

**课 程 实 验 报 告**

**课程名称： 串并行程序设计实验**

**专业班级： ACM1601**

**学 号： U201614787**

**姓 名： 宁嘉**

**指导教师： 陆枫**

**报告日期： 　 2018/1/24**

**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

|  |
| --- |
| 先用一个函数去判断括弧是否匹配（从左往右扫描即可），然后对每一个左括号，都做这样一个处理，去找到与它匹配的右括号在哪（距离它多少个括号），然后取出距离的最大值即可。 |

#### 2.2 分治法求解思路

|  |
| --- |
| 将括弧不断平均分成左右两部分，其中merge操作只需将左右已经得到的最大距离与crossing的最大距离进行一个比较便可以得到新的最大距离。但这中间有一个trick的操作，便是通过保留适当的数据项使得我们在merge的时候只要O(1)的Work and Span便可以完成。 |

### 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).

|  |
| --- |
| 1. functor MkBruteForcePD (structure P : PAREN\_PACKAGE) : PAREN\_DIST = 2. struct 3. structure P = P 4. open P 5. open Seq 6. fun parenDist (parens : paren seq) : int option = 7. let 8. fun parenMatch a=(\*判断是否匹配\*) 9. let 10. fun count (s,x) = case (s,x) of 11. (NONE,\_)=>NONE 12. |(SOME n,CPAREN)=>if (n=0) then NONE else SOME(n-1) 13. |(SOME n,OPAREN)=>SOME(n+1) 14. in 15. iter count (SOME 0) a=SOME 0 16. end 17. fun count\_match (n:int ,s:paren seq)=(\*n表示第n个括号,且nth为左括号\*) 18. let fun helper(add:int,flag)= 19. if flag=0 then add-2 20. else case nth s (n+add) of 21. CPAREN =>helper(add+1,flag-1) 22. |OPAREN =>helper(add+1,flag+1) 23. in helper(1,1) 24. end 25. fun max\_match(s)= 26. let fun helper(n,s:paren seq,max)= 27. if n<length s then 28. case nth s n of 29. CPAREN=>helper(n+1,s,max) 30. |OPAREN=> 31. let val newmax=(fn(a,b)=>if a>b then a else b )(count\_match(n,s),max) 32. in helper(n+1,s,newmax) 33. end 34. else max 35. in helper(0,s,0) 36. end 37. in 38. if length parens =0 then NONE 39. else 40. case parenMatch parens of 41. false => NONE 42. |true => SOME (max\_match(parens)) 43. end 45. 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.

|  |
| --- |
| **复杂度分析：**   * Work 的分析：（此处分析最坏的情况，即括弧完全匹配的情况）  1. parenMatch所用的work为O(n) 2. #20行count\_match函数 的work由递归的次数决定，即为该左括号到与其匹配的右括号的距离，其为O(n)。 在#35该函数遇到左括号就调用一次，共调用了O(n)次，所以调用该函数总的work为O(n2) 3. #29行调用的max\_match函数，对括号索引从左往右扫描，遇到左括号才调用count\_match函数，并更新最大值，所以用的work为O(n2) 4. 最后=>parenDist函数调用一次prenMatch函数与max\_match函数（如果匹配成功的话），所用work为O(n2)，即总work为O(n2)  * Span的分析：（此处分析最坏的情况，即括弧完全匹配的情况）   由于设计的算法是基本串行的算法，所以其span仍然是O(n2) |

#### 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.

|  |
| --- |
| * **Part 1 程序代码**  1. functor MkDivideAndConquerPD (structure P : PAREN\_PACKAGE) : PAREN\_DIST = 2. struct 3. structure P = P 4. open Primitives 5. open P 6. open Seq 7. fun parenDist (parens : paren seq) : int option = if length parens = 0 then NONE else 8. let 9. val (max,lnot\_match\_num,llocal\_num,rnot\_match\_num,rlocal\_num)= 10. let 11. fun helper(C) = 12. case showt C of 13. EMPTY =>(0,0,0,0,0) 14. |ELT n => if n=OPAREN then (0,1,0,0,0) 15. else (0,0,0,1,0) 16. |NODE(A,B) => 17. let 18. val (a\_max,a\_lnot\_match\_num,a\_llocal\_num,a\_rnot\_match\_num,a\_rlocal\_num)=helper(A) 19. val (b\_max,b\_lnot\_match\_num,b\_llocal\_num,b\_rnot\_match\_num,b\_rlocal\_num)=helper(B) 20. in if a\_lnot\_match\_num = b\_rnot\_match\_num then (Int.max((a\_llocal\_num+b\_rlocal\_num),Int.max(a\_max,b\_max)),b\_lnot\_match\_num,b\_llocal\_num,a\_rnot\_match\_num,a\_rlocal\_num) 21. else if a\_lnot\_match\_num < b\_rnot\_match\_num then (Int.max(a\_max,b\_max),b\_lnot\_match\_num,b\_llocal\_num,b\_rnot\_match\_num - a\_lnot\_match\_num + a\_rnot\_match\_num,b\_rlocal\_num+length A) 22. else (Int.max(a\_max,b\_max),a\_lnot\_match\_num - b\_rnot\_match\_num + b\_lnot\_match\_num,a\_llocal\_num +length B,a\_rnot\_match\_num,a\_rlocal\_num) 23. end 24. in helper(parens) 25. end 26. in if lnot\_match\_num = 0 andalso rnot\_match\_num = 0 then SOME(max) 27. else NONE 28. end 29. end  * **Part 2 算法正确性证明**   在parenDist函数中间调用helper函数进行分治的递归：helper函数传递的参数为括号组成的sequence，其返回值为(max,lnot\_match\_num,llocal\_num,rnot\_match\_num,rlocal\_num)，其中max为sequence中**潜在的[[1]](#footnote-1)**最大的匹配括号之间的距离，lnot\_match为未匹配的左括号的数目，rnot\_match为未匹配的右括号的数目，llocal\_num为未匹配的最左的左括号距离右端的距离，rlocal\_num为未匹配的最右的右括号距离左端的距离。下面我们将证明算法递归的每一层这五个属性量都成立。   * + **递归基础：**  1. 当sequence中无元素的时候，所有指标均为0 2. 当sequence中只有左括号的时候，除了lnot\_match\_num为1外均为0 3. 当sequence中只有右括号的时候，除了rnot\_match\_num为1外均为0    * **递归步骤：**将C平均地分为A,B两串(A串位于B串左边)，由归纳假设（对sequence长度进行归纳）helper(A)与helper(B)的返回值均满足我们的归纳结论，下面证明helper(C)的返回值也满足归纳结论。分如下三种情况讨论： 4. A串没有匹配的左括号数量 < B串没有匹配的右括号数量：那么这时A串中没有匹配的左括号都将被B串中没有匹配的右括号匹配，所以C串没有匹配的左括号就是原来B串中没有匹配的左括号，既有关系C中lnot\_match\_num=b\_lnot\_matchnum，B串中有a\_lnot\_match\_num个右括号被匹配掉了，于是新的C串中未匹配的右括号数目rnot\_match\_num即为b\_rnot\_match\_num+a\_rnot\_match\_num-a\_lnot\_match\_num。对于C串中未匹配的最左的左括号的距离最右端的距离就B串中未匹配的最左的左括号距离最右端的距离，C串中未匹配的最右的右括号就是B串中未匹配的右括号的位置，于是C串中未匹配的最右的右括号距离最左端的距离即为b\_rlocal\_num+ length A。最后讨论潜在的最大括号匹配长度，这时最右端留存有括号未得到匹配，于是新产生的匹配的括号中的距离一定会小于最右端留存的括号与其在最终sequence中对应匹配的左括号之间的距离，所以潜在最长括号的长度不需要记入crossing A B串的括号，我们只要取定为A串中的max与B串中的max的最大值。 5. A串没有匹配的左括号数量 > B串没有匹配的右括号数量：分析同上。 6. A串没有匹配的左括号数量 = B串没有匹配的右括号数量：此时与之前两种情况不同的地方在于我们需要更新max项的值，因为这时A串中最左端的括号与B串中最右端的括号正好匹配，这时潜在的最大匹配距离需要加上该匹配的距离取定最大值。   这样我们得到了新合成的C串也使得helper(C)返回值满足我们的归纳的结论。  有了这些后，便可以继续算法正确性的证明，剩下的步骤是显然的：第一，括号完全匹配等价于未匹配的左括号与右括号数皆为0；第二，如果最终sequence是匹配的，我们得到的max项即为所求。 |

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.

|  |
| --- |
| Solve  1. 易知归并处理过程中的Work为O(1)  W(n) = 2W(n/2) + *Θ(lg n)+O(1)*  因为n->∞时2lg(n/2)/lg n->2，所以这是一个leaf domain tree，而induction tree最后一层共有*Θ(n)*项，每项都是*Θ(1)*，所以W(n) = *O(n)*  2. 易知归并处理过程中的Work为O(1)  W(n) = 2W(n/2) + *Θ(n)+O(1)*  由主定理知W(n) = *O(nlg n)*  3.tree sequence  4.list sequence |

#### 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：val tests = [  "",  "()",  "()()",  "(",  ")(",  "(()))",  "((()))()",  "))((",  "()()(())",  "((((()))))",  "()()()",  "(((()()()())))()",  "((())",  "()((())()()()))()())",  "()()()()()",  ")())))(()(()(()",  "()()((()))()((((()))))((()))()(()()()())",  "(()))(",  "()()()(((((((()))",  "(()((()))()(()))",  "()(())()((",  ")))(())()(",  "()(())()(()))(())()",  "()()()()()()()()()(",  "()()()()()()()()()(()(())()(()))(())()",  "()())(()()(()()((((()))))((()()",  "((((((()))))))"  ]  BF and DC 都已pass. |

#### 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!*

|  |
| --- |
| 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)* .

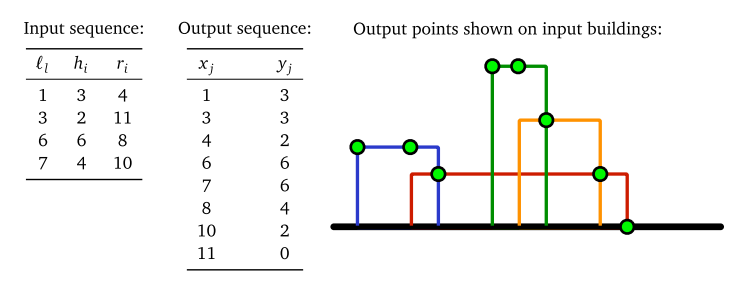
3. *O* is an anti-symmetric relation on functions. That is to say, for any functions *f* and *g* , if *f∈O(g)* and *g∈O(f)*, then *f*=*g*.

|  |
| --- |
| Statement 1 是正确的，而2 and 3是错误的。   * + 1. 如果 *f∈O(g)* and *g∈O(h)*, 那么存在 N>0,x0使得x>x0时，|f|<=N|g|，存在M>0,x1使得x>x1时,|g|<=M|h|，于是当x>max{x0,x1}时|f|<=NM|h|，也即*f∈O(h)*.     2. counterexample:f(x) = 1 ,g(x) = x     3. counterexample:f(x) = x , g(x) = x+1 |

## 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

|  |
| --- |
| 按楼房数量平均分来进行分治，对两个已经得到结果的楼房，去merge出新的结果序列，其中利用copy scan去查找每个点在另外一个集合中离得最近的前一个点，利用其前一个去判断该点是否要变化（上升到边界），最后去重。详细说明可见算法正确性证明。 |

### 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.

|  |
| --- |
| 1. functor MkSkyline(structure S : SEQUENCE) : SKYLINE = 2. struct 3. structure Seq = S 4. open Primitives 5. open Seq 6. datatype side = left | right 7. fun skyline (buildings : (int \* int \* int) seq) : (int \* int) seq = 8. case showt buildings of 9. EMPTY => empty() 10. |ELT (l,h,r) => %[(l,h),(r,0)] 11. |NODE (lseq,rseq) => 12. let val lresult = map (fn(x,y)=>(x,y,left)) (skyline lseq) 13. val rresult = map (fn(x,y)=>(x,y,right)) (skyline rseq) 14. val xys = merge (fn((x,\_,\_),(y,\_,\_))=>if x>y then GREATER else if x=y then EQUAL else LESS) lresult rresult 15. (\* 通过copyscan找到前边最接近某个点的另一个集合的点 16. copy (x,self) = x |copy (x,y) = y ，当scan/reduce到第i个点时，我们便找到了这样的点\*) 17. fun copy\_fits(self) = fn(x,(a,b,y)) => if y=self then x else (a,b,y) (\*单位元为???没有，但是没影响！！！\*) 18. val front\_points\_right = scani (copy\_fits right) (0,0,right) xys 19. val front\_points\_left = scani (copy\_fits left) (0,0,left) xys 20. (\*利用这些点去产生新的点\*) 21. fun pairmax((a,b),(a',b'))=(Int.max(a,a'),Int.max(b,b')) 22. fun produce (i,point) = case point of (x,y,right) => pairmax((fn(a,b,\_)=>(a,b))(nth front\_points\_right i),(x,y)) 23. |(x,y,left)=>pairmax((fn(a,b,\_)=>(a,b))(nth front\_points\_left i),(x,y)) 24. val result' = mapIdx produce xys 25. in filterIdx (fn (i,(\_,y)) => i=0 orelse (#2 (nth result' (i-1)) <> y)) result' 26. end 27. 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.

|  |
| --- |
| 1. [(1,1,2)], 2. [(1,1,3),(2,1,4)], 3. [(4,5,20),(1,3,5),(2,4,6),(8,7,11),(12,11,13),(10,10,14),(17,2,21)], 4. [(20,10000,90000)], 5. [(1,3,4),(3,2,11),(7,4,10),(6,6,8)], 6. [(1,1,2)], 7. [(1,1,3),(2,1,4)], 8. [(4,5,20),(1,3,5),(2,4,6),(8,7,11),(12,11,13),(10,10,14),(17,2,21)], 9. [(20,10000,90000)], 10. [(183, 63, 362), (95, 179, 191), (76, 218, 251), (104, 133, 235), (247, 255, 422), (119, 11, 174), (233, 103, 396), (121, 67, 171), (241, 192, 276), (185, 38, 189), (232, 239, 287), (250, 206, 297), (225, 250, 242), (211, 235, 275), (86, 2, 257), (88, 70, 289), (203, 212, 327), (109, 77, 173), (161, 256, 366), (88, 213, 139), (6, 120, 87), (107, 152, 190), (75, 23, 84), (232, 26, 296), (9, 178, 164), (97, 250, 128), (211, 122, 386), (68, 214, 164), (116, 207, 347), (87, 79, 148), (224, 226, 358), (77, 213, 295), (97, 180, 341), (190, 49, 208), (175, 74, 320), (37, 30, 112), (236, 60, 394), (177, 64, 416), (191, 230, 224), (14, 93, 108), (256, 215, 268), (112, 173, 220), (186, 100, 213), (132, 143, 337), (100, 96, 124), (25, 12, 26), (165, 102, 303), (152, 239, 303), (4, 62, 258), (167, 91, 396), (74, 62, 177), (142, 33, 272), (132, 189, 204), (48, 77, 90), (229, 235, 396), (186, 156, 398), (2, 173, 251), (90, 219, 115), (14, 84, 147), (151, 93, 228), (42, 60, 290), (30, 13, 233), (230, 195, 247), (40, 201, 143), (127, 16, 317), (193, 4, 201), (133, 163, 382), (147, 226, 355), (231, 168, 297), (15, 75, 114), (162, 125, 312), (251, 173, 290), (120, 244, 156), (216, 94, 366), (131, 48, 189), (157, 226, 403), (130, 50, 155), (175, 123, 201), (39, 112, 222), (46, 213, 286)], 11. []] |

#### 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.

|  |
| --- |
| 首先将问题化归为找到外围轮廓线同一高度线的最左边的端点，此即为要找的所有点,接下来用数学归纳法进行证明skyline函数返回的值是正确的。   * + **递归基础：**  1. 当sequence为空的时候，返回空串。 2. 当sequence中只有一个元素的时候返回左上角点和右下角点    * **递归步骤：**将 building sequence 平均地分为两个sequence，记为lseq和rseq，由归纳假设skyline(lseq)和skyline(rseq)皆返回了满足要求的序列，下面证明该skyline(building sequence)也满足要求。我们有如下结论： 3. xys是按将lseq与rseq 得到的点按x坐标排序合并得到的，其中x（第一个）坐标代表题目中的横坐标，y（第二个）坐标代表题目中的高度，s（第三个）坐标代表其属于showt中哪个子树，其中左子树为left，右子树为right。 4. copy\_fits函数：copy\_fits函数接受一个参数self，返回一个函数，由scani的具体**实现**可以知道scani (copy\_fits self) (0,0,self) xys(\*self为left or right\*)得到的sequence中的第i项为xys :sequence中0-i中项不属于self集合的最后一个元素（若不存在不属于self集合的元素，则返回(0,0,self)）。 5. 在2)中取self = right和self = left便得到了满足这样性质的两个序列front\_points\_right(前i个点中最后一个left集合的点组成的sequence或者说xys序列中第i个点前面的第一个left集合的点构成的sequence)和front\_points\_left(前i个点中最后一个right集合的点组成的sequence或者说xys序列中第i个点前面的第一个right集合的点组成的sequece)。现在可以利用xys，front\_points\_right, front\_points\_left这三个集合去生成我们最后想要得到的点。produce函数将点及其index作为参数，如果这个点属于right集合，就去front\_points\_right :sequence中查找出其前面的那个属于left集合的点(若没有，则查找到的点为(0,0,right)，若这个点属于left集合同理去查找另外一个集合对应点即可。   **准备阶段**至此结束，下面证明我们生成的新的点即为所求。  事实上，函数mapIdx produce xys 生成了这样一系列点，如果这个点没有“被覆盖”（即仍属于新的外轮廓边界），则将该点保留；如果这个点“被覆盖”（即不属于新的外轮廓边界），则将该点往上移动至新的外轮廓边界处。（其中判断有无覆盖以及移动到的新边界的位置只需要由之前准备阶段得到的每个点在另外一个集合中的前一个点的高度进行比较即可）。将这样得到的点的sequence记为result’。一方面，新的外轮廓中同高度轮廓线左端点（即为需要得到的点）一定属于result’中，因为该点所在角处的轮廓线就两种情况，一种是它比左边外轮廓线上的点高，这种点一定属于一个集合中的点高于左边集合中的点而没有变的情况；第二种是它比左边外轮廓线上的点低，这时易验证该点也属于集合result’。另一方面，其它产生的点都在外轮廓线上，中间高度重叠而不要的点filter掉，剩下的点正是我们要求的点。  综上知归纳结论成立，亦及算法正确。 |

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.

|  |
| --- |
| Wshowt = O(1) and Sshowt = O(1)  Wskyline(n) = 2Wskyline(n/2) + Wshowt + Wcombine(n)  Sskyline(n) = 2Sskyline(n/2) + Sshowt + Scombine(n)  其中计算Wcombine与Scombine包含如下过程：   1. 两次map函数，work 为 O(n)，span 为 O(1) 2. merge函数work为O(n)，span为O(log n) 3. 两次scani，work为O(n)，span为O(log n) 4. mapIdx函数，事实上为一次tabulate，work为O(n)，span为O(1) 5. filter函数，work为O(n)，span为O(1)   综上知  Wskyline(n) = 2Wskyline(n/2) + O(n)  Sskyline(n) = 2Sskyline(n/2) + O(log n)  容易解得Wskyline = O(n log n)  Sskyline = O(log2n) |

## 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

|  |
| --- |
| 先把两个串的长度变成相同，且往高位补一个零，方便后续进位时候用。将两个串逐位相加，考虑某一位什么时候会得到进位。我们可以把进位看成一列波，先是从两个1开始产生了进位，然后如果中途仍是两个1(GEN)或者一个1一个0(PROP)，则该进位波会继续往后传递，直到遇到了两个0停止(STOP)（注意加法中进位的量不可能超过1）。所以对于判断一位有没有得到进位只需要判断它前面到底是GEN还是STOP离它近，这用与之前类似的copyscan即可跳过PROP，实现该效果。 |

### 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.

|  |
| --- |
| 1. functor MkBigNumAdd(structure U : BIGNUM\_UTIL) : BIGNUM\_ADD = 2. struct 3. structure Util = U 4. open Util 5. open Seq 6. infix 6 ++ 7. datatype carry = GEN | PROP | STOP 8. fun x ++ y = 9. let 10. val lx = length x 11. val ly = length y 12. fun appone (x,y) = append(x,singleton y) 13. val (u,v) = if lx > ly then (appone(x,ZERO),appone(tabulate (fn i => if i < ly then nth y i else ZERO) lx,ZERO)) 14. else (appone(y,ZERO),appone(tabulate (fn i => if i < lx then nth x i else ZERO) ly,ZERO)) (\*长度变为相同\*) 15. fun judge(x,y) = case (x,y) of 16. (ZERO,ZERO)=>STOP 17. |(ZERO,ONE)=>PROP 18. |(ONE,ZERO)=>PROP 19. |(ONE,ONE)=>GEN 20. val preres = map2 judge u v(\*供查找序列\*) 21. fun copy (x,PROP) = x 22. |copy (\_,y) = y 23. val (a,last) = scan copy PROP preres 24. fun interact (x,y) = case (x,y) of 25. (PROP,PROP)=>ONE | (PROP,\_)=>ZERO 26. |(GEN,PROP)=>ZERO | (GEN,\_)=>ONE 27. |(STOP,PROP)=>ONE | (STOP,\_)=>ZERO 28. val result = map2 interact a preres 29. fun deal s =(\*处理加法或减法运算或分治时拆分产生的前导零\*) 30. let val idxs' = enum s 31. val idxs = filter (fn (\_,x)=>x = ONE) idxs' 32. in if length idxs = 0 then empty() 33. else take (s,(#1(nth idxs (length idxs - 1))+1)) 34. end 35. in deal result 36. end  39. val add = op++ 40. 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.

|  |
| --- |
| 1. functor MkBigNumSubtract(structure BNA : BIGNUM\_ADD) : BIGNUM\_SUBTRACT = 2. struct 3. structure Util = BNA.Util 4. open Util 5. open Seq 6. infix 6 ++ -- 7. fun x ++ y = BNA.add (x, y) 8. fun x -- y = 9. let 10. val ly = length y 11. fun v ONE = ZERO 12. |v ZERO = ONE 13. val res' = (map v y)++singleton(ONE)++x 14. val restocount = enum res' 15. val temp = filter (fn(i,x)=>i>ly-1 andalso x = ONE) restocount 16. val res'' = inject (singleton(#1(nth temp 0),ZERO)) res' 17. val res = inject (tabulate (fn i=>(ly+i,ONE)) (#1(nth temp 0)-ly)) res'' 18. fun deal s =(\*处理加法或减法运算或分治时拆分产生的前导零\*) 19. let val idxs' = enum s 20. val idxs = filter (fn (\_,x)=>x = ONE) idxs' 21. in if length idxs = 0 then empty() 22. else take (s,(#1(nth idxs (length idxs - 1))+1)) 23. end 24. in deal res 25. end 27. val sub = op-- 28. 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.

|  |
| --- |
| 1. functor MkBigNumMultiply(structure BNA : BIGNUM\_ADD 2. structure BNS : BIGNUM\_SUBTRACT 3. sharing BNA.Util = BNS.Util) : BIGNUM\_MULTIPLY = 4. struct 5. structure Util = BNA.Util 6. open Util 7. open Seq 8. open Primitives 9. exception NotYetImplemented 10. infix 6 ++ -- 11. infix 7 \*\* 12. fun x ++ y = BNA.add (x, y) 13. fun x -- y = BNS.sub (x, y) 14. fun x \*\* y = 15. if length x = 0 orelse length y = 0 then empty() 16. else if length x = 1 andalso nth x 0= ONE then y 17. else if length y = 1 andalso nth y 0= ONE then x(\*induction base\*) 18. (\* (2^n \* a + b) \* (2^n \* c + d) = (2^2n - 2^n) \* ac + 2^n \* (a+b)(c+d) - (2^n - 1) \* bd \*) 19. else 20. let 21. val lx = length x 22. val ly = length y 23. val (u,v) = if lx > ly then (x,tabulate (fn i => if i < ly then nth y i else ZERO) lx) 24. else (y,tabulate (fn i => if i < lx then nth x i else ZERO) ly) (\*长度变为相同\*) 25. val hn = (length u) div 2 26. (\*divide\*) 27. fun deal s =(\*处理加法或减法运算或分治时拆分产生的前导零\*) 28. let val idxs' = enum s 29. val idxs = filter (fn (\_,x)=>x = ONE) idxs' 30. in if length idxs = 0 then empty() 31. else take (s,(#1(nth idxs (length idxs - 1))+1)) 32. end 33. val a = deal (drop(u,hn)) 34. val b = deal (take(u,hn)) 35. val c = deal (drop(v,hn)) 36. val d = deal (take(v,hn)) 37. fun power2 n = tabulate (fn x=>ZERO) n 38. (\*val ac = a\*\*c 39. val bd = b\*\*d 40. val s = (a++b)\*\*(c++d)\*) 41. val (ac,bd,s) = par3(fn()=>a\*\*c,fn()=>b\*\*d,fn()=>(a++b)\*\*(c++d)) 42. in append (power2 (2\*hn),ac) -- append (power2 hn,ac) ++ append(power2 hn,s) -- append(power2 hn,bd) ++ bd 43. end 45. val mul = op\*\* 46. 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. 由主定理T(n) =*Θ*(*n3/2*) 2. 由主定理T(n) =*Θ*(*n1/2log n*) 3. substitution method：   T(n) ≤ 4T(n/4) + cn1/2 （k为常数)  下面用数学归纳法证明T(n) < m\*n - cn1/2  不失一般性，可设T(n) < m\*n - cn1/2对base case成立(m足够大)  设T(n) < m\*n- cn1/2对n<k成立  下证T(n) < m\*n- cn1/2对n = k也成立  T(n) ≤ 4T(n/4) - cn1/2 < 4(m\*n/4- c(n/4)1/2) + cn1/2  =m\*n – cn1/2  所以结论成立  所以得到T(n) ∈*O*(n)成立  同理可以证明T(n) ∈Ω(n)成立  综合可知T(n) =*Θ(n)* |

1. 所谓潜在的是指有可能成为最后合并的整个sequence中的有最大的distance的匹配的括号，我们将在之后的证明中看到这样规定是合法的，并且这样规定能大大降低我们的复杂度，因为我们关心的只是最终串中的maximum distance. [↑](#footnote-ref-1)