# 栈

## 32最长有效括号

这个题目还是适合用栈来做，用动态规划太难想了，首先遇到(的时候，我们就入栈，等待和它匹配的)，如果是)那就可以出栈了，但是我们要考虑的就是如何计算这个有效括号串的长度，这个是难点，我也是看了答案才理解的

要计算有效括号的长度可以利用索引，比如

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| ( | ) | ( | ( | ( | ) | ) |

这样3-6 就是匹配的，用6-3=3 还是不对，我们要**记录的是匹配位置的前一个位置的索引2**，然后用6即现在的)减掉匹配位置的前一个位置6-2=4

如果遇到的是( 就是等待匹配的)，把它的索引入栈，这个索引记录了匹配的前一个位置

如果遇到的是) 那就是要匹配了，先把和它匹配的那个元素弹栈出来，就露出了栈顶，栈顶其实就是和它匹配的前一个位置的索引，

1.如果弹栈以后，栈是空的，并没有匹配的前一个位置的索引，那就说明其实遇到的)和弹出来的元素并没有匹配上，我们就把)存入栈中作为下一次匹配的前一个位置的索引

2.如果弹栈以后，栈不空，那就说明匹配上了，栈顶保存的就是和它匹配的前一个位置的索引，只要把遇到的)的索引，减掉栈顶元素，就是新得到的局部的最长有效括号，如果比以前得到的还要长，就更新

public class Solution {

public int longestValidParentheses(String s) {

int maxans = 0;

Stack<Integer> stack = new Stack<>();

stack.push(-1);

for (int i = 0; i < s.length(); i++) {

if (s.charAt(i) == '(') {

stack.push(i);

} else {

stack.pop();

if (stack.empty()) {

stack.push(i);

} else {

maxans = Math.max(maxans, i - stack.peek());

}

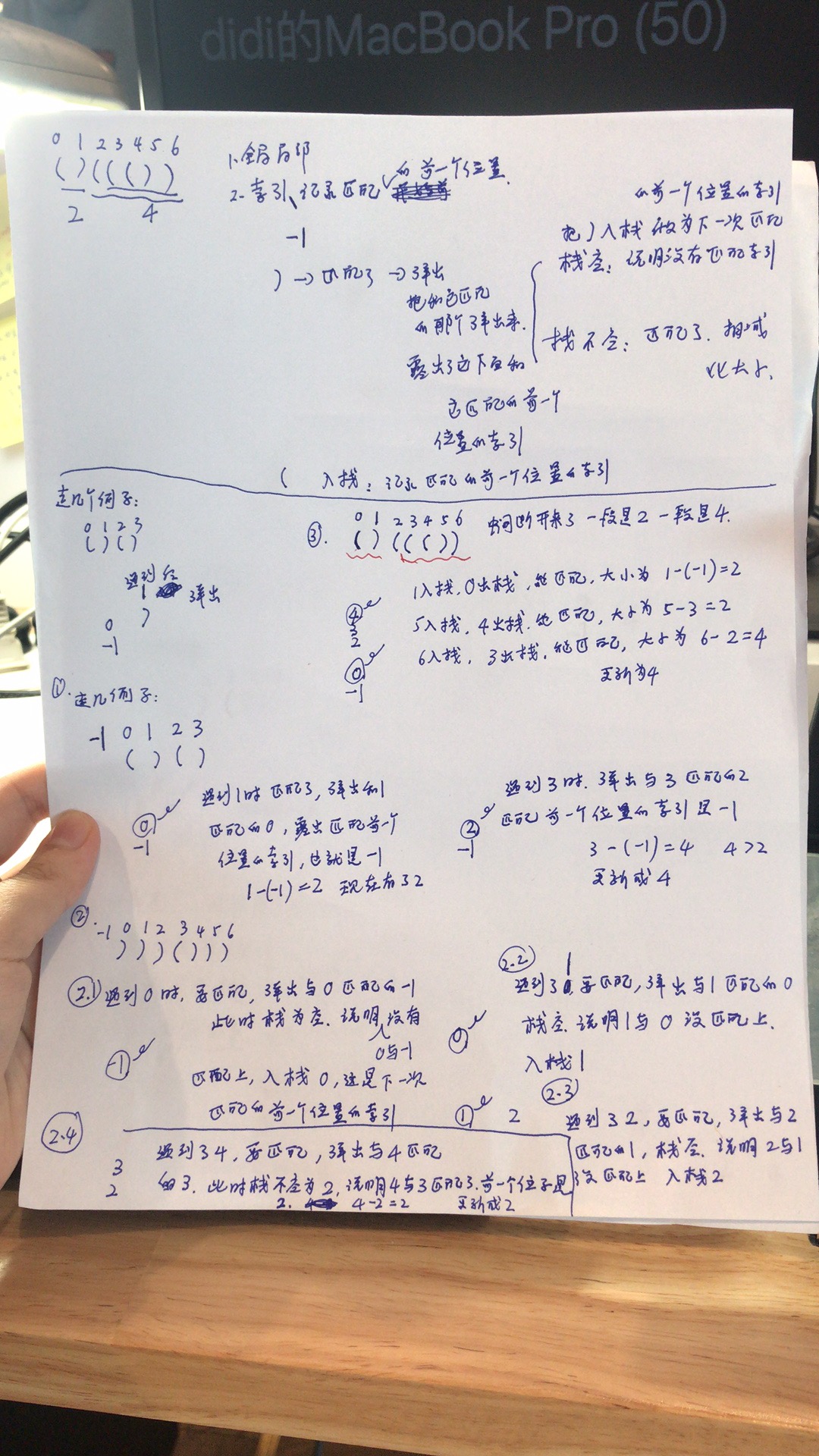
}

}

return maxans;

}

}



# 动态规划

## 70.爬楼梯

f(n)表示爬n个楼梯的方法数，动态规划都是反着想最后一步，最后一步有两种（1+1和一步登天），想到加法原理，所以爬n个楼梯的方法数等于我先到n-1的方法数+我先到n-2的方法数，也就是f(n)=f(n-1)+f(n-2) ，我担心会不会算重了，但是并不会，因为最后一步不一样，就算前面步骤都一样，也不算重复的方法

f(1)=1

f(2)=2

f(3)=f(2)+f(1)=3

f(4)=f(3)+f(2)=5

f(5)=f(4)+f(3)=8

…

看着眼熟，这就是**斐波那契数列**咩，那就简单了，只要存上次和上上次的结果，就能算出这次了，这是最省空间的了

class Solution {

public:

int climbStairs(int n) {

if(n==1)

return 1;

if(n==2)

return 2;

else

{

int now=-1;

int last=2;

int lastlast=1;

for(int i=3;i<=n;i++)

{

now=last+lastlast;

lastlast=last;

last=now;

}

return now;

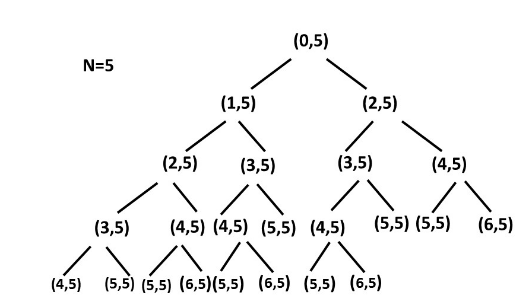
}

}

};

时间O(n) 空间O(1)

虽然简单但是看看暴搜的思路也很有意思，暴搜就是从第一步开始搜，**与动态规划恰好相反**，我或者走1步，或者走2步，走了1步之后再走1步或两步，这样形成一棵搜索树，然后把能走到n的分支数都加起来



public class Solution {

public int climbStairs(int n) {

climb\_Stairs(0, n);

}

public int climb\_Stairs(int i, int n) //这个函数就返回在第i个阶梯上有几个分支能成功的

{

if (i > n) {

return 0;//不成功啥也没有

}

if (i == n) {

return 1;

}

return climb\_Stairs(i + 1, n) + climb\_Stairs(i + 2, n);//每一个小函数都会递归调用下去形成分支

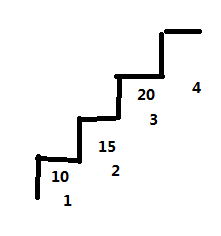
}

}

时间复杂度O(2^n)

## 71.最小代价爬楼梯

首先这个题有个小坑，



它所谓的cost={10,15,20} 20的那个台阶不是最后的那个台阶，是最后一个台阶的前一个台阶，10代表的是从台阶1到台阶2或台阶3，我需要付出的代价是10

所以为了好理解，我的索引从0开始:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| C  (原costvector) |  | 10 | 15 | 20 |  |
| costResult |  | 0 | 0 | 10 |  |

状态转移方程:

costResult[n]=min{costResult[n-1]+c[n-1],costResult[n-2]+c[n-2]}

costResult[1]=0

costResult[2]=0

例如costResult[3]=10=min{15,10}

#include<iostream>

#include<vector>

using namespace std;

int main()

{

vector<int> cost{1,100,1,1,1,100,1,1,100,1};

if(cost.size()==2)

{

//cout<<"cost= 0"<<endl;

return 0;

}

int c[cost.size()+1];

c[0]=0;

for(int i=1;i<=cost.size();i++)

{

c[i]=cost[i-1];

//cout<<c[i]<<" "<<endl;

}

int n=cost.size()+1;

//cout<<"n="<<n<<endl;

int costResult[n+1];

costResult[1]=0;

costResult[2]=0;

for(int i=3;i<=n;i++)

{

int first=costResult[i-1]+c[i-1];

int sec=costResult[i-2]+c[i-2];

//cout<<"first="<<first<<endl;

//cout<<"second="<<sec<<endl;

costResult[i]=first<sec?first:sec;

//cout<<"costResult="<<costResult[i]<<endl;

}

cout<<"cost="<<costResult[n]<<endl;

}