划分石料递归算法

孙嘉玺

22920162204038

2018 年 3 月 10 日

目录

1 题目描述

给定一块宽度为 W 的石料，石料的高度为 H, 现要从板上分别切出 n 个高度为 h_i, 宽度为 W_i 的矩形石砖。切割的规则是石料的高度方向与石砖的高度方向保持一致，同时满足一刀切的约束。问如何切割使得所使用的石板利用率最高？

2 算法设计

2.1 算法思想

这个题目根据要求，应该设计成递归的算法。即设计一个函数 $V(w, h, size)$,w 是当前石料的宽度，h 是当前石料的高度，$size$ 是当前切割的石板大小，如果当前的石料能够切割出这块石板，则继续将切割后的石板

以此方式递归下去，如果不能切割则返回一个空值（切割空值）表示不能切割。当递归结束后，在所有的返回值中，选择最好的切割方案，返回给上一层递归函数。

2.1.1 题目改进

如果完全按照题目要求，可能达不到最大利用的效果，因为实际过程中，石砖的高度方向，并不一定要和石料的高度方向一致，而且对于一个石砖或石料，很难说出那边是他的高，那边是他的宽，所以我设计的算法是考虑，石料和石板的旋转的。又因为石料的旋转和石板的旋转效果相同（石料顺时针旋转 90° 等价于石砖逆时针旋转 90°），所以编程实现过程中仅仅旋转石板即可。当然，如果非要求方向相同仅仅需要更改一行代码即可。

2.2 数据结构

算法的基本思想理解以后，还有一个问题就是如何存储我们得到的答案，首先想到的就是树状的结构在这里对答案结构做一个定义。

```
[
  (usage, nowSize, window, asWhat),
  [],
  []
]
```

答案的结构可以看作一个嵌套的列表（Python），每个列表只有三个值，列表元素 1 是切割的方案有关信息（元组），列表元素 2 是在元素 1 切割方案下的分成的两块材料中的第一块的切割结果，元素 3 与元素 2 意义类似。其实答案的结构也能看成递归。

- usage 当前切割方案下能利用多少石料
- nowSize 递归到这一步时的石料大小
- window 用那一块石板的规格切割石料
- asWhat 沿着高度的方向切割，或者沿着宽度的方向切割 ($True/False$)

3 代码实现

本题目用 Python 代码实现, 因为在 python 下可以很好的作图，直观的呈现切割方案

4 代码

```
# coding: utf-8
```

```
from __future__ import division
import matplotlib.pyplot as plt
import numpy as np
from pprint import pprint


def plotAns(Ans, Map, color):
    window = Ans[0][2]
    if Ans[0][0] == 0 or Ans[0][1][0] == 0 or Ans[0][1][1] == 0:
        return
    for i in range(Ans[0][2][0]):
        for j in range(Ans[0][2][1]):
            Map[i][j] = color
    if Ans[0][0] != 0:
        if Ans[0][3] == AS_WIDTH:
            plotAns(Ans[1], Map[window[0]:,:window[1]],color + 10)
            plotAns(Ans[2], Map[:,window[1]:],color + 5)
        else:
            plotAns(Ans[1], Map[window[0]:,:], color + 10)
            plotAns(Ans[2], Map[:window[0]:,window[1]:], color + 5)


def selectBest(array):
    best = array[0]
    bestSum = best[0][0][0] + best[1][0][0]
    for candidate in array:
        if candidate[0][0][0] + candidate[1][0][0] > bestSum:
            best = candidate
            bestSum = best[0][0][0] + best[1][0][0]
    return best, bestSum



def bestSolve(Width, High, window, asWhat):
    nextSolves = []
    if Width >= window[0] and High >= window[1]:
        nowSolve = (window[0] * window[1], (Width, High), window, asWhat)

        if asWhat == AS_WIDTH:
```

```
            newWidth = [Width - window[0], Width]
            newHigh  = [window[1], High - window[1]]
        else:
            newWidth = [Width - window[0], window[0]]
            newHigh  = [High, High - window[1]]

        for s in Sizes:
            for asNext in AS:
                nextSolves.append([
                    bestSolve(newWidth[0], newHigh[0], s, asNext[0]),
                    bestSolve(newWidth[1], newHigh[1], s, asNext[1])
                ])


        bestNext, nextSum = selectBest(nextSolves)

        # update the nowSolve
        nowSolve = (nowSolve[0] + nextSum, nowSolve[1], nowSolve[2], nowSolve[3])
        nowSolve = [nowSolve, bestNext[0], bestNext[1]]
        return nowSolve
    else:
        nowSolve = [(0, (Width, High), window, asWhat), [], []]
        return nowSolve


if __name__ == "__main__":

    Width = 130
    High = 100

    Sizes = [(54, 29), (70, 50), (30, 41)]
    Sizes.extend([(s[1], s[0]) for s in Sizes]) # 如果不考虑旋转，将该语句注释掉即可，不会影响程序逻辑
    # we need to range the Size, so we don't need to range the Map

    Map = np.ones((Width, High))

    AS_WIDTH = True
    AS_HIGH  = False
    AS = [(True, True), (False, False), (True, False), (False, True)]
```

```
ans = bestSolve(Width, High, (0,0), True)
plotAns(ans,Map,0)
pprint(ans)

plt.imshow(Map)
percent = ans[0][0] / (Width * High) * 100
title = "%.2f" % percent
plt.title(title + "%")
plt.show()
print ans[0][0]/(Width * High) * 100 , "%"
```

5 实验数据 & 实验结果

5.1 石版大小

- 54×29
- 70×50
- 30×41

5.2 石料大小 & 切割方案

5.2.1 100 × 100

结果如图 1

5.2.2 100 × 119

结果如图 2

5.2.3 110 × 100

结果如图 3

5.2.4 130 × 100

结果如图 4

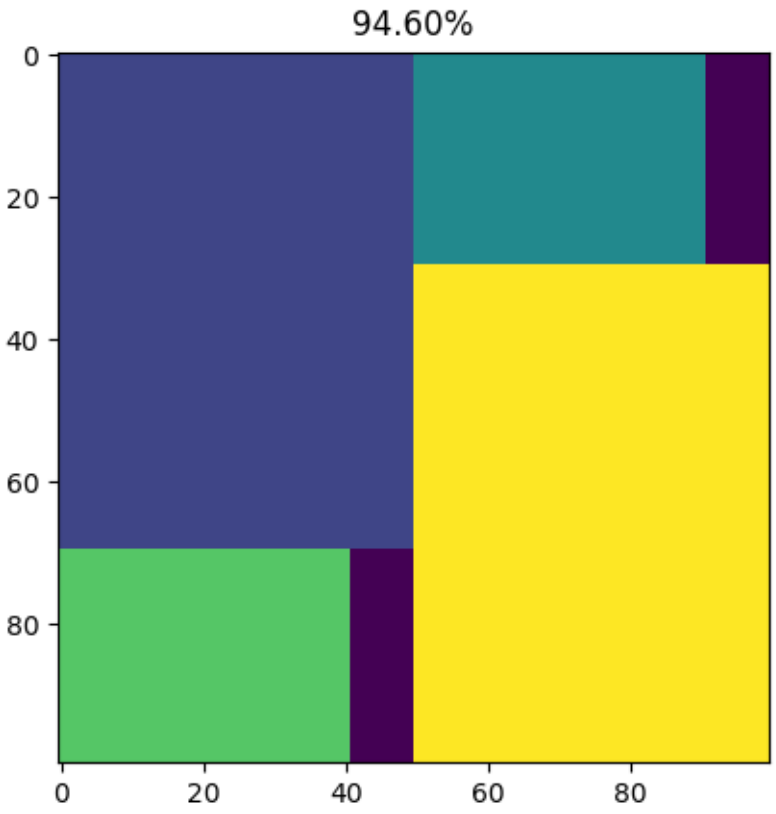

图 1:

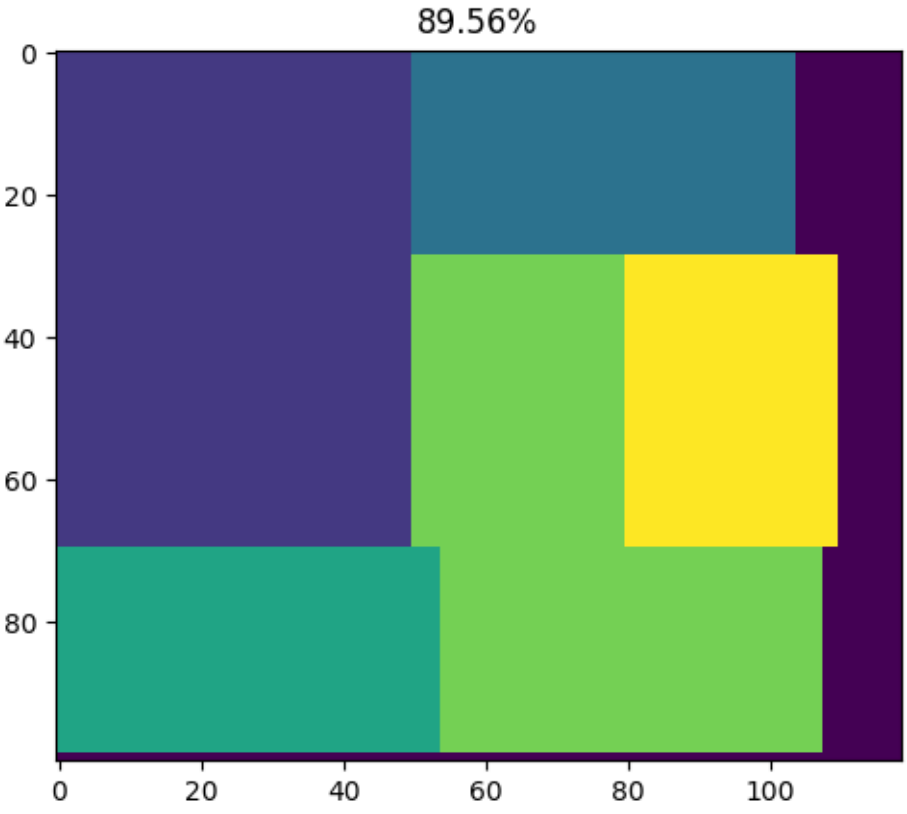

图 2:

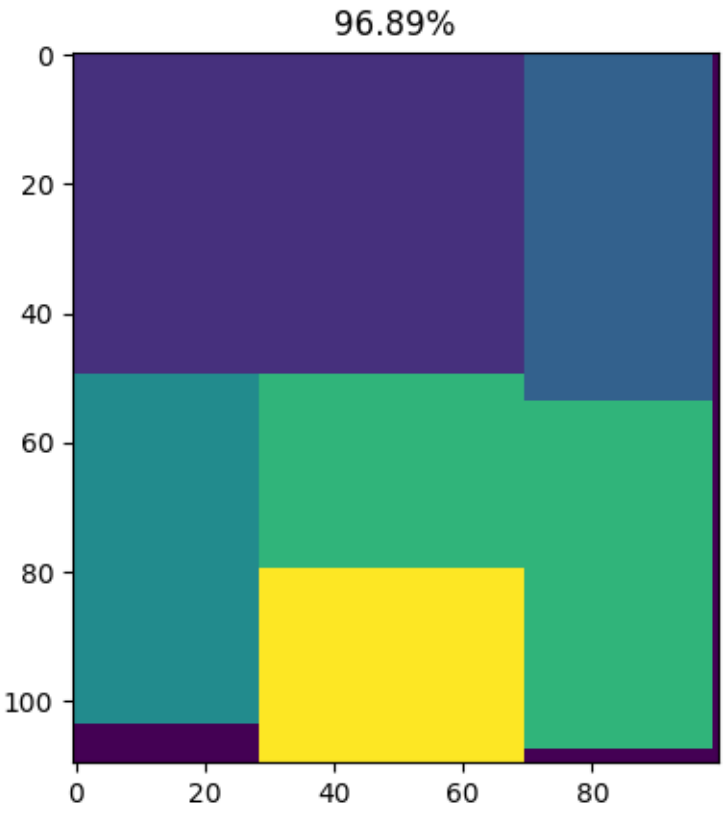

图 3:

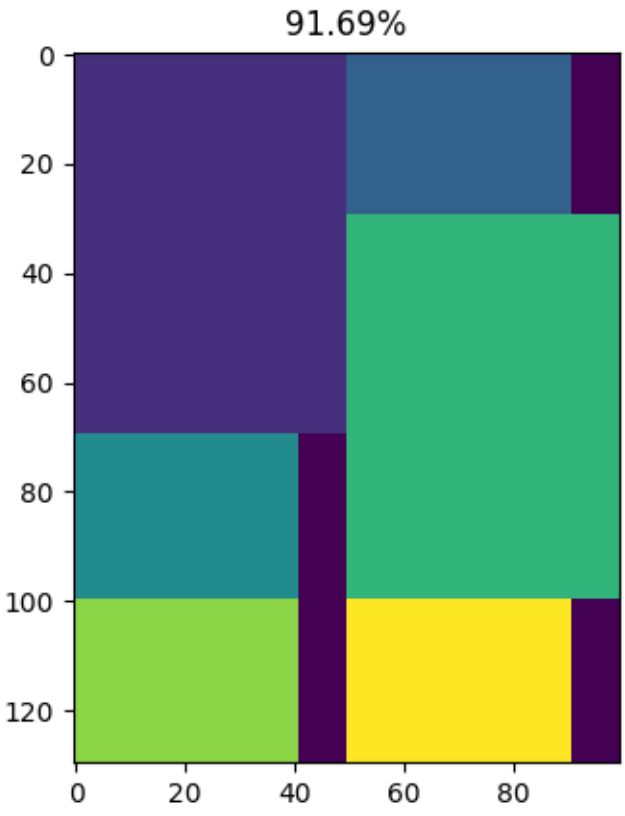

图 4:

5.2.5 200×200

没有跑出来结果

6 实验分析

当数据规模比较小的时候，问题都可以完美的求解，但是当石料的面积变大时，因为递归的算法复杂的太高，是指数级的，因此当石料的规模变大，问题将变得很难求解，因此递归算法对于此题只适合在小的数据规模下使用，如果数据规模较大，应该考虑其他的算法。而本题的算法复杂的为 $O((8k)^n)$ 假设有 k 种石砖。

7 实验总结

递归算法，是一种很优雅的算法，并且广泛应用于各种问题，如马踏棋盘，八皇后。但是使用递归算法的时候需要注意分析算法的时间复杂度。结合具体的问题规模使用。