

**课 程 实 验 报 告**

**课程名称： 串并行与数据结构及算法**

**专业班级： ACM1601**

**学 号： U201614831**

**姓 名： 苏墨馨**

**指导教师： 陆枫**

**报告日期： 　2018年1月24日**

**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 枚举法求解思路

|  |
| --- |
| 对括号中两个位置i j( i <= j )判断从i开始到j的子串是否匹配，如果匹配则记录i和j的距离作为左右括号匹配的距离。求出每两个位置之间的距离，找出最大值。 |

#### 2.2 分治法求解思路

|  |
| --- |
| 利用divide and conquer求解分为divide和combine两个部分。divide的时候用showt函数分为空串，只有左括号的串，只有右括号的串以及两个子串的情况。前三个情况作为base case，记录剩余的未匹配左括号数，左括号的最大匹配距离，剩余的未匹配右括号数，右括号的最大匹配距离，以及最大匹配距离。 两个子串的时候combine，combine的时候根据左子串左括号未匹配数和右子串右括号未匹配数计算新串左右括号未匹配数和左右括号最大匹配距离，以及最大匹配距离。最后得到整个串的最大匹配距离。 |

### 3. 回答问题

#### 3.1 关于枚举法求解

Task 5.1 (15%). Complete the functor MkBruteForcePD in the ﬁle MkBruteForcePD.sml with a brute-force solution to the maximum parenthesis distance problem. You may use the the solution to the parenthesis matching problem from recitation 1. You may also ﬁnd Seq.subseq to be useful for your solution. Please ensure that you understand the deﬁnition of a brute-force solution before attempting this task (we have received many non-brute-force solutions in the past).

|  |
| --- |
| functor MkBruteForcePD (structure P : PAREN\_PACKAGE) : PAREN\_DIST =  struct  structure P = P  open P  open Seq  fun matchParens parens =  let  val s = map (fn x=>if x = OPAREN then 1 else ~1) parens  val (s1,ans) = scan (op +) 0 s  fun copy(a,b) = if a <b then a else b  val (s2,ans2) = scan copy 0 s1  in  if ans <> 0 orelse ans2 < 0 then false else true  end  fun dis (parens,i,j) = (\*左右括号最大匹配的距离\*)  if nth parens j = CPAREN andalso nth parens i = OPAREN andalso matchParens (subseq parens (i+1,j-i-1))  then SOME(j-i-1)  else NONE    fun cmp (NONE,NONE) = NONE  |cmp(SOME a,NONE) = SOME a  |cmp(NONE,SOME a) = SOME a  |cmp(SOME a,SOME b) = if (a>b) then SOME a else SOME b  fun parenDist (parens : paren seq) : int option =(\*o(n4)\*)  let  val max = 0  fun loop (i,j) =  if i = length parens then NONE  else if j = length parens  then loop (i+1,i+1)  else cmp(dis (parens,i,j), loop (i,j+1))  in  if matchParens parens = false then NONE  else loop (0,0)  end  end |

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.

|  |
| --- |
| Work = O(n4) Span = O(n2lgn) |

#### 3.2 关于分治法求解

Task 5.3 (30%). Complete the functor MkDivideAndConquerPD in the corresponding ﬁle with a divide-and-conquer solution as described above. For this assignment, you are not required to submit a proof of correctness of your implementation. However, we advise that you work out a proof by mathematical induction for your solution as an exercise.

|  |
| --- |
| functor MkDivideAndConquerPD (structure P : PAREN\_PACKAGE) : PAREN\_DIST =  struct  structure P = P  open Primitives  open P  open Seq  fun cmp s1 s2 = if s1 = s2 then EQUAL  else if s1 > s2 then GREATER  else LESS  fun parenDist (parens : paren seq) : int option =  let  fun dis par = case showt par of  EMPTY => (0,0,0,0,SOME 0)  |ELT OPAREN => (1,0,0,0,SOME 0)  |ELT CPAREN => (0,0,1,0,SOME 0)  |NODE (parens1,parens2) =>  let  val ((l1,ldis1,r1,rdis1,max1),(l2,ldis2,r2,rdis2,max2))= Primitives.par(fn()=>dis parens1,fn()=> dis parens2)  val (l3,ldis3,r3,rdis3,max3) =  case cmp l1 r2 of  EQUAL =>(l2,ldis2,r1,rdis1,SOME(ldis1 + rdis2))  |GREATER =>(l1 + l2 - r2,ldis1 + length parens2,r1,rdis1,Option210.intMax(max1,max2))  |LESS => (l2,ldis1,r1 + r2 - l1,rdis2 + length parens1,Option210.intMax(max1,max2))  in  (l3,ldis3,r3,rdis3,max3)  end    val (l,\_,r,\_,maxf) = dis parens  in  if length parens > 0 andalso l = 0 andalso r = 0 then maxf  else NONE  end  end |

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.

|  |
| --- |
| WparenDist(n) = 2 · WparenDist(n/2) + Wshowt(n) + O(1)   1. 当*Wshowt∈Θ(lg n)*时，*W(n) = 2W(n/2)+O(lgn)* 则 W(n) = O(lgn) 2. 当*Wshowt∈Θ (n)*时，*W(n) = 2W(n/2)+O(n)* 则 W(n) = O(nlgn) 3. Tree sequence可以用*Θ(lg n)*的work实现showt 4. List sequence 可以用*Θ(n)*的work实现showt |

#### 3.2 关于代码测试

Task 5.5 (5%). In this course you will be expected to test your code extensively. For this assignment you should make sure you thoroughly and carefully test both of your implementations of the PAREN\_DIST signature. Your tests should include both edge cases and more general test cases on speciﬁc sequences.

|  |
| --- |
| 测试样例：    测试结果：   1. Brute-force:      1. Divide-and-conquer: |

#### 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!*

|  |
| --- |
| Ordering:3>7>4>1>6>2>5 |

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)* .

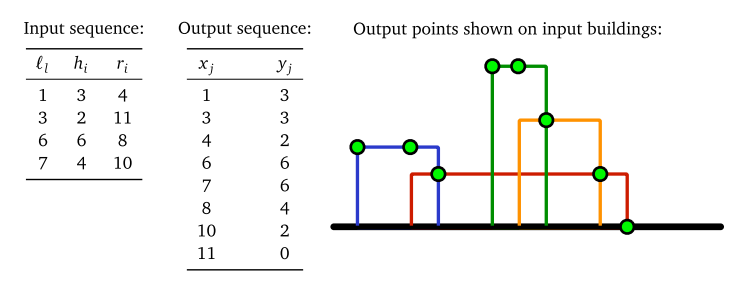
3. *O* is an anti-symmetric relation on functions. That is to say, for any functions *f* and *g* , if *f∈O(g)* and *g∈O(f)*, then *f*=*g*.

|  |
| --- |
| 证明：  1.∵*f∈O(g)*∴ 存在正常数n1,c1使得当n > n1时，f(n) ≤ c1·g(n)  ∵*g∈O(h)*∴ 存在正常数n2,c2使得当n > n2时，g(n) ≤ c2·h(n)  ∴当n > max{n1,n2}时，f(n) ≤ c1·g(n) ≤ c1·c2·g(n) 由定义知，*f∈O(h)*  2. 反例：f(x) = log x g(x) = x  取n0 = 1 c = 1当n > n0时f(n) ≤ g(n)，因此满足*f∈O(g)*但是找不到n0使它满足*g∈O(f)* 因此，O不是对称关系。  3. 反例：f(x) = x g(x) = 2x  取n0 = 1 c1 = 1，当n > n0时f(n) ≤ g(n) 满足*f∈O(g)*, 取n0 = 1，c2 = 4，当n > n0时g(n) ≤ f(n) 满足*g∈O(f)* 但是f≠g 因此O不是反对称关系 |

## Lab2 轮廓线匹配实验

### 1. 实验要求

给出平面上有若干矩形，矩形的底部都与*x*轴重合，求整个图形的外轮廓，如下图所示。矩形的输入用三元组(*l, h, r*)表示；其中*l*为矩形左端坐标，*h*为矩形高度，*r*为右端坐标。输出用一组点构成的串表示，这些点是从左到右的过程中外轮廓高度发生变化时的转折点（如图中的绿点），按照点的横坐标升序排列。要求使用分治法求解，对输入序列按照二分法进行分解。设输入序列长为*n*，要求算法的时间复杂度满足 *work*=*O*(*n log n*)，*span*=*O*(*log2 n*)。



### 2. 实验思路

|  |
| --- |
| 利用divide and conquer求解分为divide和combine两个部分。divide的时候用showt函数分为空串，单元素的串，以及两个子串的情况。前两个情况作为base case，记录左右的边界及边界高度。两个子串的时候combine，根据两个子串生成两个只记录位置的串，然后利用copy\_scan对排好序的位置串和高度串分别copy高度，生成左右的copy串，再取两个串中高度高的坐标生成新串。然后对多余的进行filter去重，filter的时候比较第i个横坐标和第i+1个横坐标的高度，如果相等则去掉第i个横坐标的元素。最后得到一个(x,h)且没有多余坐标的串。 |

### 3. 回答问题

#### 3.1 提供代码和注释

Task 4.1 (50%).

Note: You may receive zero credit for task 4.1 if you do any of the following:

• Hard code to the public tests.

• Submit a sequential solution.

• Submit the reference solution or a similar solution as your own.

|  |
| --- |
| functor MkSkyline(structure S : SEQUENCE) : SKYLINE =  struct  structure Seq = S  open Primitives  open Seq  fun combine s1 s2= (\*w=n s=log n \*)  let  fun cmp ((l1,r1),(l2,r2)) = if l1 < l2 then LESS else if l1 = l2 then EQUAL else GREATER    (\*s3 s4 仅保存需要的位置\*)  val s3 = map (fn (x1,y1) => (x1,~1)) s1 (\*work O(n) span O(log n)\*)  val s4 = map (fn (x2,y2) => (x2,~1)) s2  (\*左右分别合并位置信息然后copy高度\*)  fun copy ((x3,y3),(x4,y4)) = if y4 = ~1 then (x4,y3) else (x4,y4)  (\*merge w=O(n) s=O(log n) scan w=O(n) s=O(logn)\*)  val building1 = scani copy (0,~1) (merge cmp s1 s4)  val building2 = scani copy (0,~1) (merge cmp s2 s3)  (\*选择更高的，并且一样高的去重\*)  fun maxheight ((x1,y1),(x2,y2)) = if y1 > y2 then (x1,y1) else (x2,y2)  (\*w=o(n) s=o(1)\*)  val rem1 = map2 maxheight building1 building2  fun judge (0,\_) = true  |judge (i,a:int\*int) = if (#2 a) <>(#2 (nth rem1 (i-1))) then true else false  in(\*去重\*)  (\*w=o(n) s=log n\*)  filterIdx judge rem1  end    fun skyline (buildings : (int \* int \* int) seq) : (int \* int) seq =  (\*w(n) = 2w(n/2) + o(n) s(n) = s(n/2) + o(log n) w = o(nlogn) s = o(log2n)\*)  if length buildings = 0 then empty()  else  case showt buildings of(\*w= lgn or n\*)  EMPTY => singleton((0,0))  |ELT (l,h,r) => fromList [(l,h),(r,0)]  |NODE (s1,s2) => combine (skyline s1) (skyline s2)  end |

#### 3.2 关于代码测试

Task 4.2 (10%). To aid with testing we have provided a testing structure, Tester, which should simplify the testing process. Tester will look at the ﬁle Tests.sml, in which you should put your test input. At submission time there must not be any testing code in the same ﬁle as your SKYLINE implementation, as it can make it difﬁcult for us to test your code. In order to test your code, after running CM.make, you will need to run Tester.testSkyline() to test your implementation.

|  |
| --- |
| 测试样例：    测试结果： |

#### 3.2 关于正确性和复杂度证明

Task 4.3 (30%).

Prove the correctness of your divide-and-conquer algorithm by induction. Be sure to carefully state the theorem that you’re proving and to note all the algorithm steps in your proof. You should use the following structural induction principle for abstract sequences:

Let *P* be a predicate on sequences. To prove that *P* holds for every sequence, it sufﬁces to show the following:

1. *P* (〈〉) holds,

2. For all *x*, *P*(〈*x*〉) holds, and

3. For all sequences *S1* and *S2*, if *P*(*S1*) and *P*(*S2*) hold, then *P*(*S1*@*S2* ) holds.

|  |
| --- |
| 证明：1.当buildings = <>时，showt之后得到EMPTY=>singleton(0,0) 正确  2.当buildings只有一个元素时，showt之后得到ELT (l,h,r) => <(l,h),(r,0)> 正确  3.当buildings 含有多个元素时，showt之后递归调用skyline直到1和2的情况再递归combine，在combine的过程中，将两个已经排好的串合并位置信息，取最大的高度并且去重，得到一个结果串。所以如果1和2的结果正确，那么combine就能得到一个正确的结果，因此当buildings含有多个元素时能得到正确的结果。 |

Task 4.4 (10%).

Carefully explain why your divide-and-conquer steps satisfy the speciﬁed recurrence and prove a closed-form solution to the recurrence.

|  |
| --- |
| 在combine的过程中，主要调用了map,scan,merge和filter四个函数，它们的work都是O(n)，span都是O(logn)，因此combine的work为O(n)，span为O(logn)。初始串在divide之后成为两个长度为n/2的串(或者base case),再并行递归计算，所以W (n) = 2W(n/2) + Wcombine 即W (n) = 2W(n/2) + O(n)，由tree method解得W(n) = O(nlogn)，S (n) = S(n/2) + Scombine 即S (n) = S(n/2) + O(logn)，递归代入解得) |

## Lab3 bignumlab

### 1. 实验要求

实现n位二进制大整数的加法运算。输入a, b和输出s都是二进制位的串。要求算法的时间复杂度满足work=O(n)，span=O(log n)。

### 2. 实验思路

|  |
| --- |
| 1)加法：给较短的数补零然后zip形成(x,y)串，用map得到产生(GEN)，传递(PROP)和停止(STOP)进位的位置，产生的进位会传递到停止的位置(即在这些位置上会出现进位1)，用copy\_scan找到这些位置并且转换为二进制01串，再对它和初始的串进行不考虑进位的加法得到结果。  2)减法：给较短的数补零，用取反加一的方法(补码)将减法转换为加法得到结果。用zip和tabulate得到一个位置和数字的二元组，用filter找到最后一个(高位)ONE的位置，再用take去掉结果后面的0。  3)乘法：给短的数补0得到(x,y)串，再求乘积。分成空串，单元素串和两个子串三种情况。前两个情况为base case，记录乘积。当有两个子串时，用*AB = pr·2n+(ps+rq)·2n/2+qs* 以及*ps* + *rq* = (*p* + *q*) *∗* (*r* + *s*) *- pr – qs*将大数乘积转换为三个小数的乘积的加减运算，其中在进行乘2n时采用移位的方法，给数的前面补n位0。 |

### 3. 回答问题

#### 3.1 提供加法计算的代码和注释

Task 4.1 (35%). Implement the addition function

++ : bignum \* bignum -> bignum

in the functor MkBigNumAdd in MkBigNumAdd.sml. For full credit, on input with *m* and *n* bits, your solution must have *O*(*m*+*n*) work and *O*(lg(*m*+*n*)) span. Our solution has under 40 lines with comments.

|  |
| --- |
| functor MkBigNumAdd(structure U : BIGNUM\_UTIL) : BIGNUM\_ADD =  struct  structure Util = U  open Util  open Seq  infix 6 ++  datatype carry = GEN | PROP | STOP  fun x ++ y =  let  (\*给短的序列补0\*)  val s = tabulate (fn x => ZERO) (Int.max (length x,length y)-Int.min (length x,length y)+ 1)(\*work O(n) span O(1)\*)  val x1 = append (x,s)(\*work O(n) span O(log n)\*)  val y1 = append (y,s)  val sum = zip x1 y1  (\*映射为carry序列\*)  fun find (ZERO,ZERO) = STOP  |find (ONE,ONE) = GEN  |find (\_,\_) = PROP  val sum1 = map find sum(\*work O(n) span O(log n)\*)  (\*找进位\*)  fun copy (a,PROP) = a  |copy (\_,GEN) = GEN  |copy (\_,STOP) = STOP  val sum2 = scani copy STOP sum1  val carry1 = append ((singleton STOP),sum2)(\*和比原数多一位 w=O(n) s=1\*)  fun bit\_add (ZERO,(ONE,ONE)) = ZERO  |bit\_add (ZERO,(ZERO,ZERO)) = ZERO  |bit\_add (ONE,(ONE,ZERO)) = ZERO  |bit\_add (ONE,(ZERO,ONE)) = ZERO  |bit\_add (\_,(\_,\_)) = ONE  (\*进位1 无进位0\*)  val num1 = map (fn x => if x = STOP then ZERO else ONE ) carry1 (\*work O(n) span O(log n)\*)  val result1 = map2 bit\_add num1 sum (\*work O(n) span O(log n)\*)  in  result1  end    val add = op++  end |

#### 3.2 提供减法计算的代码和注释

Task 4.2 (15%). Implement the subtraction function

-- : bignum \* bignum -> bignum

in the functor MkBigNumSubtract in MkBigNumSubtract.sml, where *x* -- *y* computes the number

obtained by subtracting *y* from *x*. We will assume that *x*≥*y*; that is, the resulting number will always be non-negative. You should also assume for this problem that ++ has been implemented correctly. For full credit, if *x* has *n* bits, your solution must have *O*(*n*) work and *O*(lg*n*) span. Our solution has fewer than 20 lines with comments.

|  |
| --- |
| functor MkBigNumSubtract(structure BNA : BIGNUM\_ADD) : BIGNUM\_SUBTRACT =  struct  structure Util = BNA.Util  open Util  open Seq  infix 6 ++ --  fun x ++ y = BNA.add (x, y)  fun x -- y =  let  (\*给短的序列y补0，xy符号位为0\*)  val y1 = if length x = length y then y else append (y,tabulate (fn x => ZERO) (length x-length y))  val neg\_y = map (fn x => if x = ZERO then ONE else ZERO) y1 (\*取反\*)  val sub = x ++ (neg\_y ++ singleton ONE)  val result1 = take (sub,length x)  (\*找到第一个,高位1\*)  val cancel = zip (tabulate (fn i => i) (length x)) result1  val find\_one = filter (fn (x,y) => if y = ONE then true else false) cancel  val (a,b) = nth find\_one (length find\_one-1)  in  take (result1,a + 1)  end    val sub = op--  end |

#### 3.3 提供乘法计算的代码和注释

Task 4.3 (30%). Implement the function

\*\* : bignum \* bignum -> bignum

in MkBigNumMultiply.sml. For full credit, if the larger number has n bits, your solution must satisfy *W∗∗*(*n*) = 3 · *W*∗∗(*n*/2)+ O(*n*) and have O (lg2*n* ) span. You should use the following function in the Primitives structure:

val par3 : (unit -> ’a) \* (unit -> ’b) \* (unit -> ’c) -> ’a \* ’b \* ’c

to indicate three-way parallelism in your implementation of \*\*. You should assume for this problem that ++ and -- have been implemented correctly, and meet their work and span requirements. Our solution has 40 lines with comments.

|  |
| --- |
| functor MkBigNumMultiply(structure BNA : BIGNUM\_ADD  structure BNS : BIGNUM\_SUBTRACT  sharing BNA.Util = BNS.Util) : BIGNUM\_MULTIPLY =  struct  structure Util = BNA.Util  open Util  open Seq  open Primitives  exception NotYetImplemented  infix 6 ++ --  infix 7 \*\*  fun x ++ y = BNA.add (x, y)  fun x -- y = BNS.sub (x, y)  fun bit\_mul (ONE,ONE) = ONE  |bit\_mul (\_,\_) = ZERO  (\*乘法\*)  fun multiply sum =  case showt sum of  EMPTY => empty()  |ELT (a,b) => singleton (bit\_mul (a,b))  |NODE (x,y) =>  let  val p = map (fn (a,b) => a) y  val q = map (fn (a,b) => a) x  val r = map (fn (a,b) => b) y  val s = map (fn (a,b) => b) x  val t1 = p++q  val t2 = r++s  val t3 = tabulate (fn x => ZERO) (Int.max (length t1,length t2)-Int.min (length t1,length t2))  val t = zip (append (t1,t3)) (append (t2,t3))  val (pr,pqrs,qs) = par3 (fn () => multiply y,fn () => multiply t,fn () => multiply x)  fun power (m,(n:int)) = append ((tabulate (fn i => ZERO) n),m)(\*给m前补n位0 变高位\*)  val k = length x  in  qs ++ (power (pr,2\*k)) ++ (power ((pqrs--pr--qs),k))  end  fun x \*\* y =  let  (\*给短的序列补0\*)  val s = tabulate (fn x => ZERO) (Int.max (length x,length y)-Int.min (length x,length y))  val x1 = append (x,s)  val y1 = append (y,s)  val sum = zip x1 y1  in  if length x = 0 orelse length y = 0 then empty()  else if length x = 0 then y  else if length y = 0 then x  else multiply sum  end    val mul = op\*\*  end |

#### 3.4 迭代计算复杂度分析

Task 5.1 (15%). Determine the complexity of the following recurrences. Give tight *Θ*-bounds, and

justify your steps to argue that your bound is correct. Recall that *f*∈*Θ*(*g*) if and only if *f*∈*O*(*g*) and *g*∈*O*(*f*). You may use any method (brick method, tree method, or substitution) to show that your bound is correct, except that you must use the substitution method for problem 3.

1. T (*n*) = 3T(*n*/2) + *Θ*(*n*)

2. T (*n*) = 2T(*n*/4) + *Θ*()

3. T (*n*) = 4T(*n*/4) +*Θ*() (Prove by substitution.)

|  |
| --- |
| 1.tree method:递归树高为，第i层有个结点，每个结点的T为，第i层的T为 ，总的T为  2.tree method:递归树高为，第i层有个结点，每个结点的T为，第i层的T为 总的T为  3.substitute method：当n > 1时T (*n*) ≤ 4T(*n*/4) +k·且当n ≤ 1时T(n) ≤ k，假设存在常数k1和k2使得T(n) ≤ k1·n + k2  证明：n = 1时，令k1 = 4k，k2 = k，T(1) ≤ k ≤ k1 + k2成立  假设T(n/4) ≤ k1·n/4 + k2  则 T(n) ≤ 4T(n/4) + k·  ≤ 4(k1·n/4 + k2) + k·  = k1n + 4k2 + k·  ≤ 4k1n + k2 |