算法分析与设计作业

第一次

姓名

班级

学号

电话

Email

日期

1. 给定 m*n 的二维矩阵，使用动态规划算法解决最大子矩阵和问题。

算法步骤

（1）问题拆解：

将二维问题降为一维问题：通过固定矩阵的上下边界，计算每一列的和，将二维问题转化为一维最大子数组和问题。

（2）动态规划：

使用 Kadane 算法来求解一维最大子数组和问题。

（3）关键点：

对于每一个可能的上下边界组合，计算压缩的一维数组，然后在该数组中寻找最大子数组和。

代码实现

```
def max_sum_submatrix(matrix):
    if not matrix or not matrix[0]:
        return 0

    m, n = len(matrix), len(matrix[0])
    max_sum = float('-inf')

    # 枚举上边界
    for top in range(m):
        temp = [0] * n   # 压缩数组，用于存储列累积和

        # 枚举下边界
        for bottom in range(top, m):
            # 更新每列的累积和
            for col in range(n):
                temp[col] += matrix[bottom][col]

            # 使用 Kadane 算法计算一维数组最大子数组和
            current_sum = 0
            max_temp_sum = float('-inf')
            for val in temp:
                current_sum = max(val, current_sum + val)
                max_temp_sum = max(max_temp_sum, current_sum)

            # 更新全局最大值
            max_sum = max(max_sum, max_temp_sum)

    return max_sum

# 测试
```

```
matrix = [
    [1, -2, -1, 4],
    [-3, 4, 1, -1],
    [2, -1, -4, 5],
    [1, -1, 1, -2]
]
print("最大子矩阵和:", max_sum_submatrix(matrix))
```

时间复杂度

外层两重循环（上下边界）：$O(m^2)$
每次计算 Kadane 算法：$O(n)$
总时间复杂度：$O(m^2 \times n)$

空间复杂度

使用了一个长度为 n 的数组存储累积和：$O(n)$

具体步骤

以代码所给矩阵为示例：
初始矩阵为：

$$\begin{bmatrix} 1 & -2 & -1 & 4 \\ -3 & 4 & 1 & -1 \\ 2 & -1 & -4 & 5 \\ 1 & -1 & 1 & -2 \end{bmatrix}$$

1.固定上下边界，计算列累积和：

第一步：固定上边界 top，从 top = 0 开始，逐一扩大下边界 bottom，计算列累积和 temp：

①top = 0，bottom = 0，当前 temp 为：

$$\text{temp} = \begin{bmatrix} 1 & -2 & -1 & 4 \end{bmatrix}$$

对 temp 应用 Kadane 算法，遍历 temp：

$$\begin{gathered} 1 => \text{current\_sum} = 1, \text{max\_sum} = 1 \\ -2 => \text{current\_sum} = -1, \text{max\_sum} = 1 \\ -1 => \text{current\_sum} = -1, \text{max\_sum} = 1 \\ 4 => \text{current\_sum} = 4, \text{max\_sum} = 4 \end{gathered}$$

此时最大子数组和为 4；

②top = 0，bottom = 1，当前 temp 为：

$$\text{temp} = \begin{bmatrix} 1+(-3) & -2+4 & -1+1 & 4+(-1) \end{bmatrix} = \begin{bmatrix} -2 & 2 & 0 & 3 \end{bmatrix}$$

对 temp 应用 Kadane 算法，遍历 temp：

$$\begin{gathered} -2 => \text{current\_sum} = -2, \text{max\_sum} = -2 \\ 2 => \text{current\_sum} = 2, \text{max\_sum} = 2 \\ 0 => \text{current\_sum} = 2, \text{max\_sum} = 2 \\ 3 => \text{current\_sum} = 5, \text{max\_sum} = 5 \end{gathered}$$

此时最大子数组和为 5；

③top = 0，bottom = 2，当前 temp 为：

$$\text{temp} = \begin{bmatrix} -2+2 & 2+(-1) & 0+(-4) & 3+5 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -4 & 8 \end{bmatrix}$$

对 temp 应用 Kadane 算法，遍历 temp：

$$0 => \text{current\_sum} = 0, \text{max\_sum} = 0$$
$$1 => \text{current\_sum} = 1, \text{max\_sum} = 1$$
$$-4 => \text{current\_sum} =- 3, \text{max\_sum} = 1$$
$$8 => \text{current\_sum} = 8, \text{max\_sum} = 8$$

此时最大子数组和为 8；

④top = 0，bottom = 3，当前 temp 为：

$$\text{temp} = [0+1 \quad 1+(-1) \quad -4+1 \quad 8+(-2)] = [1 \quad 0 \quad -3 \quad 6]$$

对 temp 应用 Kadane 算法，遍历 temp：

$$1 => \text{current\_sum} = 1, \text{max\_sum} = 1$$
$$0 => \text{current\_sum} = 1, \text{max\_sum} = 1$$
$$-3 => \text{current\_sum} =- 2, \text{max\_sum} = 1$$
$$6 => \text{current\_sum} = 6, \text{max\_sum} = 6$$

此时最大子数组和为 6；

⑤top = 1，bottom = 1，当前 temp 为：

$$\text{temp} = [-3 \quad 4 \quad 1 \quad -1]$$

对 temp 应用 Kadane 算法，遍历 temp：

$$-3 => \text{current\_sum} =- 3, \text{max\_sum} =- 3$$
$$4 => \text{current\_sum} = 4, \text{max\_sum} = 4$$
$$1 => \text{current\_sum} = 5, \text{max\_sum} = 5$$
$$-1 => \text{current\_sum} = 4, \text{max\_sum} = 5$$

此时最大子数组和为 5；

以此类推，直至遍历所有子矩阵，调用 max 函数得到最大子矩阵和为 8，对应的矩阵为：

$$\begin{bmatrix} 4 \\ -1 \\ 5 \end{bmatrix}$$

2. A 碗中有 n 棵红豆， 要你把它转移到 B 碗中， 每次只能从 A 碗中取 1,2,或 3 粒红豆放入 B 碗,求共有多少种不同的解决方法？分别用递推法和备忘录法写出算法。

问题分析

状态转移方程：

$$f(n) = f(n-1) + f(n-2) + f(n-3)$$

$f(n-1)$代表最后一步取 1 粒红豆；$f(n-2)$代表最后一步取 2 粒红豆；

$f(n-3)$代表最后一步取 3 粒红豆。

初始条件：

由于 0 粒红豆只有 1 种方式（即什么都不取）、1 粒红豆只能取 1 次、2 粒红豆有两种方式（取 1+1 或直接取 2）、1 粒红豆有四种方式（1+1+1，1+2，

2+1，直接取 3），故有：

$$f(0) = 1$$
$$f(1) = 1$$
$$f(2) = 2$$
$$f(3) = 4$$

递推法：

思路：

递推法通过从底部开始逐步计算 f(0),f(1),...,f(n)，避免了重复计算。

代码实现：

```
def transfer_beans_recursive(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    elif n == 2:
        return 2
    elif n == 3:
        return 4

    # 初始化
    dp = [0] * (n + 1)
    dp[0], dp[1], dp[2], dp[3] = 1, 1, 2, 4

    # 递推
    for i in range(4, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3]

    return dp[n]

# 测试
n = 5
print(f"将 {n} 粒红豆从 A 碗移到 B 碗的方法数：{transfer_beans_recursive(n)}")
```

示例输出：

对于 n=5：

初始化 f(0)、f(1)、f(2)、f(3)的值，进行运算：

$$\begin{aligned} f(5) &= f(4) + f(3) + f(2) = (f(3) + f(2) + f(1)) + f(3) + f(2) \\ &= (4 + 2 + 1) + 4 + 2 = 13 \end{aligned}$$

备忘录法：

思路：

备忘录法是递归的改进，通过记录已计算的子问题结果，避免重复计算。

代码实现：

```
def transfer_beans_memo(n, memo=None):
    if memo is None:
        memo = {}

    # 基础情况
    if n == 0:
        return 1
    elif n == 1:
        return 1
    elif n == 2:
        return 2
    elif n == 3:
        return 4

    # 如果结果已在备忘录中，直接返回
    if n in memo:
        return memo[n]

    # 递归求解并存入备忘录
    memo[n] = (
        transfer_beans_memo(n - 1, memo) +
        transfer_beans_memo(n - 2, memo) +
        transfer_beans_memo(n - 3, memo)
    )
    return memo[n]

# 测试
n = 5
print(f"将 {n} 粒红豆从 A 碗移到 B 碗的方法数：{transfer_beans_memo(n)}")
```

示例输出：

对于 n=5：

初始化 f(0)、f(1)的值，进行运算：

$$
\begin{aligned}
f(5) &= f(4) + f(3) + f(2) \\
&= (f(3) + f(2) + f(1)) + (f(2) + f(1) + f(0)) + (f(1) + f(0)) \\
&= ((f(2) + f(1) + f(0)) + (f(1) + f(0)) + f(1)) + ((f(1) + f(0)) \\
&\quad + f(1) + f(0)) + (f(1) + f(0)) \\
&= (((f(1) + f(0)) + f(1) + f(0)) + (f(1) + f(0)) + f(1)) + ((f(1) \\
&\quad + f(0)) + f(1) + f(0)) + (f(1) + f(0)) = 13
\end{aligned}
$$