7주차 예비보고서

전공: 수학과,컴퓨터공학과 학년: 3학년 학번: 20181256 이름: 김도현

1.

Array

Struct 구조에 랭킹이름, 랭킹점수를 저장하고 있는 배열에 대해 생각을 해보았다. 예를 들어 1등 DoHYun 1000, 2등 DH 500 점이라면 Array[0].rankname=DoHYun, Array[0].rankscore=1000, Array[1].rankname=DH, Array[1].rankscore=500 으로 struct 형 배열에 넣으면 될 것이다. 이때 배열의 크기는 계속 새로운 ranker에 대해 insert를 받아야 하기 때문에 처음 malloc으로 동적 배열 할당을 받은 후 realloc 등을 이용하여 메모리 재 할당을 받으면 해결 될 것이다. 단점으로는 array는 서로 붙어 있는 continuous를 띄기 때문에 delete, insert 할 때마다 새롭게 배열의 순서를 결정해야 할 것이다.

Linked List

Linked list 는 생각 했을 때 이상적인 자료구조이다. 왜냐하면 ranking 시스템은 node라는 곳에 랭킹이름, 랭킹점수가 포함이 되어 있다. 또한 거기에는 다음 node를 가리키는 포인터가 들어있기 때문에 만약 점수의 흐름대로 node들이 포인터 한다고 가정 했을 때 모든 node들의 순서를 다시 설정하지 않고 링크를 따라서 해당 원하는 곳을 찾아가서 insert 시 prevbignode->newnode->prevsmallnode(newnode가 중간값이라면..)으로 prevbignode의 포인터 링크를 newnode로 새로 할당해주고 newnode의 포인터 링크는 prevsmallnode로 설정하면 끝이 난다. Delete도 마찬가지로 prevbignode의 링크를 newnode->link 인 prevsmallnode로 만들어주고 newnode 를 delete 시키면 된다.

2. 자세한 코드 설명은 //주석 표시에 코드 옆에 달아두었다.

Array

Pseudo code

Typedef struct{

char\* rankname //rankname 저장 malloc으로 할당하거나 따로 100정도의 길이제한두는것도..

int rankscore //rankscore 저장

} AA

AA\* Array=malloc(ranknumber\*sizeof(AA)) //Array struct 배열 만듬

Insert(rankname, rankscore)

Array=realloc(Array, ranknumber+1 크기 배열 재할당)

for(i=0;Array[i].rankscore>rankscore;i++) //rankscore 위치를 찾고

{}

for(j=ranknumber-1;j>i;j--)//가장 낮은 순서부터 rankscore 위치 앞까지 for 문

Array[j+1].rankname, rankscore => Array[j].rankname, rankscore

//j+1에 j 를 넣음

Array[i].rankname, rankscore=rankname, rankscore //해당 위치 I 에 받아온 rankname, rankscore 대입.

ranknumber=ranknumber+1 // ranknumber 1을 더해준다.

시간복잡도는 첫 for문은 rankscore에 따른 위치를 찾는 것이고 두번째 for문은 삽입될 때 한칸 씩 뒤로 미뤄야하기 때문에 있는 것이다. 해당 for문이 돌아갈 때 시간 복잡도는 ranknumber에 따라 달라지기 때문에 O(ranknumber)이라 생각하면 되고 공간 복잡도는 동적 배열 재할당을 ranknumber에 관련해 받기 때문에 마찬가지로 O(ranknumber) 이다.

Delete(number)

Array[number-1].rankname=NULL //해당 ([0]이 1등)Array[number-1]를 null 상태로 만듬

Array[number-1].rankscore=NULL

for(i=number-1;i<ranknumber-1;i++) //insert와 반대로 뒤에있는 rank들을 하나 앞으로 당겨옴

Array[i].rankname, rankscore => Array[i+1].rankname, rankscore

Array[i].rankname=NULL //마지막 rank가 Array[ranknumber-2]에 저장되어 있기 때문에 지움.

Array[i].rankscore=NULL

Array=realloc(Array, ranknumber-1 크기 배열 재할당)

ranknumber=ranknumber-1 //삭제했기 때문에 rankumber를 1 줄여준다

시간복잡도는 해당 for문이 Array[number-1]가 비워져 있기 때문에 다시 그곳을 채우기 위해 사용되었다. 이때 돌아갈 때 시간복잡도는 ranknumber 에 따라 달라지기 때문에 최악은 O(ranknumber) 이고 공간복잡도는 동적 배열 재할당이 ranknumber에 관련해 받기 때문에 마찬가지로 O(ranknumber) 이다.

Linked List

Insert(rankname, rankscore)

Node \*newnode->rankname, rankscore= rankname, rankscore//newnode에 해당하는 rankname과 rankscore를 넣는다.

Node \*current=top //current,prev 노드를 이용해 해당 newnode의 위치를 link를 통해 찾는다

Node \*prev=0

if(current!=NULL) //current 가 null이 아니라면

while(current->rankscore>=rankscore)

prev=current// prev가 current가 되고

current=current->link //current가 current+1 이 됨

newnode->link=current// newnode의 link는 current 즉 하나 더 작은 rankscore를 가진 노드에 연결

prev->link=newnode // prev->link 즉 newnode보다 하나 더 큰 rankscore를 가진 노드는 newnode로 연결

ranknumber=ranknumber+1 //입력했기 때문에 rankumber를 1 늘린다

시간복잡도는 while 문에서 rankscore의 크기비교로 돌아가기 때문에 rankscore가 가장 마지막에 올때(가장 작을 때) 최악의 경우로 ranknumber에 의해 결정되므로 O(ranknumber) 이 된다.

공간복잡도는 newnode, current node를 생성하고 이걸로 탐색 후 기존의 linkedlist에 insert하기 때문에 O(1) 이다. 이때 currentnode로 linked list를 탐색 할 시 global 변수인 linked list에 공간복잡도는 O(ranknumber)이 된다.

Delete(number)

Node \*current=top //current,prev 노드를 통해 해당 number가 어떤 위치인지 알아낸다

Node \*prev=0

If(number>ranknumber) //받은 숫자가 ranknumber 수보다 크면 에러 발생

Error!!

exit(-1)

else //아니면

for(i=0;i<number-1;i++) //number-1 보다 작을때 까지

prev=current //prev가 current 되고

current=current->link /current 가 current->link 가 됨

prev->link=current->link// prev node와 current->link node를 연결

Free(current) //current node 삭제

시간복잡도는 for 문에서 number의 수만큼 node를 탐색하며 돌아가는데 이때 최악의 경우는 number=ranknumber 일 때 이므로 O(ranknumber)이라 생각할 수 있다.

공간복잡도는 for 문의 i, currentnode, prev node를 가지고 linked list 를 탐색하기 때문에 이에 대한 메모리할당은 O(1) 이다. 이때 linked list를 탐색 할 시 global 변수인 linked list에 공간복잡도는 O(ranknumber)이 된다.

3. pseudo code이다.

Array

If(Y>ranknumber||X<1) error // error 출력

for( i=X-1;i<Y;i++) //rank X~Y (array 배열 0부터 시작한다고 가정)

printing(Array[i].rankname, Array[i].rankscore) // 바로 프린트 하면 된다

X~Y 의 랭킹을 출력하는 것이기 때문에 for문이 돌아갈 때 Array[X-1] 부터~Array[Y-1]까지 돌아가면 되기때문에 시간 복잡도는 O(Y-X) 인데 최악의 경우는 Y=ranknumber 이고 X=1일 경우이다. 즉 O(ranknumber)로 생각하면 되어서 O(ranknumber) 이다. 공간 복잡도는 함수 자체는 for문의 i 변수 정도밖에 없어 O(1) 이다.

이때 global 변수인 배열의 크기는 ranknumber 이기 때문에 Array배열은 O(ranknumber)가 된다.)

Linked List

Node \*current=top //current node를 통해 linked list를 탐색해 출력한다

If(Y>ranknumber||X<1) error // error 출력

for(i=1;i=<Y;i++)

if(X=<i) //i가 x보다 크거나 같을 때부터 Y보다작을 때 까지

printing(current->rankname, current->rankscore) //rankname, rankscore 출력

current=current->link //계속 탐색 해간다

위의 array 와 마찬가지로 노드 개수로 X보다 크거나 같은 숫자부터 출력을 시작해서 Y의 수만큼노드를 탐색 및 출력한다. 이때 시간 복잡도는 O(Y-X) 인데 최악의 경우는 Y=ranknumber 이고 X=1일 경우이다. 즉 O(ranknumber)로 node를 for문이 ranknumber 만큼 돌아갈 수 있기 때문에 O(ranknumber) 이라 생각하면 된다.

공간복잡도는 currentnode를 새로 할당해 linkedlist를 탐색하여 출력하기 때문에 O(1)이다. 이때 linked list를 탐색 할 시 global 변수인 linked list에 공간복잡도는 O(ranknumber)이 된다.