****

**《算法设计与分析》实验报告**

**题目：多种算法求解子集和数问题**

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 学号 | 任务分工 | 成绩 |
| 江熠 | 3020207160 | 动态规划算法实现、论文写作 |  |
| 罗奥成 | 3019207021 | 回溯算法、分支限界算法实现 |  |

**摘 要**

本实验的主要目的是利用多种算法求解子集和数问题。子集和数问题给定了一个正整数和由个正整数组成的集合，要求我们找到所有满足条件的集合，使得，且中元素之和（即和数）为。若找不到满足条件的集合，则找到使得和数尽量大且不超过的的子集。针对子集和数问题的特性，我们分别使用了动态规划算法、回溯算法以及分支限界算法，利用C++语言编写代码对问题进行求解，并使用Florida State University提供的7个数据集中的6个以及1个自编数据集（为了验证找不到和数刚好等于的子集的情况）对算法正确性进行验证。除此以外，我们还分析了不同算法在解决这一问题上时间复杂度的差异，并归纳了三种算法各自的优缺点。

**关键词**:

子集和数，动态规划，回溯，分支限界

# 1. 实验目的

**1.1 问题的背景及意义**

想象一下如下的场景：假如你是一名超市收银员，现在你需要给一位顾客找出一定数额的钱（例如87元）。收银机里有若干张面额不同的纸币（例如2张50元，3张20元，2张10元，5张5元和2张1元），你是否能刚好将钱找给顾客？如果能，有几种可行的找零方案？如果不能，你最多能找给顾客多少钱？这个找零问题就可以被抽象为子集和数问题，并利用算法知识和计算机程序解决。

首先我们先给出子集和数问题的定义。在这个问题中，给定一个正整数和由个正整数组成的集合，要求我们找到所有满足条件的集合，使得，且中元素之和（即和数）为。若找不到满足条件的集合，则找到使得和数尽量大且不超过的的子集。

**1.2 课题的任务与目的**

本次课题的任务就是利用在本课程中学习到的算法知识解决上述的子集和数问题。通过本次实验，我们可以从实践的角度掌握利用算法求解复杂问题的流程。在分析问题的过程中，我们能够强化对课堂所学几大基础算法的原理的理解。在编写代码时，我们可以提升利用算法原理进行编码实践的能力。最后，我们还能从本次课题中拓展算法改进的思路，真正掌握《算法设计与分析》这门课的核心思想。

**1.3 解决方案的主要思路**

子集和数问题是一个NP-难问题，使用暴力求解的方法需要对其每一个子集进行和数的计算，时间复杂度高达。为了避免较高的时间复杂度，我们可以利用算法对其进行优化。常见的解决NP-难问题的算法有动态规划算法、回溯算法和分支限界算法。这三种算法都可以解决子集和数问题。下面对这三种算法解决子集和数问题的主要思路进行阐述。

**1.3.1 动态规划算法**

动态规划算法可以解决的另一个经典的NP-难问题就是0-1背包问题。我们发现，子集和数问题可以转化为一个特殊的0-1背包问题。在0-1背包问题中，向背包中装入某物品，背包容量减少了，而价值增加了。而子集和数问题中，将某正整数“装入”子集，子集剩余能装入的数的和，即“容量”减少了，而目前子集的和数，即“价值”增加了。因此，子集和数问题可以看作是每个物品重量和价值相等的特殊0-1背包问题。我们就可以仿照0-1背包问题，写出相应的状态转移方程，并编写代码进行求解。

**1.3.2 回溯算法**

子集和数问题可以被展开为一个解空间树。对于第层的节点来说，其左孩子表示子集中存在第个数，右孩子表示不存在。容易得到这个解空间树为一棵层的完全正则二叉树，从根节点到叶节点的每一条路径都是解空间的一个元素。回溯算法使用深度优先搜索（DFS）的方法对该树进行搜索。为了剪枝，我们需要对每个节点进行限界。节点的下界即为当前子集的和数，上界当前子集的和数与中所剩元素的和数之和。如果某个节点的下界刚好等于，则找到一个可行解；如果节点的下界大于或是上界小于，则这个节点是不可行的，可以将其“杀死”；否则，则可以继续扩展，直到找出可行解。

**1.3.3 分支限界算法**

分支限界算法解决该问题的思想与回溯算法很类似，区别在于分支限界算法使用的是广度优先搜索（BFS）的方法对解空间树进行搜索。用于剪枝的限界函数与上述的回溯算法中的限界函数是一样的。需要注意的是，由于我们需要找到每一个可行解，而不是找出最优解，故不能使用LC分支限界算法，只能使用FIFO分支限界算法。

# 2. 实验设计流程

下面将解释以上的三种算法具体如何求解子集和数问题。

**2.1 准备工作**

为了实现接下来的算法，首先我们需要把文件中的数据集读入程序中。为此编写testcase\_read()函数，该函数接受一个代表当前读取数据集编号的字符串作为参数。对于每个数据集，需要读入两个.data文件，分别代表正整数和正整数集。例如，第一个数据集包含两个文件subsetsum-1-1.data和subsetsum-1-2.data。前者保存的是正整数的值，读入整型变量c中；第2个文件保存的是正整数集的值，读入整型数组w[1:n]中。除此以外，还定义了布尔型数组x[1:n]表示符合条件的子集，x[i]为1表示的第i个元素在子集当中，为0则表示不在子集当中。初始化x[1:n]中每个元素为0。

**2.2 动态规划算法**

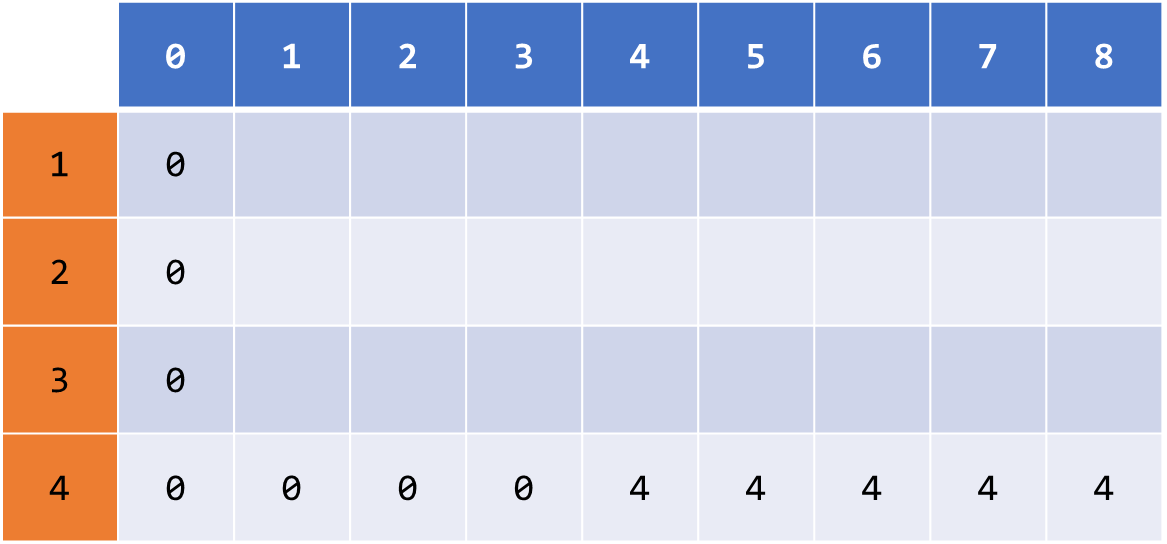
设表示条件为的子集和数问题的最大子集和数的值，则初始条件为

状态转移方程为

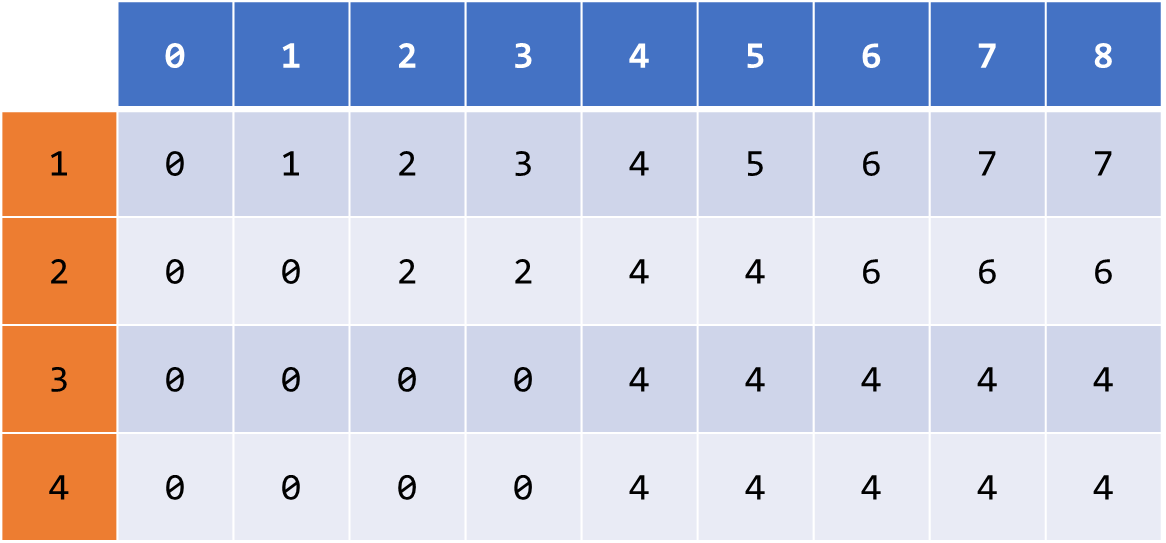
求出的值，即可得到符合条件的最大子集和数的值。

我们现在已经能得到最大子集和数的值了，接下来该如何找到满足最大子集和数的呢？我们只需要观察每次状态转移方程中的值从何而来即可。例如，对于，若，则表明第1个数一定不在子集当中，继续考虑。反之，如果，则表明第1个数不在子集当中，继续考虑；如果，则表明第1个数在子集当中，继续考虑；如果，则表明第1个数在或者不在子集当中都能够得到最优解，和这两条路径都需要考虑。这样一步步回溯得出最优解的过程，直到，若，则表明第n个数不在子集当中；反之则表示第n个数在子集当中。至此就得到了一个或数个满足条件的解向量x[1:n]。

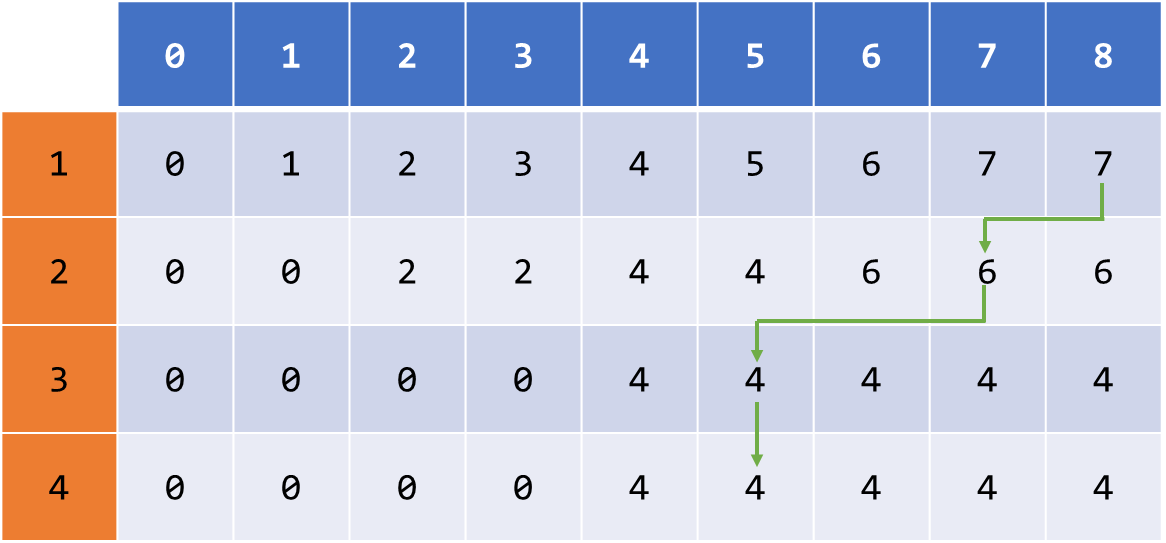
为了进行上述的回溯过程，我们需要建立一个二维数组dp\_table[][]来存储所有的值以便于回溯。初始化dp\_table[1:n][0]均为0，dp\_table[n][0:c]根据初始条件赋值。例如，对于第3个数据集，初始化dp\_table如下：



初始化之后，根据状态转移方程填充dp\_table。注意填充顺序是从左到右，填充完下一行之后再填充上一行，在上述例子中得到的dp\_table如下：

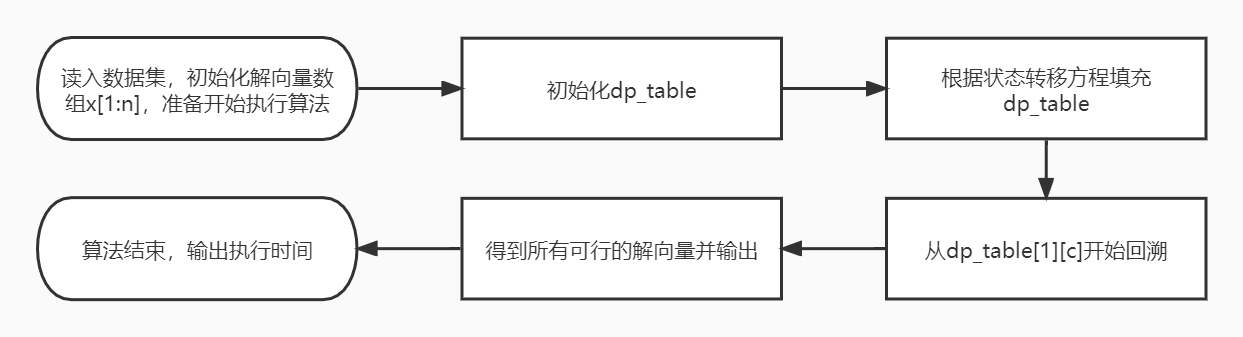


至此我们已经得到了最大子集和数。接下来，从dp\_table的右上角元素dp\_table[1][c]开始，按照之前所叙述的方法，对得到最优解的过程回溯：



即得到解向量x[1:n]。上述例子中得到的解向量为。

整个动态规划算法的流程图表示为：



**2.3 回溯算法**

**2.4 分支限界算法**

# 3. 实验结果及复杂性分析

**3.1 实验环境设置**

本实验在Ubuntu虚拟机下进行，Ubuntu版本号为20.04。使用的编译器为g++，版本号为9.3.0。为了更加方便地编写代码，我们还使用了Visual Studio Code的RemoteSSH功能连接虚拟机。

为了便于编译和测试程序，我们使用了Makefile文件进行自动化编译和链接。在Linux系统中工程的根目录下执行make命令，将自动编译src文件夹下的dp-subsetsum.cpp、bt-subsetsum.cpp和bb-subsetsum.cpp三个代码文件并逐个运行，输出运行的结果。

**3.2 测试方法及性能评价指标**

**3.2.1 测试方法**

为了验证算法的正确性，我们使用Florida State University提供的7个数据集中的6个（P01、P02和P04~P07）以及1个自编数据集（为了验证找不到和数刚好等于的子集的情况）对算法正确性进行验证。所有数据集的存储于data文件夹下文件名以“-1”结尾的.data文件中，存储于文件名以“-2”结尾的.data文件中，并以0作为数据结尾的标志。

**3.2.2 性能评价指标**

为了测试三种算法的实际耗时，我们还需要计算每种算法测试每一个样例时的运行时间。由于运行时间太短，不能用自然时间来表示运行时间，我们利用了ctime库中的clock()函数来计算运行时间。clock()函数的返回值为从开始此程序到调用clock()函数之间的CPU时钟计时单元（clock tick）数。对于每一个样例，在数据集读入完毕时将当前clock()函数的返回值存储至time1中，直至算法执行完毕时，计算当前clock()的返回值与time1的差值，此差值即可表示算法对于该样例的实际运行时间。

**3.3 实验结果分析**

# 4. 结论与展望

**4.1 解决方案的整体评价**

**4.1.1 解决方案的优势**

**4.1.2 解决方案的不足**

**4.2 进一步工作的展望**

# 5. 参考文献

[1] Sartaj Sahni. Data Structures, Algorithms, and Applications in C++. Gainesville, Florida. 2004.

[2] Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein. Introduction to Algorithms. Massachusetts Institute of Technology. 2009.

[3] 严蔚敏,吴伟民.数据结构[M].北京:清华大学出版社,1997.