**算法课程设计实验报告**

学院：计算机与信息科学学院

专业：计算机科学与技术

年级：2021级3，4班

组员：

袁晨航

曹天宇

谭燕霜

冉嘉麟

**理论：**

**算法思想理论介绍**

**设计步骤**

**具体实例**

**总结反思**

**内容拓展**

**实践：**

**代码实现**

**运行结果**

**动态规划算法理论介绍：**

递归分治介绍：

在介绍动态规划之前，首先要提及到递归分治算法，所谓分治算法也就是将一个可以分解的大规模问题分解为多个小规模问题。那我们该以什么标准来进行判断当前问题是否可以进行分解呢？（也就是如何判断该问题是否满足最优子结构性质）

一般有如下几个标准：

1：当我们缩小问题规模后是否可以解决问题，问题是否变得简单。

2：我们缩小后的子问题是否和原问题是一个类型，一样的同种问题。（如递归求解某数的阶乘，我们递归分解减小规模后依旧是求小于当前数据的阶乘。）

3：我们是否可以使用分解后子问题的解，来组成原问题的解。（当然大多数情况，满足前两条的问题，都会满足第三条）

4：我们分解后的子问题，是不是相互独立，有没有交叉，有没有多次计算，或者，各子问题之间存不存在相互影响。

递归与分治主要的过程在于如何分解，以及分解为子问题解决后，如何回归组成原问题的解。对于分解我们一般会采取逐次减小，或者折半分解，最后就是按随机的标准分。对于前者，例如递归求解某个数的阶乘，递归求解一个序列的全排列。对于后者，例如我们的归并排序，按随机标准分的最典型例子就是快速排序以及线性选择。

这里还有一个问题，那就是如何对子问题的解进行合并呢？

对于前面不同的分解方式，我们需要使用不同的合并方式，对第一种（求全排列），代码中最重要的部分就是交换数据，当问题分解到最小的单位时，就输出当前的解。但是为了当前子问题不会影响到别的子问题，我们需要将交换的数据还原。对于折半分解，（归并排序），最好的方法是使用与当前求解问题的规模相匹配的数组，来存储当前的解，然后通过改变子问题开始结束位置来进行组合。

而动态规划的递进过程与递归分治基本一致，但是又有很大的区别。对于分治算法我们适用的条件是当前问题可以分解为小问题，动态规划也是利用这一特点。但是递归分治算法在分解问题时我们可以很清晰的看到，分解后的子问题是没有重叠的。其子问题相互独立，互不影响，这就不存在一个问题存在多个解这一说法。

而我们在分解某些问题时，如前面判断是否可以使用递归分治算法时，可能会遇到，子问题之间不是独立的，几个子问题之间存在联系，或者一个问题被计算多次，这就存在重复计算，或者一个问题纯在多个解的情况。比如矩阵连乘问题，假设有三个可以连续相乘的矩阵1，矩阵2，矩阵3，当求解当前问题解时，假设乘法次序是（1\*2）\*3，这与1\*（2\*3）的结果大概率是不同的。原因就是组成原问题的两个子问题不相同。而在三个矩阵的情况下，子问题重复计算还没有完全体现，当连乘规模扩大后，我们就会发现，越小的子问题，出现重复计算机的概率就越大。

这不代表这些问题就不能使用动态规划了，而是在动态规划中，我们需要进行重要的一步，那就是记录历数据。在分解递进过程中，加入记录历史数据这一功能。

使多阶段决策过程最优。交叠的子问题。

\* 书本上讲的太难懂了，我们需要做的就是利用历史数据来避免重复计算。

\* 动态规划算法通常用于求解具有某种最优性质的问题。在这类问题中，可能会有许多可行解。每一个解都对应于一个值，我们希望找到具有最优值的解。动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。与分治法不同的是，适合于用动态规划求解的问题，经分解得到子问题往往不是互相独立的，正因为不是独立的子问题，所以我们才会出现大量重复计算的东西，对于动态规划的思想，如若想掌握，我们必须通过题目来实践掌握，对于算法都是这样不可避免的，不要妄图自己会很容易就掌握这些算法。若用分治法来解这类问题，则分解得到的子问题数目太多，有些子问题被重复计算了很多次。如果我们能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，这样就可以避免大量的重复计算，节省时间。我们可以用一个表来记录所有已解的子问题的答案。不管该子问题以后是否被用到，只要它被计算过，就将其结果填入表中。这就是动态规划法的基本思路。

\* 通过组合子问题的解而解决整个问题的解。（所以该问题是要可以分解为小的子问题的） 。分治法与之类似，但是分治法会做很多不必要的工作，会重复多遍计算很多子结果。动态规划就是所有子问题就计算一遍，把结果利用变量存储起来。

**设计步骤：**

按照书上的介绍，我们假设能够使用动态规划来解决遇到的问题，那么主要有以下几个步骤：

1. 描述最优解的结构
2. 建立递推关系，递归定义最优解的值
3. 自底向上的方式来计算最优解的值
4. 由计算结果来构造一个最优解。

但是这四个步骤，乍一听，一脸懵，假设你遇到了一个问题到底该怎么办。理论都太过于学术，抽象，我们必须要将理论映射到实例中去。我认为，运用动态规划最重要的几个步骤应该是：

1 利用历史数据，已经存储好的历史数据，如何存储数据。（需要定义什么记录表）

2 明确数组元素含义，即明确记录表的含义。（往往使用的是一维数组，暂时使用二维数组还比较少。）

3 找出数组元素之间的关系，这一步是最难的。这一步也就是我们需要找出那所谓的递推公式。

4 找出初始值。

但是如上的描述还有还有很多的问题，首先，什么情况下才有最优解，在某些问题中，是否存在最优解这一说法，比如我们下面的例子，智障机器人的行进路线条数，在没有障碍的情况下，何谈最优这一概念。第二，建立递推关系，一般都有什么方法，如斐波那契数列，如果你没有知道递归公式的情况下，我们应该怎么寻找到序列的规律，换言之，我们在遇到与之类似的问题时，该采取什么策略来寻找到规律，推到出公式。第三，自底向上的由子问题解组成原问题的解是现在的主要想法，那是因为我们在解决子问题时越是分解到小的子问题重复的概率越大，那么我们是否可以在分解时就记录重复的子问题。而不是在分解完成后，再逐个解决子问题进行保留。最后如何通过最后的记录来组成原问题的解，重点在于如何回溯，如何重组。

**具体实例：路径条数的问题**

先介绍什么叫路径条数问题。（现在我们需要做的是暗示自己不知道这个题目可以使用动态规划来解决，我们需要使用上面几个步骤来得出，改题目可以使用动态规划。）



如图，有一个人工智障机器人，位于一个m\*n的网格左上角，即位于一个m行n列的表格左上角，假设该点为开始位置。这个智障每次只能向右或者向下一格，请问智障为了到达Finish终点，且在路线不重复的情况下，它能怎么走，共有多少种走法？

理论建立：

首先，分析最优子结构性质及判断子问题重叠性

我们来看看这个题目是否可以使用递归分治的策略。若可以，那么我们再看是否可以使用动态规划。

1. 假设机器人现在所在格子为2\*2，更或者为1\*2，或2\*1。该问题是否变得简单了，答案显而易见。
2. 如前，不管格子的规模多大，我们将其缩小后，问题依旧是机器人能够到达中终点，所能够行进的路线条数。很明显，格子2\*2，和格子1\*2是同种类型问题。
3. 还是和前面一样，我们很明显能看见，在2\*2的问题里面，包含了1\*2，和2\*1的小问题，其实就是子问题，如果我们解决这两个子问题就又可能解决2\*2这个大问题，2\*2的子问题的解，就是由前面两个子问题组成，我们可以将问题规模进一步扩大，在3\*2，2\*3，3\*3的问题中，都存在2\*2的子问题，如果我们解决了2\*2的子问题，我们就结局的了2\*2这个原问题，那么我们就有可能解决上面扩展的三个更大规模的问题。
4. 由上面个的前面两点，我们可以看出来，该问题满足最优子结构性质，尤其是第三点，当问题规模扩大后，我们发现在3\*3的子问题中，肯定会重复求解2\*2这一子问题。这一点满足动态规划的特性。

然后，分析完该问题可以使用动态规划后，我们需要来对问题的细节进行描述。也就是设计步骤的实现。

1. 首先我们需要描述最优解，就是解会长什么样子，如在矩阵连乘中，也就是确定乘法次序的矩阵序列；在本题中就是机器人走的路径。但是按照另外一种设计步骤我们应该要做的就是，假设定义一个数组dp[i][j]，并将其定义为i行j列的表格有dp[i][j]条重左上角到右下角的路径可供机器人选择。
2. 我们来推到递推关系，也就是状态转移方程。由上面对于子结构的分析。我们可以先假设dp[i][j]为1\*1，此时只有一种方式，初始化为1，因为当前就已经在终点了，完全不需要进行移动即可到达终点，当然这是第三步的事情。在这个问题中，我们需要考虑的是怎么走到中终点，而条件也给出了限制，每次的选择要么是向右或者向下。所以我们的结果也只可能由向上或者向左组成。

一：当i=1时，即现在只有1行。此时不管j为几，（前提是为正）都只有一种选择，机器人只能选择一直向右这一种方式。假如用公式表示就是dp[i][j]=dp[i][j-1].

二：同理当j=1时，也是一样。dp[i][j]=dp[i-1][j]。

三：当i>1,j>1时，如2\*2的表格来看，dp[2][2]的结果可以由子问题dp[1][2]和dp[2][1]组成。而dp[1][2]和dp[2][1]又分别属于上述前两种情况。当机器人处于2\*2的位置时，不管向右还是向下做出选择，都会进入其中一个子问题。

dp[1][j]=1,dp[i][1]=1,dp[1][1]=1。

dp[i-1][j]+dp[i][j-1] =dp[i][j] i>1,j>1

（当然这是重终点作为子问题起点经行分析，我们也可以重起点作为子问题起点来分析）

一：当i=1时，j>1时或者i>1,j=1时，机器人只能一直向右或者一直向下，路径只有一条。

二：当i>1,j>1时，假设还是2\*2，我们可以当前由两种选择，向右或者向下，路径可供机器人选择。和明显，这时我们需要以开点为初始位置，我们现在想要走到2\*2的右下角。dp[2][2]这个较大规模问题需要向右dp[1][2]或者向下dp[2][1]才能进入。

3.已经解决了前两步，第三步其实也觉解决了。那就是递推公式中的那两种特殊情况，加上就在原点的那种情况。即

dp[1][j]=1,dp[i][1]=1,dp[1][1]=1。

dp[i][j]=dp[i-1][j]+dp[i][j-1] i>1,j>1

最后的总结与反思：

1. 在什么情况下可以使用动态规划，首先需要先进行基本的判断。遇到的问题是否可以进行分解，是否可以分解为同类型且更简单的小问题。（当然这方面我更建议多看几个例子，最好多看几个，达到自己能够先判断出该问题是可以分解的。子问题是否有重复先别考虑。
2. 在推到递推公式时，重最开始的状态开始考虑，考虑最简形态，并且可以得出问题初始化值。以及是否全部涉及所有最简初始状态，也就是递推公式是否考虑周全，有没有漏掉什么情况。
3. 程序员大忌，理论建立于实践切不可分而谈之，不要认为自己理论建立会了，代码就会敲了，一定义将理论付诸到实践。比如，在

这个问题其实是一个非常简单的动态规划问题，但是对于我们建立思维过程非常重要，题目的目的不在于会解答，在于总结与反思，自己是否建立了一个逻辑正确且清晰的思维过程。问题也不在于困扰自己，在于有没有引导自己想到这方面的缺陷，有没有考虑得当。学习也不在于分数，而是在于自己到底有没有学到知识，有没有建立自己的认知系统，有没有打开自己的眼界，有没有对以前的认知进行纠偏。