**《算法设计与分析》实验报告模板**

我保证没有抄袭别人作业!

1. **实验名称**

回溯法

1. **问题描述**

1.01背包问题

1. **算法设计与分析**
2. **算法选择**

回溯法解决找出问题的解的集合或者要求回答什么解是满足某些约束条件的最佳解，通过构造合适的解空间（树），深度优先搜索不断利用约束函数和限界函数排除不是解和不是最优解的情况

1. **算法设计**

**算法名称**：01背包问题

**输 入：**物品价值数组p，重量数组w，背包容量

**输 出：**获取价值最大的组合

**算法步骤：**

(1) 构建解集合

(2) 每对一个物品进行选择时，先进行约束函数（是否超过背包容量）和限界函数（是否已经不可能是最优解）排除该种情况。

(3) 然后在上一步解的情况下，继续递归向下选择

**算法名称**：装载问题

**输 入：**船的容量c1，c2，货物的重量w

**输 出：**得到最优装载方案， solution(标记了该船装载哪些货物)

**算法步骤：**

1. 先将第一艘船的容量作为01背包问题的背包容量传入，借助01背包问题解决，得到最大装载方案
2. 最后判断剩余货物能否装入最后一艘船，如果可以，则为最优方案，否则
3. **算法复杂度分析**

1.01背包问题

从递归树的角度分析，递归树的深度为n（物品个数），而每一个节点有两个选择，所以整个解空间有2^n个节点，而节点的处理过程是常数时间，虽然剪枝函数可以排除一些情况，但并不固定，最差情况需要2^n，时间复杂度为O(2^n)

而空间复杂度考虑额外的函数递归栈调用时O(n)

2.装载问题

由于装载问题借用了上述的01背包问题的算法，所以复杂度分析和01背包问题相同，第二艘船仅需要进行判断，是线性时间，所以总的时间复杂度为O(2^n),空间复杂度为O(n)

四.实验测试

**测试用例与结果分析**

测试数据和用例同之前的实验一样，这里说明同之前实验。

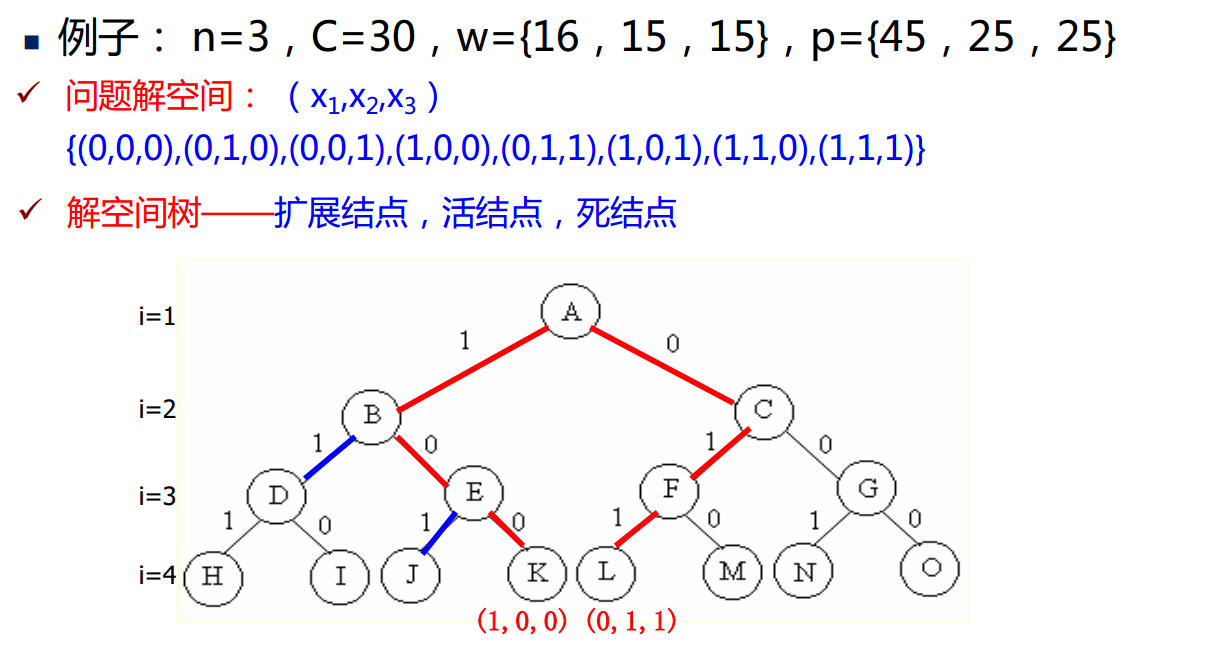
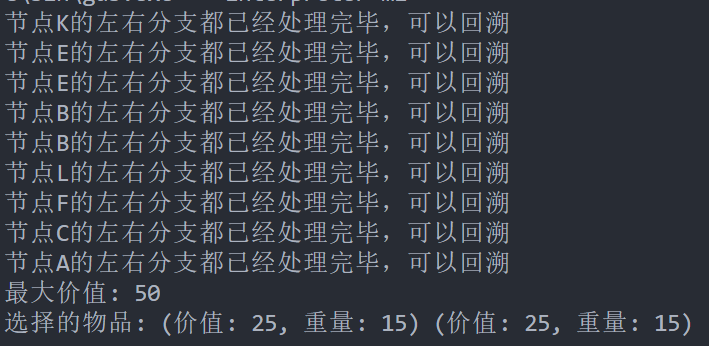
Testdata.hpp文件中的Testdata类函数生成，可以生成一维数组，乱序或有序，二维数组，以及随机字符串，可以根据数据所需求的规模设定生成测试用例。而测试时间由Timetracker.hpp中的measuretime函数实现，可以根据具体算法可以调整运行次数，以得到函数较为准确的运行时间，并且避免运行时间过久。

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（与cpp文件同名），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。

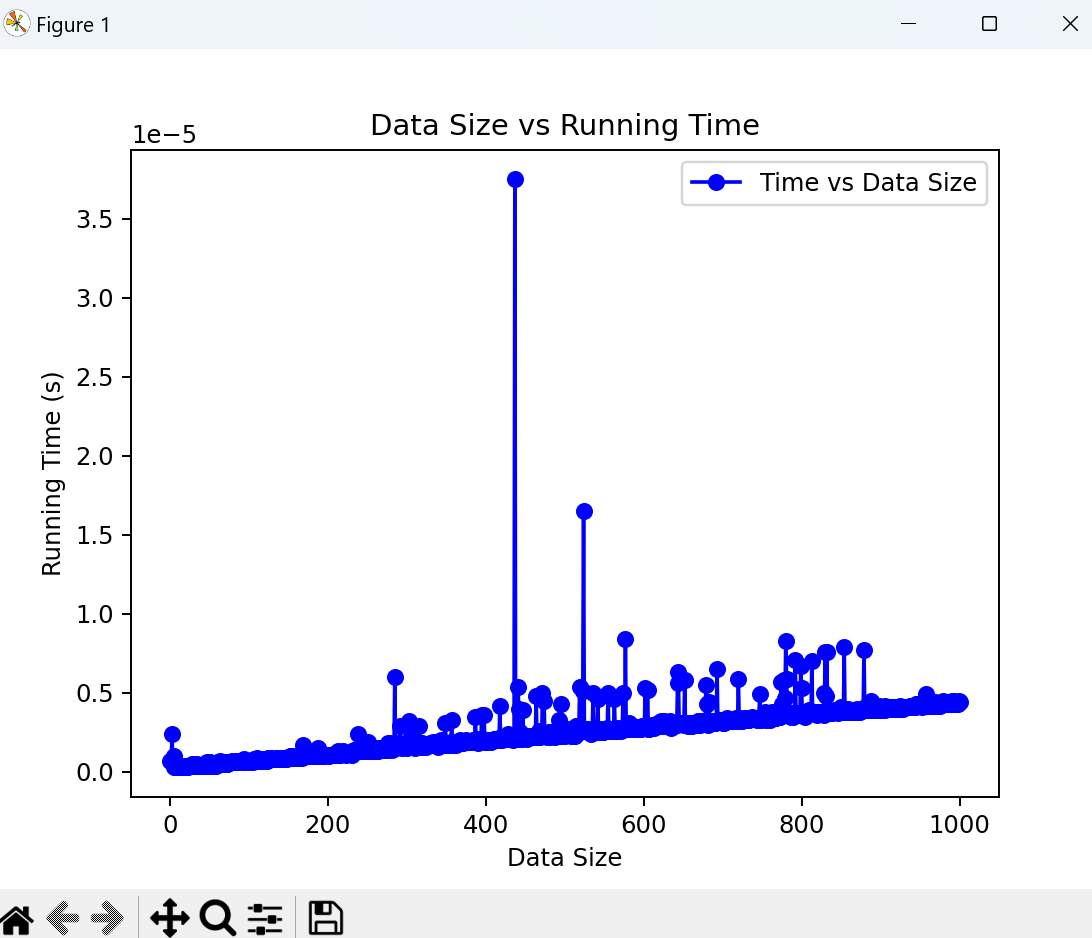
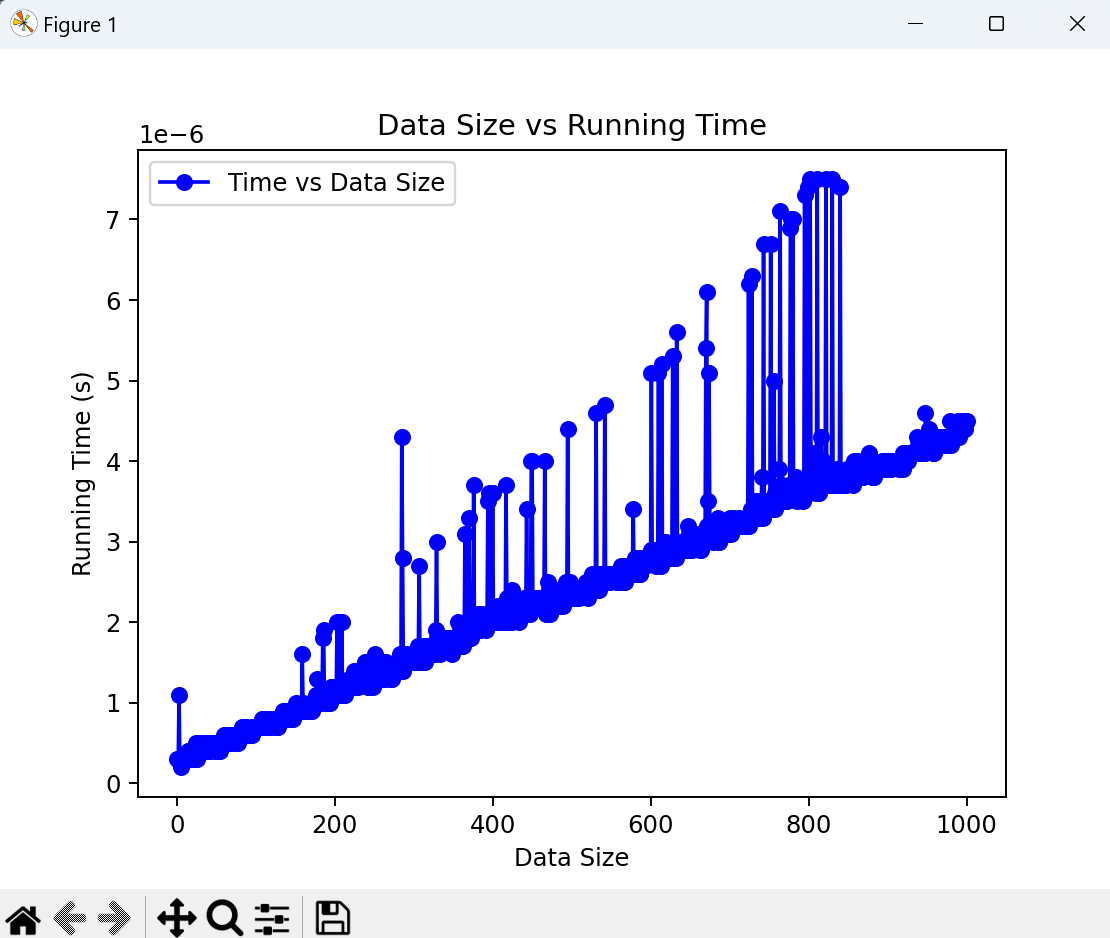
操作步骤，先运行算法.cpp文件，生成两个txt数据文件，再运行python文件根据数据文件进行绘制，即可复现以下测试用例的结果（由于测试数据每次随机生成，可能略微不同，为确保数据来源真实以及结果准确，会保留展示的测试用例数据，放在工作目录下的testdata.txt，同时便于进行算法的正确性验证）

1.01背包问题

首先使用课上例题中的数据验证代码实现的正确性



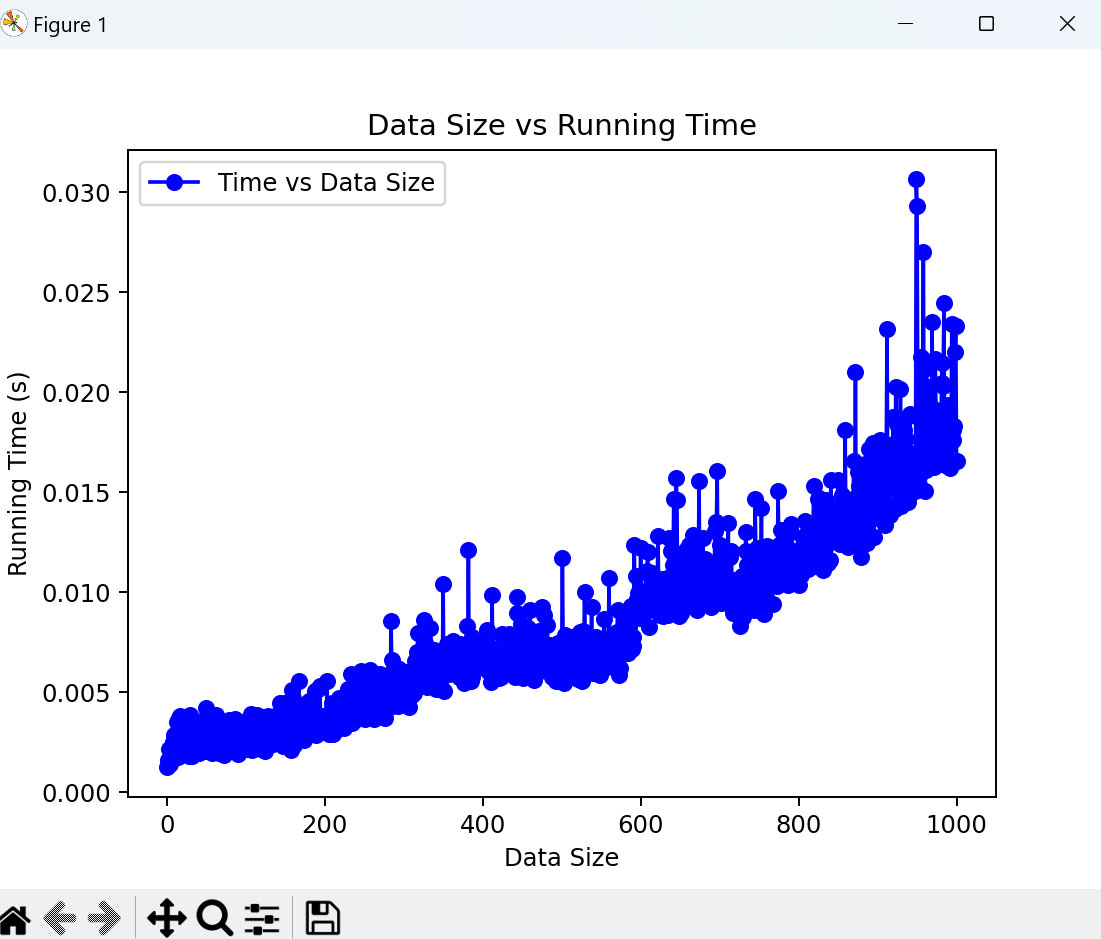
两种上界计算方法都得到验证，正确性无误。下面扩大数据规模进行性能测试，

看到结果几乎是成线性增长，并不符合时间复杂度为O(2^n)，但两个剪枝函数导致的运行时间可以看到，第二种剪枝函数更好，时间基本都比第一种更快。可以看到剪枝函数的选择会对回溯法的效率产生很大影响。

2.装载问题

测试数据对c2不变，c1容量随机生成，测试1000组数据得到的图像如下



看到在800之前的变化仍然是类似线性的，但是在800之后有非线性的趋势，符合前面的时间复杂度分析。并且借此分析可能上述测试数据不够大，再扩展数据需要的数据记录量过大，未防止最后的压缩文件过大上传麻烦，不再测试。

五.实验总结

刚接触回溯法，理论上的最差情况让人的印象挺差，总觉得时间性能不会太好，但经过实验实现，发现虽然这种方法本质上还是穷举，但是通过设计良好的剪枝函数可以极大提升性能，甚至在数据规模没有太大时，接近线性时间，这是做实验前万万想不到的结果。本次测试用例实际上会生成的偶然性数据，且运行时间较长（经调试一部分猜想这一特殊测试用例几乎遍历了所有情况），这会导致运行时间异常长，导致最终图形不美观，后续我删除了这个极特殊的数据，重新画图，可以更好的分析结果。

