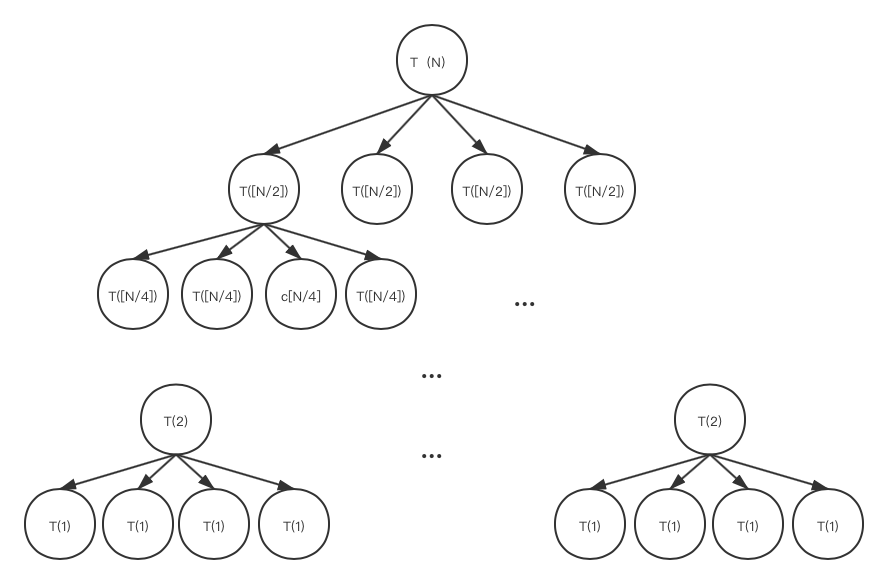
**Assignment Two**

|  |  |
| --- | --- |
| Name | 周子龙 |
| Student number | 1851201 |

1. **递归算法**

对递归式（c为常数），画出递归树，并给出其解的一个渐近紧确界。用代入法进行验证。



该递归数如上图所示。

我们考虑n = 2k这样的n。

对式（1）进行后项代替，得到：

由等比数列求和公式，我们有：

将式（3）代入式（2），并假设 得到：

由于,故而：

根据master theory，可以认为对所有的n均有式（5）成立。

下面用代入法对式（5）进行验证，即证明：

由式（1），可以得到：

我们取：,则式（7）恒成立。

证毕。

1. **整数线性规划问题**

考虑下面的整数线性规划问题

试设计一个解此问题的动态规划算法，并分析该算法的计算复杂性。

解：

设为前项中满足约束条件的最大的，即：

为了方便考虑，我们令

现在考虑与前一项的关系：

可能包含。如果包含，则说明明，此时有：

且

如果不包括，说明，此时有：

且

综上，我们可以得到的递推关系式为：

且满足初始条件：.

对于本问题我们只要输出第一个使得时的F（i）,即为结果。故我们可以由此设计一个解决该问题的动态规划算法，其伪码描述如下：

**Algorithm** integerProgramming(A[], C[], X[], b)

//Implement of solving integer programming problem using dynamic programming.

//**Input**: array A[], C[], X[], b, in which X contains

non-negative integer.

//**Output**: under the constraint that .

F[0] 🡨0，i 🡨1，w🡨0

**while** i < n

w +=A[i]\*X[i]

**if**

F[i] = F[i-1] +

w+=

i++

**else**

**return** F[i-1]

本算法的时间复杂度为，空间复杂度为，这是显而易见的。

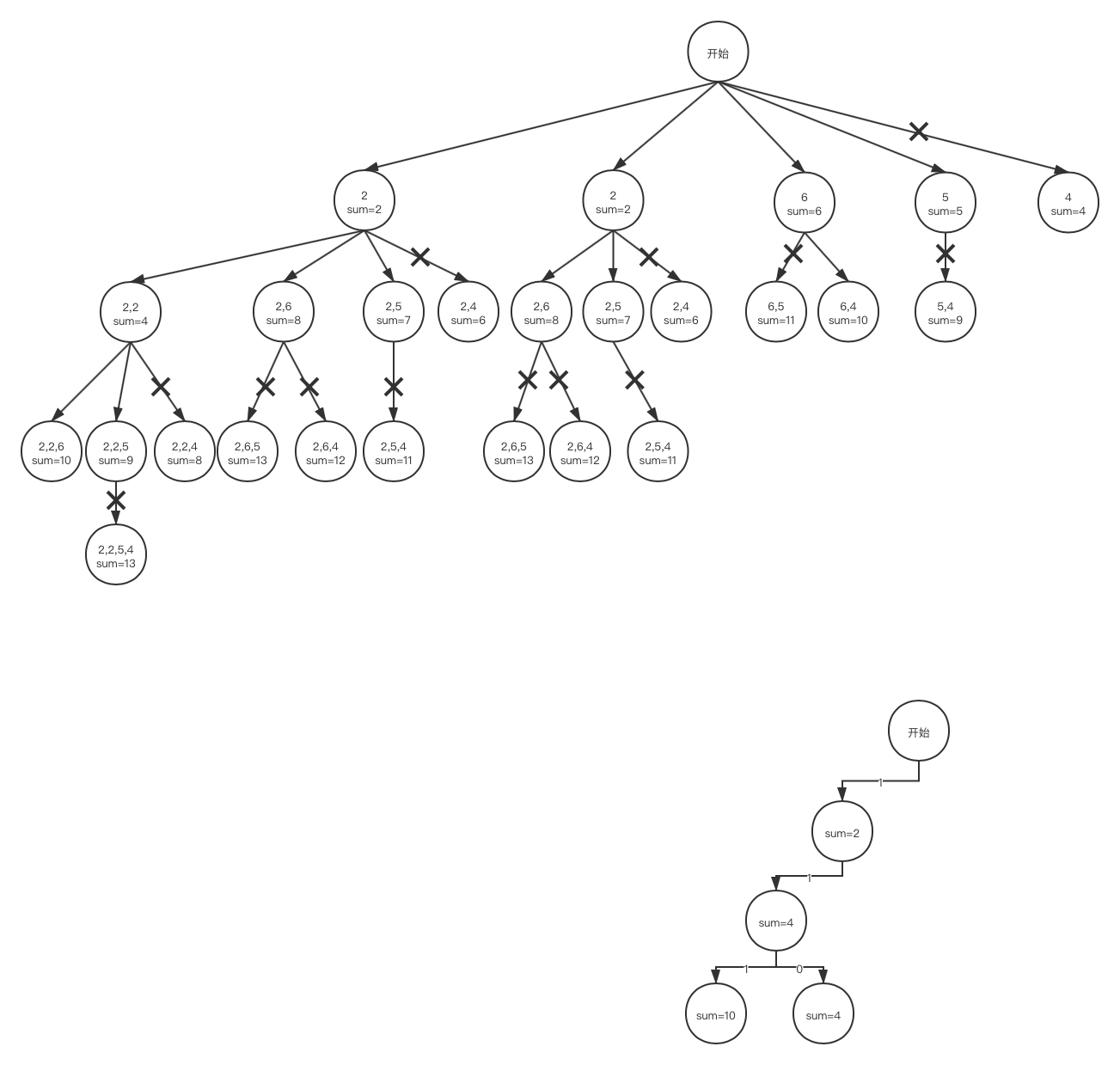
1. **回溯法**

集合是一个正整数集合（这里S用重复元素，应该理解为重复集），c=10，子集和问题判定是否存在S的一个子集，使中的元素之和为c。求所有可能的子集。要求：

1. 给出该问题的形式化描述。
2. 利用回溯法深度搜索解空间树，利用剪枝函数求其所有解，要求画出其实际生成的解空间树（即要标出被剪枝的部分）。
3. 形式化描述：

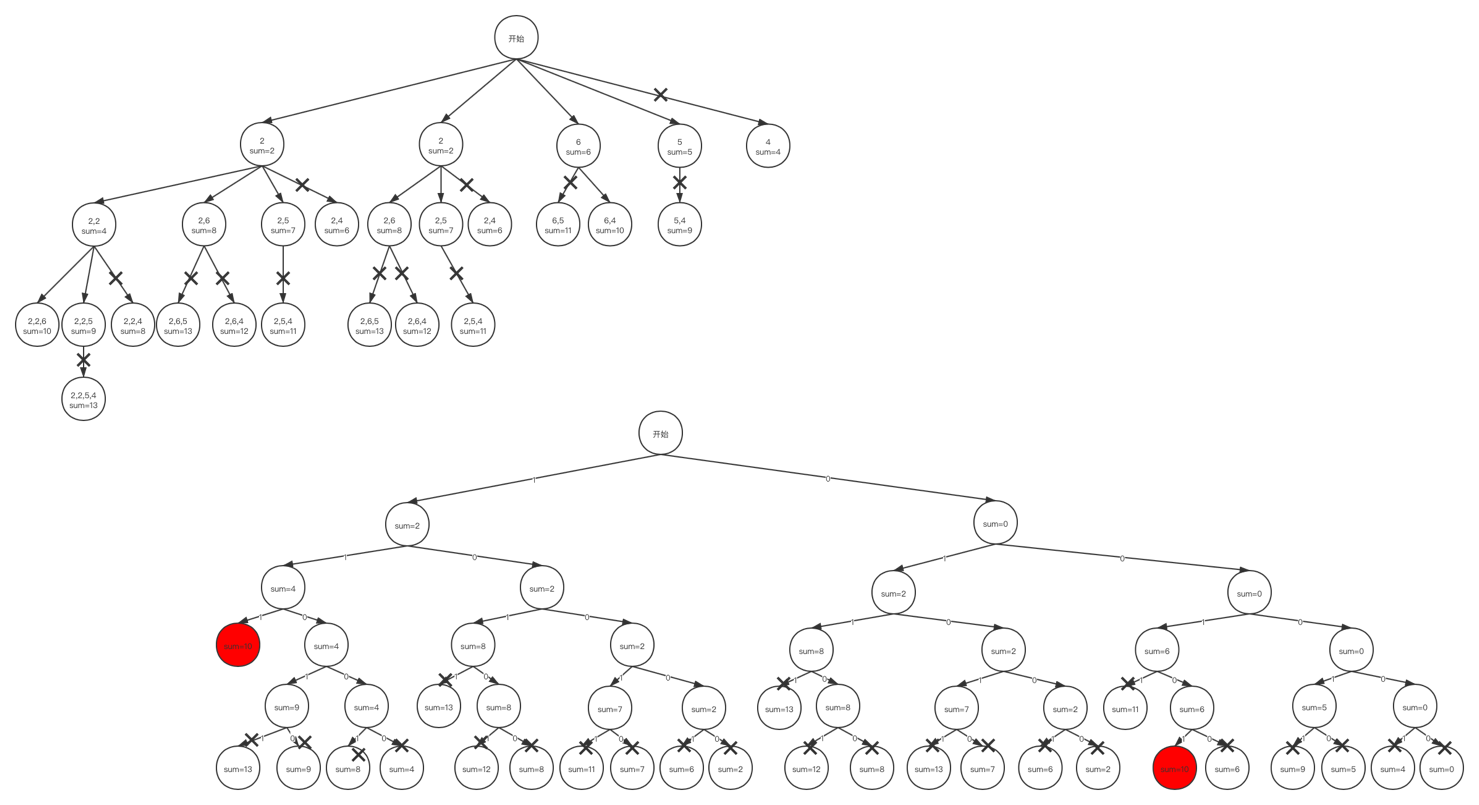
对于一个给定重复集S={，求该集合中元素的这样的全部组合（重复元素视不相同元素）：C={, 使得。

1. 对该问题的回溯解空间树如下：



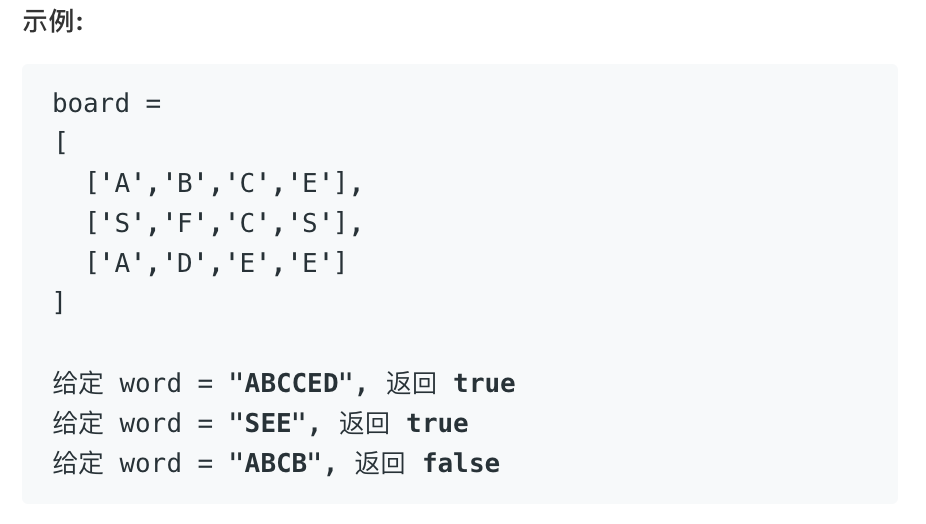
此外，可以使用01编码的形式来表示第i个元素是否包含在集合中进行编码，则全部解空间可以表示为一个n（集合中元素个数）维的向量的所有可能排列的集合:{00000, 00001, 00010,…, 11111}

这样，可以用一个二叉树来表示其解空间，得到：



1. **编程题**

给定一个二维网格和一个单词，找出该单词是否存在于网格中。单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

****

提示：

board和word中只包含大写和小写英文字母。

* 1 <= board.length <= 200
* 1 <= board[i].length <= 200
* 1 <= word.length <= 10^3

算法的自然语言描述：

输入：一个二位字符网格和一个待查找的单词。

输出：判断该单词是否可以通过水平相邻或者垂直相邻的方式由网格中的字符构建出来。

**Step1**: 依次遍历棋盘上的所有位置，如果当前位置和单词中的第0位字符相等，若相等则

进入**step2** 并令i = 0, 否则遍历下一个位置.

**Step2**: 将当前位置标记为已检查，并依次判断该位置的四个相邻位置是否和单词中的第i个字符相等，若相等，则进入**Step3**，否则返回**Step1**.

**Step3**: 如果i等于单词的长度，返回查找成功，若否，返回**step2**，并令i=i+1

**Step3**: 重复**Step1**-**Step2**，直到棋盘上所有位置被遍历完毕。若均以遍历完毕，返回查找失败.

空间复杂度：

算法中使用了一个大小的二维数组存放检查信息，此外的存储内容均为常数级，本算法的空间复杂度为.

时间复杂度：

本算法和给定的棋盘信息和待查找的单词相关的排布有关，复杂度与棋盘的行数、列数以及单词的长度k有关。综上，应当分别讨论最好情况、最坏情况和平均情况：

在最好情况下，棋盘的前k个相邻的位置（k位单词的长度）碰巧可以构成单词，故在最好情况下，有：

在最坏情况下，单词的前k-1个位置恰好均有匹配，但第k个位置不匹配，这就相当于生成了全部的解空间。为了简单期间，认为棋盘上的任意一个位置均有四个相邻位置，即忽略边界上的特殊情况。其次，由于单词具有顺序，应当认为不同的搜索顺序是不一样的解。同时，解的数量和棋盘的形状也有关系，处于简单考虑，我们认为棋盘较为方正，即m和n相差不太大。故而，在的棋盘上，共有：

个连续的k个位置，可以构成一个长度为k的单词。在实际情况中，考虑到边界以及每一个位置上的字母只能在单词中出现一次，所有这样的位置个数应当小于上式。

在这样的考虑下，有：

平均情况下，为了简单期间，延用上面的假设。在棋盘的全部个位置上，每一个位置有48种可能的字母出现（区分大小写），对单词亦然，故而单词的第i个位置的棋盘上的某一位置匹配的概率为：，故而可以得到：

如果认为，m和n近似相等，则问题规模可以简化为n。由此，可以近似地得到：

我们看到，在平均情况下时间复杂度竟然要小于最坏情况下的时间复杂度，这是由于不一样的计算方法和假设下得到的，在最换情况下，我们假设k个字母必须要相邻出现，而平均情况下仅仅是计算了概率，没有考虑这一实现细节。如果统一忽略这一细节，则可以得到如下的修正：

这与我们直观上的理解类似。

运行结果：

共设置四个测试点：

