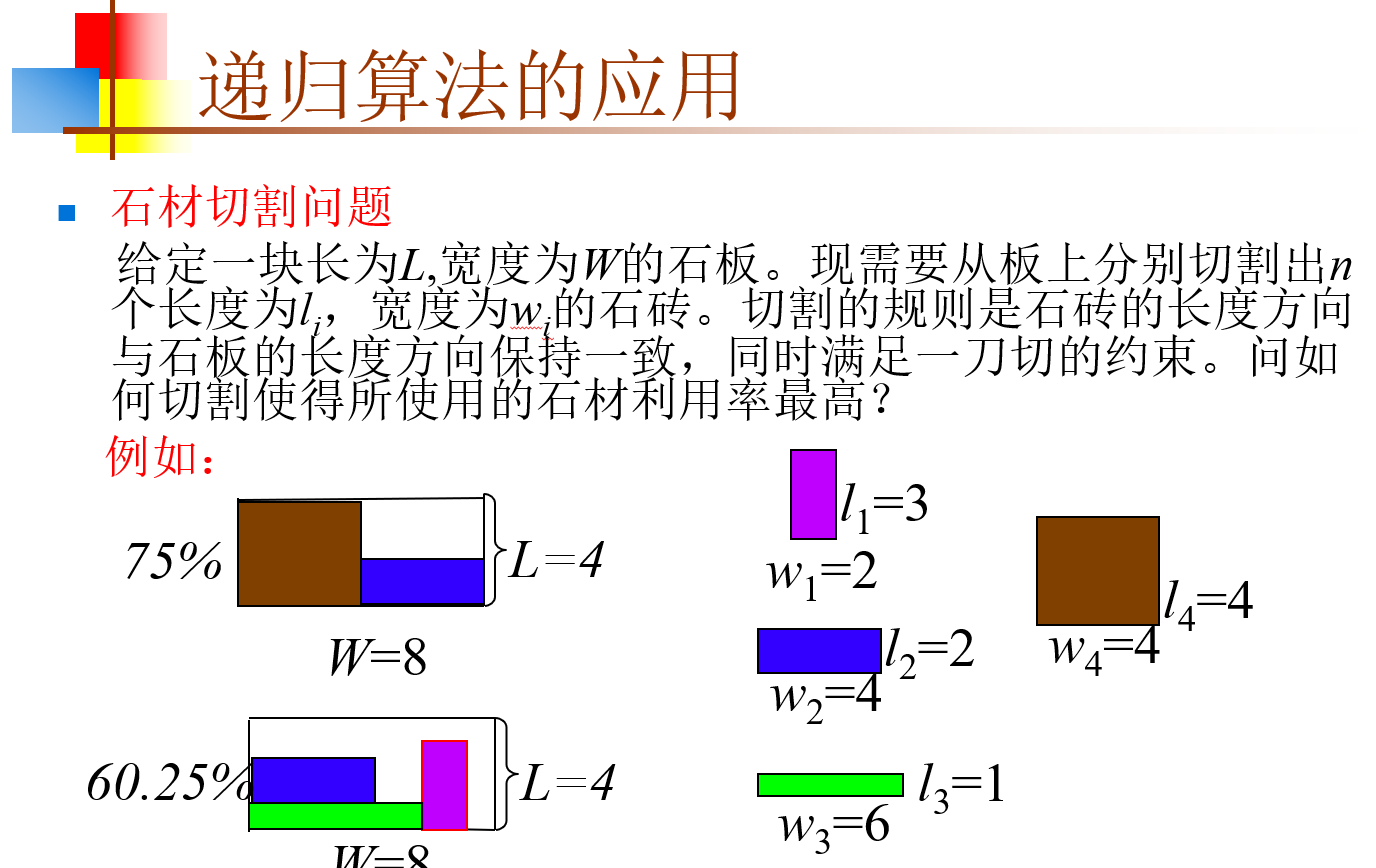
**使用动态规划算法解决切石块问题**

王柳依 37120222203439

#### 基本思路

1. **动态规划**



代码使用python语言编写，用**动态规划算法**，解决切石块问题。对于给定的石材切割问题，目标是在给定尺寸的石材上切割出给定数量的石块，使得石材的利用率最大化。

这里将问题分解为子问题。对于每个子问题，即给定“棋盘”的长和宽，对于石材在不同位置和尺寸下，能够切割出的最大价值——石块的面积。通过逐步解决这些子问题，最终得到整个问题的最优解。

由于每个石块都只需要一个，所以这里应该用01背包。**状态转移方程**：

**变量含义**如下：

：从第1块石头到第c块石头，每块石头选择放或者不放，每块长为i，宽为j的原料能切出石块的最大总面积。

：第c块需求石头的长度。

：第c块需求石头的宽度。

：第c块需求石头的价值即面积。

选择到第c个石块进行切割，有三种**切割方法**，即：

(1): 垂直于长度方向一刀切，上面为从第1块石头到第c-1块石头，能切割出的最大面积。下面是新的第c块石块；

(2): 垂直于宽度方向一刀切，左面为从第1块石头到第c-1块石头，能切割出的最大面积。右面是新的第c块石块；

(3): 本轮回不进行一刀切，最大面积不变。

算法具有最优子结构性质，时间复杂度为O(n\*l\*w) （n为石块数量）。

01背包可以对数组降维，进行空间优化。典型的01背包问题，通过**倒序遍历**容量，来达到不重复装入背包的效果。题目要求采用“一刀切”的策略，所以这里我们也倒序遍历作为“背包容量”的长和宽，空间复杂度由O(l\*w\*n)优化为O(l\*w),状态转移方程变为：

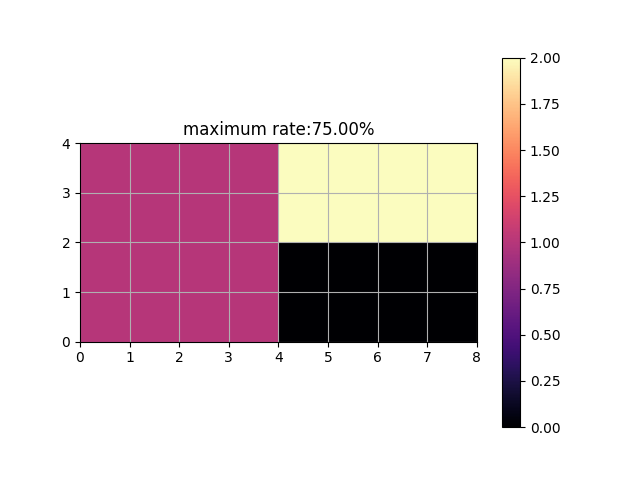
核心代码：

1. **def** solve():
2. **for** i **in** range(rockL[0], l + 1):
3. **for** j **in** range(rockW[0], w + 1):
4. f[i][j] = rockV[0]
5. method[i][j].append((0, how.origin))
7. **for** chose **in** range(1, nums):
8. **for** i **in** range(l, rockL[chose]-1, -1):
9. **for** j **in** range(w, rockW[chose]-1, -1):
10. lastmethod = deepcopy(method[i][j])
11. s = f[i - rockL[chose]][j] + rockV[chose]
12. t = f[i][j - rockW[chose]] + rockV[chose]
13. **if** s > f[i][j] **and** s >= t:
14. f[i][j] = s
15. method[i][j] = deepcopy(lastmethod) + [(chose, how.horizon)]
16. **elif** t > f[i][j] **and** t > s:
17. f[i][j] = t
18. method[i][j] = deepcopy(lastmethod) + [(chose, how.vertical)]
19. **结果呈现**

method二维数组用于记录达到最大利用率的切割方式，计算并输出最大利用率。使用matplotlib库绘制切割情况的可视化图像，并保存图像，非常直观。

#### 程序运行结果

测试例：



测试例正确。

#### 和其它算法的对比

枚举算法：代码简单，理解最容易，但是时间复杂度过高，完全无法有效解决问题。

递归算法：递归层数多，容易造成很大的时间复杂度。也可能因为内存占用过多而导致运行问题。可以用在小规模的问题，效率低。需要多种方式的剪枝。

贪心算法：逻辑简单，容易理解。有长度优先、宽度优先、面积优先三种贪心策略，但是无论采取哪种策略，都无法保证正确性。适合粗略计算，用在小作坊切石头，大切割厂需要更靠谱的算法。

动态规划算法：逻辑难度一般。能有效得到最优解，也适合记录切割方式。优化方式有：可以配合贪心策略。