算法作业：石块切割

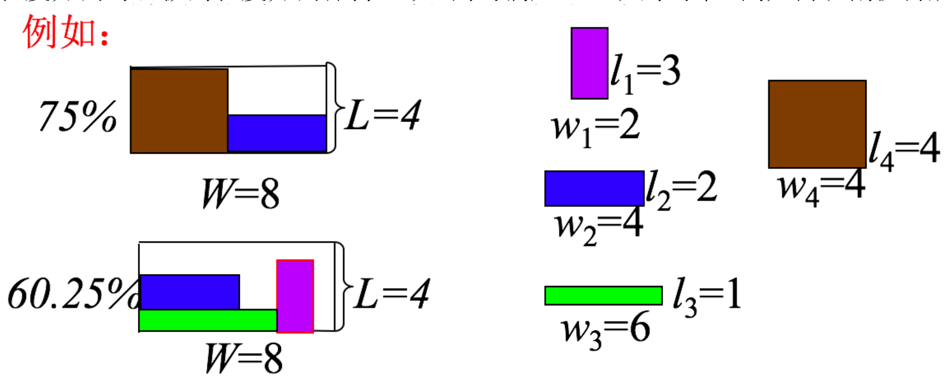
张富威 22920162204078

2018/3/12

目录

1. 问题描述

石材切割问题给定一块长为 L, 宽度为 W 的石板。现需要从板上分别切割出 n 个长度为 li, 宽度为 wi 的石砖。切割的规则是石砖的长度方向与石板的长度方向保持一致, 同时满足一刀切的约束。问如何切割使得所



使用的石材利用率最高?

2. 算法分析

2.1 算法基本思想

算法主要运用递归的思想，从最初的一块大石块，一刀切成两个石块，这样就把问题分解成两个比较简单的问题，在把这两个石块继续往下切，直到找到符合规格的石块，并且计算出使用率，如果使用率更大，则存储答案的数据结构就要进行更新，在不考虑石块翻转的情况下，算法复杂度为 $\{(4k)^n\}$。

2.2 算法的优化

如果是要找到绝对最优解，那么必须全部遍历一遍，如果是相对最优解，我们可以使用启发式算法，先搜索大的方块，因为大的方块能够更快的提高使用率。我们增加一个递归结束条件，就是达到 90% 就结束递归，这样的话就能以最快的速度找到结果。

2.3 数据结构

实验中，我使用的数据结构是一个字典树，大致是这样的一个情况:

```
result = { 'info':[usage, usedarea, fullarea], 'size':[x1, x2, y1, y1],
                'nextone':{
                    'info':[usage, usedarea, fullarea], 'size':[x1, x2, y1, y1],
                    'nextone':{......},'nexttwo':{......}
                }, 'nexttwo':{
                    'info':[usage, usedarea, fullarea], 'size':[x1, x2, y1, y1],
                    'nextone':{......},'nexttwo':{......}
                }
                }
            info: 一个列表，包括使用率，使用面积和总面积。
            size: 分别是石块四个端点的坐标。
            nextone: 一刀切下去之后，两个石块中的一块
            nexttwo: 两个石块的另一块
```

这是一个树状结构，每个大的石块包含两个小的石块的信息。

3. 代码实现

3.1 编程语言

本题使用 python 实现，虽然时间上肯定比 C 慢，但是更容易实现，也便于使用 matplotlib 库和 numpy 库画出分割之后的结果。

3.2 具体操作

算法采用坐标标记的形式，方便后续画图，相当于把原材料放进二维坐标轴中去切，每次递归传进切出来的两块石块的顶点坐标，直到切出满足要求的石块回溯，或者再也切不下去回溯。遍历完所有种可能，就能找到最优解。

4. 代码

4.1 python 代码

```python
# coding:utf-8
import matplotlib as mpl
```

```
import matplotlib.pyplot as plt
import numpy as np
import time
class Brick():
    '''
        brick_list saves the length and width of all the bricks as a dictionary.
            example:
            d = {1:[4, 6], 2:[3,2], 3:[1,1]}
            there are three types of bricks, as the length and width in d[1],
            d[2], d[3]raw_material saves the length and width of the raw material as a list
            example:
                [100, 100] the length and width of the raw material is 100 and 100
        bricknums is the sum types of the bricks
        result save the max usage plan
            example:
                dic = { 'info':[usage, usedarea, fullarea], 'size':[x1, x2, y1, y1],
                'nextone':{
                    'info':[usage, usedarea, fullarea], 'size':[x1, x2, y1, y1],
                    'nextone':{......},'nexttwo':{......}
                }, 'nexttwo':{
                    'info':[usage, usedarea, fullarea], 'size':[x1, x2, y1, y1],
                    'nextone':{......},'nexttwo':{......}
                }
                }
            info: usage is use ratio of the bricks,
                  usearea is area of the useful bricks, fullarea
                  is calculate by (x2 - x1)*(y2 - y1)
            size: there are four coordinates ,
                  x1 x2 is the absciass, y1 y2 is the ordinate.
            nextone: one of the brick cut by knife
            nexttwo: another brick cut by knife
        result_point: save the coordinates of all the found bricks
            example:
                we find two bricks and the coordinates are
                [0, 1, 0, 1], [1, 2, 0, 1], .........
                so the result_point is [[0, 1, 0, 1], [1, 2, 0, 1], [.........]]
    '''
    def __init__(self, raw_material, bricknums, brick_list, result, result_point):
        self.raw_material = raw_material
```

```
        self.bricknums = bricknums
        self.brick_list = brick_list
        self.result = result
        self.result_point = result_point
    '''
        inputBrickInfo():  input the infomation of the raw material
                           and all types of the cut-bricks
    '''
    def inputBrickInfo(self):
        leng = 88
        print leng*'*'
        self.raw_material = input(' 请输入原材料的规格（长和宽）：')
        self.bricknums = input(' 请输入要切割的砖头种类总数:')
        for i in range(1, self.bricknums + 1):
            brick_list[i] = list(input(' 请输入第 {} 种砖块的规格 (长和宽):'.format(i)))
    '''
        sortBrickSize(): sort the bricks from large size to small size
            example:
                the brick_list is {1:[1, 1], 2:[3, 4], 3:[5, 6]}
                after finishing this function, we get the brick_list:
                    {1:[5, 6], 2:[3, 4], 3:[1, 1]}
    '''
    def sortBrickSize(self):
        temp_list = ()
        for i in range(1, self.bricknums + 1):
            max_area = self.brick_list[i][0] * self.brick_list[i][1]
            target = 0
            for j in range(i + 1, self.bricknums + 1):
                if max_area < self.brick_list[j][0] * self.brick_list[j][1]:
                    target = j
                    max_area = self.brick_list[j][0] * self.brick_list[j][1]
            if target:
                self.brick_list[target], self.brick_list[i] = self.brick_list[i],\
                self.brick_list[target]
    '''
        findContentBrick():
            judge whether the brick now is accorded with
            one of the given bricks in self.brick_list
            if it contents, the usage change to 1
```

```
    '''
    def findContentBrick(self, x1, x2, y1, y2):
        side_length = [x2 - x1, y2 - y1]
        for i in range(1, self.bricknums + 1):
            # find it
            if side_length == self.brick_list[i]:
                return 1
        return 0
    '''
        calArea(): return the area of the brick
    '''
    def calArea(self, x1, x2, y1, y2):
        return (x2 - x1)*(y2 - y1)
    '''
        updateInfo(): update the dict['info'][0] (usage), dict['info'][1] (usedarea)
    '''
    def updateInfo(self, usedarea, dict, fullarea):
        dict['info'][0] = (float)(usedarea) / (float)(fullarea)
        dict['info'][1] = usedarea
    '''
        replaceArrayxy():replace the xy matrix from (x1, y1) to (y1, y2)
            example:
                wo got xy [[0, 0],
                            [0, 0]]
                then replaceArrayxy(xy, 0, 0, 1, 1, 5)
                we got xy [[5, 5],
                            [5, 5]]
    '''
    def replaceArrayxy(self, xy, x1, x2, y1, y2, val):
        for i in range(x1, x2):
            for j in range(y1, y2):
                xy.itemset((j, i), val)
    '''
        plotGraph(): using matplotlib and numpy to draw
        the result of the bricks we just cut
    '''
    def plotGraph(self):
        fig = plt.figure()
        ax = fig.add_subplot(111)
```

```
        xy = np.zeros([self.raw_material[1], self.raw_material[0]])
        countlist = {}
        for i in range(1, len(self.brick_list) + 1):
            countlist[i] = 0
        for i in self.result_point:
            l = [i[1] - i[0], i[3] - i[2]]
            for j in range(1, len(self.brick_list) + 1):
                if l == self.brick_list[j]:
                    countlist[j] += 1
                    self.replaceArrayxy(xy, i[0], i[1], i[2], i[3], j)
        leng = 88
        print leng*'-'
        for j in range(1, len(self.brick_list) + 1):
            print ' 大小为{0}的石块颜色对应的序号为{1}, 个数为{2}'\
            .format(self.brick_list[j], j, countlist[j])
        print ' 利用率为{0:.2%}, 利用面积为{1}, 原材料总面积为{2}'\
        .format((float)(self.result['info'][0]), self.result['info'][1]\
        , self.result['info'][2])
        print leng*'*'
        plt.imshow(xy)
        plt.colorbar()
        plt.show()

    '''
        getResult(): get the result_point through dictionary result
    '''
    def getResult(self, dict, blank):
        # print '-'*blank, 'info:', dict['info']
        # print '-'*blank, 'size', dict['size']
        # print '*********************************'
        if dict['nextone']:
            self.getResult(dict['nextone'], blank + 4)
        if dict['nexttwo']:
            self.getResult(dict['nexttwo'], blank + 4)
        if dict['nextone'] == {} and dict['nexttwo'] == {} and dict['info'][0] == 1:
            self.result_point.append(dict['size'])
    '''
        findOptimalSolution():
            its function is just like the name it is. through
```

```
        this function, we try all the probability to find the optimal solution
'''
def findOptimalSolution(self, x1, x2, y1, y2, resdic):
    resdic['size'] = [x1, x2, y1, y2]
    resdic['nextone'] = {}
    resdic['nexttwo'] = {}
    resdic['info'] = [0, 0, self.calArea(x1, x2, y1, y2)]
    # judge if it is accorded with one of the given bricks in self.brick_list
    if self.findContentBrick(x1, x2, y1, y2):
        resdic['info'] = [1.0, self.calArea(x1, x2, y1, y2), \
        self.calArea(x1, x2, y1, y2)]
        return 0
    for i in range(1, self.bricknums + 1):
        max = 0
        '''
            across cutting
        '''
        if x1 + self.brick_list[i][0] < x2:
            dic1 = {'info':[], 'size':[], 'nextone':{}, 'nexttwo':{}}
            dic2 = {'info':[], 'size':[], 'nextone':{}, 'nexttwo':{}}
            self.findOptimalSolution(x1, x1 + self.brick_list[i][0], y1, y2, dic1)
            self.findOptimalSolution(x1 + self.brick_list[i][0], x2, y1, y2, dic2)
            # find out a more useful result
            usedarea = dic1['info'][1] + dic2['info'][1]
            if usedarea > resdic['info'][1]:
                self.updateInfo(usedarea, resdic, self.calArea(x1, x2, y1, y2))
                resdic['nextone'] = dic1
                resdic['nexttwo'] = dic2
        '''
            rip cutting
        '''
        if y1 + self.brick_list[i][1] < y2:
            dic1 = {'info':[], 'size':[], 'nextone':{}, 'nexttwo':{}}
            dic2 = {'info':[], 'size':[], 'nextone':{}, 'nexttwo':{}}
            self.findOptimalSolution(x1, x2, y1, y1 + self.brick_list[i][1], dic1)
            self.findOptimalSolution(x1, x2, y1 + self.brick_list[i][1], y2, dic2)
            usedarea = dic1['info'][1] + dic2['info'][1]
            if usedarea > resdic['info'][1]:
                self.updateInfo(usedarea, resdic, self.calArea(x1, x2, y1, y2))
```

```
                    resdic['nextone'] = dic1
                    resdic['nexttwo'] = dic2
```

执行代码:

```
if __name__ == '__main__':
    #initialize the property of the living example
    raw_material = []
    brick_list = {}
    bricknums = 0
    result = {'info':[], 'size':[], 'nextone':{}, 'nexttwo':{}}
    result_point = []

    brick = Brick(raw_material, bricknums, brick_list, result, result_point)
    brick.inputBrickInfo()

    time_start = time.time()

    brick.sortBrickSize()
    brick.findOptimalSolution(0, brick.raw_material[0], 0, brick.raw_material[1], brick.result)
    brick.getResult(brick.result, 0)
    time_end = time.time()
    print '找到结果'
    print '用时:{0:.3f}s'.format(time_end - time_start)
    brick.plotGraph()
```

5. 实验用例以及实验结果

5.1 实验用例

5.1.1 用例一

原材料规格:120, 110

所需要石块的规格:

- 石块一：33,55
- 石块二：27,50
- 石块三：40,60
- 石块四：30,60

结果详见 5.2.1

5.1.2 用例二

原材料规格:130,140

所需要石块的规格:

- 石块一：33,44
- 石块二：44,55
- 石块三：55,66
- 石块四：23,56

结果详见 5.2.2

5.1.3 用例三

原材料规格:130,150

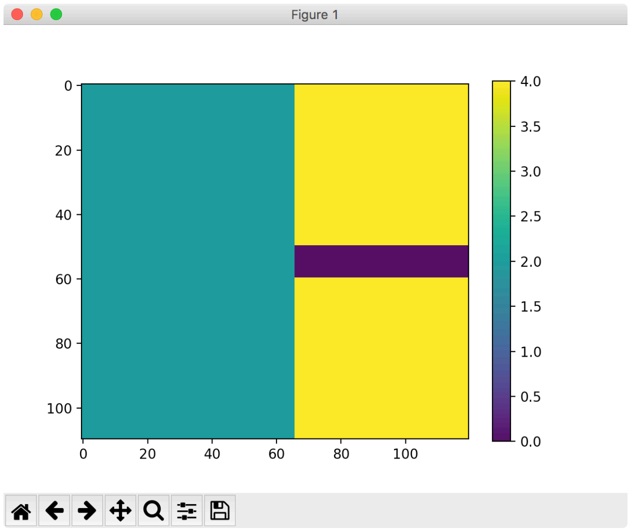
所需要石块的规格:

- 石块一：29,69
- 石块二：18,34
- 石块三：44,55
- 石块四：67,45

结果详见 5.2.3

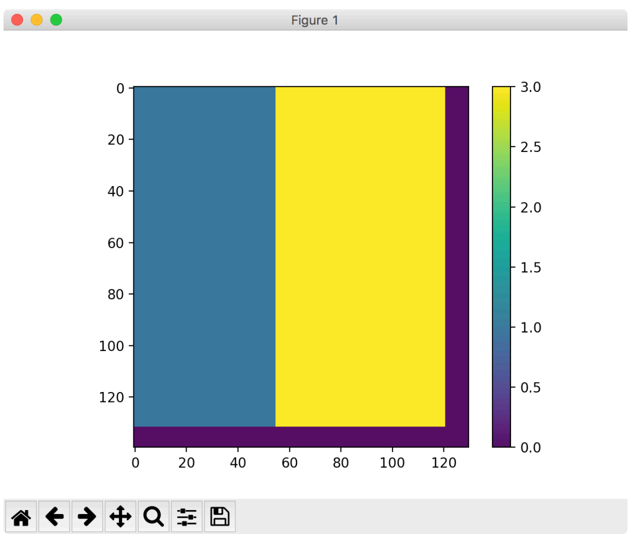
5.2 实验结果

5.2.1 用例一结果



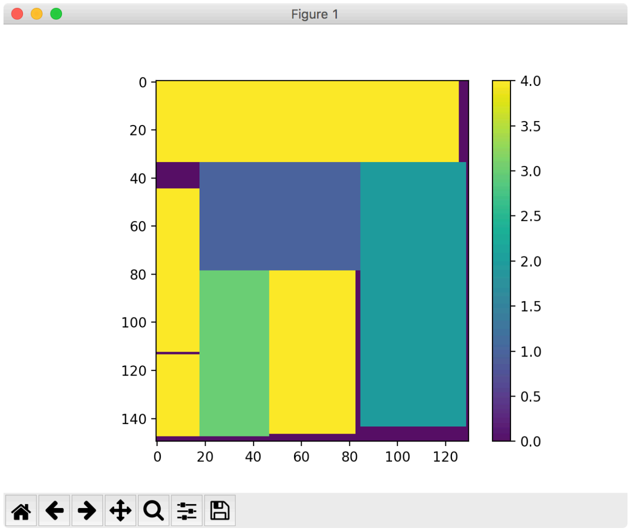

```
******************************************************************************
请输入原材料的规格（长和宽）:120,110
请输入要切割的砖头种类总数:4
请输入第 1 种砖块的规格(长和宽):33,55
请输入第 2 种砖块的规格(长和宽):27,50
请输入第 3 种砖块的规格(长和宽):40,60
请输入第 4 种砖块的规格(长和宽):30,60
找到结果
用时:0.723s
------------------------------------------------------------------------------
大小为[40, 60]的石块颜色对应的序号为1, 个数为0
大小为[33, 55]的石块颜色对应的序号为2, 个数为4
大小为[30, 60]的石块颜色对应的序号为3, 个数为0
大小为[27, 50]的石块颜色对应的序号为4, 个数为4
利用率为95.91%, 利用面积为12660, 原材料总面积为13200
******************************************************************************
```

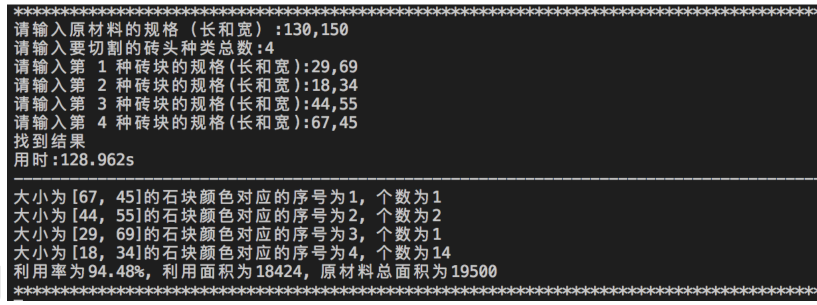
5.2.2 用例二结果




```
******************************************************************************
请输入原材料的规格（长和宽）:130,140
请输入要切割的砖头种类总数:4
请输入第 1 种砖块的规格(长和宽):33,44
请输入第 2 种砖块的规格(长和宽):44,55
请输入第 3 种砖块的规格(长和宽):55,66
请输入第 4 种砖块的规格(长和宽):23,56
找到结果
用时:9.229s
------------------------------------------------------------------------------
大小为[55, 66]的石块颜色对应的序号为1, 个数为2
大小为[44, 55]的石块颜色对应的序号为2, 个数为0
大小为[33, 44]的石块颜色对应的序号为3, 个数为6
大小为[23, 56]的石块颜色对应的序号为4, 个数为0
利用率为87.76%, 利用面积为15972, 原材料总面积为18200
******************************************************************************
```

5.2.3 用例三结果





6. 实验总结

6.1 总结

本次算法实验，考验的是递归的运用，但是递归有一个缺点，就是算法复杂度过大，递归也有好处，就是代码量小，几行代码就能搞定，可以说是有利有弊。如果用启发式算法加上递归结束条件，我觉得代码运行时间会大大缩小，不会像用例三一样花费将近 2 分钟。## 6.2 扩展如果石块中有小孔或者其他破损使得不能被切下去，可以增加一个判断条件，如果坐标有经过那个点（小孔），就不要继续往下切。

7. 参考资料

7.1 python 库

- numpy
- matplotlib
- time

7.2 方法

- python 类，字典，列表等
- 递归
- 实例属性