課題3

1029232337 杉本風斗

May 2, 2012

1 Ex3.21

示されたqueueの実装では,queueの先頭の要素へのポインタと末尾の要素へのポインタをペアとしてqueueのデータを表現しているため, あたかも2つ追加されているように見える.

```
(define (front-ptr queue) (car queue))
(define (rear-ptr queue) (cdr queue))
(define (set-front-ptr! queue item) (set-car! queue item))
(define (set-rear-ptr! queue item) (set-cdr! queue item))
(define (empty-queue? queue) (null? (front-ptr queue)))
(define (make-queue) (cons '() '()))
(define (front-queue queue)
  (if (empty-queue? queue)
    (error "FRONT called with an empty queue" queue)
    (car (front-ptr queue))))
(define (insert-queue! queue item)
  (let ((new-pair (cons item '())))
    (cond ((empty-queue? queue)
           (set-front-ptr! queue new-pair)
           (set-rear-ptr! queue new-pair)
           queue)
      (else
        (set-cdr! (rear-ptr queue) new-pair)
        (set-rear-ptr! queue new-pair)
        queue))))
(define (delete-queue! queue)
  (cond ((empty-queue? queue)
         (error "DELETE! called with an empty queue" queue))
    (else
      (set-front-ptr! queue (cdr (front-ptr queue)))
      queue)))
```

```
27
28 (define (print-queue queue)
29   (print (front-ptr queue)))
30
31 ;; test
32 (define q1 (make-queue))
33 (print-queue (insert-queue! q1 'a))
34 (print-queue (insert-queue! q1 'b))
35 (print-queue (delete-queue! q1))
36 (print-queue (delete-queue! q1))
```

2 Ex3.22

記憶を頼りにがんばる.

```
1  (define (make-queue)
2    (let ((front-ptr '())
3          (rear-ptr '()))
4      (define (empty-queue?)
5        (null? front-ptr))
6      (define (set-front-ptr! item)
7        (set! front-ptr item))
8      (define (set-rear-ptr! item)
9        (set! rear-ptr item))
10     (define (front-queue)
11       (if (empty-queue?)
12         (error "FRONT called with an empty queue" queue)
13         (car front-ptr)))
14     (define (insert-queue! item)
15       (let ((new-pair (cons item '())))
16         (cond ((empty-queue?)
17                (set-front-ptr! new-pair)
18                (set-rear-ptr! new-pair)
19                front-ptr)
20           (else
21             (set-cdr! rear-ptr new-pair)
22             (set-rear-ptr! new-pair)
23             front-ptr))))
24     (define (delete-queue!)
25       (cond ((empty-queue?)
26              (error "DAME!!!!!!" front-ptr))
27         (else
```

```
28            (set-front-ptr! (cdr front-ptr))
29            front-ptr)))
30      (define (dispatch m)
31        (cond ((eq? m 'empty?) empty-queue?)
32          ((eq? m 'front) front-queue)
33          ((eq? m 'insert) insert-queue!)
34          ((eq? m 'delete) delete-queue!)
35          (else front-ptr)))
36      dispatch))
37  ;; test
38  (define q (make-queue))
39  ((q 'empty?))
40  ((q 'insert) 'a)
41  ((q 'insert) 'b)
42  ((q 'delete))
43  ((q 'insert) 'c)
44  ((q 'front))
```

3 Ex3.23

リストの要素を (ひとつ前 . (要素 . ひとつ後ろ)) というように表現し, queue と同じように先頭と末尾へのポインタのペアを deque とする.
そうすると, 双方向に要素をたどることができるから示された操作を O(1) でできる.
要素が双方向にポインタを持っていることに注意して先頭・末尾への追加・削除の際にポインタを適切につなぎかえる.

```
1  (define (front-ptr deque) (car deque))
2  (define (rear-ptr deque) (cdr deque))
3  (define (set-front-ptr! deque item) (set-car! deque item))
4  (define (set-rear-ptr! deque item) (set-cdr! deque item))
5  (define (empty-deque? deque) (null? (front-ptr deque)))
6  (define (make-deque) (cons '() '()))
7  (define (front-deque deque)
8    (if (empty-deque? deque)
9      (error "FRONT called with an empty deque" deque)
10     (car (front-ptr deque))))
11 (define (front-insert-deque! deque item)
12   (let ((new-pair (cons '() (cons item '()))))
13     (cond ((empty-deque? deque)
14            (set-front-ptr! deque new-pair)
```

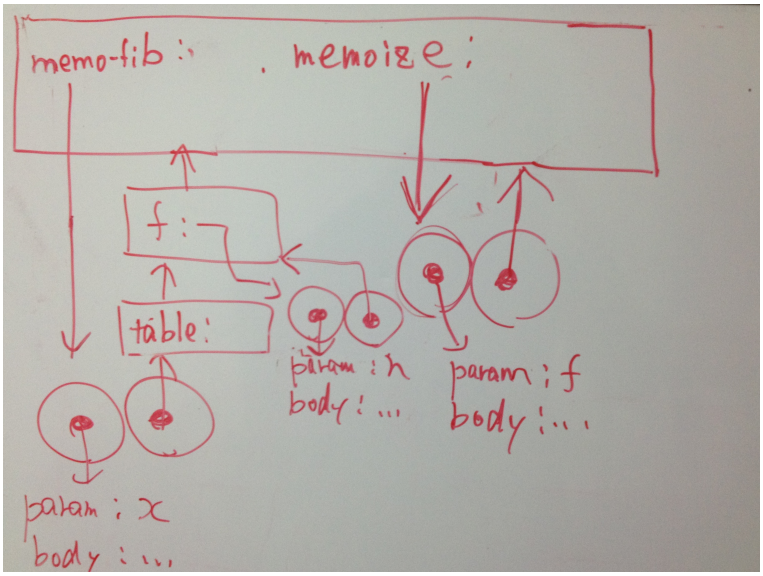
```
15            (set-rear-ptr! deque new-pair)
16            deque)
17       (else
18         (set-cdr! (cdr new-pair) (front-ptr deque))
19         (set-car! (front-ptr deque) new-pair)
20         (set-front-ptr! deque new-pair)
21         deque))))
22 (define (front-delete-deque! deque)
23   (cond ((empty-deque? deque)
24          (error "DELETE! called with an empty deque" deque))
25     (else
26       (set-car! (cddr (front-ptr deque)) '())
27       (set-front-ptr! deque (cddr (front-ptr deque)))
28       deque)))
29 (define (rear-insert-deque! deque item)
30   (let ((new-pair (cons '() (cons item '()))))
31     (cond ((empty-deque? deque)
32            (set-front-ptr! deque new-pair)
33            (set-rear-ptr! deque new-pair)
34            deque)
35       (else
36         (set-cdr! (cdr (rear-ptr deque)) new-pair)
37         (set-car! new-pair (rear-ptr deque))
38         (set-rear-ptr! deque new-pair)
39         deque))))
40 (define (rear-delete-deque! deque)
41   (cond ((empty-deque? deque)
42          (error "DELETE! called with an empty deque" deque))
43     (else
44       (set-cdr! (cdr (car (rear-ptr deque))) '())
45       (set-rear-ptr! deque (car (rear-ptr deque)))
46       deque)))
47
48 (define (print-deque deque)
49   (define (to-list item)
50     (cond ((null? item) '())
51       (else
52         (cons (cadr item) (to-list (cddr item))))))
53   (print (to-list (front-ptr deque))))
54
55 ;; test
56 (define deq (make-deque))
57 (empty-deque? deq)
```

```
58 (print-deque (front-insert-deque! deq 'a))
59 (print-deque (front-insert-deque! deq 'b))
60 (print-deque (front-insert-deque! deq 'c))
61 (print-deque (front-delete-deque! deq))
62 (print-deque (rear-insert-deque! deq 'd))
63 (print-deque (rear-delete-deque! deq))
```

4 Ex3.27

4.1 図

ホワイトボードに書いたのをカメラで写したので割とひどい.



4.2 O(n)

(memo-fib n) は, (memo-fib (- n 1)) と (memo-fib (- n 2)) を計算するが, (memo-fib (- n 1)) で計算した結果がメモ化されるので (memo-fib (- n 2)) は定数時間で計算できる.
よって, n に比例するステップ数で計算できる.

4.3 (memoize fib)

メモ化されていない fib が再帰的に呼び出されるためうまく計算時間を抑えられない.