DATA STRUCTURE

HAMM

2018147518

Measuring Program Efficiency

Two ways of measuring Program Efficiency

① Empirically (경험적으로)

② Analytically

```
// input: a[0] ... a[n-1]
// output: min

for (i = 0; i < n; i++) {
  // check if a[i] is the min
  isMin = true;
  for (j = 0; j < n; j++) {
    if (a[j] < a[i]) isMin = false;
  }
  if (isMin) min = a[i];
}
```

(n번 loop; n번 addition, (n+1)번 comparison; n번 addition, (n+1)번 comparison; n번 comparison)

n{n+(n+1)+n} + n+(n+1)

→ n번 addition, (n+1)번 comparison, n번 comparison이 n번 진행

& n번 addition, (n+1)번 comparison

```
// input: a[0] ... a[n-1]
// output: min

// a[0] is the tentative min
min = a[0];
for (i = 1; i < n; i++) {
  // see if a[i] is smaller than min-so-far
  if (a[i] < min) min = a[i];
}
```

((n-1)번 addition, n 번 comparison; (n-1) comparison)

(n-1)+(n-1)+n = 3n-2

→ (n-1)번 addition, (n-1)번 comparison, n번 comparison

⇒ different platform (실행조건 차이), detail문제로 meaningful ✕

↳ 실제로 a[i] 호출시 a[0]에서 addition이 일어남.

⇒ a[i]호출 대신 미리 변수를 선언해서 넣어줌으로 해결 가능.

Order of growth

Big-O notation

If f(n) can be bounded by g(n) from above after multiplying an latter by an appropriate constant, f(n) = O(g(n))

For $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$, if there exist constants k and C s.t $f(n) \leq C \cdot g(n)$ for all $n > k$, we say $f(n)=O(g(n))$

For $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$, if there exist constants k and C s.t. $f(n) \geq C \cdot g(n)$ for all $n > k$, we say $f(n)=\Omega(g(n))$

If $f(n)=O(g(n))$ and $f(n)=\Omega(g(n))$, we say $f(n)=\Theta(g(n))$

→ input의 size뿐만 아니라 input의 contents도 running time에 영향.

- usually focus on the worst case (= worst-case complexity)

```
// preprocessing
// do nothing

// query (x, y) comes

answer = 0;
for (i = x; i <= y; i++) {
  answer += a[i];
}
```

```
// preprocessing
for (i = 0; i < n; i++) {
  sum[i][i] = a[i];
  for (j = i+1; j < n; j++) {
    sum[i][j] = sum[i][j-1] + a[j];
  }
}
// now for all x ≤ y, sum[x][y] holds a[x]+…+a[y]

// query (x, y) comes
answer = sum[x][y];
```

q개의 queries가 왔다고 하자.

	첫방법	둘째 방법
time complexity	O(nq)	O(n+q)
space complexity	O(1)	O(n²)

⇒ n와 p에 따라서 더 이득되는 방법이 달라짐.

실제적으로 time complexity 말고도 고려할 것들이 많지만 주로 problem은 time complexity에 의함.

- 또, overflow가 일어날 가능성도 있음.

JAVA FOR C++ PROGRAMMERS

date : 2019 . 04 . 13

Basic Java Structures

Hello world!

```
Class Hello {
    public static void main (String[] args)
    { System.out.println("Hello, world!"); }
```

- Class 밖에 함수 선언 X
- Class 이름과 파일 이름이 같아야 함

Primitives and objects

- Object variables work like "pointers"

```
class MyInt{
    private int val;
    public int getVal() { return val; }
    public void setVal(int x) { val = x; }
}
```

선언시 : `MyInt a = new MyInt();`

```
MyInt a = new MyInt(), b = new MyInt();
a.setVal(3);  b.setVal(3);
if(a==b) { System.out.println("true"); }
else { System.out.println("false"); }
```

⇒ 출력 : false

- == 기호는 같은 object를 가리키고 있는지 확인한다.

Strings

```
String a="abcdeabcde";
```

- a.length() : String의 길이 return.
- a.indexOf("bc") : first accurance return.
- a.substring(2,4) : index 2이상 4미만의 String 리턴.

Arrays

```
int[] a;    a = new int[10];
```

✲ in JAVA, there is no delete operation.

FILE I/O

```
import java.io.*;
Class Hello {
    public static void main (String[] args){
        String s; int i, x;
        try {
            BufferedReader rd = new BufferedReader(new FileReader("input.txt"));
            BufferedWriter wr = new BufferedWriter(new FileWriter("output.txt"));
            :
            rd.close(); wr.close();
        }
        catch (Exception e) {
            System.out.println("ERROR");
        }
    }
}
```

ARRAYS

date : 2019 . 04 . 14

Array as a Data Structure

- Array : a set of "homogeneous" data
- Indexed by consecutive integers
- O(1) time access

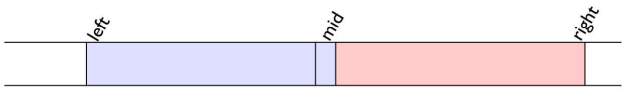
☆ the size needs to be specified when creation.

Binary Search

```
left = 0;
right = n - 1;
while (left < right) {
  mid = (left + right) / 2;
  if (a[mid] < x) left = mid + 1; else right = mid;
}
if ((left == right) && (a[left] == x)) {
  found = true;
  foundpos = left;
} else found = false;
```

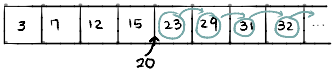
a[mid]도 x 아닌 무한루프에 빠질 수 있음. ex) left = 6, right = 7, a[mid] = x인 경우

n=0일때를 걸러내기 위해 필요

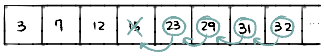


- $O(\log_2 n) = O(\log n)$으로 표시.
- 하지만 Sorted Array로 계속 유지 필요.

∴ Insertion : O(n)



∴ deletion : O(n)



LINKED LISTS

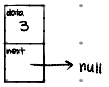
Curve of forgetting study planner

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

date : 2019 . 04 . 14

Linked Lists

Node : data + next pointer to next node



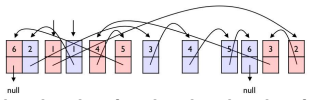
- 하나의 object로 취급

Linked List

access : O(n) ⇒ 첫 포인터부터 순차적인 접근

insertion : O(1) ⇒ 삽입 위치를 미리 알고 있을 때

deletion : O(1)




```
class Node {
  public int data; public Node next;
}
class LinkedList {
  public Node first, last;
  public LinkedList() {
    first = null; }
  public boolean isEmpty() {
    return (first == null); }
  public void InsertAtFront(int x){
    Node newnode = new Node();
    newnode.data = x;
    newnode.next = first;
    first = newnode;
    if(last == null) last = newnode; }
  public int DeleteFirst() {
    int ret;
    if (first == null) return -1; // if the list is empty
    ret = first.data;
    first = first.next;
    if (first = null) last = null; // 원래 list 원소가 1개 였을 경우
    return ret; } // JAVA는 자동 garbage 기능이 있으므로 explicitly handle X
  public void DisplayAll(){
    Node cur;
    for(cur = first; cur != null; cur = cur.next) {
      System.out.println(cur.data);}}
  public void InsertAfter(Node n, int x) {
    Node newnode;
    if(n == null) return;
    newnode = new Node();
    newnode.data = x;
    newnode.next = n.next;
    n.next = newnode;
    if(newnode.next == null) last = newnode;
  public void InsertAtEnd(int x){
    Node newnode;
    newnode = new Node();
    newnode.data = x;
    newnode.next = null;
    if(last == null) { first = last = newnode; }
    else { last.next = newnode; last = newnode }
}
```

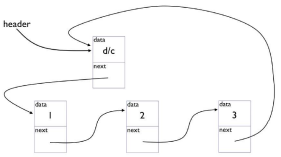
Iterator

↱ LinkedList 클래스에 iterator 선언부 추가.

```
Class LinkedList {
  public Iterator getIterator() {
    return new Iterator(first); }

Class Iterator {
  private Node cur, prev;
  private LinkedList llist;
  public Iterator(Node first, LinkedList caller) {
    llist = caller; cur = first; prev = null; }
  public boolean atEnd() {
    return (cur == null); }
  public int getData() {
    if(atEnd()) { return 0; }
    return cur.data; }
  public void next() {
    if(!atEnd()) { prev = cur; cur = cur.next; }}
  public void InsertAfter(int x) {
    llist.InsertAfter(x); }
  public int DeleteCurrent() {
    int ret;
    if(cur == null) { return -1; }
    ret = cur.data; cur = cur.next;
    if(prev == null) { llist.first = cur; } // 만일 cur가 first 였다면
    else { prev.next = cur; }
    if(cur == null) llist.last = prev; // 만약 cur가 last였다면
    return ret; }
```

Circular linked lists



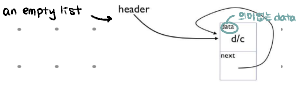
- no inherent "first" & "last" item

ex) Round-robin scheduling

- code becomes simpler

- Header node : never deleted

an empty list




```
Class CircularLinkedList {
  private final Node header;
  public CircularLinkedList() {
    Node newnode = new Node();
    newnode.next = newnode;
    header = newnode; }
  public boolean isEmpty() {
    return (header.next == header); }
  public CircularIterator getIterator(){
    return new CircularIterator(header); }
Class CircularIterator{
  private Node header, prev, cur;
  public CircularIterator(Node h) {
    header = h; prev = h; cur = h.next; }
  public boolean atEnd() {
    return (cur == header); }
  public int getData() {
    if(atEnd()) { return -1; }
    return cur.data; }
  public void next() {
    prev = cur; cur = cur.next; }
```

...to be continued...