7주차 예비보고서

전공: 수학/컴퓨터공학 학년: 3학년 학번: 20181288 이름: 윤성호

1. <linkedList>

linkedList를 이용하여 각 노드에 이름과 점수정보를 입력하여 연결시킨다. 이때, 연결은 점수가 가장 높은 사람을 가장 앞에 놓고, 점점 점수가 낮아지도록 연결을 시킨다. 이렇게 되면 탐색이나, 삭제 및 삽입과정이 매우 용이해진다.

<heap>

Max heap로 사람들의 점수정보를 저장해놓는다. Heap자료구조는 linkedList 자료구조보다 탐색에 유리하다. linkedList의 경우에는 특정 점수를 검색하기 위해서는 가장 앞에서부터 차례로 탐색해 나가야 한다. 그러나 heap의 경우에는 트리의 두갈래 중 하나로 쭉쭉 들어가므로 O(logN)만큼의 시간만에 탐색이 가능해진다.

1. <linkedList>

linkedList에 자료를 삽입하기 위해서는 먼저 현재 점수가 가장 높은지 아닌지 판별해야 한다. 가장 높다면 head node로 위치시키고, 가장 높지 않으면 중간에 삽입시켜야 하기 때문이다. 먼저, 가장 점수가 높다면 새로운 노드의 link를 head node의 link로 주고, head를 새로운 노드로 주면 된다. 만약 점수가 가장 높지는 않아서 중간에 삽입해야 한다면 새로운 노드의 link를 바로 전 노드의 link로 주고, 바로 전 노드의 link를 새로운 노드로 향하도록 해주면 된다. 이는 잘 알려진 linkedList에서 노드를 삽입하는 과정이고, 시간복잡도는 최악의 상황에서는 가장 끝에 삽입되는 것이므로 linkedList의 길이에 영향을 받는다. linkedList의 길이를 len이라고 했을때 시간복잡도는 O(len)가 된다. 공간복잡도는 새로운 노드 하나를 생성하므로 그정도의 메모리가 필요하다. 따라서 O(1)이라고 이야기할 수 있다. 글로벌 변수로 사용한 linkedList를 탐색하는 과정도 있으므로, 이를 위한 공간복잡도는 O(len)이다.

자료를 삭제하기 위해서는 현재 삭제하는 노드가 head node인지 아닌지 구분해야 한다. 먼저, head node라면, temp 라는 이름으로 빈 노드를 하나 생성하여, temp에 head를 저장하주고, head->link = head라고 해준다. 그 이후 삭제된 노드는 메모리를 free해주어야 한다. 이를 위해서 Temp변수를 도입한 것이다. 만약 head node가 아니라면, 현재 노드를 temp에 담아주고 이전 노드의 link를 현재 노드의 링크에 연결 시켜준 후, 현재 노드의 메모리를 free해준다. 이 또한 삽입과정과 마찬가지로 시간보잡도는 O(len)이고, 공간복잡도는 temp 노드를 생성하므로 O(1)이다. 여기서 temp라는 이름으로 글로벌 변수인 linkedList를 탐색하므로, 공간복잡도는 O(len)이다.

아래는 linkedList 자료구조의 삽입, 삭제 과정의 pseudo code이다.

<삽입>

IF NEW NODE SHOULD BE FIRST NODE

NEW NODE’s LINK = HEAD

HEAD = NEW NODE

ELSE

NEW NODE’s LINK = PREVIOUS NODE’s LINK

PREVIOUS NODE’s LINK = NEW NODE

<삭제>

IF FIRST NODE SHOULD BE DELETED

TEMP = HEAD

HEAD = HEAD’s LINK

Free(TEMP)

ELSE

TEMP = CURR NODE

PREVIOUS’s LINK = CURR NODE’s LINK

Free(TEMP)

<heap>

Heap은 항상 complete tree이다. 따라서 새로운 원소를 삽입할때는 일단 가장 끝에 삽입을 한 후 알맞게 올라오는 과정을 거치게 된다. 부모노드와 현재 노드와의 점수비교를 하며 현재 노드가 더 크면 교체하고, 크지 않다면 교체하지 않고 삽입과정을 종료한다.

삭제과정은 트리를 타고 내려가면서 검색한 점수를 찾은 후, 그 점수를 가장 마지막 노드와 교체한 후 현재 노드를 삭제한다. 그 다음, 교체된 가장 마지막 노드는 트리를 타고 내려가며 올바른 위치에 위치하게 된다. 삽입과정과 삭제과정 모두 트리를 타고 올라가거나 내려가는 과정이 있는데, 이는 한 level을 이동할때마다 절반을 탐색하지 않아도 된다. 즉, 전체 노드의 개수를 N이라고 하면, O(logN)의 시간복잡도가 걸린다. 공간복잡도는 삽입과정은 빈 노드를 생성하므로 O(1)이지만 삭제과정은 빈 노드를 생성하지 않으므로 O(0)이다. 그리고 글로벌 변수로 힙이 생성되어 있고, 이를 탐색에 사용하므로 힙의 공간복잡도는 O(N)이다.

아래는 heap 자료구조의 삽입과 삭제과정의 pseudo code이다.

<삽입>

SWAP newNODE, lastNODE

While (1)

IF newNODE’s score > parentNODE’s score

SWAP newNODE, parentNODE

ELSE

Break

<삭제>

SEARCH THE INPUT SCORE

IF NOT FOUND

Printf(“Error : Input score not found)

ELSE

SWAP inputNODE, lastNODE

Free(inputNODE);

While (1)

IF lastNODE’s score < childNODE’s score

SWAP lastNODE, childNODE

ELSE

Break

1. <linkedList>

Count 변수를 하나 두고, head노드부터 차례로 찾아가며 지나갈때마다 count변수를 1증가시킨다. Count변수가 x, y 사이일때, 그 노드에 저장되어있는 점수와 이름을 출력해주는 형태로 구현할 수 있다. 이때 시간복잡도는 최악으로 걸릴때 모든 원소를 다 출력해야 하므로 linkedList의 길이를 len이라고 했을때 O(len)이라고 할 수 있다. 공간복잡도는 새로운 노드를 하나 만들어서 그 노드를 이동시키므로 O(1)이라고 할 수 있다. 탐색과정에서 사용되는 LinkedList의 공간복잡도는 O(len)이다.

아래는 linkedList 자료구조를 이용하여 정보 출력하는 pseudo code이다.

COUNT = 0

CURR NODE = HEAD NODE

for \_ in range(COUNT)

CURR NODE = CURR NODE’s link

While (CURR NODE IS NOT NULL)

IF (COUNT < Y) PRINT NAME and SCORE

ELSE break

<heap>

현재 저장되어있는 heap을 복사한다. 그리고 그 복사된 heap에서 매번 최댓값을 꺼내가며 linkedList 와 마찬가지로 count변수를 두고, 꺼낼때마다 count변수를 1증가시킨다. Count 변수가 x, y 사이일때 노드의 이름과 점수를 출력하게 된다. 이때 시간복잡도는 모두 출력되게 하는 상황으로 현재 heap의 크기가 len이라고 하면 O(len)이 된다. 또한, 공간복잡도는 힙 전체를 복사하는 과정이 필요하므로 O(len \* sizeof(NODE))가 된다.

아래는 heap자료구조를 이용하여 정보를 출력하는 pseudo code이다.

COPY HEAP TO TEMPHEAP

for \_ in range(X)

POP MAXIMUM NODE FROM TEMPHEAP

While (TEMPHEAP IS NOT EMPTY)

IF (COUNT < Y) {

PRINT NAME and SCORE

POP MAXIMUM NODE FROM TEMPHEAP

}

ELSE break