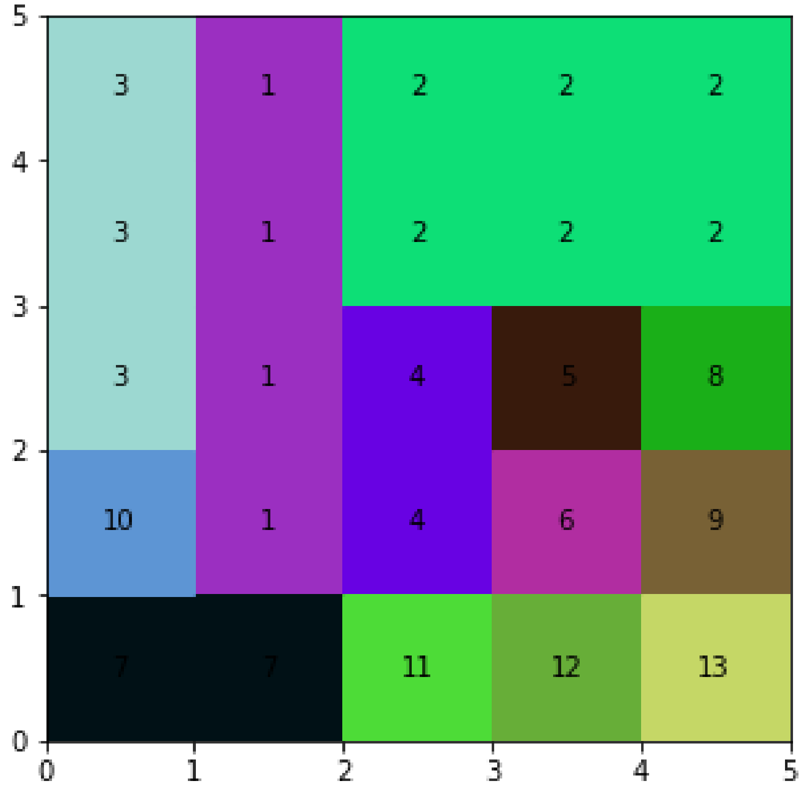


5
4
3
2
1
0
0
1
2
3
4
5
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12

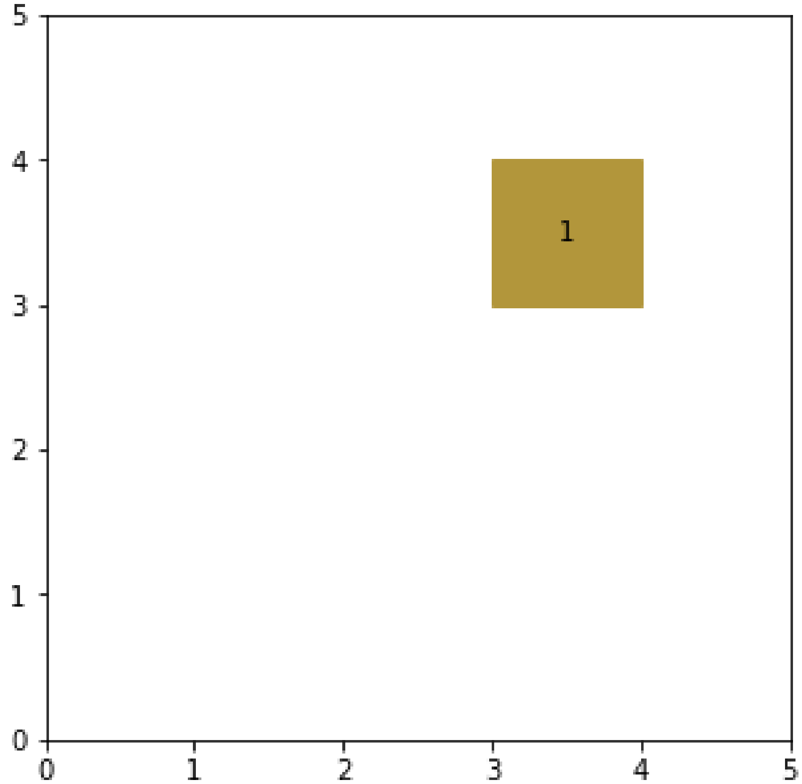

3 1 2 2 2
3 1 2 2 2
3 1 4 5 8
10 1 4 6 9
7 7 11 12
0 1 2 3 4 5
0 1 2 3 4 5


5
4
3
2
1
0
0
1
2
3
4
5
3 1 2 2 2
3 1 2 2 2
3 1 4 5 8
10 1 4 6 9
7 7 11 12

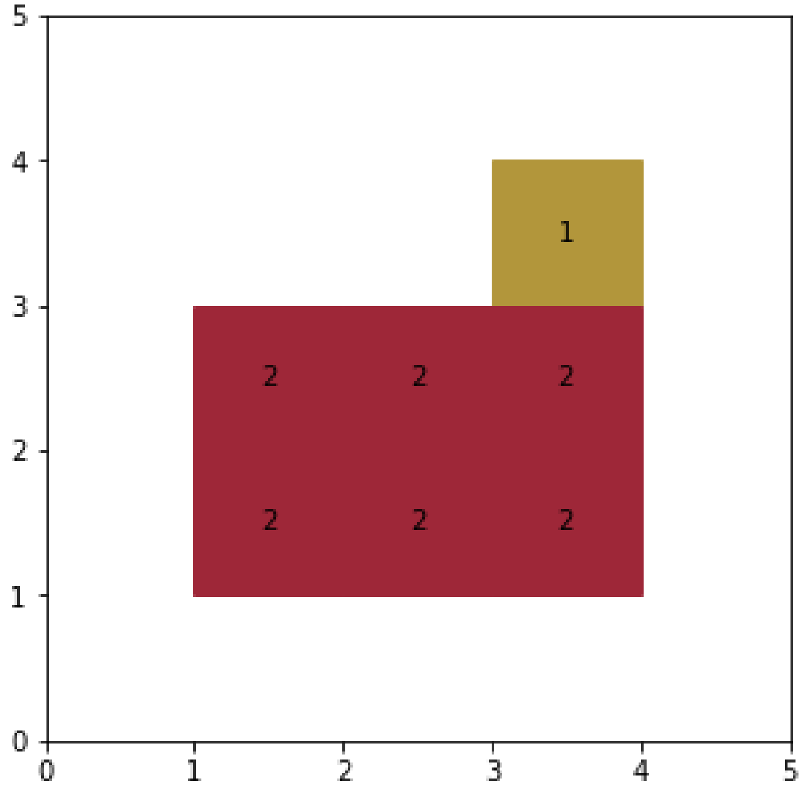

3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
0
1
2
3
4
5



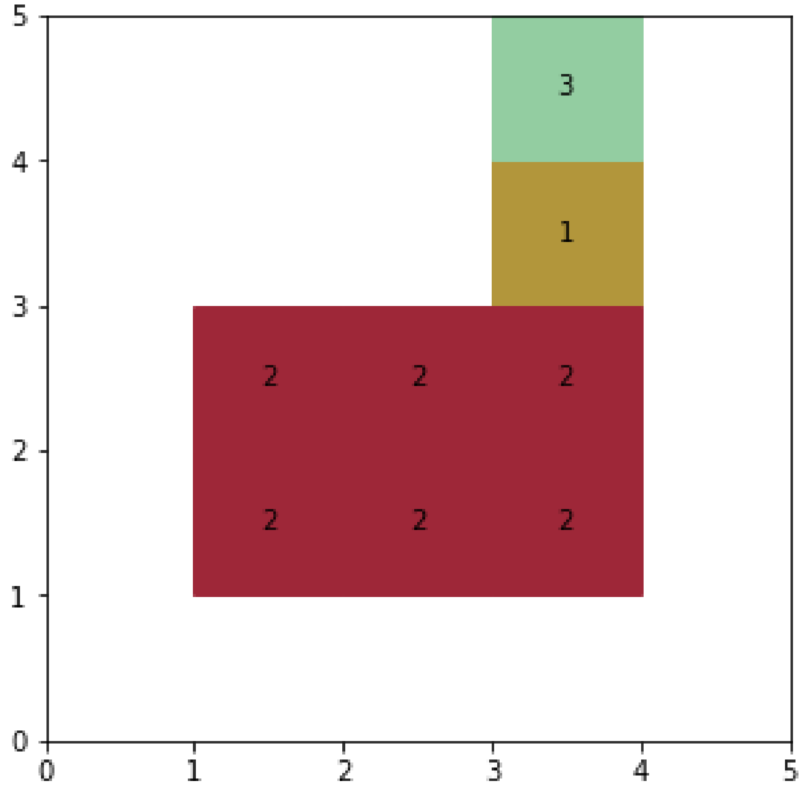
3
1
2
2
2
3
1
2
2
2
3
1
4
5
8
10
1
4
6
9
7
7
11
12
13
0
1
2
3
4
5

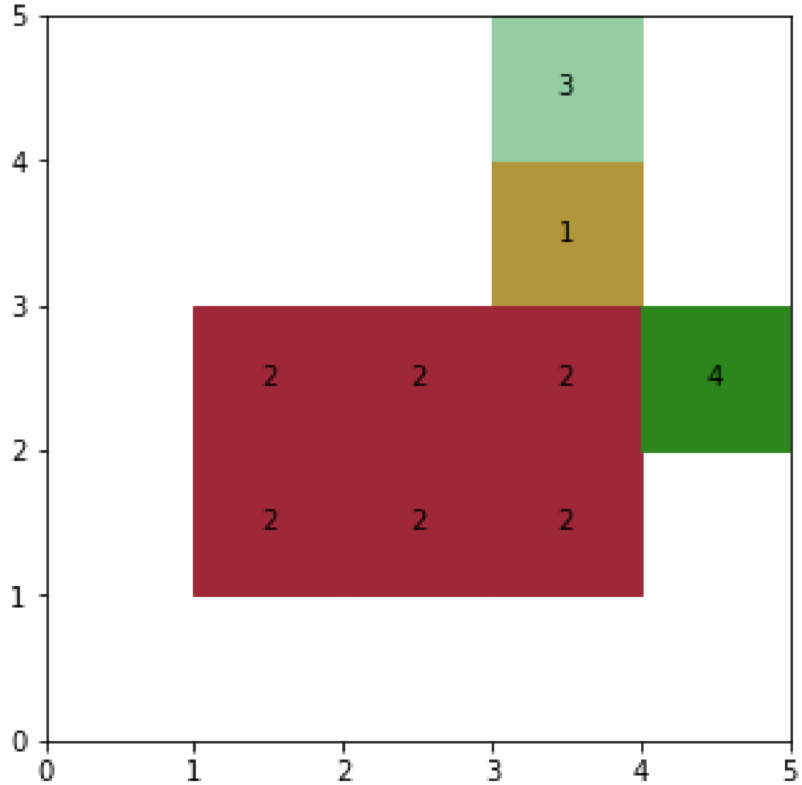


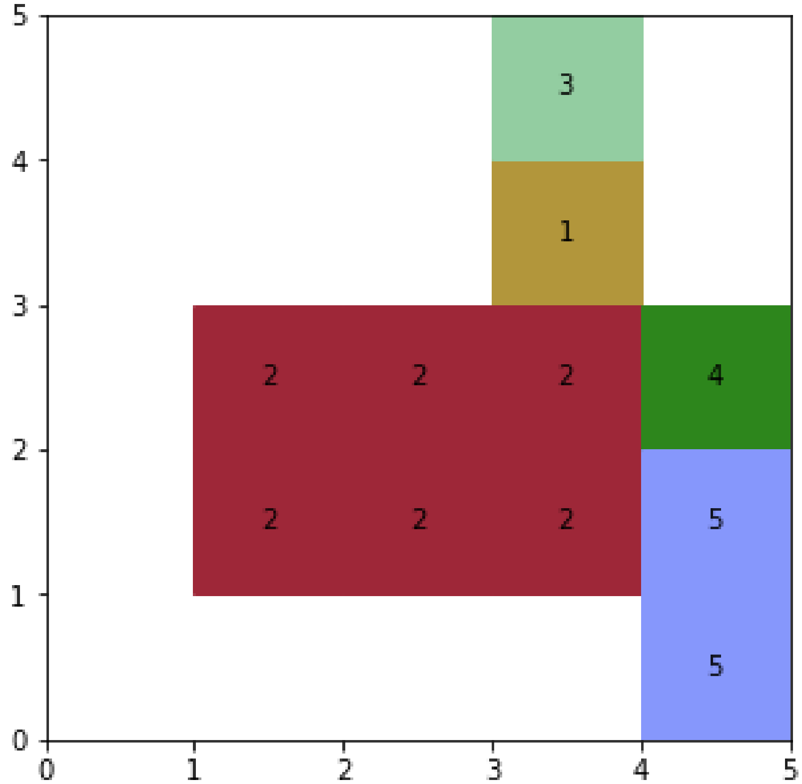
0
1
2
3
4
5
0
1
2
3
4
5
1



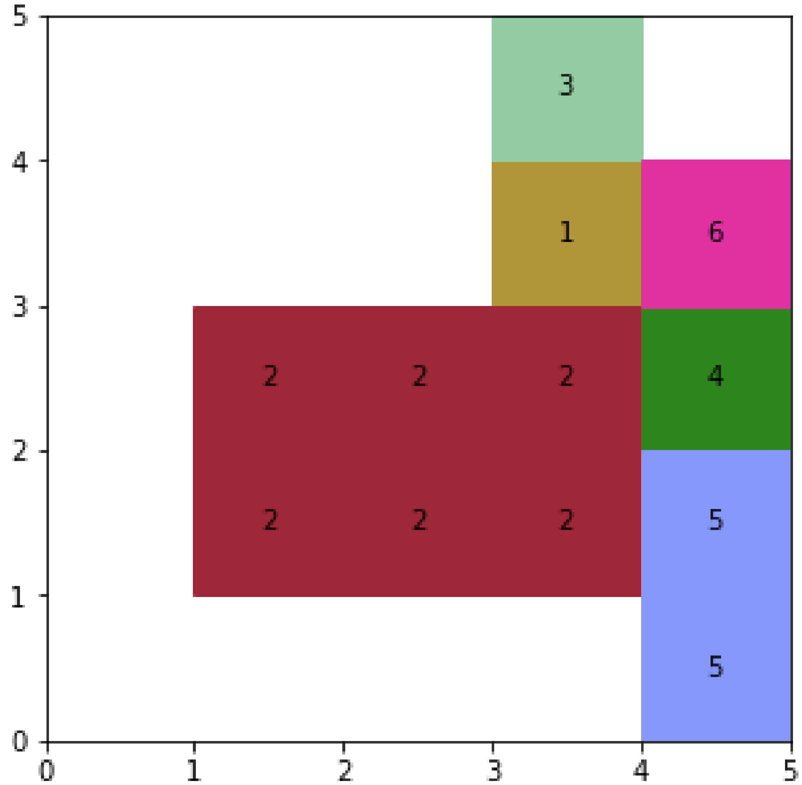
5
4
3
2
1
0
0
1
2
3
4
5
1
2
2
2
2
2
2


3
1
2
2
2
2
2
2
0
1
2
3
4
5

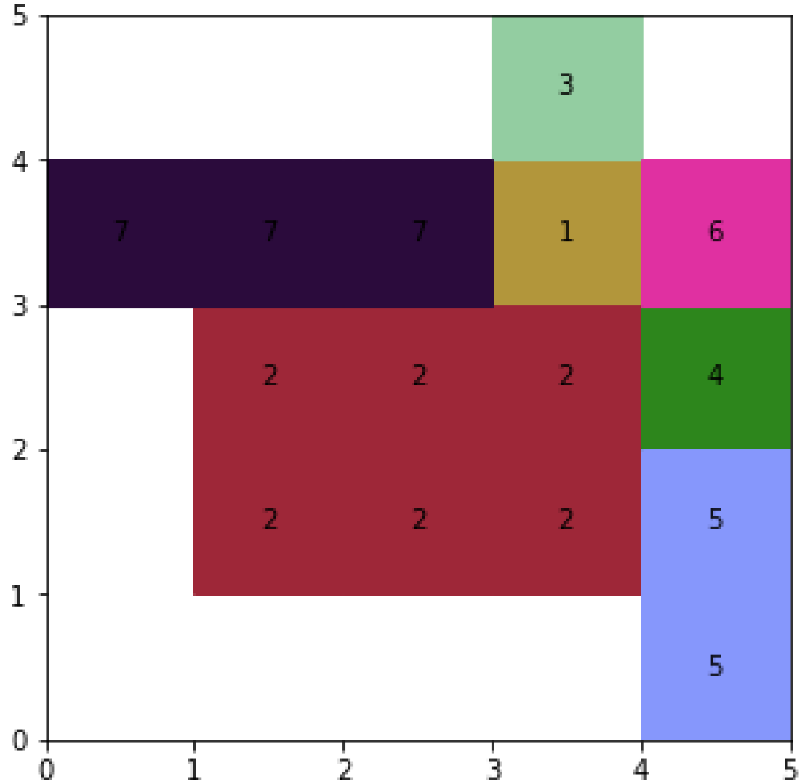

3
1
2
2
2
4
2
2
2
0
1
2
3
4
5
0
1
2
3
4
5



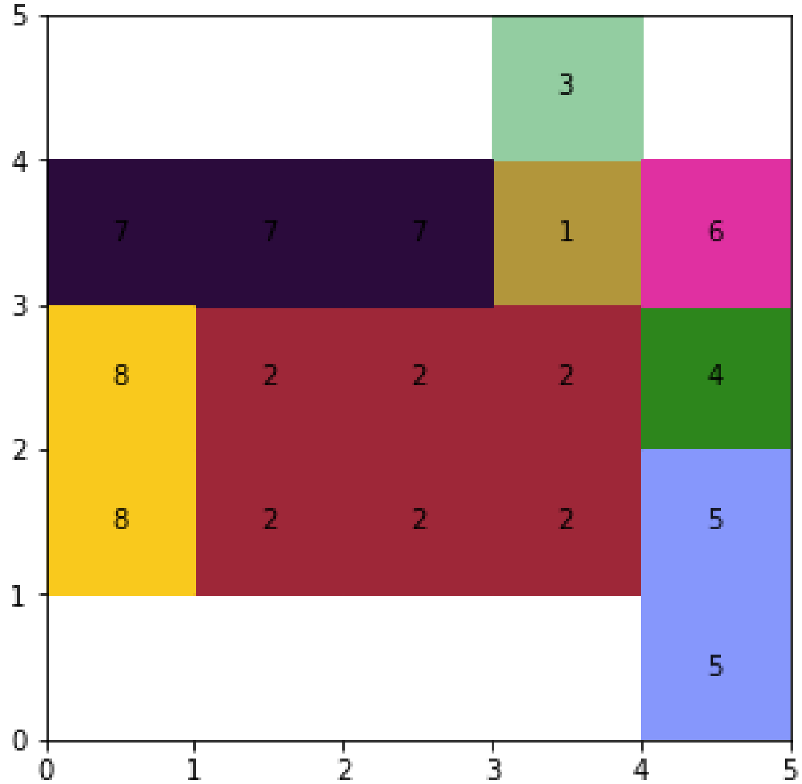
3
1
2 2 2 4
2 2 2 5
5
0 1 2 3 4 5
0 1 2 3 4 5



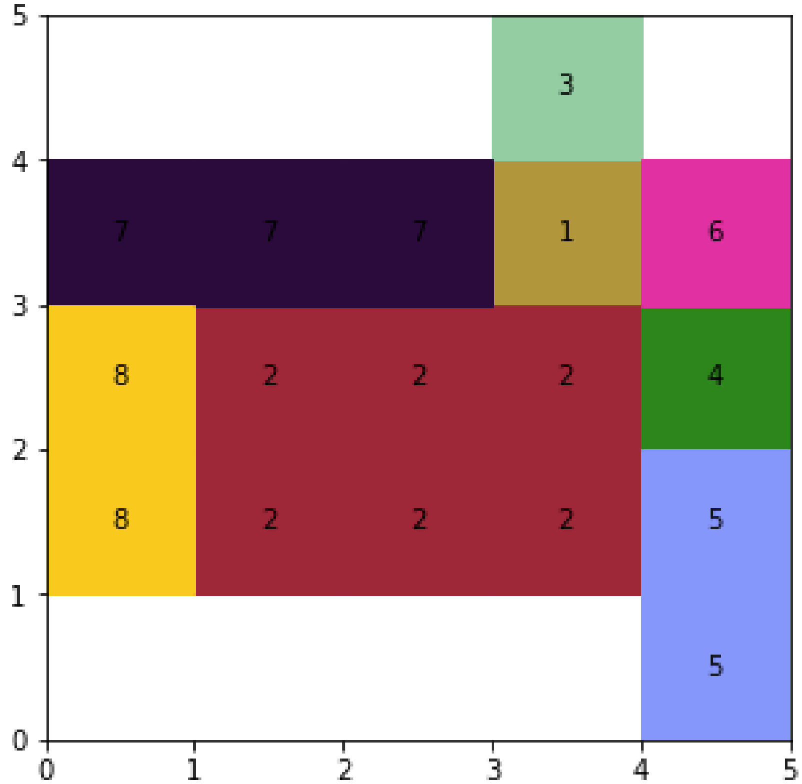
3
1
6
2
2
2
4
2
2
2
5
5
0
1
2
3
4
5

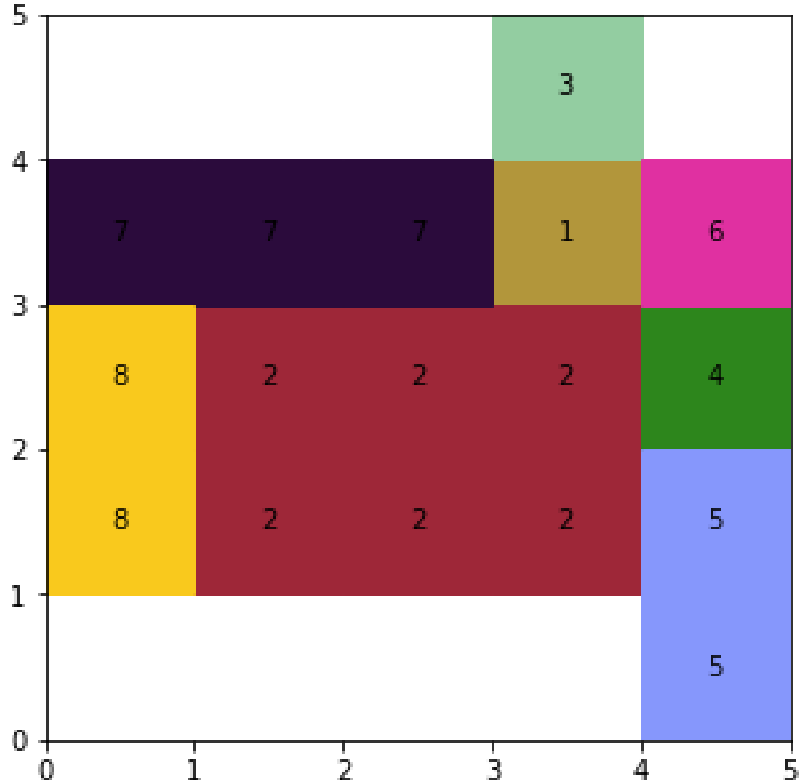


5
4
3
2
1
0
0
1
2
3
4
5
3
7
7
7
1
6
2
2
2
4
2
2
2
5
5

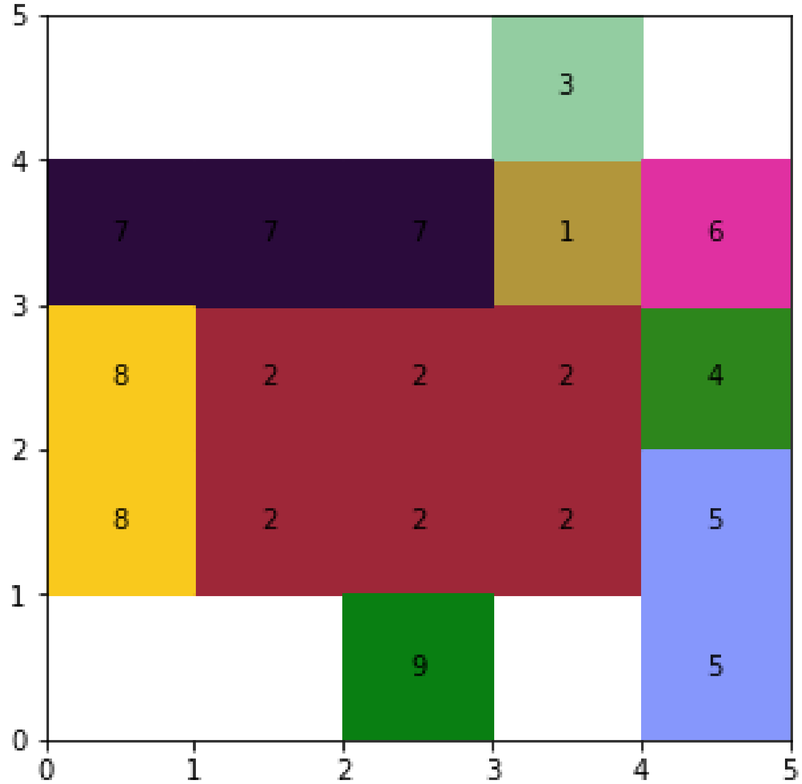


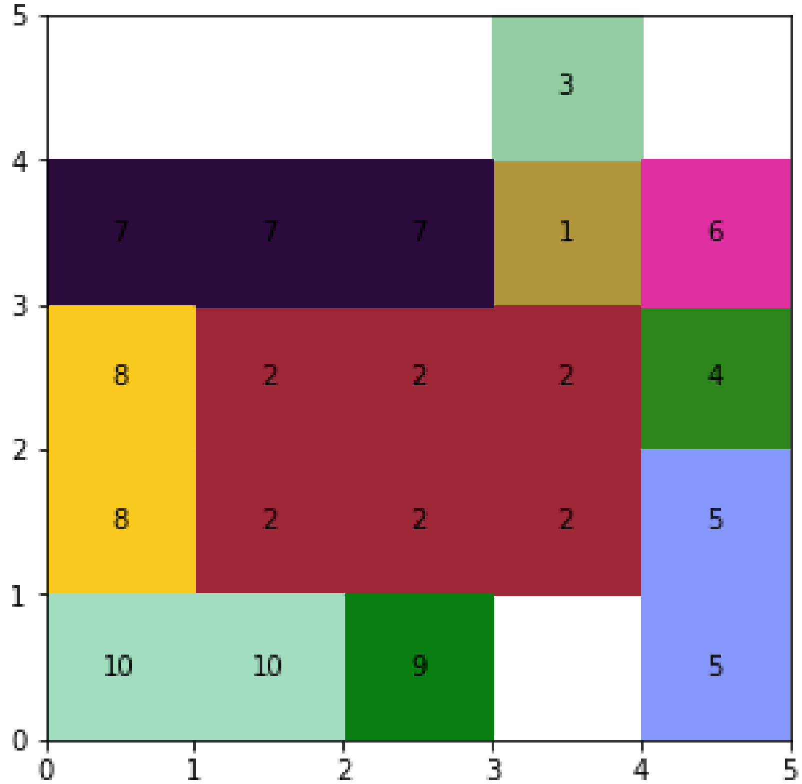
5
4
3
2
1
0
0
1
2
3
4
5
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
5


3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
5
0
1
2
3
4
5

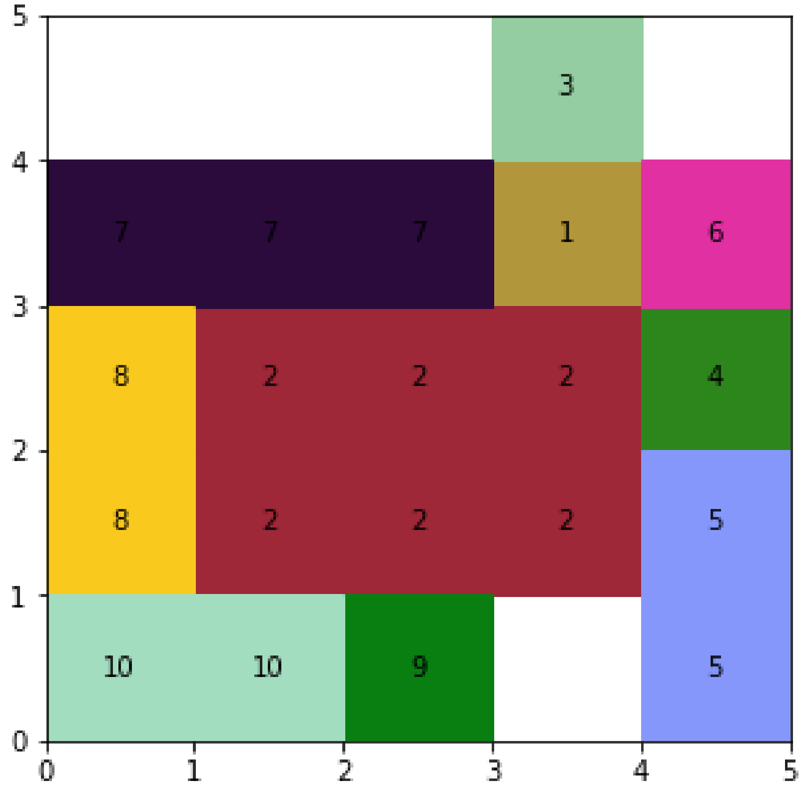


3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
5
0
1
2
3
4
5
0
1
2
3
4
5

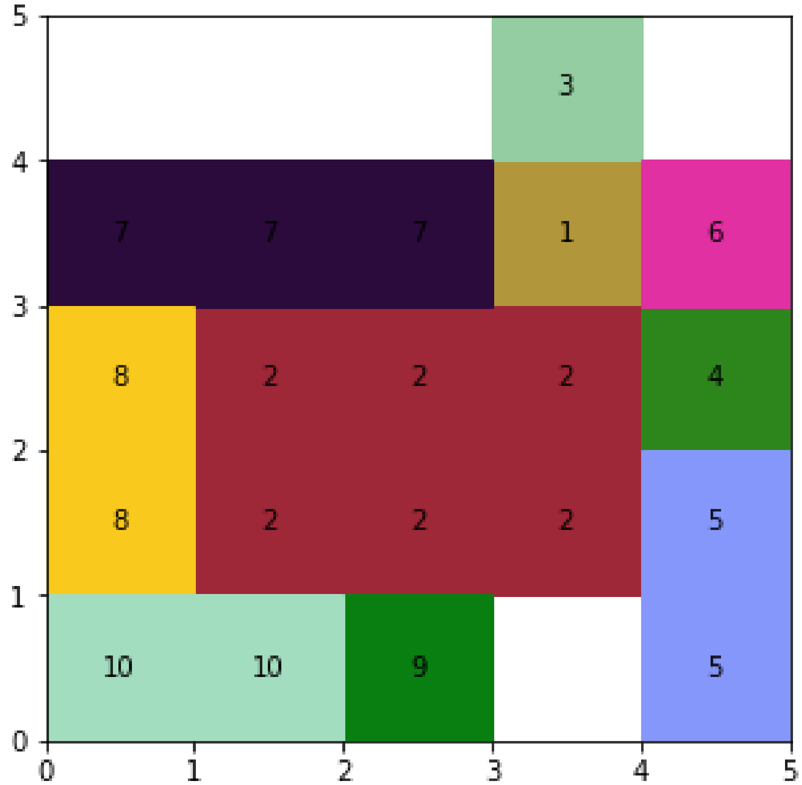

5
4
3
2
1
0
0
1
2
3
4
5
3
7 7 7 1 6
8 2 2 2 4
8 2 2 2 5
9
5



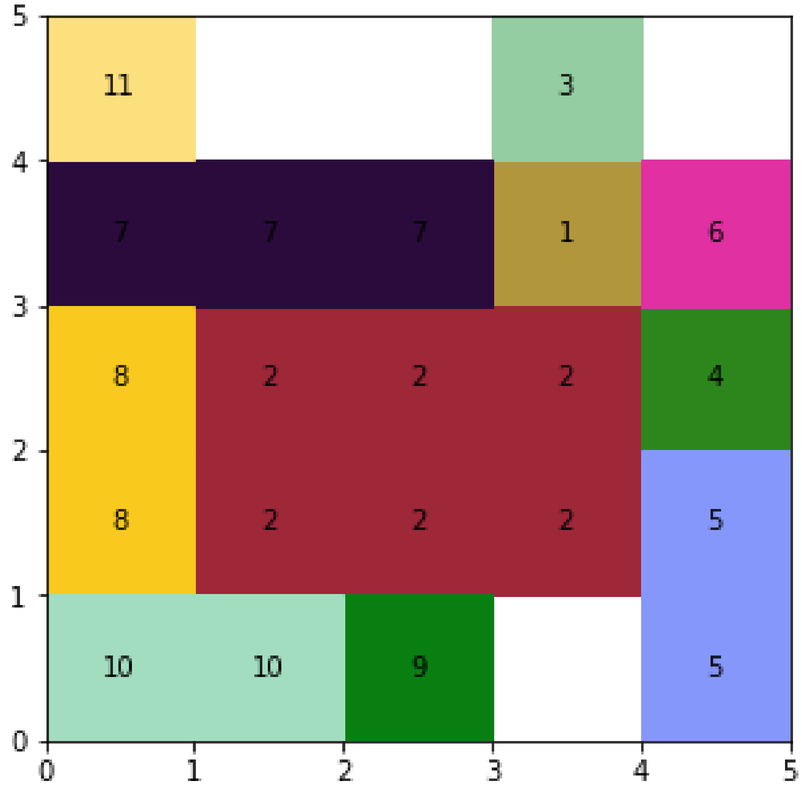
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
5
0
1
2
3
4
5

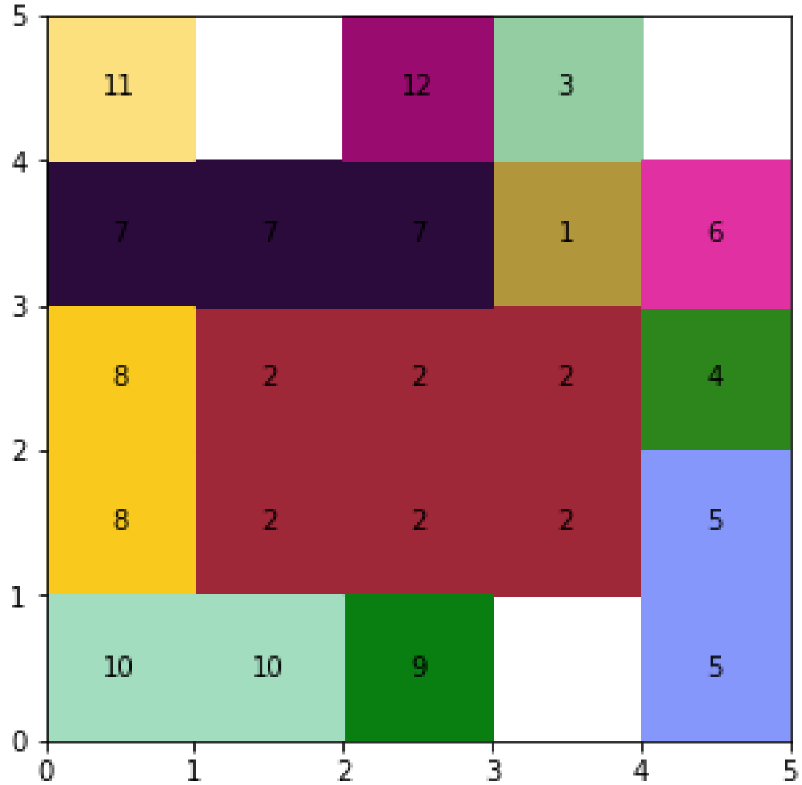


3
7 7 7 1 6
8 2 2 2 4
8 2 2 2 5
10 10 9 5
0 1 2 3 4 5

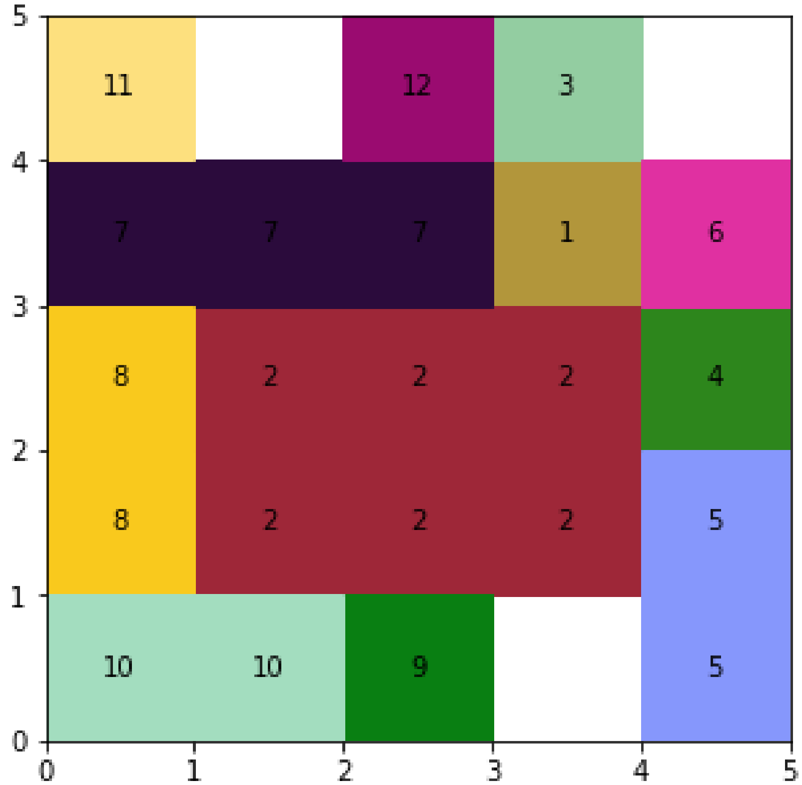


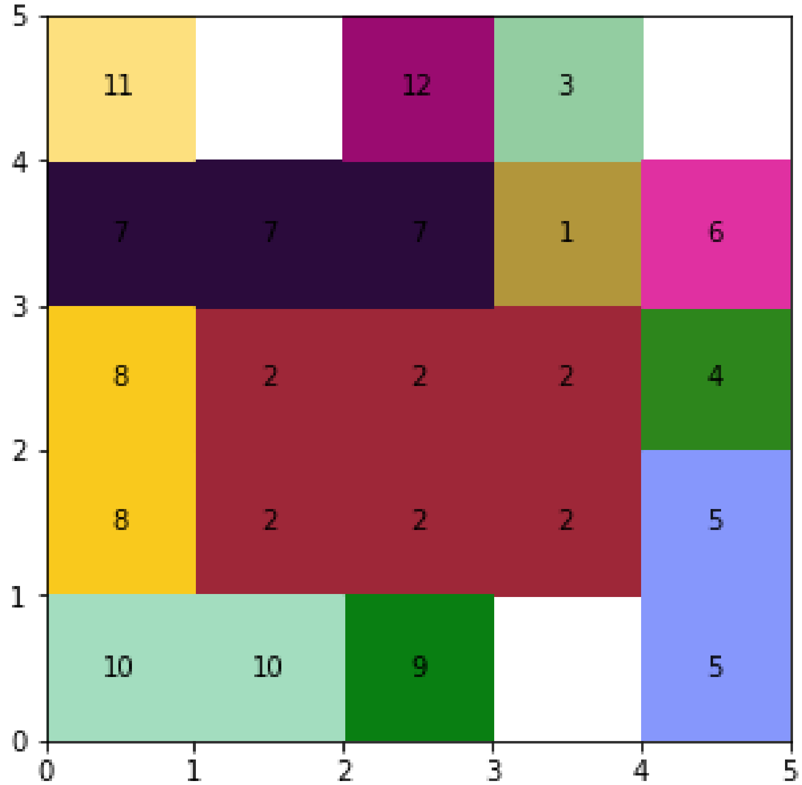
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
5
0
1
2
3
4
5


11
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
5
0
1
2
3
4
5

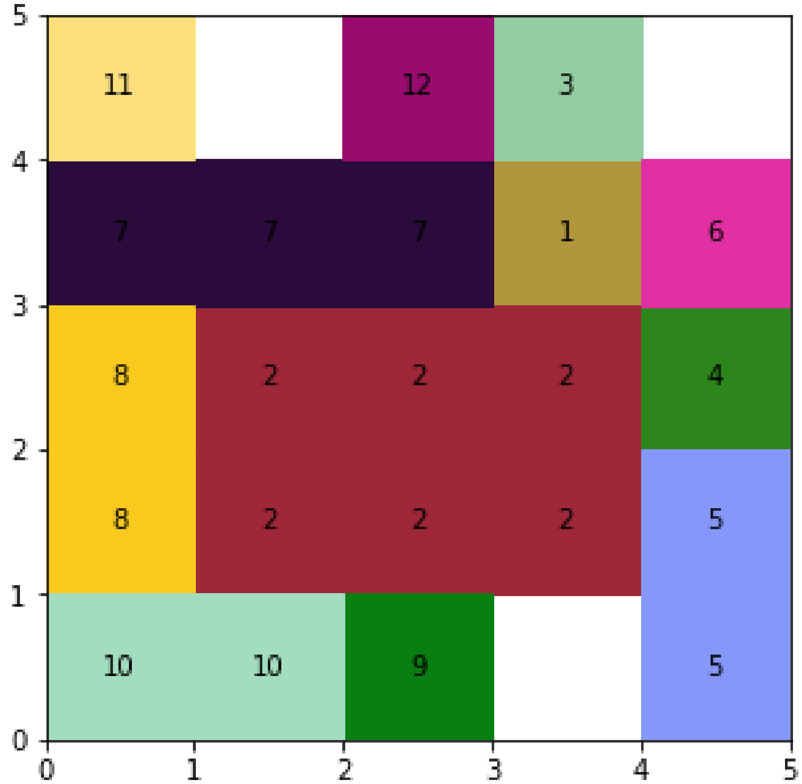


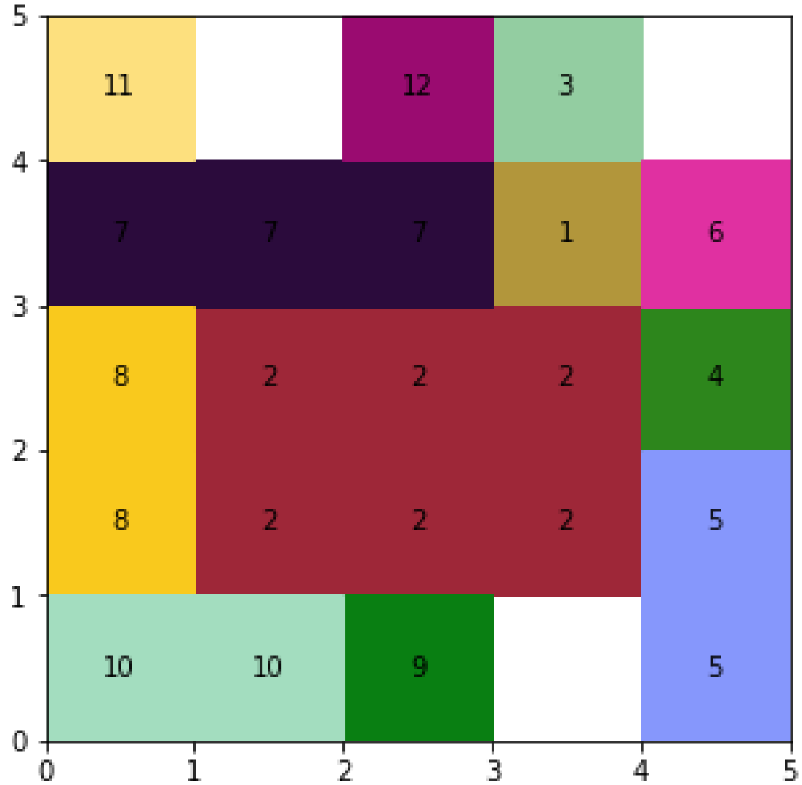
11
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
5
0
1
2
3
4
5


11
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
5
0
1
2
3
4
5

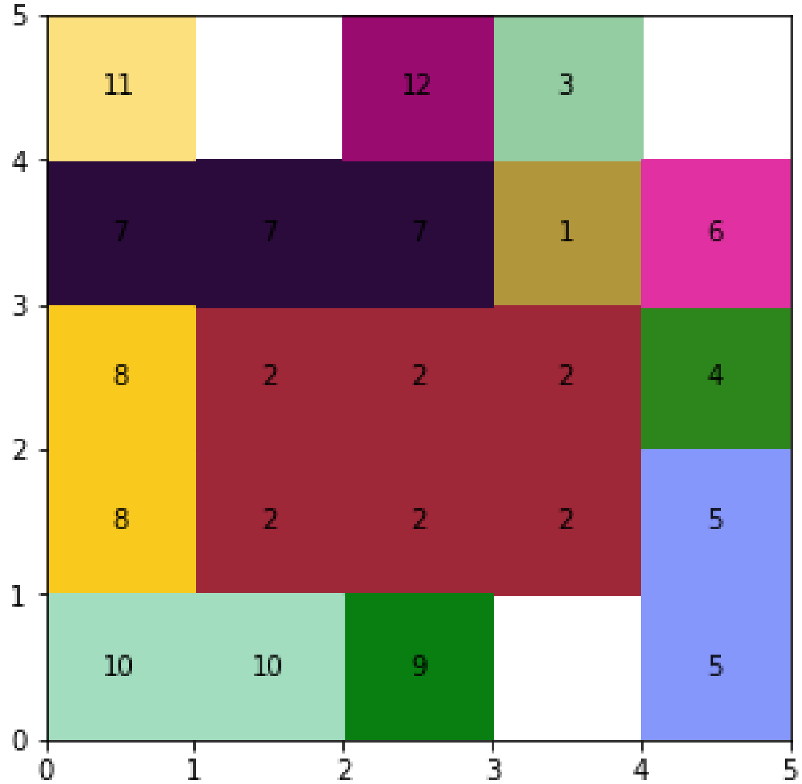


11
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
5
0
1
2
3
4
5
0
1
2
3
4
5

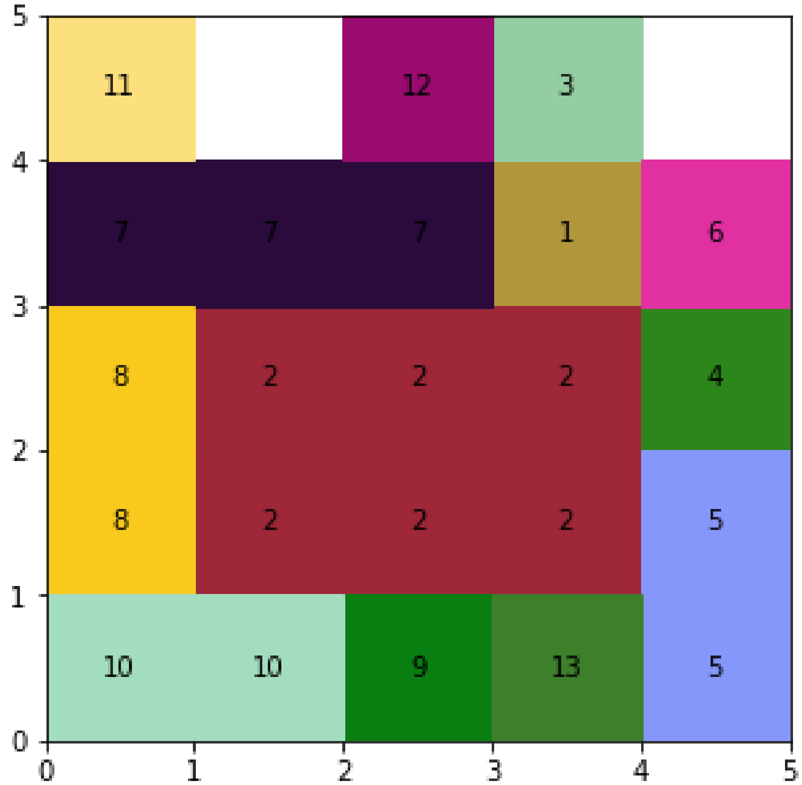

11
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
5
0
1
2
3
4
5



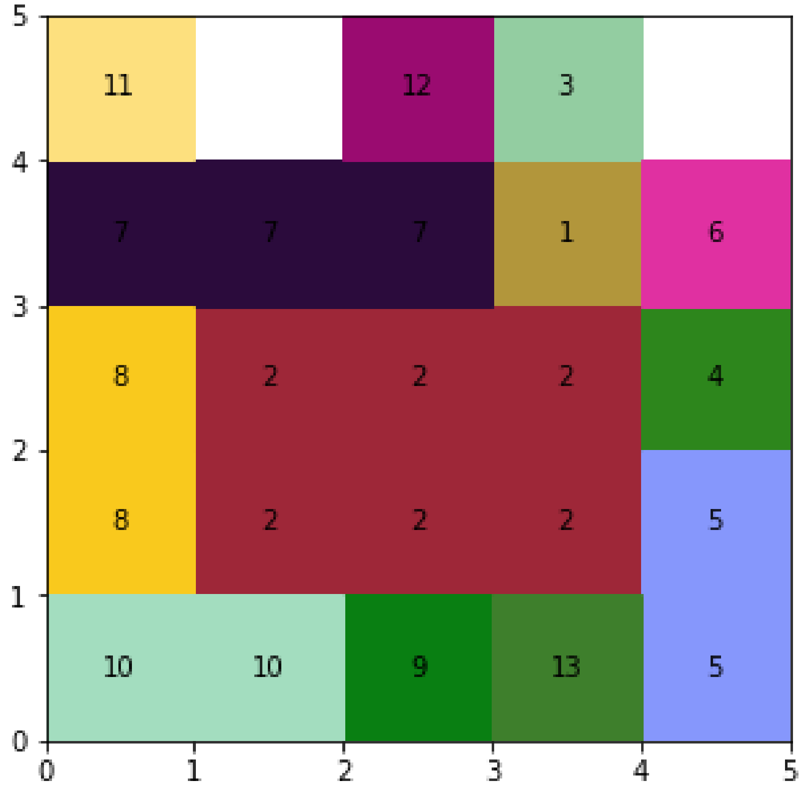
11
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
5
0
1
2
3
4
5
0
1
2
3
4
5



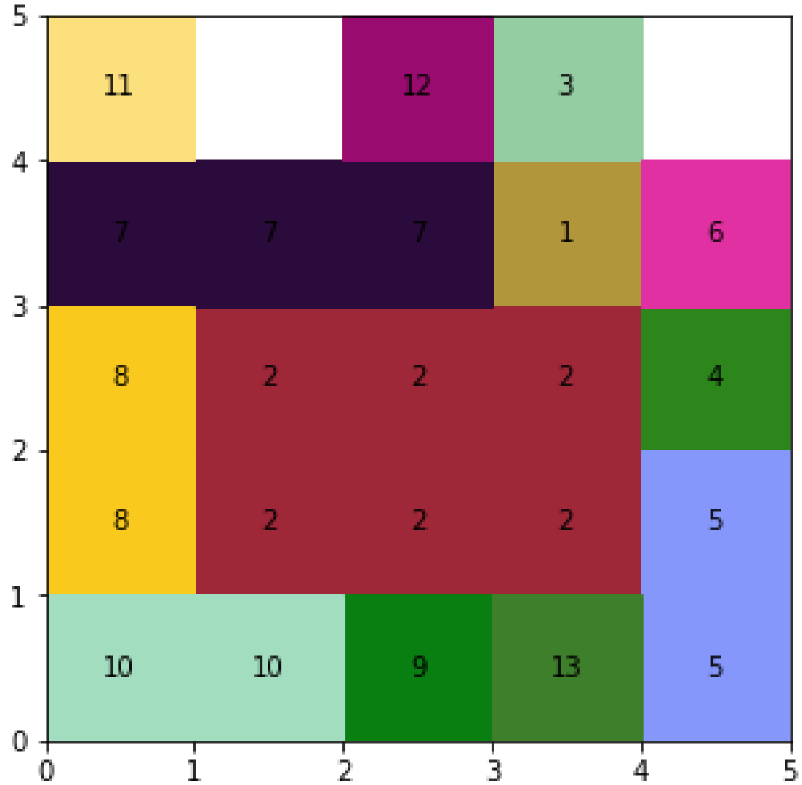
11
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
5
0
1
2
3
4
5



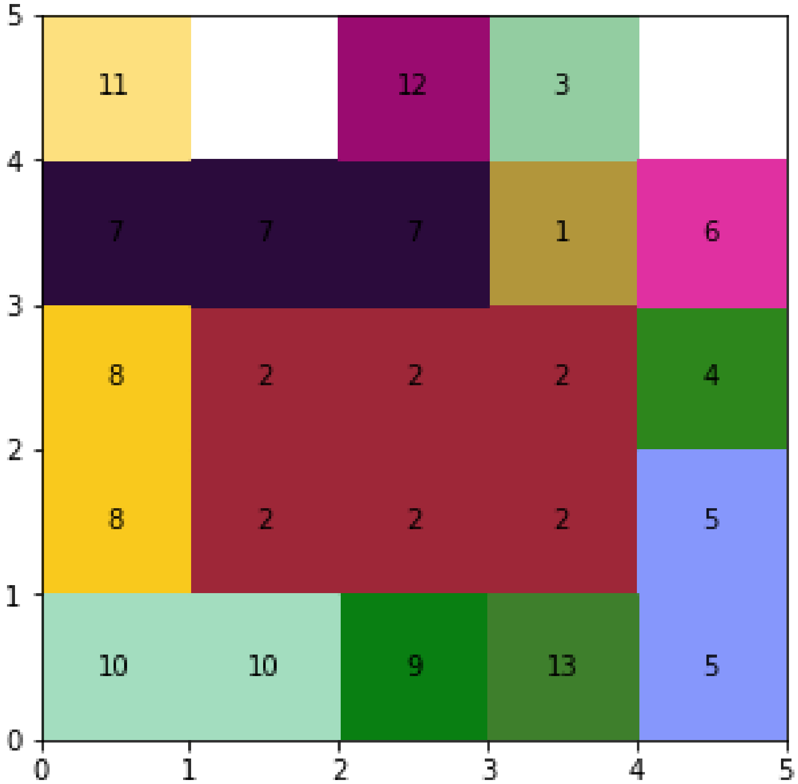
11
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
13
5
0
1
2
3
4
5

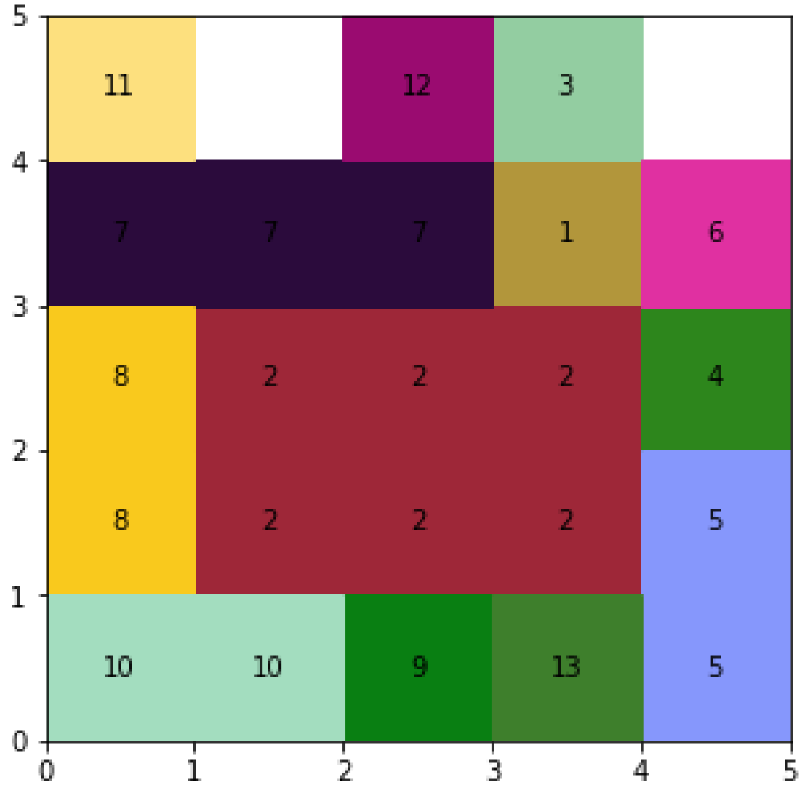


11
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
13
5
0
1
2
3
4
5
0
1
2
3
4
5

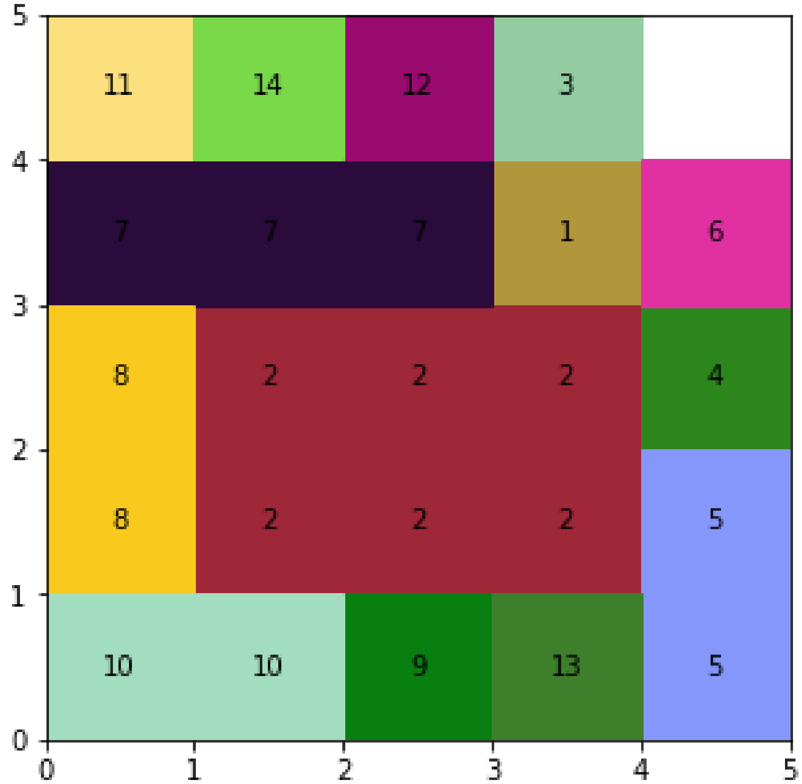


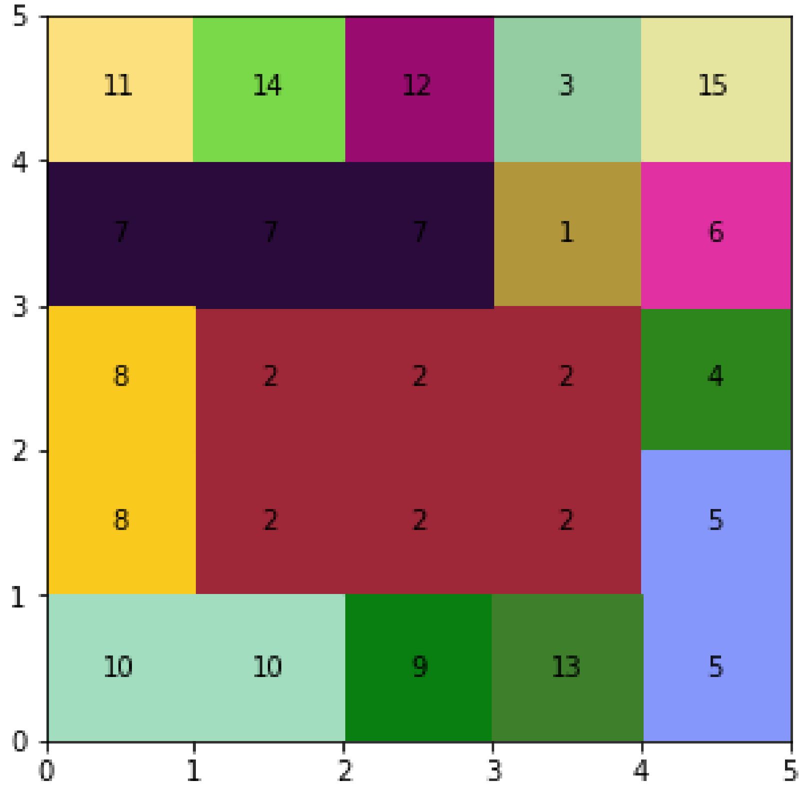
11
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
13
5
0
1
2
3
4
5


11
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
13
5
0
1
2
3
4
5

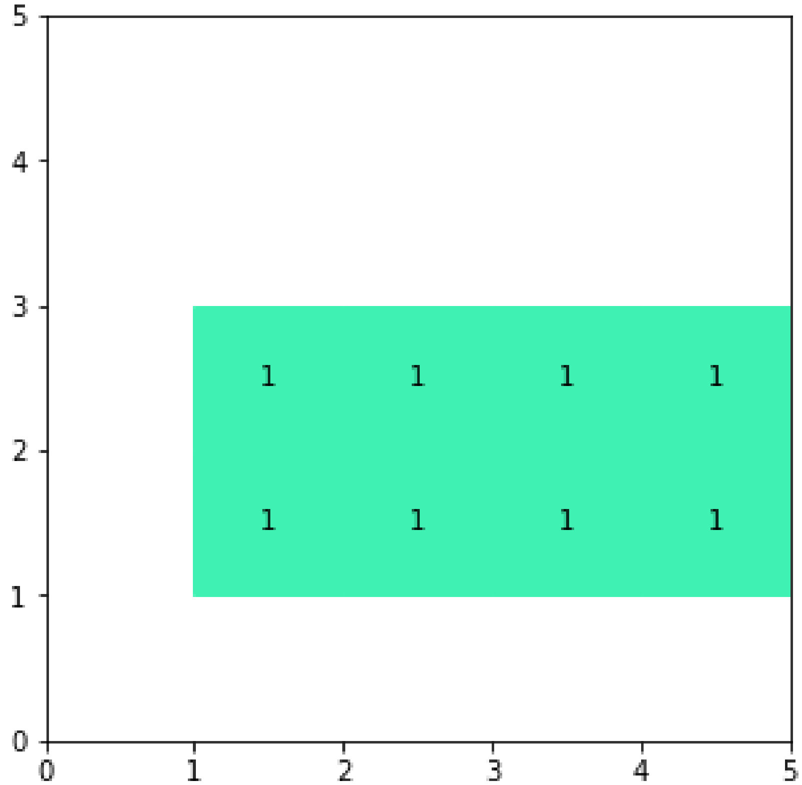


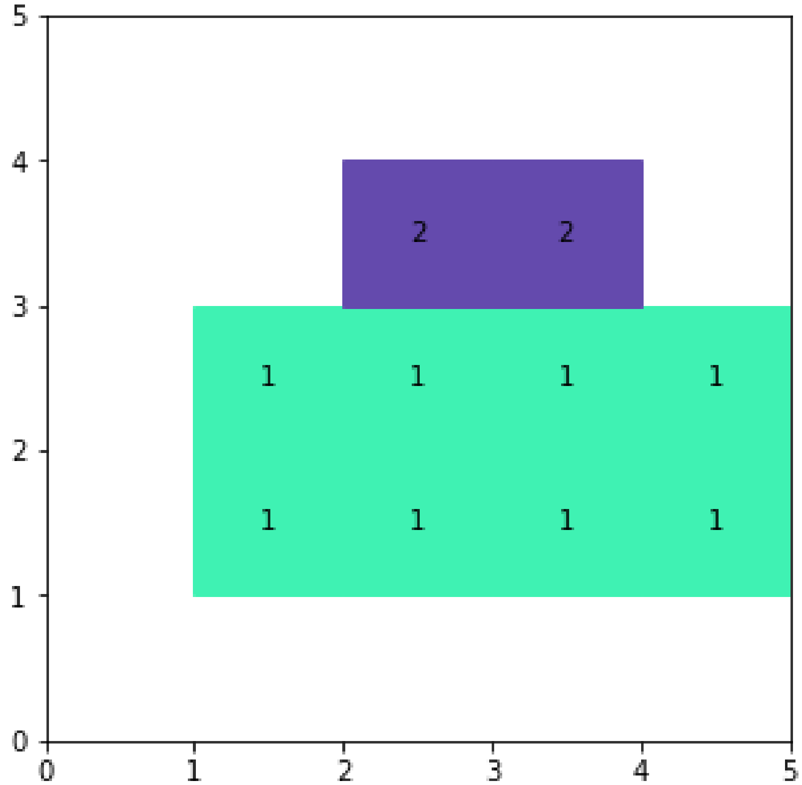
11
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
13
5
0
1
2
3
4
5

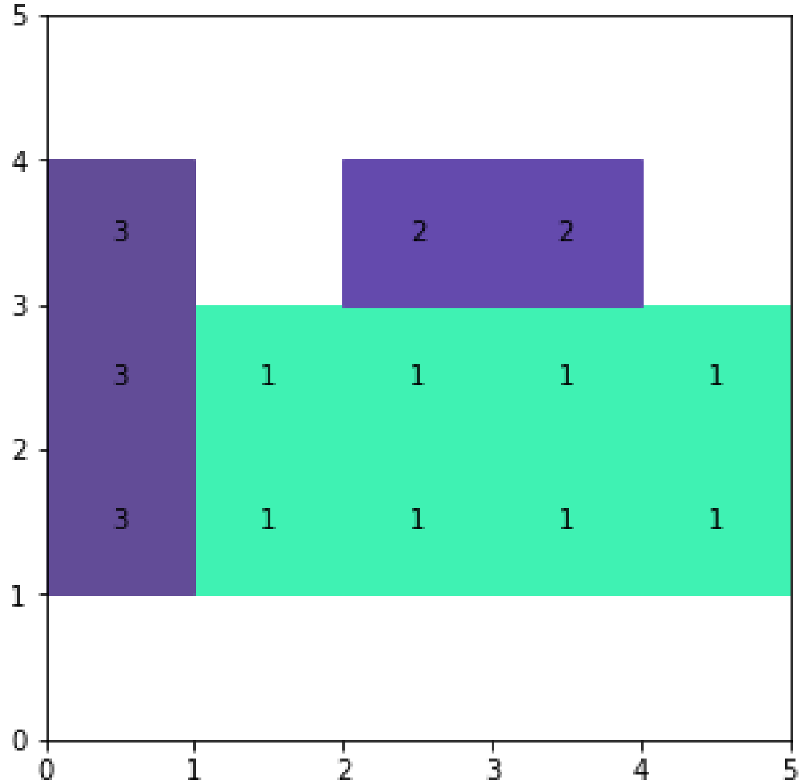

11
14
12
3
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
13
5
0
1
2
3
4
5
0
1
2
3
4
5



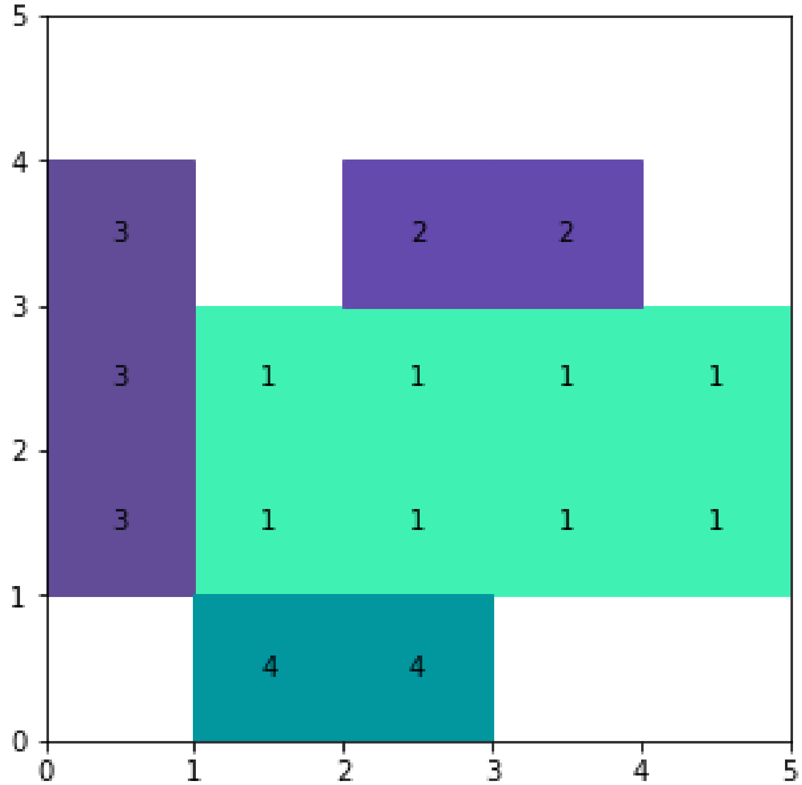
11
14
12
3
15
7
7
7
1
6
8
2
2
2
4
8
2
2
2
5
10
10
9
13
5
0
1
2
3
4
5


5
4
3
2
1
0
1 1 1 1
1 1 1 1
0 1 2 3 4 5

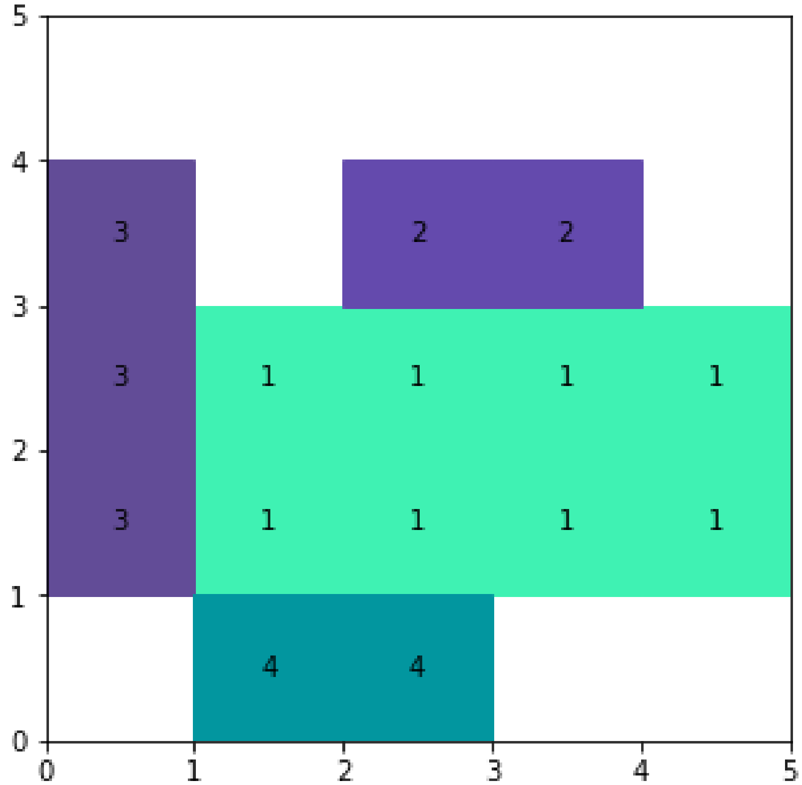

5
4
3
2
1
0
0
1
2
3
4
5
2
2
1
1
1
1
1
1
1
1



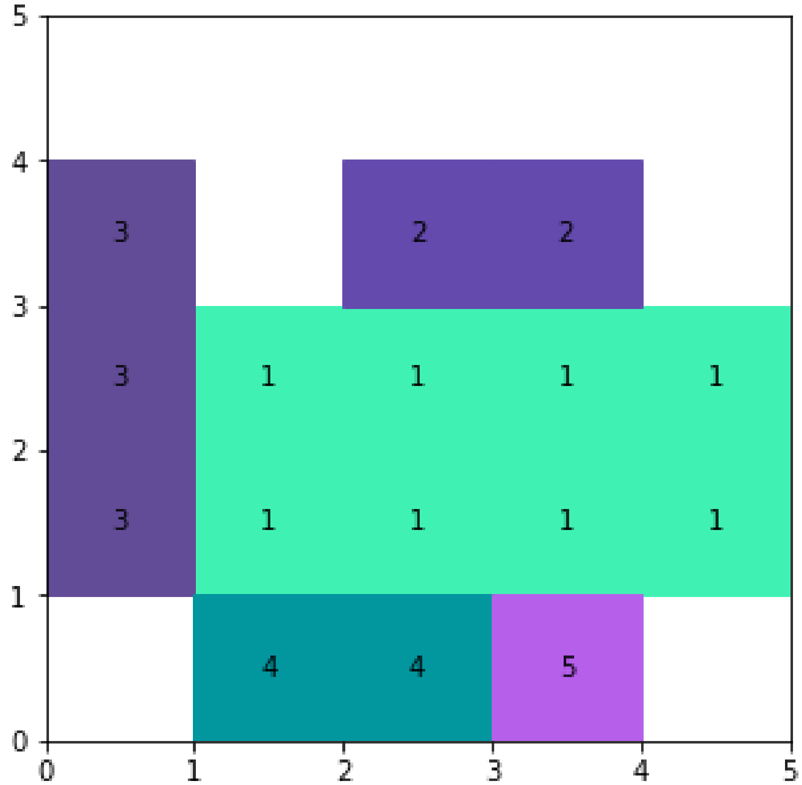
5
4
3
2
1
0
3
2
2
3
1
1
1
1
3
1
1
1
1
0
1
2
3
4
5

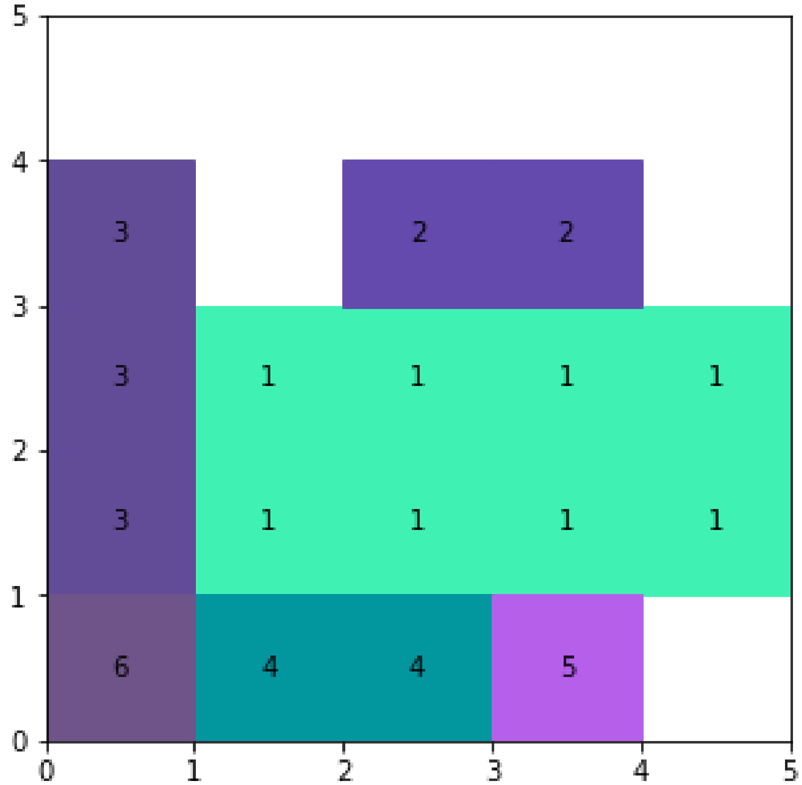


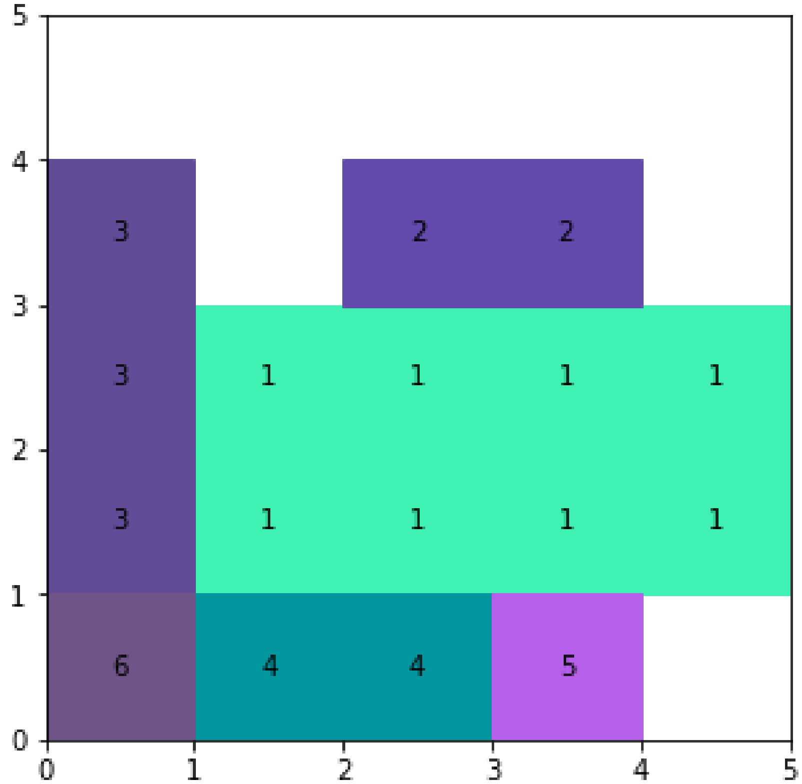
5
4
3
2
1
0
0
1
2
3
4
5
3
2
2
3
1
1
1
1
3
1
1
1
1
4
4



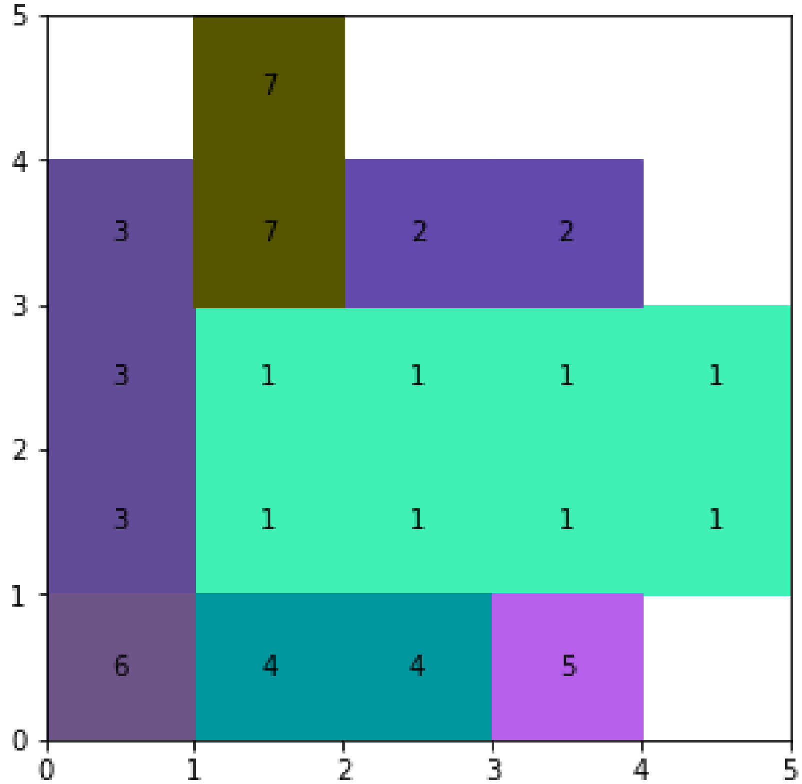
5
4
3
2
1
0
0
1
2
3
4
5
3
2
2
3
1
1
1
1
3
1
1
1
1
4
4

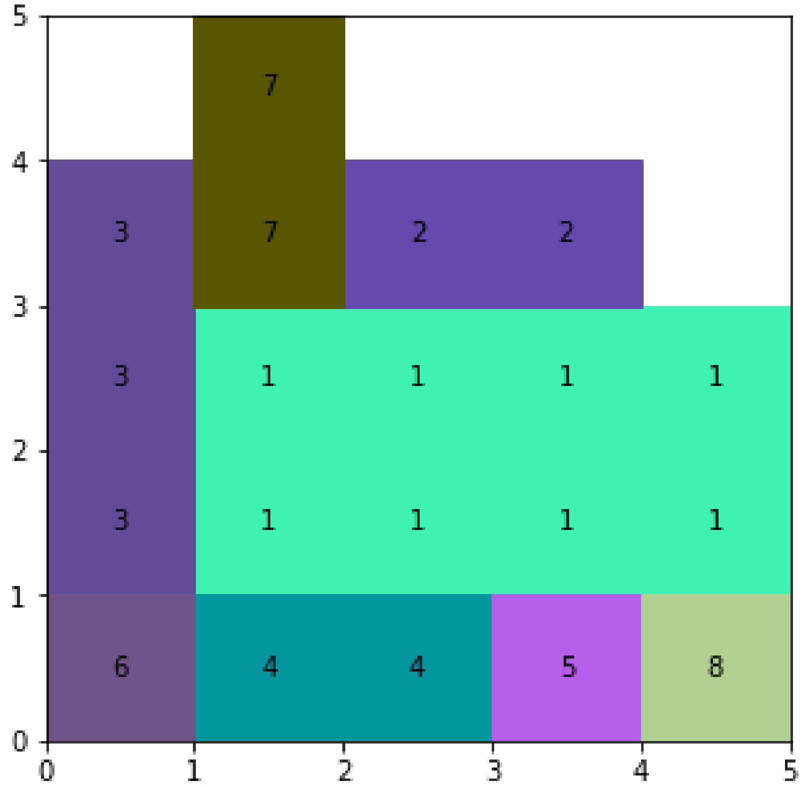

5
4
3
2
1
0
0
1
2
3
4
5
3
2
2
3
1
1
1
1
3
1
1
1
1
4
4
5


3
2
2
3
1
1
1
1
3
1
1
1
1
6
4
4
5
0
1
2
3
4
5

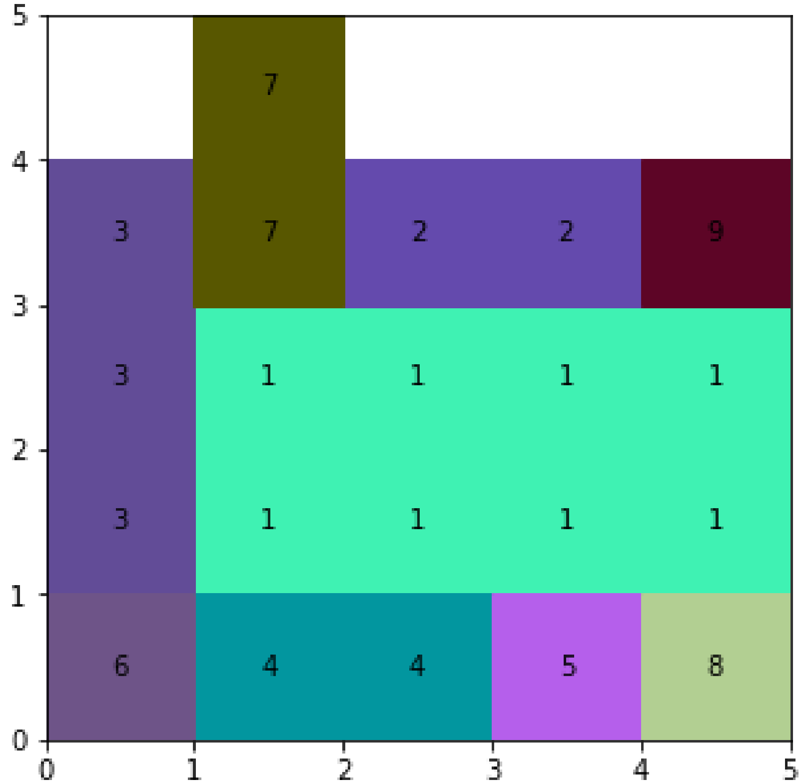


5
4
3
2
1
0
3
2
2
3
1
1
1
1
3
1
1
1
1
6
4
4
5
0
1
2
3
4
5

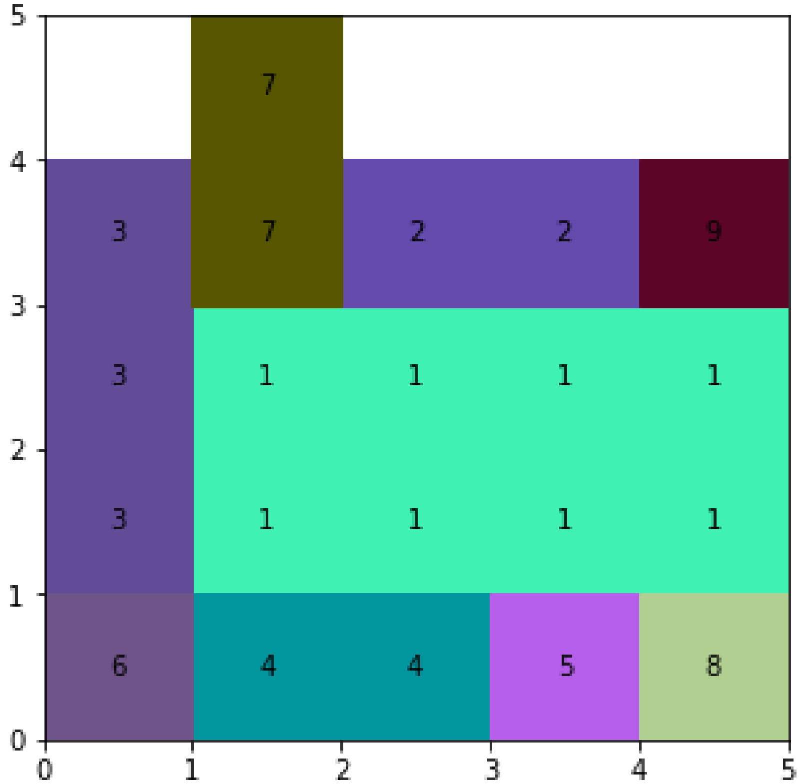

7
3
7
2
2
3
1
1
1
1
3
1
1
1
1
6
4
4
5
0
1
2
3
4
5

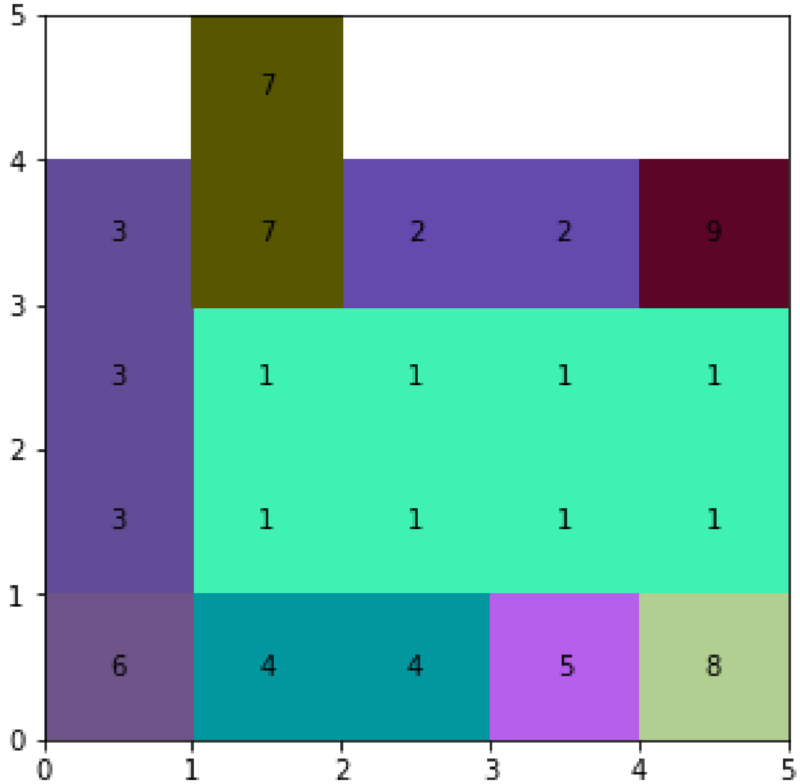


7
3
7
2
2
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

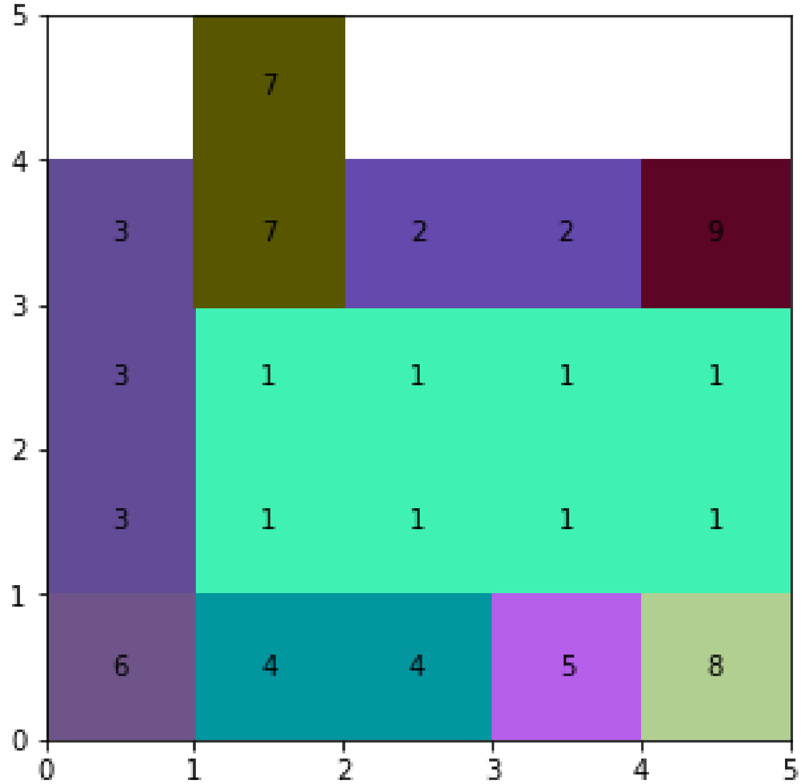


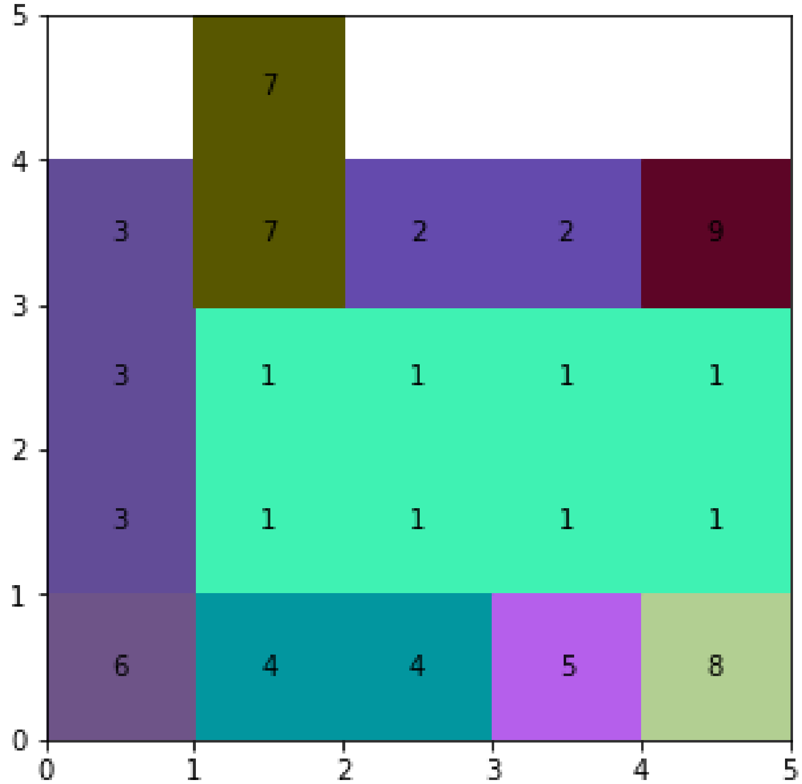
7
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5


5
7
4
3 7 2 2 9
3
3 1 1 1 1
2
3 1 1 1 1
1
6 4 4 5 8
0
0 1 2 3 4 5

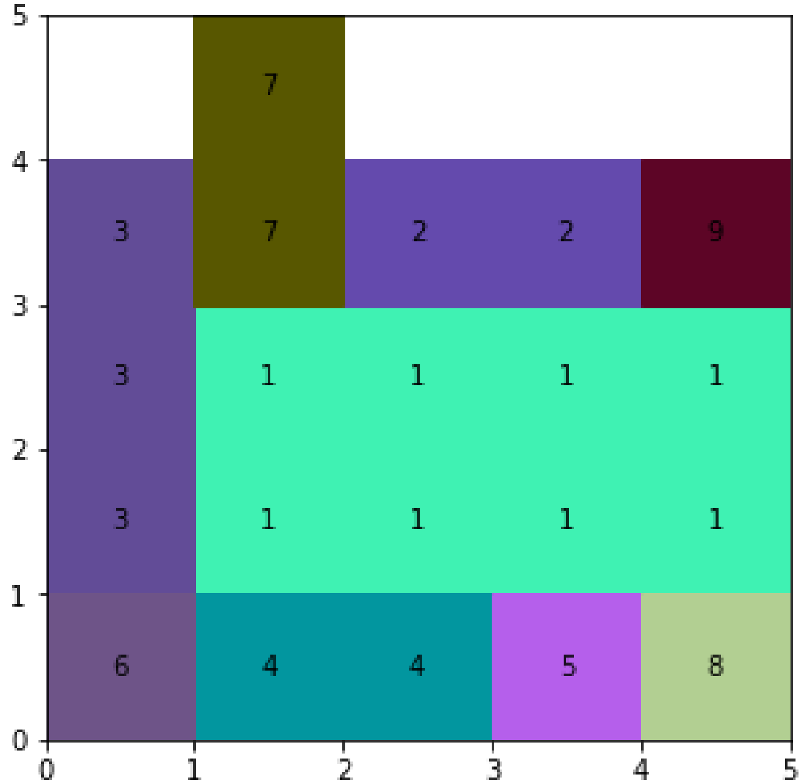


5
4
3
2
1
0
0
1
2
3
4
5
7
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8

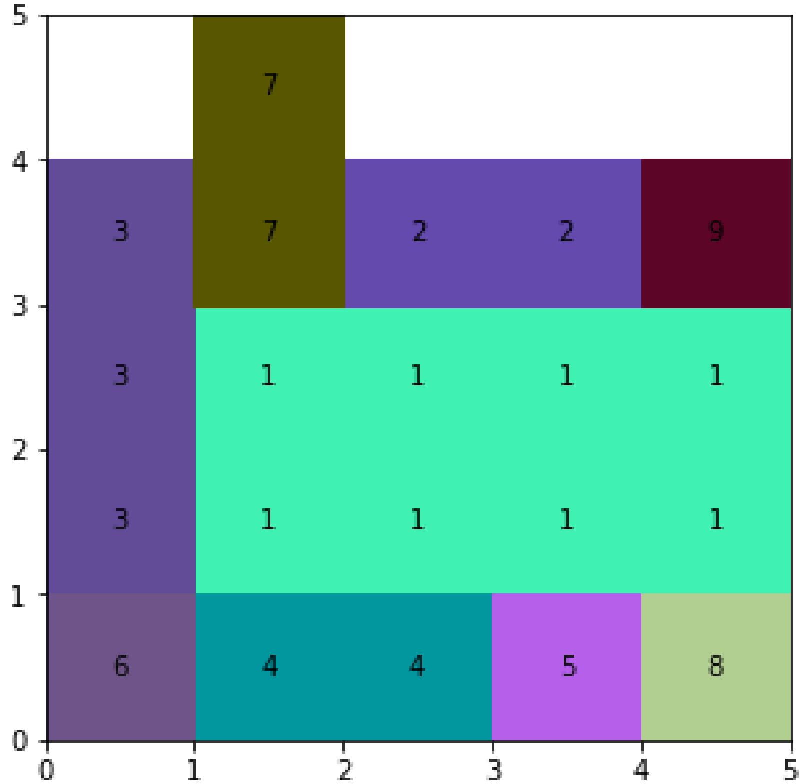

7
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

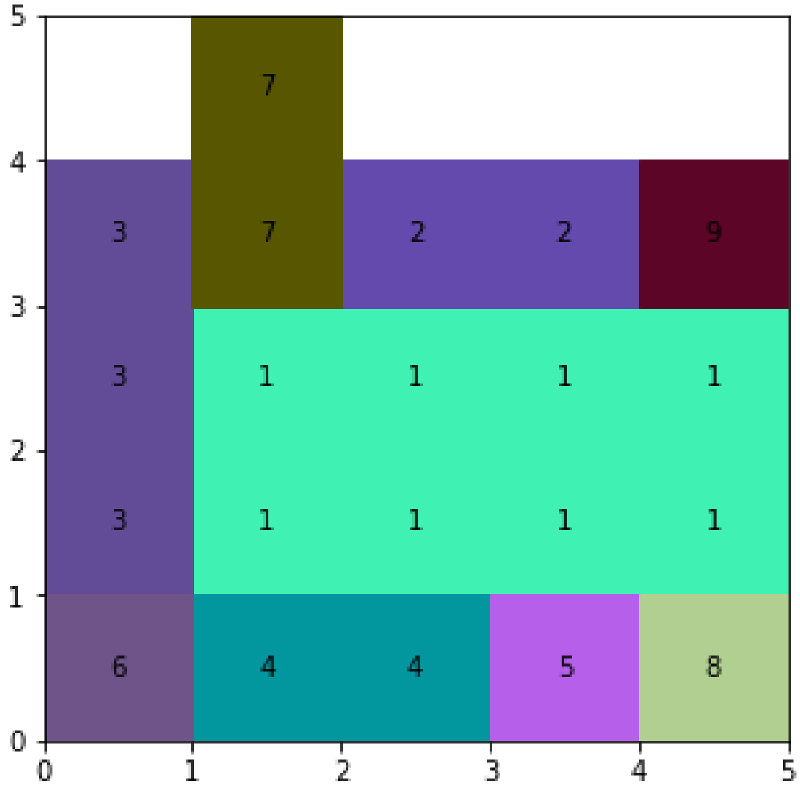


7
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

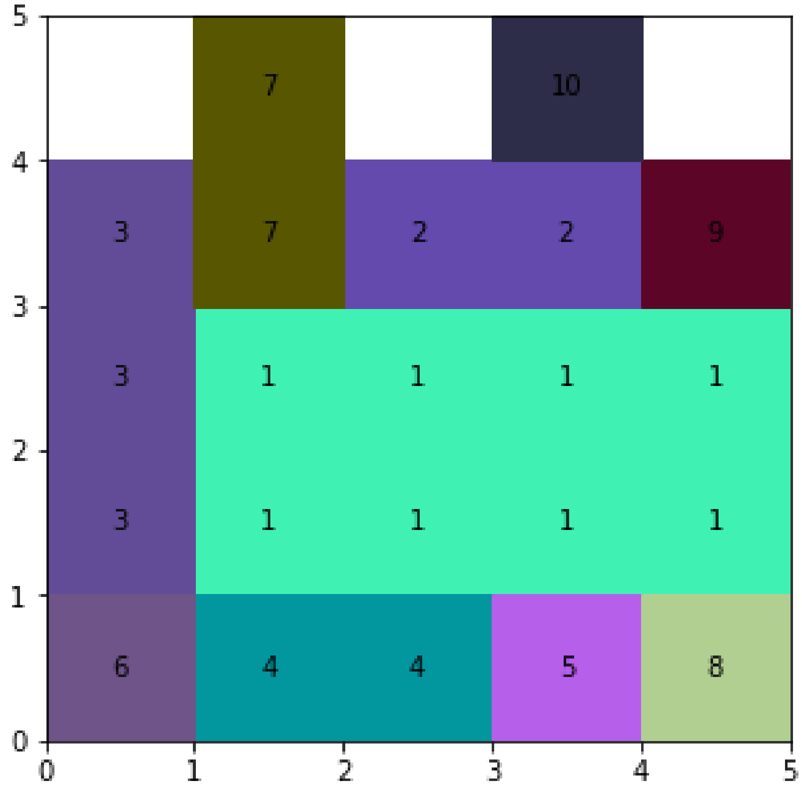


7
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8

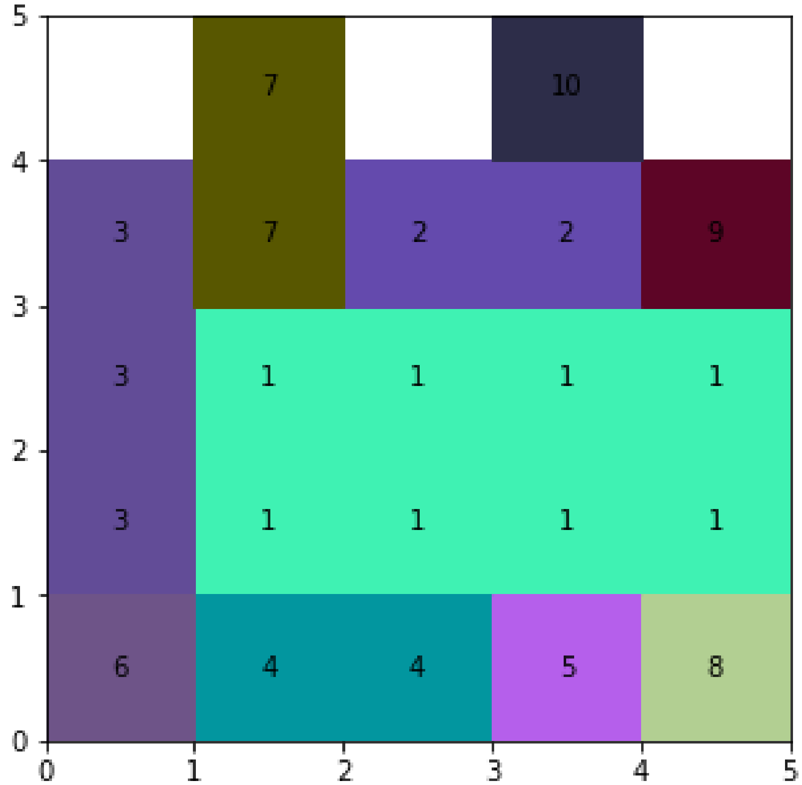

5
4
3
2
1
0
7
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5



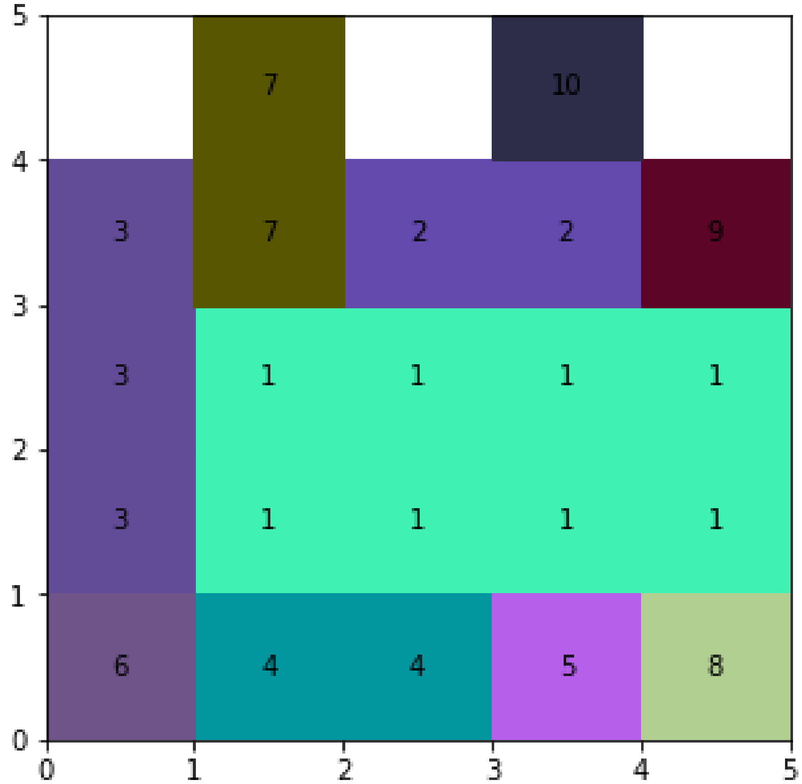
5
4
3
2
1
0
0
1
2
3
4
5
7
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8



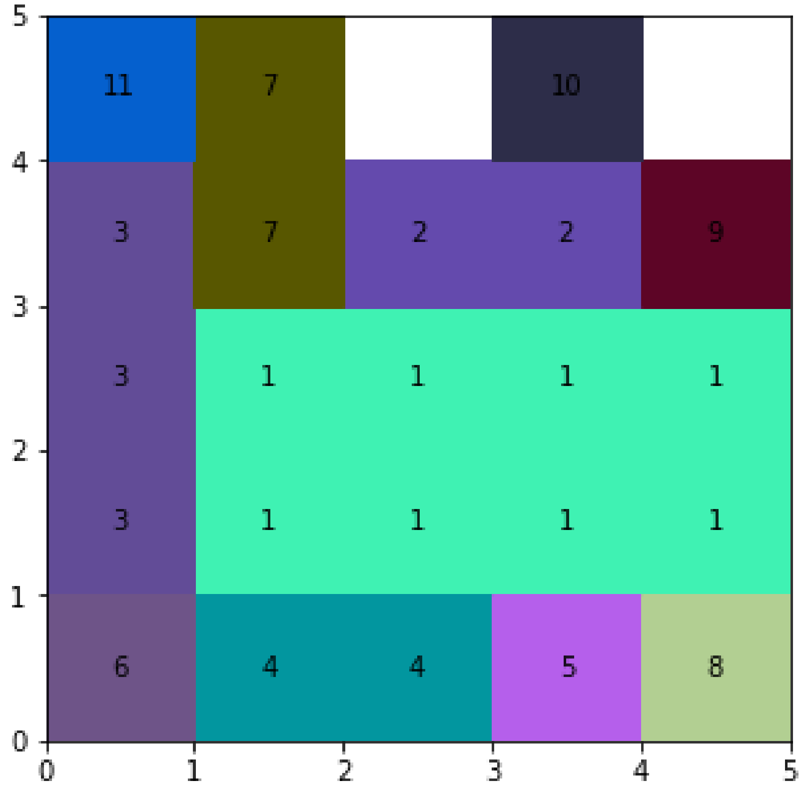
7
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5



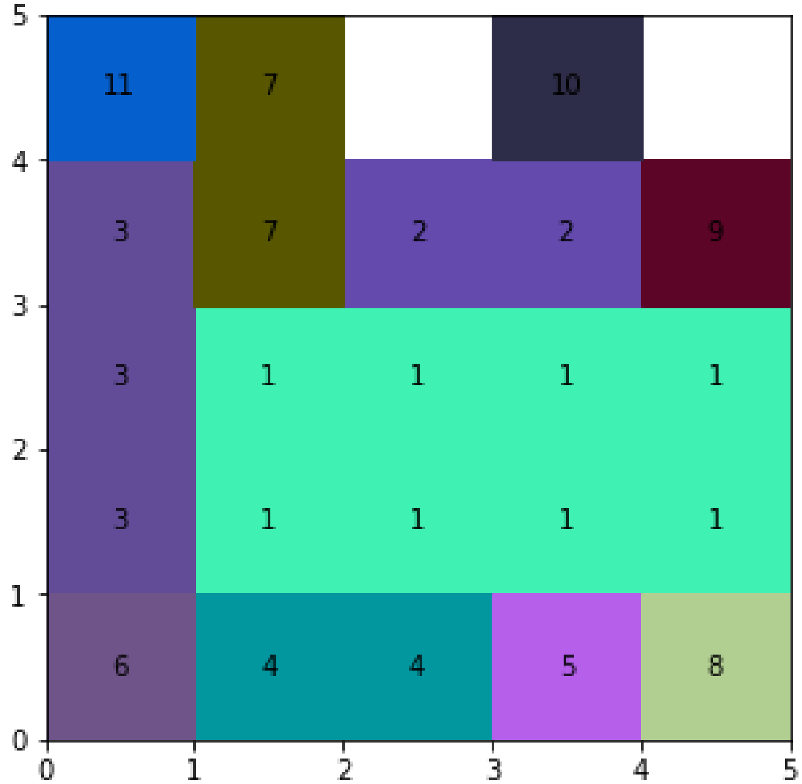
7
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5



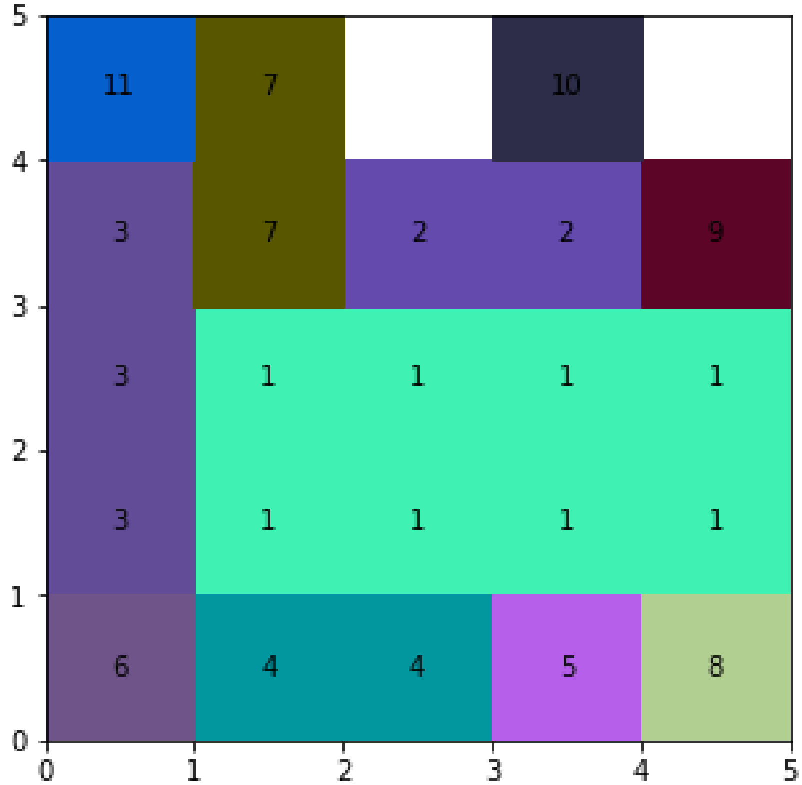
7
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5



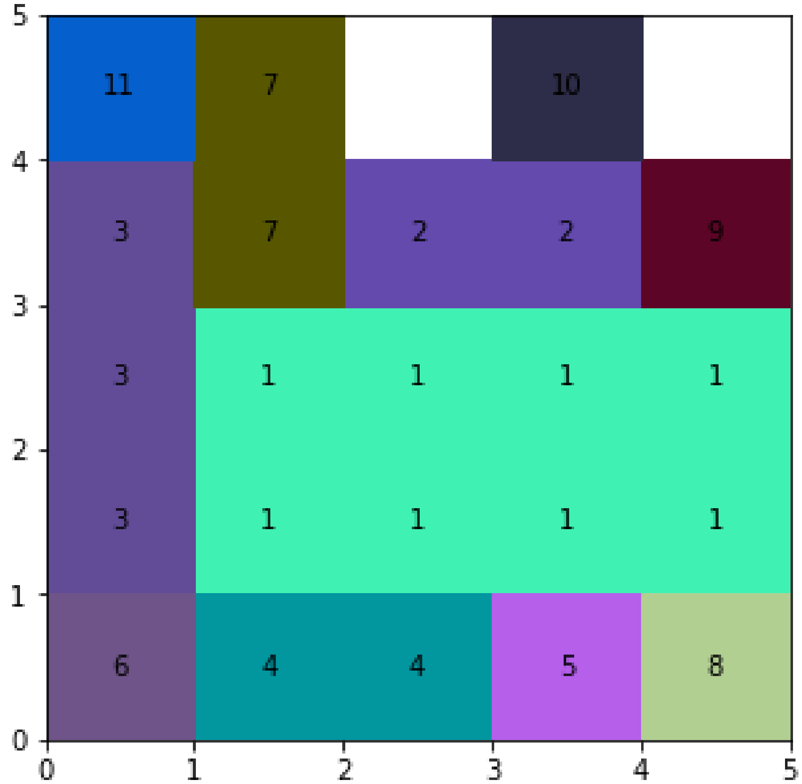
11
7
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5



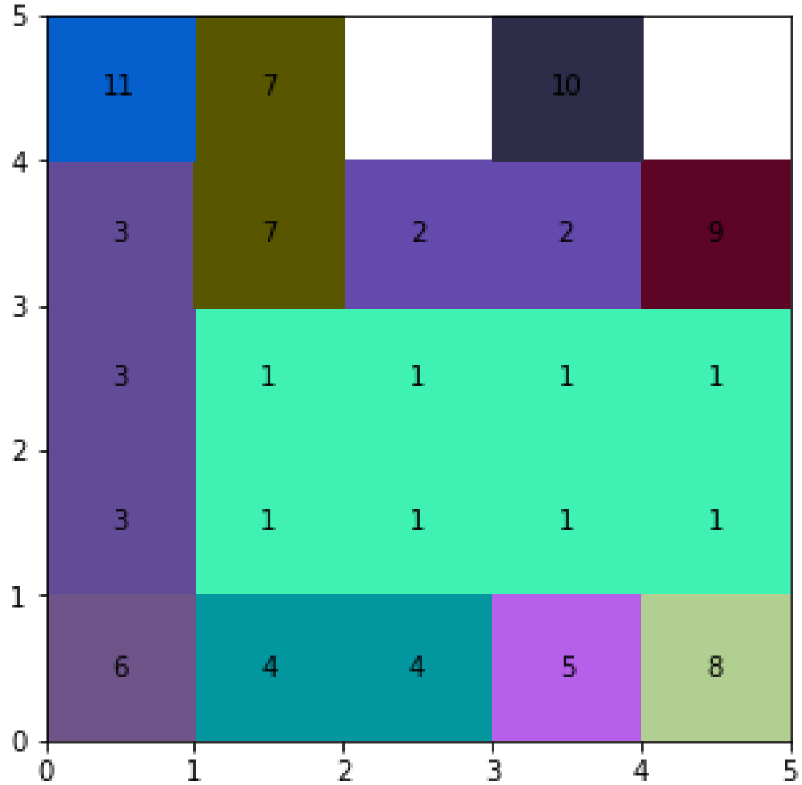
11
7
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

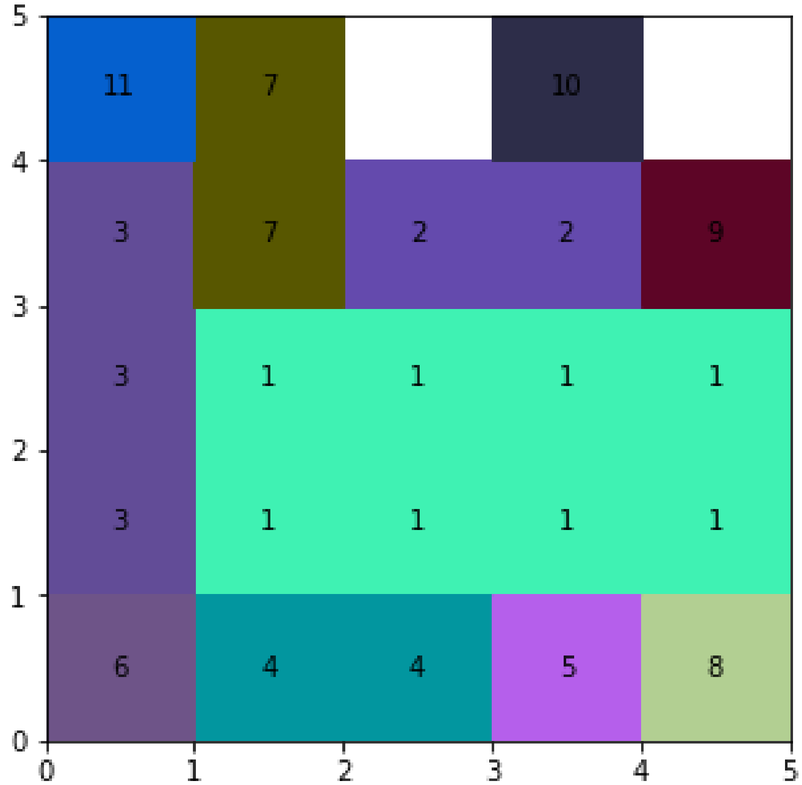


11
7
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

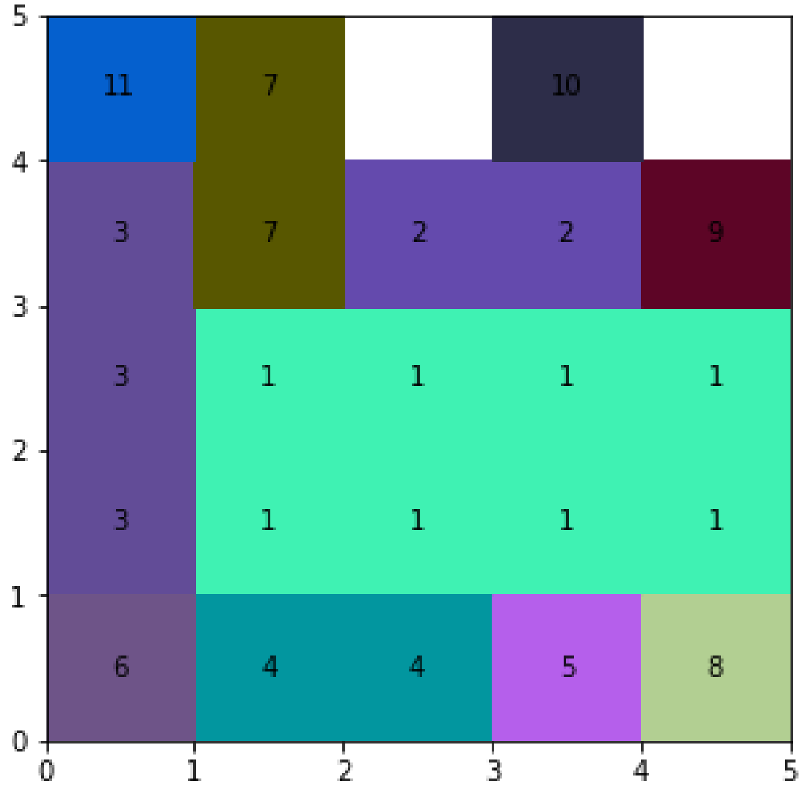


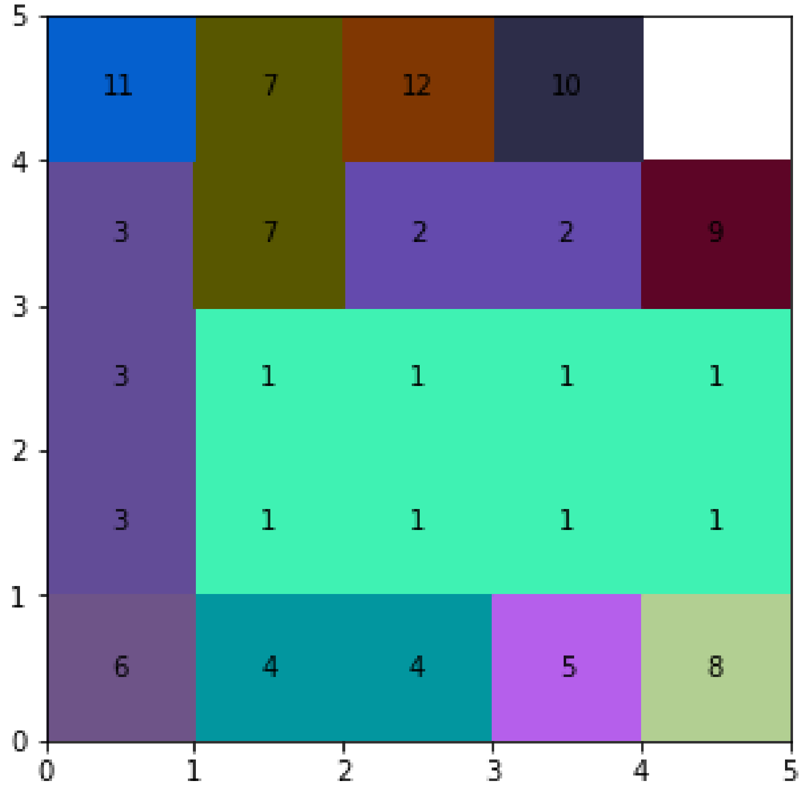
11
7
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5


11
7
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

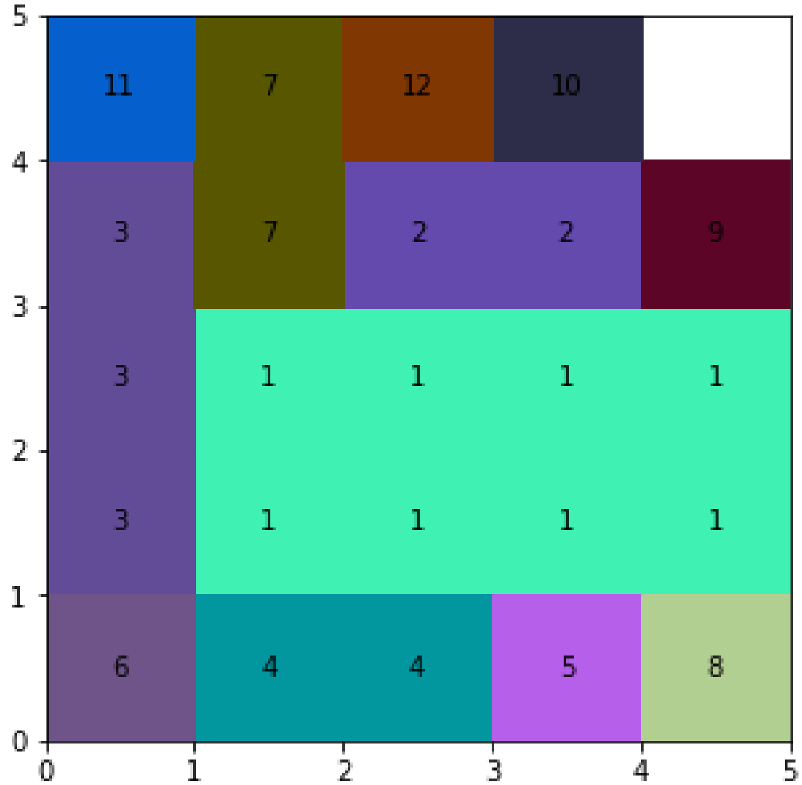


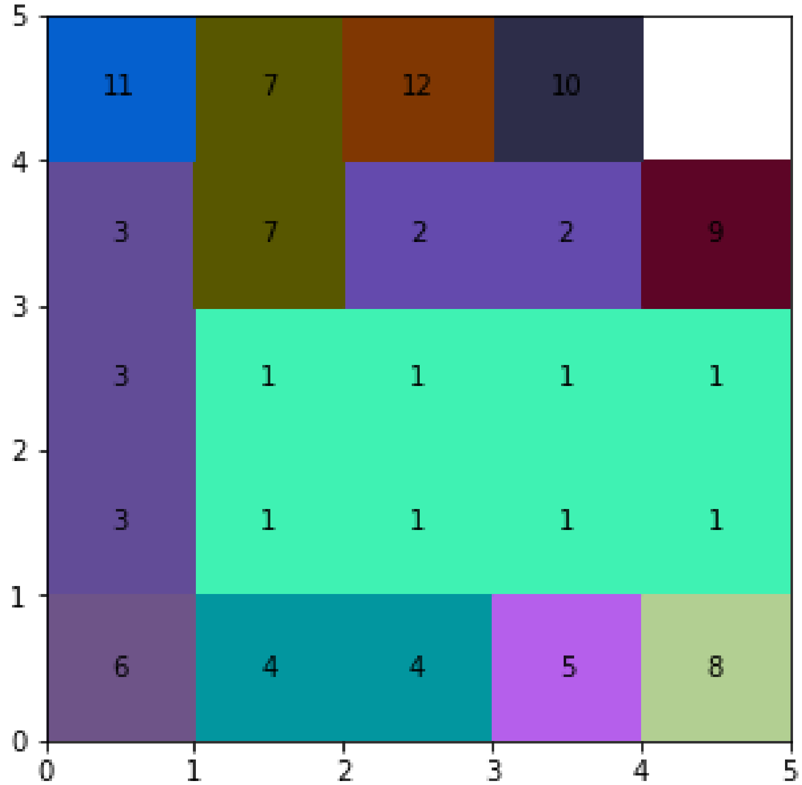
11
7
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5
0
1
2
3
4
5


11
7
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

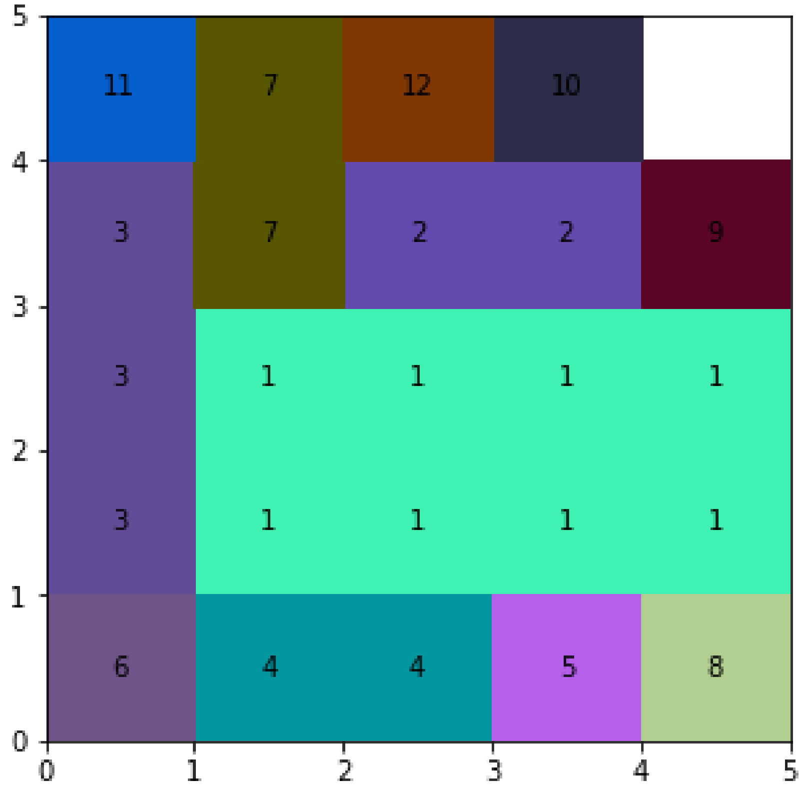


11
7
12
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

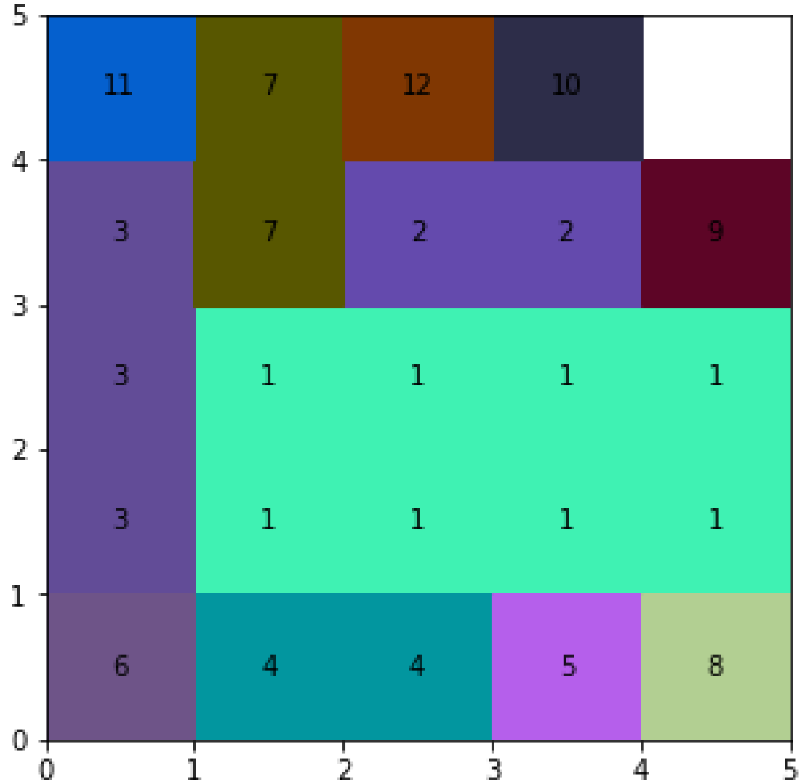

11
7
12
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5



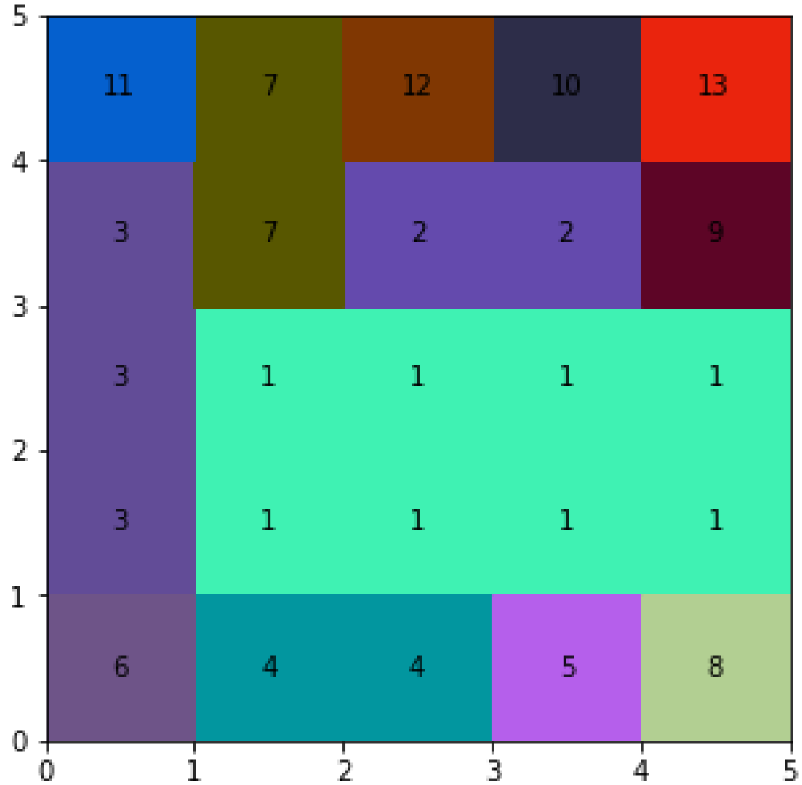
11
7
12
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

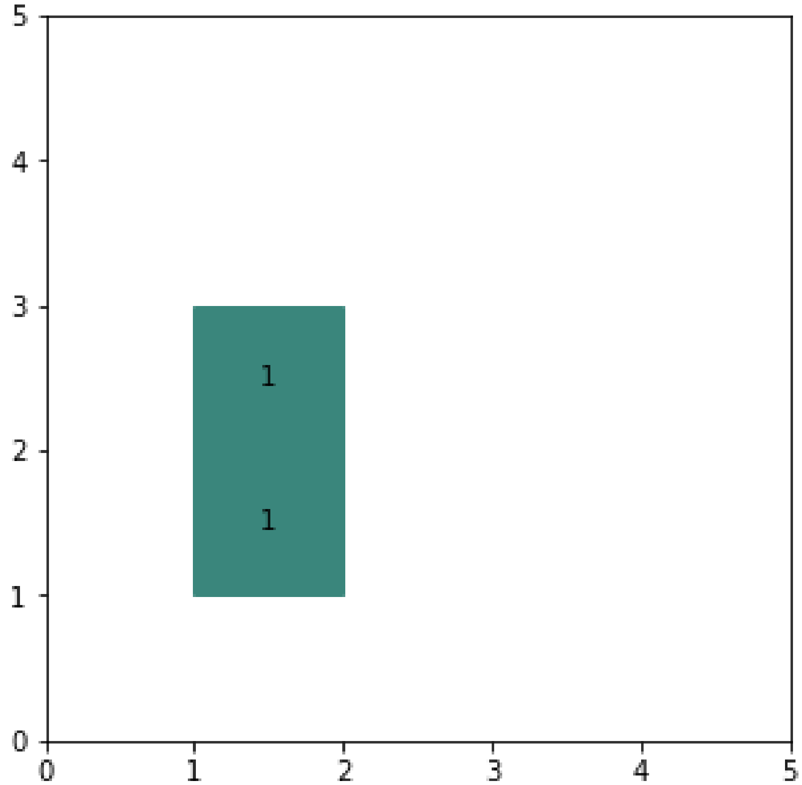


11
7
12
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

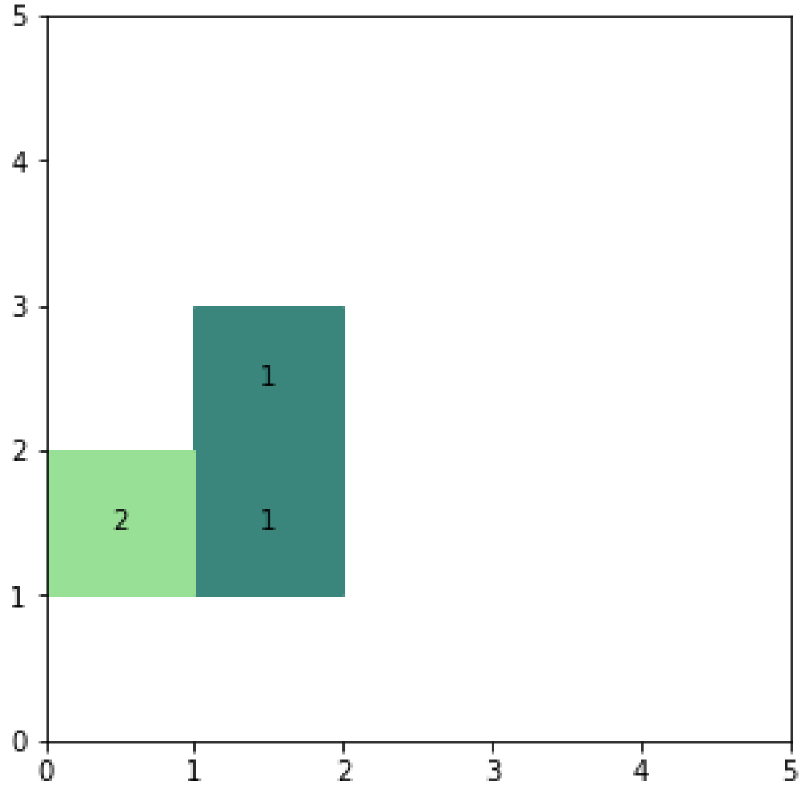


11
7
12
10
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

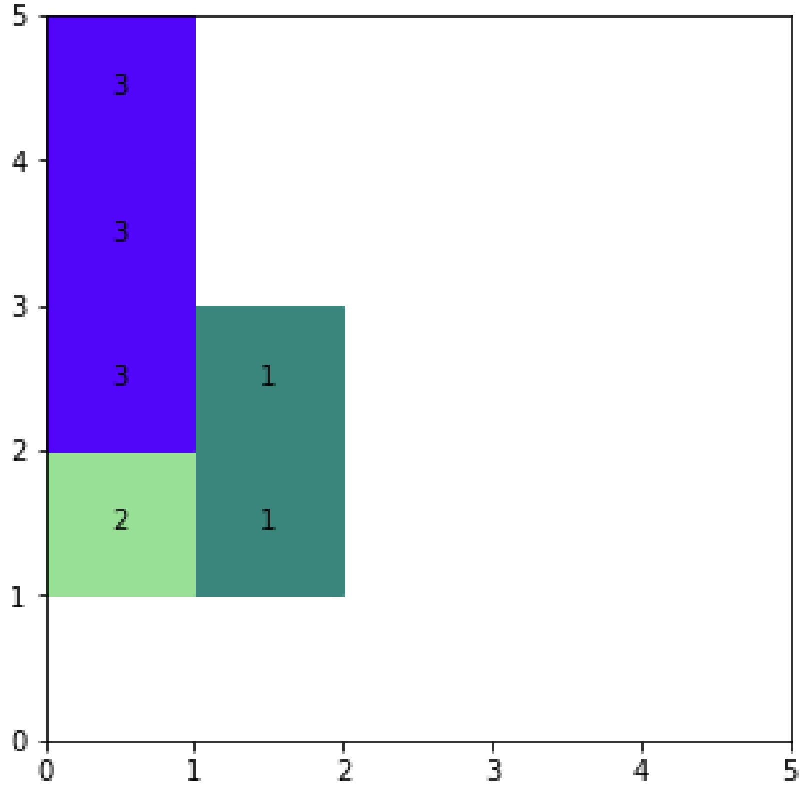

11
7
12
10
13
3
7
2
2
9
3
1
1
1
1
3
1
1
1
1
6
4
4
5
8
0
1
2
3
4
5

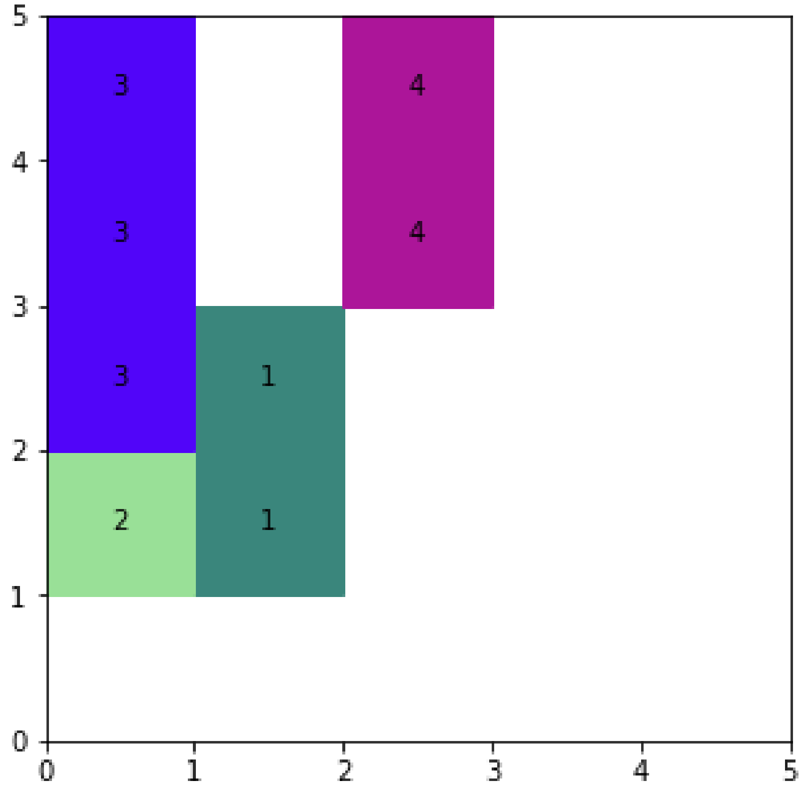


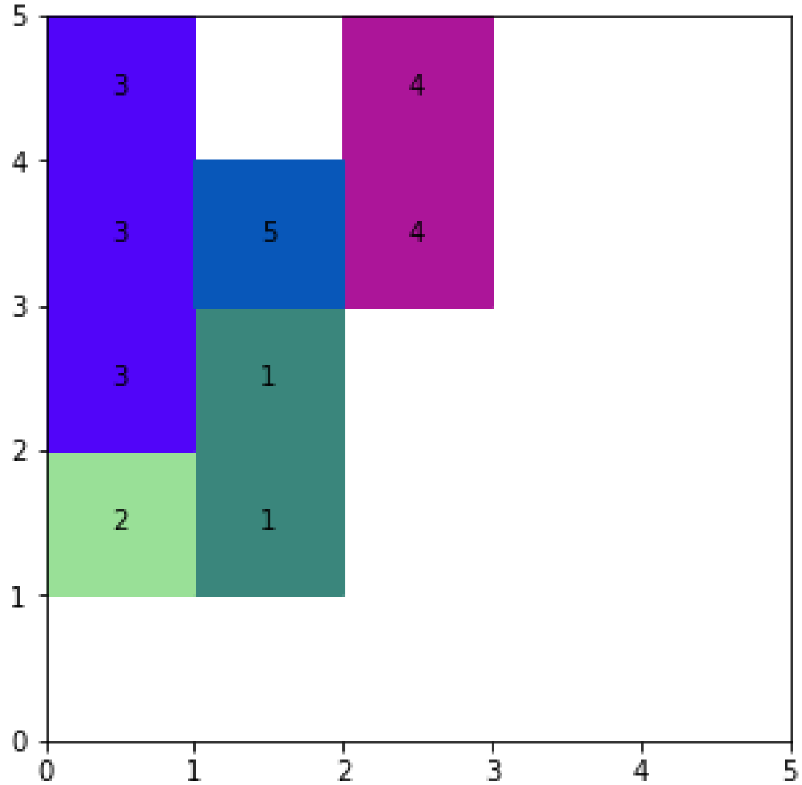
5
4
3
2
1
0
1
1
0
1
2
3
4
5

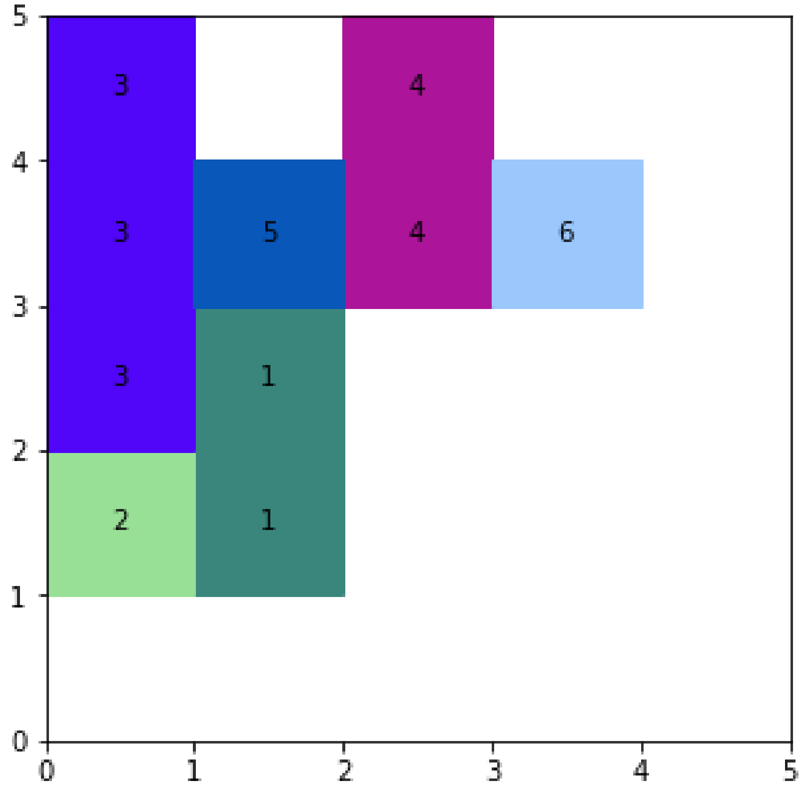


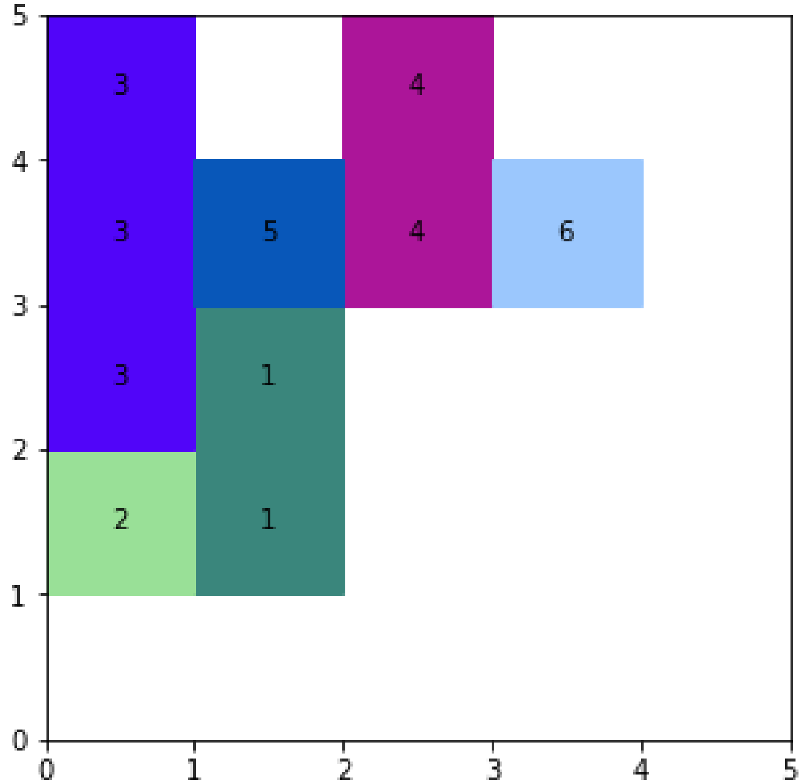
5
4
3
2
1
0
0
1
2
3
4
5
1
2
1

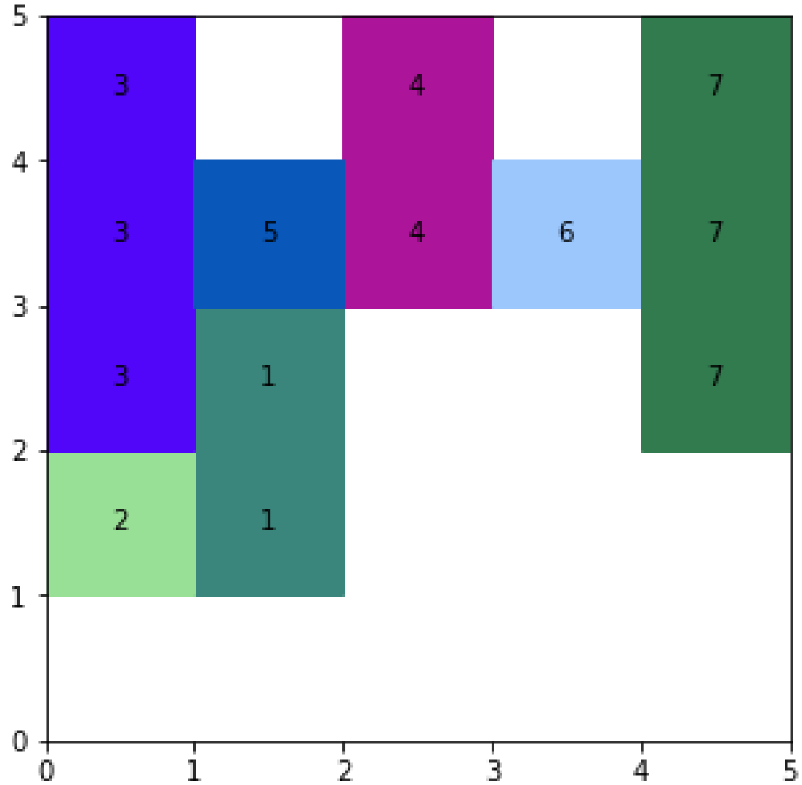

3
3
3
1
2
1
0
1
2
3
4
5


3
3
3
4
4
1
1
2
0
1
2
3
4
5
0
1
2
3
4
5

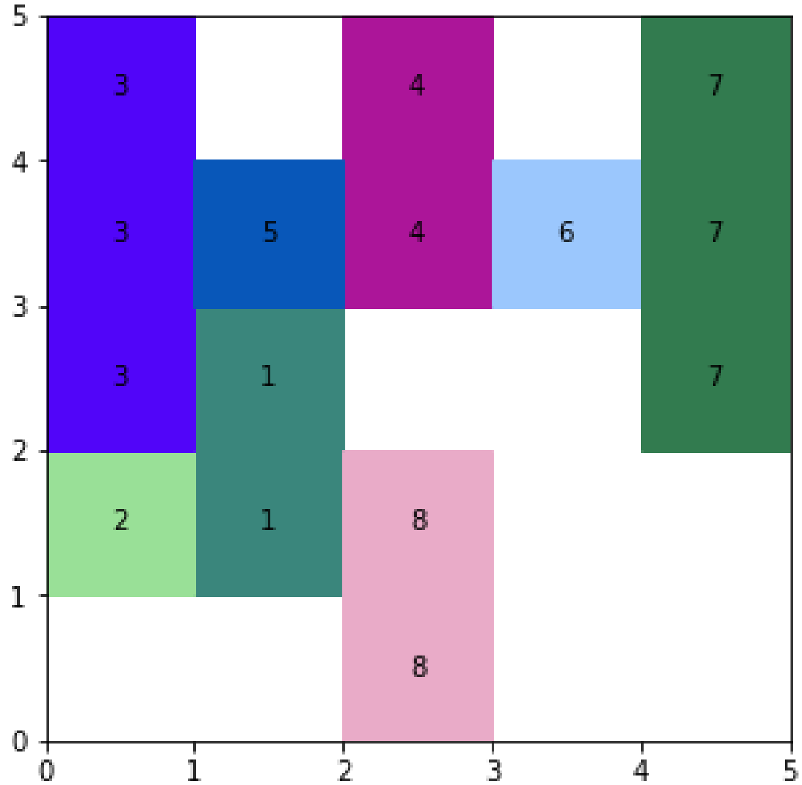

3
4
3
5
4
3
1
2
1
0
1
2
3
4
5
0
1
2
3
4
5

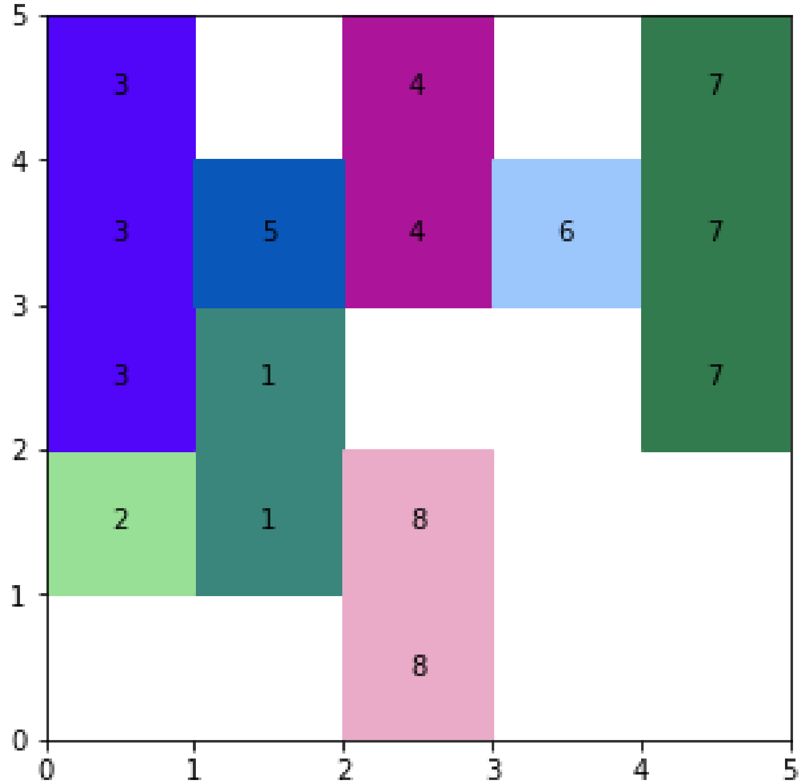

3
4
3
5
4
6
3
1
2
1
0
1
2
3
4
5

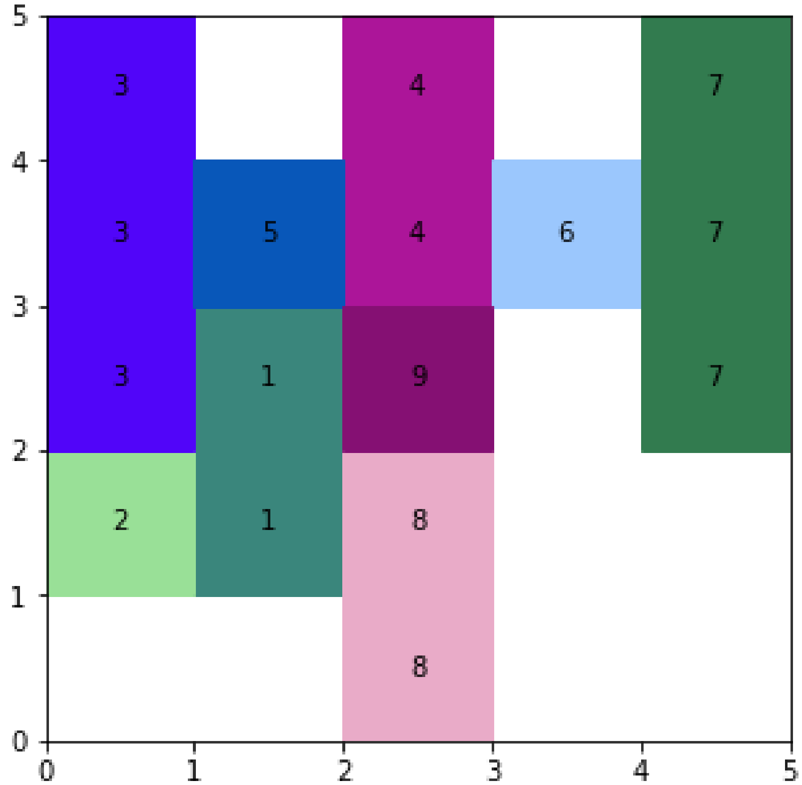

3
4
3
5
4
6
3
1
2
1
0
1
2
3
4
5

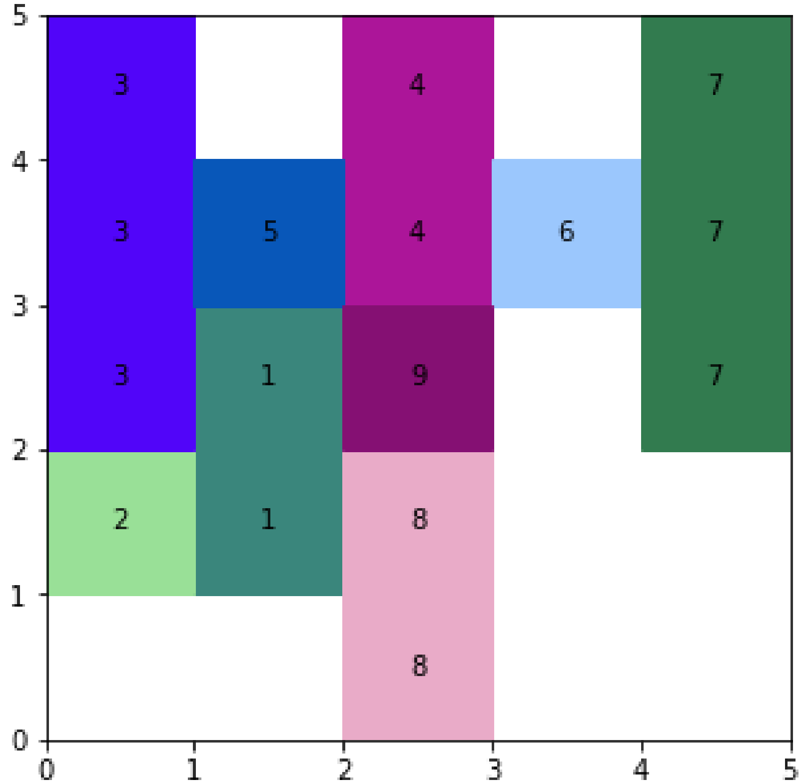


3
4
7
3
5
4
6
7
3
1
7
2
1
0
1
2
3
4
5

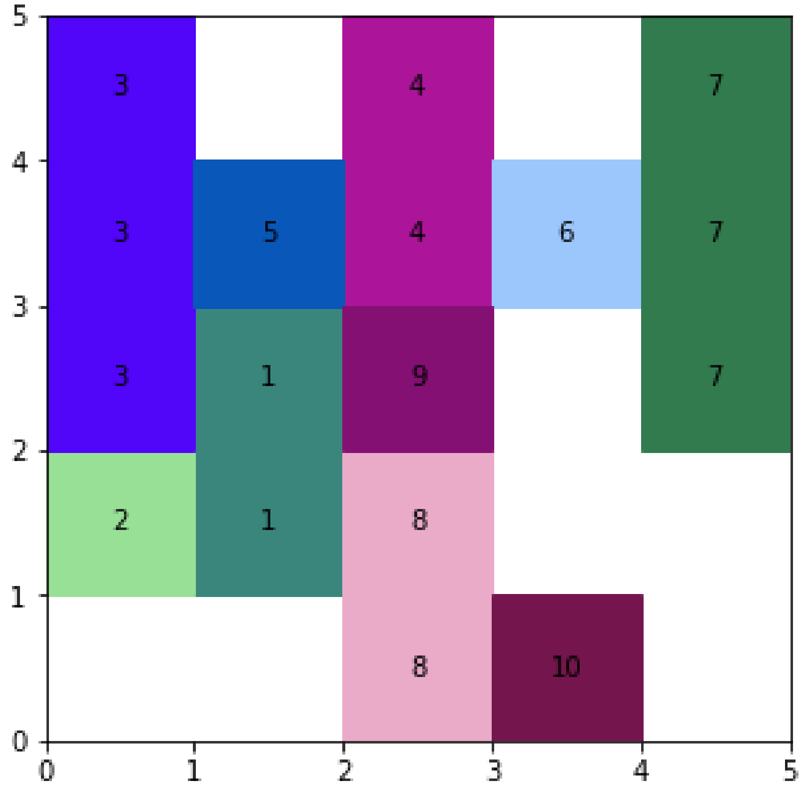

3
4
7
3
5
4
6
7
3
1
7
2
1
8
8
0
1
2
3
4
5
0
1
2
3
4
5

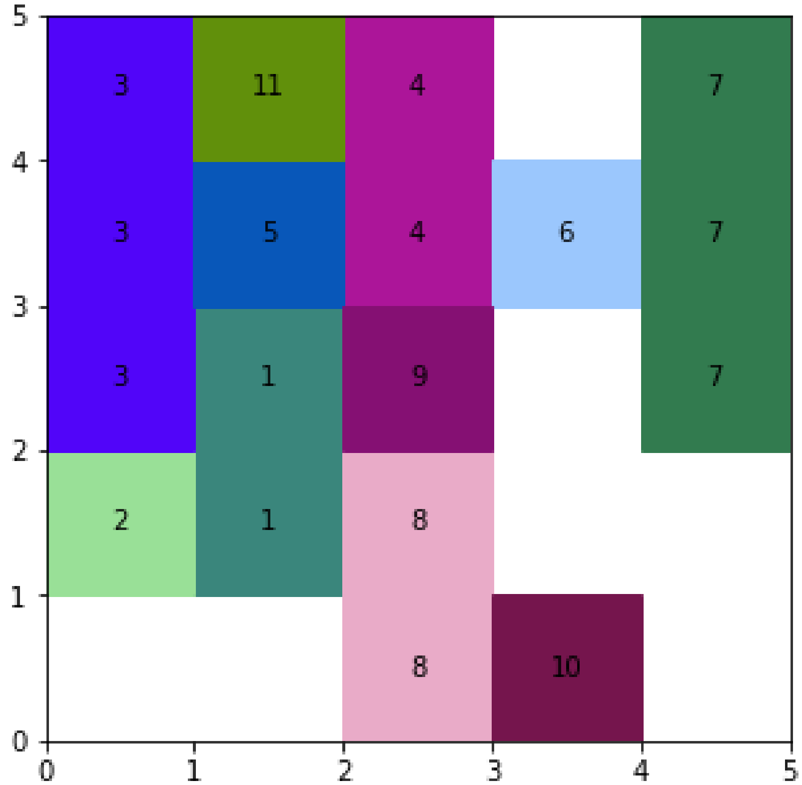

3
4
7
3
5
4
6
7
3
1
7
2
1
8
8
0
1
2
3
4
5
0
1
2
3
4
5

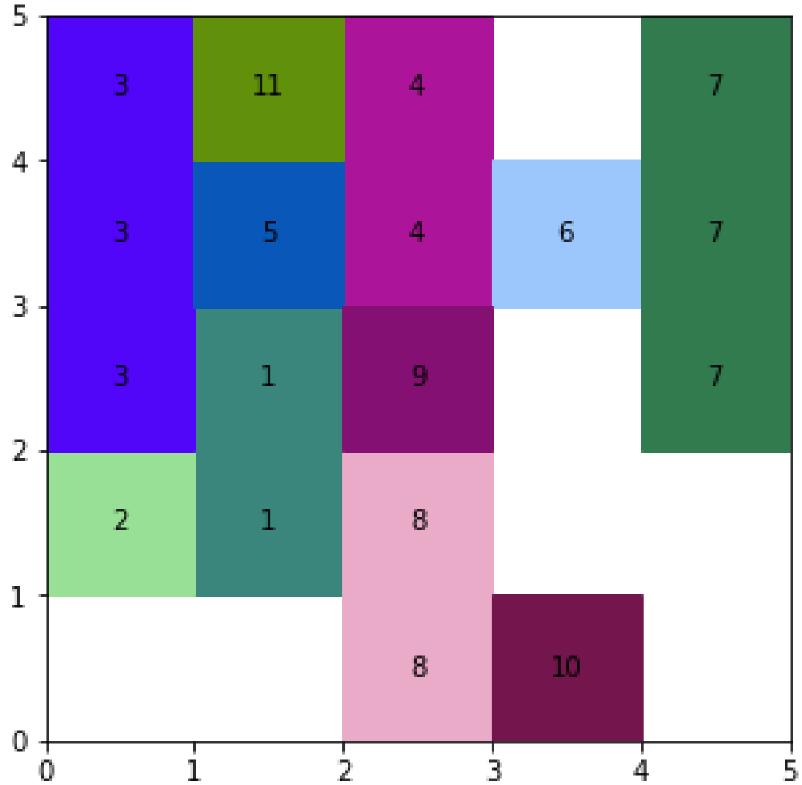

3
4
7
3
5
4
6
7
3
1
9
7
2
1
8
8
0
1
2
3
4
5
0
1
2
3
4
5



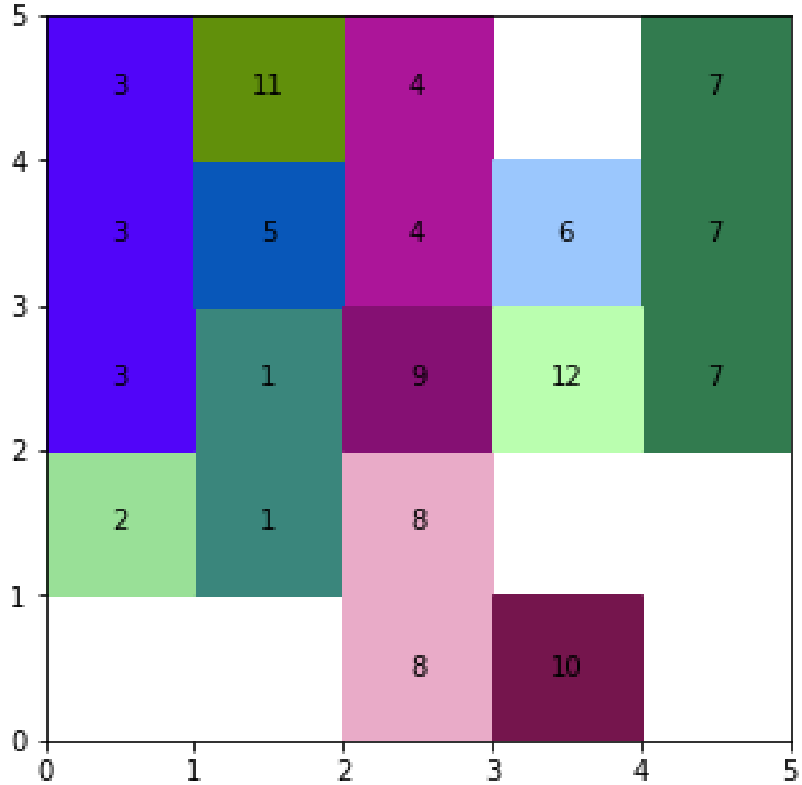
3
4
7
3
5
4
6
7
3
1
9
7
2
1
8
8
0
1
2
3
4
5


3
4
7
3
5
4
6
7
3
1
9
7
2
1
8
8
10
0
1
2
3
4
5

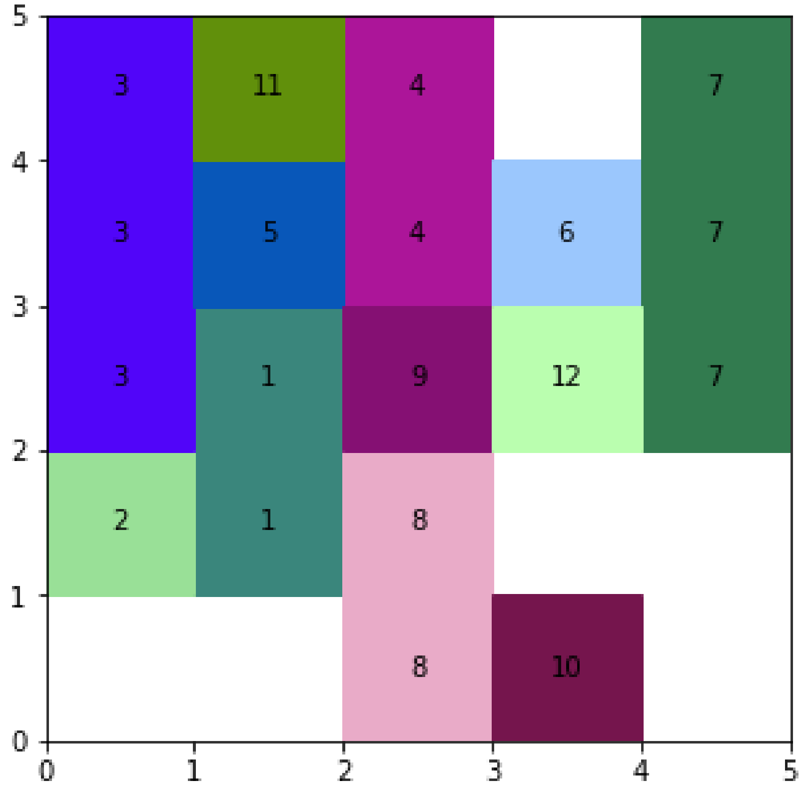

3
11
4
7
3
5
4
6
7
3
1
9
7
2
1
8
8
10
0
1
2
3
4
5



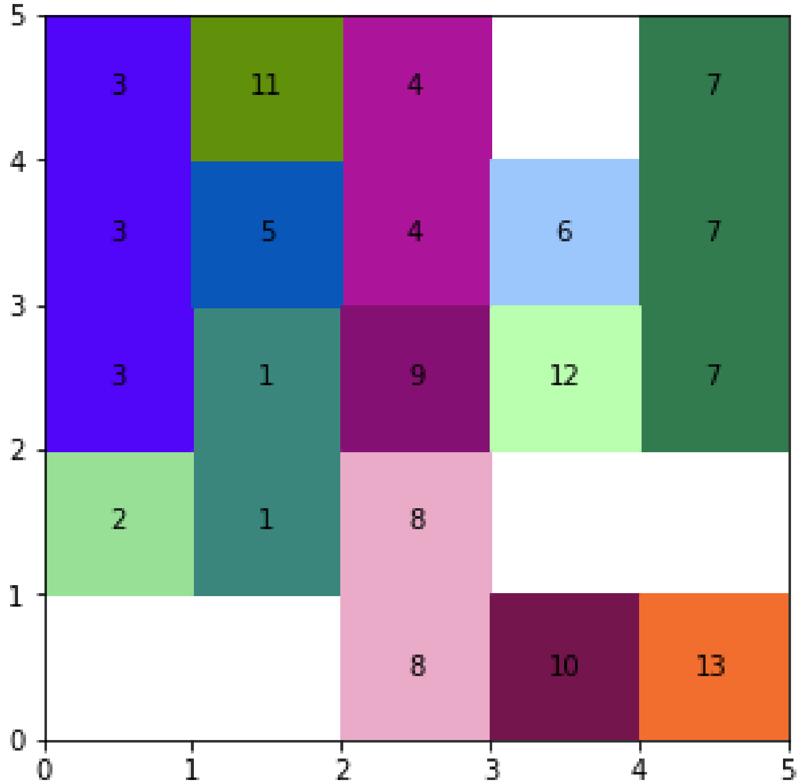
3
11
4
7
3
5
4
6
7
3
1
9
7
2
1
8
8
10
0
1
2
3
4
5



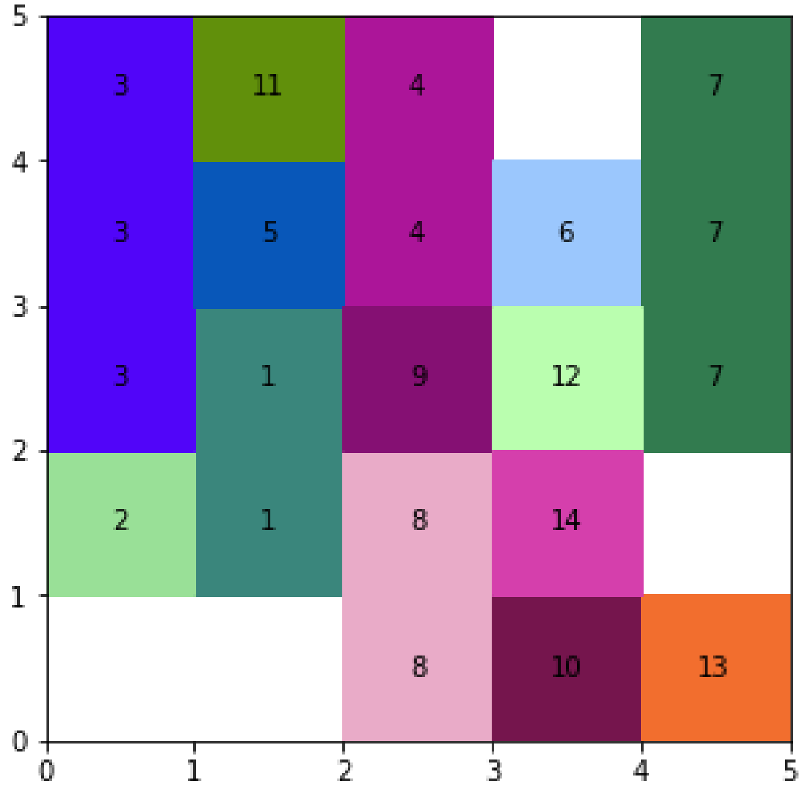
3
11
4
7
3
5
4
6
7
3
1
9
12
7
2
1
8
8
10
0
1
2
3
4
5



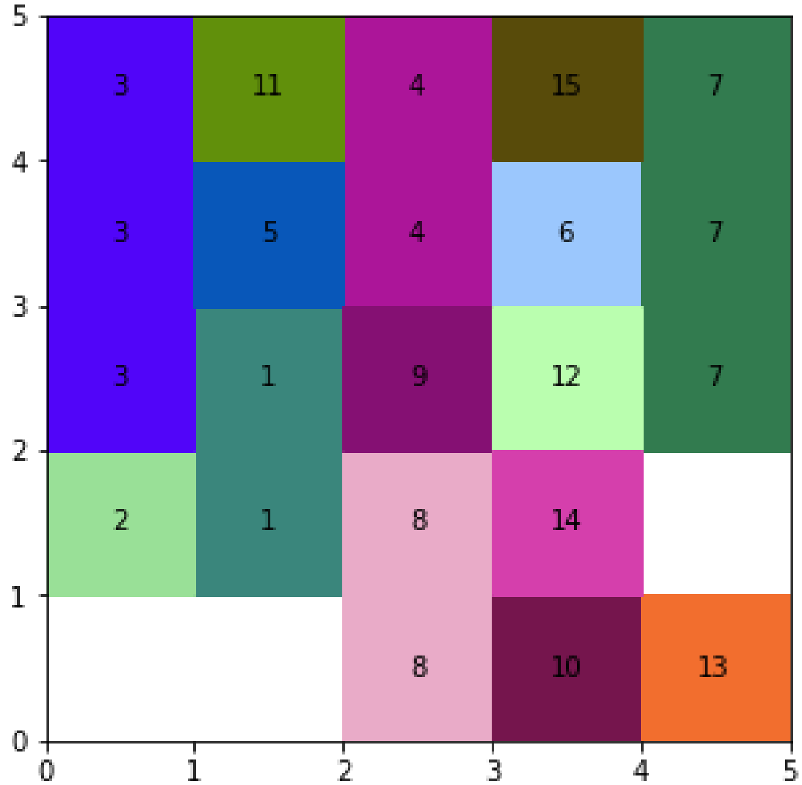
3
11
4
7
3
5
4
6
7
3
1
9
12
7
2
1
8
8
10
0
1
2
3
4
5
0
1
2
3
4
5



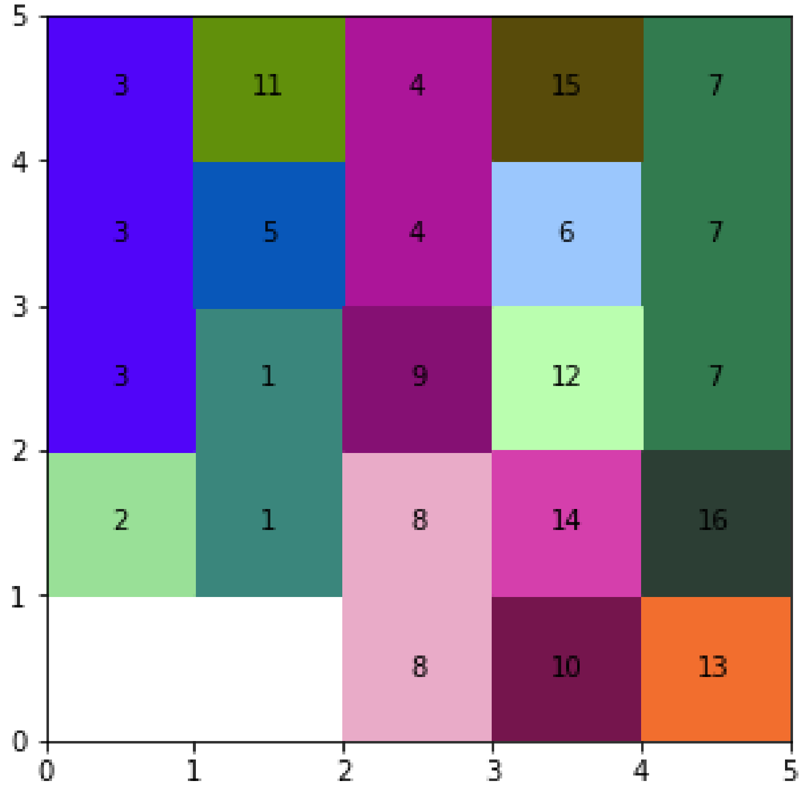
3
11
4
7
3
5
4
6
7
3
1
9
12
7
2
1
8
8
10
13
0
1
2
3
4
5

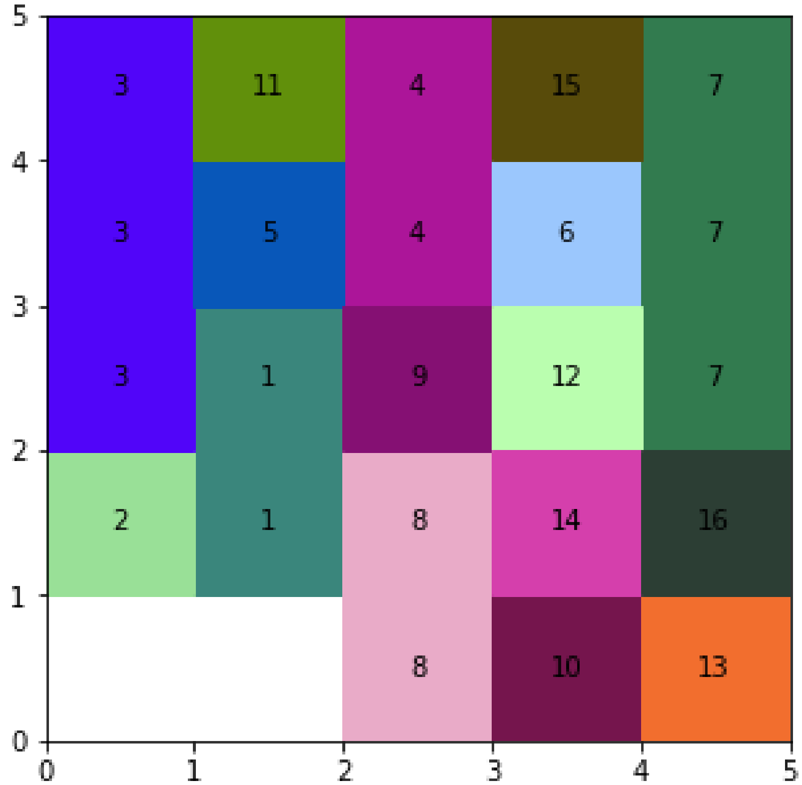


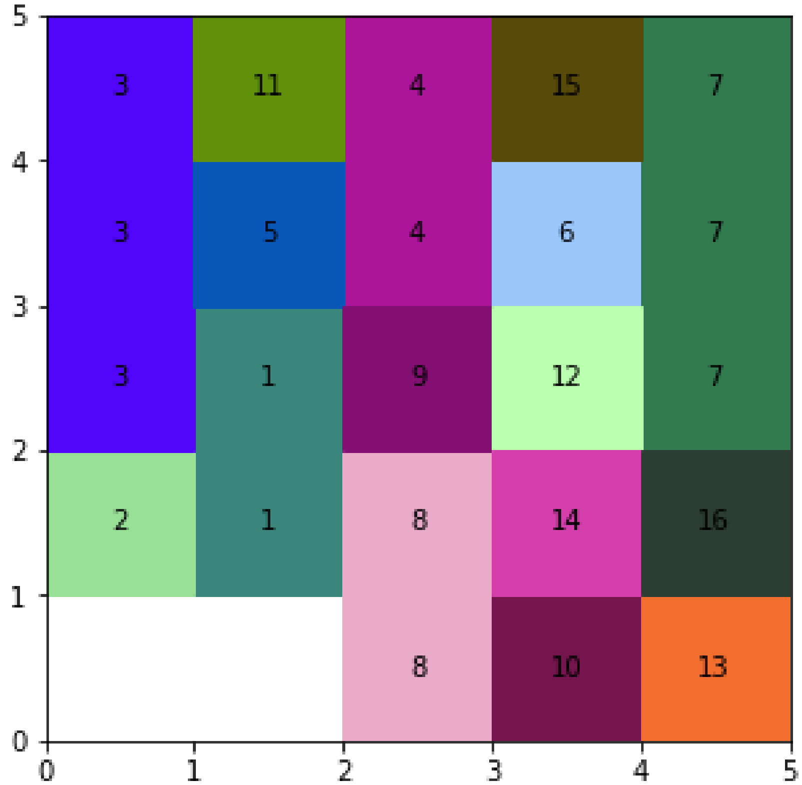
3
11
4
7
3
5
4
6
7
3
1
9
12
7
2
1
8
14
8
10
13
0
1
2
3
4
5



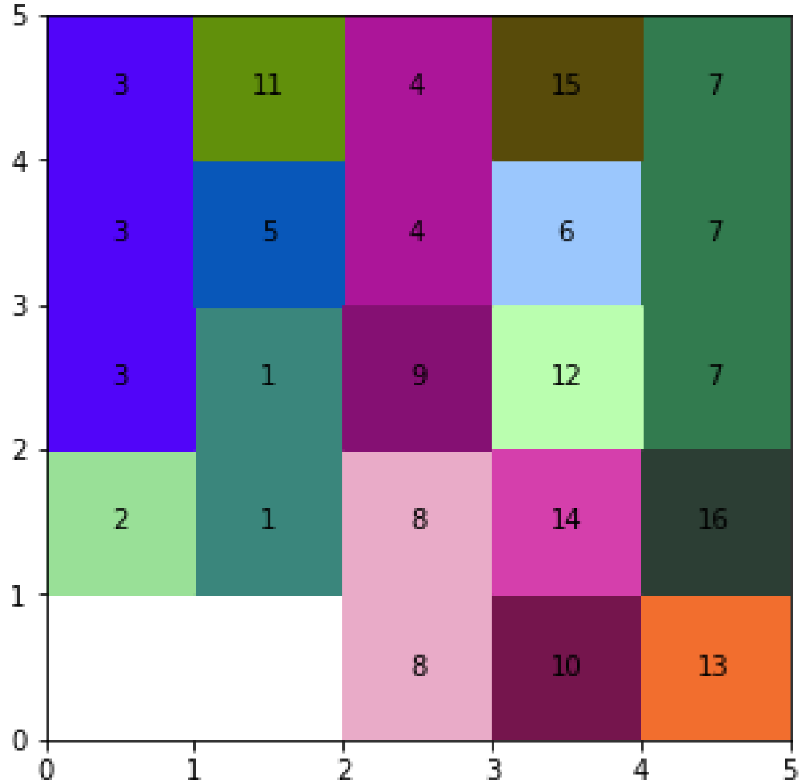
3
11
4
15
7
3
5
4
6
7
3
1
9
12
7
2
1
8
14
8
10
13
0
1
2
3
4
5


3
11
4
15
7
3
5
4
6
7
3
1
9
12
7
2
1
8
14
16
8
10
13
0
1
2
3
4
5

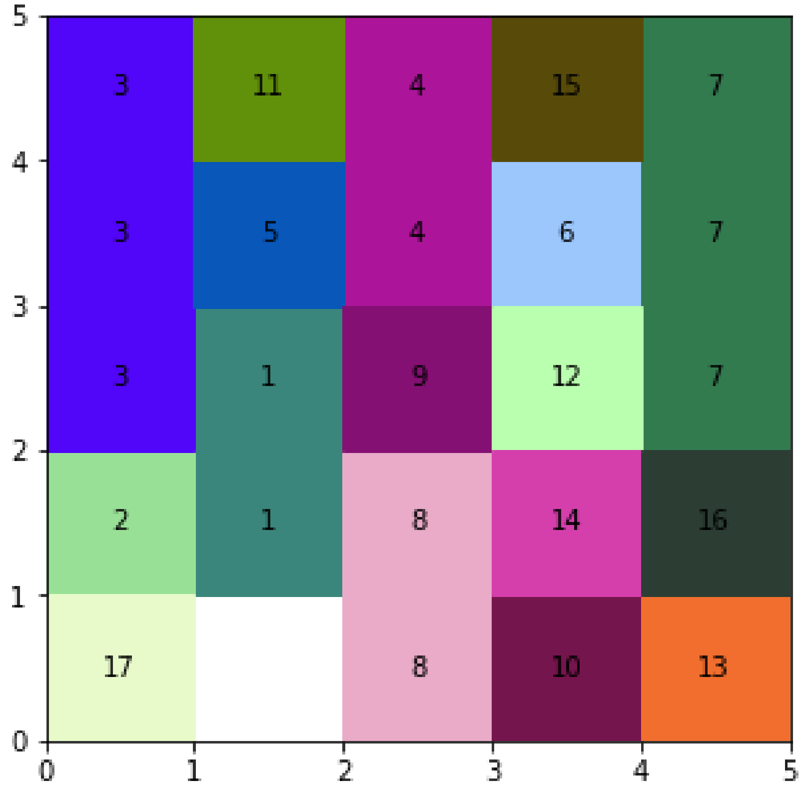

3
11
4
15
7
3
5
4
6
7
3
1
9
12
7
2
1
8
14
16
8
10
13
0
1
2
3
4
5

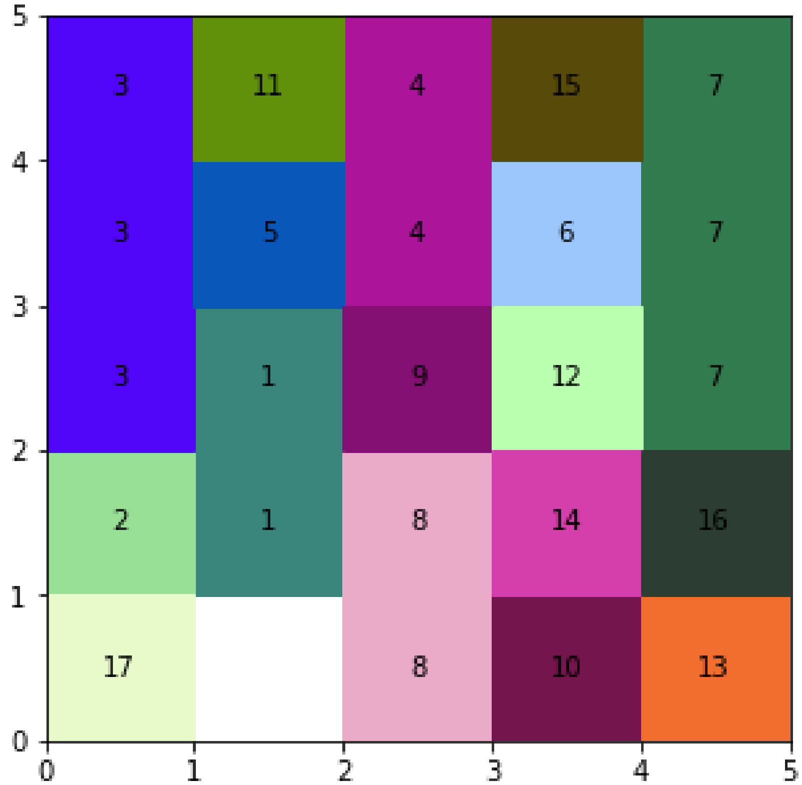


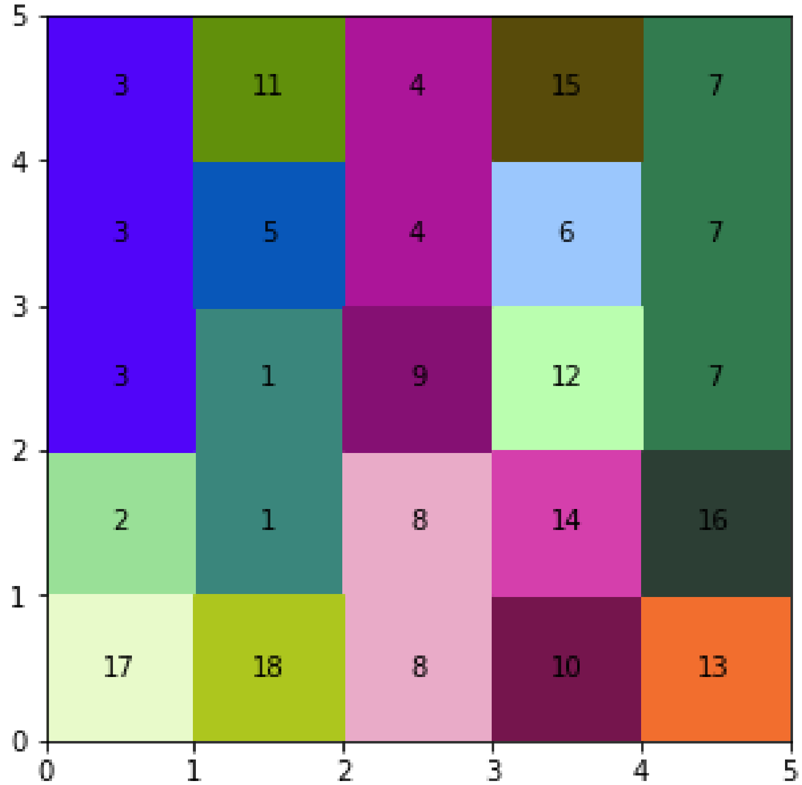
3
11
4
15
7
3
5
4
6
7
3
1
9
12
7
2
1
8
14
16
8
10
13
0
1
2
3
4
5



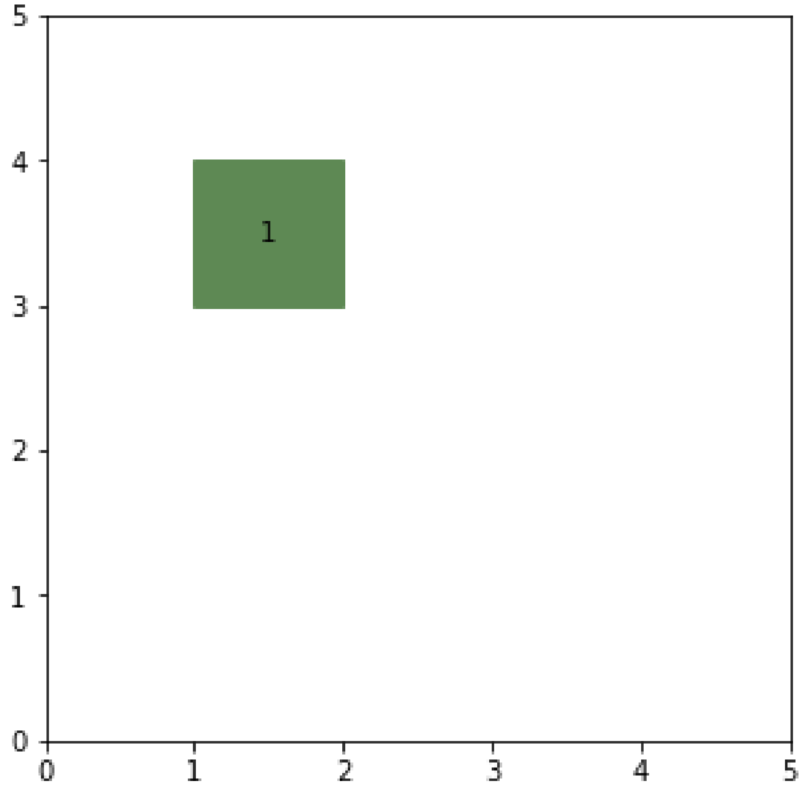
3
11
4
15
7
3
5
4
6
7
3
1
9
12
7
2
1
8
14
16
8
10
13
0
1
2
3
4
5


3
11
4
15
7
3
5
4
6
7
3
1
9
12
7
2
1
8
14
16
17
8
10
13
0
1
2
3
4
5

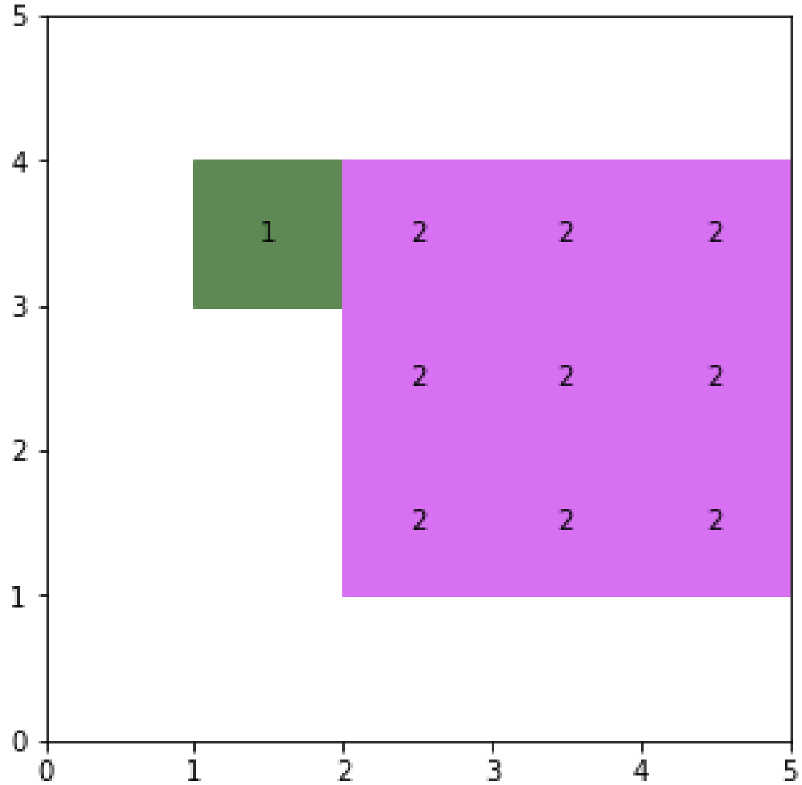

3
11
4
15
7
3
5
4
6
7
3
1
9
12
7
2
1
8
14
16
17
8
10
13
0
1
2
3
4
5

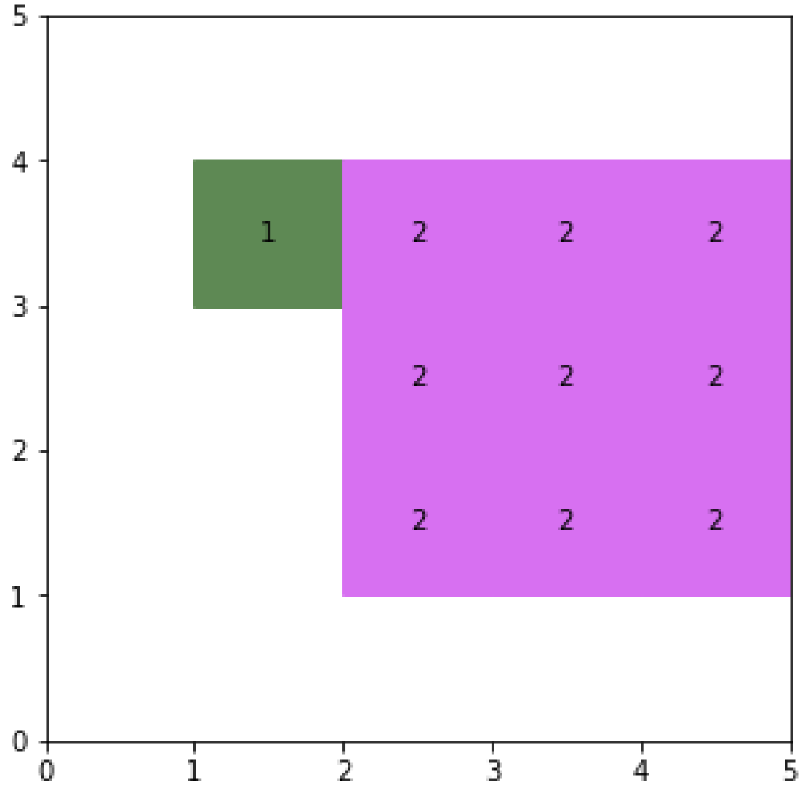


3
11
4
15
7
3
5
4
6
7
3
1
9
12
7
2
1
8
14
16
17
18
8
10
13
0
1
2
3
4
5

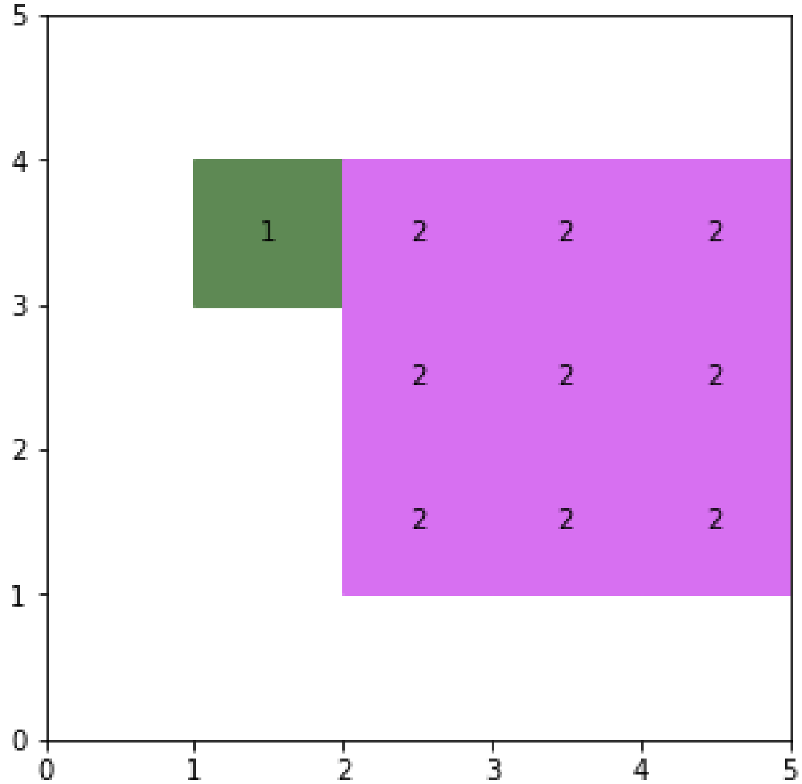


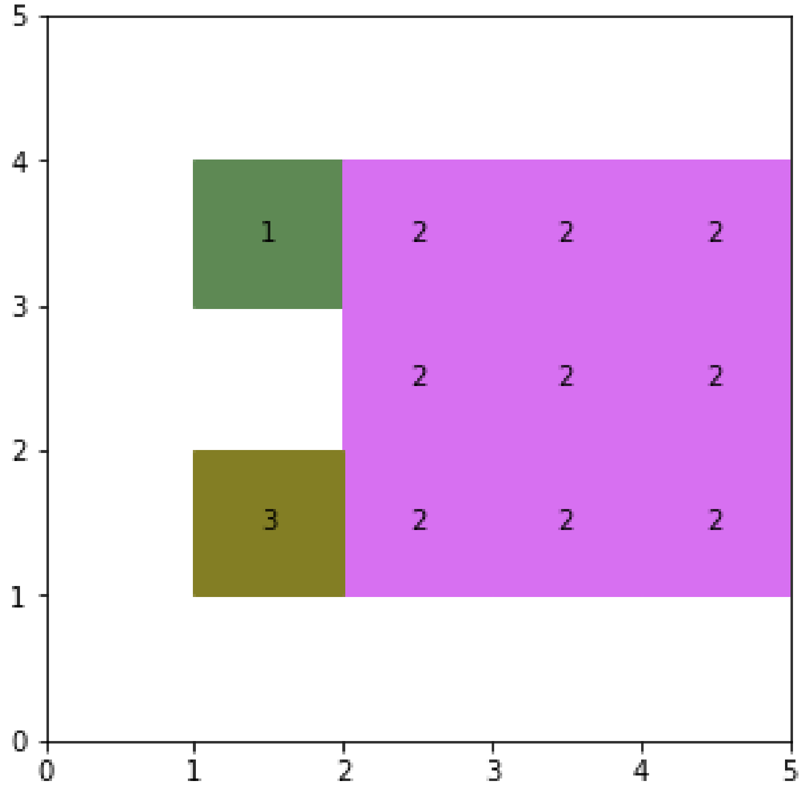
1
0
1
2
3
4
5
0
1
2
3
4
5


1
2
2
2
2
2
2
2
2
2
0
1
2
3
4
5

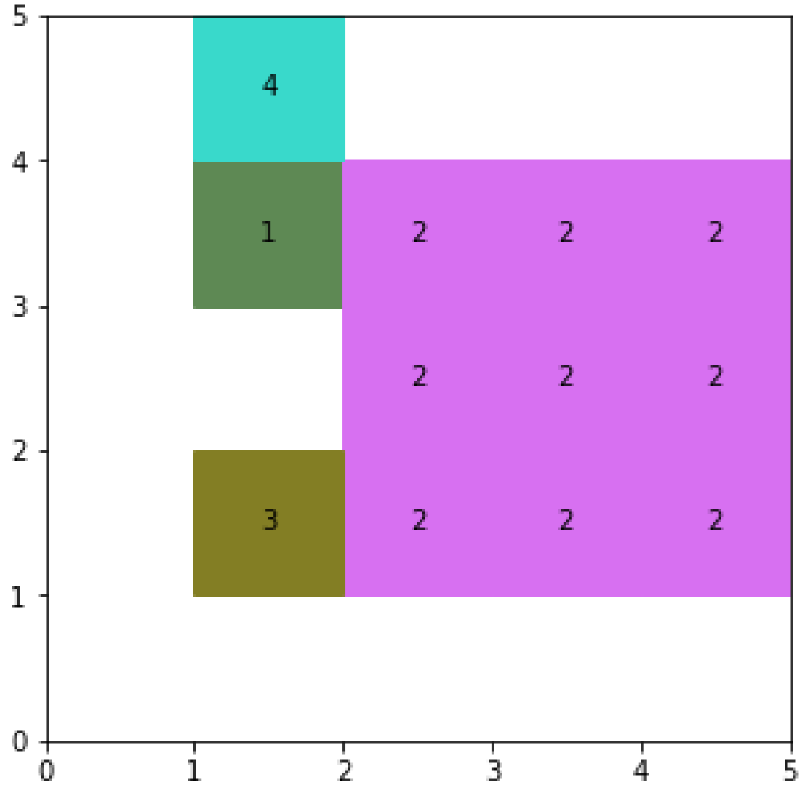


1
2
2
2
2
2
2
2
2
2
0
1
2
3
4
5

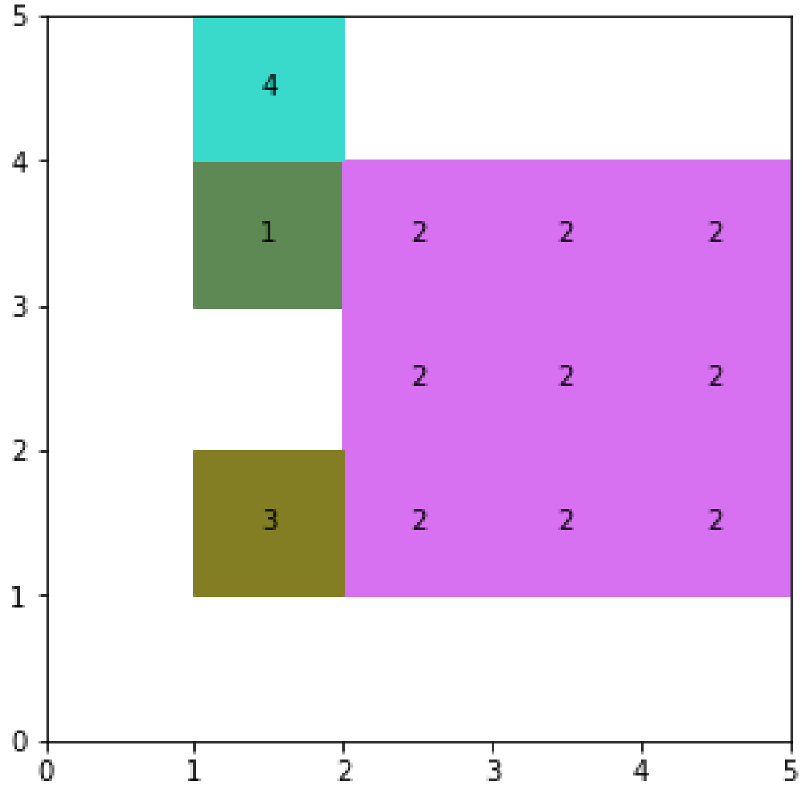

1
2
2
2
2
2
2
2
2
2
0
1
2
3
4
5

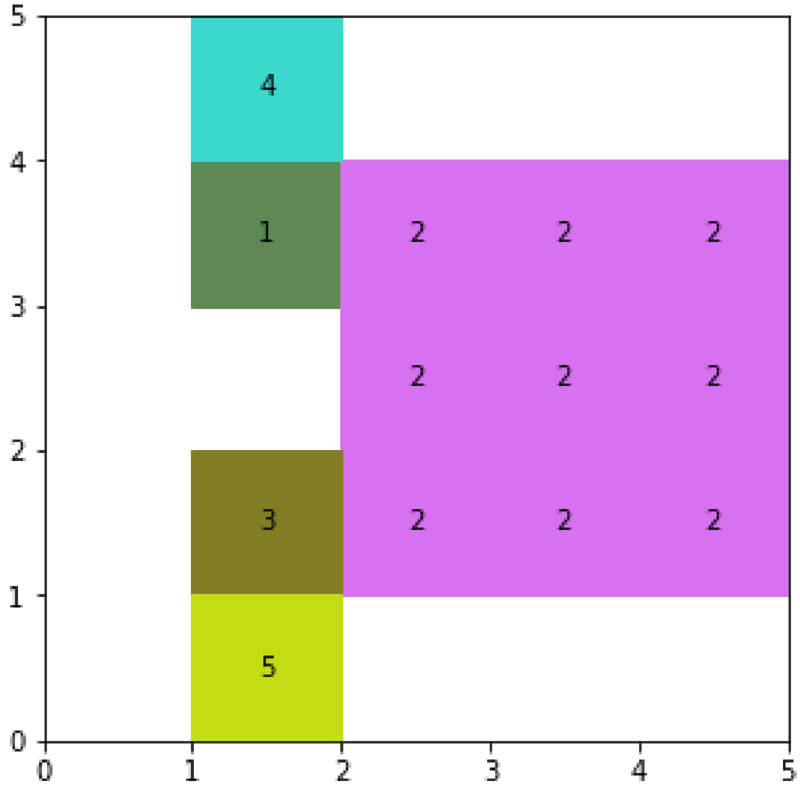


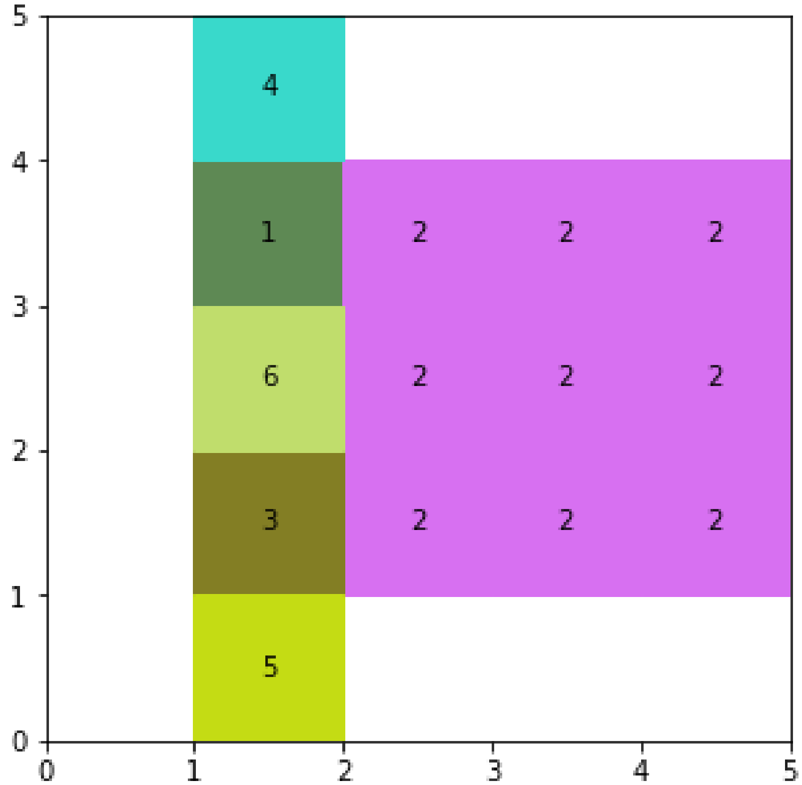
5
4
3
2
1
0
0
1
2
3
4
5
1
2
2
2
2
2
2
3
2
2
2



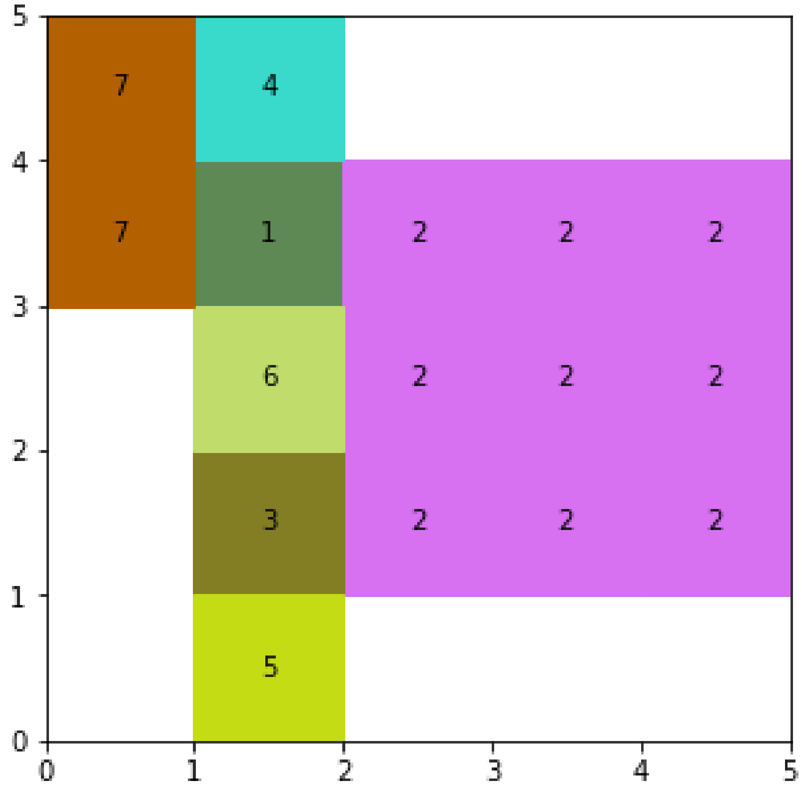
4
1
2
2
2
2
2
2
3
2
2
2
0
1
2
3
4
5

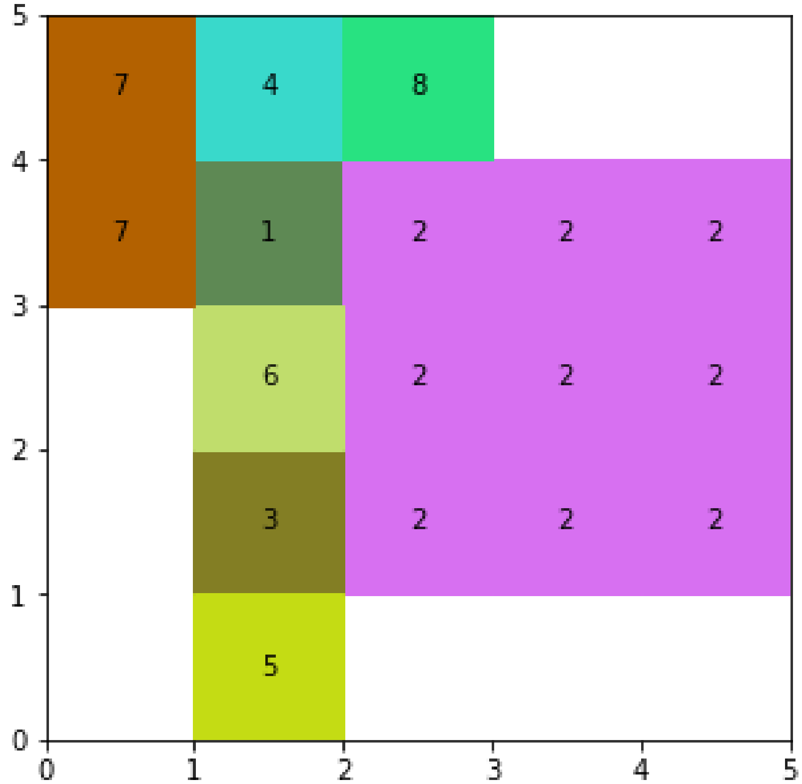

4
1
2
2
2
2
2
2
3
2
2
2
0
1
2
3
4
5

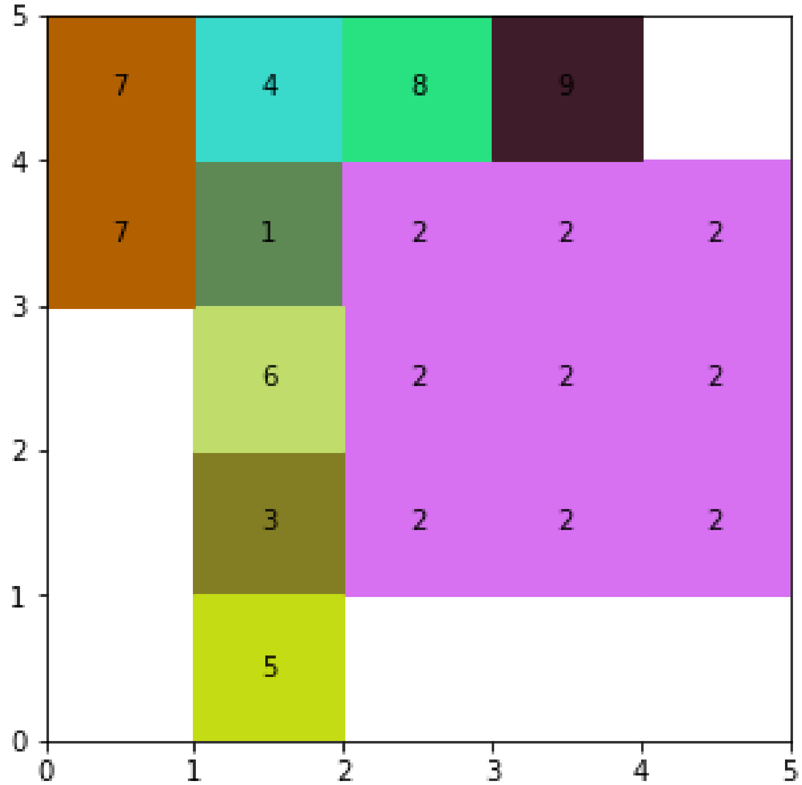

4
1
2
2
2
2
2
2
3
2
2
2
5
0
1
2
3
4
5
0
1
2
3
4
5

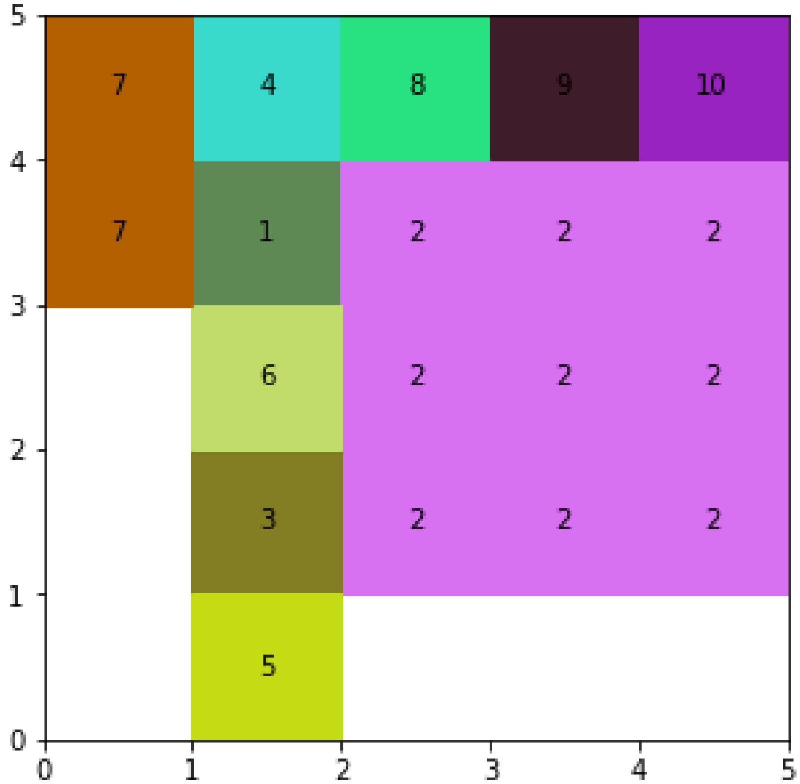


4
1
2
2
2
6
2
2
2
3
2
2
2
5
0
1
2
3
4
5
0
1
2
3
4
5

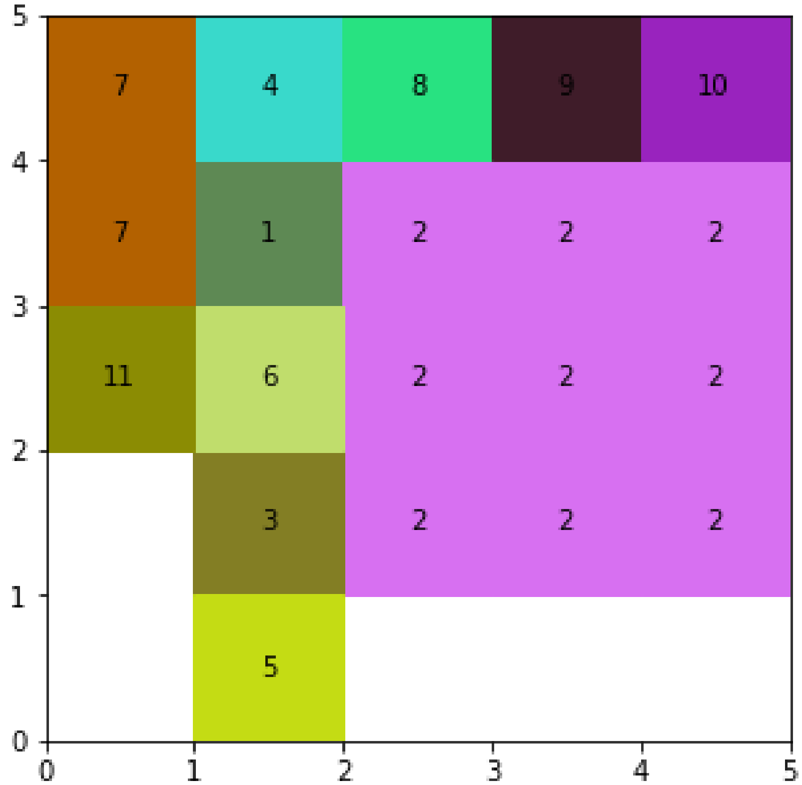

7
4
7
1
2
2
2
6
2
2
2
3
2
2
2
5
0
1
2
3
4
5

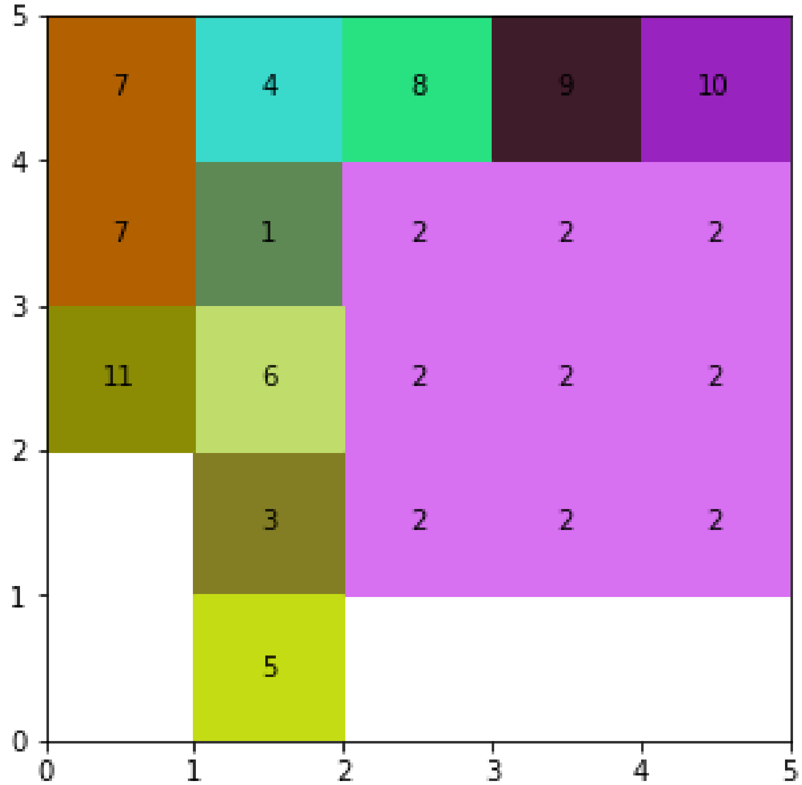

7
4
8
7
1
2
2
2
6
2
2
2
3
2
2
2
5
0
1
2
3
4
5
0
1
2
3
4
5


7
4
8
9
7
1
2
2
2
6
2
2
2
3
2
2
2
5
0
1
2
3
4
5
0
1
2
3
4
5

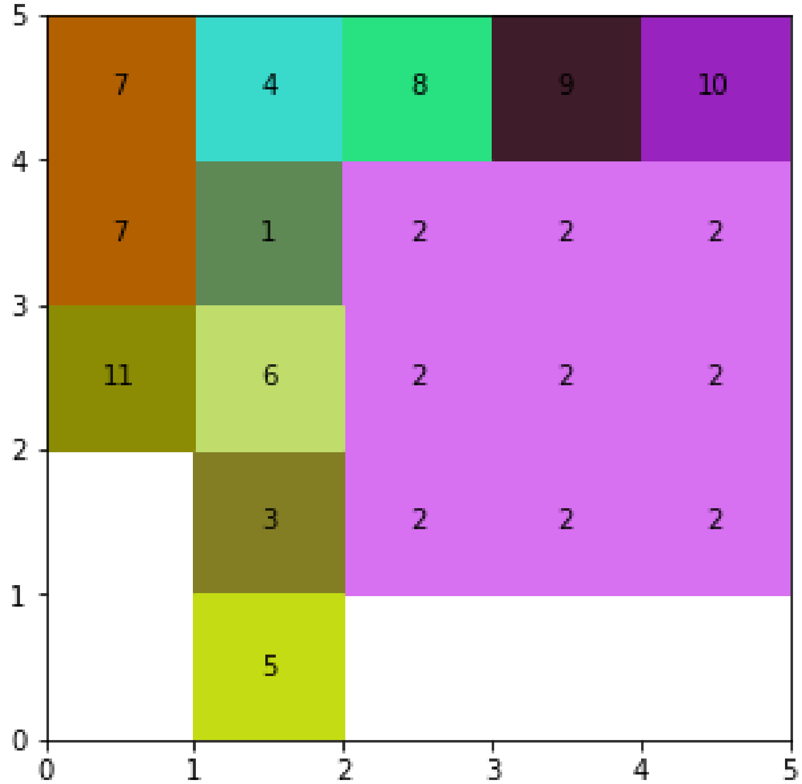


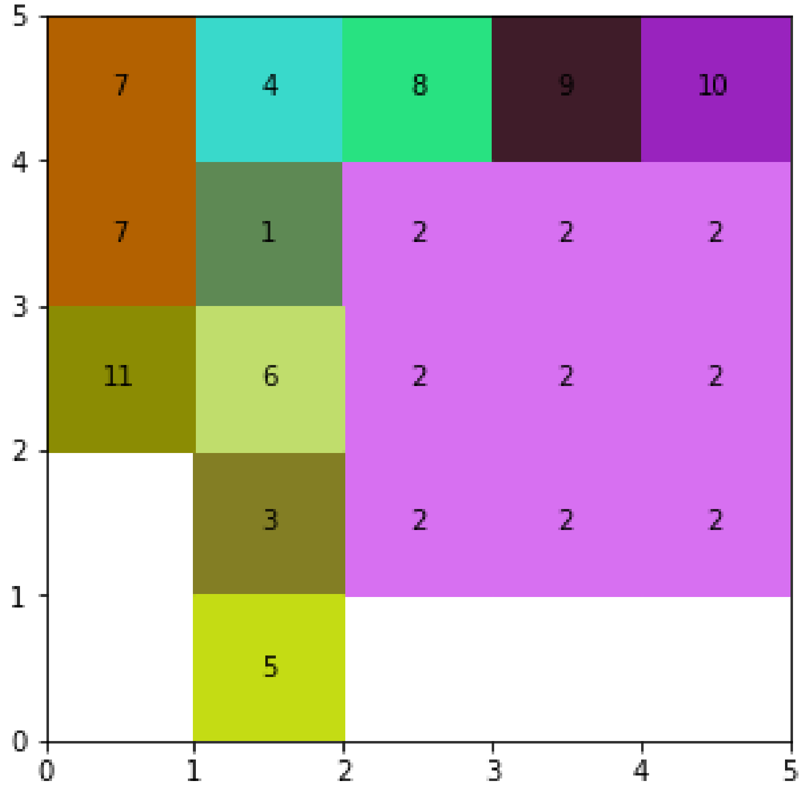
7
4
8
9
10
7
1
2
2
2
6
2
2
2
3
2
2
2
5
0
1
2
3
4
5

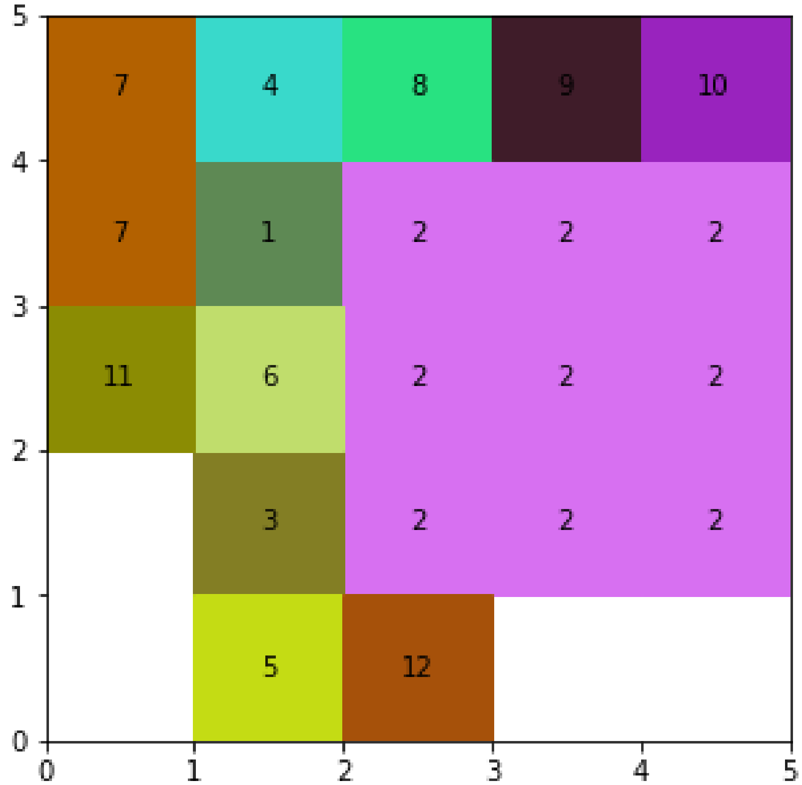

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
3
2
2
2
5
0
1
2
3
4
5

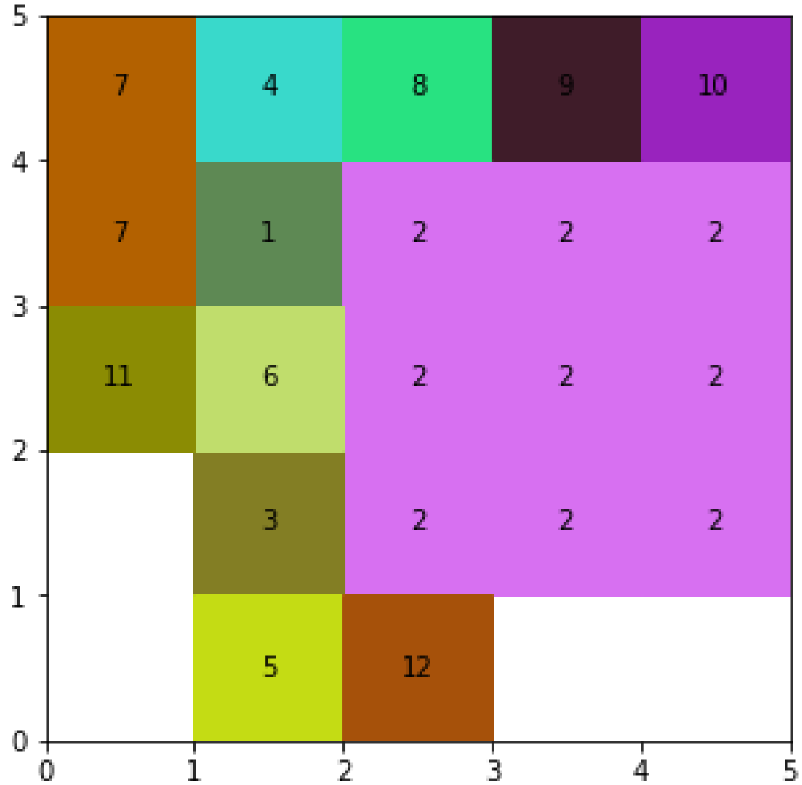


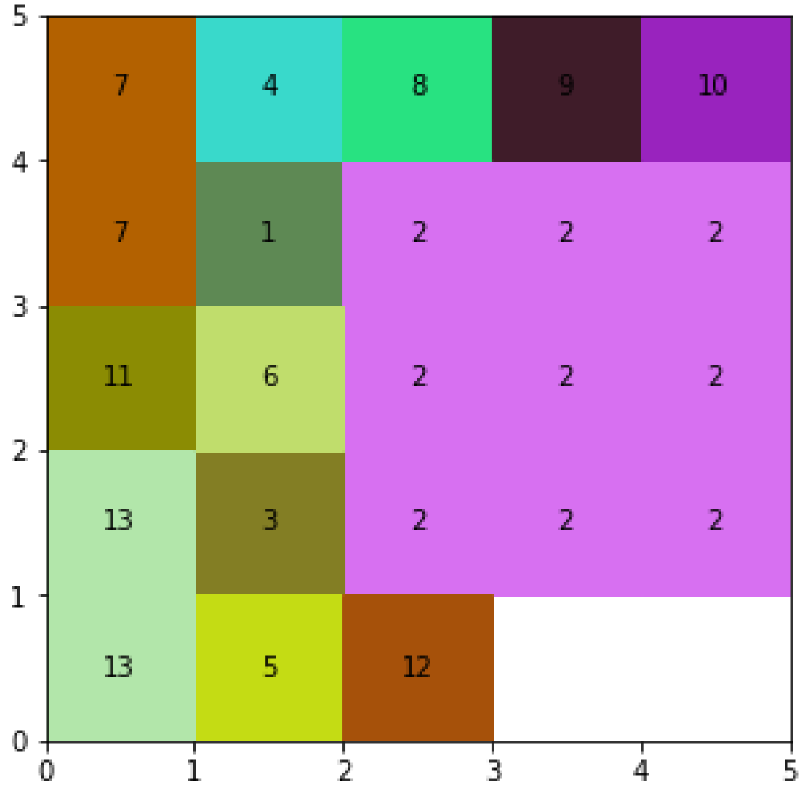
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
3
2
2
2
5
0
1
2
3
4
5

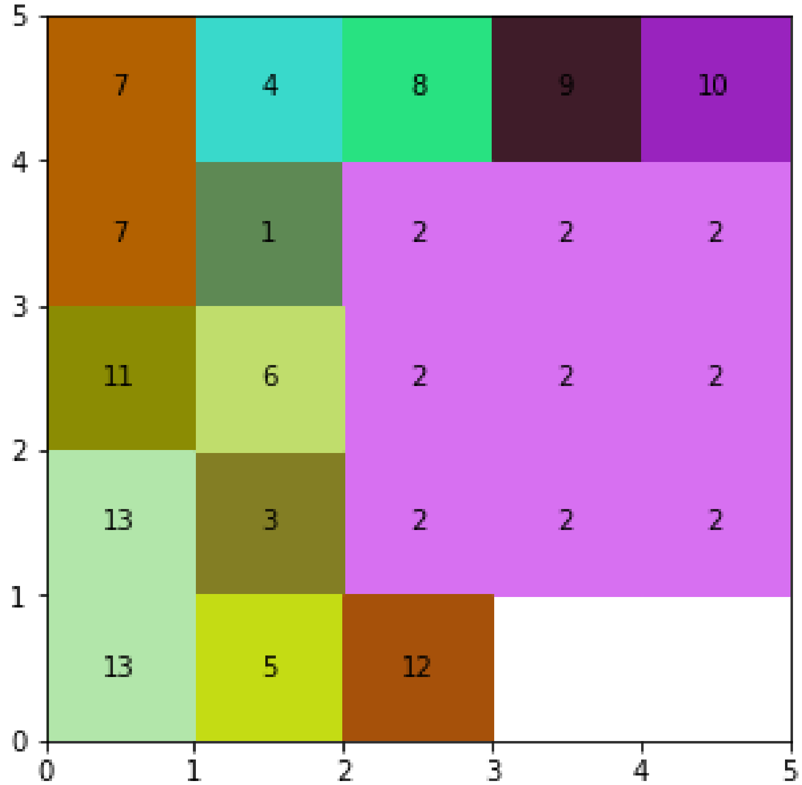

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
3
2
2
2
5
0
1
2
3
4
5

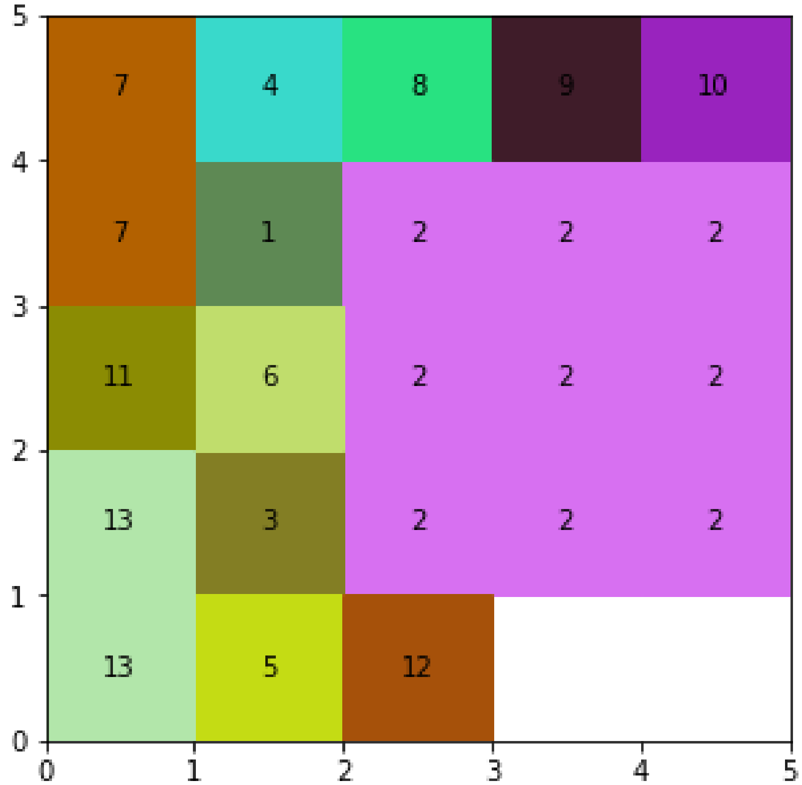

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
3
2
2
2
5
0
1
2
3
4
5


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
3
2
2
2
5
12
0
1
2
3
4
5

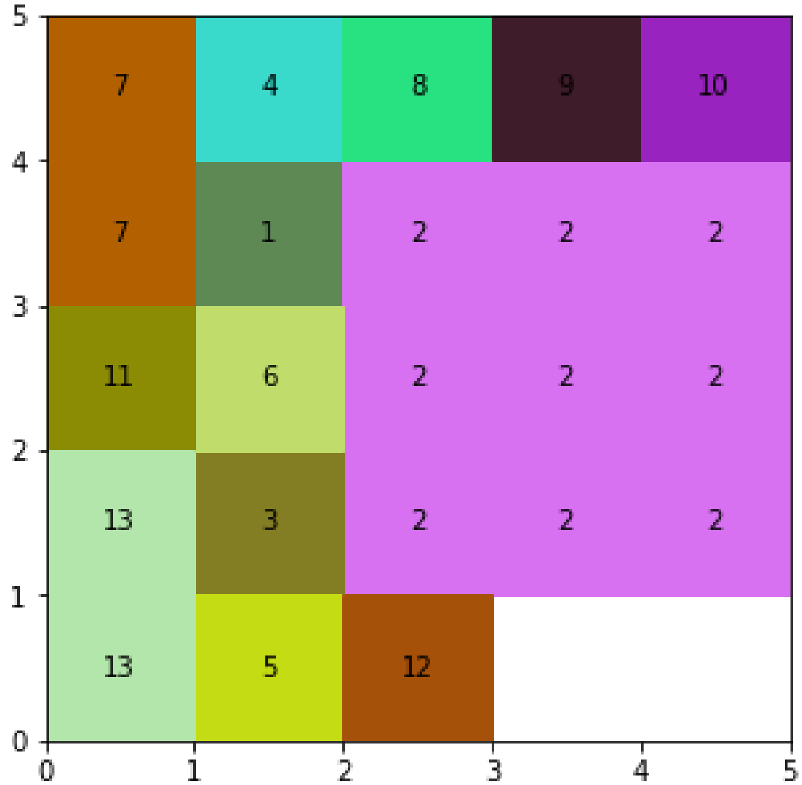

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
3
2
2
2
5
12
0
1
2
3
4
5

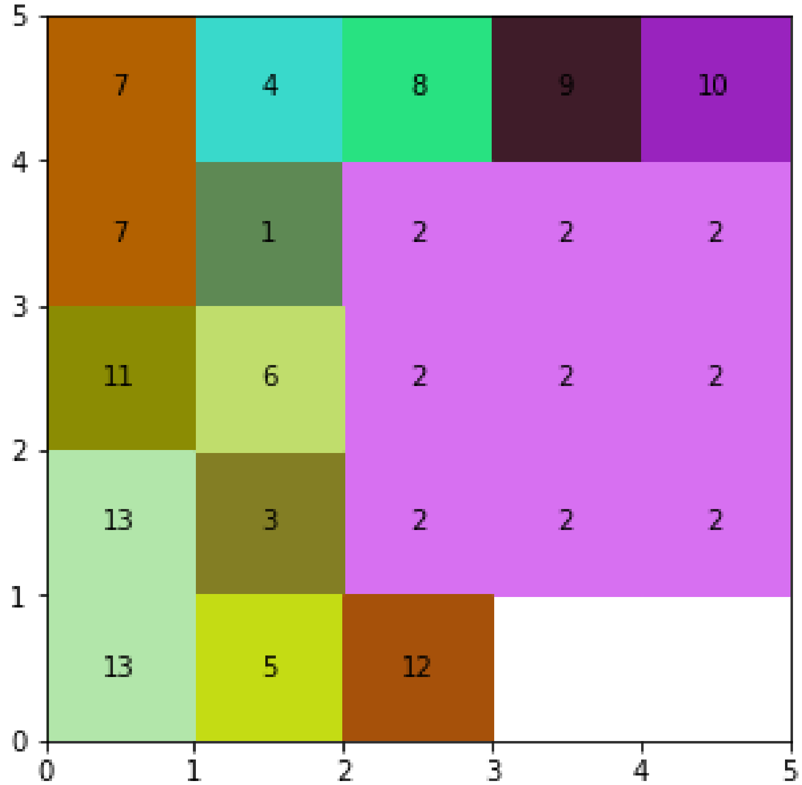

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5
0
1
2
3
4
5


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5

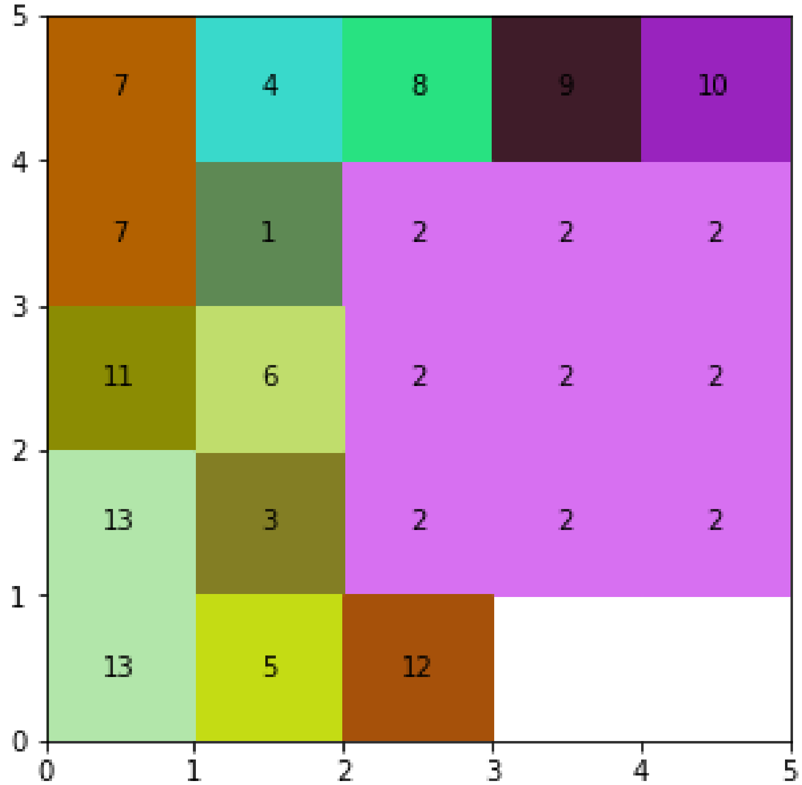


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5

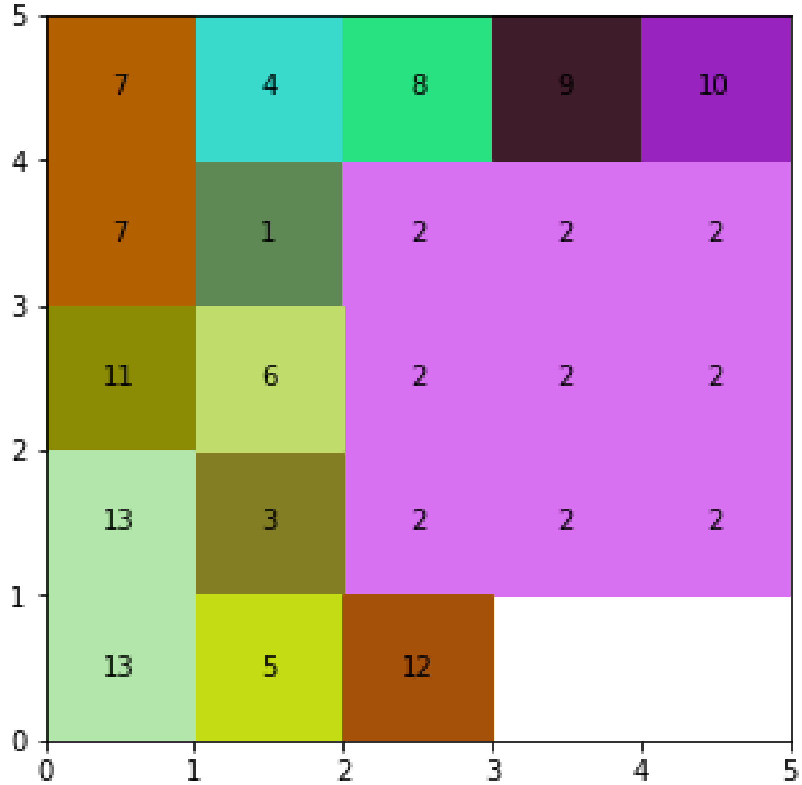

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12



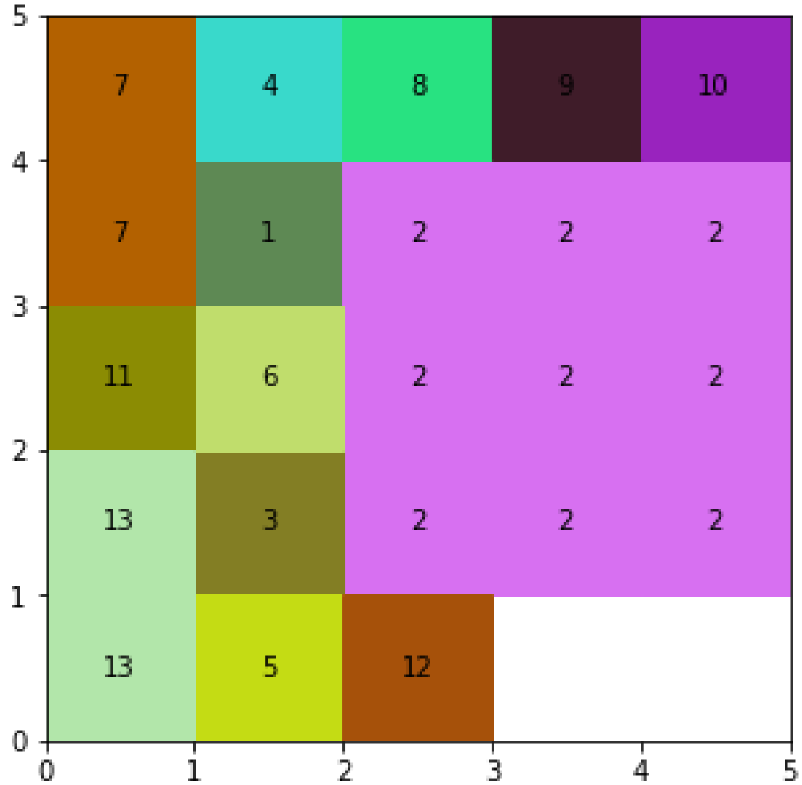
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5

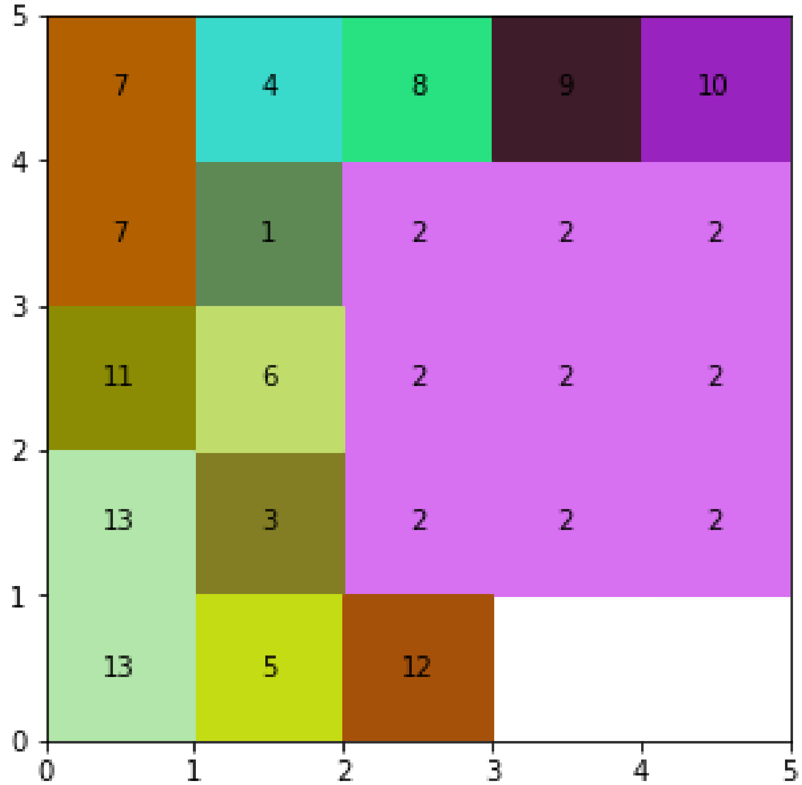


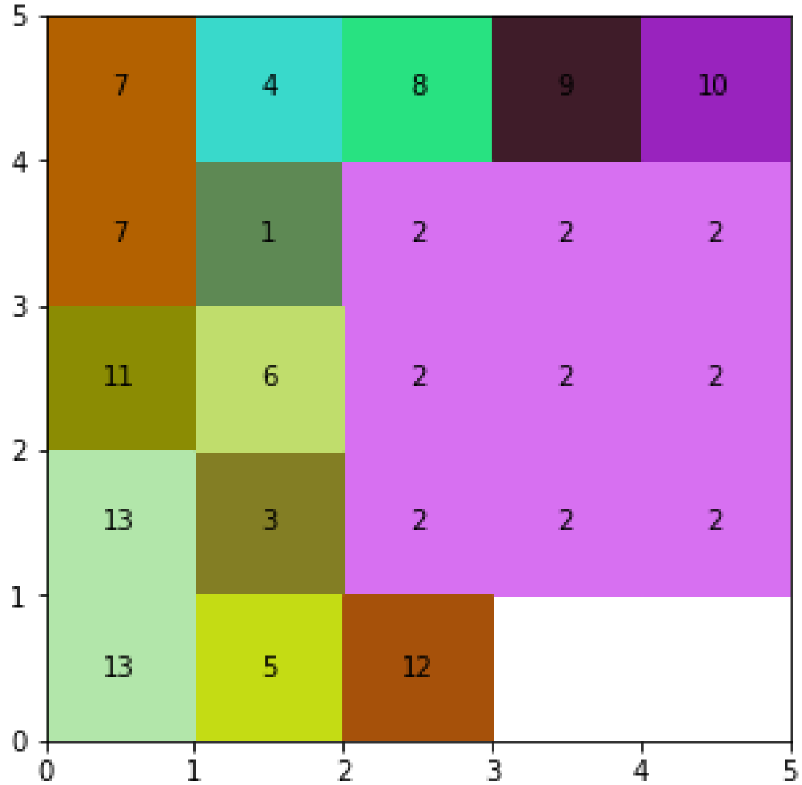
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5
0
1
2
3
4
5

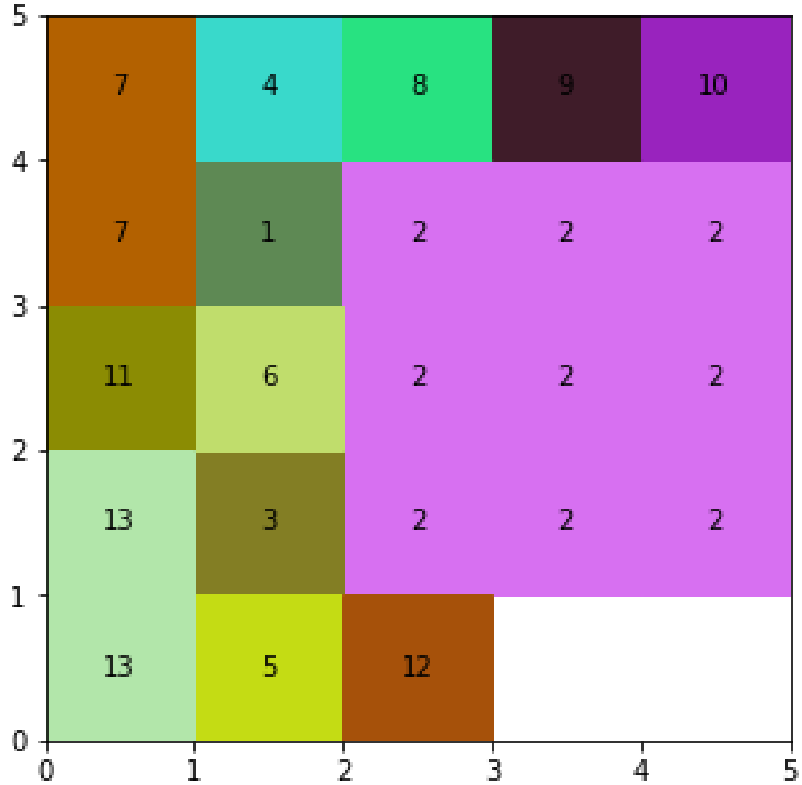


5
4
3
2
1
0
0
1
2
3
4
5
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12

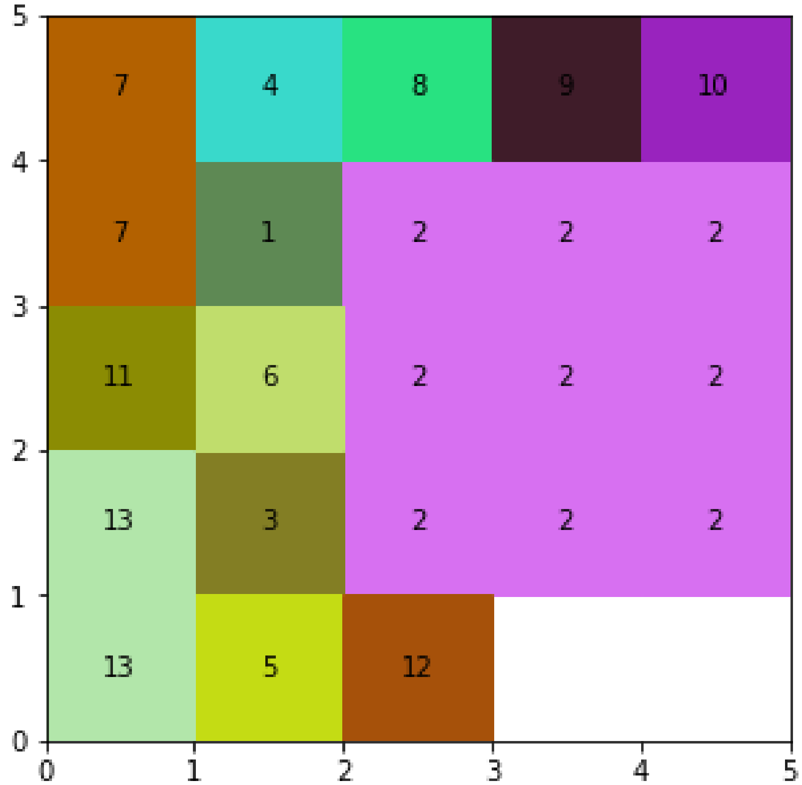

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5

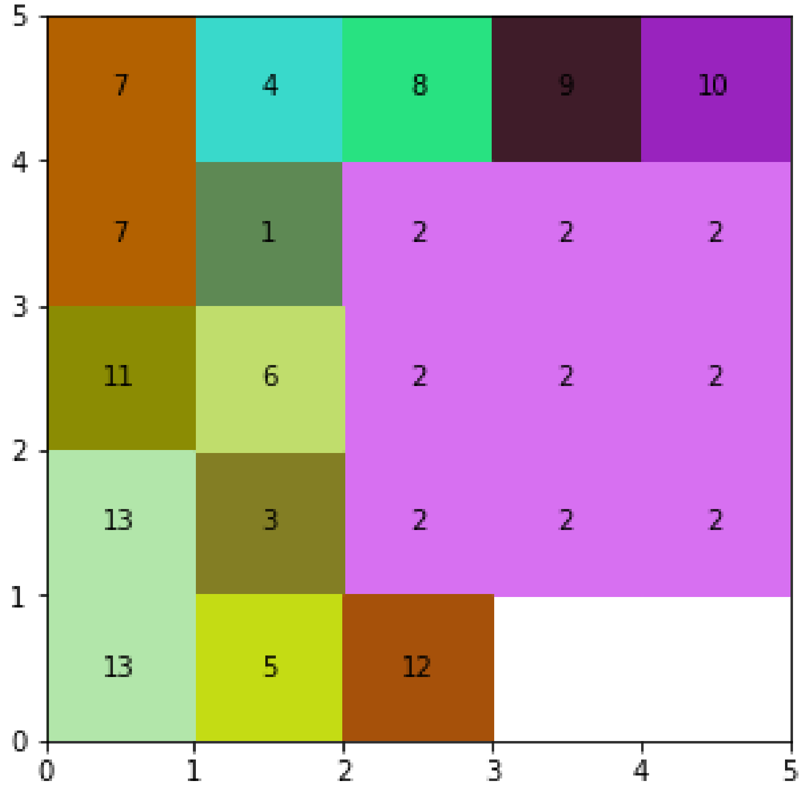

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5
0
1
2
3
4
5



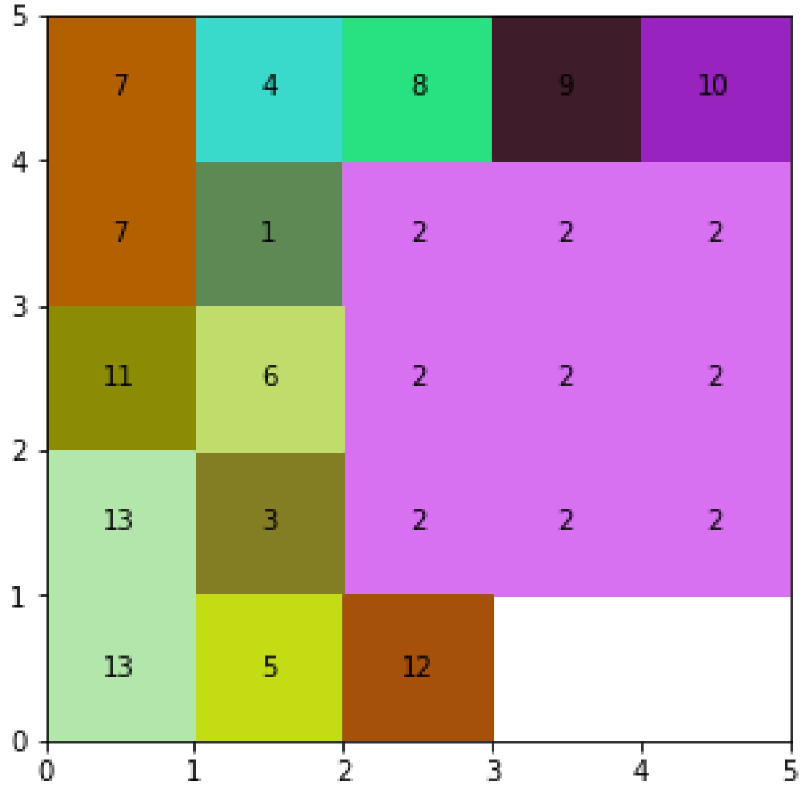
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5



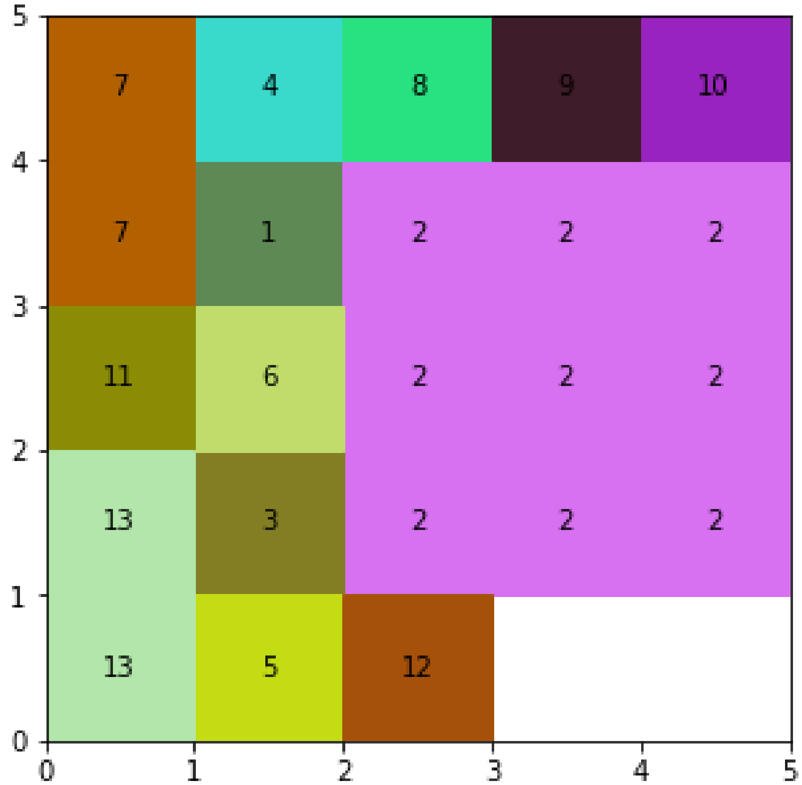
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5
0
1
2
3
4
5



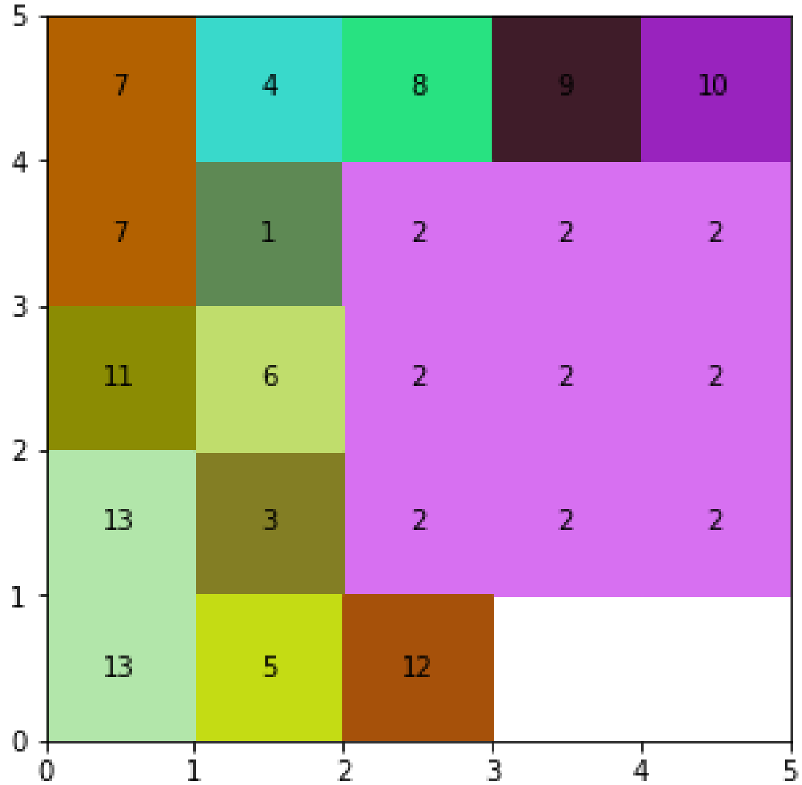
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5

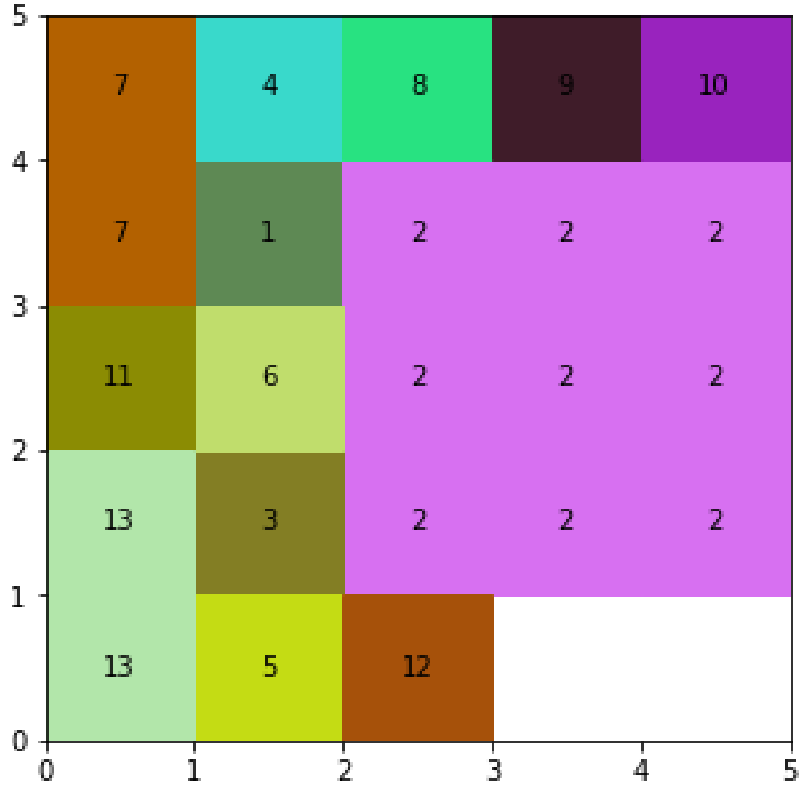


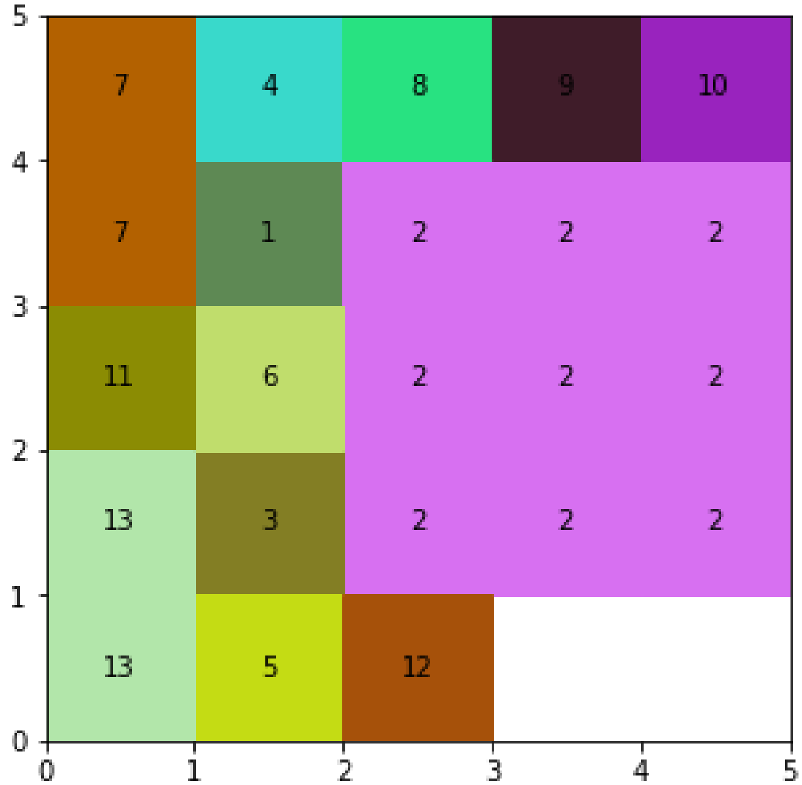
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5
0
1
2
3
4
5



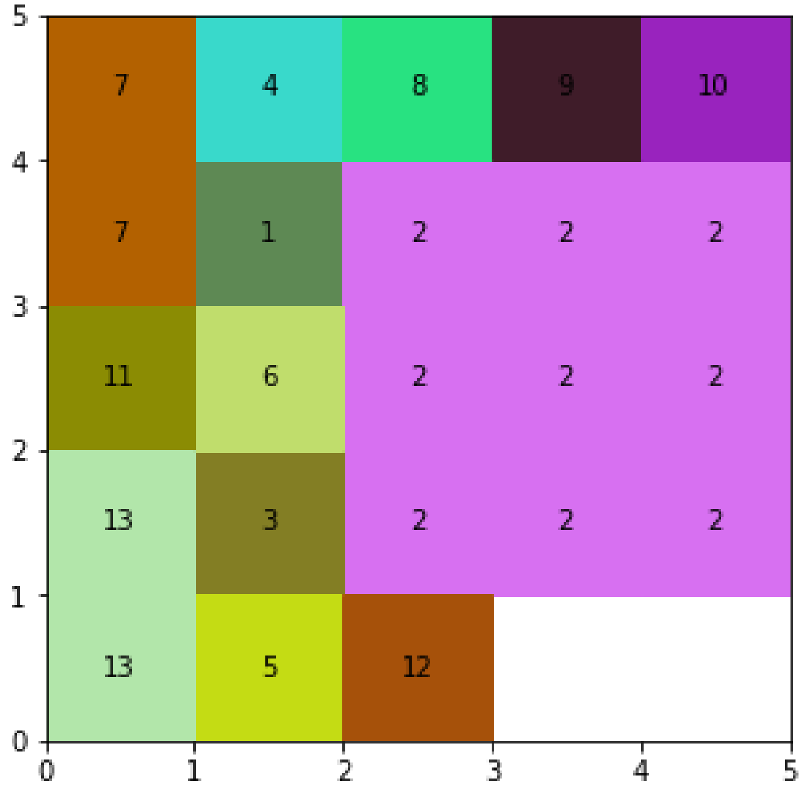
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5

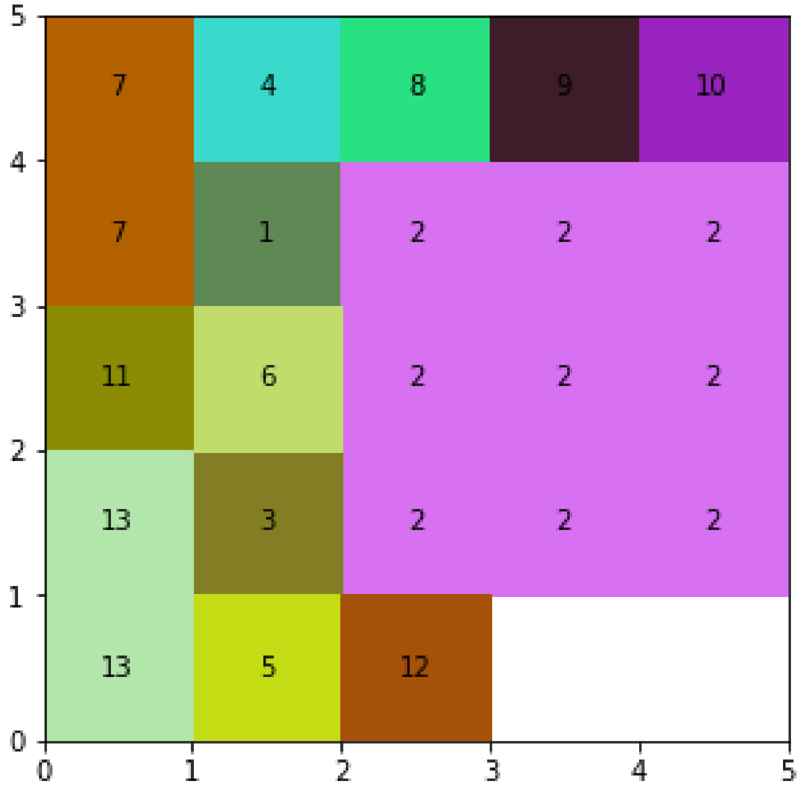

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5
0
1
2
3
4
5

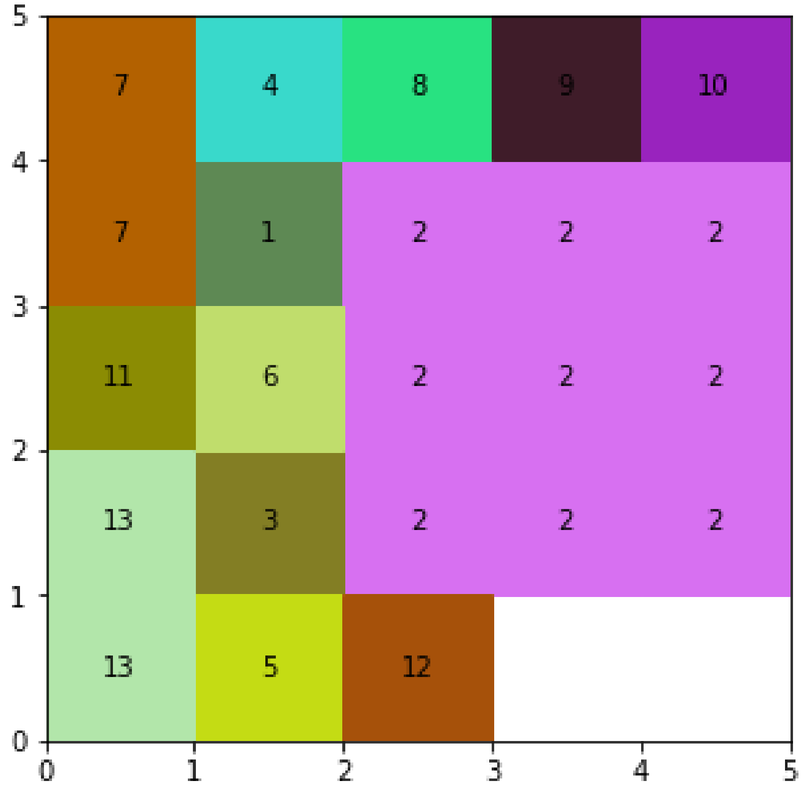


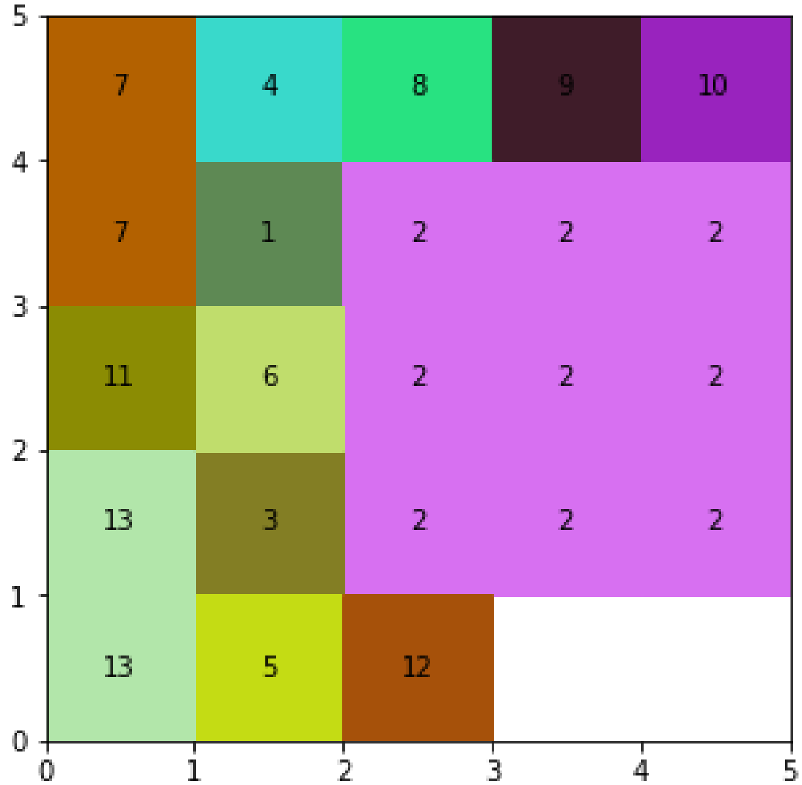
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5

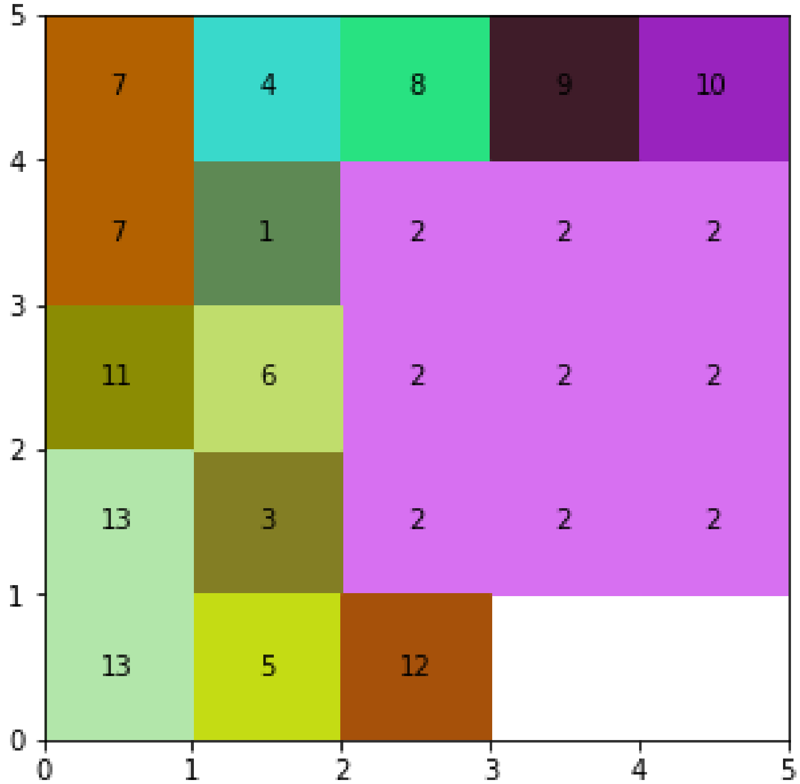


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5

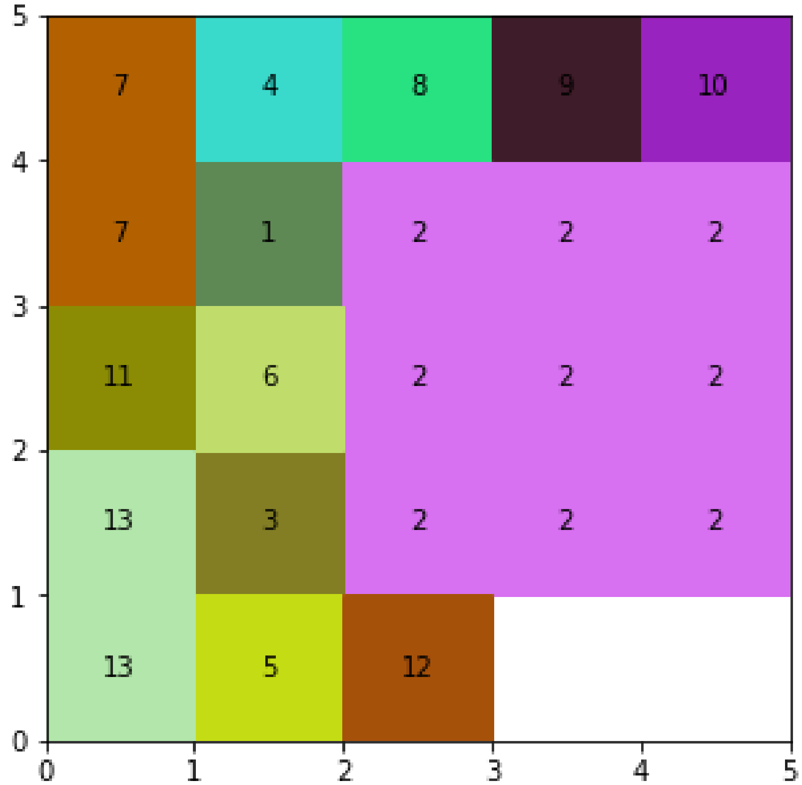

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5



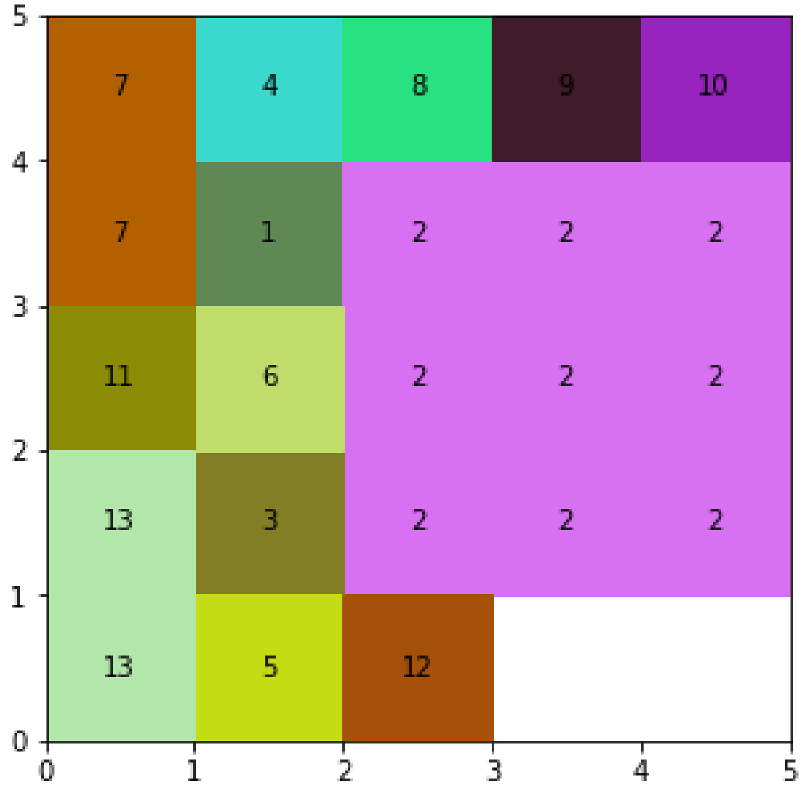
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5



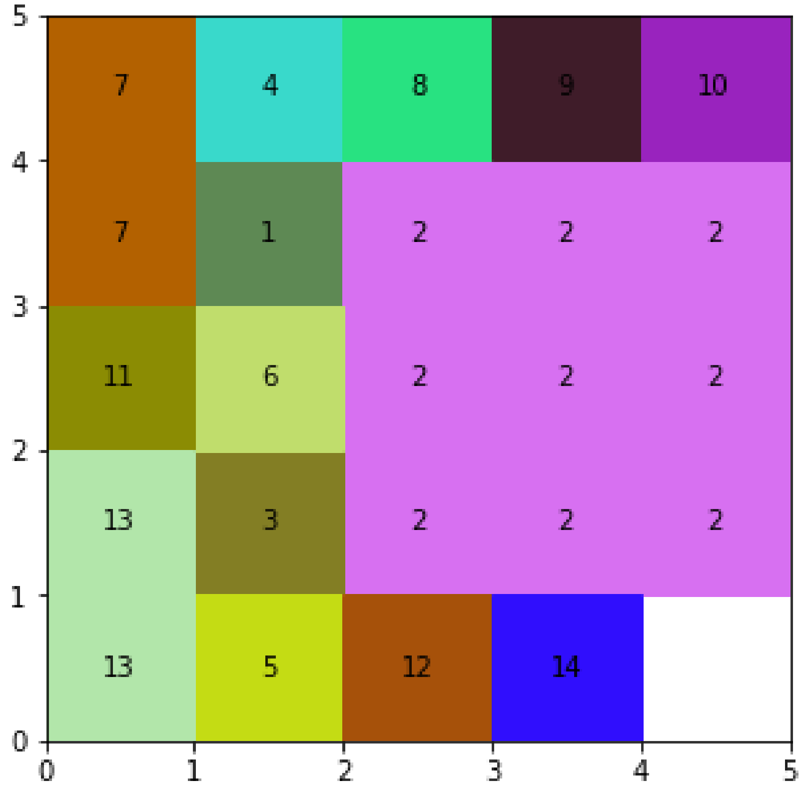
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5
0
1
2
3
4
5

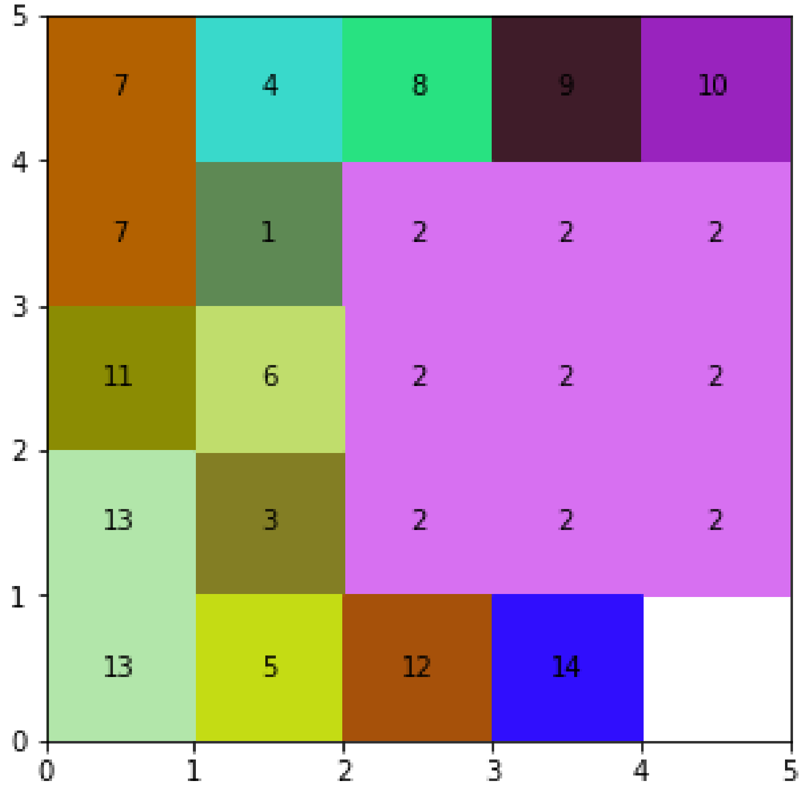


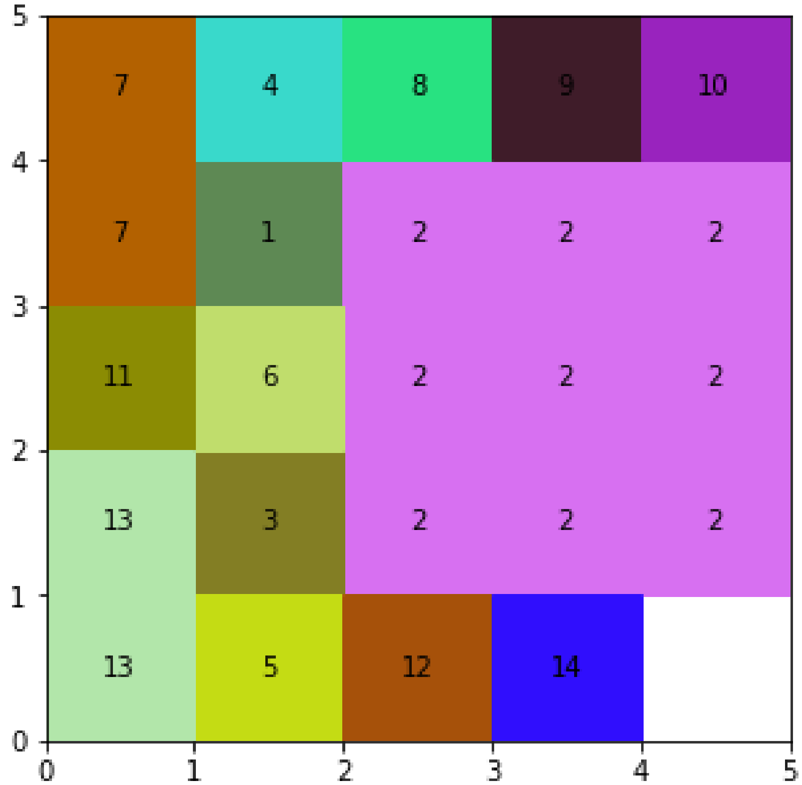
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5

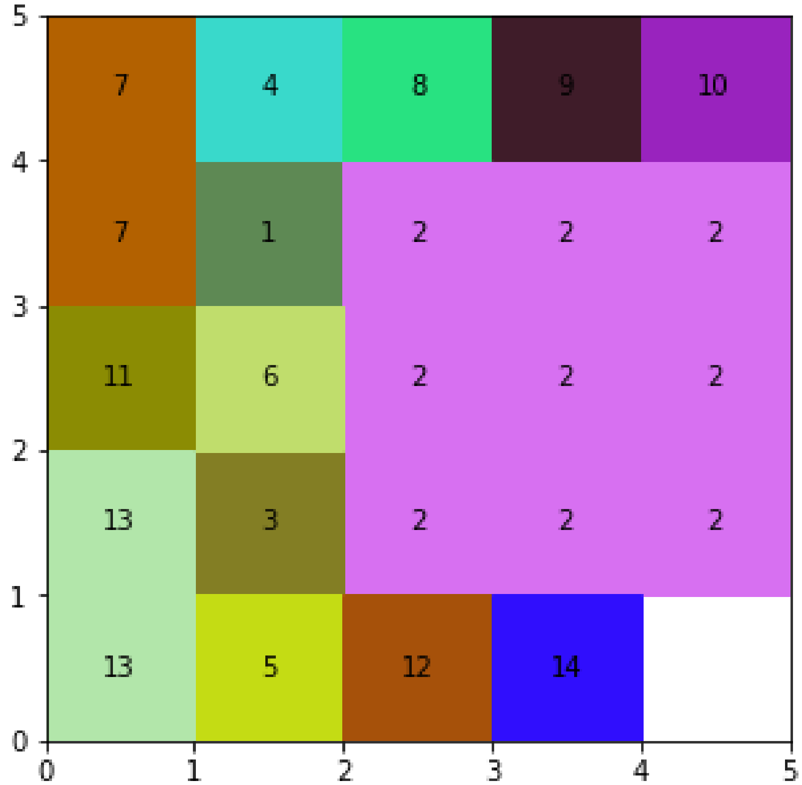


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
0
1
2
3
4
5

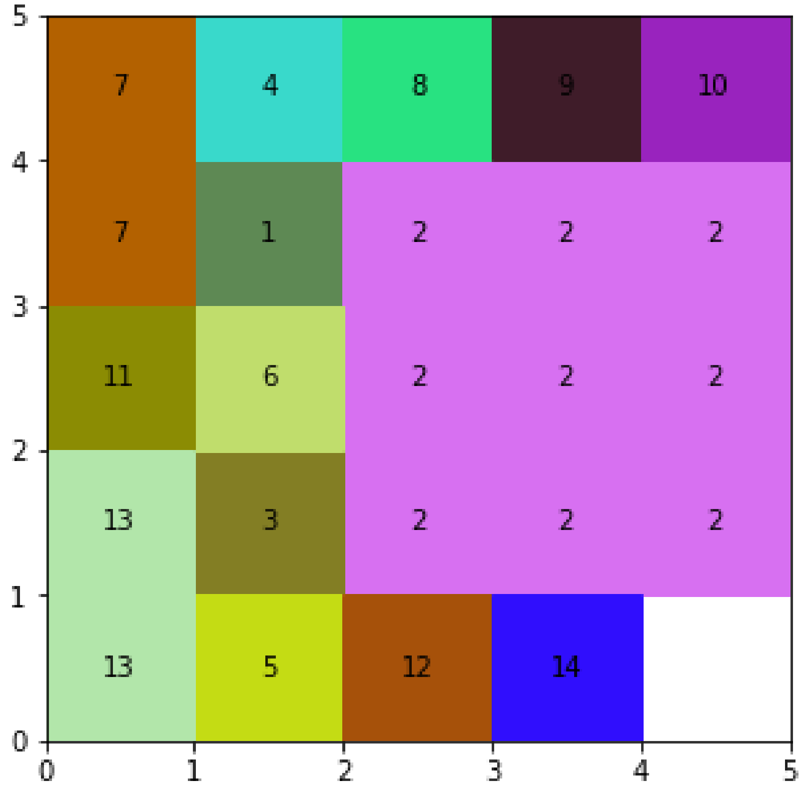

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5

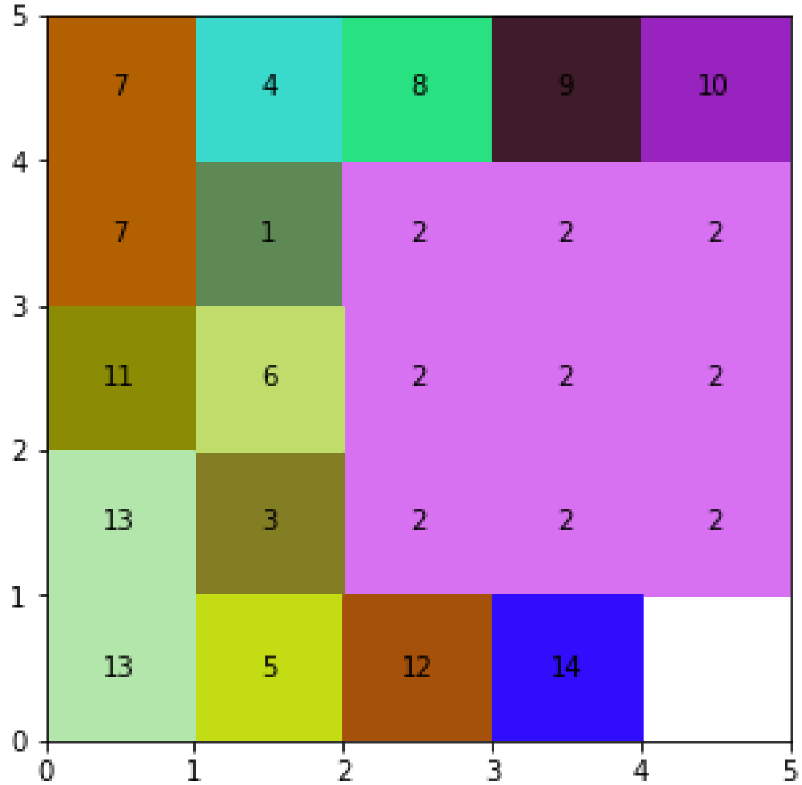

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5
0
1
2
3
4
5



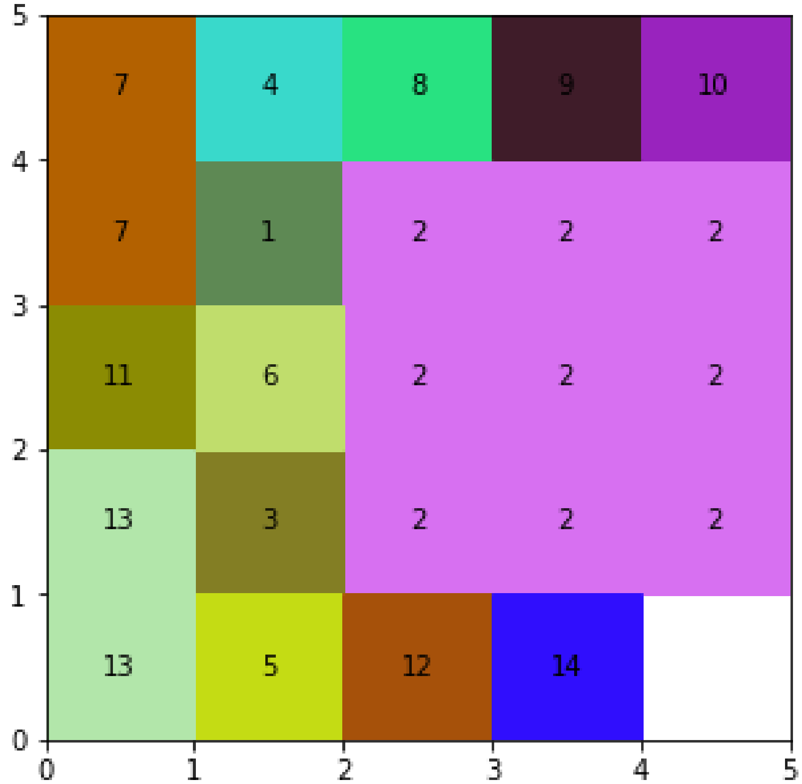
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5



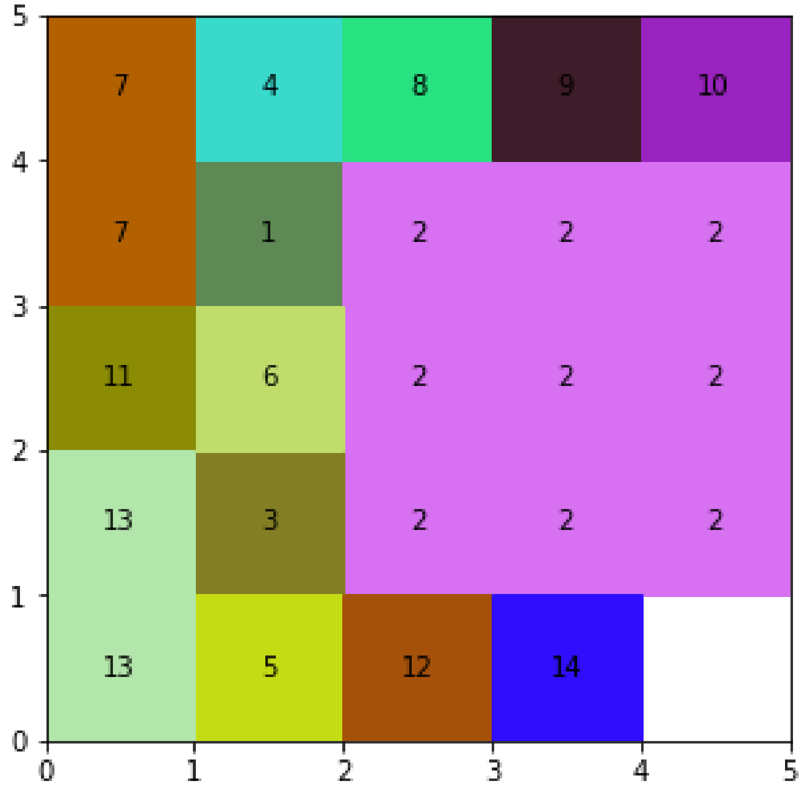
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5

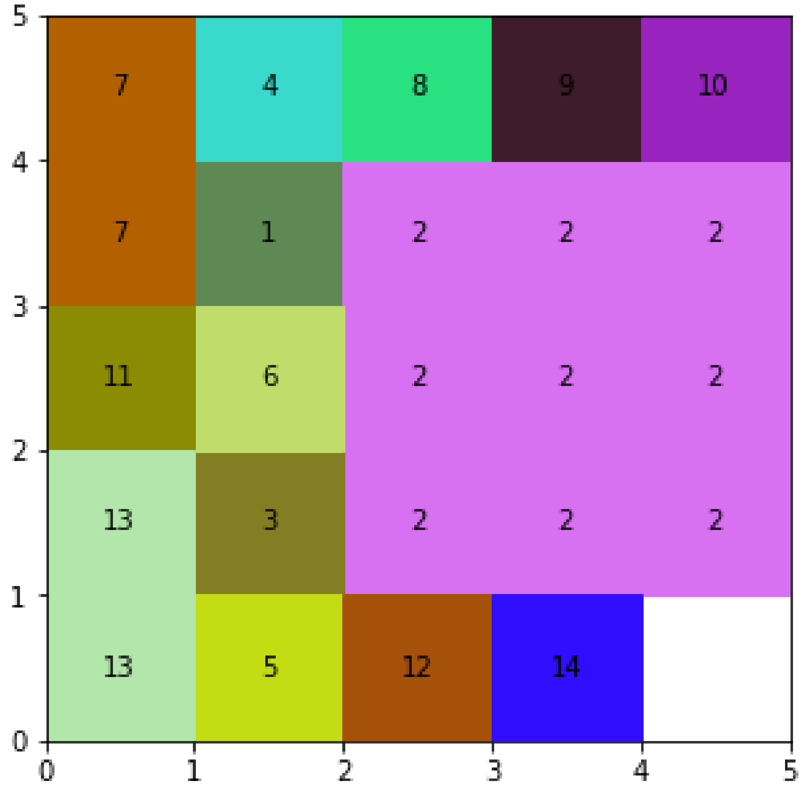


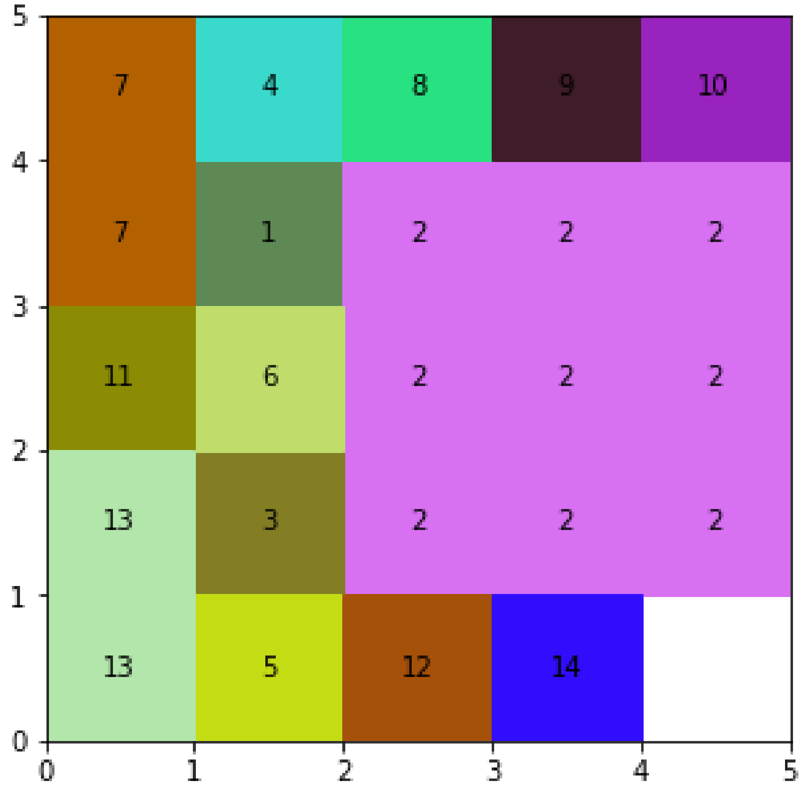
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5



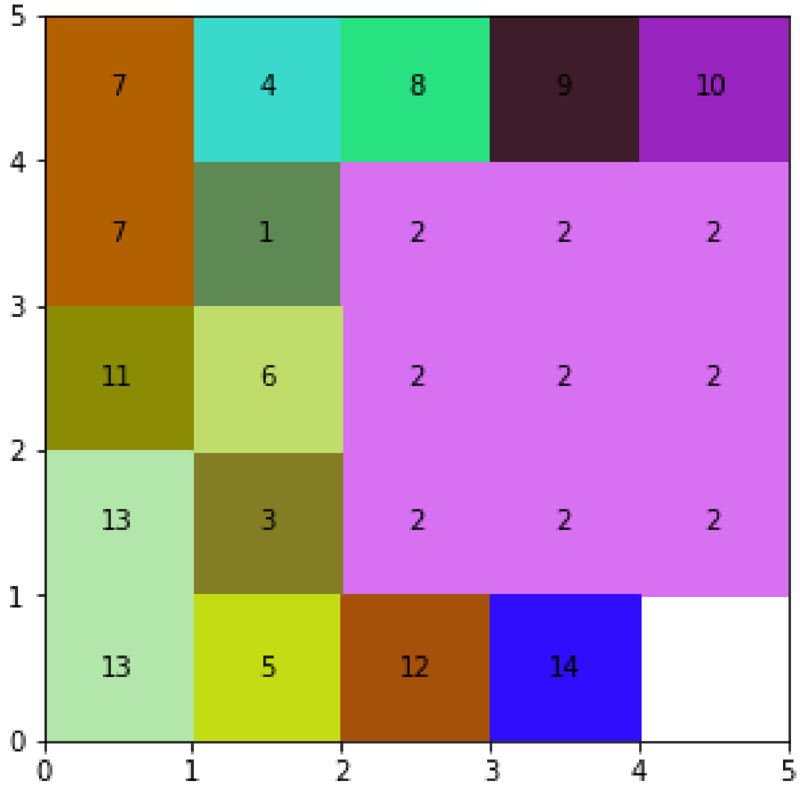
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5

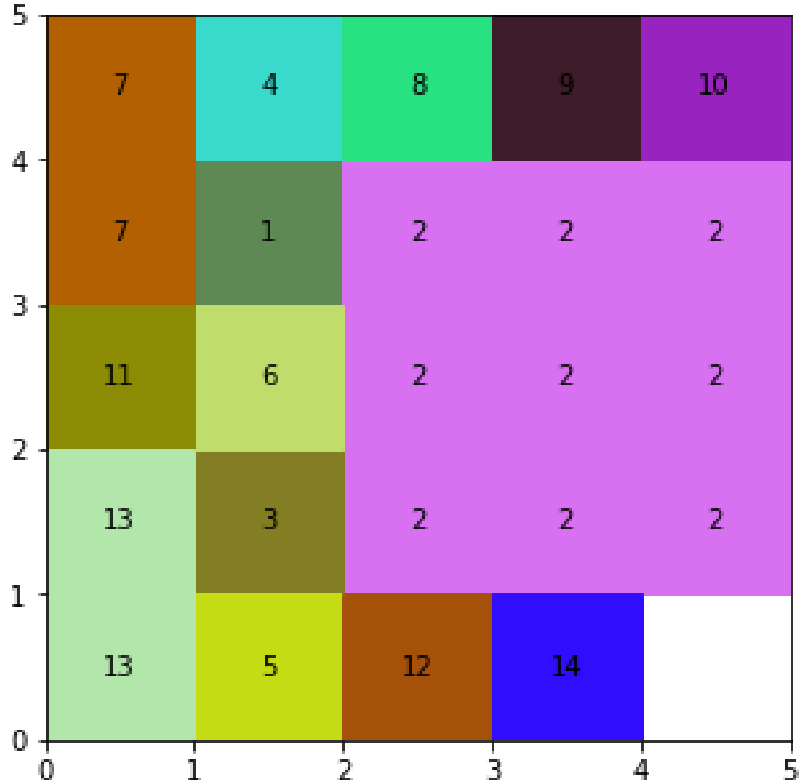

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5

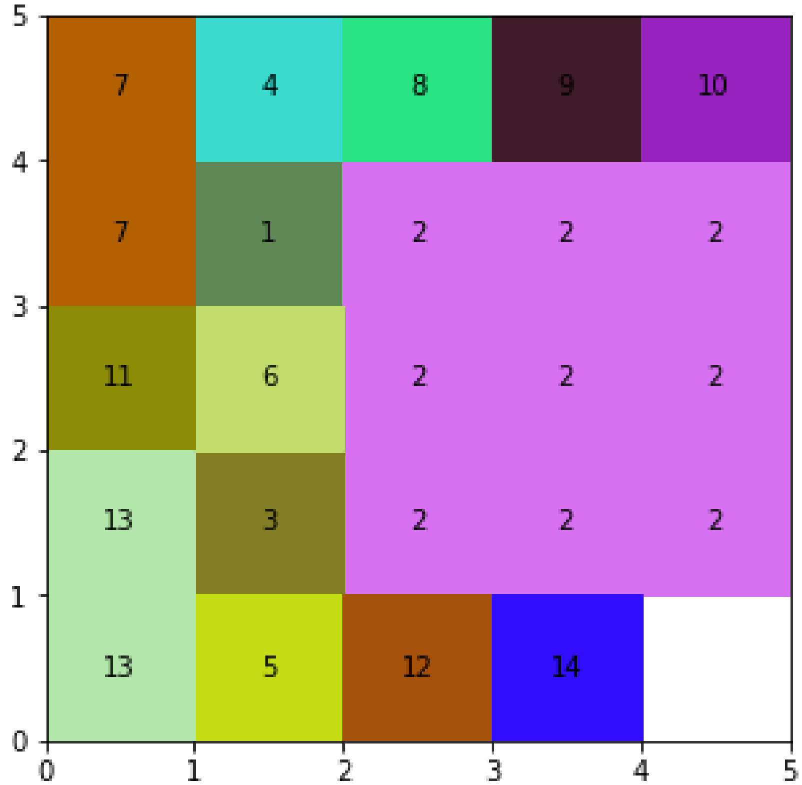


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5

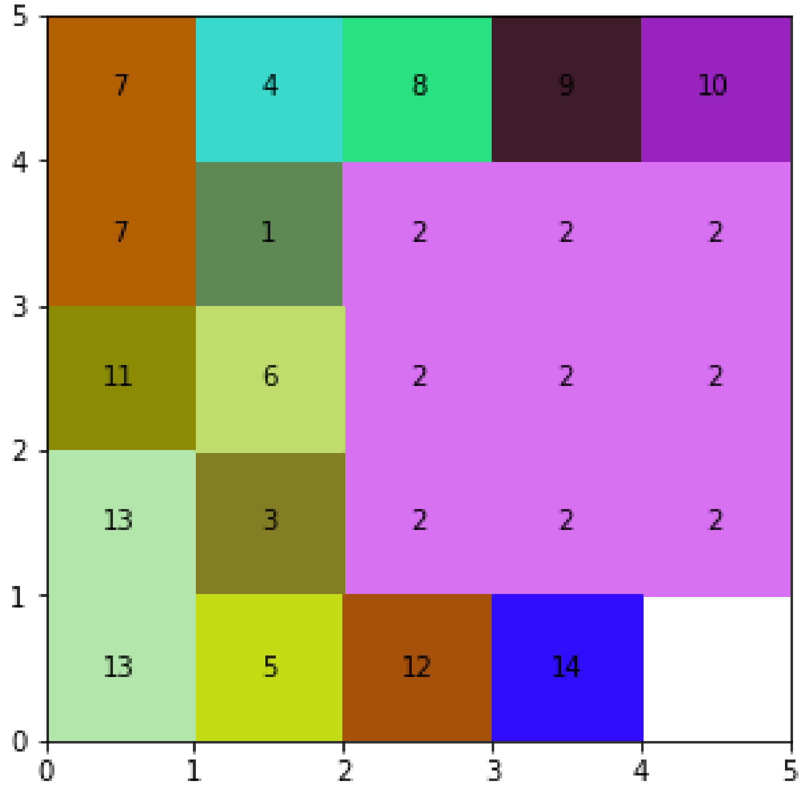


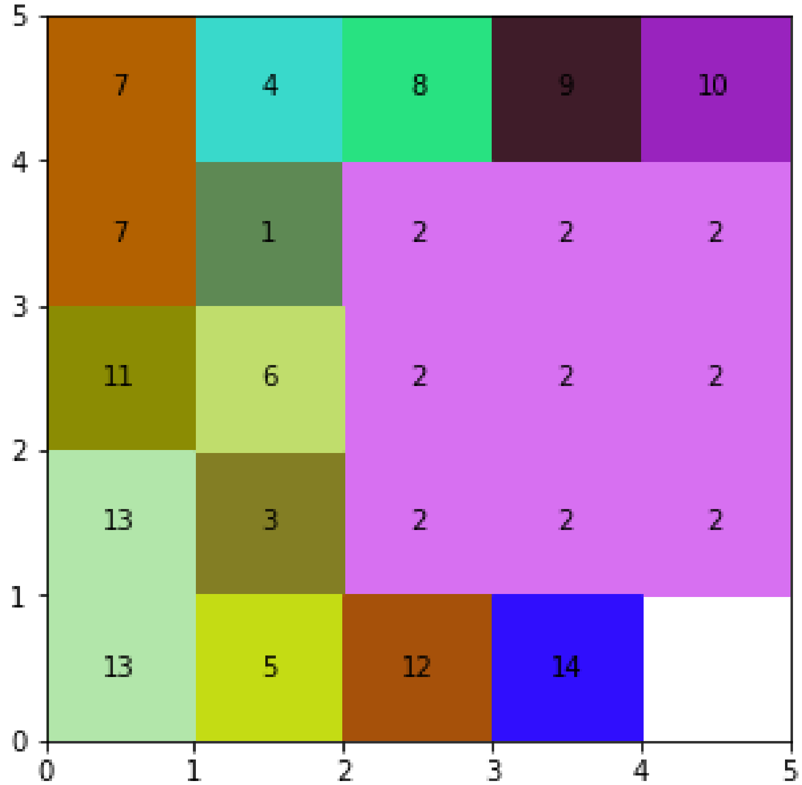
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5

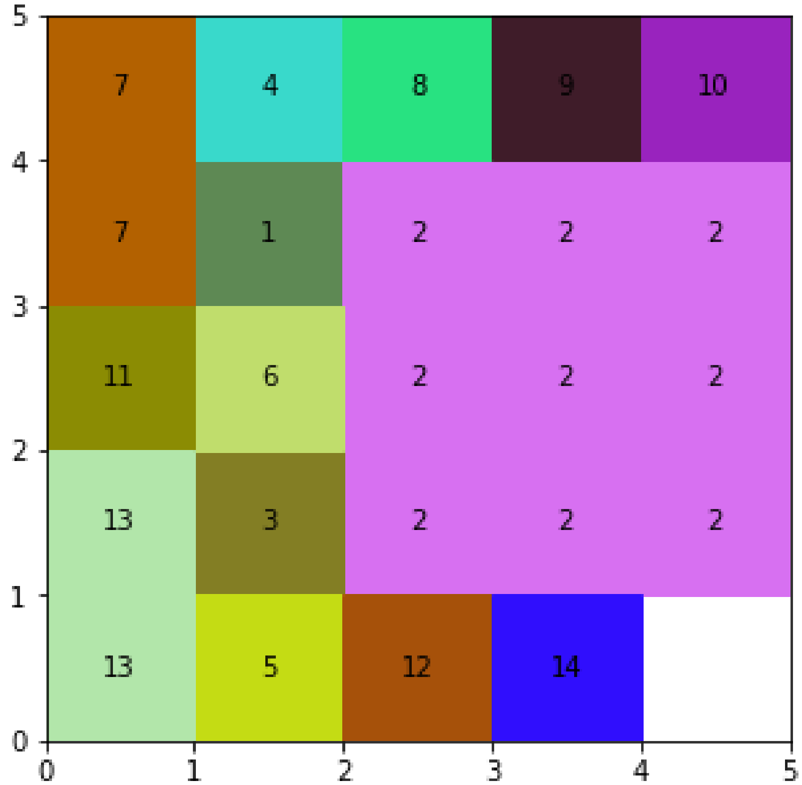

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5



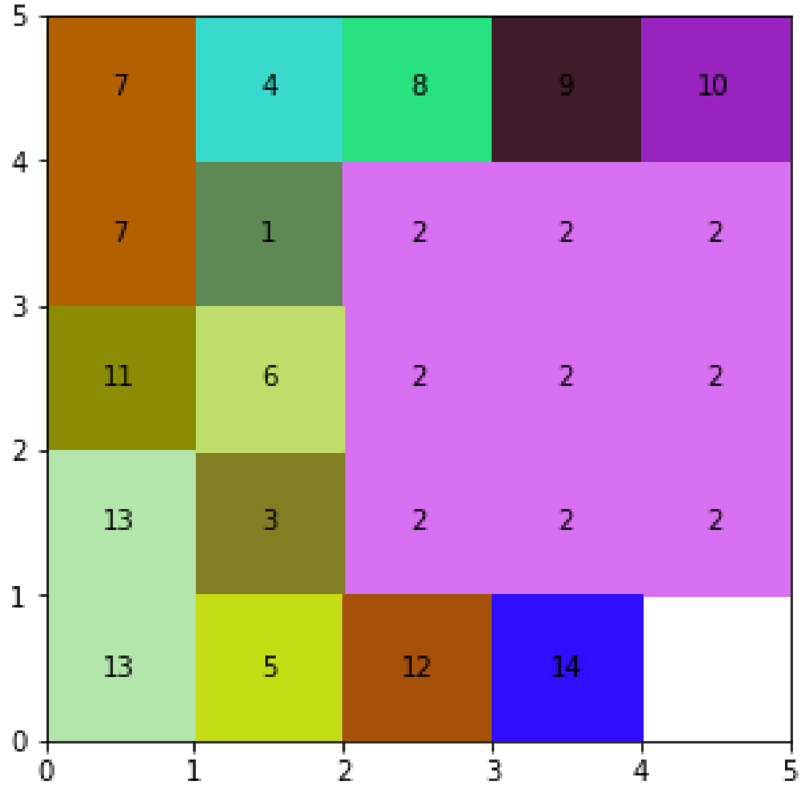
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5

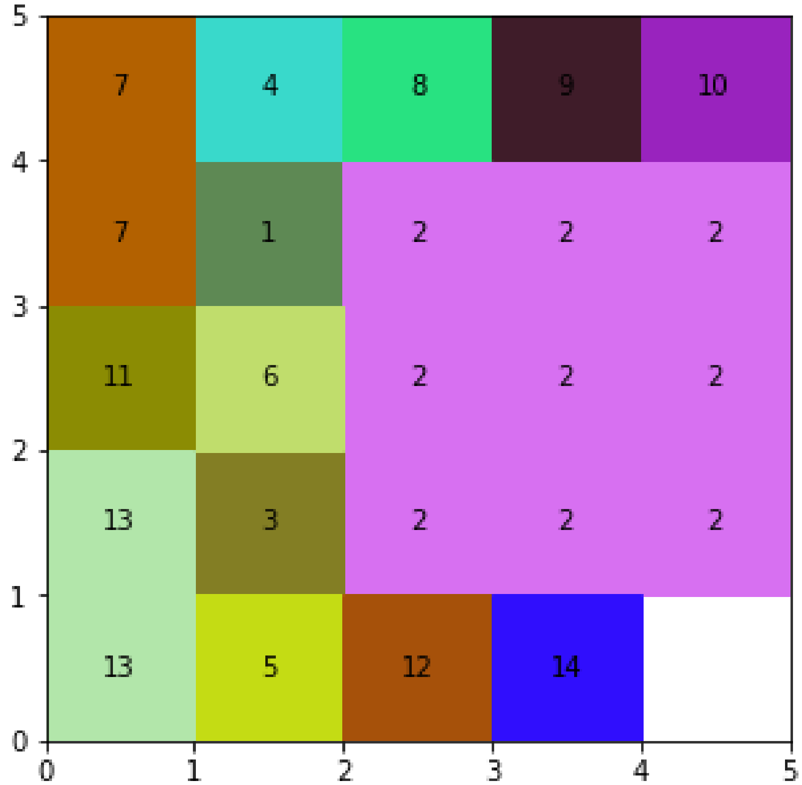

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5

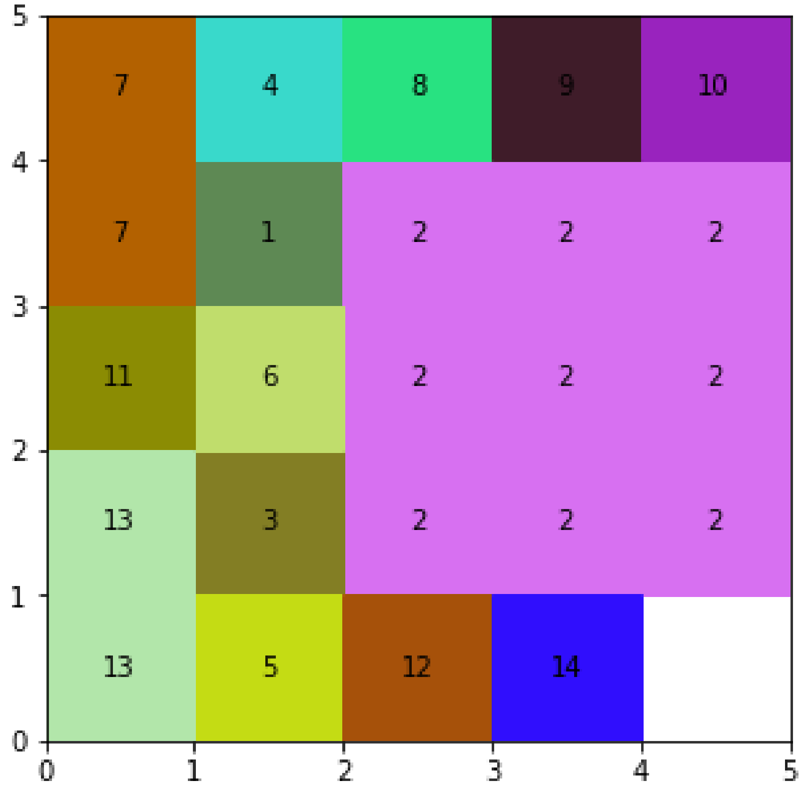

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5



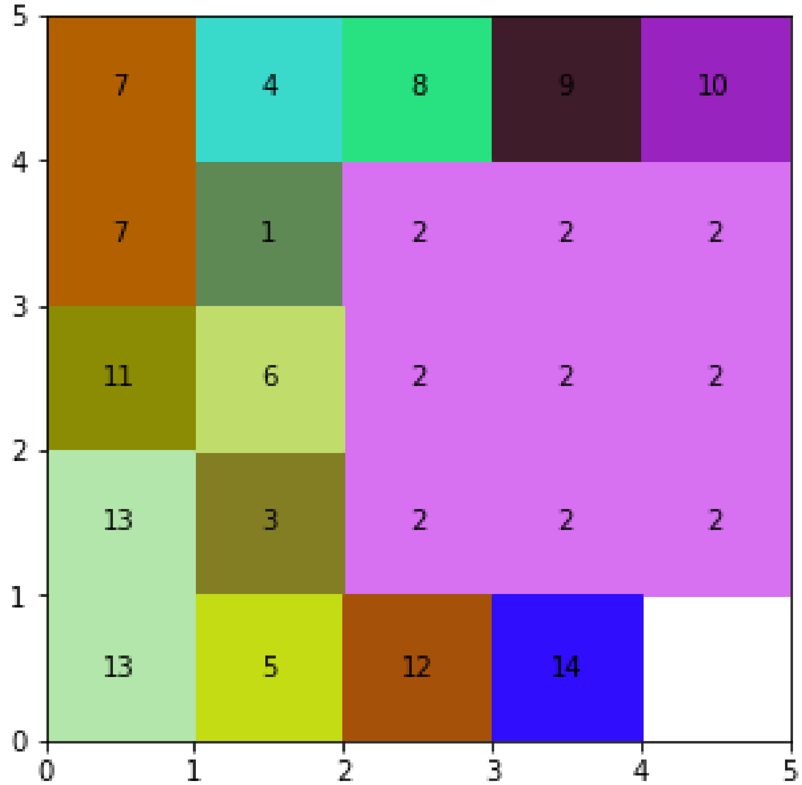
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5

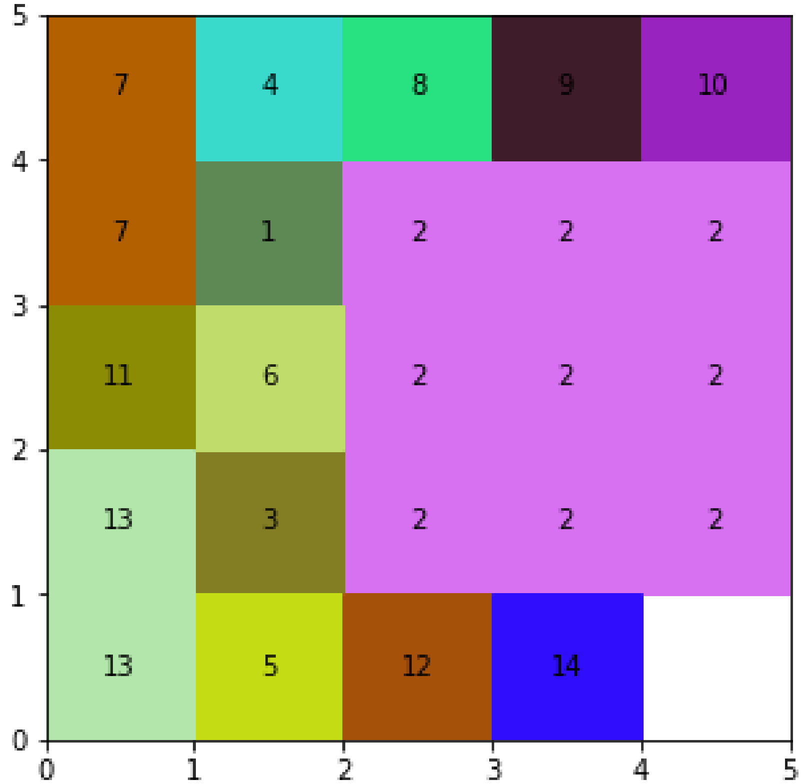

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5

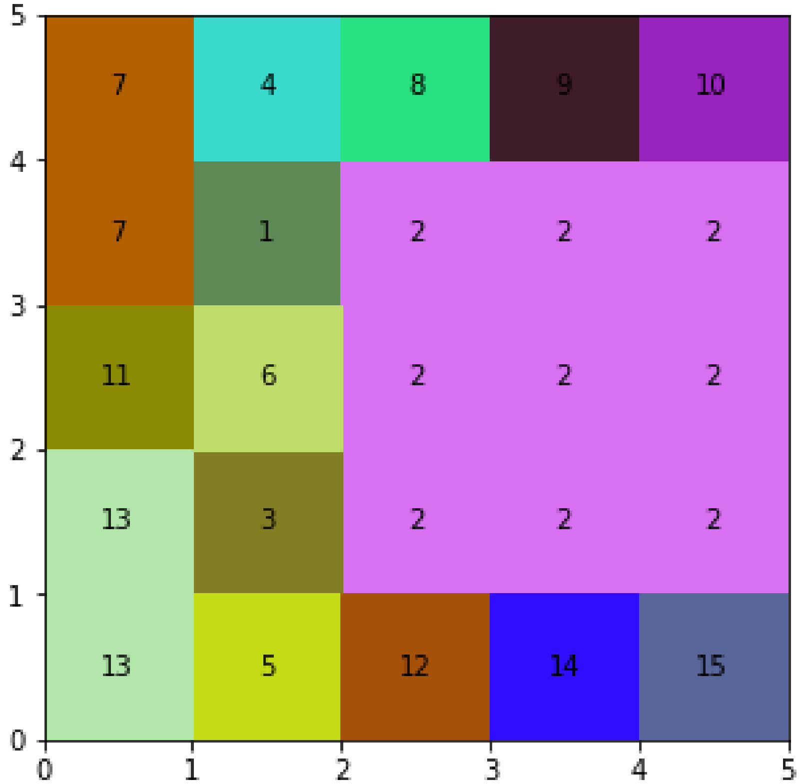


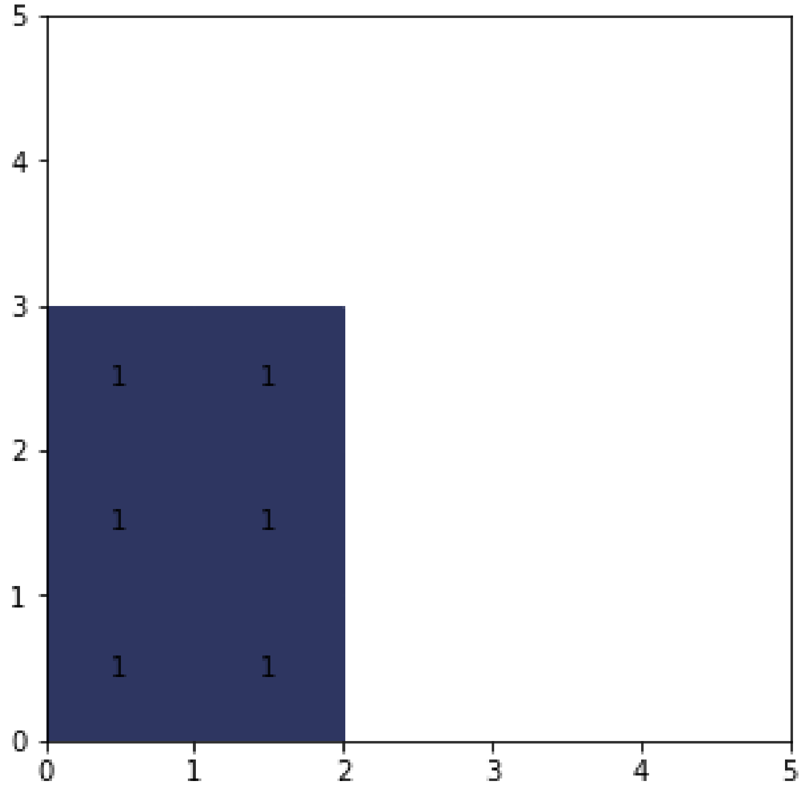
5
4
3
2
1
0
0
1
2
3
4
5
7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14

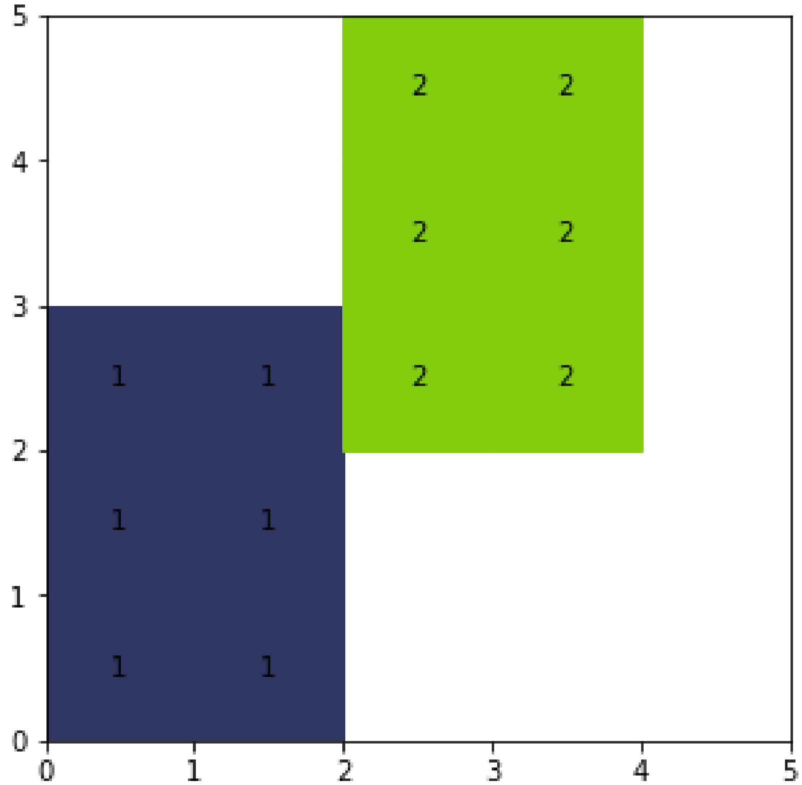


7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
0
1
2
3
4
5

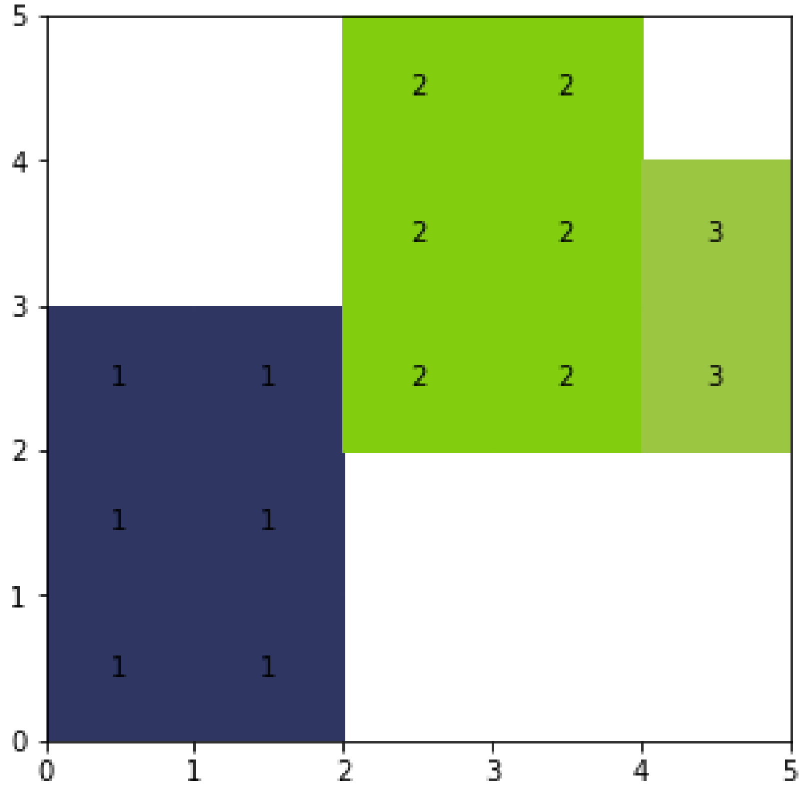

7 4 8 9 10
7 1 2 2 2
11 6 2 2 2
13 3 2 2 2
13 5 12 14
0 1 2 3 4 5
0 1 2 3 4 5

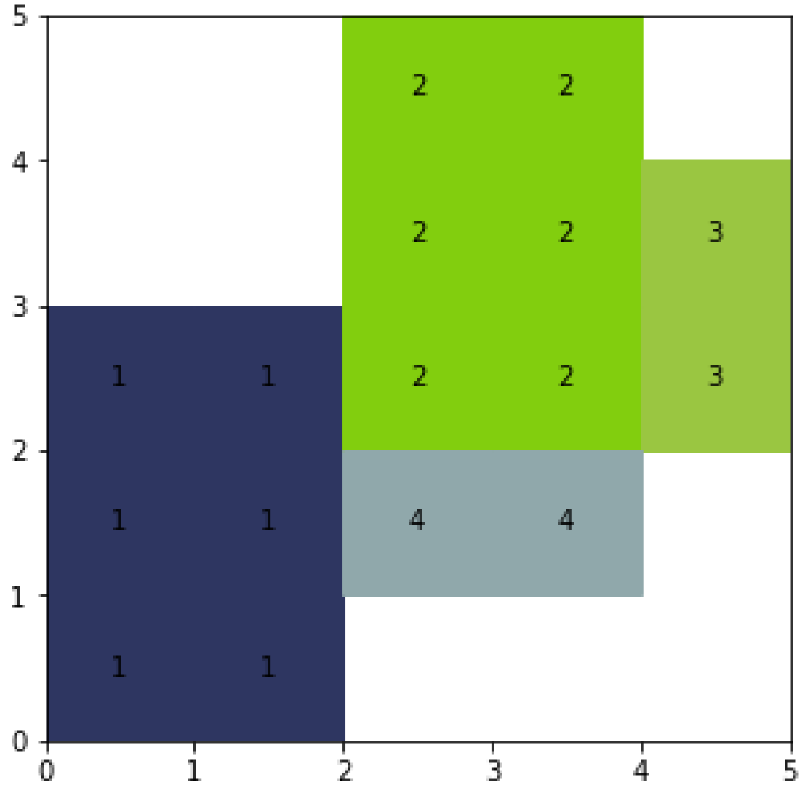

7
4
8
9
10
7
1
2
2
2
11
6
2
2
2
13
3
2
2
2
13
5
12
14
15
0
1
2
3
4
5

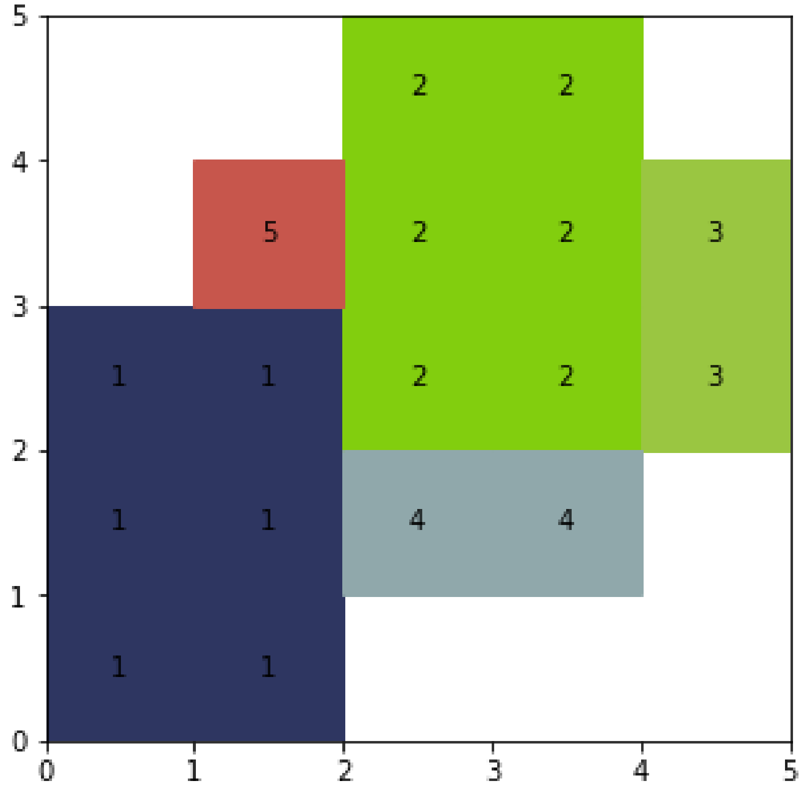

5
4
3
2
1
0
1 1
1 1
1 1
0 1 2 3 4 5

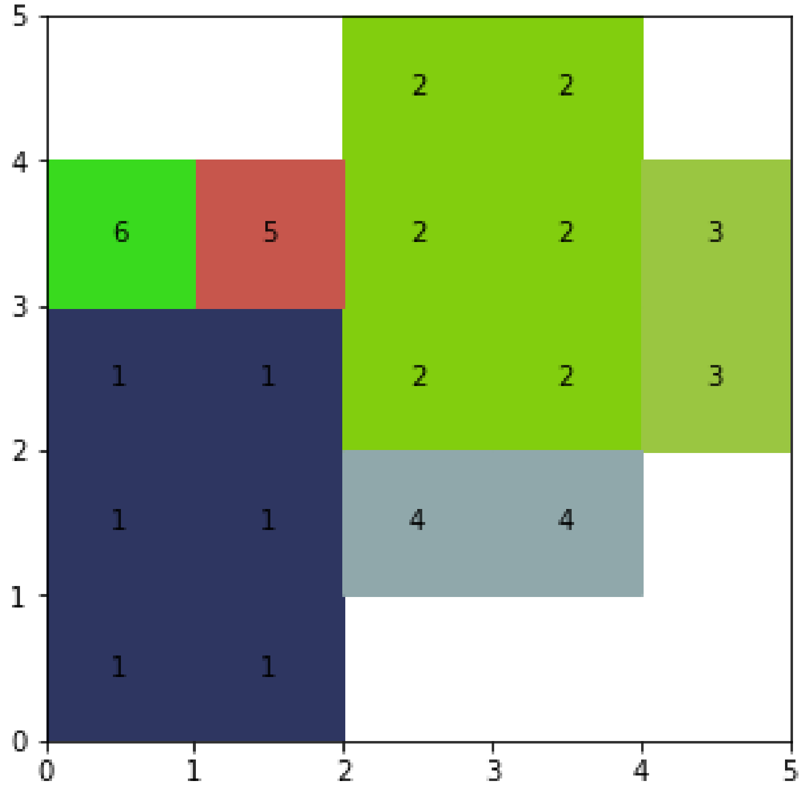


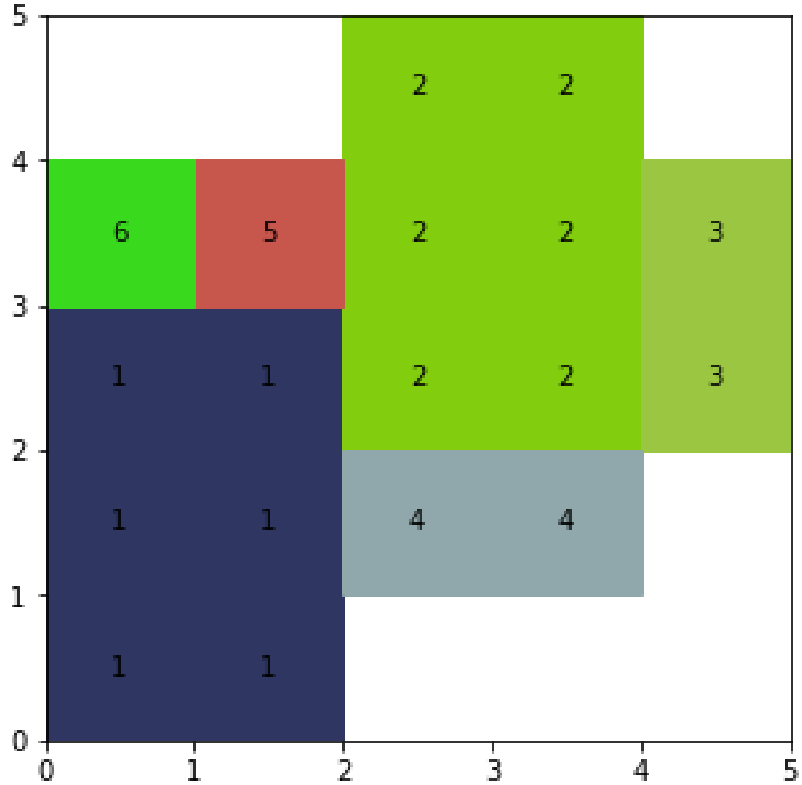
5
4
3
2
1
0
0
1
2
3
4
5
2 2
2 2
2 2
1 1
1 1
1 1


0
1
2
3
4
5
1
1
1
1
1
1
2
2
2
2
2
2
3
3

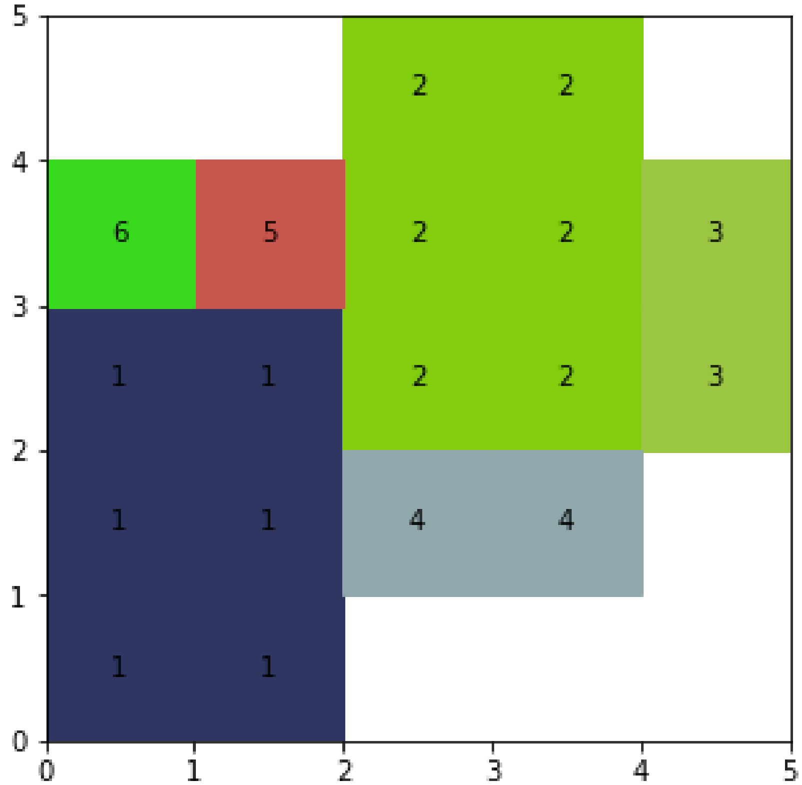

5
4
3
2
1
0
0
1
2
3
4
5
2 2
2 2 3
1 1 2 2 3
1 1 4 4
1 1


5
4
3
2
1
0
0
1
2
3
4
5
2
2
5
2
2
3
1
1
2
2
3
1
1
4
4
1
1

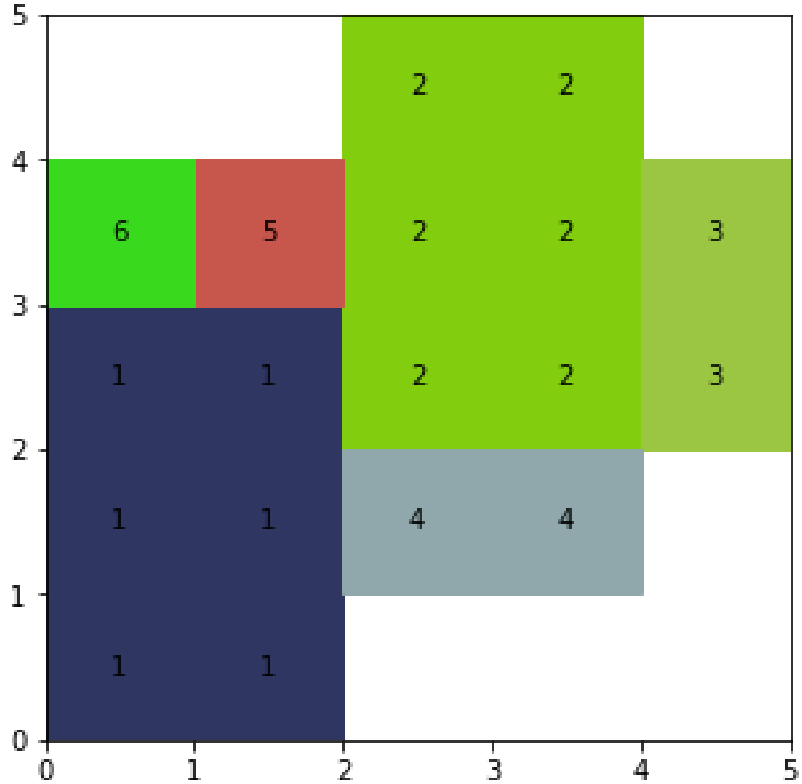

5
4
3
2
1
0
0
1
2
3
4
5
2 2
6 5 2 2 3
1 1 2 2 3
1 1 4 4
1 1



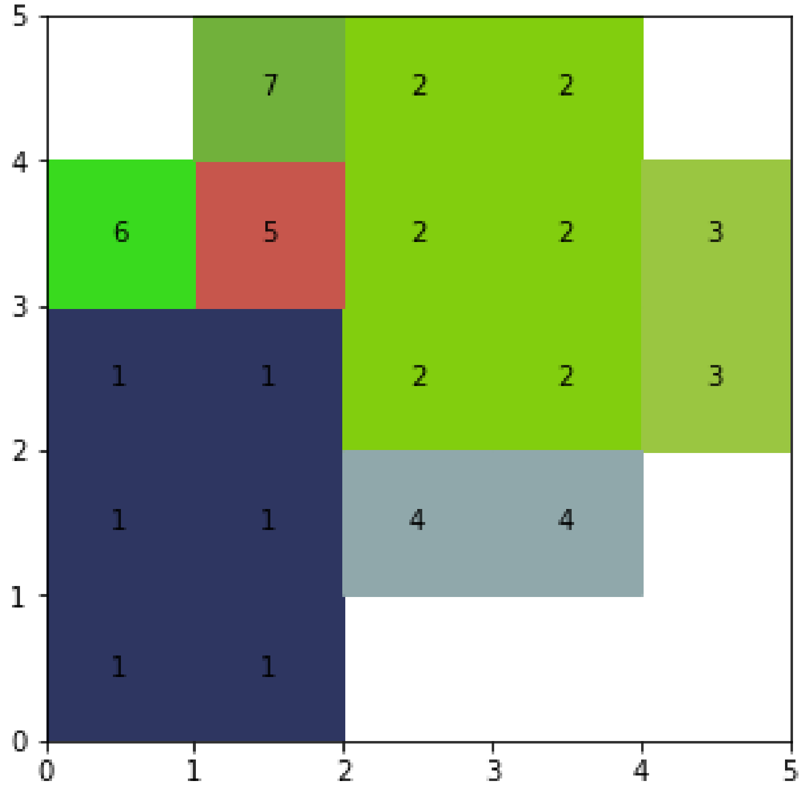
5
4
3
2
1
0
0
1
2
3
4
5
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
1
1

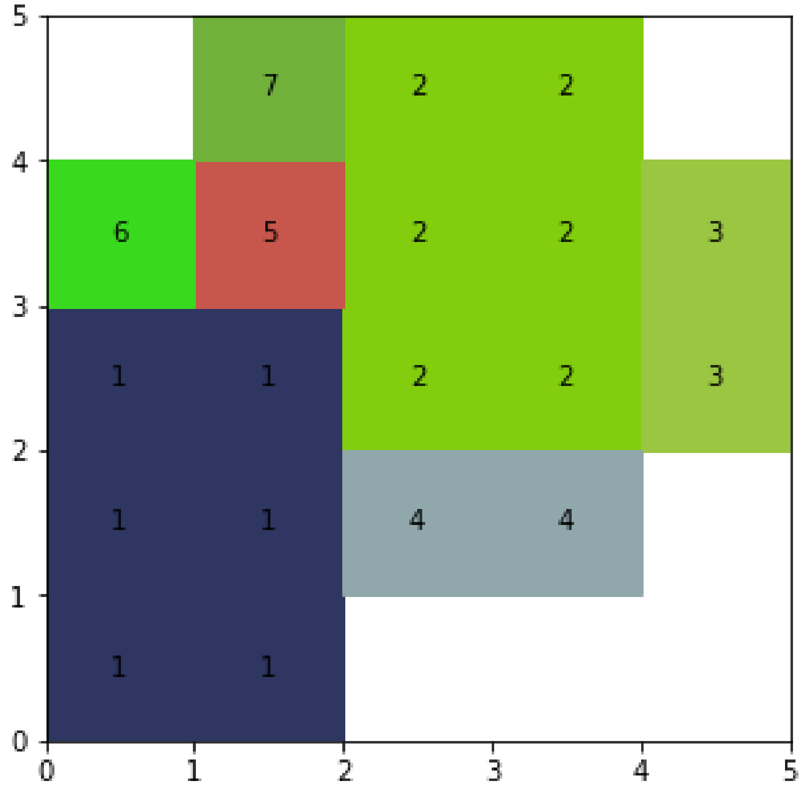


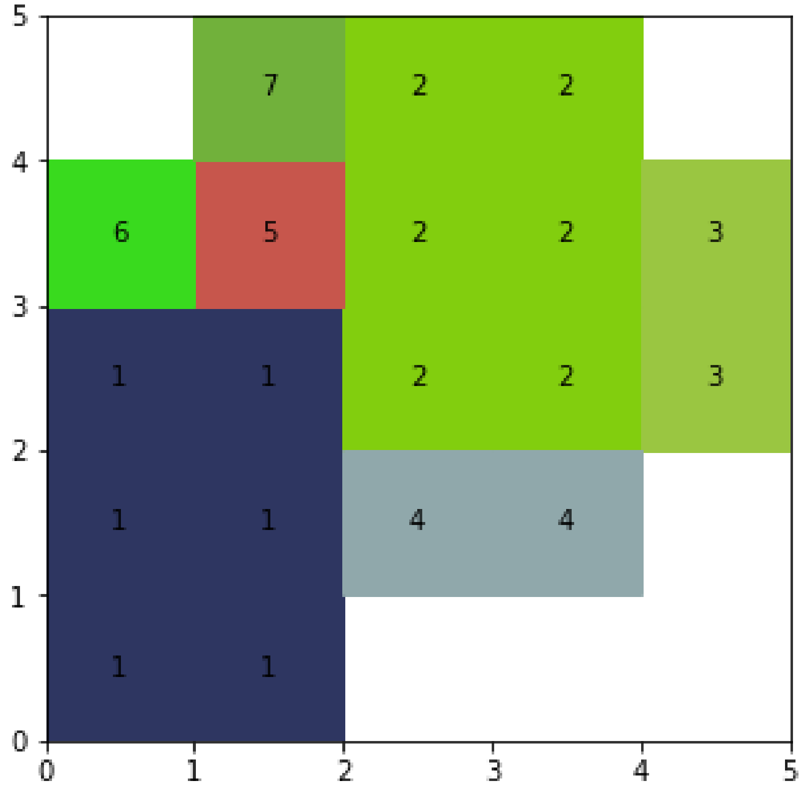
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
1
1
0
1
2
3
4
5



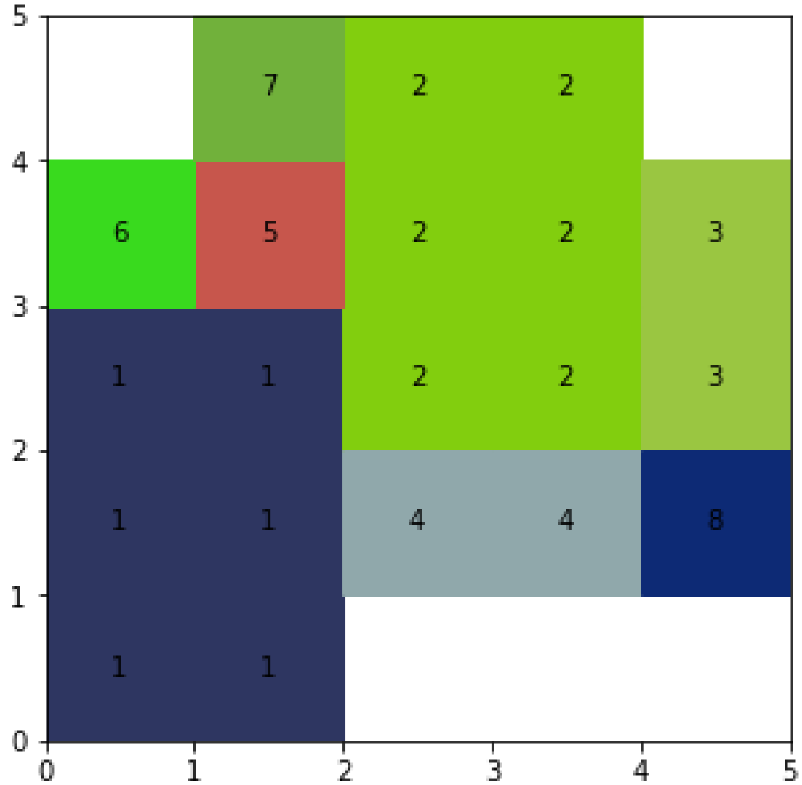
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
1
1
0
1
2
3
4
5


7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
1
1
0
1
2
3
4
5

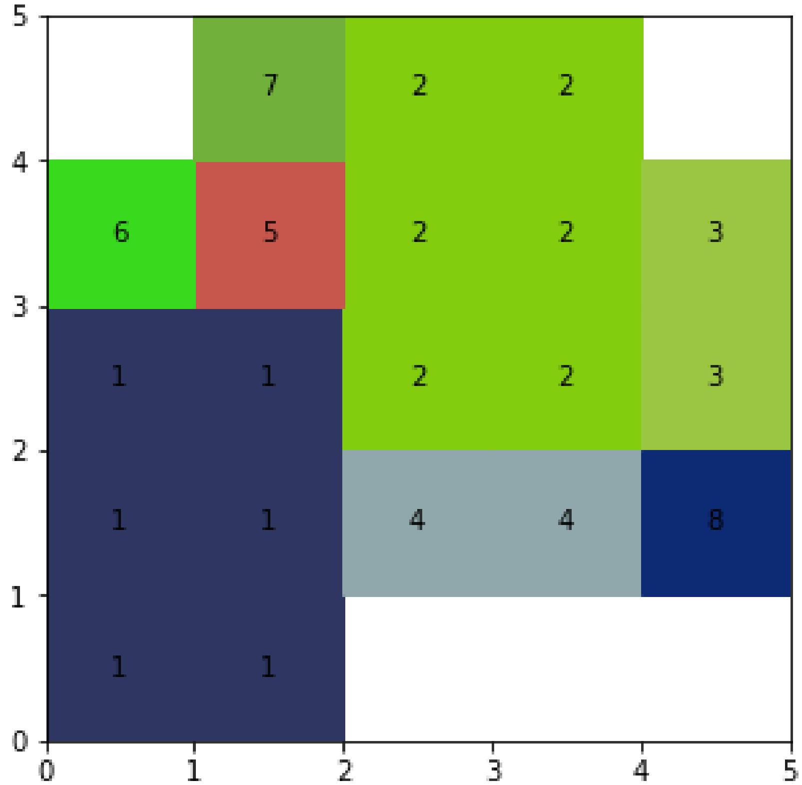

7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
1
1
0
1
2
3
4
5

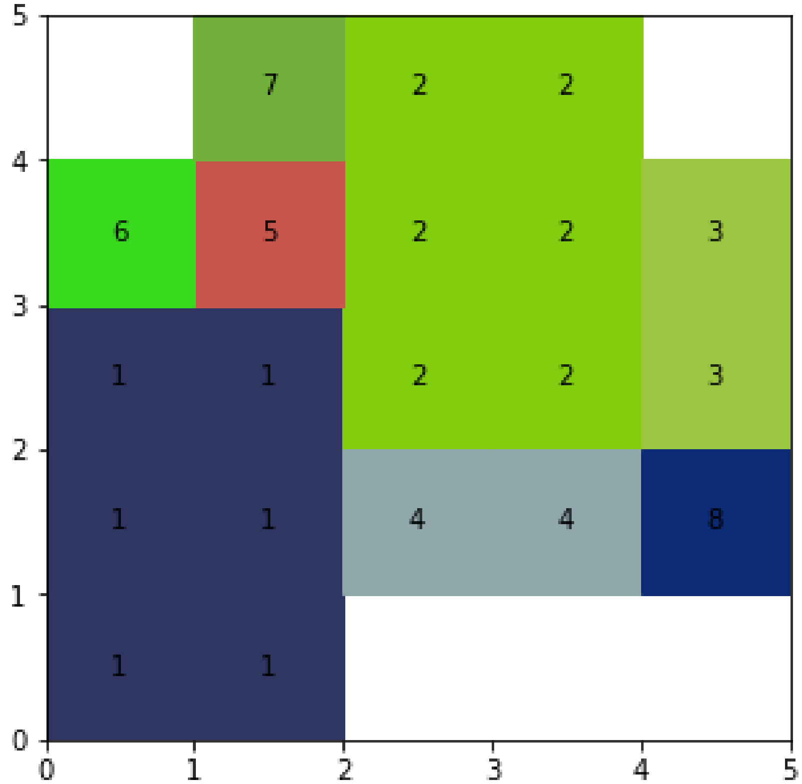


7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
1
1
0
1
2
3
4
5

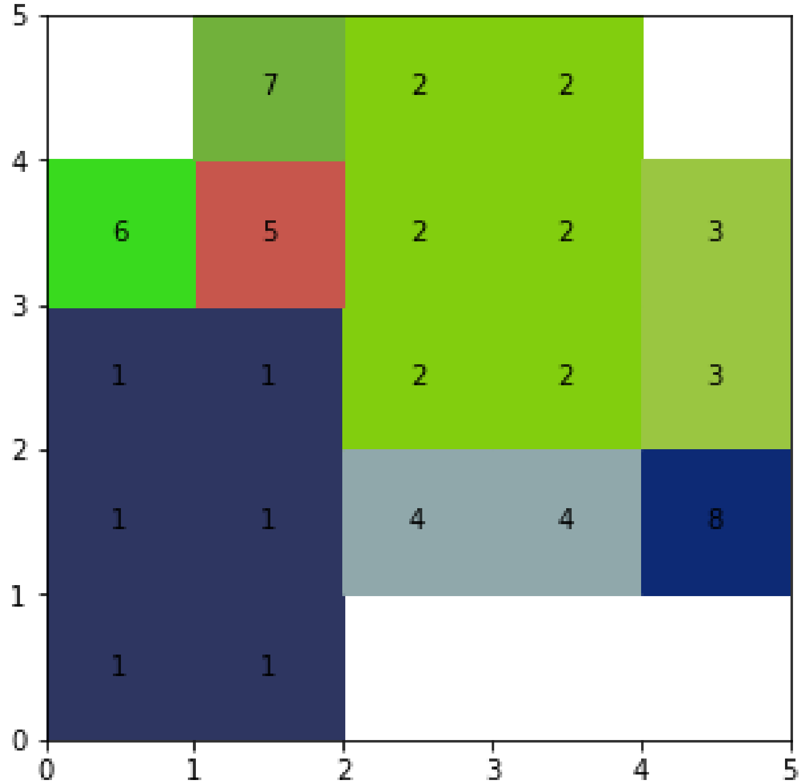


7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5

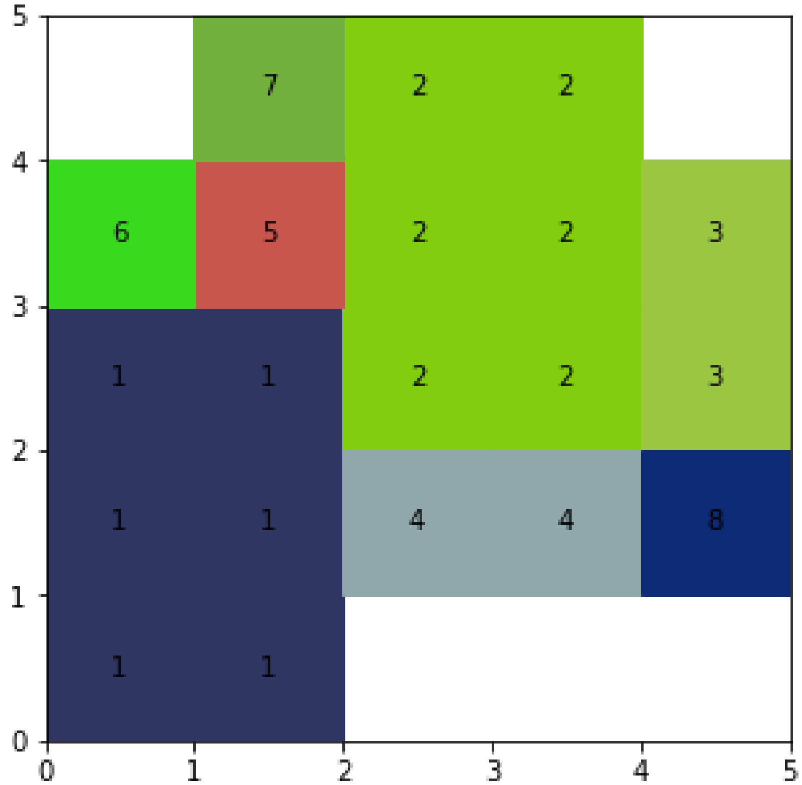

7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5



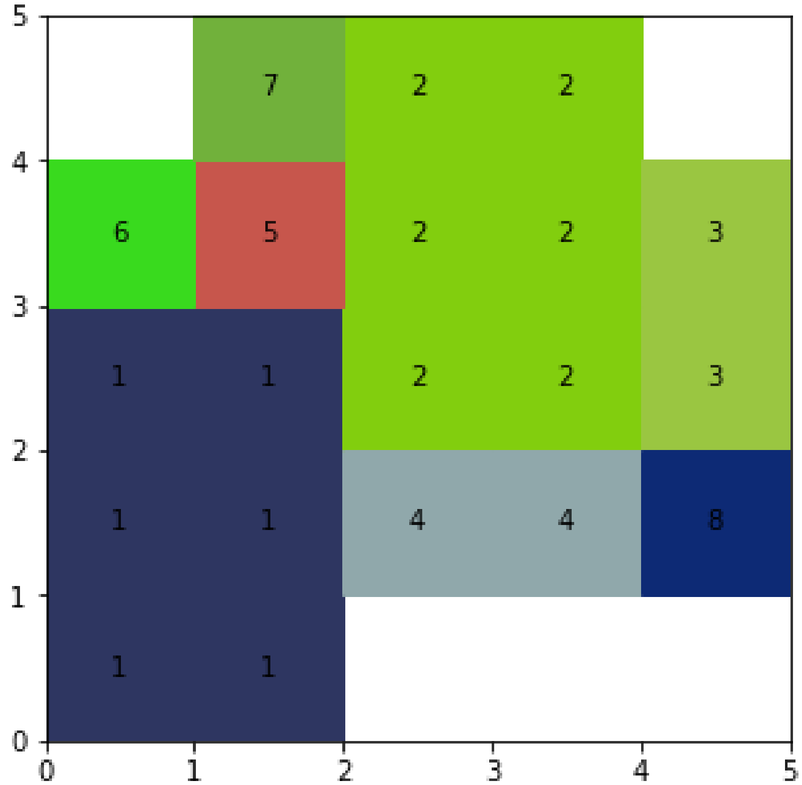
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5



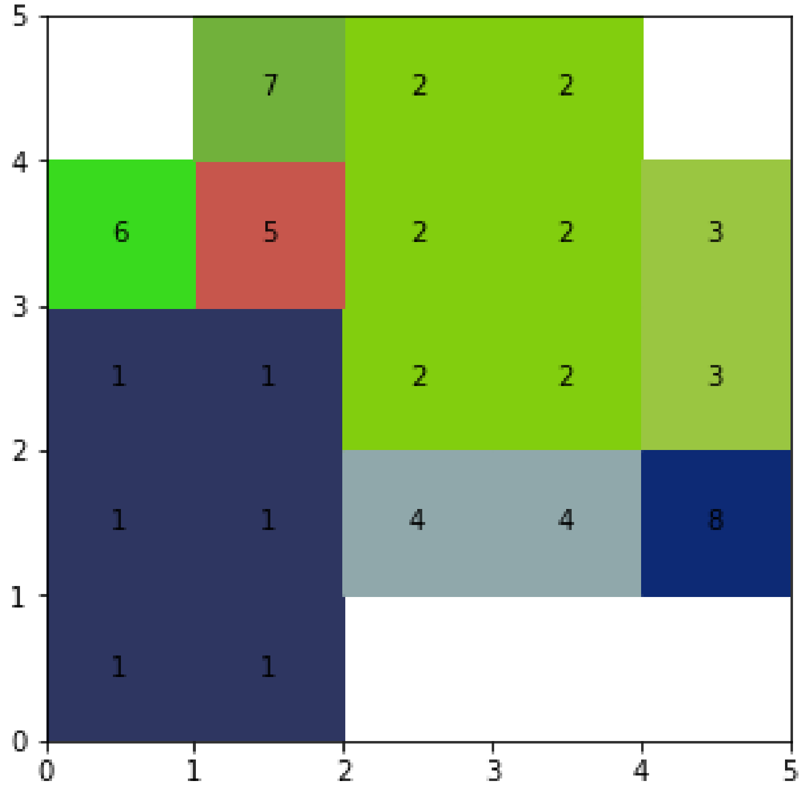
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5



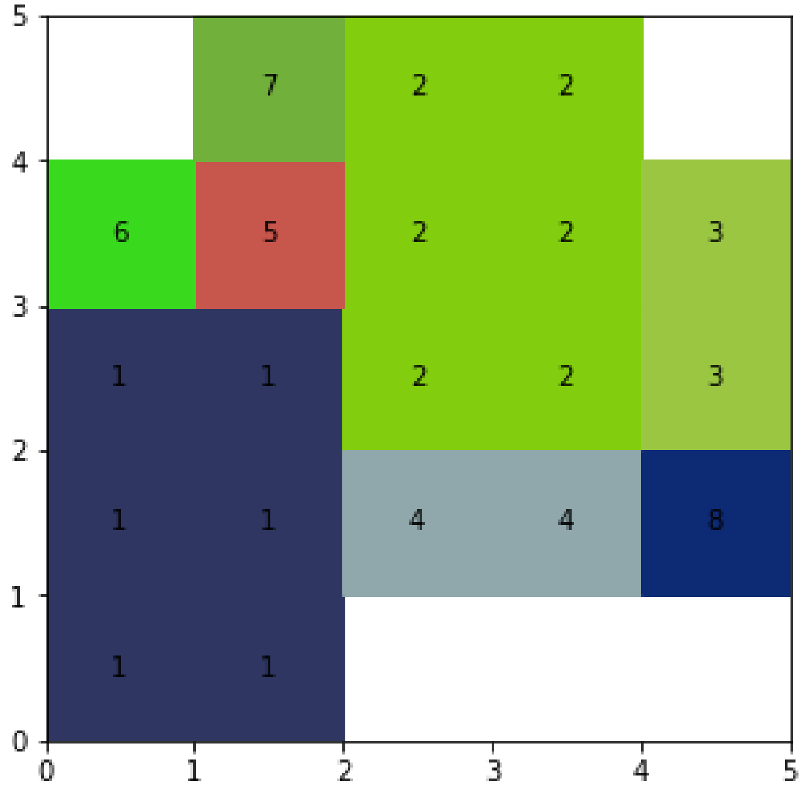
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5

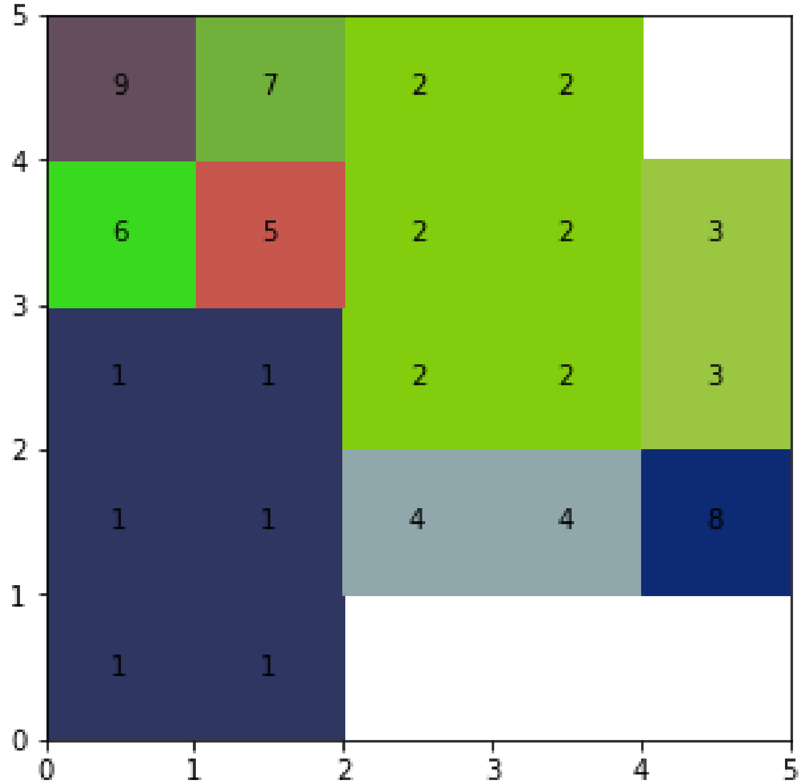


7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5

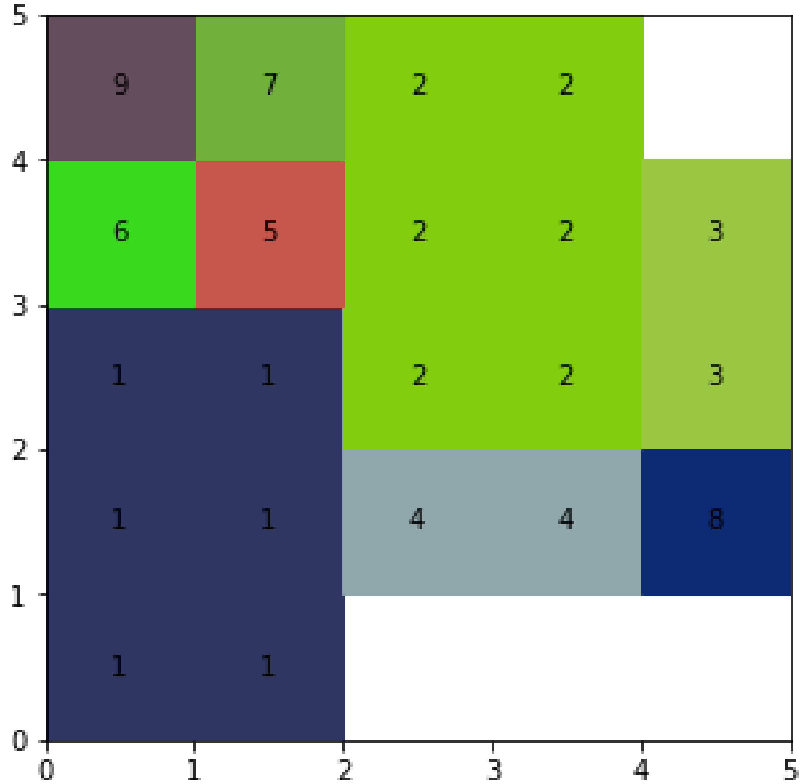


7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5

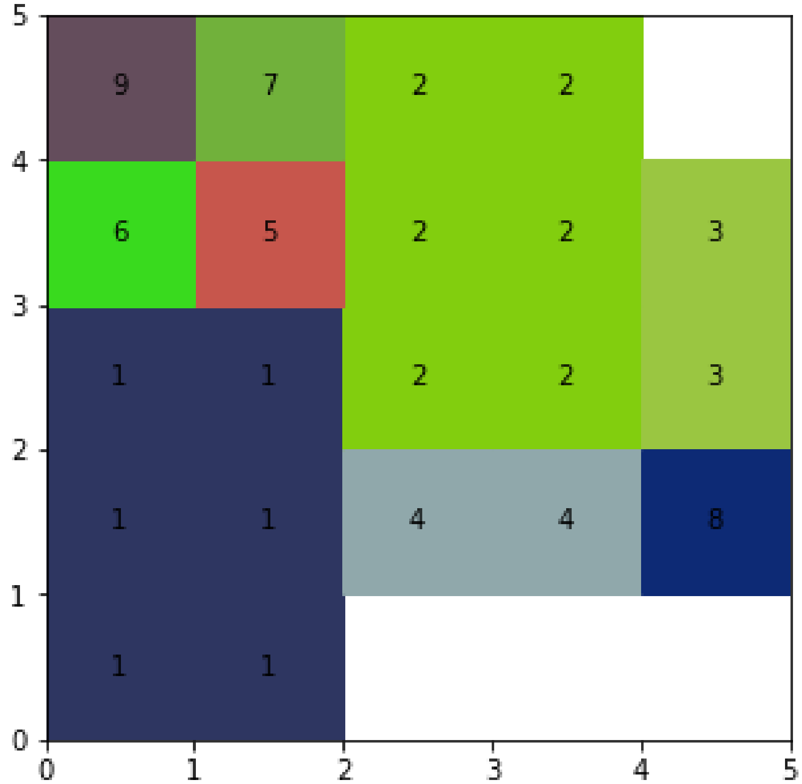

7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5

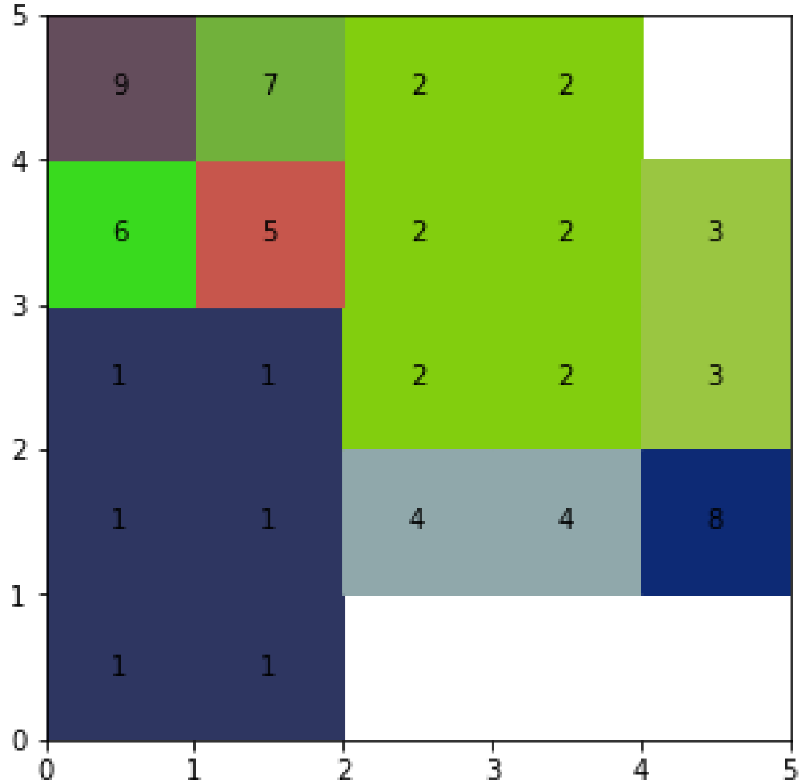


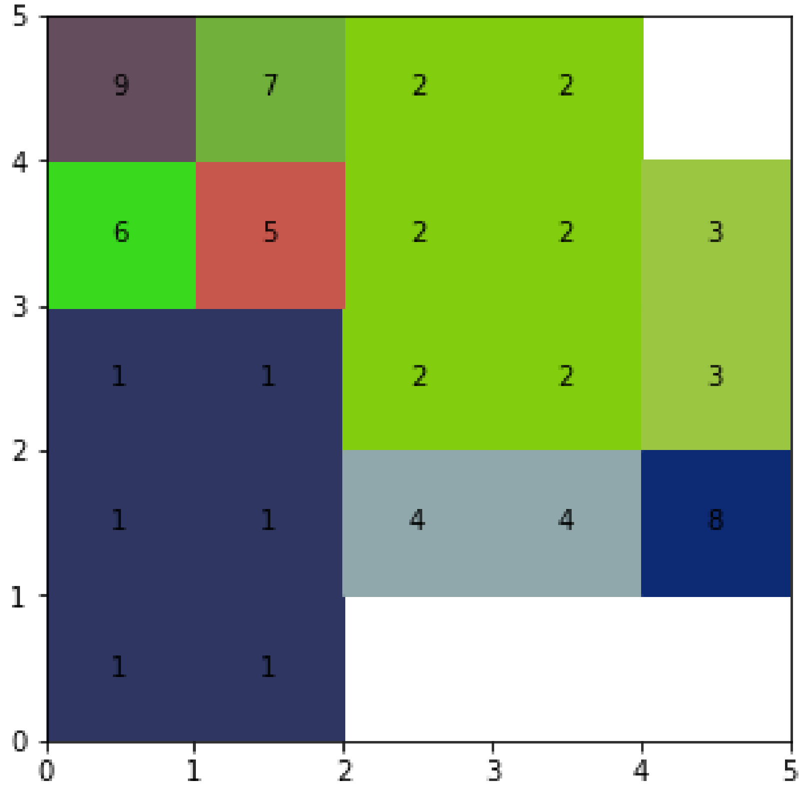
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5



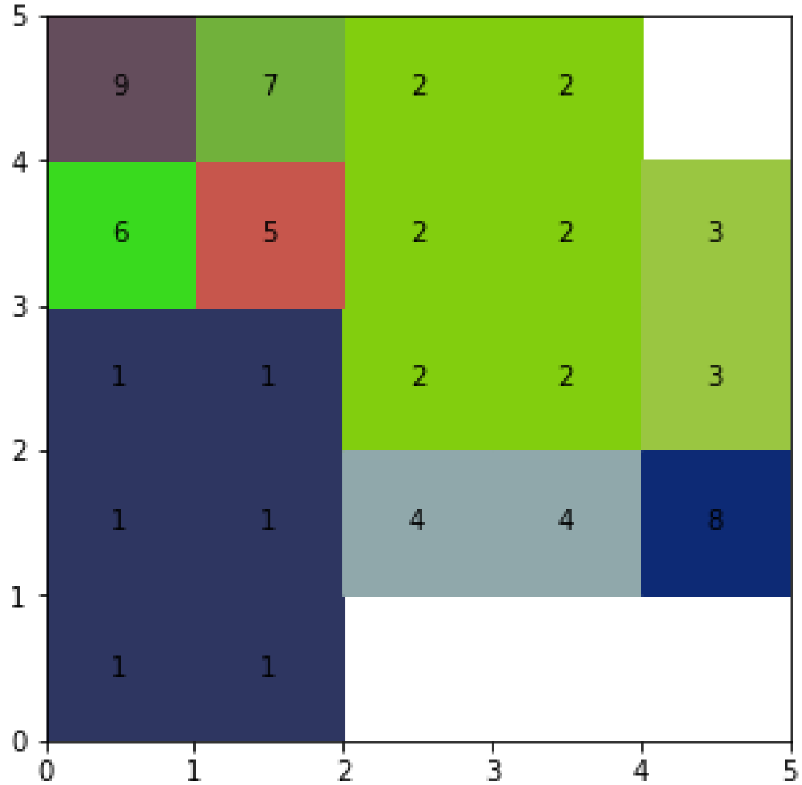
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5


9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5

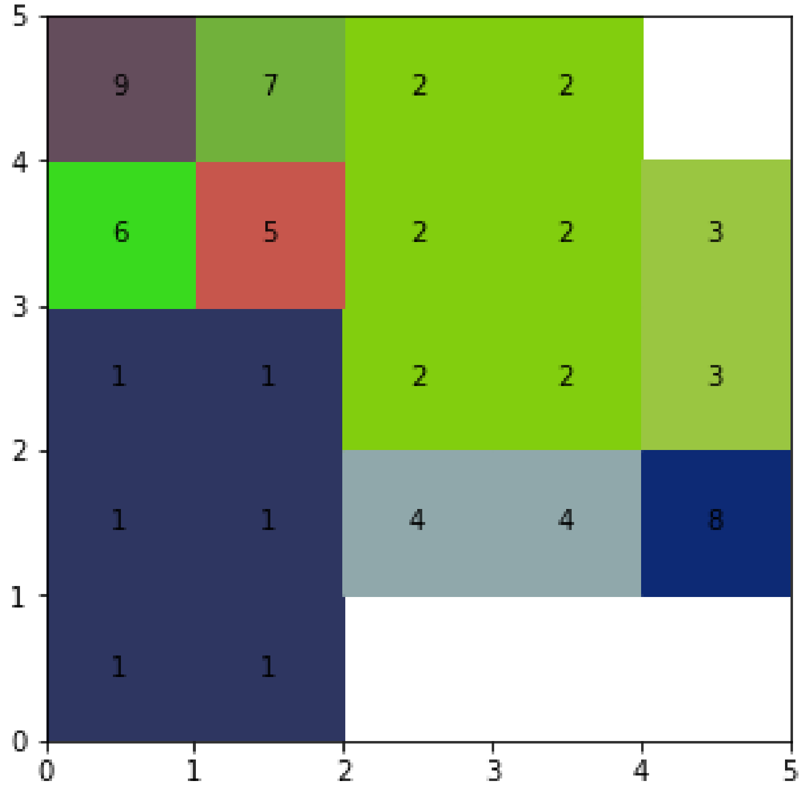

9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5



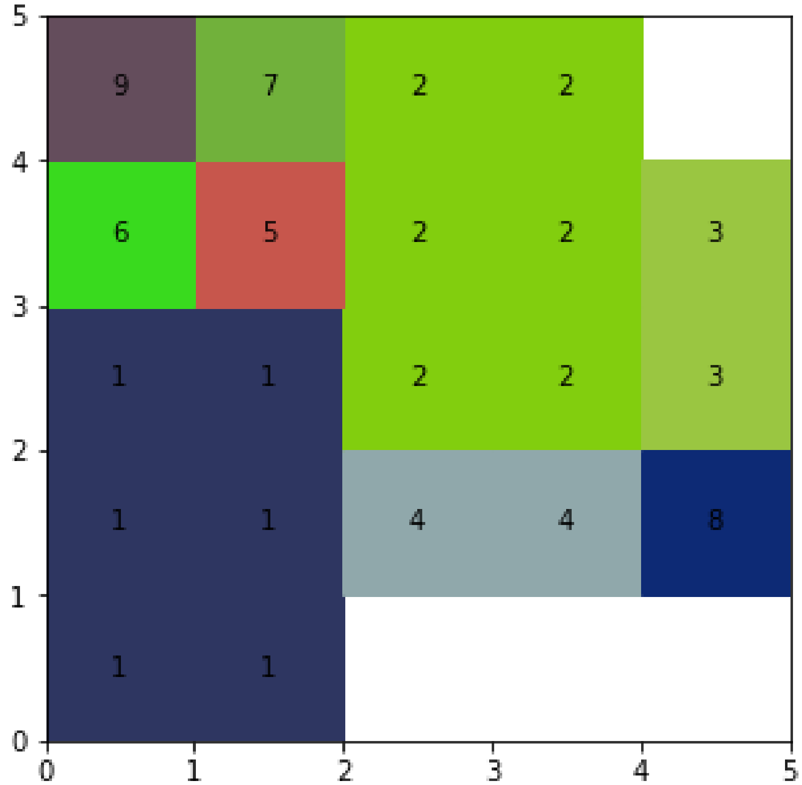
5
4
3
2
1
0
0
1
2
3
4
5
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1

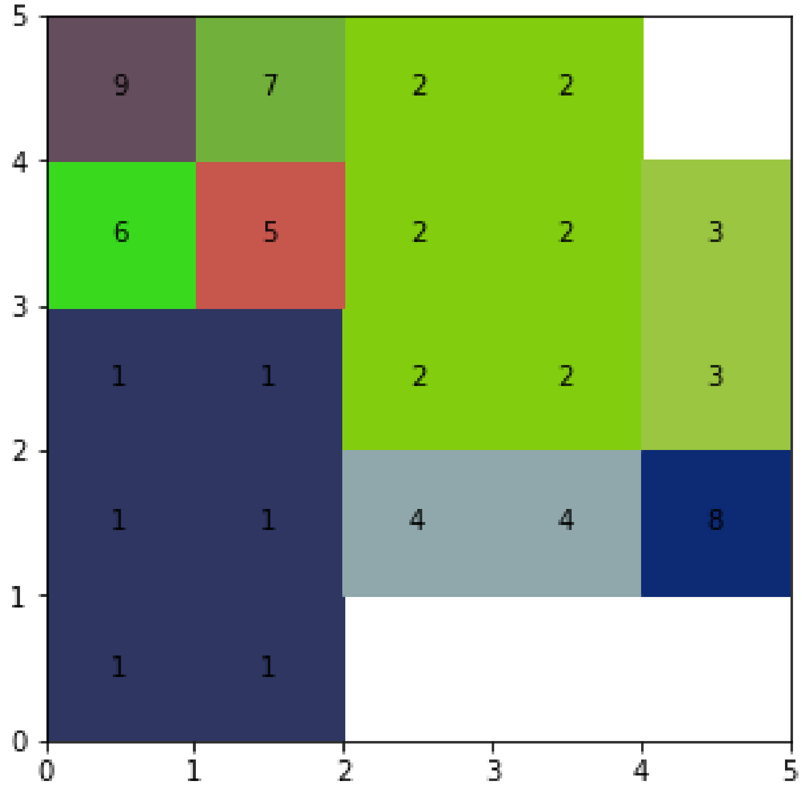


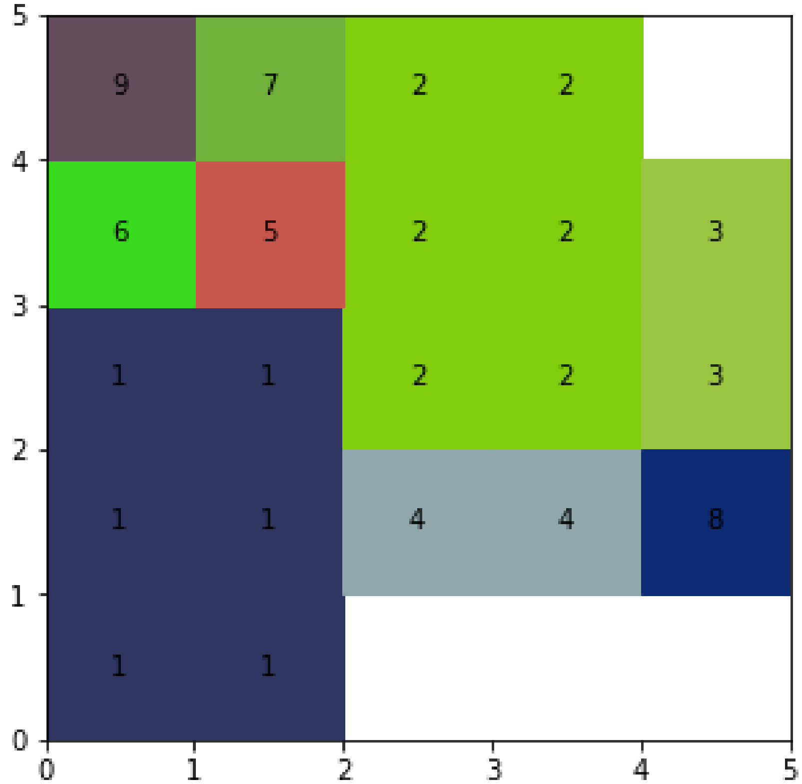
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5

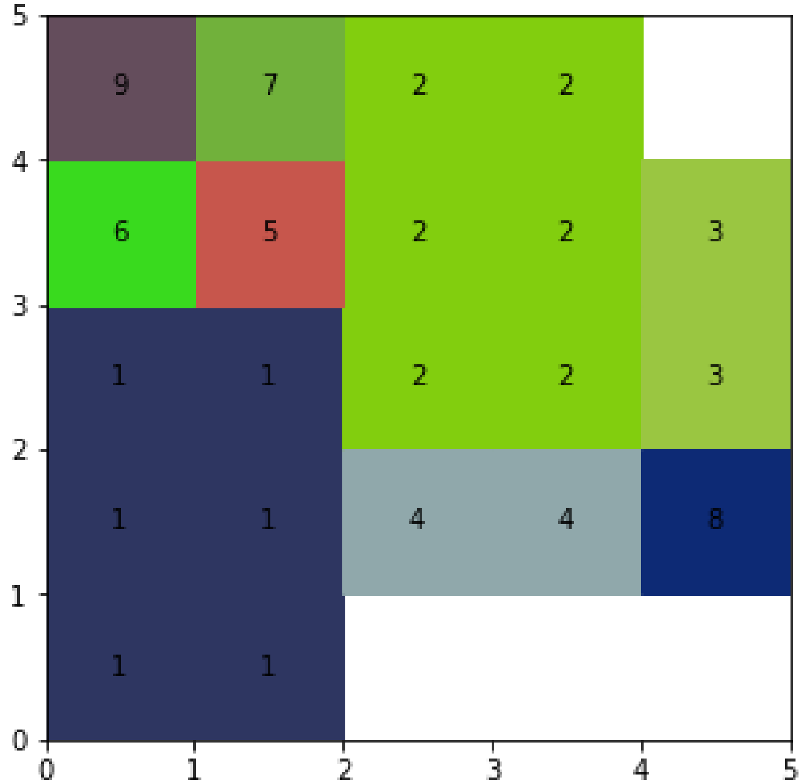


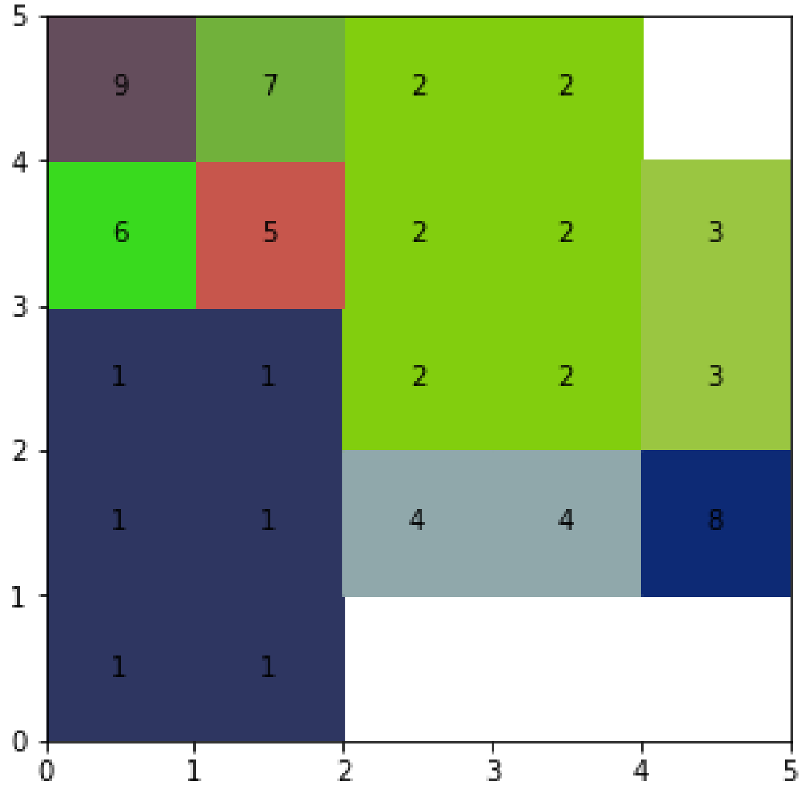
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5

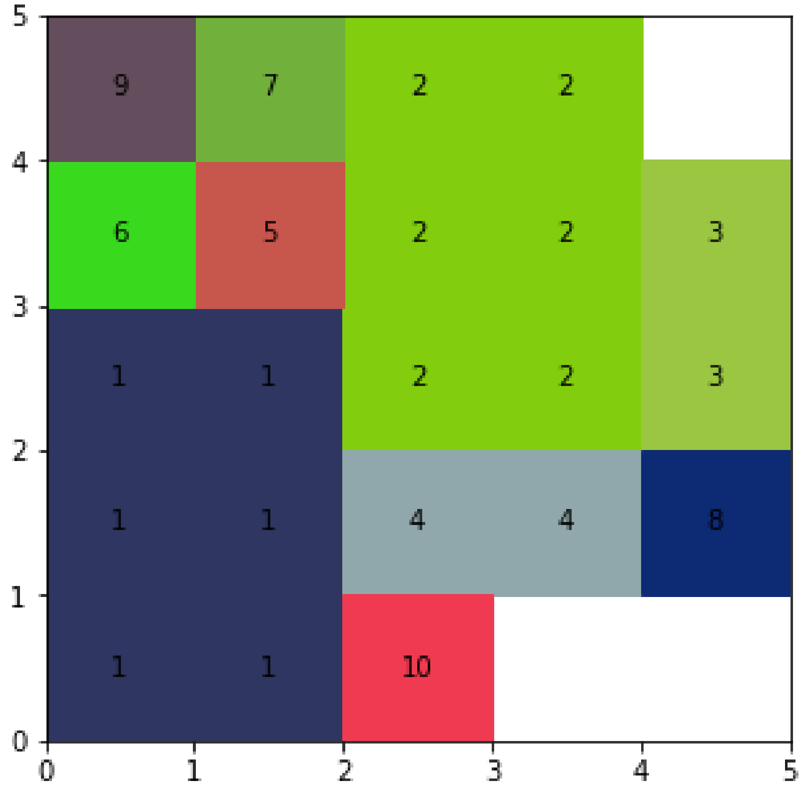

9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5
0
1
2
3
4
5


9 7 2 2
6 5 2 2 3
1 1 2 2 3
1 1 4 4 8
1 1
0 1 2 3 4 5

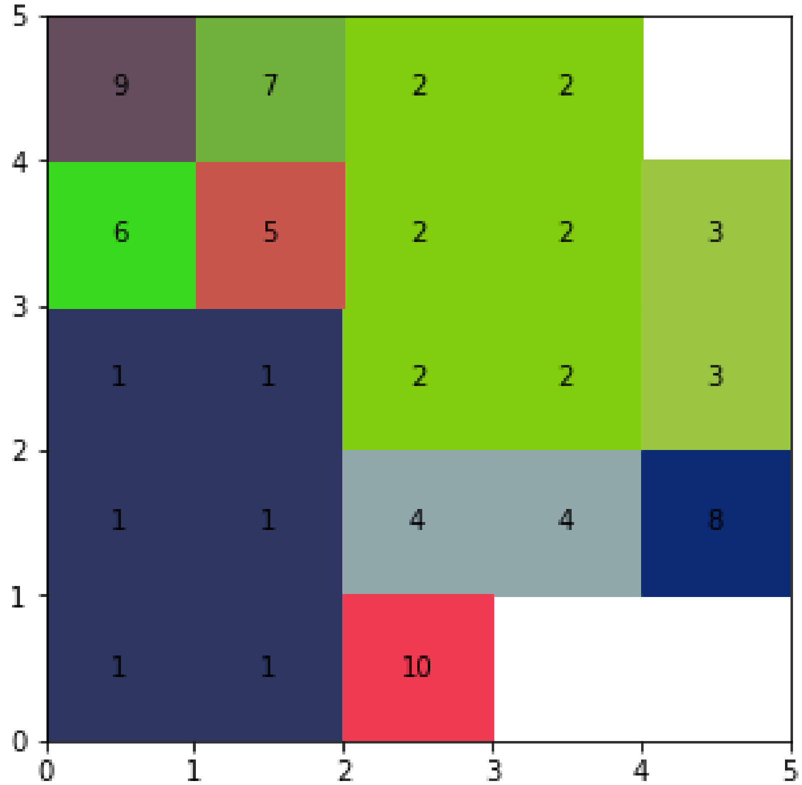

9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5
0
1
2
3
4
5


9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
0
1
2
3
4
5

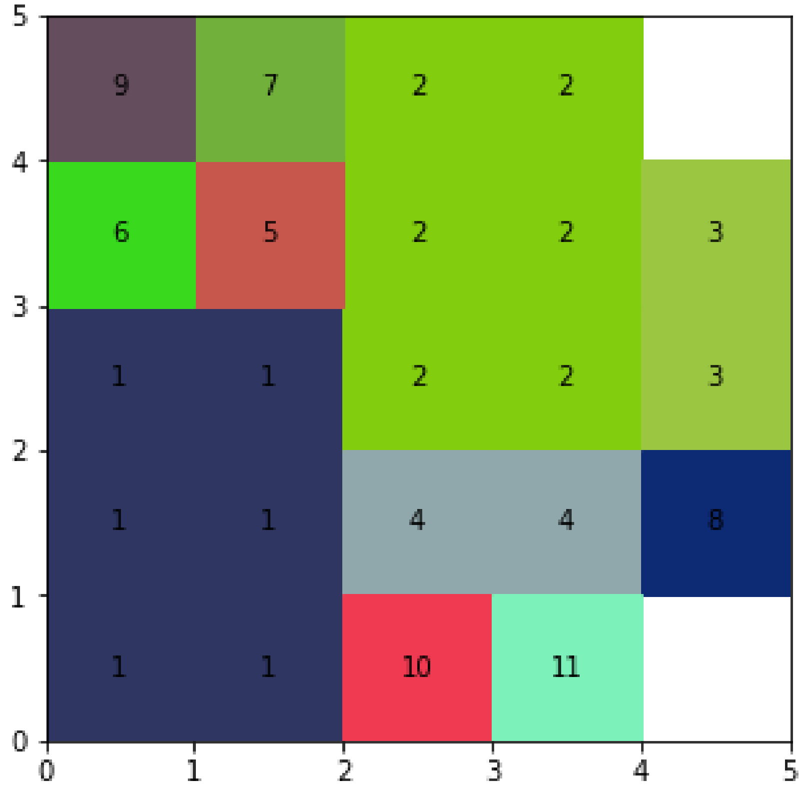

9 7 2 2
6 5 2 2 3
1 1 2 2 3
1 1 4 4 8
1 1
0 1 2 3 4 5
0 1 2 3 4 5

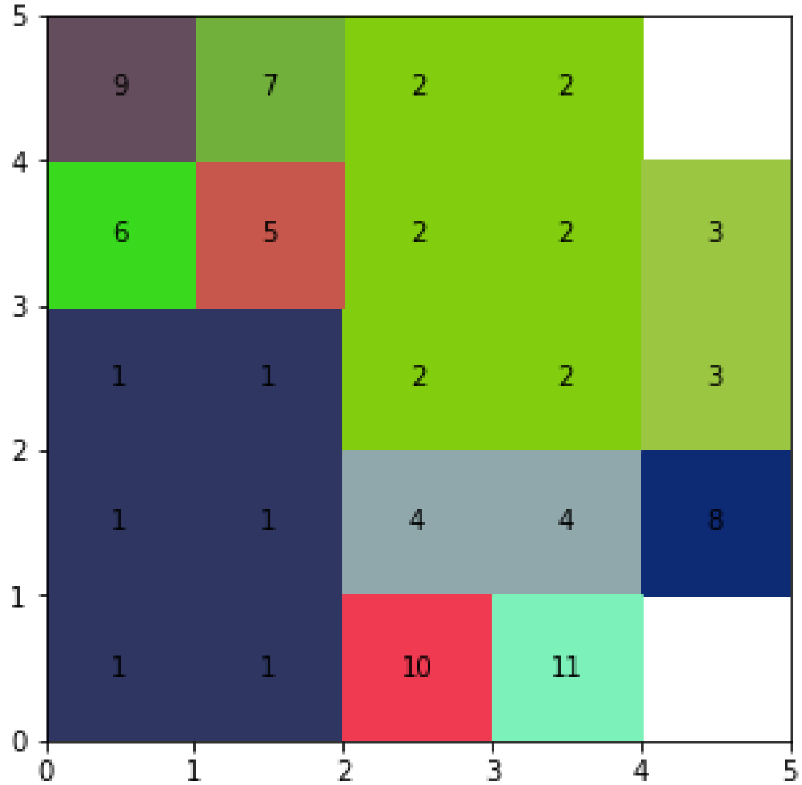


9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
0
1
2
3
4
5

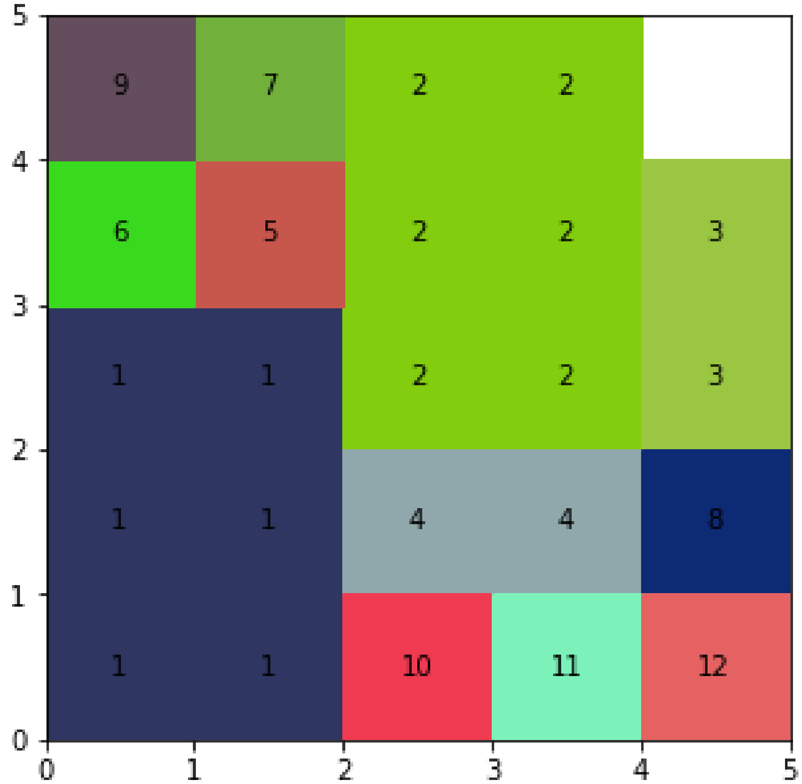


5
4
3
2
1
0
0
1
2
3
4
5
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10

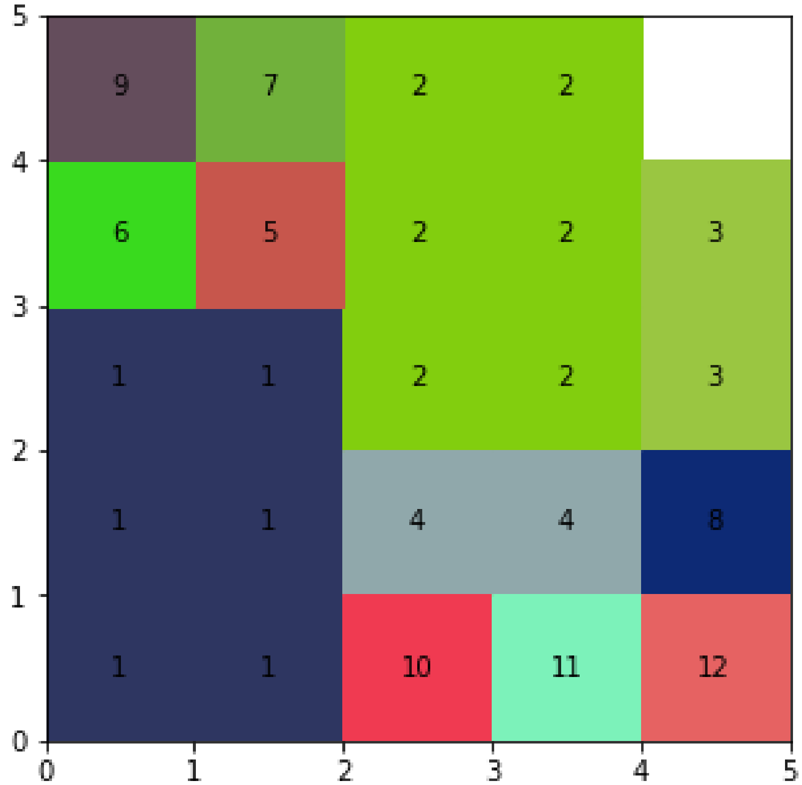

9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
0
1
2
3
4
5



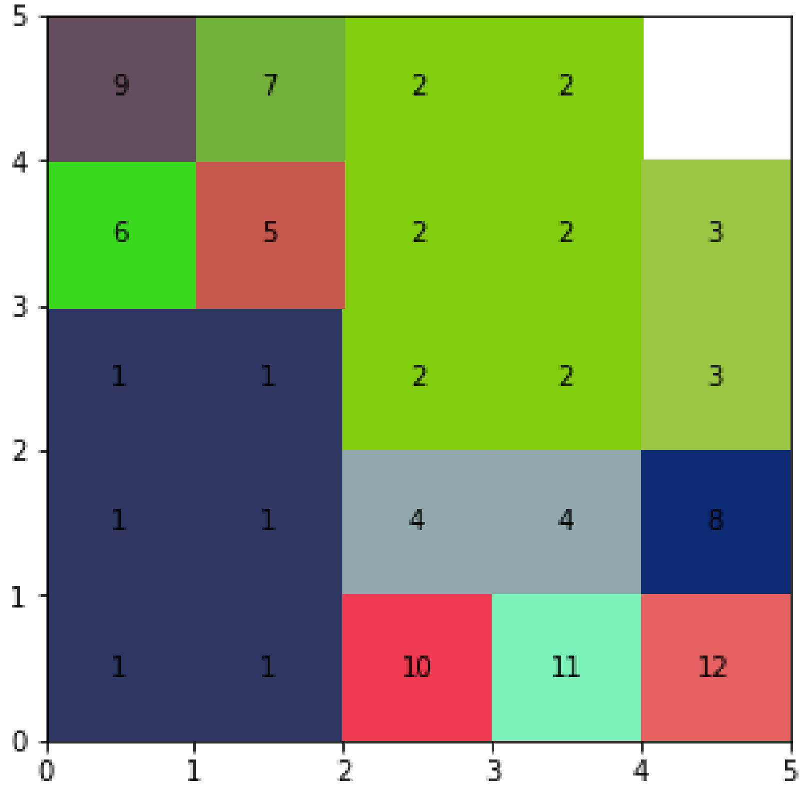
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
0
1
2
3
4
5



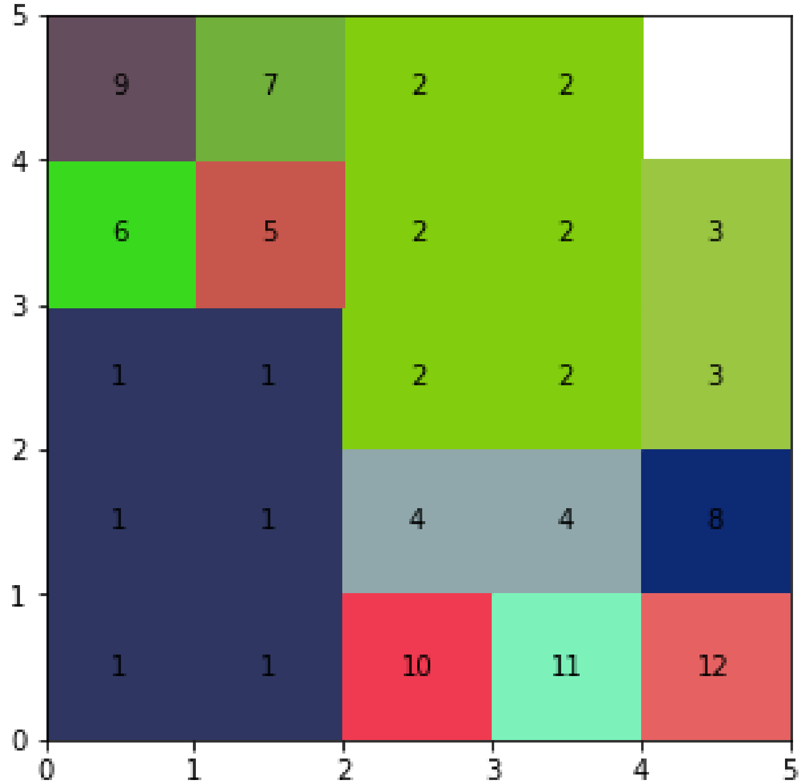
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5

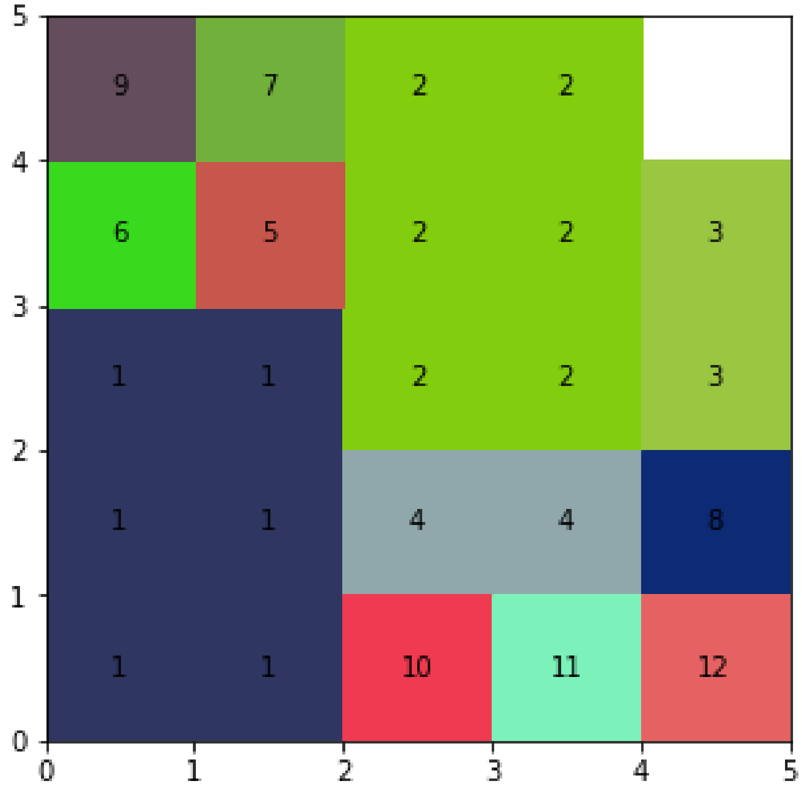


9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5

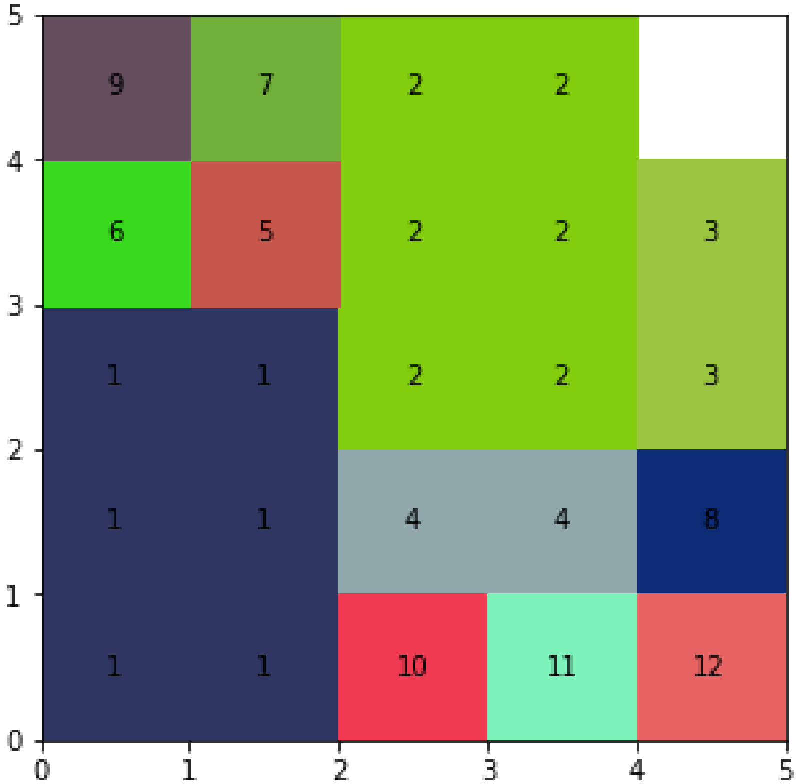


9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5

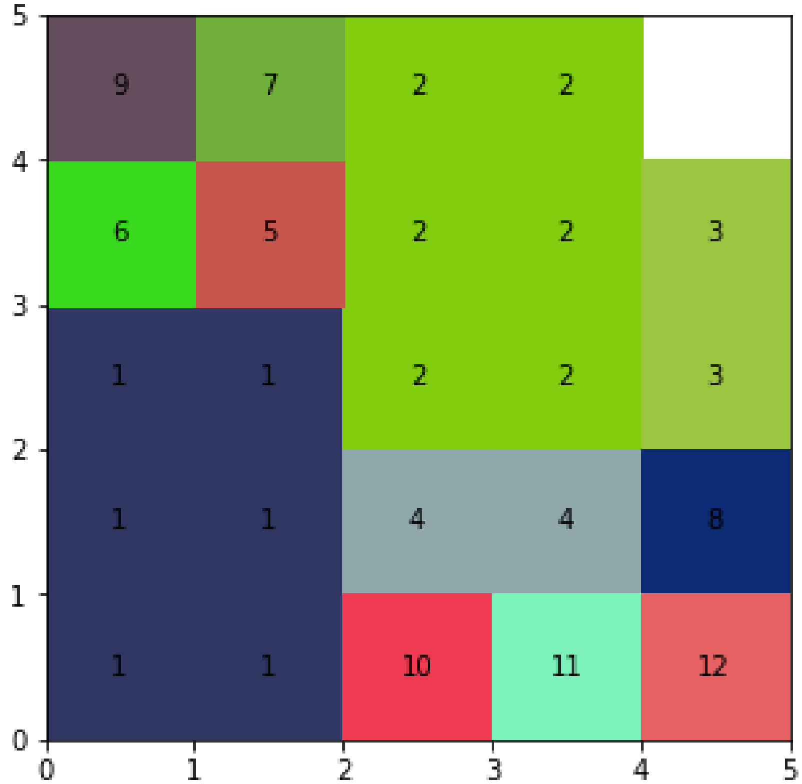

9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5

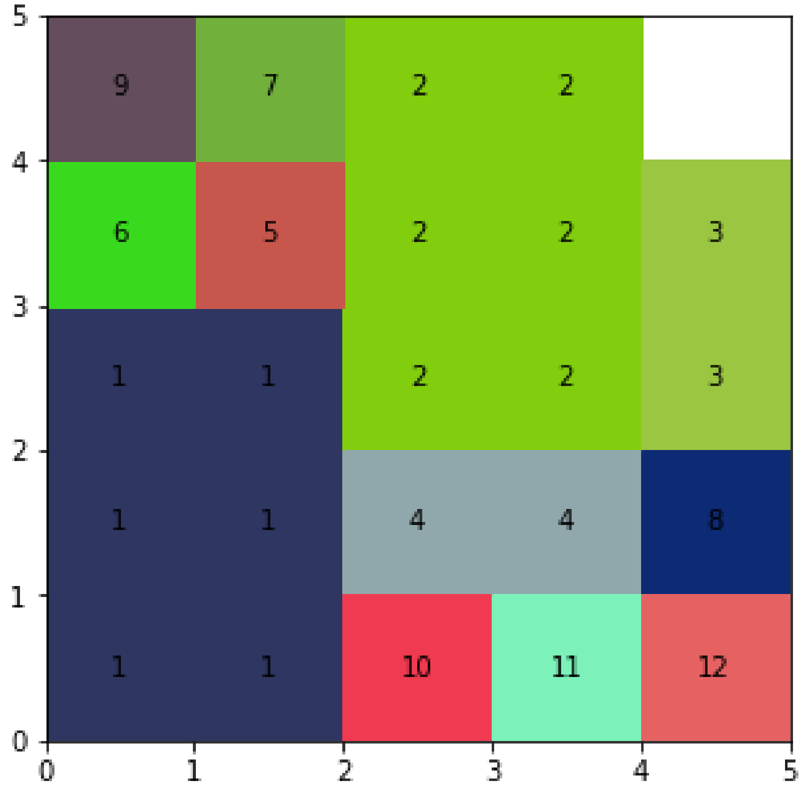


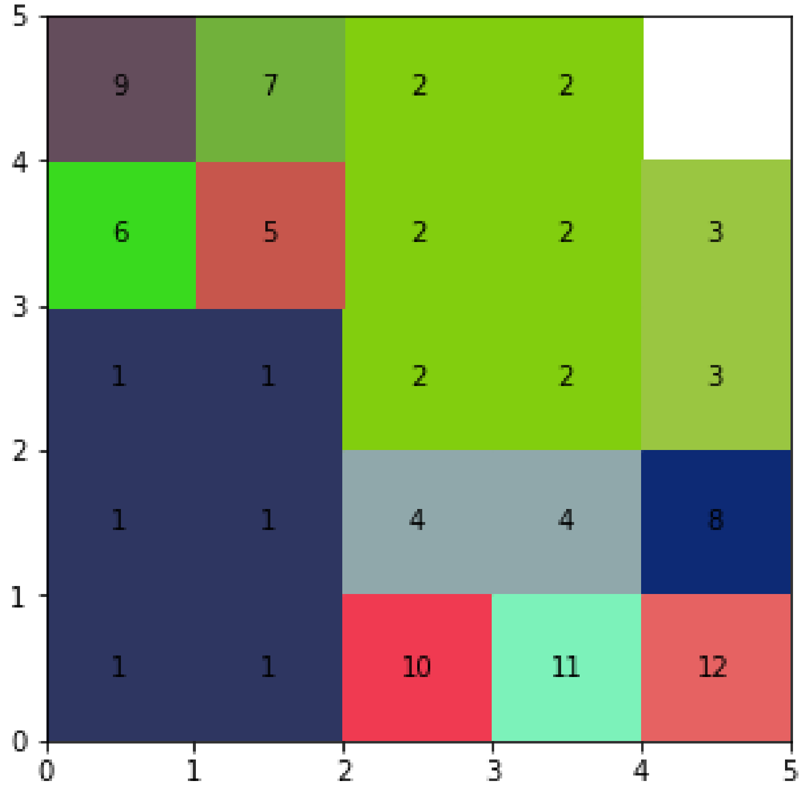
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5



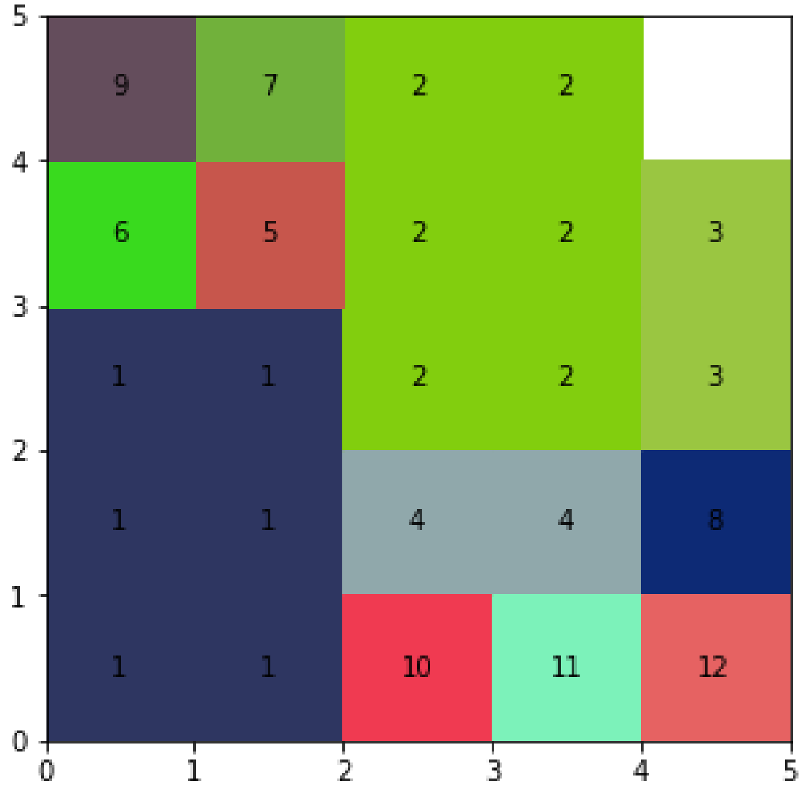
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5


9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5

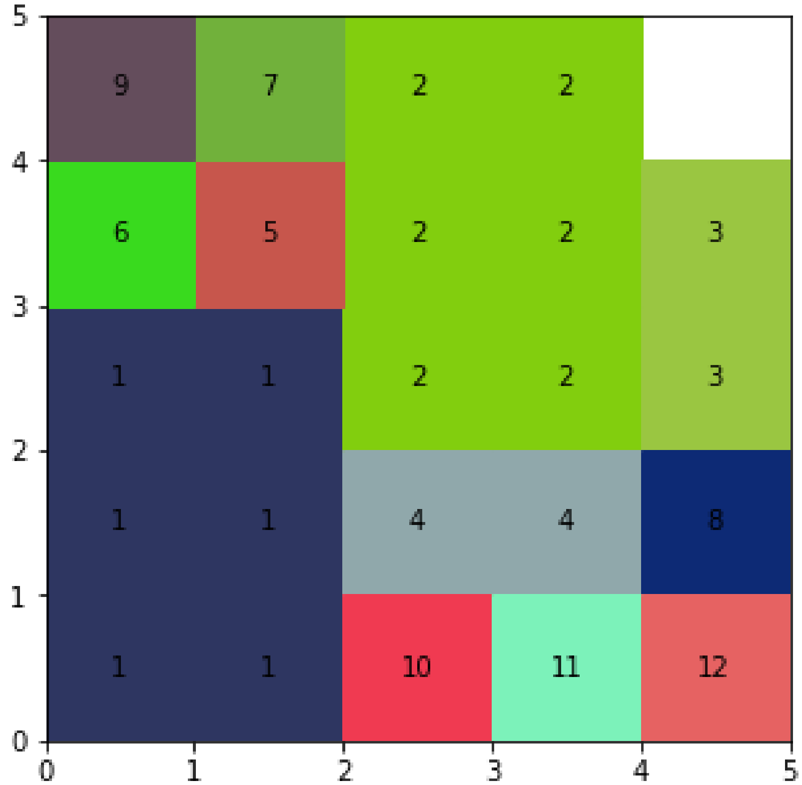

9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5



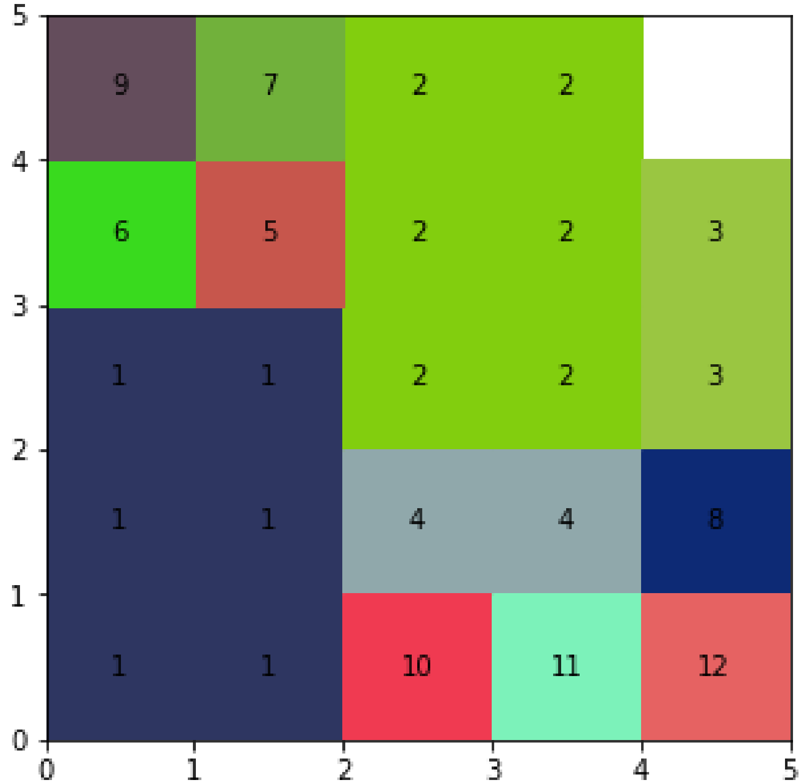
5
4
3
2
1
0
0
1
2
3
4
5
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12



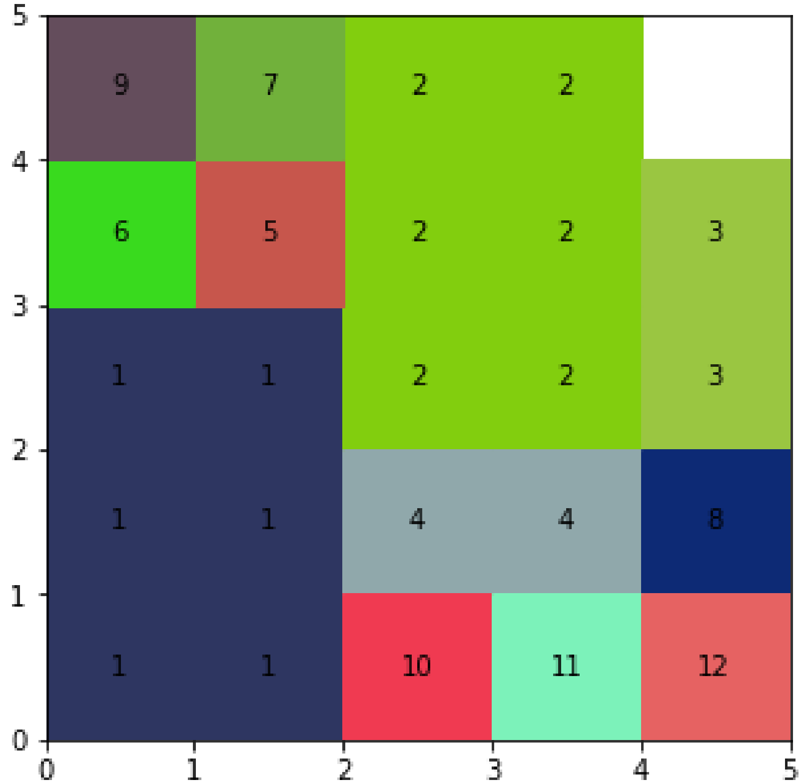
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5



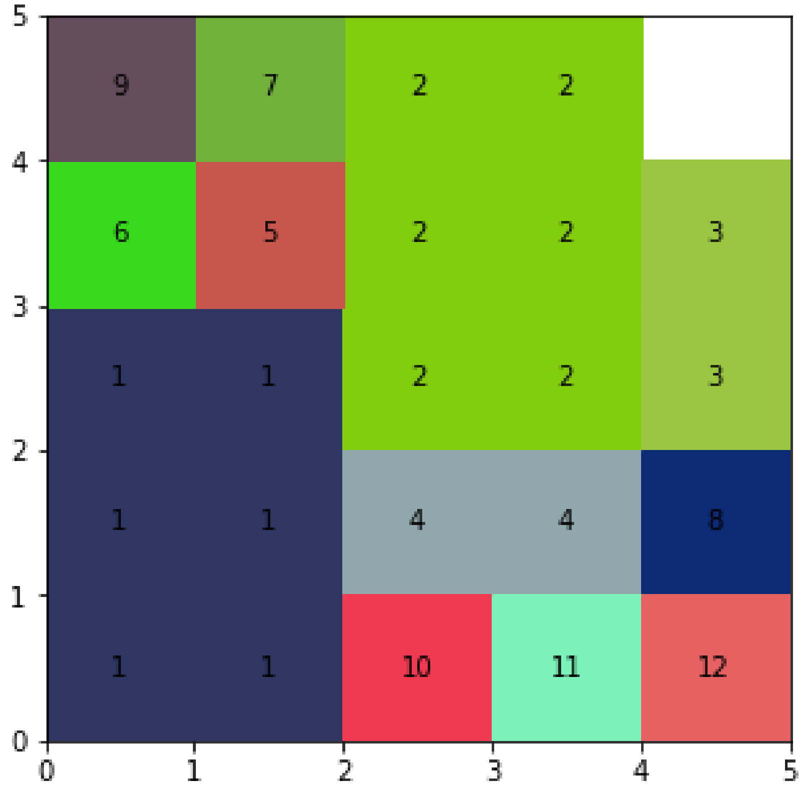
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5



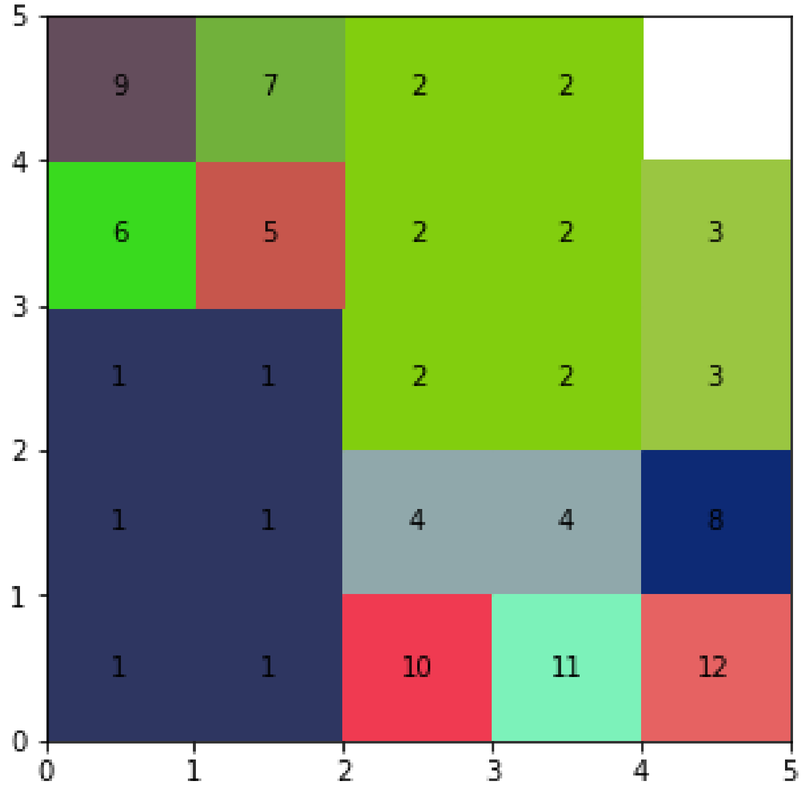
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5
0
1
2
3
4
5

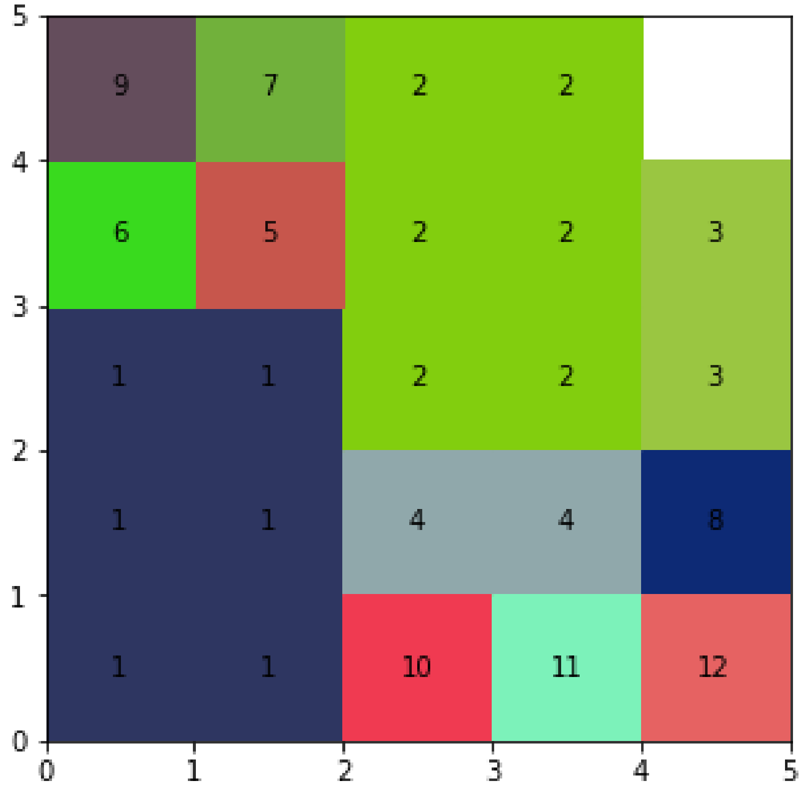


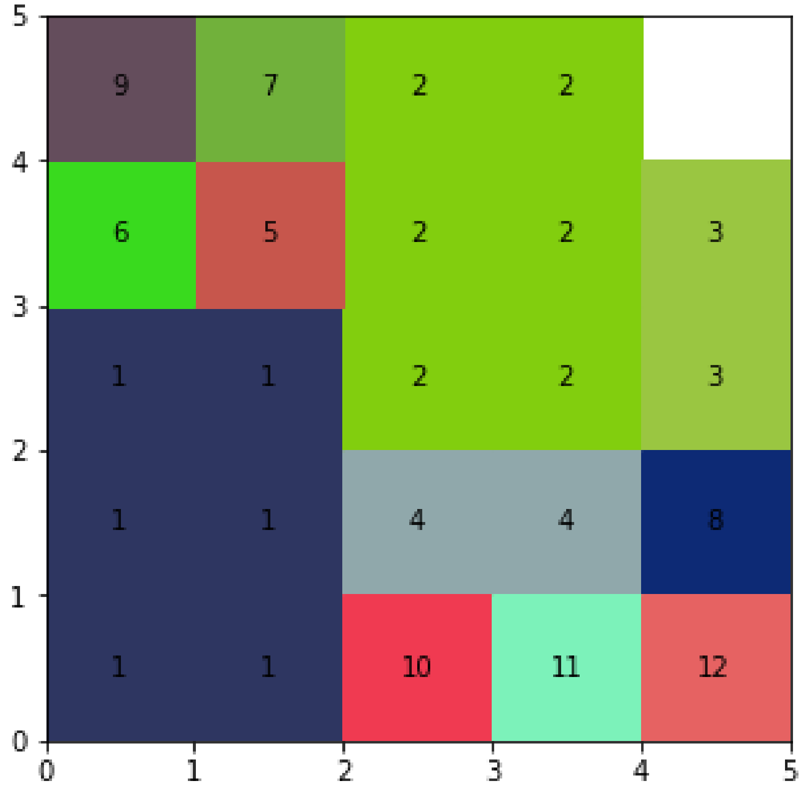
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5



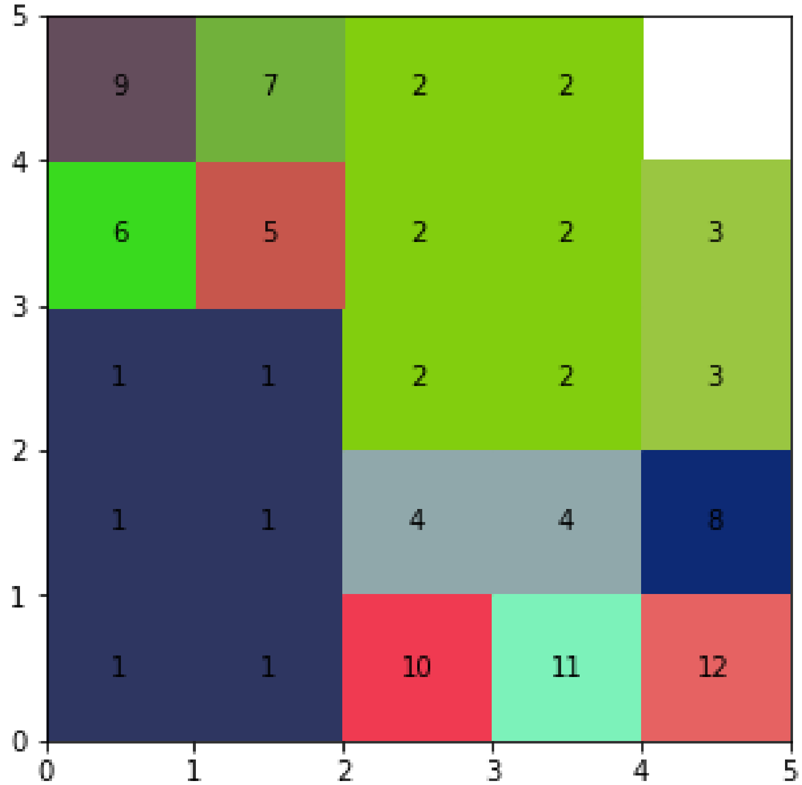
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5

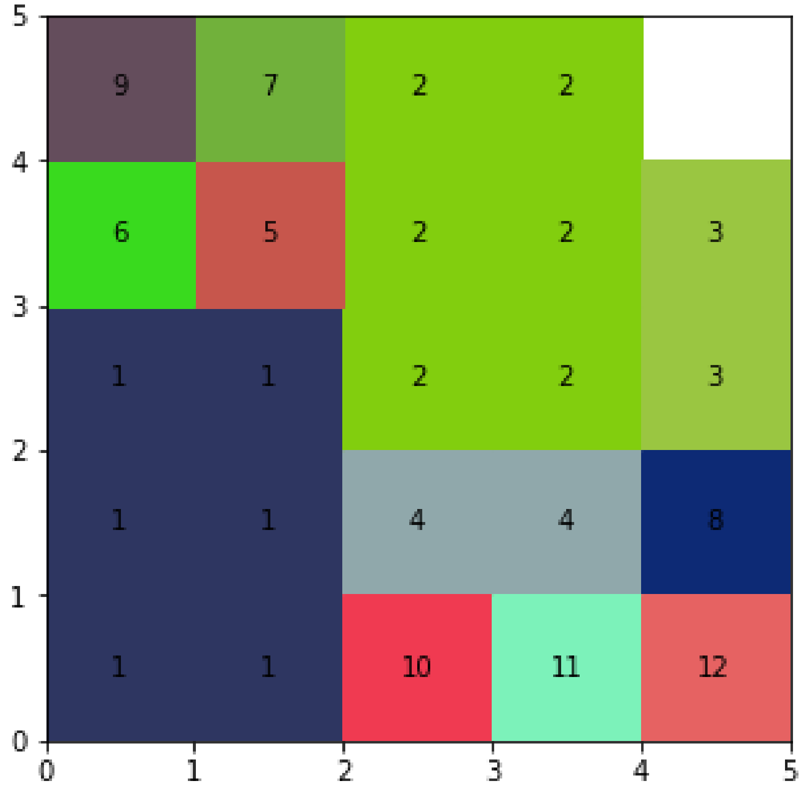

9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5


9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5

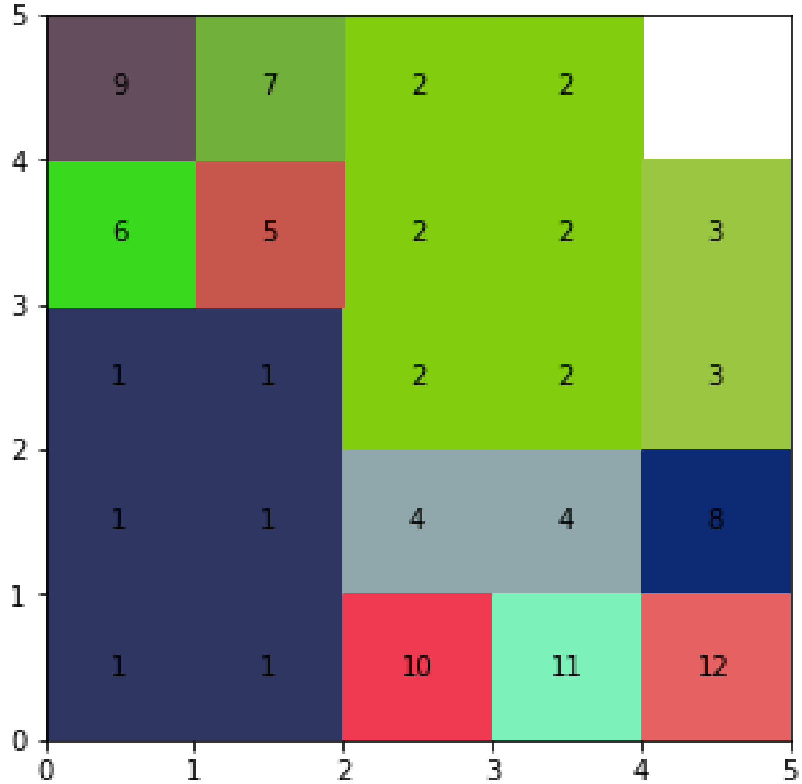


9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5

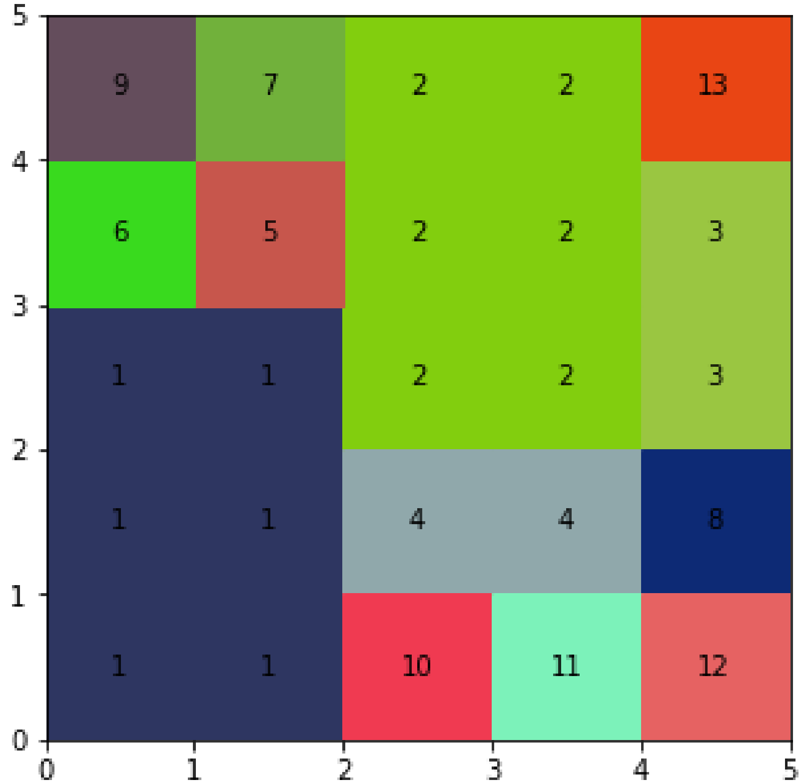

9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5

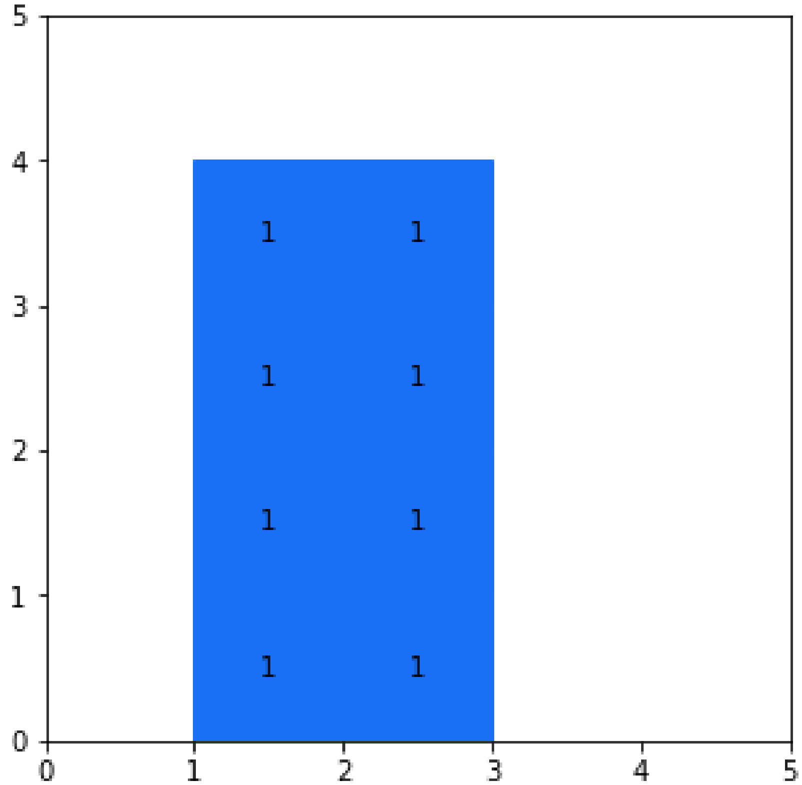


9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5

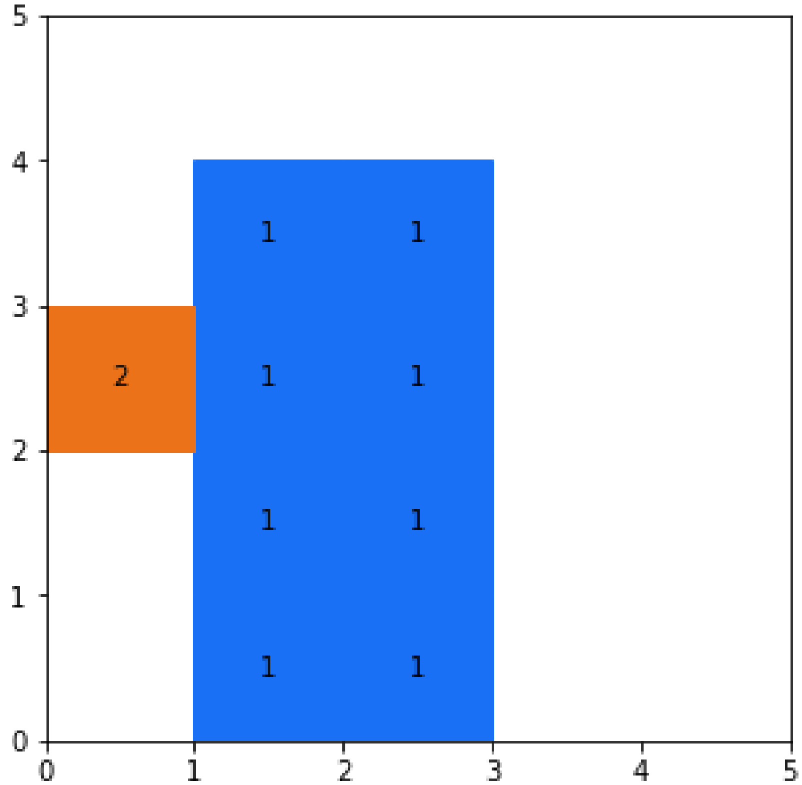


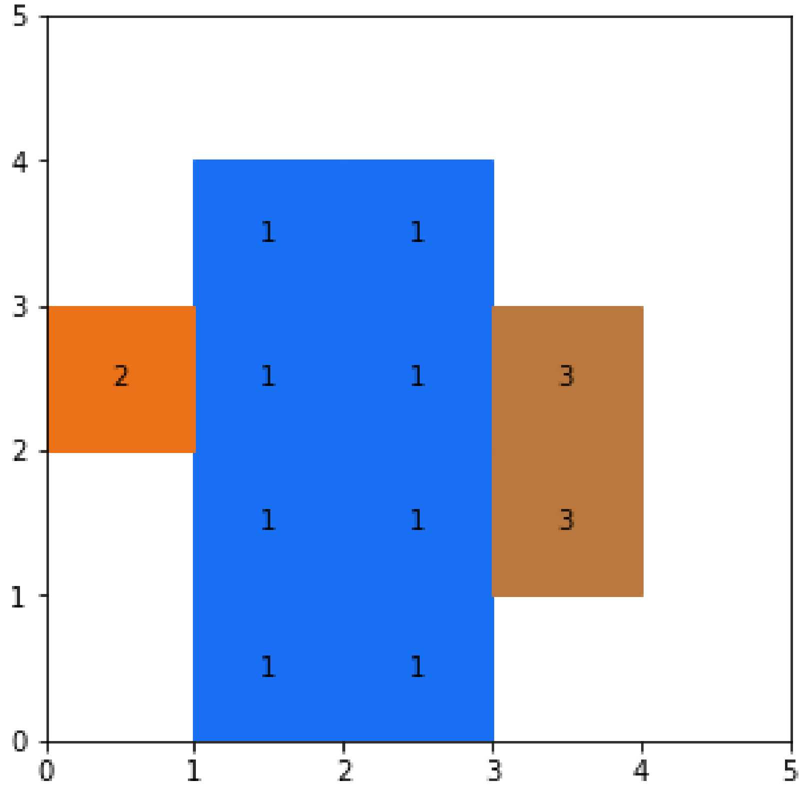
9
7
2
2
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5

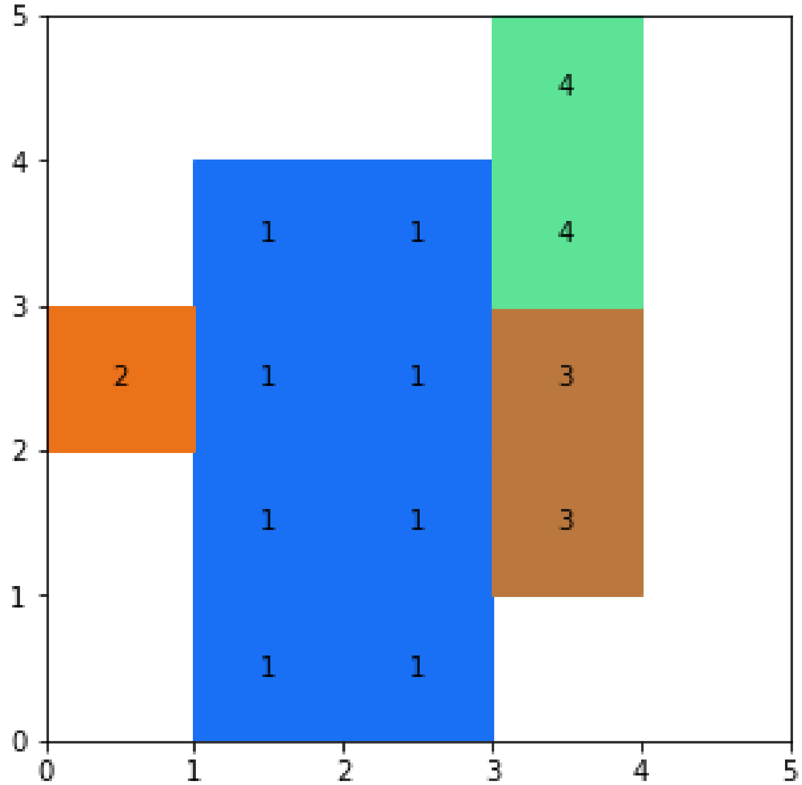

9
7
2
2
13
6
5
2
2
3
1
1
2
2
3
1
1
4
4
8
1
1
10
11
12
0
1
2
3
4
5



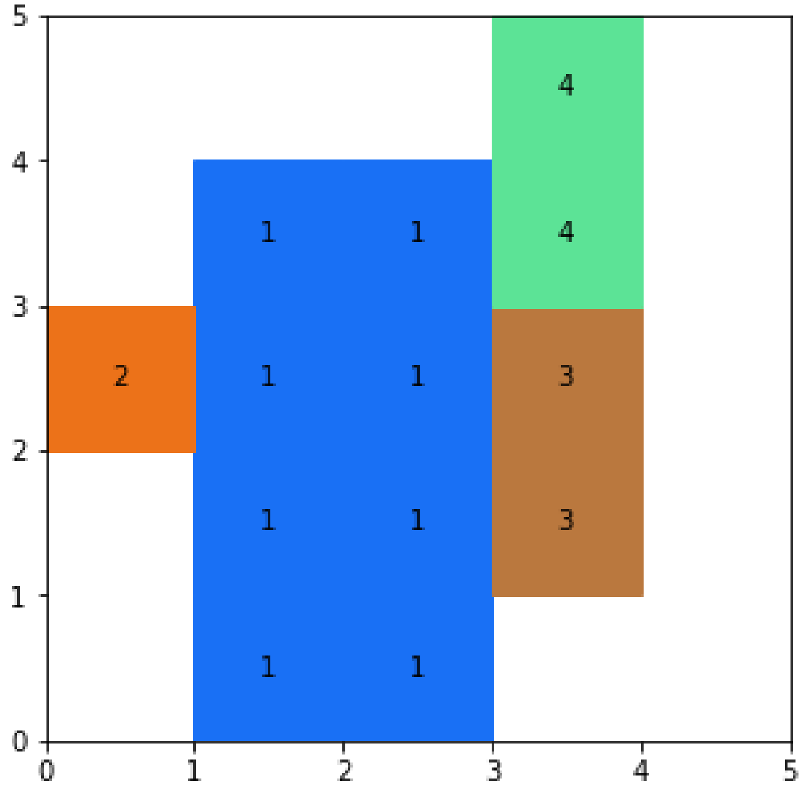
5
4
3
2
1
0
1 1
1 1
1 1
1 1
0 1 2 3 4 5


5
4
3
2
1
0
0
1
2
3
4
5
1
1
1
1
1
1
1
1
2

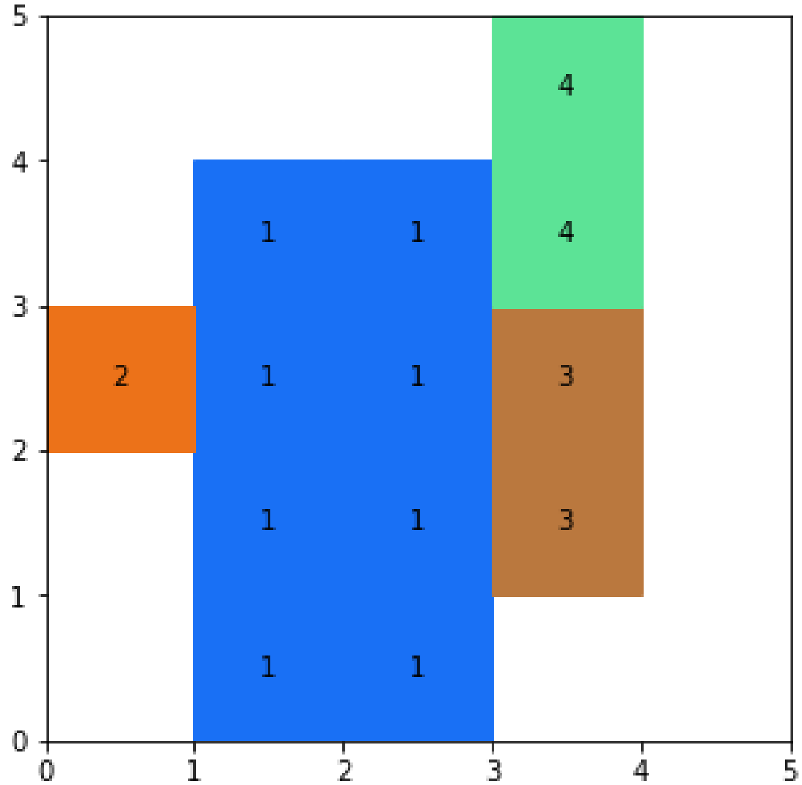

1
2
3
4
5
0
1
2
3
4
5
1
1
2
1
1
3
1
1
3
1
1



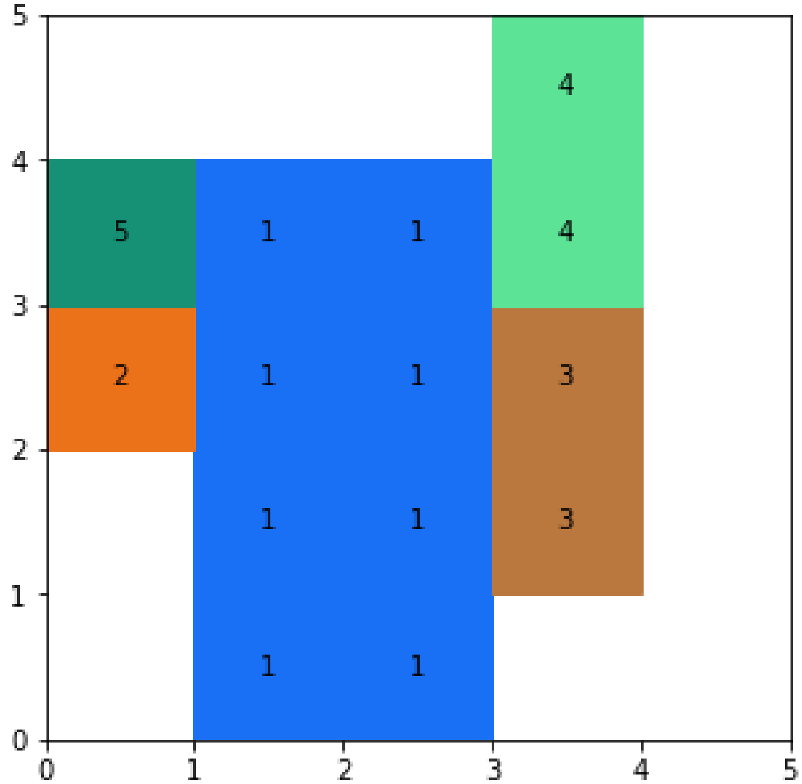
5
4
3
2
1
0
0
1
2
3
4
5
4
4
1
1
2
1
1
3
1
1
3
1
1



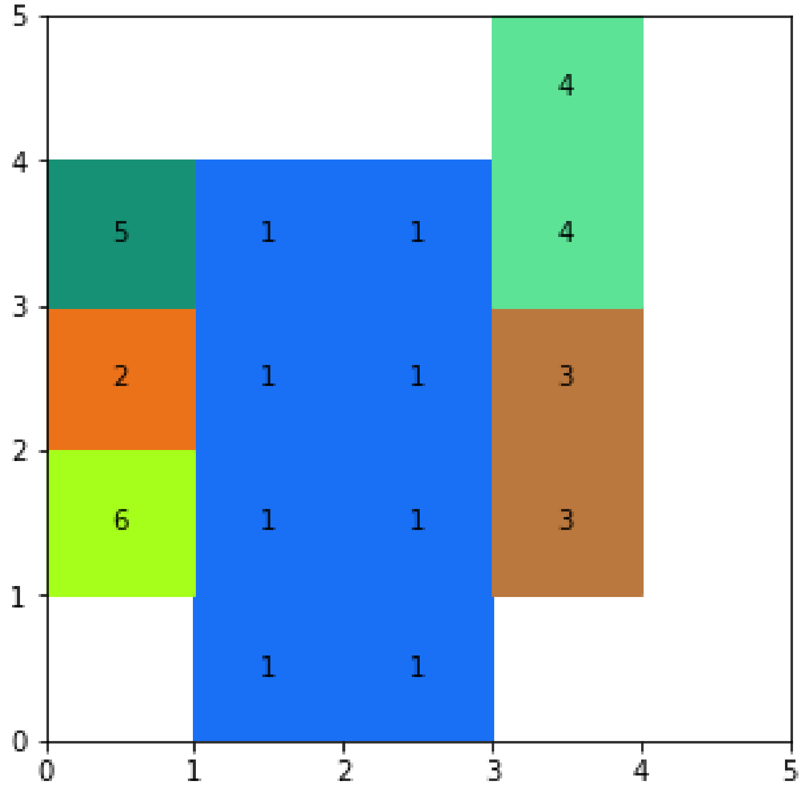
5
4
3
2
1
0
0
1
2
3
4
5
4
4
1
1
2
1
1
3
1
1
3
1
1



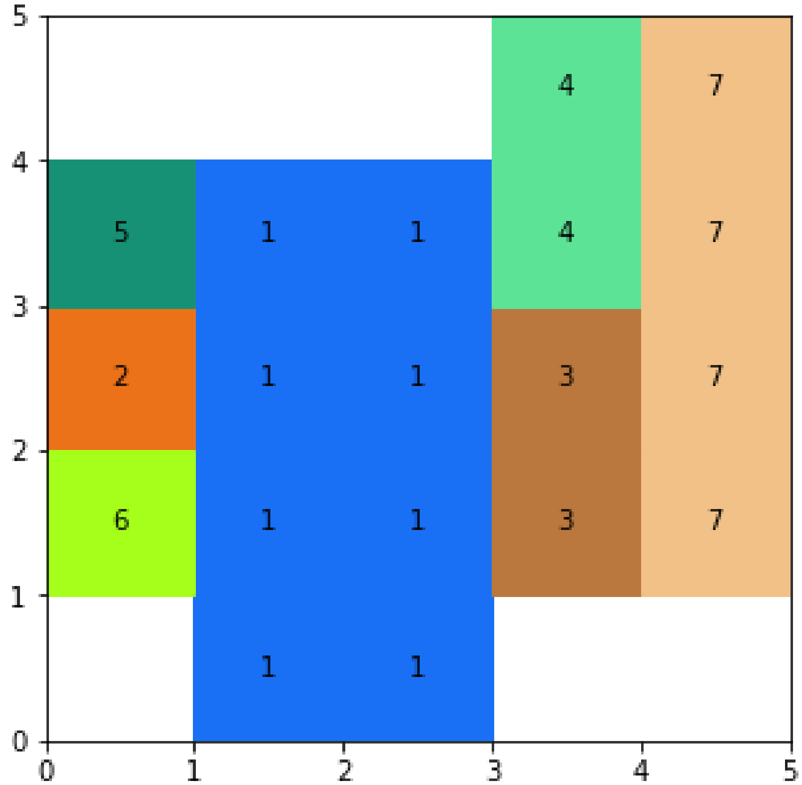
4
4
1
1
2
1
1
3
1
1
3
1
1
0
1
2
3
4
5

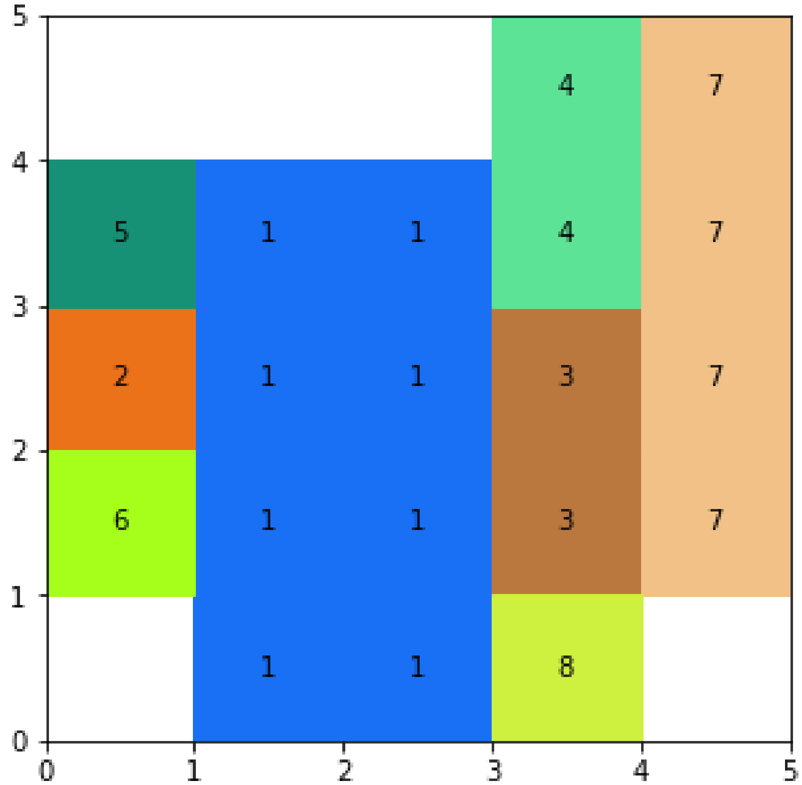


5
4
3
2
1
0
0
1
2
3
4
5
4
4
5
1
1
2
1
1
3
1
1
3
1
1

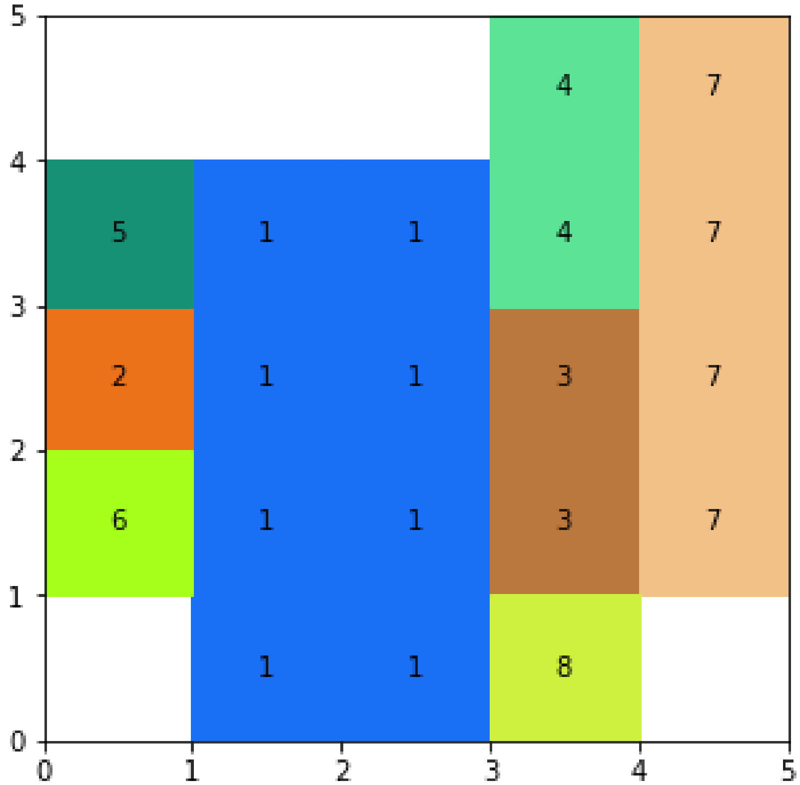


5
4
3
2
1
0
0 1 2 3 4 5
4
5 1 1 4
2 1 1 3
6 1 1 3
1 1

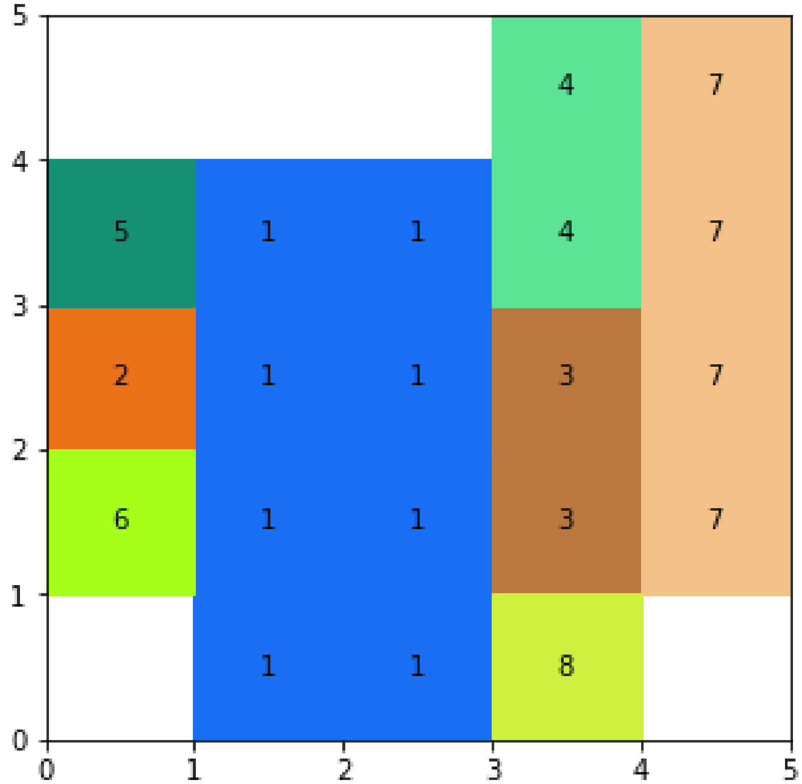

4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
0
1
2
3
4
5



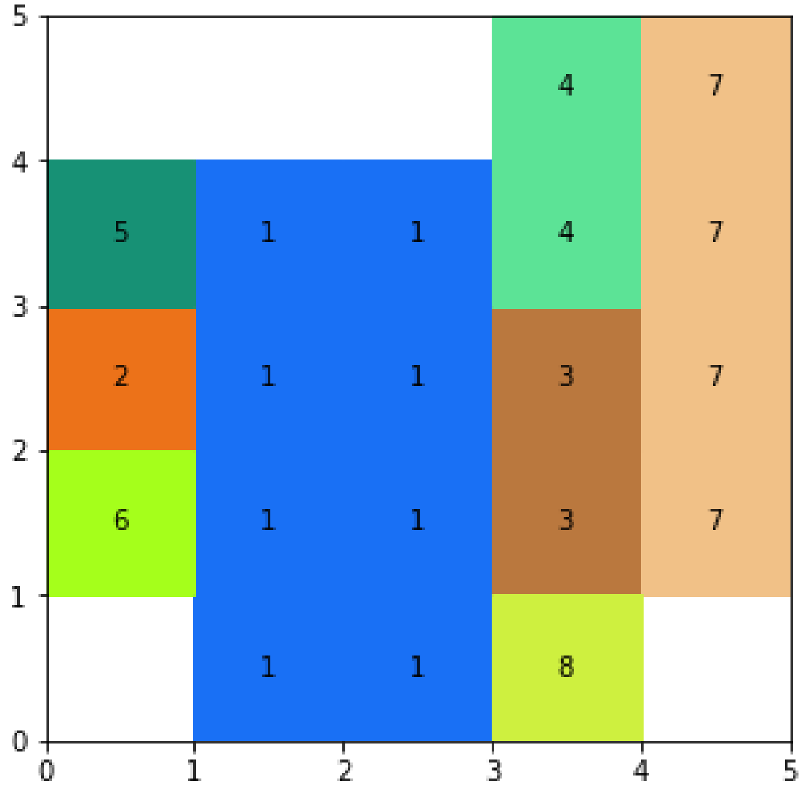
0
1
2
3
4
5
4 7
5 1 1 4 7
2 1 1 3 7
6 1 1 3 7
1 1 8

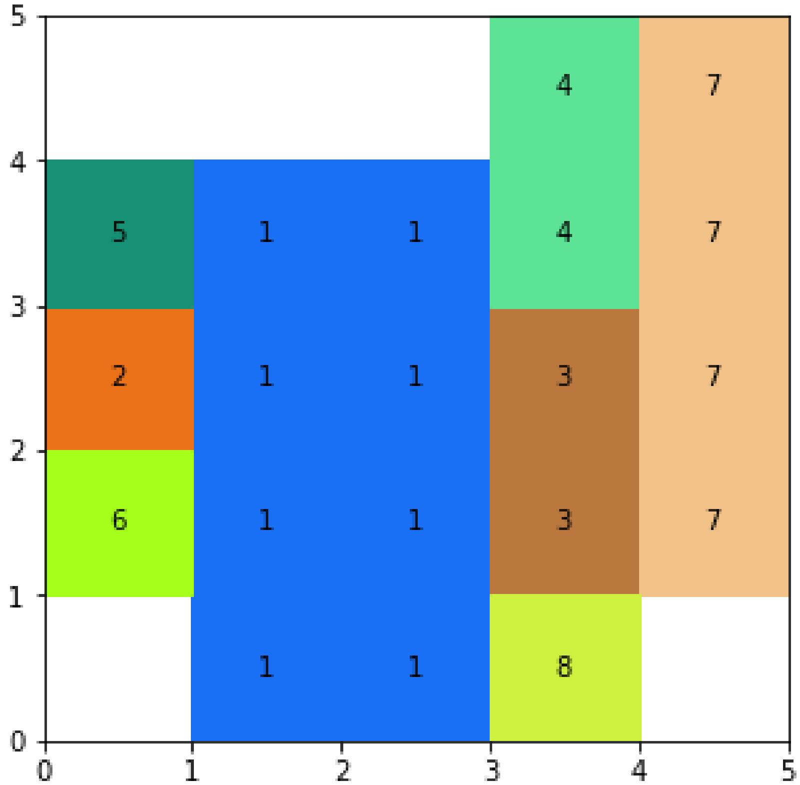


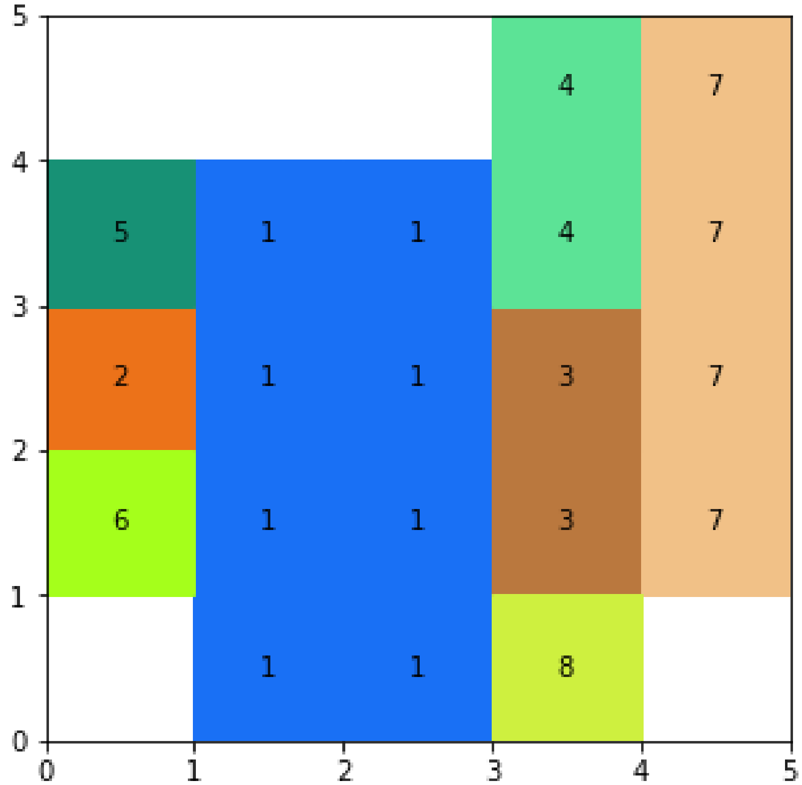
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0
1
2
3
4
5



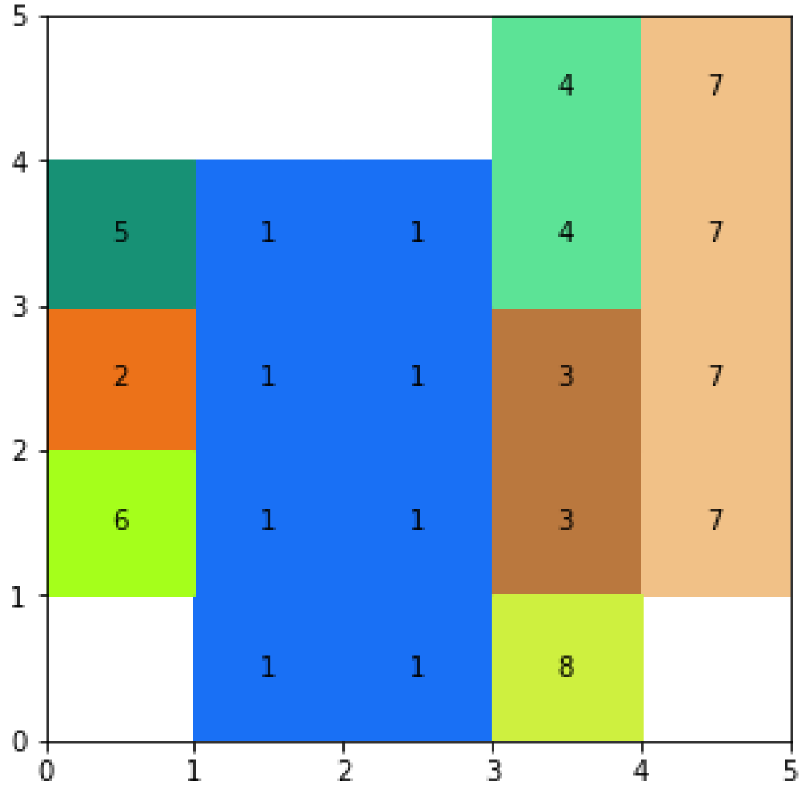
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0
1
2
3
4
5


4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0 1 2 3 4 5
0 1 2 3 4 5

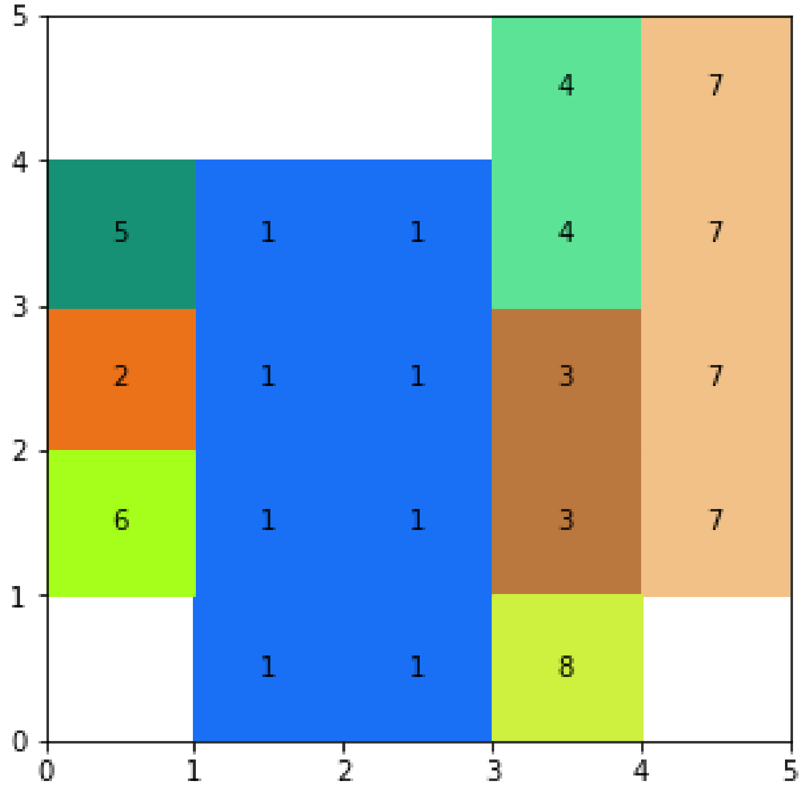

4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0
1
2
3
4
5
0
1
2
3
4
5

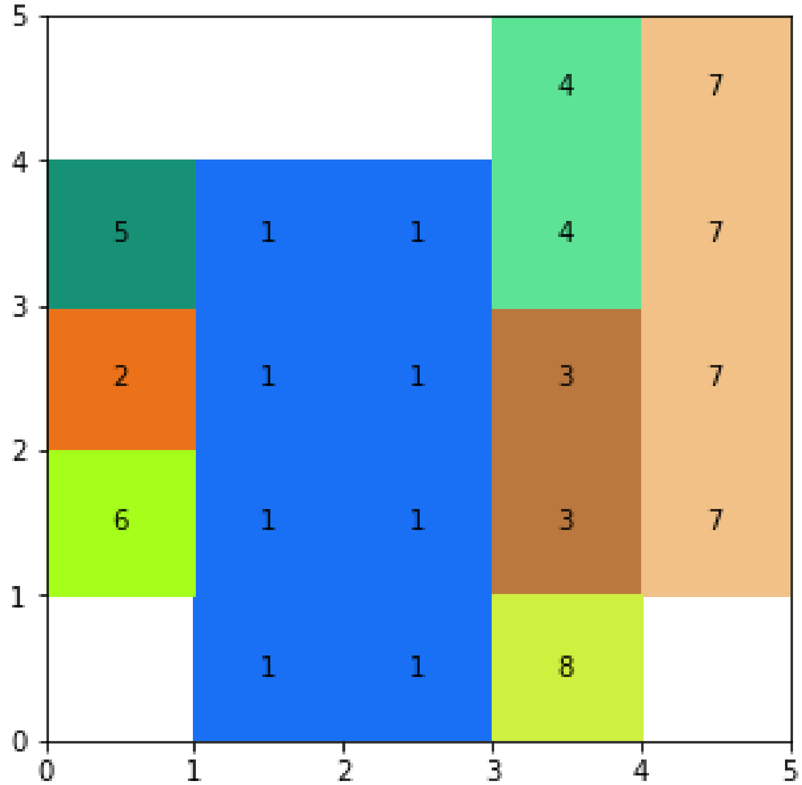


4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0
1
2
3
4
5

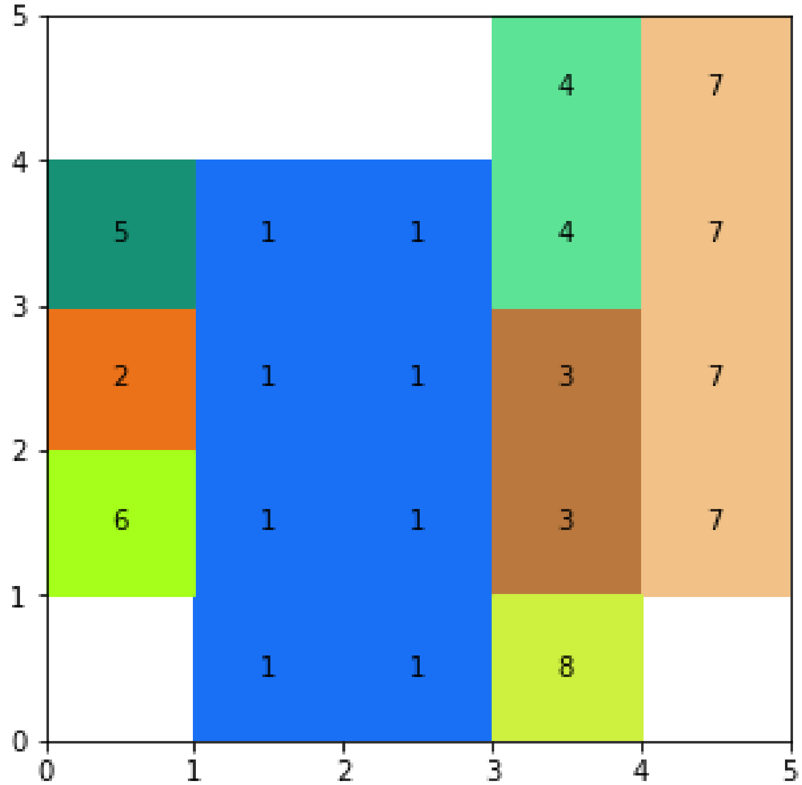


5
4
3
2
1
0
0
1
2
3
4
5
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8

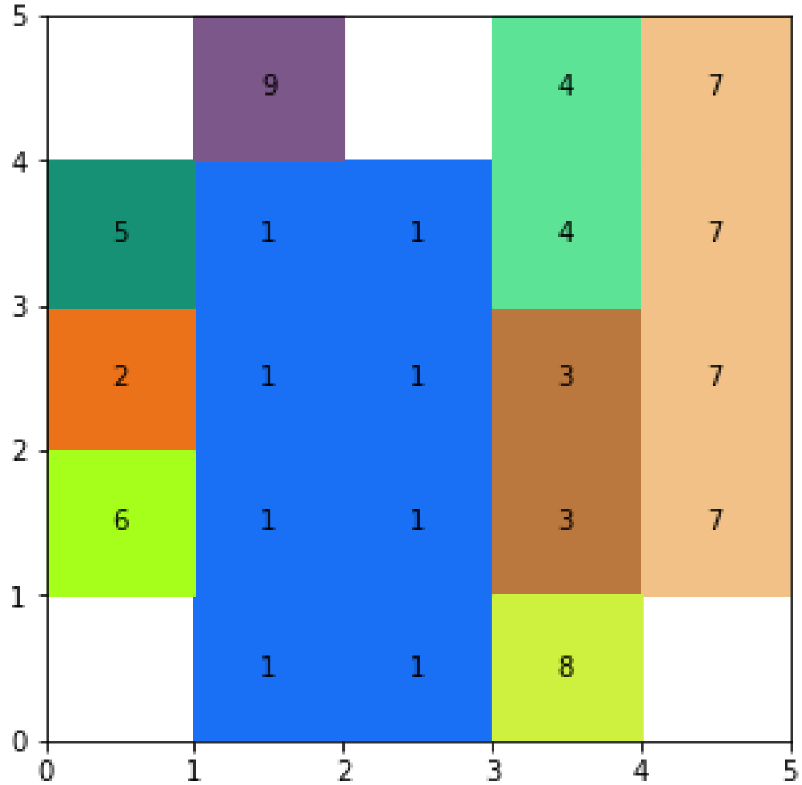

5
4
3
2
1
0
0
1
2
3
4
5
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8

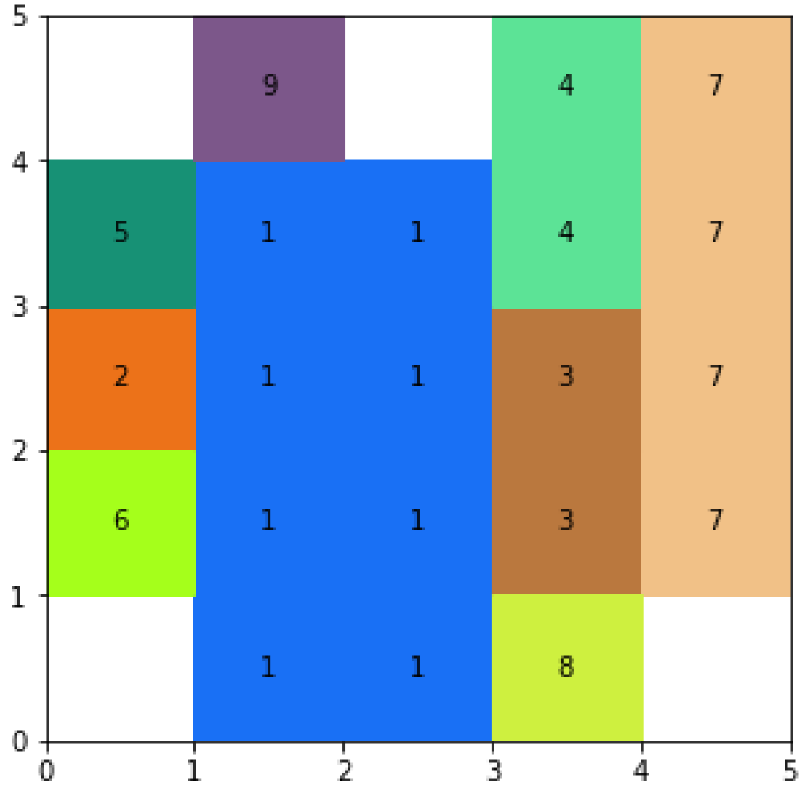


5
4
3
2
1
0
0
1
2
3
4
5
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8

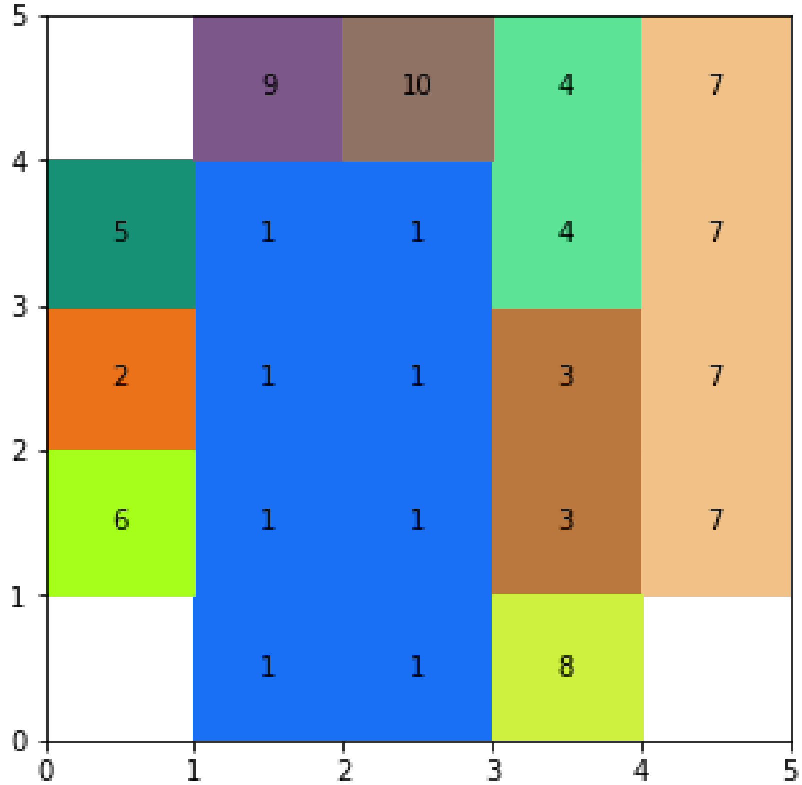


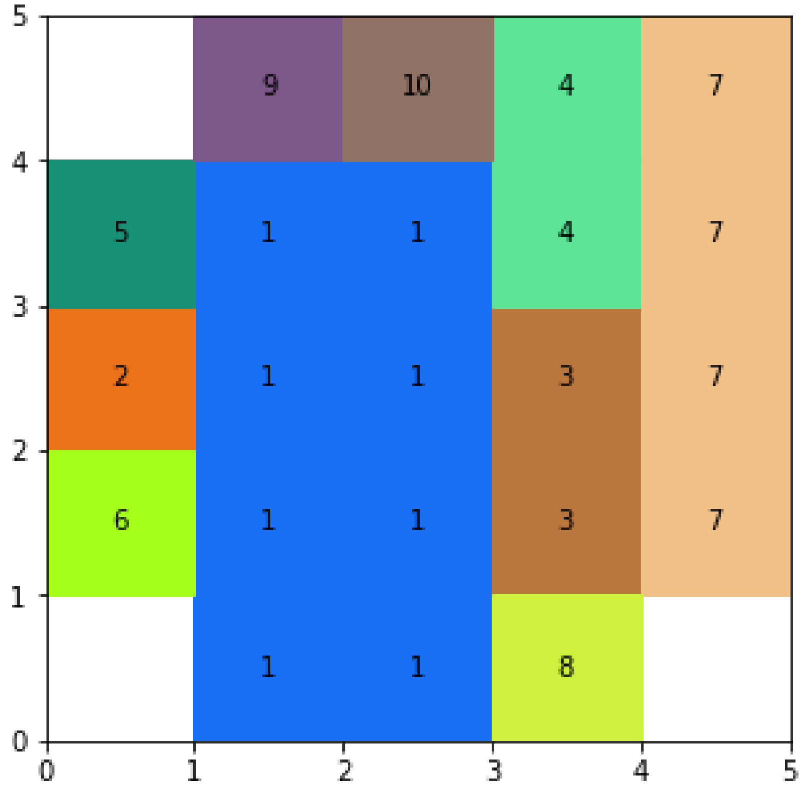
5
4
3
2
1
0
0
1
2
3
4
5
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8


9
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0
1
2
3
4
5

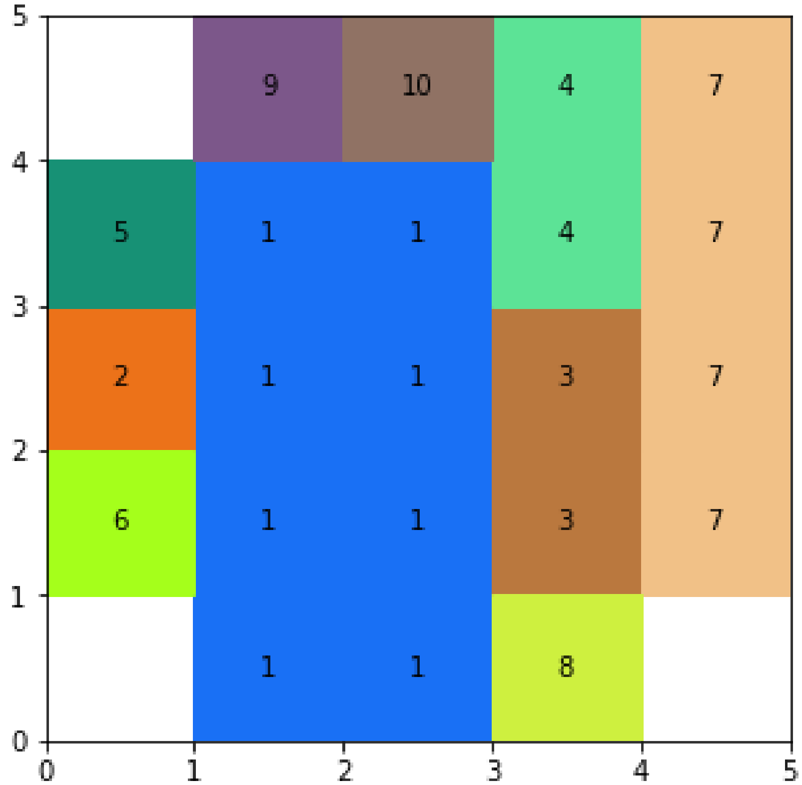


5
4
3
2
1
0
0
1
2
3
4
5
9
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8

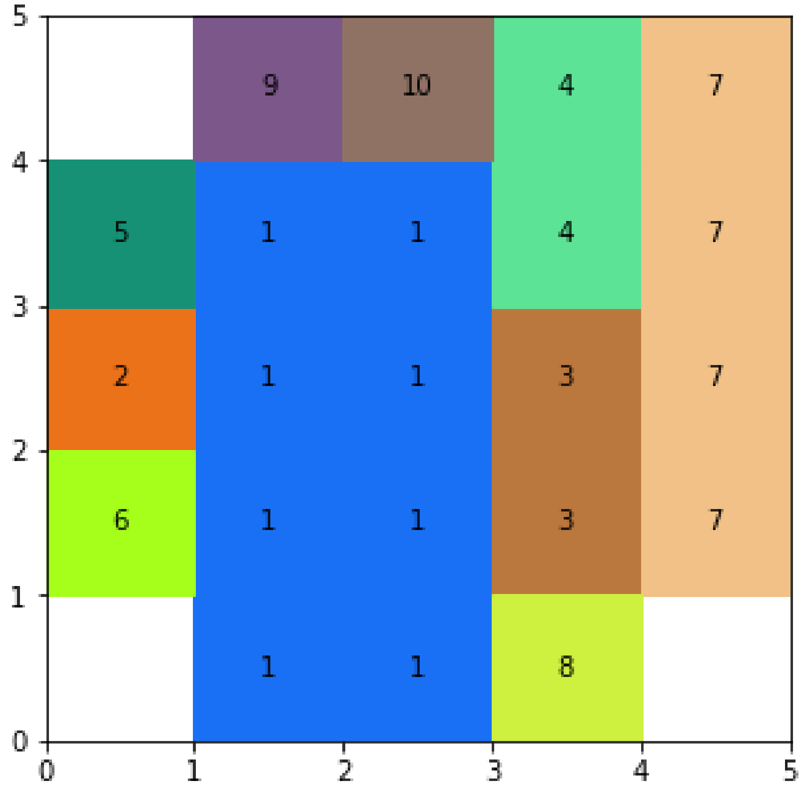

9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0
1
2
3
4
5



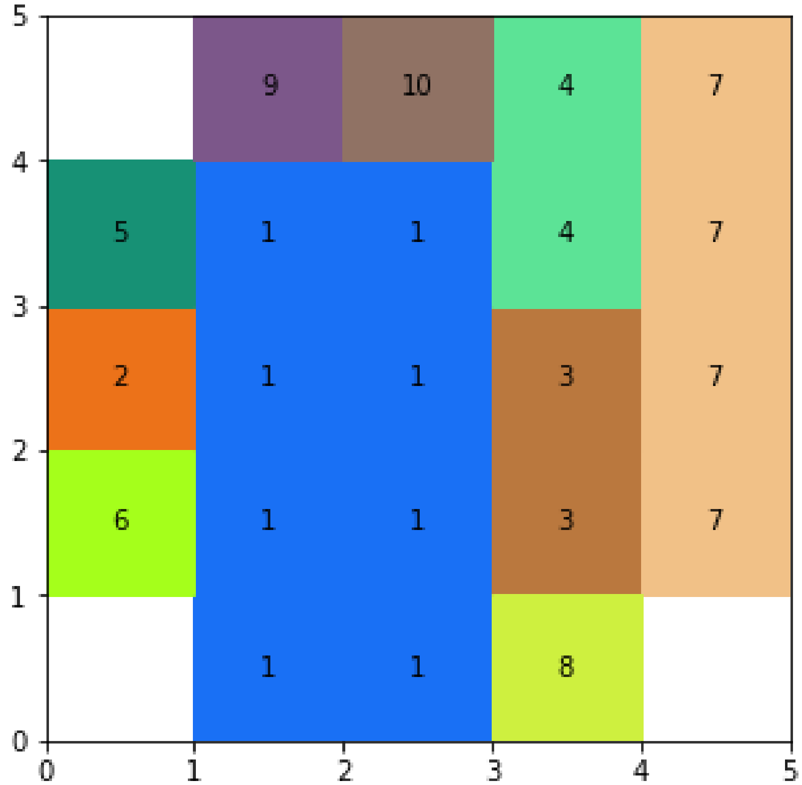
9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8

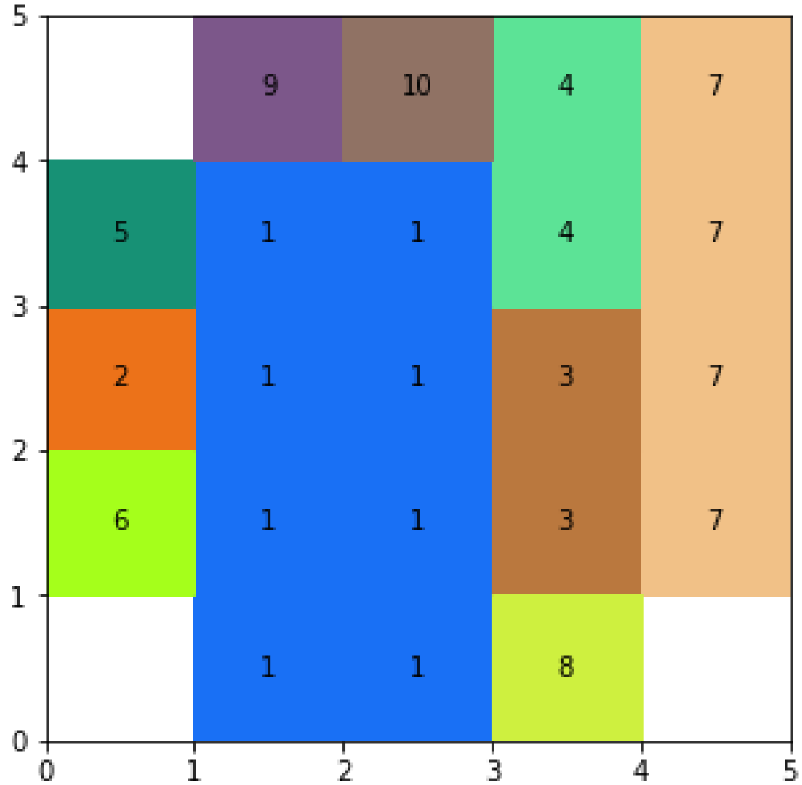


9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0
1
2
3
4
5

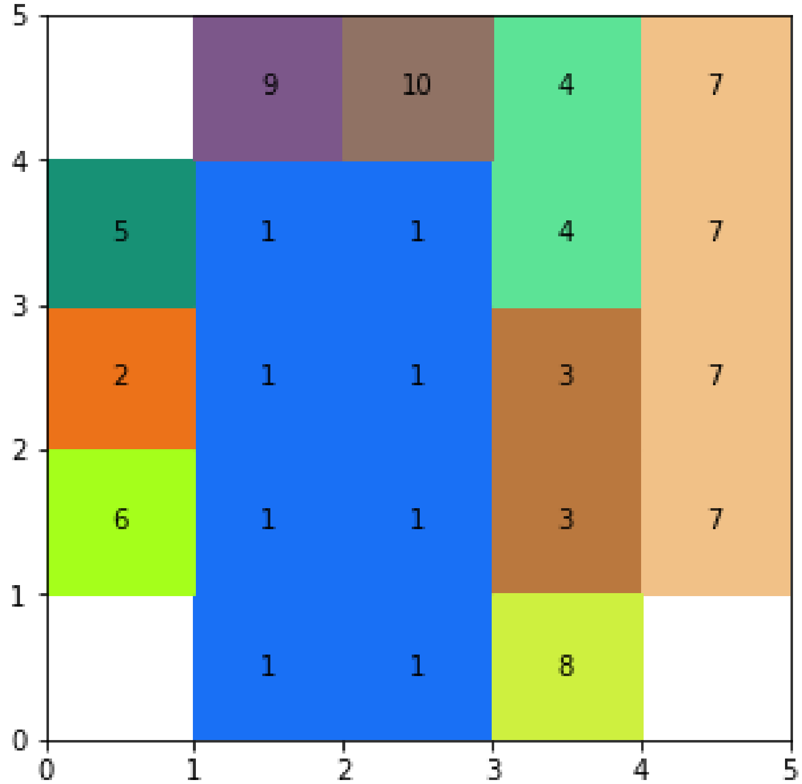


9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0
1
2
3
4
5

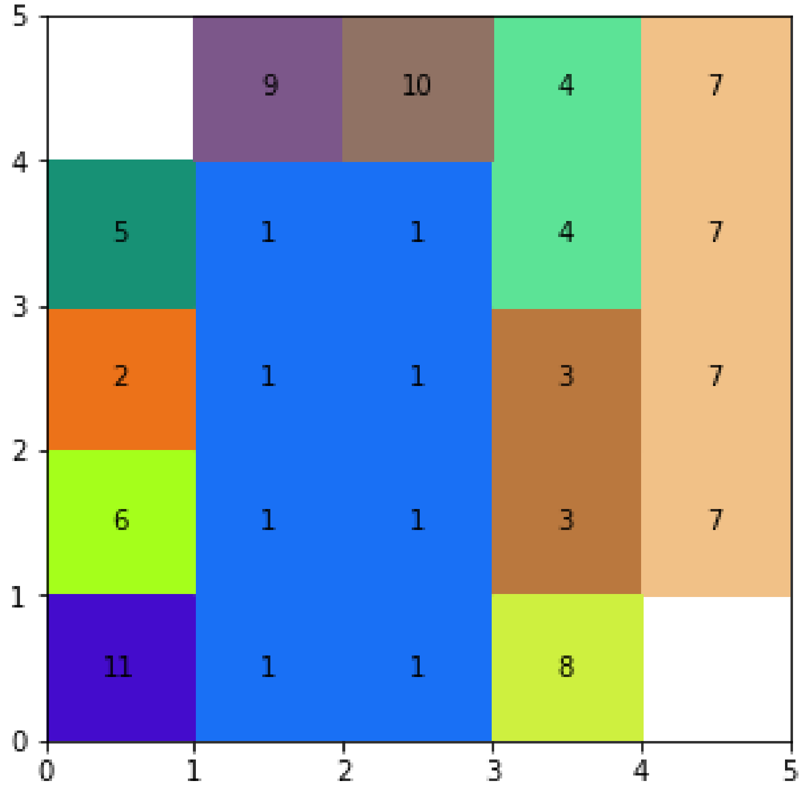

9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0
1
2
3
4
5



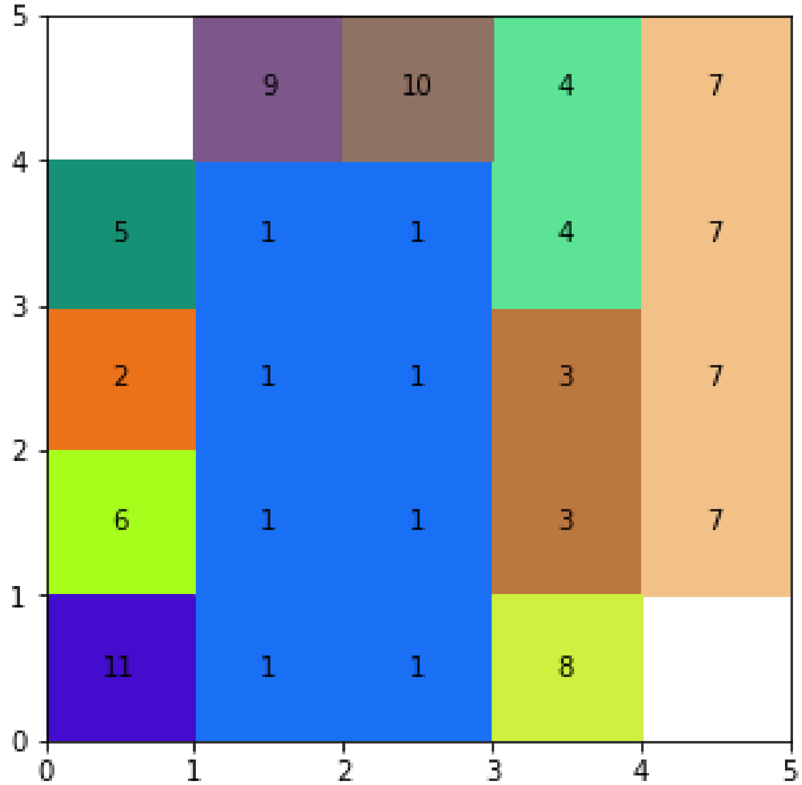
9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0
1
2
3
4
5

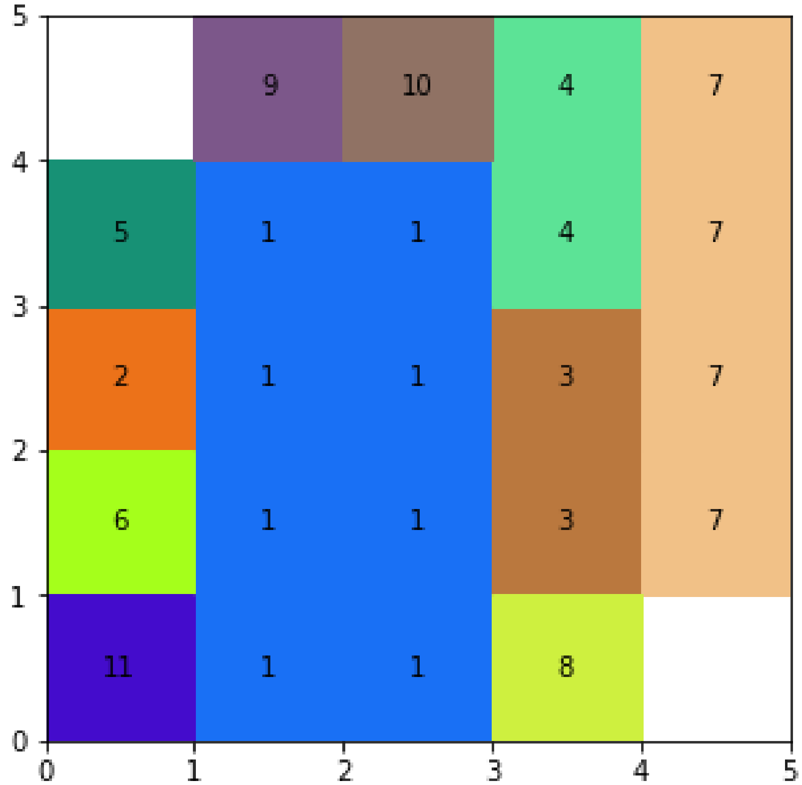


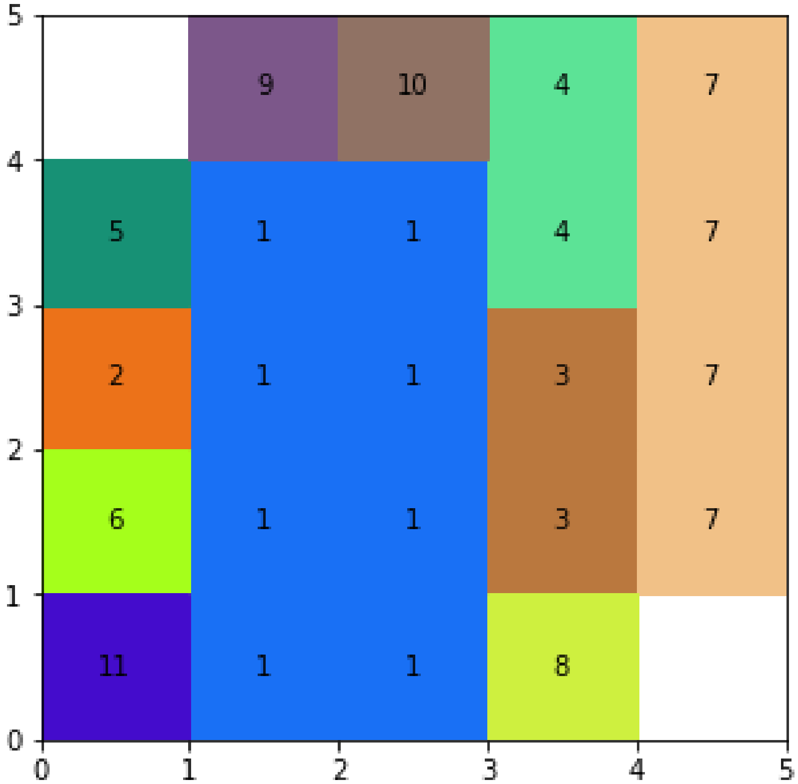
9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
1
1
8
0
1
2
3
4
5



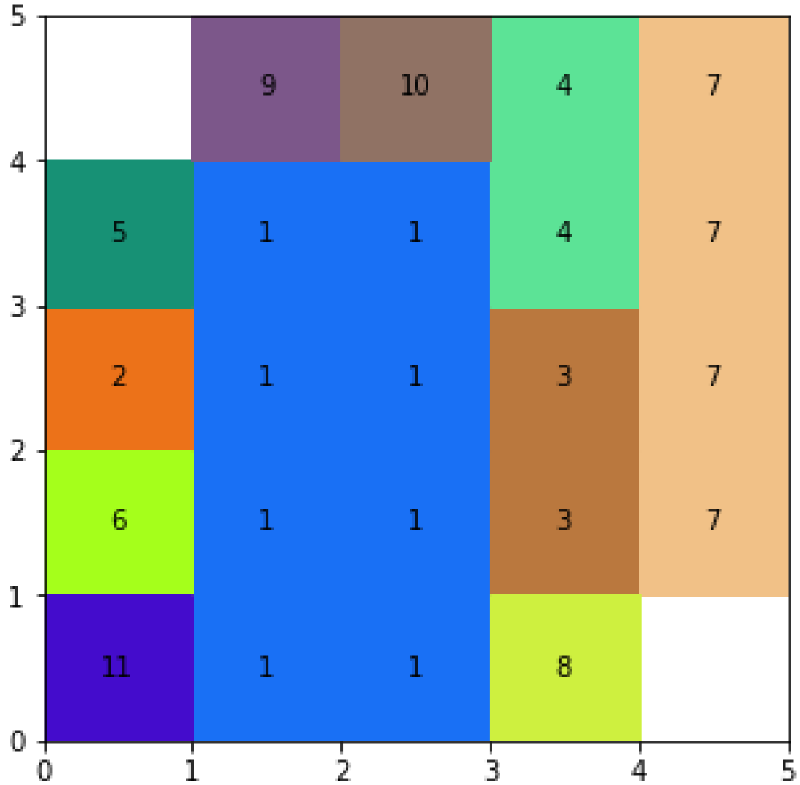
9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
0
1
2
3
4
5
0
1
2
3
4
5


9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
0
1
2
3
4
5

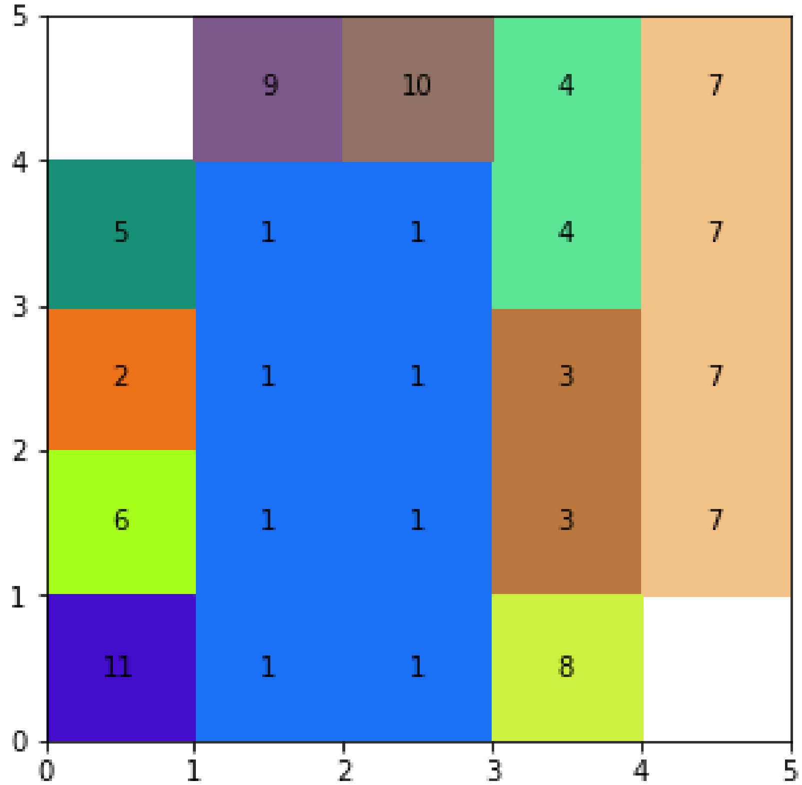

9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
0
1
2
3
4
5
0
1
2
3
4
5



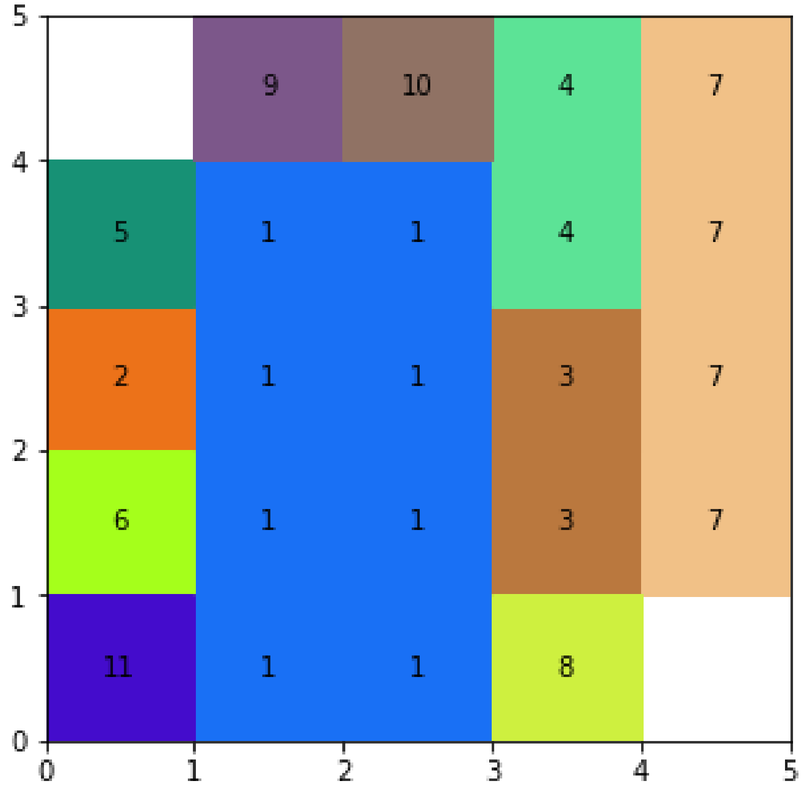
5
4
3
2
1
0
0
1
2
3
4
5
9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8



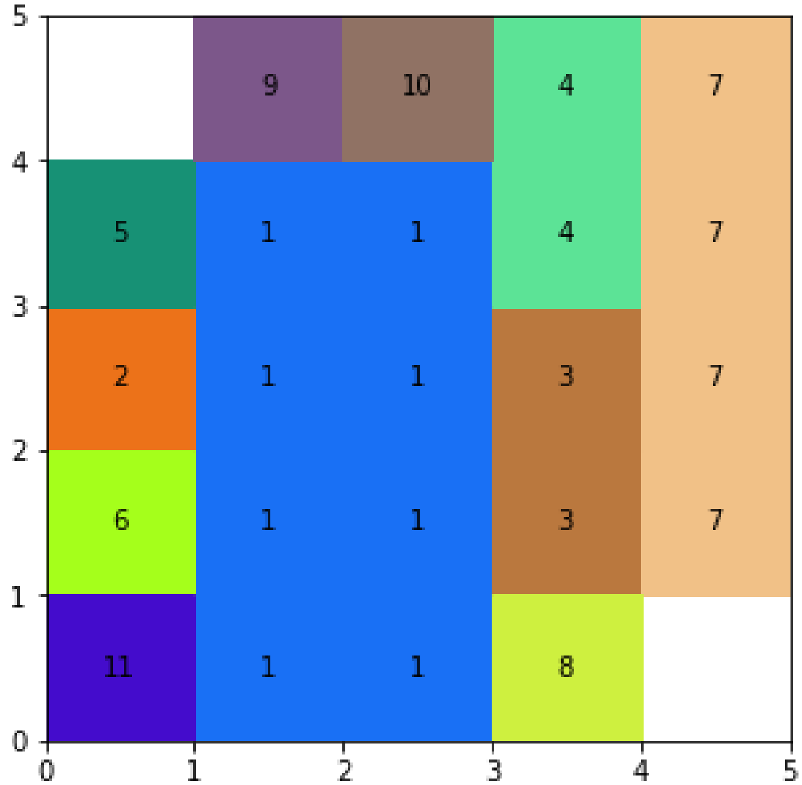
9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
0
1
2
3
4
5



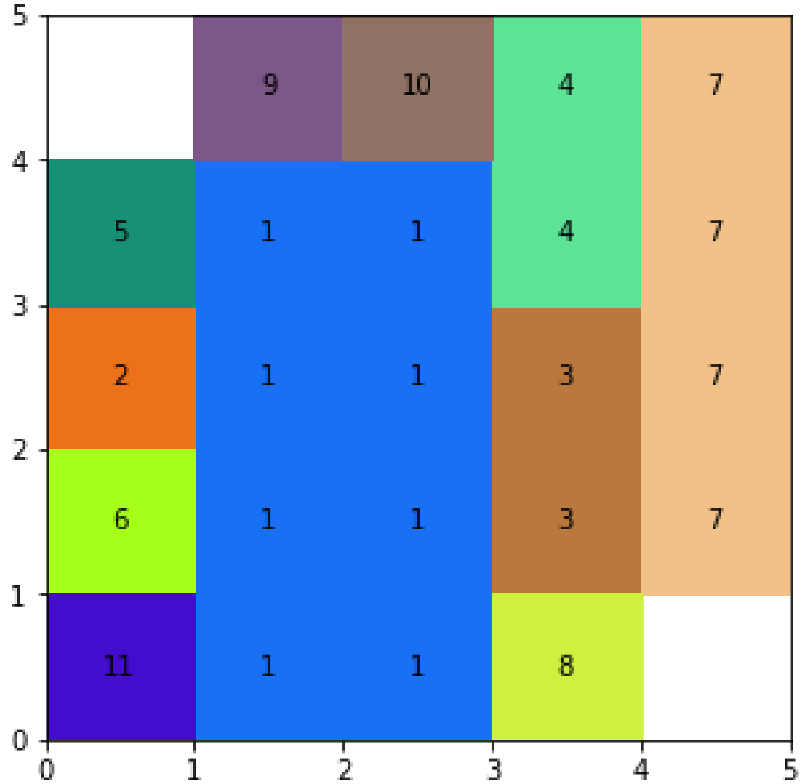
9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
0
1
2
3
4
5

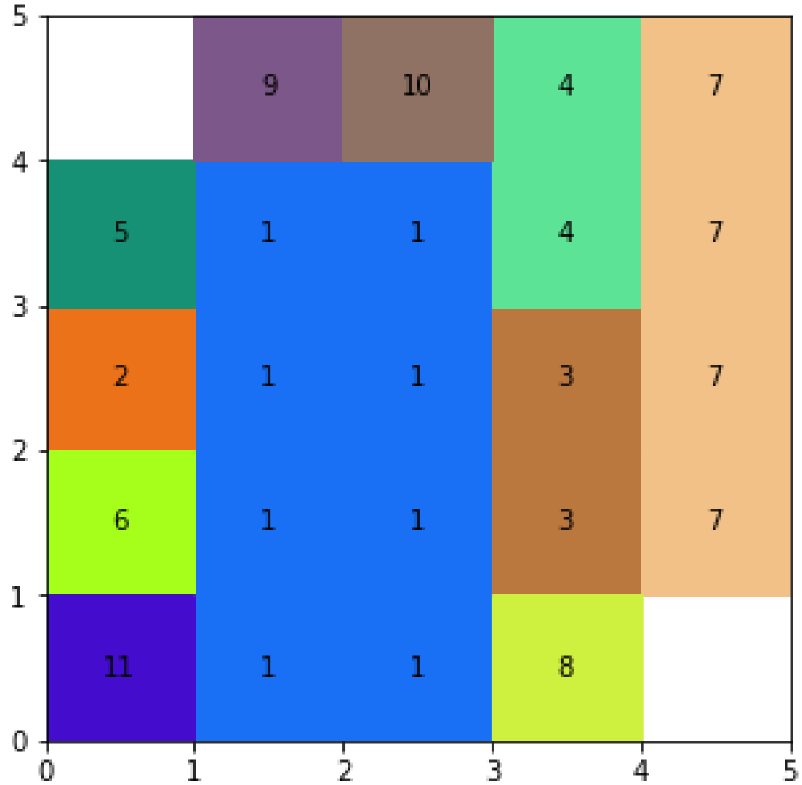


9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
0
1
2
3
4
5

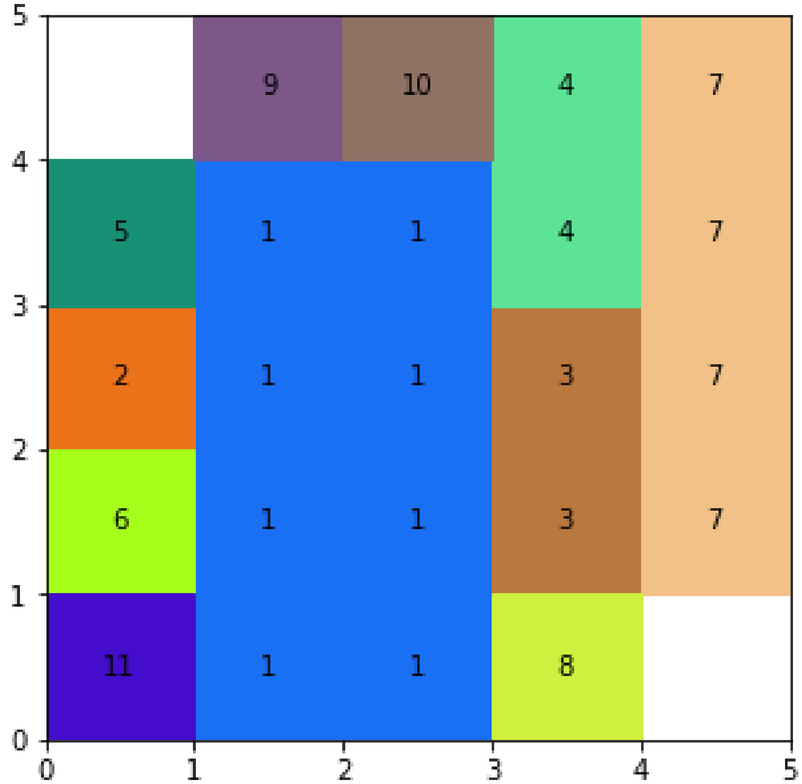


9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
0
1
2
3
4
5

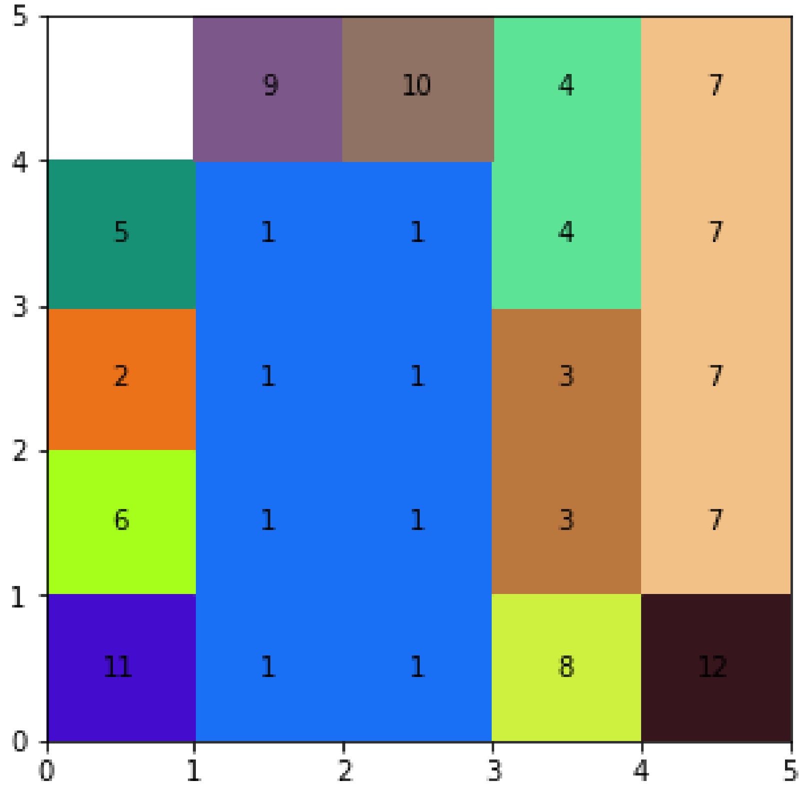

9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
0
1
2
3
4
5



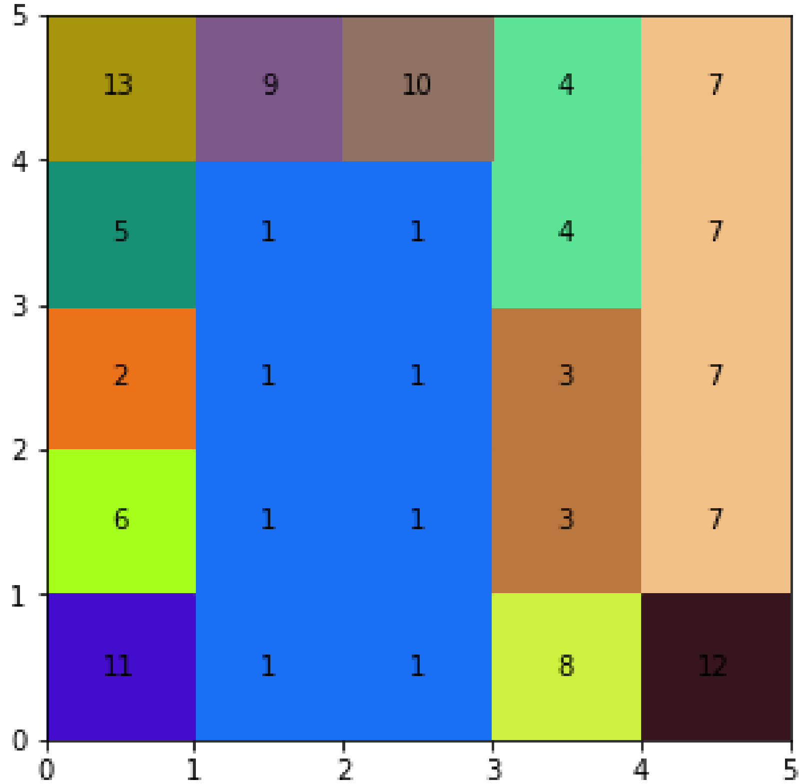
9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
0
1
2
3
4
5



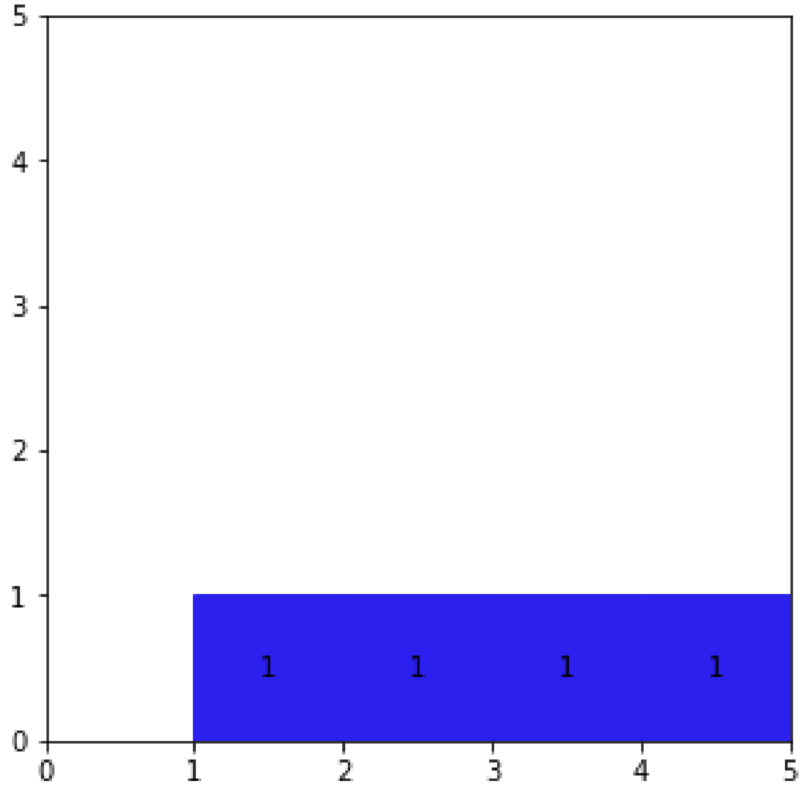
9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
0
1
2
3
4
5



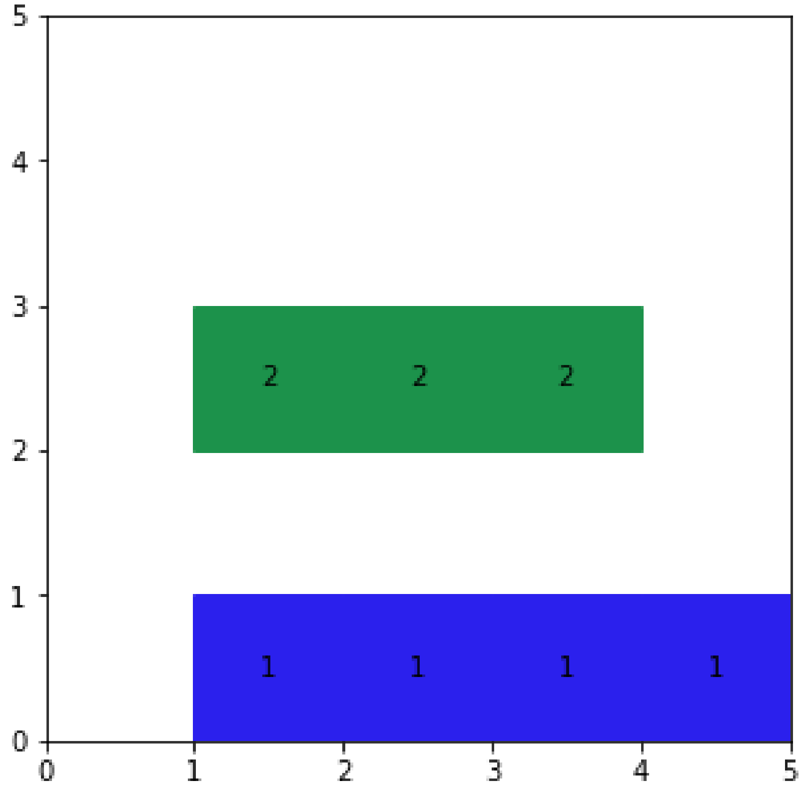
5
4
3
2
1
0
0
1
2
3
4
5
9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
12



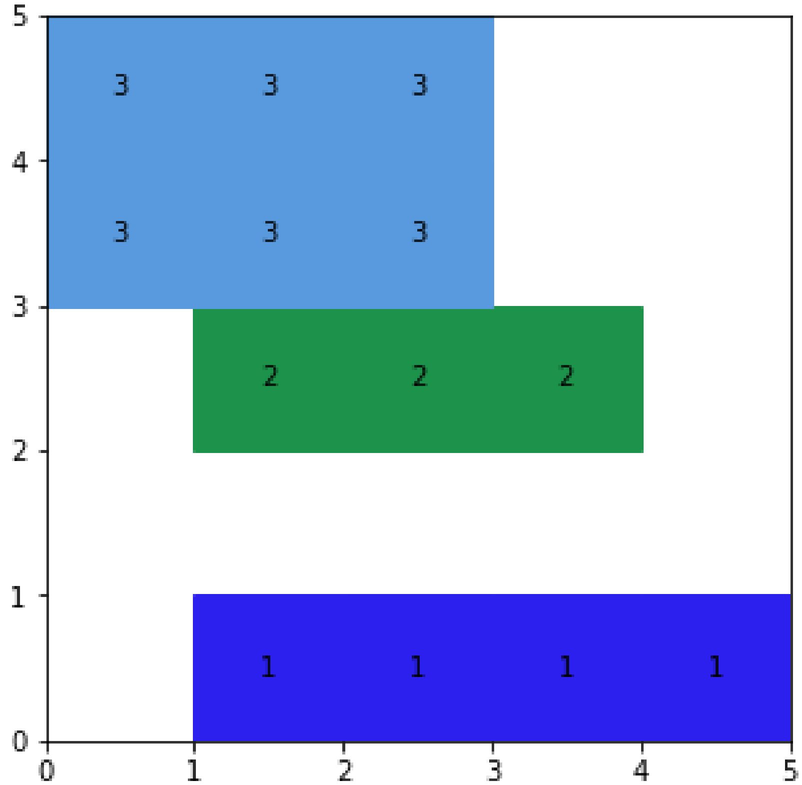
13
9
10
4
7
5
1
1
4
7
2
1
1
3
7
6
1
1
3
7
11
1
1
8
12
0
1
2
3
4
5



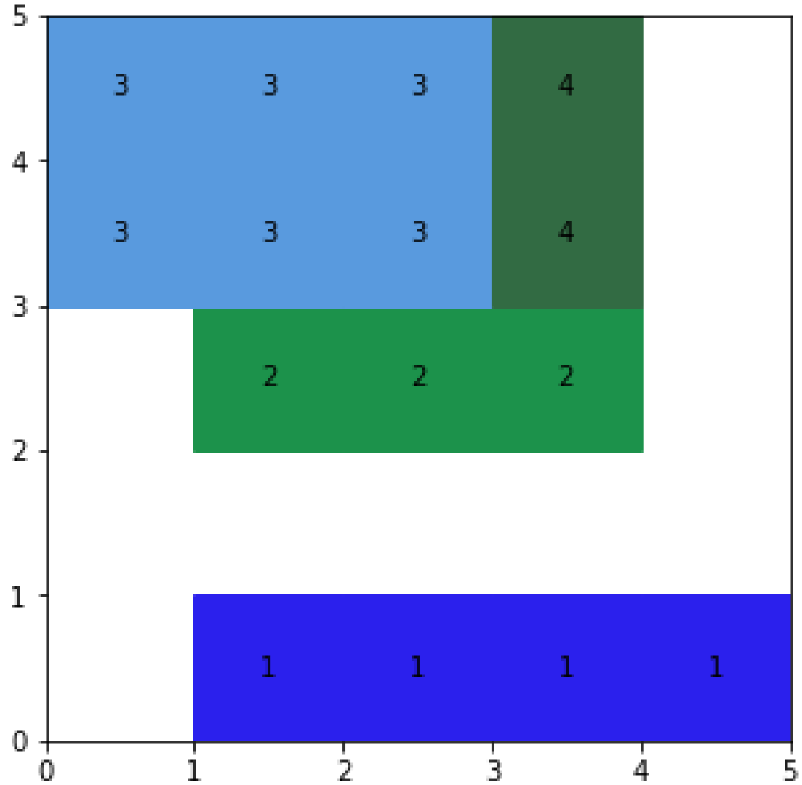
5
4
3
2
1
0
1 1 1 1
0 1 2 3 4 5



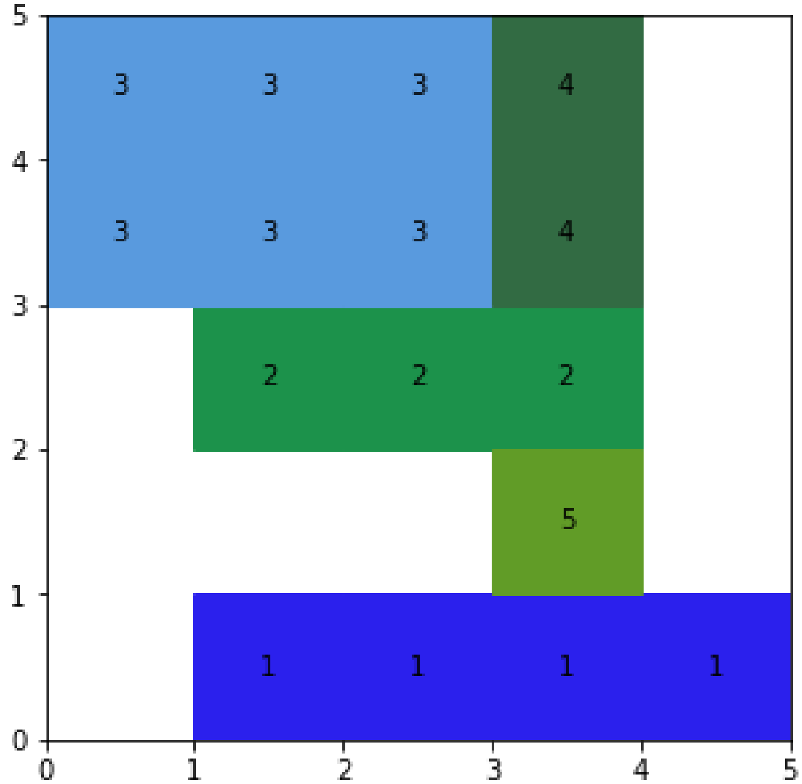
2
2
2
1
1
1
1
0
1
2
3
4
5



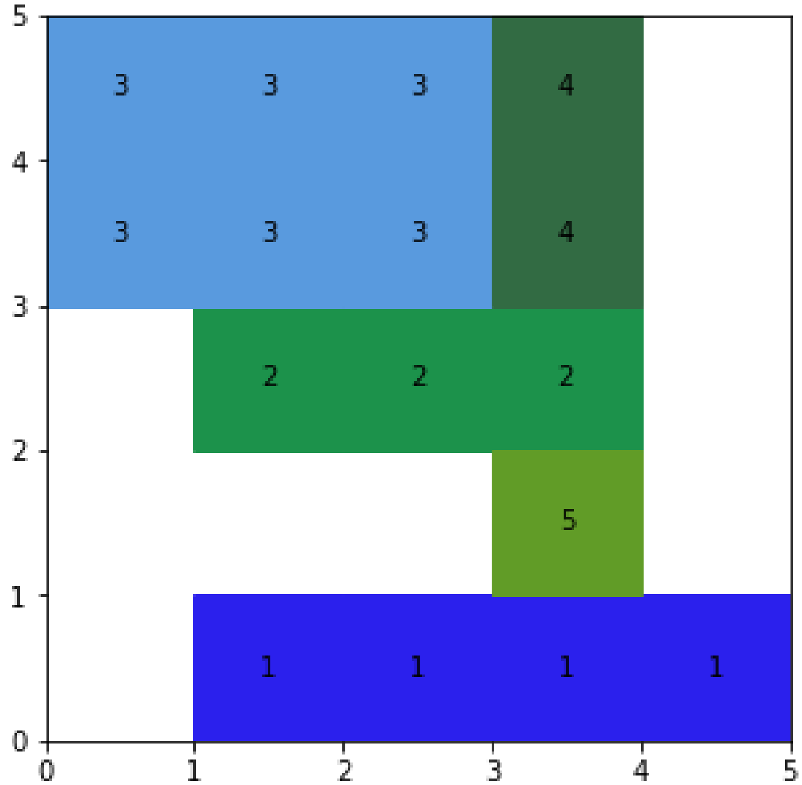
5
4
3
2
1
0
0
1
2
3
4
5
3 3 3
3 3 3
2 2 2
1 1 1 1



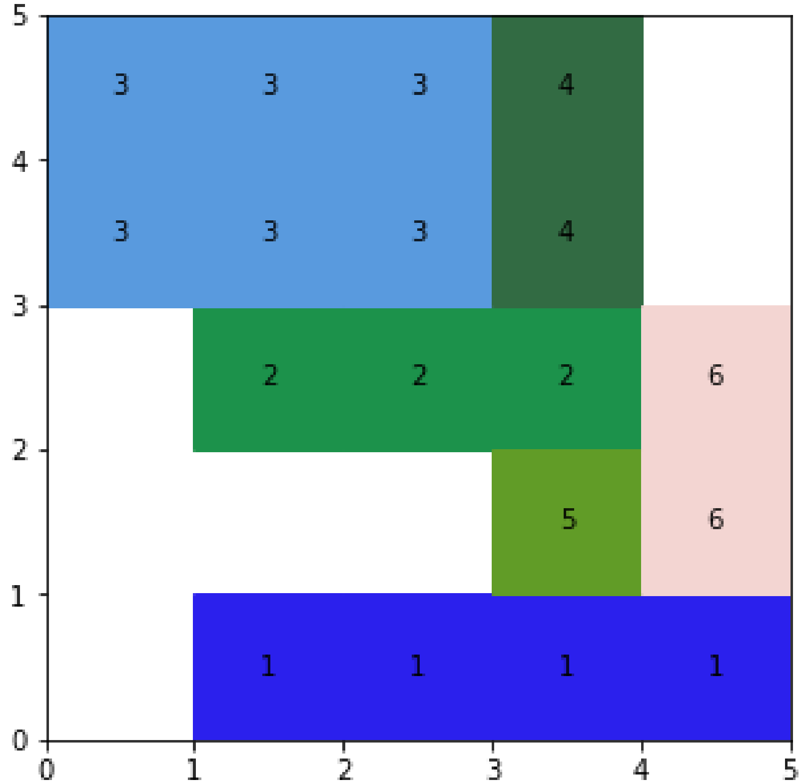
3 3 3 4
3 3 3 4
2 2 2
1 1 1 1
0 1 2 3 4 5

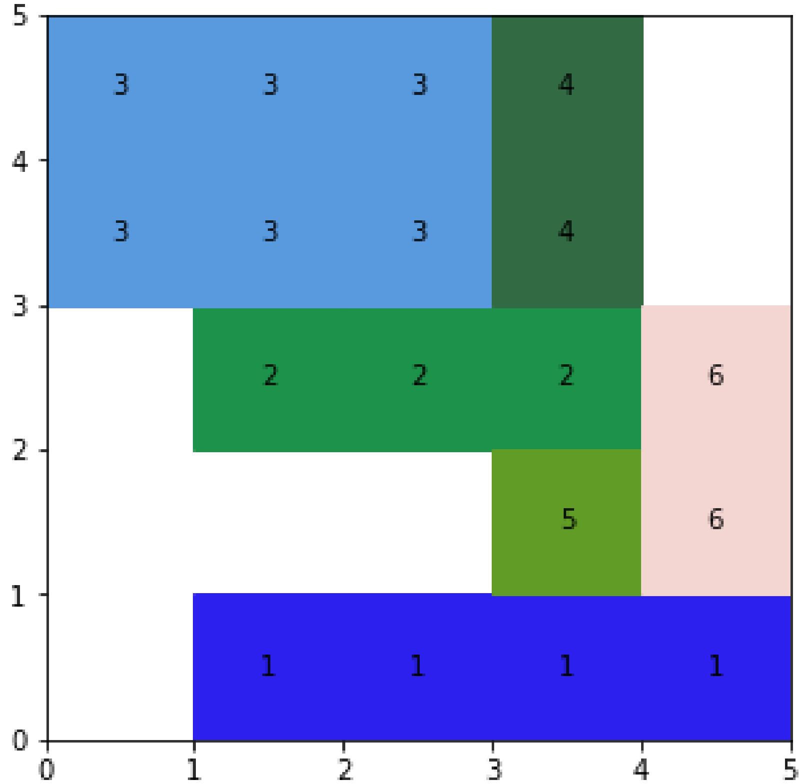


5
4
3
2
1
0
0
1
2
3
4
5
3
3
3
4
3
3
3
4
2
2
2
5
1
1
1
1

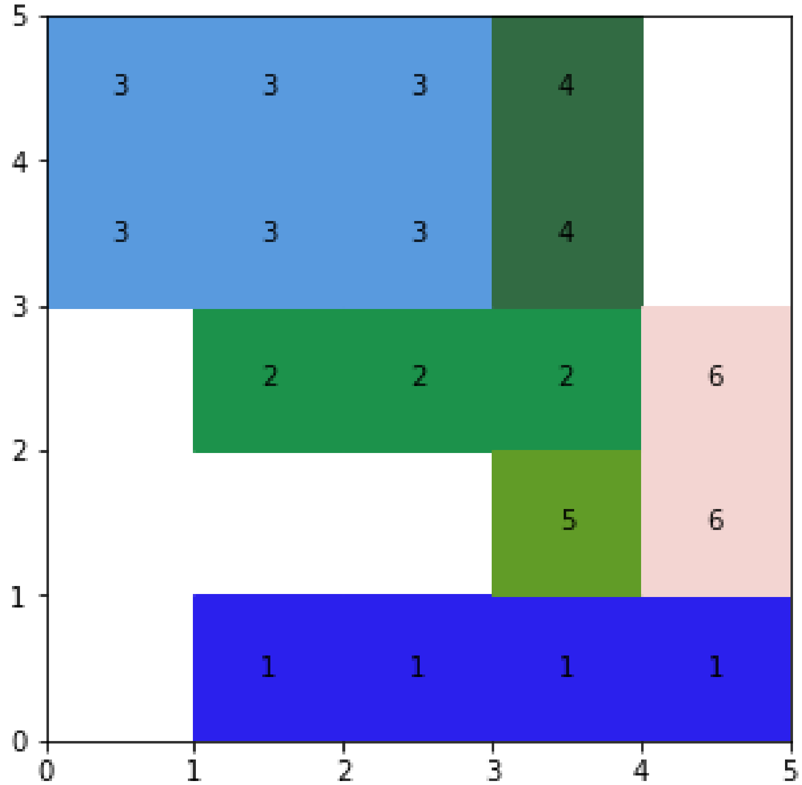


3 3 3 4
3 3 3 4
2 2 2
5
1 1 1 1
0 1 2 3 4 5

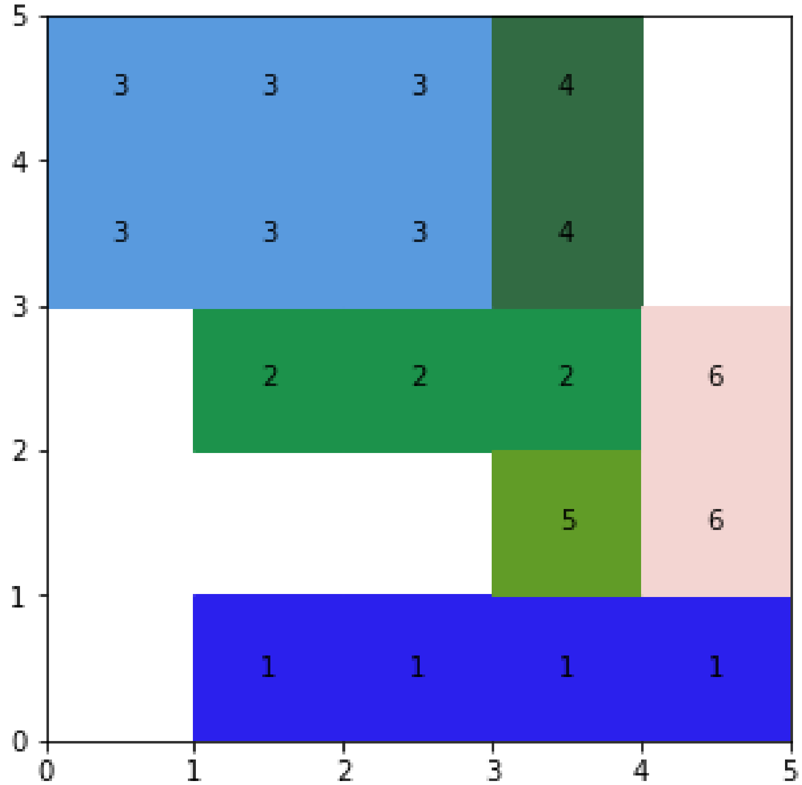

3
3
3
4
3
3
3
4
2
2
2
6
5
6
1
1
1
1
0
1
2
3
4
5
0
1
2
3
4
5

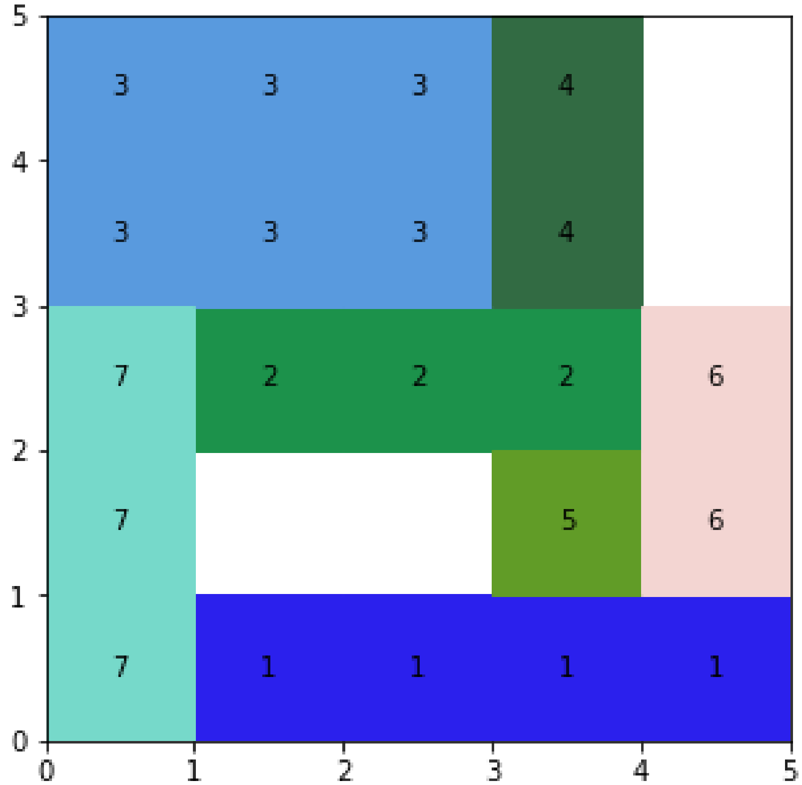


3
3
3
4
3
3
3
4
2
2
2
6
5
6
1
1
1
1
0
1
2
3
4
5

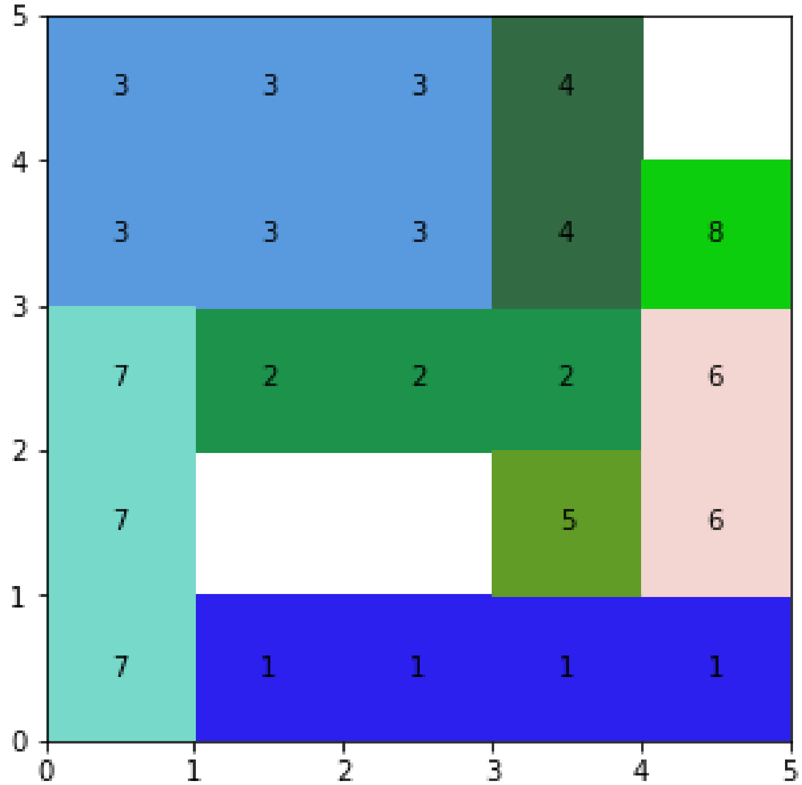


3
3
3
4
3
3
3
4
2
2
2
6
5
6
1
1
1
1
0
1
2
3
4
5

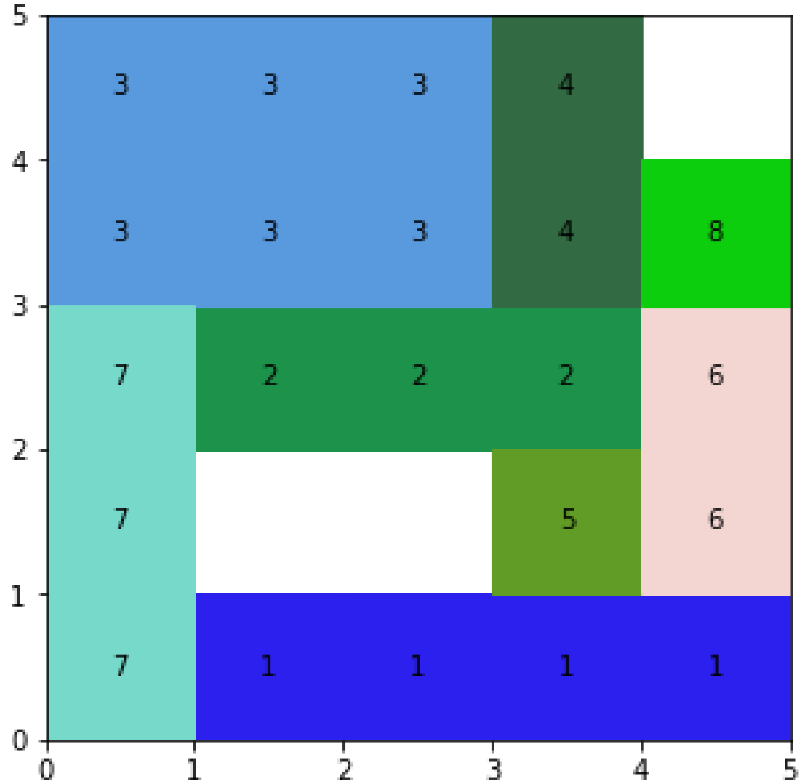

3
3
3
4
3
3
3
4
2
2
2
6
5
6
1
1
1
1
0
1
2
3
4
5
0
1
2
3
4
5



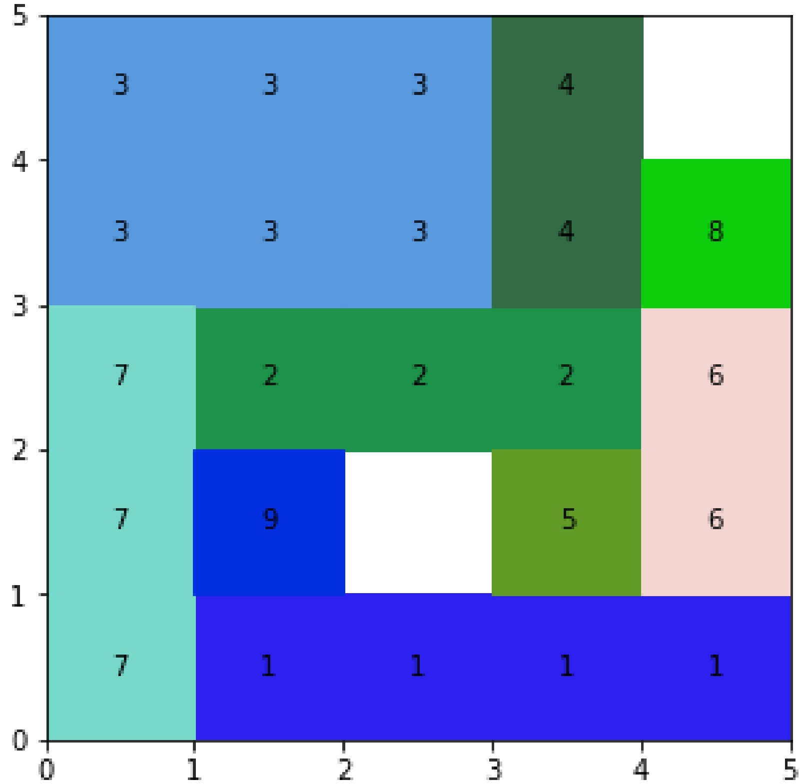
3 3 3 4
3 3 3 4
7 2 2 2 6
7 5 6
7 1 1 1 1
0 1 2 3 4 5



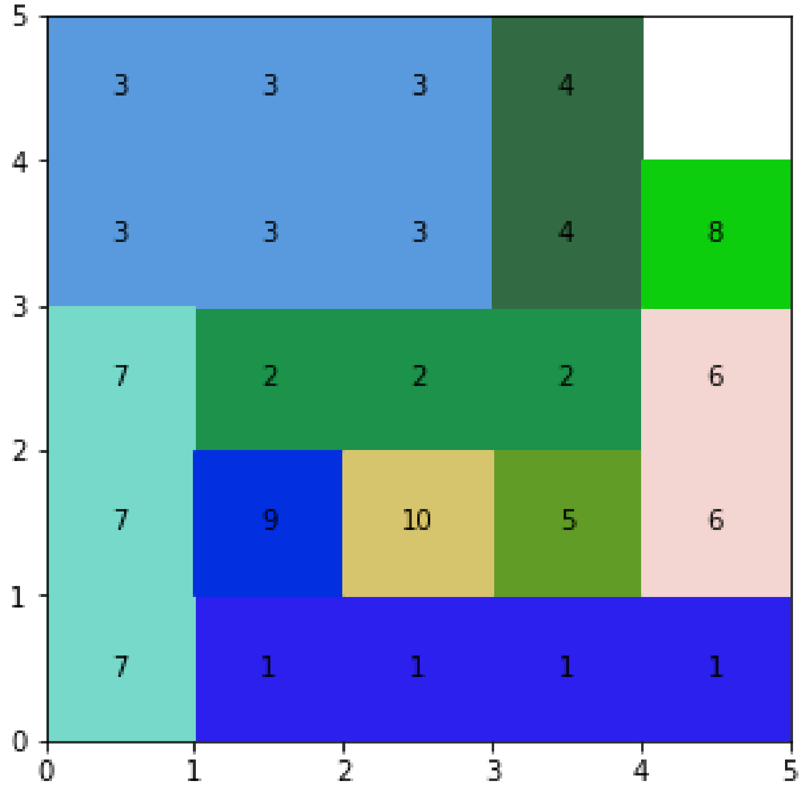
3 3 3 4
3 3 3 4 8
7 2 2 2 6
7 5 6
7 1 1 1 1
0 1 2 3 4 5

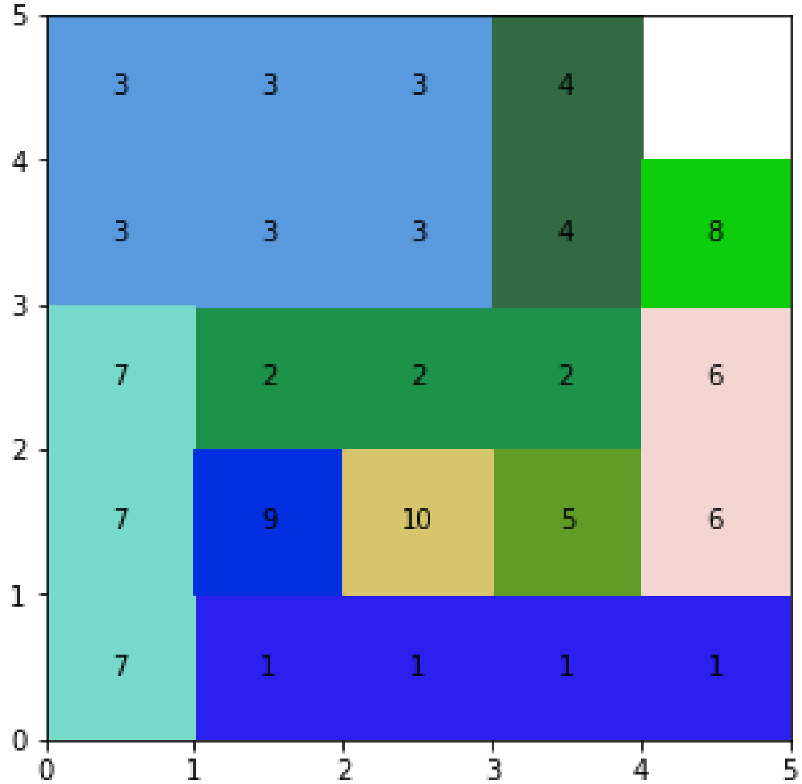


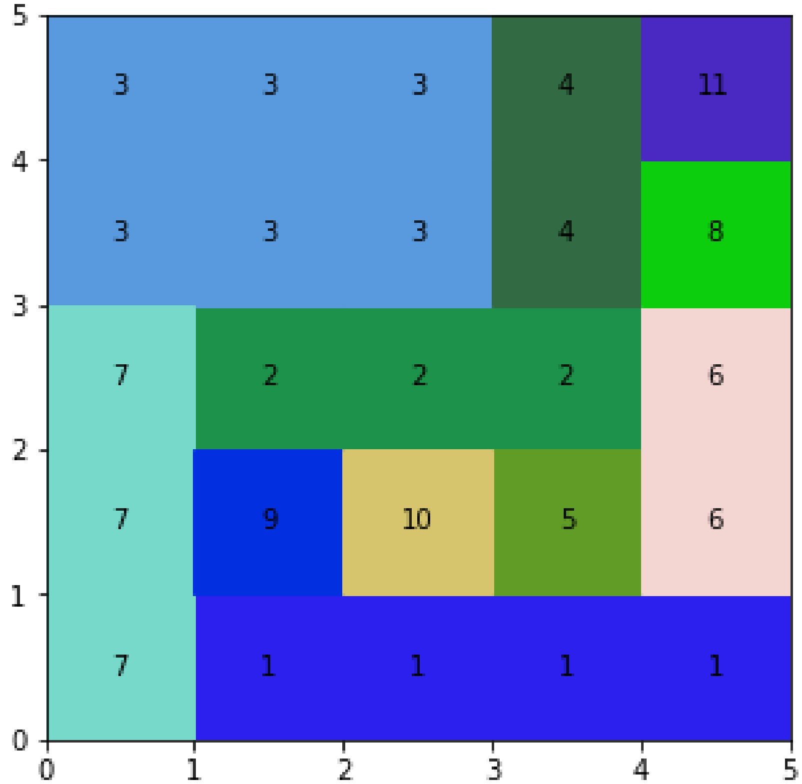
3
3
3
4
3
3
3
4
8
7
2
2
2
6
7
5
6
7
1
1
1
1
0
1
2
3
4
5

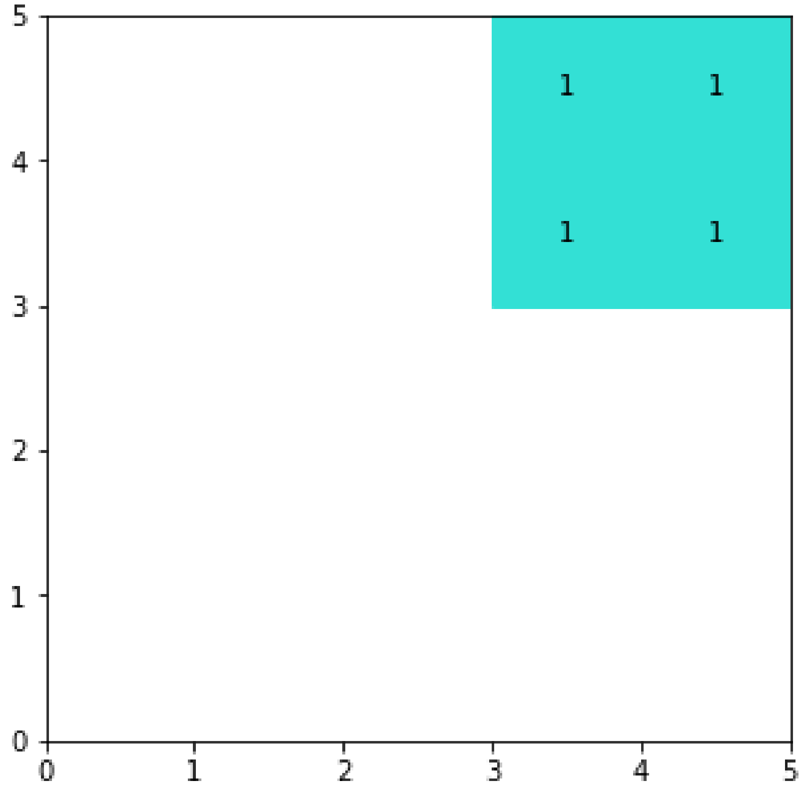


3
3
3
4
3
3
3
4
8
7
2
2
2
6
7
9
5
6
7
1
1
1
1
0
1
2
3
4
5

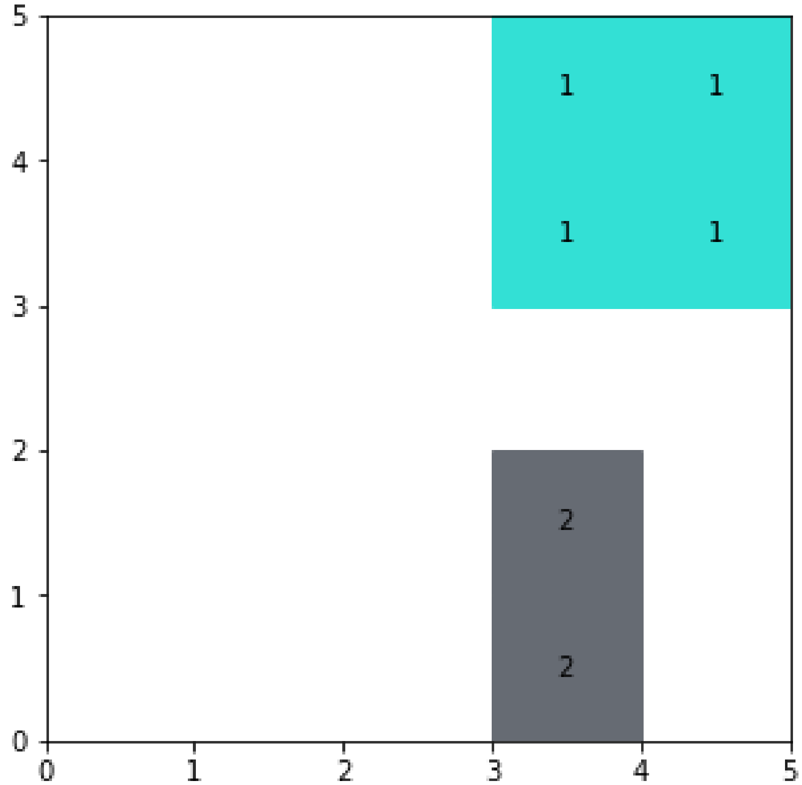

3
3
3
4
3
3
3
4
8
7
2
2
2
6
7
9
10
5
6
7
1
1
1
1
0
1
2
3
4
5

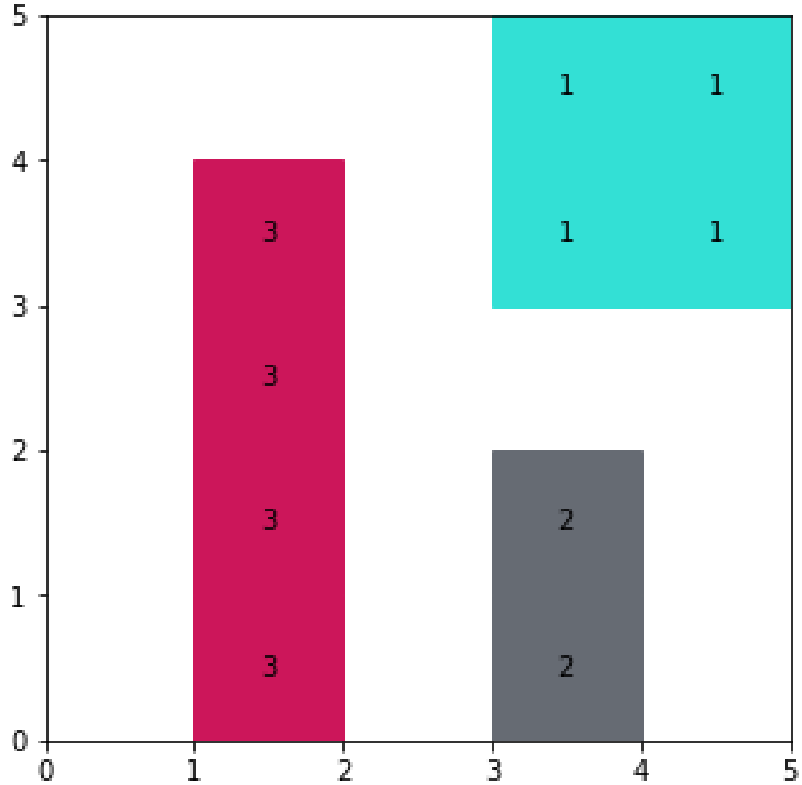

3
3
3
4
3
3
3
4
8
7
2
2
2
6
7
9
10
5
6
7
1
1
1
1
0
1
2
3
4
5
0
1
2
3
4
5


3 3 3 4 11
3 3 3 4 8
7 2 2 2 6
7 9 10 5 6
7 1 1 1 1
0 1 2 3 4 5
0 1 2 3 4 5

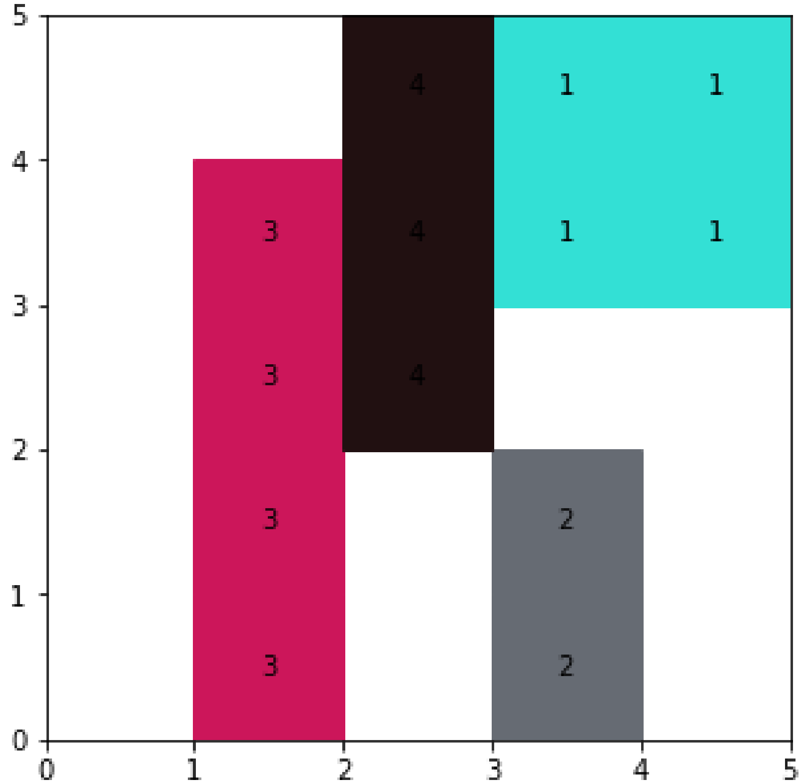


5
4
3
2
1
0
0
1
2
3
4
5
1
1
1
1

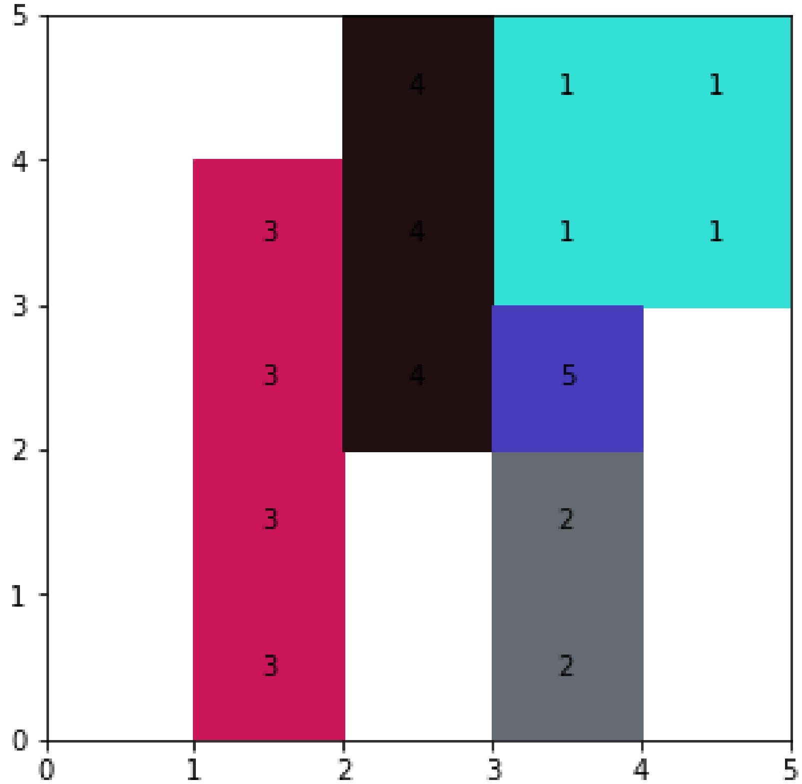

1
1
1
1
2
2
0
1
2
3
4
5
0
1
2
3
4
5



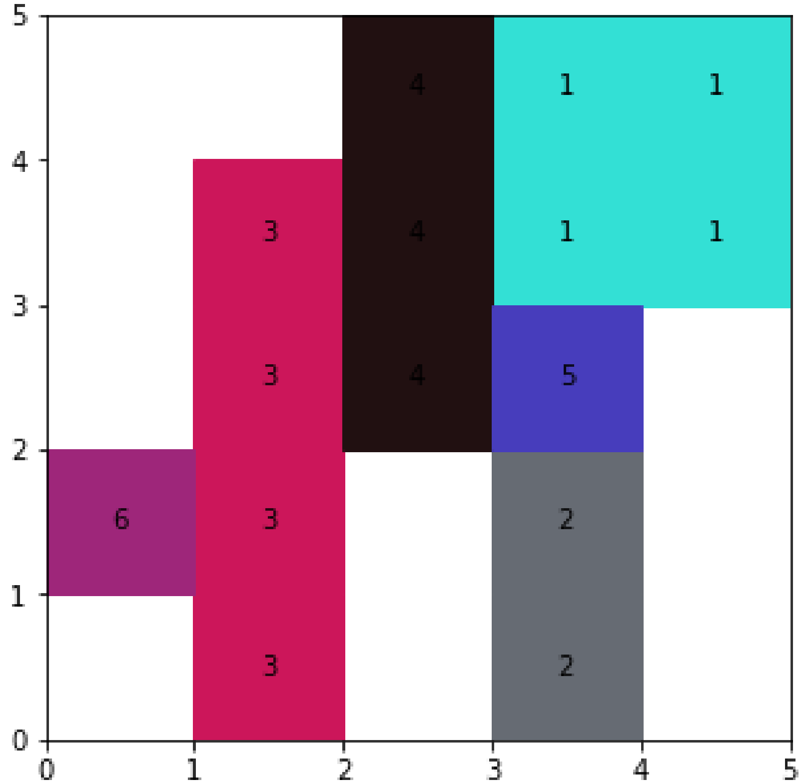
5
4
3
2
1
0
0
1
2
3
4
5
1
1
1
1
3
3
3
3
2
2



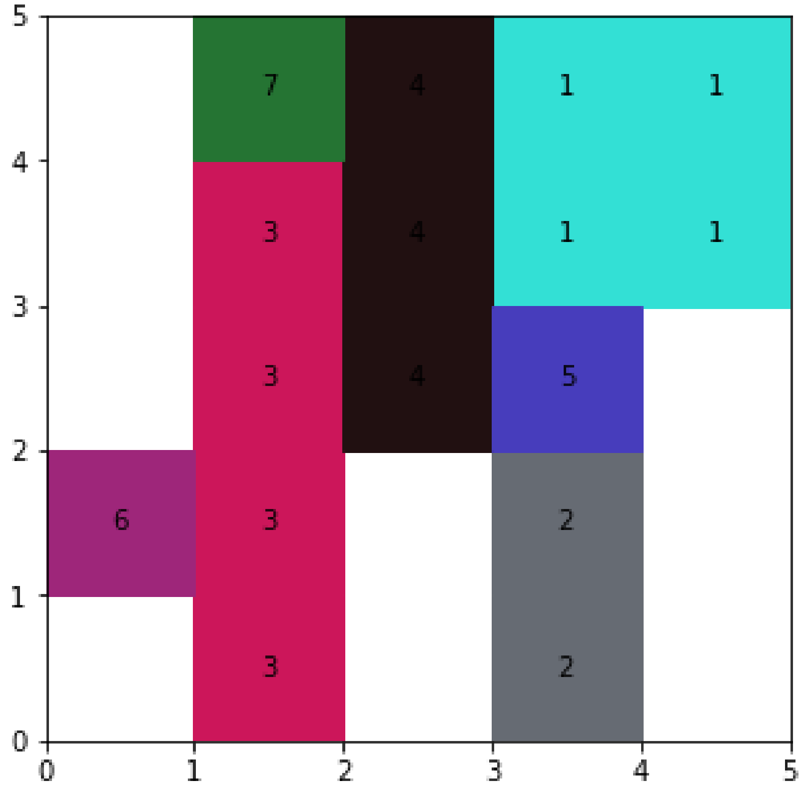
4
1
1
3
4
1
1
3
4
3
2
3
2
0
1
2
3
4
5



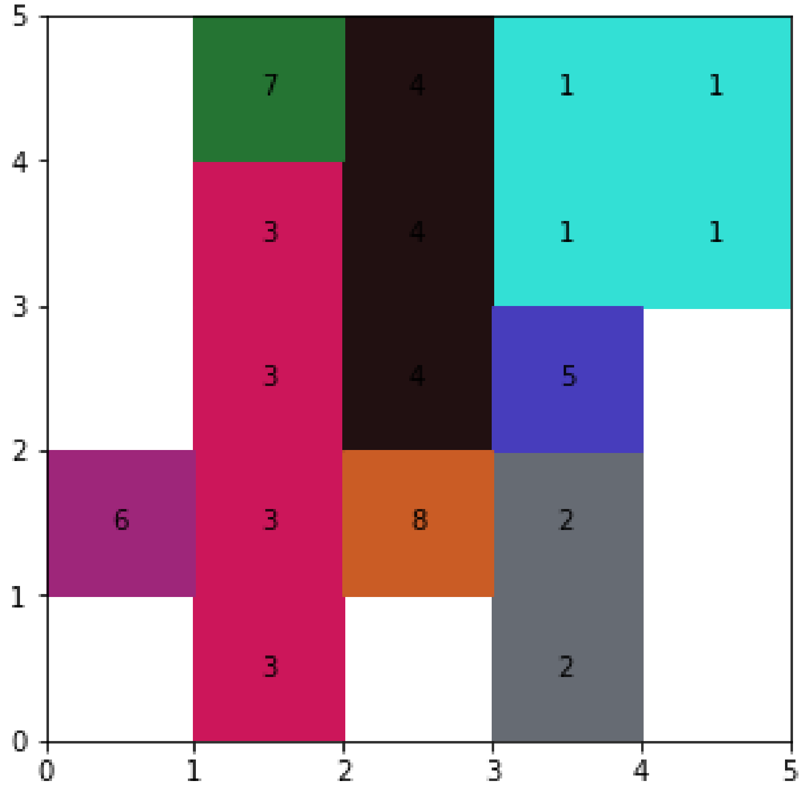
4
1
1
3
4
1
1
3
4
5
3
2
3
2
0
1
2
3
4
5

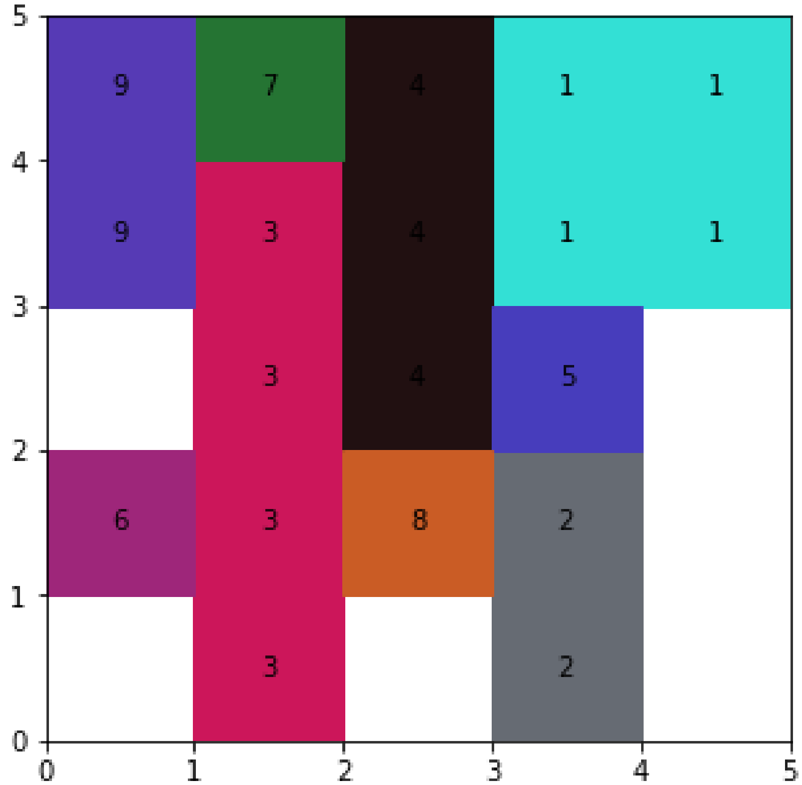


5
4
3
2
1
0
0
1
2
3
4
5
4
1
1
3
4
1
1
3
4
5
6
3
2
3
2

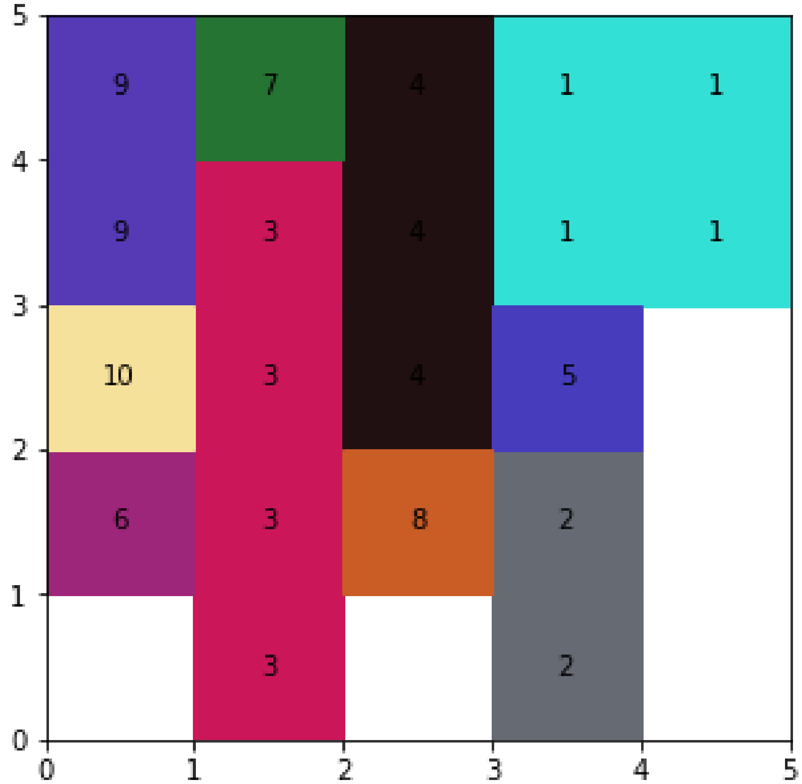


7
4
1
1
3
4
1
1
3
4
5
6
3
2
3
2
0
1
2
3
4
5

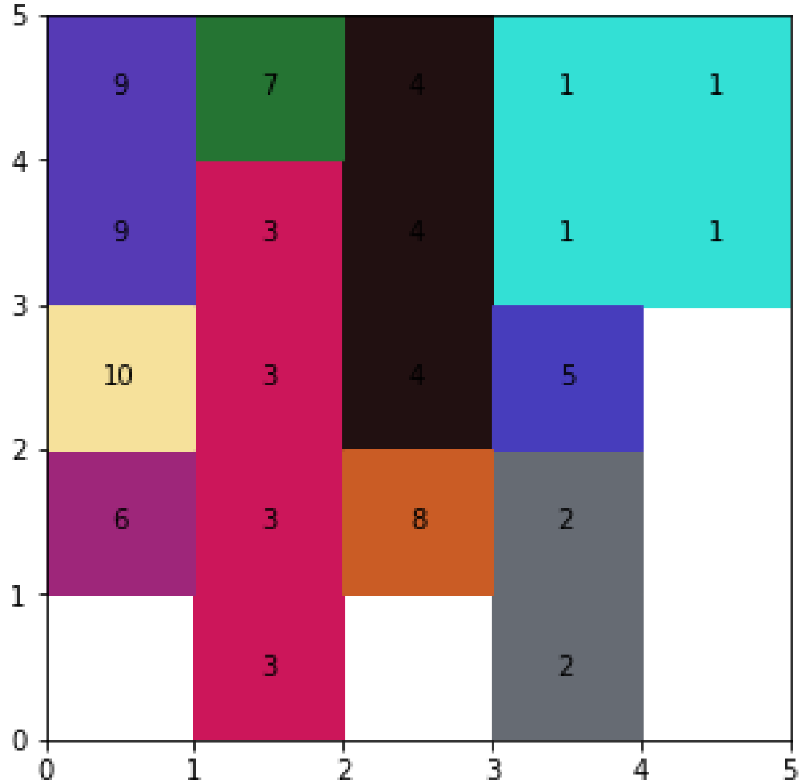

7
4
1
1
3
4
1
1
3
4
5
6
3
8
2
3
2
0
1
2
3
4
5



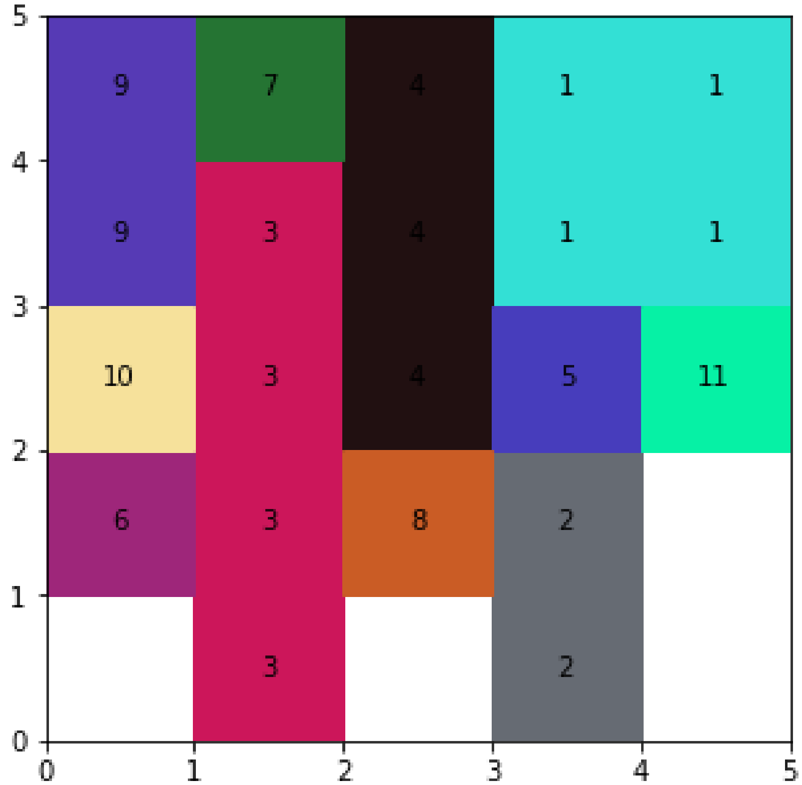
9
7
4
1
1
9
3
4
1
1
3
4
5
6
3
8
2
3
2
0
1
2
3
4
5



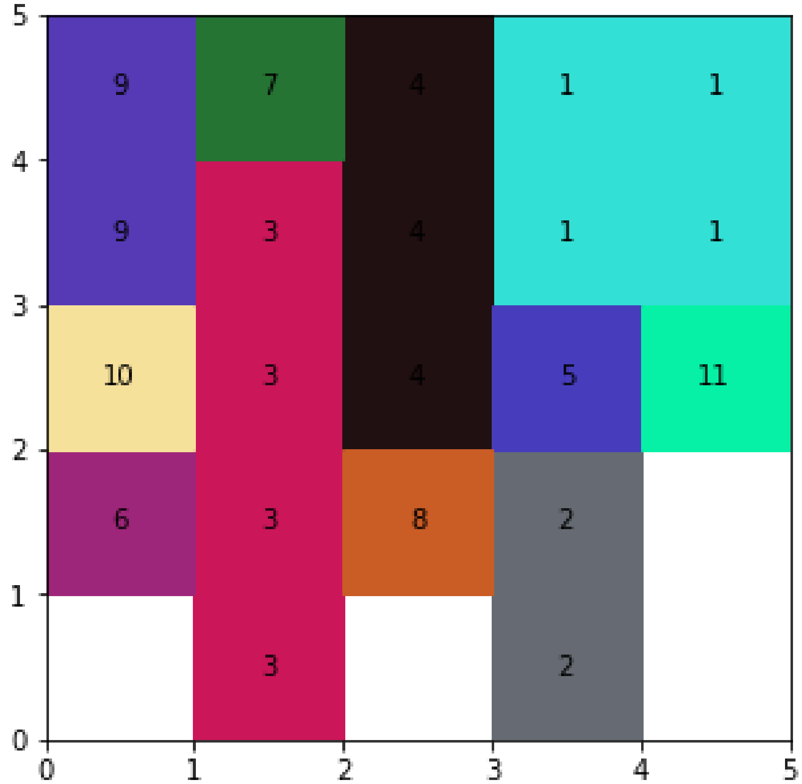
9
7
4
1
1
9
3
4
1
1
10
3
4
5
6
3
8
2
3
2
0
1
2
3
4
5



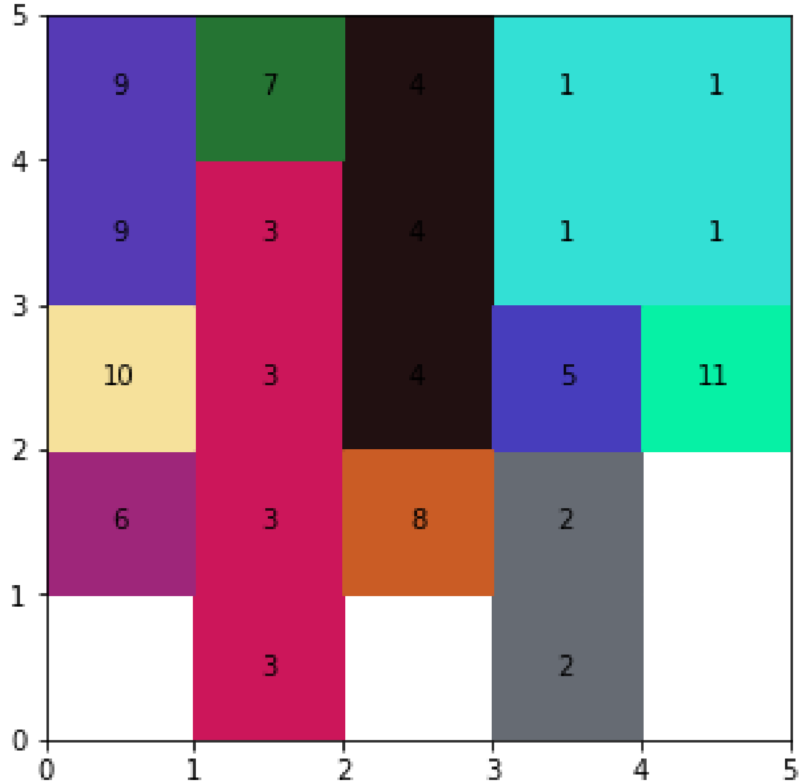
9
7
4
1
1
9
3
4
1
1
10
3
4
5
6
3
8
2
3
2
0
1
2
3
4
5

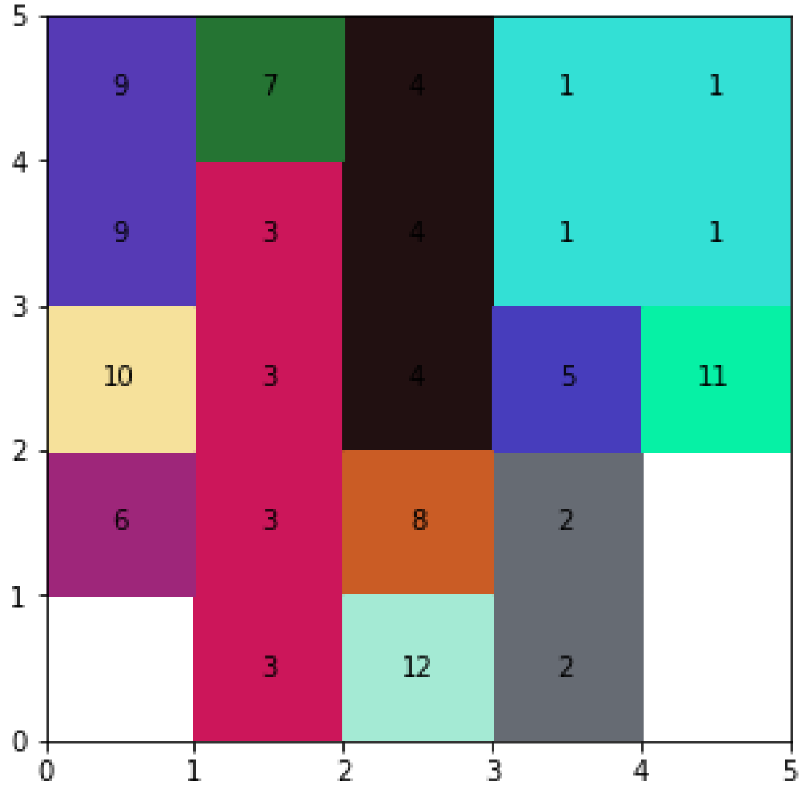


5
4
3
2
1
0
0
1
2
3
4
5
9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
3
2

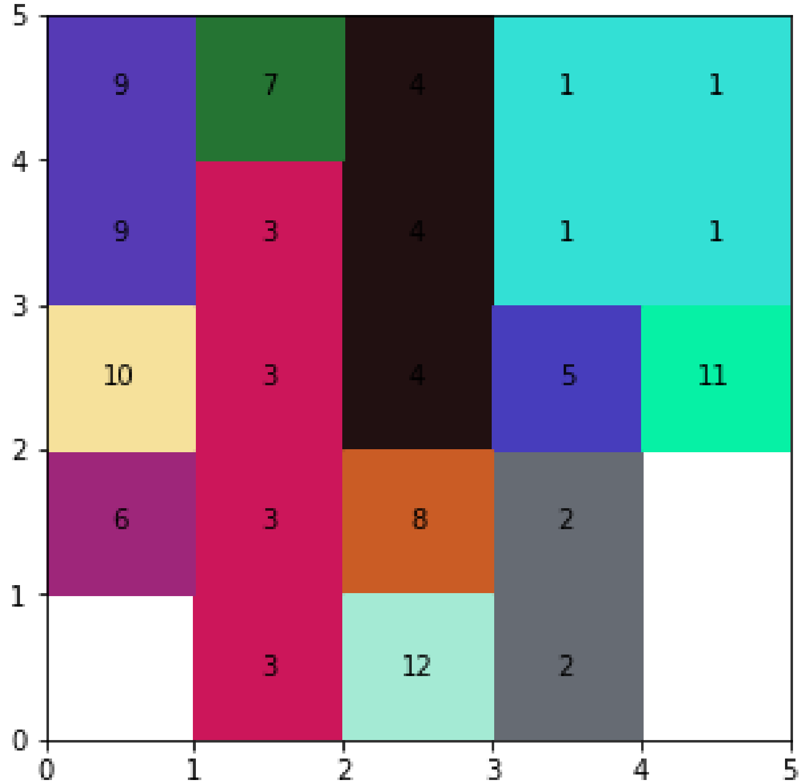


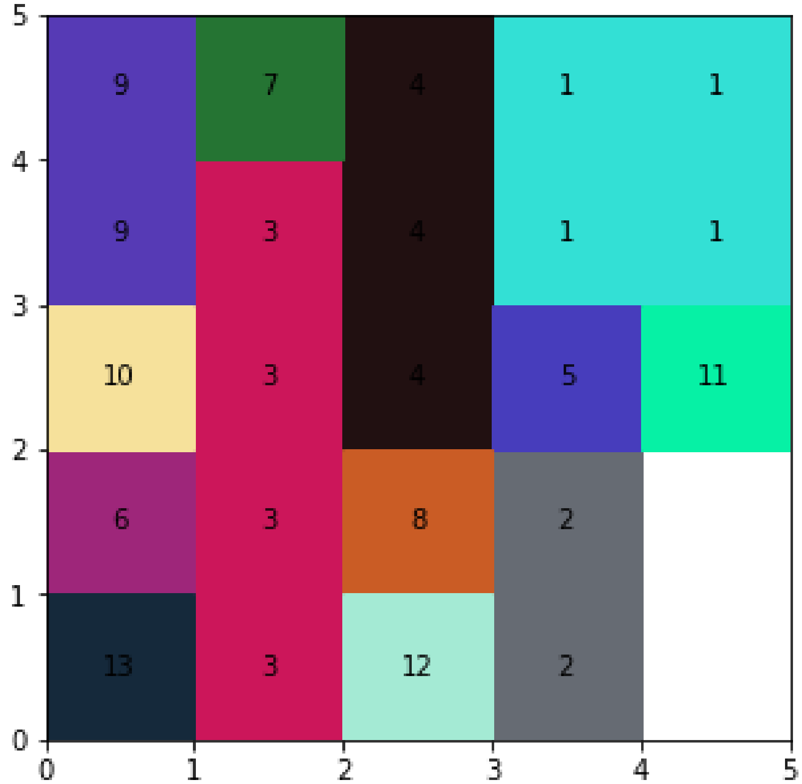
9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
3
2
0
1
2
3
4
5

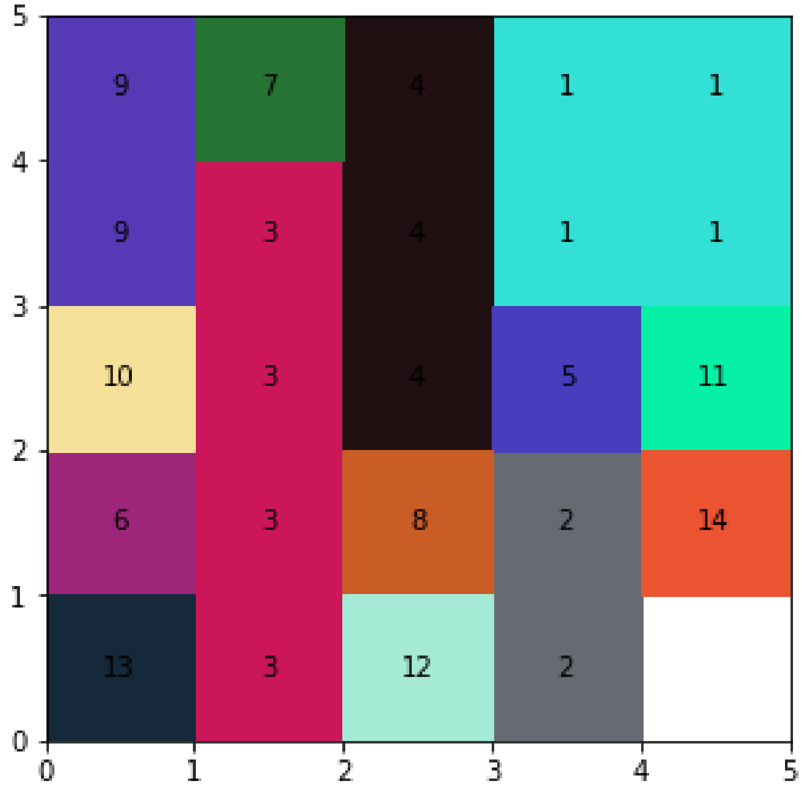

9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
3
2
0
1
2
3
4
5

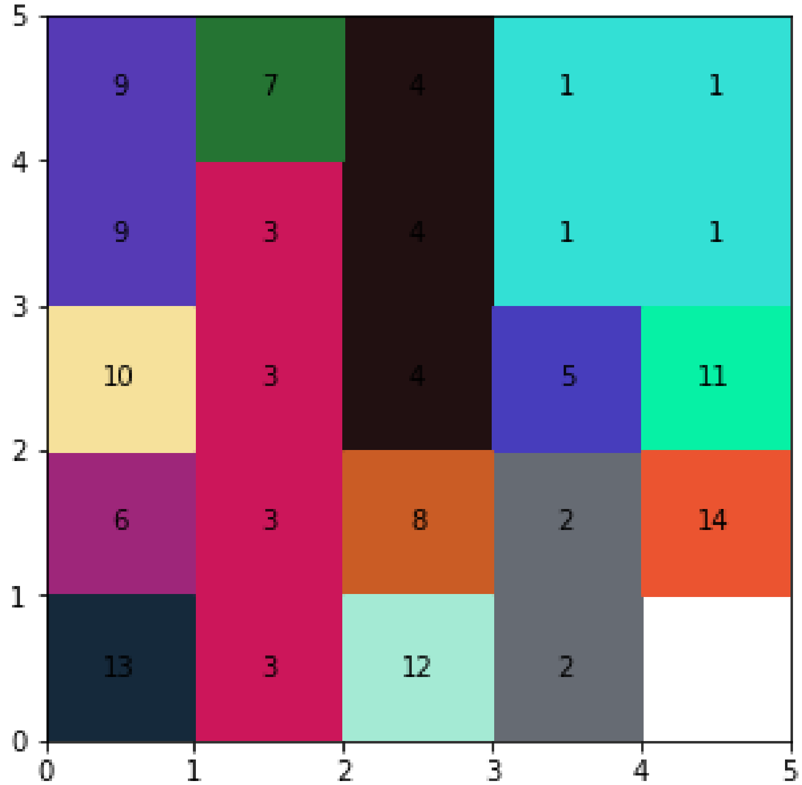


9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
3
12
2
0
1
2
3
4
5

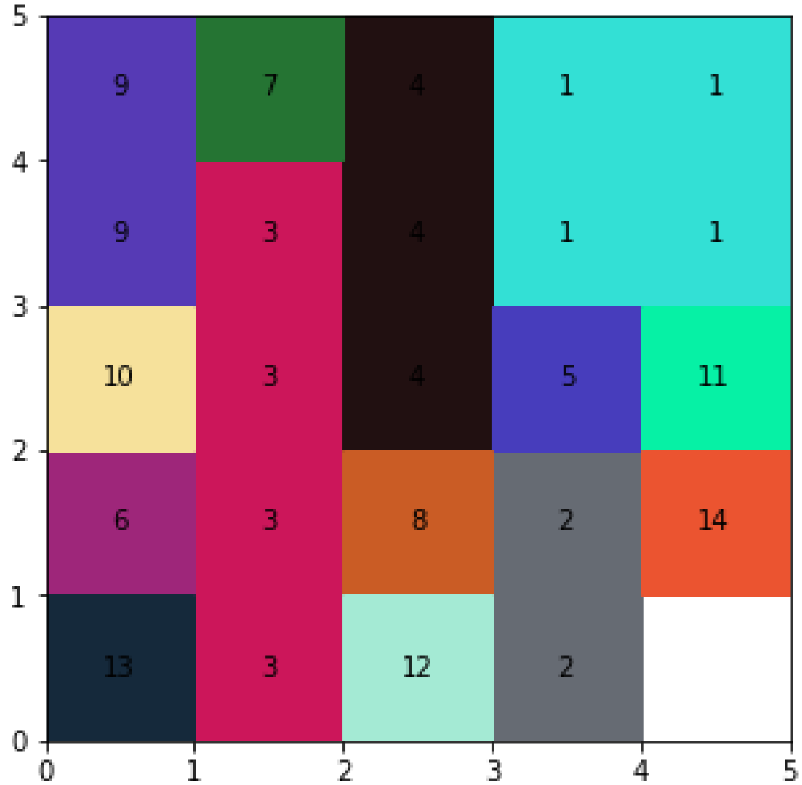

9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
3
12
2
0
1
2
3
4
5


9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
13
3
12
2
0
1
2
3
4
5

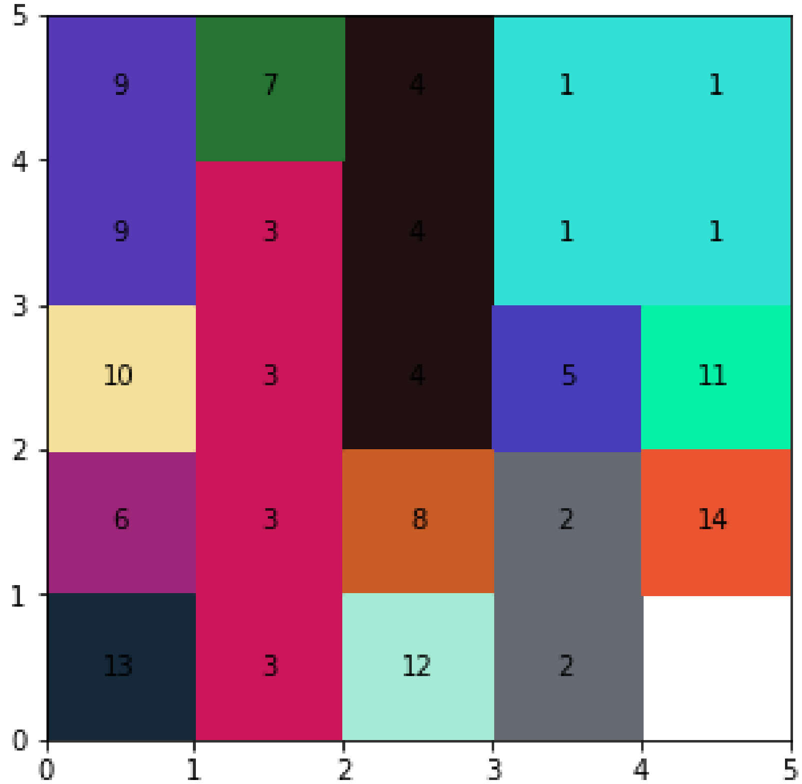

9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2
0
1
2
3
4
5

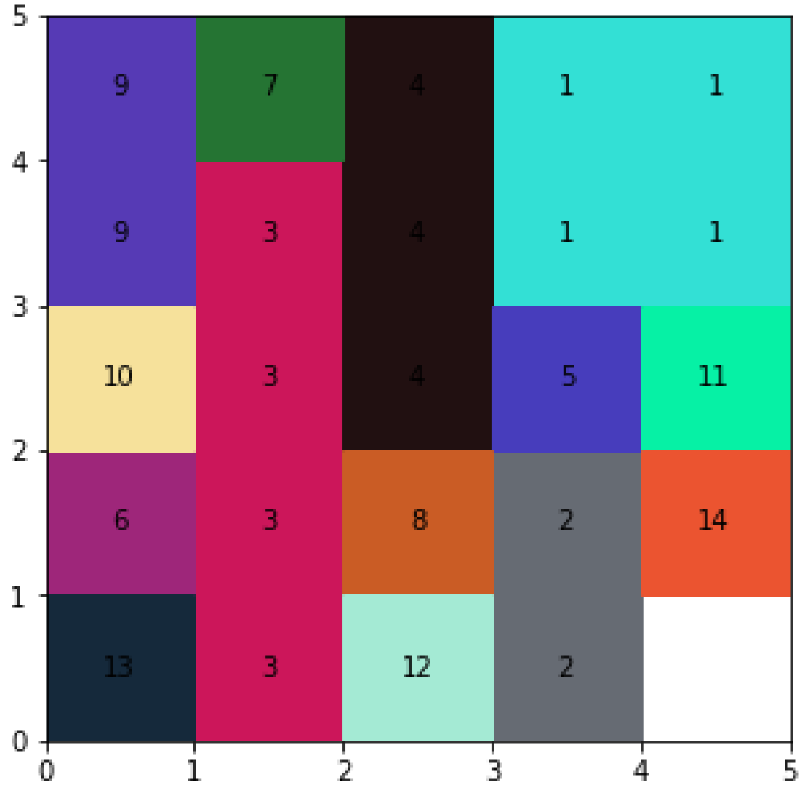


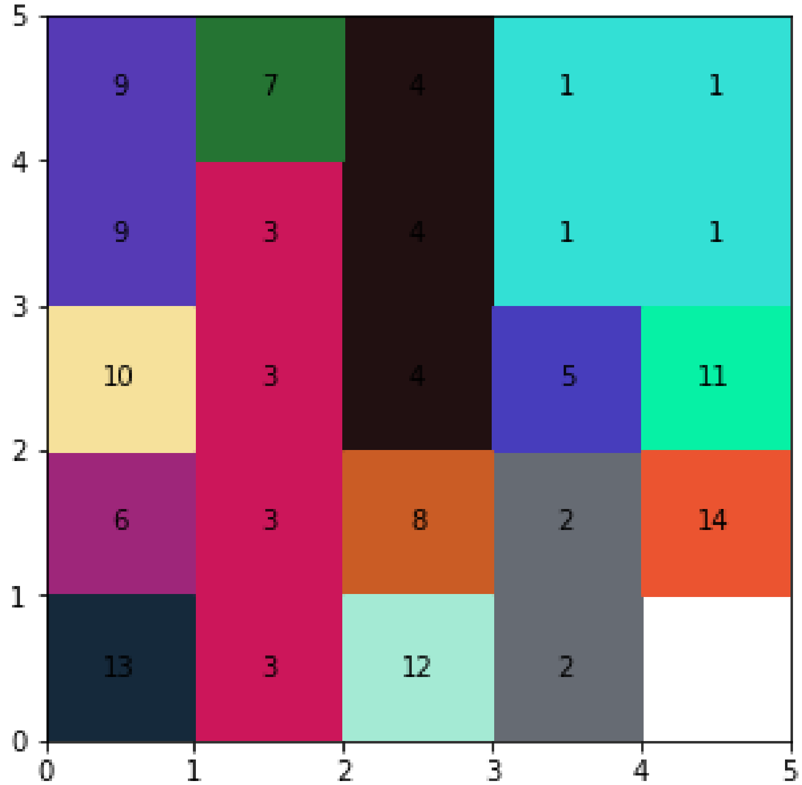
9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2
0
1
2
3
4
5



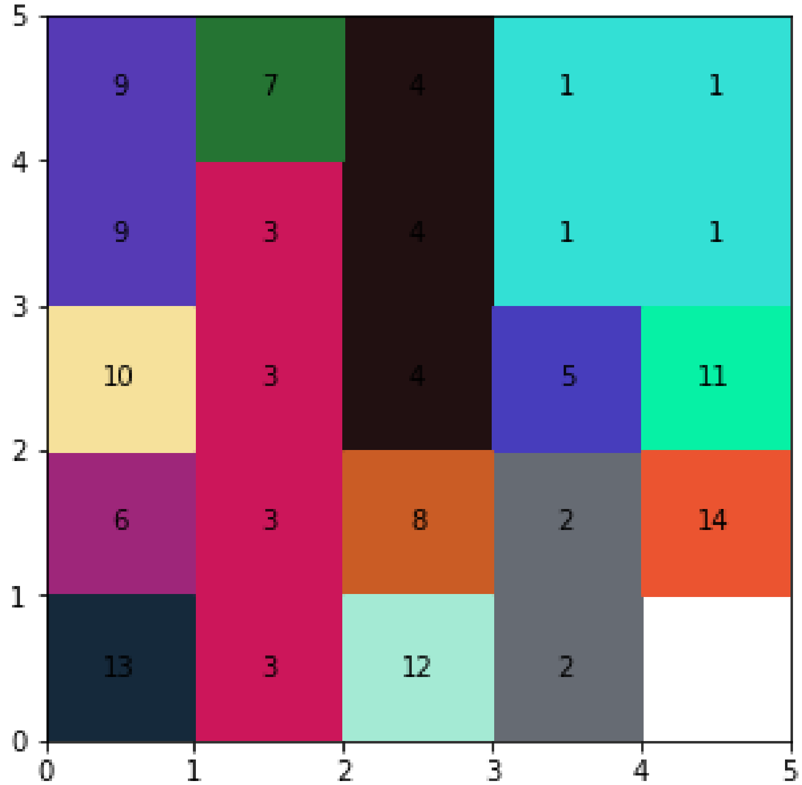
9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2
0
1
2
3
4
5

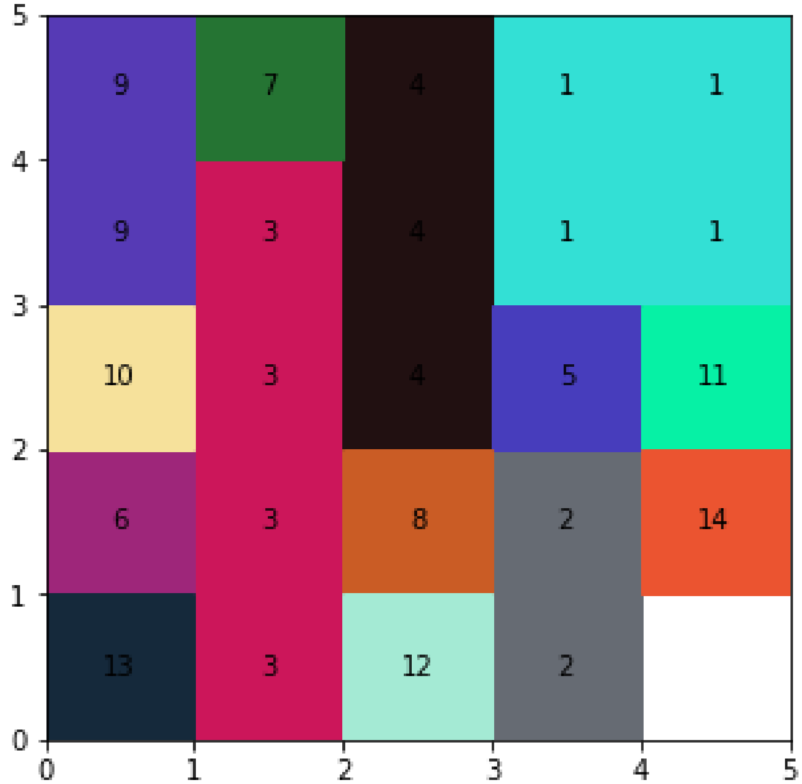

9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2
0
1
2
3
4
5

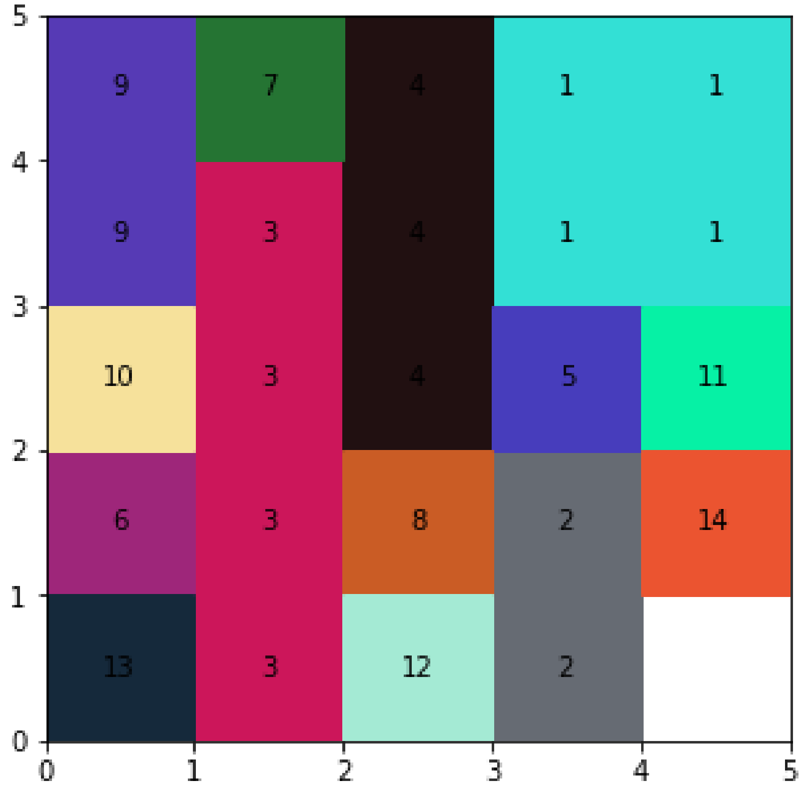

9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2
0
1
2
3
4
5

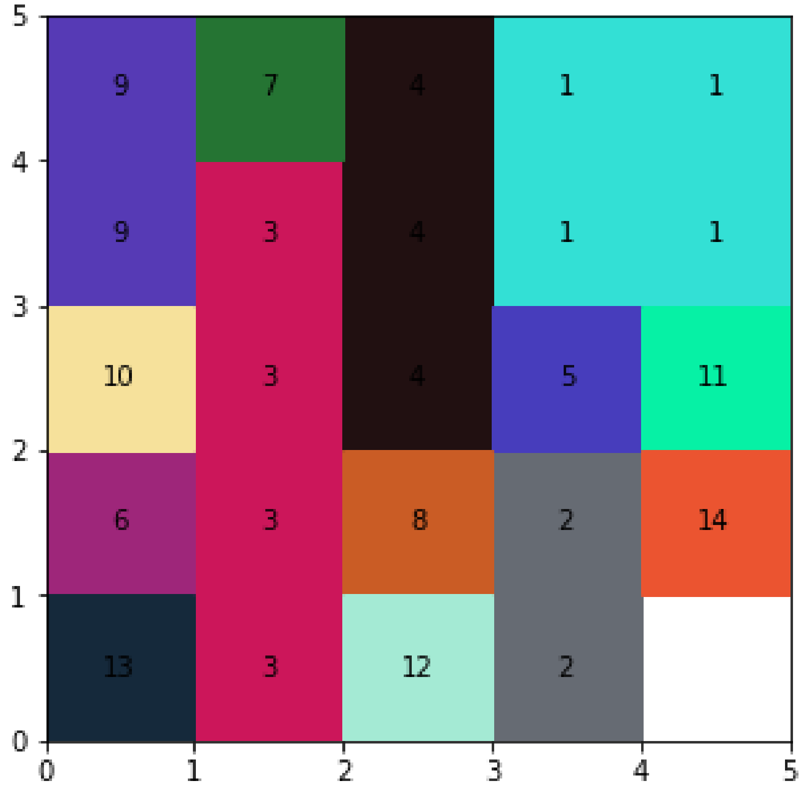


9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2
0
1
2
3
4
5

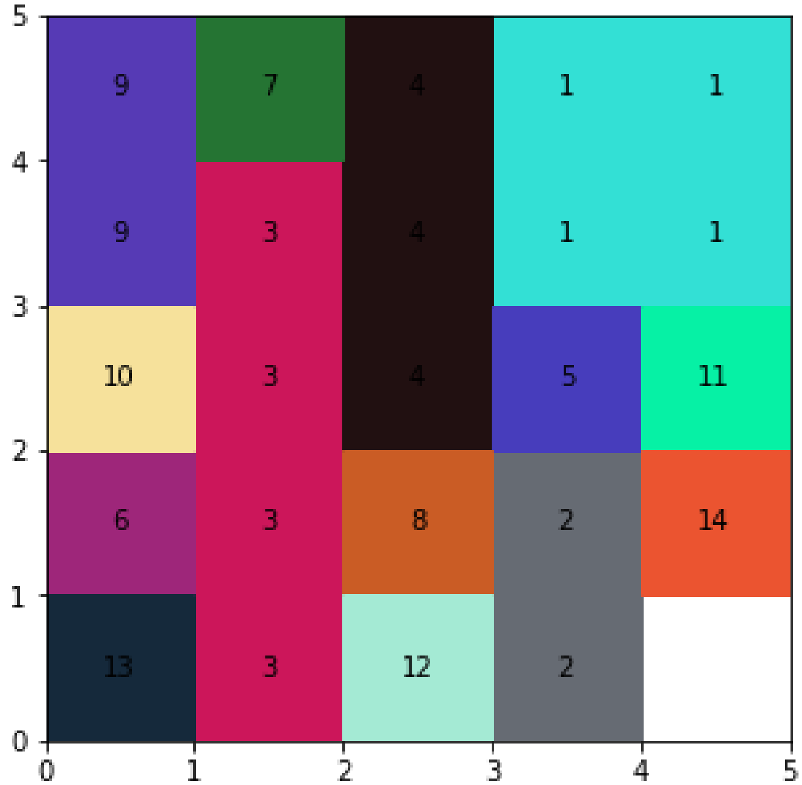

9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2
0
1
2
3
4
5


5
4
3
2
1
0
0
1
2
3
4
5
9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2

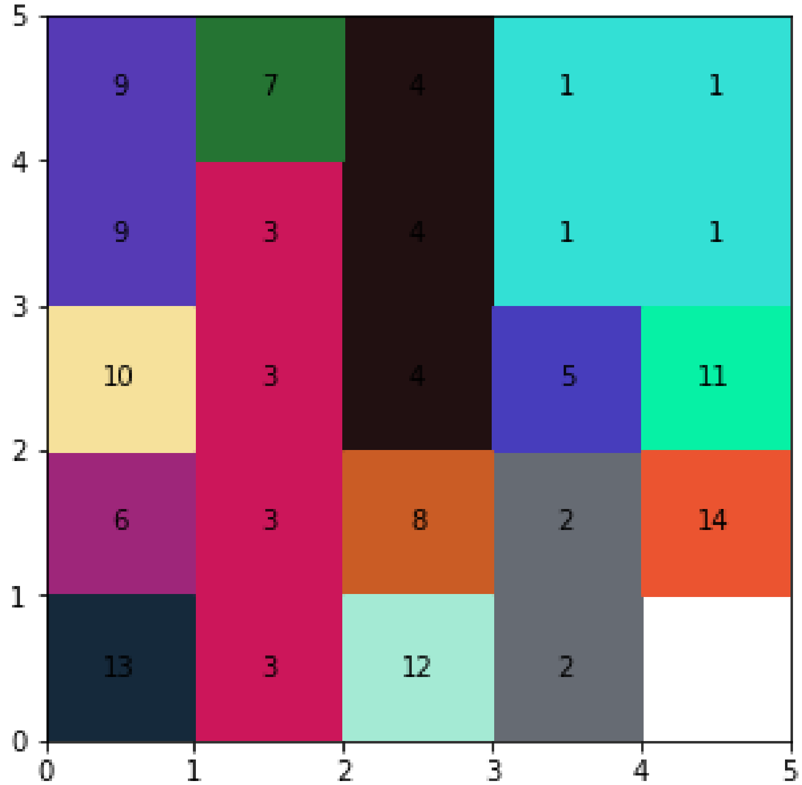

9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2
0
1
2
3
4
5



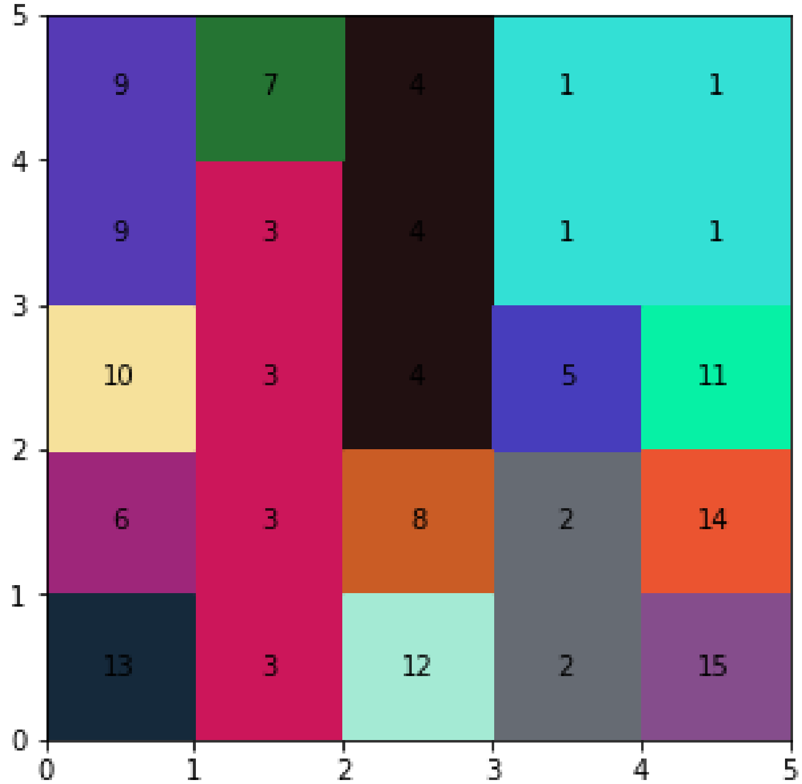
5
4
3
2
1
0
0
1
2
3
4
5
9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2

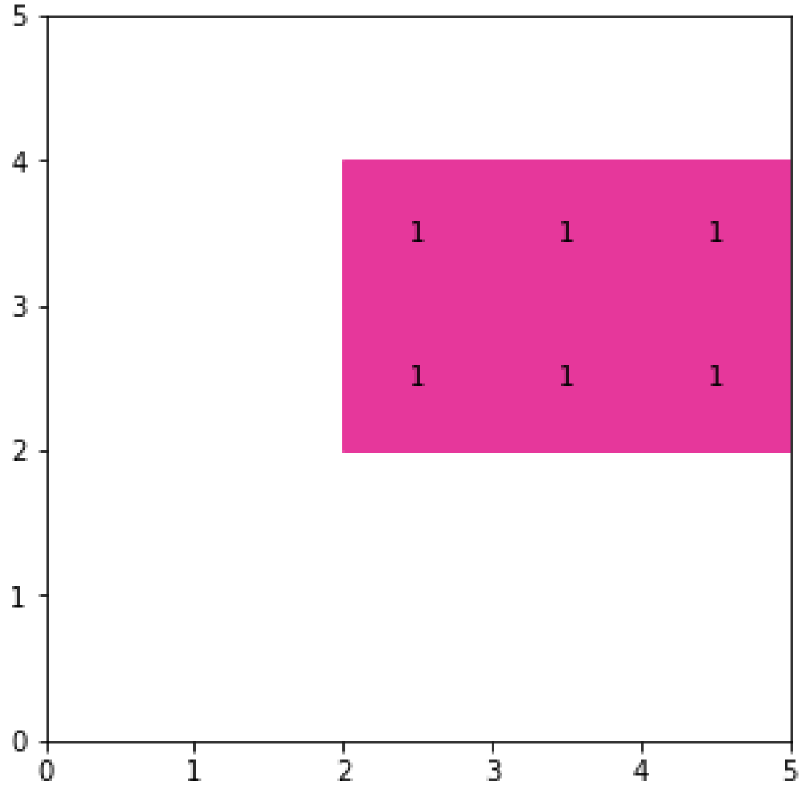


9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2
0
1
2
3
4
5

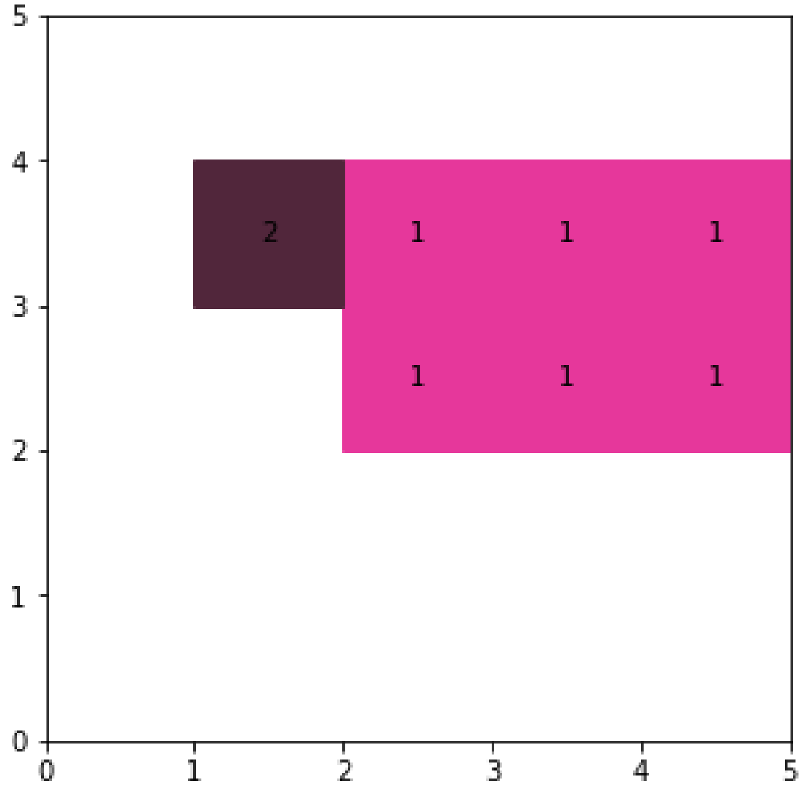


9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2
0
1
2
3
4
5

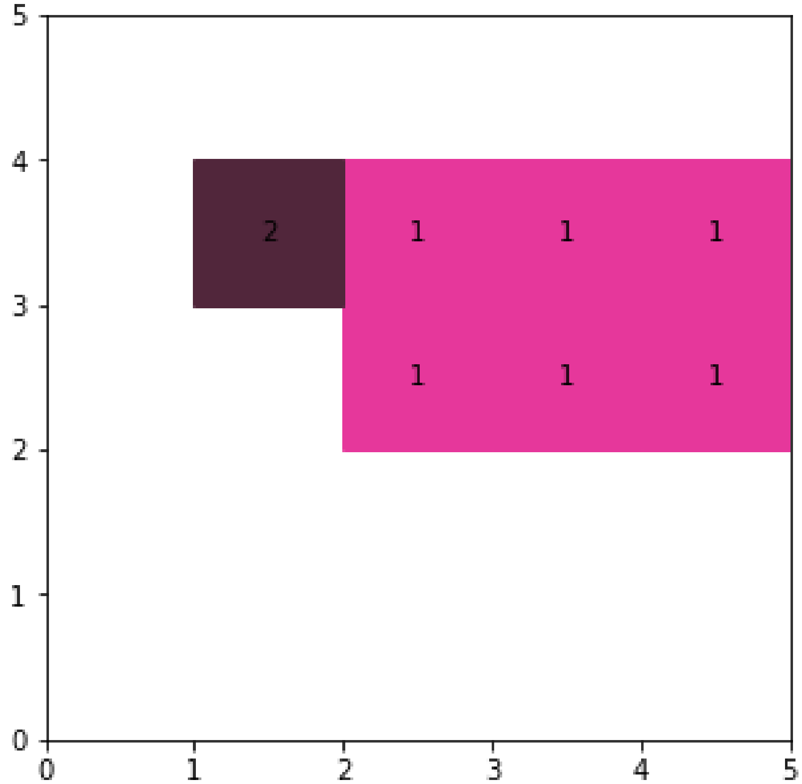

9
7
4
1
1
9
3
4
1
1
10
3
4
5
11
6
3
8
2
14
13
3
12
2
15
0
1
2
3
4
5



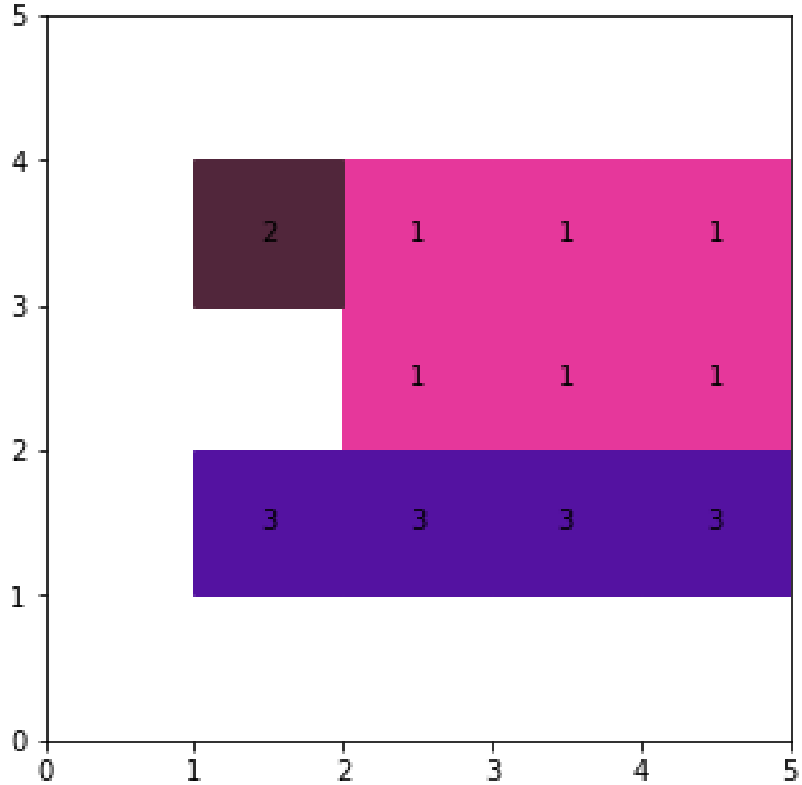
5
4
3
2
1
0
1 1 1
1 1 1
0 1 2 3 4 5



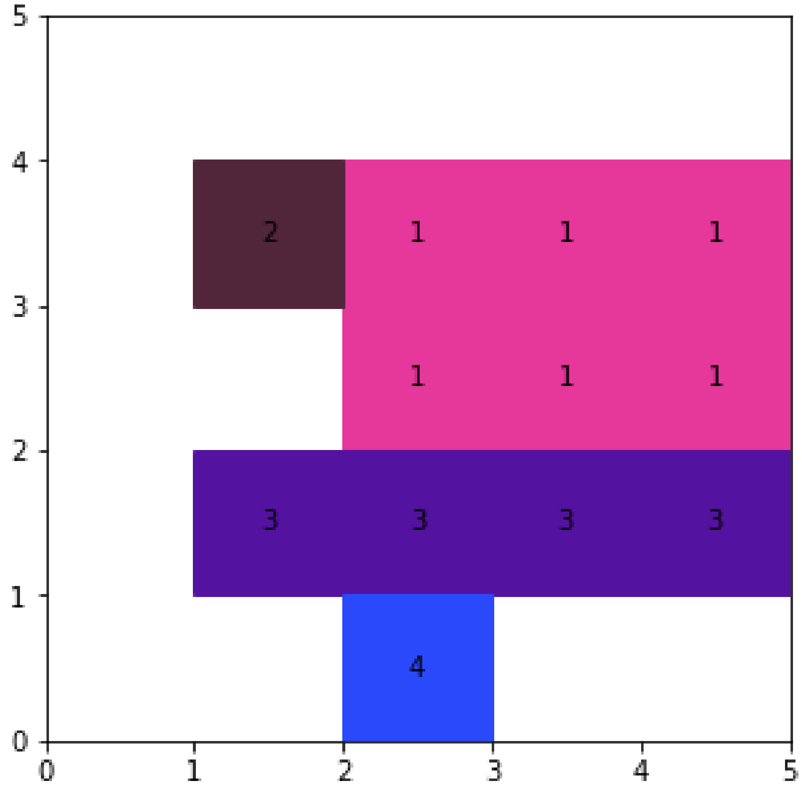
5
4
3
2
1
0
0
1
2
3
4
5
2
1
1
1
1
1
1



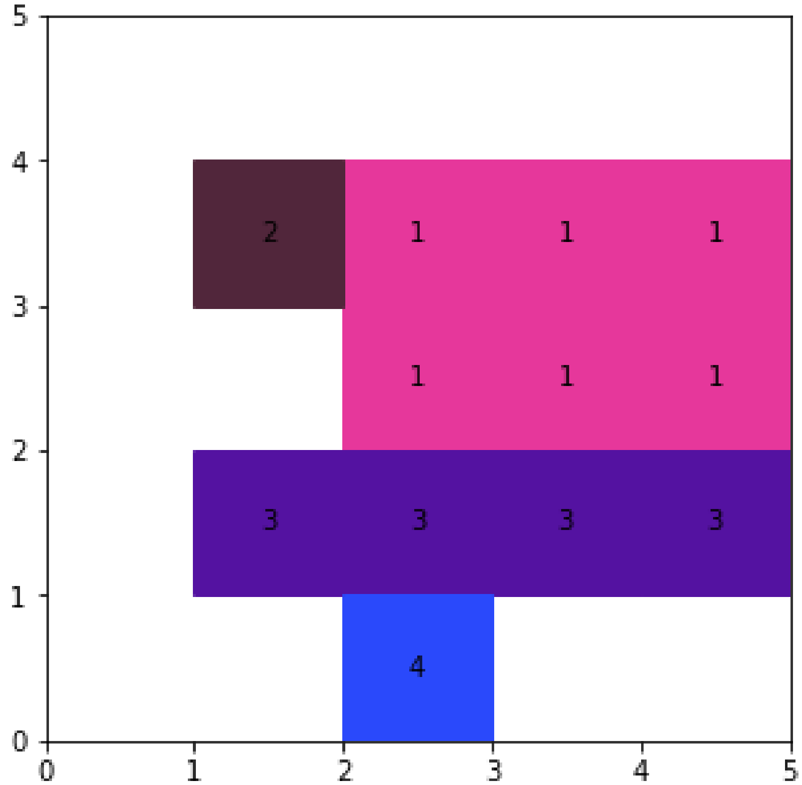
5
4
3
2
1
0
0
1
2
3
4
5
2
1
1
1
1
1
1



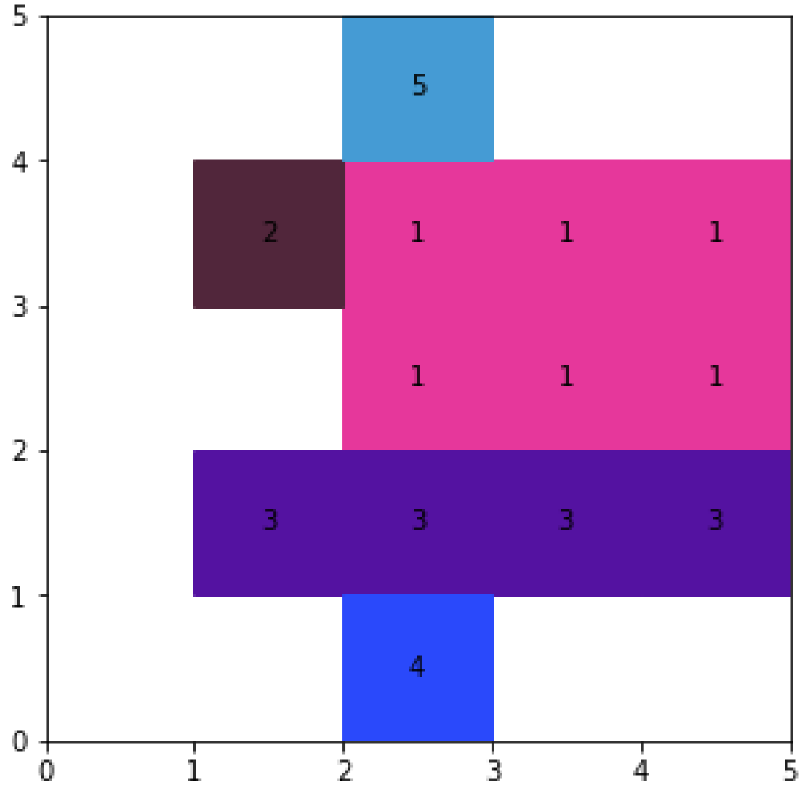
5
4
3
2
1
0
0
1
2
3
4
5
2
1
1
1
1
1
1
3
3
3
3



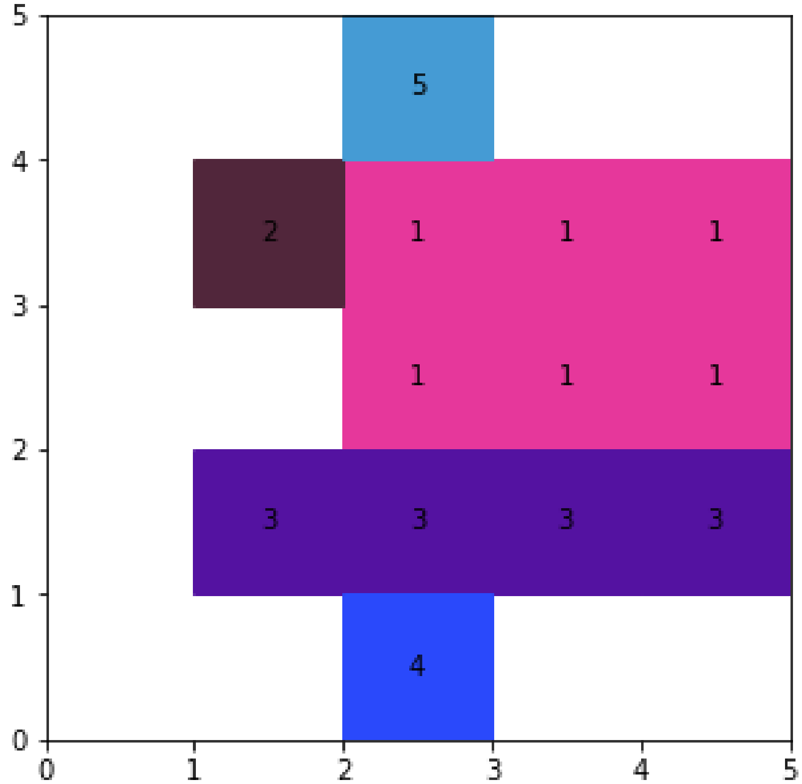
5
4
3
2
1
0
0
1
2
3
4
5
2
1
1
1
1
1
1
3
3
3
3
4

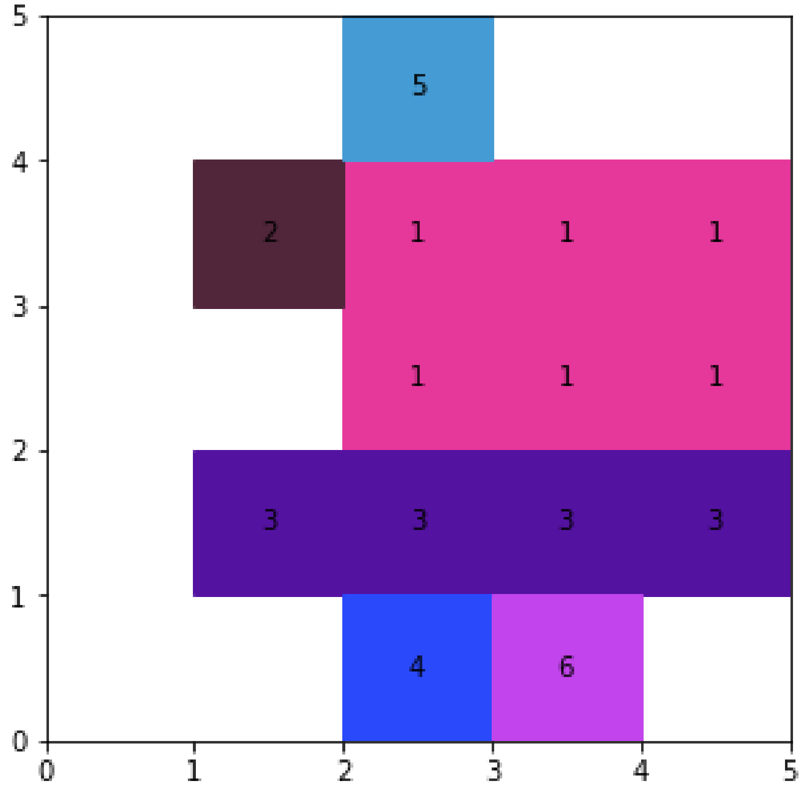


5
4
3
2
1
0
0
1
2
3
4
5
2
1
1
1
1
1
1
3
3
3
3
4

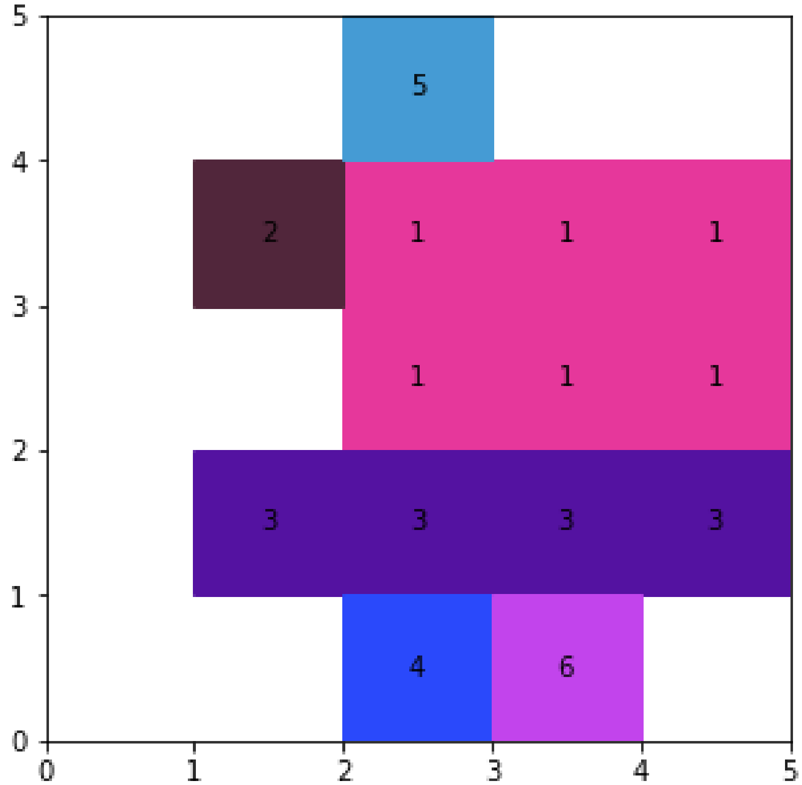


5
4
3
2
1
0
5
2 1 1 1
1 1 1
3 3 3 3
4
0 1 2 3 4 5

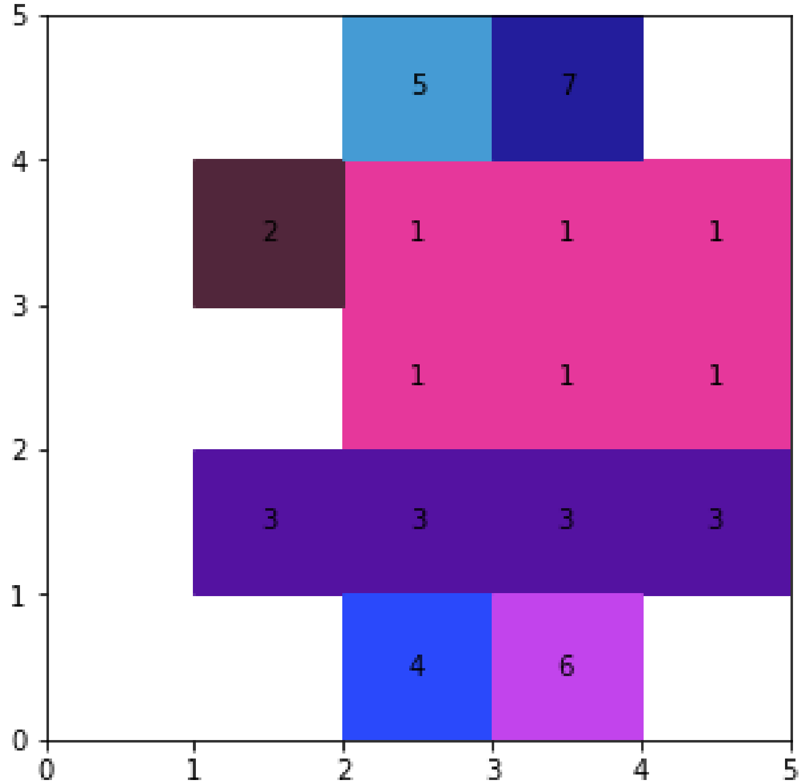

5
2
1
1
1
1
1
1
3
3
3
3
4
0
1
2
3
4
5
0
1
2
3
4
5



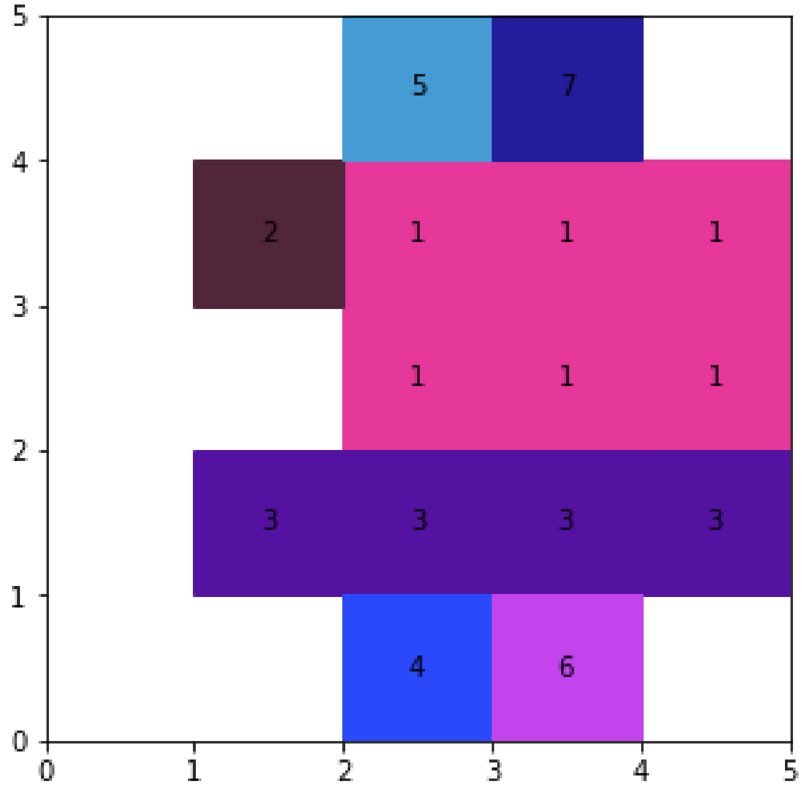
5
2
1
1
1
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5

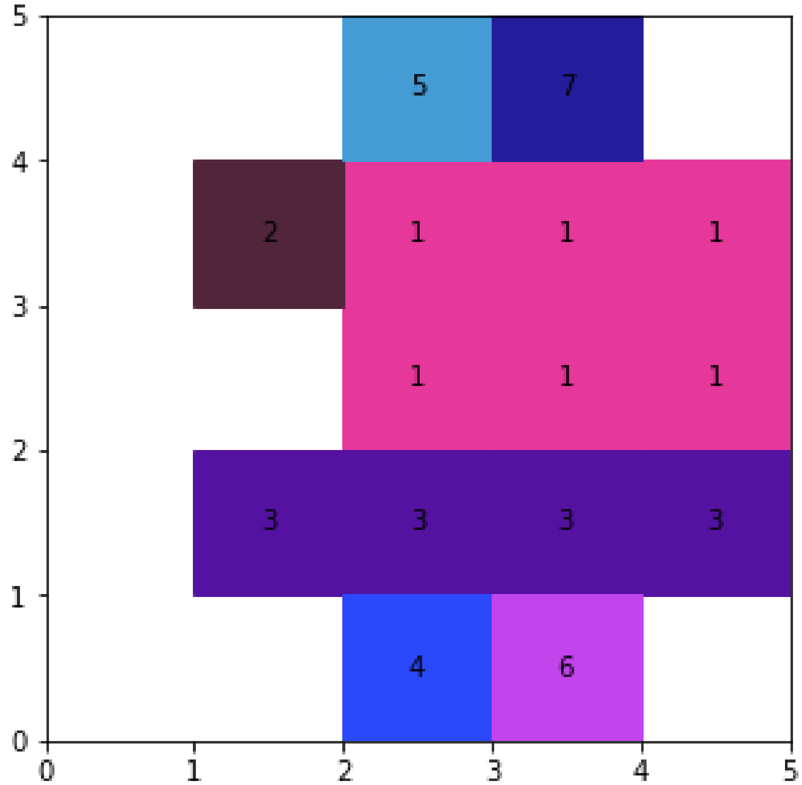


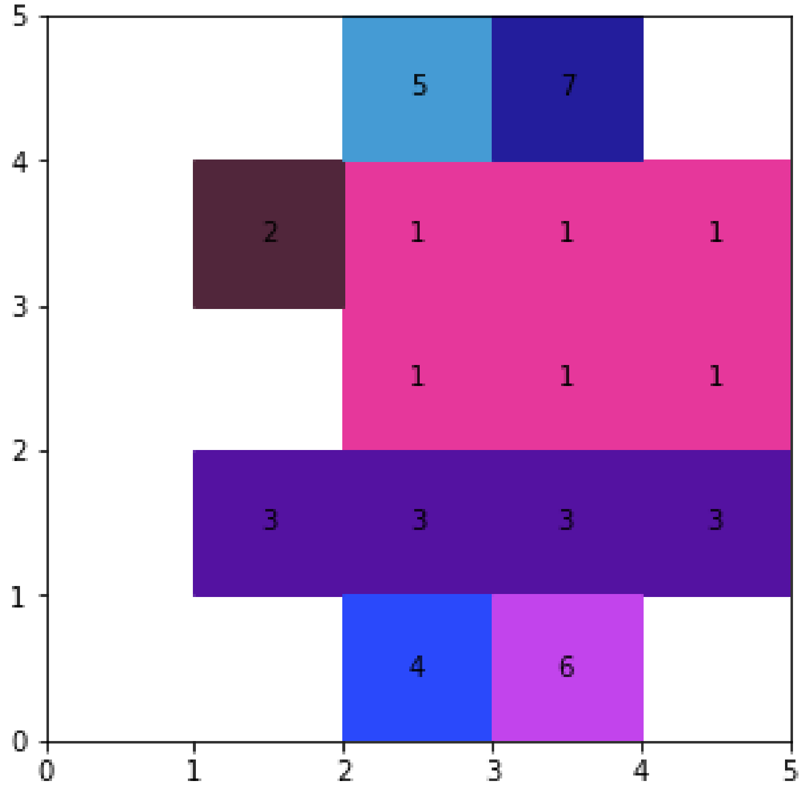
5
2
1
1
1
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5

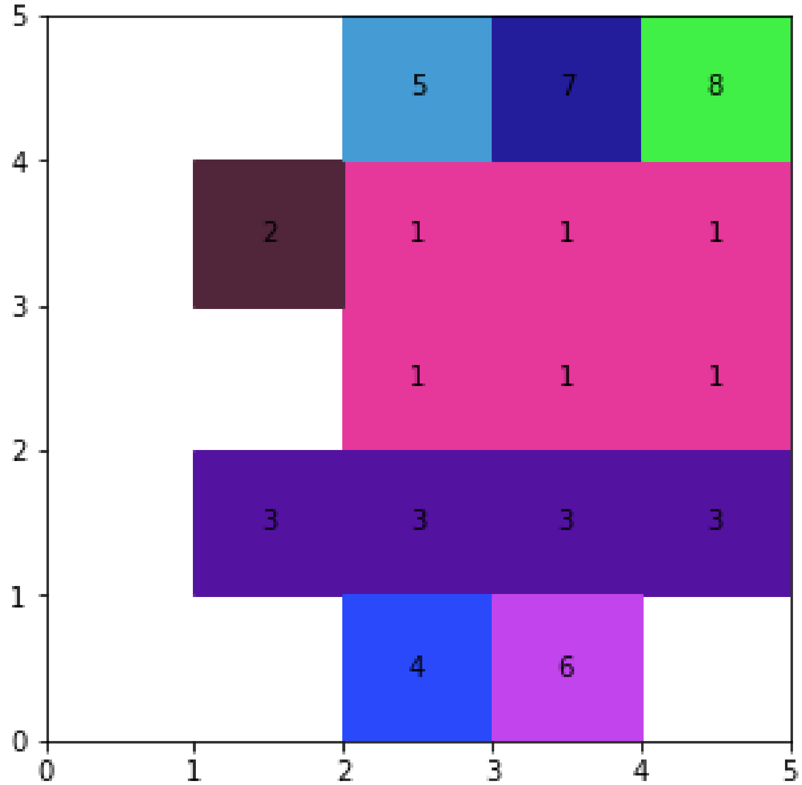


5
7
2
1
1
1
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5

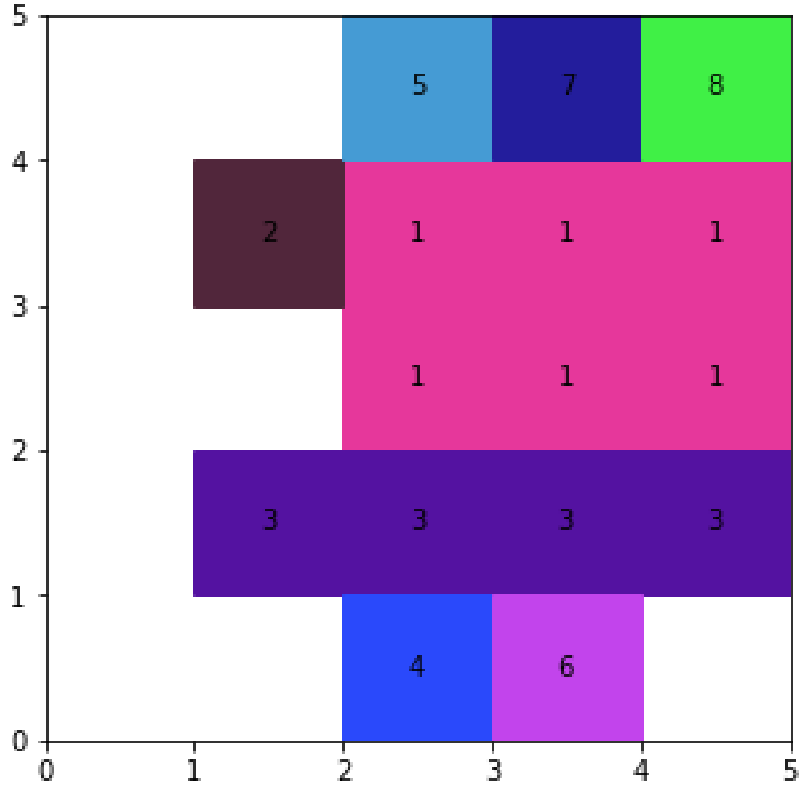

5
4
3
2
1
0
0
1
2
3
4
5
5
7
2
1
1
1
1
1
1
3
3
3
3
4
6


5
7
2
1
1
1
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5
0
1
2
3
4
5

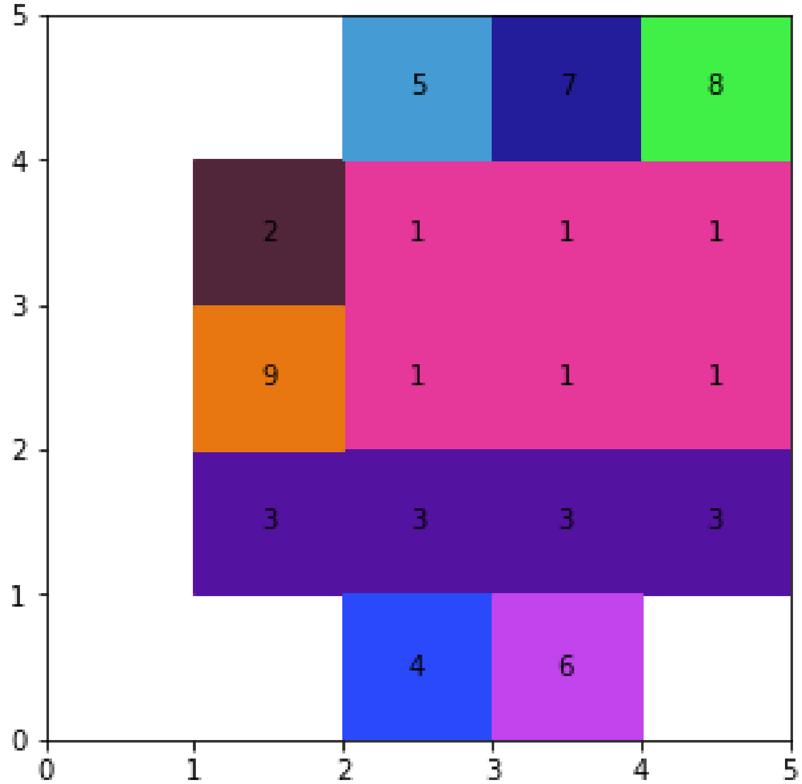

5
7
2
1
1
1
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5
0
1
2
3
4
5

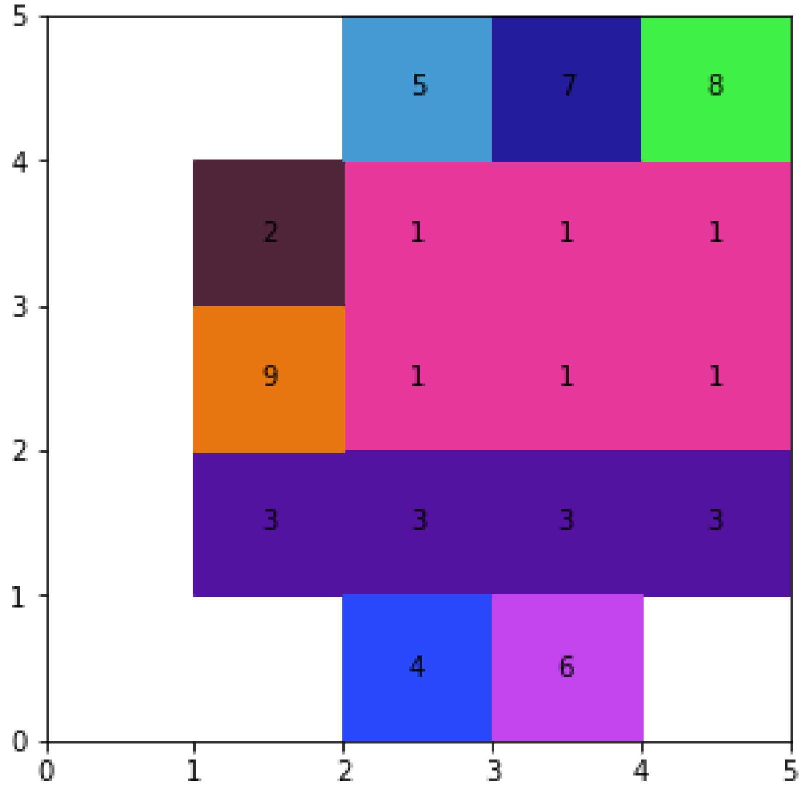


5
7
8
2
1
1
1
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5

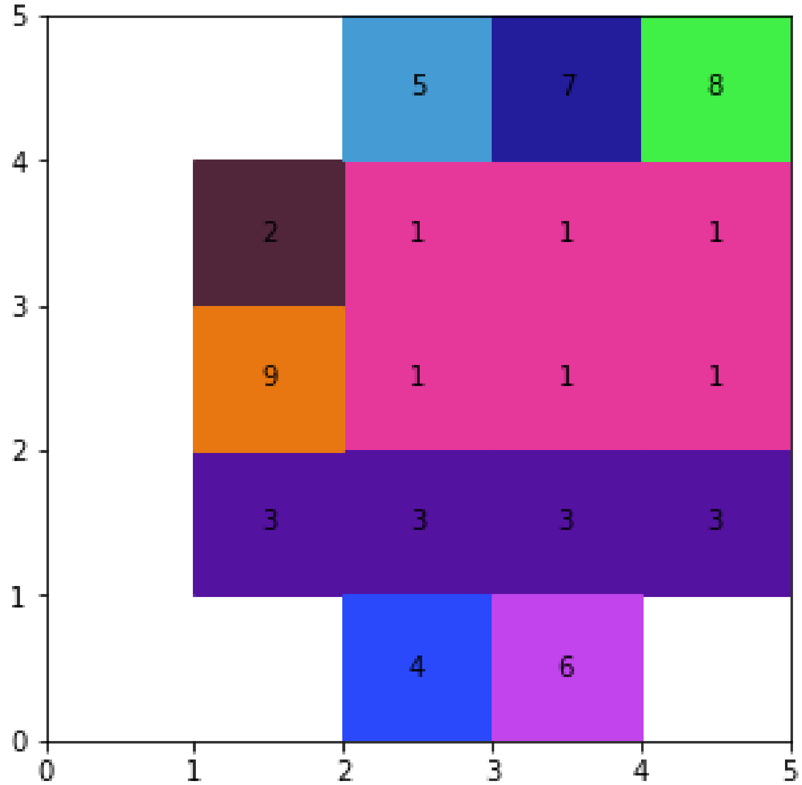


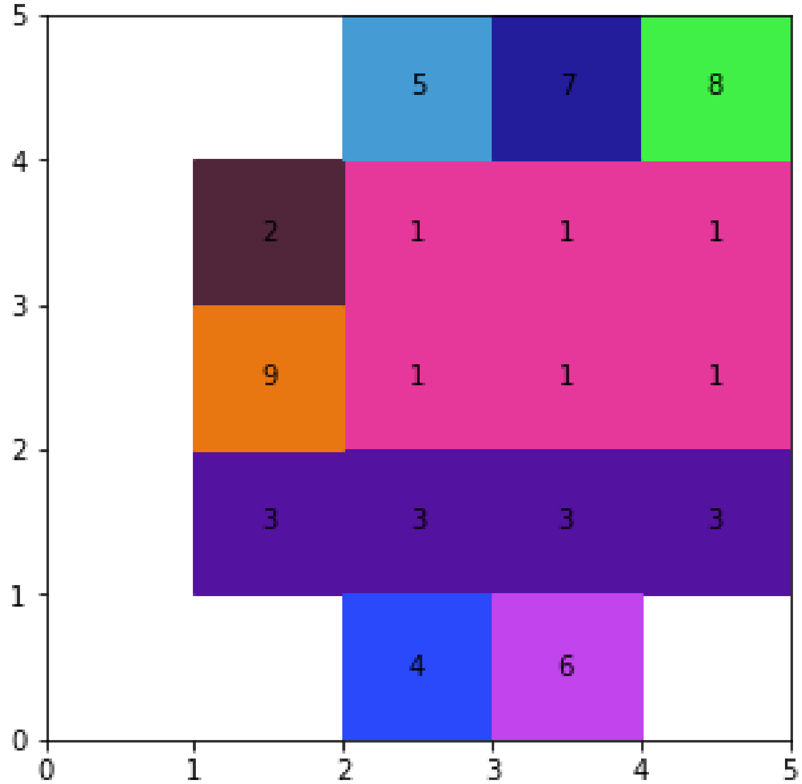
5
7
8
2
1
1
1
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5


5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5

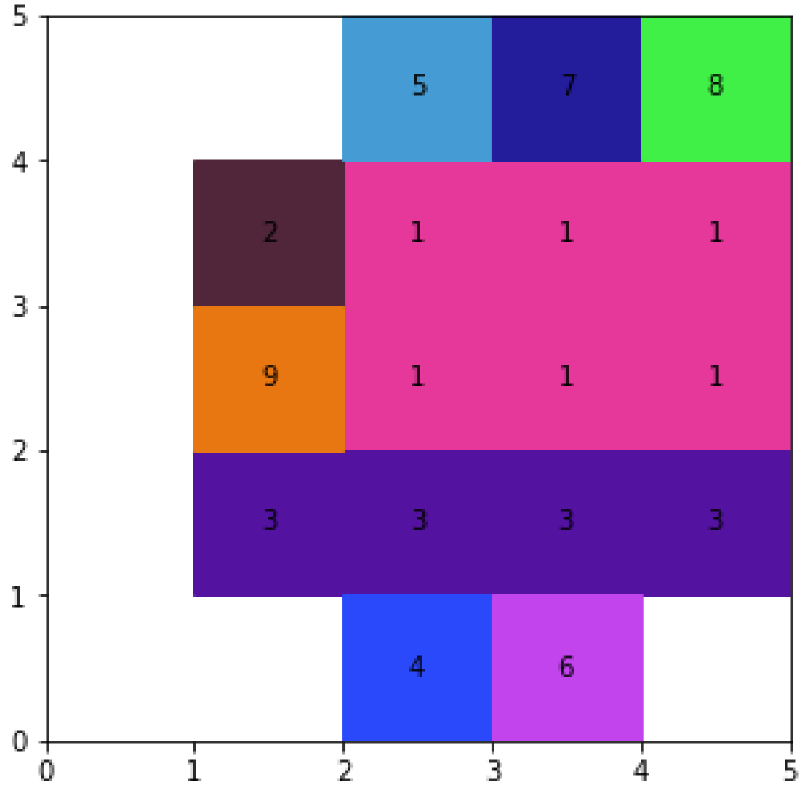


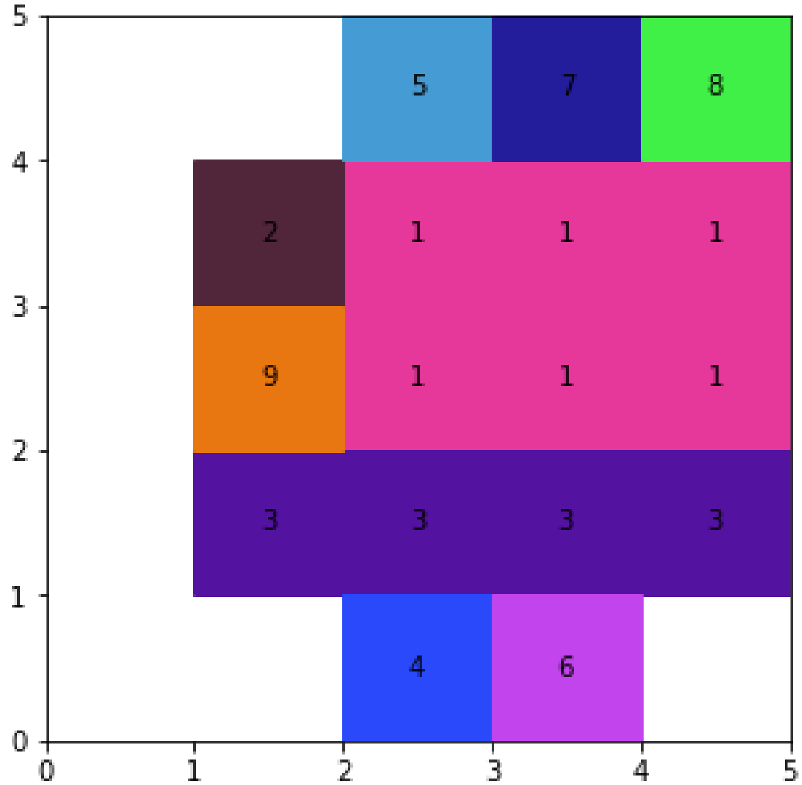
5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5

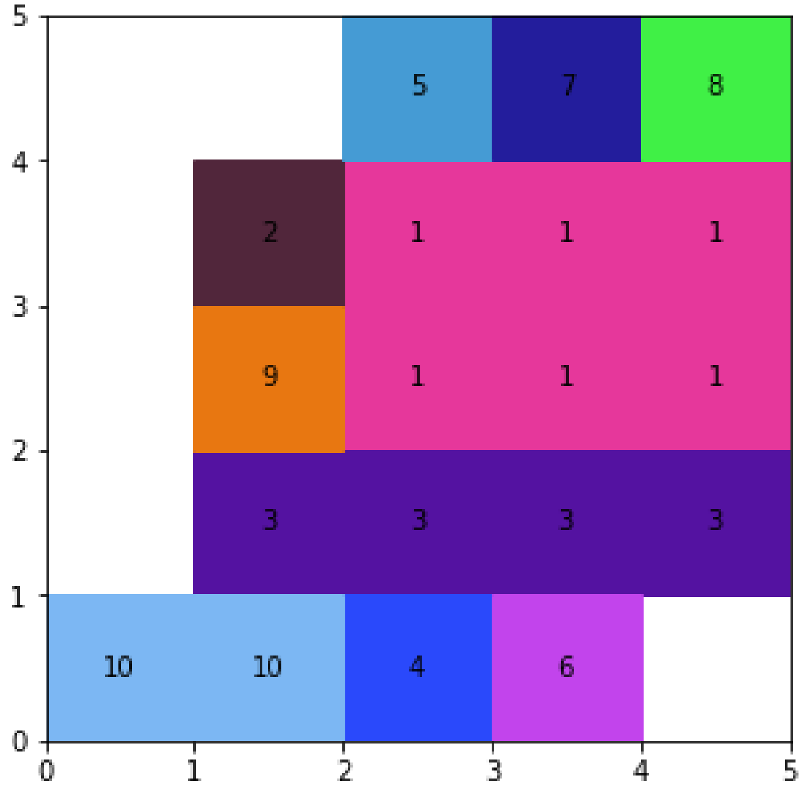

5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5



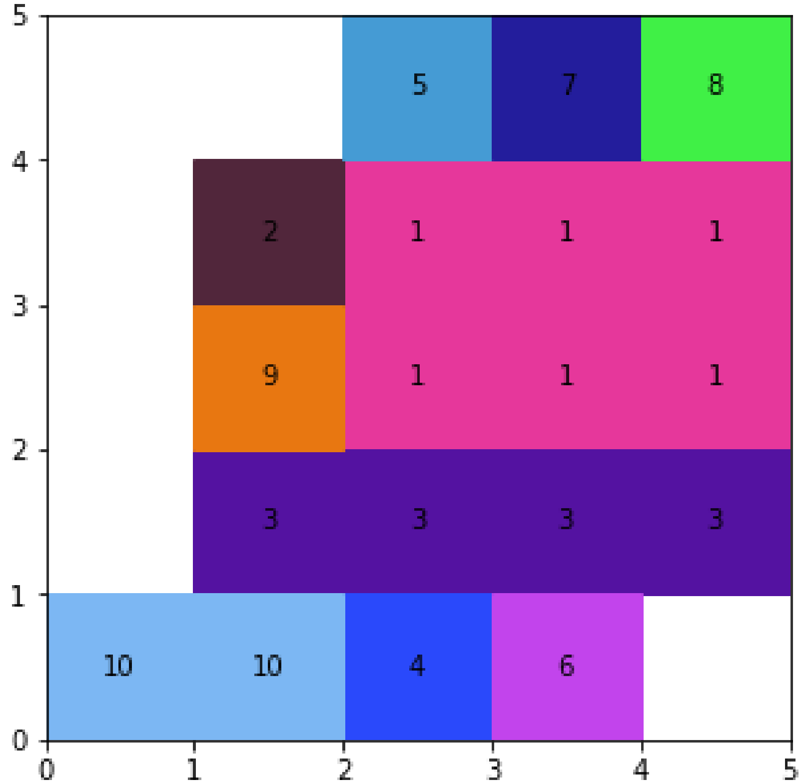
5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5

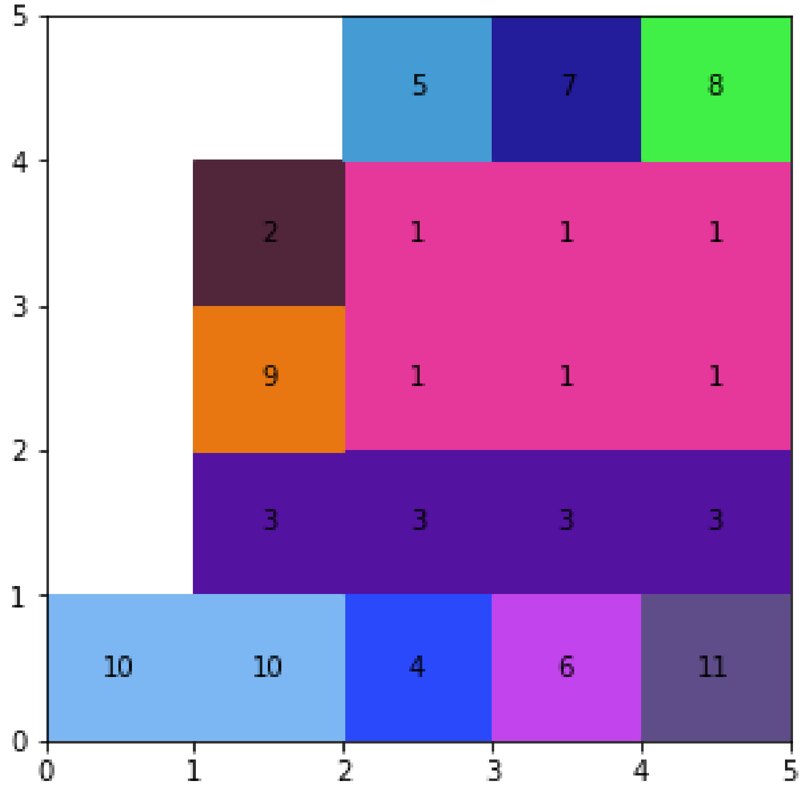

5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5
0
1
2
3
4
5

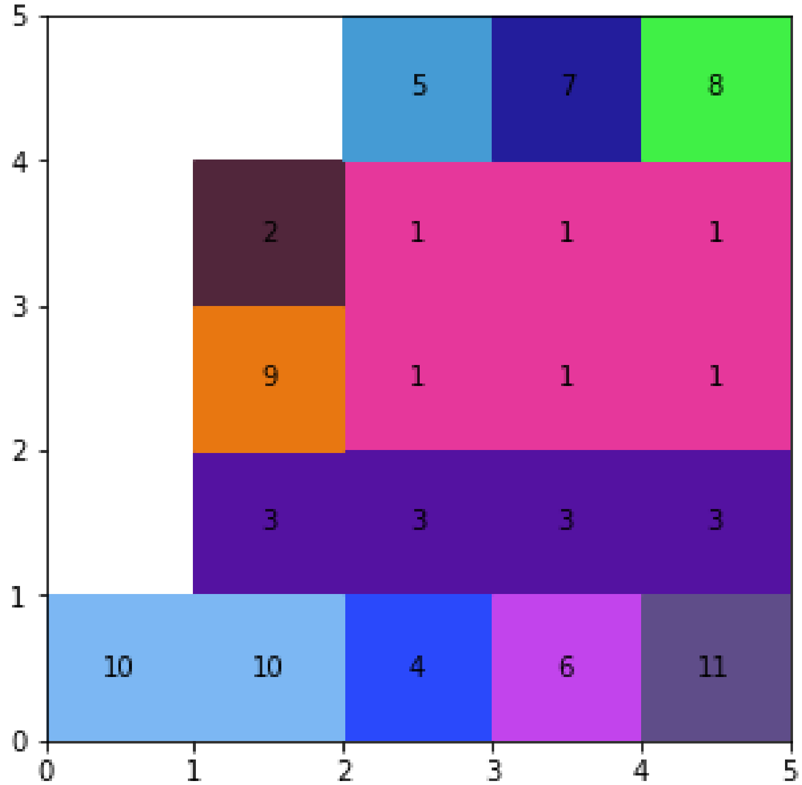

5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
4
6
0
1
2
3
4
5

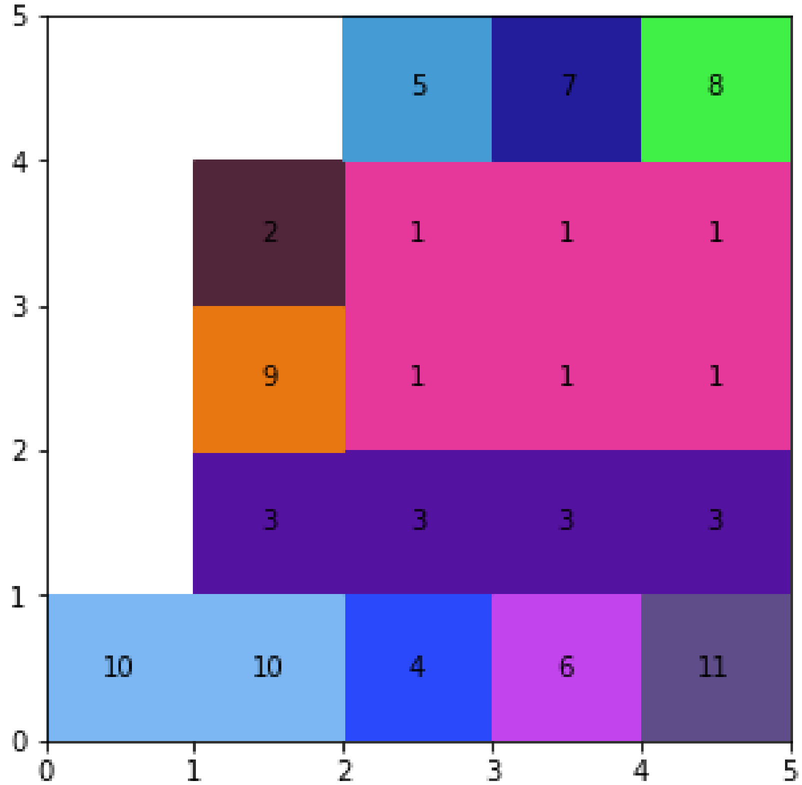


5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
10
10
4
6
0
1
2
3
4
5

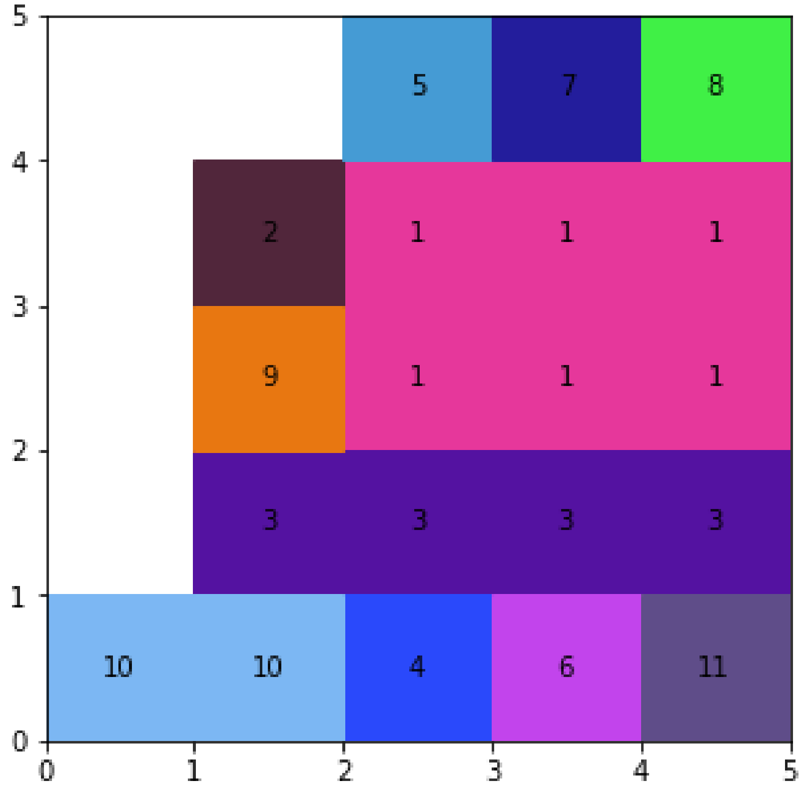

5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
10
10
4
6
0
1
2
3
4
5


5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5

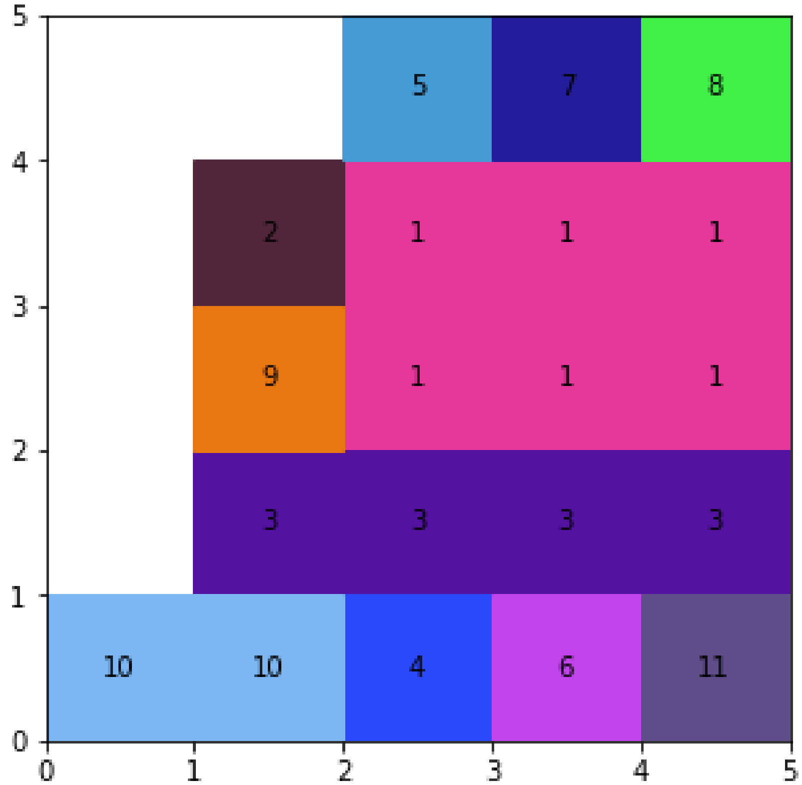

5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5

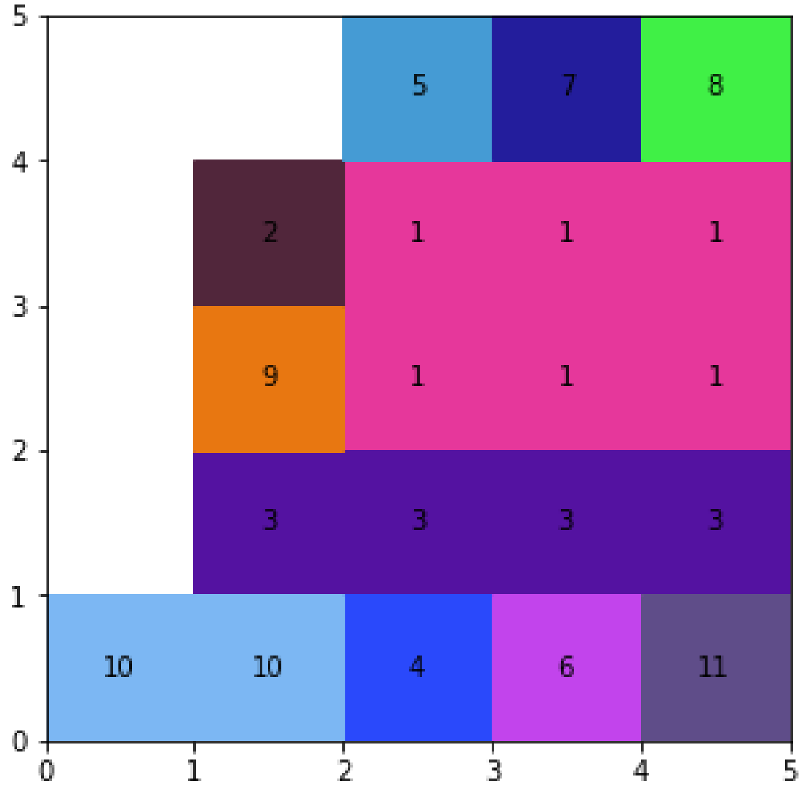


5
4
3
2
1
0
0
1
2
3
4
5
5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
10
10
4
6
11

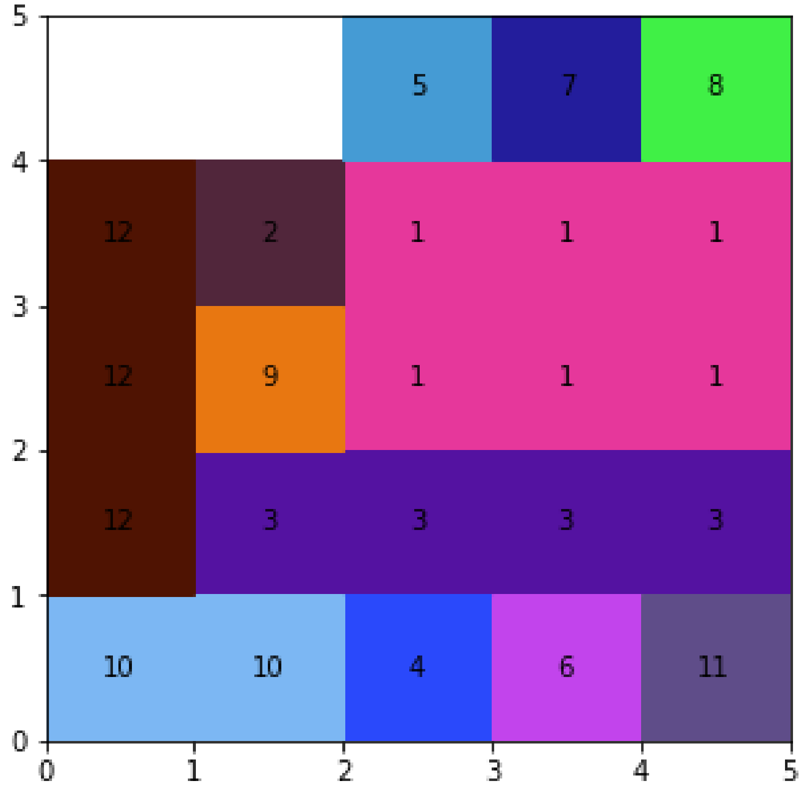


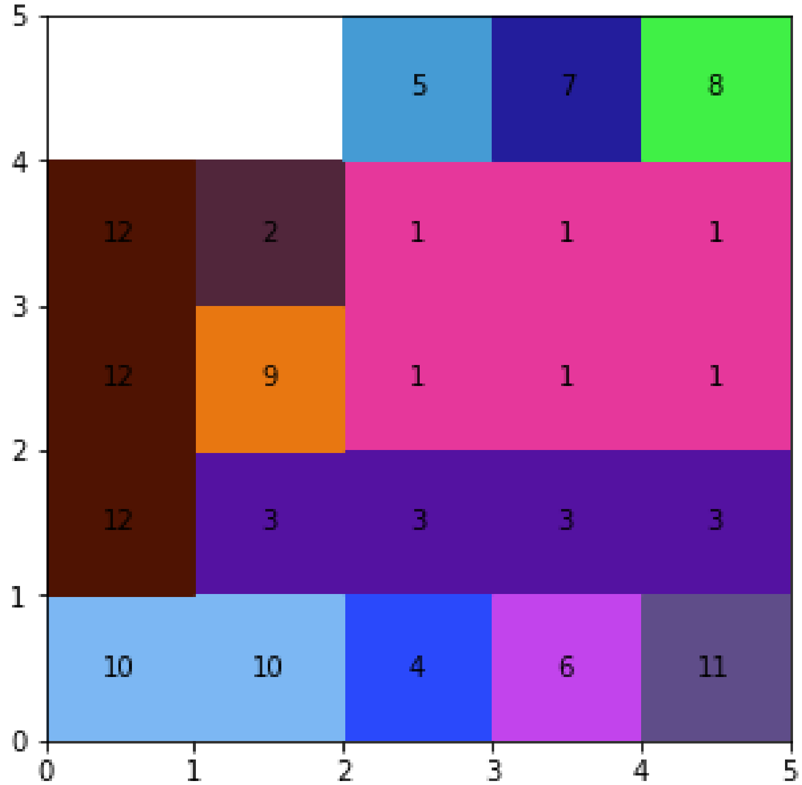
5
4
3
2
1
0
0
1
2
3
4
5
5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
10
10
4
6
11


5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5

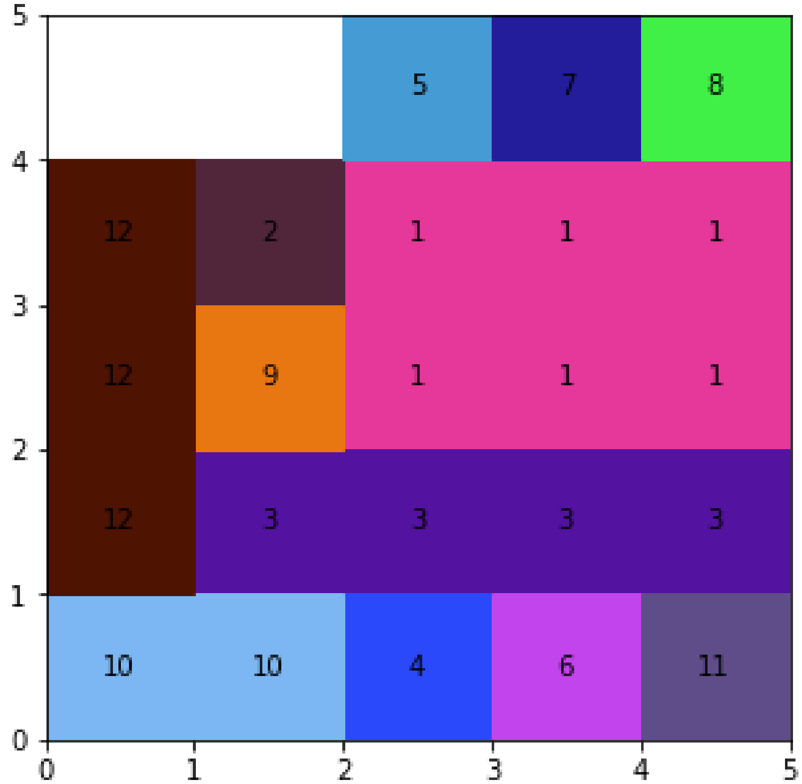


5
7
8
2
1
1
1
9
1
1
1
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5

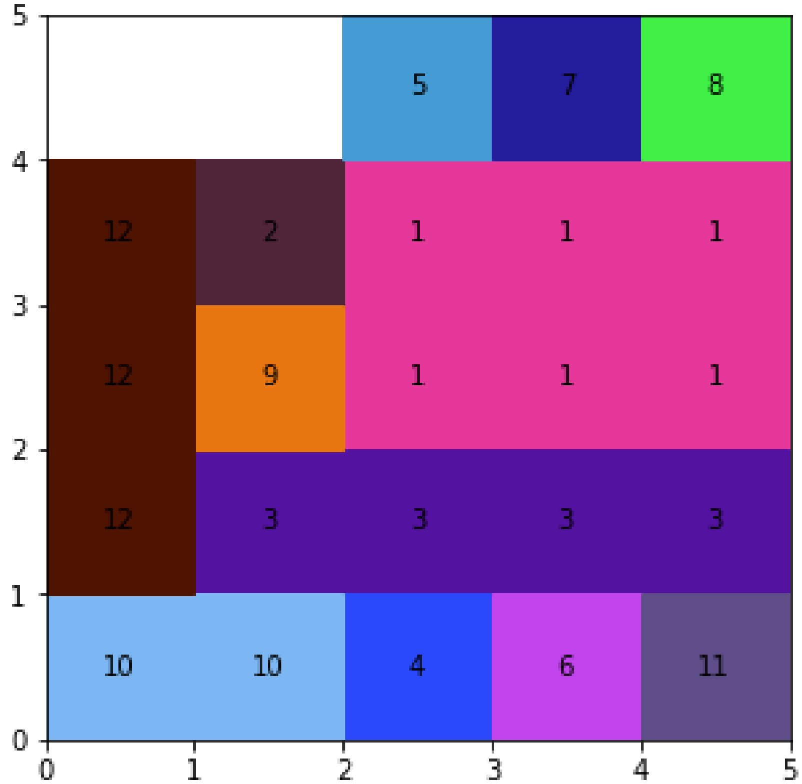

5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5

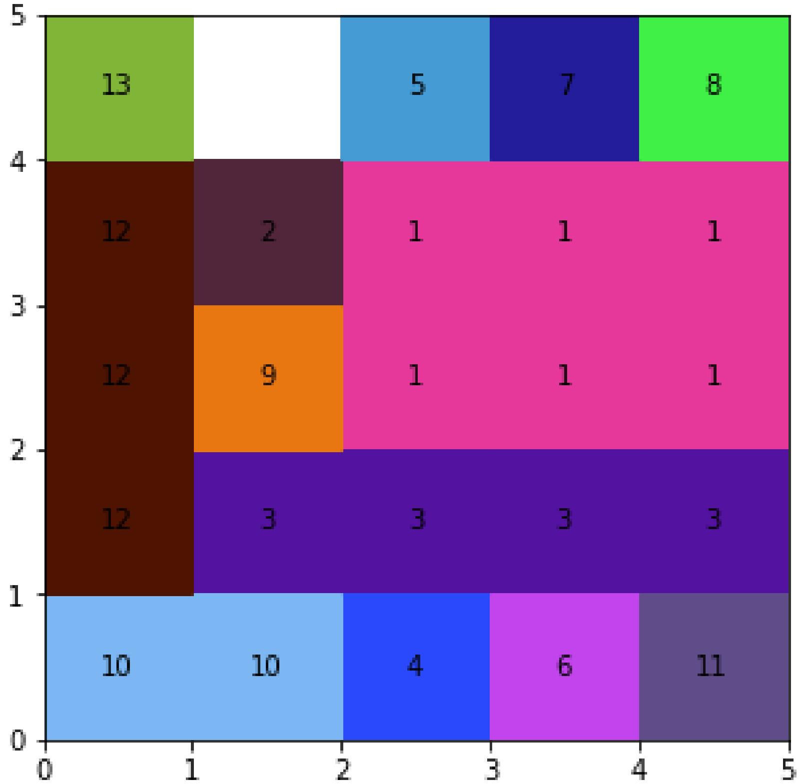


5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5

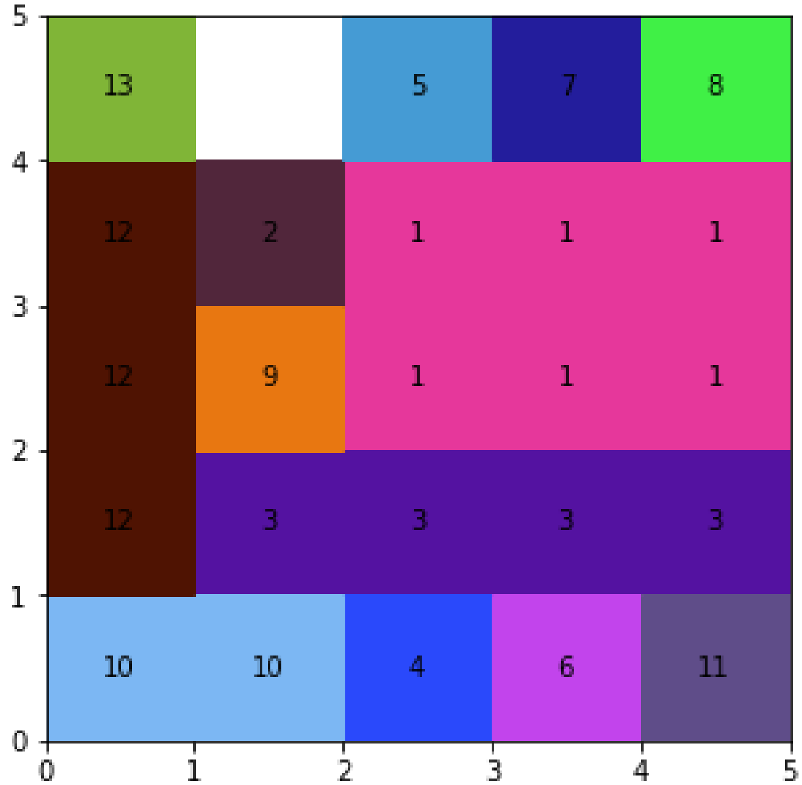


5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5

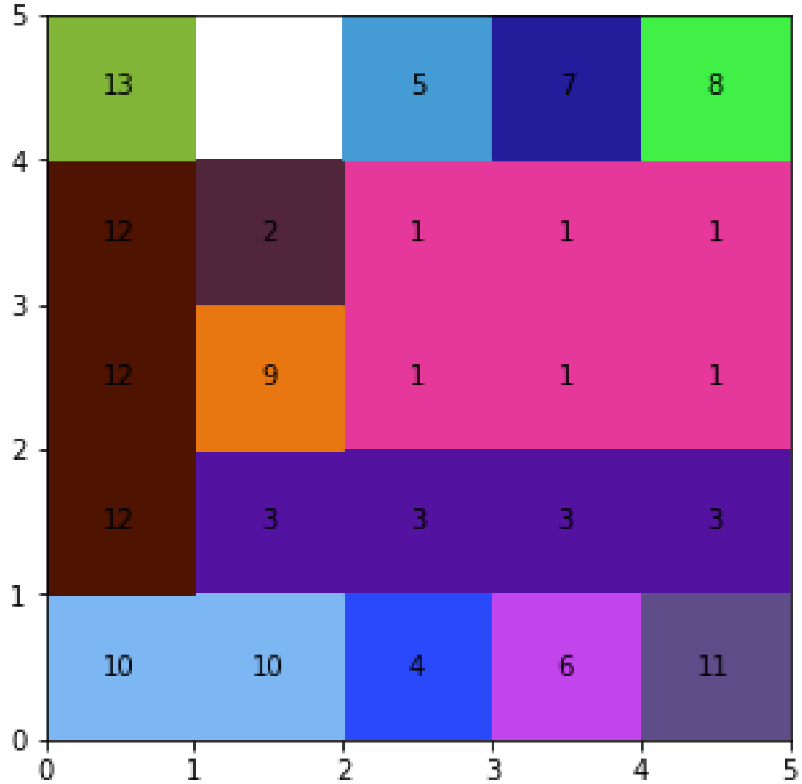

5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5

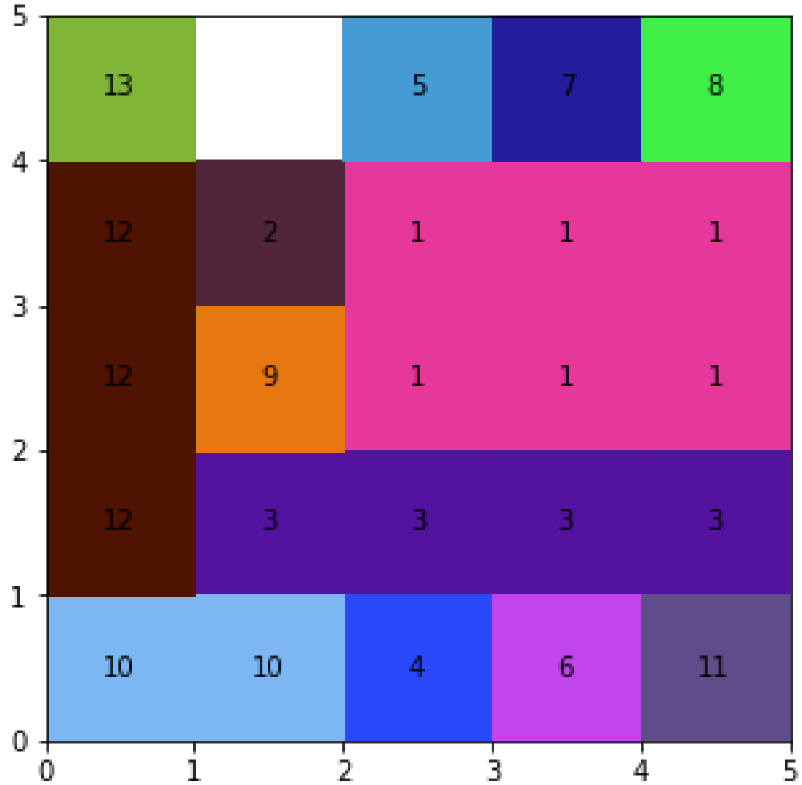


13
5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5

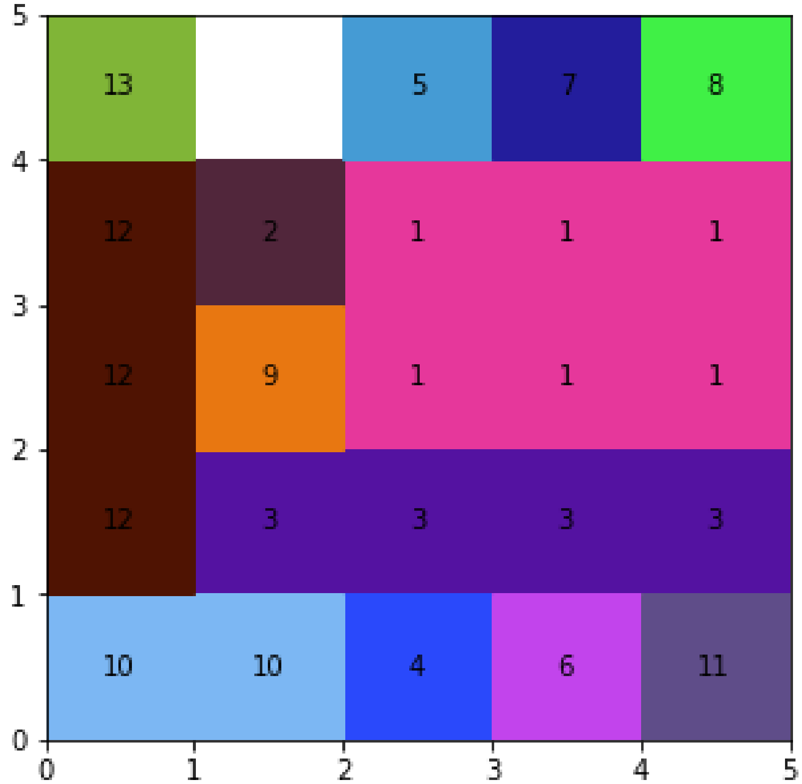


13
5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5

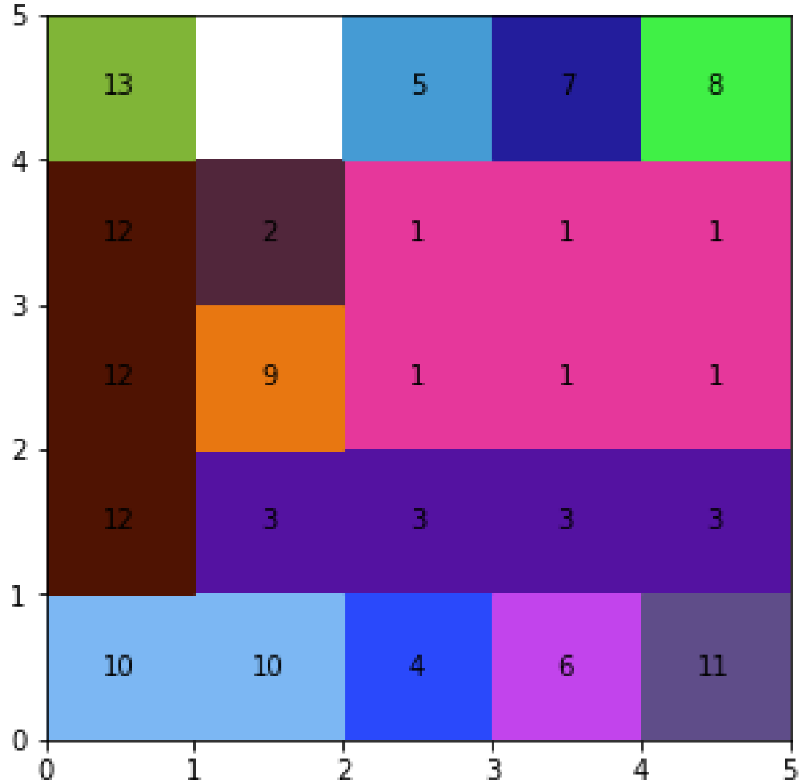

13
5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5



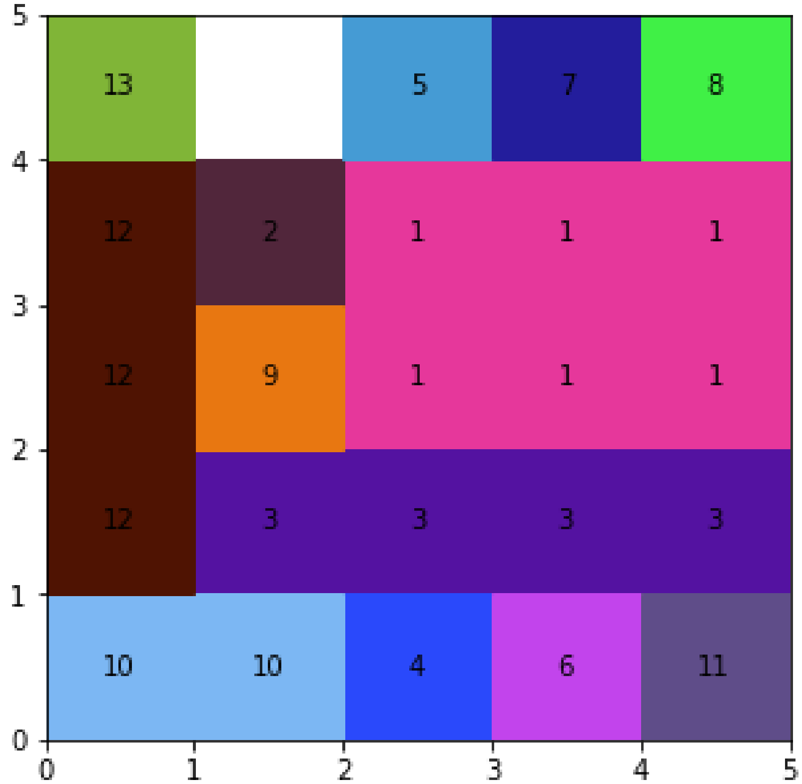
13
5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5



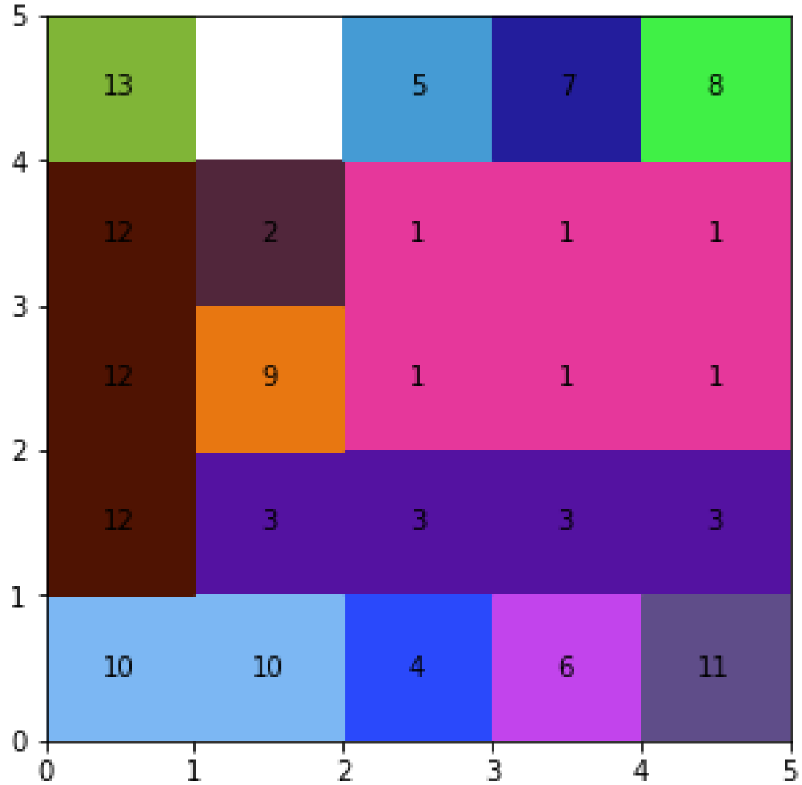
13
5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5

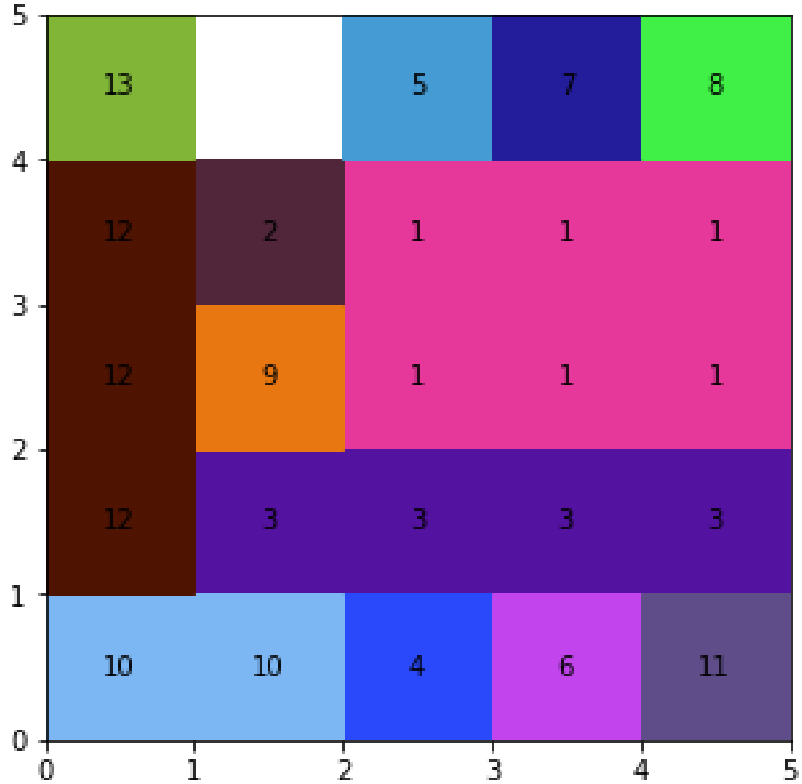


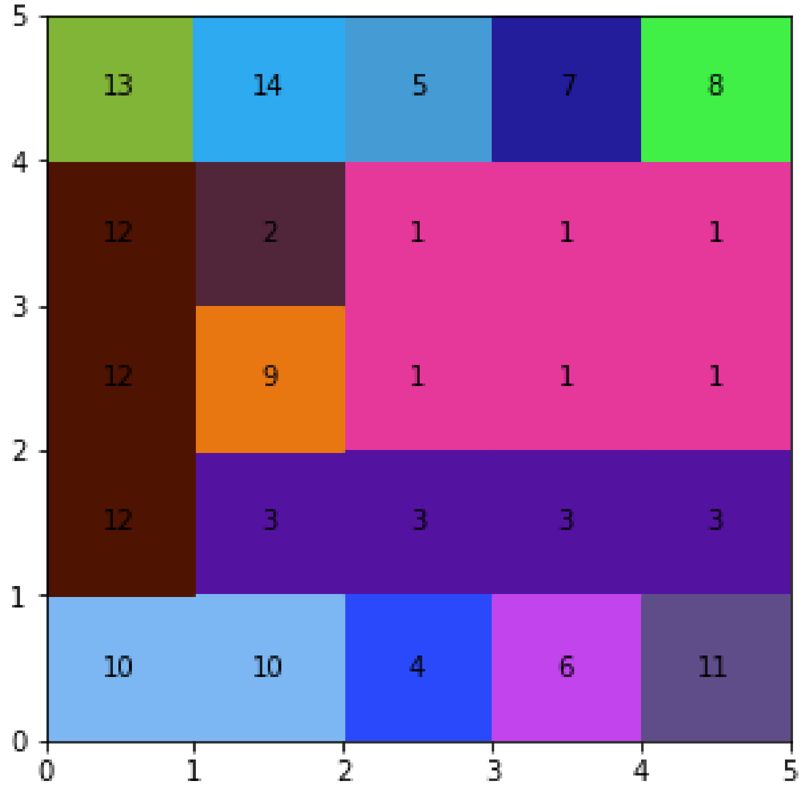
13
5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5



13
5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5
0
1
2
3
4
5


5
4
3
2
1
0
13
5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11
0
1
2
3
4
5


13
5
7
8
12
2
1
1
1
12
9
1
1
1
12
3
3
3
3
10
10
4
6
11




```
On average across 10 experiments, it takes 14.0 communication towers before full coverage is obt
```

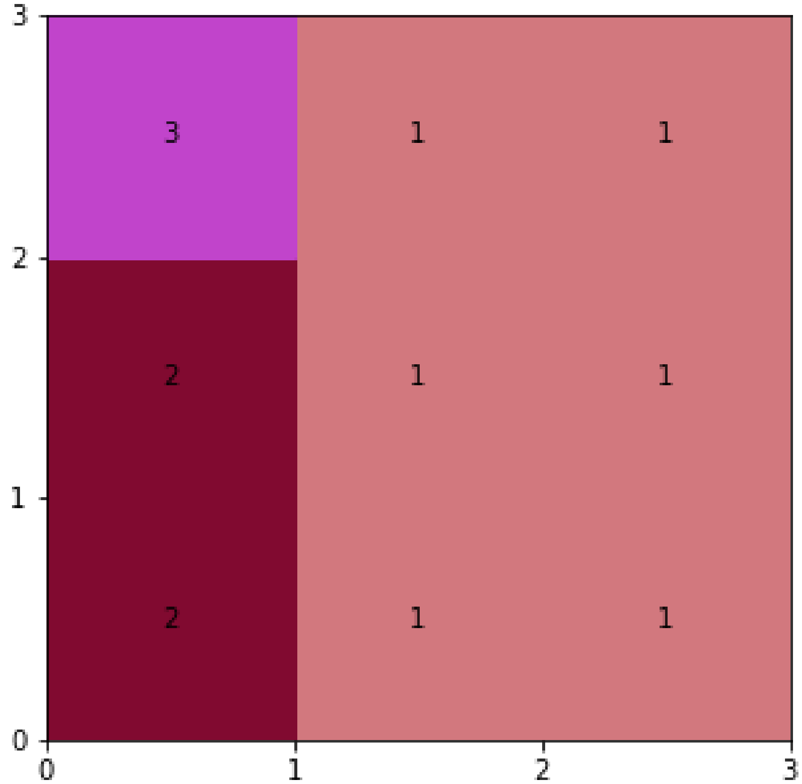
(3) On average, how many communications towers are required before full coverage is obtained? (additional analysis)

```
As shown, the code shows across variable-valued number of experiments, I have printed out the nu
of experiments and average number of towers in each iteration. On the plot, it shows that, while
initially there are outliers, the law of large number dictates that, as the number of experiment
larger, the average number of towers would approach to some number, and evidently the variabilit
in the average number of towers would also decrease as the number of experiments incerase. Thus,
say that the number of experiments is inversely proportional to the variability of the average
number towers across various iterations and the state of the map is shown step-by-step.
Note: The numbers and colors show the type of color and tower numbers associated with the rectan
```

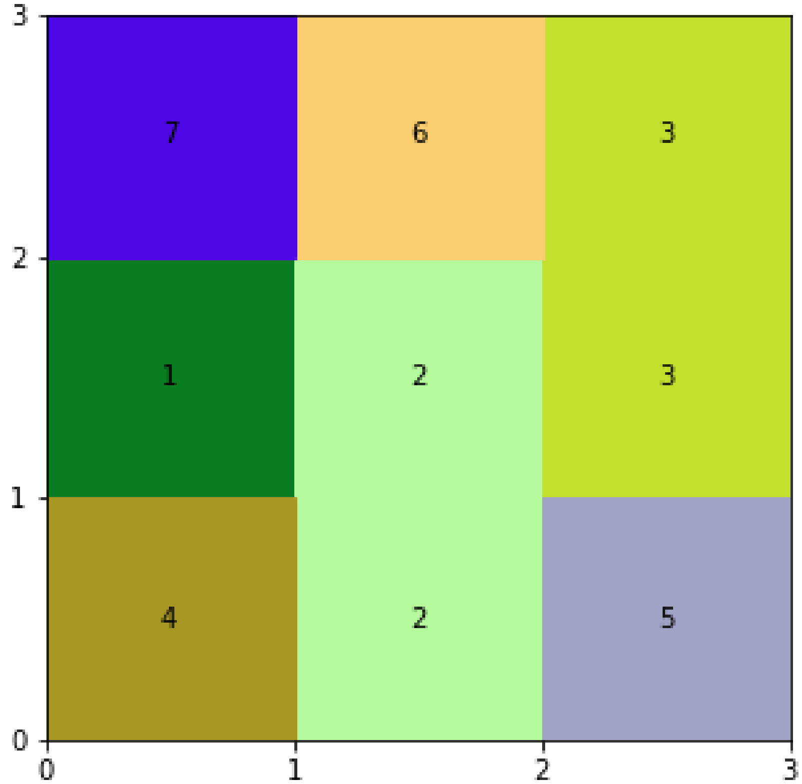
```
In [52]: # define the length and width dimensions of map and number of iterations
         xLim = yLim = 3
         numIteration = 200
         myList = list()
         for i in range(1, numIteration):
             obj = MyClass(xLim, yLim)
             numExperiment = (50 * i)
```

```
        maxTrial = 100
        aList = list([0] * (numExperiment))
        currentSum = 0
        # after this loop found numExperiment full maps
        for i in range(0, numExperiment):
        # after this loop we found one full map
            for j in range(0, maxTrial):
                startLength = random.randint(0, xLim - 2)
                startWidth = random.randint(0, yLim - 2)
                endLength = random.randint(startLength + 1, xLim)
                endWidth = random.randint(startWidth + 1, yLim)
                value = obj.add(startLength, startWidth, endLength, endWidth)
                if value == 0:
                    currentSum += obj.tower
                    break
        # put (# experiment, avg # of towers) tuples in myList
        myTuple = (numExperiment, currentSum * 1.0/numExperiment)
        myList.append(myTuple)
        print ('On average across ' + str(numExperiment) + ' experiments, it takes ' + str(
        obj.displayMap()
    plt.plot(*zip(*myList))
    plt.title('My Plot')
    plt.xlabel('Number of Experiments')
    plt.ylabel('Average Number of Towers')
    plt.show()
```

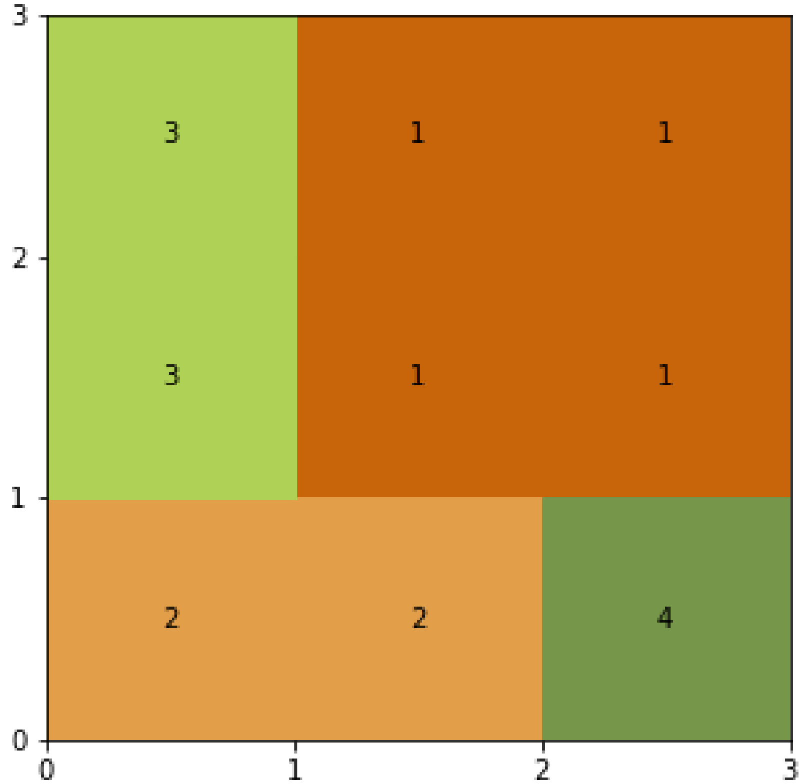
```
On average across 50 experiments, it takes 3.0 communication towers before full coverage is obta
```



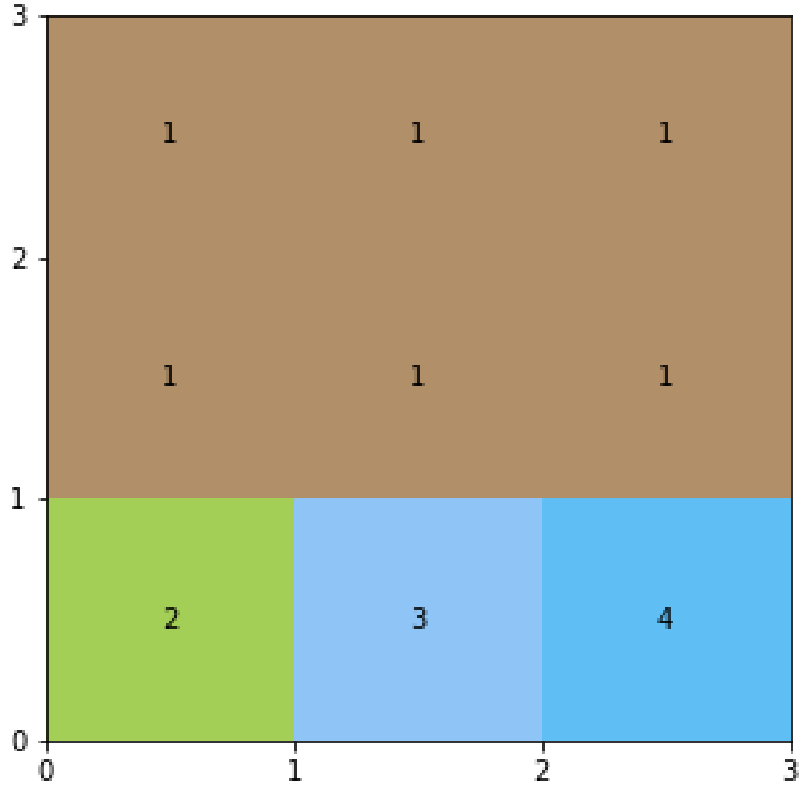

```
On average across 100 experiments, it takes 7.0 communication towers before full coverage is obt
```



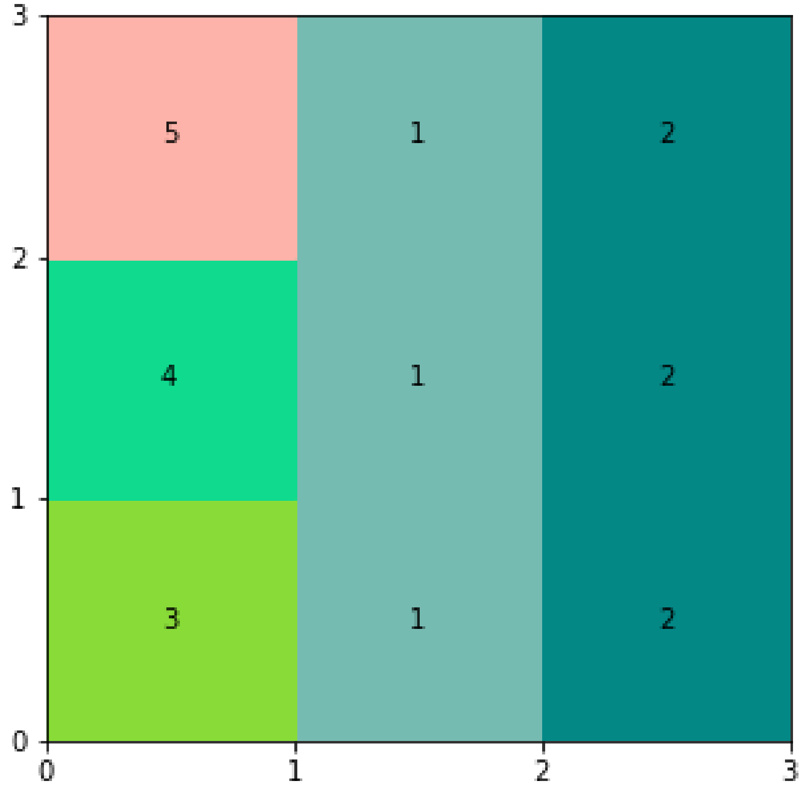

```
On average across 150 experiments, it takes 4.0 communication towers before full coverage is obt
```



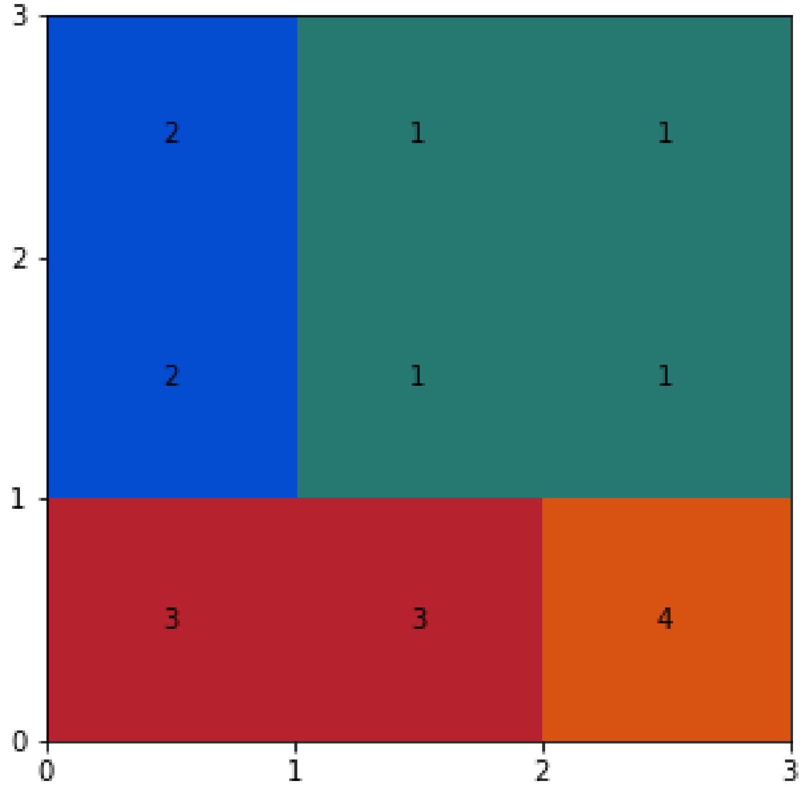
On average across 200 experiments, it takes 4.0 communication towers before full coverage is obt



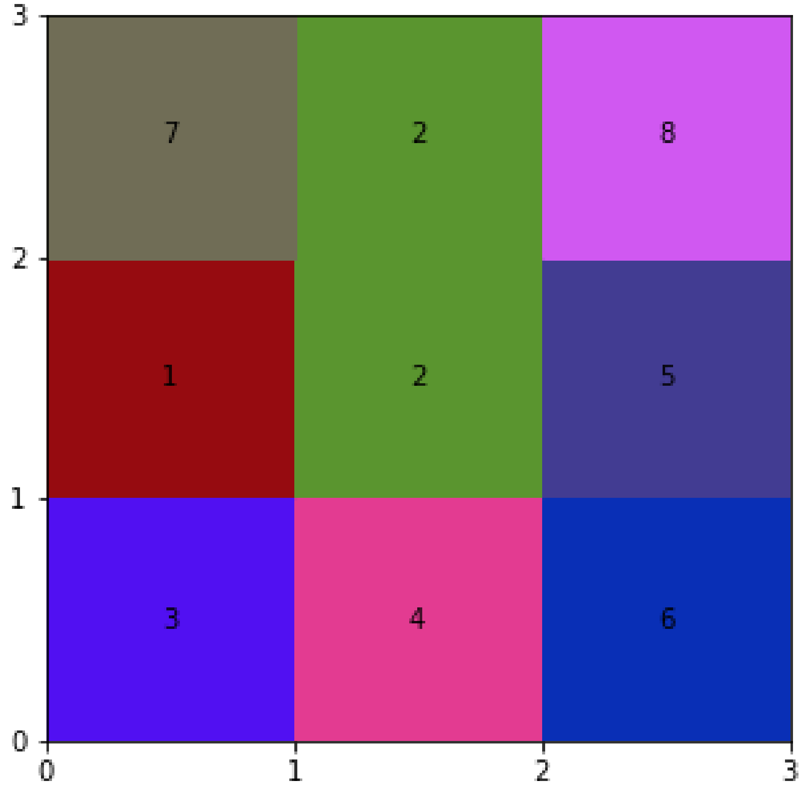

```
On average across 250 experiments, it takes 5.0 communication towers before full coverage is obt
```



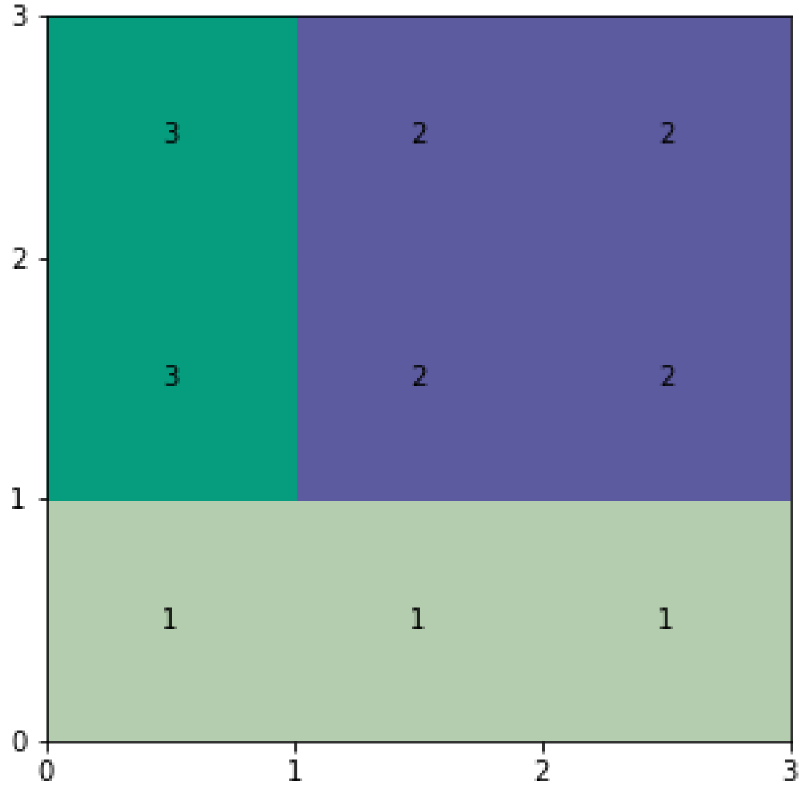
On average across 300 experiments, it takes 4.0 communication towers before full coverage is obt



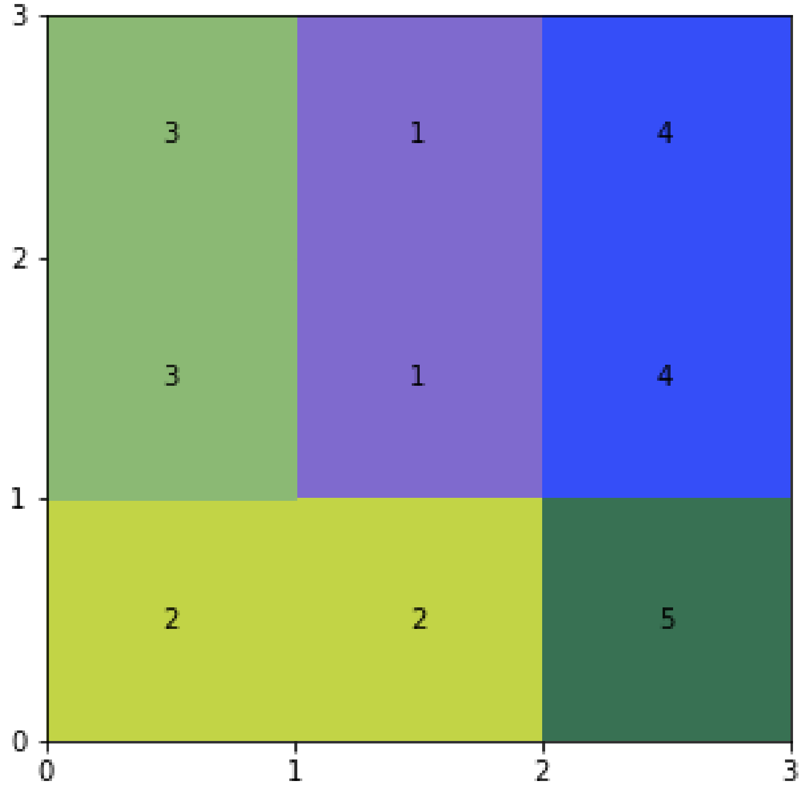

```
On average across 350 experiments, it takes 8.0 communication towers before full coverage is obt
```



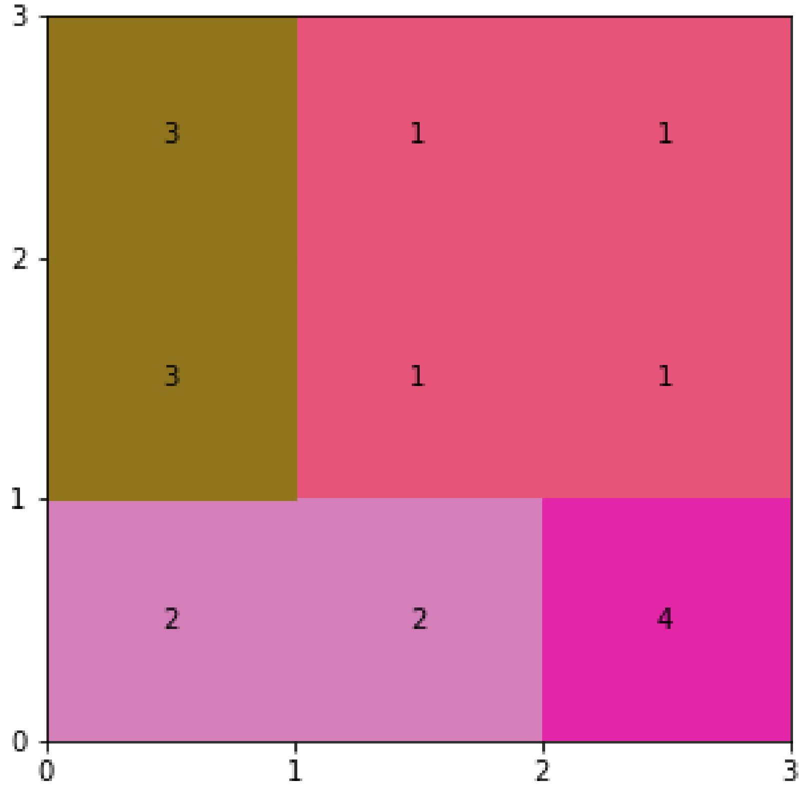

```
On average across 400 experiments, it takes 3.0 communication towers before full coverage is obt
```



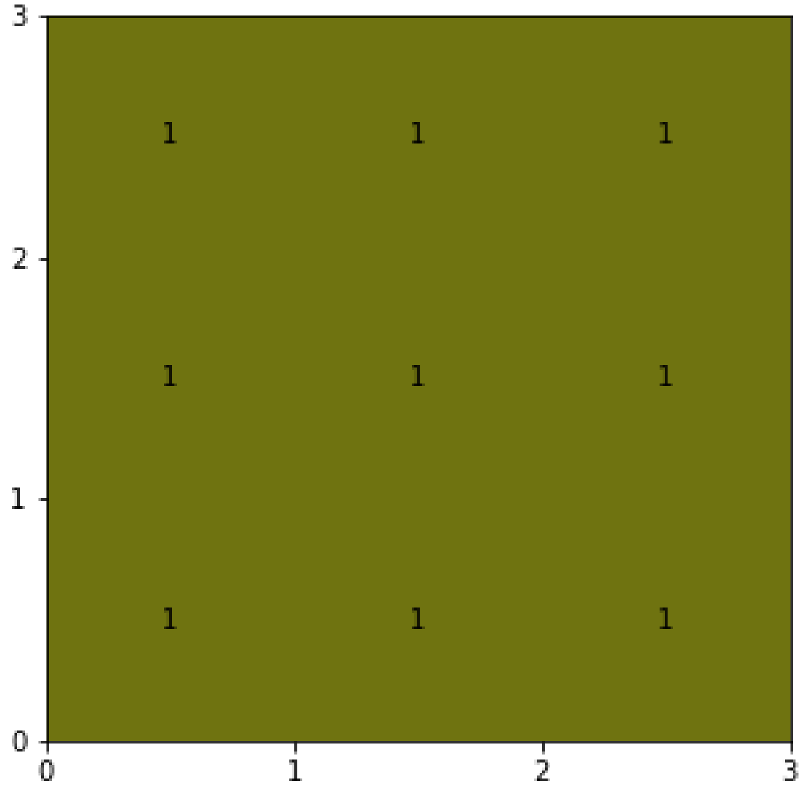
On average across 450 experiments, it takes 5.0 communication towers before full coverage is obt



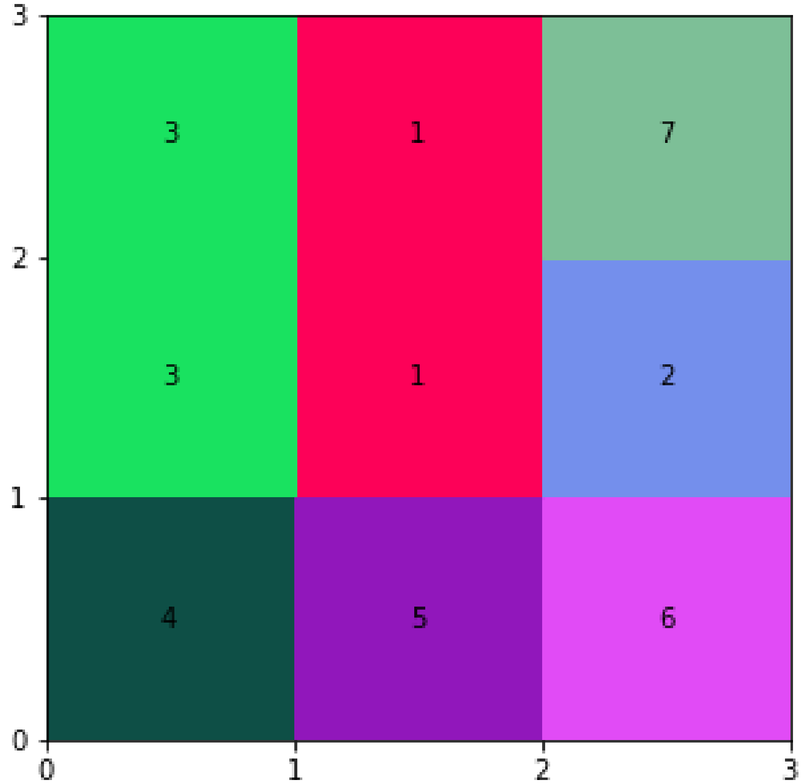
`On average across 500 experiments, it takes 4.0 communication towers before full coverage is obt`



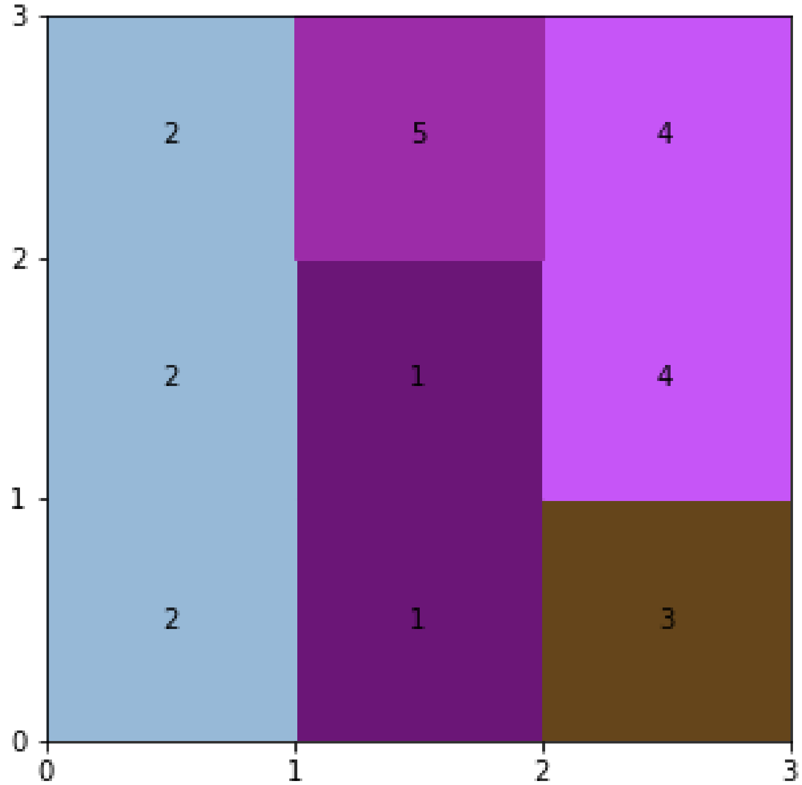
On average across 550 experiments, it takes 1.0 communication towers before full coverage is obt



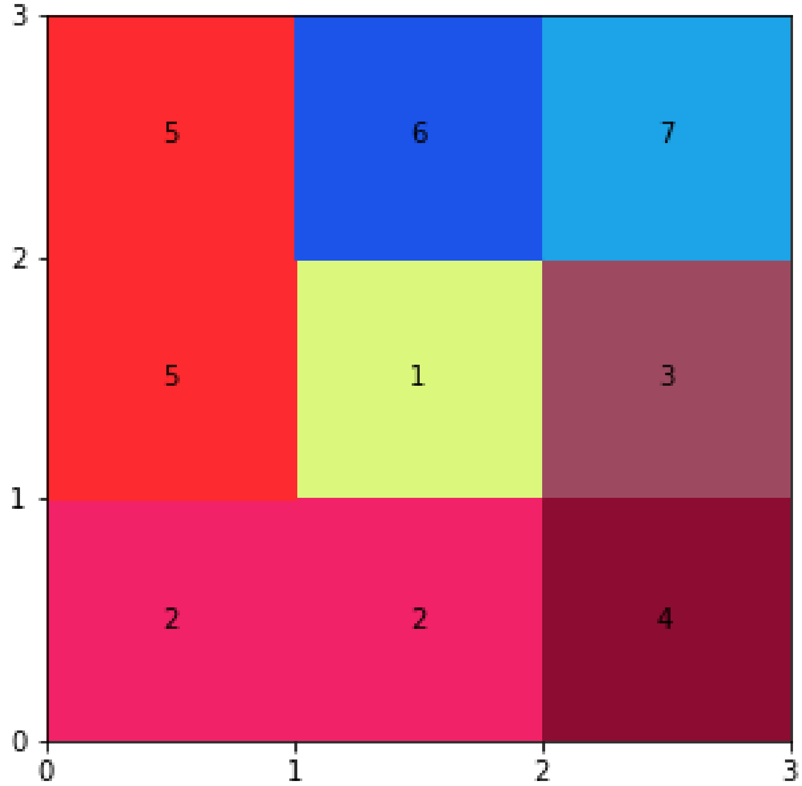

```
On average across 600 experiments, it takes 7.0 communication towers before full coverage is obt
```



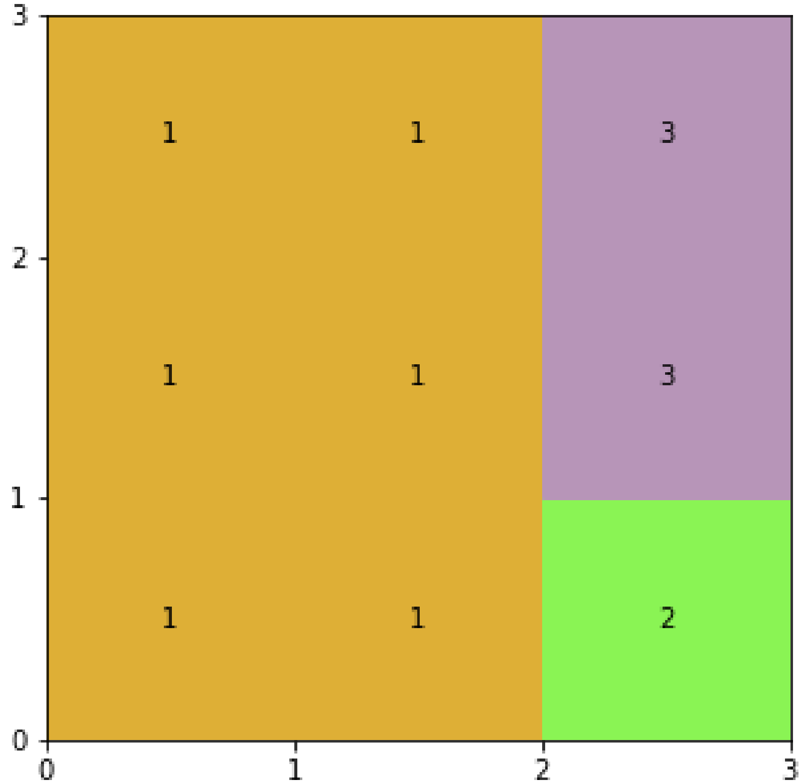

```
On average across 650 experiments, it takes 5.0 communication towers before full coverage is obt
```



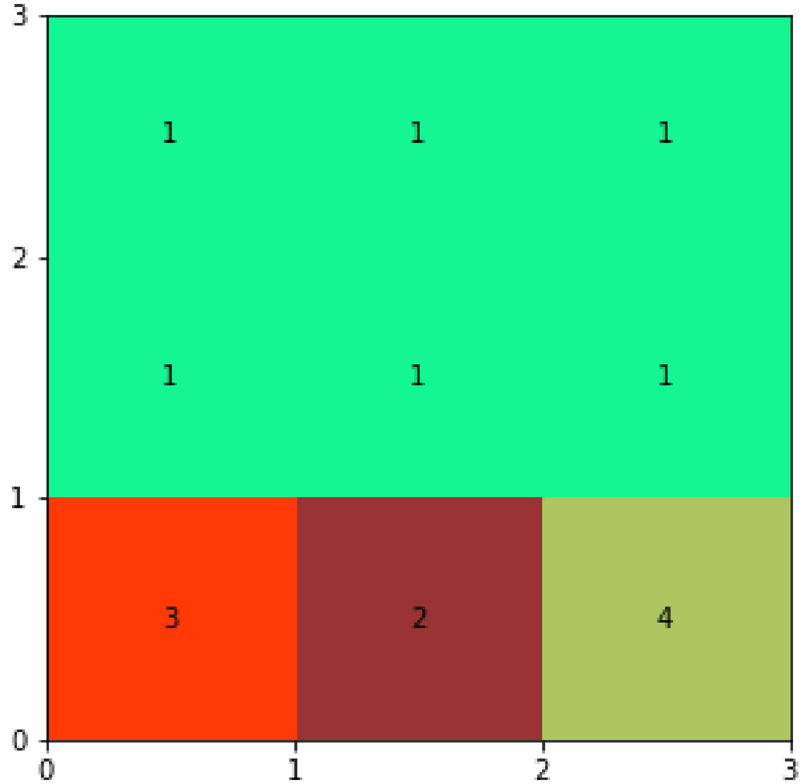

```
On average across 700 experiments, it takes 7.0 communication towers before full coverage is obt
```



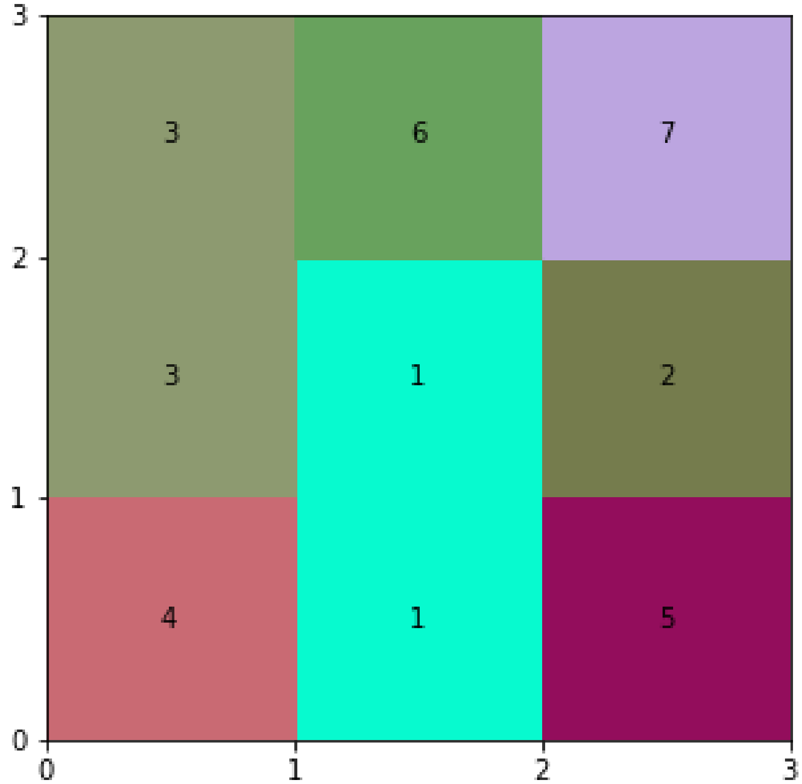
On average across 750 experiments, it takes 3.0 communication towers before full coverage is obt



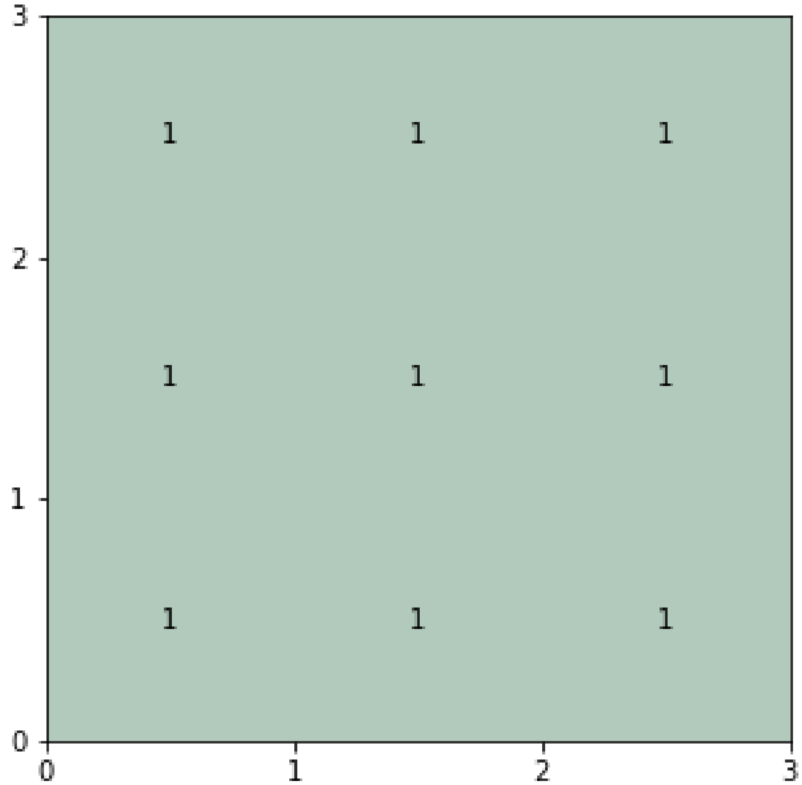

```
On average across 800 experiments, it takes 4.0 communication towers before full coverage is obt
```



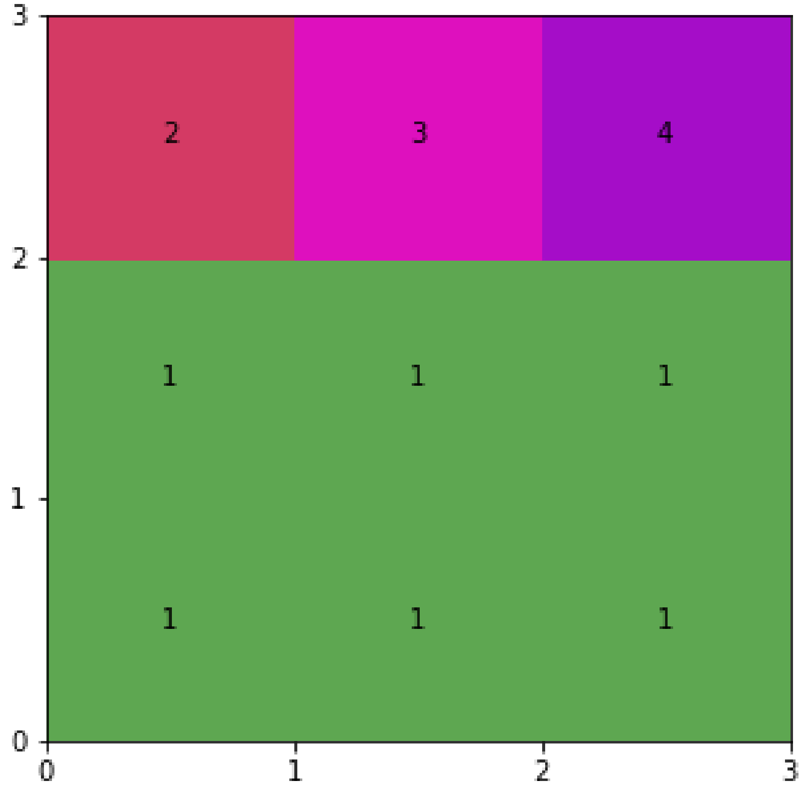
On average across 850 experiments, it takes 7.0 communication towers before full coverage is obt



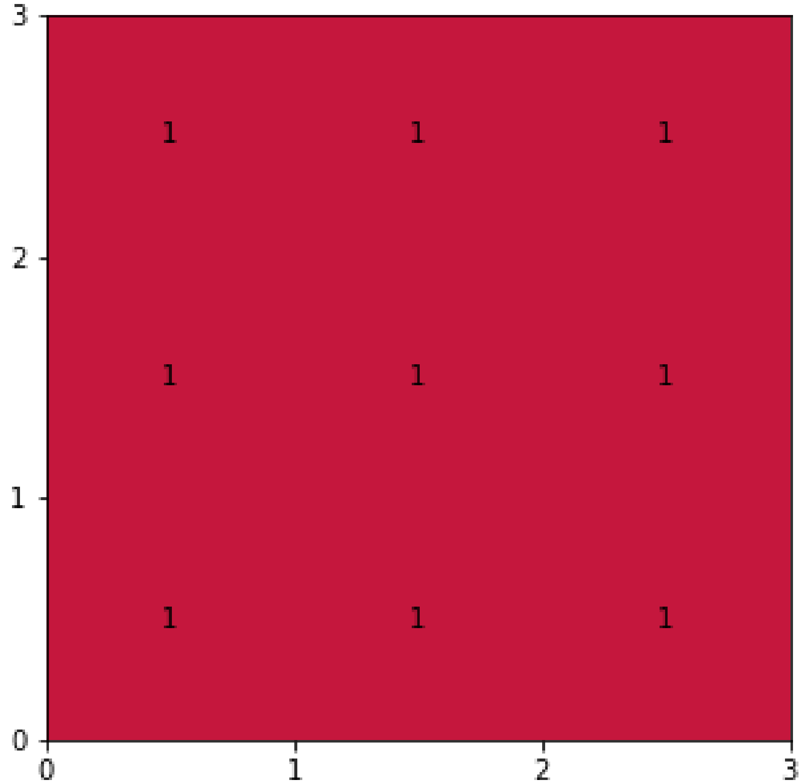
On average across 900 experiments, it takes 1.0 communication towers before full coverage is obt



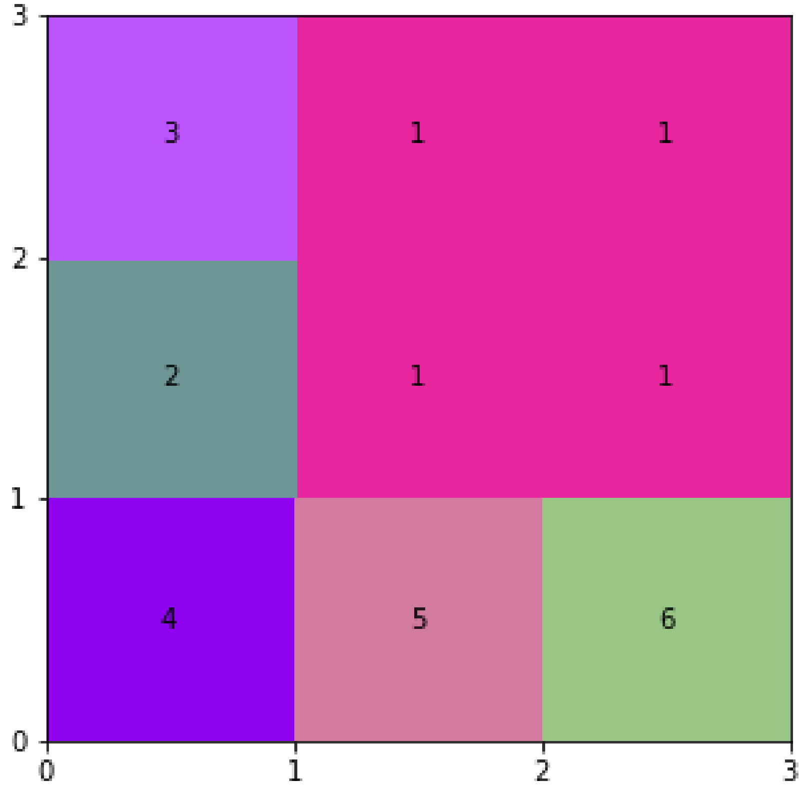

```
On average across 950 experiments, it takes 4.0 communication towers before full coverage is obt
```



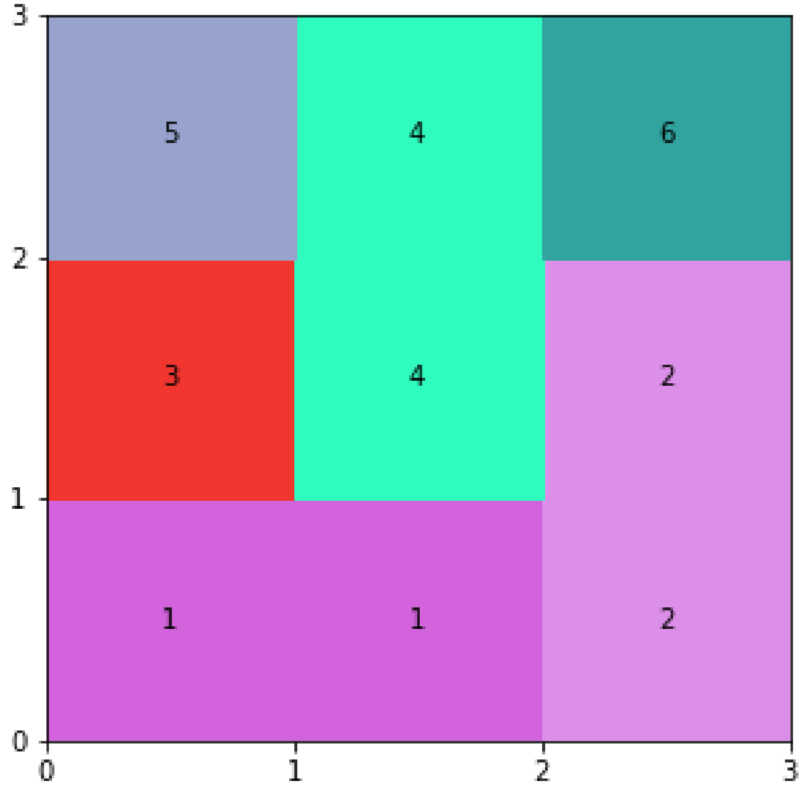
On average across 1000 experiments, it takes 1.0 communication towers before full coverage is ob



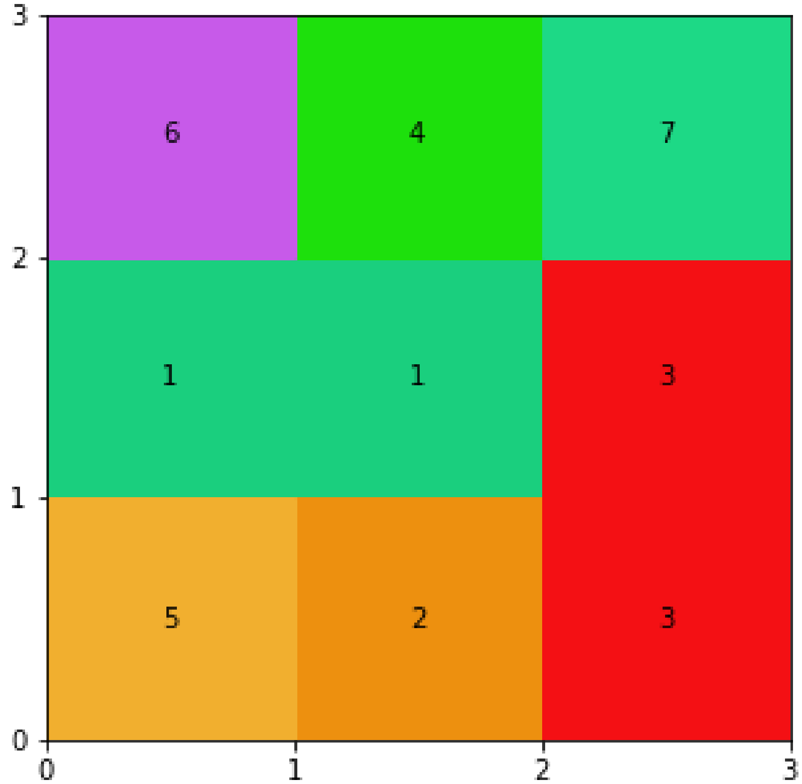

```
On average across 1050 experiments, it takes 6.0 communication towers before full coverage is ob
```



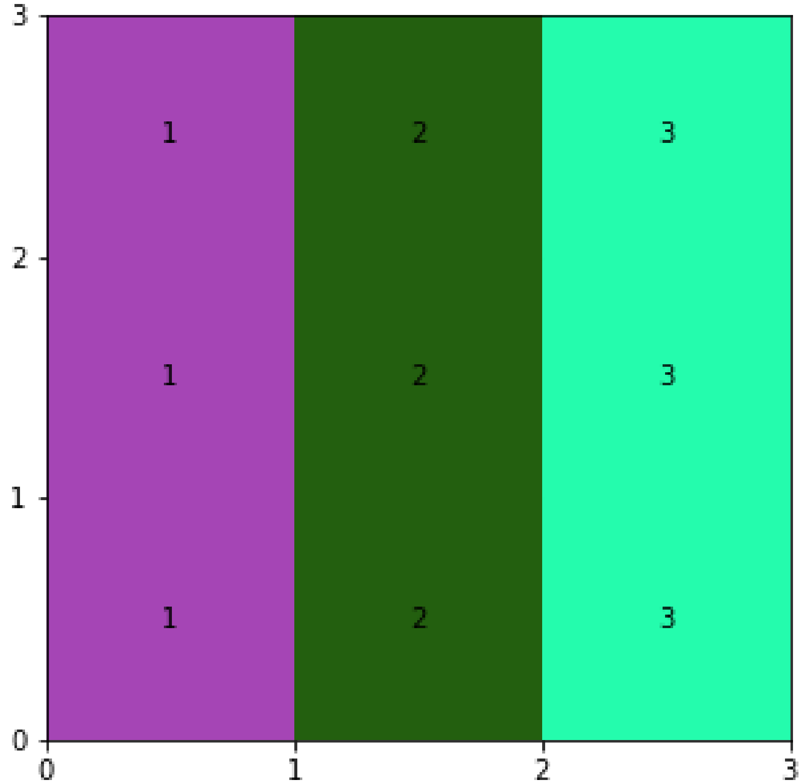

```
On average across 1100 experiments, it takes 6.0 communication towers before full coverage is ob
```



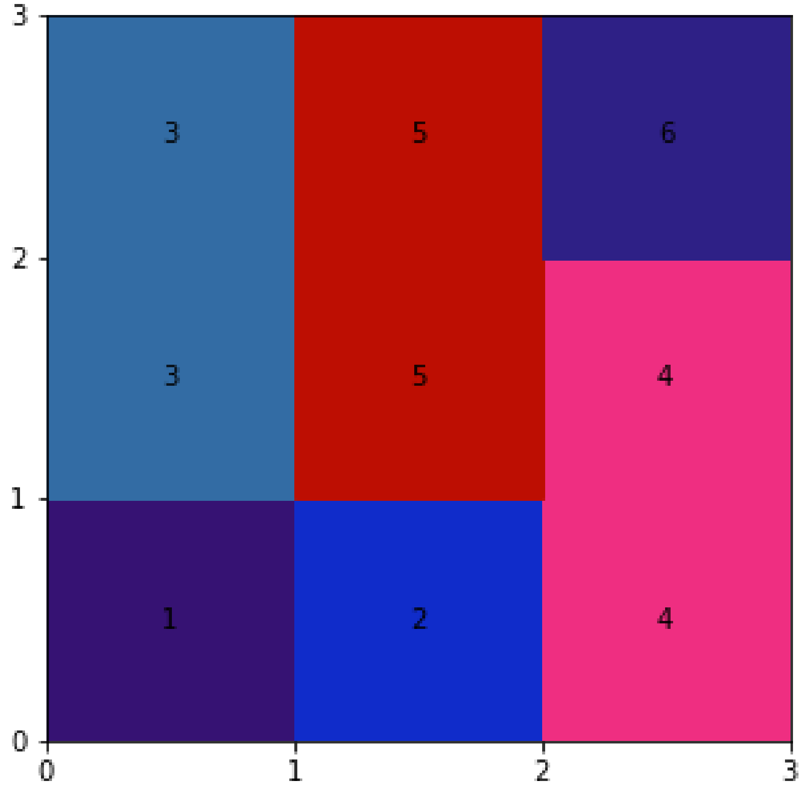
On average across 1150 experiments, it takes 7.0 communication towers before full coverage is ob



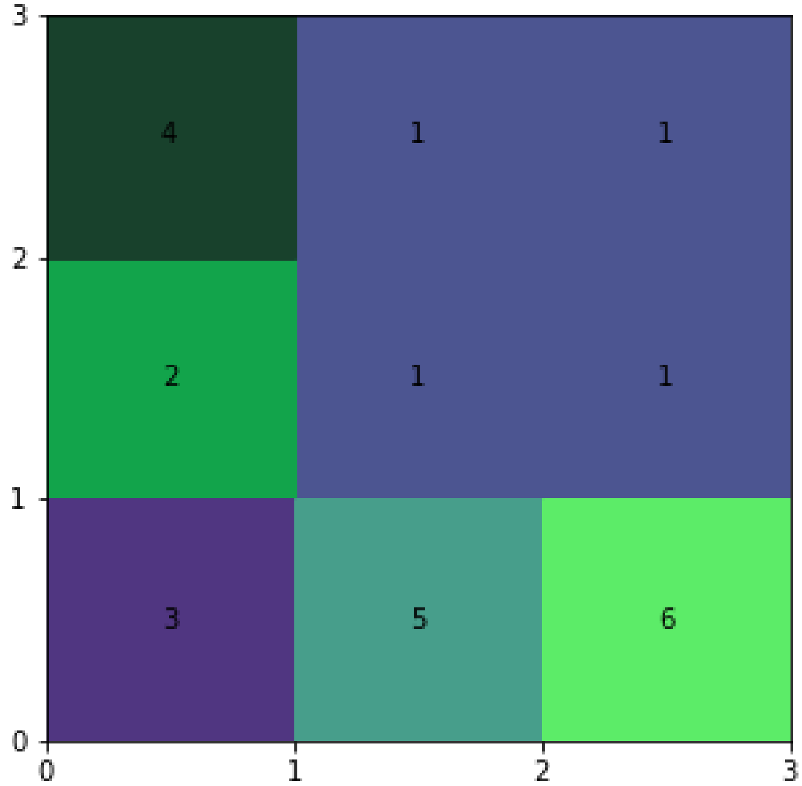
On average across 1200 experiments, it takes 3.0 communication towers before full coverage is ob



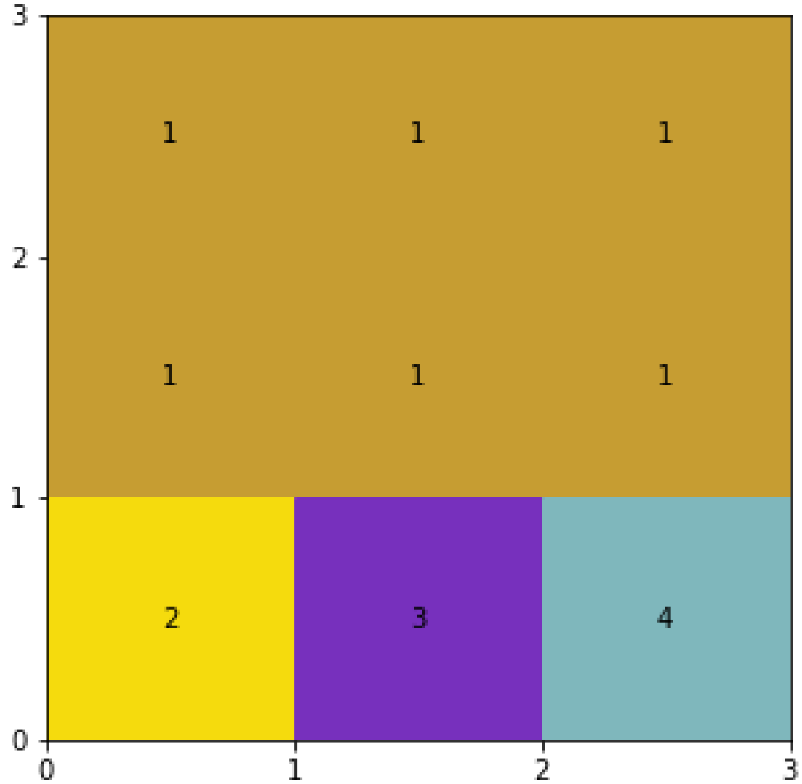

```
On average across 1250 experiments, it takes 6.0 communication towers before full coverage is ob
```



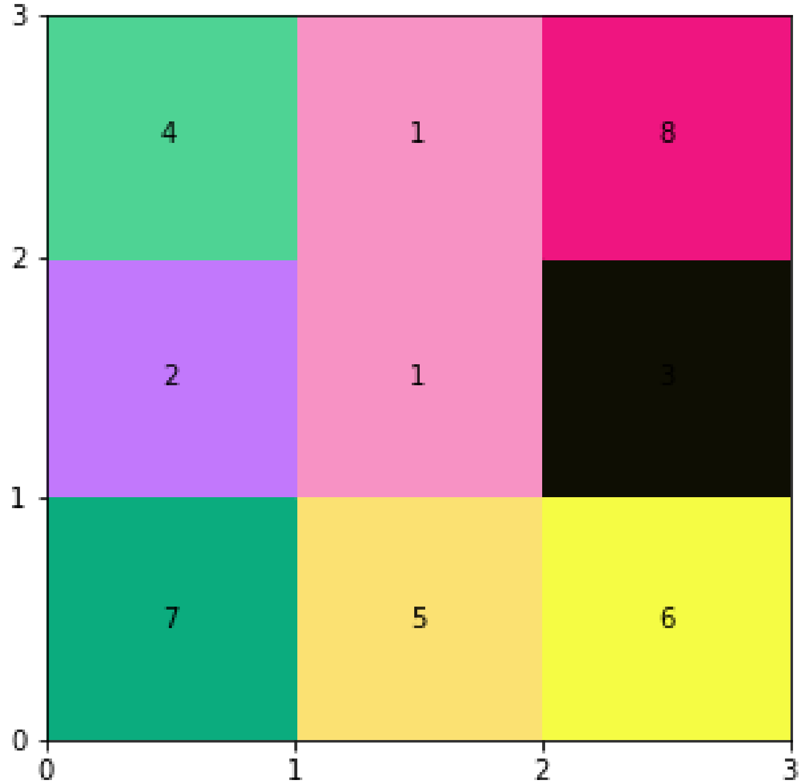

```
On average across 1300 experiments, it takes 6.0 communication towers before full coverage is ob
```



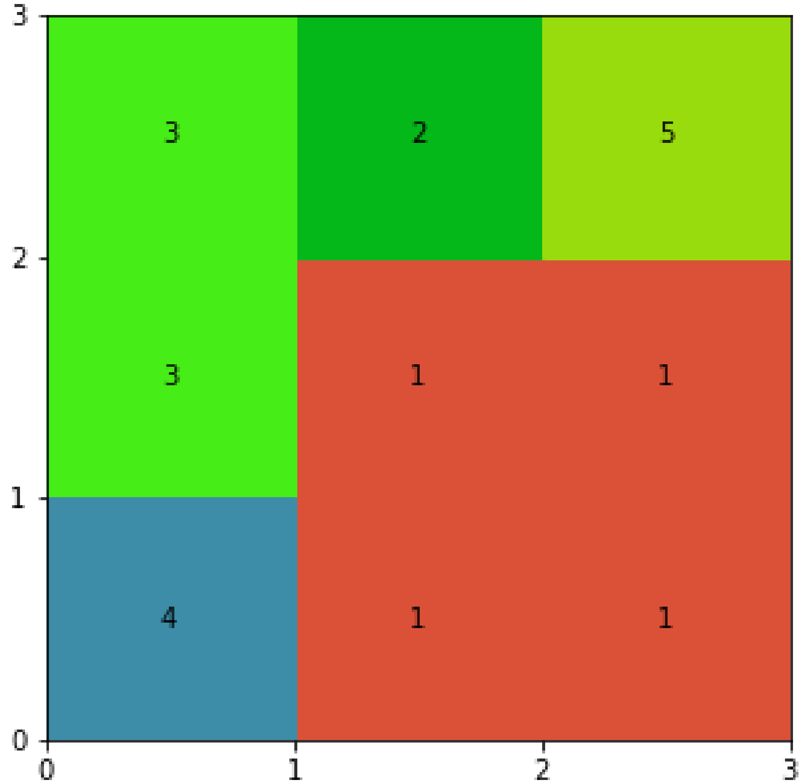
On average across 1350 experiments, it takes 4.0 communication towers before full coverage is ob



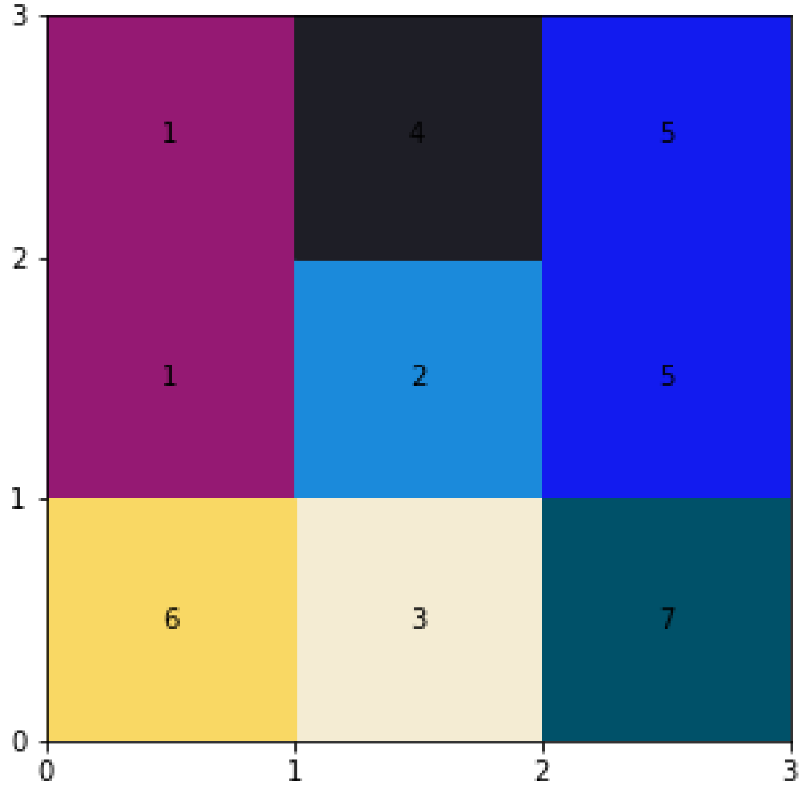

```
On average across 1400 experiments, it takes 8.0 communication towers before full coverage is ob
```



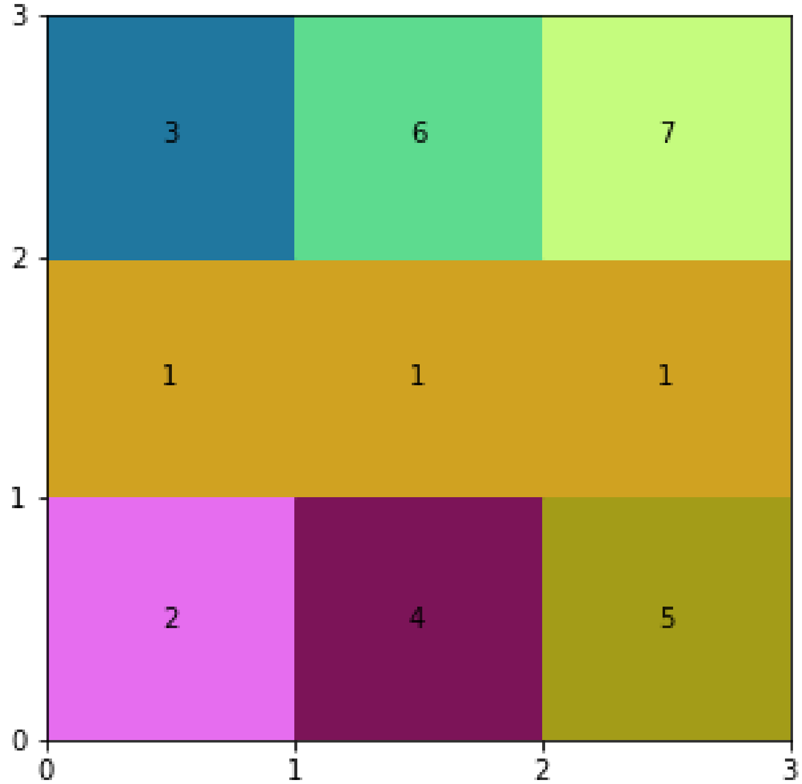
On average across 1450 experiments, it takes 5.0 communication towers before full coverage is ob



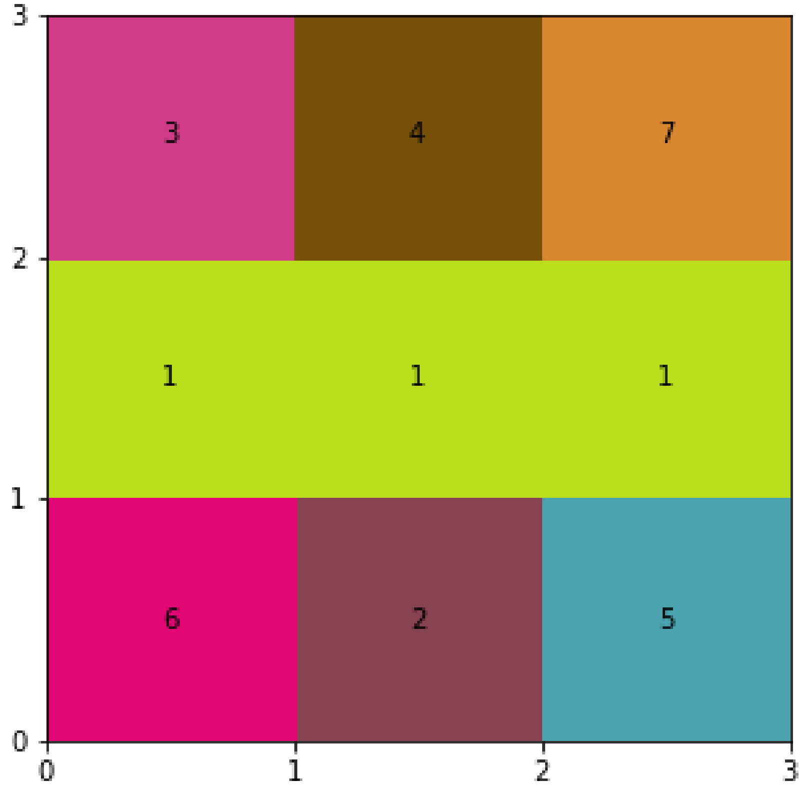
On average across 1500 experiments, it takes 7.0 communication towers before full coverage is ob



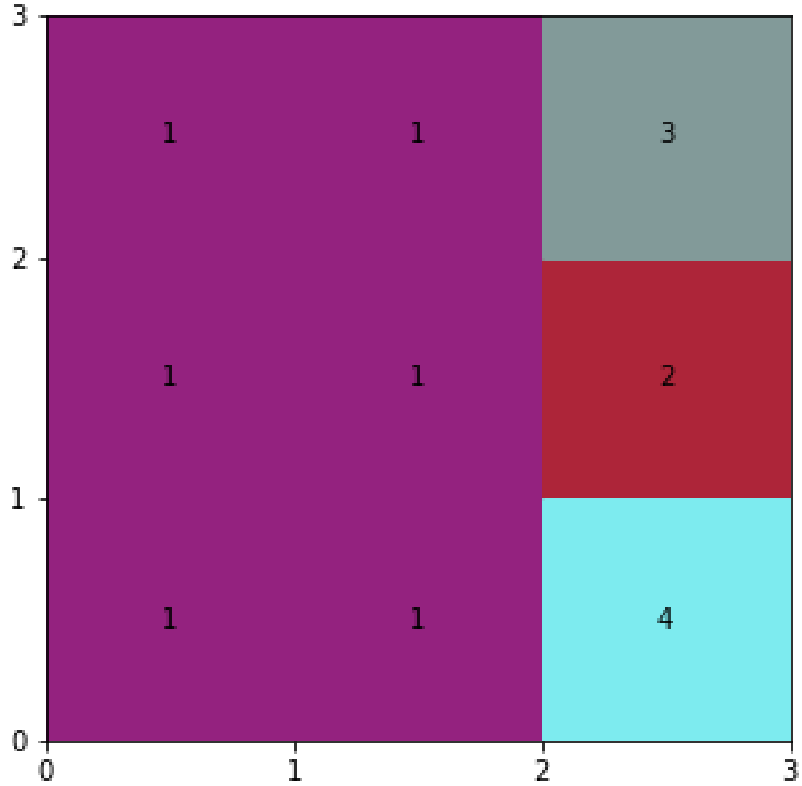
On average across 1550 experiments, it takes 7.0 communication towers before full coverage is ob



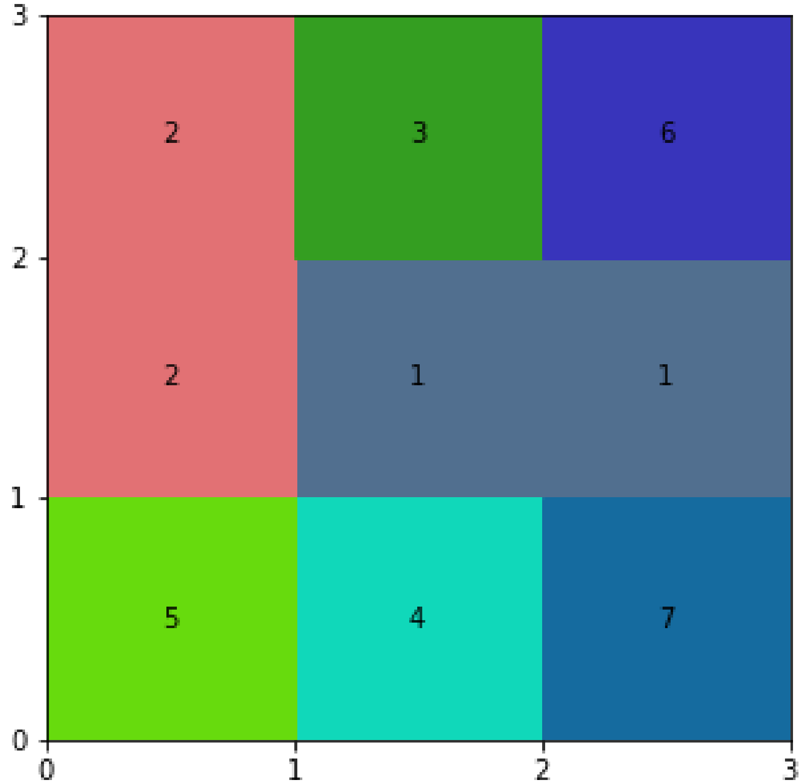

```
On average across 1600 experiments, it takes 7.0 communication towers before full coverage is ob
```



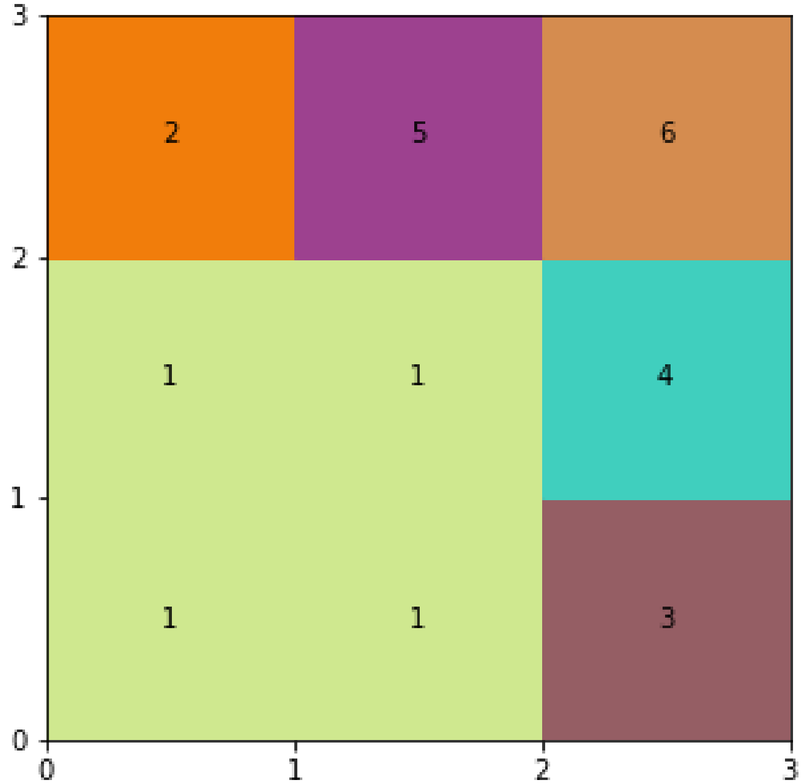
On average across 1650 experiments, it takes 4.0 communication towers before full coverage is ob



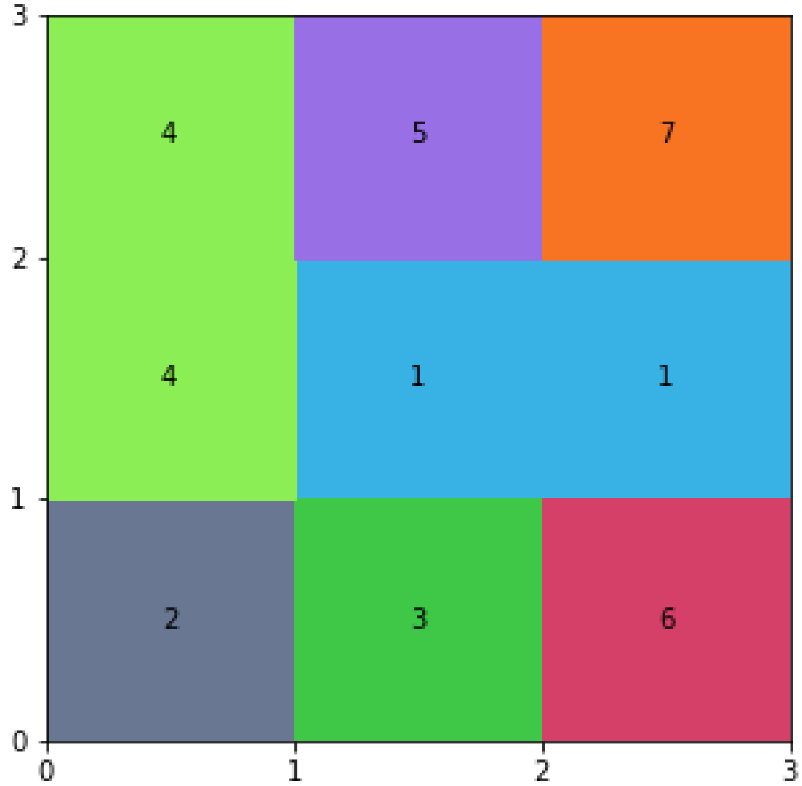
On average across 1700 experiments, it takes 7.0 communication towers before full coverage is ob



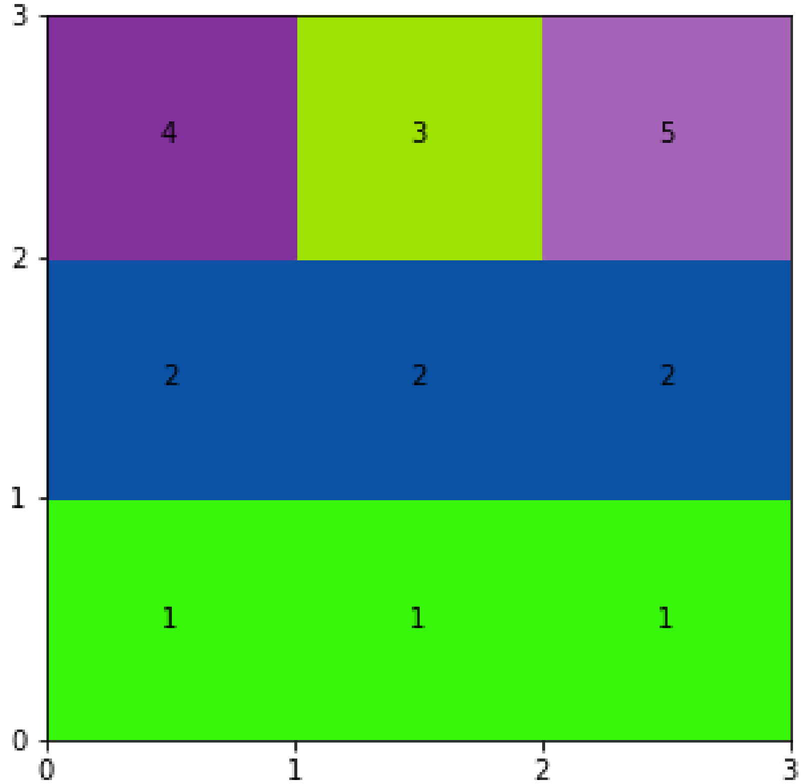
On average across 1750 experiments, it takes 6.0 communication towers before full coverage is ob



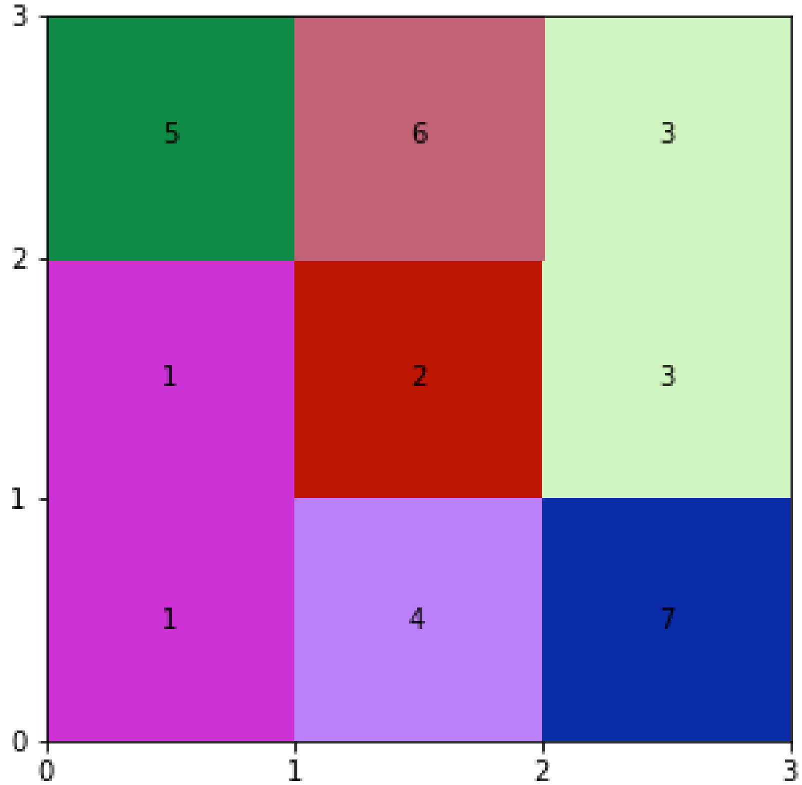

```
On average across 1800 experiments, it takes 7.0 communication towers before full coverage is ob
```



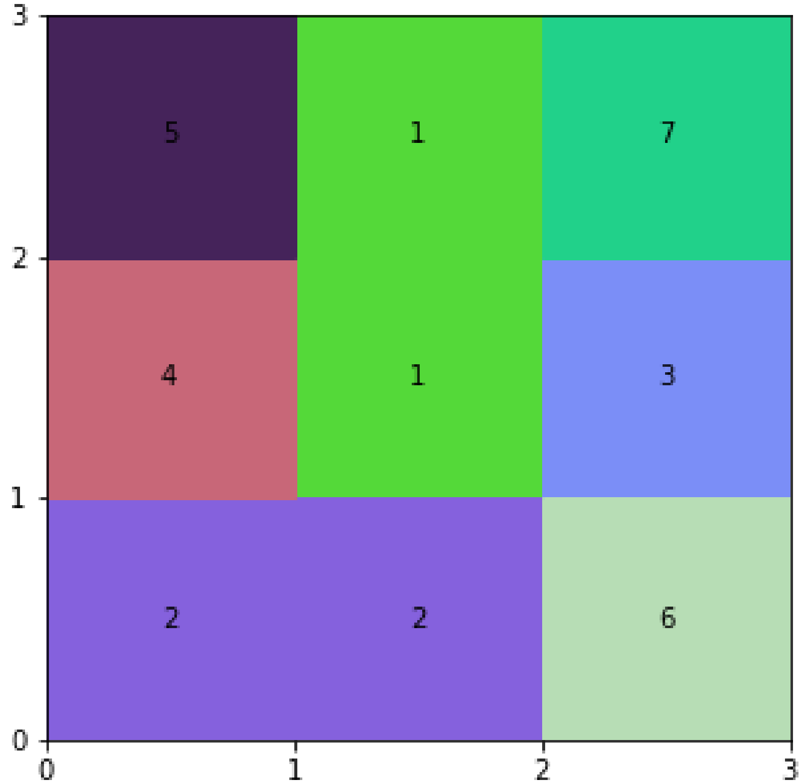
On average across 1850 experiments, it takes 5.0 communication towers before full coverage is ob



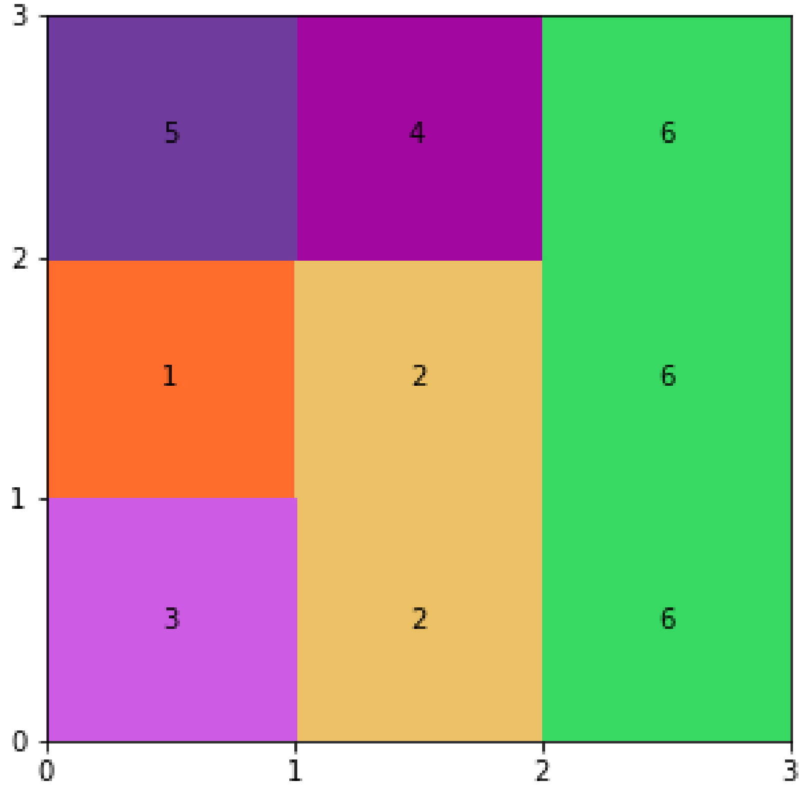
On average across 1900 experiments, it takes 7.0 communication towers before full coverage is ob



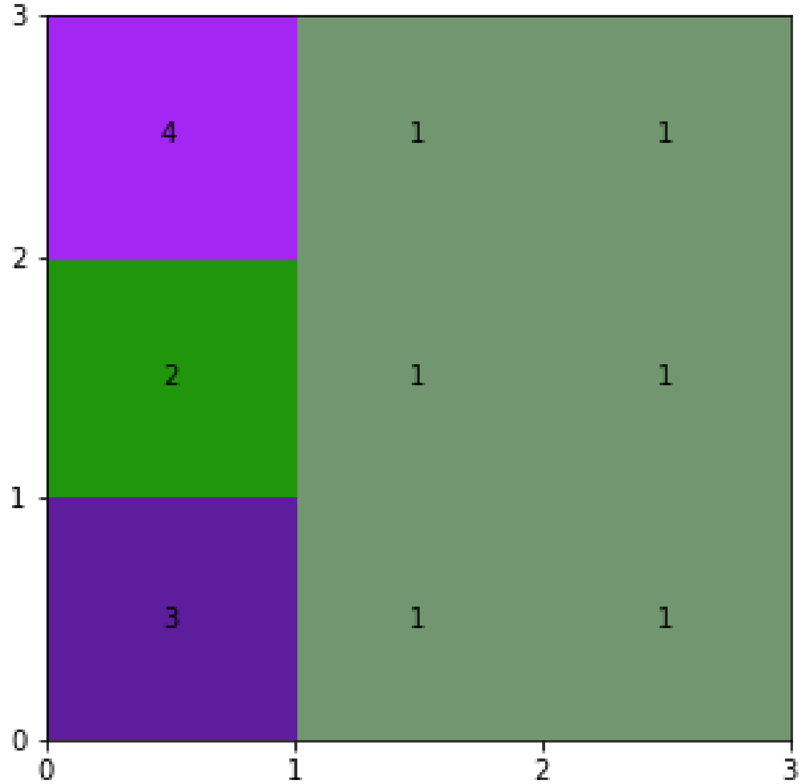
On average across 1950 experiments, it takes 7.0 communication towers before full coverage is ob



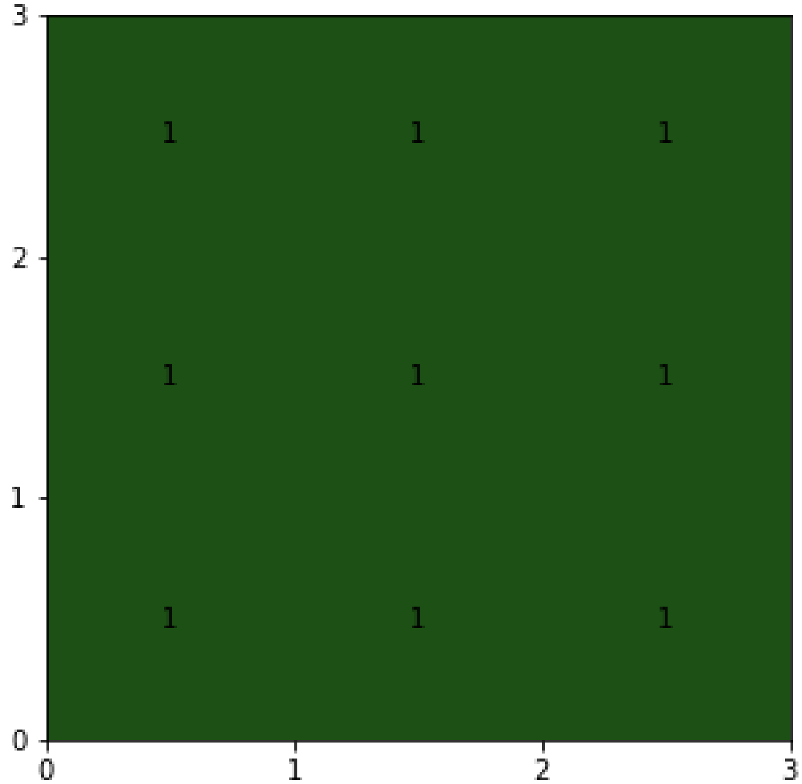
`On average across 2000 experiments, it takes 6.0 communication towers before full coverage is ob`



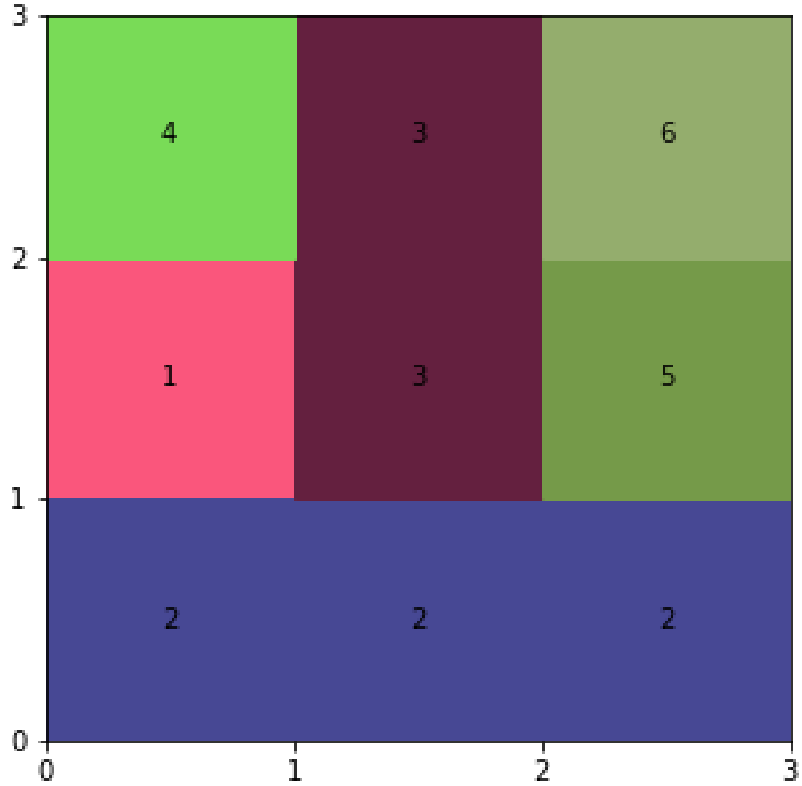

```
On average across 2050 experiments, it takes 4.0 communication towers before full coverage is ob
```



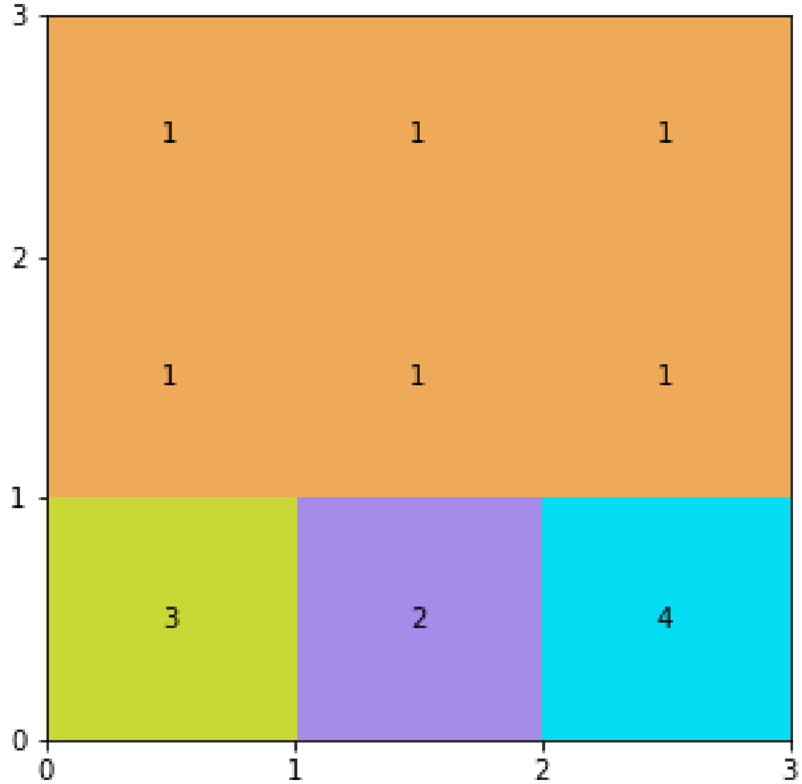
On average across 2100 experiments, it takes 1.0 communication towers before full coverage is ob



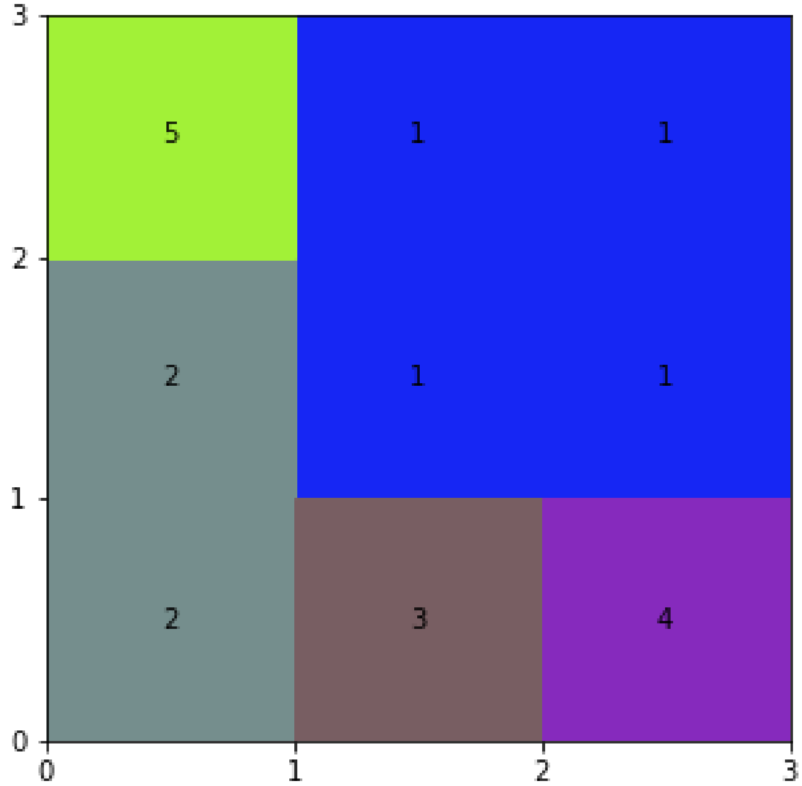

```
On average across 2150 experiments, it takes 6.0 communication towers before full coverage is ob
```



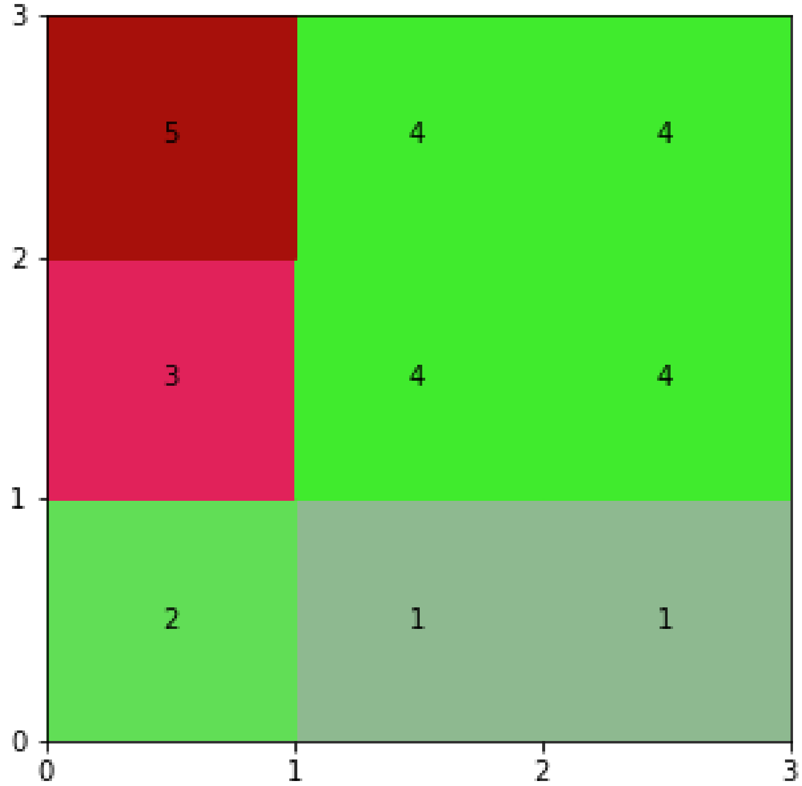

```
On average across 2200 experiments, it takes 4.0 communication towers before full coverage is ob
```



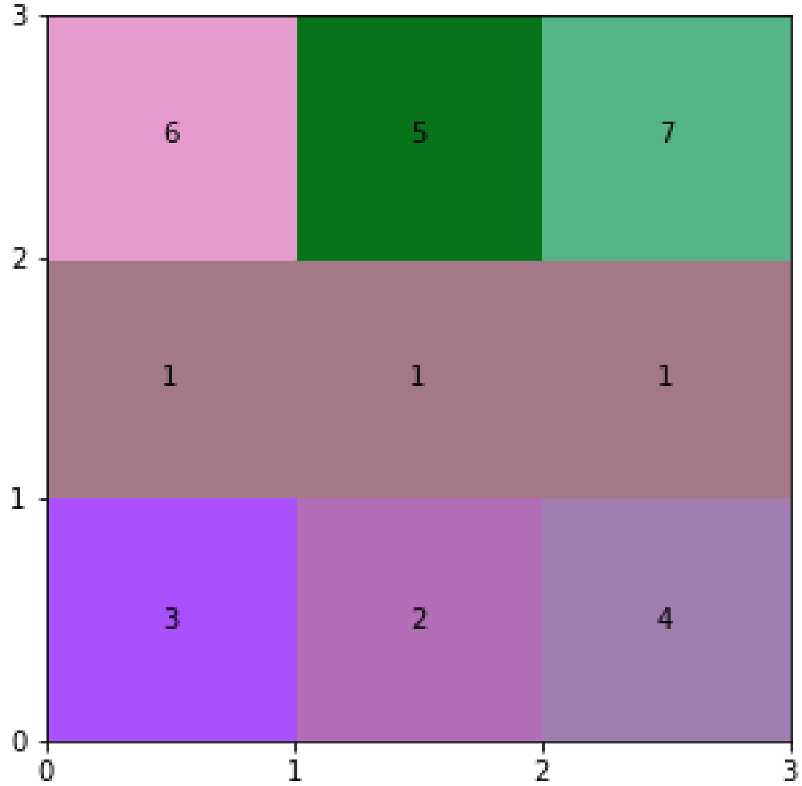
On average across 2250 experiments, it takes 5.0 communication towers before full coverage is ob



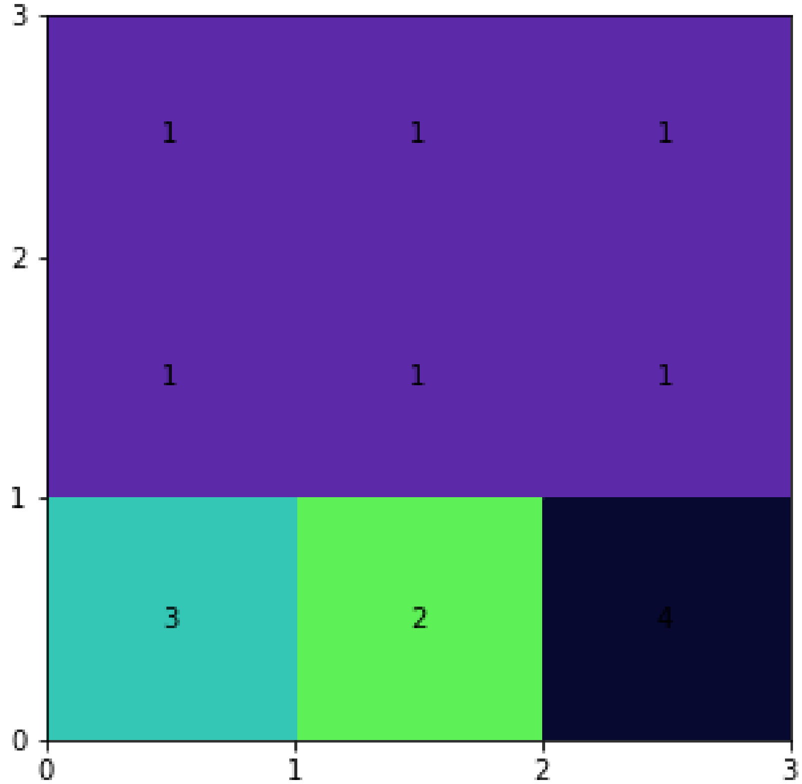

```
On average across 2300 experiments, it takes 5.0 communication towers before full coverage is ob
```



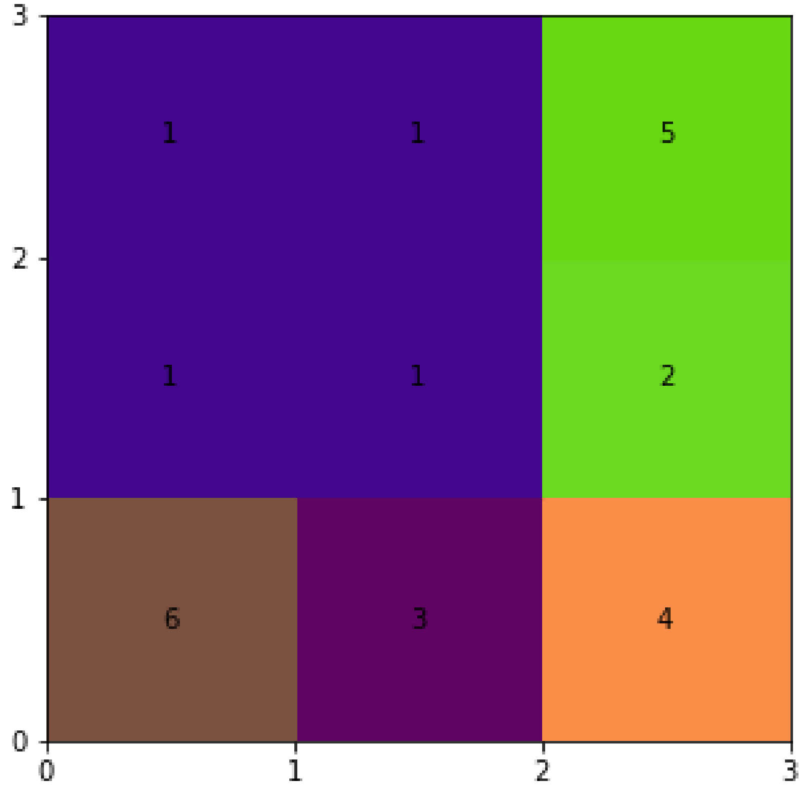

```
On average across 2350 experiments, it takes 7.0 communication towers before full coverage is ob
```



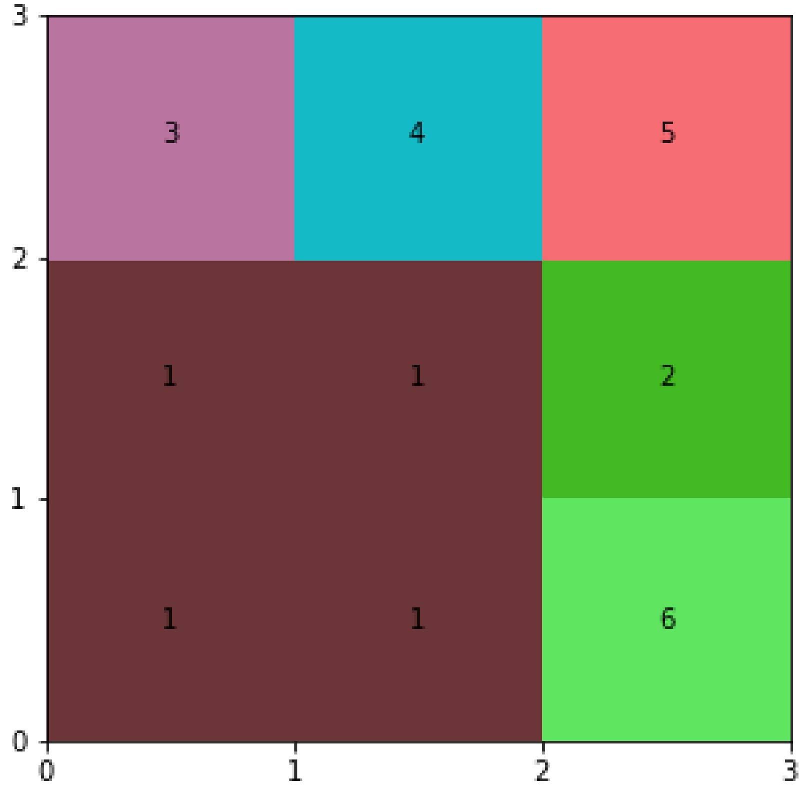

```
On average across 2400 experiments, it takes 4.0 communication towers before full coverage is ob
```



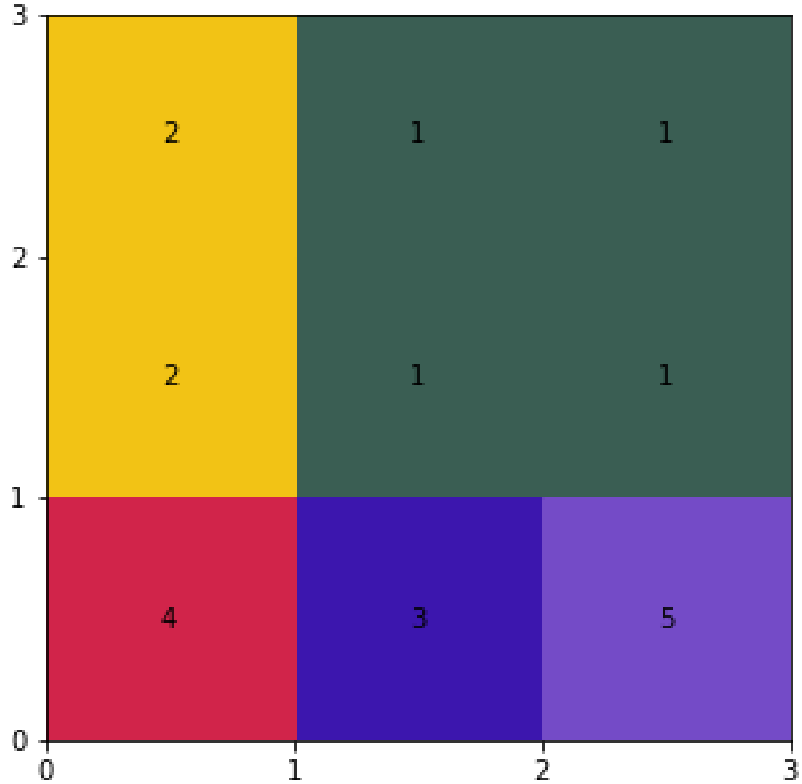
On average across 2450 experiments, it takes 6.0 communication towers before full coverage is ob



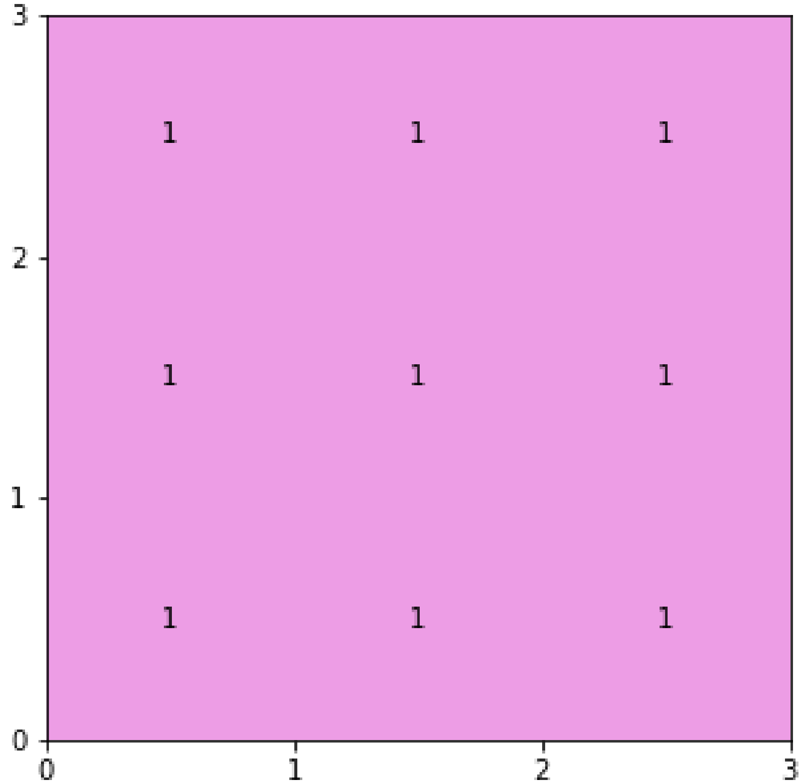
On average across 2500 experiments, it takes 6.0 communication towers before full coverage is ob



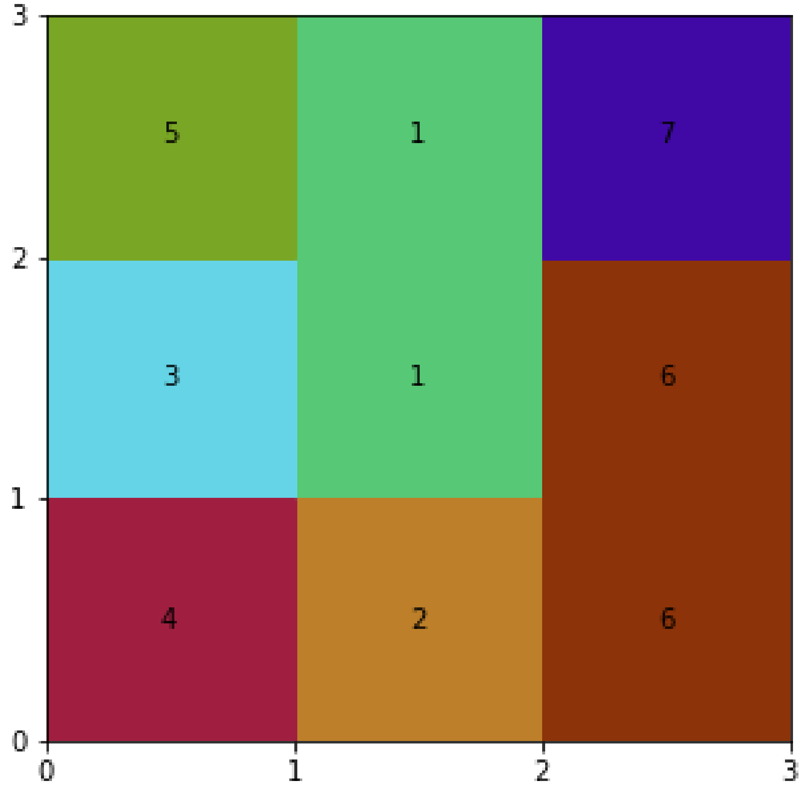
On average across 2550 experiments, it takes 5.0 communication towers before full coverage is ob



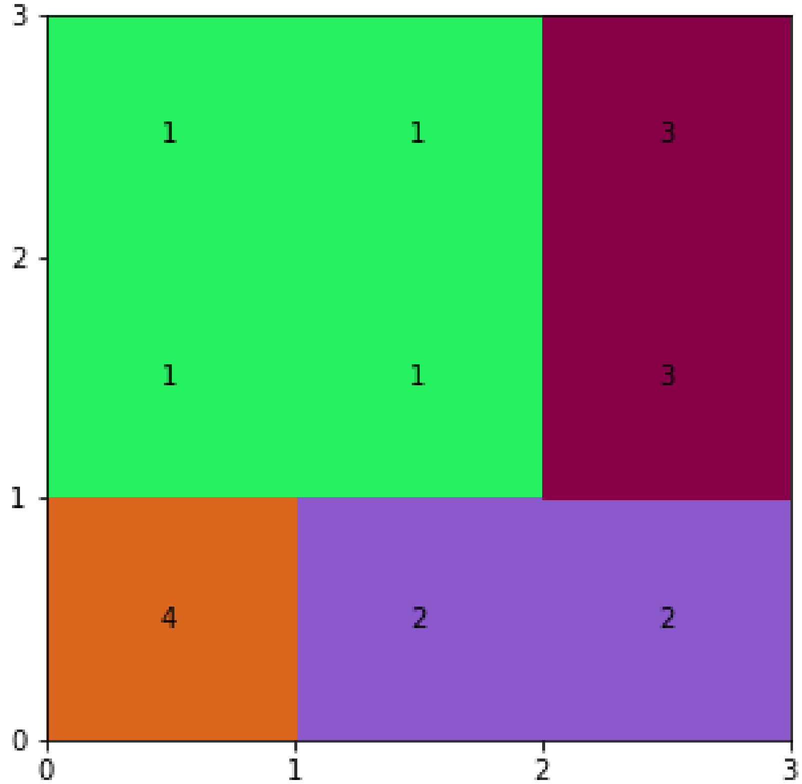

```
On average across 2600 experiments, it takes 1.0 communication towers before full coverage is ob
```



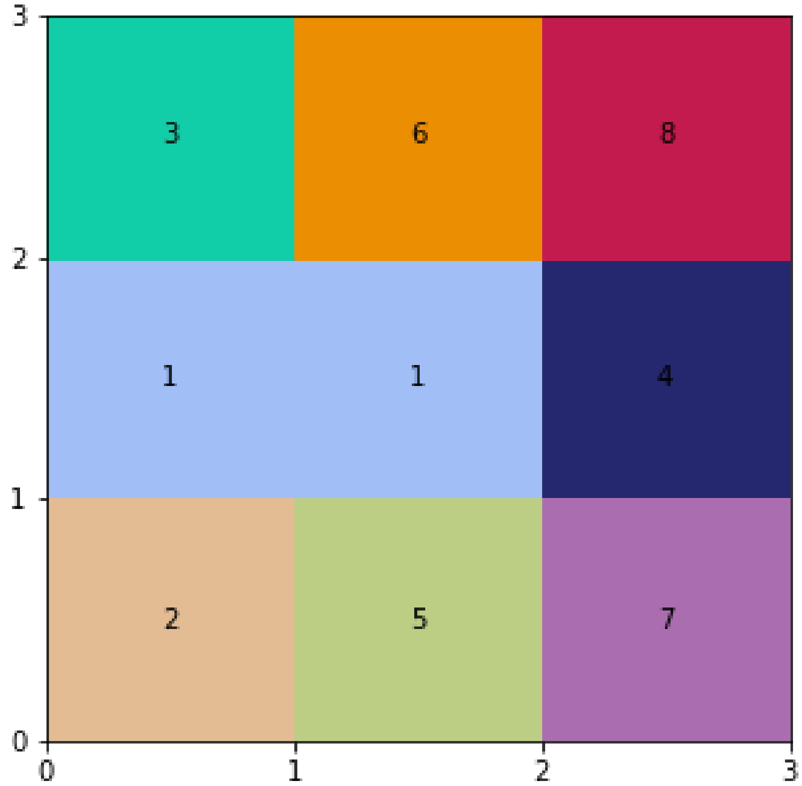

```
On average across 2650 experiments, it takes 7.0 communication towers before full coverage is ob
```



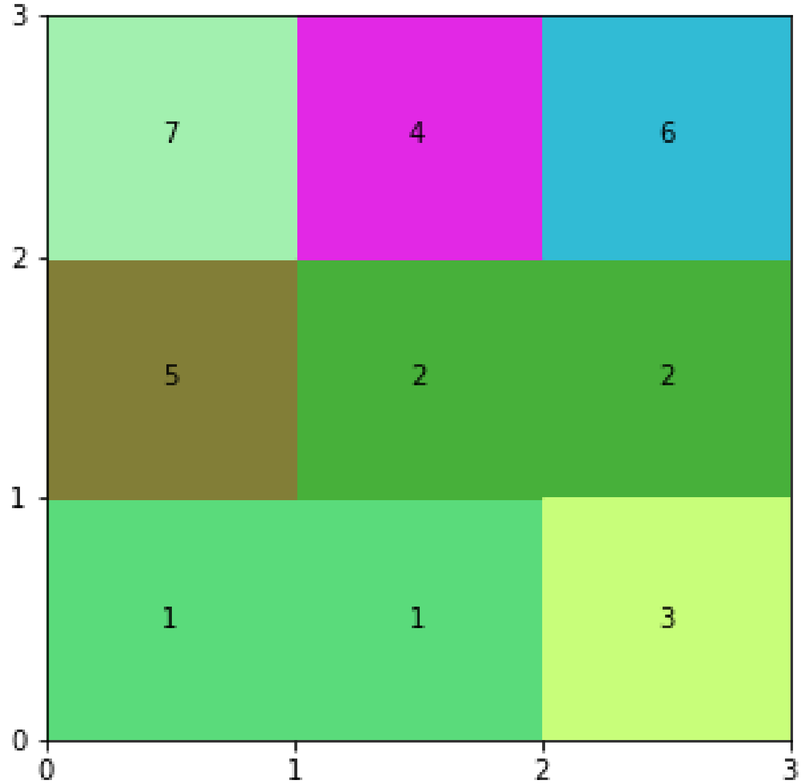
`On average across 2700 experiments, it takes 4.0 communication towers before full coverage is ob`



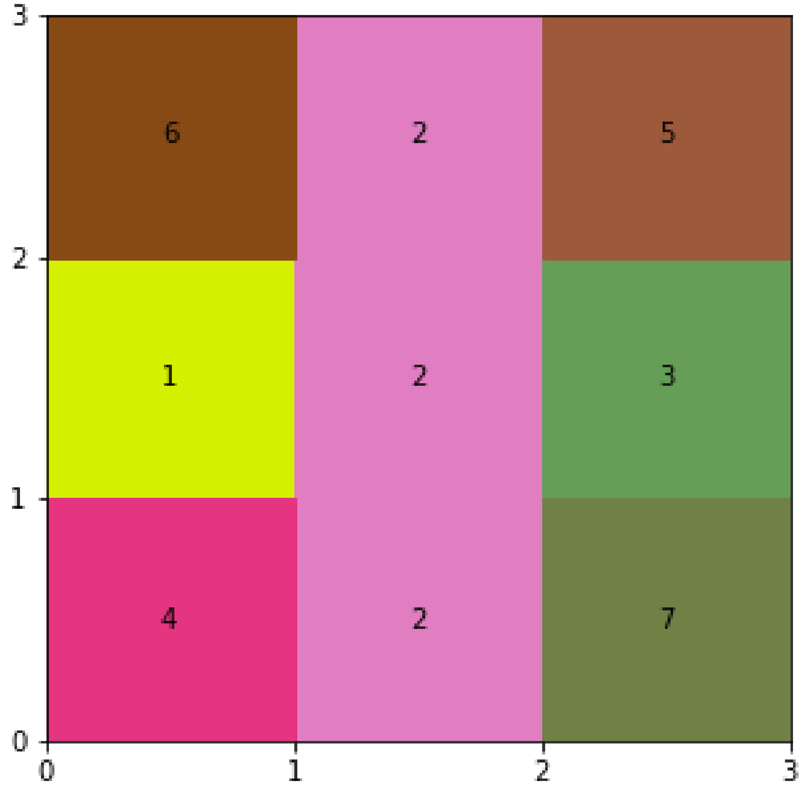

```
On average across 2750 experiments, it takes 8.0 communication towers before full coverage is ob
```



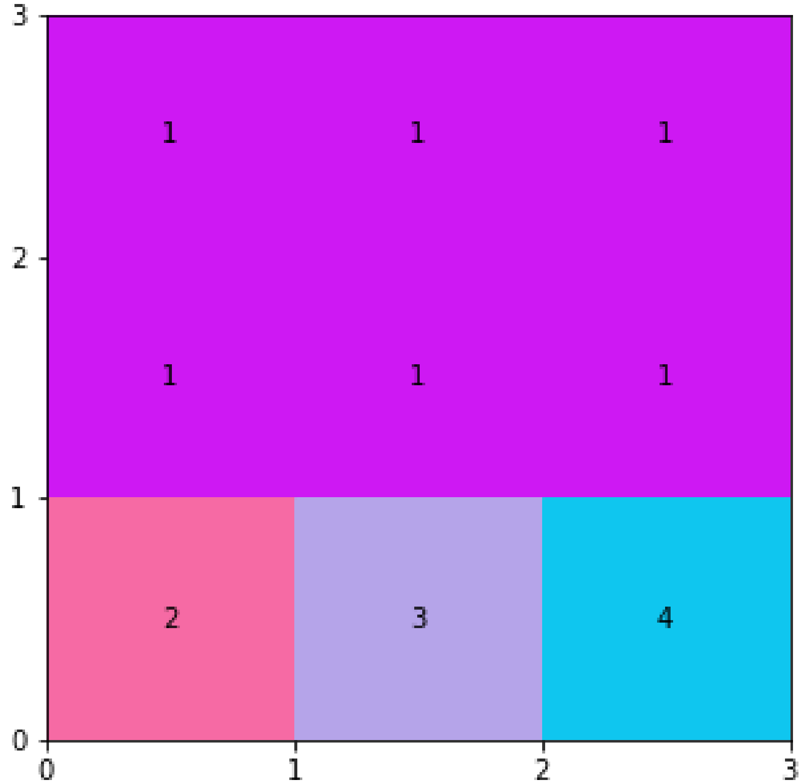
On average across 2800 experiments, it takes 7.0 communication towers before full coverage is ob



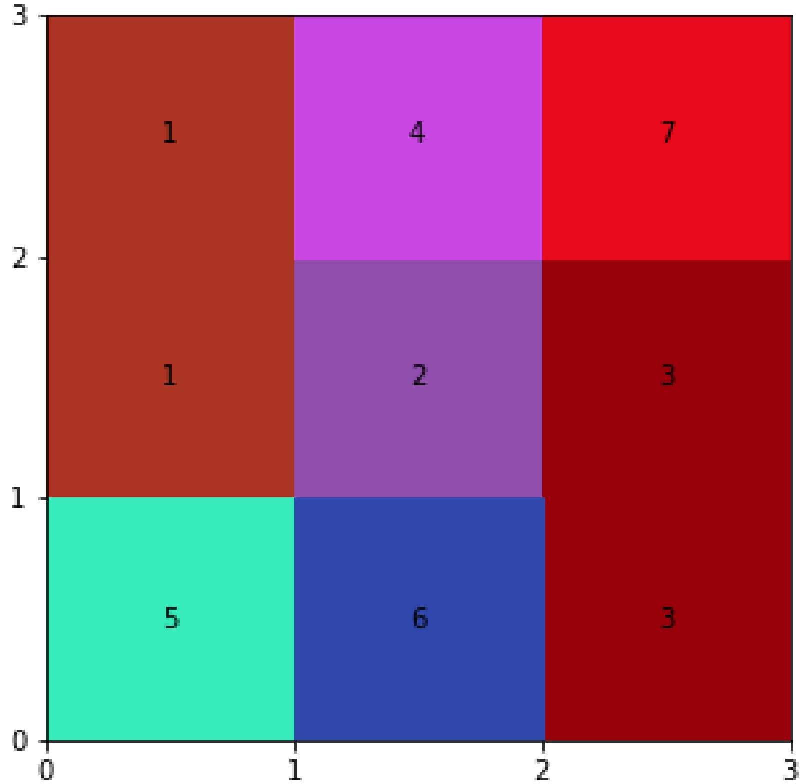

```
On average across 2850 experiments, it takes 7.0 communication towers before full coverage is ob
```



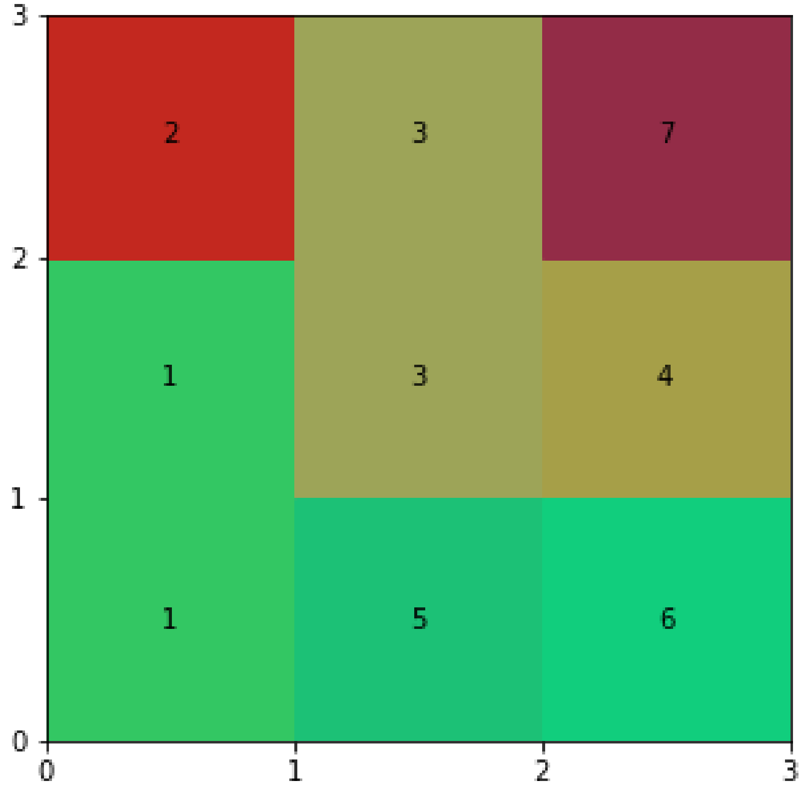
On average across 2900 experiments, it takes 4.0 communication towers before full coverage is ob



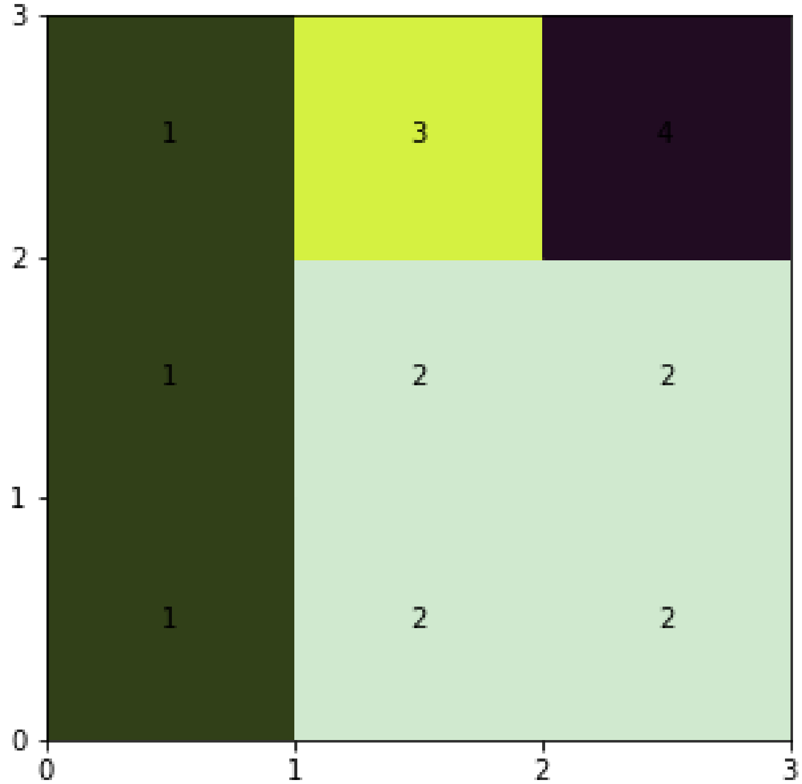

```
On average across 2950 experiments, it takes 7.0 communication towers before full coverage is ob
```



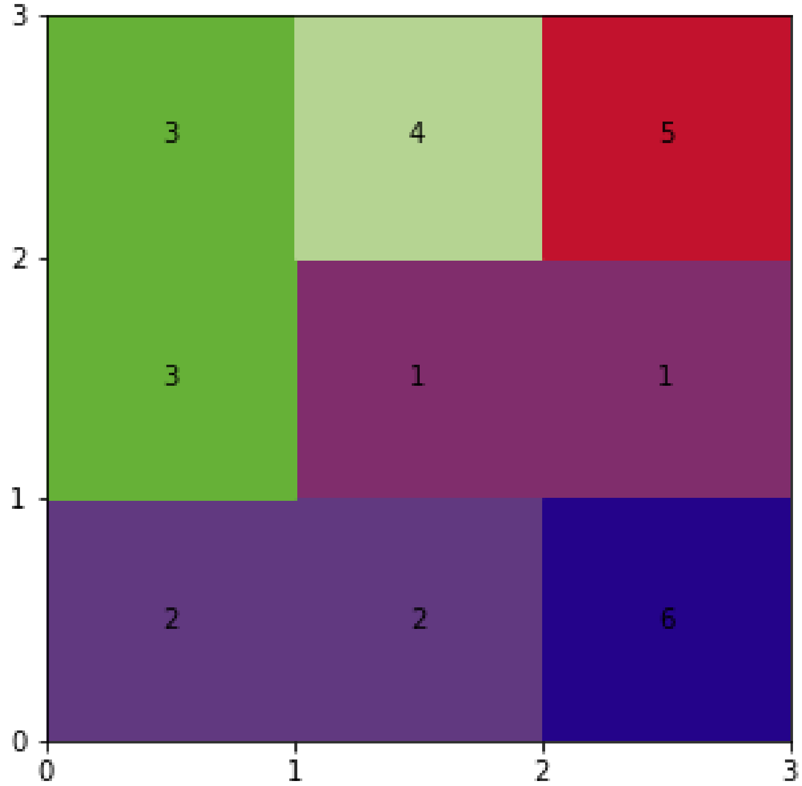
On average across 3000 experiments, it takes 7.0 communication towers before full coverage is ob



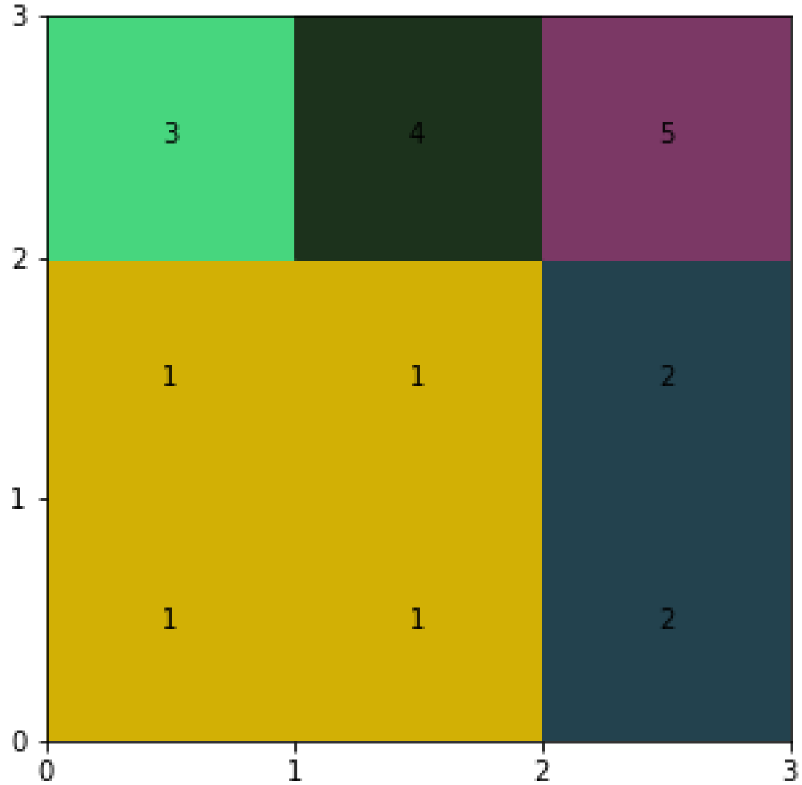
On average across 3050 experiments, it takes 4.0 communication towers before full coverage is ob



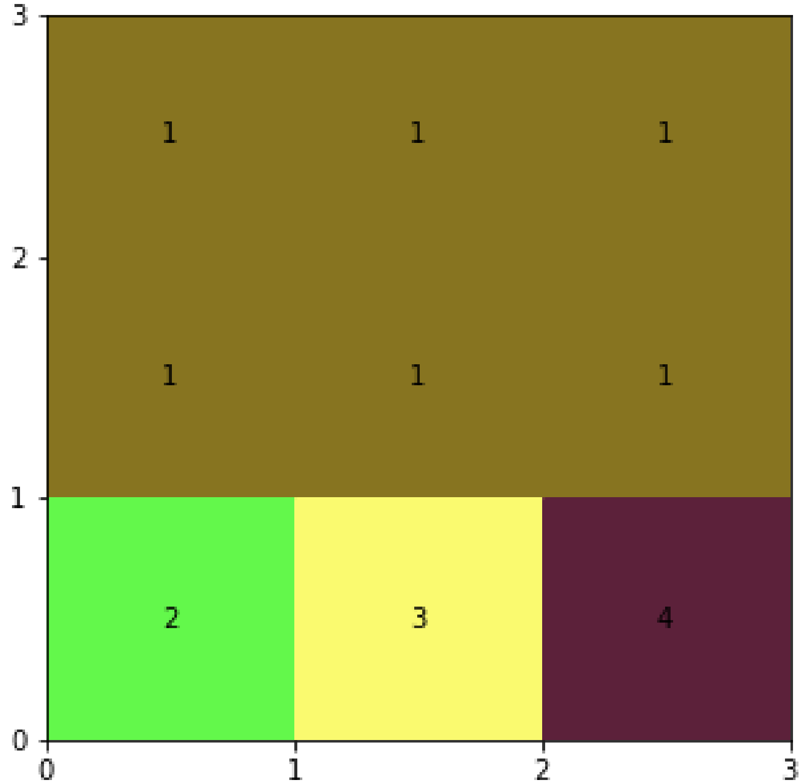

```
On average across 3100 experiments, it takes 6.0 communication towers before full coverage is ob
```



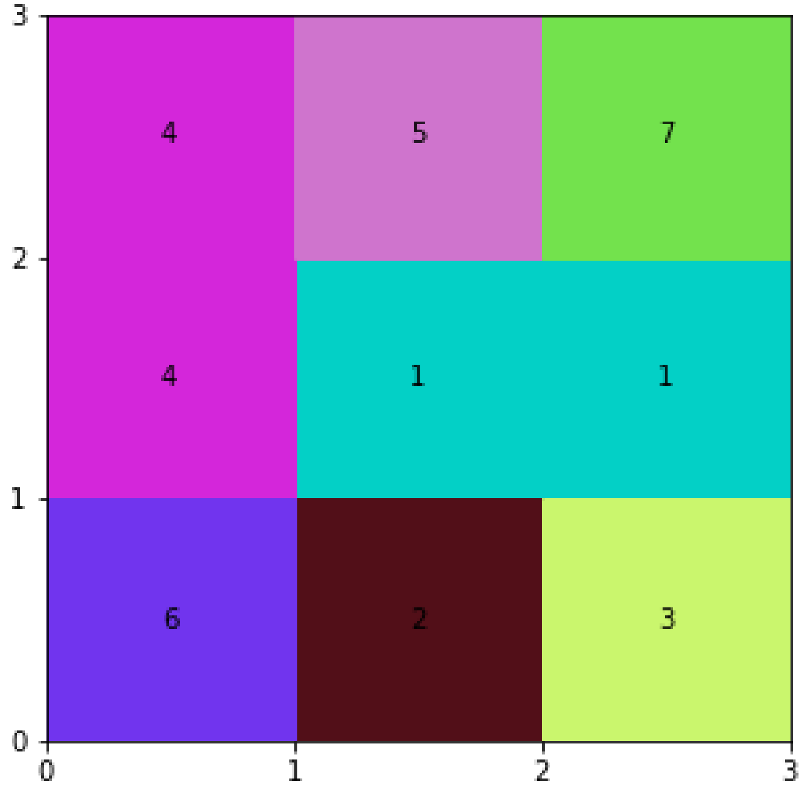
On average across 3150 experiments, it takes 5.0 communication towers before full coverage is ob



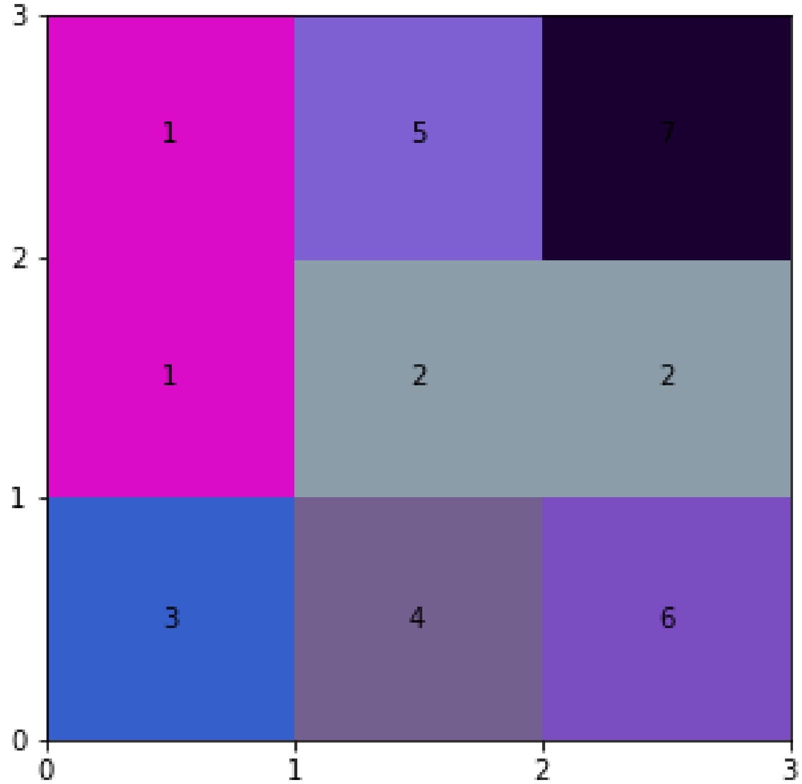

```
On average across 3200 experiments, it takes 4.0 communication towers before full coverage is ob
```



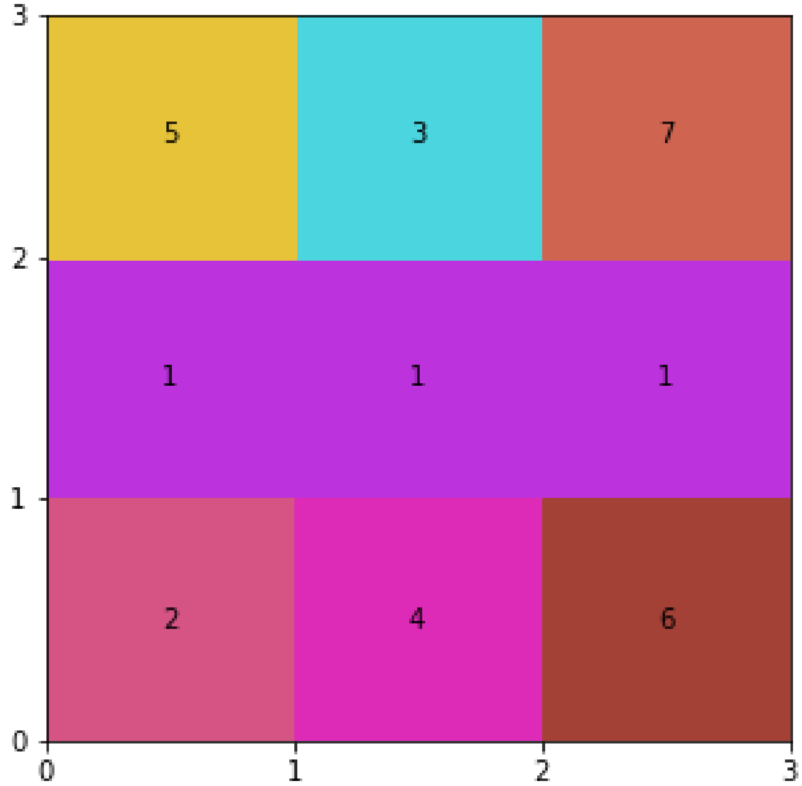
On average across 3250 experiments, it takes 7.0 communication towers before full coverage is ob



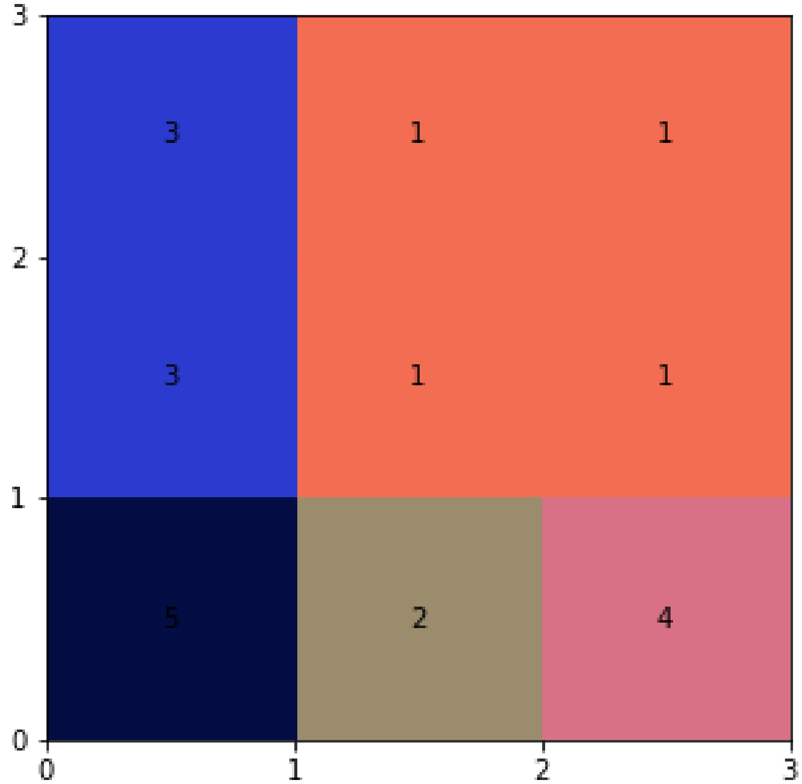
`On average across 3300 experiments, it takes 7.0 communication towers before full coverage is ob`



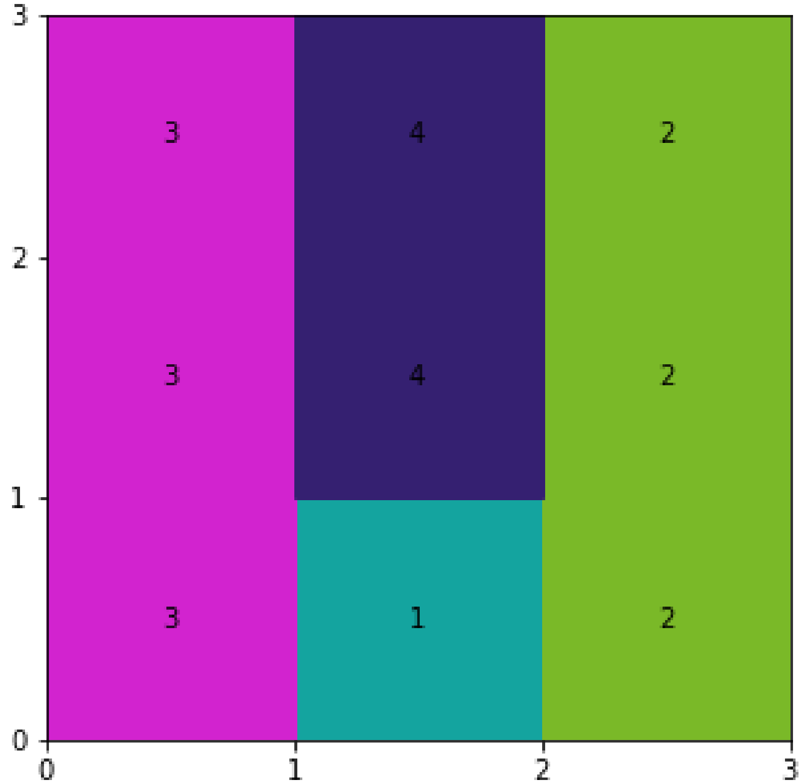
On average across 3350 experiments, it takes 7.0 communication towers before full coverage is ob



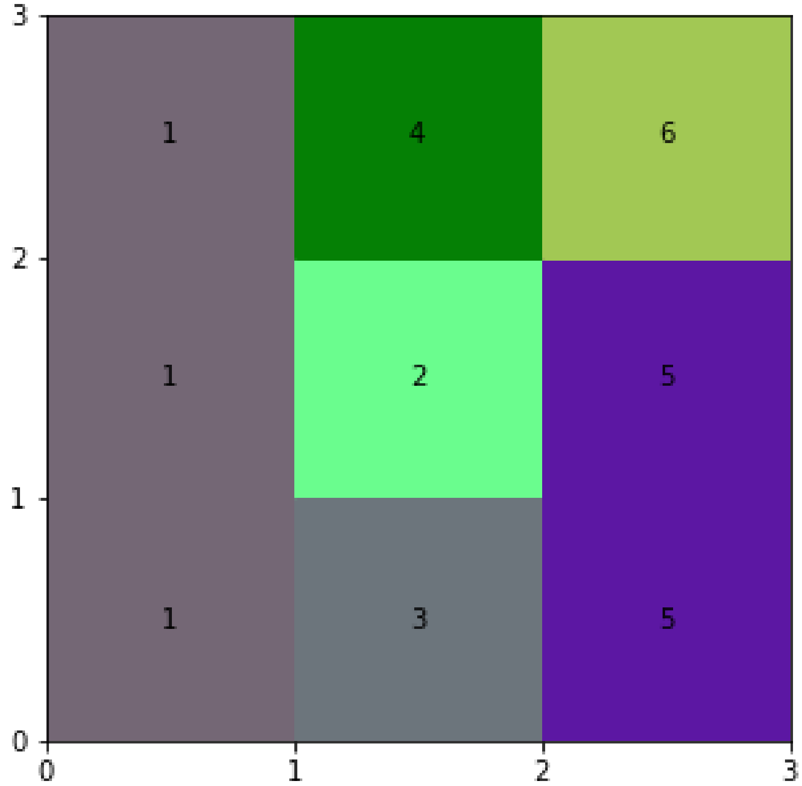
On average across 3400 experiments, it takes 5.0 communication towers before full coverage is ob



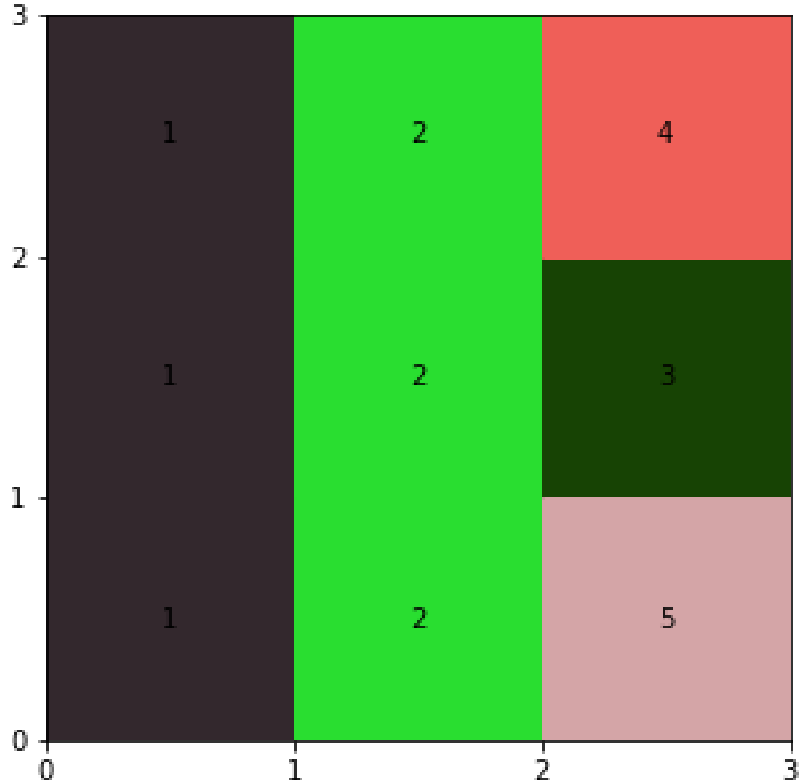

```
On average across 3450 experiments, it takes 4.0 communication towers before full coverage is ob
```



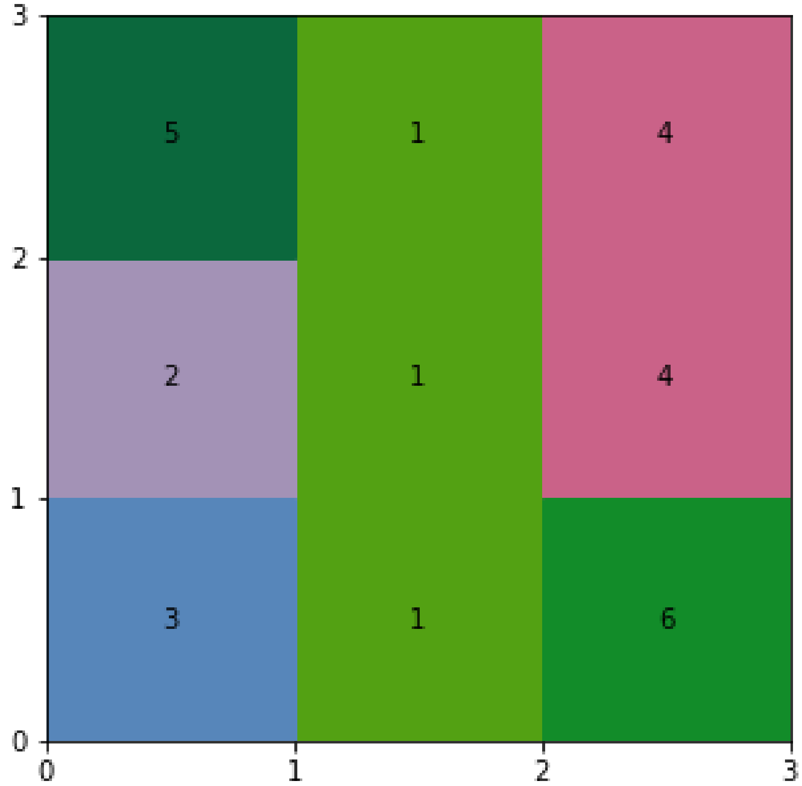
On average across 3500 experiments, it takes 6.0 communication towers before full coverage is ob



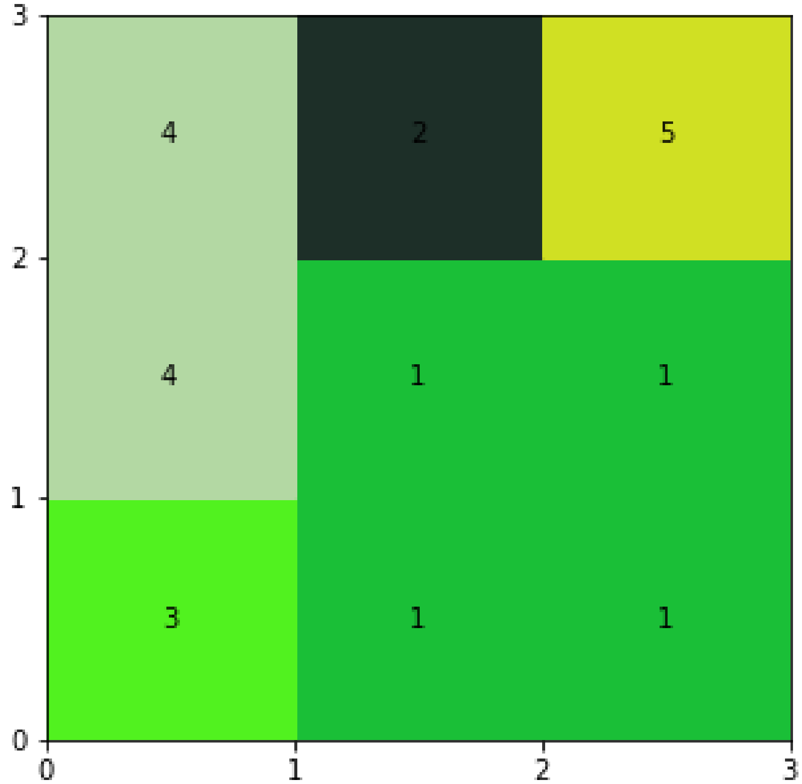
On average across 3550 experiments, it takes 5.0 communication towers before full coverage is ob



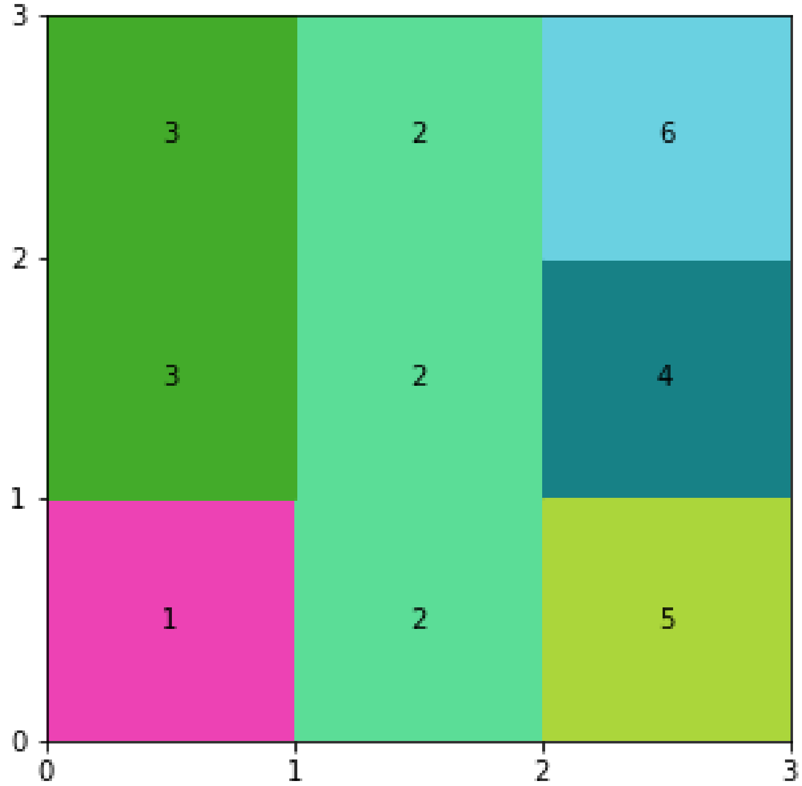

```
On average across 3600 experiments, it takes 6.0 communication towers before full coverage is ob
```



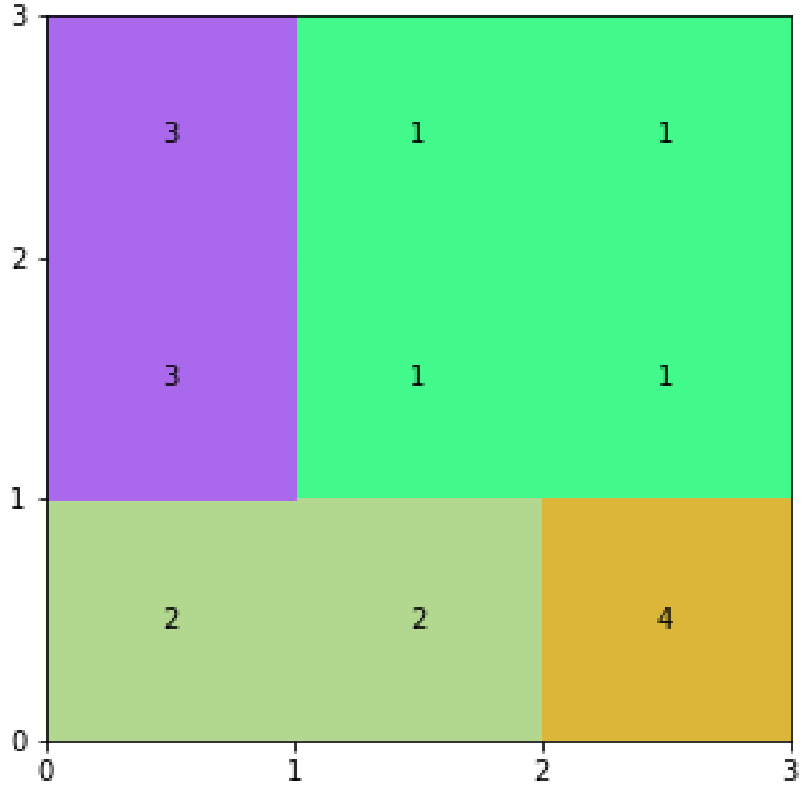
On average across 3650 experiments, it takes 5.0 communication towers before full coverage is ob



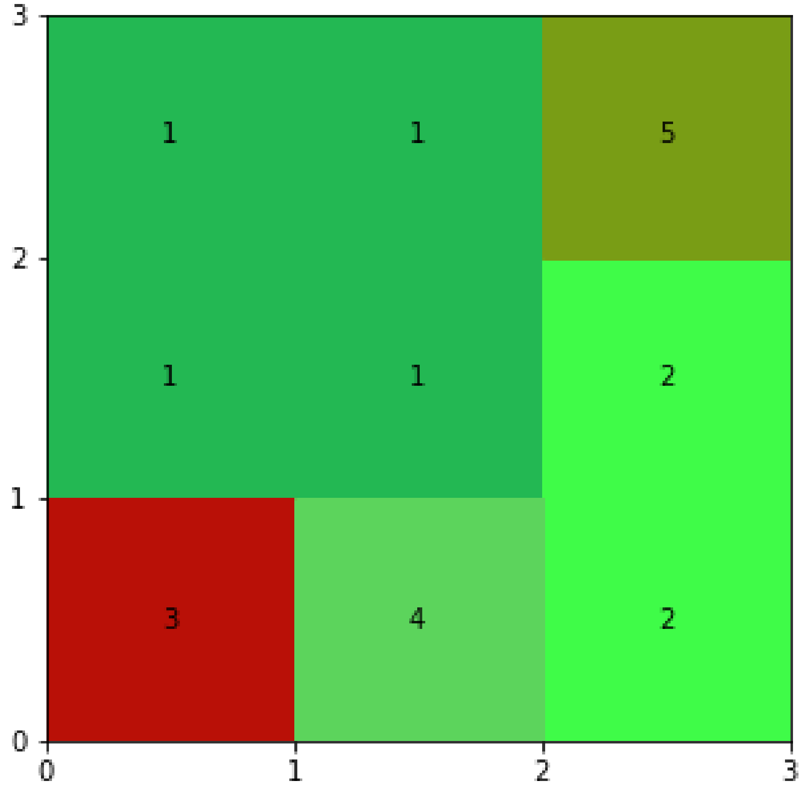
On average across 3700 experiments, it takes 6.0 communication towers before full coverage is ob



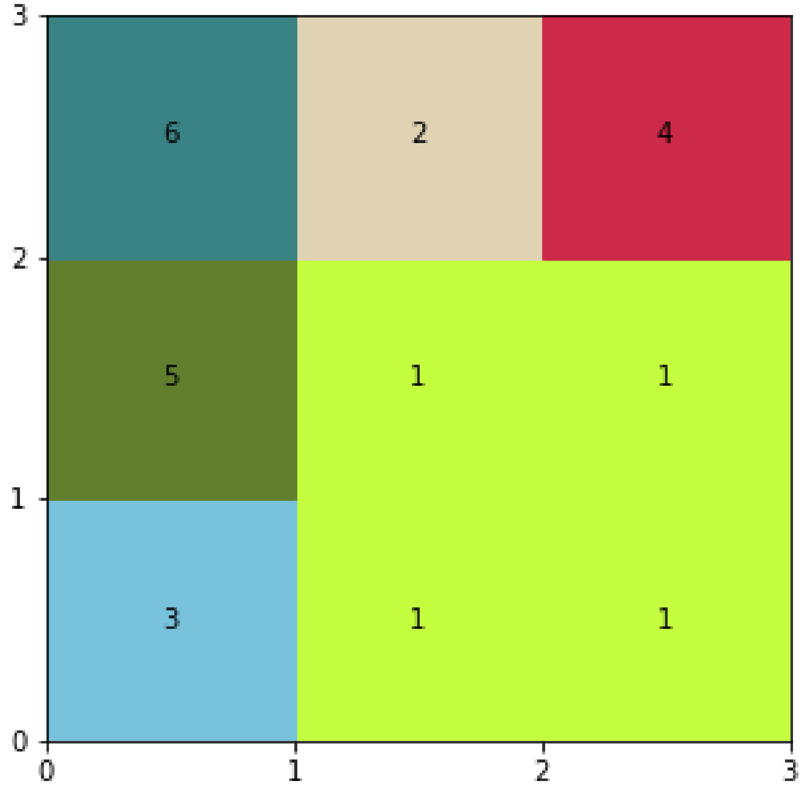

```
On average across 3750 experiments, it takes 4.0 communication towers before full coverage is ob
```



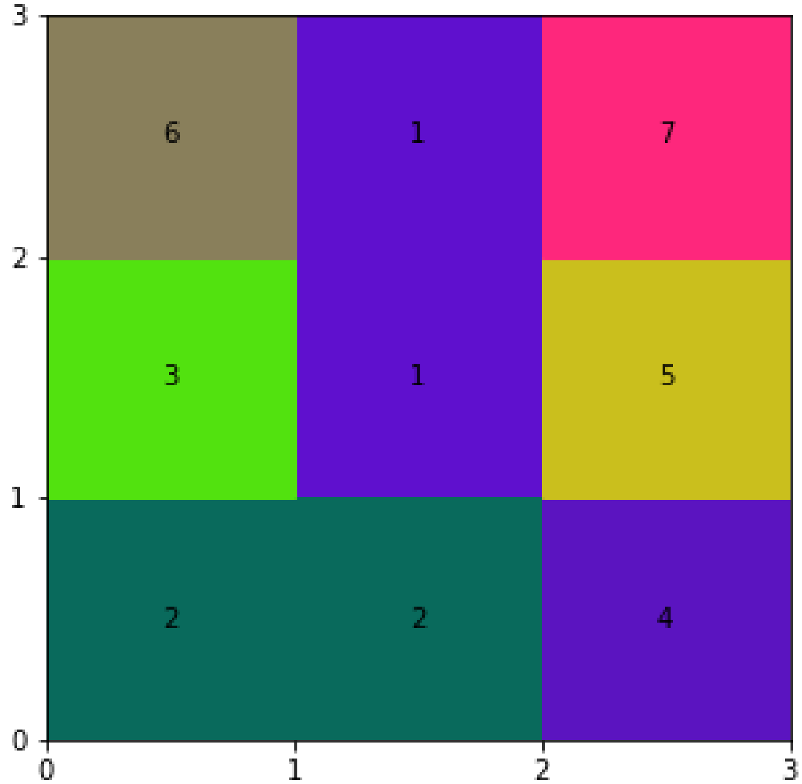

```
On average across 3800 experiments, it takes 5.0 communication towers before full coverage is ob
```



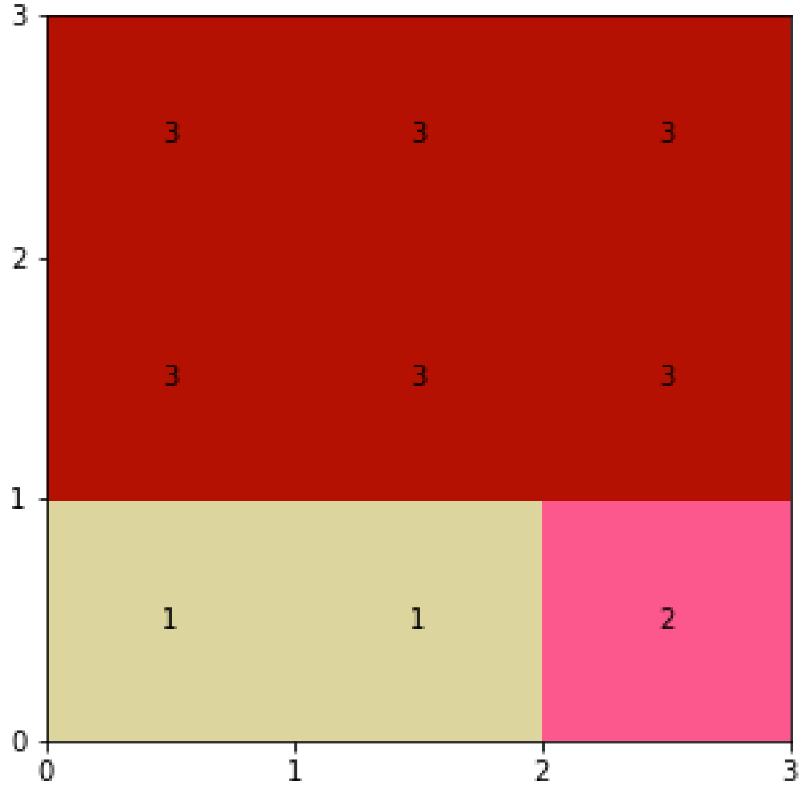
On average across 3850 experiments, it takes 6.0 communication towers before full coverage is ob



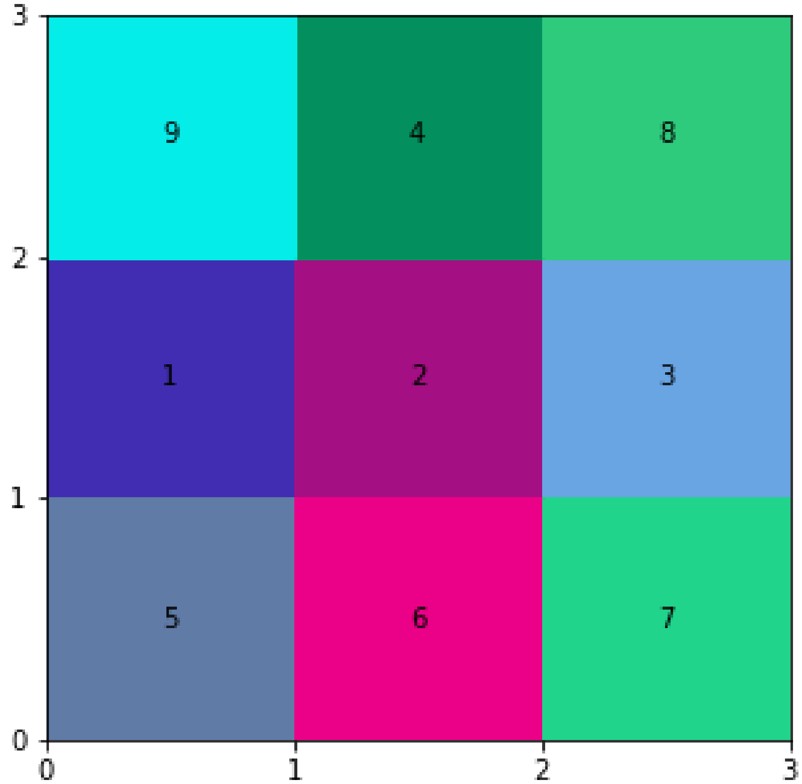
On average across 3900 experiments, it takes 7.0 communication towers before full coverage is ob



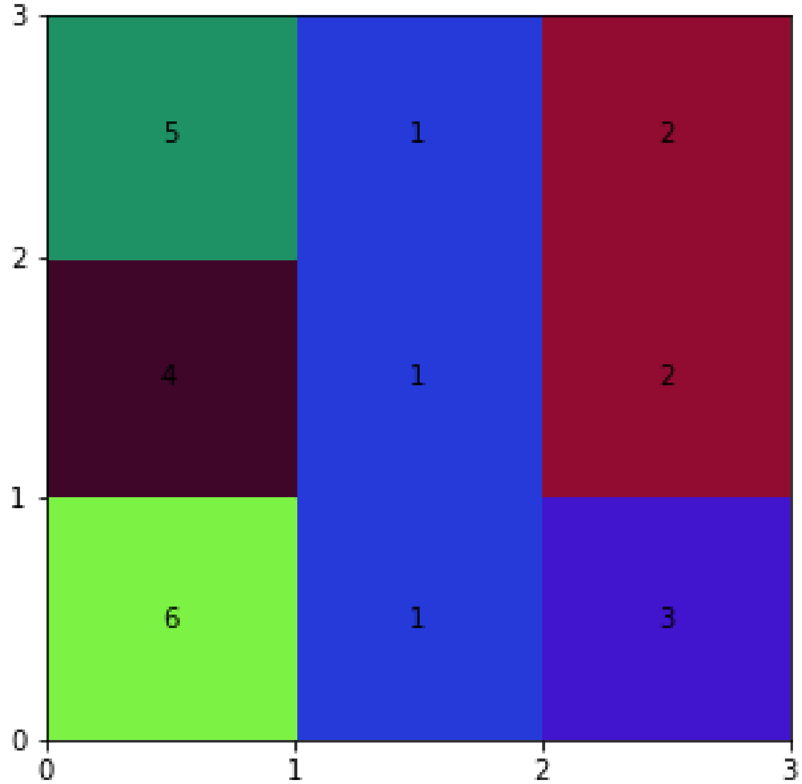
On average across 3950 experiments, it takes 3.0 communication towers before full coverage is ob



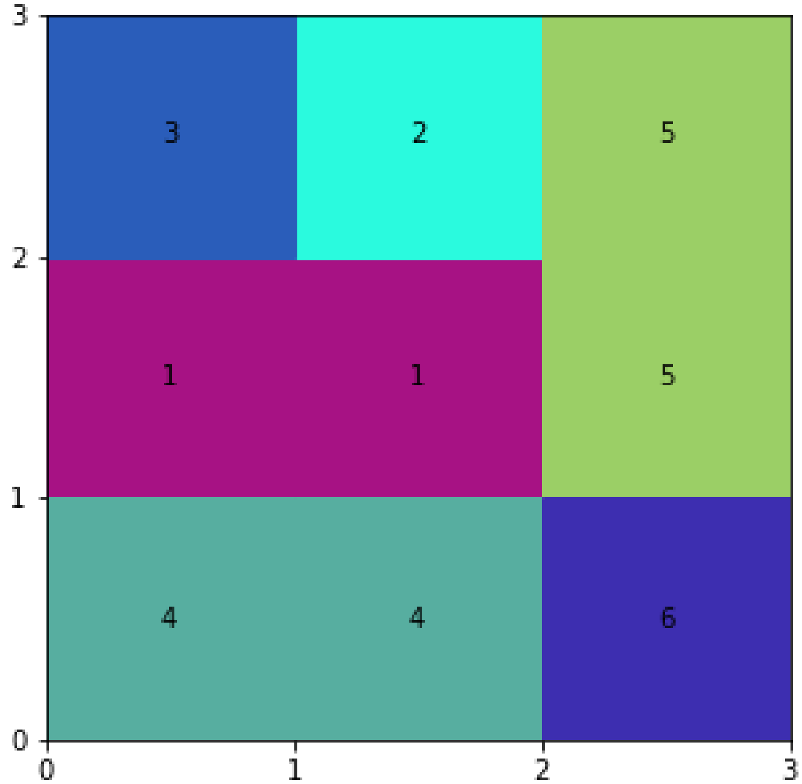

```
On average across 4000 experiments, it takes 9.0 communication towers before full coverage is ob
```



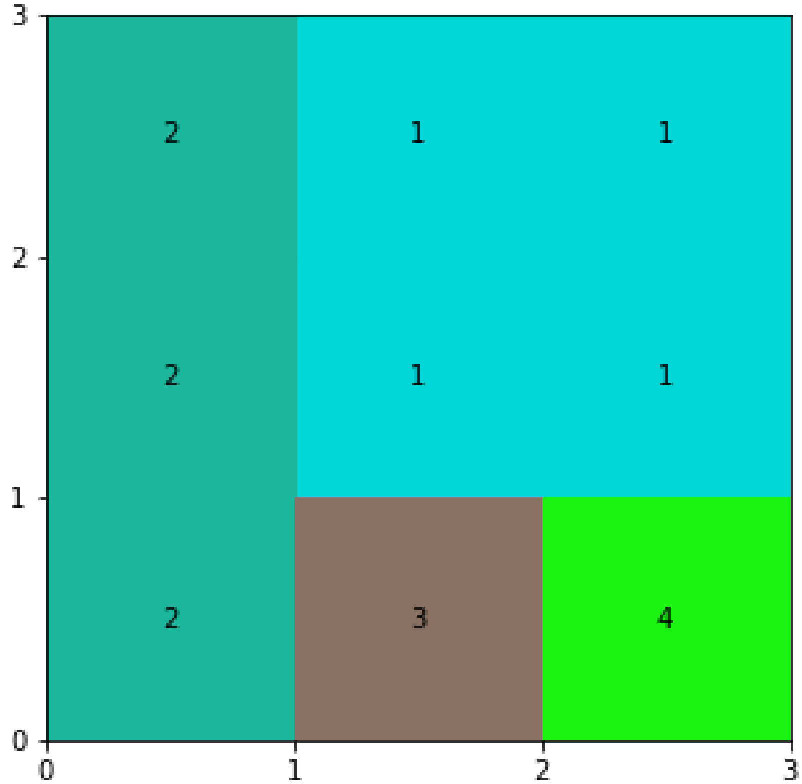
On average across 4050 experiments, it takes 6.0 communication towers before full coverage is ob



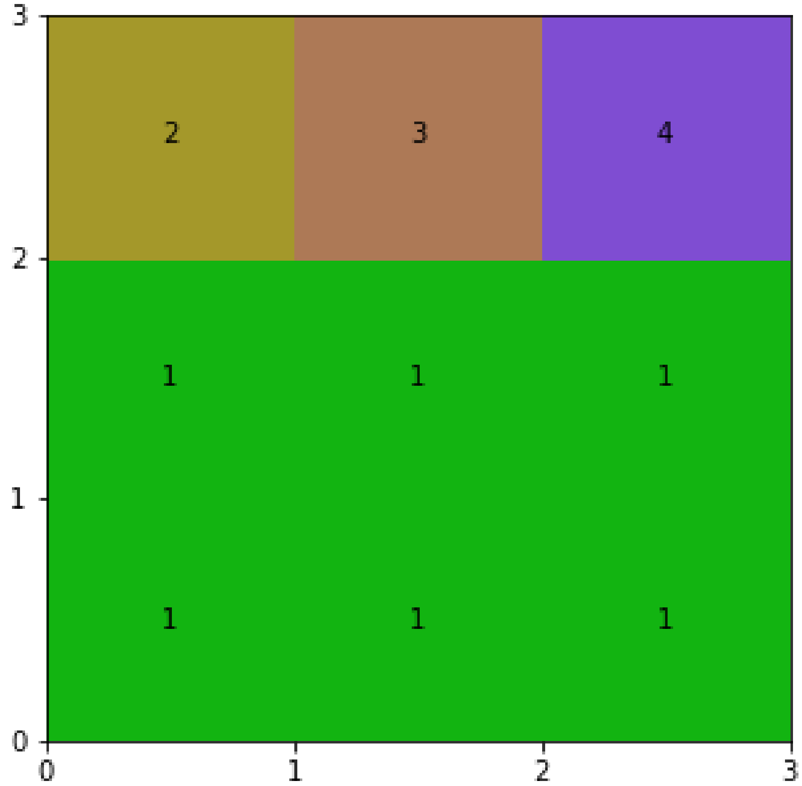
On average across 4100 experiments, it takes 6.0 communication towers before full coverage is ob



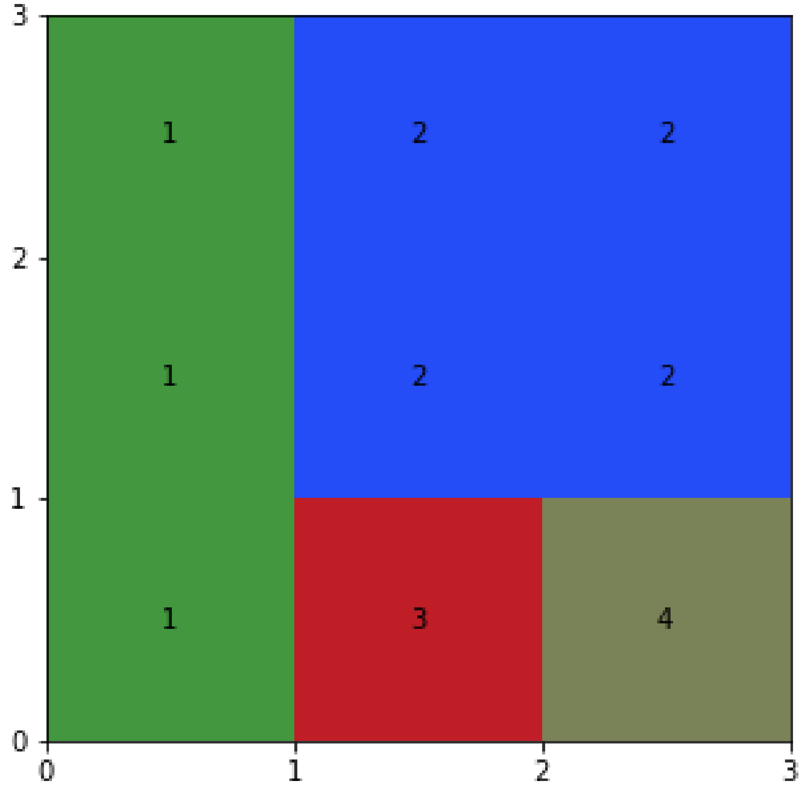
On average across 4150 experiments, it takes 4.0 communication towers before full coverage is ob



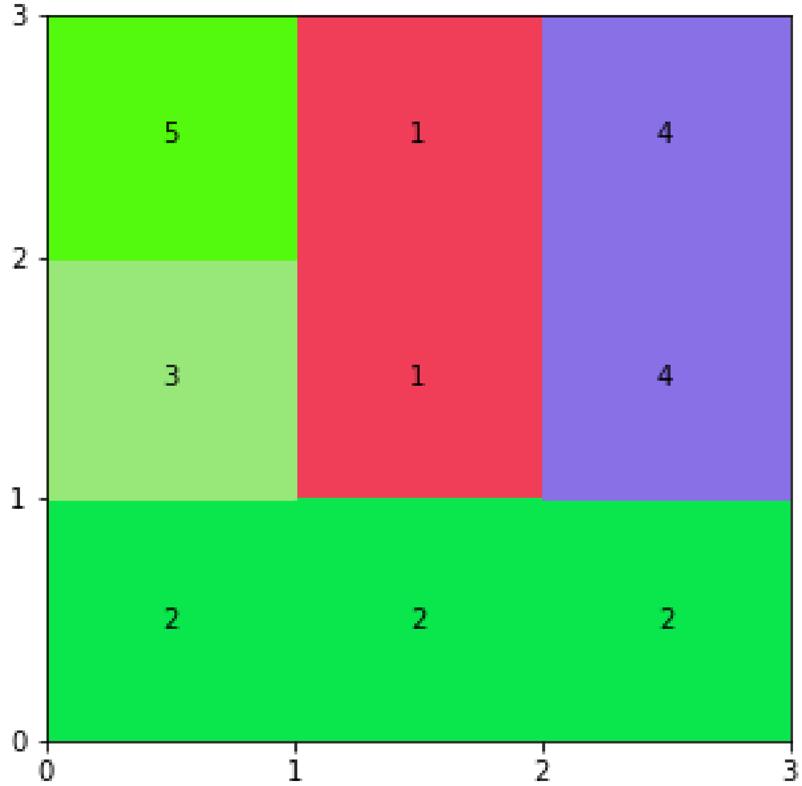

```
On average across 4200 experiments, it takes 4.0 communication towers before full coverage is ob
```



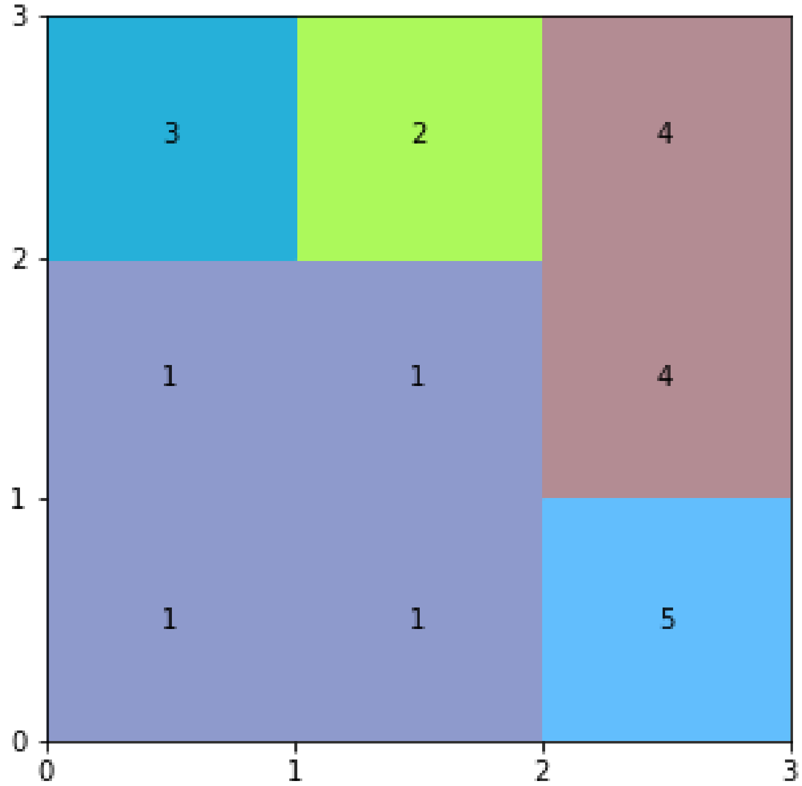
On average across 4250 experiments, it takes 4.0 communication towers before full coverage is ob



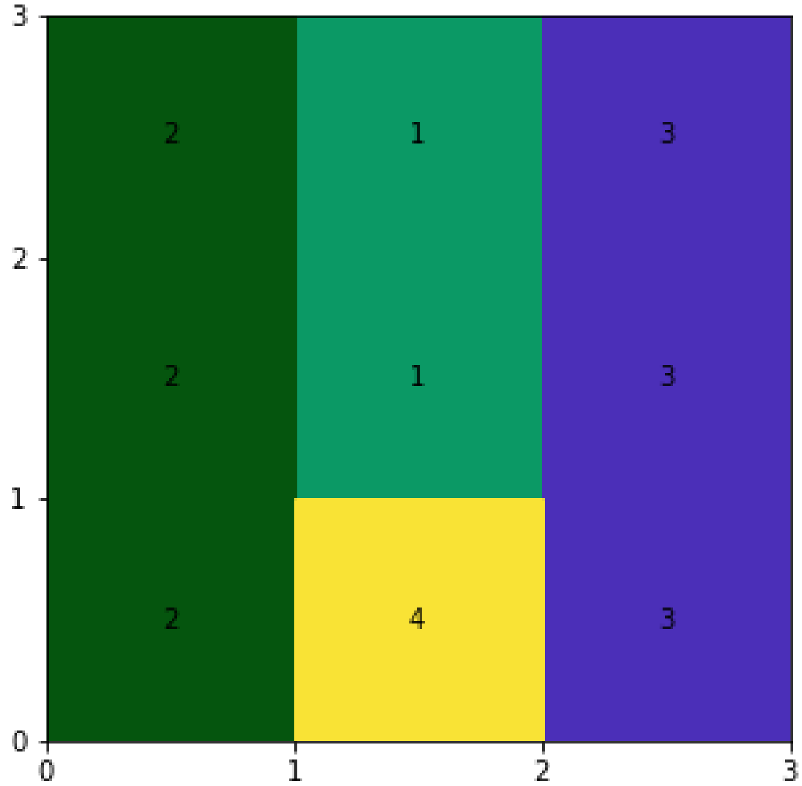
On average across 4300 experiments, it takes 5.0 communication towers before full coverage is ob



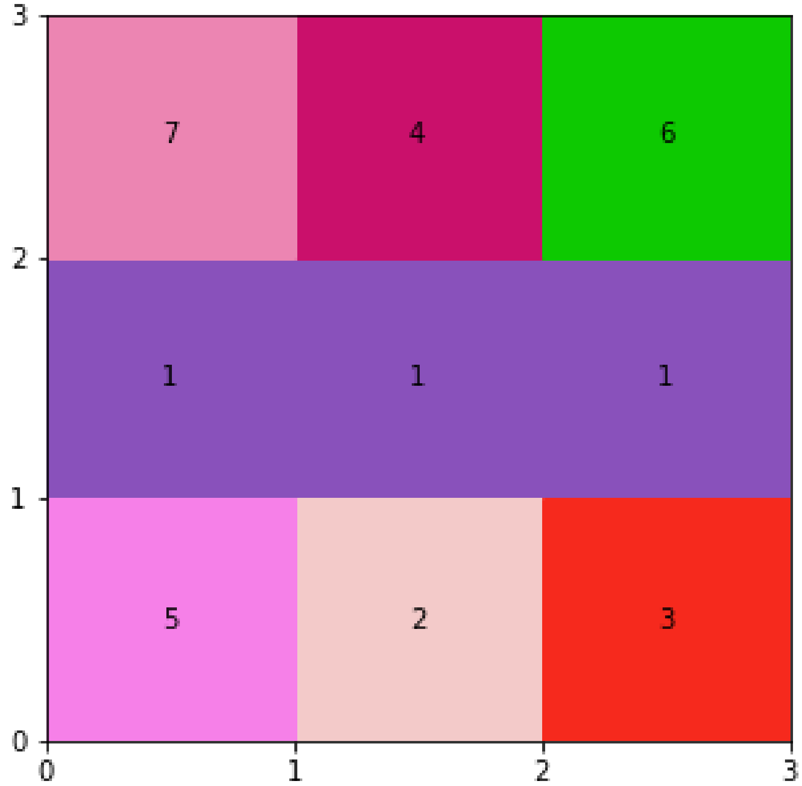
On average across 4350 experiments, it takes 5.0 communication towers before full coverage is ob



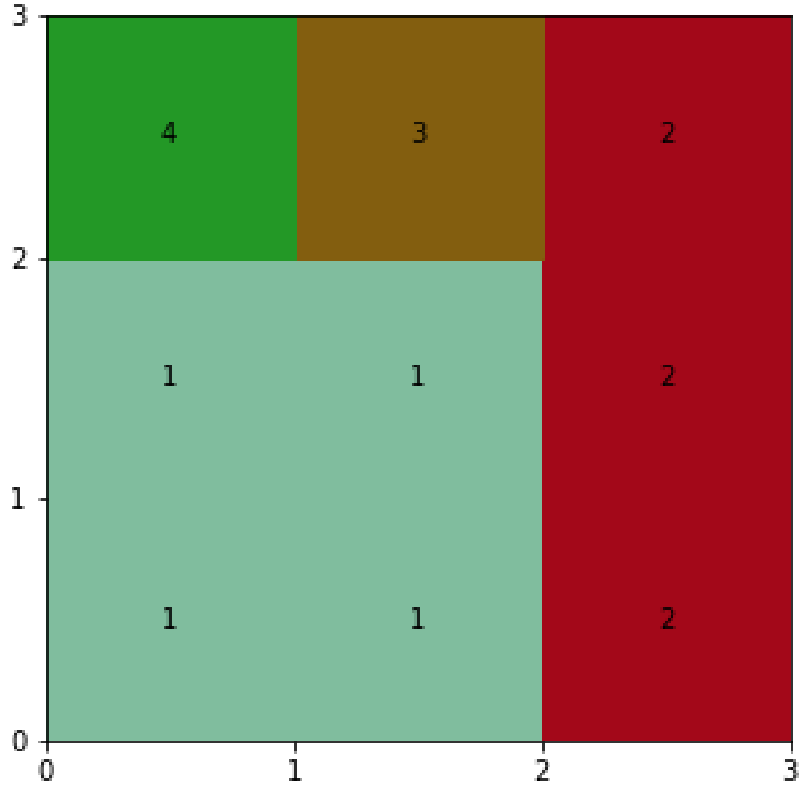
On average across 4400 experiments, it takes 4.0 communication towers before full coverage is ob



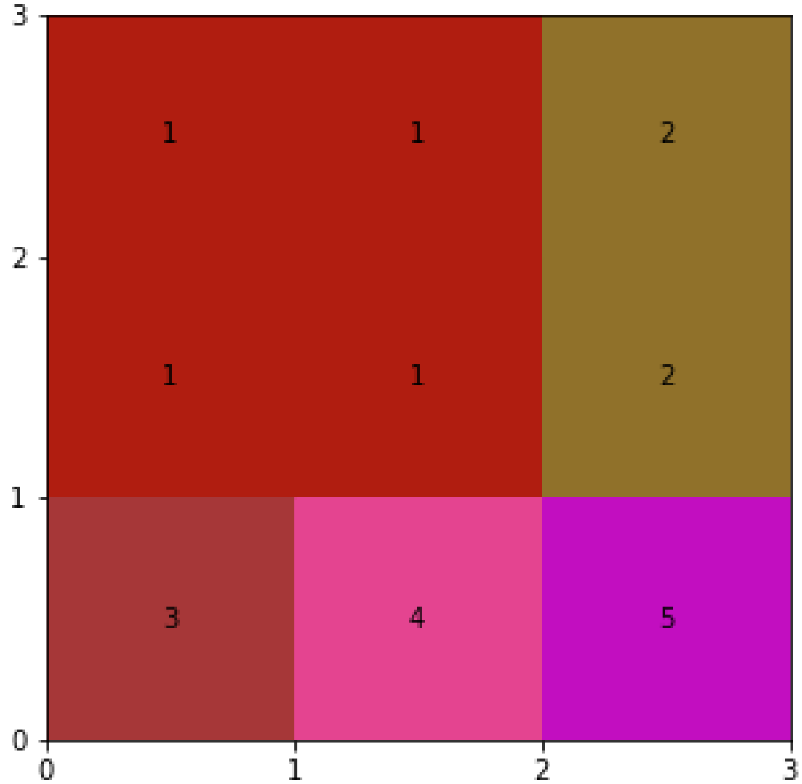
On average across 4450 experiments, it takes 7.0 communication towers before full coverage is ob



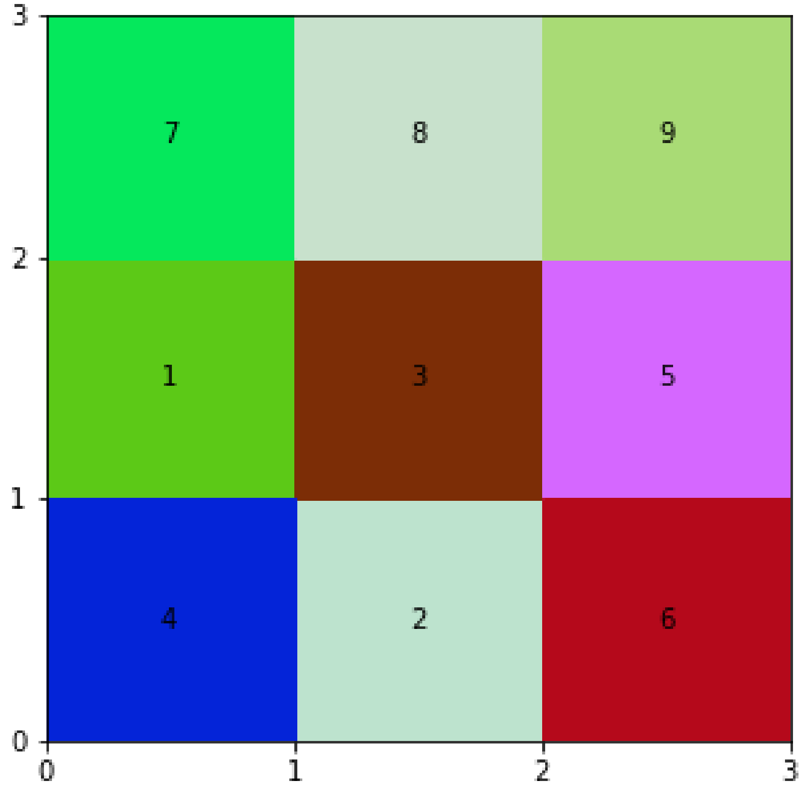
On average across 4500 experiments, it takes 4.0 communication towers before full coverage is ob



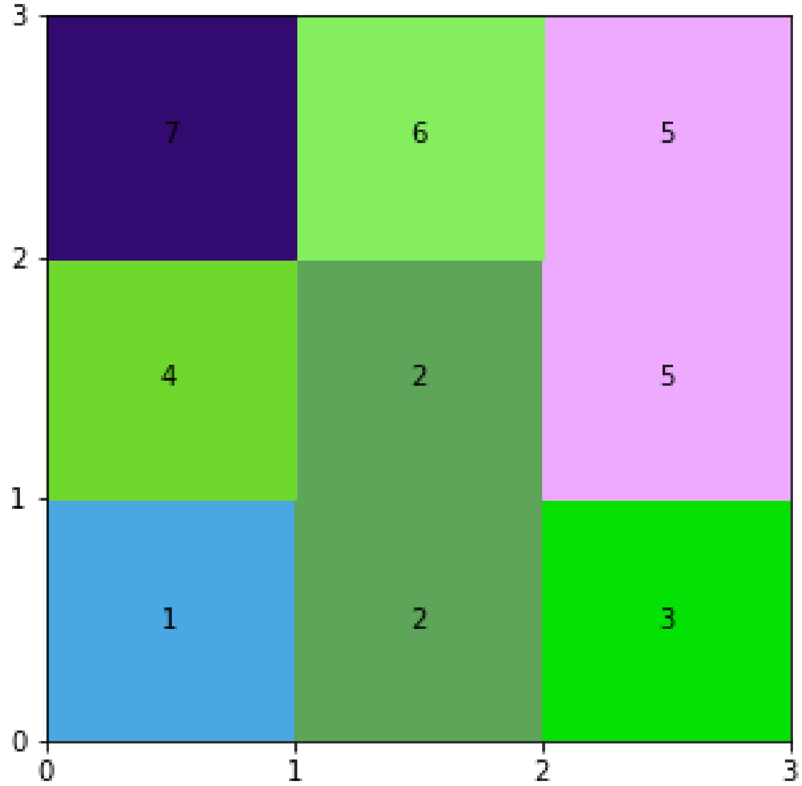

```
On average across 4550 experiments, it takes 5.0 communication towers before full coverage is ob
```



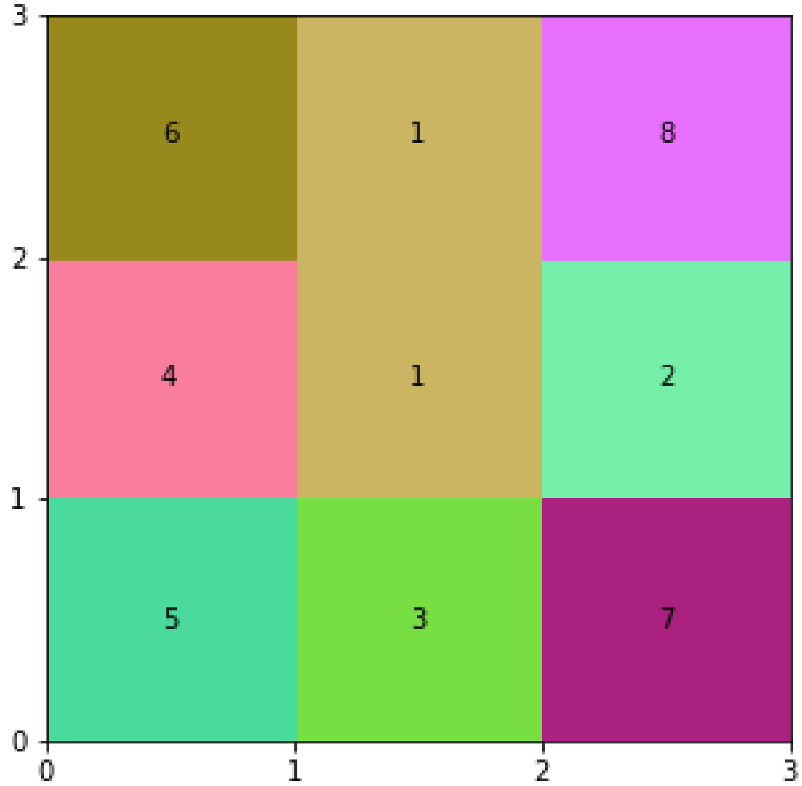
On average across 4600 experiments, it takes 9.0 communication towers before full coverage is ob



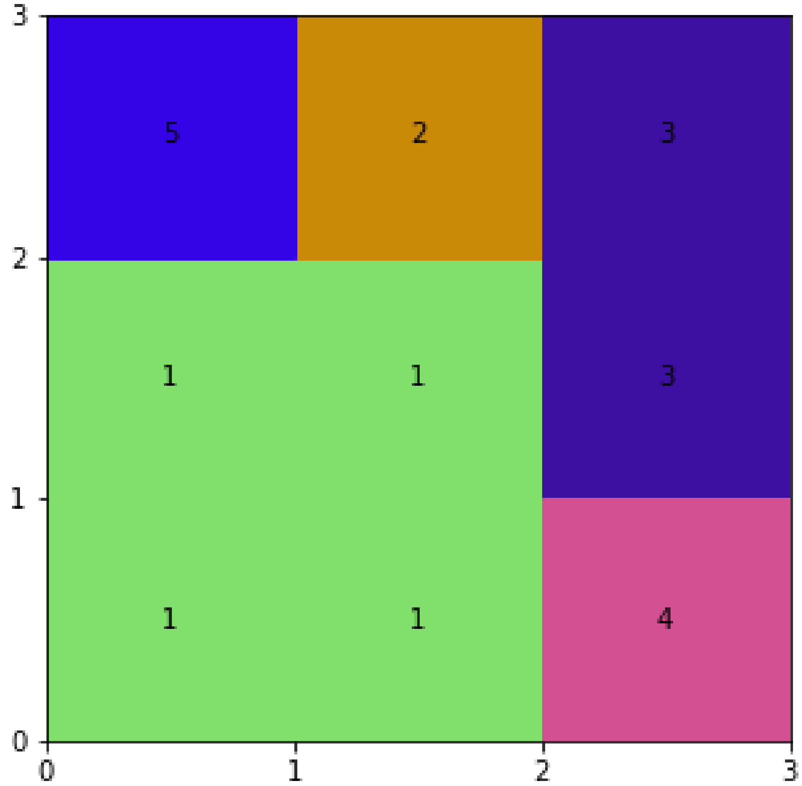
On average across 4650 experiments, it takes 7.0 communication towers before full coverage is ob



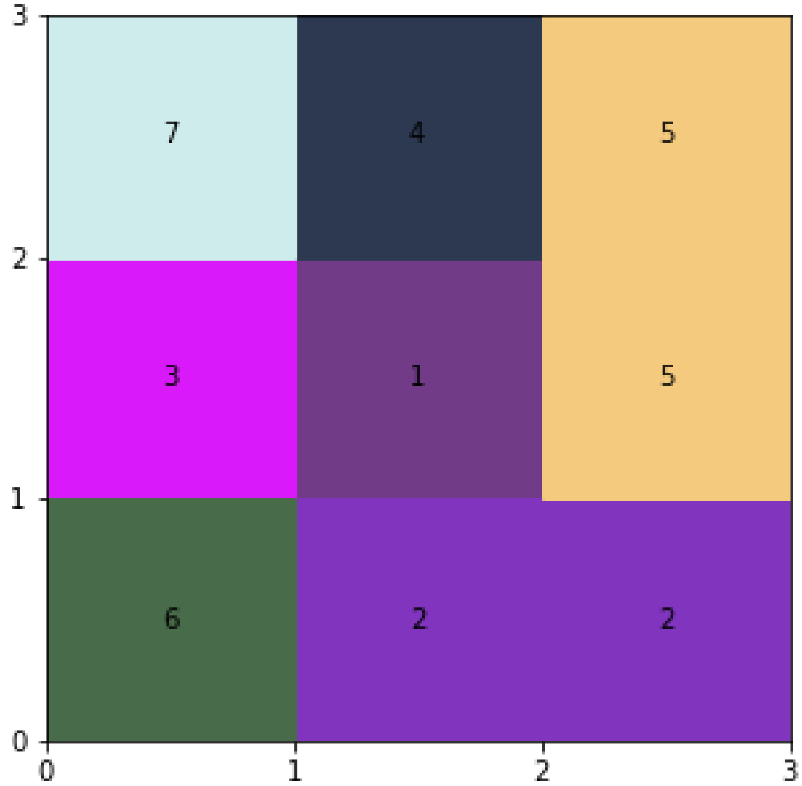
On average across 4700 experiments, it takes 8.0 communication towers before full coverage is ob



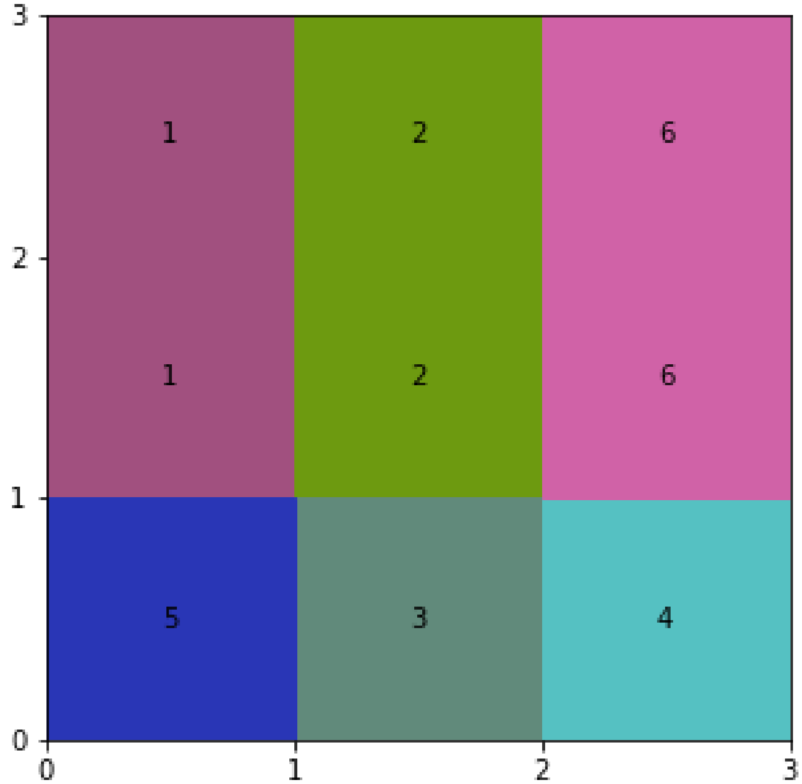
On average across 4750 experiments, it takes 5.0 communication towers before full coverage is ob



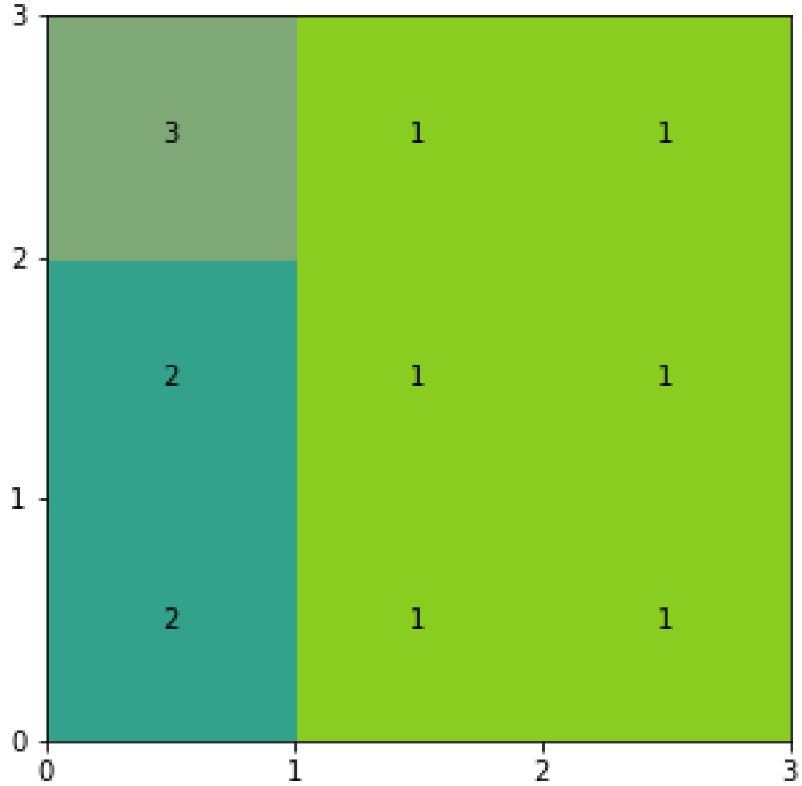
On average across 4800 experiments, it takes 7.0 communication towers before full coverage is ob



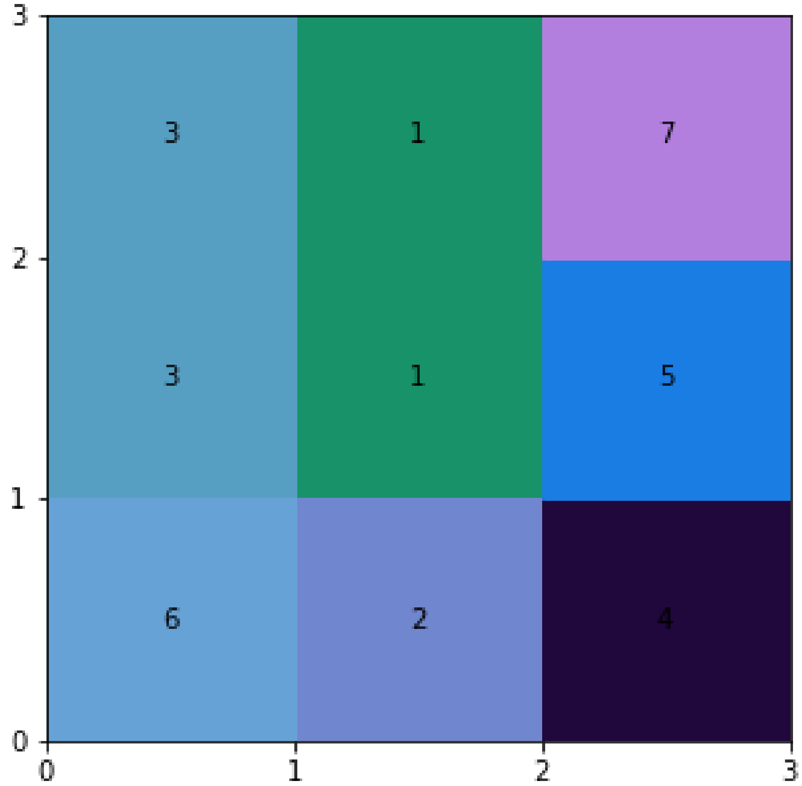
On average across 4850 experiments, it takes 6.0 communication towers before full coverage is ob



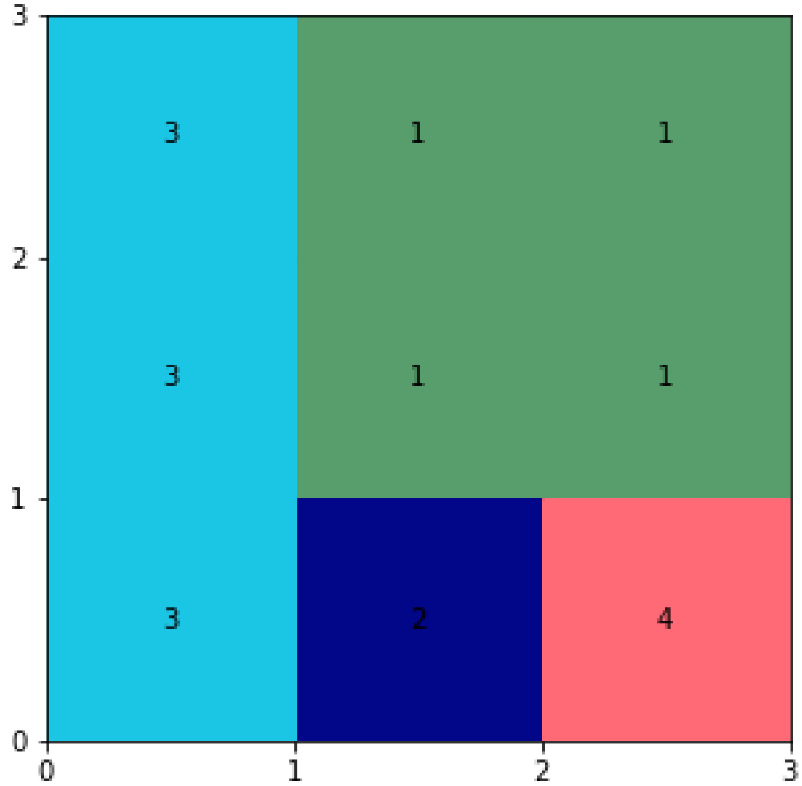
On average across 4900 experiments, it takes 3.0 communication towers before full coverage is ob



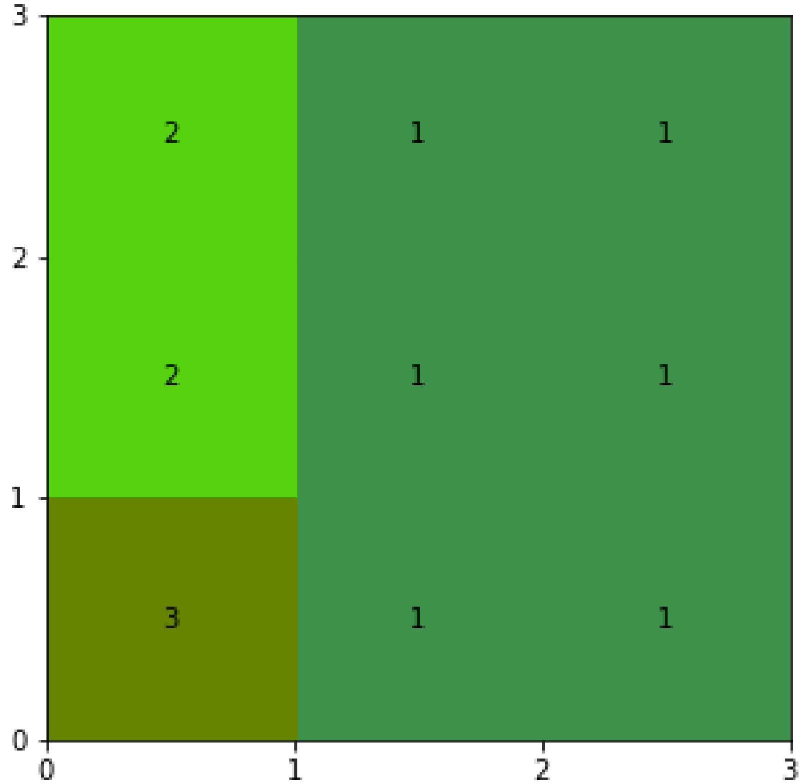
On average across 4950 experiments, it takes 7.0 communication towers before full coverage is ob



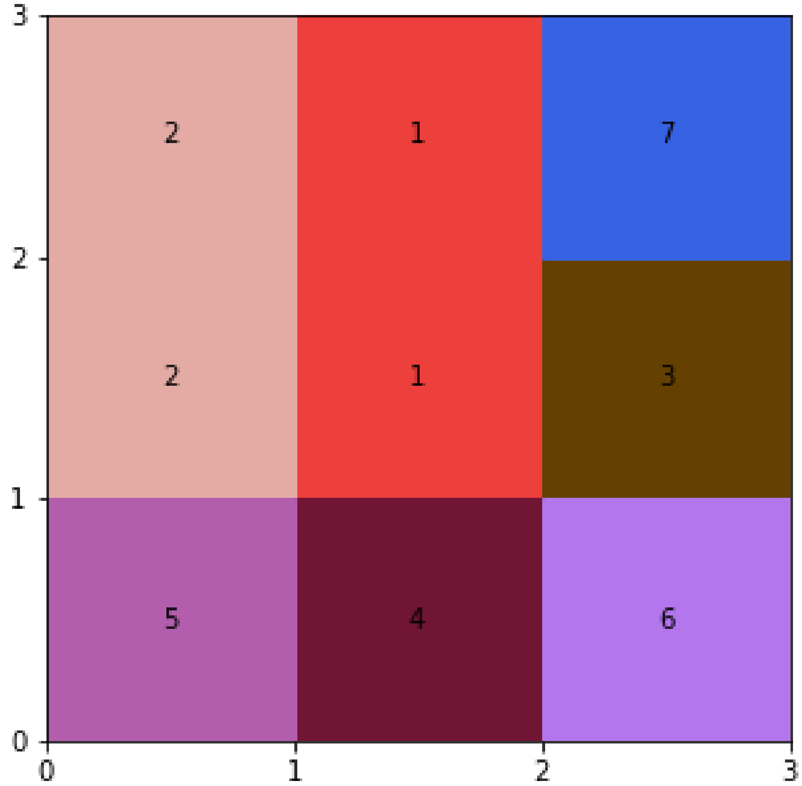

```
On average across 5000 experiments, it takes 4.0 communication towers before full coverage is ob
```



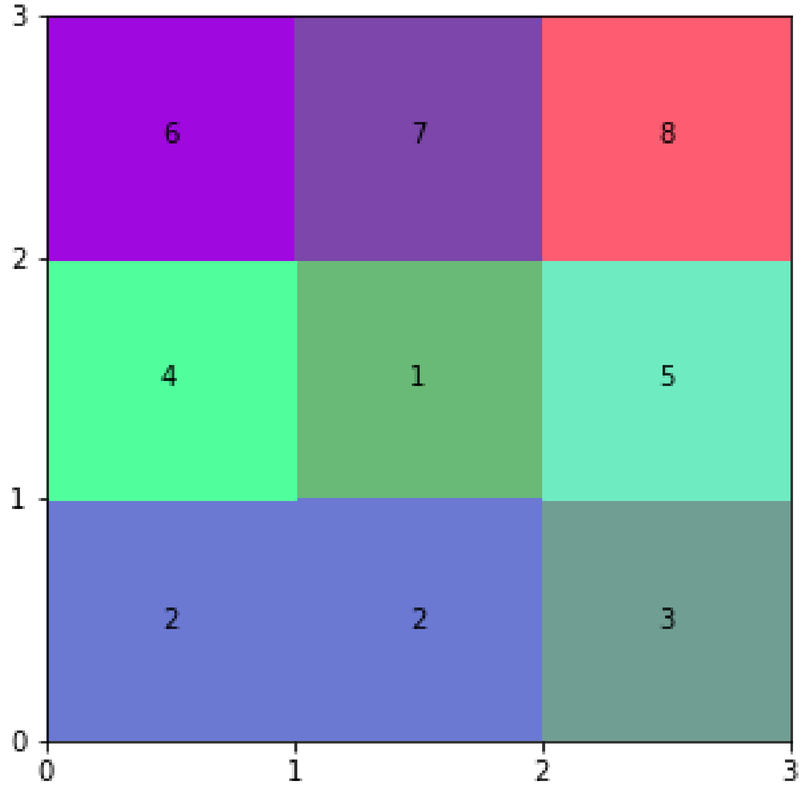

```
On average across 5050 experiments, it takes 3.0 communication towers before full coverage is ob
```



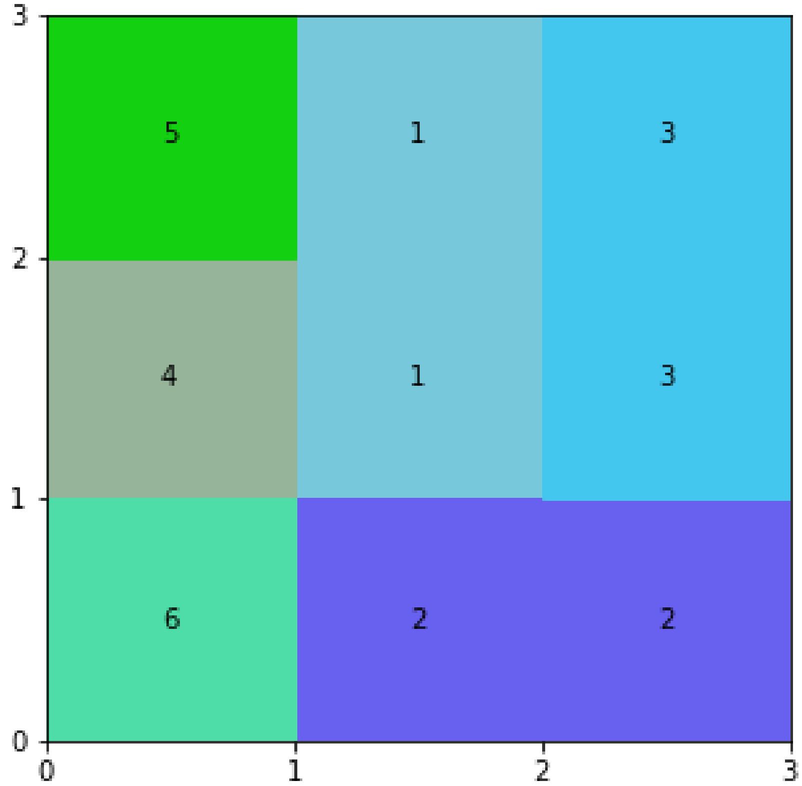
On average across 5100 experiments, it takes 7.0 communication towers before full coverage is ob



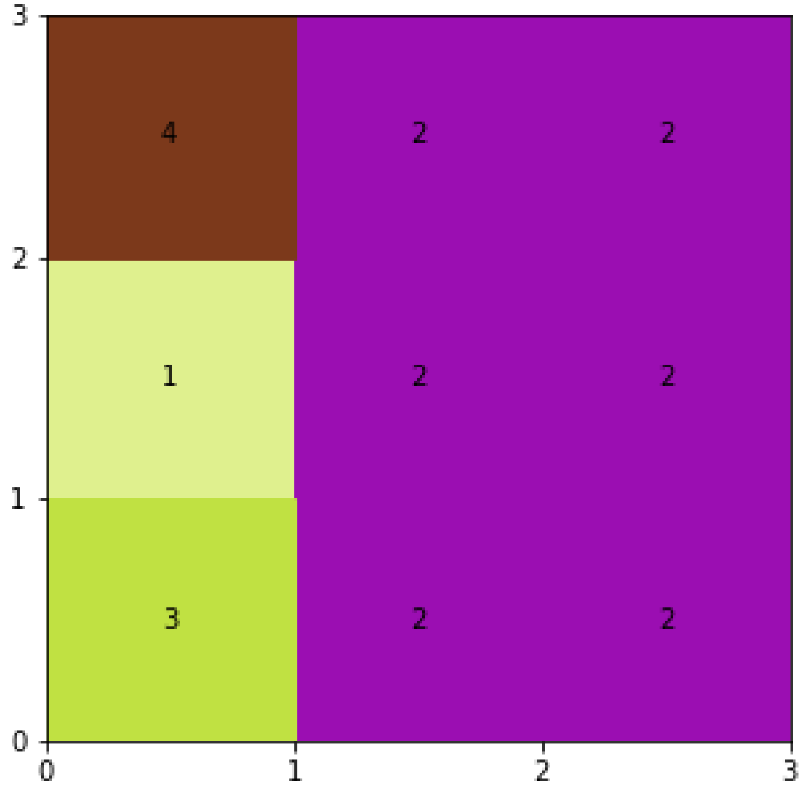
On average across 5150 experiments, it takes 8.0 communication towers before full coverage is ob



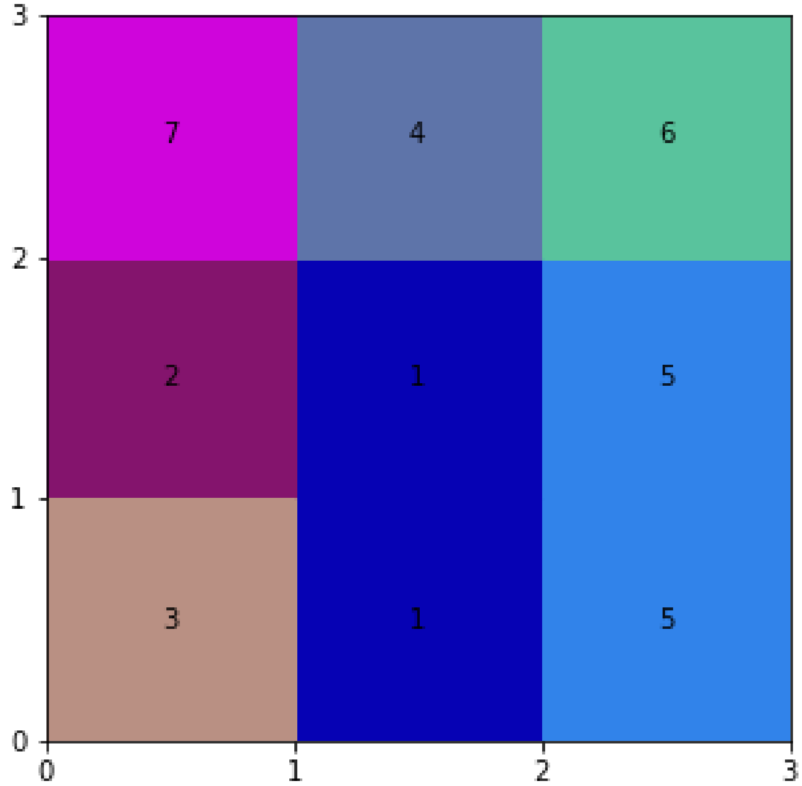

```
On average across 5200 experiments, it takes 6.0 communication towers before full coverage is ob
```



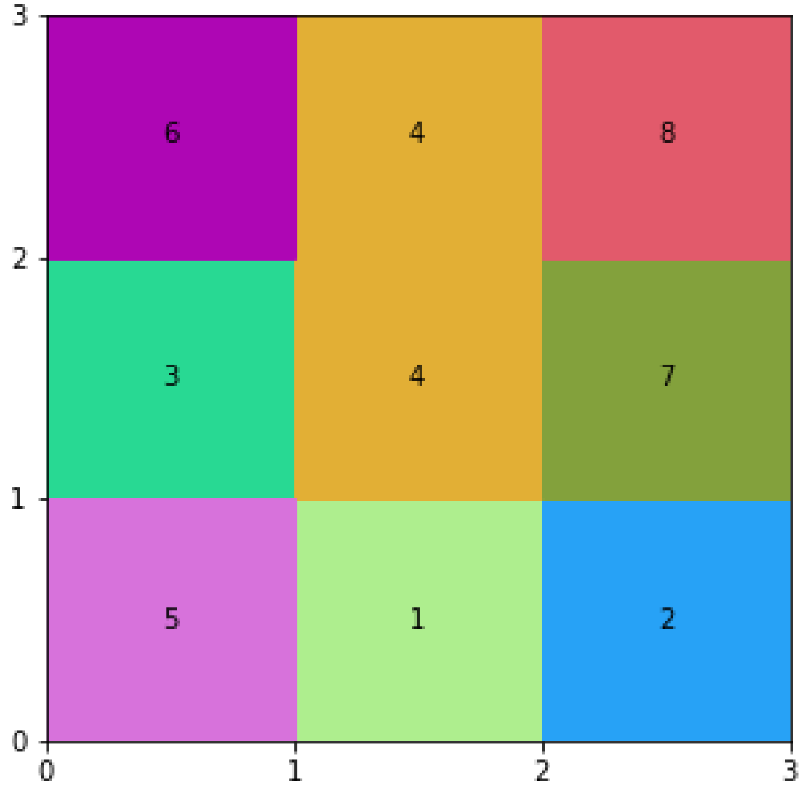

```
On average across 5250 experiments, it takes 4.0 communication towers before full coverage is ob
```



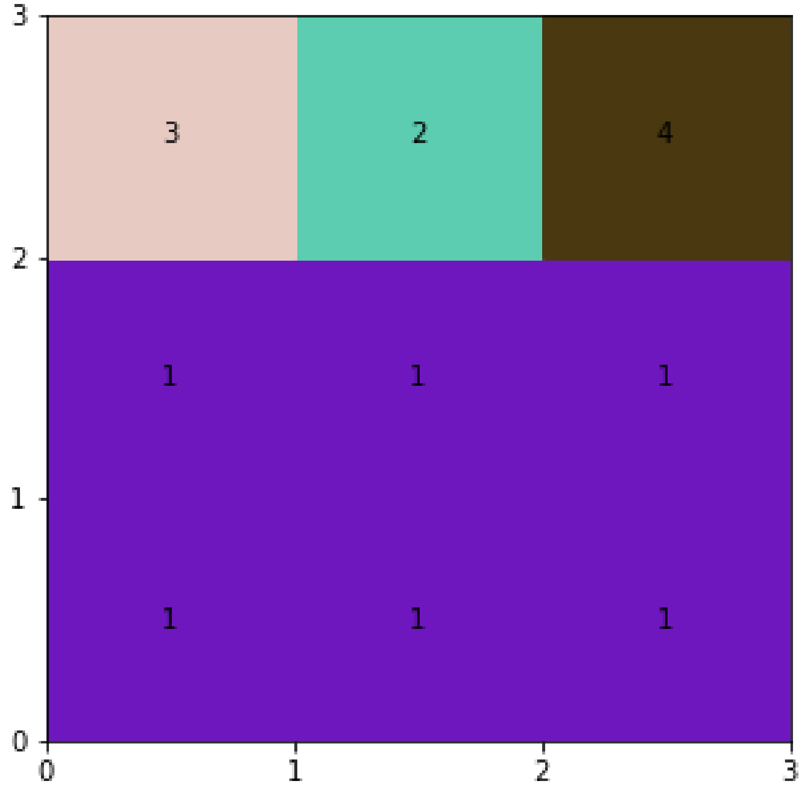
`On average across 5300 experiments, it takes 7.0 communication towers before full coverage is ob`



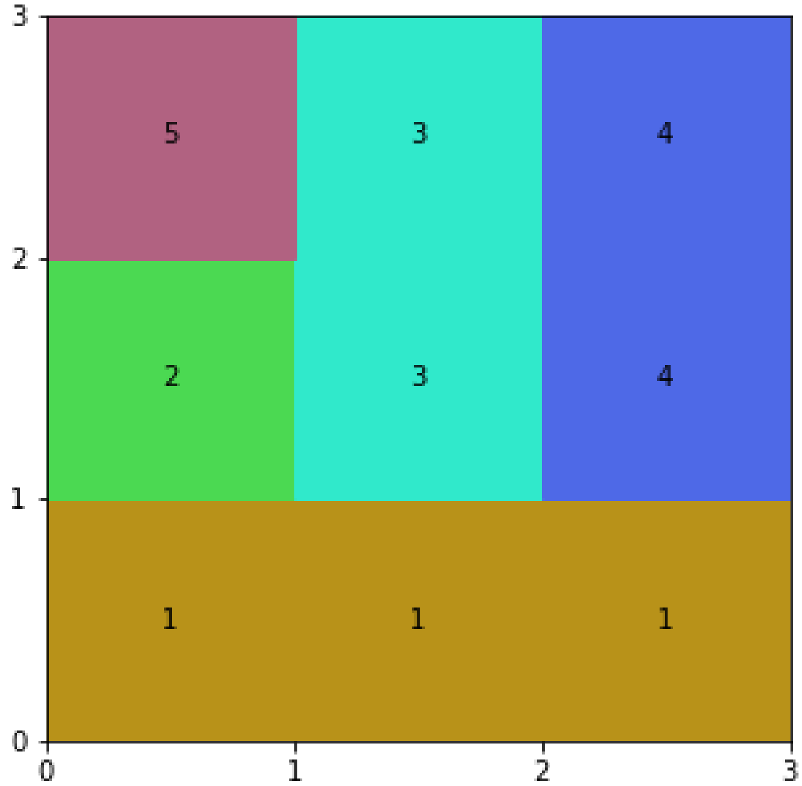

```
On average across 5350 experiments, it takes 8.0 communication towers before full coverage is ob
```



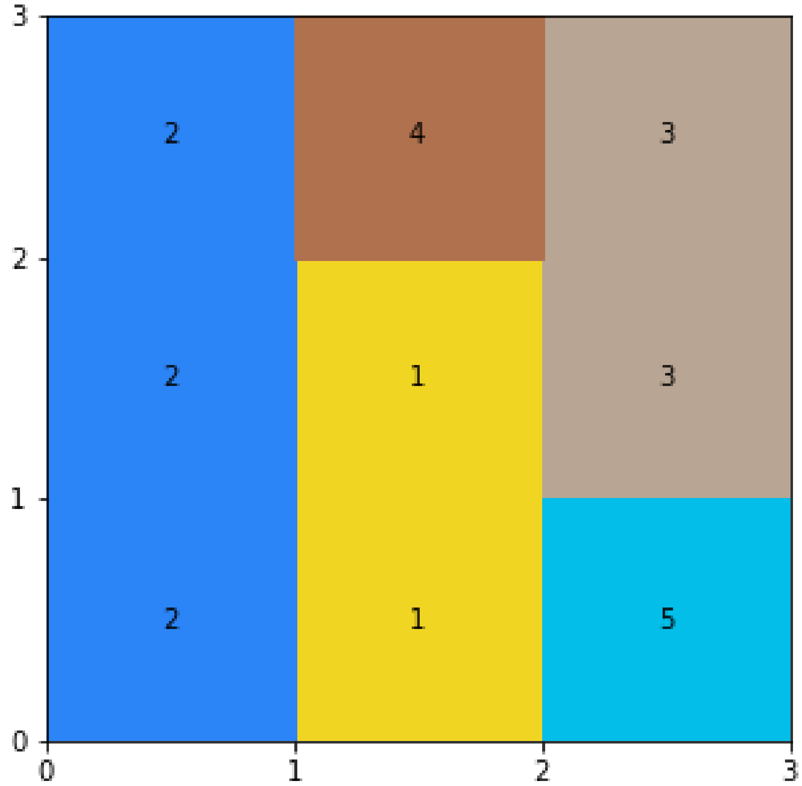
On average across 5400 experiments, it takes 4.0 communication towers before full coverage is ob



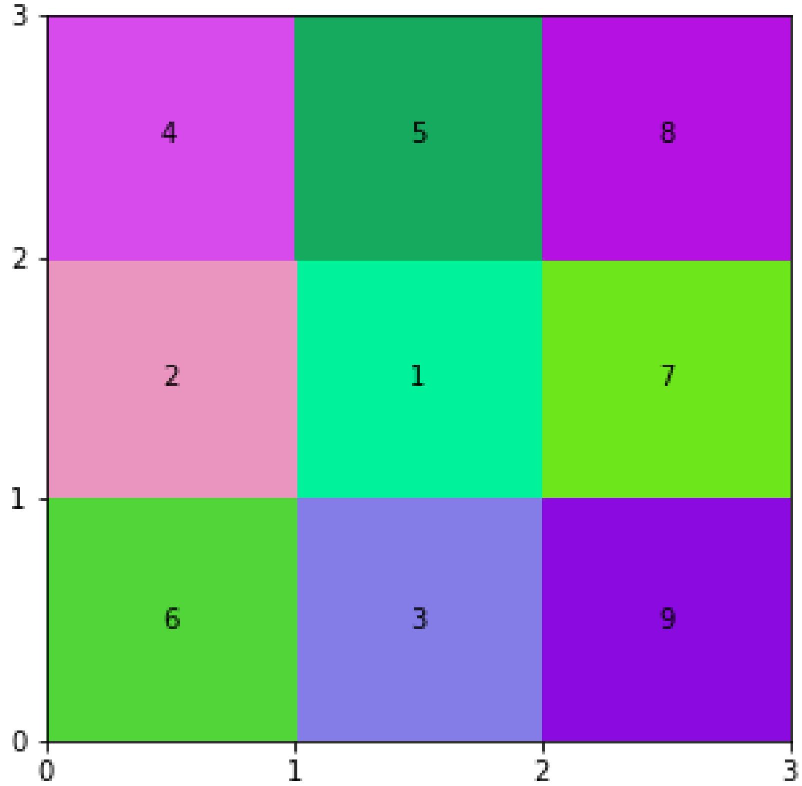
On average across 5450 experiments, it takes 5.0 communication towers before full coverage is ob



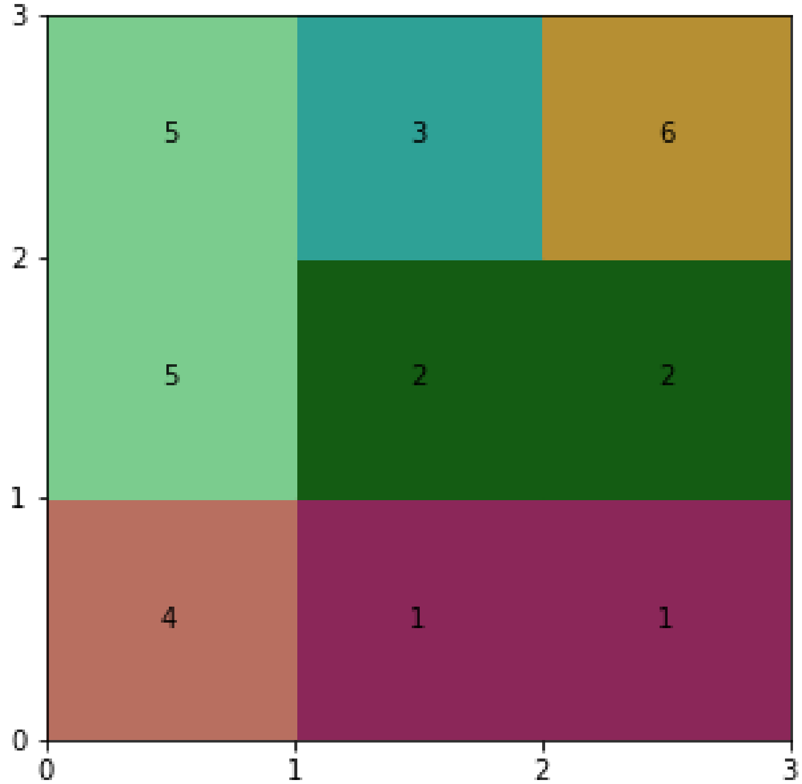
On average across 5500 experiments, it takes 5.0 communication towers before full coverage is ob



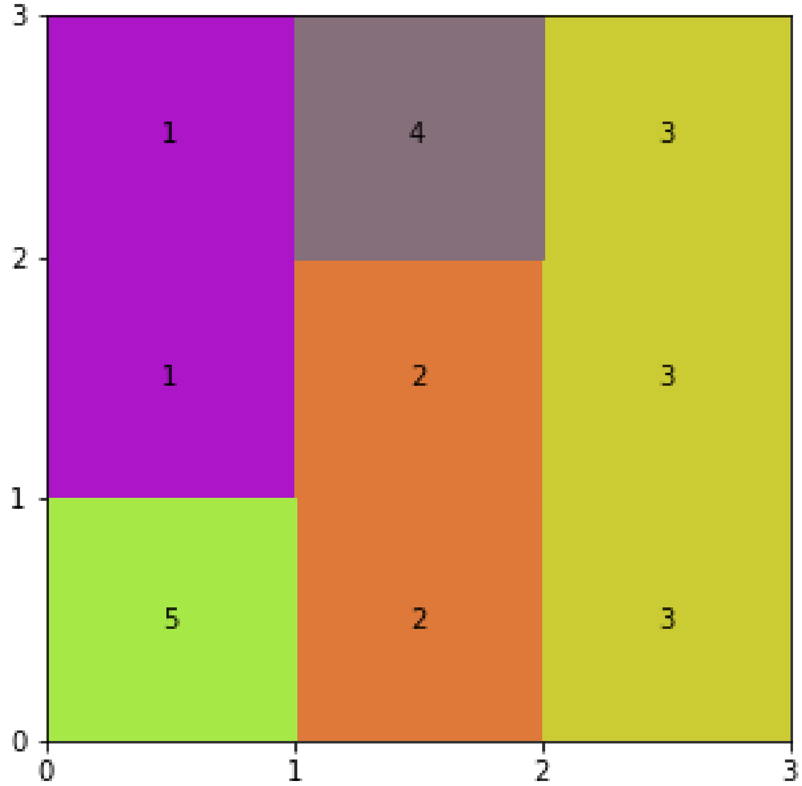
On average across 5550 experiments, it takes 9.0 communication towers before full coverage is ob



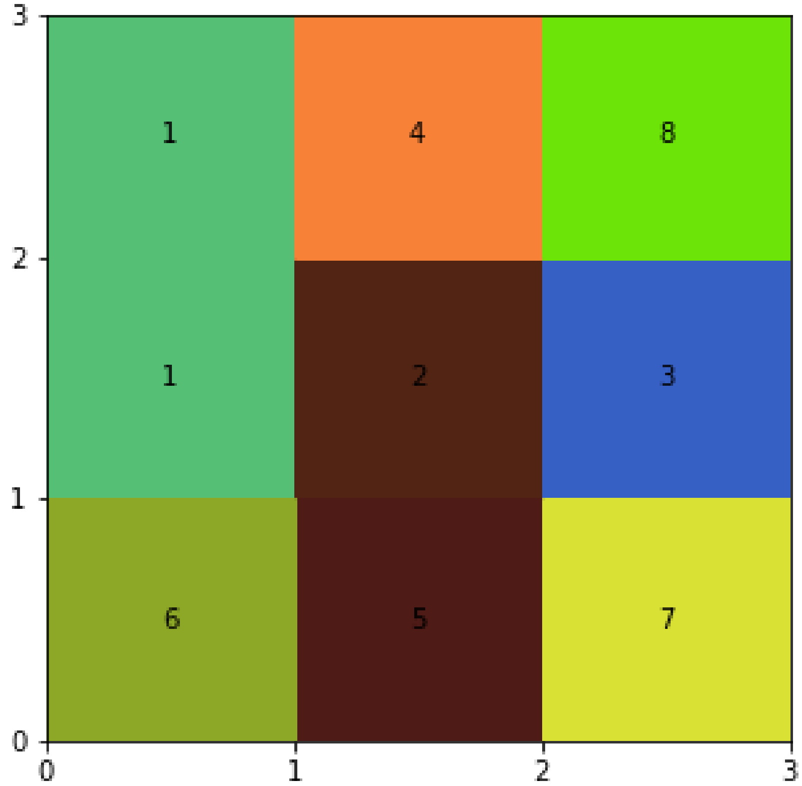
On average across 5600 experiments, it takes 6.0 communication towers before full coverage is ob



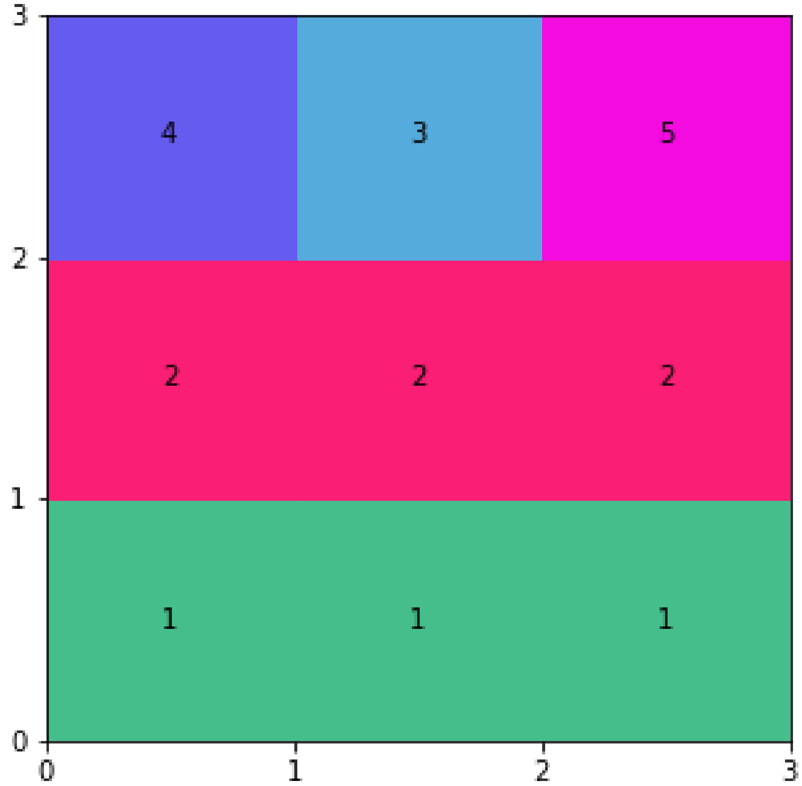
On average across 5650 experiments, it takes 5.0 communication towers before full coverage is ob



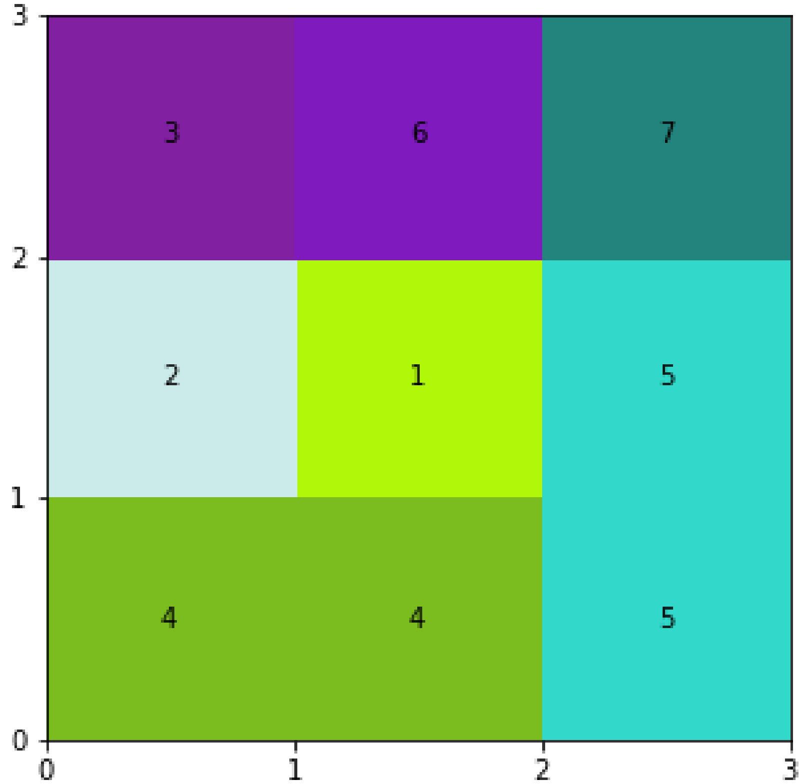
On average across 5700 experiments, it takes 8.0 communication towers before full coverage is ob



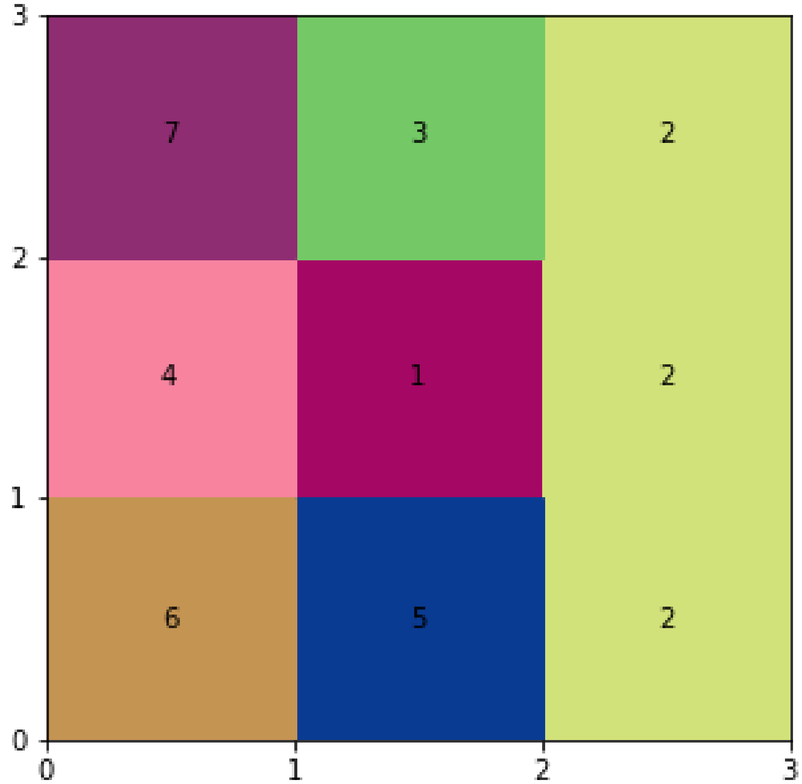
On average across 5750 experiments, it takes 5.0 communication towers before full coverage is ob



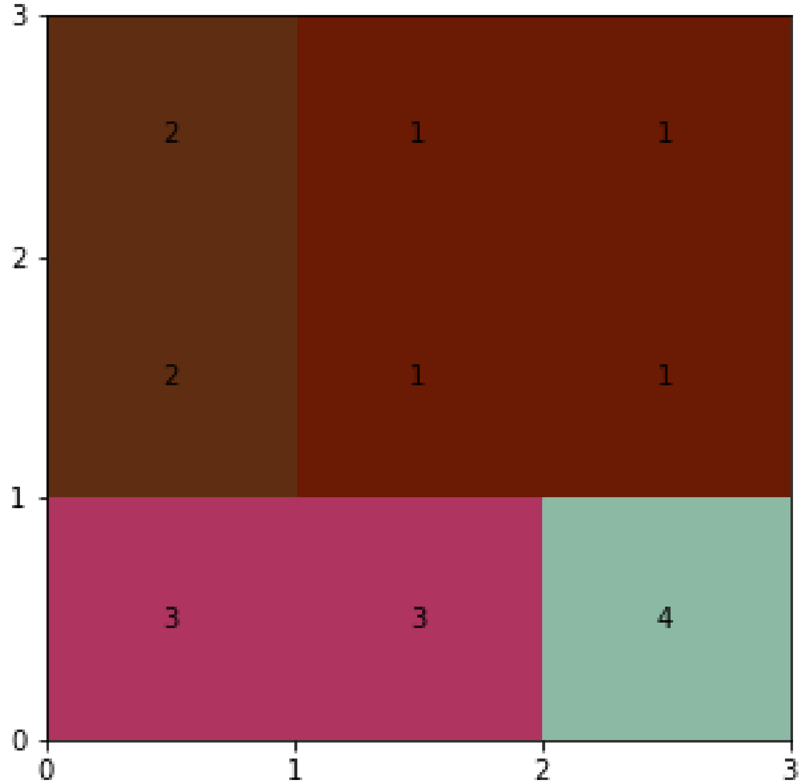

```
On average across 5800 experiments, it takes 7.0 communication towers before full coverage is ob
```



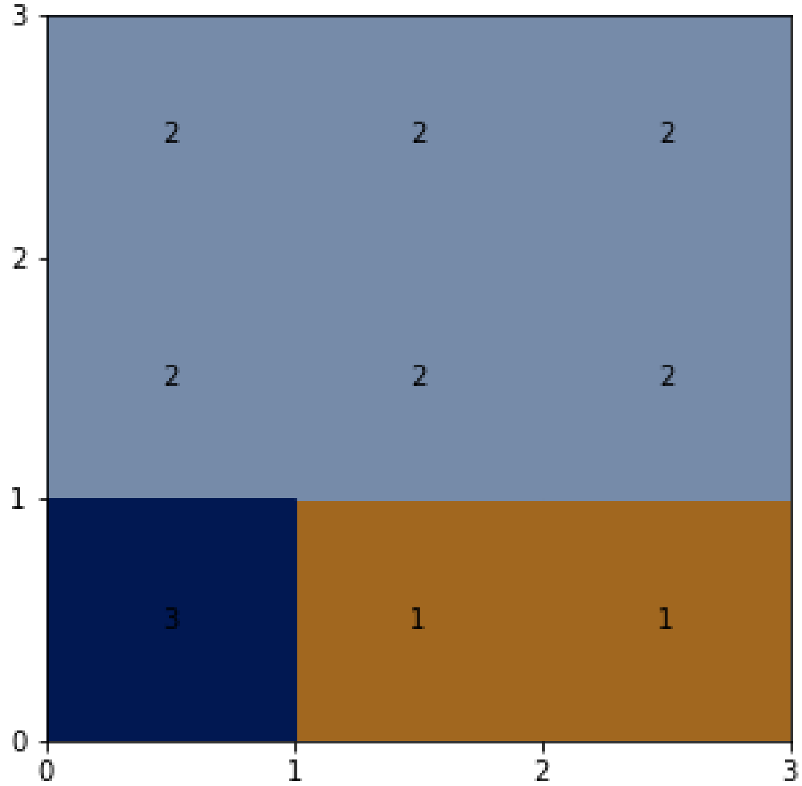
On average across 5850 experiments, it takes 7.0 communication towers before full coverage is ob



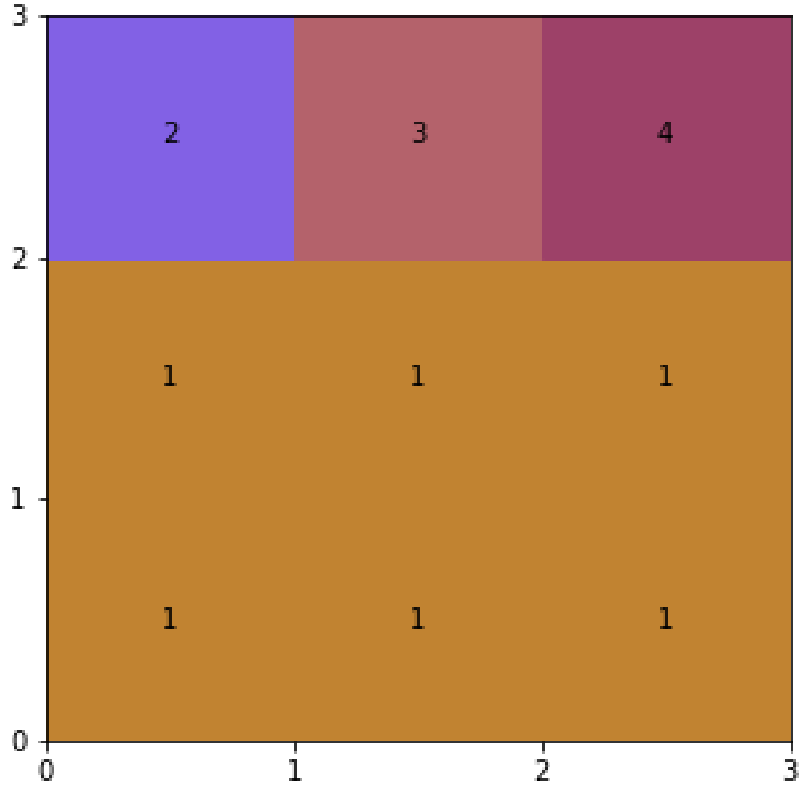
On average across 5900 experiments, it takes 4.0 communication towers before full coverage is ob



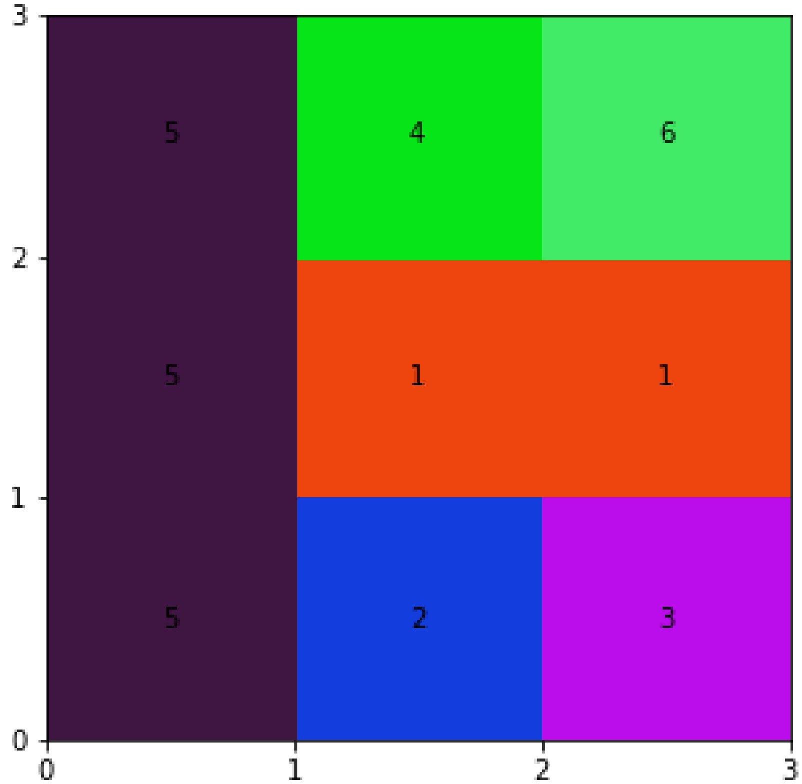

```
On average across 5950 experiments, it takes 3.0 communication towers before full coverage is ob
```



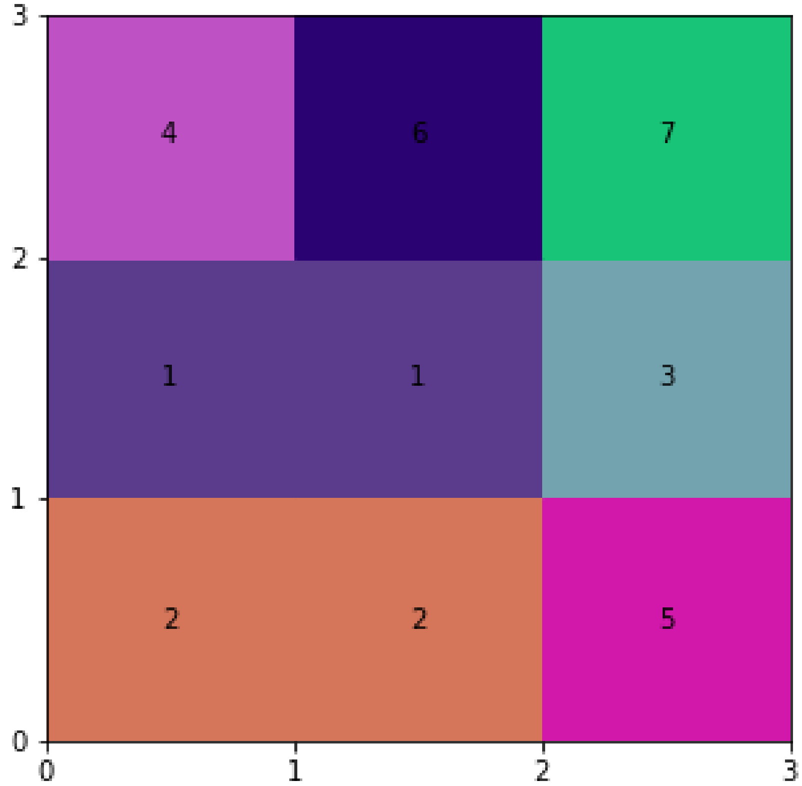
On average across 6000 experiments, it takes 4.0 communication towers before full coverage is ob



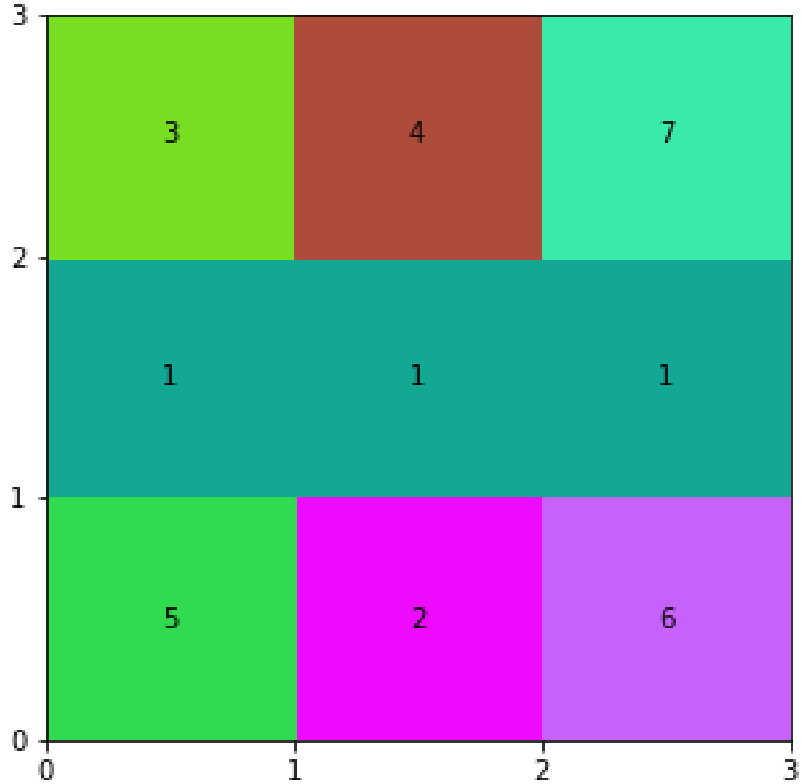

```
On average across 6050 experiments, it takes 6.0 communication towers before full coverage is ob
```



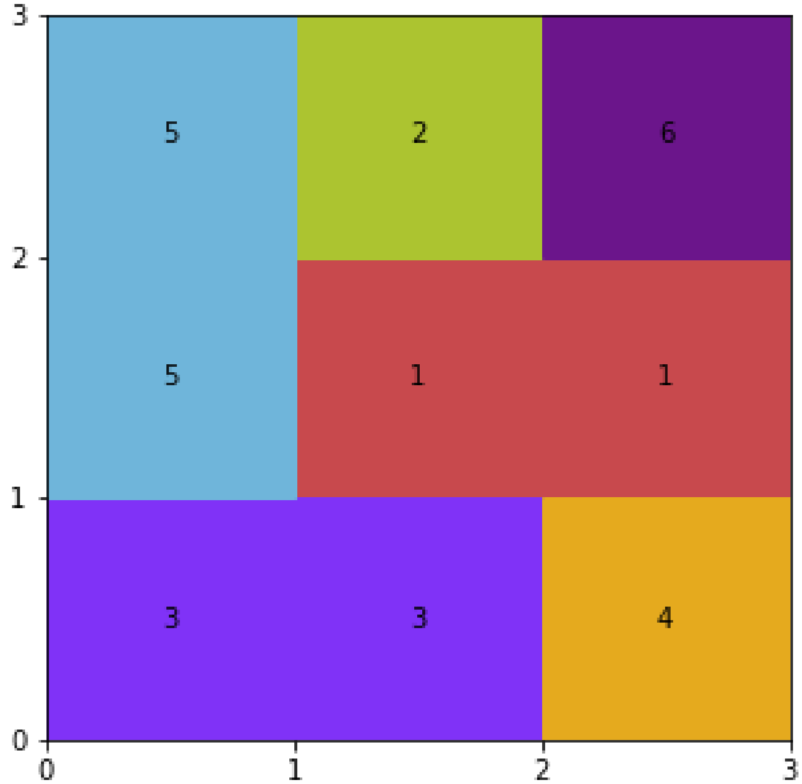
On average across 6100 experiments, it takes 7.0 communication towers before full coverage is ob



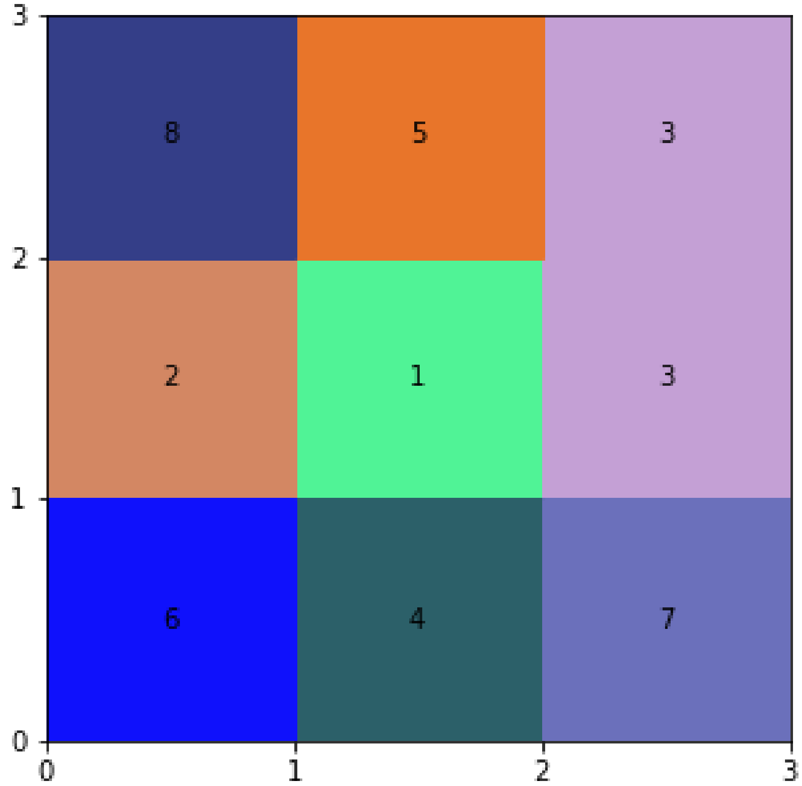

```
On average across 6150 experiments, it takes 7.0 communication towers before full coverage is ob
```



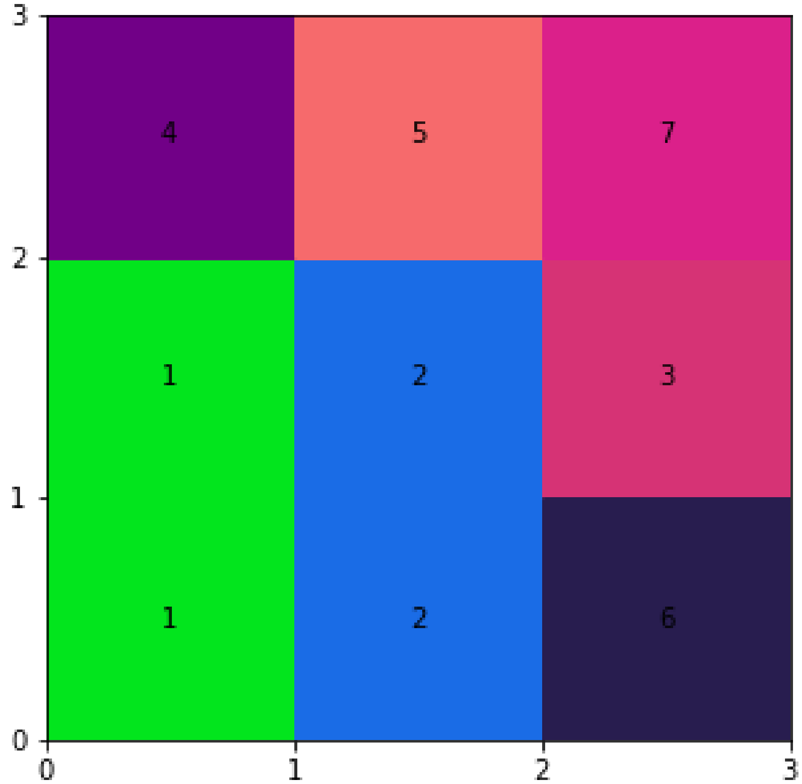
On average across 6200 experiments, it takes 6.0 communication towers before full coverage is ob



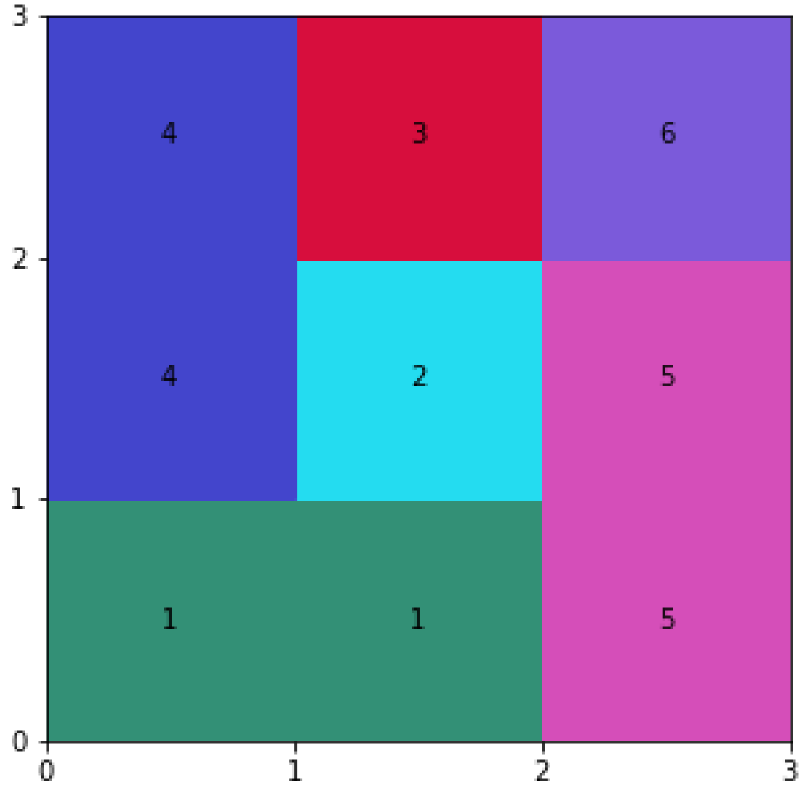

```
On average across 6250 experiments, it takes 8.0 communication towers before full coverage is ob
```



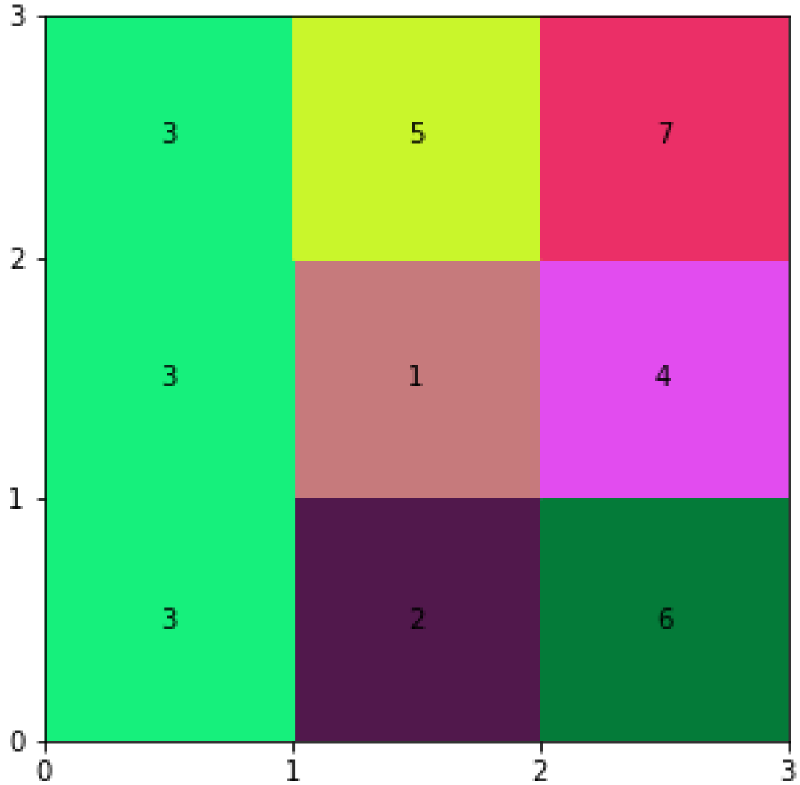
On average across 6300 experiments, it takes 7.0 communication towers before full coverage is ob



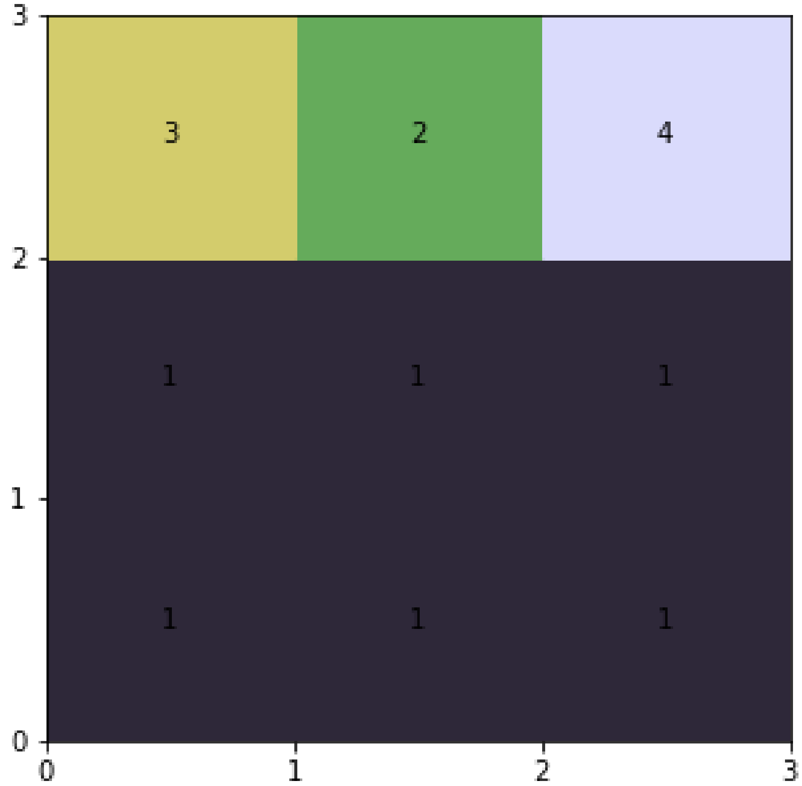

```
On average across 6350 experiments, it takes 6.0 communication towers before full coverage is ob
```



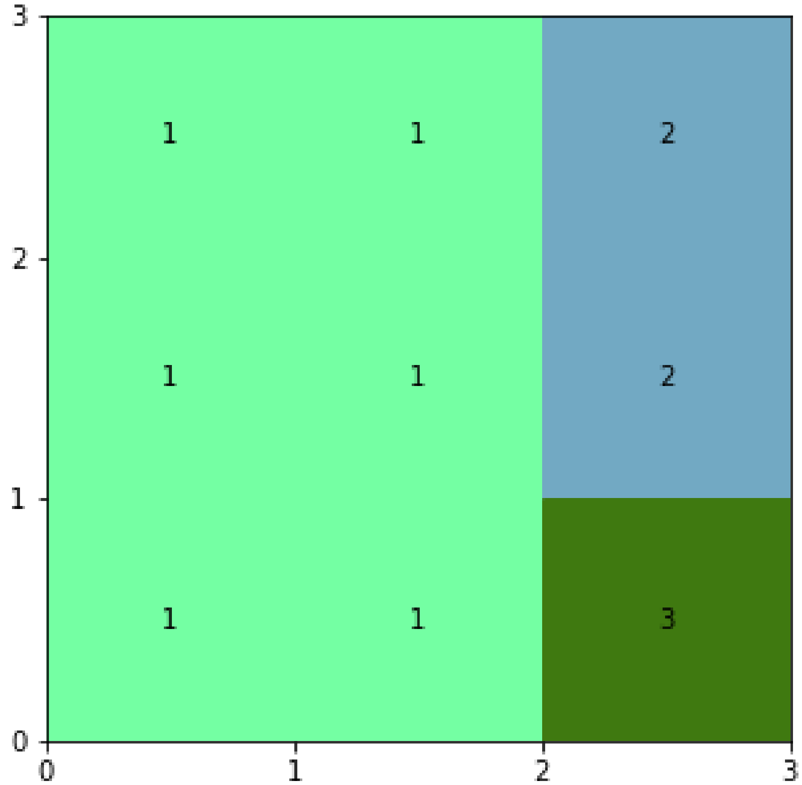

```
On average across 6400 experiments, it takes 7.0 communication towers before full coverage is ob
```



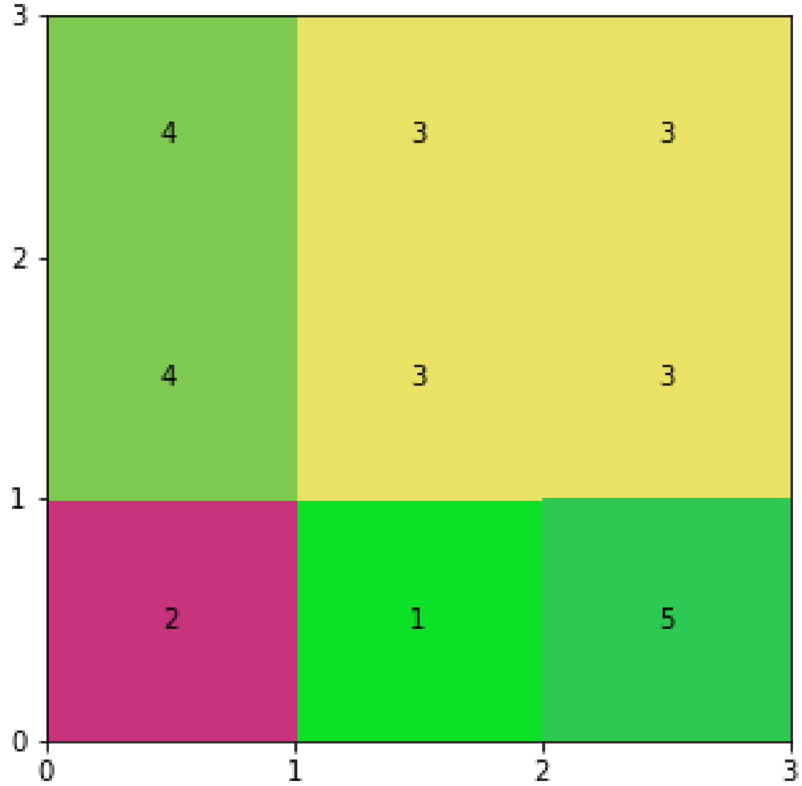
On average across 6450 experiments, it takes 4.0 communication towers before full coverage is ob



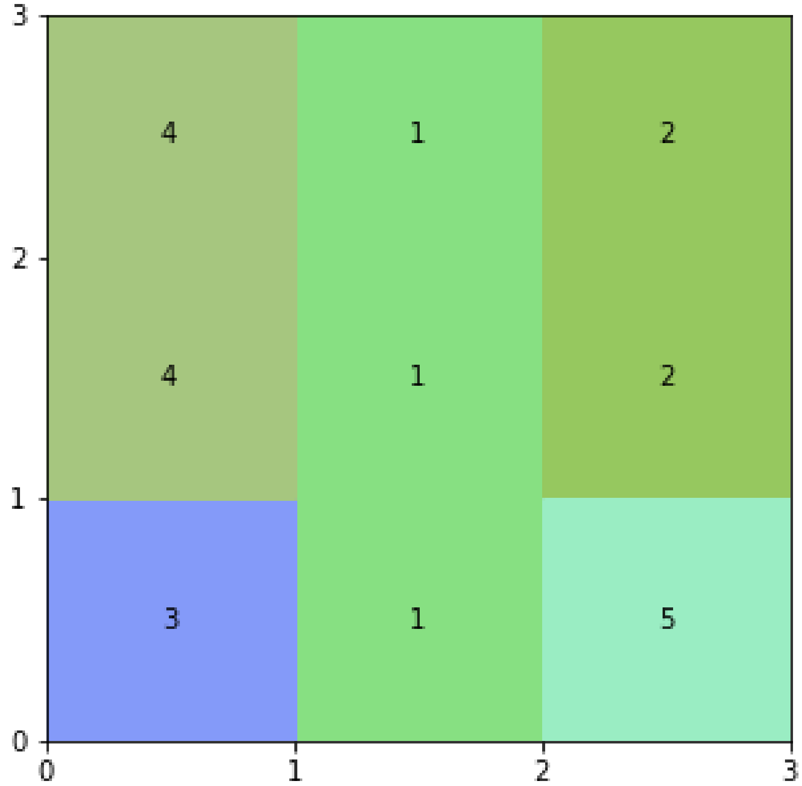
On average across 6500 experiments, it takes 3.0 communication towers before full coverage is ob



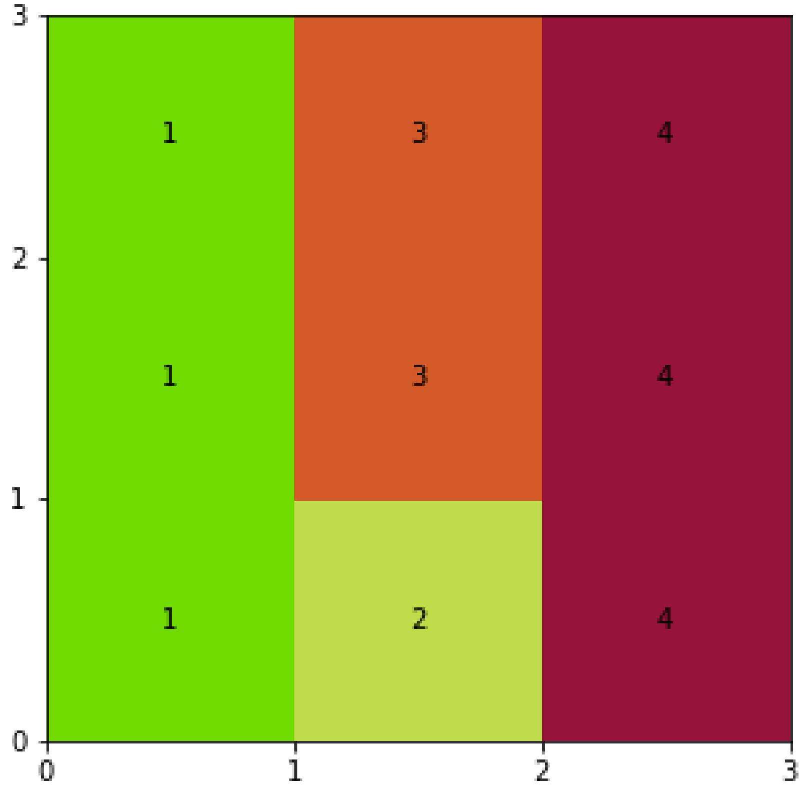
On average across 6550 experiments, it takes 5.0 communication towers before full coverage is ob



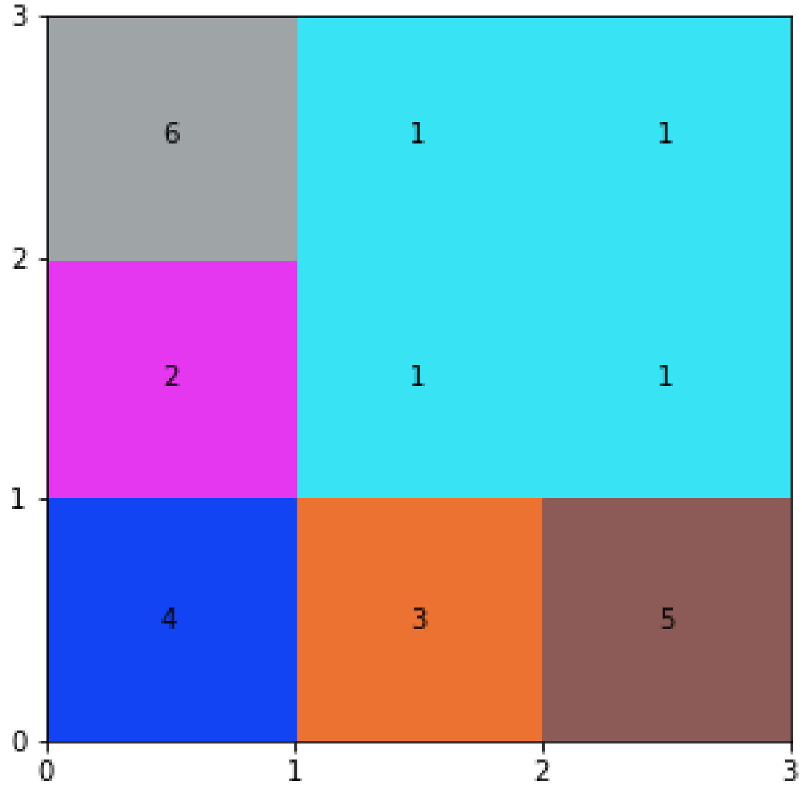

```
On average across 6600 experiments, it takes 5.0 communication towers before full coverage is ob
```



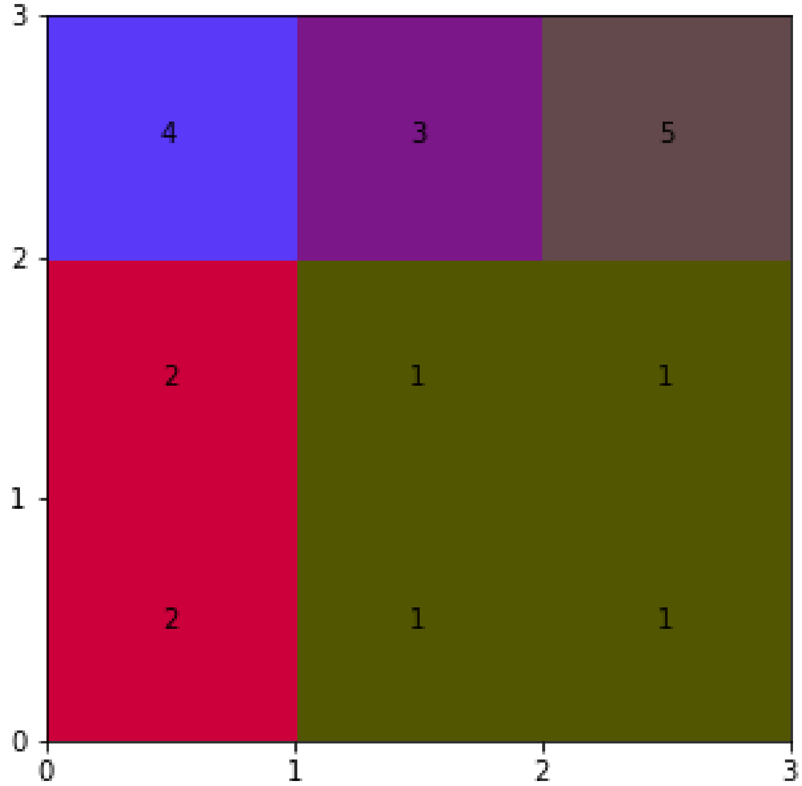
On average across 6650 experiments, it takes 4.0 communication towers before full coverage is ob



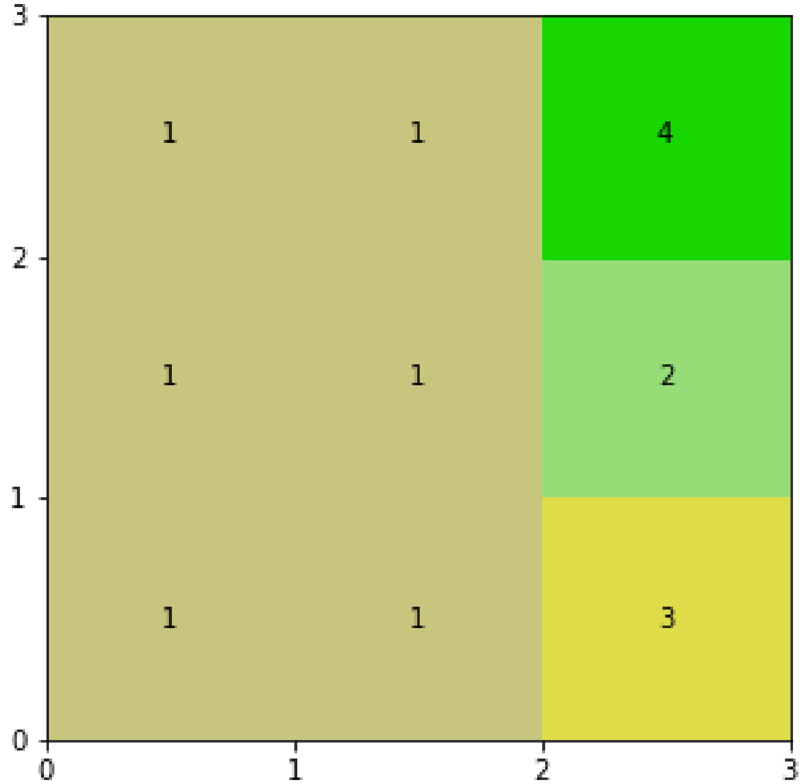
On average across 6700 experiments, it takes 6.0 communication towers before full coverage is ob



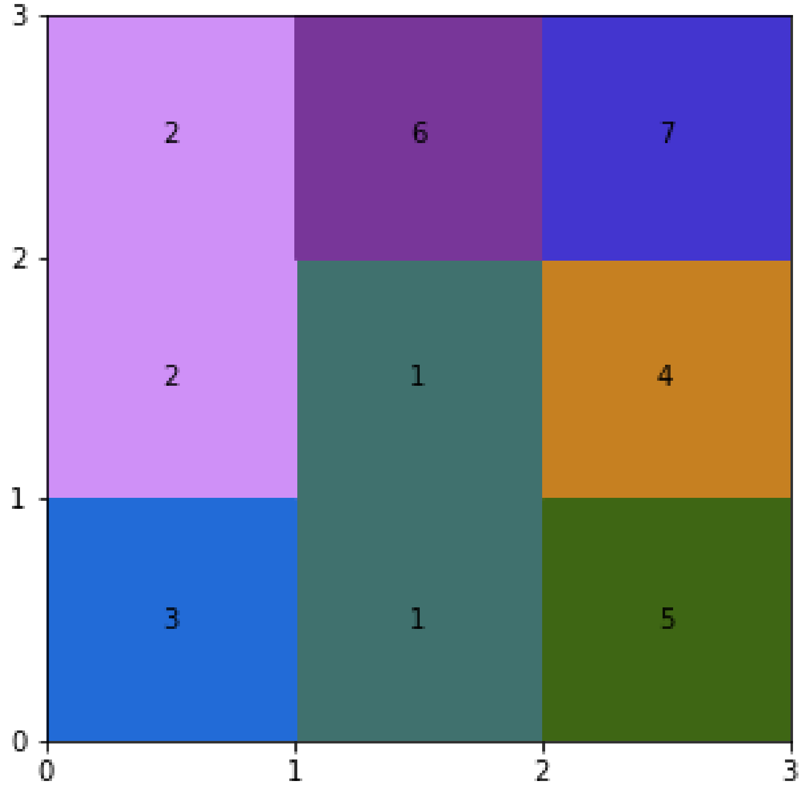

```
On average across 6750 experiments, it takes 5.0 communication towers before full coverage is ob
```



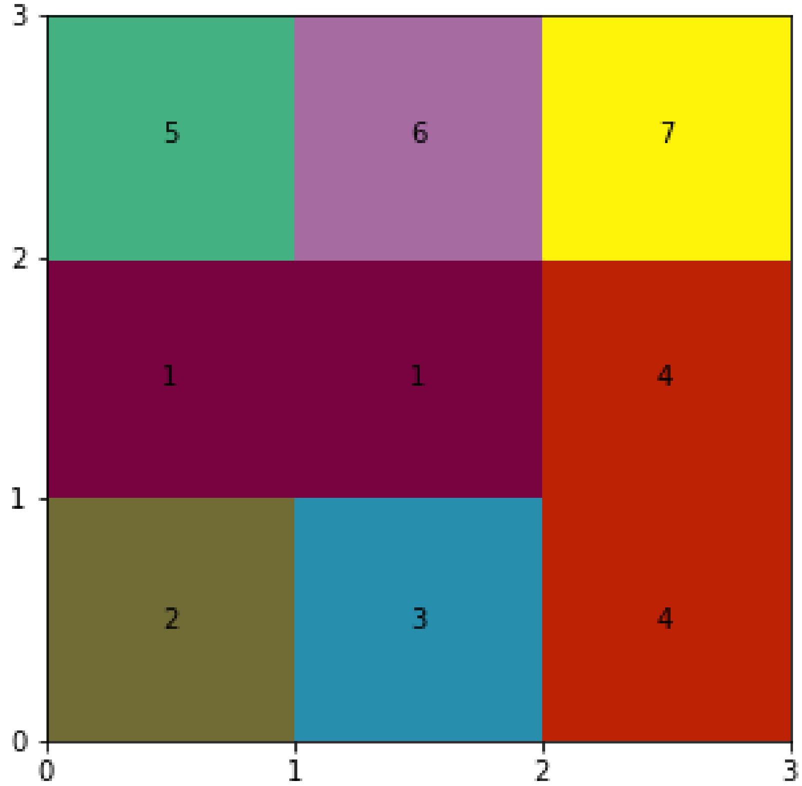
On average across 6800 experiments, it takes 4.0 communication towers before full coverage is ob



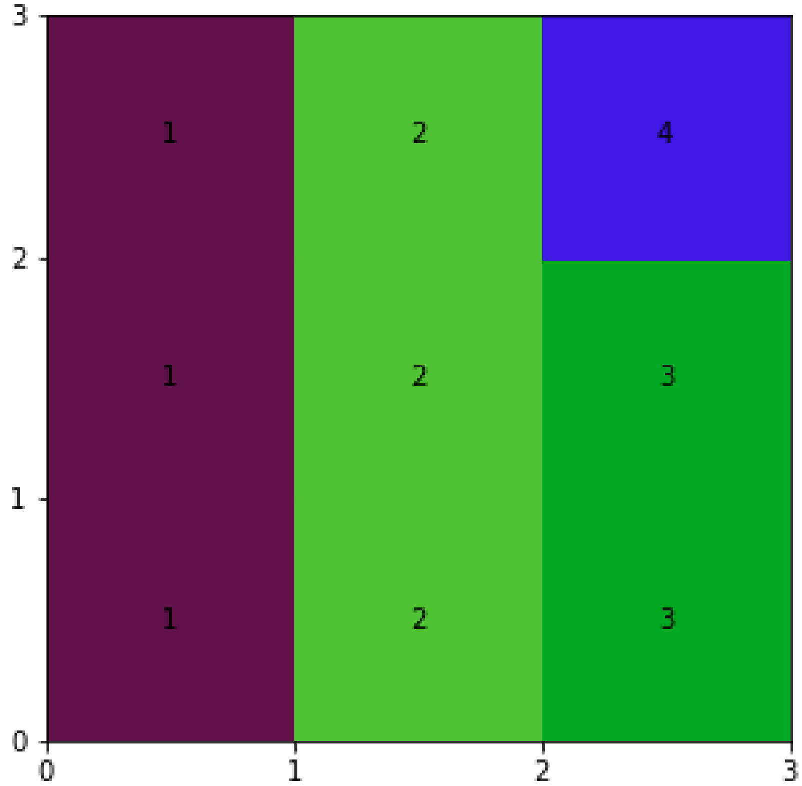
On average across 6850 experiments, it takes 7.0 communication towers before full coverage is ob



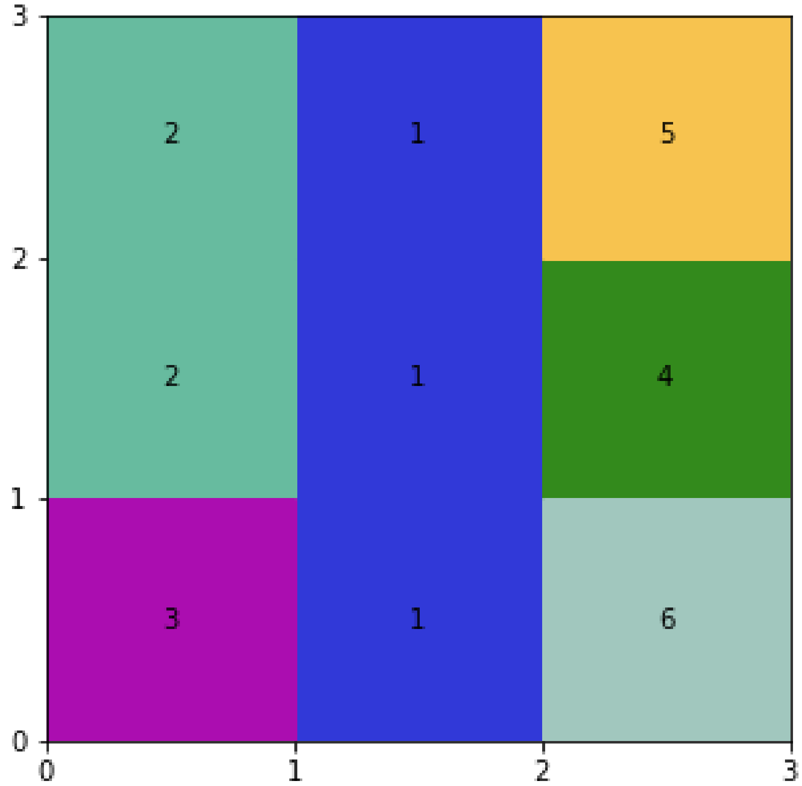
On average across 6900 experiments, it takes 7.0 communication towers before full coverage is ob



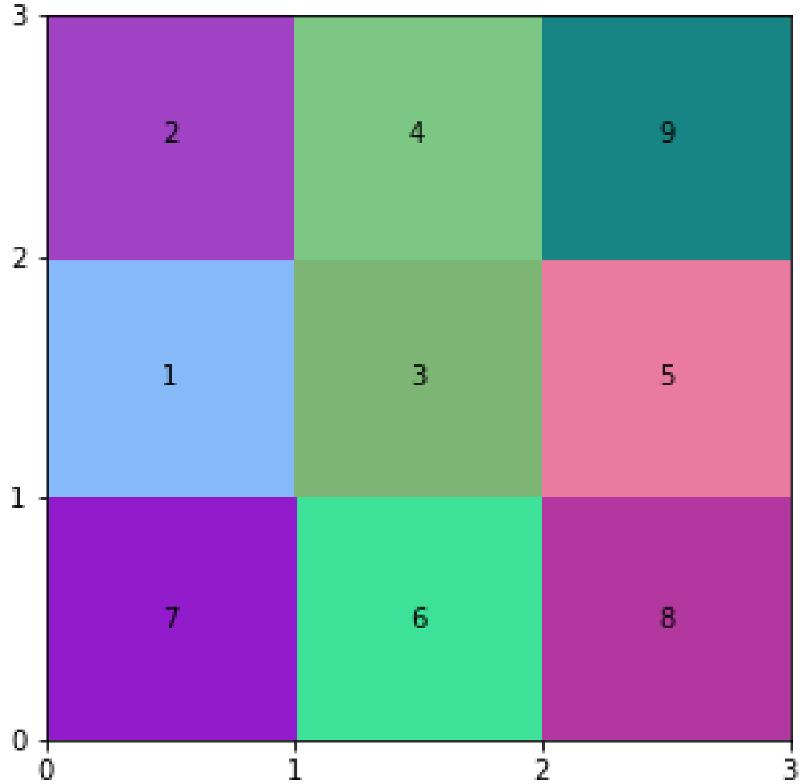
On average across 6950 experiments, it takes 4.0 communication towers before full coverage is ob



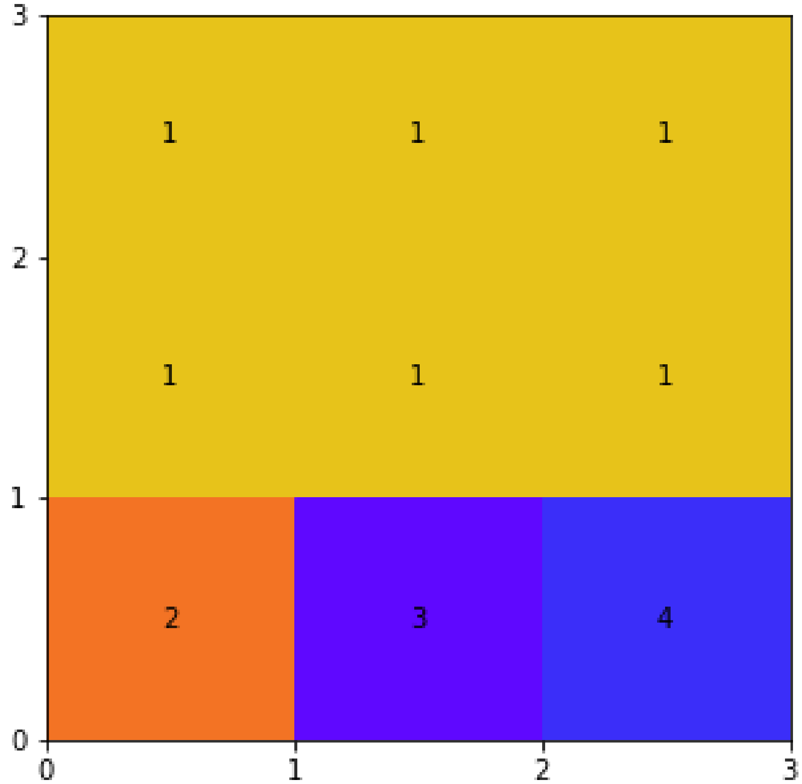
On average across 7000 experiments, it takes 6.0 communication towers before full coverage is ob



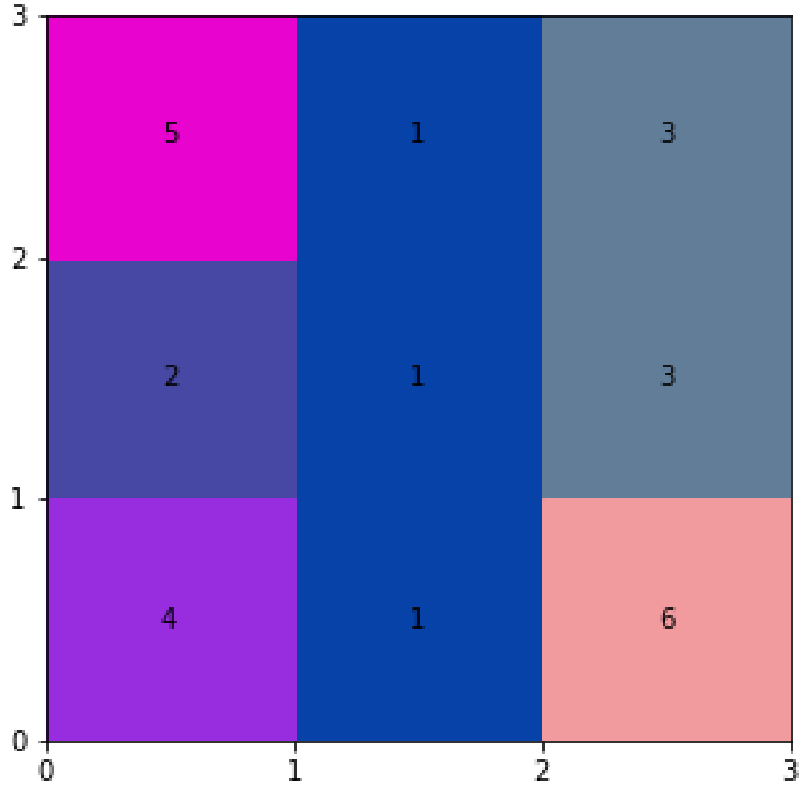

```
On average across 7050 experiments, it takes 9.0 communication towers before full coverage is ob
```



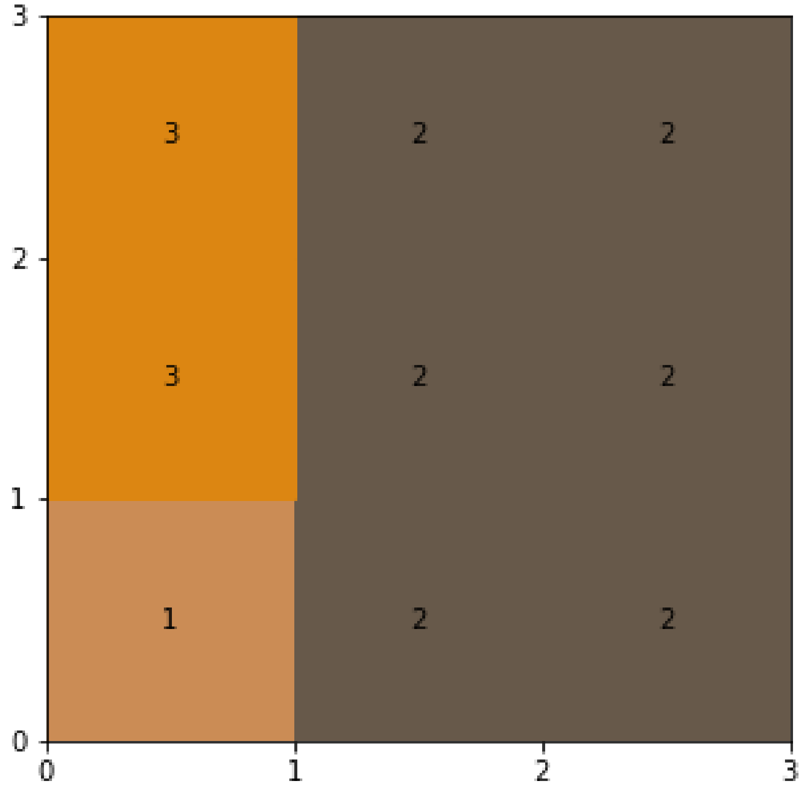

```
On average across 7100 experiments, it takes 4.0 communication towers before full coverage is ob
```



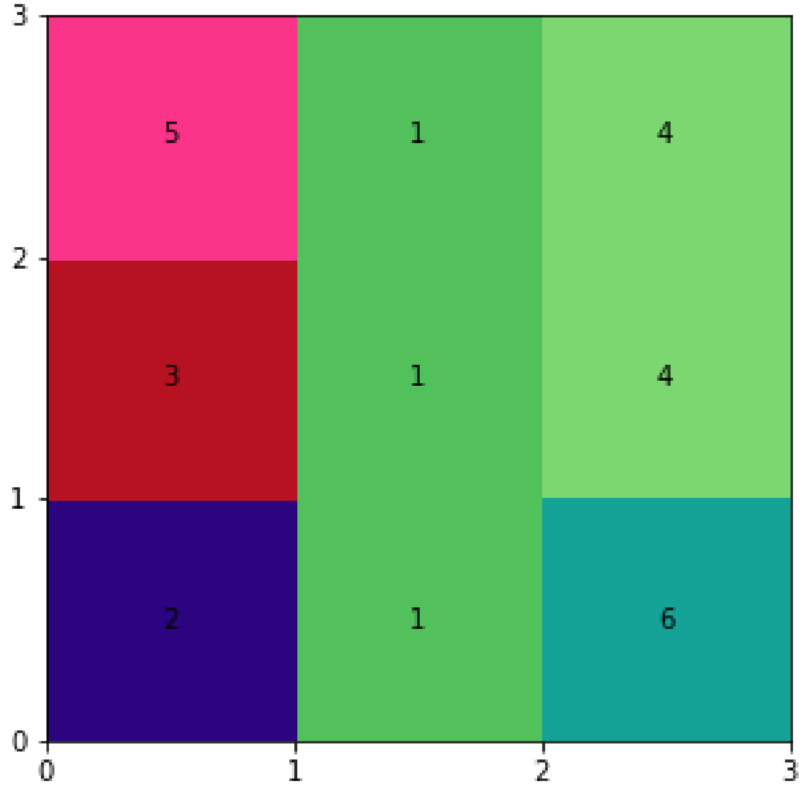

```
On average across 7150 experiments, it takes 6.0 communication towers before full coverage is ob
```



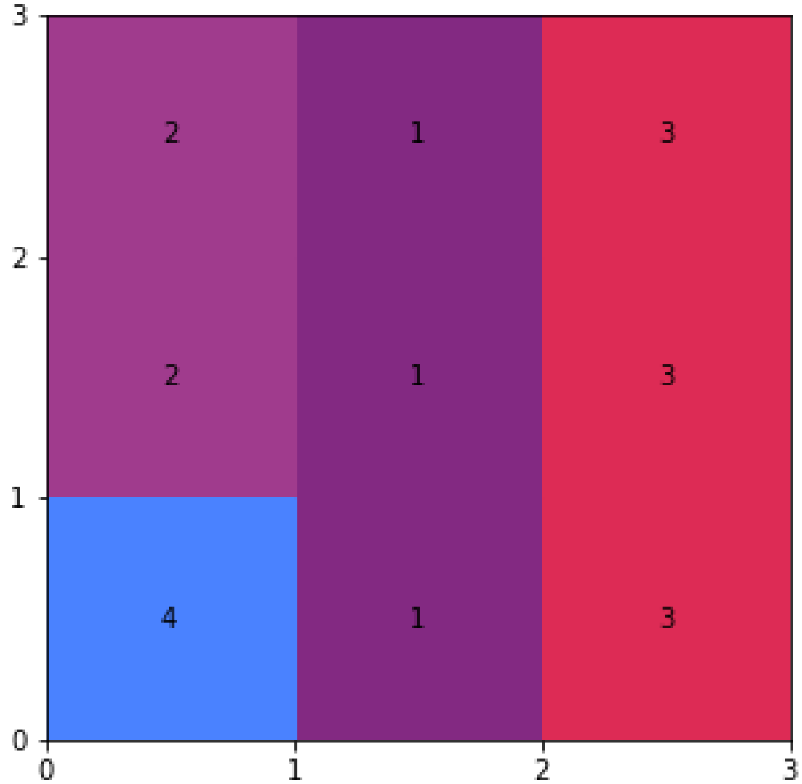

```
On average across 7200 experiments, it takes 3.0 communication towers before full coverage is ob
```



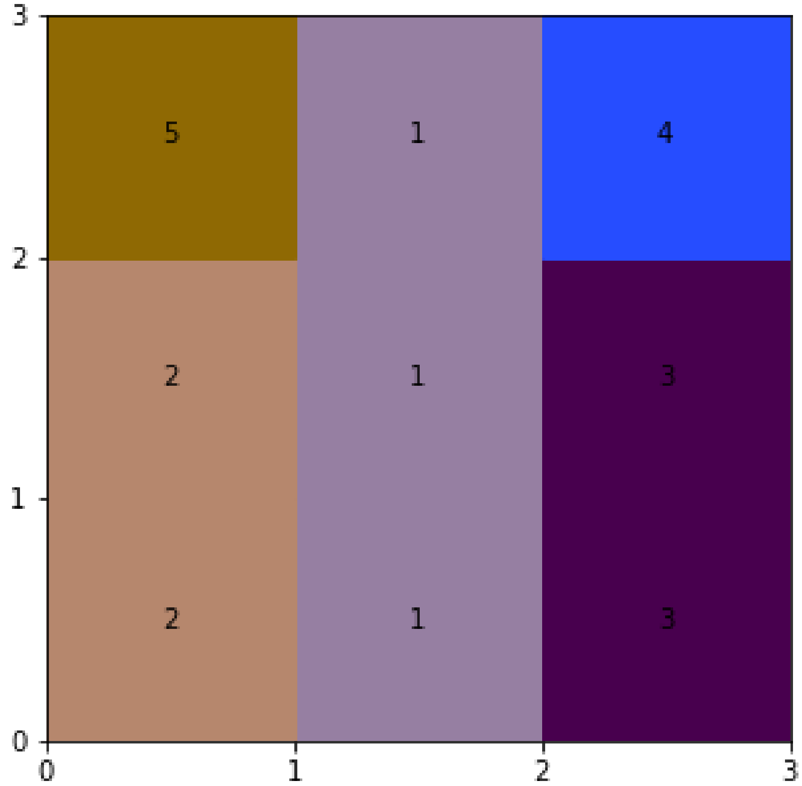

```
On average across 7250 experiments, it takes 6.0 communication towers before full coverage is ob
```



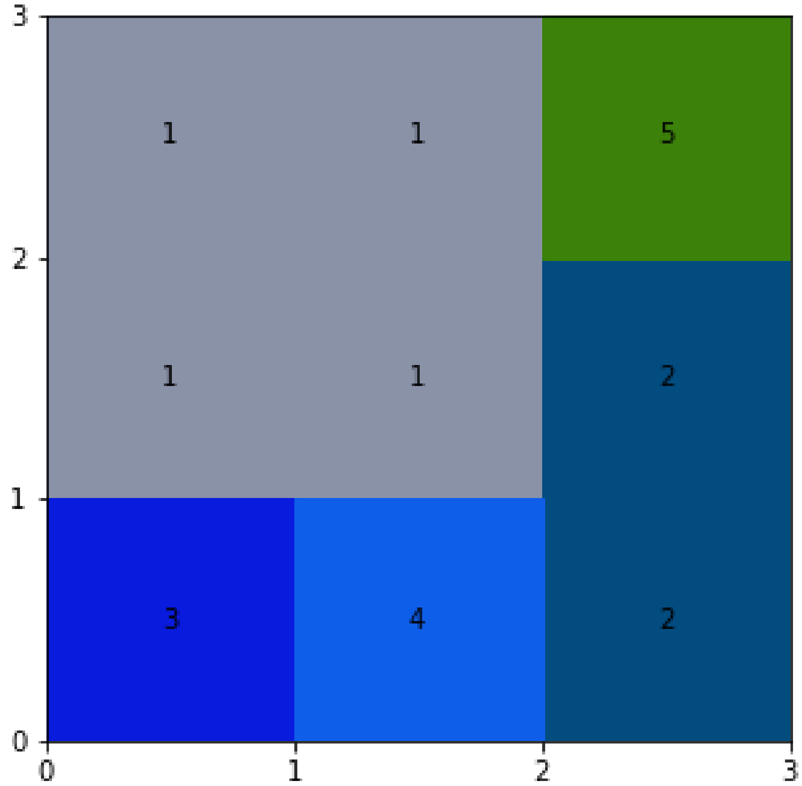
On average across 7300 experiments, it takes 4.0 communication towers before full coverage is ob



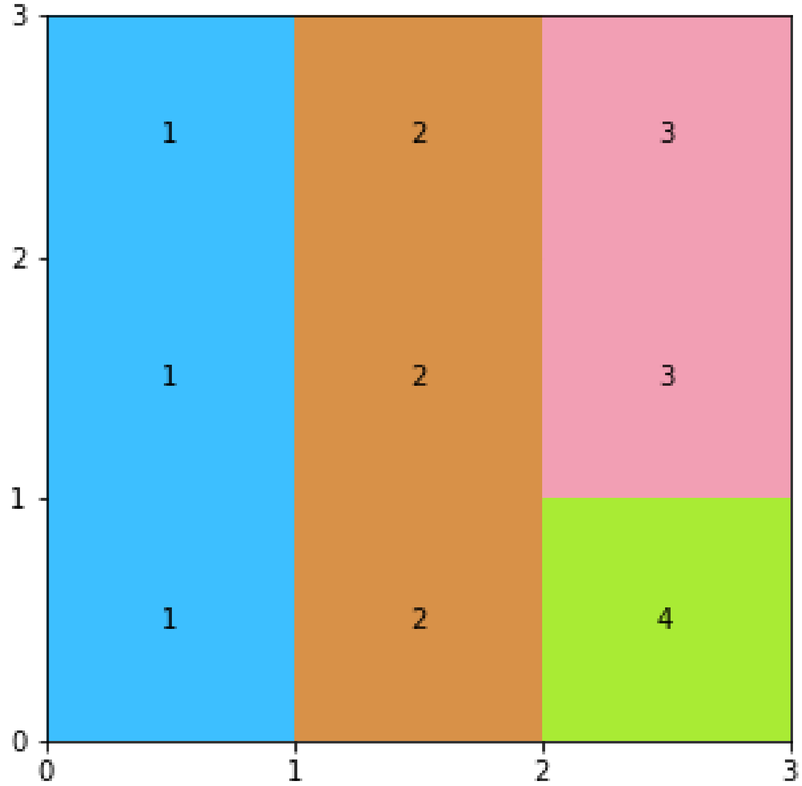
On average across 7350 experiments, it takes 5.0 communication towers before full coverage is ob



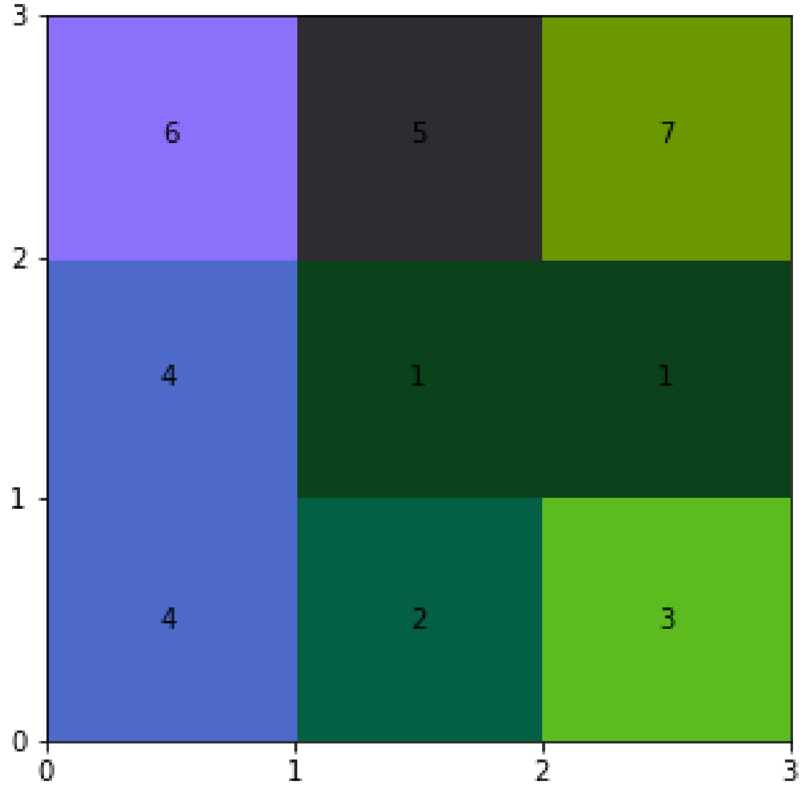
On average across 7400 experiments, it takes 5.0 communication towers before full coverage is ob



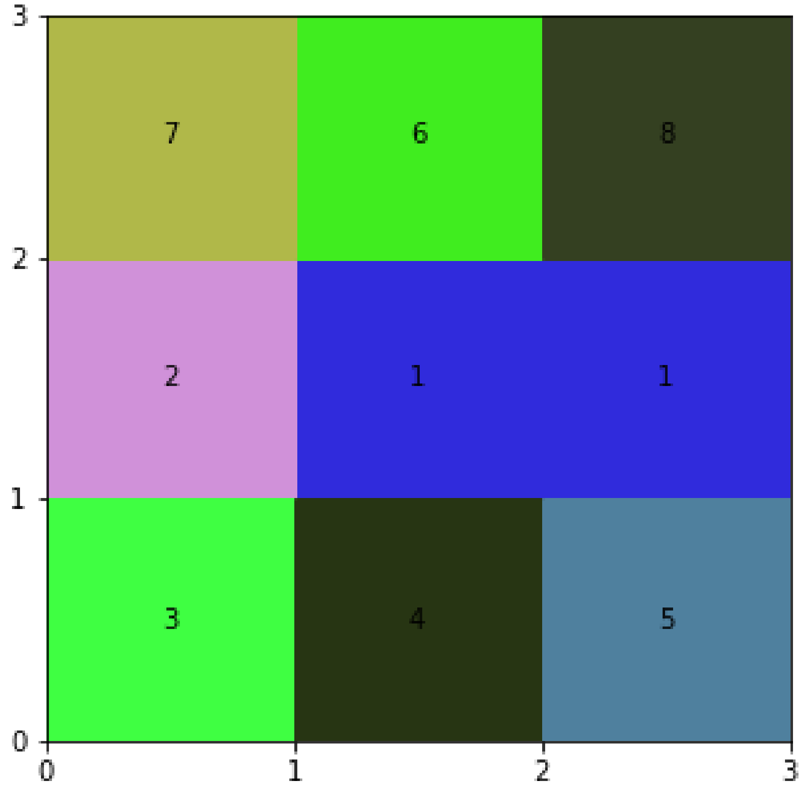

```
On average across 7450 experiments, it takes 4.0 communication towers before full coverage is ob
```



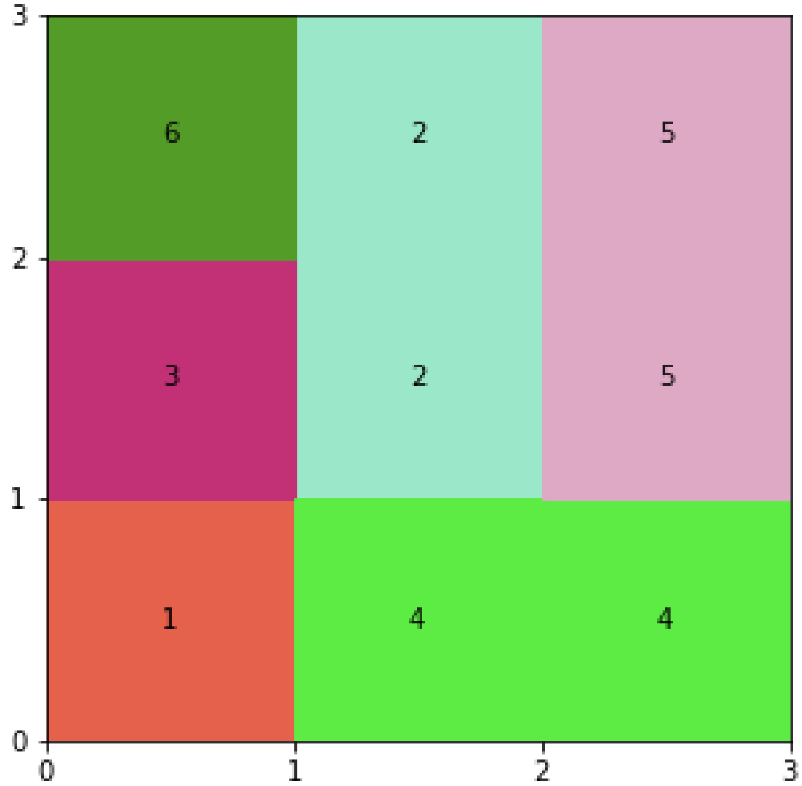

```
On average across 7500 experiments, it takes 7.0 communication towers before full coverage is ob
```



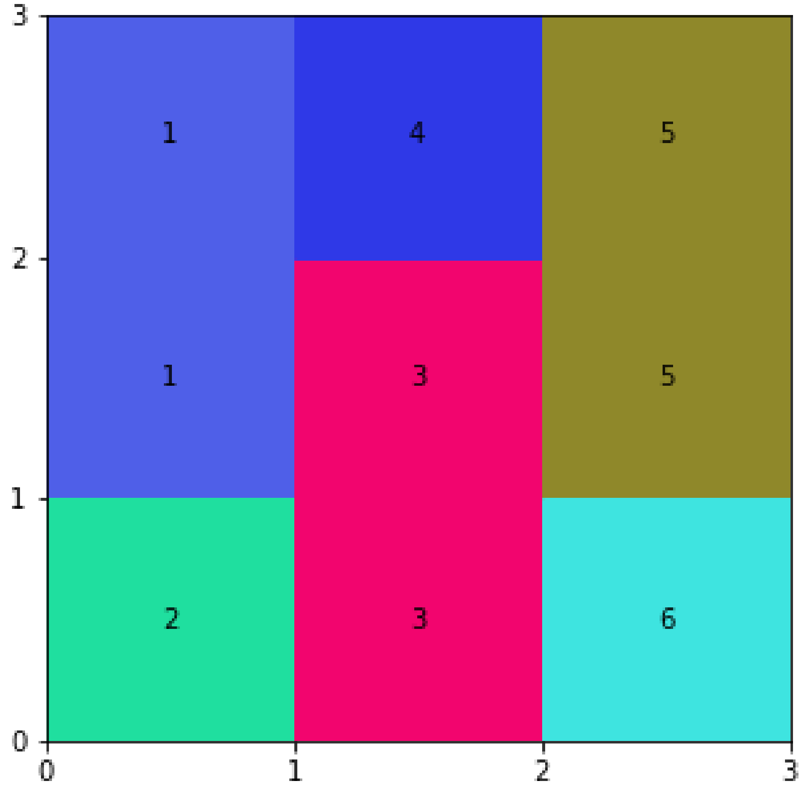
On average across 7550 experiments, it takes 8.0 communication towers before full coverage is ob



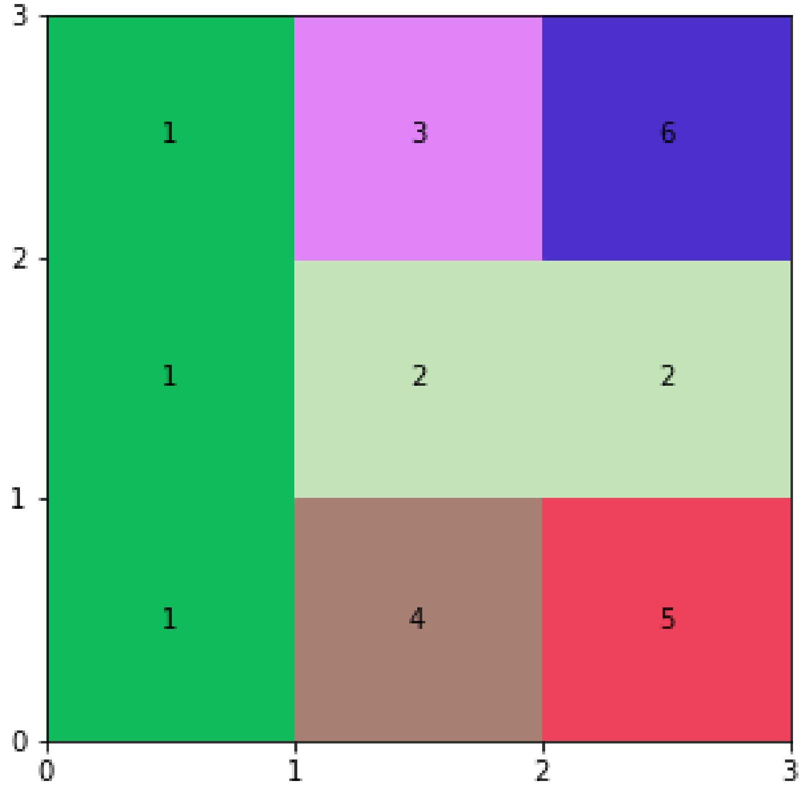
On average across 7600 experiments, it takes 6.0 communication towers before full coverage is ob



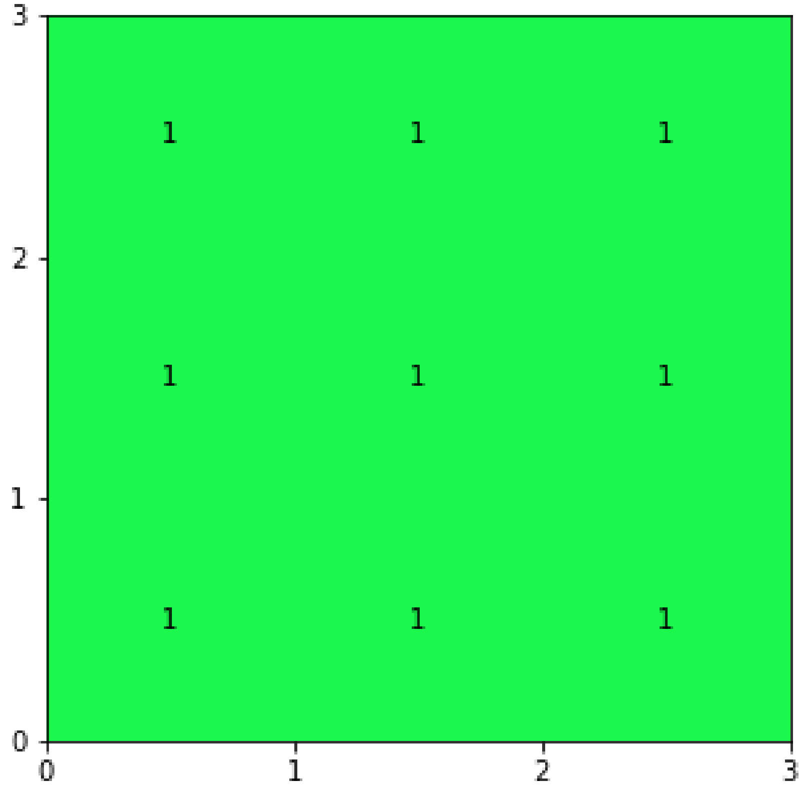
On average across 7650 experiments, it takes 6.0 communication towers before full coverage is ob



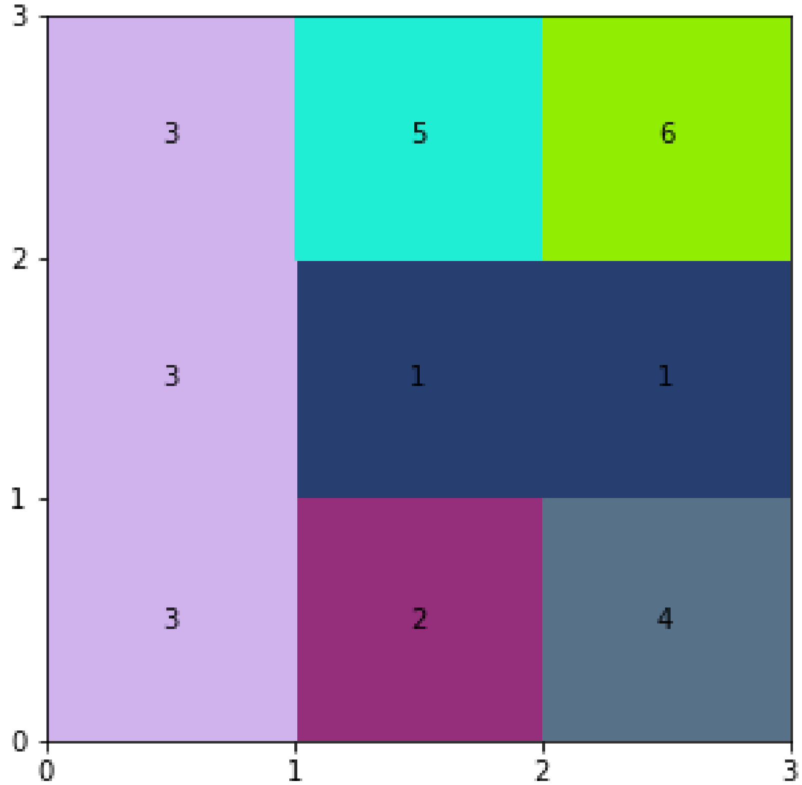
On average across 7700 experiments, it takes 6.0 communication towers before full coverage is ob



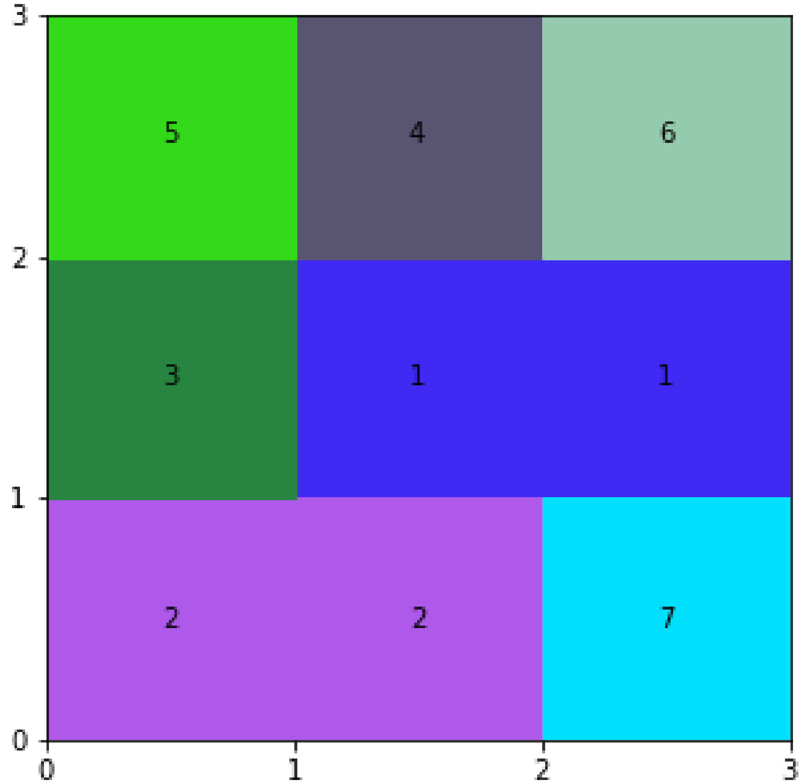
On average across 7750 experiments, it takes 1.0 communication towers before full coverage is ob



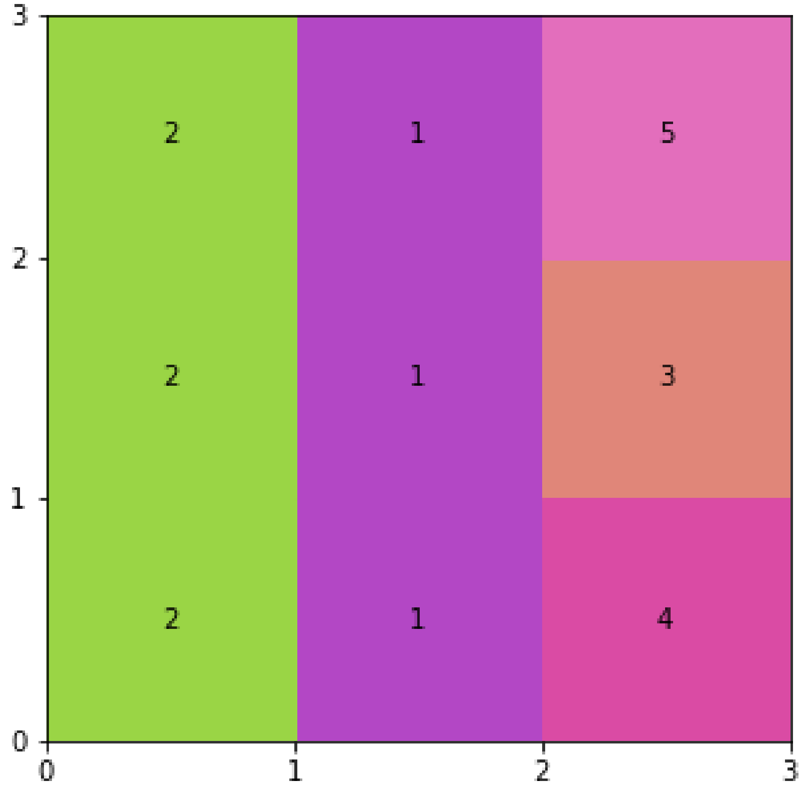
On average across 7800 experiments, it takes 6.0 communication towers before full coverage is ob



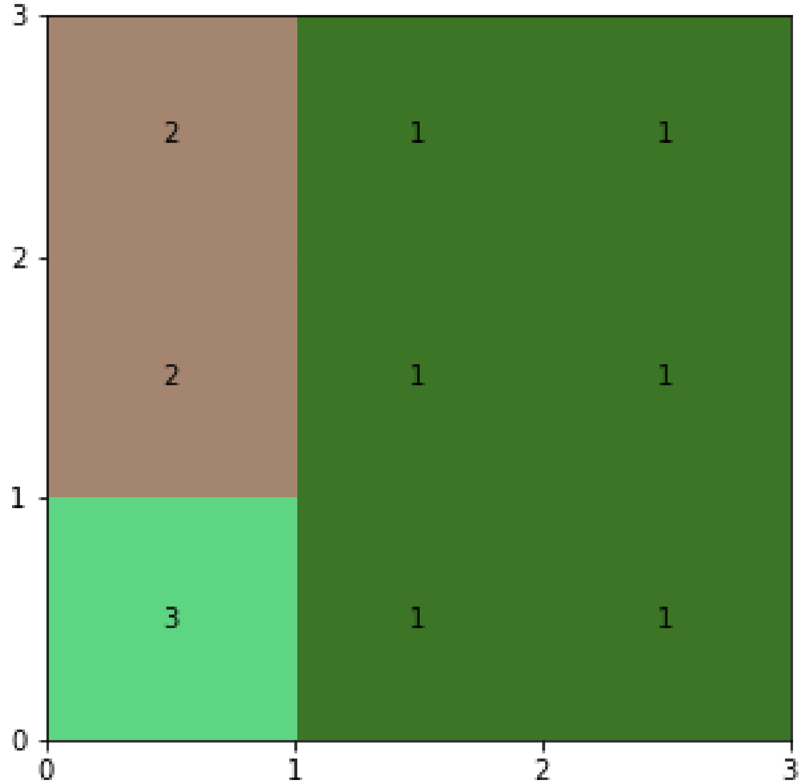
On average across 7850 experiments, it takes 7.0 communication towers before full coverage is ob



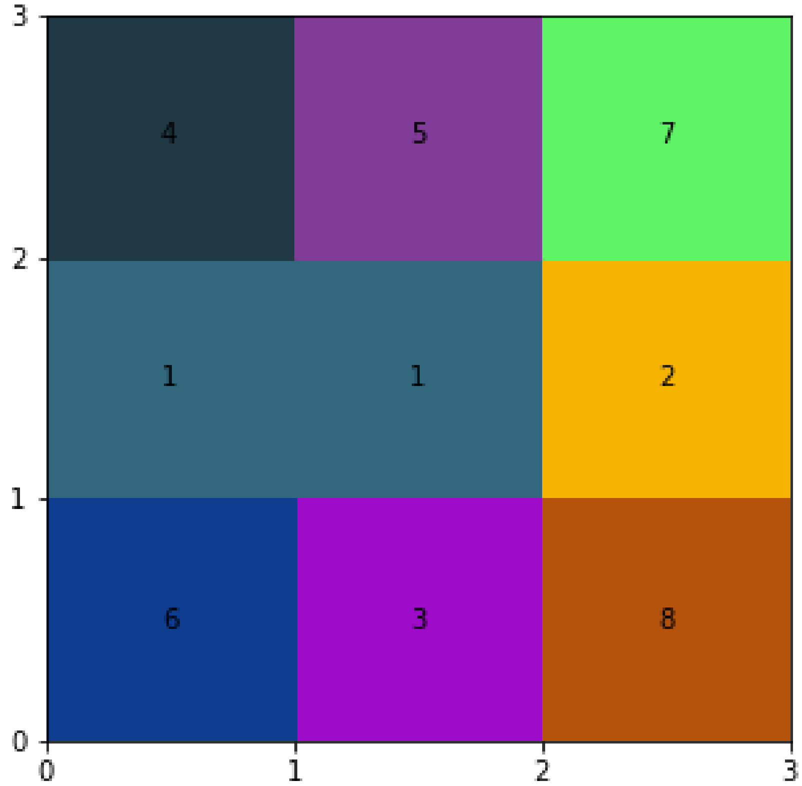
On average across 7900 experiments, it takes 5.0 communication towers before full coverage is ob



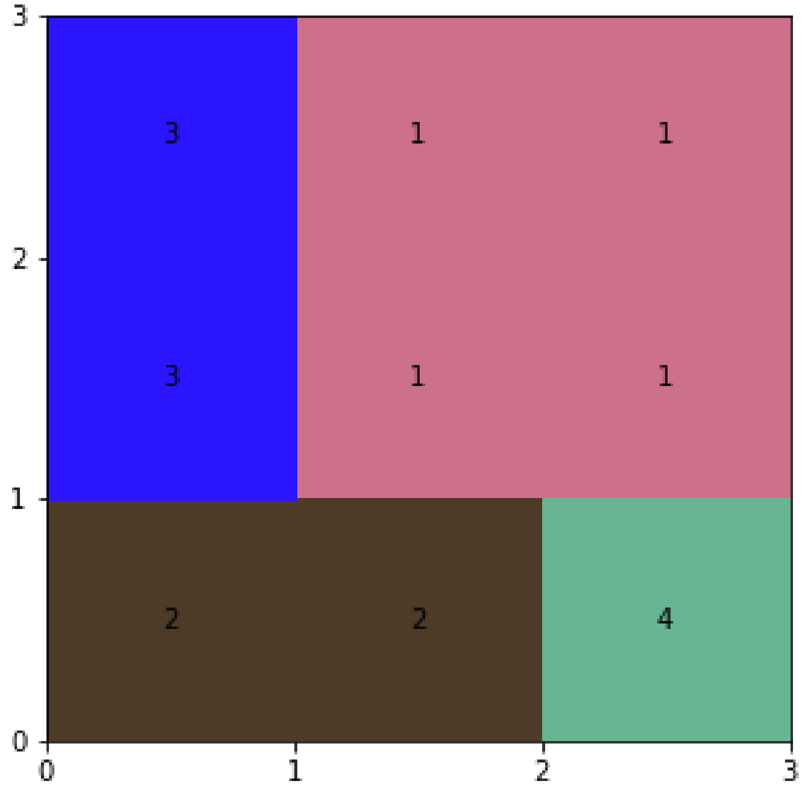

```
On average across 7950 experiments, it takes 3.0 communication towers before full coverage is ob
```



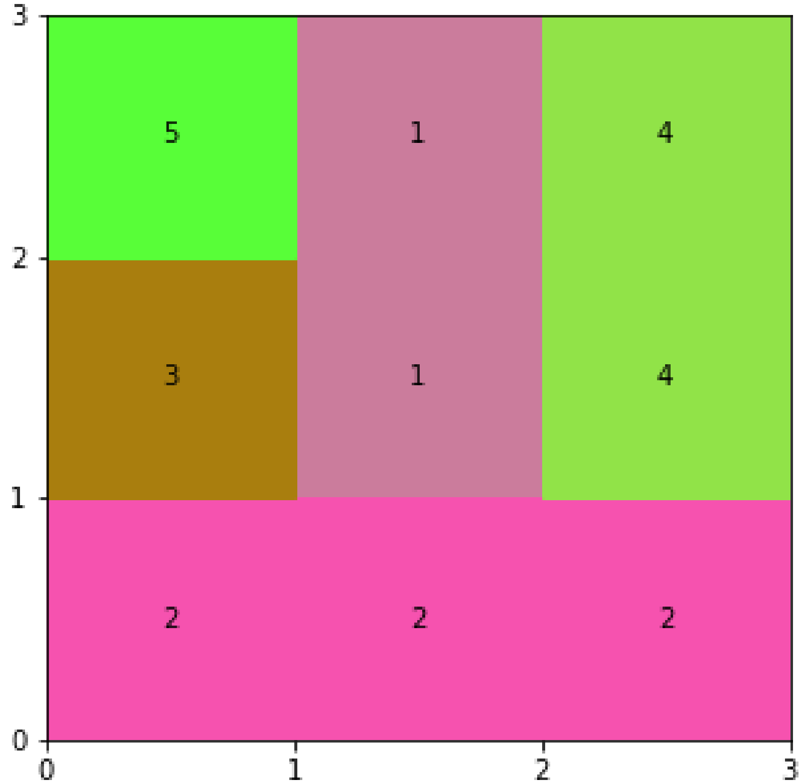

```
On average across 8000 experiments, it takes 8.0 communication towers before full coverage is ob
```



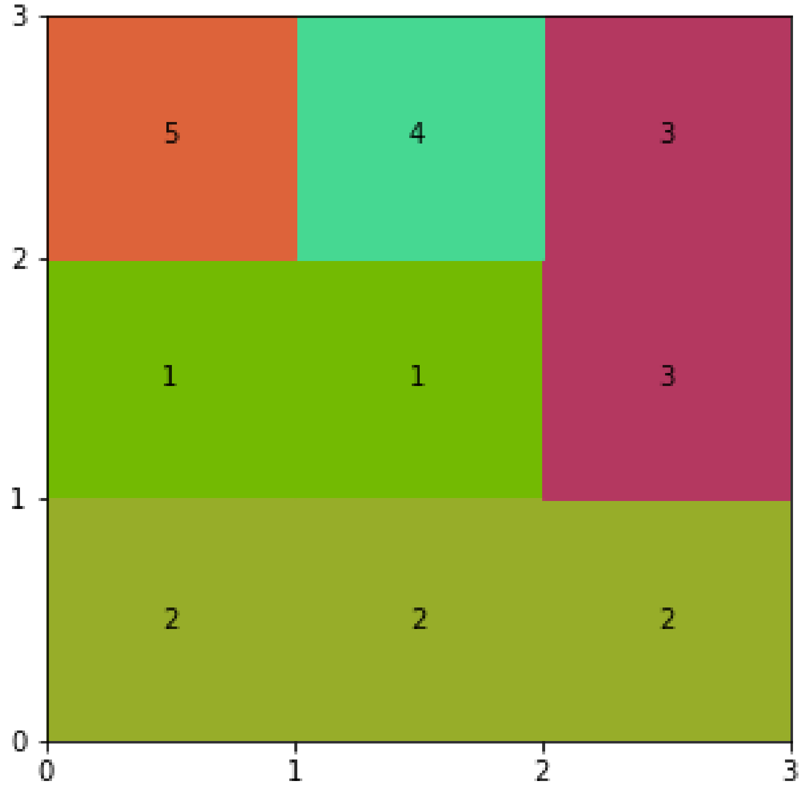
On average across 8050 experiments, it takes 4.0 communication towers before full coverage is ob



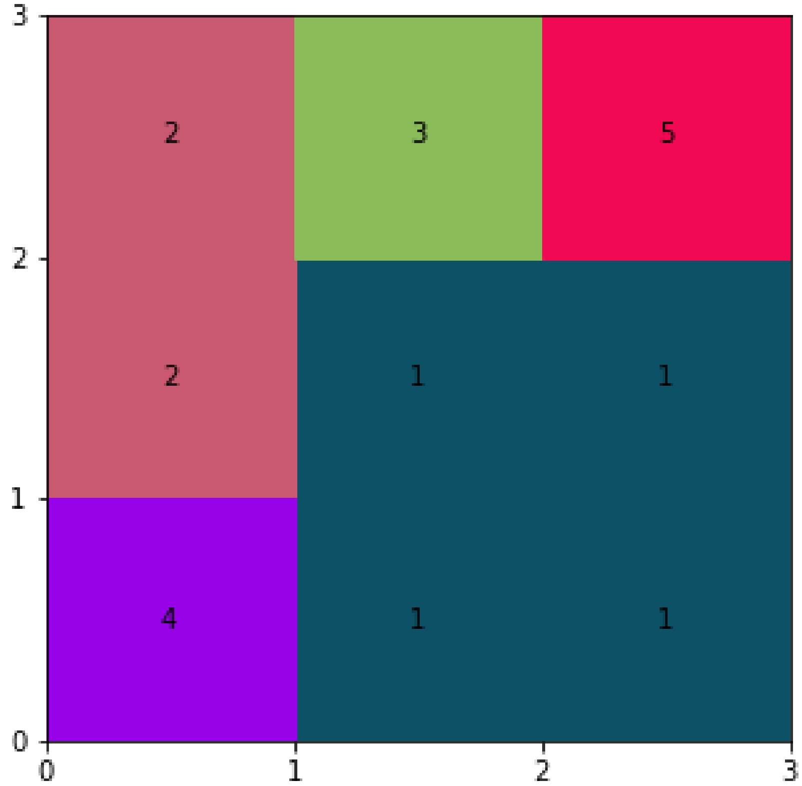
On average across 8100 experiments, it takes 5.0 communication towers before full coverage is ob



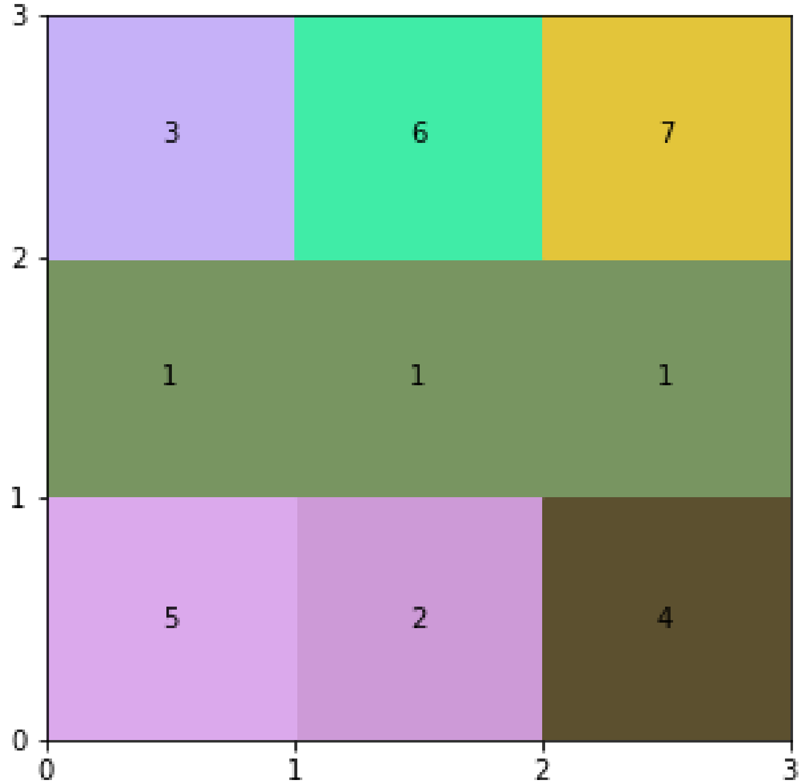
On average across 8150 experiments, it takes 5.0 communication towers before full coverage is ob



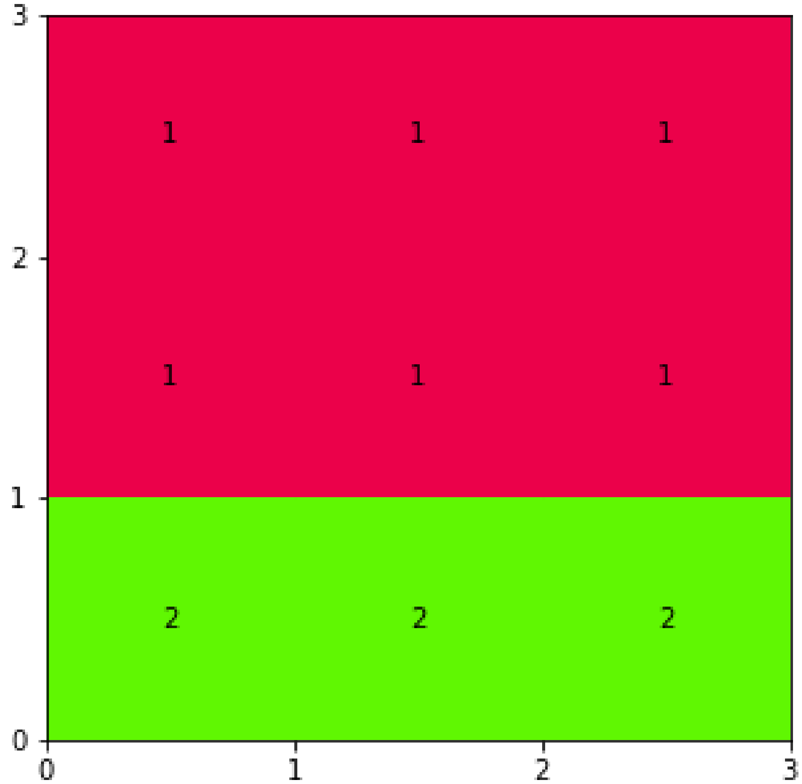
On average across 8200 experiments, it takes 5.0 communication towers before full coverage is ob



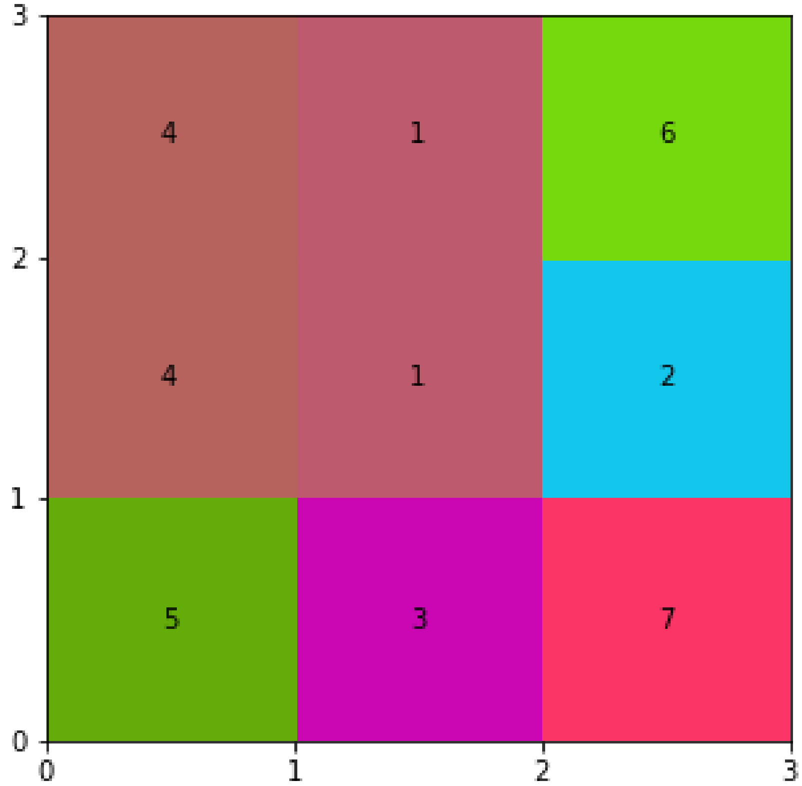
On average across 8250 experiments, it takes 7.0 communication towers before full coverage is ob



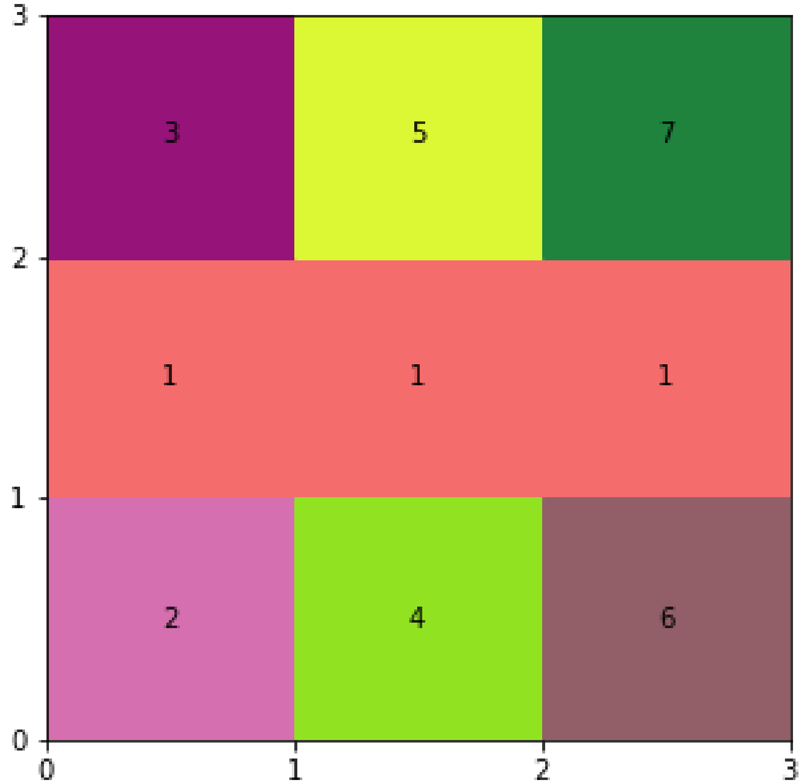
On average across 8300 experiments, it takes 2.0 communication towers before full coverage is ob



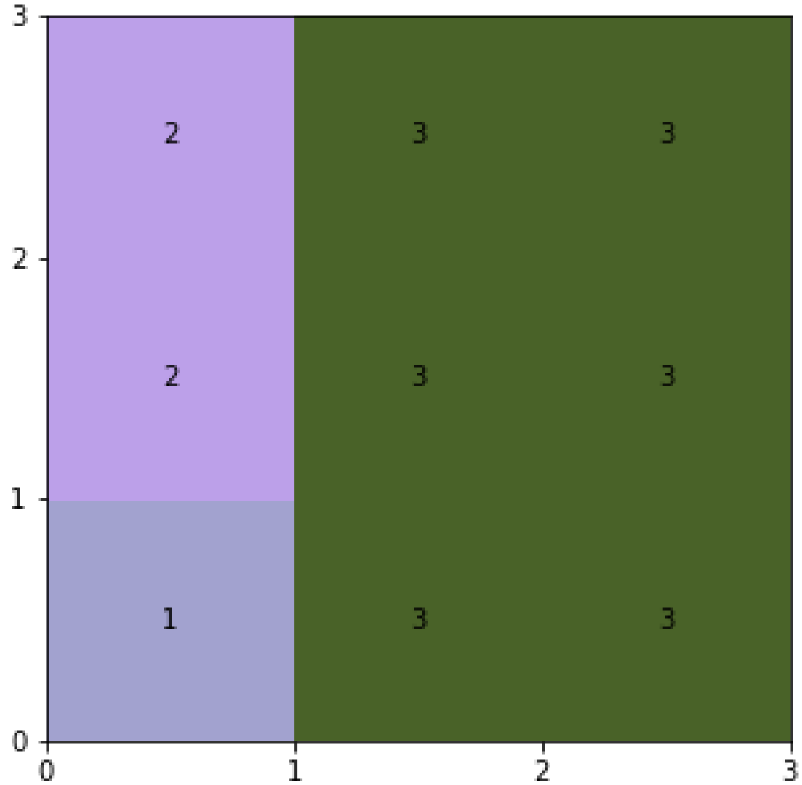

```
On average across 8350 experiments, it takes 7.0 communication towers before full coverage is ob
```



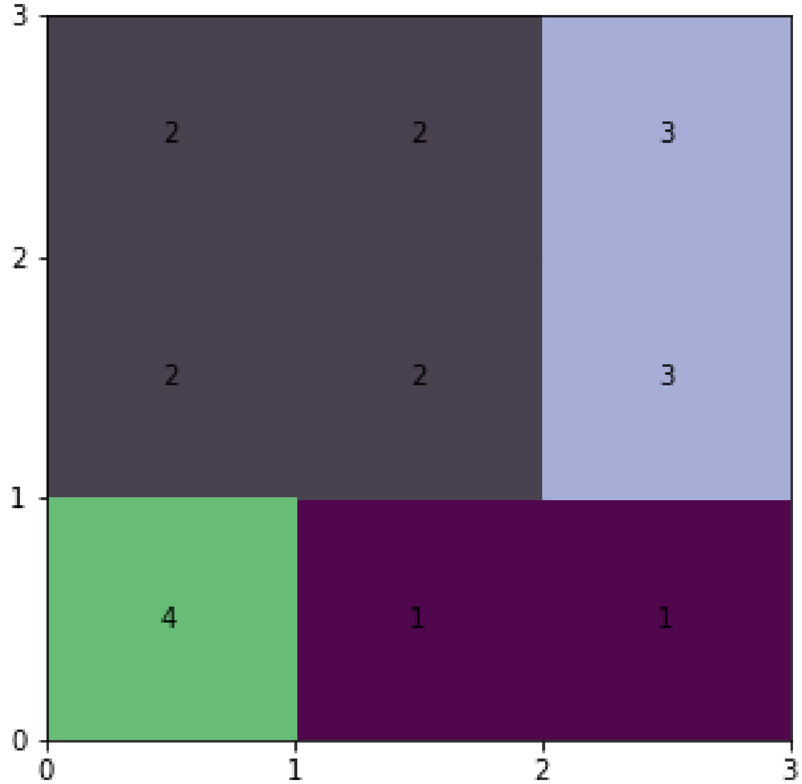
On average across 8400 experiments, it takes 7.0 communication towers before full coverage is ob



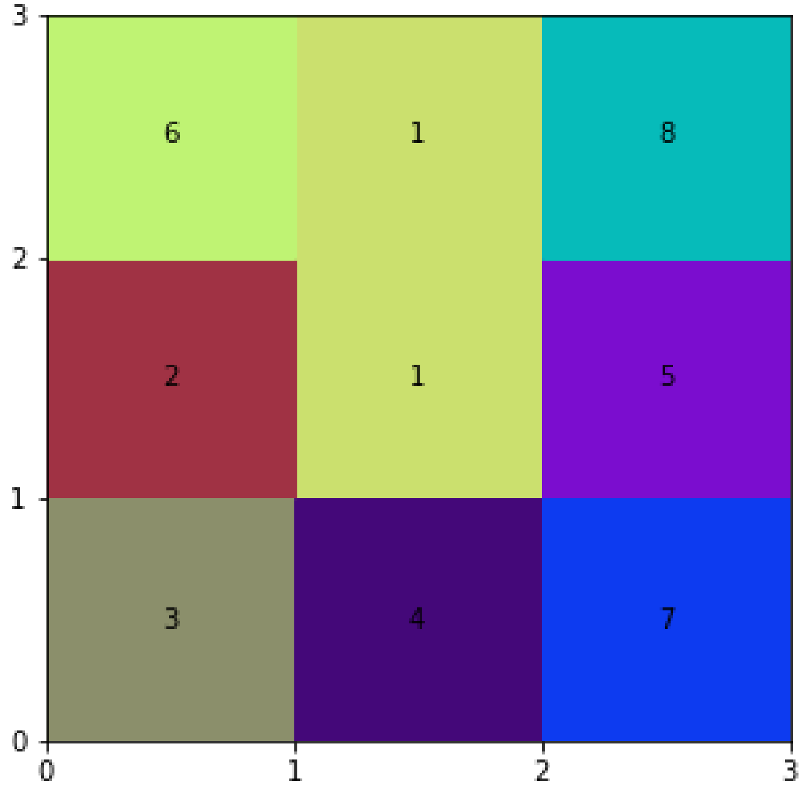
On average across 8450 experiments, it takes 3.0 communication towers before full coverage is ob



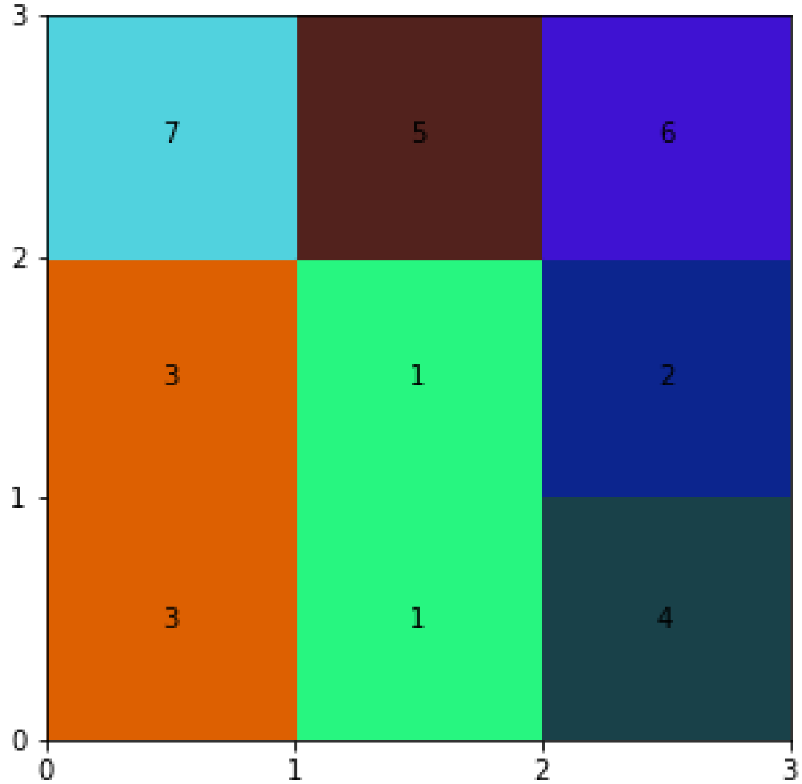
On average across 8500 experiments, it takes 4.0 communication towers before full coverage is ob



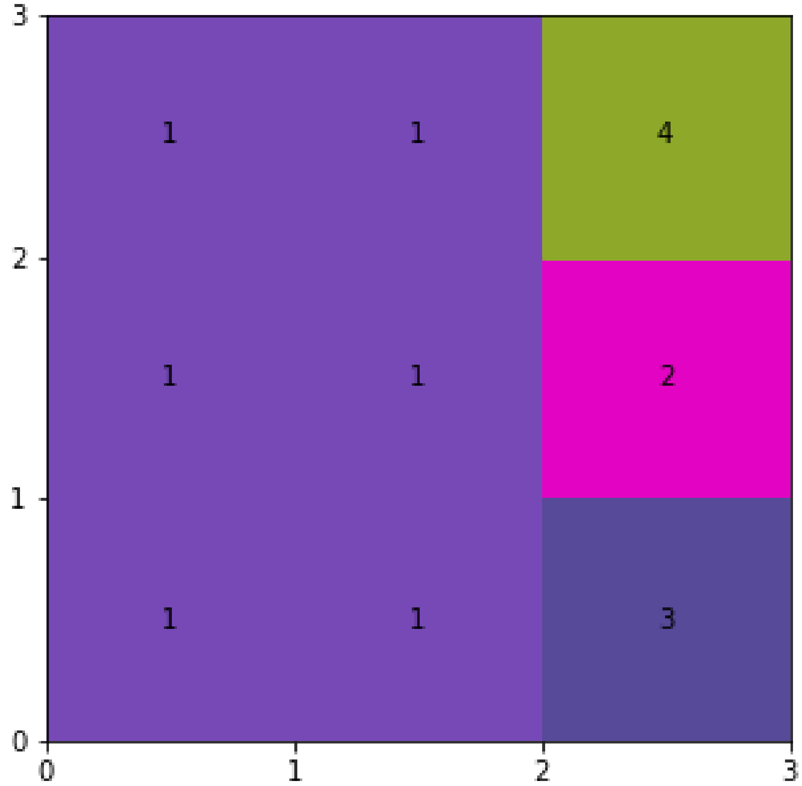
On average across 8550 experiments, it takes 8.0 communication towers before full coverage is ob



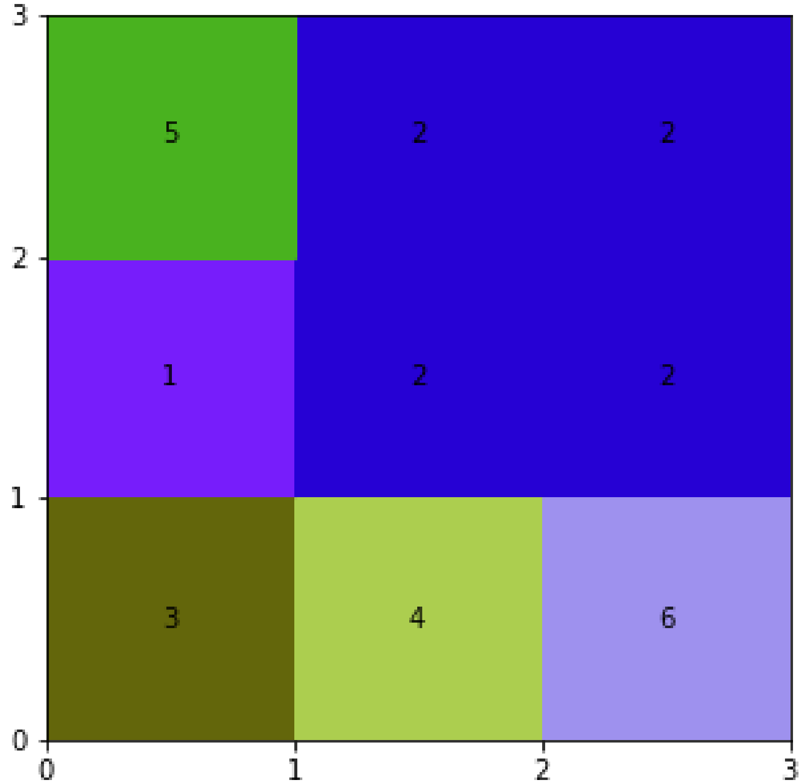
On average across 8600 experiments, it takes 7.0 communication towers before full coverage is ob



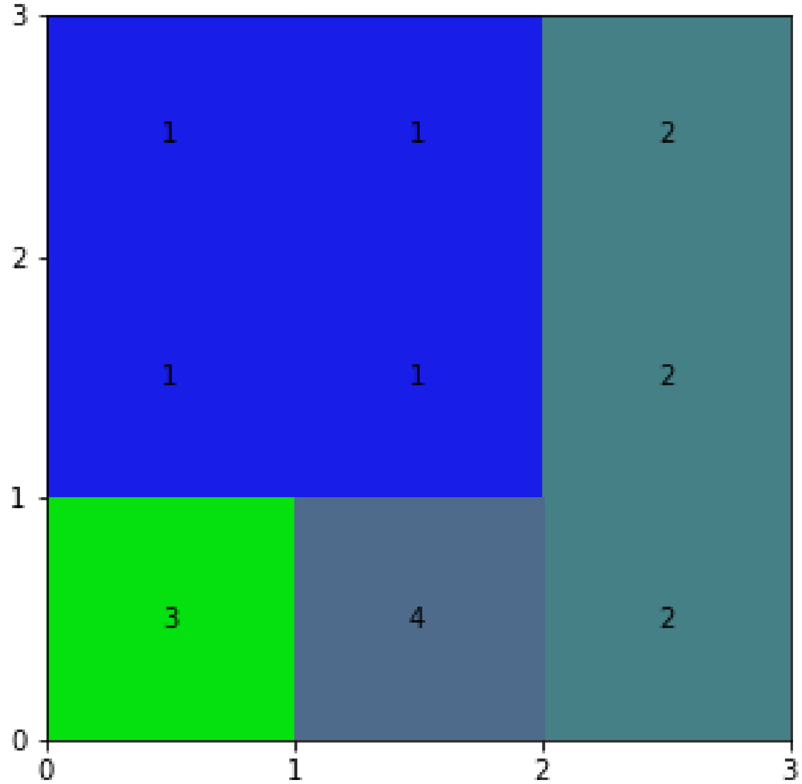
On average across 8650 experiments, it takes 4.0 communication towers before full coverage is ob



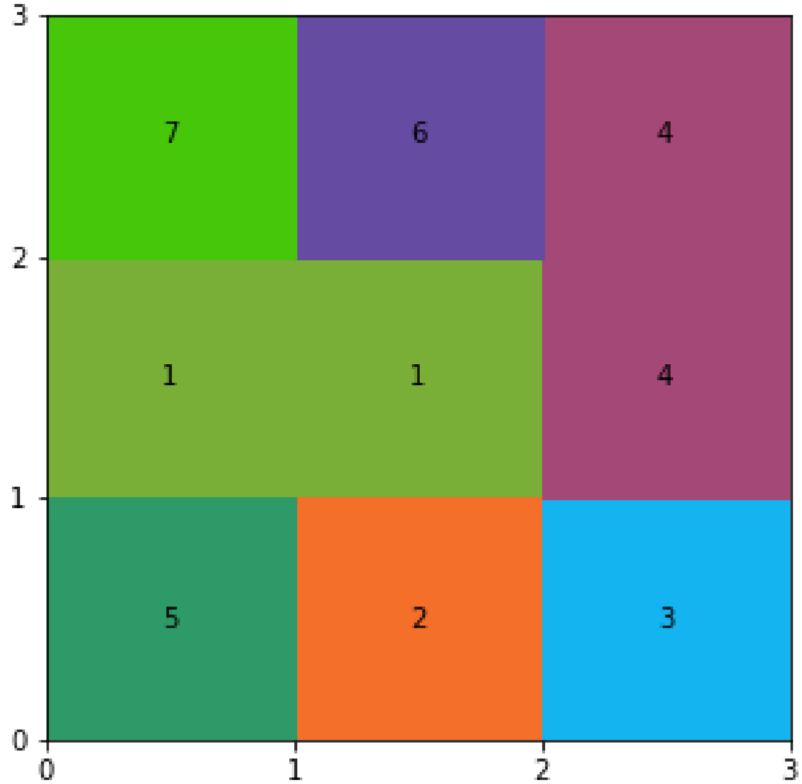
On average across 8700 experiments, it takes 6.0 communication towers before full coverage is ob



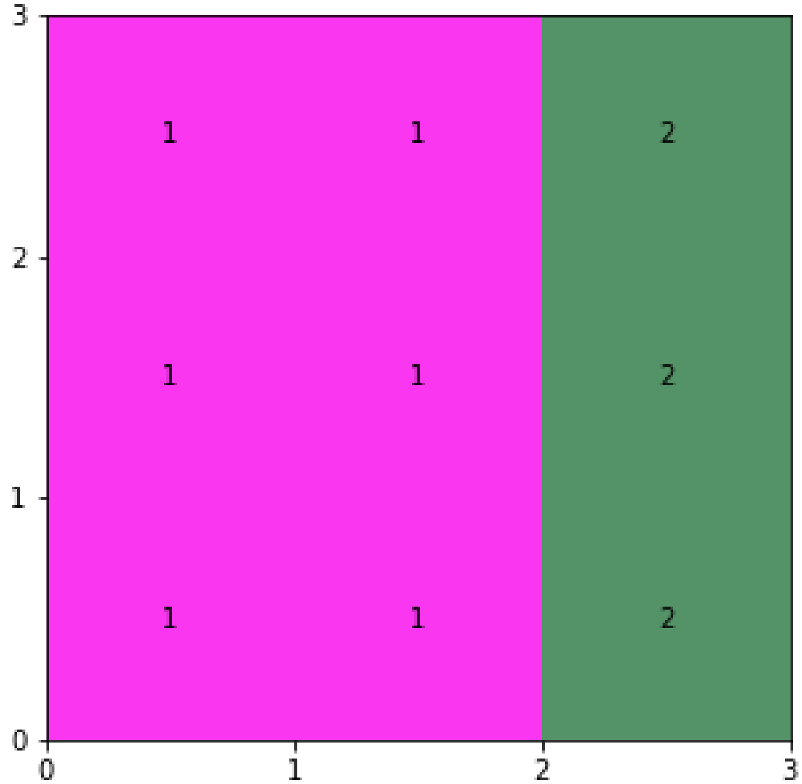

```
On average across 8750 experiments, it takes 4.0 communication towers before full coverage is ob
```



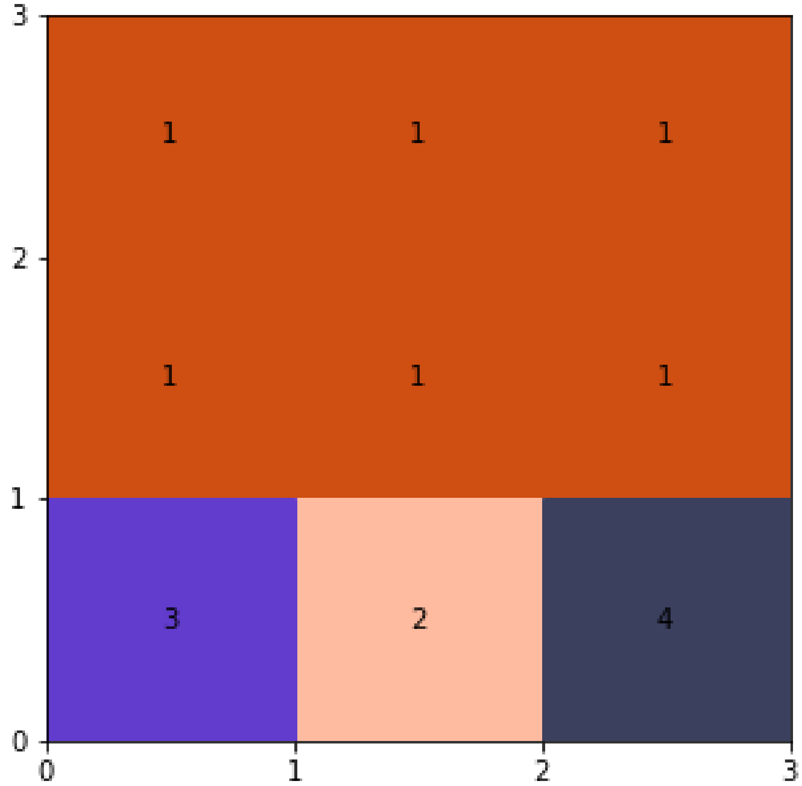
On average across 8800 experiments, it takes 7.0 communication towers before full coverage is ob



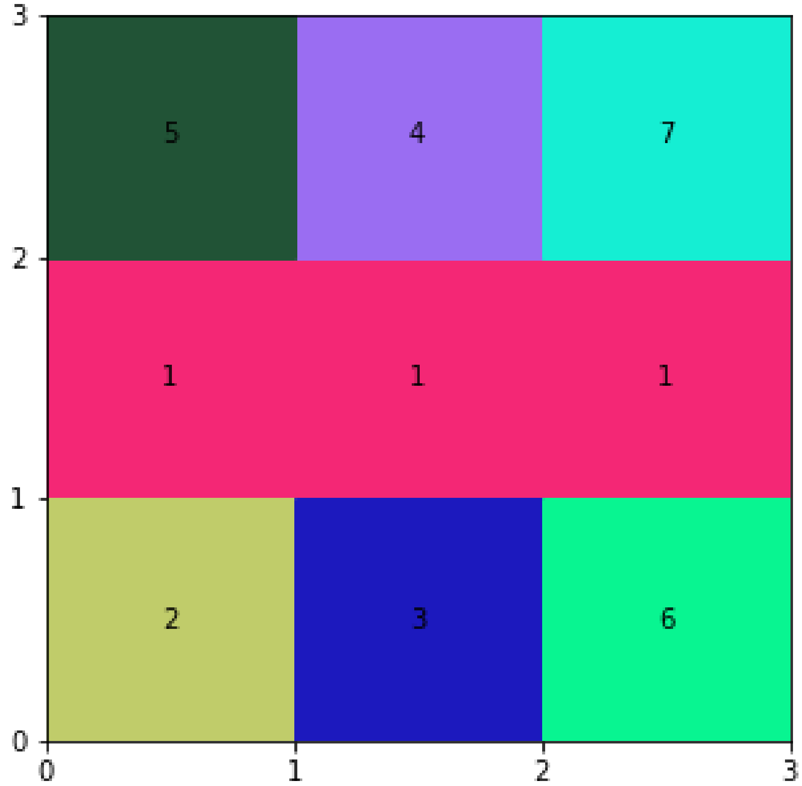

```
On average across 8850 experiments, it takes 2.0 communication towers before full coverage is ob
```



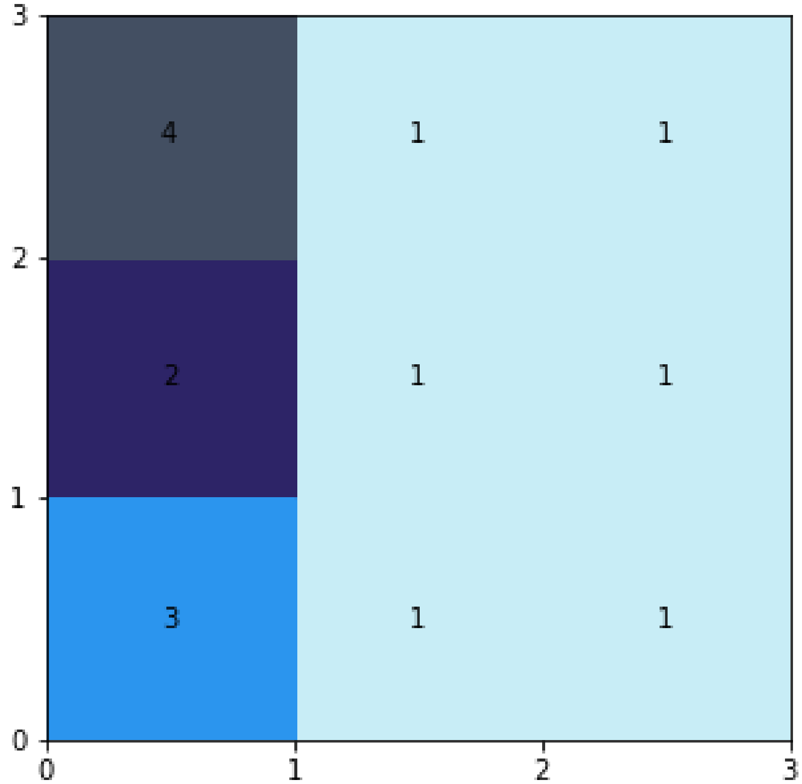
On average across 8900 experiments, it takes 4.0 communication towers before full coverage is ob



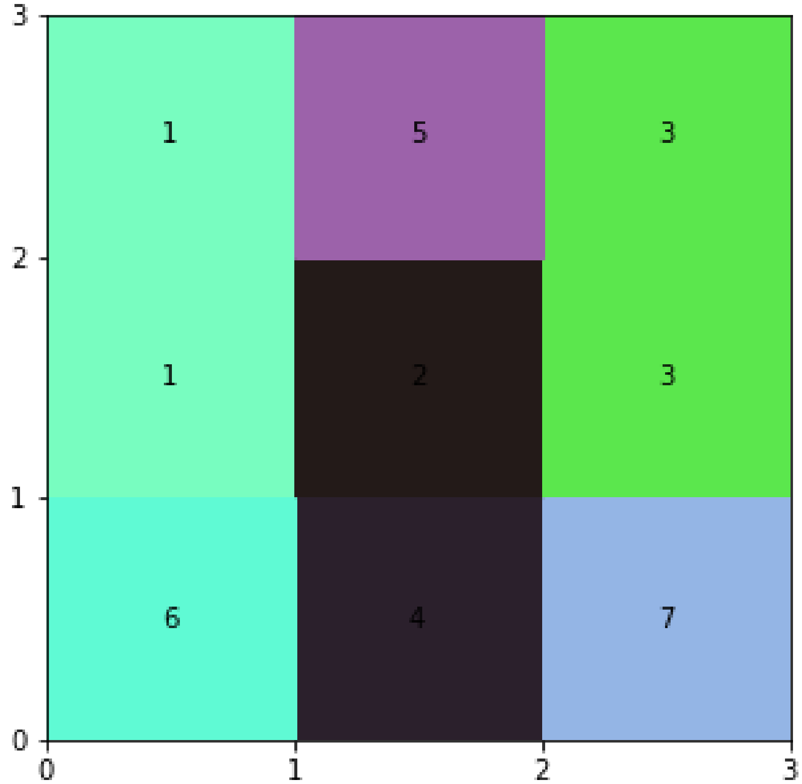
On average across 8950 experiments, it takes 7.0 communication towers before full coverage is ob



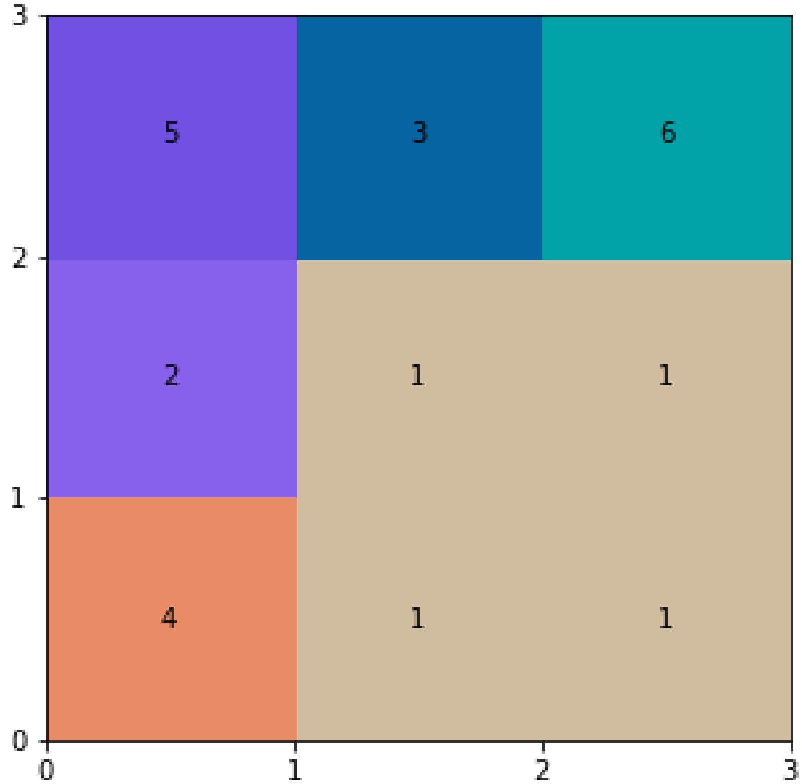
On average across 9000 experiments, it takes 4.0 communication towers before full coverage is ob



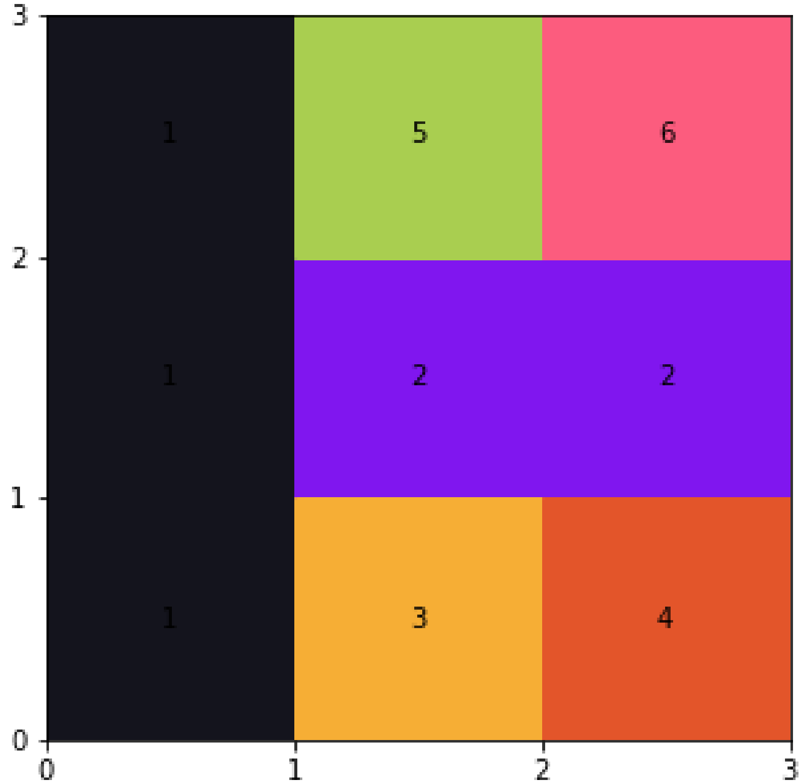
On average across 9050 experiments, it takes 7.0 communication towers before full coverage is ob



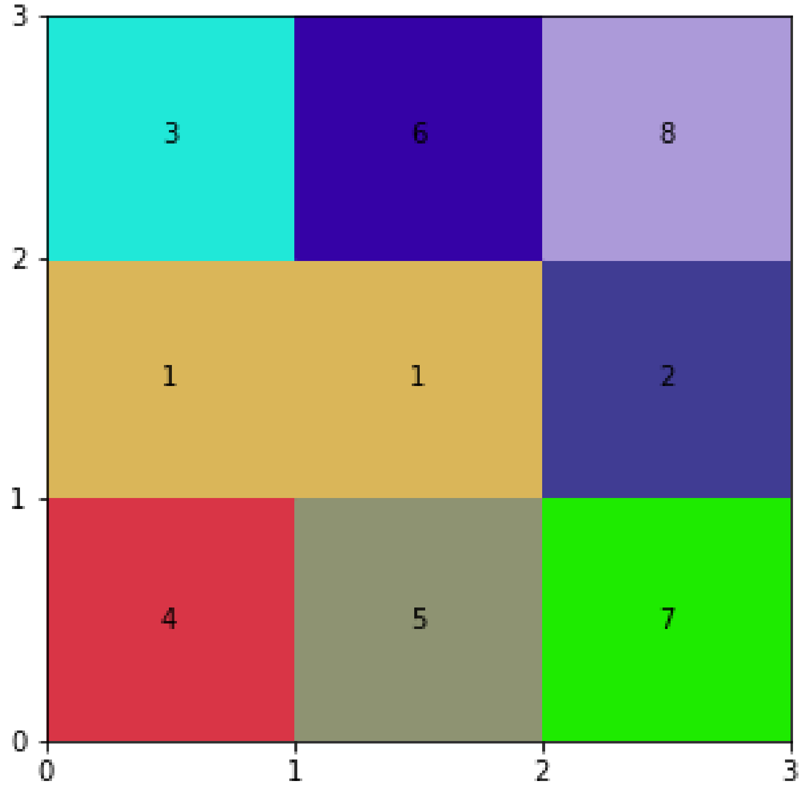
On average across 9100 experiments, it takes 6.0 communication towers before full coverage is ob



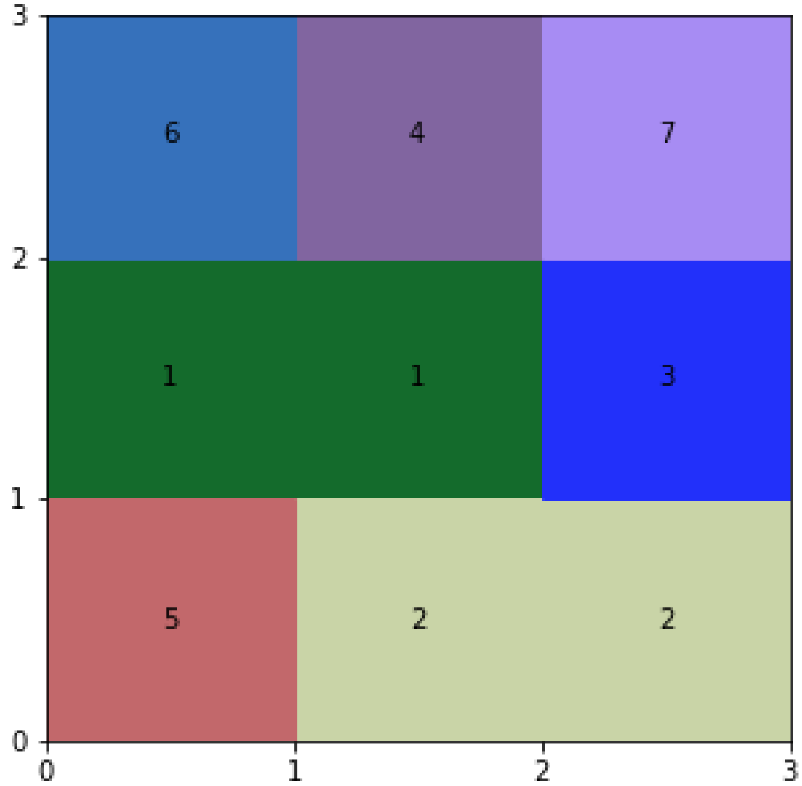

```
On average across 9150 experiments, it takes 6.0 communication towers before full coverage is ob
```



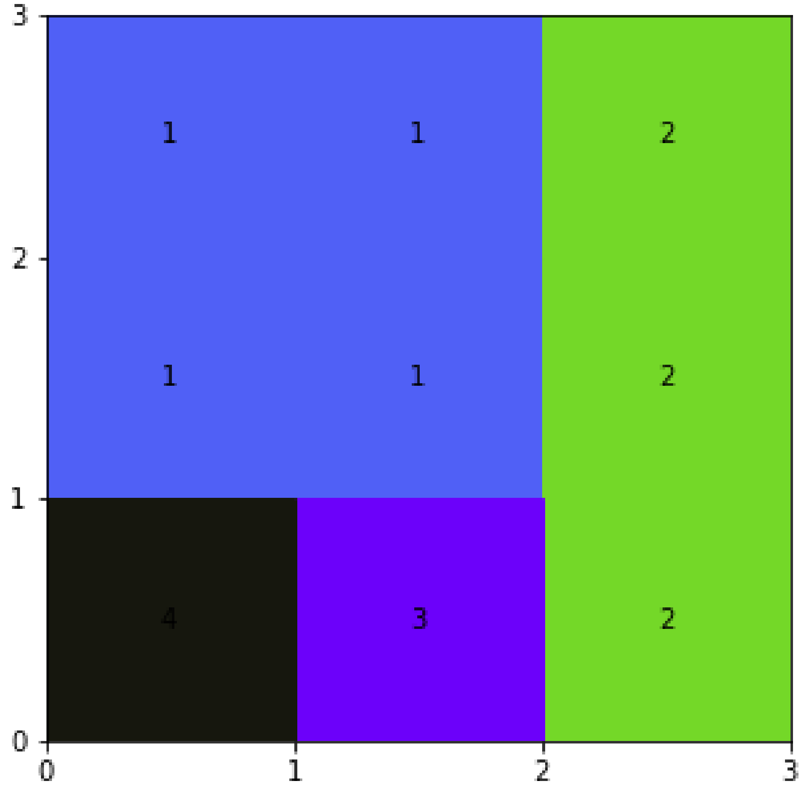
On average across 9200 experiments, it takes 8.0 communication towers before full coverage is ob



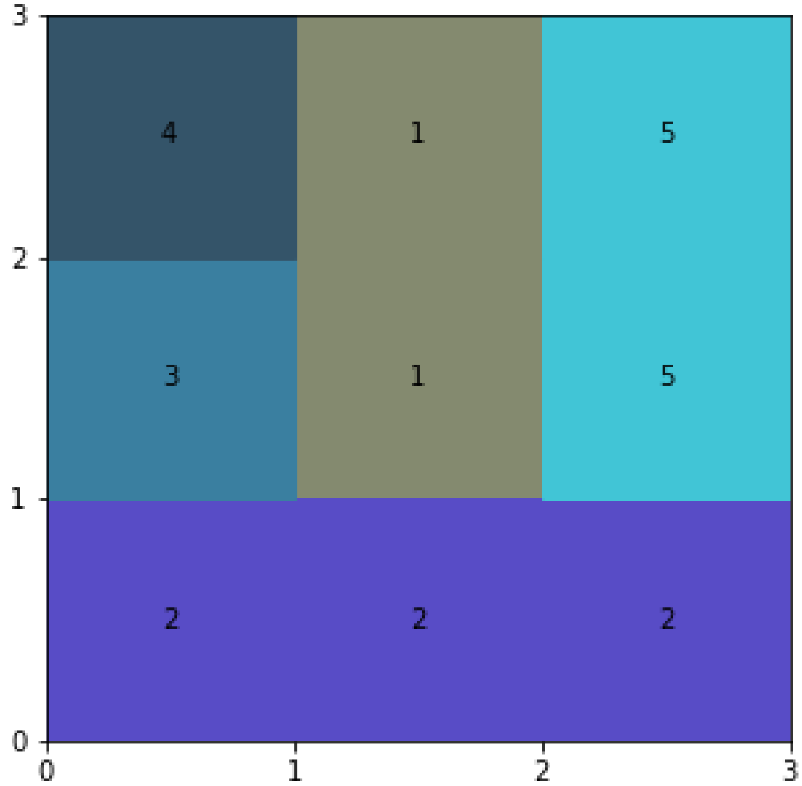
On average across 9250 experiments, it takes 7.0 communication towers before full coverage is ob



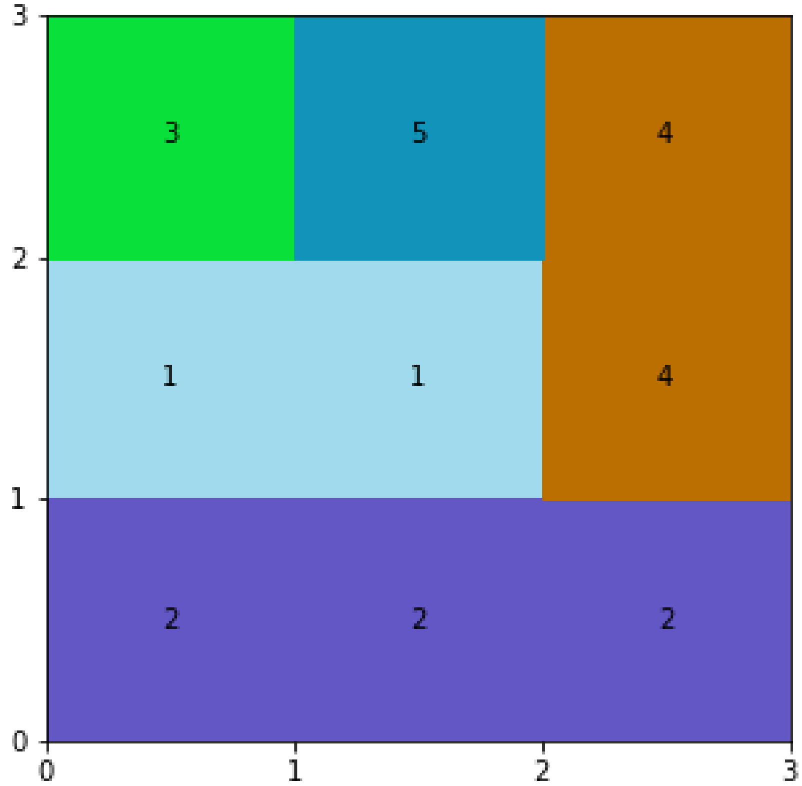
On average across 9300 experiments, it takes 4.0 communication towers before full coverage is ob



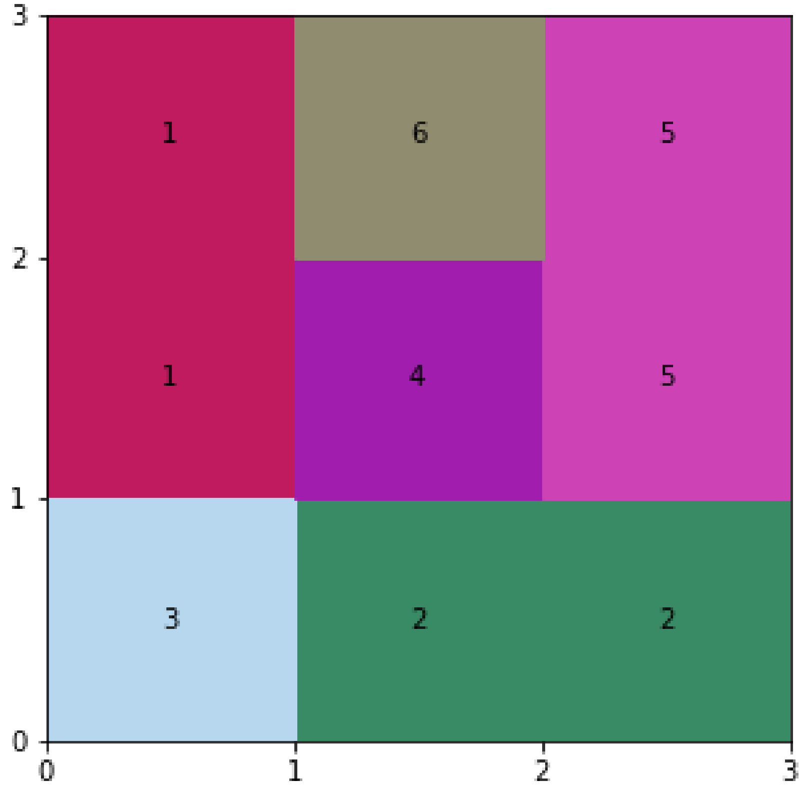

```
On average across 9350 experiments, it takes 5.0 communication towers before full coverage is ob
```



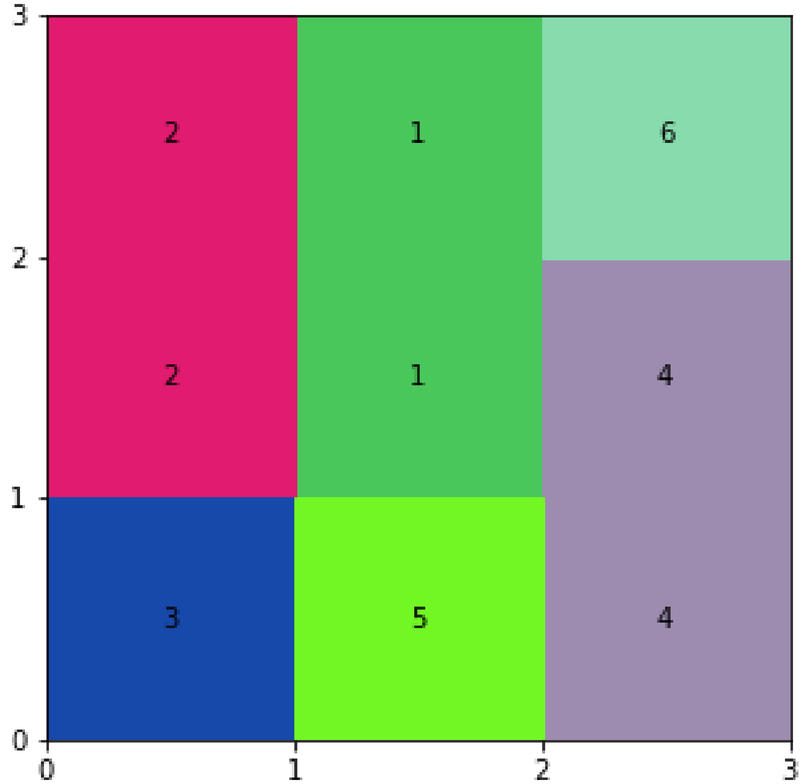
On average across 9400 experiments, it takes 5.0 communication towers before full coverage is ob



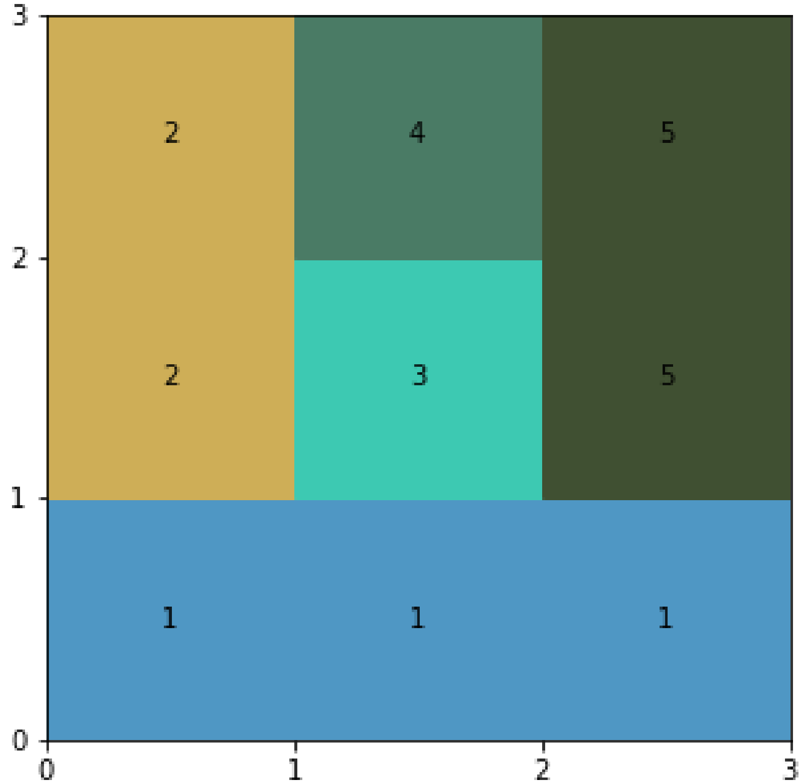
On average across 9450 experiments, it takes 6.0 communication towers before full coverage is ob



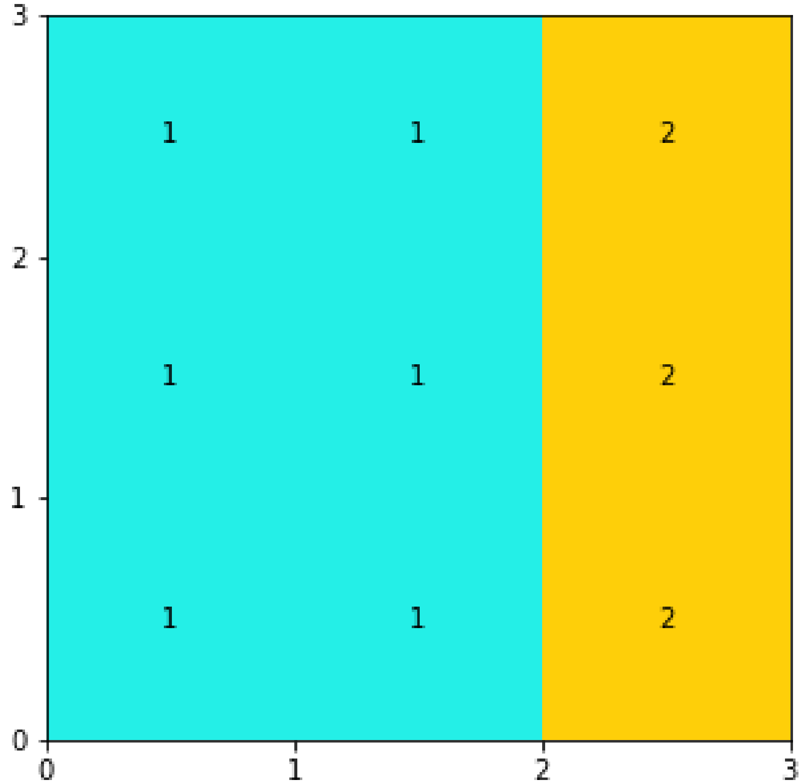
On average across 9500 experiments, it takes 6.0 communication towers before full coverage is ob



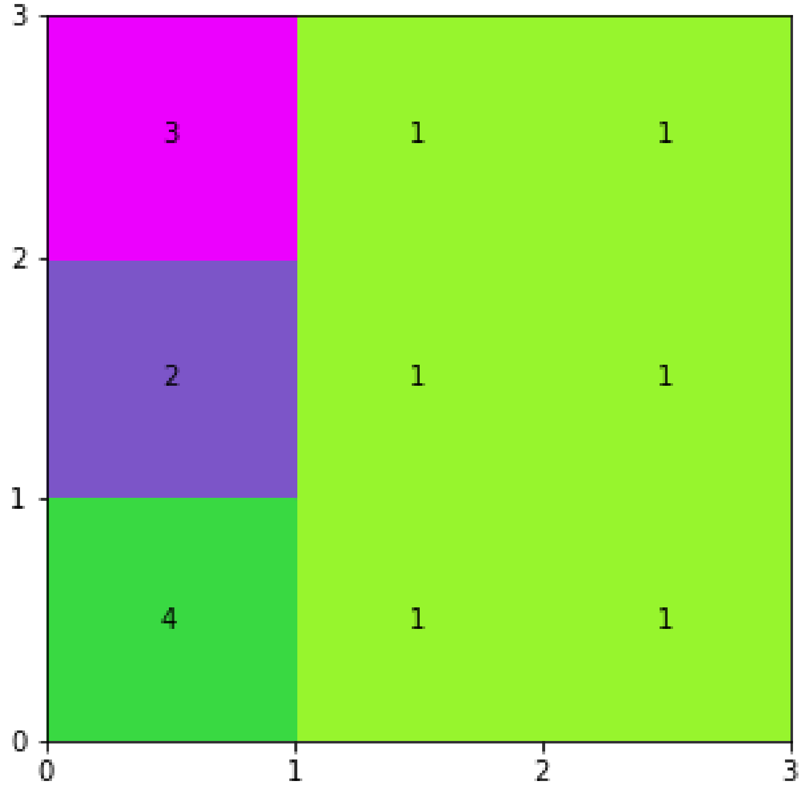
On average across 9550 experiments, it takes 5.0 communication towers before full coverage is ob



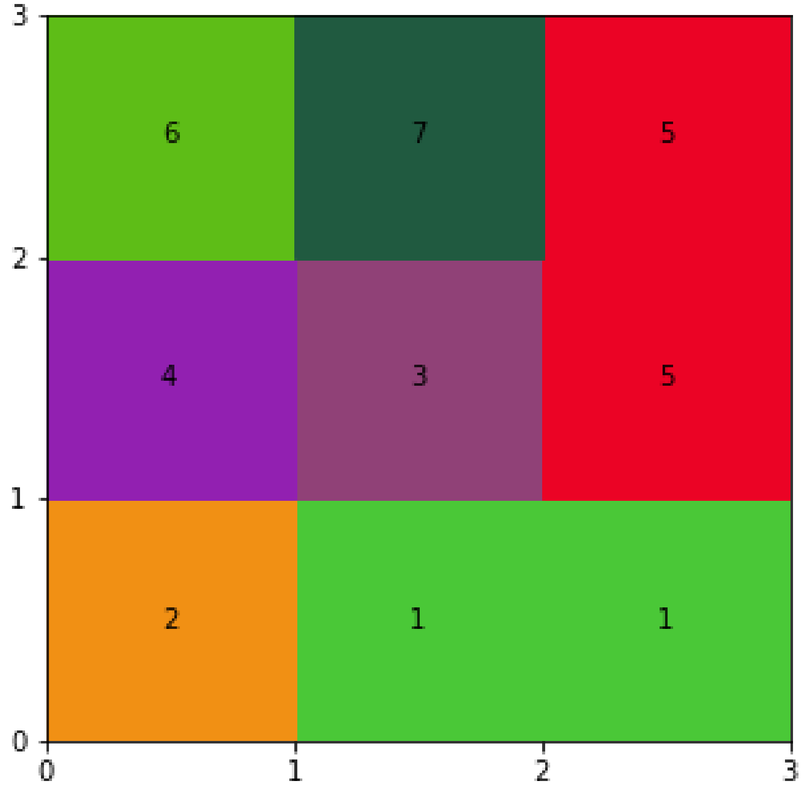

```
On average across 9600 experiments, it takes 2.0 communication towers before full coverage is ob
```



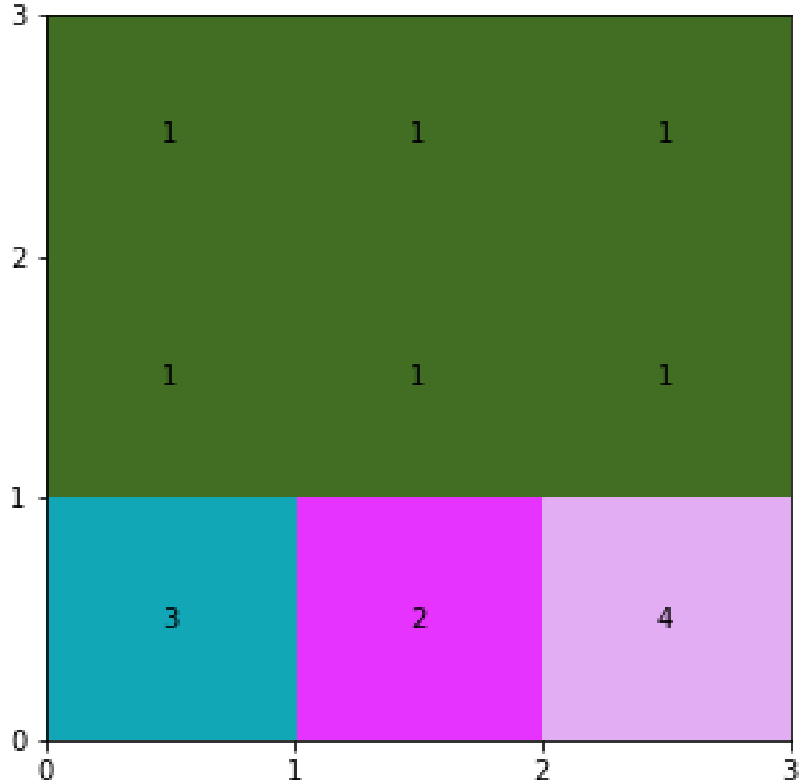

```
On average across 9650 experiments, it takes 4.0 communication towers before full coverage is ob
```



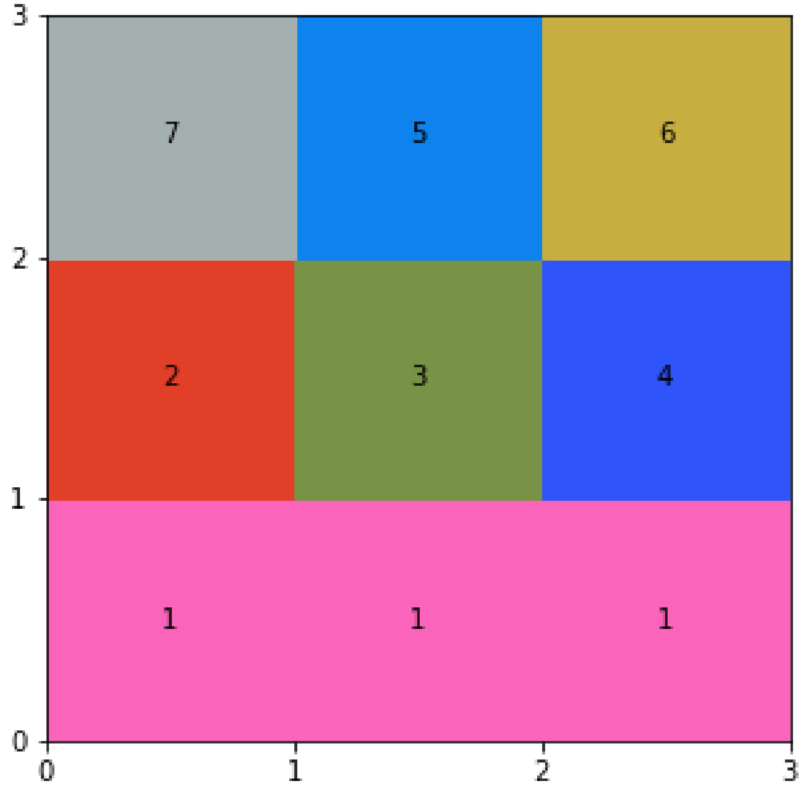

```
On average across 9700 experiments, it takes 7.0 communication towers before full coverage is ob
```



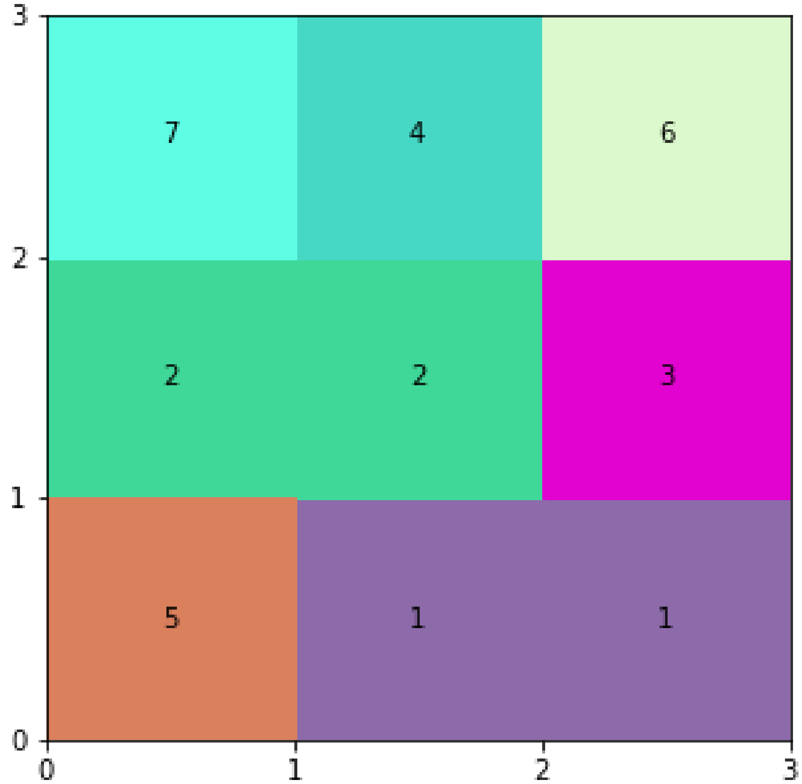
On average across 9750 experiments, it takes 4.0 communication towers before full coverage is ob



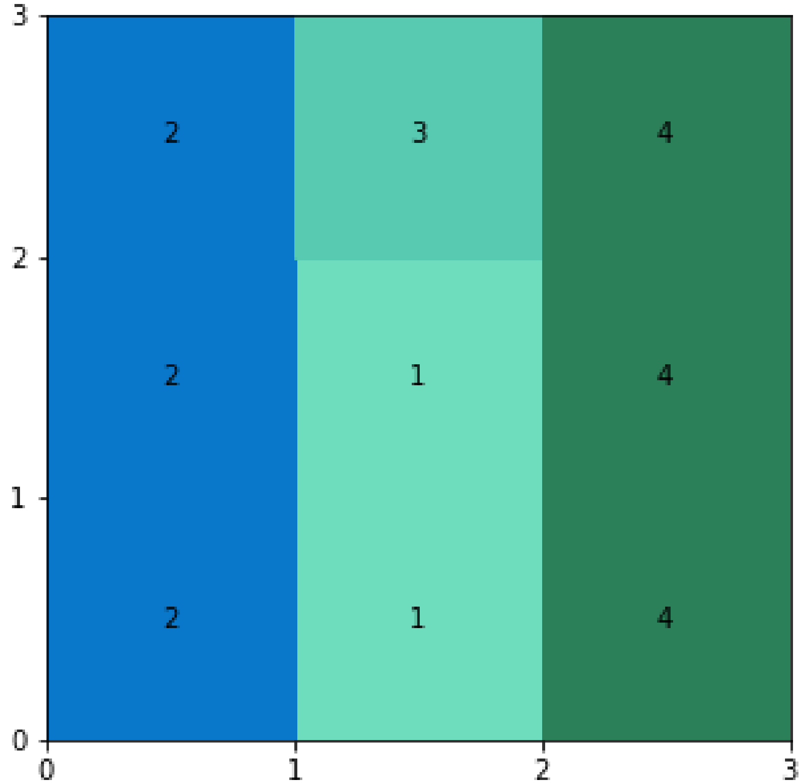
On average across 9800 experiments, it takes 7.0 communication towers before full coverage is ob



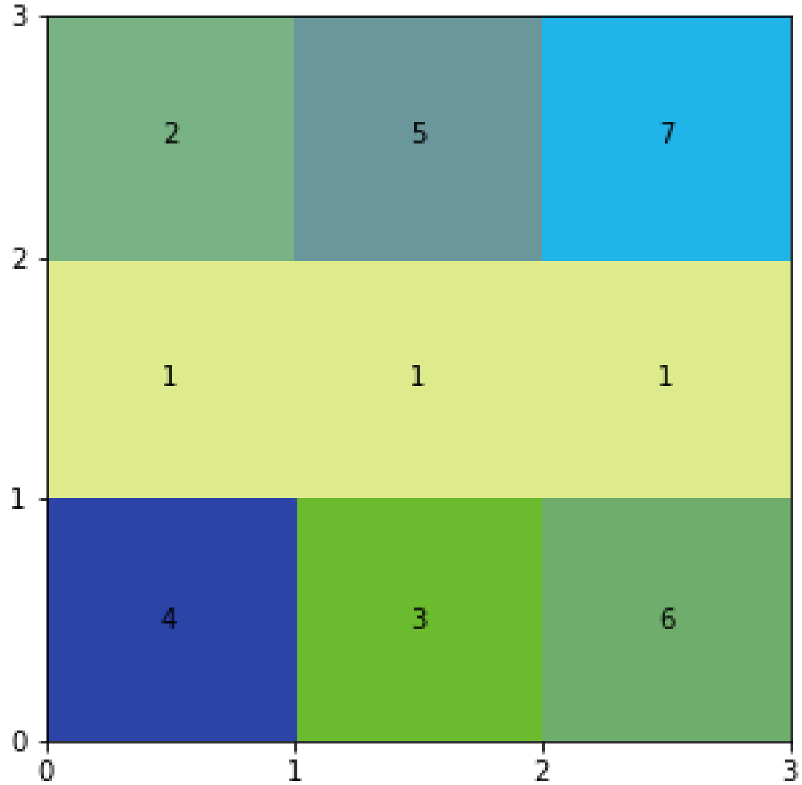
`On average across 9850 experiments, it takes 7.0 communication towers before full coverage is ob`



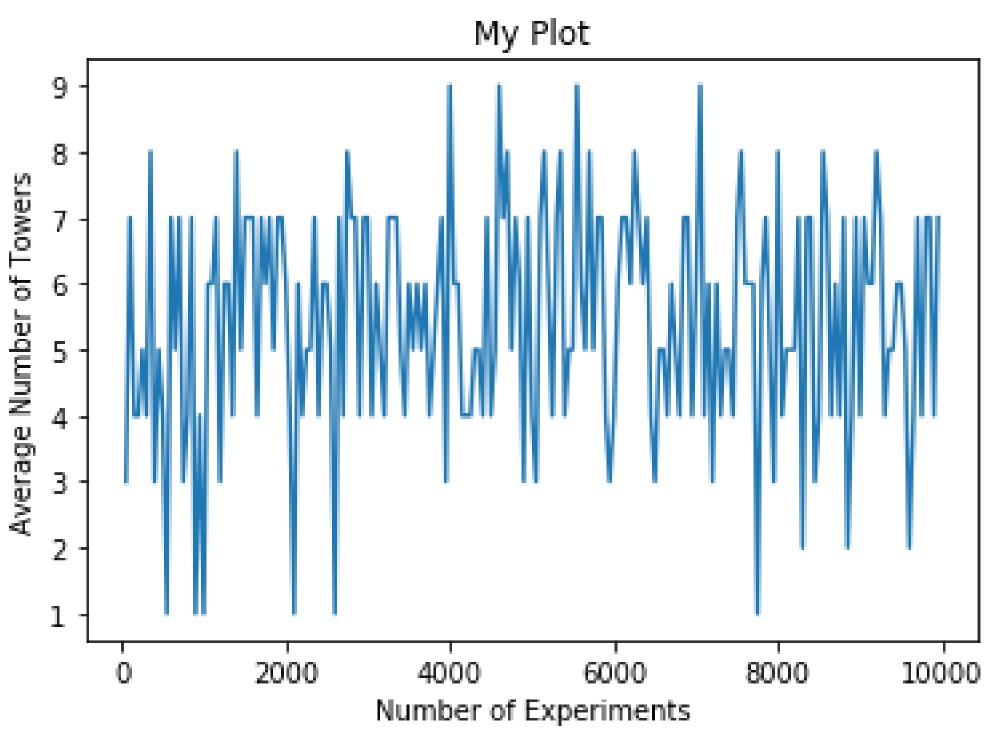
On average across 9900 experiments, it takes 4.0 communication towers before full coverage is ob




```
On average across 9950 experiments, it takes 7.0 communication towers before full coverage is ob
```



2
5
7
1
1
1
4
3
6

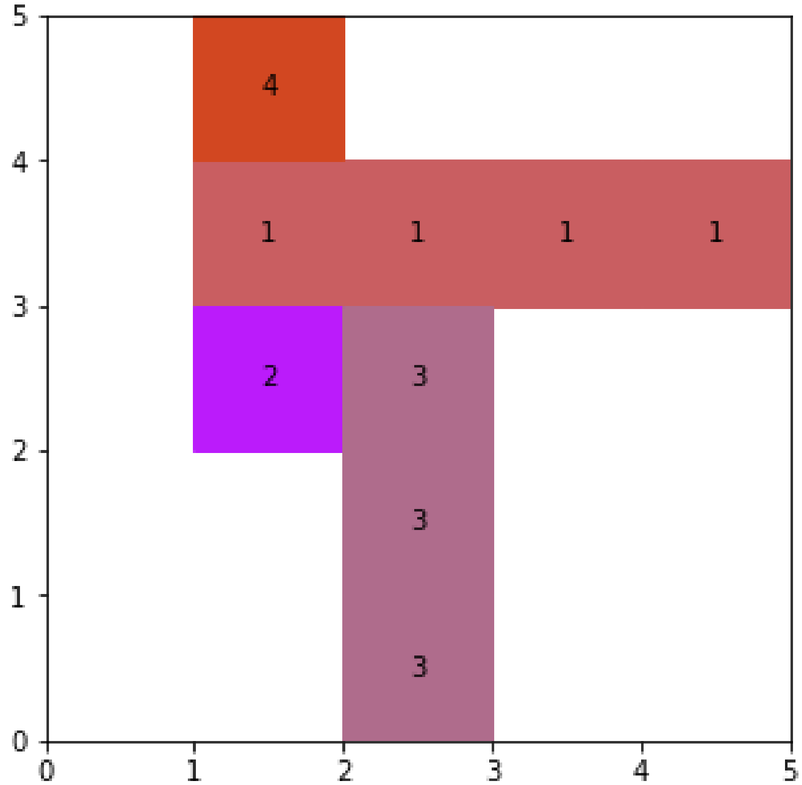


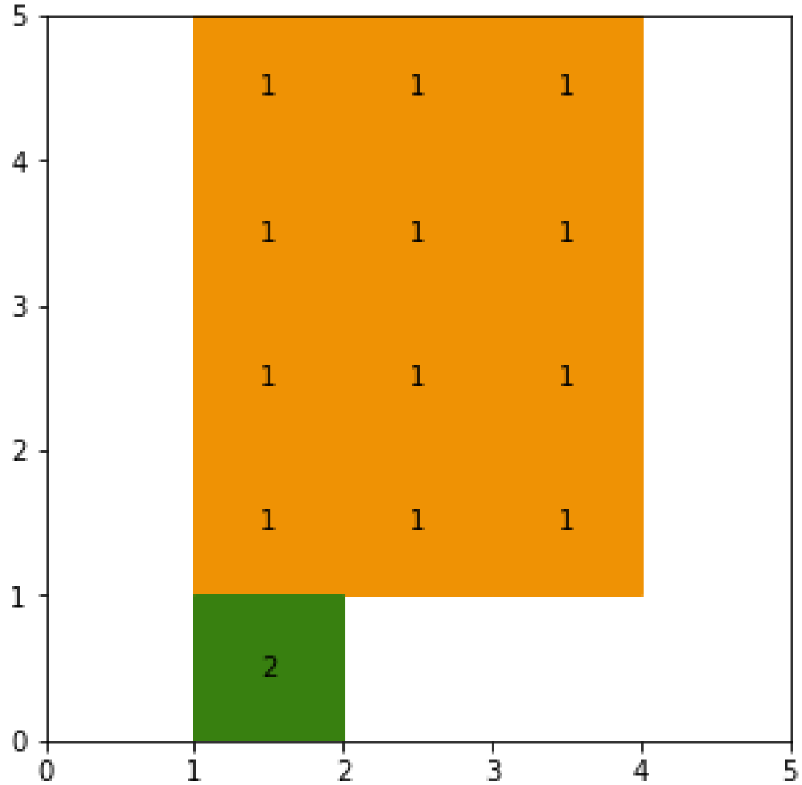
My Plot
Average Number of Towers
Number of Experiments

(3) On average, how many communications towers are required before full coverage is obtained? (additional analysis)

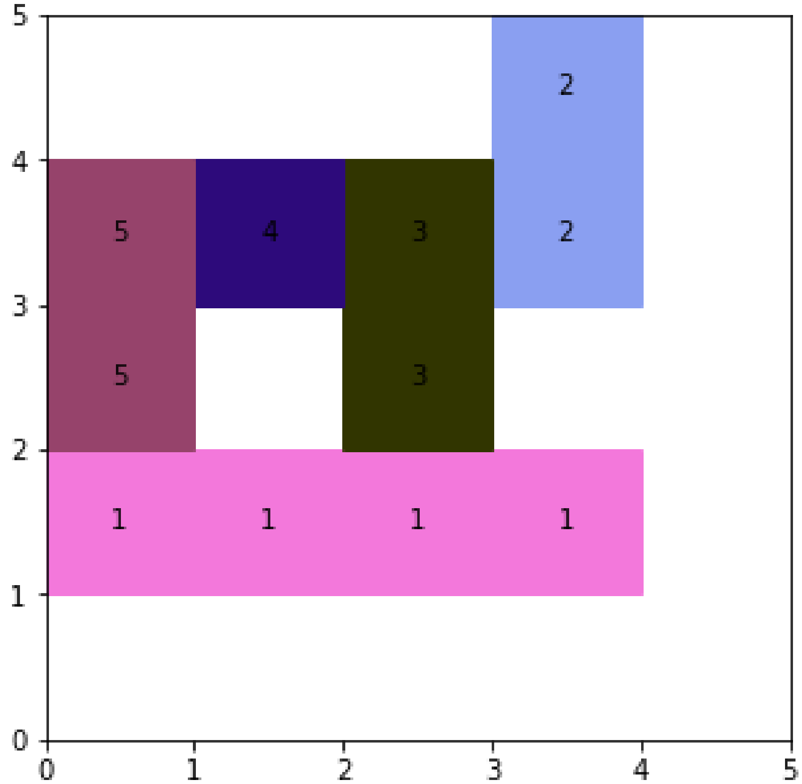
As shown, the code shows across variable-valued map dimensions, the % of total coverage goes dow
assuming a fixed number of trials to take average off of, and this makes sense because we can as
that, as the area gets larger, it would be harder and harder to cover a bigger portion of the de
coverage. Thus, the value of dimension is inversely proportional to the total % of area coverage
Note: The numbers and colors show the type of color and tower numbers associated with the rectan

```
In [64]: # define necessary variables
         startIndex = 5
         endIndex = 20
         numExperiment = 5
         maxTrial = 5
         myList = list()
         # dimension of the grid
         for k in range(startIndex, endIndex):
             # after this loop found numExperiment full maps
             averageArea = 0
             for i in range(0, numExperiment):
                 xLim = yLim = k
                 obj = MyClass(xLim, yLim)
                 # after this loop we found one full map
                 for j in range(0, maxTrial):
                     startLength = random.randint(0, xLim - 2)
                     startWidth = random.randint(0, yLim - 2)
                     endLength = random.randint(startLength + 1, xLim)
                     endWidth = random.randint(startWidth + 1, yLim)
                     value = obj.add(startLength, startWidth, endLength, endWidth)
                     if value == 0:
                         break
                 obj.displayMap()
                 if value != 0:
                     averageArea += (obj.getCurrentArea() * 1.00 / obj.getTotalArea())
             averageArea /= numExperiment
             myTuple = (k, averageArea)
             myList.append(myTuple)
         # plot the graph
         plt.plot(*zip(*myList))
         plt.title('My Plot')
         plt.xlabel('Dimension of map')
         plt.ylabel('% of total coverage')
         plt.show()
```

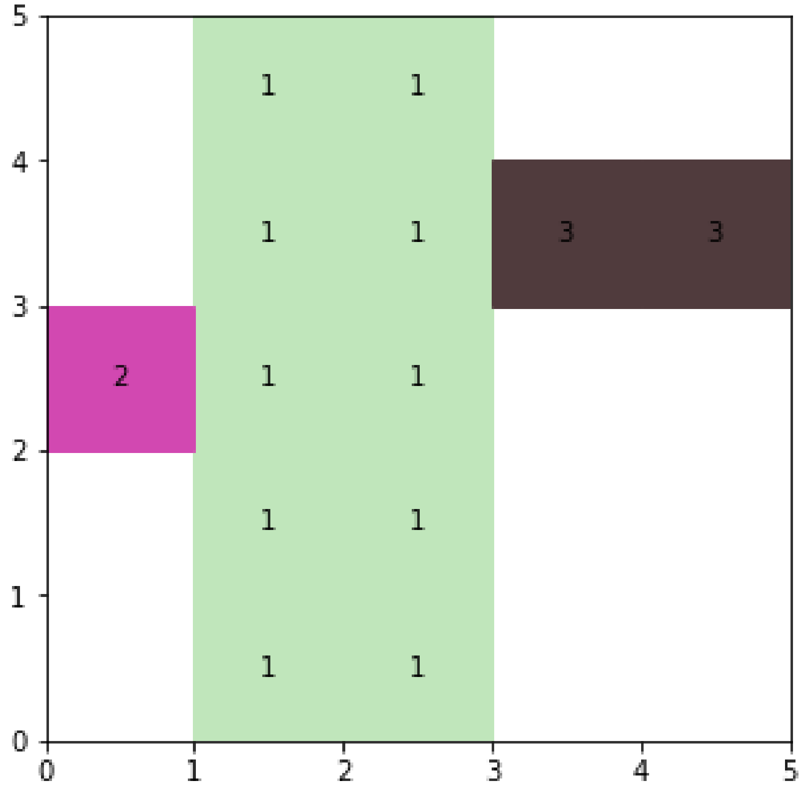

4
1
1
1
1
2
3
3
3
0
1
2
3
4
5
0
1
2
3
4
5



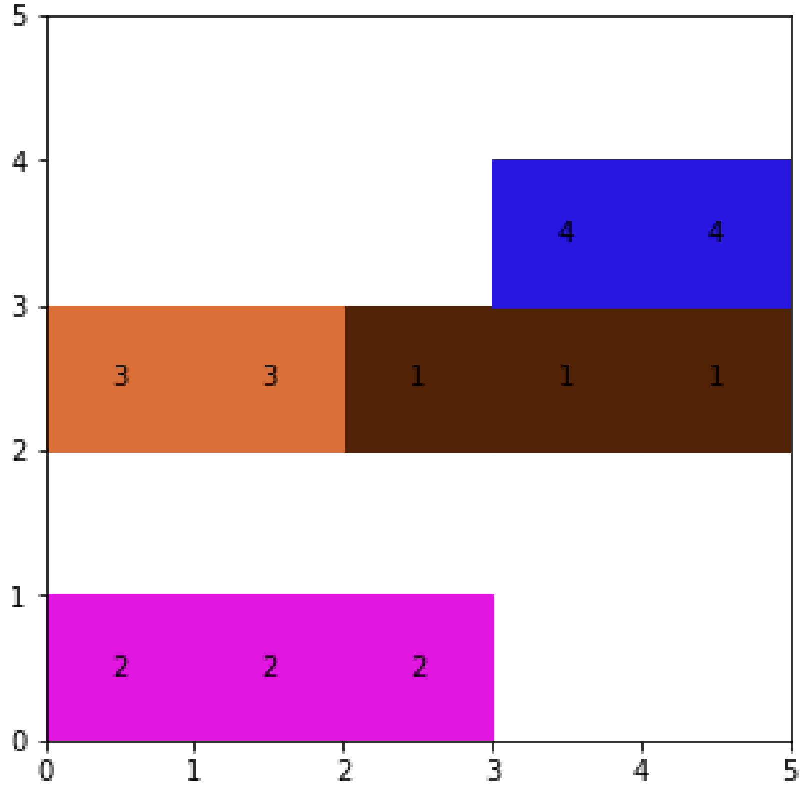
1 1 1
1 1 1
1 1 1
1 1 1
2
0 1 2 3 4 5
0 1 2 3 4 5



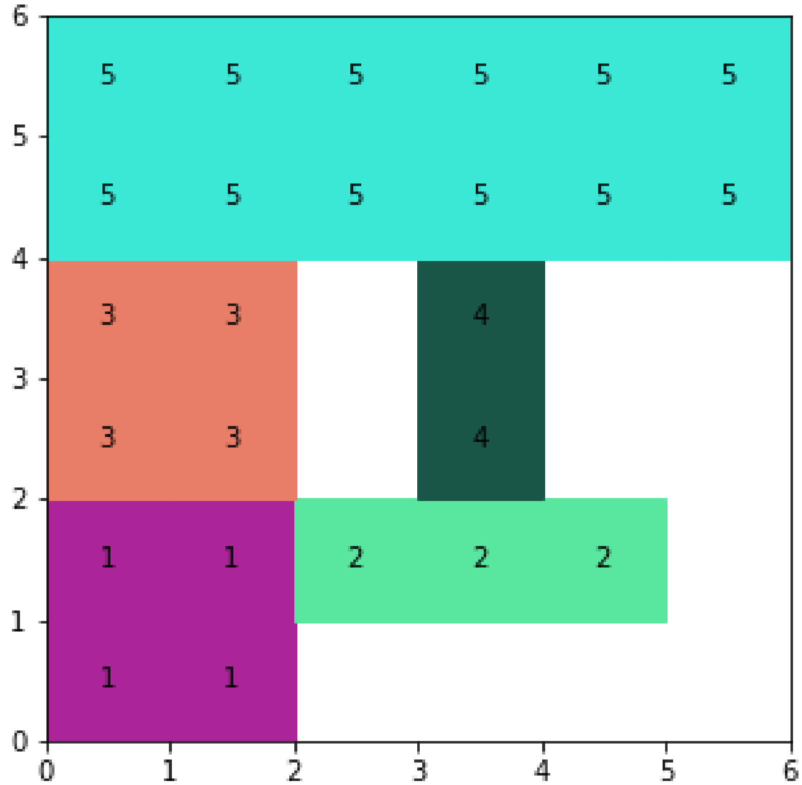
2
2
5
4
3
5
3
1
1
1
1
0
1
2
3
4
5



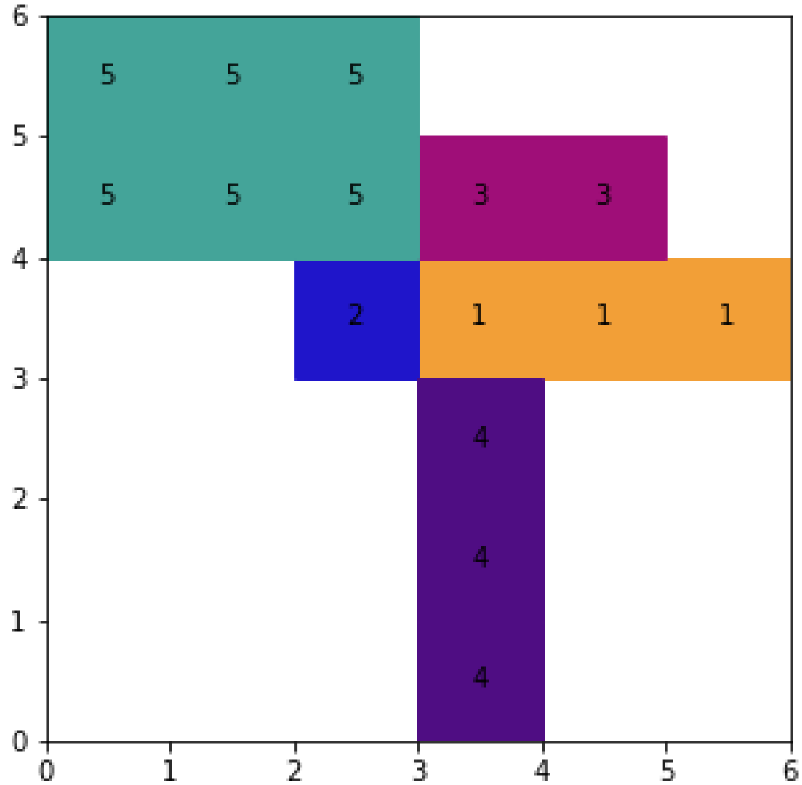
1
1
1
1
3
3
2
1
1
1
1
1
1
0
1
2
3
4
5



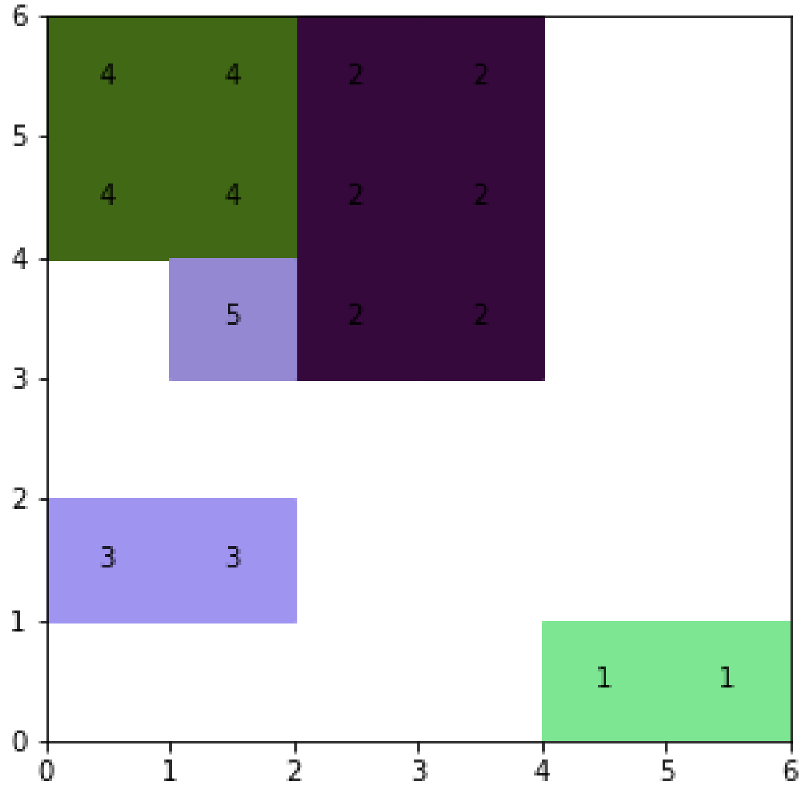
5
4
3
2
1
0
4 4
3 3 1 1 1
2 2 2
0 1 2 3 4 5



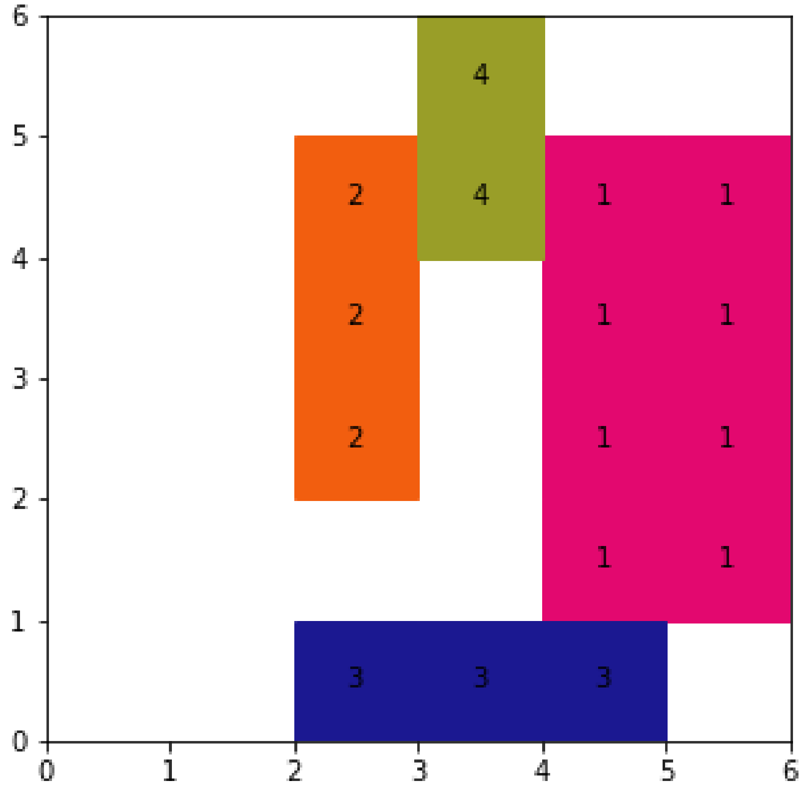
5 5 5 5 5 5
5 5 5 5 5 5
3 3 4
3 3 4
1 1 2 2 2
1 1
0 1 2 3 4 5 6



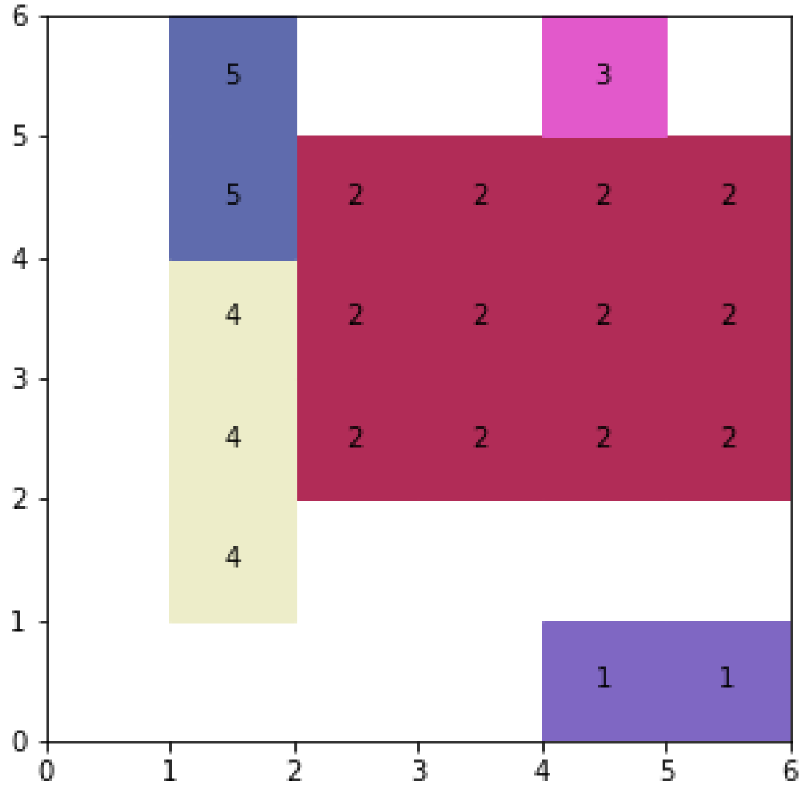
5 5 5
5 5 5 3 3
2 1 1 1
4
4
4
0 1 2 3 4 5 6

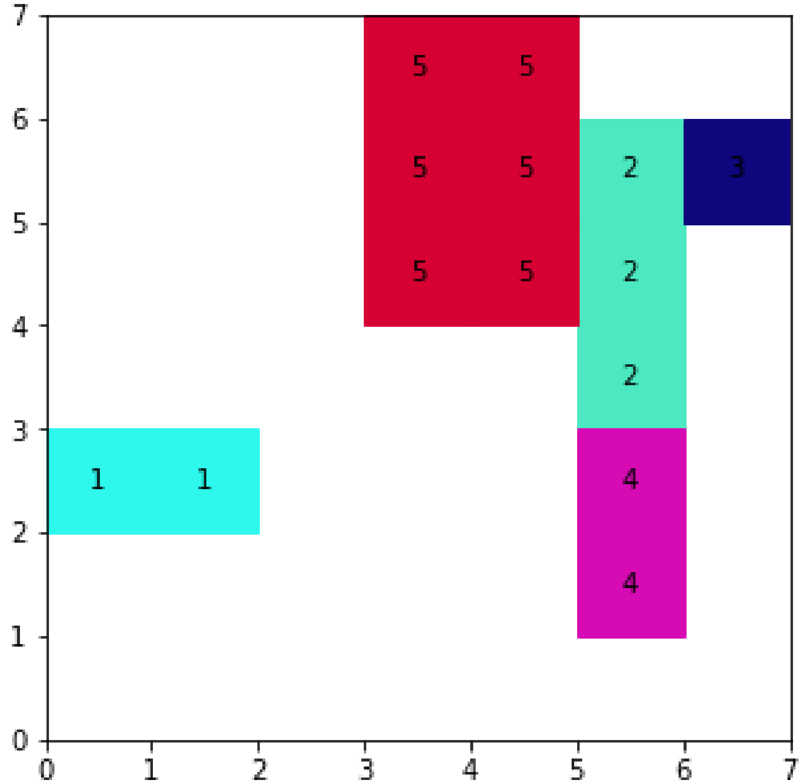


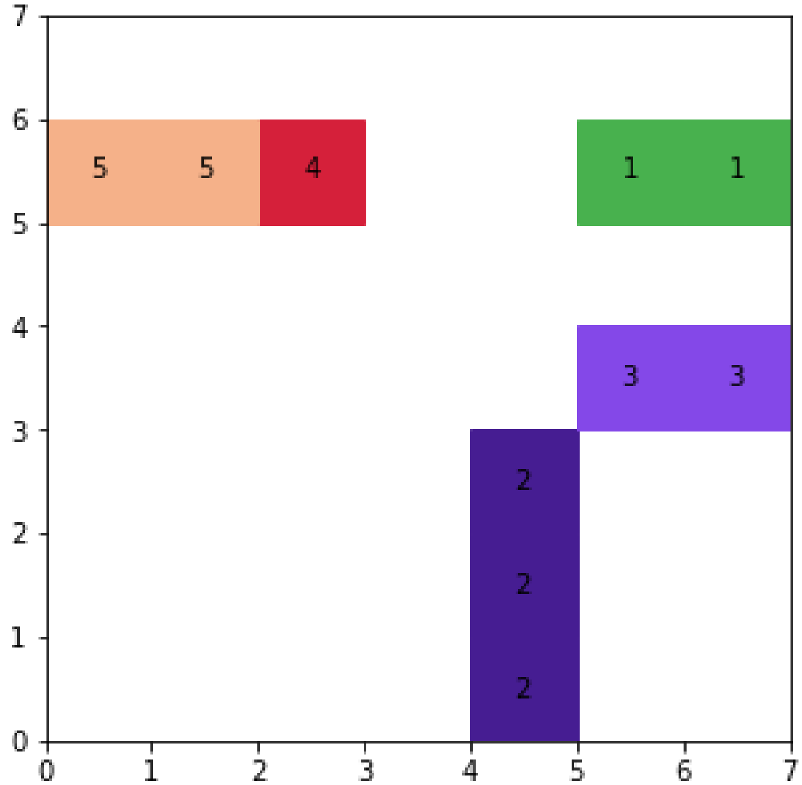
6
5
4
3
2
1
0
0 1 2 3 4 5 6
4 4 2 2
4 4 2 2
5 2 2
3 3
1 1



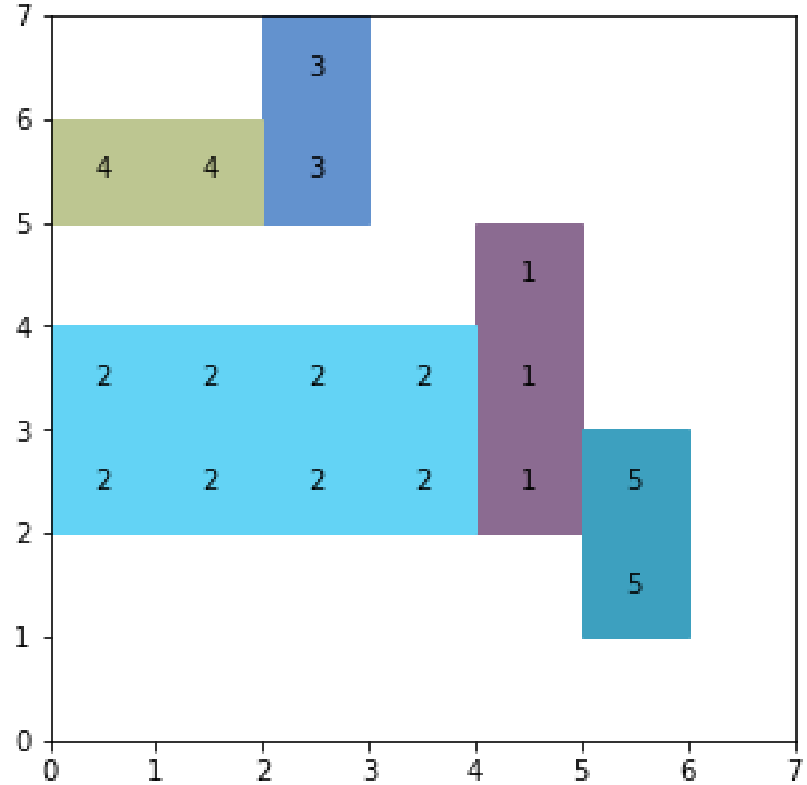
6
5
4
3
2
1
0
0 1 2 3 4 5 6
4
2 4 1 1
2 1 1
2 1 1
1 1
3 3 3


5
5
3
2
2
2
2
4
2
2
2
2
4
2
2
2
2
4
1
1
0
1
2
3
4
5
6

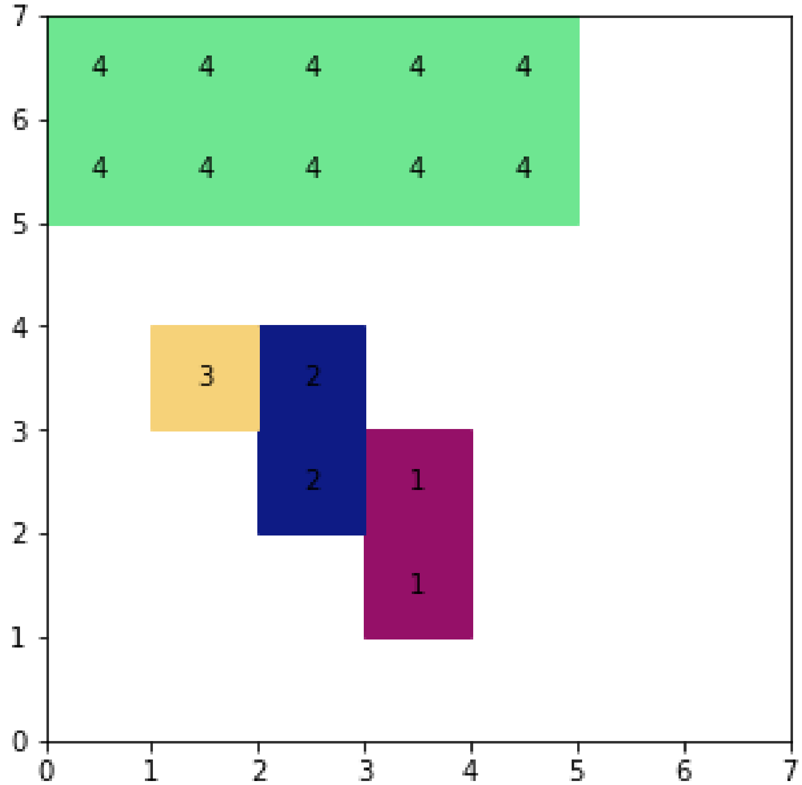

5
5
5
5
2
3
5
5
2
2
1
1
4
4
0
1
2
3
4
5
6
7



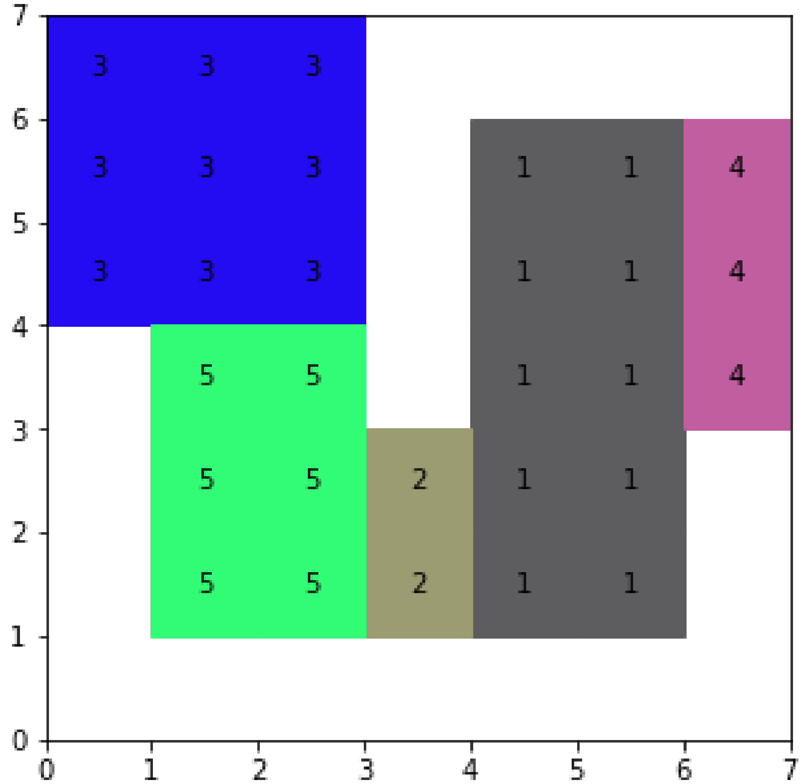
5
5
4
1
1
3
3
2
2
2
0
1
2
3
4
5
6
7



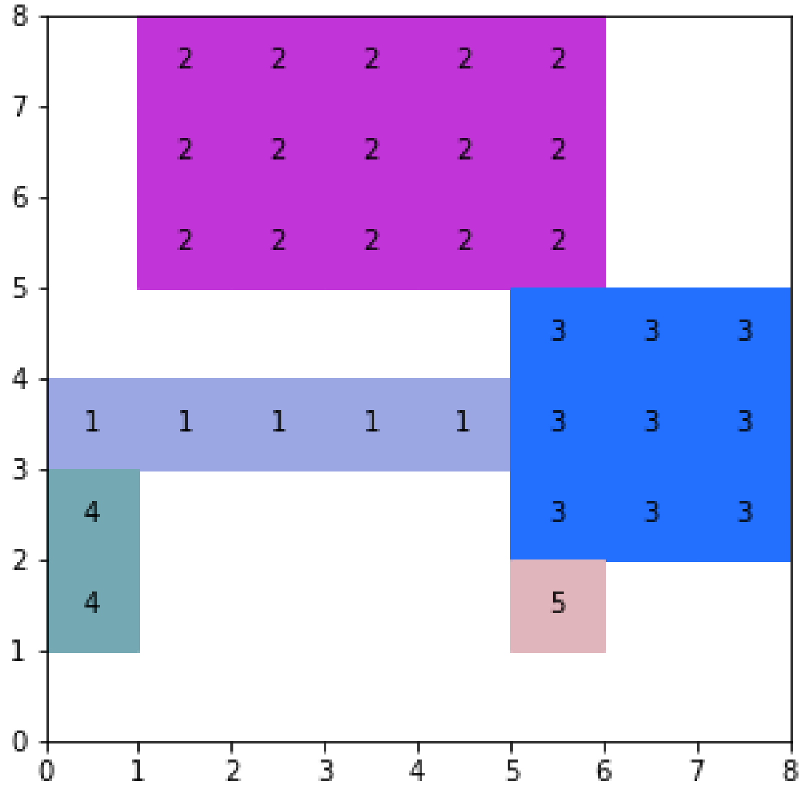
7
6
5
4
3
2
1
0
3
3
4
4
1
2
2
2
2
1
2
2
2
2
1
5
5
0
1
2
3
4
5
6
7



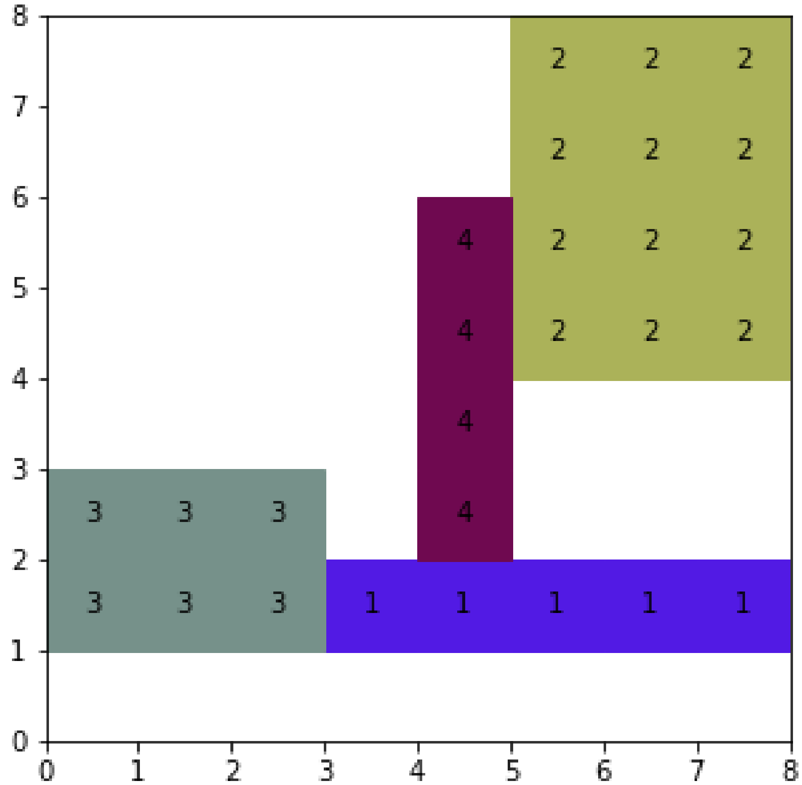
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
4
4
4
4
4
4
4
4
4
4
3
2
2
1
1



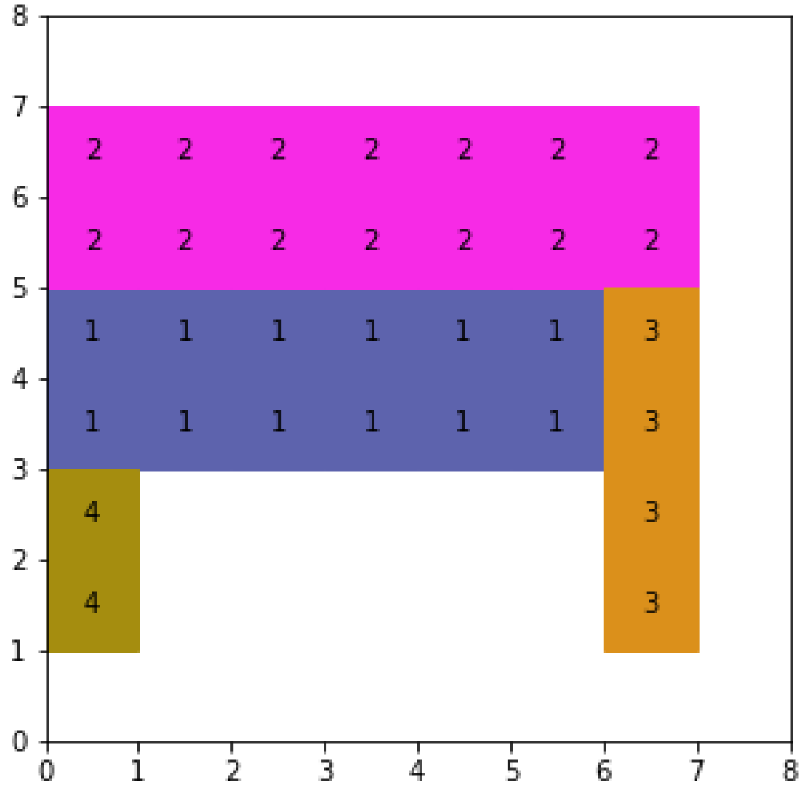
3 3 3
3 3 3 1 1 4
3 3 3 1 1 4
5 5 1 1 4
5 5 2 1 1
5 5 2 1 1
0 1 2 3 4 5 6 7



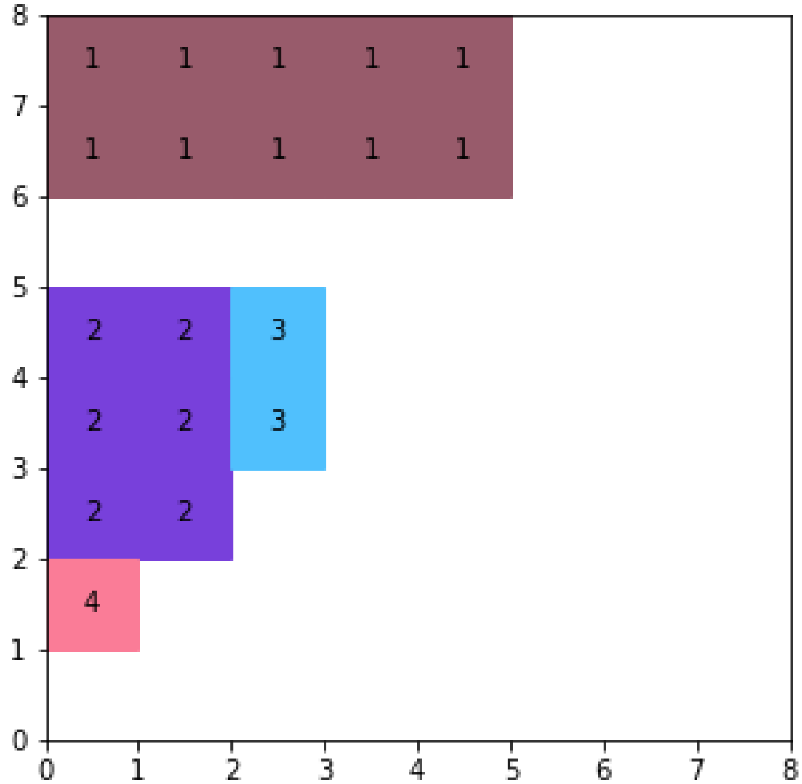
0
1
2
3
4
5
6
7
8
1 1 1 1 1
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
3 3 3
3 3 3
3 3 3
4
4
5



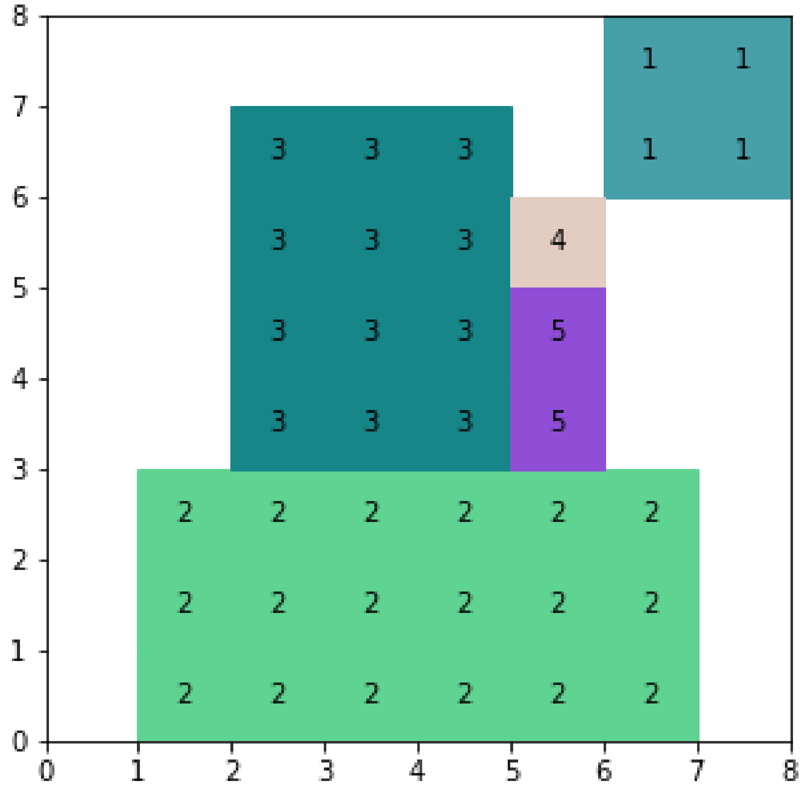
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8
2 2 2
2 2 2
4 2 2 2
4 2 2 2
4
3 3 3 4
3 3 3 1 1 1 1 1



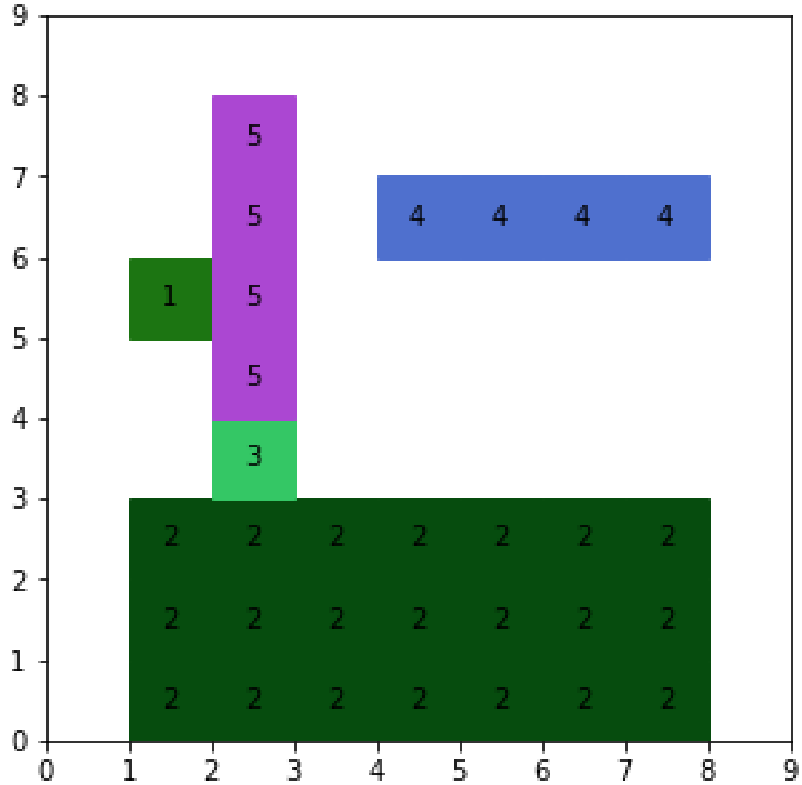
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8
2 2 2 2 2 2 2
2 2 2 2 2 2 2
1 1 1 1 1 1 3
1 1 1 1 1 1 3
4 3
4 3



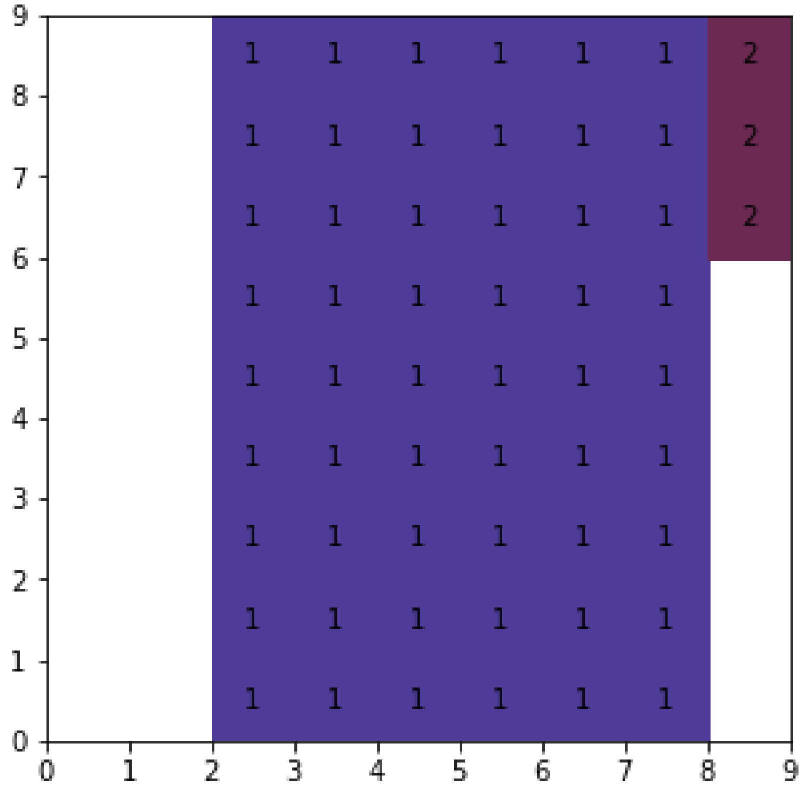
8
7
6
5
4
3
2
1
0
1 1 1 1 1
1 1 1 1 1
2 2 3
2 2 3
2 2
4
0 1 2 3 4 5 6 7 8



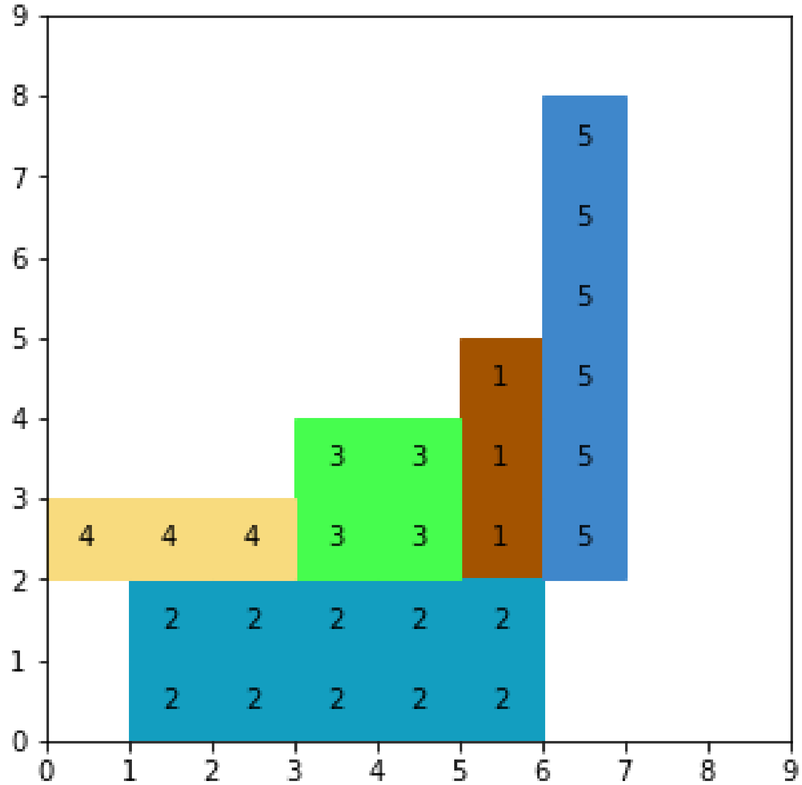
1 1
1 1
3 3 3
3 3 3 4
3 3 3 5
3 3 3 5
2 2 2 2 2 2
2 2 2 2 2 2
2 2 2 2 2 2
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8

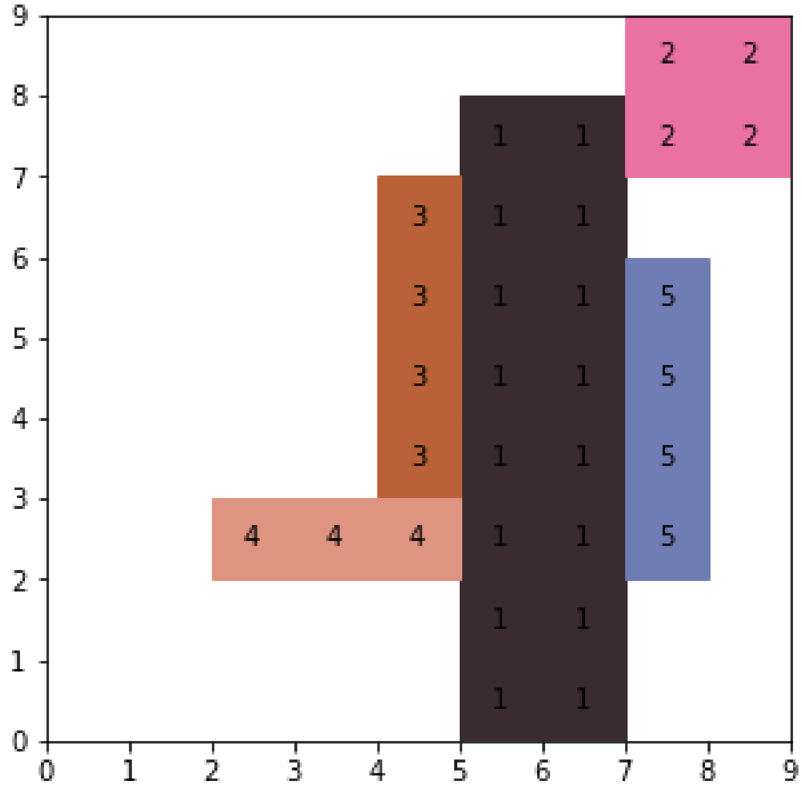


9
8
7
6
5
4
3
2
1
0
5
5
5
5
4 4 4 4
1
3
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
0 1 2 3 4 5 6 7 8 9

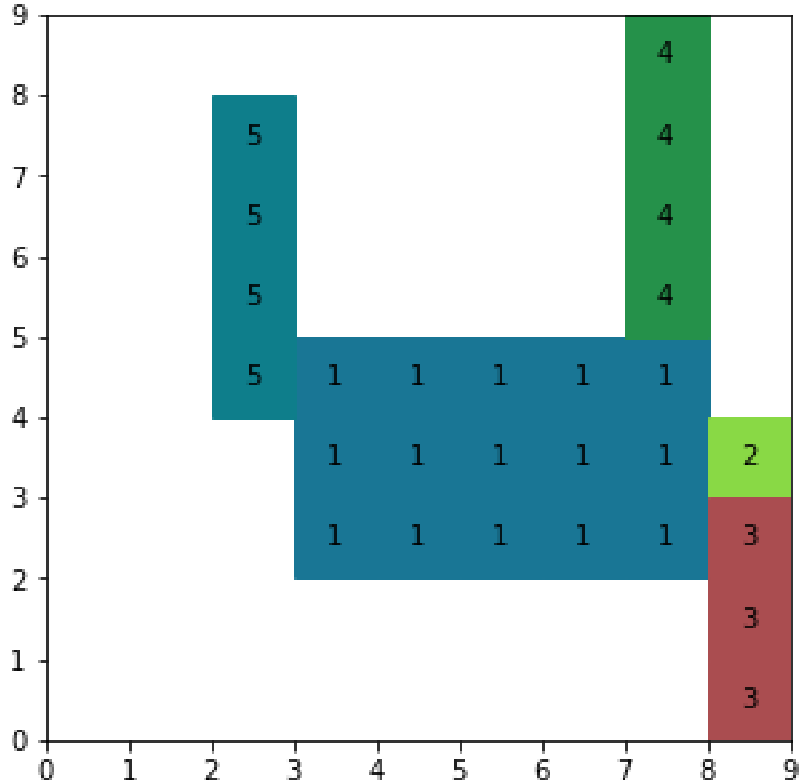


2
2
2

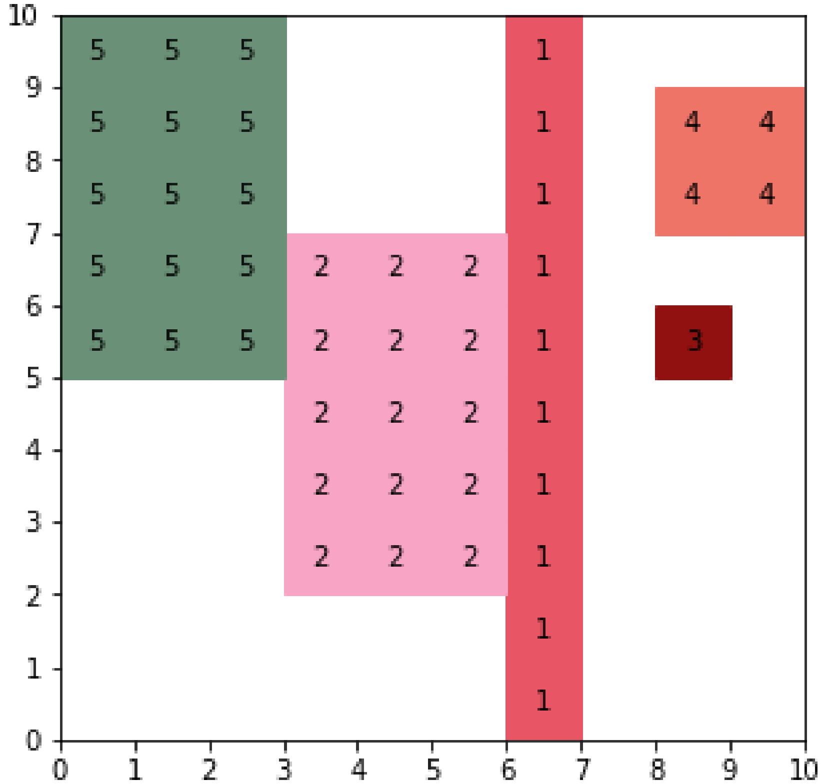

9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9
5
5
5
1 5
3 3 1 5
4 4 4 3 3 1 5
2 2 2 2 2
2 2 2 2 2



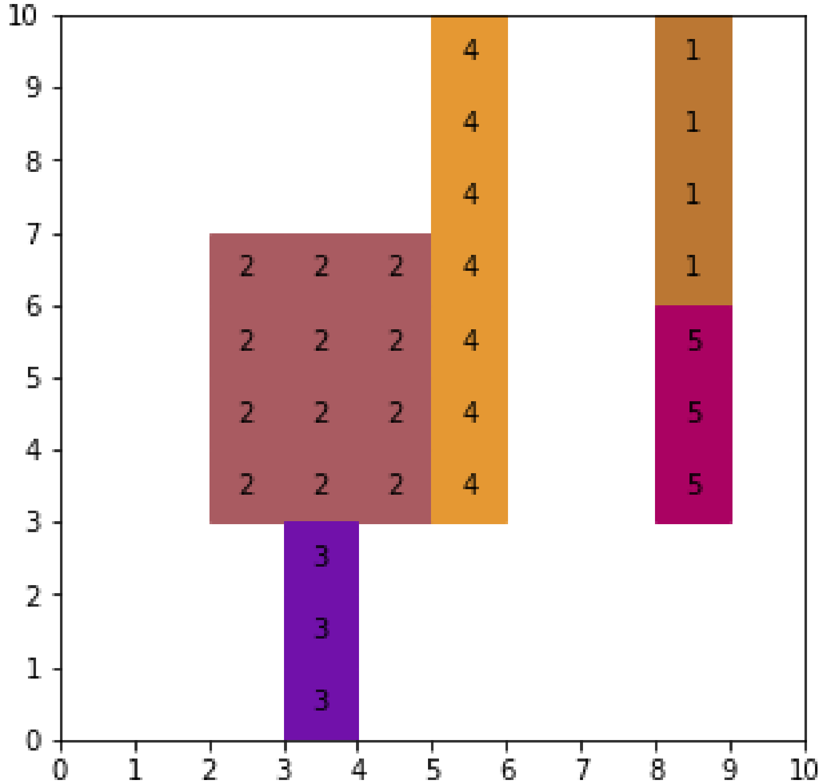
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9
2 2
1 1 2 2
3 1 1
3 1 1 5
3 1 1 5
3 1 1 5
4 4 4 1 1 5
1 1
1 1



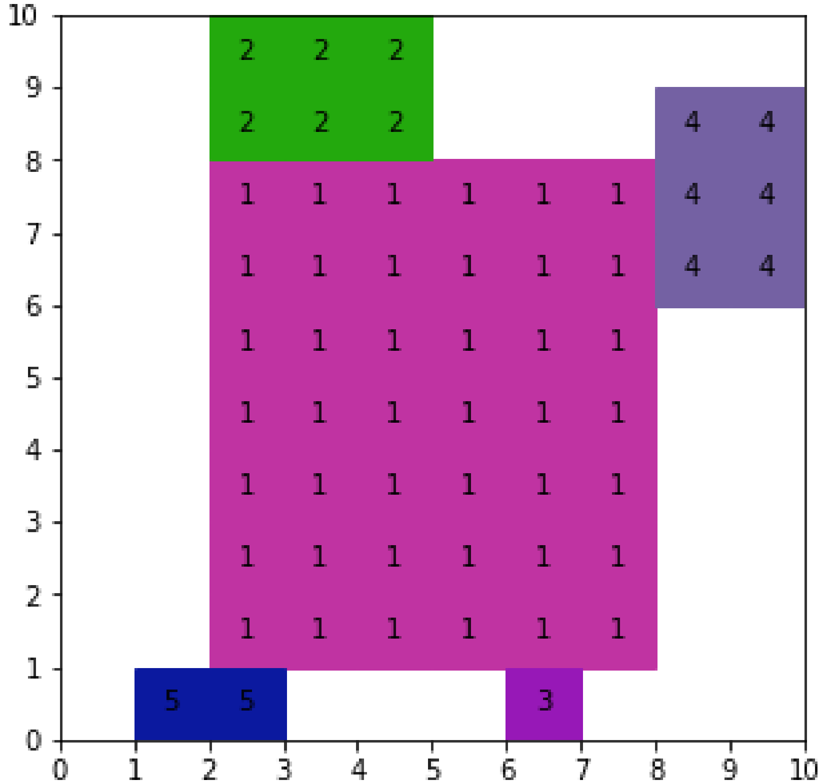
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9
4
4
4
4
5
5
5
5
1 1 1 1 1
2
1 1 1 1 1
3
1 1 1 1 1
3
3



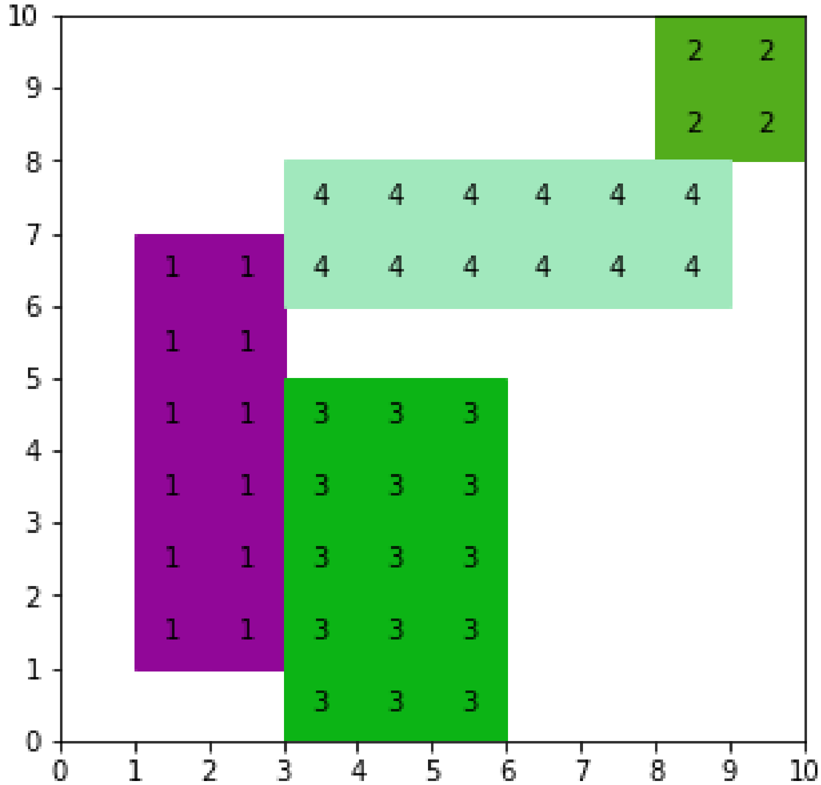
10
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9
10
5 5 5
5 5 5
5 5 5
5 5 5
5 5 5
2 2 2
2 2 2
2 2 2
2 2 2
2 2 2
1
1
1
1
1
1
1
1
1
1
4 4
4 4
3

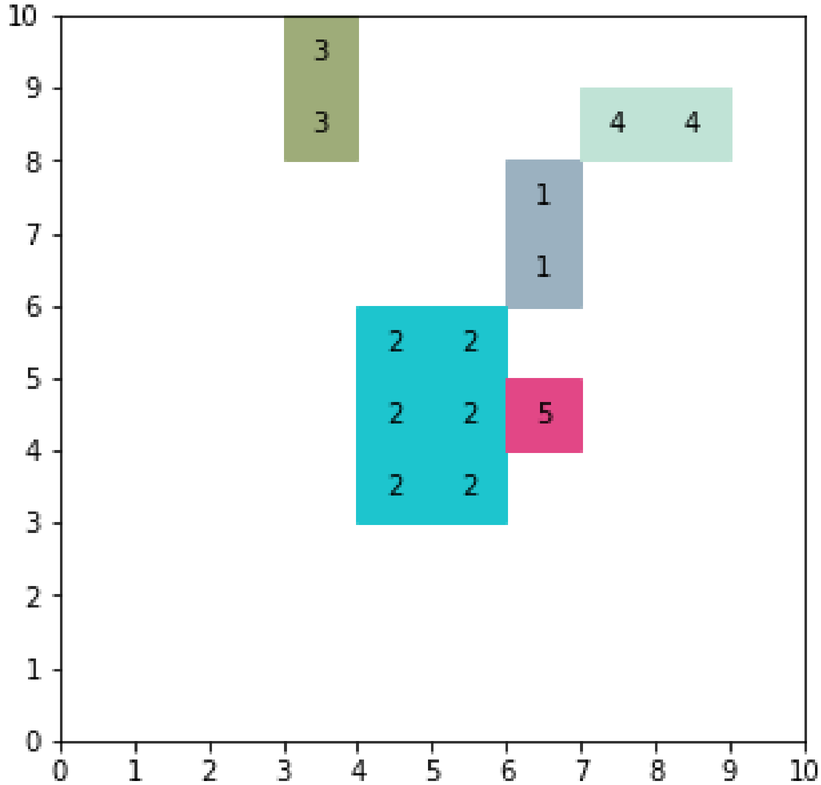


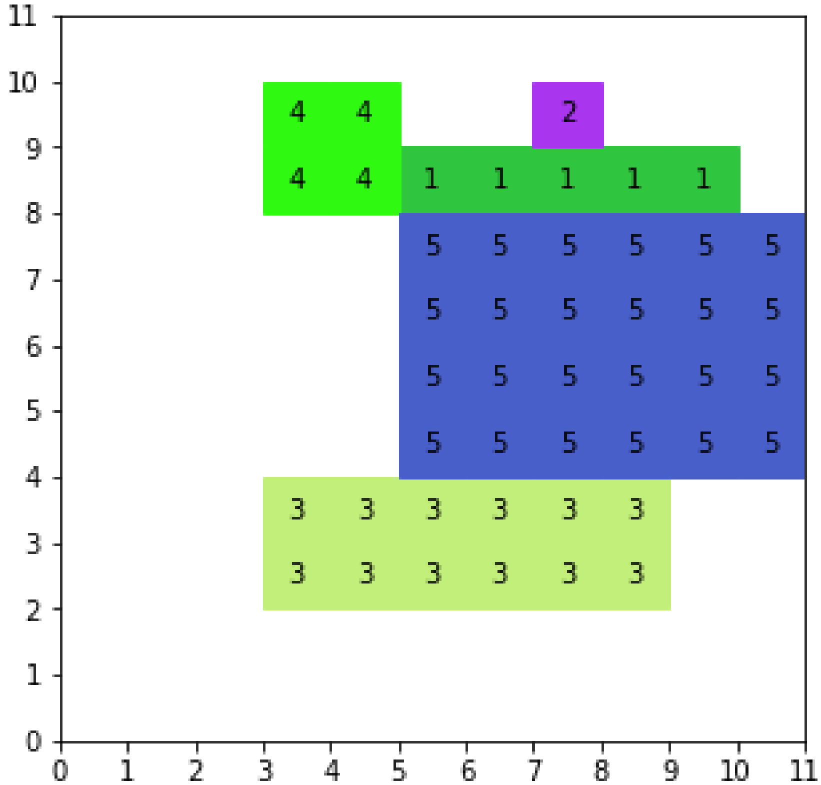
4
4
4
4
4
4
4
1
1
1
1
2 2 2
2 2 2
2 2 2
2 2 2
5
5
5
3
3
3
0 1 2 3 4 5 6 7 8 9 10
0 1 2 3 4 5 6 7 8 9 10

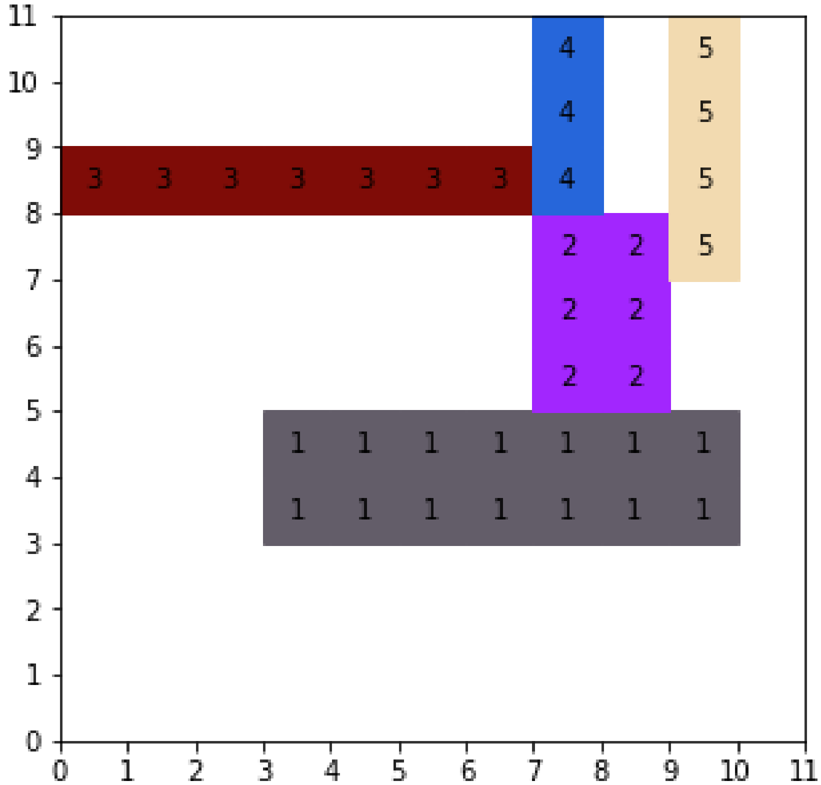


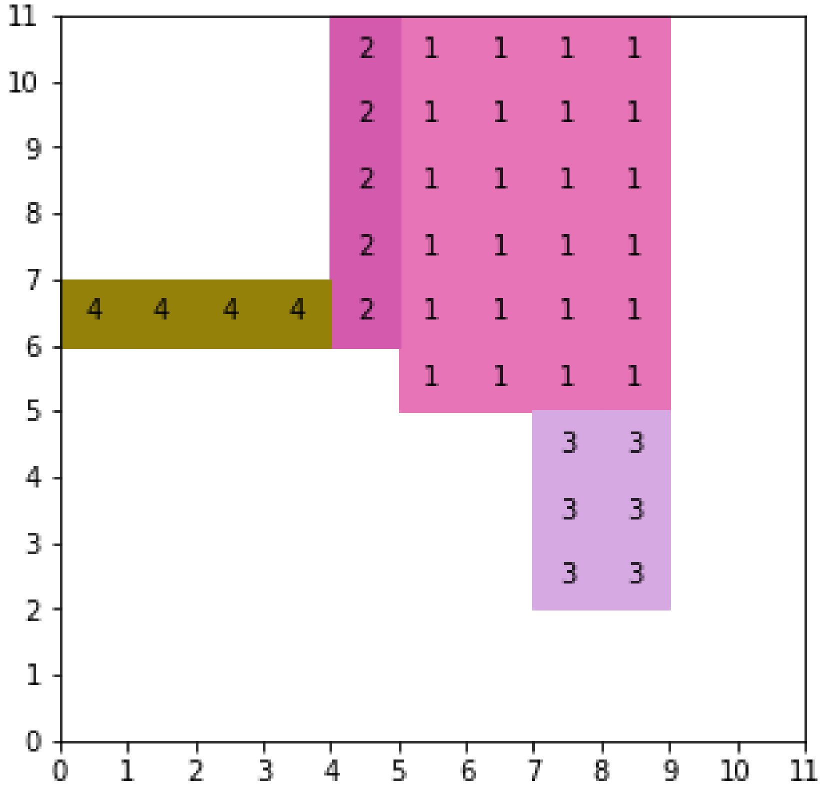
10
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9
10
2 2 2
2 2 2
4 4
1 1 1 1 1 1 4 4
1 1 1 1 1 1 4 4
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
5 5 3


10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10
2 2
2 2
4 4 4 4 4 4
4 4 4 4 4 4
1 1
1 1
1 1
1 1
1 1
1 1
3 3 3
3 3 3
3 3 3
3 3 3
3 3 3

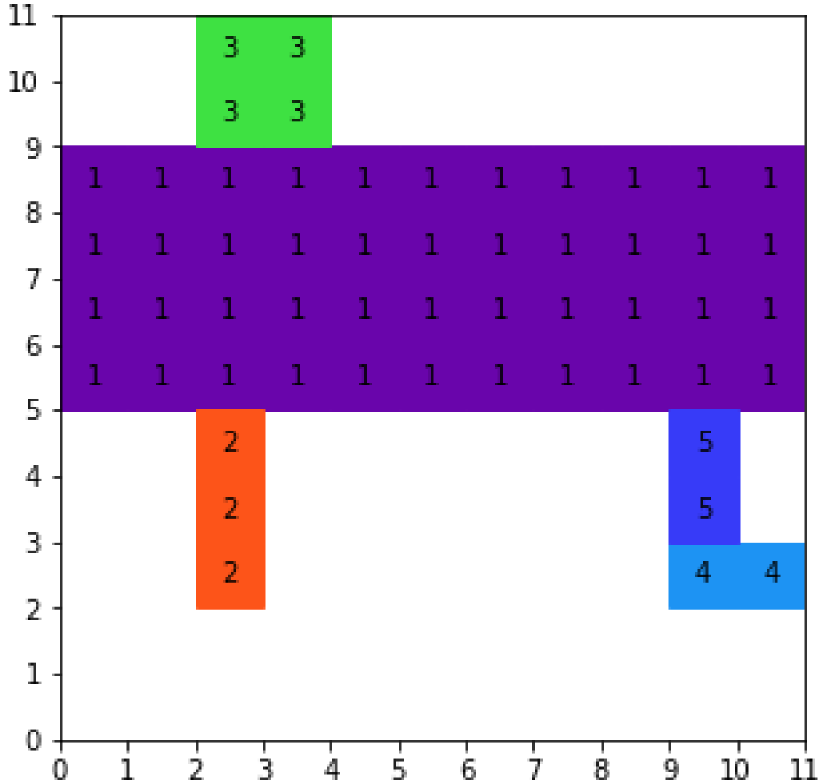

3
3
4 4
1
1
2 2
2 2 5
2 2


4 4 2
4 4 1 1 1 1 1
5 5 5 5 5 5
5 5 5 5 5 5
5 5 5 5 5 5
5 5 5 5 5 5
3 3 3 3 3 3
3 3 3 3 3 3
0 1 2 3 4 5 6 7 8 9 10 11

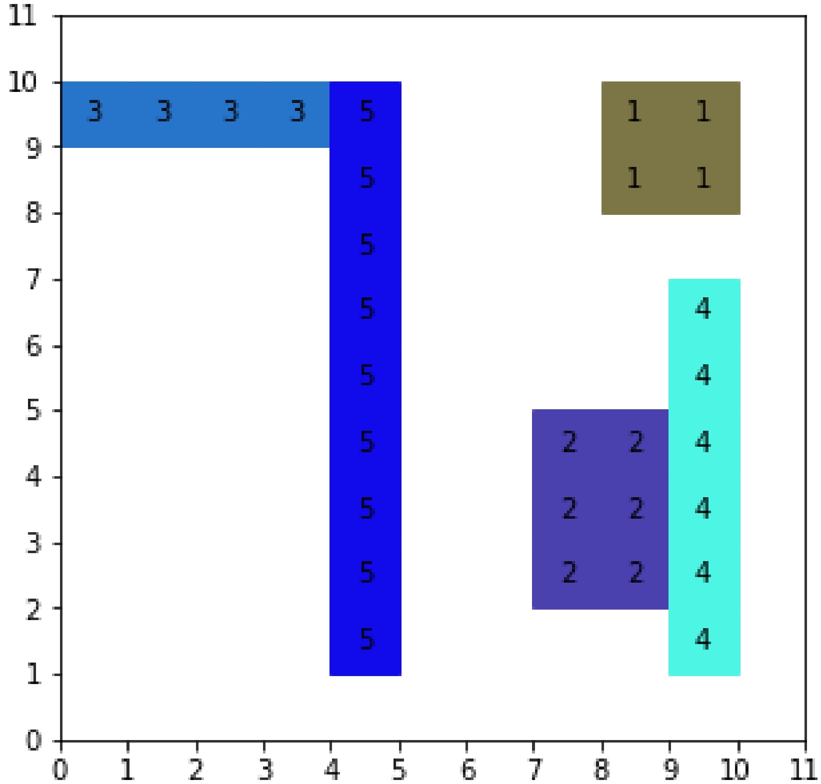

11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11
4 4 4
5 5 5 5
3 3 3 3 3 3 3
2 2 2 2 2 2
1 1 1 1 1 1 1
1 1 1 1 1 1 1



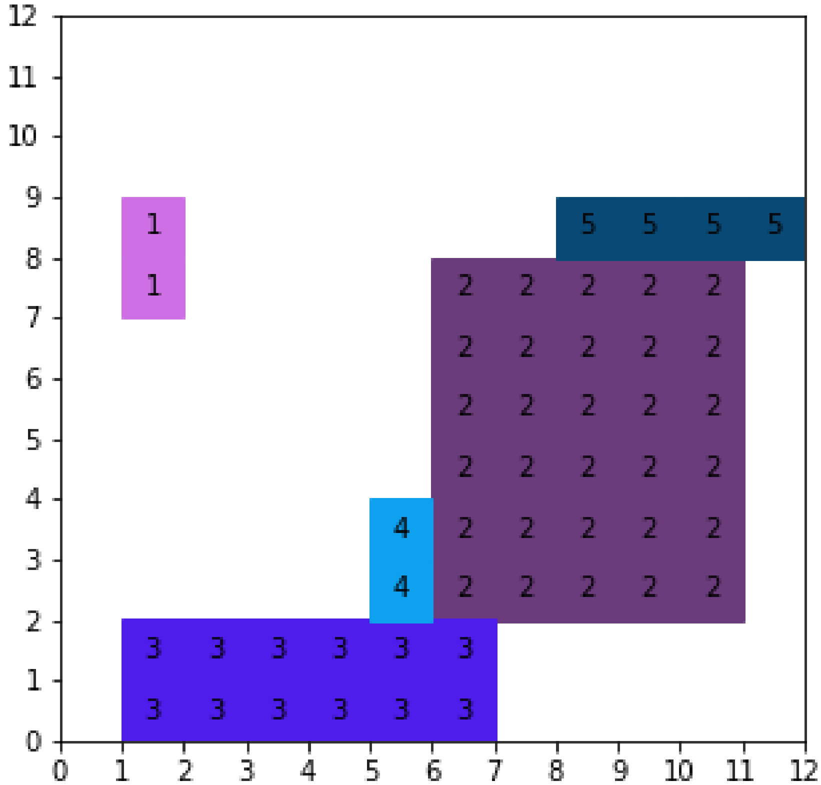
2
1
1
1
1
2
1
1
1
1
2
1
1
1
1
2
1
1
1
1
4
4
4
4
2
1
1
1
1
1
1
1
1
3
3
3
3
3
3



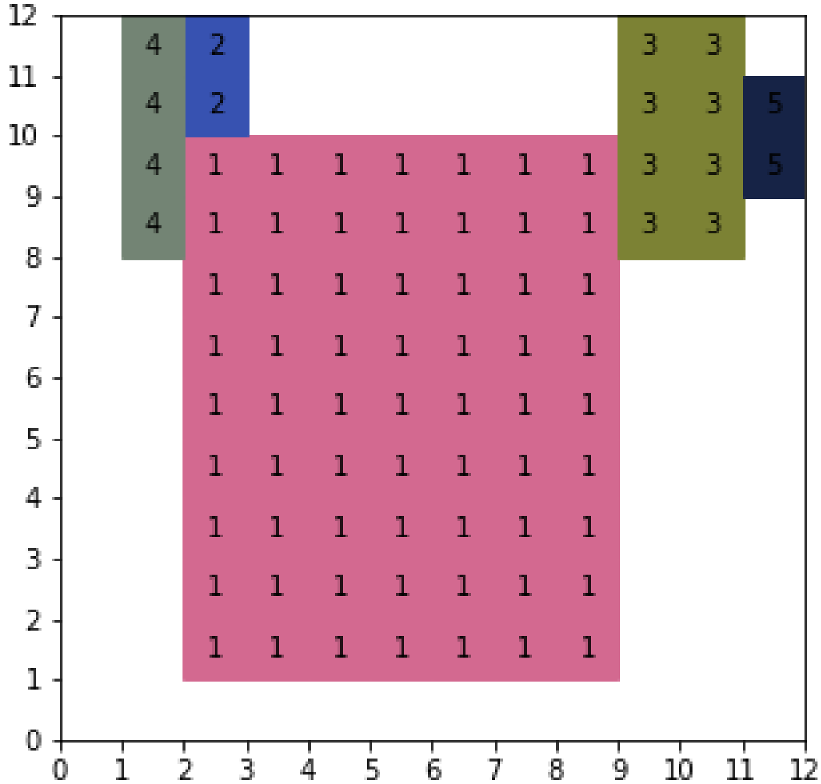
3 3
3 3
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
2
2
2
5
5
4 4
0 1 2 3 4 5 6 7 8 9 10 11



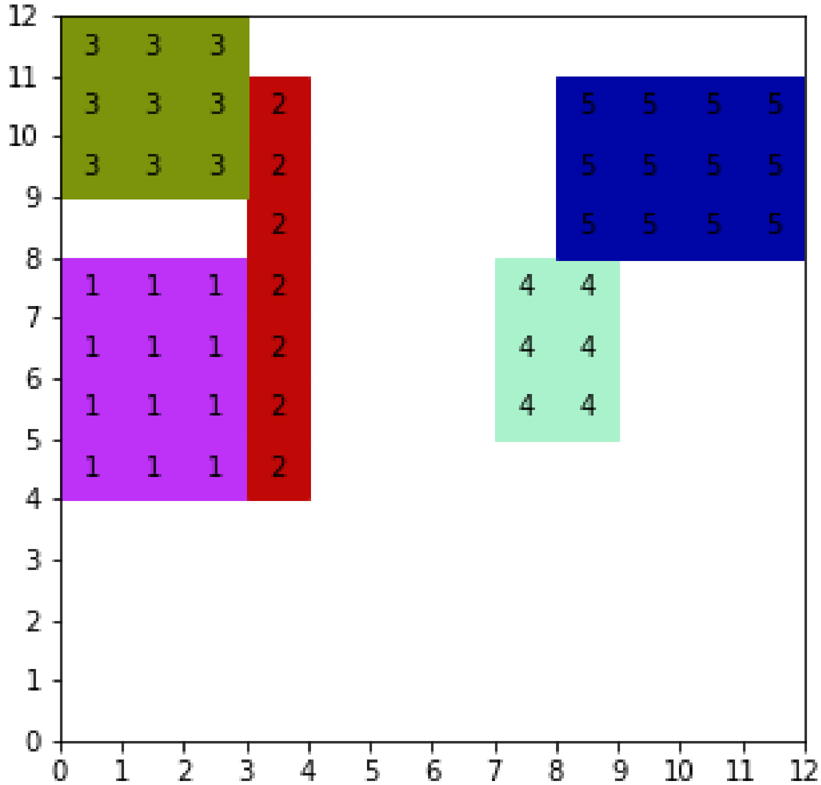
3 3 3 3 5
5
5
5
5
5
5
5
5
1 1
1 1
4
4
2 2 4
2 2 4
2 2 4
4
0 1 2 3 4 5 6 7 8 9 10 11

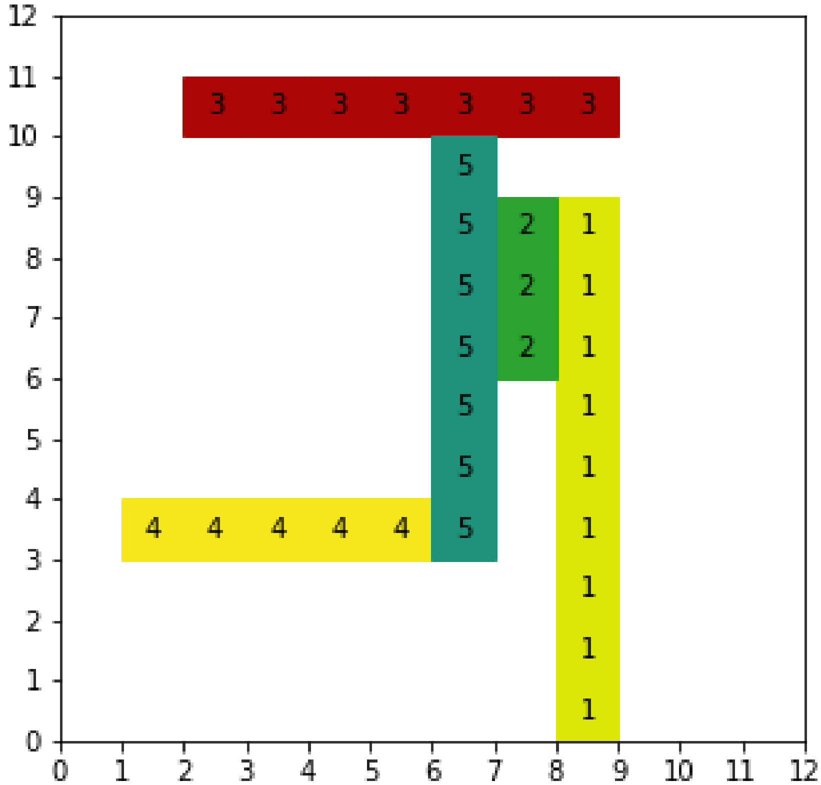


1
1
5 5 5 5
2 2 2 2 2
4
3 3 3 3 3 3
0 1 2 3 4 5 6 7 8 9 10 11 12

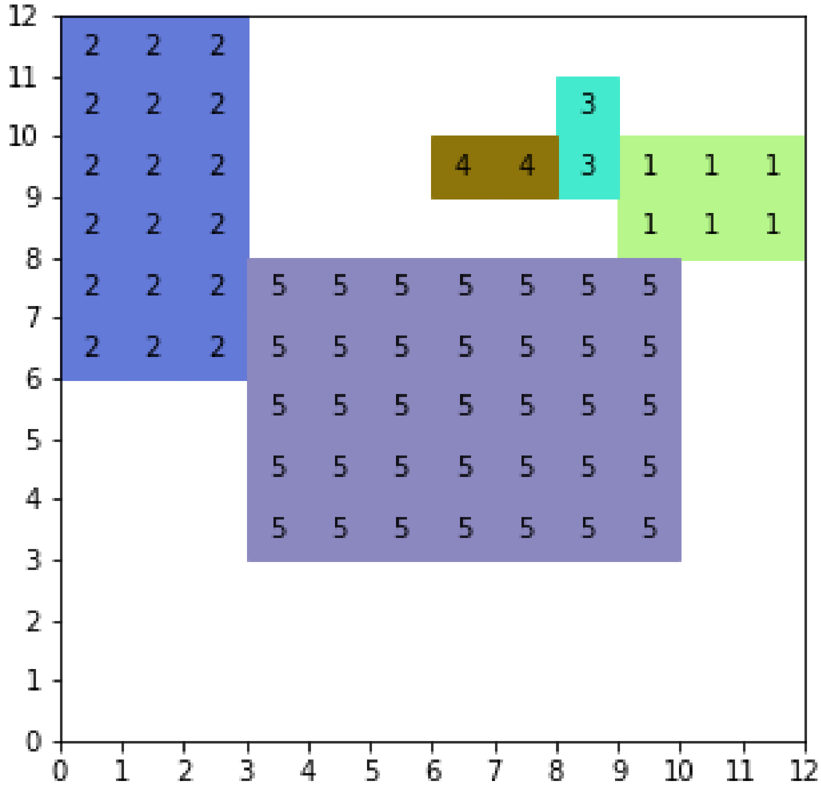


12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12
4 2 3 3
4 2 3 3 5
4 1 1 1 1 1 1 1 3 3 5
4 1 1 1 1 1 1 1 3 3
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1

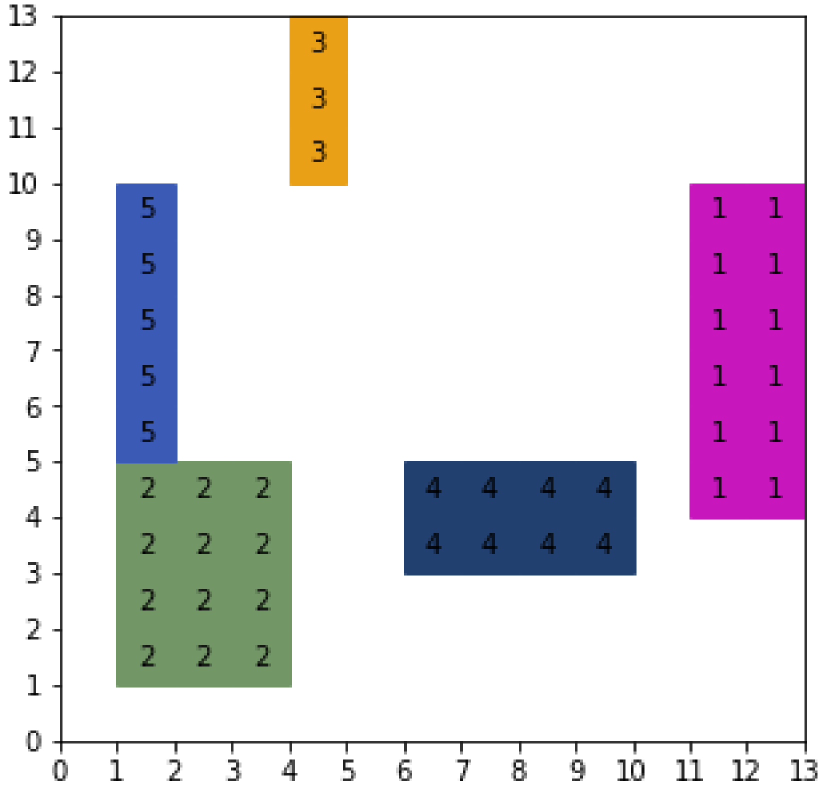

3 3 3
3 3 3
3 3 3
2 2 2 2 2 2 2
1 1 1
1 1 1
1 1 1
1 1 1
4 4
4 4
4 4
5 5 5 5
5 5 5 5
5 5 5 5
0 1 2 3 4 5 6 7 8 9 10 11 12



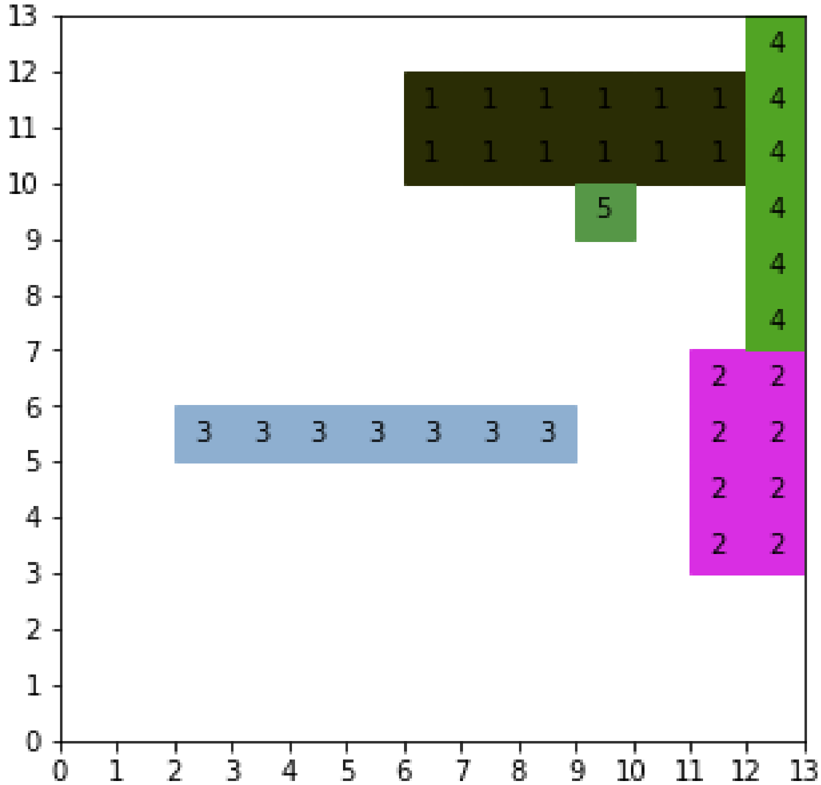
3 3 3 3 3 3 3
5
5 2 1
5 2 1
5 2 1
5 1
5 1
4 4 4 4 4 5 1
1
1
1
0 1 2 3 4 5 6 7 8 9 10 11 12

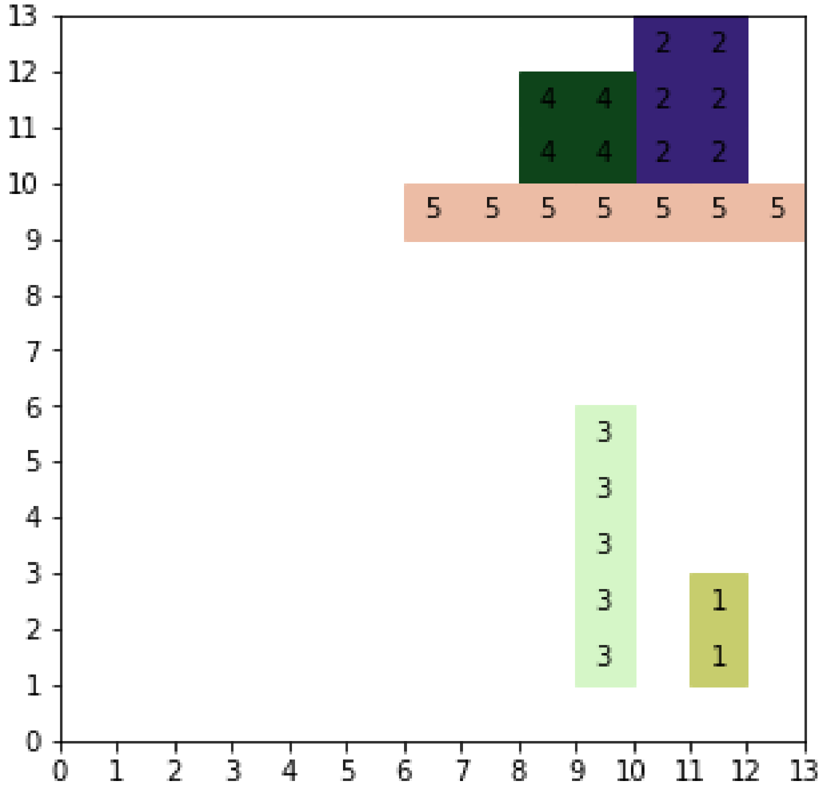


2
3
4
1
5
0
1
2
3
4
5
6
7
8
9
10
11
12

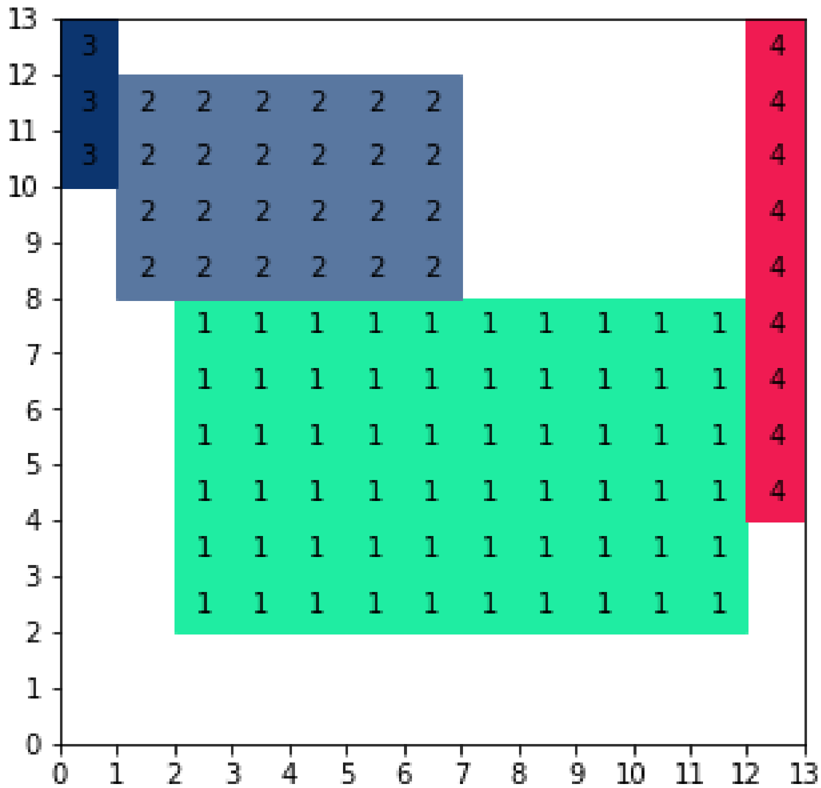


13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13
3 3 3
5 5 5 5 5
1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2
4 4 4 4 4 4 4 4

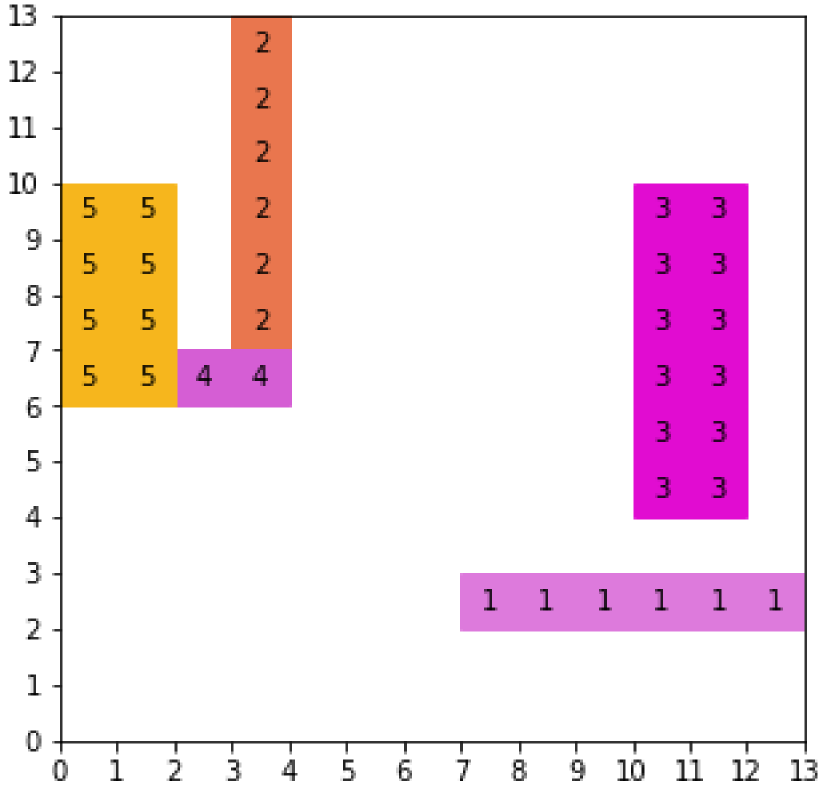

13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13
4
1 1 1 1 1 1 4
1 1 1 1 1 1 4
5 4
4
4
2 2
3 3 3 3 3 3 3 2 2
2 2
2 2

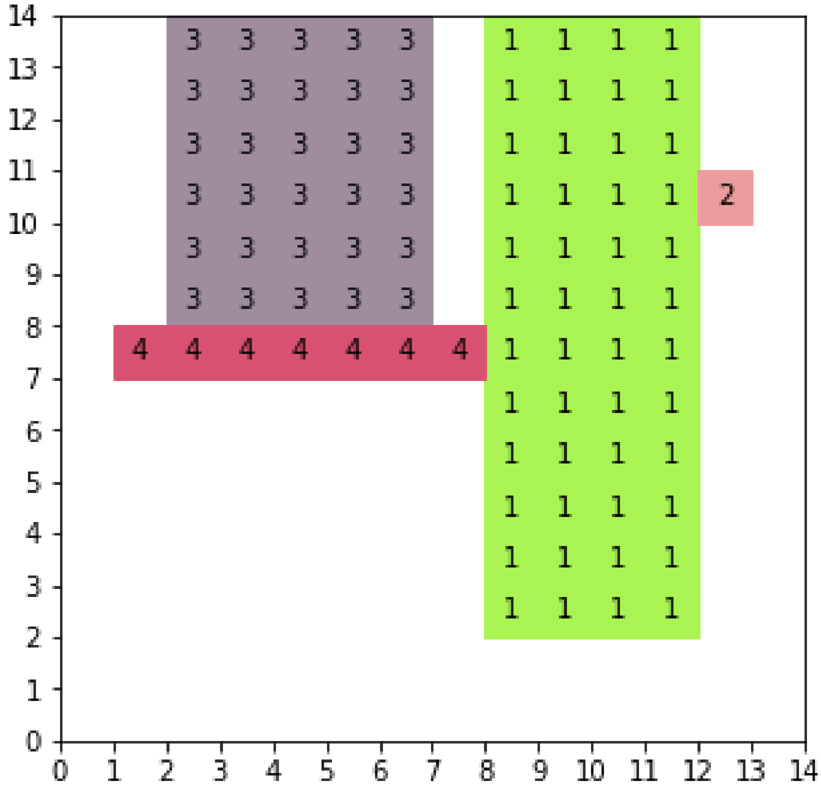


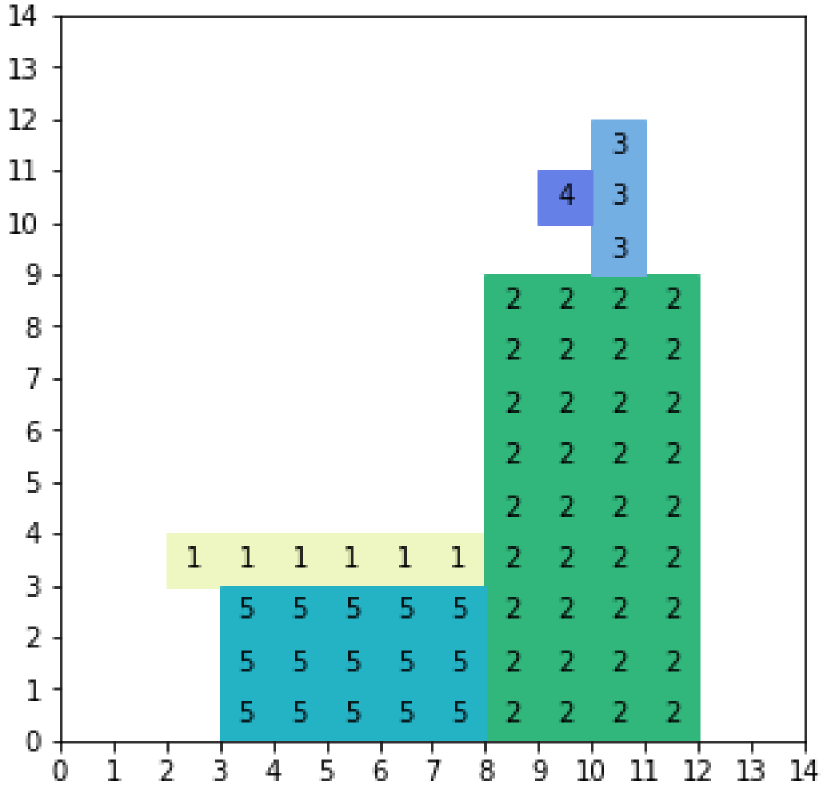
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13
2 2
4 4 2 2
4 4 2 2
5 5 5 5 5 5 5
3
3
3
3 1
3 1

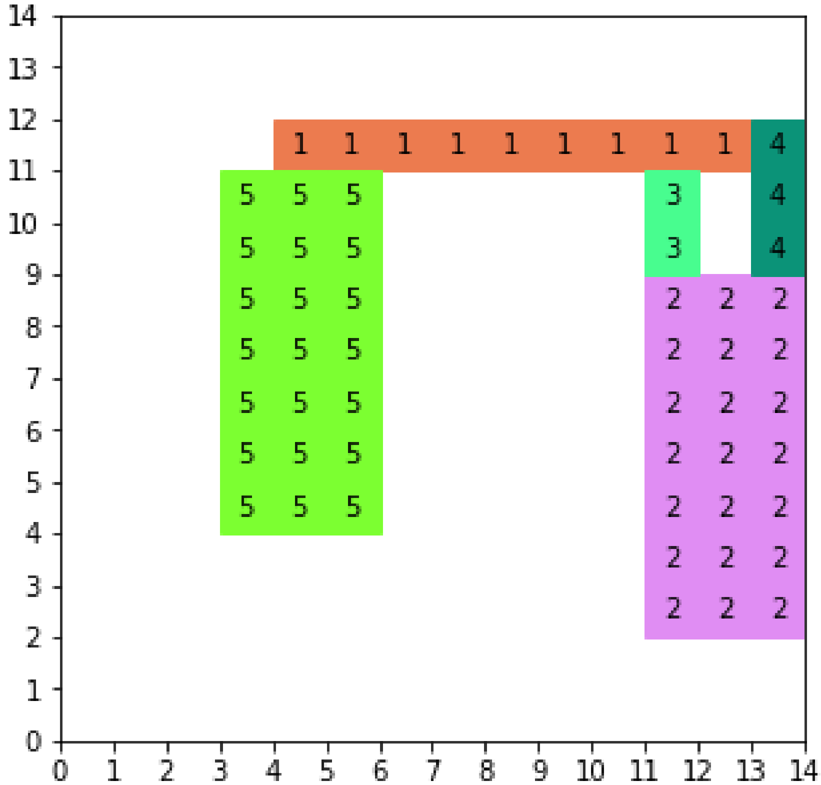


3
3 2 2 2 2 2 2
3 2 2 2 2 2 2
2 2 2 2 2 2
2 2 2 2 2 2
1 1 1 1 1 1 1 1 1 1
4
0 1 2 3 4 5 6 7 8 9 10 11 12 13

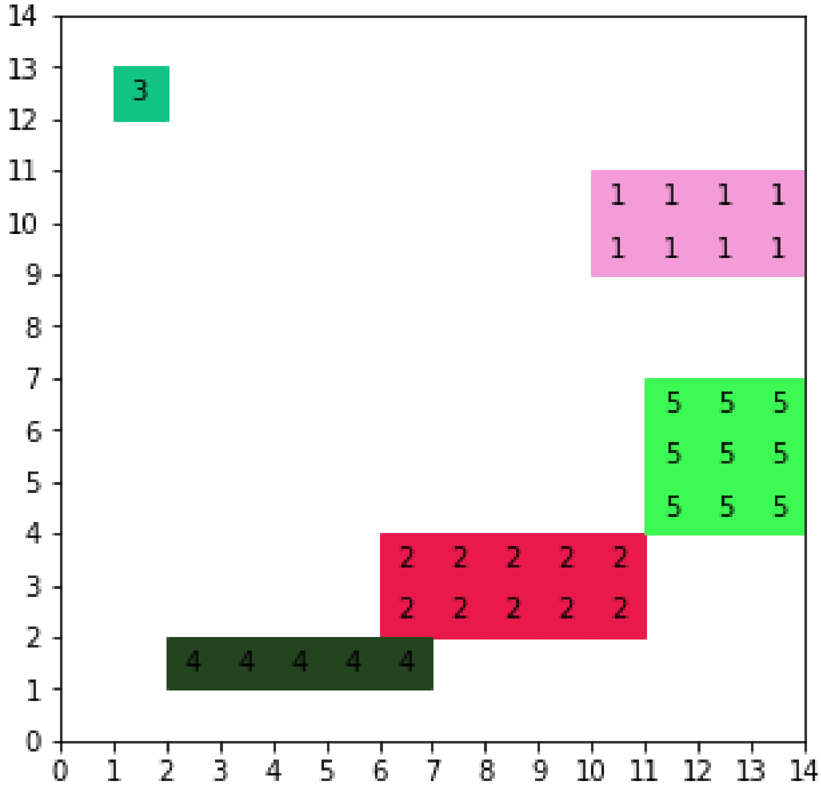

5 5
5 5
5 5
5 5
4 4
2 2 2 2 2 2
3 3
3 3
3 3
3 3
3 3
3 3
1 1 1 1 1 1
0 1 2 3 4 5 6 7 8 9 10 11 12 13



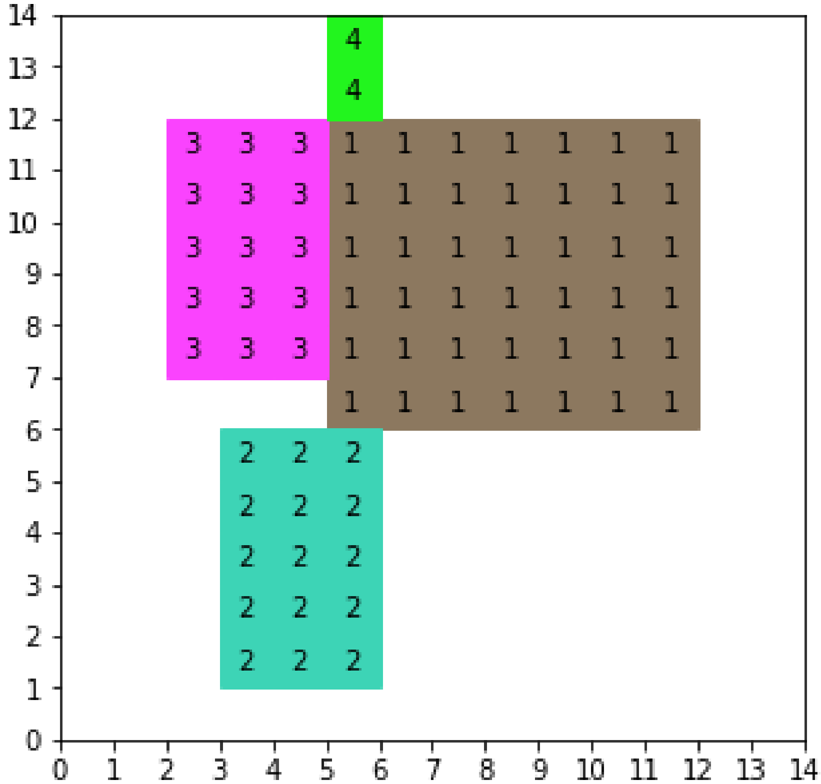

14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
3 3 3
4
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 1
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14



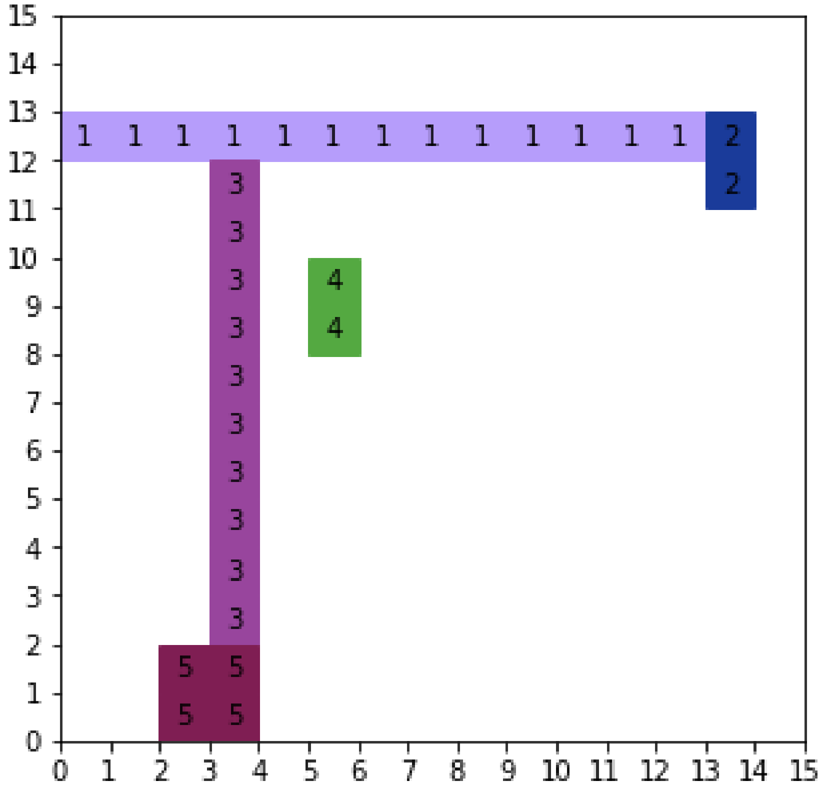
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
1 1 1 1 1 1 1 1 1 4
5 5 5 3 4
5 5 5 3 4
5 5 5 2 2 2
5 5 5 2 2 2
5 5 5 2 2 2
5 5 5 2 2 2
5 5 5 2 2 2
2 2 2
2 2 2

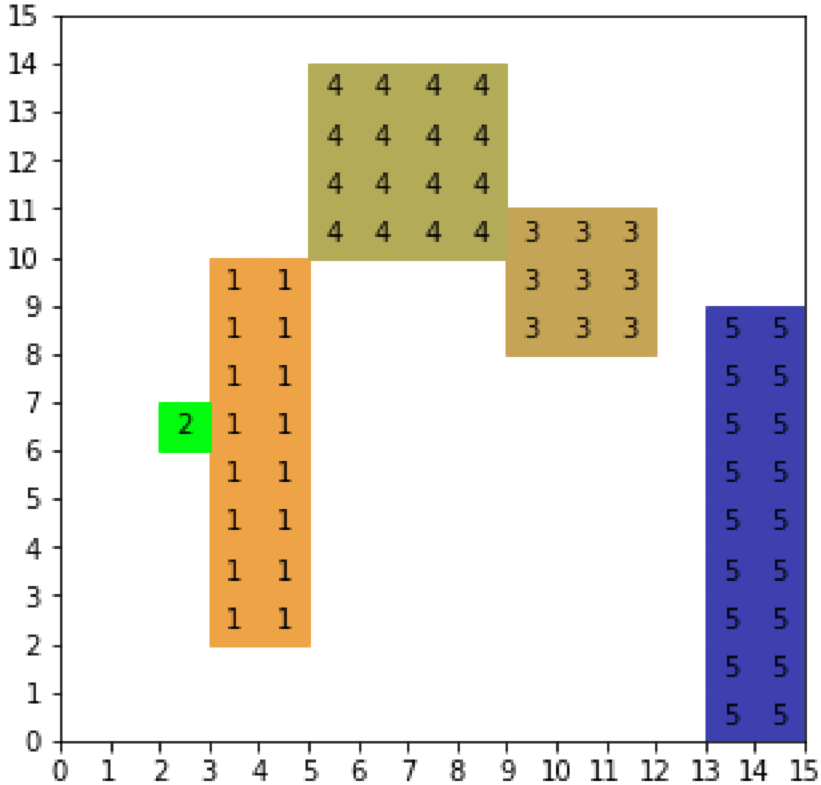


14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
3
1 1 1 1
1 1 1 1
5 5 5
5 5 5
5 5 5
2 2 2 2 2
2 2 2 2 2
4 4 4 4 4

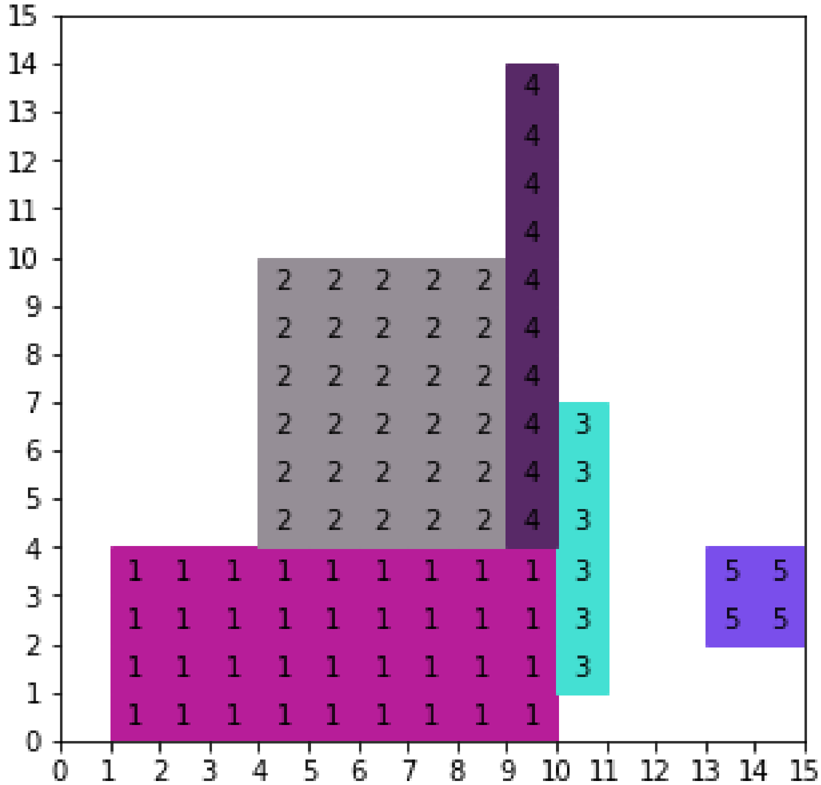


4
4
3 3 3 1 1 1 1 1 1 1
3 3 3 1 1 1 1 1 1 1
3 3 3 1 1 1 1 1 1 1
3 3 3 1 1 1 1 1 1 1
3 3 3 1 1 1 1 1 1 1
1 1 1 1 1 1 1
2 2 2
2 2 2
2 2 2
2 2 2
2 2 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

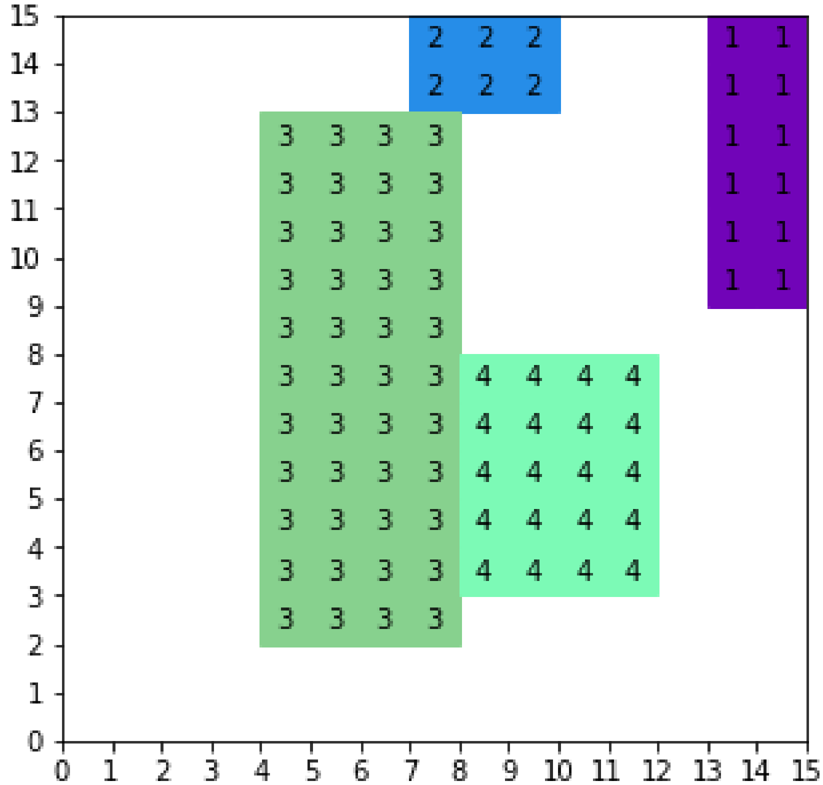

1 1 1 1 1 1 1 1 1 1 1 1 1
2
2
3
3
3
3
3
3
3
3
3
3
4
4
5 5
5 5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

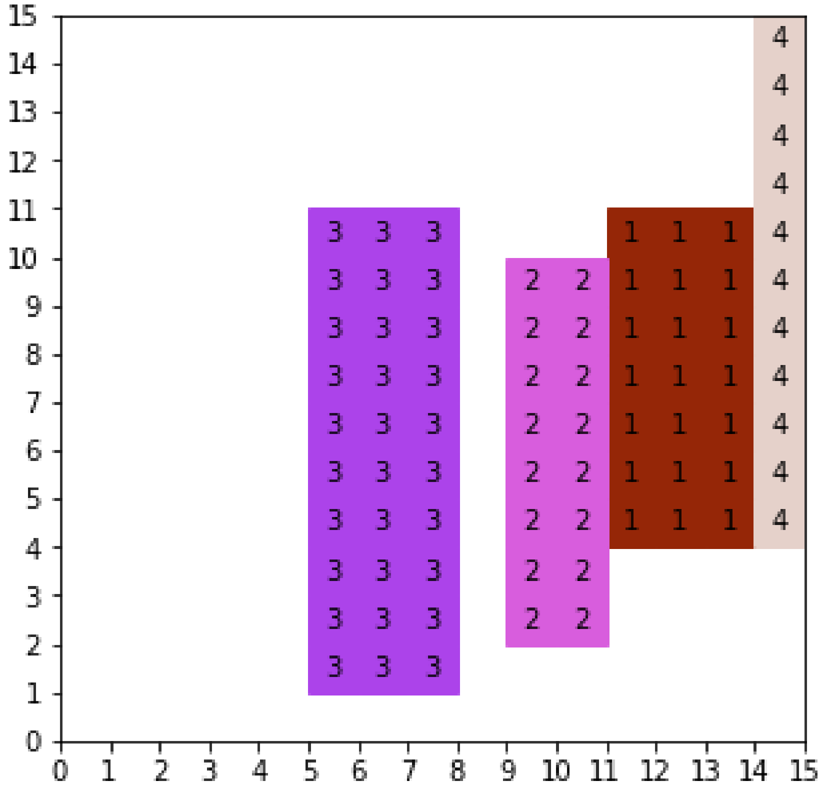


4 4 4 4
4 4 4 4
4 4 4 4
4 4 4 4
3 3 3
3 3 3
3 3 3
1 1
1 1
1 1
2
1 1
1 1
1 1
1 1
1 1
5 5
5 5
5 5
5 5
5 5
5 5
5 5
5 5
5 5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

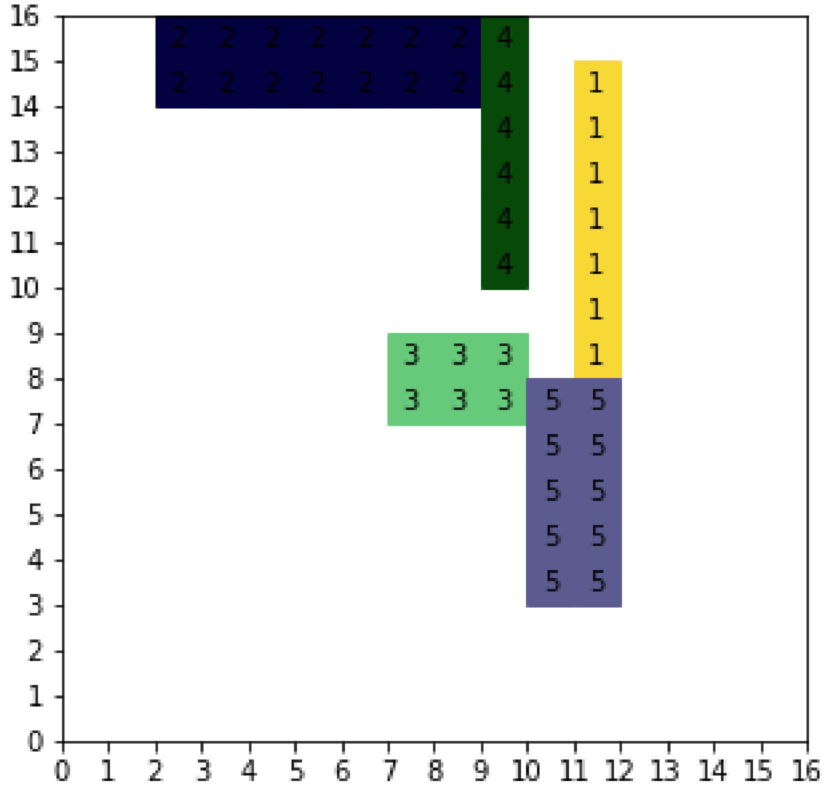


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

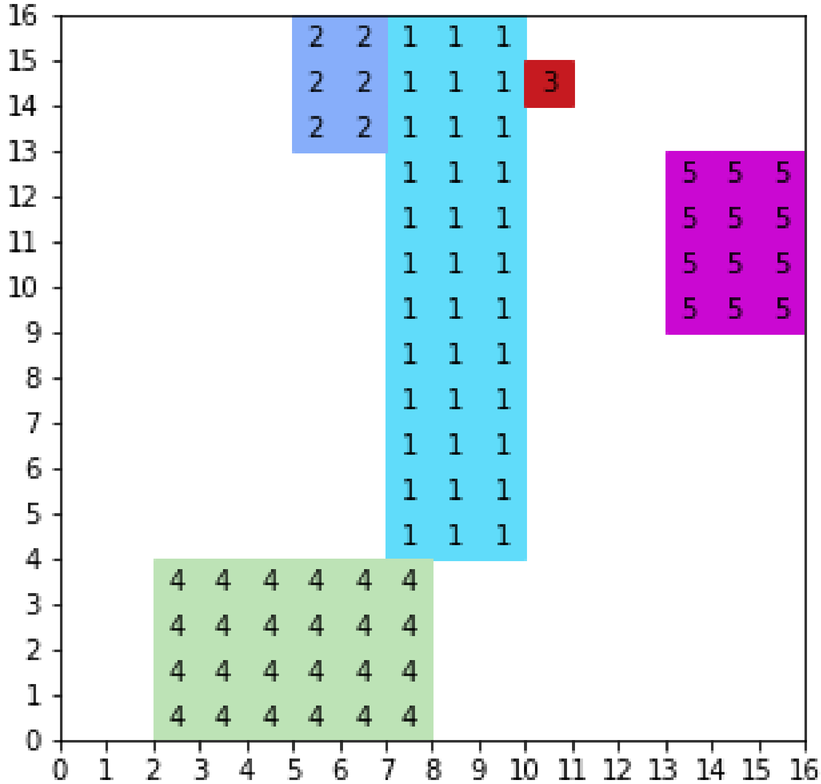

2 2 2
2 2 2
1 1
1 1
1 1
1 1
1 1
1 1
3 3 3 3
4 4 4 4
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15



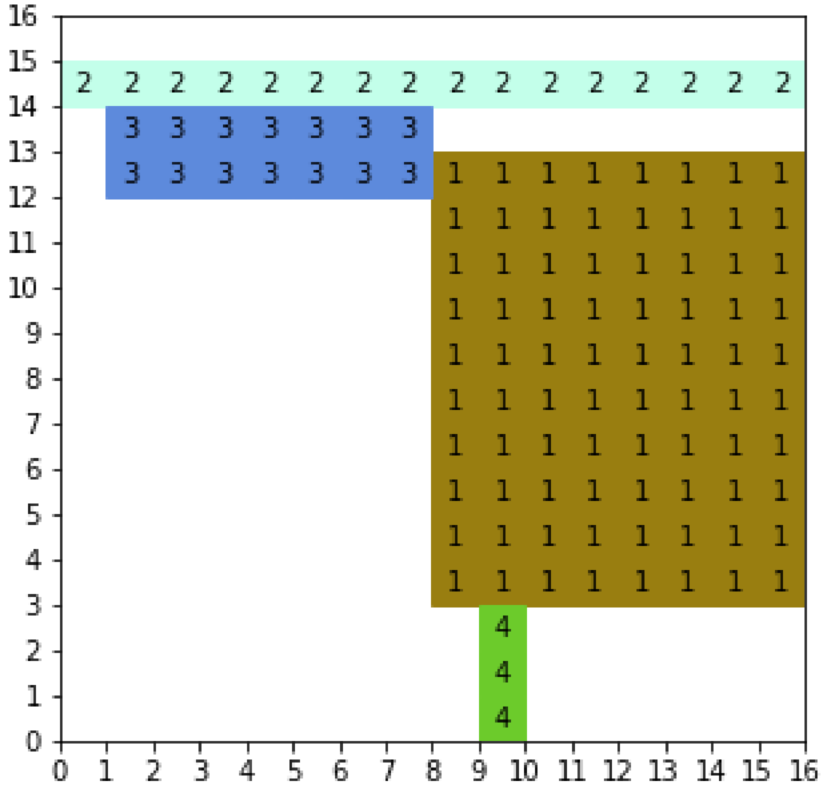
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
3 3 3
2 2
1 1 1
4



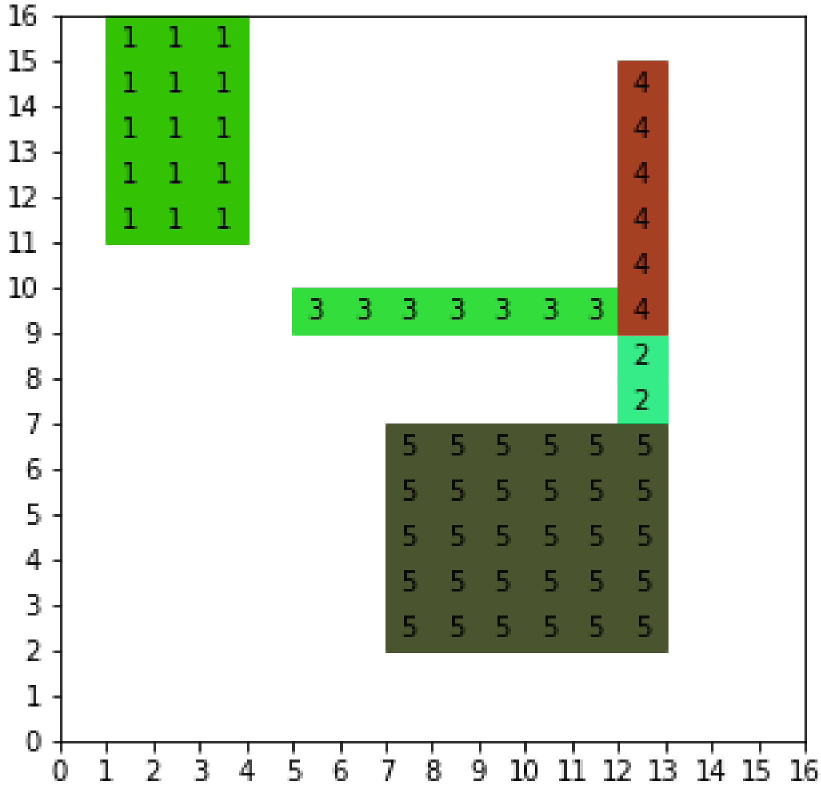
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
2 2 2 2 2 2 2 4
2 2 2 2 2 2 2 4
1 1 1 1 1 1 1
4 4 4 4
3 3 3
3 3 3 5 5
5 5
5 5
5 5
5 5

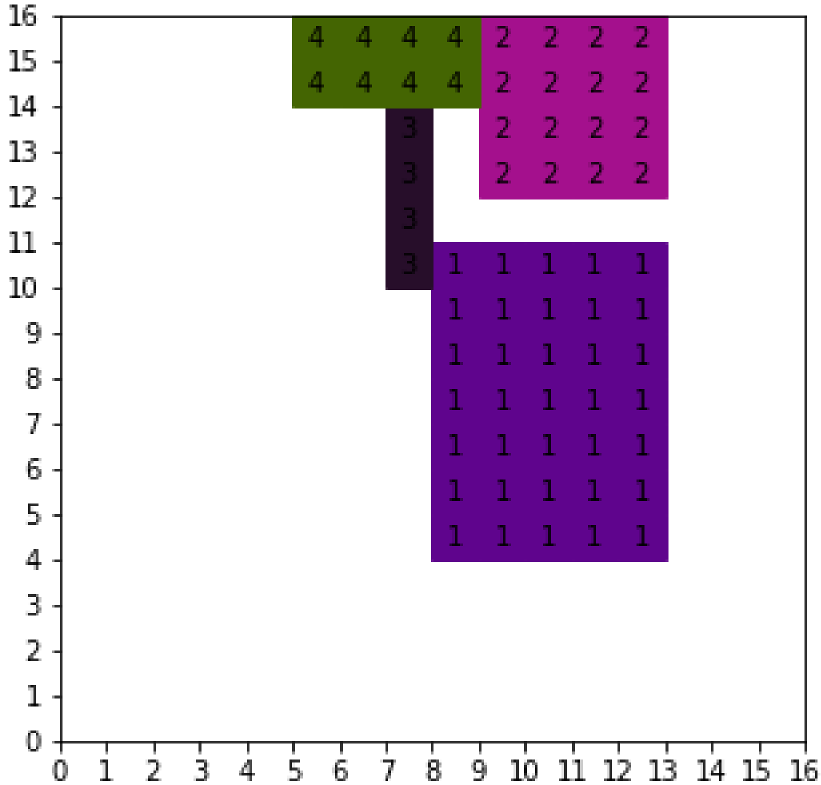


16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

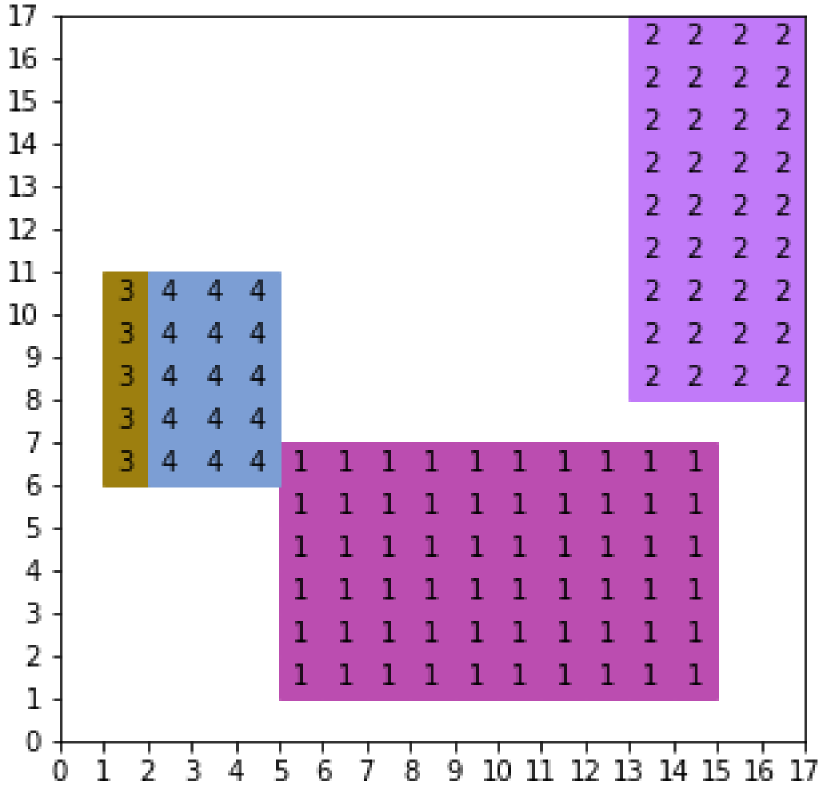


16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3
3 3 3 3 3 3 3 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
4
4
4

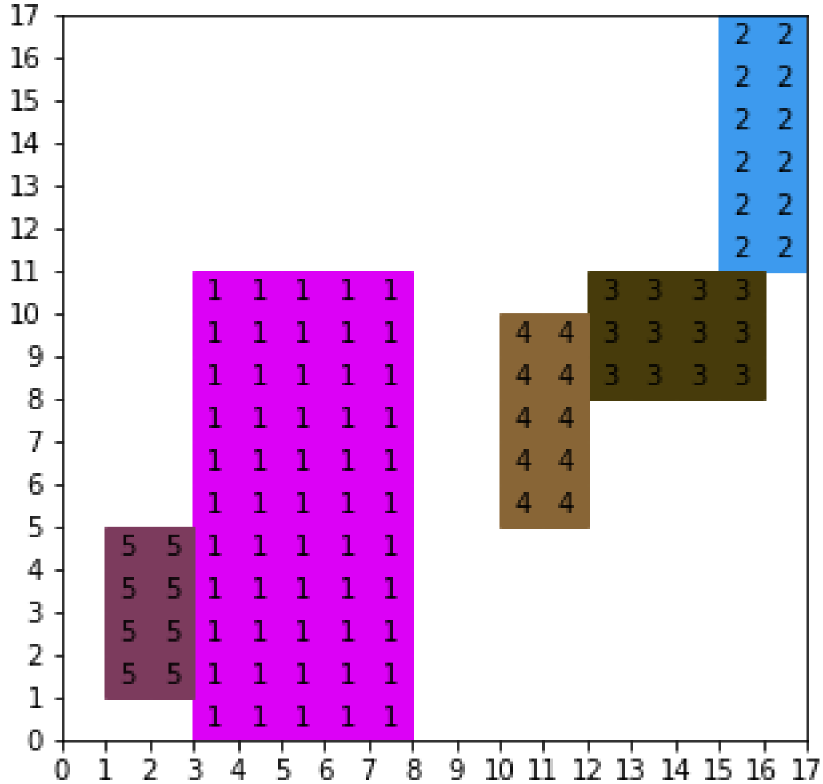

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
1 1 1
3 3 3 3 3 3 3
4
2
5 5 5 5 5 5



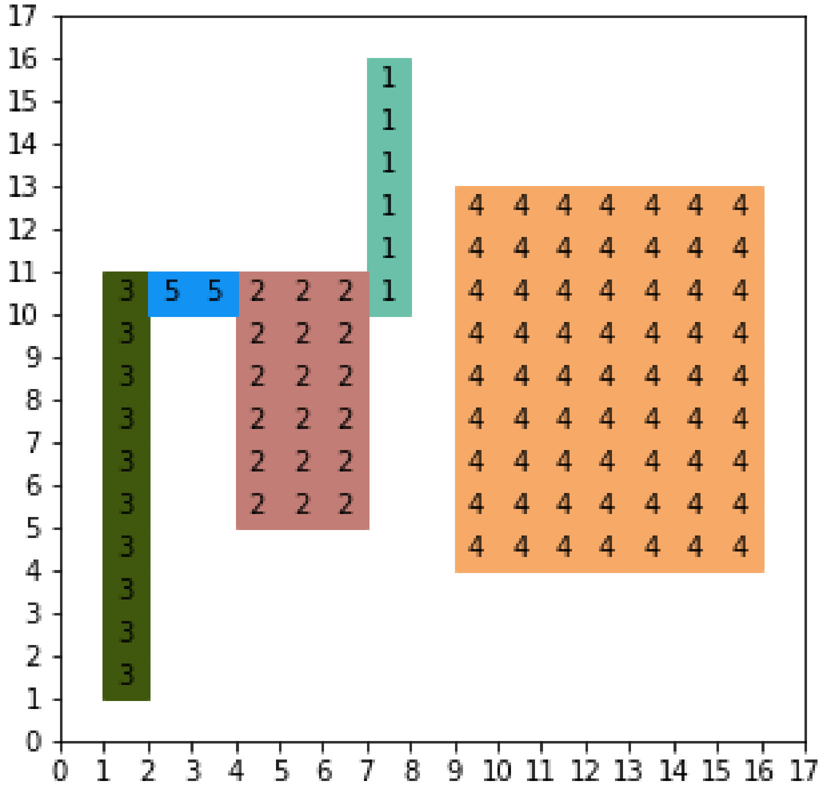
4 4 4 4
4 4 4 4
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
3
3
3
3
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

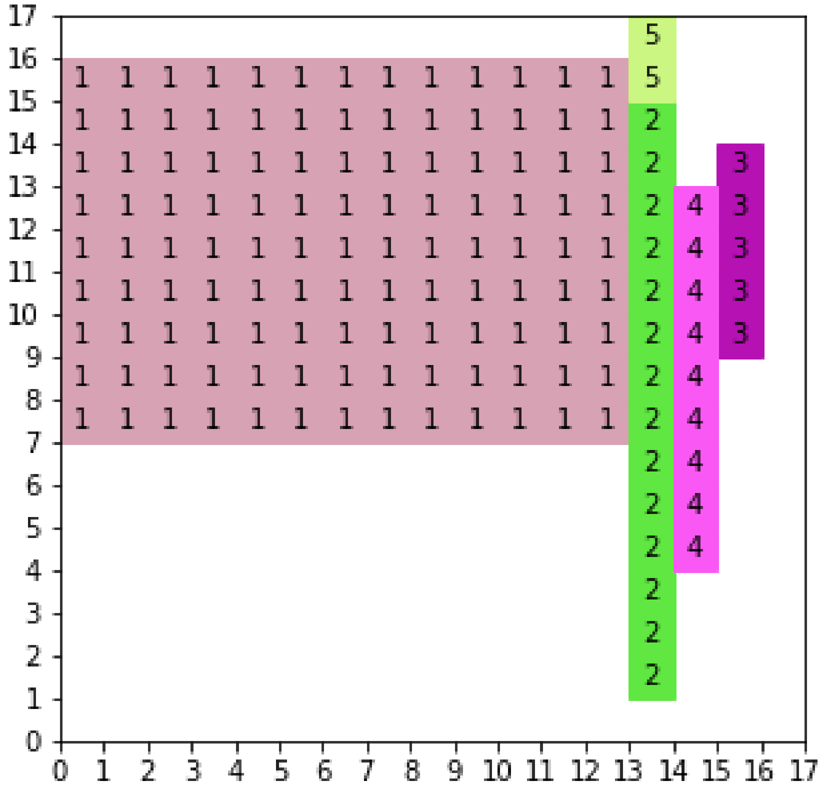


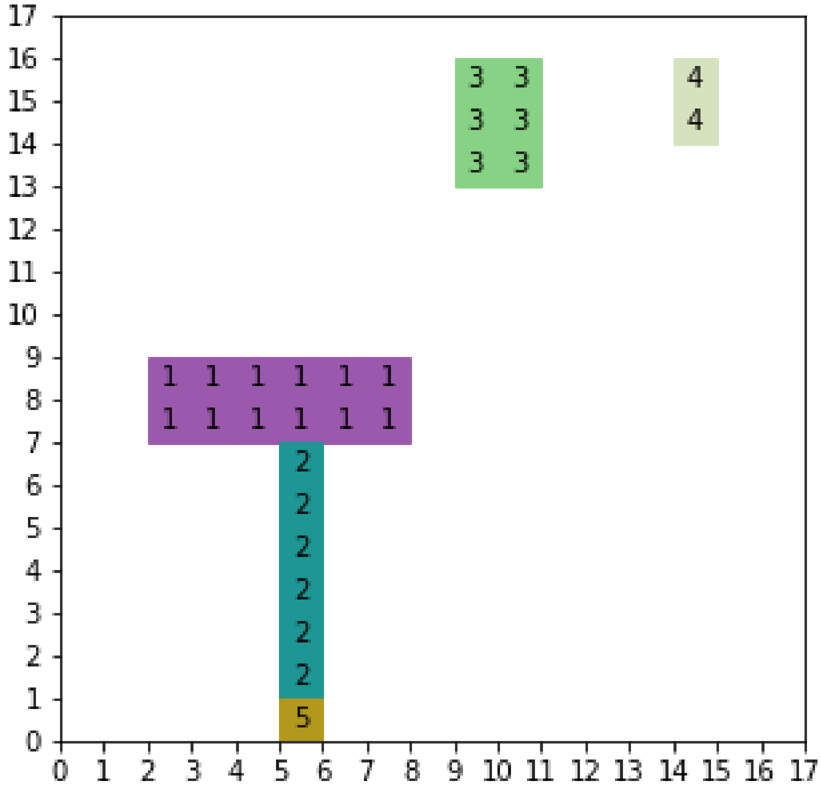
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17



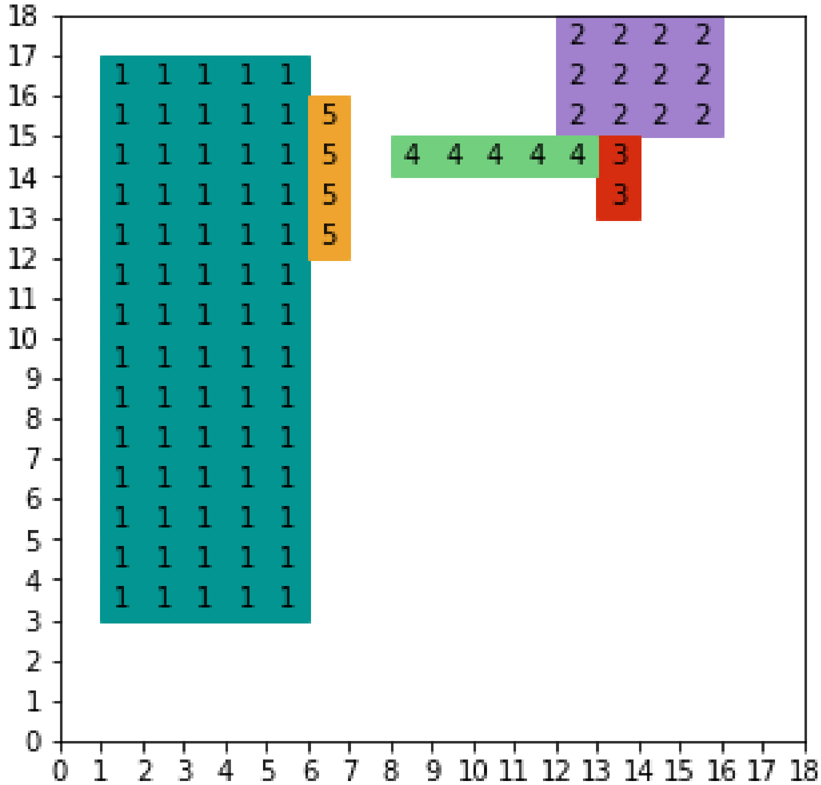
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17


17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
1
1
1
1
1
1
3 5 5 2 2 2
2 2 2
4 4 4 4 4 4 4
3

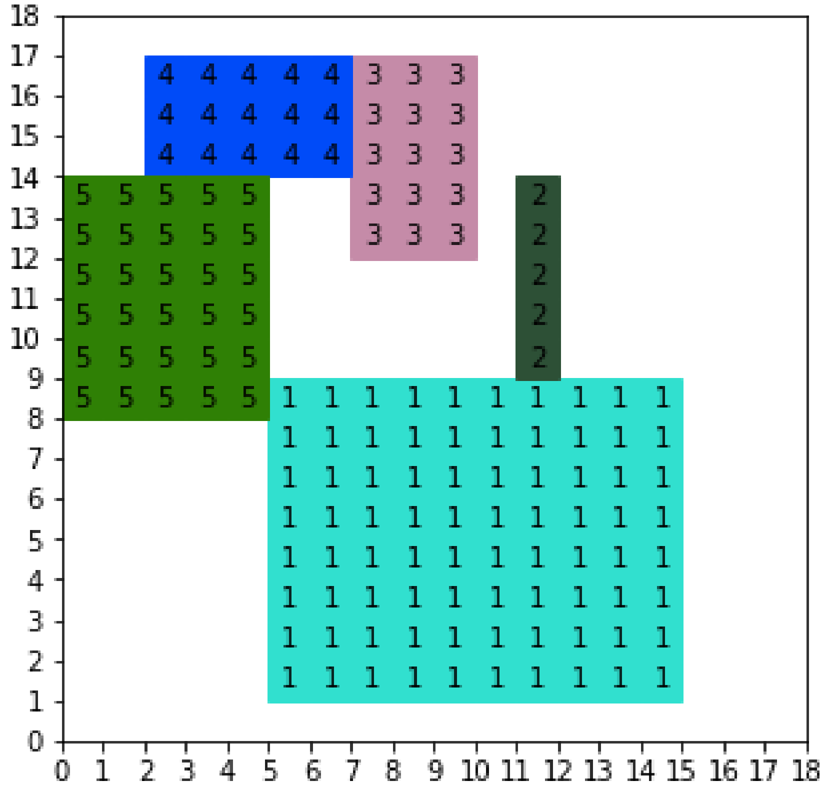


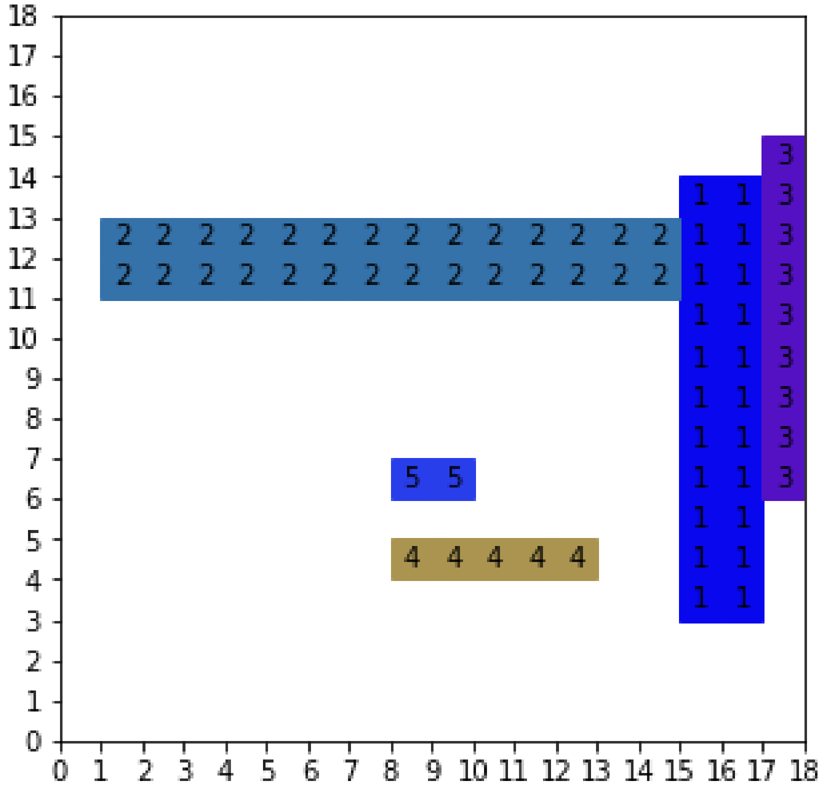


17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
3 3
3 3
3 3
4
4
1 1 1 1 1 1
1 1 1 1 1 1
2
2
2
2
2
2
5

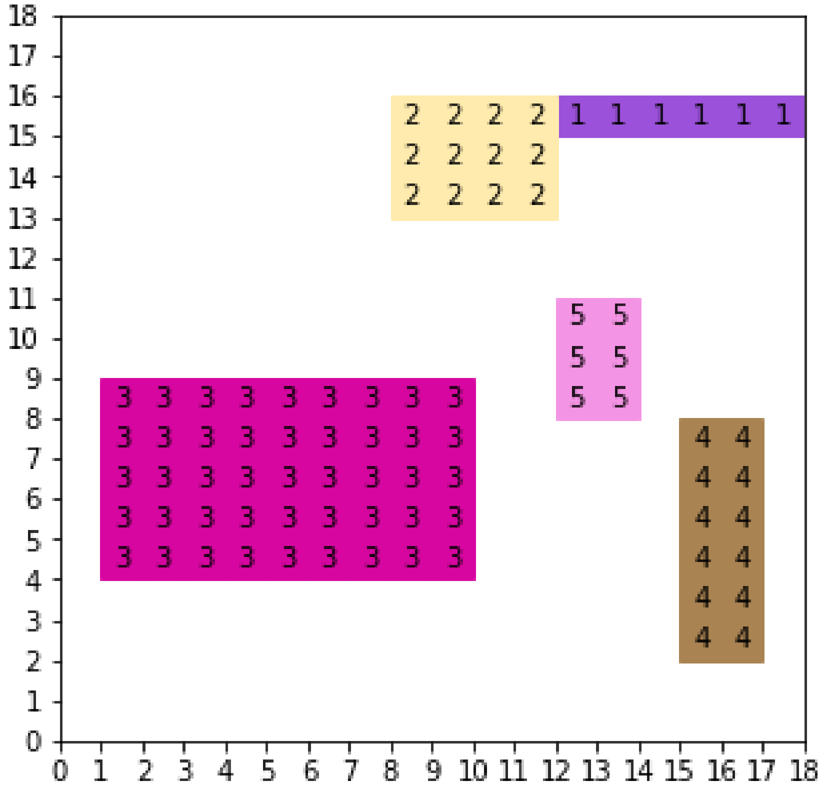


18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

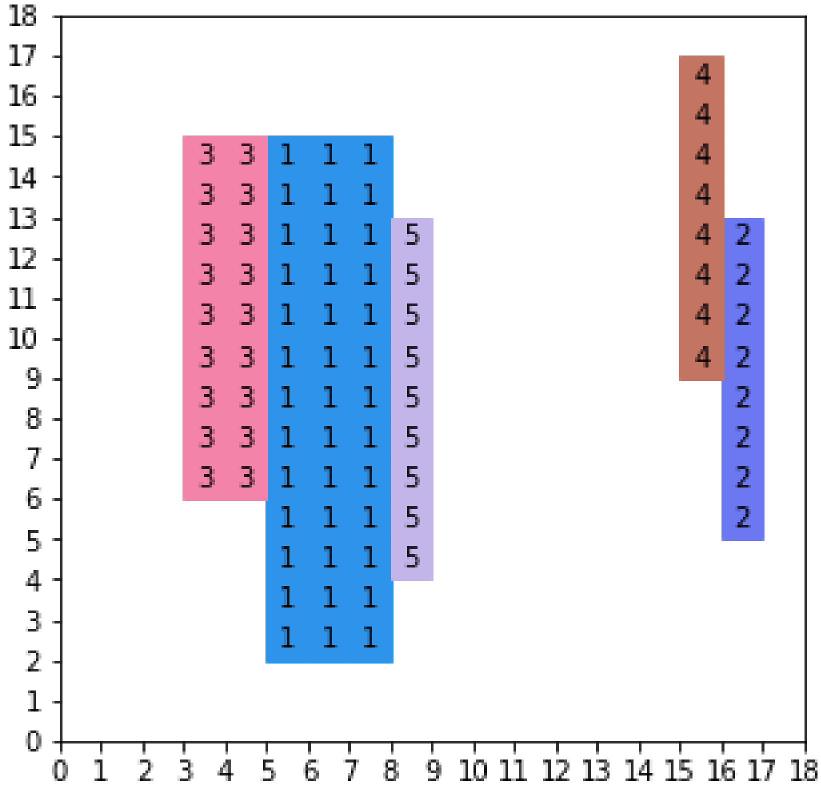


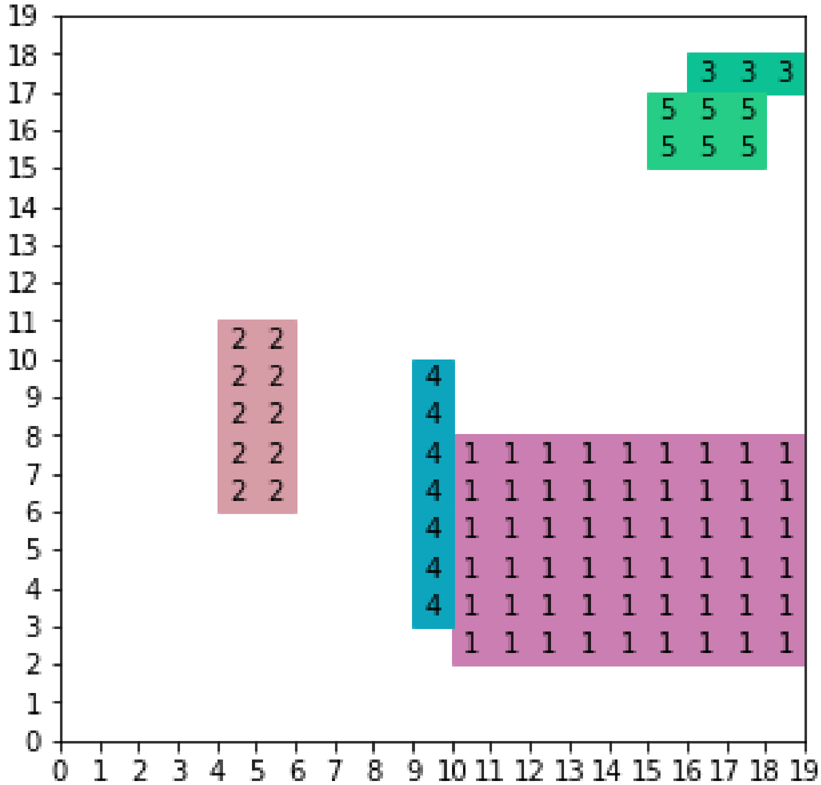


18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2
1 1
3
5 5
4 4 4 4 4

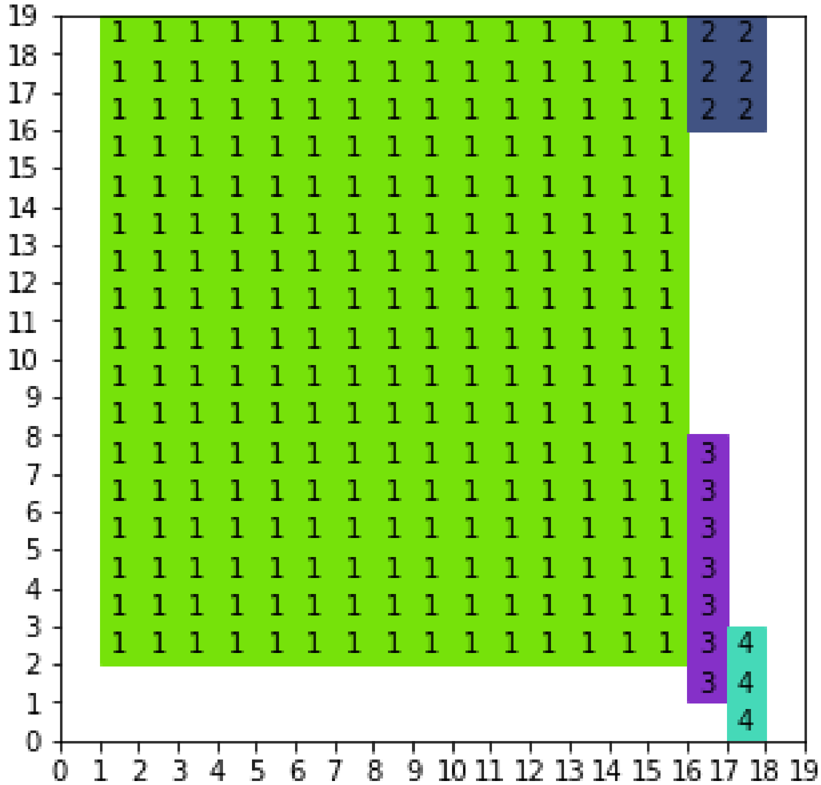


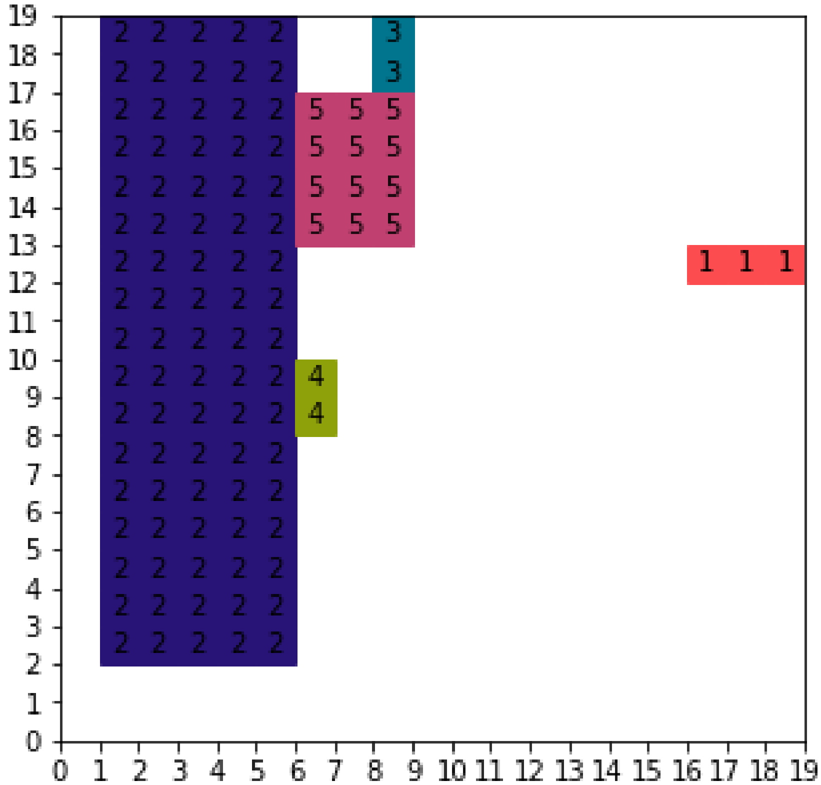
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
2 2 2 2
2 2 2 2
2 2 2 2
1 1 1 1 1 1
5 5
5 5
5 5
3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3
4 4
4 4
4 4
4 4
4 4
4 4

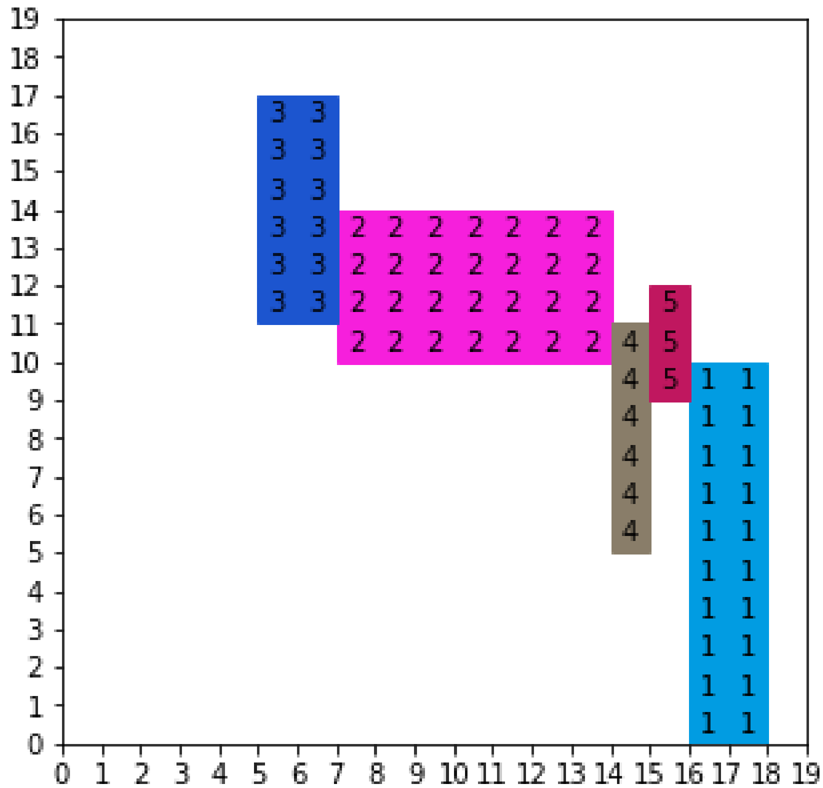


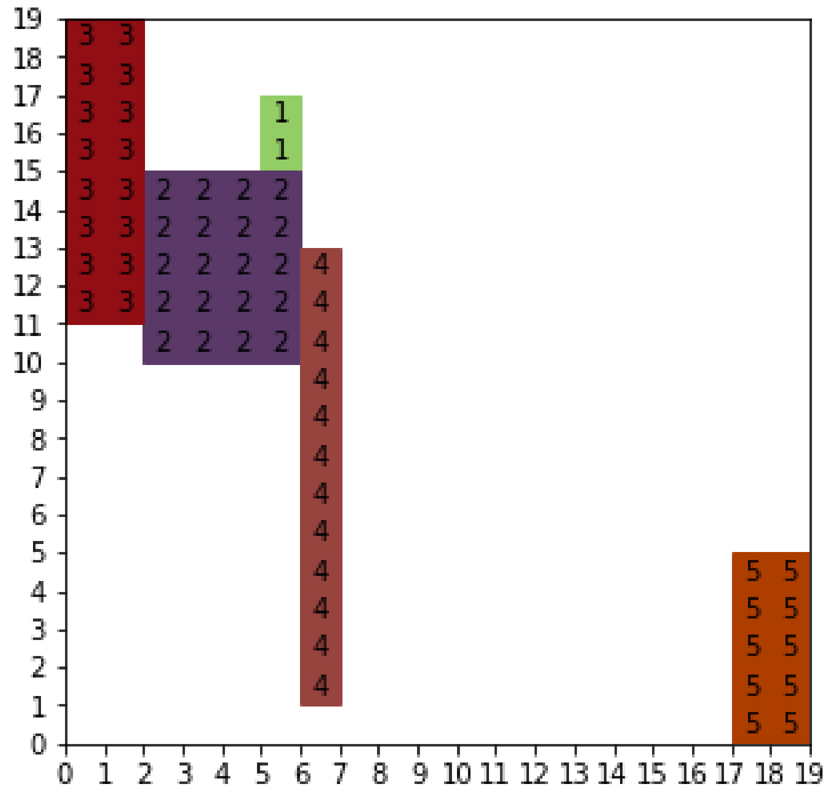


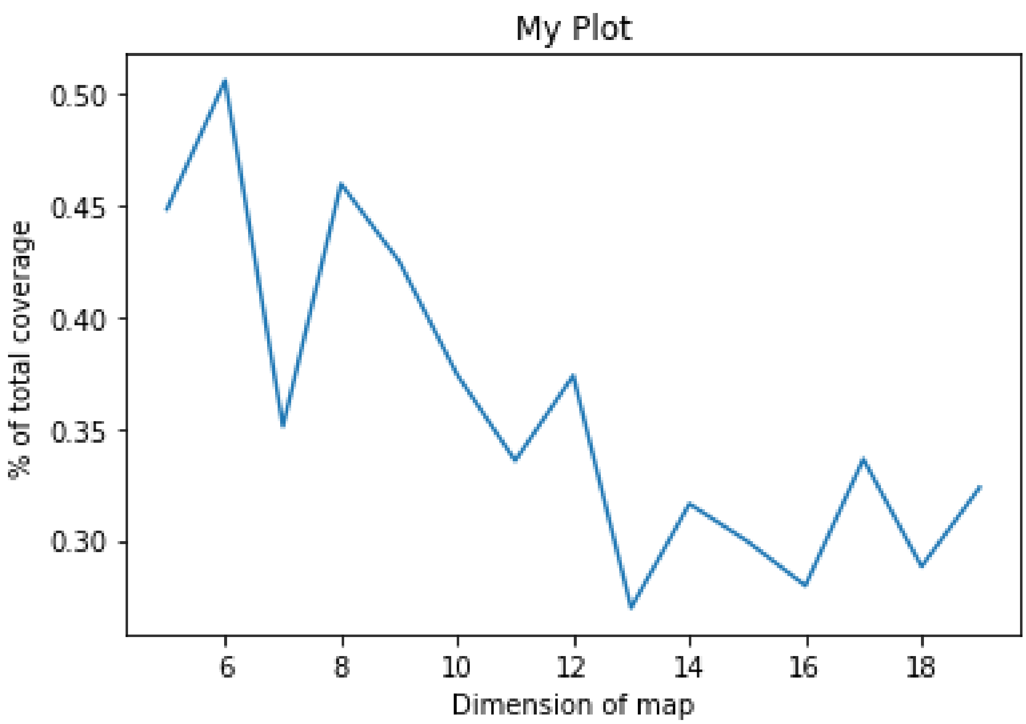
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
3 3 3
5 5 5
5 5 5
2 2
2 2
2 2
2 2
2 2
4
4
4 1 1 1 1 1 1 1 1 1
4 1 1 1 1 1 1 1 1 1
4 1 1 1 1 1 1 1 1 1
4 1 1 1 1 1 1 1 1 1
4 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1








3
2
1
4
5


My Plot
% of total coverage
Dimension of map