****



**算法设计与分析课程报告**

**关于****动态规划算法的算法验证**

**专 业 工科试验班**

**班 级 7班**

**小组成员 王红阳 3019244233**

**艾力扎提·艾合卖提**

**史峪 3019244234**

**侯乐岩**

目录

[一、 摘要 3](#_Toc11477)

[二、 贡献 4](#_Toc23637)

[三、 实验目的 5](#_Toc21263)

[四、 实验设计流程 6](#_Toc864)

[五、 实验原理及实现 8](#_Toc2145)

[1. 0/1背包问题 8](#_Toc32047)

[（1） 代码结构 8](#_Toc584)

[（2） 算法实现原理 8](#_Toc15268)

[（3） 递归实现和迭代实现的区别 8](#_Toc7179)

[2. 矩阵乘法链问题 9](#_Toc25108)

[（1）文件结构 9](#_Toc29743)

[（2）算法思想 9](#_Toc873)

[（3）代码实现 9](#_Toc28297)

[3. TSP问题 11](#_Toc26225)

[（1）文件结构 11](#_Toc2001)

[（2）算法思想 11](#_Toc10652)

[（3）代码实现 13](#_Toc3814)

[4. 所有点对最短短路径问题 14](#_Toc10679)

[（1）文件结构 14](#_Toc14630)

[（2）算法思想 14](#_Toc27490)

[（3）代码实现 14](#_Toc29314)

[六、 实验结果及分析 17](#_Toc1364)

[1. 0/1背包 17](#_Toc3897)

[2. 矩阵乘法链 18](#_Toc14765)

[3. All\_pairs最短路径问题 19](#_Toc29787)

[4. TSP问题 20](#_Toc7907)

[5. 数据集分布特点对算法性能的影响 21](#_Toc13704)

[七、 动态规划算法的特点 24](#_Toc32014)

[1. 最优子结构 24](#_Toc22922)

[2. 无后效性 25](#_Toc27295)

[3. 子问题重叠性质 25](#_Toc17582)

[4. 实现策略 25](#_Toc19402)

[（1）自底向上策略 25](#_Toc32529)

[（2）自顶向下(备忘录)策略 25](#_Toc12830)

[5. 优缺点分析 25](#_Toc32319)

[八、 实验总结 26](#_Toc17967)

1. **摘要**

**为充分落实在本课程上的所学内容，我们小组通过实验对动态规划算法进行了算法验证。**

**本实验利用动态规划算法，针对四个问题（0/1背包问题及其9个数据集，矩阵乘法链问题及其6个数据集，所有点对最短路径问题及其9个数据集，TSP问题及其3个数据集），进行代码实现，并对程序进行性能分析，从而实现对动态规划算法的验证。**

**在实验过程中，我们分析了数据集分布特点对算法性能的影响（待定），归纳了算法解决不同问题的优缺点，即：递归实现该算法时，即自顶向下运行，只需要求解该问题需要的解，不需要求解所有子问题，但是它可能会重复计算某些子问题，需要额外的递归开销；迭代法实现该算法时，即自底向上运行，必须对所有子问题进行求解，不需重复计算子问题，可有效减少计算时间和所需的存储空间。**

**通过本次实验，我们掌握了利用动态规划算法进行问题求解的实验流程，强化了对课堂所学算法原理的理解，并且提升了利用算法原理进行编码实践的能力。**

1. **贡献**

数据集输入与结果输出（史峪、艾力扎提）

递归、迭代算法实现（艾力扎提）

0/1背包问题

负责人:艾力扎提

算法分析（艾力扎提）

算法实现（王红阳）

实验报告

负责人:史峪

实验报告总体排版(摘要、贡献、实验目的、实验设计流程):史峪

四项实验实验原理：王红阳、艾力扎提

实验结果分析：史峪、王红阳、艾力扎提、侯乐岩

实验总结:史峪

数据集输入与结果输出（王红阳）

矩阵乘法链

负责人:王红阳

统筹规划将整个实验进行拆分

算法分析（王红阳、侯乐岩）

数据集收集（侯乐岩）

算法及头文件（艾力扎提）

All-pairs最短路径

负责人:艾力扎提、

王红阳

数据集的输入和结果输出、算法实现（王红阳）

算法分析(史峪、侯乐岩)

算法实现（王红阳）

数据集的输入和结果输出（王红阳）

TSP问题

负责人:王红阳

算法分析（王红阳）

1. **实验目的**
2. **掌握利用算法进行问题求解的实验流程**
3. **强化对课堂所学算法原理的理解**
4. **提升利用算法原理进行编码实践的能力**
5. **实验设计流程**

分析数据集分布特点对算法性能的影响

实验报告撰写

实验结果汇总

实验设计流程

运用模块化方法编写

TSP问题的实现

对矩阵乘法链问题的实现

**对所有点对最短路径问题的实现**

对0/1背包问题的递归与迭代实现

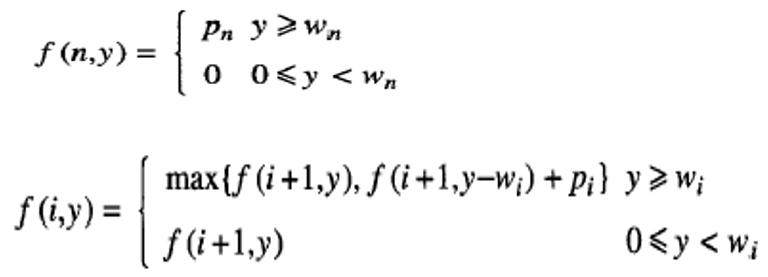
归纳算法解决不同问题的优缺点

1. **实验原理及实现**
2. 0/1背包问题
3. 代码结构

该算法实现在DP\_knapsack文件夹中。背包问题动态规划算法实现分为递归实现和迭代实现两部分。递归实现代码在recursion.cpp文件中，迭代实现算法在iteration.cpp文件中，上述两个.cpp文件是整个背包问题动态规划实现的核心算法文件。除此之外，read\_data.cpp、output.cpp中分别包含了数据集输入和结果输出代码、knapsack.h中包含了所有全局变量和一些常用的初始化函数、test\_all.cpp文件中包含了整个项目运行的主函数。

1. 算法实现原理

该算法的实现分为递归实现和迭代实现两部分。两种实现方法原理相同即令f(i,y)等于从第i到第n个物品约束为y的背包问题的优化值并利用下述递归表达式：



计算f(1,c)就等于背包问题的优化效益值。再利用回溯算法得到优化解。回溯算法原理为若f(i,y) = f(i+1,y)，则xi = 0，否则xi = 1并且y = y-wi。

1. 递归实现和迭代实现的区别

递归实现不需要存储计算的数据，这导致重复计算的数据过多。而迭代实现利用n\*c的二维数组存储数据，解决了重复计算的问题，但又产生了当n或c过大时，算法空间复杂度过大的问题。

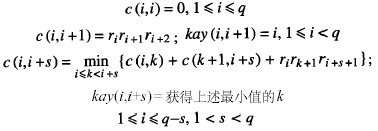
1. 矩阵乘法链问题

（1）文件结构

用于求解该问题的文件夹为：MatrixMultiplicationChain 该文件夹内包含文件夹DataSet（其中包含输入数据集文件），文件夹Answers（其中包含输入文件对应的输出文件，即问题的解），Makefile，readme.md，path.h（包含输入文件及输出文件的路径），DP.exe,DP.cpp(包含实现该算法的主程序）

（2）算法思想

定义c(i,j)为计算M(i,j)的优化乘法数(优化值)

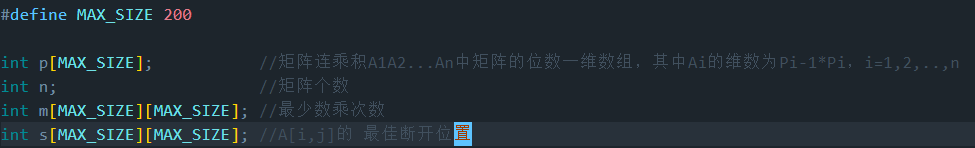


（3）代码实现

因为递归法可能重复计算子问题，浪费不必要的时间和空间，所以我选择了使用迭代法实现该问题(即三层for-loop)，在计算过程中，保存计算的每个c 和kay值, 可避免大量重复计算.但需O(n2)的存储空间。

DP.cpp中包含m[n][n](m[i][j]为计算M(i,j)的优化乘法数）等变量，计算这些变量的函数DPMatrixChain（），输出最优解的函数PRINT\_OPTIMAL\_PARENS（），用于输出数组的函数print\_array（），输入数据集的函数inputData（），输出计算结果及运行时间的函数outputAnswers（）。在main()函数中，对不同数据集进行输入，求解，输出等流程。

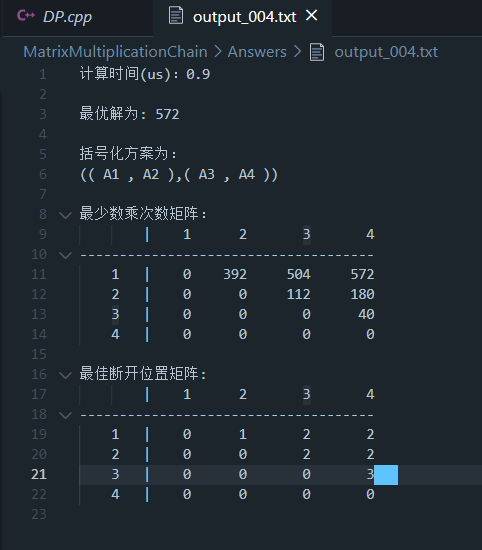
主程序DP.cpp的主要变量：



DPMatrixChain为计算最少乘法数等变量的函数：



输出的解的格式如下：



1. TSP问题

（1）文件结构

用于求解该问题的文件夹为：TSP问题 该文件夹内包含文件夹DataSet（其中包含输入数据集文件），文件夹Answers（其中包含输入文件对应的输出文件，即问题的解），（由于在老师发放的该问题的数据集网站中，对每个数据集都配有有标准答案，所以该文件夹中还包含）文件夹Standard\_Answers（用于存储每个输入文件的标准答案），Makefile，readme.md，path.h（包含输入文件及输出文件的路径），global.h(用于存储全局变量及全局变量的初始化）DP.exe,DP.cpp(包含实现该算法的主程序），verify.exe,verify.cpp(验证DP.cpp对每个输入的数据集计算的答案是否正确）

（2）算法思想

TSP问题是指旅行家要旅行n个城市，要求各个城市经历且仅经历一次然后回到出发城市，并要求所走的路程最短。各个城市间的距离可以用代价矩阵来表示。

假设从顶点s出发，令d(i, V’)表示从顶点i出发经过V’(是一个点的集合)中各个顶点一次且仅一次，最后回到出发点s的最短路径长度。

推导：(分情况来讨论)

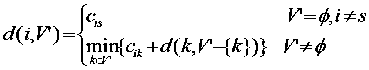
①当V’为空集，那么d(i, V’)，表示从i不经过任何点就回到s了，如上图的 城市3->城市0(0为起点城市)。此时d(i, V’)=Cis(就是 城市i 到 城市s 的距离)

②如果V’不为空，那么就是对子问题的最优求解。你必须在V’这个城市集合中，尝试每一个，并求出最优解。

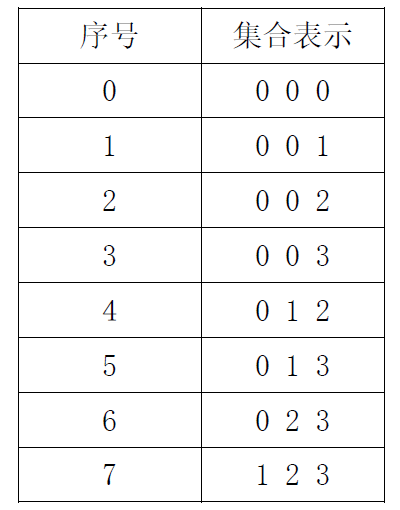
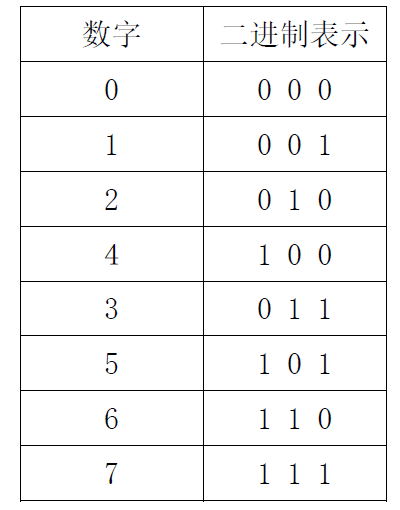
d(i, V’)=min{Cik + d(k, V’-{k})}

注：Cik表示你选择的城市和城市i的距离，d(k, V’-{k})是一个子问题。

综上所述，TSP问题的动态规划方程就出来了：

[](https://p-blog.csdn.net/images/p_blog_csdn_net/gfaiswl/612517/o_image_6.gif)

接下来，需要使用位与运算。为对n个顶点分别用0~n-1的数字编号，按个数为1,2,……,n-1的顺序生成1~n-1个元素的子集存放在数组V[2^n-1]中。这样，在判断时就可以将当前点与数组编号按位与运算（如下图所示），为0则为需要进行计算



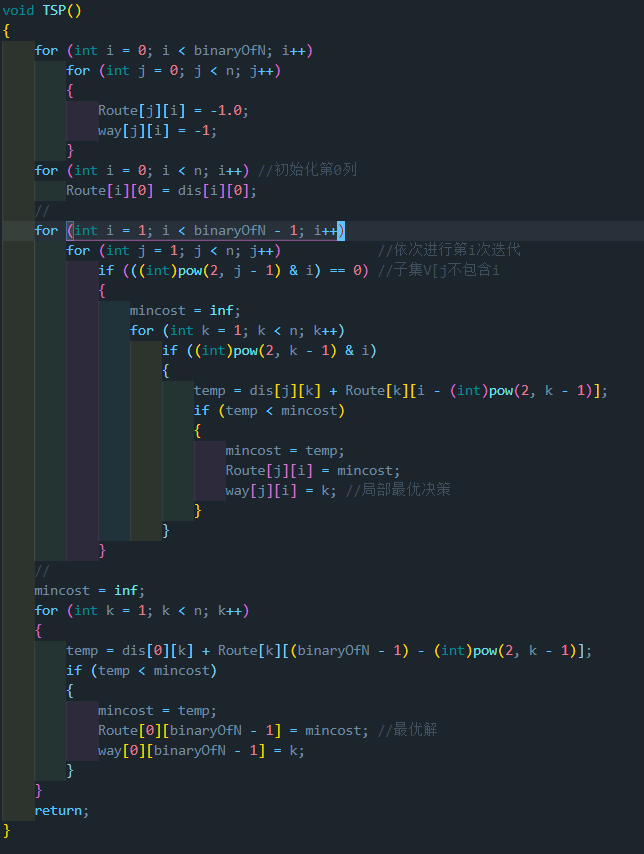
（3）代码实现

使用迭代法实现该问题(即三层for-loop)，可计算所有可能的解，没有遗漏。在计算过程中，保存计算的每个变量的值,但需O(n2)的存储空间且时间复杂度为O(n22n)。

DP.cpp中，计算这些变量的函数TSP（），输入数据集的函数inputData（），输出计算结果及运行时间的函数outputAnswers（）。在main()函数中，对不同数据集进行输入，求解，输出等流程。

Verify.cpp中，对不同输出答案文件进行验证。

该程序的核心部分TSP函数实现如下：



1. 所有点对最短短路径问题

（1）文件结构

用于求解该问题的文件夹为：所有点对最短短路径问题 该文件夹内包含文件夹DataSet（其中包含输入数据集文件），文件夹Answers（其中包含输入文件对应的输出文件，即问题的解），Makefile，readme.md，path.h（包含输入文件及输出文件的路径），global.h(用于存储全局变量及全局变量的初始化）DP.exe,DP.cpp(包含实现该算法的程序）,all\_pairs.h(包含程序所需的主要函数）

（2）算法思想

最短路径: 假设G为有向图,其中每条边都有一个成本(cost),图中每条有向路径的长度(或成本)定义为该路径上各边的成本之和.

定义c(i,j)=i到j的最短路长度，用kay(i,j)记录该路径经历的节点

因为只考虑简单路径, 所以初始化:c(i,i)=0，c(i,j)=∞(i≠j)，kay ={0}

接下来，建立动态递归方程：c(i,j)=min{c(i, j), c(i, k)+c(k, j)}，kay(i,j)为c(i,j)取得最小值的k值。

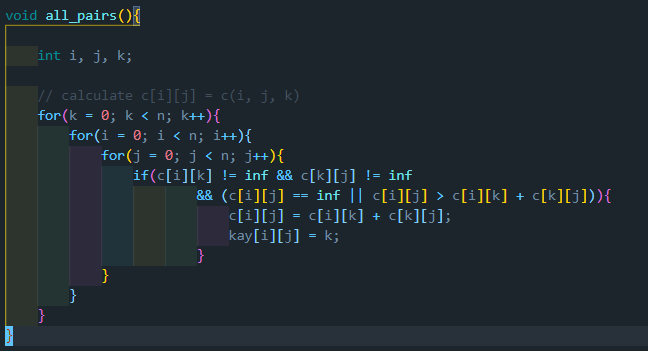
（3）代码实现

使用迭代法实现该问题(即三层for-loop)，在计算过程中，保存计算的每个c 和kay值, 可避免大量重复计算.但时间复杂度较大为O(n3)。

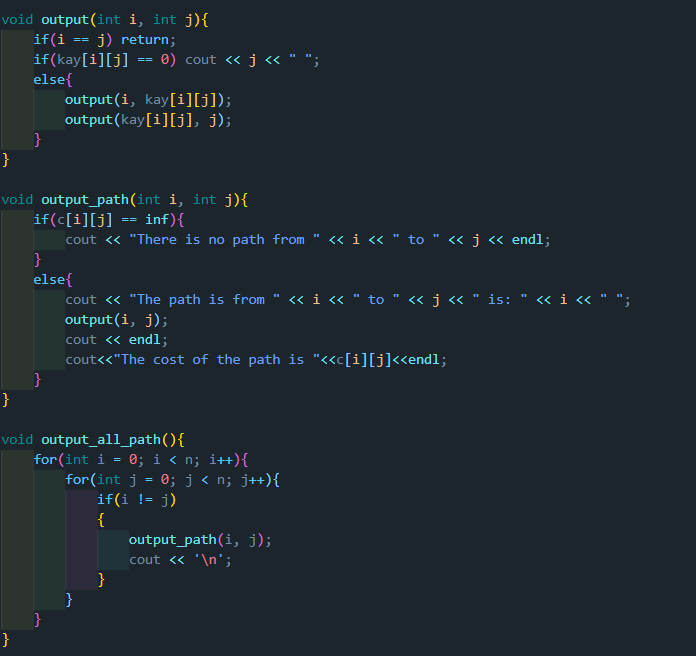
DP.cpp中输入数据集的函数inputData（），输出计算结果及运行时间的函数outputAnswers（）。在main()函数中，对不同数据集进行输入，求解，输出等流程。

all\_pairs.h中包含用于输出路径及成本的系列函数output\_all\_path()，output\_path(),output()；计算c和kay的函数all\_pairs()

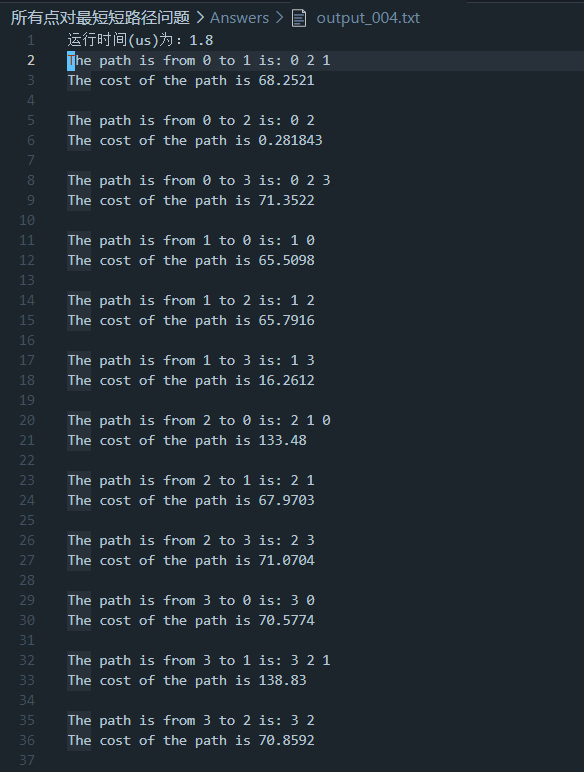
下面为all\_pairs函数的代码实现：



下图为用于输出路径的函数：



输出后的格式如下：



1. **实验结果及分析**
2. **0/1背包**

**Iteration方法**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **n** | **c** | **时间消耗(ms)** | **n\*c/t** |
| **1** | **4** | **20** | **0.00060** | **133333** |
| **2** | **5** | **26** | **0.00110** | **118182** |
| **3** | **10** | **165** | **0.00660** | **250000** |
| **4** | **10** | **269** | **0.01110** | **242342** |
| **5** | **15** | **750** | **0.05720** | **196678** |
| **6** | **20** | **878** | **0.07810** | **224840** |
| **7** | **23** | **10000** | **1.00730** | **286826** |
| **8** | **100** | **995** | **0.34690** | **228333** |
| **9** | **500** | **2543** | **5.39230** | **235799** |

**递归**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **n** | **c** | **时间消耗(ms)** | **总递归次数** | t/2^n |
| **1** | **4** | **20** | **0.00050** | **44** | 31.25 |
| **2** | **5** | **26** | **0.00120** | **94** | 37.5 |
| **3** | **10** | **165** | **0.00770** | **1593** | 7.51953125 |
| **4** | **10** | **269** | **0.01350** | **3209** | 13.1835938 |
| **5** | **15** | **750** | **0.34240** | **98141** | 10.4492188 |
| **6** | **20** | **878** | **13.82320** | **4177384** | 13.1828308 |
| **7** | **23** | **10000** | **74.72390** | **22631729** | 8.90778303 |
| **8** | **100** | **995** | **7239.06620** | **2145625934** | t/2^n |

**可以看到时间消耗随着n的增长，呈现指数型增长，同时，t/2^n的值趋向平缓，在n增长的情况，值有所下降。符合T=O（2^n）的情况，t的增长渐进小于2^n的增长。验证了对时间复杂度的估计**

**N\*c/t的值完美的在一条水平线附近，说明，t的增长速率与n\*c的增长速率趋于一致。验证了t=Ɵ（n\*c）**

1. **矩阵乘法链**

|  |  |  |
| --- | --- | --- |
| **n** | **消耗时间（us）** | t/n^3 |
| **4** | **0.3** | 468.75 |
| **8** | **1.2** | 234.375 |
| **16** | **5.9** | 144.043 |
| **32** | **32.6** | 99.4873 |
| **64** | **216** | 82.39746 |
| **128** | **1525.3** | 72.73197 |

**随着n的增长，时间开销也增长，但不如指数型增长的快。t/n^3的值随着n的增长，逐渐减小，但减小速率越来越慢，向水平线趋近。由此，时间复杂度=O（n^3）得到验证**

1. All\_pairs最短路径问题

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| T1(us) | 0.8 | 5 | 32.6 | 241 | 1899 | 15358 | 108370 | 845842 | 6.61754e+06 |
| T2(us) | 0.8 | 5.1 | 34.8 | 239.2 | 1877.4 | 14837 | 108963 | 839878 | 6.6288e+06 |

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **T1** | **T2** | averageT |
| **4** | **0.8** | **0.8** | 0.8 |
| **8** | **5** | **5.1** | 5.05 |
| **16** | **32.6** | **34.8** | 33.7 |
| **32** | **241** | **239.2** | 240.1 |
| **64** | **1899** | **1877.4** | 1888.2 |
| **128** | **15358** | **14837** | 15097.5 |
| **256** | **108370** | **108963** | 108666.5 |
| **512** | **845842** | **839878** | 842860 |

**本问题与矩阵乘法链的图的趋势走向近乎无差，随着n的增长，时间开销与n的关系为n^3，由图像可以轻松看出这一结论**

1. **TSP问题**

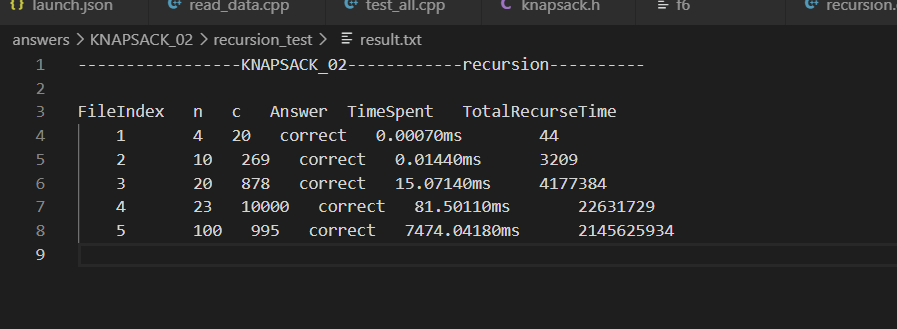
|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **5** | **15** | **17** |
| **T(us)** | **5.5** | **57700** | **285019** |
| **C\*T/n^2\*2^n** | **88000** | **100173.611** | **96311.0267** |

**随着n 的增大，时间开销随着增大，且增长速率非常惊人，当时通过算法估计的为n^2\*2^n，现在进行验证。通过图表计算得到C\*T/n^2\*2^n，其中C为常数，目的使图表更加直观，否则原结果太小，难以在图表上观察。看到C\*T/n^2\*2^n为一条近似的直线，可能是因为n的值还是偏小，难以得到准确结果，但由于时间复杂度增长太快，难以在电脑上进行更大数据量的计算。不过，这也近似验证了估计T(n)=O(n2\*2n)**

1. **数据集分布特点对算法性能的影响**

主要以0\1背包问题为例

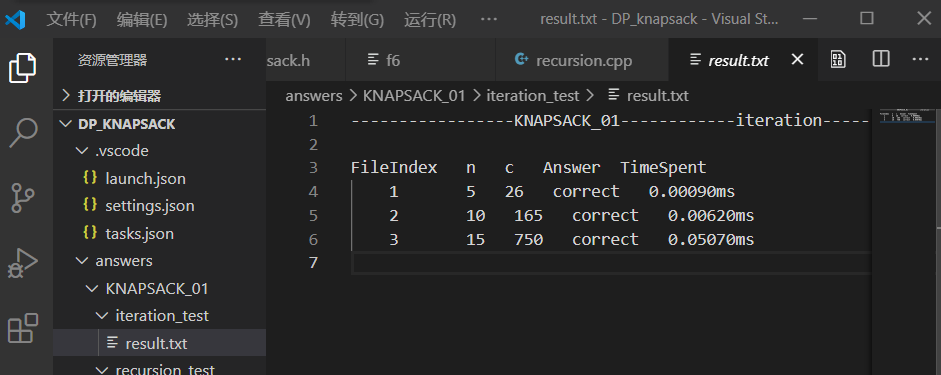
（1）在数据规模上

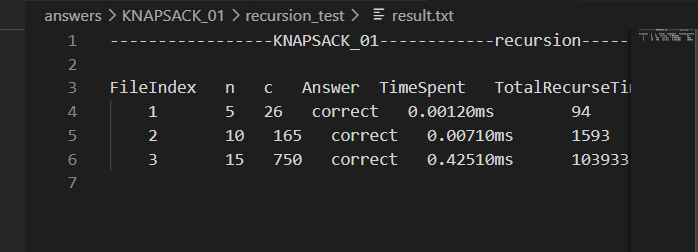


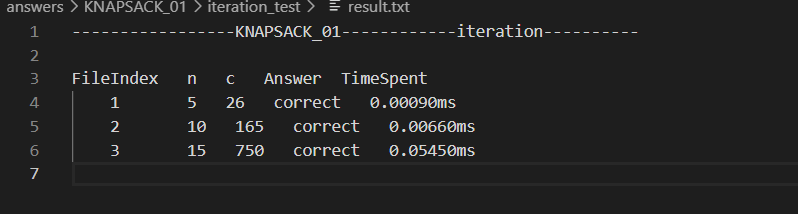
由图，随着数据规模的增加，时间开销也随之增长。

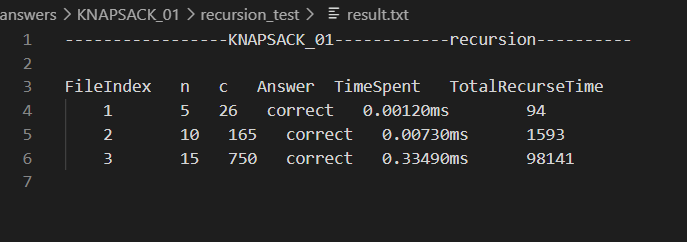
（2）在分布顺序上

将KNAPSACK\_01中的p03数据集数据顺序修改，按照最优解的顺序从上到下排列，前两张图为修改前iteration与recursion部分，后两张图为修改后的

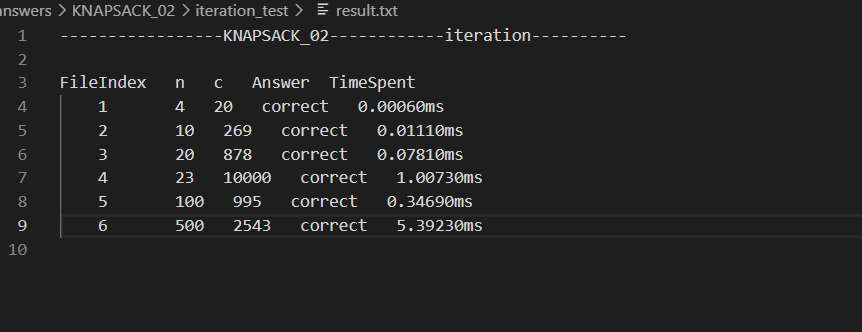


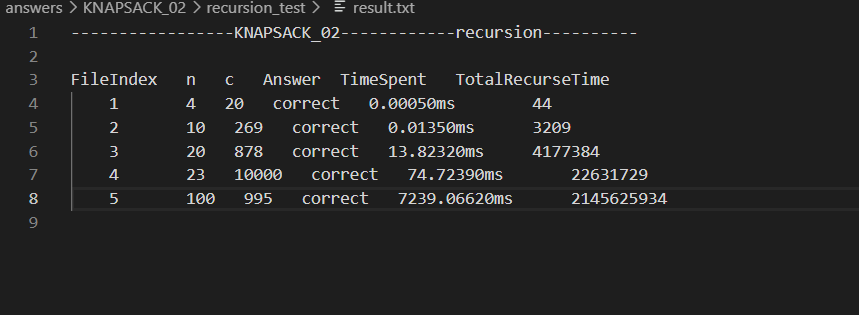


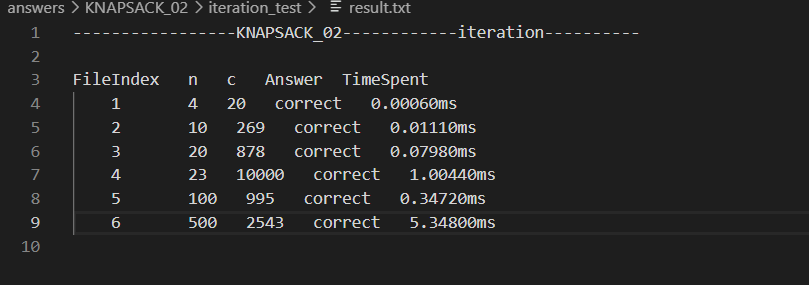


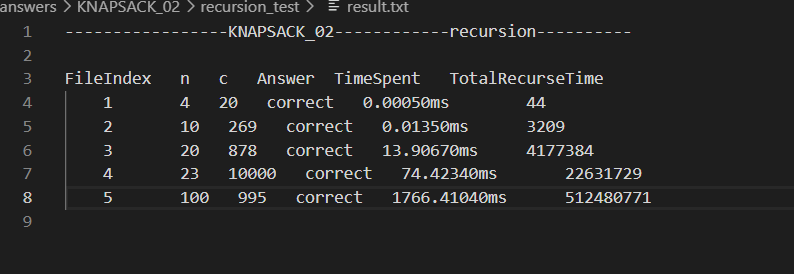


将KNAPSACK\_02中的f5数据集数据顺序修改, 按照最优解的顺序从上到下排列，前两张图为修改前iteration与recursion部分，后两张图为修改后的

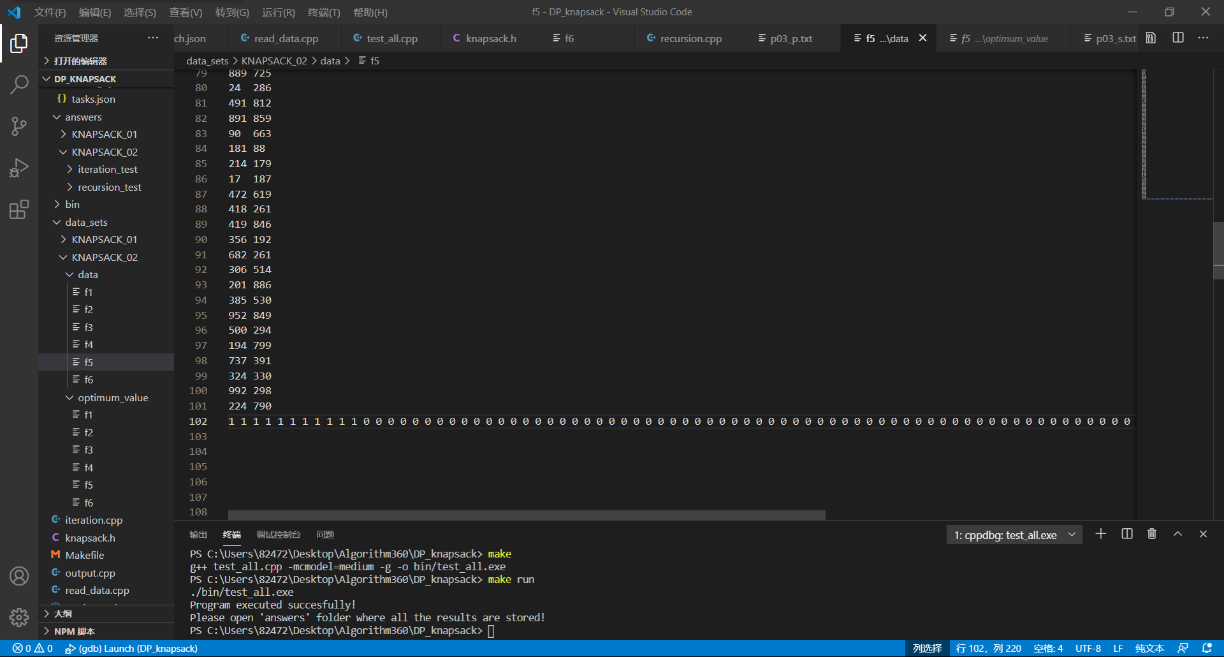








修改过程中



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | **改变顺序前** | | **改变顺序后** | |
| **消耗时间** | **递归次数** | **消耗时间** | **次数** |
| **P03** | **Iteration** | **0.0507** |  | **0.0545** |  |
| **Recursion** | **0.4251** | **103933** | **0.3349** | **98141** |
| **F5** | **Iteration** | **0.3469** |  | **0.3472** |  |
| **Recursion** | **7239.0622** | **2145625934** | **1766.4104** | **521480771** |

**得出结论：数据规模越大，算法时间开销越大，性能越低。数据集分布对算法性能有影响，初始数据越贴合最优解的码放次序，算法时间开销越小，性能越高。**

1. **动态规划算法的特点**
2. **最优子结构**

**如果问题的最优解所包含的子问题的解也是最优的，我们就称该问题具有最优子结构性质（即满足最优化原理）。最优子结构性质为动态规划算法解决问题提供了重要线索。**

1. **无后效性**

**即子问题的解一旦确定，就不再改变，不受在这之后、包含它的更大的问题的求解决策影响。**

1. **子问题重叠性质**

**子问题重叠性质是指在用递归算法自顶向下对问题进行求解时，每次产生的子问题并不总是新问题，有些子问题会被重复计算多次。动态规划算法正是利用了这种子问题的重叠性质，对每一个子问题只计算一次，然后将其计算结果保存在一个表格中，当再次需要计算已经计算过的子问题时，只是在表格中简单地查看一下结果，从而获得较高的效率。**

1. **实现策略**

（1）自底向上策略

**一般动态规划问题都是基于此策略。在用这种方法时一般需要恰当的定义子问题的规模，使得任何子问题都只依赖于更小的子问题的求解。我们可以将问题的规模按照由大到小排列依次求解。每个子问题都只求解一次**

（2）自顶向下(备忘录)策略

本实验0/1背包中Recursion方法使用了该方法。

动态规划有一个性质为子问题重叠性质，就是对于子问题在递归过程中不断求解，虽然问题规模很小，但是求解次数会非常多，造成程序运行非常慢。在使用自顶向下的求解过程中，我们一般要设计一个备忘录，在递归求解过程中对于已经求解过的问题保存在备忘录中，当下次要使用时直接拿出来，不用再次求解。

1. **优缺点分析**

**递归实现该算法时，即自顶向下运行，只需要求解该问题需要的解，不需要求解所有子问题，但是它可能会重复计算某些子问题，需要额外的递归开销；迭代法实现该算法时，即自底向上运行，必须对所有子问题进行求解，不需重复计算子问题，可有效减少计算时间和所需的存储空间。**

1. **实验总结**

**通过这次实验，我们掌握了通过实际问题验证算法的实验流程,强化了对课堂所学算法原理的理解,提升了利用算法原理进行编码实践的能力，增强了协调组织与队员们模块化编程的能力。**

**同时，我们对C++编程语言的理解更加深入，对处理中等项目有了足够的经验，也体会到算法在实际问题中的非凡效用，相信在以后的学习生活中会对我们大有裨益。**

**本次实验，我们利用四种问题验证了动态规划算法的合理性、正确性、高效性，分工明确，配合高效，总体来说，符合预期地完成了本次实验。**

**在此，要特别感谢互联网的开放共享，老师的答疑解惑及倾囊相授，我们受益匪浅，感激不尽。**