```
#lang racket

(define (prime?-h n m)
  (if (> m (/ n 2))
      #t
      (if (= (modulo n m) 0)
          #f
          (prime?-h n (+ m 1)))))

(define (prime? n)
  (prime?-h (abs n) 2))

(define (prime-count-h a b x)
  (define next (+ (expt x 2) (* a x) b))
  (if (prime? next)
      (+ 1 (prime-count-h a b (+ x 1)))
      0))

(define (prime-count a b)
  (prime-count-h a b 0))

(define (max-prime-count ab1 ab2)
  (if (> (cdr ab1) (cdr ab2))
      ab1
      ab2))

(define (best-coeffs min max gen-coeffs)
  (define coeffs (gen-coeffs min))
  (if (= min max)
       coeffs
      (max-prime-count coeffs (best-coeffs (+ min 1) max gen-coeffs))))


(define (gen-ab a-min a-max b-min b-max)
  (define a-to-ab (lambda (a)
                    (begin
                      (define coeffs
                        (best-coeffs b-min b-max (lambda (b) (cons (cons a b)
(prime-count a b)))))
                      ;(printf "(~s, ~s). count: ~s~n" (caar coeffs) (cdar coeffs) (cdr
coeffs))
                      coeffs)))
  (best-coeffs a-min a-max a-to-ab))

(define (solve)
  (define ab (gen-ab -999 999 -1000 1000))
  (* (caar ab) (cdar ab)))

(solve)
```