10주차 결과보고서

전공 : 컴퓨터공학 학년 : 2학년 학번 : 20151616 이름 : 최승환

1. 실험시간에 작성한 랭킹 시스템의 자료구조와 랭킹 시스템의 각 기능에 대한 알고리즘을 요약하여 기술하시오. 본인이 선택한 랭킹 시스템을 구현하기 위한 자료구조가 왜 효율적인지 시간 및 공간 복잡도를 통해 보이고, 설명하시오.

링크드 리스트를 사용해 입력 받은 자료들을 저장했다. 이 때 점수는 내림차순으로 정렬하도록 되어 있고, 이는 입력을 받는 동시에 처리한다. 입력을 받으면 자신보다 큰 노드를 만나기 전까지 노드를 하나씩 옮겨 가고, 두 노드 사이에 새로운 노드를 만들어 점수와 이름을 저장한다. 이런 방식을 사용하면 항상 정렬되어 있는 상태를 유지할 수 있다.

또한 입력을 받아 노드를 먼저 정리한 다음 그 노드들을 찾아가며 프린트 하는 것이 아니라 노드들을 입력받는 동시에 출력을 하고 노드 정리를 하는 방식이기 때문에 처음부터 끝까지 두 번 참조할 필요가 없이 단 한 번 노드를 도는 것만으로 모든 작업이 끝나게 된다.

따라서 새로운 노드를 입력하는데 시간복잡도는 O(n)이다. 하지만 이는 search와 print를 모두 합친 시간복잡도이므로 시간 상으로 이득이 있다고 할 수 있다.

void newRank(int score){

Rank\_node\* currnode;

Rank\_node\* bef=NULL;

Rank\_node\* newnode = malloc(sizeof(Rank\_node));

int ncnt,flag=1;

FILE\* wfp;

//FILE\* rfp = fopen("rank.txt","r");

clear();

printw("enter your name : ");

scanw("%s",newnode -> name);

newnode -> score = score;

newnode -> next\_node = NULL;

createRankList();

wfp = fopen("rank.txt","w");

currnode = head;

ranknum++;

fprintf(wfp,"%d\n",ranknum);

if(ranknum == 1)

{

fprintf(wfp,"%s %d\n",newnode->name,newnode->score);

head = newnode;

return;

}

for(ncnt=0;ncnt<ranknum-1;++ncnt)

{

if(currnode -> score < newnode -> score)

{

if(currnode != head)

bef -> next\_node = newnode;

newnode -> next\_node = currnode;

currnode = newnode;

//fprintf(wfp,"%s %d\n",newnode->name,newnode->score);

break;

}

fprintf(wfp,"%s %d \n",currnode -> name,currnode -> score);

bef = currnode;

currnode = currnode -> next\_node;

}

if(currnode == NULL)

{

return;

}

while(currnode != NULL)

{

fprintf(wfp,"%s %d\n",currnode -> name,currnode -> score);

if(currnode -> next\_node == NULL)

break;

currnode = currnode -> next\_node;

}

currnode = currnode -> next\_node;

fclose(wfp);

return;

// user code

}

2. 본 실험 및 숙제를 통해 습득한 내용을 기술하시오.

기존에 밝혔듯이 이 실험을 진행할 때 사용할 수 있는 방법은 array와 linked list 두 가지가 있다. 하지만 array를 사용할 경우 데이터를 추가하거나 삭제할 때 번거롭고 시간복잡도도 크기 때문에 linked list를 사용해 추가와 삭제를 용이하도록 했다. array를 사용하면 숫자를 하나 받아 그 노드를 검색하는데는 용이하겠지만 이를 삭제할 때는 linked list에서 노드를 검색할 때와 비슷한 시간이 걸리게 된다. 따라서 linked list를 이용해 받은 숫자만큼 노드를 찾아가고 해당 노드의 링크를 이용해 삭제한다.

void delete\_node(int temp)

{

//todo

Rank\_node\* currnode = head;

Rank\_node\* befnode;

int cnt=temp;

int chk = 0;

echo();

ranknum--;

wgetch(stdscr);

while(temp > 1)

{

temp--;

befnode=currnode;

currnode = currnode -> next\_node;

}

if(currnode -> next\_node == NULL)

{

if(cnt == 1)

{

free(currnode);

head = NULL;

}

else

befnode -> next\_node =NULL;

}

else if(cnt == 1)

{

head = currnode -> next\_node;

free(currnode);

}

else

{

befnode -> next\_node = currnode -> next\_node;

//free(currnode);

}

noecho();

}