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

  fun parenMatch p =
      let
        val init = map (fn CPAREN => (1,0) | OPAREN => (0,1))
        fun combine ((i,j), (k,l)) =
            if j > k then (i, l+j-k) else (i+k-j, l)
      in reduce combine (0,0) (init p) = (0,0)
      end

  fun optMax ((NONE, x) | (x, NONE)) = x
    | optMax (SOME x, SOME y) = SOME (Int.max (x, y))

  fun parenDist (parens : paren seq) : int option =
      if not (parenMatch parens) then NONE
      else let
        fun prefixes s = tabulate (fn i => take (s, i+1)) (length s)
        fun suffixes s = tabulate (fn i => drop (s, i)) (length s)
        val all = flatten ((map suffixes o prefixes) parens)

        fun tryDist parens' =
            let val n = length parens'
            in if n >= 2 andalso parenMatch parens' andalso
                  parenMatch (subseq parens' (1, n-2))
               then SOME (n-2)
               else NONE
            end
      in reduce optMax NONE (map tryDist all)
      end
end
```

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

  fun optMax ((NONE, x) | (x, NONE)) = x
    | optMax (SOME x, SOME y) = SOME (Int.max (x, y))

  fun parenDist (parens : paren seq) : int option =
      let
        (* pd : paren seq -> (int option * (int * int) * (int * int))
         *
         * pd s ==> (r, (i,j), (xL,xR))
         * where r ==> (SOME max) where max is the maximum distance
         *                between matching parentheses in s such that
         *                there are no unmatched CPAREN's to the right
         *                and no unmatched OPAREN's to the left,
         *              (NONE) otherwise.
         *       i is the number of unmatched CPAREN's,
         *       j is the number of unmatched OPAREN's,
         *      xL is the distance to the rightmost unmatched CPAREN,
         *  and xR is the distance to the leftmost unmatched OPAREN.
         *)
        fun pd s =
            case showt s
              of EMPTY => (NONE,(0,0),(0,0))
               | ELT OPAREN => (NONE,(0,1),(0,0))
               | ELT CPAREN => (NONE,(1,0),(0,0))
               | NODE (lp, rp) =>
                 let
                   val ((dL,(i,j),(x1,x2)), (dR,(k,l),(x3,x4))) =
                     Primitives.par (fn () => pd lp, fn () => pd rp)
                   val (mid, remain, dists) =
                     case Int.compare (j,k)
                       of EQUAL => (SOME (x2+x3), (i, l), (x1, x4))
                        | LESS => (NONE, (i+k-j, l), (x3 + length lp, x4))
                        | GREATER => (NONE, (i, l+j-k), (x1, x2 + length rp))
                 in (optMax (mid, optMax (dL, dR)), remain, dists)
                 end

        val (dist,remain,_) = pd parens
      in if remain = (0,0) then dist else NONE
      end
end
```