Subject : ....

No. :

Date : ..../........./..........

04/11/2011

* what we've done

- stack for expression evaluation
- stack for infix ⇒ postfix
- queue
- mazing problem : recursive, stack, queue
- STL vector/stack/queue
- list
- Reading Assignments : circular array / dynamic array for queue
  multipl stack / queue in array
  list in C

* midterm

- 04/24/2011 (Sunday) 3pm ~ 5:30 pm
- 2.5 hrs, no extension
- open book or any printout material
- no electronic device
- English questions, English/Chinese answers
- range : textbook & slides until 04/19/2011 class
  (taught & reading assignments)

can ask TA questions on English meaning

can use pen/pencil

Linked List

* sparse polynomial w/ dense index array (Subsec 4.4.1)

$x^3 + 2x^2 + 5$

| (0,5) | (2,2) | (3,1) |

- remove $2x^2$?

| (0,5) | (3,1) |

need "moving" elements

- insert $4x^1$?

| (0,5) | (1,4) | (2,2) | (3,1) |

also need "moving"

(Sec 4.1)

* if "no moving" but still want to access sequentially?

location	0	1	2	3
	(0,5)	(2,2)	(3,1)	(1,4)
next	none	3	1	0

add a "next" field

graphically, first → $\boxed{(3,1)} \xrightarrow{next} \boxed{(2,2)} \xrightarrow{next} \boxed{(1,4)} \xrightarrow{next} \boxed{(0,5)} \xrightarrow{next}$ none

$\boxed{\text{singly linked list (chain)}}$

originally $\boxed{(2,2)} \xrightarrow{next} \boxed{(0,5)}$

after insertion

① $\boxed{(1,4)}$. next = $\boxed{(2,2)}$. next

② $\boxed{(2,2)}$. next = location of $\boxed{(1,4)}$

* how about deletion?

$\boxed{(3,1)} \xrightarrow{next} \boxed{(2,2)} \xrightarrow{next} \boxed{(0,5)} \xrightarrow{next}$ none

① $\boxed{(3,1)}$. next = $\boxed{(2,2)}$. next

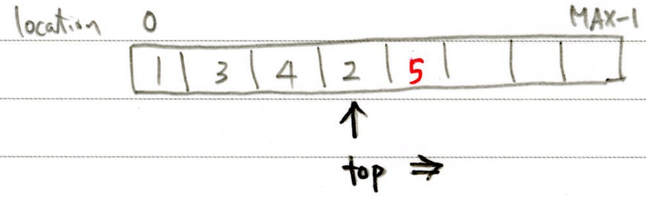
② free $\boxed{(2,2)}$

Subject : .... No. : 4-2
Date : ..../........../..........

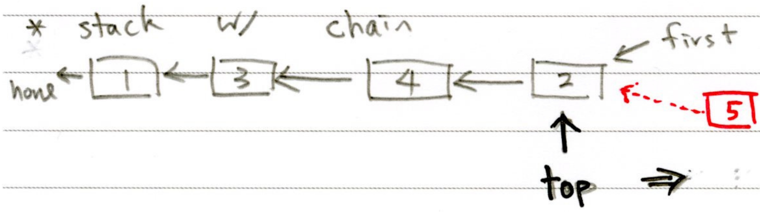
(Sec 4.2) Linked List in C
READING ASSIGNMENT

(Sec 4.3)

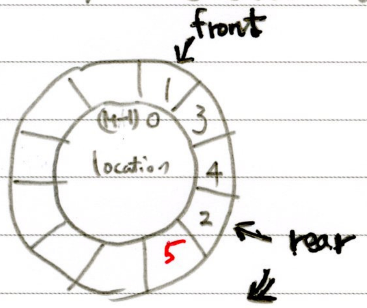
* stack w/ array



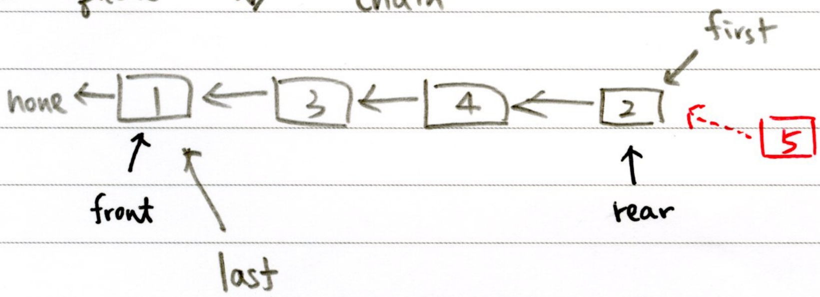
* stack w/ chain



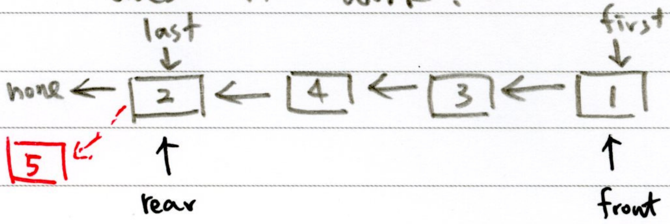
* queue w/ circular array



* queue w/ chain

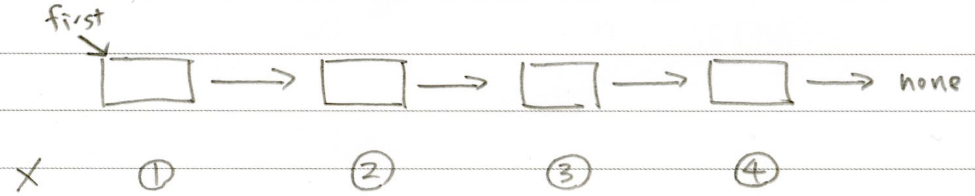


does it work?

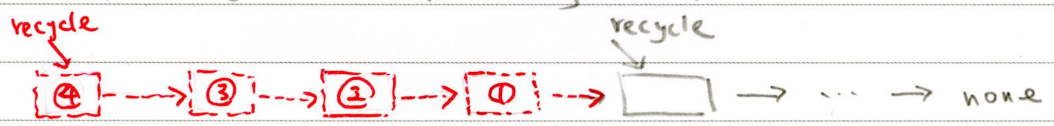


(Subsec 4.4.3)

* erase a whole chain



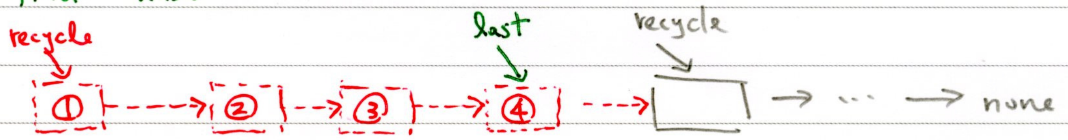
* erase a whole chain w/ recycle bin



* erase a whole chain w/ recycle bin at once

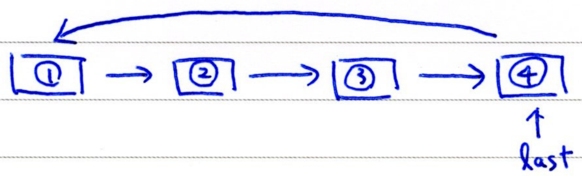
① find "last"

②



* erase a whole chain w/ recycle bin "by keeping first & last"

* erase a whole circular list w/ recycle bin ~~by~~ by only "last"



(because last.next is first)

① temp = last.next

② last.next = recycle

③ recycle = temp

* erase a whole circular list w/ recycle bin by only "first"

(how?)

(Sec 4.4) Other parts

* poly add
* "0" poly in circular list

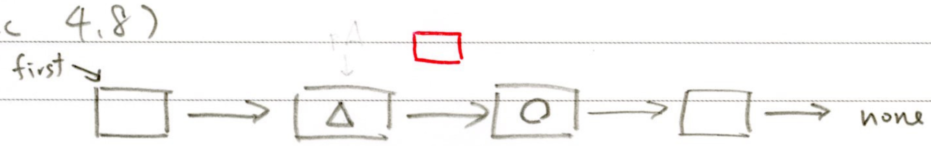
READING ASSIGNMENT

(Sec 4.5)

* chain inverting
* chain concatenation
* circular list insert front
* circular list length

READING ASSIGNMENT

(Sec 4.8)



* chain: insertAfter(Δ) is easy

  □.next = Δ.next

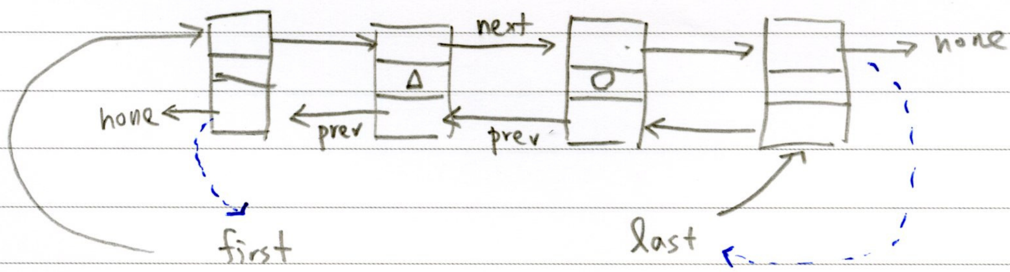
  Δ.next = location of □

* InsertBefore(○) is hard

  ① find ? such that (?.next is ○)

  ② InsertAfter(?)

* doubly linked list



"- - ->" circular

trade-off:

more pointers, more expensive to maintain, (Sec 4.6) Equivalence Class

"$\equiv$"

- symmetric: $b \equiv a \iff a \equiv b$
- reflxive: $a \equiv a$
- transitive: $a \equiv b$ and $b \equiv c \iff a \equiv c$

* if $0 \equiv 4$, $3 \equiv 1$, $7 \equiv 4$, $3 \equiv 5$, $2 \equiv 6$

$\{0, 4, 7\}$, $\{1, 3, 5\}$, $\{2, 6\}$

* how ?

① expand by symmetry

$0 \equiv 4$, $4 \equiv 0$
$3 \equiv 1$, $1 \equiv 3$
$7 \equiv 4$, $4 \equiv 7$
$3 \equiv 5$, $5 \equiv 3$
$2 \equiv 6$, $6 \equiv 2$

② group

$0 \equiv 4$
$1 \equiv 3$
$2 \equiv 6$
$3 \equiv 1$, $3 \equiv 5$
$4 \equiv 0$, $4 \equiv 7$
$5 \equiv 3$
$6 \equiv 2$
$7 \equiv 4$

can use lists to be dynamic

③ search by transitivity

$0 \equiv 4$, $4 \equiv 0$, $4 \equiv 7$, $7 \equiv 4$
$1 \equiv 3$, $3 \equiv 1$, $3 \equiv 5$, $5 \equiv 3$
$2 \equiv 6$, $6 \equiv 2$

like maze search, can use stack (by list) to store

* analysis

- n lists
- m pairs, 2m expanded pairs
- search n "starting pos." w/ 2m "moves"

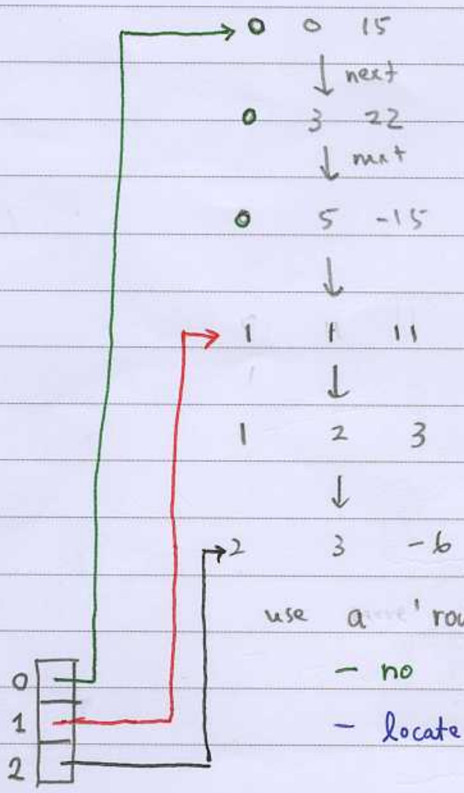
$O(m+n)$

(Sec 4.7) Sparse Matrix Again

* dense 2D : waste storage
* ordered triples : hard to insert & delete
  w/ dense array

* ordered triples w/ list

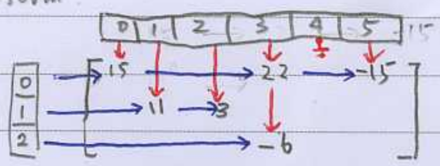
$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \end{bmatrix}$$



- easy to insert & delete
- '0' shows up many times
- hard to locate a[1][2] (linear search of O(#element))

* use a 'row entry' array
  - no need to store '0'
  - locate a[1][2] takes linear search of O(# col)

* if lazy and don't want to study fast Transform ?
  ① change next to next_of_same_row
  ② add next_of_same_col
  ③ maintain 'col entry' (remove col idx)



Q: need to store (row, col) ?

transpose : O(# elements) by swapping next_of_same_row for each ele.
& next_of_same_col

can be easily improved to O(1) (how ?)

* Subsec 4.7.2 ~ 4.7.4 READING ASSIGNMENT