7주차 예비보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20191619 이름: 이동석

**1. 자료구조**

**1-1. Linked List**

링크드 리스트를 이용해 랭킹 시스템을 구현할 수 있고 다음과 같은 자료구조를 생각 할 수 있다.

typedef struct node {

int data;

char name[NAMELEN];

struct node \*link;

} NODE;

이 자료구조에 대해서 랭킹의 삽입과 삭제를 구현하기 위한 pseudo code는 다음과 같다.

Insert(x) {

NODE\* nNode = (NODE\*)malloc(sizeof(NODE))

nNode -> data = x

nNode -> link = NULL

NODE \* curr = head

If x > curr -> data

nNODE -> link = head

head = nNODE

else

while curr ->link != NULL && x <= curr->link->data

curr = curr -> link

nNODE->link = curr -> link

curr -> link = nNODE -> link

endwhile

endif

}

이때, 시간복잡도는 정렬되어 있는 최악의 경우 O(n) , 최선의 경우 O(1)의 복잡도를 가진다. 공간복잡도는 원소의 개수 O(n) 이다.

Delete(x) {

/\* Delete xth node \*/

NODE \*curr = head

If head == NULL

return

endif

NODE \* curr = head;

for i=1 to x-1

curr = curr -> link

endfor

NODE \*temp = curr -> link

curr -> link = temp -> link

free temp

}

이때, 시간복잡도는 x번째 원소를 삭제하므로, O(x) 이다. 공간복잡도 역시 원소 x까지 가므로 O(x) 라 할 수 있다.

**1-2. Binary Tree**

이진트리의 자료구조는 다음과 같이 생각 할 수 있다.

typedef struct node{

int data;

char name[NAMELEN];

struct node \*left;

struct node \*right;

}NODE;

이 자료구조에 대해서 랭킹의 삽입과 삭제를 구현하기 위한 pseudo code는 다음과 같다.

Insert(x) {

NODE \*newNODE=(NODE \*)malloc(sizeof(NODE));

newNODE -> data = x

newNODE -> left = NULL

newNODE -> right= NULL

if root == NULL

root = newNODE

else

NODE \*curr = root

while true

if curr -> data >= newNODE -> data

if curr -> right == NULL

curr->right = newNODE

break

endif

else curr = curr -> right

endif

else if curr -> data < newNODE -> data

if curr -> left == NULL

curr -> left = newNODE

break

endif

else curr = curr -> left;

endif

endfor

endif

}

일반적으로 이진트리의 시간복잡도는 트리의 깊이와 연관되어있다. 깊이는 실행마다 1/2 된다 할 수 있으므로 O(log n) 이다. 물론 최악의 경우, 한쪽방향으로 몰려 있을 수 있다. 이때는, O(n) 이 된다. 공간복잡도는 배열이나 링크드 리스트와 마찬가지로 O(n) 이다.

삭제하는 알고리즘의 경우 삽입에 비해 까다롭다. 총 3가지의 경우로 나누어서 생각해야한다. 먼저, 삭제할 노드가 단말노드일 경우, 두번째로 하나의 서브트리만을 가지고 있는 경우, 마지막으로, 두개의 서브트리를 가지고 있는 경우로 나뉜다.

Delete(NODE \*root, int data){

If root == NULL

return root

if data < root->data

root->left = Delete(root->left, data)

else if data > root -> data

root->right = Delete(root->right, data)

else

if root -> left == NULL

NODE\* temp = root -> right

Free root

return temp

else if root -> right == NULL

NODE\* temp = root->left

Free root

return temp

endif

NODE\* temp = node closet to the node to be deleted

root -> data = temp -> data

root -> right = Delete(root->right, temp->data)

return root

}

이 경우 역시 삽입의 경우와 같다.

각각의 경우에 대해 랭킹을 x~y위 까지의 정보를 얻는 방법은 다음과 같다.

**Linked List**

void print\_list(){

NODE \*curr = head

for i=1 to x-1

curr = curr -> link;

endfor

for i=x to y-1

print the data

curr = curr -> link

endfor

print the data

}

이 경우 시간복잡도와 공간복잡도는 모두 y번째 원소까지 체크해야하므로 , O(y) 라 할 수 있다.

**Binary Tree**

void print\_list(NODE \*root,int X,int Y,int \*check)

/\* check=1 in this function \*/

{

if root == NULL return

if root != NULL

if root -> left !=NULL

print\_list(root->left,X,Y,check)

if \*check >= X && \*check <= Y

print the data

(\*check)++

print\_list(root->right,X,Y,check)

}

}

이 경우 시간복잡도와 공간복잡도는 마찬가지로 처음부터 Y번째 원소까지 확인하므로, O(Y) 이다.