# Week 05 - Lecture 2 Slides


# Dr. Yeganeh Bahoo



Created: 2023-10-04 Wed 11:07


# Table of Contents

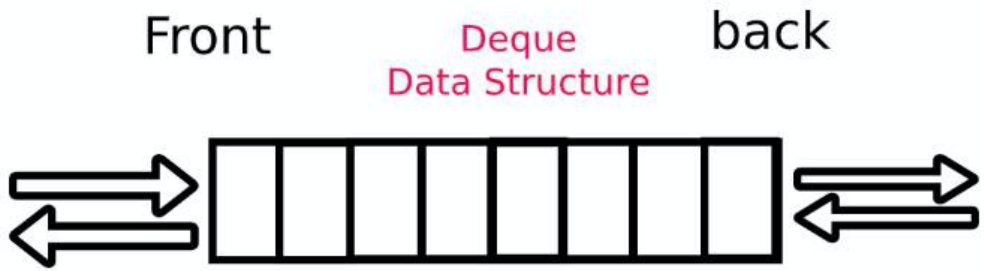


# Lecture 2: Deque and Lists as Sets

Learning objectives: By the end of this lecture you should be able to

- implement the deque data structures
- write programs that operate on lists as sets

## Double-Ended Queue (deque)




- A hybrid linear structure that provides all the capabilities of stacks and queues in a single data structure.
- It can be traversed as a FIFO (queue) or LIFO (stack)
- 4 operators: PUSH-FRONT, PUSH-BACK, POP-FRONT, POP-BACK
- Application example: **job-stealing algorithms**, where where the main worker is processing items from the front, while other workers, when idle, may steal the lowest-priority items from the back.

## Implementing a deque using two stacks: example

| Deque operation | Front stack | Back stack | Deque abstraction |
|---|---|---|---|
| | () | () | |
| push-front 1 | (1) | () | 1 |
| push-front 2 | (2 1) | () | 2, 1 |
| push-back 3 | (2 1) | (3) | 2, 1, 3 |
| push-back 4 | (2 1) | (4 3) | 2, 1, 3, 4 |
| push-back 5 | (2 1) | (5 4 3) | 2, 1, 3, 4, 5 |
| pop-front | (1) | (5 4 3) | 1, 3, 4, 5 |
| pop-front | () | (5 4 3) | 3, 4, 5 |

| Deque operation | Front stack | Back stack | Deque abstraction |
|---|---|---|---|
| pop-front | (4 5) | () | 4, 5 |
| pop-back | () | (4) | 4 |

## Example and exercise

Based on the example shown in the previous slide, complete the blanks in POP-FRONT.

```
(defstruct deque
  front
  back)

(defun push-front (item deque)
  (push item (deque-front deque)))

(defun push-back (item deque)
  (push item (deque-back deque)))

(defun pop-front (deque)
  "If front stack is empty, pushes the items from the back to the front,
   then pops it"
  (unless (deque-front deque)
    (do ()
        ((null (deque-back deque)))
      (push (pop ...) ...)))
  (pop (deque-front deque)))
```

### Solution

```
(defun pop-front (deque)
  "If front stack is empty, pushes the items from the back to the front,
   then pops it"
  (unless (deque-front deque)
    (do ()
        ((null (deque-back deque)))
      (push (pop (deque-back deque)) (deque-front deque))))
  (pop (deque-front deque)))
```

## Example

Two stacks deque simple application: a palindrome checker

```
(defstruct deque
  front ; the front stack
  back) ; the back stack

(defun pal-checker (list)
  (do* ((d (make-deque :front list))
        (first (pop-front d) (pop-front d))
        (last (pop-back d) (pop-back d)))
       ((not (and (equal first '[) (equal last ']))))
        (and (null first) (null last)))))
```

```
CL-USER> (pal-checker '([ [ ] ]))
T
CL-USER> (pal-checker '([ [ ] ] ]))
NIL
CL-USER> (pal-checker '())
T
```

## Sets

- A set is an unordered collection of items.
- Each item appears only once in the set.
- Use case examples:
    - to track items we have already seen and processed
    - when we want to calculate some relations between groups of elements
- The set interface:
    - ELEMENT-OF-SET? checks whether an item is in the set
    - INSERT-SET/REMOVE-SET adds/removes an item
    - SUBSET-SET checks whether a set is a subset of another set
    - UNION, INTERSECTION, DIFFERENCE, etc.

## ELEMENT-OF-SET?

```
(defun element-of-set? (x set)
  "Returns the rest of the set if x is in set; otherwise returns nil"
  (when set  ; if set is null (empty) WHEN returns nil (false)
    (if (equal x (car set)) (cdr set)
        (element-of-set? x (cdr set)))))
```

```
RTL-USER> (element-of-set? 2 ())
NIL
RTL-USER> (element-of-set? 4 '(3 4 2 6))
(2 6)
RTL-USER> (element-of-set? 2 '(3 4 21 6))
NIL
```

Run time of ELEMENT-OF-SET?

- It may have to scan the entire set (in the worst case, the object turns out not to be in the set).
- If the set has $n$ elements, ELEMENT-OF-SET? might take up to $n$ steps
- Hence the order of growth of the run time is $O(n)$.

Lisp as a MEMBER built-in function that works like ELEMENT-OF-SET?

## Exercise - INSERT-SET

Complete the blanks in the program below that defines function INSERT-SET.

```
(defun insert-set (x set)
  "Insert x in set"
  (if (...)
      set
      (cons ... set)))
```

```
RTL-USER> (insert-set 3 '(4 3 2))
(4 3 2)
RTL-USER> (insert-set 3 '(1 4 2))
(3 1 4 2)
```

### Solution

```
(defun insert-set (x set)
  (if (element-of-set? x set)
```

```
      set
      (cons x set)))
```

## Exercise - REMOVE-SET

Complete the blanks in the program below that defines REMOVE-SET.

```
(defun remove-set (x set &optional (acc ()))
  (cond ((null set) acc)
        ((equal x ...) (append acc ...))
        (t (remove-set x (cdr set) (cons ... acc)))))
```

```
RTL-USER> (remove-set 3 '(2 4 5))
(2 4 5)
RTL-USER> (remove-set 3 '(2 4 3))
(2 4)
```

### Solution

```
(defun remove-set (x set &optional (acc ()))
  (cond ((null set) acc)
        ((equal x (car set)) (append acc (cdr set)))
        (t (remove-set x (cdr set) (cons (car set) acc)))))
```

## Exercise - SUBSET-SET

In mathematics, a set $A$ is a subset of a set $B$ if **all** elements of $A$ are also elements of $B$.

Complete the blanks in the program below that defines SUBSET-SET.

```
(defun subset-set (a b)
  "Returns T if a is a subset of b; otherwise returns nil"
  (if (null a) T
      (and (element-of-set? ... ...)
           (subset-set ... b))))
```

```
RTL-USER> (subset-set () '(1 2))
T
RTL-USER> (subset-set '(1) '(3 4 1 3))
T
```

```
RTL-USER> (subset-set '(1 7) '(3 4 7))
NIL
RTL-USER> (subset-set '(1 7) '(1 7))
T
RTL-USER> (subset-set '(1 7) '(1))
NIL
```

### Solution

```
(defun subset-set (a b)
  (if (null a) T
      (and (element-of-set? (car a) b)
           (subset-set (cdr a) b))))
```

## An iterative implementation of SUBSET-SET

Notice that the program uses DOLIST.

```
(defun subset-set (a b)
  (dolist (item a t)
    (unless (element-of-set? item b)
      (return)))) ; breaks from the loop
```