9주차 예비보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20231515 이름: 김다은

1.

1-1. Linked List

typedef struct \_Node {

    char name[NAMELEN];

    int score;

    struct \_Node \*link;

} Node;

Node \*head = NULL;

Node \*tail = NULL;

위와 같은 Node 구조체를 만들고 사용자명과 점수를 저장한 후, 각 node를 link를 통해 연결하여 Linked List 형태로 정보를 저장할 수 있을 것이다. 또한 랭킹을 삽입할 때, linked list의 node의 score와 대소 비교하여 정렬된 형태로 삽입한다.

1-2. Array

typedef struct \_Node {

    char name[NAMELEN];

    int score;

} Node;

Node array[MAX\_ARRAY\_SIZE];

위와 같이 Node 배열을 만들어 랭킹을 관리하는 것도 가능하다. 또한 새로운 ranking 정보를 삽입할 때마다 배열이 정렬되게 유지되도록 구현한다.

2.

설명은 주석으로 대체한다.

2-1 Linked List로 구현한 경우의 pseudo code

<삽입>

    Node \*new = (Node\*)malloc(sizeof(Node));

    Node \*pre = NULL;

    Node \*ptr = head;

    strcpy(new->name, str);

    new->score = score;

    new->link = NULL;

    if(head == NULL) { *// linked list의 노드가 0개인 경우*

        head = new;

        tail = new;

    }

    else {

*// 대소비교 후 삽입*

        for(; ptr!=NULL; pre=ptr, ptr=ptr->link) {

            if(ptr->score < score) {

                new->link = pre->link;

                pre->link = new;

                break;

            }

        }

        if(pre == NULL) { *// linked list의 맨 첫번째 노드로 삽입되는 경우*

            new->link = head;

            head = new;

        }

        if(ptr == NULL) { *// linked list의 맨 마지막 노드로 삽입되는 경우*

            pre->link = new;

            tail = new;

        }

    }

    score\_number++; *// 랭킹의 개수(linked list의 Node 개수)*

시간복잡도: 최악의 경우 전체 linked list를 모두 돌아야 한다. 따라서 linked list의 원소 수를 n이라고 할 때, O(n)이다.

공간복잡도: linked list의 원소 수만큼 메모리를 할당하게 되므로 O(n)이다.

<삭제>

Node \*ptr = head;

for(; ptr->link!=NULL; ptr=ptr->link) {

}

free(ptr);

score\_number--;

linked list의 맨 끝까지 이동한 뒤, 마지막 노드인 ptr를 free해준다.

시간복잡도: 전체 linked list를 모두 돌아야 한다. 따라서 linked list의 원소 수를 n이라고 할 때, O(n)이다.

공간복잡도: linked list의 원소 수만큼 메모리를 할당하게 되므로 O(n)이다.

2-2. Array로 구현한 경우의 pseudo code

<삽입>

    for(int i=0; i<score\_number; i++) {

        if(array[i].score < score) {

            for(int j=score\_number; j>i; j--) {

                strcpy(array[j].name, array[j-1].name);

                array[j].score = array[j-1].score;

            }

            strcpy(array[i].name, str);

            array[i].score = score;

            break;

        }

    }

score\_number++;

시간복잡도: 최악의 경우 전체 Array를 모두 돌아야 한다. 따라서 Array의 원소 수를 n이라고 할 때, O(n)이다.

공간복잡도:

Node array[MAX\_ARRAY\_SIZE];

맨 처음에 위와 같이 선언하였으므로 O(MAX\_ARRAY\_SIZE)이다.

<삭제>

score\_number--;

배열의 맨 마지막 원소만을 없애주면 된다. 이때 굳이 해당 배열의 원소를 초기화해줄 필요가 없으므로 score\_number만 -1해주면 된다.

시간복잡도: O(1)

공간복잡도: 삽입에서와 같은 이유로 O(MAX\_ARRAY\_SIZE)이다.

3.

3-1. linked list의 경우

*// X, Y를 입력받기*

X, Y의 값을 사용자로부터 입력받는다.

if(X가 Y보다 크거나, X가 1보다 작거나, Y가 전체 랭킹의 개수보다 큰 경우) {

에러 메시지 출력

}

else {

    int count = 1;

    printw("      name      |   score\n");

    printw("----------------------------\n");

    for(Node \*ptr = head; ptr != NULL; ptr = ptr->link) {

        if(count가 X와 같거나 크고, Y보다 작거나 같은 경우) {

해당 노드(ptr)에 저장된 name과 score를 형식에 맞추어 출력한다.

        }

        else if(count가 Y보다 큰 경우)break;

        count++;

    }

}

시간복잡도: 최악의 경우 배열 전체를 돌아야 한다. 따라서 linked list의 원소 수를 n이라고 할 때, O(n)이다.

공간복잡도: 추가로 생성되는 Node는 없다. 따라서 따라서 linked list의 원소 수를 n이라고 할 때, O(n)이다.

3-2. Array의 경우

*// X, Y를 입력받기*

X, Y의 값을 사용자로부터 입력받는다.

if(X가 Y보다 크거나, X가 1보다 작거나, Y가 전체 랭킹의 개수보다 큰 경우) {

에러 메시지 출력

}

else {

    for(int i=X-1; i<=Y-1; i++) {

array[i]에 저장된 name과 score를 형식에 맞추어 출력

}

}

시간복잡도: 배열의 인덱스 X-1부터 Y-1까지 탐색하므로 O(Y-X+1)

공간복잡도: 처음에 할당된 array만 사용되므로 2-2에서와 같은 이유로 O(MAX\_ARRAY\_SIZE)이다.