7주차 예비보고서

전공: 수학/컴퓨터공학 학년: 3학년 학번: 20181294 이름: 임승섭

**1.**

1 - 1 Array

먼저 배열을 이용할 수 있다. 저장해야 하는 정보는 사용자의 이름 (name)과 점수(score)이다. 만약 두 정보가 같은 자료형이라면 2차원 배열을 이용하겠지만 다르기 때문에 배열을 두 개를 만든다. 문자열을 저장하는 배열과 점수를 저장하는 배열을 각각 만든다. 초기에 적당한 크기를 malloc을 이용하여 배열에 할당해주고, 만약 그 이상으로 정보가 들어온다면 realloc을 이용하여 크기를 키워준다. 이 때, 새로운 정보가 들어오면 점수가 높은 순서대로 저장해주어야 한다. 먼저 점수가 저장되어 있는 배열로 몇 번째 인덱스에 새로운 값이 들어가야 하는지 확인하고, 이를 이용해 이름이 저장되어 있는 배열도 수정해준다. 아무래도 나중에 보다 쉬운 출력을 위해 미리 점수가 높은 순서대로 저장해주는 것이 낫다.

1 - 2 Linked List

다음은 Linked List를 이용할 수 있다. 두 가지 정보를 하나의 노드에 저장해주면 되기 때문에 배열 두 개를 사용하는 것보다는 간편하다. 노드에는 name, score, next를 부여하고 name은 사용자 이름, score는 점수, next는 다음 노드를 가리키는 포인터를 할당한다. 새로운 정보가 들어오면 새로운 노드를 만들어서 추가해주면 되기 때문에 초기에 크기를 지정해 줄 필요가 없다. 마찬가지로 나중에 간편한 출력을 위해 새로운 정보를 저장할 때 점수 순서대로 저장해준다. 각 노드의 점수들을 비교해가며 알맞은 위치에 새로운 노드를 저장해준다.

**2.**

2 - 1 Array

// name array and score array (global variable)

int rank\_num = 1;

char \*\*N = (char\*\*)malloc(sizeof(char \*) \* rank\_num)

int \*S = (int\*)malloc(sizeof(int) \* rank\_num)

S[0] = 0;

// INSERT\_ARRAY(char \*name, int score)

if (rank\_num == 1)

S[0] = score

N[0] = (char\*)malloc(sizeof(char) \* full\_name\_num)

strcpy(N[0], name)

rank\_num++

else

for (int i = 0; i < rank\_num; i++)

if (S[i+1] < score)

break;

N = realloc(N, rank\_num+1)

N[rank\_num+1] = (char\*)malloc(sizeof(char) \* full\_name\_num)

for (int j = rank\_num; j > i; j--)

S[j] = S[j-1]

strcpy(N[j], N[j-1])

S[i] = score

strcpy(N[i], name)

<시간복잡도>

현재 순위의 개수를 나타내는 rank\_num에 대해서만 for loop을 돌기 때문에 시간복잡도는 O(rank\_num)이라 할 수 있다.

<공간복잡도>

마찬가지로 현재 순위의 개수를 나타내는 rank\_num 크기의 배열 두개만 사용하기 때문에 공간복잡도도 마찬가지로 O(rank\_num)이라 생각할 수 있지만 name 을 저장하는 배열은 문자열 2차원 배열이다. 따라서 공간복잡도는 O(rank\_num \* full\_name\_num)이다.

// DELETE\_ARRAY(char \*\*name)

int i = 0

while(strcmp(N[i], name) != 0)

i++

if (i == rank\_num)

printf(“No %s\n”, name)

break

for (int j = i; j < rank\_num; j++)

S[j] = S[j+1];

strcpy(N[j], N[j+1])

free(S[rank\_num])

free(N[rank\_num])

rank\_num--

<시간복잡도>

worst case인 경우는 for loop 에서 i가 0부터 rank\_num까지 갈 때이다. 따라서 시간복잡도는 O(rank\_num) 이다.

<공간복잡도>

Delete할 원소가 있으면 배열의 마지막 인덱스를 메모리 해제해주기만 하고 있고, 따로 추가되는 메모리 할당은 없다. 여전히 O(rank\_num \* full\_name\_num)이다.

2 - 2 LinkedList

// name and sore node

typedef struct \_Node {

char name[full\_name\_num]

int score

Node \*next

} Node;

// INSERT\_LINKED\_LIST(char \*name, int score)

Node \*newnode = (Node\*)malloc(sizeof(Node))

strcpy(newnode->name, name)

newnode->score = score

prev = HeadNode

while(prev != NULL)

if (prev.Score > score)

prev = prev->next

else

break

newnode->next = prev->next;

prev->next = newnode

<시간복잡도>

worst case일 때 prev가 node들의 맨 마지막까지 가는 경우를 생각할 수 있다. while loop이 돌기 때문에 시간복잡도는 O(rank\_num)이라 할 수 있다.

<공간복잡도>

newnode에 메모리를 추가하긴 하지만 node 하나에 대한 메모리기 때문에 기존과 변함이 없다고 할 수 있다. 공간복잡도는 O(rank\_num)이다.

// DELETE\_LINKED\_LIST(char \*\*name)

prev = HeadNode

if (strcmp(prev->name, name)

HeadNode = prev->next

free(prev)

curr = prev->next

while (prev != NULL)

if (strcmp(curr->name, name) == 0)

break;

prev = prev->next

curr = curr->next

if prev == NULL

printf(“No %s\n”, name)

else

prev->next = curr->next;

free(curr)

<시간복잡도>

worst case일 때 linkedlist의 맨 마지막까지 가는 경우를 생각할 수 있다. 따라서 O(rank\_num)이다.

<공간복잡도>

새로 추가되는 메모리 할당은 없다. 그대로 O(rank\_num)이다.

**3.**

3 - 1 Array

Get\_rank(int x, int y)

for (int i = x; i < y; i++)

printf(“%s %d\n”, N[i], S[i])

<시간복잡도 & 공간복잡도>

매개변수로 받은 x부터 y 사이에서만 for loop을 돌기 때문에 시간복잡도는 O(y - x)라고 할 수 있고, 공간복잡도는 이미 있는 배열을 사용하는 것이기 때문에 여전히 O(rank\_num \* full\_name\_num)이다.

3 - 2

Get\_rank(int x, int y)

prev = HeadNode

for (int i = 0; i < x; i++)

prev = prev->next

for (int i = x; i < y; i++)

printf(“%s %d\n”, prev->name, prev->score)

prev = prev->next

<시간복잡도 & 공간복잡도>

i가 0부터 y까지 loop을 돌기 때문에 시간복잡도는 O(y)라고 할 수 있고, 공간복잡도는 이미 존재하는 링크드리스트를 사용하기 때문에 여전히 O(rank\_num)이다.