最优二叉搜索树输出实现报告

王资 18214668 计算机技术

1.基本思想

1.1 穷举有序序列所有可能的二叉搜索树

设有序序列为 S，包含 n 个元素，简化地，设 k 为序列 S 的第 k 个元素，那么算法为对 k=1:n 依次输出以 k 为根，1:k-1 为左子树，k+1:n 为右子树，递归地重复这一过程，则可以穷举有序序列 S 的所有二叉搜索树。

1.2 输出最优二叉搜索树

先使用动态规划的算法，计算最优二叉搜索树的平均路径长度，同时记录每个子问题的最优解的分割。该问题的动态规划算法具体为初始化 w[i+1][i] = a[i]，0 <= i <= n, 即是一个空树，但假设含有第 i 个叶子节点；由于是空树，其深度为 0，所以初始化 m[i+1][i] = 0，0 <= i <= n. 随后与矩阵链相乘的动态规划算法类似，对 r=1:n, j = i + r, （r 即二叉搜索树的节点数（不包括叶子节点），i, j 分别为有序序列的第 i 和第 j 个元素），根据 w[i][j] = w[i][j-1] + a[j] + b[j], $m[i][j]=\min_{i\le k\le j}\{m[i][k-1]+m[k+1][j]\}+w[i][j]$ ，来得到和更新 w[i]j]和 m[i][j]二个值，直到循环结算，就得到了 m[1][n]即需要求解的最优二叉搜索树的平均路径长度，在更新 m[i][j]的同时记录子问题 m[i][j]的最优分割。

已知最优分割后，只需递归地读取当前问题的最优分割，再继续读取二个子问题的最优分割，递归地重复这一个过程就能得到并可以输出动态规划算法计算出来的最优二叉搜索树。

2.代码中的数据结构

2.1 穷举有序序列所有可能的二叉搜索树

代码中使用一个向量，作为储存序列有序序列 S 的容器，并作为枚举二叉搜索树函数的输入，该向量的主要作用是正确输出序列中的元素且避免重复。

枚举二叉搜索树函数的返回值是一个储存字符串的向量，作为该子问题的所有可能的二叉搜索树的临时储存容器。

2.2 输出最优二叉搜索树

double **m 对于 m[i][j]储存从序列第 i 个元素到第 j 个元素的子问题的最优解；int **s 对于 s[i][j]储存从序列第 i 个元素到第 j 个元素的子问题的最优解的分割方式；double **w，对于 w[i][j]储存从序列第

i 个元素到第 j 个元素的子问题的各个节点（包括叶子节点）的搜索概率之和。double *a, double *b 分别存储叶子节点（搜索失败）和非叶子节点（搜索成功）的搜索概率。

3.程序流程

3.1 穷举有序序列所有可能的二叉搜索树

输入一个正整数，代表有序序列包含的元素数量 n，随后程序生成 0:n-1 的有序序列 S 作为枚举二叉搜索树函数的输入，该函数根据基本思想中的算法，递归地完成各个子树的构造，最终的返回结果即为枚举长度为 n 的有序序列 S 的所有二叉搜索树的结果。

3.2 输出最优二叉搜索树

在代码中规定 a, b 各个元素的值，即各个元素搜索成功的概率或各个范围搜索失败的概率，然后输入根据基本思想中的动态规划算法实现的函数中，得到矩阵 m 和矩阵 s。根据 m[1][n]得到最短平均长度，将 s 输入到基本思想中设计的递归分割函数，其返回结果即是最优二叉搜索树。

4.调试过程出现的问题和解决方法

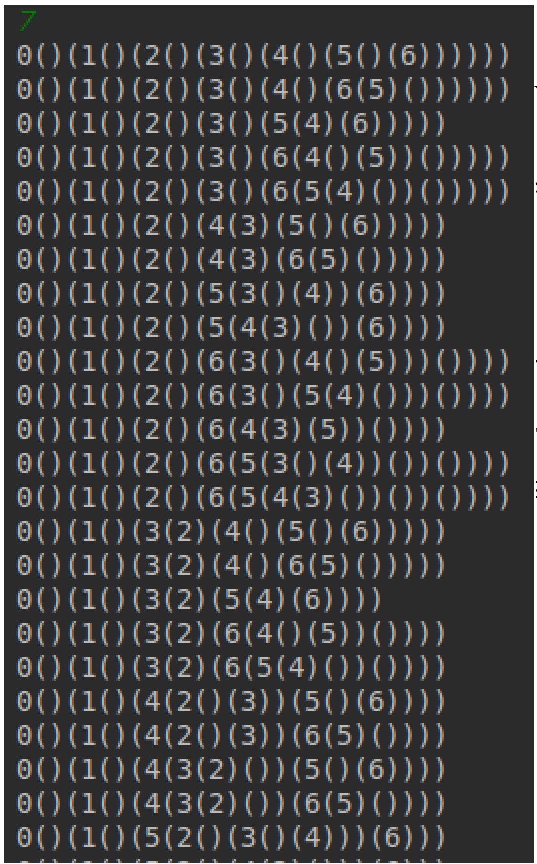
1. 原本想用递推的方式存储穷举二叉搜索树的情况，但由于每个子问题的根节点都有多种选择，导致这种想法处理多个根的情况很复杂，于是还是使用递归的方法，每个递归的返回值储存了该子问题穷举二叉搜索树的情况，于是最终也能合并各个子问题，得到原问题的穷举结果。

2.动态规划中的 w[i+1][i]和 m[i+1][i]的初始化问题，要正确理解这个节点的意思，(i+1,i)是一个实际上不存在的节点，是一棵空树，也是一个假设的第 i 个叶子节点（没有父节点），理解了这个含义之后，就知道它们的初始化，w[i+1][i]=a[i]，而由于是空树 m[i+1][i]=0。初始化这一列（斜列）节点是为了后续的动态更新。如果这 2 列元素初始化错误，就会导致后续的结果都出错。

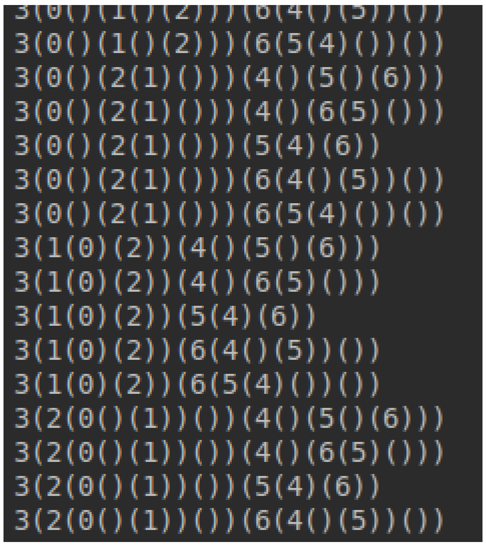
5.运行结果

5.1 穷举有序序列所有可能的二叉搜索树

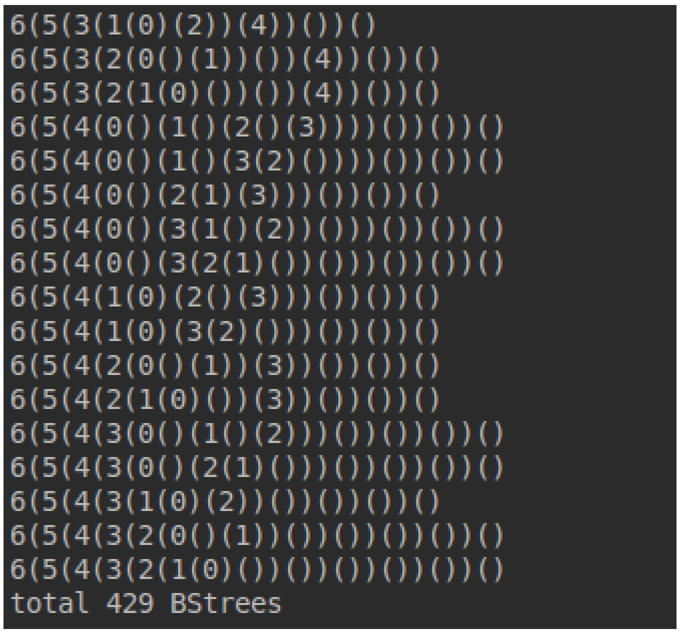
输入为 7



```
7
0()(1()(2()(3()(4()(5()(6)))))))
0()(1()(2()(3()(4()(6(5)()))))))
0()(1()(2()(3()(5(4)(6)))))
0()(1()(2()(3()(6(4()(5))())))))
0()(1()(2()(3()(6(5(4)())())))))
0()(1()(2()(4(3)(5()(6)))))
0()(1()(2()(4(3)(6(5)()))))
0()(1()(2()(5(3()(4))(6))))
0()(1()(2()(5(4(3)())(6))))
0()(1()(2()(6(3()(4()(5)))()))))
0()(1()(2()(6(3()(5(4)()))()))))
0()(1()(2()(6(4(3)(5))())))
0()(1()(2()(6(5(3()(4))())()))))
0()(1()(2()(6(5(4(3)())())()))))
0()(1()(3(2)(4()(5()(6)))))
0()(1()(3(2)(4()(6(5)()))))
0()(1()(3(2)(5(4)(6))))
0()(1()(3(2)(6(4()(5))())))
0()(1()(3(2)(6(5(4)())())))
0()(1()(4(2()(3))(5()(6))))
0()(1()(4(2()(3))(6(5)())))
0()(1()(4(3(2)())(5()(6))))
0()(1()(4(3(2)())(6(5)())))
0()(1()(5(2()(3()(4)))(6)))
```

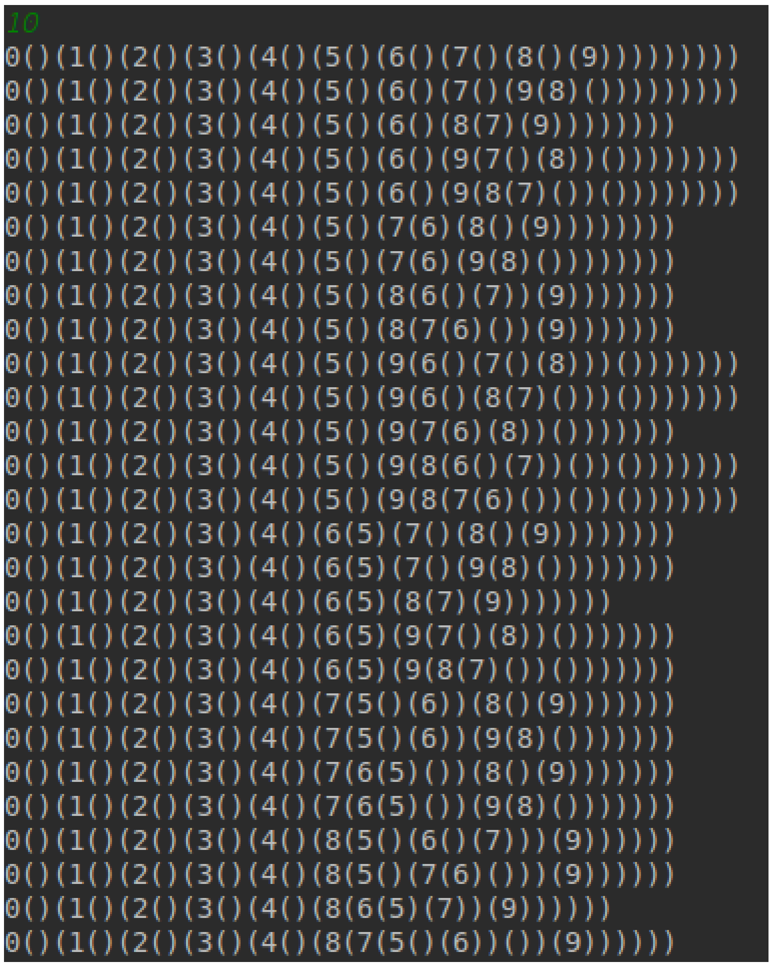


```
3(0()(1()(2)))(6(4()(5))())
3(0()(1()(2)))(6(5(4)())())
3(0()(2(1)()))(4()(5()(6)))
3(0()(2(1)()))(4()(6(5)()))
3(0()(2(1)()))(5(4)(6))
3(0()(2(1)()))(6(4()(5))())
3(0()(2(1)()))(6(5(4)())())
3(1(0)(2))(4()(5()(6)))
3(1(0)(2))(4()(6(5)()))
3(1(0)(2))(5(4)(6))
3(1(0)(2))(6(4()(5))())
3(1(0)(2))(6(5(4)())())
3(2(0()(1))())(4()(5()(6)))
3(2(0()(1))())(4()(6(5)()))
3(2(0()(1))())(5(4)(6))
3(2(0()(1))())(6(4()(5))())
```

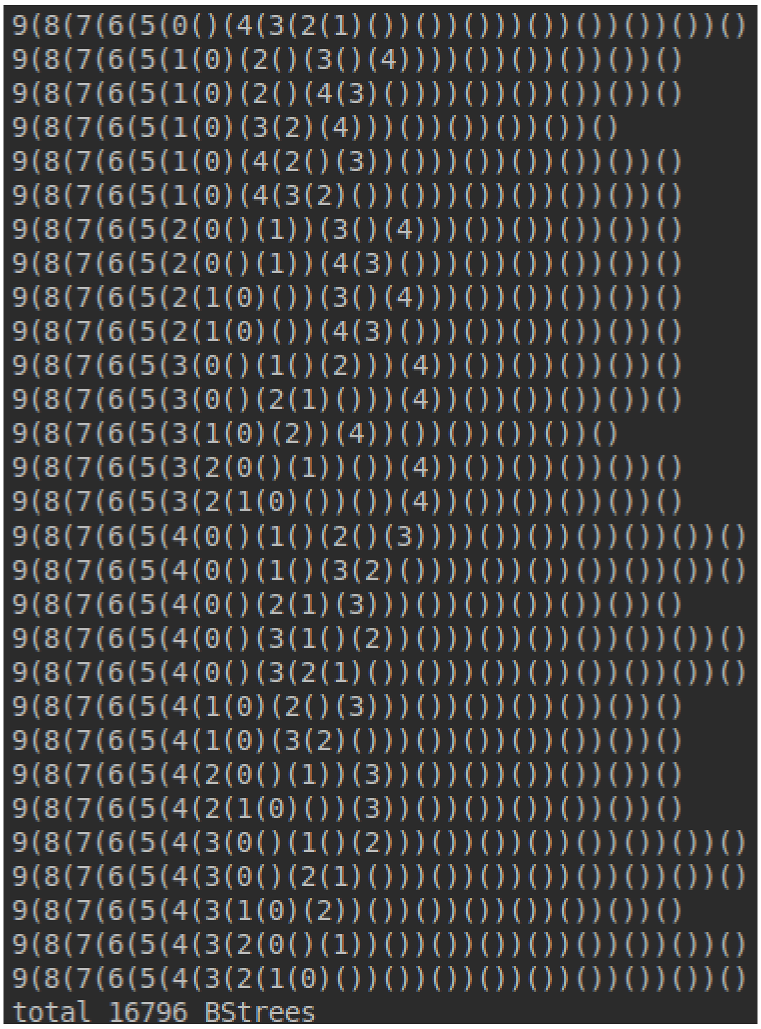


```
6(5(3(1(0)(2))(4))())()
6(5(3(2(0()(1))())(4))())()
6(5(3(2(1(0)())())(4))())()
6(5(4(0()(1()(2()(3))))())())()
6(5(4(0()(1()(3(2)())))())())()
6(5(4(0()(2(1)(3)))())())()
6(5(4(0()(3(1()(2))()))())())()
6(5(4(0()(3(2(1)())()))())())()
6(5(4(1(0)(2()(3)))())())()
6(5(4(1(0)(3(2)()))())())()
6(5(4(2(0()(1))(3))())())()
6(5(4(2(1(0)())(3))())())()
6(5(4(3(0()(1()(2)))())())())()
6(5(4(3(0()(2(1)()))())())())()
6(5(4(3(1(0)(2))())())())()
6(5(4(3(2(0()(1))())())())())()
6(5(4(3(2(1(0)())())())())())()
total 429 BStrees
```

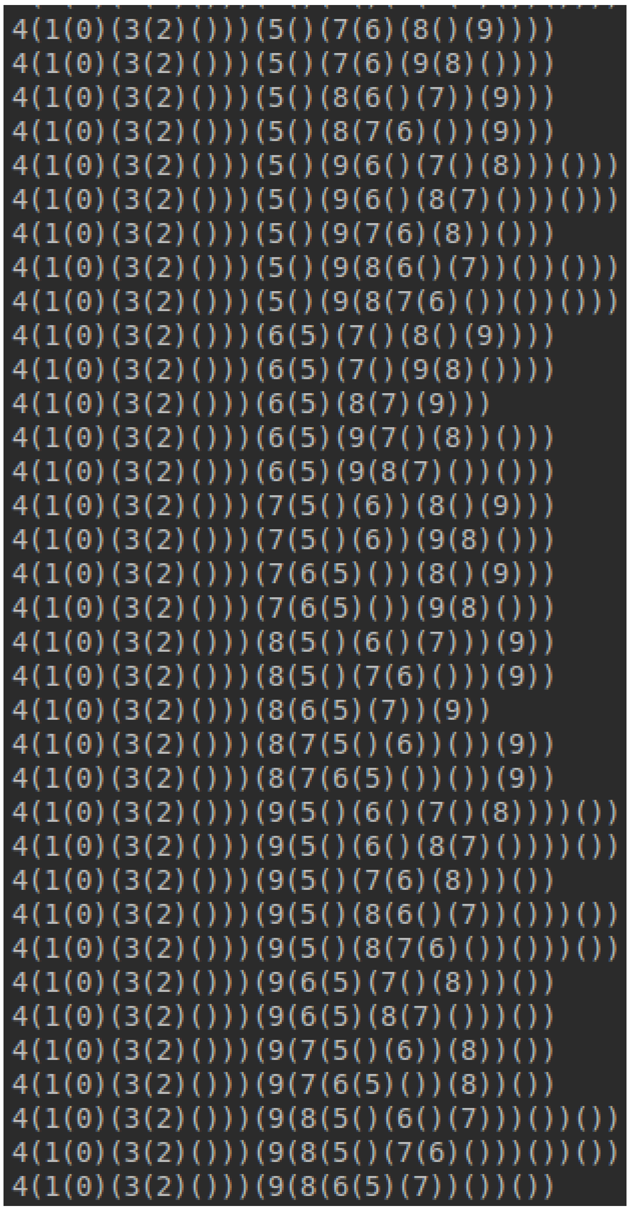
输入为 10



```
10
0()(1()(2()(3()(4()(5()(6()(7()(8()(9)))))))))
0()(1()(2()(3()(4()(5()(6()(7()(9(8)()))))))))
0()(1()(2()(3()(4()(5()(6()(8(7)(9))))))))
0()(1()(2()(3()(4()(5()(6()(9(7()(8))())))))))
0()(1()(2()(3()(4()(5()(6()(9(8(7)())())))))))
0()(1()(2()(3()(4()(5()(7(6)(8()(9))))))))
0()(1()(2()(3()(4()(5()(7(6)(9(8)())))))))
0()(1()(2()(3()(4()(5()(8(6()(7))(9)))))))
0()(1()(2()(3()(4()(5()(8(7(6)())(9)))))))
0()(1()(2()(3()(4()(5()(9(6()(7()(8)))()))))))
0()(1()(2()(3()(4()(5()(9(6()(8(7)()))()))))))
0()(1()(2()(3()(4()(5()(9(7(6)(8))()))))))
0()(1()(2()(3()(4()(5()(9(8(6()(7))())()))))))
0()(1()(2()(3()(4()(5()(9(8(7(6)())())()))))))
0()(1()(2()(3()(4()(6(5)(7()(8()(9))))))))
0()(1()(2()(3()(4()(6(5)(7()(9(8)())))))))
0()(1()(2()(3()(4()(6(5)(8(7)(9)))))))
0()(1()(2()(3()(4()(6(5)(9(7()(8))()))))))
0()(1()(2()(3()(4()(6(5)(9(8(7)())()))))))
0()(1()(2()(3()(4()(7(5()(6))(8()(9)))))))
0()(1()(2()(3()(4()(7(5()(6))(9(8)()))))))
0()(1()(2()(3()(4()(7(6(5)())(8()(9)))))))
0()(1()(2()(3()(4()(7(6(5)())(9(8)()))))))
0()(1()(2()(3()(4()(8(5()(6()(7)))(9))))))
0()(1()(2()(3()(4()(8(5()(7(6)()))(9))))))
0()(1()(2()(3()(4()(8(6(5)(7))(9))))))
0()(1()(2()(3()(4()(8(7(5()(6))())(9))))))
```

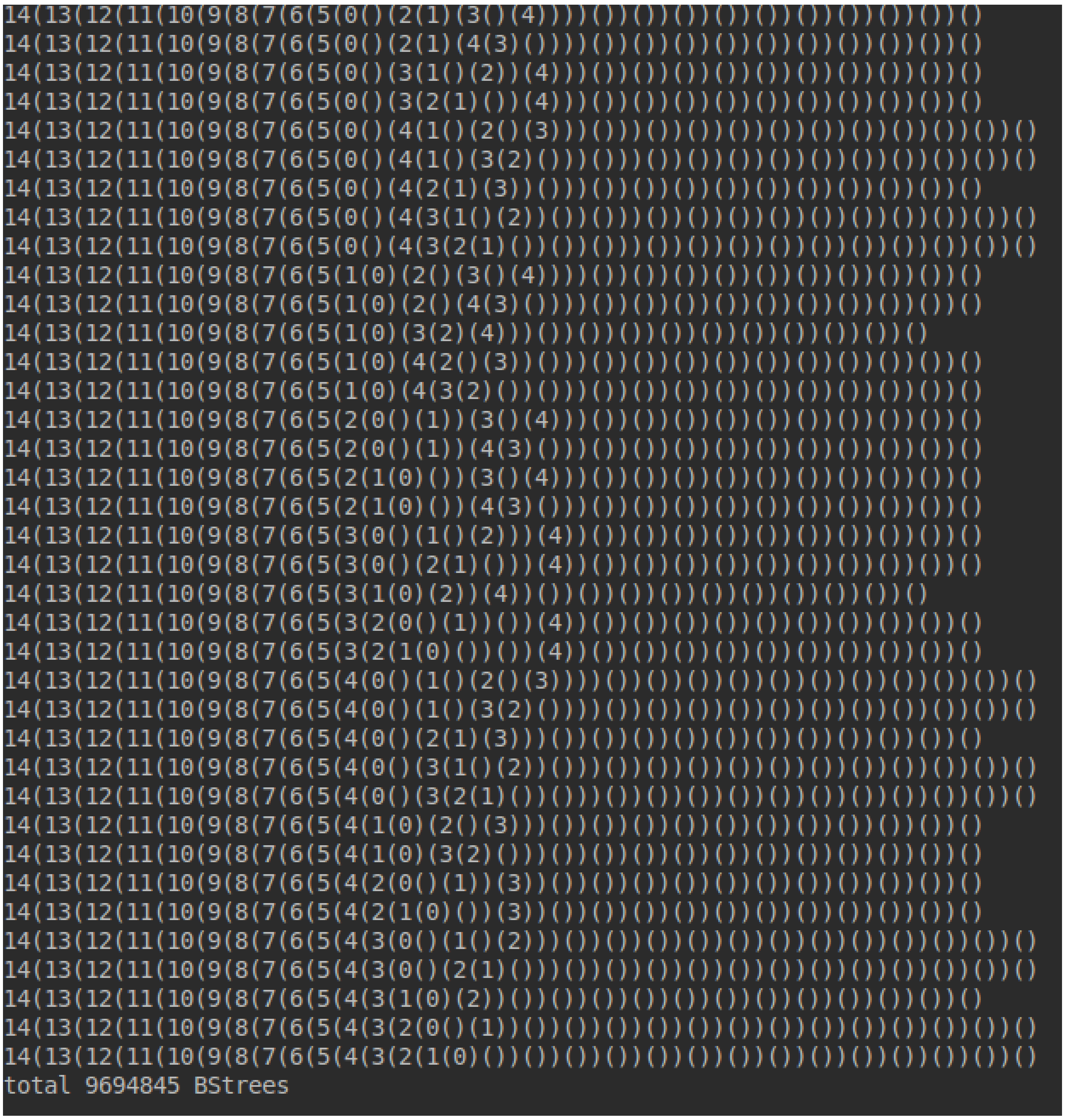


```
9(8(7(6(5(0()(4(3(2(1)())())()))())())())())()
9(8(7(6(5(1(0)(2()(3()(4))))())())())())()
9(8(7(6(5(1(0)(2()(4(3)())))())())())())()
9(8(7(6(5(1(0)(3(2)(4)))())())())())()
9(8(7(6(5(1(0)(4(2()(3))()))())())())())()
9(8(7(6(5(1(0)(4(3(2)())()))())())())())()
9(8(7(6(5(2(0()(1))(3()(4)))())())())())()
9(8(7(6(5(2(0()(1))(4(3)()))())())())())()
9(8(7(6(5(2(1(0)())(3()(4)))())())())())()
9(8(7(6(5(2(1(0)())(4(3)()))())())())())()
9(8(7(6(5(3(0()(1()(2)))(4))())())())())()
9(8(7(6(5(3(0()(2(1)()))(4))())())())())()
9(8(7(6(5(3(1(0)(2))(4))())())())())()
9(8(7(6(5(3(2(0()(1))())(4))())())())())()
9(8(7(6(5(3(2(1(0)())())(4))())())())())()
9(8(7(6(5(4(0()(1()(2()(3))))())())())())())()
9(8(7(6(5(4(0()(1()(3(2)())))())())())())())()
9(8(7(6(5(4(0()(2(1)(3)))())())())())())()
9(8(7(6(5(4(0()(3(1()(2))()))())())())())())()
9(8(7(6(5(4(0()(3(2(1)())()))())())())())())()
9(8(7(6(5(4(1(0)(2()(3)))())())())())())()
9(8(7(6(5(4(1(0)(3(2)()))())())())())())()
9(8(7(6(5(4(2(0()(1))(3))())())())())())()
9(8(7(6(5(4(2(1(0)())(3))())())())())())()
9(8(7(6(5(4(3(0()(1()(2)))())())())())())())()
9(8(7(6(5(4(3(0()(2(1)()))())())())())())())()
9(8(7(6(5(4(3(1(0)(2))())())())())())())()
9(8(7(6(5(4(3(2(0()(1))())())())())())())())()
9(8(7(6(5(4(3(2(1(0)())())())())())())())())()
total 16796 BStrees
```



```
4(1(0)(3(2)()))(5()(7(6)(8()(9))))
4(1(0)(3(2)()))(5()(7(6)(9(8)())))
4(1(0)(3(2)()))(5()(8(6()(7))(9)))
4(1(0)(3(2)()))(5()(8(7(6)())(9)))
4(1(0)(3(2)()))(5()(9(6()(7()(8)))()))
4(1(0)(3(2)()))(5()(9(6()(8(7)()))()))
4(1(0)(3(2)()))(5()(9(7(6)(8))()))
4(1(0)(3(2)()))(5()(9(8(6()(7))())()))
4(1(0)(3(2)()))(5()(9(8(7(6)())())()))
4(1(0)(3(2)()))(6(5)(7()(8()(9))))
4(1(0)(3(2)()))(6(5)(7()(9(8)())))
4(1(0)(3(2)()))(6(5)(8(7)(9)))
4(1(0)(3(2)()))(6(5)(9(7()(8))()))
4(1(0)(3(2)()))(6(5)(9(8(7)())()))
4(1(0)(3(2)()))(7(5()(6))(8()(9)))
4(1(0)(3(2)()))(7(5()(6))(9(8)()))
4(1(0)(3(2)()))(7(6(5)())(8()(9)))
4(1(0)(3(2)()))(7(6(5)())(9(8)()))
4(1(0)(3(2)()))(8(5()(6()(7)))(9))
4(1(0)(3(2)()))(8(5()(7(6)()))(9))
4(1(0)(3(2)()))(8(6(5)(7))(9))
4(1(0)(3(2)()))(8(7(5()(6))())(9))
4(1(0)(3(2)()))(8(7(6(5)())())(9))
4(1(0)(3(2)()))(9(5()(6()(7()(8))))())
4(1(0)(3(2)()))(9(5()(6()(8(7)())))())
4(1(0)(3(2)()))(9(5()(7(6)(8)))())
4(1(0)(3(2)()))(9(5()(8(6()(7))()))())
4(1(0)(3(2)()))(9(5()(8(7(6)())()))())
4(1(0)(3(2)()))(9(6(5)(7()(8)))())
4(1(0)(3(2)()))(9(6(5)(8(7)()))())
4(1(0)(3(2)()))(9(7(5()(6))(8))())
4(1(0)(3(2)()))(9(7(6(5)())(8))())
4(1(0)(3(2)()))(9(8(5()(6()(7)))())())
4(1(0)(3(2)()))(9(8(5()(7(6)()))())())
4(1(0)(3(2)()))(9(8(6(5)(7))())())
```

输入为 15



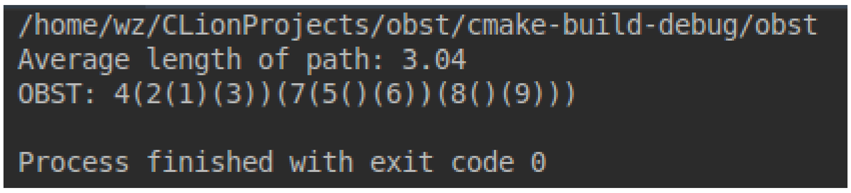
```
14(13(12(11(10(9(8(7(6(5(0()(2(1)(3()(4))))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(0()(2(1)(4(3)())))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(0()(3(1()(2))(4)))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(0()(3(2(1)())(4)))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(0()(4(1()(2()(3)))()))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(0()(4(1()(3(2)()))()))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(0()(4(2(1)(3))()))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(0()(4(3(1()(2))())()))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(0()(4(3(2(1)())())()))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(1(0)(2()(3()(4))))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(1(0)(2()(4(3)())))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(1(0)(3(2)(4)))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(1(0)(4(2()(3))()))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(1(0)(4(3(2)())()))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(2(0()(1))(3()(4)))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(2(0()(1))(4(3)()))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(2(1(0)())(3()(4)))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(2(1(0)())(4(3)()))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(3(0()(1()(2)))(4))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(3(0()(2(1)()))(4))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(3(1(0)(2))(4))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(3(2(0()(1))())(4))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(3(2(1(0)())())(4))())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(0()(1()(2()(3))))())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(0()(1()(3(2)())))())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(0()(2(1)(3)))())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(0()(3(1()(2))()))())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(0()(3(2(1)())()))())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(1(0)(2()(3)))())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(1(0)(3(2)()))())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(2(0()(1))(3))())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(2(1(0)())(3))())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(3(0()(1()(2)))())())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(3(0()(2(1)()))())())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(3(1(0)(2))())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(3(2(0()(1))())())())())())())())())())())()
14(13(12(11(10(9(8(7(6(5(4(3(2(1(0)())())())())())())())())())())())()
total 9694845 BStrees
```

5.2 输出最优二叉搜索树

输入为

第 k 个元素	0	1	2	3	4	5	6	7	8	9
b		.06	.06	.06	.06	.06	.06	.06	.06	.06
a	.1	.04	.04	.04	.04	.04	.04	.04	.04	.04

即课件上的例子

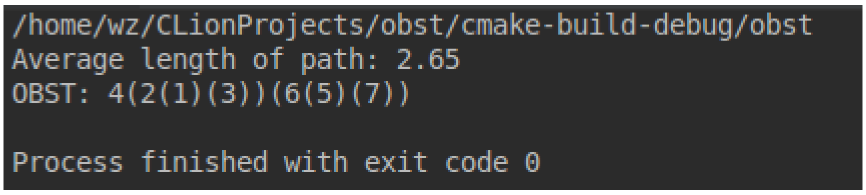

```
/home/wz/CLionProjects/obst/cmake-build-debug/obst
Average length of path: 3.04
OBST: 4(2(1)(3))(7(5()(6))(8()(9)))

Process finished with exit code 0
```

长度与结构都与例子相符。

输入为

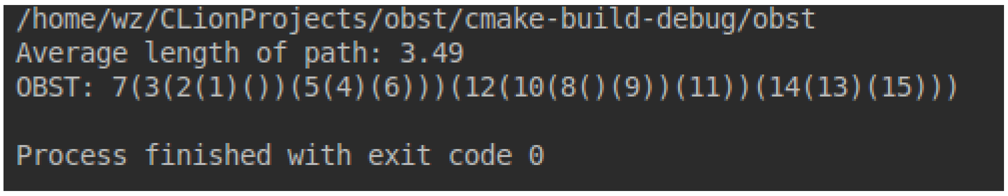
第 k 个元素	0	1	2	3	4	5	6	7	sum
b		.1	.08	.09	.11	.1	.05	.07	0.600
a	.05	.06	.03	.04	.06	.07	.05	.04	0.400


```
/home/wz/CLionProjects/obst/cmake-build-debug/obst
Average length of path: 2.65
OBST: 4(2(1)(3))(6(5)(7))

Process finished with exit code 0
```

输入为

第 k 个元素	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	sum
b		.02	.03	.045	.04	.045	.025	.06	.035	.025	.045	.05	.04	.045	.055	.035	0.5950
a	.025	.03	.035	.01	.015	.02	.025	.03	.015	.01	.02	.045	.03	.035	.02	.04	0.4050


```
/home/wz/CLionProjects/obst/cmake-build-debug/obst
Average length of path: 3.49
OBST: 7(3(2(1)())(5(4)(6)))(12(10(8()(9))(11))(14(13)(15)))

Process finished with exit code 0
```

6.总结

本次实验的输出问题主要是复习递归的算法，只要搞清除原问题与子问题之间的递归关系，就能简单的构造出递归函数，完成输出。另外主要是学习到了最优二叉搜索树的构造算法，该算法与矩阵链相乘的相似，但比矩阵链多了一个储存概率的矩阵 w，该矩阵也是这个算法中比较烧脑的部分，正确理解这个 w 的含义，并在实现中正确的初始化是实现该动态规划算法的关键。

7.存在问题和改进设想

这次实验的最优二叉搜索树的输出使用了中间信息 s，同矩阵链相乘一样，这个中间信息也是可以省略并直接依靠 m 找到最优二叉搜索树的结构的，但是需要查找的表元素同样较多，需要找元素所在的同一列和同一行中的所有靠前的元素，即对于 m[i][j]要对比所有 m[1:i-1][j]和 m[i][1:j-1]的元素，才能确定该问题是由那些子问题构造而成的，可以牺牲时间的情况下，获得了空间。

穷举二叉搜索树的递归表达式

对于有序序列 S 中的每个元素 k，需要递归前 k-1 个元素作为左子树，后 n-k 个元素作为右子树

因此根据该定义得到递归表达式为 $f(n)=\sum_{k=1}^{n} f(k-1)*f(n-k)=\sum_{k=0}^{n-1} f(k)*f(n-1-k)$

根据递归式随后得到递推式 $f(n)=\frac{C(2n,n)}{n+1}$

所以 $f(n-1)=\frac{C(2n-2,n-1)}{n}$

所以 $f(n)=\frac{4n-2}{n+1}f(n-1)$

所以 $f(n)=\frac{4n-2}{n+1}\frac{4n-6}{n}\cdots\frac{2}{2}=2^{n}\frac{2n-1}{n+1}\frac{2n-3}{n}\cdots\frac{1}{2}=2^{n}\prod_{k=1}^{n}\frac{2k-1}{k+1}=2^{n}\prod_{k=2}^{n+1}\frac{2k-3}{k}=2^{n}\prod_{k=2}^{n+1}2-\frac{3}{k}$

由于 $1\le 2-\frac{3}{k}<2, for\, 3\le k\le n+1$

所以 $2^{n}\le f(n)<4^{n}$

所以 $f(n)=O(4^{n})$ 或 $f(n)=\Omega(2^{n})$