모바일 기반 R5RS Scheme 인터프리터의 성능 비교에 사용될 알고리즘 소개

한상곤

December 11, 2023 updated: December 14, 2023

1 Pi Digits(pidigits)

이 알고리즘은 파이의 자릿수를 생성하는 것입니다. 해당 알고리즘은 연산 속도를 비교하기 위해서 순차 알고리즘을 사용하여 계산을 진행합니다. 파이의 자릿수를 구하는 알고리즘은 Unbounded Spigot Algorithms for the Digits of Pi [1]를 참고하였습니다.

Listing 1: Pi Digits 예제

```
#lang racket/base

(require racket/cmdline)

(define (floor_ev q r s t x)
  (quotient (+ (* q x) r) (+ (* s x) t)))

(define (comp q r s t  q2 r2 s2 t2)
  (values (+ (* q q2) (* r s2))
          (+ (* q r2) (* r t2))
          (+ (* s q2) (* t s2))
          (+ (* s r2) (* t t2))))

(define (next q r s t) (floor_ev q r s t 3))
(define (safe? q r s t n) (= n (floor_ev q r s t 4)))
(define (prod q r s t n) (comp 10 (* -10 n) 0 1  q r s t))
(define (mk q r s t k) (comp q r s t k (* 2 (add1 (* 2 k))) 0 (add1
   (* 2 k))))

(define (digit k  q r s t  n row col)
  (if (> n 0)
      (let ([y (next q r s t)])
        (if (safe? q r s t y)
            (let-values ([(q r s t) (prod q r s t y)])
              (if (= col 10)
                  (let ([row (+ row 10)])
                    (printf "\t:~a\n~a" row y)
                    (digit k q r s t (sub1 n) row 1))
                  (begin
                    (printf "~a" y)
                    (digit k q r s t (sub1 n) row (add1 col)))))
            (let-values ([(q r s t) (mk q r s t k)])
              (digit (add1 k) q r s t n row col))))
      (printf "~a\t:~a\n"
              (make-string (- 10 col) #\space)
              (+ row col))))

(define (digits n)
  (digit 1  1 0 0 1  n 0 0))

(digits (command-line #:args (n) (string->number n)))
```

2 FANNKUCH

판쿠흐 벤치마크(FANNKUCH Benchmark)는 Kenneth R. Anderson과 Duane Rettig의 Performing Lisp Analysis of the FANNKUCH Benchmark[2]에 소개된 알고리즘입니다. n이 무한대가 될 때 $n * log(n)$일 것으로 추측됩니다

Listing 2: FANNKUCH 예제

```
#lang racket/base


(require (for-syntax (only-in racket/base
                              lambda
                              syntax
                              syntax-case
                              make-rename-transformer
                              #%app)))
(require racket/unsafe/ops
         racket/future)
(require racket/cmdline)

(define-sequence-syntax unsafe-in-fxrange
  (lambda () #'in-fxrange/proc)
  (lambda (stx)
    (syntax-case stx ()
      [[(d) (_ nat)]
       #'[(d)
          (:do-in ([(n) nat])
                  #f
                  ([i 0])
                  (unsafe-fx< i n)
                  ([(d) i])
                  #t
                  #t
                  [(unsafe-fx+ 1 i)])]])))

(define (unsafe-in-fxrange/proc n)
  (make-do-sequence (lambda () (values (lambda (x) x)
                                       (lambda (x) (unsafe-fx+ 1 x))
                                       0
                                       (lambda (x) (unsafe-fx< x n))
                                       #f
                                       #f))))


(define-syntax-rule (define/0st-bool (name arg0 rest ...) body ...)
  (begin
    (define-syntax-rule (name arg0/v rest ...)
      (if arg0/v (name/t rest ...) (name/f rest ...)))
    (define (name/t rest ...) (let ([arg0 #t]) body ...))
    (define (name/f rest ...) (let ([arg0 #f]) body ...))
    ))

(define (fannkuch n)
  (let ([future-slices (for/list ([k (unsafe-in-fxrange n)])
                         (let ([pi (for/vector #:length n ([i (
                            unsafe-in-fxrange n)])
                                     (unsafe-fxmodulo (unsafe-fx+ i
                                        k) n))]
                               [tmp (make-vector n)]
                               [count (make-vector (unsafe-fx- n 1))
                                  ]
                               [retval (mcons #f #f)])
                           (future (lambda ()
                                     (fannkuch/slice n pi tmp count
                                        retval))))))])
    (for/fold ([flips 0] [checksum 0]) ([f (in-list future-slices)])
      (let-values ([(flips2 checksum2) (touch f)])
          (values (unsafe-fxmax flips flips2) (unsafe-fx+ checksum
             checksum2))))))


(define (fannkuch/slice n pi tmp count retval)
  (define/0st-bool (loop even-parity? flips r checksum n-1 pi tmp
     count retval)
    (for ([i (unsafe-in-fxrange r)])
      (unsafe-vector-set! count i (unsafe-fx+ 1 i)))
    (let* ([next-flips (count-flips pi tmp n)]
           [flips2 (unsafe-fxmax next-flips flips)]
           [next-checksum (if even-parity?
                              (unsafe-fx+ checksum  next-flips)
                              (unsafe-fx- checksum next-flips))])
      (let loop2 ([r 1])
        (if (unsafe-fx= r n-1)
            (values flips2 next-checksum)
            (let ([perm0 (unsafe-vector-ref pi 0)])
              (for ([i (unsafe-in-fxrange r)])
                (unsafe-vector-set! pi i (unsafe-vector-ref pi (
                   unsafe-fx+ 1 i))))
              (unsafe-vector-set! pi r perm0)
              (unsafe-vector-set! count r (unsafe-fx- (
                 unsafe-vector-ref count r) 1))
              (if (unsafe-fx= (unsafe-vector-ref count r) 0)
                  (loop2 (unsafe-fx+ 1 r))
                  (loop (not even-parity?)
                        flips2
                        r
                        next-checksum
                        n-1
                        pi
                        tmp
                        count
                        retval)))))))
  (loop #t 0 (unsafe-fx- n 1) 0 (unsafe-fx- n 1) pi tmp count retval
     ))


(define (count-flips pi rho n)
  (vector-copy-all! rho pi n)
  (let loop ([k 0])
    (if (unsafe-fx= (unsafe-vector-ref rho 0) 0)
        k
        (let loop2 ([i 0]
                    [j (unsafe-vector-ref rho 0)])
          (if (unsafe-fx> j i)
              (begin
                (vector-swap! rho i j)
                (loop2 (unsafe-fx+ 1 i) (unsafe-fx- j 1)))
              (loop (unsafe-fx+ 1 k)))))))

(define (vector-copy-all! dest src n)
 (for ([i (unsafe-in-fxrange n)])
   (unsafe-vector-set! dest i (unsafe-vector-ref src i))))

(define-syntax-rule (vector-swap! v i j)
  (let ([t (unsafe-vector-ref v i)])
    (unsafe-vector-set! v i (unsafe-vector-ref v j))
    (unsafe-vector-set! v j t)))

; assume that n>=3
(command-line #:args (n)
              (define-values (answer checksum)
                (fannkuch (string->number n)))
              (printf "~a\nPfannkuchen(~a) = ~a\n"
                      checksum
                      n
                      answer))
```

3 Spectral Norm

$a_{11} = 1$, $a_{12} = 1/2$, $a_{21} = 1/3$, $a_{13} = 1/4$, $a_{22} = 1/5$, $a_{31} = 1/6$ 등의 항목이 있는 무한 행렬 A의 스펙트럼 노름의 값을 구하는 문제로 Hundred-Dollar, Hundred-Digit Challenge Problems의 3번 문제[3]입니다. 구글의 입사문제로 유명하며, 행렬 및 벡터 연산을 비교하는데 주로 사용됩니다.

Listing 3: Spectral Norm 예제

```
#lang racket/base

(require racket/cmdline
         racket/flonum)

(define (Approximate n)
  (let ([u (make-flvector n 1.0)]
        [v (make-flvector n 0.0)])
    (for ([i (in-range 10)])
      (MultiplyAtAv n u v)
      (MultiplyAtAv n v u))

    (let loop ([i 0][vBv 0.0][vv 0.0])
      (if (= i n)
          (flsqrt (fl/ vBv vv))
          (let ([vi (flvector-ref v i)])
            (loop (add1 i)
                  (fl+ vBv (fl* (flvector-ref u i) vi))
                  (fl+ vv (fl* vi vi))))))))

(define (A i j)
  (fl/ 1.0 (fl+ (fl* (->fl (+ i j))
                     (fl/ (->fl (+ i (+ j 1))) 2.0))
                (->fl (+ i 1)))))

(define (MultiplyAv n v Av)
  (for ([i (in-range n)])
    (flvector-set! Av i
                   (for/fold ([r 0.0])
                       ([j (in-range n)])
                     (fl+ r (fl* (A i j) (flvector-ref v j)))))))

(define (MultiplyAtv n v Atv)
  (for ([i (in-range n)])
    (flvector-set! Atv i
                   (for/fold ([r 0.0])
                       ([j (in-range n)])
                     (fl+ r (fl* (A j i) (flvector-ref v j)))))))

(define (MultiplyAtAv n v AtAv)
  (let ([u (make-flvector n 0.0)])
    (MultiplyAv n v u)
    (MultiplyAtv n u AtAv)))

(printf "~a\n"
        (real->decimal-string
         (Approximate (command-line #:args (n) (string->number n)))
         9))
```

4 mandelbrot

프랙털의 일종으로, 수열 Z_n의 절대값이 무한대로 발산하지 않는 복소수 c의 집합에 관한 점화식($Z_0 = 0$, $Z_{n+1} = Z_n^2 + C$)에 관한 문제입니다. 관련 알고리즘은 위키피디아를 참고하였습니다.

References

[1] Jeremy Gibbons. Unbounded spigot algorithms for the digits of pi. *The American Mathematical Monthly*, 113(4):318–328, 2006.

[2] Kenneth R Anderson and Duane Rettig. Performing lisp analysis of the fannkuch benchmark. *ACM SIGPLAN Lisp Pointers*, 7(4):2–12, 1994.

[3] Keith Briggs. Solutions to trefethen's problems from "hundred-dollar, hundred-digit challenge", siam news volume 35, number 1., 2002.