테트리스 2주차 예비보고서

전공: 철학과 학년: 3학년 학번: 20180032 이름: 남기동

1. 2주차 실습에 구현하는 랭킹 시스템을 위한 자료구조 2가지

1) 스택을 통한 구현

스택을 통해서 랭킹 시스템을 구현할 수 있을 것 같다. 그러나 스택의 구조 상 add와 delete 하려는 item이 스택의 중간에 있으면 add와 delete를 하기가 용이하지 않다. 그러므로 temp라는 하나의 스택을 더 만들어서 활용한다면 add와 delete를 적절하게 수행할 수 있다.

2) 연결리스트를 통한 구현

자료에서 활용하고 있는 연결리스트를 활용하여서도 랭킹 시스템을 구현할 수 있다. 스택보다는 각 노드들의 정보에서 link를 통해 노드를 오가면서 add와 delete를 하거나 search를 하면 보다 용이하리라 예상된다.

2. 각 자료구조에서 랭킹 삽입 및 삭제를 위한 pseudo code와 시간 공간 복잡도

1) 스택으로 구현하는 경우

```
Typedef struct {
Char name[30]
Int score
} element;
Element stack [MAX_STACK_SIZE];
Element temp {MAX_STACK_SZIE};

Add( element item)
```

```
For i: 0 to n
    If ( Stack[i].score < item.score )
    Push(stack, item)
    While ( temp empty )
            Push(stack, pop(temp))
    Else push(temp, pop(stack))
```

스택으로 add함수를 구현한 것에는 stack과 temp라는 이름의 두 스택이 활용된다. Stack이 값을 넣는 메인 스택이고 temp는 add를 하기 위해 스택에 쌓인 값들을 임시로 넣어두기 위해 활용한다. Add 함수의 경우 worst case를 고려한다면 stack의 맨 아래까지 조사를 하고 원하는 item을 넣은 후 temp에 stack에 쌓여 있던 모든 element를 다시 stack에 쌓아야 하기 때문에 stack에 n개의 element가 있다고 가정할 때 O(n^2)의 시간 복잡도를 갖는다.

stack과 temp라는 스택이 활용되기는 하지만 실질적으로 값이 차 있던 것은 stack이고 stack에서 pop한 것들이 temp에 임시로 들어갔다가 나오는 구조이다. 따라서 stack에 n개의 정보가 있다고 가정하면 O(n)의 공간 복잡도를 갖는다.

```
Delete (element item)
For i:0 to n
    If ( Stack[i].score == item.score )
    Pop(stack)
    While ( temp empty )
            Push(stack, pop(temp))
    Else push(temp, pop(stack))
```

delete함수도 add함수와 유사하게 stack과 temp를 활용한다. 찾으려는 item을 조사하기 위해 stack의 정보를 찾고 해당 값을 찾으면 pop을 한 뒤 그 동안 temp로 보내놨던 stack의 정보들을 다시 stack에 쌓는 구조를 갖는다. 따라서 시간 복잡도는 stack에 n개의 정보가 있다고 가정할 때

O(n^2)의 구조를 갖는다.

공간 복잡도도 add함수와 유사하게 stack의 n개의 정보가 저장되는 것이 핵심이기 때문에 O(n)의 공간 복잡도를 갖는다.

2) 연결리스트로 구현한 경우

```
typedef struct a* Node_pointer;
typedef struct a {
        char name[NAMELEN];
        int score;
        Node_pointer link;
}Node;
Node_pointer head = NULL;

Add(element item)
Node_pointer ptr;
Node_pointer trail;
for (ptr = head; ptr != NULL; ptr = ptr->link) {
        if (item.score > ptr->score) break;
        trail = ptr;
        }
trail->link = temp;
temp->link = ptr;
```

add함수의 시간 복잡도는 O(n)이다. 헤드가 가리키는 노드부터 시작하여 ptr을 이동시켜가면서 원하는 값을 찾으면 그곳에 item의 정보를 넣은 새로운 노드를 삽입하고 이전 노드와 다음 노드와의 링크를 변경하기 때문이다. 즉, worst case의 경우에도 연결 리스트에 n개의 노드가 있다고

가정했을 때 마지막 노드까지 이동하기 때문에 시간 복잡도는 O(n)이 되는 것이다.

공간 복잡도는 n개의 노드 정보를 저장해야 하고 나머지 값들은 상수이기 때문에 O(n)이다.

```
Delete(element item)
Node_pointer ptr;
Node_pointer trail;
for (ptr = head; ptr != NULL; ptr = ptr->link) {
        if (item.score == ptr->score) break;
        trail = ptr;
        }
trail->link = ptr->link;
free(ptr)
```

delete함수도 헤드부터 시작하여 ptr를 활용하여 노드들을 탐색하게 된다. 이 때 원하는 item의 score와 같은 값을 갖고 있는 노드를 찾으면 그 노드의 이전 노드를 삭제할 노드가 가리키는 노드를 연결시키고 삭제할 노드를 free함수를 통해 메모리 반환을 하여 값을 삭제하면 된다.

시간 복잡도는 n개의 노드가 있다고 가정할 때 O(n)이며 공간 복잡도도 나머지는 상수이고 n개의 노드 정보를 저장해야 하기 때문에 O(n)이다.

3. 정렬된 랭킹의 정보를 얻기 위한 pseudo code와 시간 공간 복잡도

1) 스택을 통한 구현

```
Search ( int x, int y )
If( x <= y ) {
For i:x to y
        Pop(stack)
```

Pop한 값을 출력

}

Else 오류 메시지 출력

이미 add와 delete함수에서 정렬을 해 놓은 상태이고 stack은 결국 배열이기 때문에 index를 이용하여 찾으려는 해당 값들에 바로 접근할 수 있다. 시간 복잡도는 x와 y가 처음과 마지막을 가리켜서 총 n개의 정보를 출력하게 하는 경우 for loop을 통해 stack의 n개의 정보를 탐색하고 출력해야 하기 때문에 O(n)의 시간 복잡도를 갖는다. 공간 복잡도는 stack의 정보만을 갖고 있으면 되기 때문에 O(n)이다.

2) 연결 리스트를 통한 구현

Search (int x, int y)

Node_pointer ptr;

Node_pointer trail;

If(x <= y) {

For(ptr = head, i = 1; i == x; ptr = ptr -> link, i++)

For(j = i ; j <= i+y ; ptr = ptr -> link, j ++)

Ptr의 값 출력

}

Else 오류 메시지 출력

search함수는 일단 x와 y 값을 입력 받아와서 x가 y보다 작거나 같은, 즉 적합한 정보일 때는 해당하는 랭킹의 값들을 출력하고 그렇지 않은 경우에는 오류 메시지를 출력하는 구조를 갖는다. 시간 복잡도는 n개의 노드가 있다고 할 때 n개의 노드를 탐색해야 하기 때문에 O(n)이며 공간 복잡도도 다른 값들은 모두 상수이고 n개의 노드를 저장하기 위한 값을 고려해야 하기 때문에 O(n)이다.