


04-D 括号匹配

#数据结构邓神

什么是栈结构? 

逆序输出	<ul style="list-style-type: none">• conversion• 输出次序与处理过程颠倒；递归深度和输出长度不易预知
递归嵌套	<ul style="list-style-type: none">• stack permutation + parenthesis• 具有自相似性的问题可递归描述，但分支位置和嵌套深度不固定
延迟缓冲	<ul style="list-style-type: none">• evaluation• 线性扫描算法模式中，在预读足够长之后，方能确定可处理的前缀
栈式计算	<ul style="list-style-type: none">• RPN• 基于栈结构的特定计算模式

Data Structures & Algorithms (Fall 2013), Tsinghua University

实例

实例

❖ $(a[i-1][j+1]) + a[i+1][j-1]) * 2$ //失配

$(a[i-1][j+1] + a[i+1][j-1]) * 2$ //匹配

我们可以忽略除了括号的另外符号

尝试

尝试

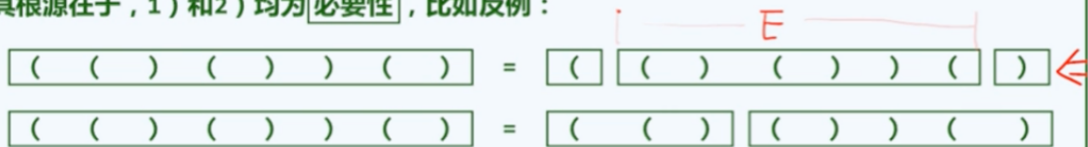
0) 平凡：无括号的表达式是匹配的 \emptyset ✓

1) 减治？ \boxed{E} 匹配，仅当 (\boxed{E}) 匹配

2) 分治？ \boxed{E} 和 \boxed{F} 均匹配，仅当 $\boxed{E} \boxed{F}$ 匹配

这些性质我们都很难使用之前的策略

❖ 究其根源在于，1) 和 2) 均为必要性，比如反例：



分别否定了减而治之和分而治之

构思

❖ 颠倒以上思路：消去一对紧邻的左右括号，不影响全局的匹配判断

亦即：

L

 ()

R

 匹配，仅当

L

R

 匹配

构思

❖ 那么，如何找到这对括号？

再者，如何使问题的这种简化得以持续进行？

用栈处理

❖ 顺序扫描表达式，用栈记录已扫描的部分

//实际上只需记录左括号

反复迭代：凡遇 (，则进栈；凡遇)，则出栈



最后条件就是如果栈非空或者中间遇到不匹配的就是不配

最后如果栈是空的表示匹配

应用

实现

❖ bool paren(const char exp[], int lo, int hi) { //exp[lo, hi)

✓ Stack<char> S; //使用栈记录已发现但尚未匹配的左括号

for (int i = lo; i < hi; i++) //逐一检查当前字符

if ('(' == exp[i]) S.push(exp[i]); //遇左括号：则进栈

else if (!S.empty()) S.pop(); //遇右括号：若占非空，则弹出左括号

else return false; //否则（遇右括号时栈已空），必不匹配

→ return S.empty(); //最终，栈空当且仅当匹配

}

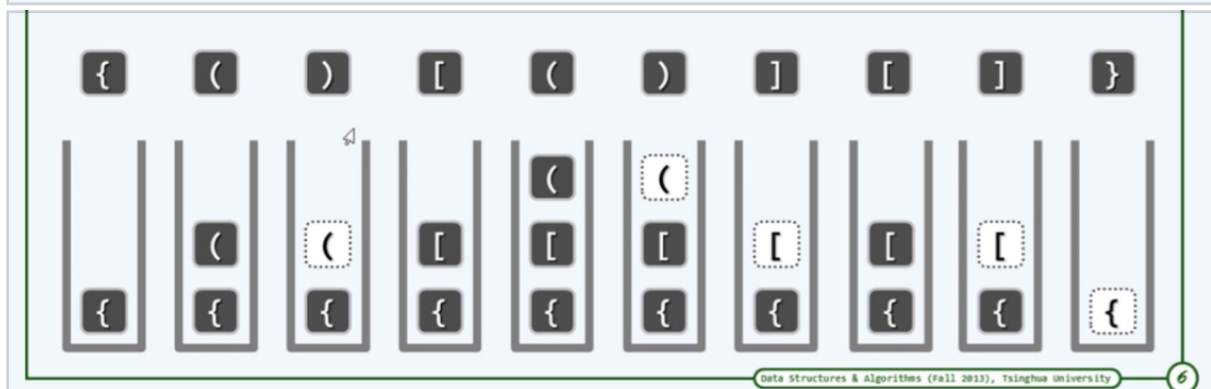
```
bool paren(const char exp[],int lo,int hi){
    stack<char> S;
    for (int i = lo; i < hi; ++i) {
        if ( '(' == exp[i]){
            S.push(exp[i]);
        }else if ( !S.empty()){
            S.pop();
        }else {
            return false;
        }
    }
    return S.empty();
}
```

实例

每次遇到左括号则是+1，每次右括号就-1，最后减1，也可以解决

而计数器则无法处理

❖ 可否，使用多个计数器？不行，反例：[()]



比如: `<body>`|`</body>`, `<h1>`|`</h1>`, ``|``, `<p>`|`</p>`, ``|``, ...