

**课 程 实 验 报 告**

**课程名称： 串行与并行数据结构及算法**

**专业班级： ACM1701**

**学 号： U201714850**

**姓 名： 丁文隆**

**指导教师： 陆枫**

**报告日期： 　2018.11.21**

**计算机科学与技术学院**

## Lab1 括弧匹配实验

### 1. 实验要求

给定一个由括号构成的串，若该串是合法匹配的，返回串中所有匹配的括号对中左右括号距离的最大值；否则返回NONE。左右括号的距离定义为串中二者之间字符的数量，即*max { j - i + 1 | (si, sj)* 是串s中一对匹配的括号}。要求分别使用枚举法和分治法求解。

### 2. 实验思路

#### 2.1 分治法求解思路

|  |
| --- |
| 1.首先对于一个定义一个(m,l,r,ld,rd),其中每个部分分别代表的意义为：  m:闭括号中最大括号长度  l:没有匹配的左括号数  r:没有匹配的右括号数  ld:最左边的左括号到串末尾的距离  rd:最右边右括号到串开始处的距离  2.设定空串：EMPTY（0,0,0,0,0）  左括号：OPAREN(0,1,0,1,0)  右括号：CPAREN(0,0,1,0,1)  3.如果不是空的或者左右括号，则分为左右两边(m1, l1, r1, ld1, rd1), (m2, l2, r2, ld2, rd2)，两边并行处理，合并规则为:  1).重新计算闭括号最大长度以及左右未匹配括号  如果左边没匹配左括号与右边没匹配右括号相等，则计算中间闭括号长度m0与m1和m2比较，得到最大值  如果左边没匹配左括号比右边没匹配右括号要少，合并之后的l0就是右边的左括号数，合并之后的r0是左右两边括号总数减去已匹配的左边左括号数  如果左边没匹配左括号不比右边没匹配右括号要少，合并之后l0就是两边括号数之和减去右边已经匹配的右括号，合并之后的r0就是左边的右括号数  2).重新计算左边最左到末尾以及右边最右到开始的距离  如果左边没匹配左括号数大于右边没匹配右括号数，则左边左括号没匹配完，新的ld就是ld1加上右边的总长度，否则就是右边的左括号数  如果右边没匹配右括号数大于左边没匹配左括号数，则右边右括号没匹配完，新的的rd就是rd2加上左边的总长度，否则就是左边的右括号数  4.对于输入的串parens，返回其最长闭括号所含元素(max)，未匹配左右括号数，如果未匹配左右括号数均为0并且输入的串本身长度大于0，则最长闭括号所含元素为max，否则左右括号是没有匹配的，返回NONE  (具体实践如代码所示) |

### 3. 回答问题

#### 3.1 关于枚举法求解

Task 5.2 (5%). What is the work and span of your brute-force solution? You should assume subseq has *O(1)* work and span, where m is the length of the resulting subsequence, and parenMatch has *O(n)* work and *O(log2n)* span where *n* is the length of the sequence.

|  |
| --- |
| 1. Work: brute force算法遍历了所有的连续闭子串，如果我们假设subseq操作用*O(1)*的work，而判断一个串是否匹配，即是parenMatch函数，用O(n)的work，那么判断我们所取出的一个子串是否匹配（即parenMatch函数）用work就是O(n)。现在我们计算MatchDist函数的复杂度，它计算第m位开始的(第m位为左括号时)闭括号的长度，由于它要判断逐一m以后括号是否匹配并返回长度，并且每次判断是都调用parenMatch函数，故执行一次这个函数的work为所以我们用tabulate从第位到第n位调用MatchDist，记录从第1位到第n位所有连续闭子串长度的work为，然后我们计算n个长度中最大的长度，由于我们是两两都要比较，这一步用的work是，所以综上，这个算法的work是 2. Span: 我们运用subseq操作用的span是O(1)，而判断一个串是否匹配，即parenMatch函数，用的span是，这个就是我们计算每个闭子串所需要的span，所以我们用tabulate记录所有连续闭子串长度所用的span为。（所有子串并行处理。）最后我们要计算n和长度中最大的元素，由于是所有n个值都两两比较，这一步的span为所以综上，这个算法的span是。 |

#### 3.2 关于分治法求解

Task 5.4 (20%). The speciﬁcation in Task 5.3 stated that the work of your solution must follow a recurrence that was parametric in the work it takes to view a sequence as a tree. Naturally, this depends on the implementation of SEQUENCE.

1. Solve the work recurrence with the assumption that *Wshowt∈Θ(lg n)* where *n* is the length of the input sequence.

2. Solve the work recurrence with the assumption that *Wshowt∈Θ(n)* where n is the length of the input sequence.

3. In two or three sentences, describe a data structure to implement the sequence α seq that allows showt to have *Θ(lg n)* work.

4. In two or three sentences, describe a data structure to implement the sequence α seq that allows showt to have *Θ(n)* work.

|  |
| --- |
| divide and conquer 的 work可以表示成:  W(n)=2\*W(n/2)+Wshowt(n)+O(1)  1.对于第一种情况，即*Wshowt∈Θ(lg n)*  W(n)=2\*W(n/2)+ Θ(lg n)，我们令n=，所以我们有这颗树的第一层work为lgn，第二层为2lg(n/2)，第k层有而我们可以看到第二层与第一层work的比为2lg(n/2)/lgn = 2-2/lgn > 1（因为n>2），所以这棵树是leaf-dominated的，即W(n)=O(    2.对于第二种情况，即*Wshowt∈Θ(n)*  即W(n)=2\*W(n/2)+ *Θ(n)* ,所以我们有这颗树第一层work为n，第二层work为2\*(n/2)=n，第k层为n。这颗树是balanced的，一共有logn层，故W(n)=(nlogn)    3.用一棵balanced tree来存这个串，这种情况下showt会产生一棵高为logn的树，复杂度也就为O(logn)  4.用一个头指针在链表表头的链表去存这个串，所以showt每次都要遍历这个链表去执行操作，因为这个串的长度是n，所以复杂度就是O(n) |
|  |

#### 3.3 关于渐进复杂度分析

Task 6.1 (5%). Rearrange the list of functions below so that it is ordered with respect to O—that is, for every index *i*, all of the functions with index less than i are in big-O of the function at index *i*. You can just state the ordering; you don’t need to prove anything.

1.

2. *f(n) = 2n1.5*

3. *f(n) =(nn)!*

4. *f(n) = 43n*

5. *f(n) = lg(lg(lg(lg(n))))*

6. *f(n) = 36n52 + 15n18 + n2*

7. *f(n) = nn!*

|  |
| --- |
| 复杂度由低到高为：  5 2 6 1 4 7 3 |

Task 6.2 (15%). Carefully prove each of the following statements, or provide a counterexample and prove that it is in fact a counterexample. You should refer to the deﬁnition of big-O. Remember that verbose proofs are not necessarily careful proofs.

1. *O* is a transitive relation on functions. That is to say, for any functions *f*, *g*, *h*, if *f∈O(g)* and *g∈O(h)*, then *f∈O(h)*.

2. *O* is a symmetric relation on functions. That is to say, for any functions *f* and *g* , if *f∈O(g)* , then *g∈O(f)* .

3. *O* is an anti-symmetric relation on functions. That is to say, for any functions *f* and *g* , if *f∈O(g)* and *g∈O(f)*, then *f*=*g*.

|  |
| --- |
| 1.传递性证明：因为 f∈O(g) 和 g∈O(h)，我们有一定存在，使得对于n>有f(n)<，且对于n>，所以得到，对于n>max{}，有f(n)<，所以f∈O(h)  2.对称性反例：由big-O的定义有，存在c和，使得对于所有的n>，有f(n)<c\*g(n)成立。我们可以看出2=O()，在（c = 1 , ）情况下成立，而我们找不出c和，让所有的n>，有f(n)<c\*g(n)成立。故这和个性质时是不存在的。  3.反对称性反例：我们可以设f(n)=n+1，g(n)=n+2, 由big-O的定义有,存在c=1，对一切n>0都有f(n)<g(n)，即f*∈O(g)*。而我们也存在c=2，对一切n>1,有g(n)<2\*f(n),即*g∈O(f)。*  而此时并没有f=g，故这个性质不成立。 |