**《算法设计与分析》实验报告**

我保证没有抄袭别人作业!

1. **实验名称**

动态规划问题

1. 最长公共子序列
2. 矩阵连乘问题
3. 收益最大化问题
4. **问题描述**
5. 给出最长公共子序列问题的最长公共子序列和长度
6. 给出矩阵连乘问题的最佳方案和次数
7. 面𝑛×𝑛个格子组成的二维表格，每个格子里有一个正整数。

让你从左上角的格子出发，只能向下或向右，走到右下角的格子，路径

上经过的数字之和作为收益。如何最大化你的收益？请1）给出问题的最优解的递归表达式。

2）设计一个动态规划算法求解该问题，并分析算法的时间和空间复杂性。

1. **算法设计与分析**

**1.算法选择**

动态规划问题具有以下2个特征：

动态规划的一个特征，重叠子问题结构（解决方法，记表)

另一特征是优化，要求问题最优解，而它依赖于子问题的最优解，如果子问题不是最优解就会导致整个问题不是最优解，即最优子结构性质

**2.算法设计**

**算法名称**：最长公共子序列

**输 入：**两个字符串X,Y

**输 出：**最长公共子序列LCS

**算法步骤：**

1.递归法

(1) 将问题拆解为更小一步, 求前缀Xi，前缀Yj的最长公共子序列Zk。

Zk总与xi和yj其中一个相等，判断xi,yj和zk与x,y是否相等，

(2) 如果x和y不等，zk和x不等，Z可以看做是Xm-1和Y的LCS(缩小问题)

如果x和y不等，zk和y不等，Z可以看做是Xm和Yn-1的LCS

如果x=y,说明LCS(X,Y)=LCS(Xm-1,Yn-1)+<xm=yn>

2.动态规划法：

(1)用表格c[i][j]来记录Xi和Yj的LCS长度，

(2)初始化表格边界为0

(2)仿照递归关系，判断x,y。但是将递归函数改为向左(Xm-1,Yn)或向上(Xm和Yn-1)或左上(Xm-1,Yn-1)

**算法名称**：矩阵连乘

**输 入：**矩阵链A1A2A3A…

**输 出：**乘法次数最少的组合

**算法步骤：**

初始化m[1,n]表对角线为0

for r=2;r<n-2矩阵链长度

for i=2;i<n-r-1起始位置范围

j=i+r+1;

计算初始m[I,j]大小，并记录初始分割点

for (k=i+1;k<j;k++)

不断尝试I,j中间断点，找到m[I,j]最优解，并记录分割点

**算法名称**：收益最大化

**输 入：**m\*n的矩阵

**输 出：**从左上角走到右下角能获取的最大收益

**算法步骤：**

检查是否为所给的收益矩阵边界，

如果是，返回该格子的收益

否则，目前的最大收益为左一格和上一格的最大值加上本格的收益

**3.算法复杂度分析**

1. **最长公共子序列**

递归算法:递归树的深度为(m+n),而两个序列的长度每层-1.总共可以生成完整的二叉树,因此时间复杂度是2^(m+n)

空间复杂度需要递归栈，而最大的递归树深度为m+n,所以为O(m+n)

动态规划法:

需要遍历整个表格，时间复杂度为O(m\*n)

空间复杂度:需要额外的m\*n的表格，O(m\*n)

1. **矩阵连乘**

递归算法:

将矩阵链分为两部分,前k个和后n-k个，再加上两部分矩阵的计算量pi-1pkpj，求解整个矩阵链的最优划分分解，求解第一层最佳划分下的第二层最佳划分..

T(n)=∑k=n(T(k)+T(n-k)+1)

该递归式可以得到T(n)是O(2^n),

动态规划:

需要填充整个表格，而每个格子需要遍历所有可能的分割点，所以时间复杂度可以看为O(n^3)

**3. 收益最大化问题**

递归算法:

从最右下角开始递归调用，每次递归只能将横纵坐标减一，最差情况递归树的深度可以达到m+n,而因为每次递归都没有存储结果，导致每次可以分出两个子树，最差情况会生成一个完整的二叉树，此时时间复杂度达到O(2^(m+n)),

没有额外调用空间，仅考虑函数栈的调用，空间复杂度为O(m+n)

而改用表格进行记录后，只需要遍历整个二维表格，时间复杂度会变为O(m\*n),因为额外多加了一个m\*n的表格，空间复杂度为O(m\*n)

1. **实验测试**

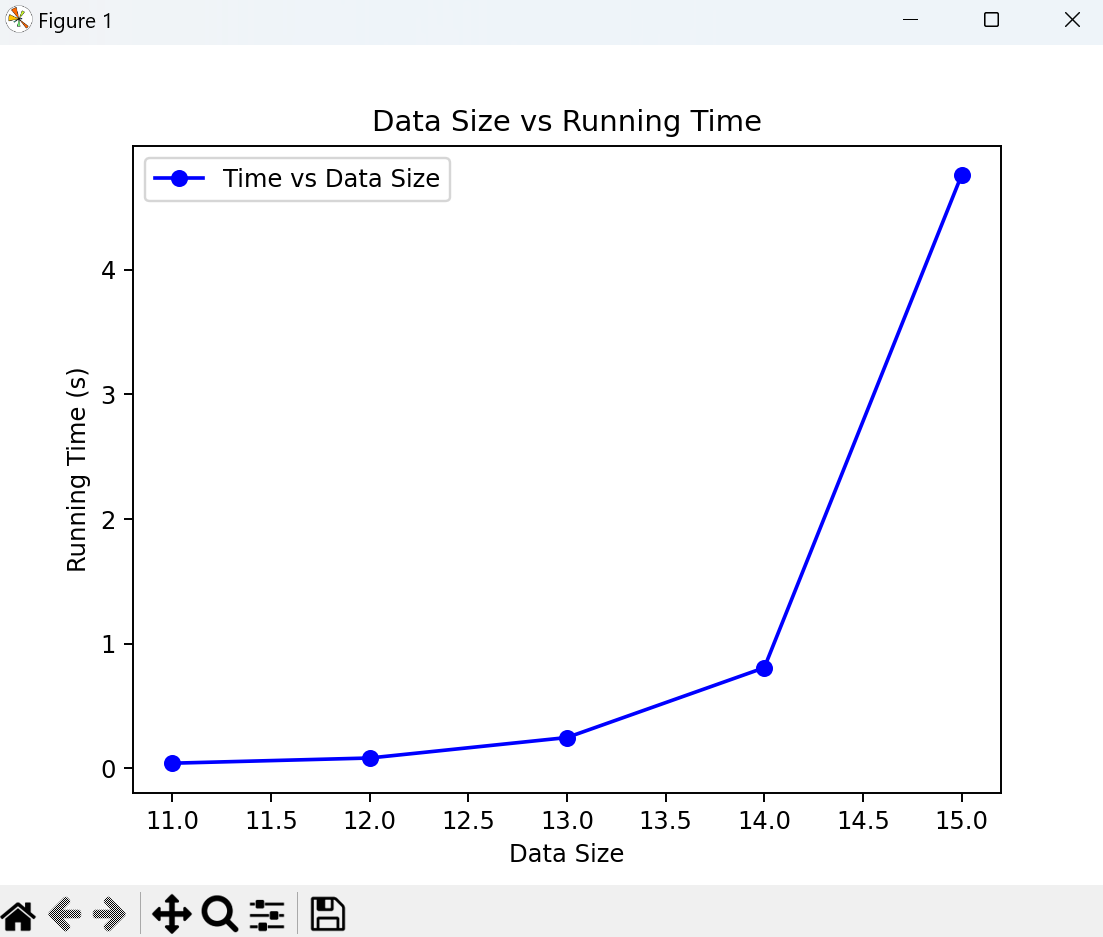
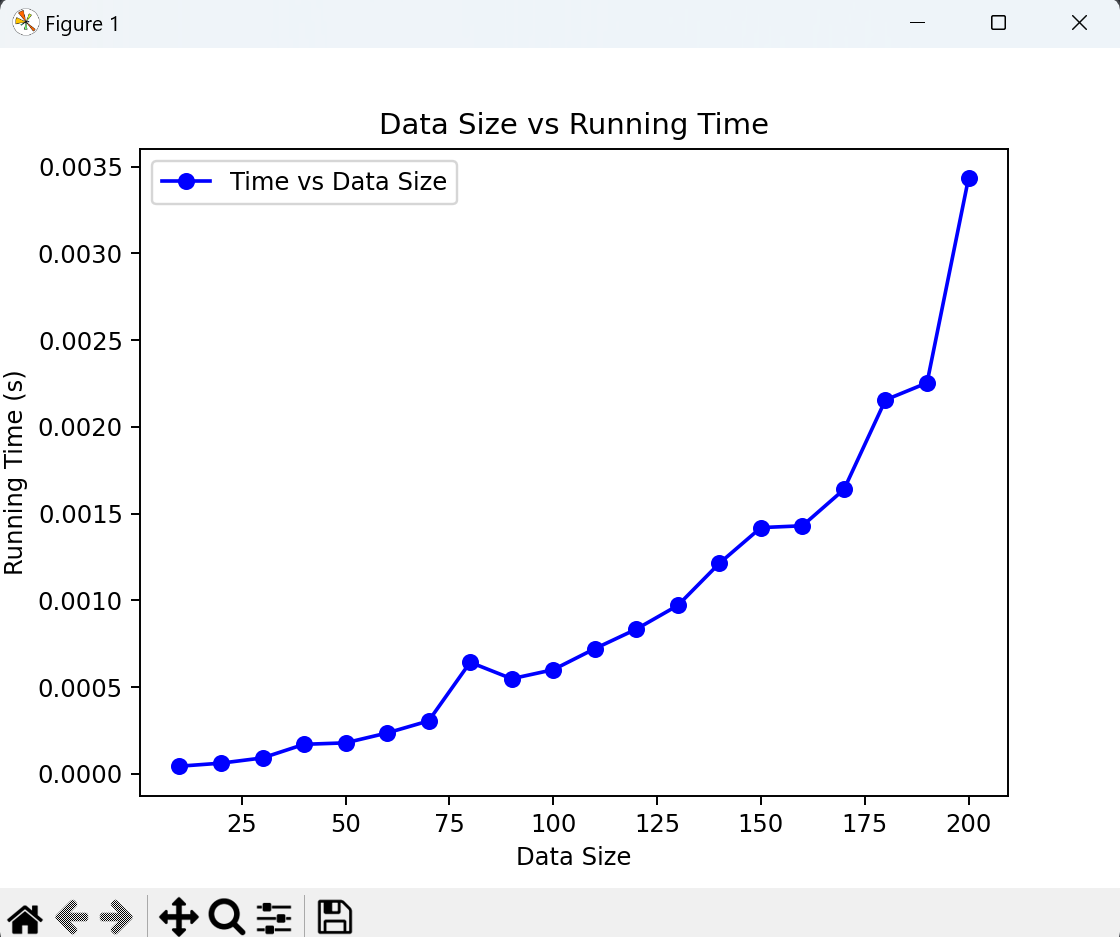
**测试用例和结果分析**

经过实验一的繁琐的数据测试，这次的测试用例改善，由Testdata.hpp文件中的Testdata类函数生成，可以生成一维数组，乱序或有序，二维数组，以及随机字符串，可以根据数据所需求的规模设定生成测试用例。而测试时间由Timetracker.hpp中的measuretime函数实现，可以根据具体算法可以调整运行次数，以得到函数较为准确的运行时间，并且避免运行时间过久。

最后测试用例的数据规模会记录到datascale.txt文件中，而对应的运行时间记录在file.txt文件中，file为算法名称，再经过Draw.py(该文件调用了python的matpilot库)将其绘制为坐标图，以更加直观的方式展现运行时间随着数据规模的变化，更好的进行性能分析。

操作步骤，先运行算法.cpp文件，生成两个txt数据文件，再运行python文件根据数据文件进行绘制，即可复现以下测试用例的结果（由于测试数据每次随机生成，可能略微不同，为确保数据来源真实以及结果准确，会保留展示的测试用例数据，放在工作目录下的testdata.txt，同时便于进行算法的正确性验证）

1.最长公共子序列

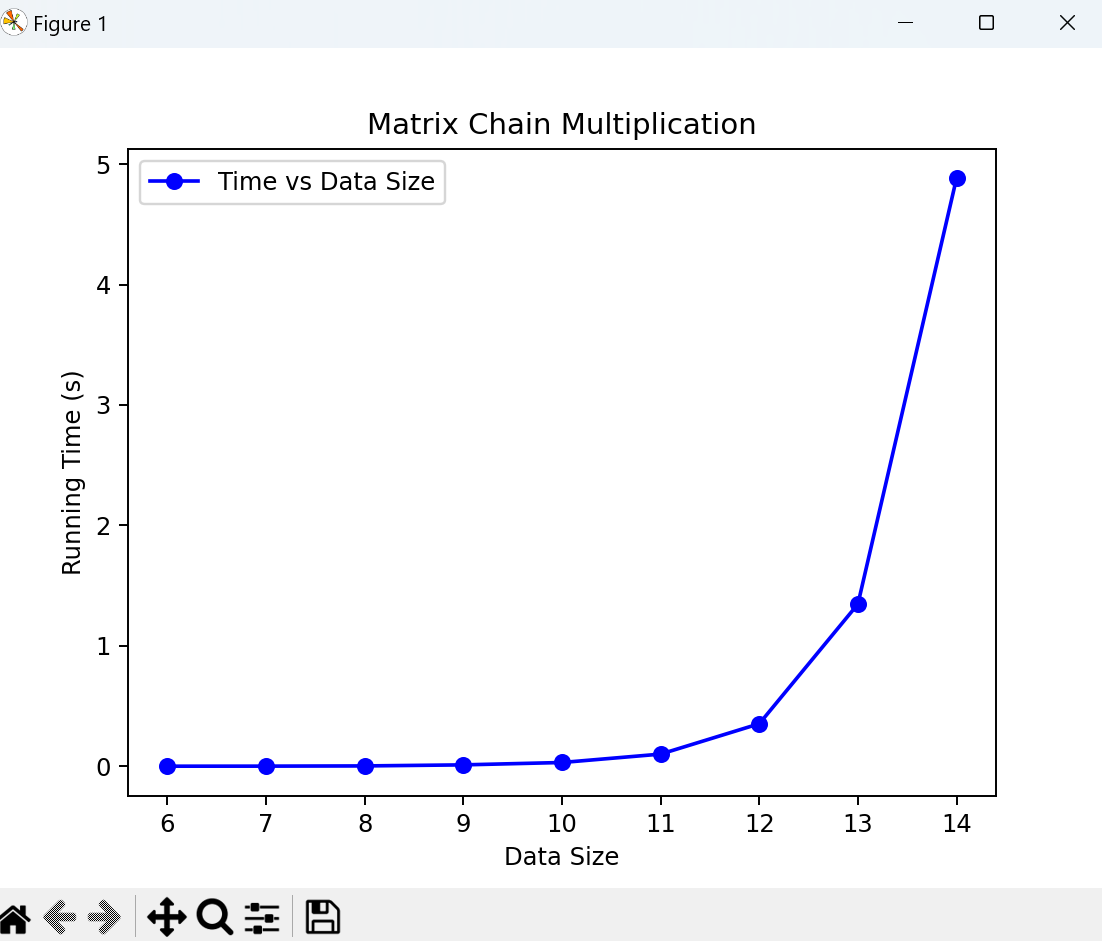
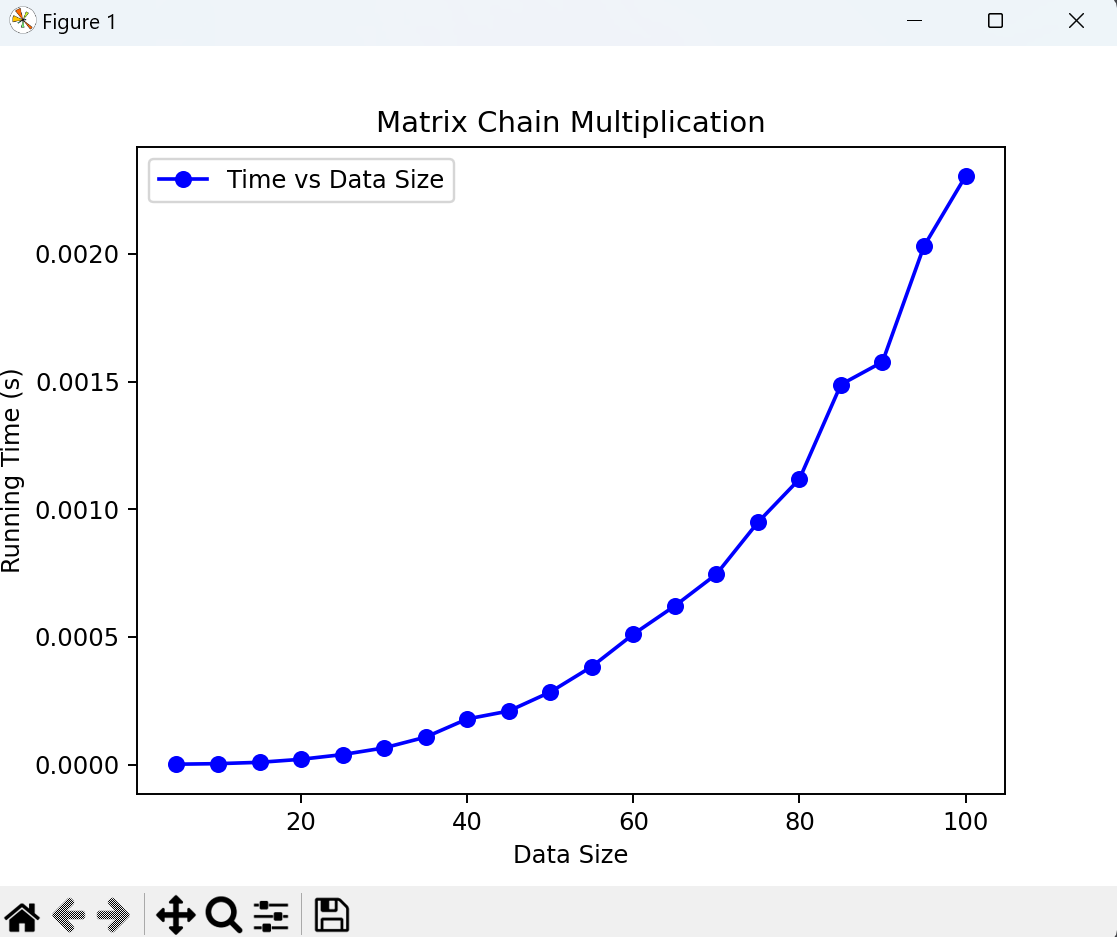


LCSR1在运行到第二个数据规模为20的时，便超出了预期时间，等待近五分钟后无果，便停止测试，仅一个数据，此时调整测试数据，将时间测试循环改为1次，并减小数据规模变化，依次+1，也只到16便超出预期，这里停止测试。

可以看到LCSR2在随着数据规模增大，符合O(nm)的时间复杂度，图中存在一些突变点也可以接受，在数据规模没有足够大时，会产生一些“偶然”情况，该组的两个字符串的LCS恰好较长等因素，会导致一些突变点，如果继续增大数据规模可以避免该情况，图像会更加标准,但这里当采用数据规模达到200后，会出现c++标准库函数的内存分配出现警告，后续便不再尝试。

采用动态规划方法要比LCSR1快很多，看到15时测试时间就已经超过了4秒,按照O(2^n)时间复杂度，16时的时间将大幅增加,也符合实验情况。

2.矩阵连乘



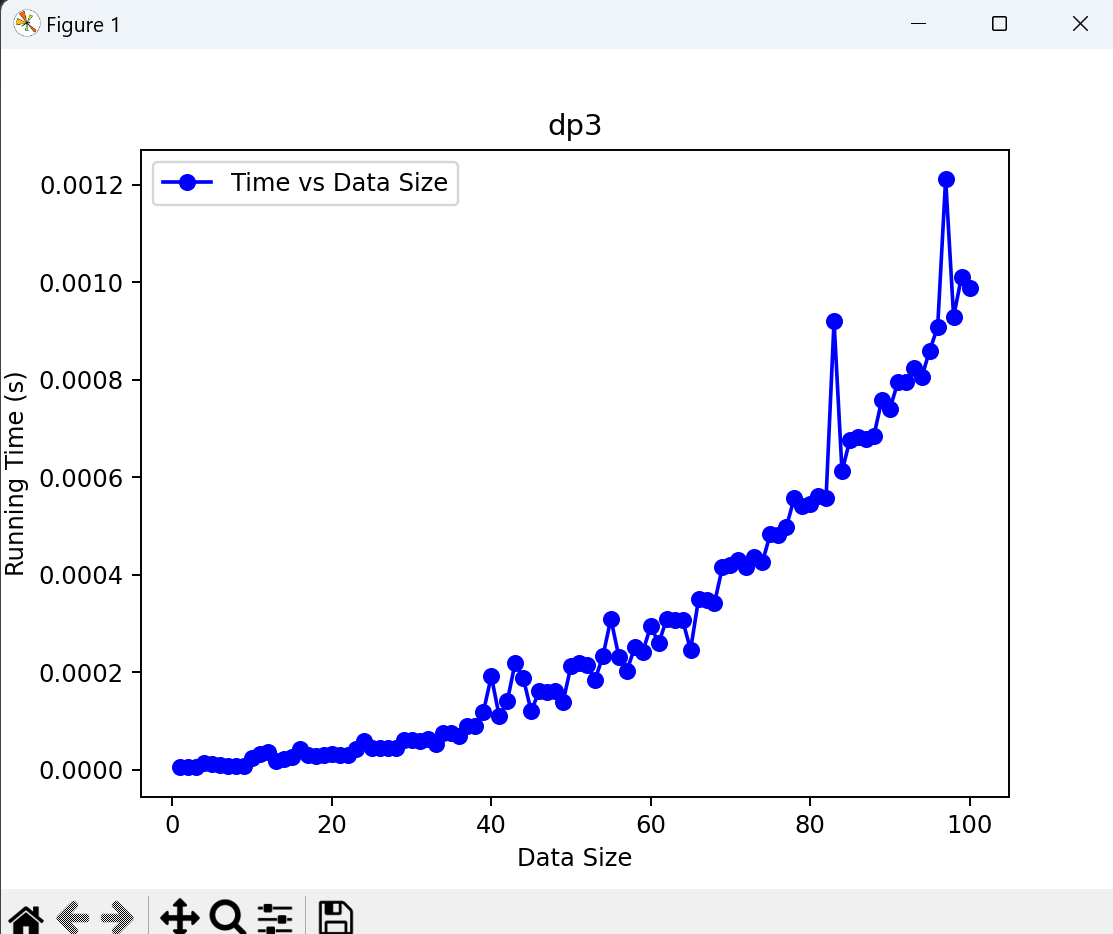
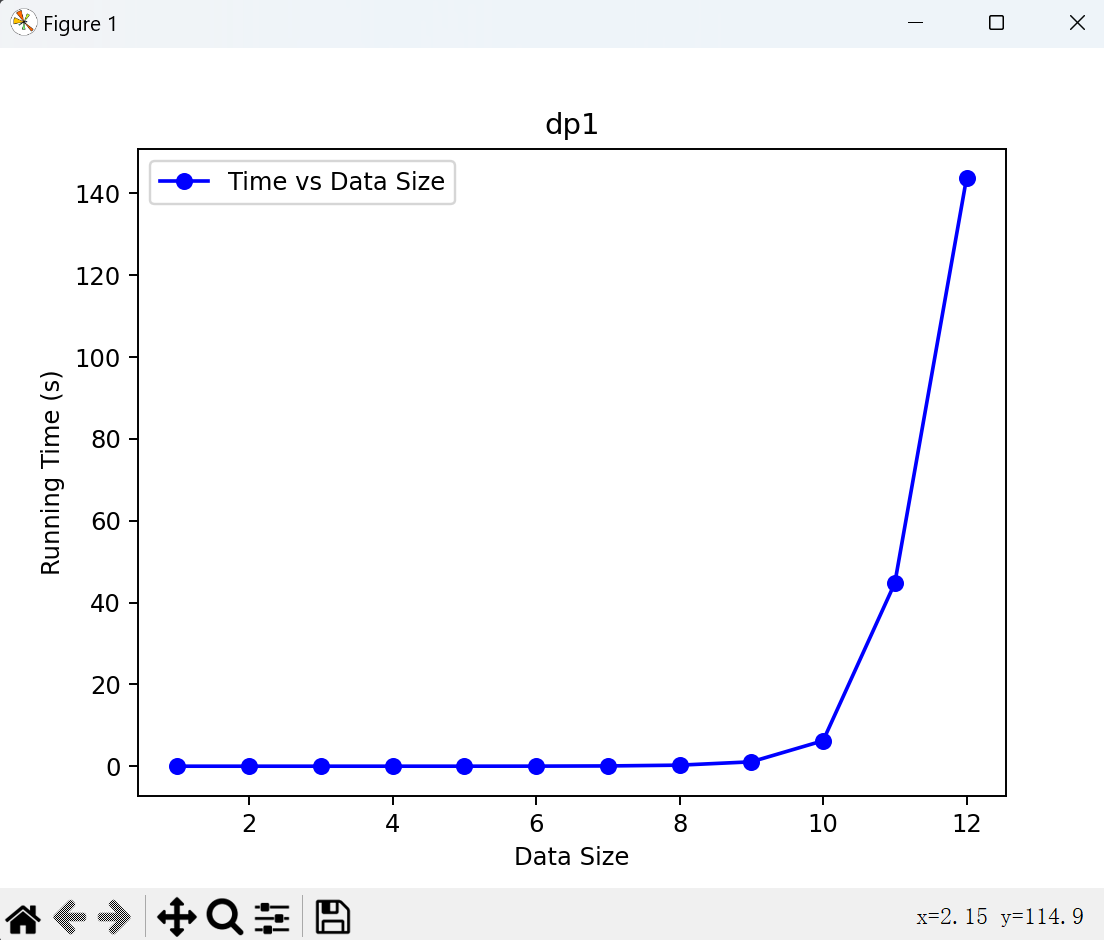
矩阵连乘的p可以看做是随机生成的一组以为数组，这里的Data Size表示该数组的大小，减一也就是矩阵的个数，二者是线性关系，且相差不大。所以可以用该数组的大小来代表矩阵个数，即数据规模。

测试用例选择到100是因为此时存储测试数据以及结果的数据文件testdata.txt已经挺大，考虑到将来文件需要打包上传，而继续增加数据规模会导致该文件过大，将来不方便上传等。并不是因为时间等性能原因截止。

但是100的规模已经可以看到该运时间随着数据规模变化的趋势，符合O(n^3)的趋势。

右图为MatrixChain2的运行时间，即采用递归而不用表格记录的方式，看到随着数据规模增大急剧增大，且运行时间要比动态规划方式要长的多，在15时等待了过久便强制停止。

1. 收益最大问题



其中dp1是仅按照递归进行的，在数据规模达到12时运行时间便超过120s，过长不再等待，符合时间复杂度为O(2^n)，dp3是按照动态规划思想设计的填表，运行时间相比有极大提升，且数据规模100时也能很快完成。为方便处理，这里测试数据都采用的n\*n,reward表格长宽相等，所以时间复杂度可以视为O(n^2),也符合图的特征，和前述一样，考虑防止testdata.txt过大，没有向其中记录更大数据。自己尝试没有记录数据的情况下，在运行到6000\*6000时，运行时间大概是5-6s,且没有溢出，后续没有继续尝试。

由于这次”偷懒”（也是想看看不采用多次循环采取运行时间平均值情况下的图像），在测试运行时间时仅运行了一次，所以会出现dp3上下跳动的情况，而之前由于都是运行100（或多次），所以图形点较为平滑

六.实验总结

在以不断方便自己的目的指引下，相比实验一，从手工整理测试数据，手工改变数据规模，手工分析每次测试的结果，这次实验做了生成测试数据，绘制结果图等一系列自动完成，虽然测试数据等函数是依靠大模型辅助写的，但是在此过程中也学习到了很多东西，例如如何让大模型生成更准确，更符合自己需求的代码，以及对c++的语法细节等更加熟悉，且在与同学的交流中得知了python的matplot库绘制图形的方便，为此又简单做了python绘制图形的函数，利用文件将由C++算法生成的结果用python绘制。

实验主题还是动态规划，动态规划没有一个固定的解法，面对不同问题有不同解法，且时间复杂度虽然大多数可以达到O(m\*n),但也不一定，例如矩阵连乘问题。他更像是一种思想，即以空间换时间，且性价比极高，特别是在今天运行时间往往比存储空间更重要的情况下，这一点也让我联想到计组中的微程序的两种设计思路，一个是微程序存储器法和组合逻辑控制法，前者利用ROM提前存储，后续用时直接查找（扯远了）。虽然数据规模不是很大，但是结果图可以很明显看出采用动态规划法后算法时间性能的急剧提升。