03/10 163 Lec1

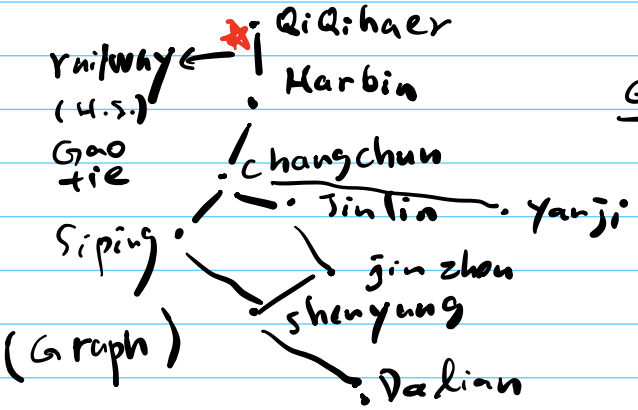
1. Intro

a. My group 方老 fang.song@pdx.edu

TCS (Theoretical Computer Szienze)

b. PSU-CCUT program : bilingual

c. A teasing example (celebrated result in TCS)



Q (starting point)

Goal: visit each city by G.T. exactly once

n: cities

$O(n!) \approx O(2^{n \log n})$

~~O(1)~~

Convinze your partner : a sequenze of zities list

[Q → H → J → .....]

size n proof/certificate

verification time :

O(1), O(log n), O(n),

✓ $O(n^2)$

(interative proof)

spaze (for storing the proof)

: O(n)

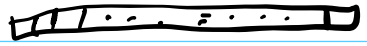
? less spaze

less time?

proof size O(log n) (Micali) {hash function, Merkle tree

★ [PCP theorem] (Probabilistic checkable Proof)

poly(n) size proof



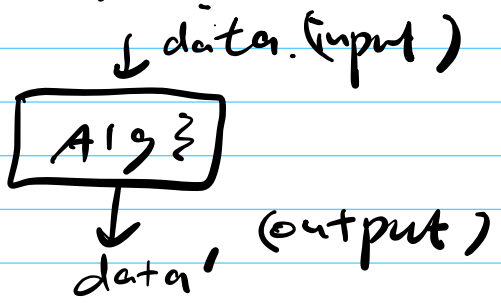
verify: O(log n) bits

☆: "Diversity" of CS!

1. D.S.

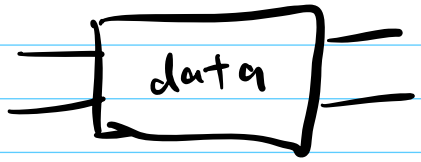
a. high-level discussion

Alg: :



D.S.: organize data so that it can be accessed quickly & usefully

ADT: (Abstract Data type)



☆ Benefits of Abstraction

Modular design.

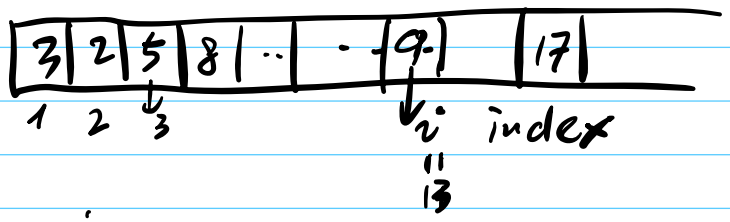
b. (linear) list 线性表.

- ADT List:

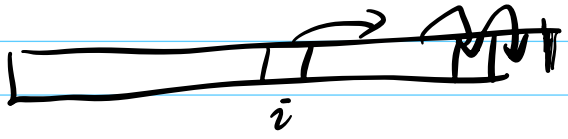
Data: - init. - get/retrieve (i)

OP's: - length - insert (x, i)
- del (i)
- output/print.

- 顺序表 (sequence list) / Array (数组)



ops: - init.
- insert(i, x) (x: 29)



O(n)

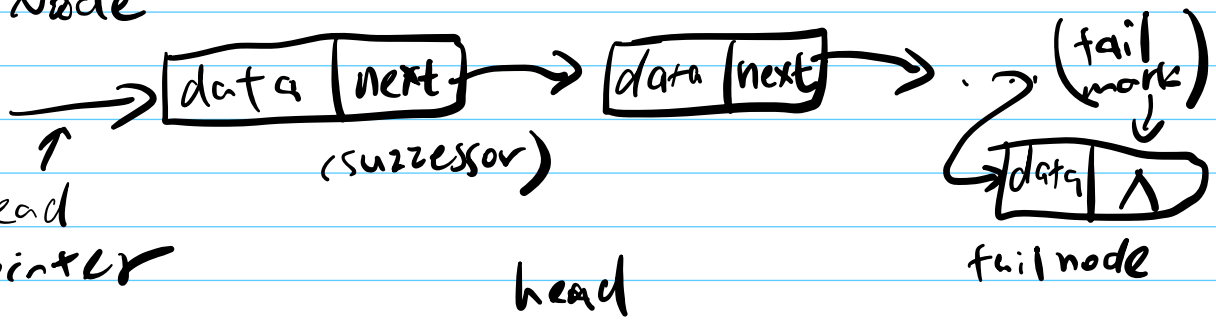
- del (i) O(n)

03/14 163 Lec 2

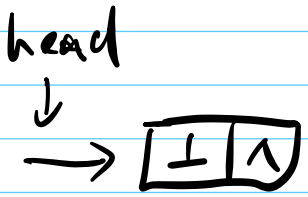
2 Linear List Cont'd

a. Linked List. (链表)

- Node

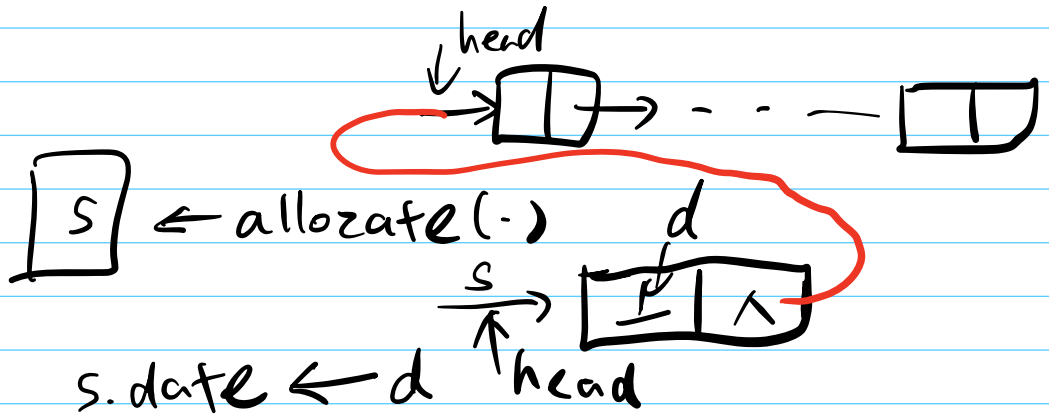


ops: - init:



- insert (head, i , d)

if i=1



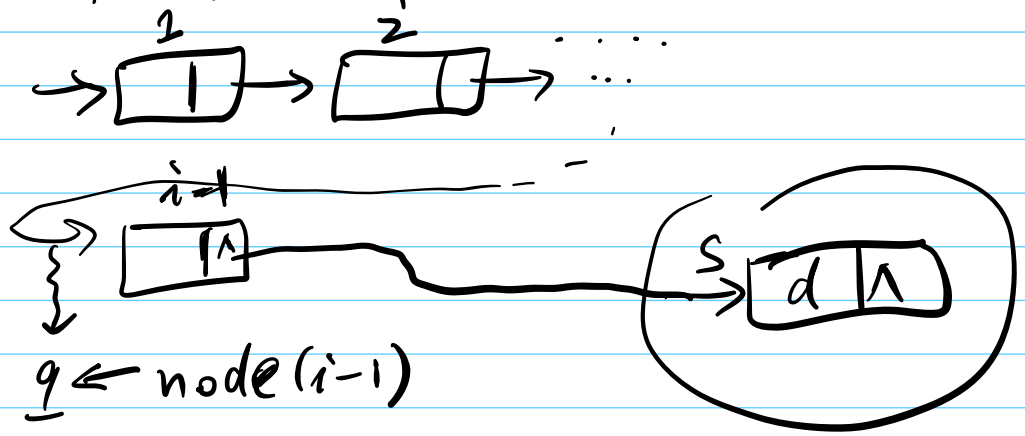
s.date ← d

s.next ← head

head ← s

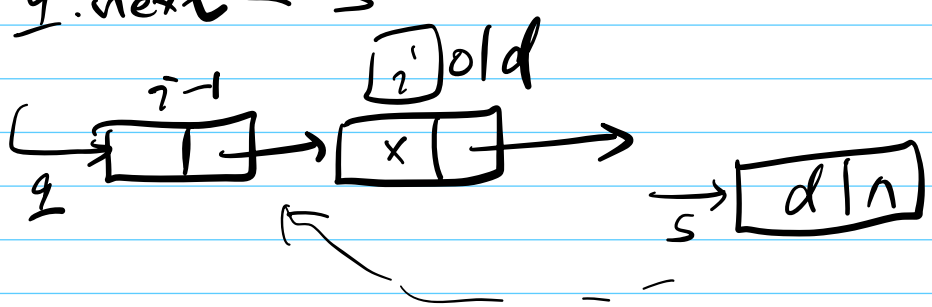
if i≠1

→ find i^{th} position.

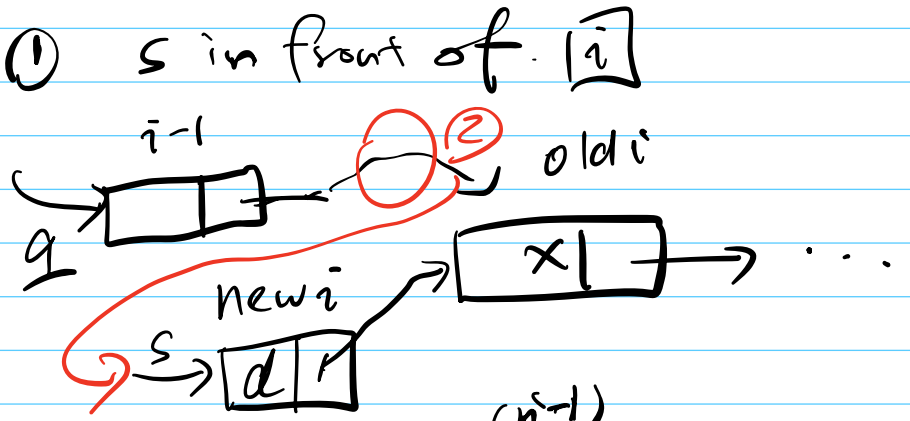


q ← node(i-1)

q.next ← s



① s in front of i

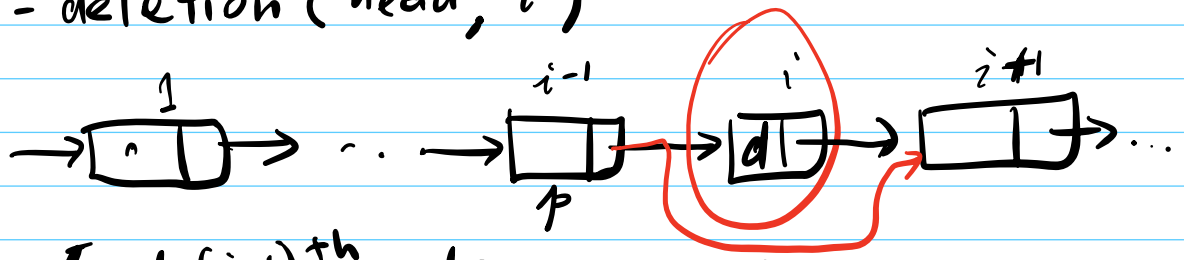


$s.next \leftarrow q.next$ (n^{i-1})

② $q.next \leftarrow s$

Time: $O(n)$.

- deletion (head, i)



- find $(i-1)^{th}$ node p
- $p.next \leftarrow q.next$ $q \leftarrow p.next$
- clean up (i+1)

Time: $O(n)$

	Array	(Singly) Linked List
- init	O(1)	O(1)
- output	O(n)	O(n)
- ins	O(n) // O(1) end	O(n) // O(1) beginning
- del.	O(n)	O(n)
- get(i)	O(1)	O(n)
Typical use case	Random access - few updates - quick read	Sequential access dynamic size freq. ins/del

Nuances:

O(n): move n data items

track n pointers

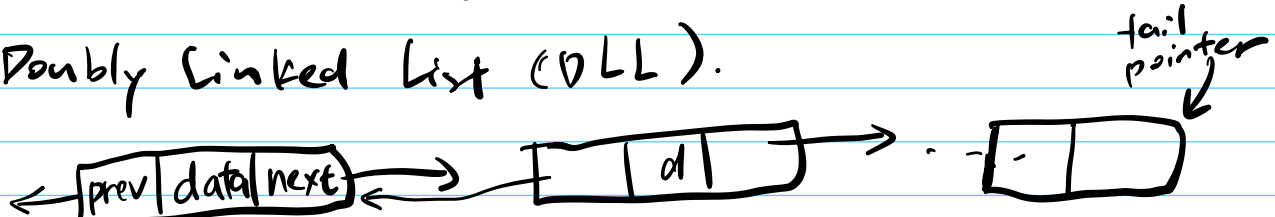
Memory:

waste if oversized

extra for pointers

250 → Cache: High Low

b. Doubly Linked List (DLL).



(pred) (suzz)

Advantage: Fast ins/del at last

- traverse in both directions

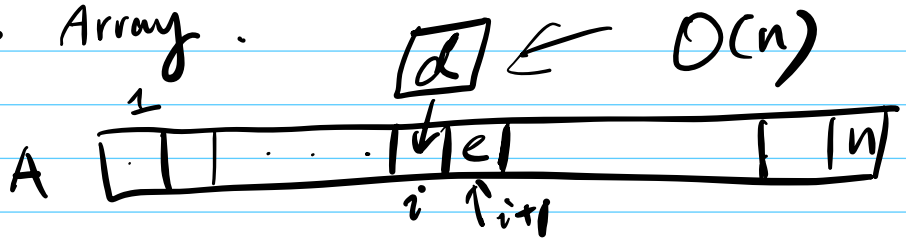
↳ update $(i-1)^{st}$ node while at i^{th} node

C. Example: swap w/ successor.

Given: List L ← Array, LL, dLL

Goal: Find first item (node) w/ value d.
& swap it w/ next item.

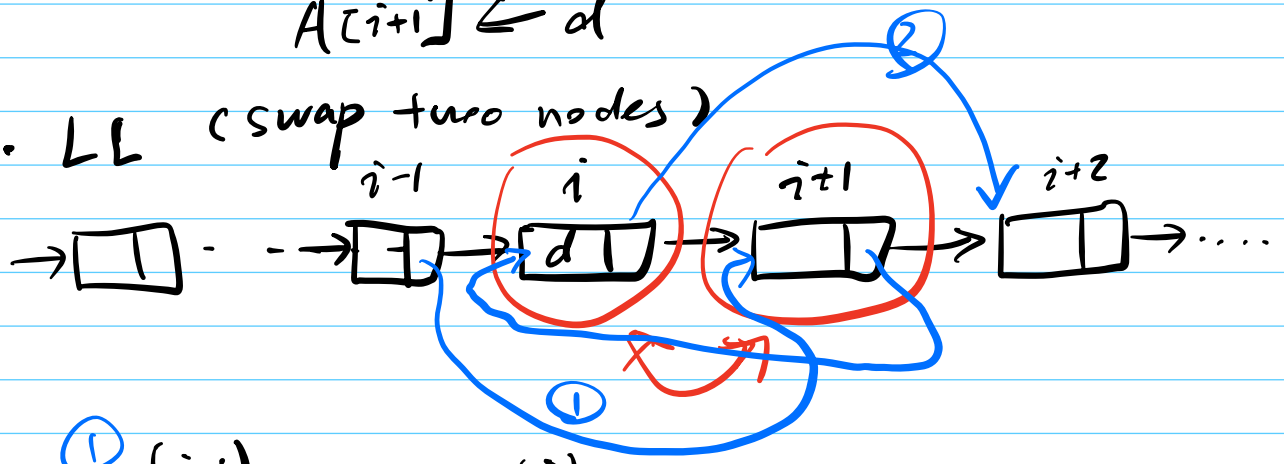
- Array.



A[i] ← A[i+1] (e)

A[i+1] ← d

- LL (swap two nodes)

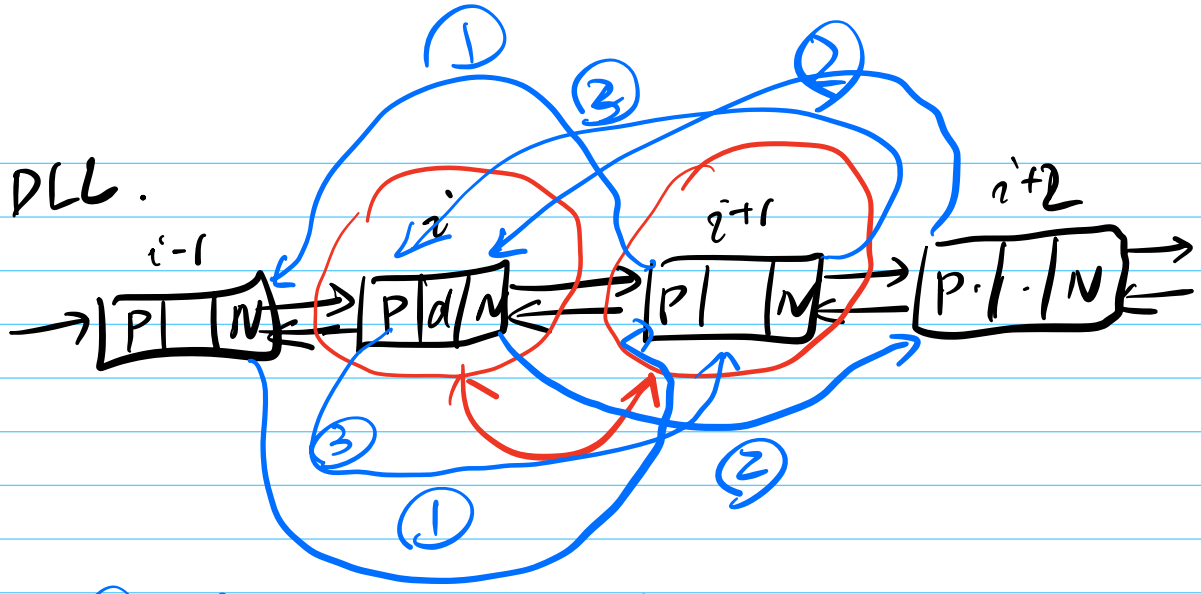


① (i-1).next ← (i).next

② (i).next ← (i+1).next

③ (i+1).next ← i

• DLL.



① (i-1).next ← i+1

(i+1).prev ← i-1

② i.next ← (i+2)

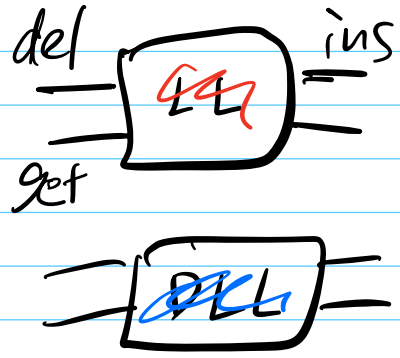
(i+2).prev ← i

③ (i+1).next ← i

i.prev ← (i+1)

Abstraction:

1. Find d (i^th)
2. del(i)
3. ins (i+1)



(modular design)

{

Apply in array:

→ Find(d) → i

→ del(A[i]) → d

→ ins(A, i+1, d)

03/17 (6) Lec3

0. Warm-up.

a. Decide Array / LL which to use?

- Store list of songs in a playlist

  Array: shuffle play on the playlist.

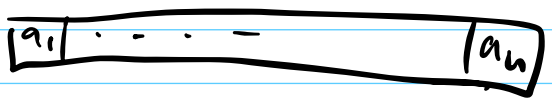
- Store reservation records at a hotel

  Array / L.L.

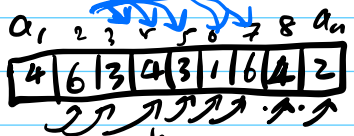
- Store a list of students & their grades

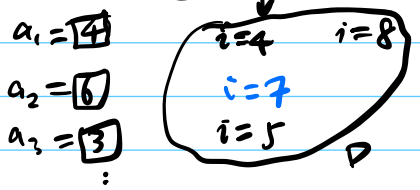
  Array

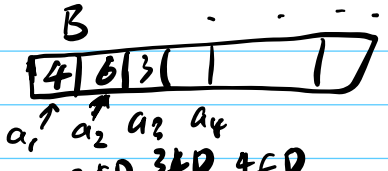
b. Remove duplicates

- Array  (integers)

- L.L. 

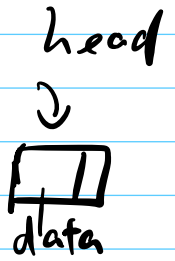
2deg: A 





```
RmDup(A)
  B ← init. array
  for i=1,···,n                          n
    if find(B, A[i]) != true
      ins(B, A[i]).                      O(n)
    i++
```

```
RmDup(head)
  nhead ← init. LL
  while head.next != ∧
    if find(nhead, head.data) != true
      ins(nhead, head)
    head ← head.next.
```



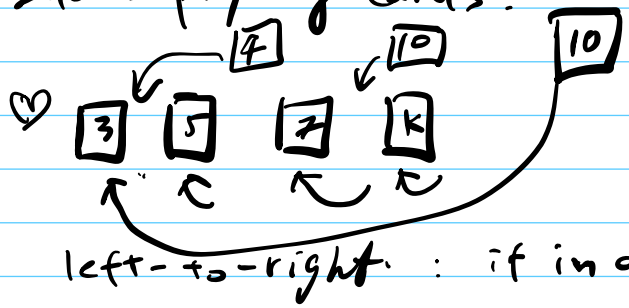
Time: $O(n^2)$

c. Cont'd from b, what if $(a_1, \cdots, a_n)$ sorted?

$a_1 \leq \cdots \leq a_n$.

1. Sorting: Insertion Sort.

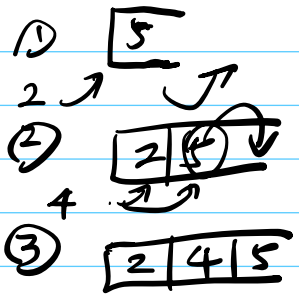
a. Idea: playing cards.



Given: A [a_1 | . . . | a_n]

Output: A' sorted $a_1' \le a_2' \le \cdots \le a_n'$

b. Ex [5 | 2 | 4 | 6 | 1 | 3]



---

c. pseudo code [Ex].

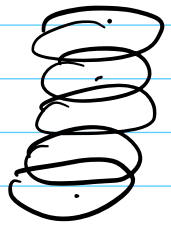
Time: $O(n^2)$

N.B. Better Sorting Alg's exist.

$O(n \log n)$ (merge sort)

---

2. Stack (栈)

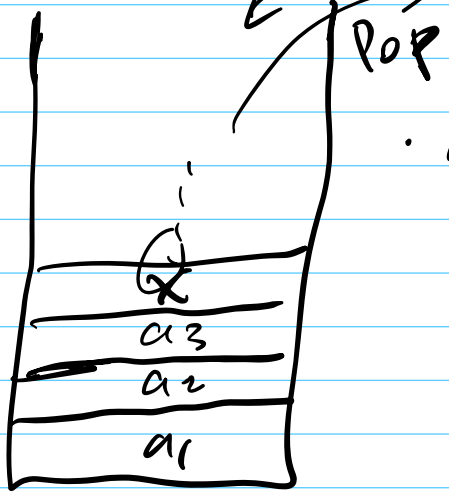


a. Motivation.

→ Browser navigation (back/forward)

→ Text editor (Vim, emacs) undo/Redo

b. basics. Push x



in-& out one-end

· ADT stack

Data:

Op's:

- init
- Push (ins)
- Pop (del)
- output

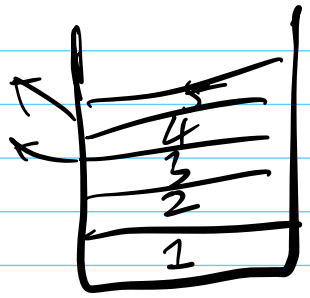
(Find, peek, size)

Last-In-First-Out (LIFO)

First-In-Last-Out (FILO)

· Example. Order in

1, 2, 3, 4, 5 →



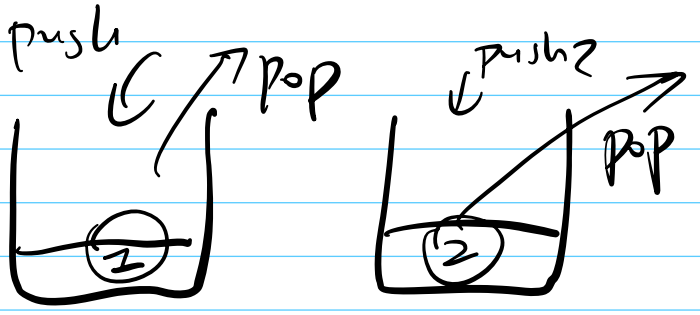
Out sequence which is possible!

✓ 5 4 3 2 1

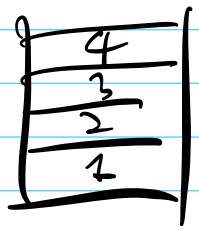
Push(1), 2, 3 ··· 4, 5

Pop (5) - - - - 1

✓. 1 2 3 4 5



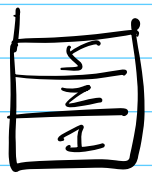
✓. 4 3 5 2 1



Push 1 ... 4

Pop 4, 3,

Push 5

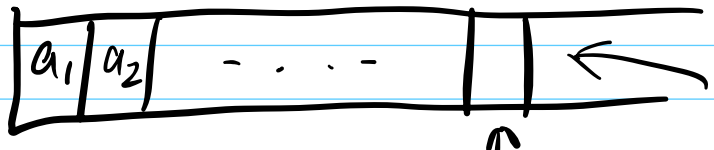


Pop 5, 2, 1

✗. 4 5 3 1 2 Ex: why NOT?

c. How to implement ?

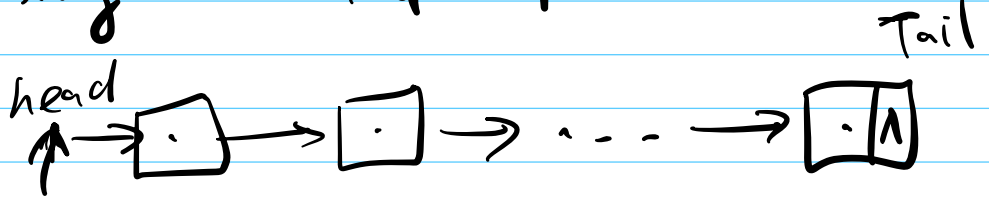
- using an array.



(top = 5) top index

Time: O(1)

- using a linked list.



(Stack Top)

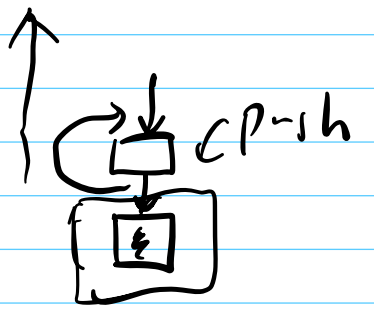
- push ↔ L.L. ins. at head
- pop ↔ ... del head.

Time: O(1)

03/21 163 Lec 4

1. Stack

a. Review



L.L. : push → ins at head.
pop del

Time: O(1)

b. Pratice.

· Parentheses.

Given: ( ) [ ] { } sequence

output: Valid

Valid if open → closed by same type & in correct order

{ } ✓ ( ), { } ✓ ([ )]
{ ( ) } ✓ (out of order)

Idea: use a stack keep track of open brackets (左括号)

- push Open (左)
- Upon closed (右): pop & match ?
- check empty stack at end.

Time: O(n)

. Min stack.

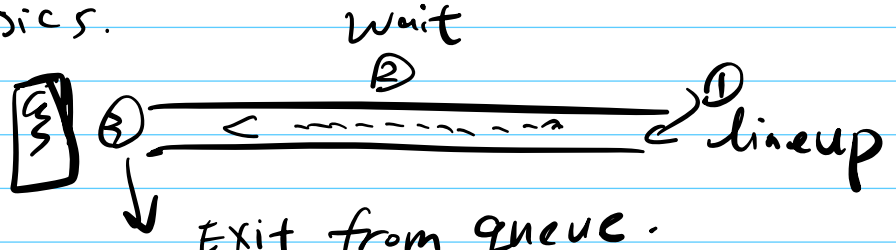
Goal: Use stack(s) to realize stack+ support stack op's & getMin(·). in O(1) time.

stack+
- push(·)
- pop(·) O(1)
- getmin(·)

Idea: Use 2nd stack to store current Min

2. Queue

a. Basics.



- head: only for exit (FIFO)
- rear: ... enter

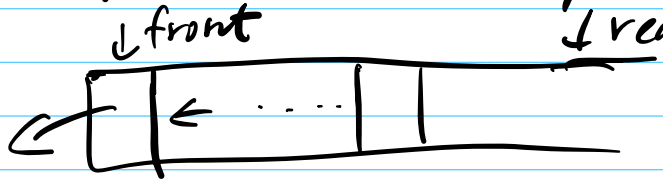
. ADT Queue.

Data: same data type

op's:
- initQ
- enqueue(x) (ins at end)
- dequeue(·) (del at front).

(isempty, - peek(·) front)

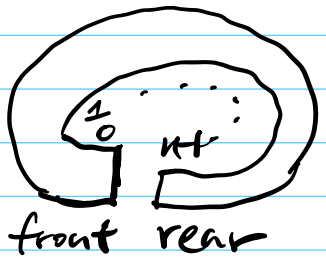
b. Implementation by array



- init array w/ n items.

OBS: multiple enq/deq.
(logical) queue shifts rightward
→ soon: "empty" queue.
no data & no space for new item.

· Solution: circular array

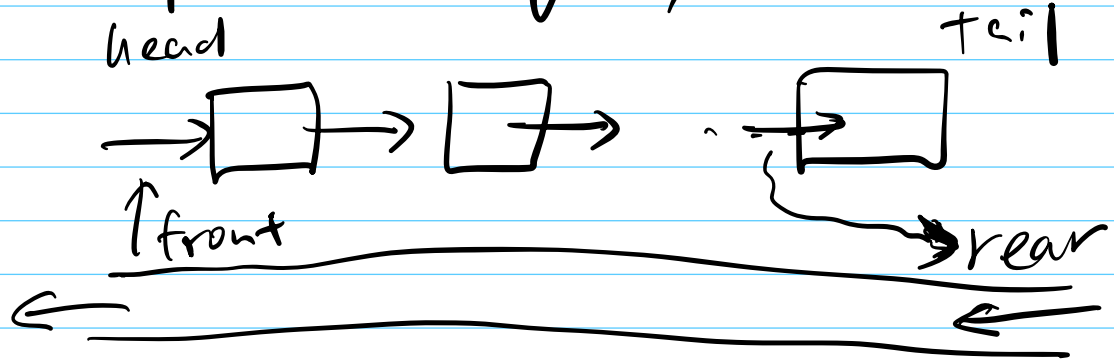


when rear/front indices progress to end, wrap around back to the beginning.

Update rule:
front ← front % sizeQ + 1
rear ← rear % sizeQ + 1
how to decide: empty front = rear
full front = rear
Convention: front: special symbol / kept vacant.
full: (rear + 1) % sizeQ = front

Time: O(1) EnQ & DeQ

c. Implementing by L.L.

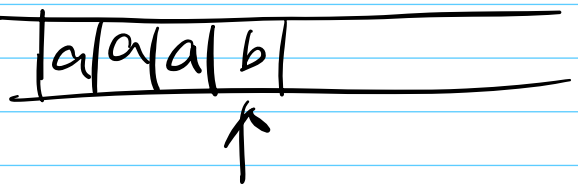


- enqueue: ins at tail of L.L.
- dequeue: del at head

Time: O(1)

d. Practice

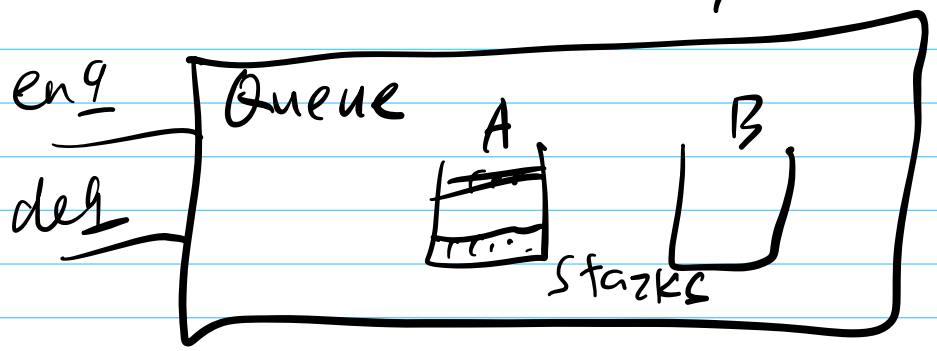
- Given a queue, find the 1st non-repeating character.



- Given a queue, reverse the order of its elem's.

Idea: use a stack

- Implement a Queue w. stacks



Idea: . use 2 stacks

- enq: push to A.
- deq: pop from A

↓ reduce to previous Ex.

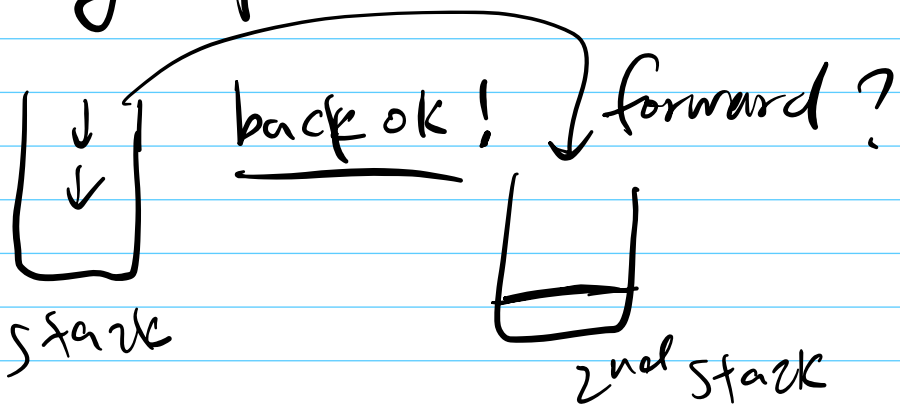
: reverse order

03/24 163 Lec 5

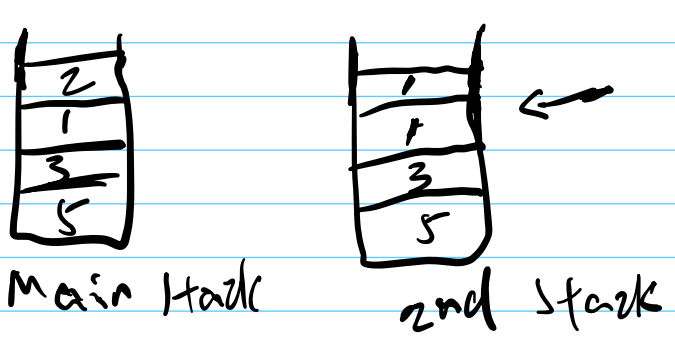
0. Warm-up

a. choose an appropriate D.S.

- check parenthesis matching
  : stack
- A cache between PC & printer to store printing assignments
  : Queue
- History of web browser



b. MinStack (stack t)
- getmin()



1. Strings.

a. Basics.

- what's a string?

  A sequence of characters.

  $s = s_1, \cdots, s_n$ $s_i \in \sum$ (alphabet 字符表) $\sum = \{0,1\}$ ASCII Unicode

  - length: $|s| = n$ finite
  - empty string (null) ε, $|\varepsilon| = 0$

- substring: a contiguous (physical/spatial adjacent) segment of s. (字串) (vs. continuous)
  - $1 \le i \le i+\ell \le n$

    $s_i \cdots s_{i+\ell}$ a substring of length ℓ.
  - prefix (前綴): substring starting at $i=1$, $(s_1 \cdots s_\ell)$
  - sufix (後綴): substring ending at $i=n$. $(s_{n-\ell}, \cdots s_n)$
  - $s = t$ iff: $|s| = |t| = n$, & $s_i = t_i$ $\forall i = 1, \cdots, n$

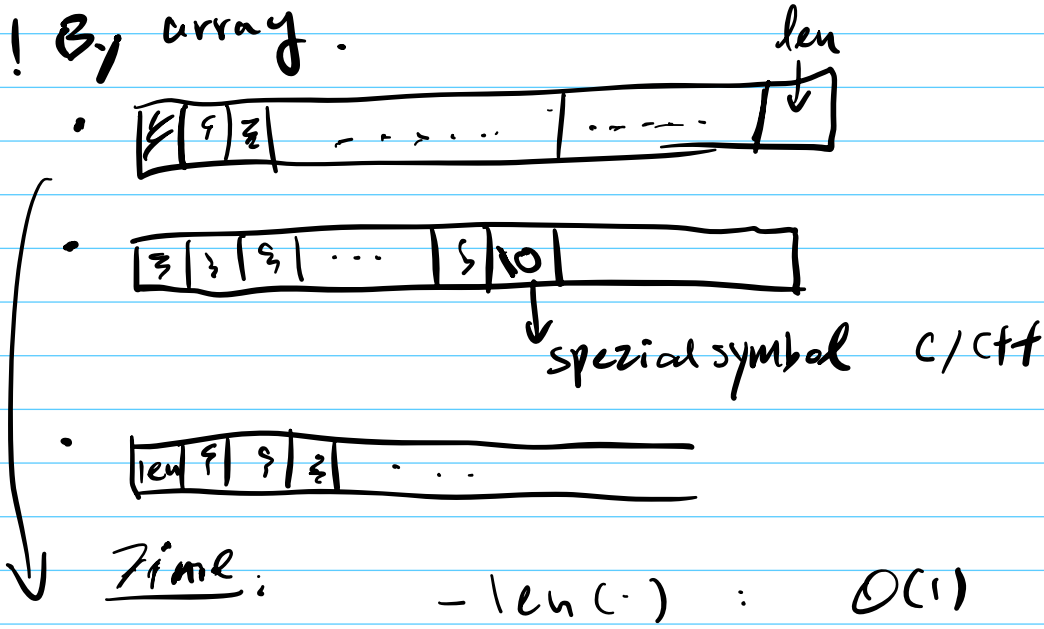
b. ADT String

- Data
- OP's:
  - len(·): length of string
  - char(i): return ith char
  - strconc(s, t): return $s \| t$ (Concatenation: 字拼接)
  - substr(i, ℓ): substring $s_i \cdots s_{i+\ell}$

(- prefix - sufix, - strcomp(s,t), - strins(s,i,t), ...)

c. Implementation.

! By array.



Time:

- len(·) : $O(1)$
- char(i) : $O(1)$
- strconc(S,t): $O(|S| + |t|)$
- substr(i,l): $O(n)$

2. String Matching.

a. Problem description

Given: (Primary) string
$t[1, \dots, n]$ of length n

Pattern
$p[1, \dots, m]$ of length m, $m \leq n$.
chars from Σ (e.g. $\{a, \dots, z\}$)

Goal: find valid shift i
s.t. p occurs in T at i. (if exists).

b. Brute-force algorithm.

- Idea: try all possible shift: $i = 1 \cdots n-m$

  starting at i ? $p[1, \cdots m] = t[i, \cdots, i+m-1]$

- correctness ✓

  if exists, will find all such i.

- Time $O((n-m) \cdot m) = O(n \cdot m)$

- Ex: Suppose all p_i's are distinct

  $p[1, \cdots, m]$, can we do better?

  idea: whenever a mismatch.

  go to that location for next matching attempt

  every char. t is compared at most twice.

  $\Rightarrow O(n)$ $\leftarrow$ vs $O(n \cdot m)$

c. KMP (Knuth-Morris-Pratt) alg.

Idea: avoid nezessary shifts from prior comparison

Side information: Longest prefix suffix

$LPS(q) := \max\{k : k < q\}$.

prefix of p which is a suffix of p.

$q = 1, \ldots, m$.