

pdf 782

week2.

wrapper class

: Integer, Double, Long,
Boolean, Character etc

Autoboxing

primitive → wrapper
ex) Integer num = 10;

Unboxing
wrapper → primitive
ex) int num = num + 50;

String → primitive
int i = Integer.parseInt(str);

String

An immutable object handling

strings.

ex) str = str + "!"

* 초기거비는게 아니라
* 새 객체 String을 만드는 것
* 한번 만들면 내용 변화 X

ArrayList

List<Integer> var
= new ArrayList<>();

var
size()
remove(i)
add(2, 5)
set(2, 5)

Generics

parameterized class

- (1) · ArrayList<BaseType>
 - Glass < Coke >
 - Multiple parameter < T1, T2 >
 - ~~자리에 대체할~~ (Textbook)
- Supper class

Bounded Type

Type Safety: 값과 타입이 일치해야 함

Sort T(n)	Worst	Avg
Bubble	$O(n^2)$	$O(n^2)$
selection	$O(n^2)$	$O(n^2)$
insertion	$O(n^2)$	$O(n^2)$
shell	$O(n^2)$	$O(n^{1.5})$
merge	$O(n \log n)$	$O(n \log n)$
heap	$O(n \log n)$	$O(n \log n)$
quick	$O(n^2)$	$O(n \log n)$

< T extends Comparable<T> > ADT LIST

T이 조상, 혹은 자손의 비교를 지원하는 경우
→ compareTo 이용

JCF: Java Collection Framework

JCF

(Interfaces
Implementations
Algorithms)

Responsibilities
add(*)
add(i, x)
remove(*)
remove(idx)
get(idx)
indexof(*)
size()
clear()
String.format("%s %d", ,)

public void add(T x) {

if (start == null) start = data;

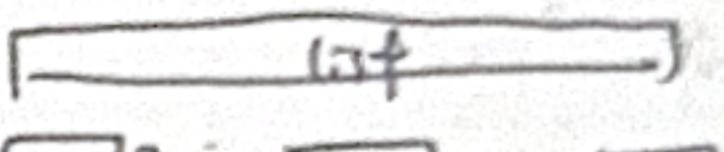
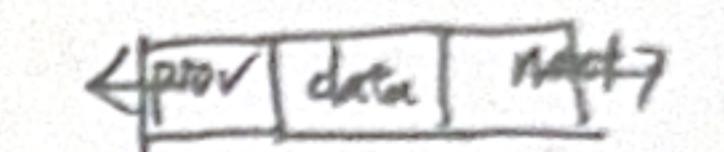
start = new Comparable
Node(T(x, start))

}

Node(T p = start);
while (p.next != null) {
if (p.next.data.compareTo(x) > 0)
p = p.next;
break;

p.next = new Node(T(x, p.next));

doubly linked list



양쪽 방향에서 찾기.

dequeue를 이용하면 쉽게 구현

List using Array

(insertion O(n)
Deletion O(n))

array를 파악하여 T로 initial 할
을 때 발생하는 문제점.

Iterator<Animal> it = animals.
iterator()

list = new T[capacity]
→ unchecked cast ~.

list = new Object[capacity]

Performance

Time Complexity: T 실행시간

Space Complexity: 메모리
설정

프로그램 시간에 영향 미치는 요소

문제크기 / 알고리즘 / 메모리 접근 속도 /

CPU/processor 수 / 컴퓨터 하드웨어

class Node<T> {

Log	Linear	Quadratic	Exponential
n			
n log n			
n^2			
n^3			
2^n			

Asymptotic Notations

O : upper

Ω : lower

Θ : upper and lower
similar

$f(n) \in O(n^2)$

$n^2, 3n^2+2n, 3n^2+n\log n, \log n, 3n$

$f(n) \in \Theta(n^2)$

$n^2, 3n^2+2n, 2^n+n\log n, 7n^3+5n$

$\Theta(n^2) = O(n^2) \wedge \Omega(n^2)$

$T(n) = O(n^2) (\theta)$

$O(n^2) = T(n) (\neq)$

public void add(T x) {

if (start == null) start = data;

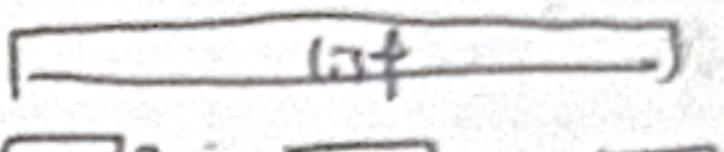
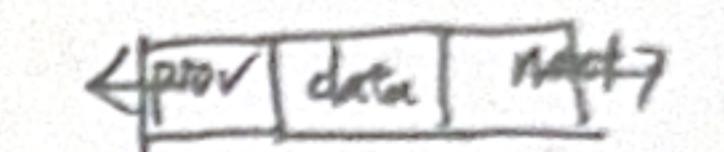
start = new Comparable
Node(T(x, start))

}

Node(T p = start);
while (p.next != null) {
if (p.next.data.compareTo(x) > 0)
p = p.next;
break;

p.next = new Node(T(x, p.next));

doubly linked list



마지막 노드가 null이 아님.

첫번째 노드 가리킴.

singly linked list ut doubly linked list 가 같은 용어.

List Iterator

ListIterator<String> it =
new LinkedList<String>().listIterator();
= new TechGiant.Iterator();

ListIterator = standard iterator

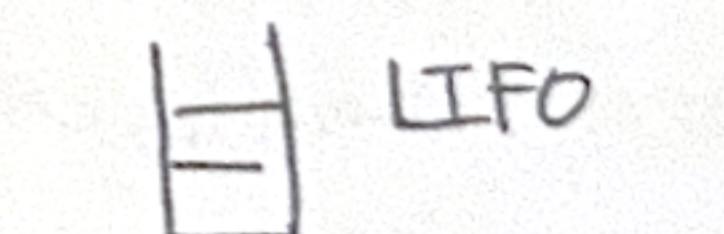
모두 강제.
양방향 접근 / nextIndex()
previous() previous()
next() next()

inplace 변경

set(E e) 교체

add(E e) 추가.

STACK.



push

pop : 빠져나오지 않으면 전부 지우기

peek : 맨위 리턴

isEmpty : 비었는지

clear : 모두 지우기

pop이나 peek 히트 %를 줄이기.

① null ② 예외 발생

현대적

프로그램 처리
 예상 출처, 만든 대로
 소스
 대기실(선택) 대기
 출가지 투고 선택 바로 옮기면 OK
 선택 → 수작업 처리
 컴퓨터는 출가지 위치를 더럽고
 3 5 1 5 7 2 4 9 8 1 2 3
 계산부로 만나면
 두개 top 개념
 계산부에 걸려온다.
Program Stack
 Activation Record (영역내에서)
 헬드레거의 구조/포트
 선택 두개로 구현됨.
 두로 가기
 Undo (두로 가기): 이전 상태 push
 Redo = 두로 가기로 추적된 주고 push
 빠트리깅 (스택)
 ↴ 가로로 옮기고

Ex-Simulating a Waiting Line

Waiting	Customer
line numberOfArrivals numberServed totalTimeWaited simulate displayResults()	arrivalTime transactionTime customerNumber getArrivalTime getTransactionTime getCustomerNumber

 Math.random()
 uniformly distributed 0~1
Linear Queue vs Circular Queue

$$(first == (last + 2) \% queue.length)$$

$$\rightarrow 2\text{번} \frac{1}{2}\text{점.}$$

$$ArrayQueue \rightarrow \text{복사}$$

$$size$$

$$(L-F+QL+1)\%QL : \text{한번의 회전.}$$

$$enqueue$$

$$L=(L+1)\%QL$$

Linked Queue
 addToFront
 addToBack
 removeFront
 removeBack
 $first = f.next$
 $f.next = null$
 size
 $last = last.prev$
 $last.next = null$
 size

Recursion
 $n=0, 1 \rightarrow n+1$
 1. 단계 조건 (base)
 2. 수렴 조건 (convergence)
 Factorial $\rightarrow T(n) = O(n)$

$$(x^0, x^{\frac{1}{2}}, x^{\frac{1}{4}}, \dots, x^{\frac{n}{2}}, \dots, x^0) \quad n = even$$

$$(0, \log n) \quad n = odd$$

링크드 스택과 비슷하나 Top 정의는
 배열 차이로 다른 사용법 ←
 O(n)

top()이면 선택법
 각각을 대소 Stack 대신 Deque로
 Fibonacci.

Dequeue <String> Stack
 $= new ArrayDeque<String>;$

- Tail recursion
- iteration
- Stack memo
- recursion
- tail
- iteration
- stack memo

Queue
 enqueue
 put, insert
 dequeue, T (poll)
 getFront: T (peek)
 isEmpty: Boolean
 clear: void

Tree = (nodes + edges)
 Empty Tree도 tree.
 Parent & children
 Root Node Edge Ancestor n descendant Degree (자식)
 Leaf: degree 0 + degree of tree

Interior node: not leaf
 the length of a path: $\ell(\text{path})$
 The height of a tree: 거상길 root
 -to-leaf path
 The depth: (= level)
 = the length of its root path

Heap (for fast comparable objects).
 priority queue
 add peek clear
 remove isEmpty

Complete binary tree
 ↳ array로 가정 (index)

$1 \quad 1 \times 2 + 1$
 $2 \times 2 \quad 3 \times 2 + 1$
 $3 \times 2 \quad 4 \times 2 + 1$

insertion ① add → ② upheap
 deletion ① Return ② last no → root
 ③ downheap

Binary tree
 $n = \frac{2^{h+1}-1}{d-1}$ (Full tree 노드수)
 maximum degree + 2 인트리
 (모든 degree가 그인 것 아님)
 Empty tree도 Binary tree.

Ordered Tree
 $\circlearrowleft \rightarrow \circlearrowright$ 순서 관계.

Sort
 pairwise distribute
 internal External.
 Java.util.Arrays → modified quick
 modified Merge
 insertion

Bubble Sort $i=n-1 \rightarrow 1$
 $j=0 \rightarrow n-1$
 $\text{if } a[i] > a[j] \text{ swap}$

Selection Sort
 가장 큰 거 찾은 다음으로

Insertion Sort 비교하면서 삽입
 Best: $O(n)$
 Worst: $O(n^2)$
 Avg: $O(n^2)$

Shell Sort: $d = n/2$
 주어진 정렬식 구현
 $\text{avg } O(n^1.5)$
 $O(n^2)$

Heap Sort
 highest를 주트로,
 꼭짓 heapify 고정 반복

Bottom up $O(n)$
 Top Remove $O(log n)$
 $O(n) + n \times O(\log n) = nO(\log n)$

Divide and Conquer
 Merge Quick (pairwise)

Binary Search Tree
 2개 노드까지 자식 가질 수 있다.
 각 노드가 같다.
 모든 원쪽 노드라는 주트다 같다
 모든 오른쪽 노드라는 주트다 같다.

Hashmap
 Var < readability
 ① only local variable
 ② declare & init at once.
 Hashmap
 Put(k, v)/remove(k)
 Get(k) containsKey(k)

Bucket (distribute)
 버킷 만들고 insertion

Counting Sort
 Radix Sort
 $O(N \cdot K)$ $O(d \cdot (n + k))$ ↳ 자릿수 유통하지 않음.

Linear Hashing
 Open addressing
 $(h_1(h_2(e)))$

Quadratic Hashing
 double hashing
 $(h_1(h_2(e)) + h_3(e))$

Separate chaining
 key % size