9주차 예비보고서

전공: 컴퓨터공학과 학년: 4학년 학번: 20192135 이름: 윤영인

**1.**

1) 연결 리스트

연결리스트는 노드들을 포인터로 연결한 구조로 이루어져 있다. 각 노드에는 사용자의 이름, 점수, 이전 노드를 가리킬 prev 포인터와 다음 노드를 가리킬 next 포인터를 저장한다. 따라서 next 포인터를 따라 노드를 탐색하면서 연결된 노드의 점수들이 내림차순으로 정렬될 수 있도록 적절한 위치를 찾아 새로운 노드를 삽입한다. 노드를 삭제할 경우, 해당 노드를 prev 혹은 next 포인터를 이동하면서 찾아서 삭제하고, 삭제한 노드의 prev 노드와 next 노드를 서로 이어준다.

2) Binary Search Tree

Binary Search Tree의 노드는 왼쪽에 해당 노드보다 작은 노드를, 오른쪽엔 해당 노드보다 큰 노드를 연결한다. 따라서 연결리스트와 마찬가지로 노드에 사용자의 정보와 이전 노드, 다음 노드를 가리킬 포인터를 저장하고 트리를 돌면서 노드의 점수를 비교하면서 작으면 왼쪽에, 크면 오른쪽에 삽입하도록 한다. 노드를 삭제하는 것도 노드들의 점수를 비교하면서 삭제하고자 하는 노드를 찾고 삭제한다.

**2.**

1) 연결리스트

void InsertNode(삽입할 노드) {

if 연결리스트가 비어있으면

then

연결리스트의 head = 삽입할 노드

else if 비어있지 않으면

then

현재 노드 = 연결리스트 -> head

while(현재 노드의 다음 노드가 존재하면) {

현재 노드 = 현재 노드 -> next

}

삽입할 노드 -> prev = 현재 노드

현재 노드 -> next = 삽입할 노드

}

연결리스트의 head 노드부터 마지막 노드까지 탐색하므로 시간 복잡도는 O(노드의 개수) = O(n)이다. 공간 복잡도는 노드의 개수만큼 공간이 필요하므로 O(n)이다.

void deleteNode(삭제할 노드의 순위) {

int count = 0

if 연결리스트가 비어있으면

then

return

else

then

현재 노드 = 현재 노드->next

while (현재 노드의 다음 노드가 존재하면) {

현재 노드 = 현재 노드->next

count++;

if(count == 삭제할 노드의 순위) {

if (현재 노드 == 연결리스트->head) {

임시 노드 = 현재 노드

현재 노드->next->prev = NULL

연결리스트->head = 현재 노드->next

free(임시노드)

}

else {

임시 노드 = 현재 노드

현재 노드->prev->next = 현재 노드->next

현재 노드->next->prev = 현재 노드->prev

free(임시노드)

}

if (연결리스트가 비어있으면)

then

free(연결리스트)

}

연결리스트의 head 노드부터 삭제할 노드를 찾아 탐색한다. 최악의 경우엔 연결리스트의 마지막 노드까지 탐색해야하므로 시간 복잡도는 O(n)이고, 공간 복잡도는 O(n)이다.

2) Binary Search Tree

void Insert(서치 트리의 루트, 삽입할 노드) {

pNode = NULL

cNode = 루트

while (cNode) {

pNode = cNode

if cNode의 점수>삽입할 노드의 점수

then

cNode = cNode의 왼쪽 노드

else

then

cNode = cNode의 오른쪽 노드

}

if pNode가 존재하면 {

if pNode의 점수>삽입할 노드의 점수

then

pNode의 왼쪽에 삽입할 노드 삽입

else

then

pNode의 오른쪽에 삽입할 노드 삽입

}

else then

서치 루트의 트리 = 삽입할 노드

}

시간 복잡도는 트리의 최대 높이만큼 이므로 O(h)이고, 최악의 경우 노드의 개수만큼 트리를 탐색하므로 O(n)이다. 만약 완벽하게 구현된 이진 트리일 경우 시간 복잡도는 O(logN)이다. 공간 복잡도는 모든 변수가 상수 변수이므로 O(1)이다.

void deleteNode(서치 트리의 루트, 삭제할 노드) {

cNode = 서치 트리의 루트

while (cNode != NULL && cNode의 점수 != 삭제할 노드의 점수) {

pNode = cNode

if 삭제할 노드의 점수>cNode의 점수

then

cNode = cNode의 오른쪽 노드

else then

cNode = cNode의 왼쪽 노드

}

if cNode != NULL

then return NULL

dNode = cNode

if dNode->left가 존재하고, d->right가 존재하지 않을 경우 // 삭제할 노드가 자식을 갖지 않을 경우

then

if pNode->left가 존재하지 않으면

then

dNode = pNode의 왼쪽 자식을 지움

else then

dNode = pNode의 오른쪽 자식을 지움

else if dNode->left가 존재하거나 dNode의 right가 존재할 경우 // 삭제할 노드가 하나의 자식 노드를 가질 경우

then

if dNode->left가 존재하면

then

dcNode = dNode의 왼쪽 노드

else then

dcNode = dNode의 오른쪽 노드

if pNode->left == cNode

then

pNode->left = dcNode

else then

pNode->right = dcNode

else

eNode = dNode의 오른쪽 노드

epNode = dNode

while eNode의 왼쪽 노드가 존재하는 동안 {

epNode = eNode

eNode = eNode의 왼쪽 노드

}

deleteData = dNode의 점수

if epNode의 왼쪽 노드 == 삭제할 노드

then

epNode의 왼쪽 노드 = eNode->right

else then

epNode의 오른쪽 노드 = eNode->right

dNode = eNode

if pRoot의 오른쪽 노드 != 이진 서치 트리의 루트

then

이진 서치 트리의 노드 = pRoot의 오른쪽 노드

free(pRoot)

return dNode

시간 복잡도는 트리의 최대 높이만큼 이므로 O(h)이고, 최악의 경우 노드의 개수만큼 트리를 탐색하므로 O(n)이다. 만약 완벽하게 구현된 이진 트리일 경우 시간 복잡도는 O(logN)이다. 공간 복잡도는 모든 변수가 상수 변수이므로 O(1)이다.

**3.**

1) 이진 서치 트리

void rank(x, y, count, 이진 서치 트리의 루트) {

if x < 1

then

x = 1

if y > count

then

y = count

if x > y 이거나 count == 0 이거나 x > count

then

에러 메시지 출력

현재 노드 = 이진 서치 트리의 루트

while (1) {

if 현재 노드가 존재하지 않으면

then break

스택에 현재 노드 push

현재 노드 = 현재 노드의 왼쪽 노드

for (j = 1 ~ y) {

현재 노드 = 스택을 pop

if x <= j

then

현재 노드의 정보를 출력

현재 노드 = 현재 노드의 오른쪽 노드

while (1) {

if 현재 노드가 존재하지 않으면 break

스택에 현재 노드 push

현재 노드 = 현재 노드의 왼쪽 노드

}

}

}

x~y 순위의 랭킹 정보를 얻기 위해 y번째 노드까지 순회하므로 시간 복잡도는 O(y) = O(n)이다. 공간 복잡도는 노드를 출력하기 위해 사용되는 변수가 상수 변수이므로 O(1)이다.

2) 연결리스트

void rank(count , x, y) {

if x < 1

then x = 1

if y > count

then y = count

if x > y || count == 0 || x > count

then

에러 메시지 출력

else

현재 노드 = 연결리스트->head

for (i = 1 ~ x) {

cur = cur->next

}

for(j = 1 ~ y) {

현재 노드의 정보 출력

}

}

x~y 순위의 랭킹 정보를 얻기 위해 y번째 노드까지 순회하므로 시간 복잡도는 O(y) = O(n)이다. 공간 복잡도는 노드를 출력하기 위해 상수 변수를 사용하므로 O(1)이다.