華中科技大学

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 $max\{j-i-1 \mid (si, sj)$ 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
```

```
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 $O(log^2 n)$ span where *n* is the length of the sequence.

Work = $O(n^4)$ Span = $O(n^2 lgn)$

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
```

```
functor MkDivideAndConquerPD (structure P : PAREN_PACKAGE) : PAREN_DIST =
struct
  structure P = P
  open Primitives
  open P
  open Seq

  (* Remove this line when you're done. *)
  exception NotYetImplemented
(*  左边的剩余,左边距离,右边的剩余,右边距离,最大值
    如果左边的（和右边的）相等 匹配
WparenDist(n) = 2 · WparenDist 2n + Wshowt(n) + O(1)
SparenDist(n) = SparenDist 2n + Sshowt(n) + O(1)*)
  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 $Wshowt \in \Theta(lg\ n)$ where n is the length of the input sequence.

2. Solve the work recurrence with the assumption that $Wshowt \in \Theta(n)$ 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 $\Theta(lg\ n)$ work.

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

$WparenDist(n) = 2 \cdot WparenDist(n/2) + Wshowt(n) + O(1)$

1. 当 $Wshowt \in \Theta(lg\ n)$时，$W(n) = 2W(n/2)+O(lgn)$ 则 W(n) = O(lgn)
2. 当 $Wshowt \in \Theta\ (n)$时，$W(n) = 2W(n/2)+O(n)$ 则 W(n) = O(nlgn)
3. Tree sequence 可以用 $\Theta(lg\ n)$的 work 实现 showt
4. List sequence 可以用 $\Theta(n)$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：

```
structure Tests =
struct
  (* Add your test cases to this list: *)
  val tests = [
    "()",
    "()()",
    "(())",
    "((()))",
    ")(((((())))",
    "(((()())))",
    "()()))(((",
    "))(()()()())",
    "(",
    ")",
    "()()(((())))",
    "()((()))()()))",
    "",
    "(()())((()))()(()))()()(((()))))"
  ]
end
```

测试结果：

1) Brute-force:

```
val it = true : bool
- Tester.testBF ();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
-
```

2) Divide-and-conquer:

```
val it = true : bool
- Tester.testDC();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
```

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. $f(n) = n^{log(n^2)}$
2. $f(n) = 2n^{1.5}$
3. $f(n) = (n^n)!$
4. $f(n) = 43^n$
5. $f(n) = lg(lg(lg(lg(n))))$
6. $f(n) = 36n^{52} + 15n^{18} + n^2$
7. $f(n) = n^{n!}$

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 $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$.
2. *O* is a symmetric relation on functions. That is to say, for any functions *f* and *g* , if $f \in O(g)$, then $g \in O(f)$.
3. *O* is an anti-symmetric relation on functions. That is to say, for any functions *f* and *g* , if $f \in O(g)$ and $g \in O(f)$, then $f=g$.

证明：

1.∵$f \in O(g)$∴ 存在正常数 n1,c1 使得当 n > n1 时，f(n) ≤ c1 g(n)

∵$g \in O(h)$∴ 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，$f \in O(h)$

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 $f \in O(g)$但是找不到 n0 使它满足 $g \in O(f)$ 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 $f \in O(g)$, 取 n0 = 1，c2 = 4，当 n > n0 时 g(n) ≤ f(n) 满足 $g \in O(f)$ 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 $work=O(n\ log\ n)$，$span=O(log^2\ n)$。

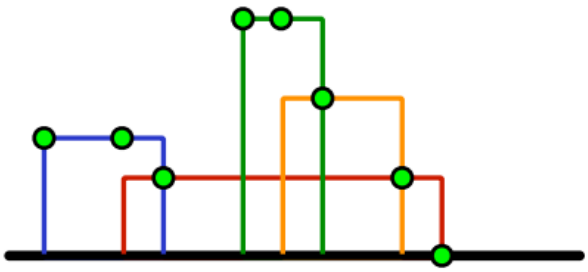
Input sequence:

ℓ_l	h_i	r_i
1	3	4
3	2	11
6	6	8
7	4	10

Output sequence:

x_j	y_j
1	3
3	3
4	2
6	6
7	6
8	4
10	2
11	0

Output points shown on input buildings:



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
```

```
functor MkSkyline(structure S : SEQUENCE) : SKYLINE =
struct
  structure Seq = S
  open Primitives
  open Seq

  exception NotYetImplemented


  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：

```
structure Tests =
struct

  val tests = List.map ArraySequence.% [
    [(1,1,2)],
    [(1,1,3),(2,1,4)],
    [(4,5,20),(1,3,5),(2,4,6),(8,7,11),(12,11,13),(10,10,14),(17,2,21)],
    [(20,10000,90000)],
    [(1,11,5), (2,6,7), (3,13,9), (12,7,16), (14,3,25),(19,18,22), (23,13,29), (24,4,28)],
    [],
    [(2,9,10),(3,7,15),(5,9,12),(16,7,24),(19,8,27)],
    [(1,1,5),(6,4,10),(3,3,9),(4,6,8),(2,2,7)],
    [(1,1,5),(3,3,9),(6,4,10),(4,6,8),(2,2,7)]

  ]

end

```

测试结果：

```
- Tester.testSkyline();
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
Test passed.
val it = () : unit
-
```

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 ($\langle\rangle$) holds,
2. For all x, $P(\langle x\rangle)$ holds, and
3. For all sequences S_1 and S_2, if $P(S_1)$ and $P(S_2)$ hold, then $P(S_1@S_2)$ 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) + $W_{combine}$ 即 W (n) = 2W(n/2) + O(n)，由 tree method 解得 W(n) = O(nlogn)，S (n) = S(n/2) + $S_{combine}$ 即 S (n) = S(n/2) + O(logn)，递归代入解得$S(n) = O(\log^2 n)$

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 $AB = pr \cdot 2^n + (ps+rq) \cdot 2^{n/2} + qs$ 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 2^n 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
```

```
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)
        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carry*)
        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 $x \geqslant y$; 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
```

```
1  functor MkBigNumSubtract(structure BNA : BIGNUM_ADD) : BIGNUM_SUBTRACT =
2  struct
3    structure Util = BNA.Util
4    open Util
5    open Seq
6
7    exception NotYetImplemented
8    infix 6 ++ --
9    fun x ++ y = BNA.add (x, y)
10   fun x -- y =
11       let
12         (*给短的序列y补0, xy符号位为0*)
13         val y1 = if length x = length y then y else append (y,tabulate (fn x => ZERO) (length x-length y))
14         val neg_y = map (fn x => if x = ZERO then ONE else ZERO) y1 (*取反*)
15         val sub = x ++ (neg_y ++ singleton ONE)
16         val result1 = take (sub,length x)
17         (*找到第一个,高位1*)
18         val cancel = zip (tabulate (fn i => i) (length x)) result1
19         val find_one = filter (fn (x,y) => if y = ONE then true else false) cancel
20         val (a,b) = nth find_one (length find_one-1)
21       in
22         take (result1,a + 1)
23       end
24
25   val sub = op--
26 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 $W_{**}(n) = 3 \cdot W_{**}(n/2) + O(n)$ and have $O(\lg^2 n)$ 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
```

```
10
11      infix 6 ++ --
12      infix 7 **
13
14      fun x ++ y = BNA.add (x, y)
15      fun x -- y = BNS.sub (x, y)
16
17      fun bit_mul (ONE,ONE) = ONE
18            |bit_mul (_,_) = ZERO
19      (*乘法*)
20      fun multiply sum =
21          case showt sum of
22              EMPTY => empty()
23              |ELT (a,b) => singleton (bit_mul (a,b))
24              |NODE (x,y) =>
25               let
26                  val p = map (fn (a,b) => a) y
27                  val q = map (fn (a,b) => a) x
28                  val r = map (fn (a,b) => b) y
29                  val s = map (fn (a,b) => b) x
30                  val t1 = p++q
31                  val t2 = r++s
32                  val t3 = tabulate (fn x => ZERO) (Int.max (length t1,length t2)-Int.min (length t1,length t2))
33                  val t = zip (append (t1,t3)) (append (t2,t3))
34                  val (pr,pqrs,qs) = par3 (fn () => multiply y,fn () => multiply t,fn () => multiply x)
35                  fun power (m,(n:int)) = append ((tabulate (fn i => ZERO) n),m)(*给m前补n位0 变高位*)
36                  val k = length x
37              in
38                  qs ++ (power (pr,2*k)) ++ (power ((pqrs--pr--qs),k))
39              end
40
41      fun x ** y =
42        let
43          (*给短的序列补0*)
44            val s = tabulate (fn x => ZERO) (Int.max (length x,length y)-Int.min (length x,length y))
45            val x1 = append (x,s)
46            val y1 = append (y,s)
47            val sum = zip x1 y1
48        in
49            if length x = 0 orelse length y = 0 then empty()
50            else if length x = 0 then y
51            else if length y = 0 then x
52            else  multiply sum
53        end
54
55      val mul = op**
56    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 $f \in \Theta(g)$ if and only if $f \in O(g)$ and $g \in O(f)$. 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) + $\Theta(n)$
2. T (n) = 2T(n/4) + $\Theta(\sqrt{n})$
3. T (n) = 4T(n/4) +$\Theta(\sqrt{n})$ (Prove by substitution.)

1.tree method:递归树高为$log_3 n + 1$，第 i 层有3^i个结点，每个结点的 T 为$\frac{n}{2^i}$，

第 i 层的 T 为 $n\left(\frac{3}{2}\right)^i$ ，总的 T 为$\sum_{i=1}^{log_3 n} n\left(\frac{3}{2}\right)^i = O(n^{log_2 3})$

2.tree method:递归树高为$log_2 n + 1$，第 i 层有2^i个结点，每个结点的 T 为$\frac{n}{4^i}$，

第 i 层的 T 为 $\frac{n}{2^i}$ 总的 T 为$\sum_{i=1}^{log_2 n} \frac{n}{2^i} = O(n^{\frac{1}{2}} log_2 n)$

3.substitute method：当 n > 1 时 T (n) ≤ 4T(n/4) +k • $\sqrt{n}$且当 n ≤ 1 时 T(n) ≤ k，

假设存在常数 k_1 和 k_2 使得 T(n) ≤ k_1 • n + k_2

证明：n = 1 时，令 k_1 = 4k，k_2 = k，T(1) ≤ k ≤ k_1 + k_2 成立

假设 T(n/4) ≤ k_1 • n/4 + k_2

则 T(n) ≤ 4T(n/4) + k • $\sqrt{n}$

≤ 4(k_1 • n/4 + k_2) + k • $\sqrt{n}$

= k_1n + 4k_2 + k • $\sqrt{n}$

≤ 4k_1n + k_2