# 520 第六周学习总结

解决课上问的三个问题，

1. 第22题（括号生成）如何用DP做？

自底向上：从小规模问题开始，逐渐得到大规模问题的解集

无后效性：后面的结果，不会影响前面的结果。

1.首先，问题拆解，自底向上的思路，从终点开始想，这时候已经产生了可以配对的n个左括号和n个右括号，称之为有i对括号，那么i对括号其实是在i-1对括号基础上得到的。i对括号的组合中，最左边第一个左括号肯定是开始，但是和它配对的右括号不一定是最后一个括号（比如n是3，极端是((()))，但是也可能是(())()的情况，和第一个左括号配对的右括号不一定是最后一个右括号）

2.定义状态，dp[i]表示，使用i对括号能够生成的组合数量。注意，每个状态都是列表的形式。

3.状态转移方程：dp[i] = ‘(’ + dp[可能的括号对数]+’)’+dp[剩下的括号对数]

注意，可能的括号对数+剩下的括号对数=i-1，假设”可能的括号对数”为j，那么因为 剩下的括号对数+j=i-1，所以 剩下的括号对数=i-j-1

“

Let us consider an example to get clear view:

f(0): ""

f(1): "("f(0)")"

f(2): "("f(0)")"f(1), "("f(1)")"

f(3): "("f(0)")"f(2), "("f(1)")"f(1), "("f(2)")"

So f(n) = "("f(0)")"f(n-1) , "("f(1)")"f(n-2) "("f(2)")"f(n-3) ... "("f(i)")"f(n-1-i) ... "(f(n-1)")" ”

所以变化上面的状态转移方程，可以得到：

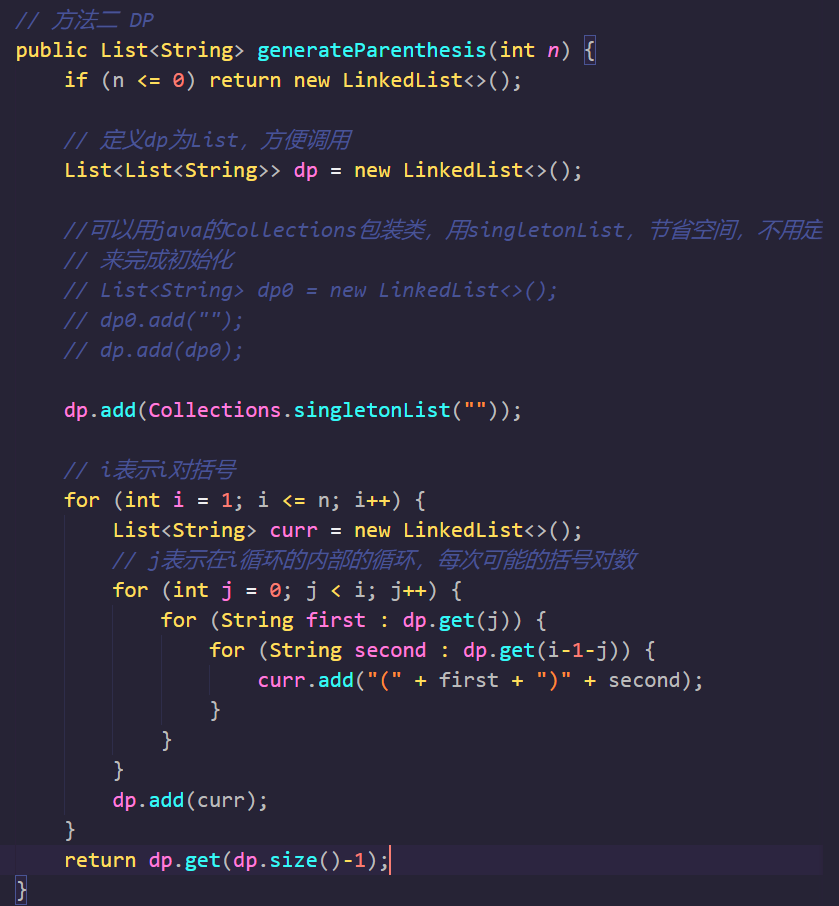
dp[i]=’(’+dp[j]+’)’+dp[i-j-1]，j=0,1,…,i-1

4.输出实现

初始状态：因为我们需要0对括号这个初始状态，因此状态数组dp从0开始，第0对括号就是空的字符串，这个状态定义成dp0.

输出内容dp[n]。

代码：



二.212题（用DFS和Trie两种方法的时间复杂度）

方法一，使用DFS，遍历words，然后到board里面查找。这种方式的时间复杂度分析：首先遍历一遍word，要O(n)，然后如果假设给得board二维数组是m\*m大小的，要对这个二维数组进行深度优先遍历，那么整个遍历的时间复杂度会是O(N\*m\*m)。而且最坏的情况下，每个字母还要向周围的上下左右四个方向散开去，所以时间复杂度还要乘上 4^k。4指四个方向，k指单词的平均长度为k。 综上，方法一的整体的时间复杂度为O(N\*m\*m\*4^k)。

方法二，使用前缀树。类似，board数组要遍历，如果是m\*n的大小，那么遍历的时间复杂度就是O(m\*n)，因为有四个方向，需要进行四次递归调用，所以要有4的指数次幂的时间复杂度：O(4^wl)，如果已经构造了字典树，可以省略N，顾整个的时间复杂度是：O(m\*n\*4^k)

1. 总结双向BFS的模板代码

注：代码来源：<https://blog.csdn.net/qq_42321579/article/details/82916543>

void BFS\_bothsides()//双向BFS

{

if(s1.state==s2.state)//起点终点相同时要特判

{

//do something

found=true;

return;

}

bool found=false;

memset(visited,0,sizeof(visited)); // 判重数组

while(!Q1.empty()) Q1.pop(); // 正向队列

while(!Q2.empty()) Q2.pop(); // 反向队列

//======正向扩展的状态标记为1，反向扩展标记为2

visited[s1.state]=1; // 初始状态标记为1

visited[s2.state]=2; // 结束状态标记为2

Q1.push(s1); // 初始状态入正向队列

Q2.push(s2); // 结束状态入反向队列

while(!Q1.empty() || !Q2.empty())

{

if(!Q1.empty())

BFS\_expand(Q1,true); // 在正向队列中搜索

if(found) // 搜索结束

return ;

if(!Q2.empty())

BFS\_expand(Q2,false); // 在反向队列中搜索

if(found) // 搜索结束

return ;

}

}

void BFS\_expand(queue<Status> &Q,bool flag)

{

s=Q.front(); // 从队列中得到头结点s

Q.pop()

for( 每个s 的子节点 t )

{

t.state=Gethash(t.temp); // 获取子节点的状态

if(flag) // 在正向队列中判断

{

if(visited[t.state]!=1)// 没在正向队列出现过

{

if(visited[t.state]==2) // 该状态在反向队列中出现过

{

各种操作；

found=true；

return;

}

visited[t.state]=1; // 标记为在在正向队列中

Q.push(t); // 入队

}

}

else // 在正向队列中判断

{

if (visited[t.state]!=2) // 没在反向队列出现过

{

if(visited[t.state]==1) // 该状态在正向向队列中出现过

{

各种操作；

found=true；

return;

}

visited[t.state]=2; // 标记为在反向队列中

Q.push(t); // 入队

}

}

}

}