动态规划、子集和数、多源最短路数据集对性能的影响

摘要

本实验针对子集和数问题，利用分支限界法，对 7 个数据集进行验证。实验结果表明，求解速度与数据集大小及数据分布有关。

对 0/1 背包问题，利用动态规划算法，对 Florida State University 的 7 个数据集进行性能分析。实验结果表明对于不同的数据集，如果数据个数越少效率越高，背包容量越小所需计算量越小。对于同一数据集内单个物品重量越大计算越简便。

对所有点对最短路径问题，利用 Floyd 算法（动态规划），对利用 Ggen.cpp 随机生成的 5 个数据集进行性能分析。实验结果表明对于不同的数据集，点数对性能有较大影响而边数变化对性能无影响。

目录

第一章 实验目的

1.1 子集和数问题

1. 探究数据集对子集和数问题求解性能的影响
2. 加深对子集和数问题的理解
3. 通过对数据集的分析探究子集和数问题的使用范围

1.2 0/1 背包问题

1. 掌握动态规划算法求解问题的一般特征和步骤。
2. 使用动态规划法编程，求解 0/1 背包问题。
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. 分析数据集分布特点对算法性能的影响

1.3 多源最短路问题

1. 掌握动态规划算法求解问题的一般特征和步骤。
2. 使用动态规划法编程，解决所有点对最短路径问题。
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. 分析数据集分布特点对算法性能的影响

第二章 实验设计流程

2.1 子集和数问题

首先利用分支限界法实现了代码。之后输入给出的数据集并且记录所需要的时间，从中找出一些规律。

2.2 0/1 背包问题

根据动态规划解题步骤（问题抽象化、建立模型、寻找约束条件、判断是否满足最优性原理、找大问题与小问题的递推关系式、填表、寻找解组成）找出 01 背包问题的最优解以及解组成，然后编写代码实现。

2.3 多源最短路问题

利用弗洛伊德算法（弗洛伊德的核心思想是：对于网中的任意两个顶点（例如顶点 A 到顶点 B）来说，之间的最短路径不外乎有 2 种情况：
直接从顶点 A 到顶点 B 的弧的权值为顶点 A 到顶点 B 的最短路径；

从顶点 A 开始，经过若干个顶点，最终达到顶点 B，期间经过的弧的权值和为顶点 A 到顶点 B 的最短路径。
所以，弗洛伊德算法的核心为：对于从顶点 A 到顶点 B 的最短路径，拿出网中所有的顶点进行如下判断：
Dis（A，K）+ Dis（K，B）< Dis（A，B）
其中，K 表示网中所有的顶点；Dis（A，B） 表示顶点 A 到顶点 B 的距离。
也就是说，拿出所有的顶点 K，判断经过顶点 K 是否存在一条可行路径比直达的路径的权值小，如果式子成立，说明确实存在一条权值更小的路径，此时只需要更新记录的权值和即可。）解决所有点对最短路径问题，然后编写代码实现。
用一个二维数组 f[i][j]表示从 i 到 j 最小路长度，初始化时输入 i，j 点的路的长度并且 f[i][i]=0.
通过观察可以发现，如果我们把一些点作为中转点的话，有可能会让路程变小。例如，我们只用 1 作为中转，可以得到
f[i][j]=min(f[i][j],f[i][1]+f[1][j]);
如果我们拿 1 和 2 作为中转点，可以得到
f[i][j]=min(f[i][j],f[i][1]+f[1][j]);
f[i][j]=min(f[i][j],f[i][2]+f[2][j]);
这段代码的意思是先拿 1 作为中转，找到 1 做中转的最小路径之后，再拿 2 做为中转

第三章 代码实现

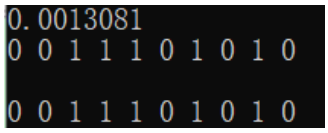
由于代码数量过多并且没有高亮，为了便于查看，将代码同步转移到网站 https://xinhecuican.github.io/post/fb98053e.html 上。

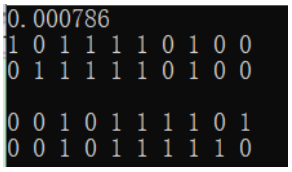
第四章 实验结果与复杂性分析

4.1 子集和数问题

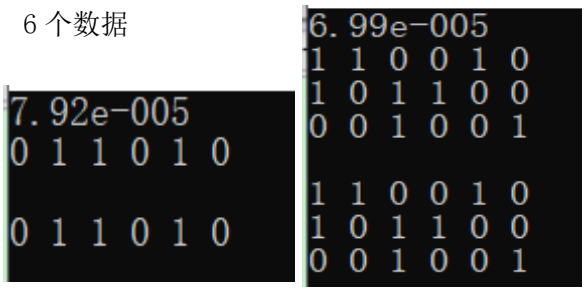
4.1.1 实验结果及分析

10 个数据

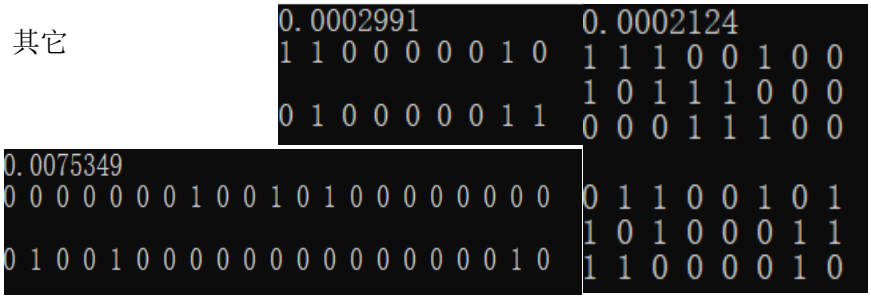




6 个数据



其它



从 10 个输入可以看出第二组与其他两组差别巨大，通过观察可以发现他和其他两组最大的区别是前面的 0 多了一位，根据子集和数问题的解空间树可以推断出第一个被选择的数据越靠前，所需时间会显著减小。

10 个输入中的第一组和第三组也有一些区别。他们都是在第 2 个位置出现 1，但是第一组比第三组所需时间少。观察数据集差异可以发现第一组的 1 的数量比第三组少，由此可以推断组成答案的数的数量越少，所需时间可能越短。可能的解释是 1 的数量少说明剪枝会比较多，因此所需时间可能会更短。而 6 个输入的两个组也验证了这一点。

之后是输入数量不同的组之间的比较。9 个输入的比 8 个输入的 1 的位置还要靠前但是所需时间更多。而 21 个输入的所需时间是 8 个输入的近 35 倍，而六个输入是八个输入的 0.37，说明数据集的大小对时间也有很大的影响。

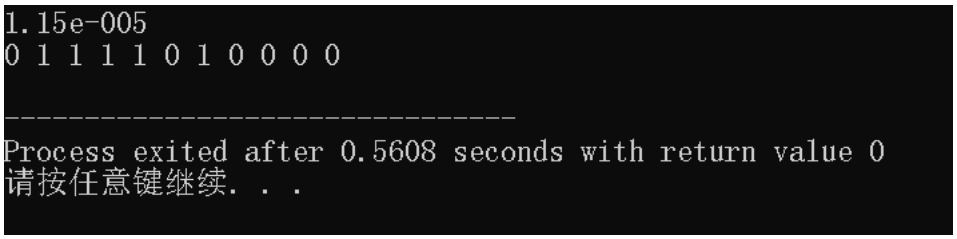
4.1.2 复杂性分析

对于最坏情况，搜索完整个问题空间需要$O(2^n)$时间。但是绝大多数情况下剪枝函数都会排除一些情况，因此实际使用时复杂度并没有到2^n甚至常数时间也可以解决。

4.2 0/1 背包问题

4.2.1 实验结果及分析

P1 数据集：数量为 10 个



P2 数据集：数量为 6 个

```
4.8e-006
0 0 1 1 1 0

--------------------------------
Process exited after 0.5319 seconds with return value 0
请按任意键继续. . .
```

P3 数据集： 数量为 7 个

```
8.1e-006
0 1 1 0 0 1 0

--------------------------------
Process exited after 0.5297 seconds with return value 0
请按任意键继续. . .
```

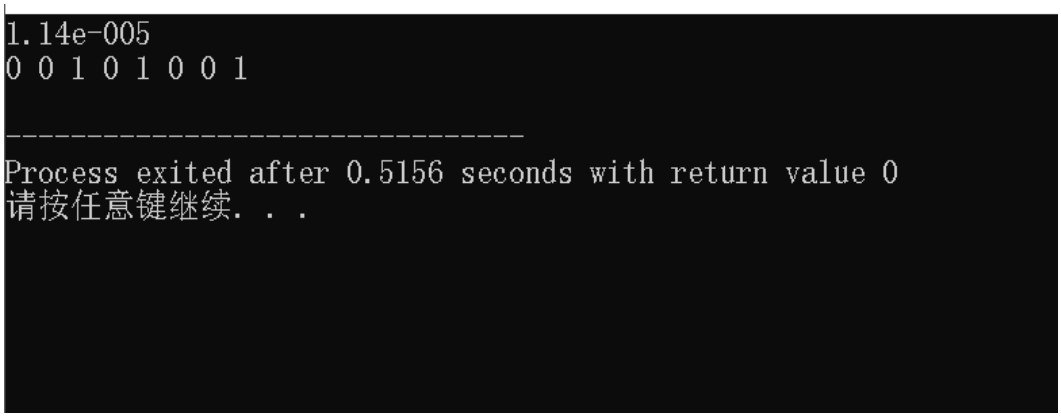
P4 数据集： 数量为 8 个

```
6.3e-006
0 1 0 0 1 0 0 0

--------------------------------
Process exited after 1.147 seconds with return value 0
请按任意键继续. . .
```

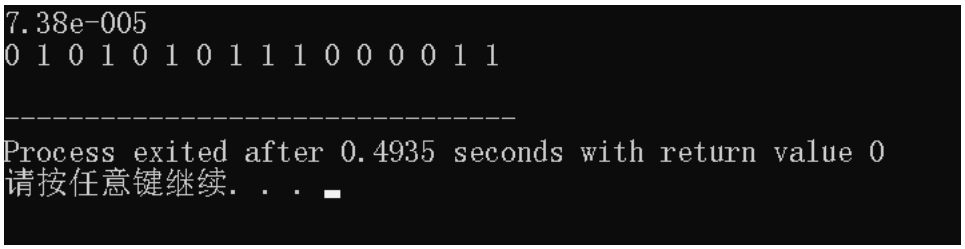
P5 数据集： 数量为 9 个

```
1.02e-005
0 1 0 1 1 1 0 1 1

--------------------------------
Process exited after 0.4738 seconds with return value 0
请按任意键继续. . .
```

P6 数据集： 数量为 8 个



P7 数据集： 数量为 16 个



通过对 P4 和 P6 这两个数据个数相同的数据集所消耗时间对比，背包容量越大所需占用时间空间资源越多；对比其余几个数据个数相差较大的数据集 ，基本呈现数据数越多所需时间越多的特点，而根据代码中信息 findMax（）函数中 j < w[i]得出在其余相同的数据集中单个物品重量越大所需计算越少。即:对于不同的数据集,如果数据个数越少效率越高，背包容量越小所需计算量越小。对于同一数据集内单个物品重量越大计算越简便。

4.2.2 复杂度分析

主要是一个二重循环，外层循环次数是 n(数量)，内层循环次数是 bagv（背包容量）,总的复杂度是 O(n*bagV)。

4.3 多源最短路问题

4.3.1 实验结果分析

数据集选择：

利用 Ggen.cpp 随机生成

P0: ./Ggen 6 2 3.1e-006

P1: ./Ggen 6 3 2.7e-006

P2: ./Ggen 7 2 3.7e-006

P3: ./Ggen 9 6 7.5e-006

P4: ./Ggen 7 7 3.4e-006

P5: 增加 P1 的边数 2.8e-006

P0 和 p1 的点数目都是 6， p2 和 p4 数目都是 7，p3 的点的数目是 9.可以发现随着点的数目增加，所花费的时间在显著增加，并且 P5 和 p0、p1 之间并没有太大区别，说明边数增加

对结果无影响。

4.3.2 复杂度分析

主要是一个三重循环，每层循环都进行 n 次，所以总的大小是$O(n^3)$

第五章 总结

子集和数问题性能和第一次出现 1 的位置、被选中的数量和数据量的大小有关。

0/1 背包问题和数据量、单个物品重量（如果很多很重的会被直接排除）、背包容量等有关。

多源最短路问题发现所用时间仅与数据集的点数目有关，与边的多少无关。并且 p2 是 p1 的 1.37 倍，p3 是 p1 的 2.7 倍，接近 n^3.