

# LINKED LISTS

Curve of forgetting study planner

1HOUR 1DAY 2DAY 1WEEK 2WEEK 1MONTH 3MONTH

date : 2019 . 04 . 14

```
public void insertAfter(int x) {  
    Node newnode = new Node();  
  
    newnode.data = x;  
    newnode.next = cur.next;  
  
    cur.next = newnode; }  
  
public int deleteCurrent() {  
    int ret;  
  
    if (cur == header) { ret=-1; }  
    else {  
        ret = cur.data;  
        prev.next = cur.next;  
        cur = cur.next; }  
  
    return ret; }
```

## DOUBLY Linked Lists

— Node의 차이



이전 노드를 가리키는 포인터 prev

# STACKS & QUEUES

## RECURSION

Curve of forgetting study planner

1HOUR 1DAY 2DAY 1WEEK 2WEEK 1MONTH 3MONTH

date : 2019. 4. 15

### Queues

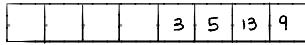
- homogeneous items forming a line
- FIFO (First In First Out)
- Enqueue : adds an item to the rear
- Dequeue : removes data from the front

#### Implementing : linked list

class Queue {

```
private final LinkedList llist;
public Queue() {llist = new LinkedList();}
public void enqueue(int x) {llist.insertAtEnd(x);}
public int dequeue() {return llist.deleteFirst();}
public boolean isEmpty() {return llist.isEmpty();}
}
```

#### Implementing 2: array



- enqueue, dequeue 하다보면 한글자로 Component가 끌리는 경우 생김.

Circular structure을 이용

```
class Queue {
private final int[] data;
private final int size;
private int front, rear;
public Queue(int s) {
    data = new int[s];
    size=s; front=0; rear=-1+s;
}
public boolean isEmpty() {return ((rear+1)%size)==front;}
public boolean isFull() {return ((rear+2)%size)==front;}
public enqueue(int x){
    if(!isfull()){
        rear=(rear+1)%size;
        data[rear]=x;
    }
}

```

```
public int dequeue(){
    int ret;
    if (isEmpty()) return -1;
    ret= data[front];
    front=(front+1)%size;
    return ret;
}
```

※ 구현시 주의할점

length를 array 길이로 접근지, 아니면 element 최대 선언 접근지에 따라 나누는 수 다름  
(front + last) 연산 X ⇒ 복잡한 + 연산, +2 연산으로 해결하니.

### Stacks

- homogeneous items forming a stack w/ one end open.
- LIFO (Last In First Out)
- push adds an item to the top
- pop removes an item from the top

#### Implementing 1 : linked list

class Stack {

```
private final LinkedList llist;
public Stack() {llist = new LinkedList();}
public void push(int x) {llist.insertAtFront(x);}
public int pop() {return llist.deleteFirst();}
public boolean isEmpty() {return llist.isEmpty();}
}
```

#### Implementing 2 : array

class Stack {

```
private final int[] data;
private final int size;
private int top;
public Stack(int s){
    data = new int[s];
    size=s;
    top=-1;
}
public boolean isEmpty() {return top== -1;}
public boolean isFull() {return top == (size-1);}
public void push(int x){if (!isFull()) data[++top] = x;}
public int top() {if (isEmpty()) return -1; else return data[top];}
}
```

### Example of using Stacks

#### ① matching parentheses

pseudo code.

```
if "(" or "{" push to a Stack
else if ")" and tos is "(" pop
else if "}" and tos is "{" pop
if stack is empty return true
else return false.
```

#### ② Recursion

##### Tower of Hanoi

X Stack

class Hanoi{

```
public static void Hanoi(int n, int from, int to){
    if(n<=0) return;
    int aux = 6- from - to;
    Hanoi(n-1, from, aux);
    System.out.println("Move a disk from rod " + from);
    System.out.println("to rod " + to);
    Hanoi(n-1, aux,to);
}
```

# STACKS & QUEUES

RECURSION

Curve of forgetting study planner

1HOUR 1DAY 2DAY 1WEEK 2WEEK 1MONTH 3MONTH

date :

with stack

```

class Context{
    public int n, from,to;
    public int whereToReturn;
    public Context(int n, int f, int t, int w){
        this.n=n, from=f, to=t, whereToReturn=w;
    }
}

public static void Hanoi (int n, int from, int to) {
    Stack s = new Stack(n);
    Context ctx; boolean calldone;
    ctx = new Context (n,from,to,0);
    while (true) {
        calldone = false;
        switch (ctx.whereToReturn) {
            case 0:
                if (ctx.n <=0) {calldone = true;}
                else {
                    s.push (new Context (ctx.n,ctx.from, ctx.to,1));
                    ctx = new Context (ctx.n-1, ctx.from, 6-ctx.from-ctx.to ,0);
                }
                break;
            case 1:
                System.out.println ("Move a disk from rod " + ctx.from + " to rod",
                        ctx.to);
                s.push (new Context (ctx.n,ctx.from,ctx.to,2));
                ctx = new Context (ctx.n-1, 6-ctx.from-ctx.to,ctx.to,0);
                break;
            case 2:
                calldone = true;
                break;
        }
        if (calldone) {
            if (s.isEmpty()) break; else ctx=s.pop();
        }
    }
}

```

③ postfix calculation.

if 이번 content operand push  
else pop 필요한 operands를 & 계산 & 결과 push.  
마지막 결과 pop

④ converting from infix to postfix.

- Stack 외부의 < 는 제일 높은 우선순위, 내부의 < 는 제일 낮은 우선순위를 갖는다

pseudo code

Initialize an empty stack s of operators

for each token x of the expression, from left to right

if x is an operand then output x

else if x is ) then

    while top of stack is not (

        pop and output an operator

        pop ※ output에 있는 괄호삭제

    else

        while top <sup>⑤</sup> of stack has higher or equal precedence than x

            pop and output an operator

            push x

repeat pop and output an operator until s is empty

# SORTING

Curve of forgetting study planner

1HOUR 1DAY 2DAY 1WEEK 2WEEK 1MONTH 3MONTH

date : 2019 . 4 . 17

## Bubble Sort

- 매번 2개씩 비교하여 order가 반대로라면 change

- 1번 시행으로 최대인 원소가 제자리를 찾고 다음엔 그 다음 큰수 이렇게 진행

### pseudo code

do

    처음부터 끝까지 비교해서 sorting property 위반시 flag change,  
    if change로 인해 또 property violated 될 수 있으니 flag = false

    while 루프를 또 돌려 하여 check

while !flag

- time complexity :  $O(n^2)$  worst case     $O(n)$  best case

## Selection Sort

- 최댓값을 찾아서 저를 앞으로 보내기

### pseudo code

array length 만큼 해당 index 전은 sorted. 뒤의 원소 중에서 최대를 찾아 현재 index값과 swap.

- time complexity :  $O(n^2)$  always

## Insertion Sort

- 현 index 앞의 element들은 sorted array이고 현 element를 앞 array의 적절한 곳에 삽입

### pseudo code

element 끝까지 index 중이며 비교하면서 값을 앞으로 테러와

if array값 < element 끄면 break;

X 나가 한 일수 : 값을 비교했을 때 element 위치의 값이 바뀌거나가 element를 다른 변수에 복사 사용

- time complexity :  $O(n^2)$

### Stable Sort

- original ordering이 바뀌지 않는 sorting method.

Ex) Bubble sort, insertion sort, merge sort, radix sort

## Merge Sort

- merging two sorted list

→ 새로운 array에 두 array 최종점부터 비교해서 새로운 array를 만든다.

두 list의 길이가  $l_1, l_2$ 일 때  $O(l_1 + l_2)$

- divide & conquer

### pseudo code

일단 mergeSort 함수를 부르면 sort된 list를 저장할 array 선언 후 실제 sort함수 인연

설계함수

if left >= right 이면 Sort끝남.

mid 계산하고 divide 해서 앞, 뒤 mergesort 함수 호출

한데 배열과 새로운 배열에 값을 넣어두고 유태배열로 값 비교하며 세 list conquer

유래 배열에 세 리스트 값 넣어주기.

- time complexity

domergeSort 함수 :  $O(n)$  worst case ( 한 list의 최대 < 다른 list의 최대 일 때 )  
호출횟수 :  $2^{n-1} \rightarrow (n\text{번 split}, n\text{번의 비교로 terminate되는 호출})$

부) domergeSort 함수 호출시마다 element 개수가 몇이 되면 너무 크게 계산

## Quick Sort

- pivot element를 설정하고 그보다 작은 element끼리 큰 element끼리 모아둔다.

### pseudo code

일단 left >= right 될 때 terminate

pivot을 잡고 이보다 작으면 맨쪽에, 크면 뒤쪽 배치

제일 마지막에 left에 있는 pivot 뒤로 사이로 Swap  $\Rightarrow$  index 저장하고 있겠어야 함

pivot 뒤 리스트 재귀 sorting 진행.

- time complexity : merge sort와 같은 능률  $O(n \log n)$  ( 딱 봤으니 나누어질 때만 )

마지막 sort된 list라면 left를 pivot 설정하면  $O(n^2)$  worst case

↳ 해결 : pivot을 random choice 하면 left element와 Swap.

## Radix Sort

- 두자리 경우 sorting을 한다고 하면 십의자리수 기준 sorting 후 일의자리 향후 sorting (MSD)

▣ 참고 : LSD (일의자리수 기준 sorting 후 십의자리 기준 sorting)

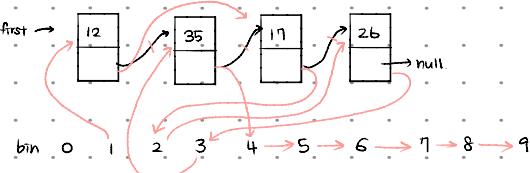
- 테이터 저장 (MSD 저장)

0에서 9까지의 bin이 있고, 각 bin은 십의자리를 나타냄

각 bin이 Node를 가리키는데 십의자리가 0인 마지막 노드는 bin 1을 가리킨다.

insertEnd() 사용

즉 하나의 linked list를 하나의 binary 이용



- time complexity :  $O(d(nr))$  ( d: number of digits, r: radix )

하나의 digit마다 모든 element를 sorting하고 그에 linked list를 combine하므로 (nrd) 시간이

걸리고, digit이 d개 이므로  $O(d(nr))$  이다.

# TREES

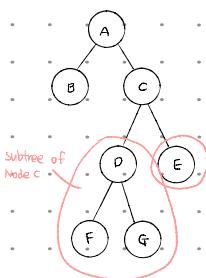
Curve of forgetting study planner

1HOUR 1DAY 2DAY 1WEEK 2WEEK 1MONTH 3MONTH

date : 2019 . 4 . 19

## Trees

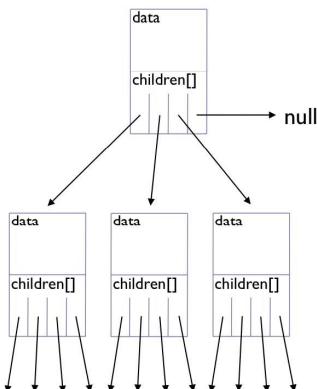
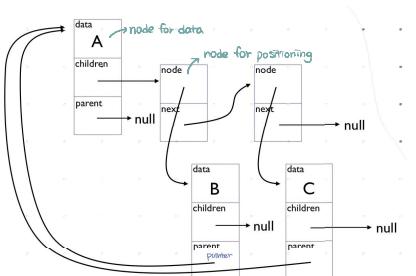
### Terminology



- Nodes : data를 담고 있는 부분
- edges : Node를 잇고 있는 선.
- child & parent : Edge로 이어져 있는 상위는 parent, 하위는 child node.
- (ancestor - parent - child - descendant)
- Sibling : parent Node가 같은 Node들
- leaves Nodes : child Node가 없는 Node ( $\leftrightarrow$  internal nodes)

- A : root Node
- tree consists D, F, G : Subtree rooted at D
- degree : number of the Node itself's children.
- path : unique(simple) line between a pair of nodes
- level : length of the unique path from root to vertex.
- height : maximum value of level of the node.

### Implementation



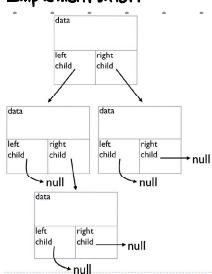
X maximum number of children is specified.

$\rightarrow$  child 없는 경우 null

### Binary Trees

- ordered tree that every node has equal or less than 2 children.
- Child node : left child, right child
- level i  $\rightarrow$  node maximum # :  $2^i$
- height  $\Rightarrow$  maximum node # :  $2^{h-1}$  ( $\rightarrow$  full binary tree)

### Implementation



#### ① pointer 사용

max # of children이 정해져 있으므로 한 노드 당 Children pointer를 두개씩 만들어 사용

### ② array

- root node : index 1
- parent node : index  $\lceil \frac{n}{2} \rceil$
- left child node : index  $2i$
- right child node : index  $2i+1$

### Traversal

- visiting every node of the given tree

preorder traversal : root-left-right

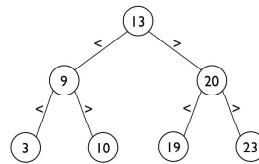
inorder traversal : left-root-right

postorder traversal : left-right-root  $\rightarrow$  postfix를 알 수 있음

## Binary Search Trees

### Binary Search

3 9 10 13 19 20 23



\* 구현시 주의할점

1. delete할 때와 insert할 때 parent노드

찾는 방법이 달라야 함!

2. 만약 right subtree의 min인 delete노드인 경우 예외 처리필요!

Right node인 경우 예외 처리필요!

- binary search tree is either

- an empty tree
- the keys of the left tree are smaller than its of the root.
- the keys of the right tree are bigger than its of the root.
- both subtrees are binary search tree
- key 값은 모두 distinct.

### Operations

#### ① insertion

- i) 원래 insert하고자 하는 값이 이미 있는가?
  - ii) 있다면 자리에 삽입해야 하는가?
- searching 수행
- 삽입 위치 찾으면 pointer 연결

$\rightarrow$  height of the tree

- time complexity : search  $O(h)$  + insert  $O(1) \Rightarrow O(h)$

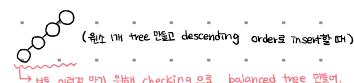
#### ② deletion

- i) delete할 노드가 tree에 있는가?
  - ii) 있다면 위치는 어떤가?
- 1) children 수 = 0인 경우 : 그냥 delete
  - 2) children 수 = 1인 경우 : child를 delete하는 자리에 옮기고 delete
  - 3) children 수 = 2인 경우 : 원쪽 subtree의 max 또는 오른쪽 subtree의 min을 delete 할 노드 자리로 옮기고 연결.
- time complexity : search  $O(h)$  +  $\begin{cases} 1) 2) delete O(1) \\ 3) delete O(h) \end{cases} \Rightarrow O(h)$

### X 외부 관계

- 평균적으로 :  $h = \log_2 n$

- worst case :  $h = n$



→ 이를 체크하는 Checking으로 balanced tree 만들기.

#### ③ Joining two trees

two binary search trees A,B  $\rightarrow$  keys : A < B

$\left[ \begin{array}{c} \text{left subtree} \leq \text{max} \\ \text{right subtree} \leq \text{min} \end{array} \right] \rightarrow \text{rootnode} \text{을 하는 new-tree 생성}$

- time complexity : search  $O(h)$  + joining  $O(1) \Rightarrow O(h)$