9주차 결과보고서

전공: 신문방송학과 학년: 3학년 학번: 20191150 이름: 전현길

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

void createRankList()
rank.txt 파일에서 랭킹 정보를 입력받아 랭킹 목록을 생성하고, linked list 형태로 저장한 뒤 파일을 닫는다. 함수의 알고리즘은 다음과 같다. 1) fscanf(fp, "%d", &rankNum) 함수를 통해 rankNum 변수에 데이터의 개수를 입력받는다. 데이터가 없다면 파일을 닫고 함수를 종료한다. 3) 데이터가 있다면, 노드 포인터 curNode를 rankRoot에 포인팅해 입력받을 준비를 한다. 4) 각 라인마다 char rank_name[NAMELEN + 1], int rank_score 변수에 각각 현재 rank의 이름, 점수 데이터를 입력받는다. 4) 노드 포인터 newNode를 생성하고, 데이터를 newNode에 복사한다. 5) 첫 포인터일 경우(rankRoot == NULL), rankRoot, curNode 포인터를 newNode에 포인팅한다. 6) 그렇지 않을 경우, 이전 노드(curNode)의 다음 포인터에 newNode를 포인팅하고 curNode를 newNode로 이동한다. 입력받은 데이터를 선형적으로 탐색하면서 linked_list 데이터를 생성하므로 **시간 복잡도는 O(n)이고, 공간 복잡도는 O(n)**이다.

void writeRankFile()
rank.txt 파일을 열고, 랭킹 정보의 수(rankNum)를 파일에 기록한 뒤, 링크드 리스트의 데이터를 line by line으로 기록한다. 1) 노드 포인터 curNode를 rankRoot에 포인팅하고, prevNode 널 포인터를 생성한다. 2) prevNode를 curNode를 따라 이동시키면서, curNode가 NULL일 때까지 데이터를 순회한다. 3) curNode의 이름, 점수를 rank.txt에 기록한다. 4) 파일을 닫는다. 링크드 리스트 데이터를 순회하면서 파일에 데이터를 저장하므로, 시**간 복잡도와 공간 복잡도는 모두 O(n)**이다.

void rank()

rank 메뉴를 출력하여, 세 가지 기능을 수행하도록 한다.
1) X위부터 Y위의 이름, 점수 데이터를 출력한다.
2) 특정한 이름을 가진 이름, 점수 데이터를 출력한다.
3) 특정한 순위의 데이터를 삭제한다.

기능 1)
a. 입력받은 X, Y 데이터를 전처리한다. X, Y 입력값이 현재 랭킹 목록의 경계를 넘는다면 넘지 않도록 초기화하고, 만약 X가 Y보다 크거나, 랭킹 목록의 데이터가 없다면 예외 처리한다.
b. curNode 노드 포인터를 rankRoot에 포인팅하고, curRank 변수를 1로 초기화한다.
c. curNode가 NULL이 될 때까지, curRank를 1씩 증가시키고, curNode를 이동시킨다.
d. X <= curRank <= Y일 때 curNode의 이름, 점수 데이터를 출력한다.
e. curRank > Y라면 break로 반복문을 종료한다.

기능 2)
a. 문자열을 str 변수에 입력받는다.
b. curNode 노드 포인터를 rankRoot에 포인팅한다.
c. curNode가 NULL이 될 때까지, 링크드 리스트를 순회한다.
d. curNode의 이름과 str이 같다면(strcmp(curNode->name, str) == 0) 이름, 점수를 출력한다.

기능 3)
a. 삭제할 데이터의 순위를 input에 입력받는다.
b. input이 1보다 작거나, rankNum(랭킹 목록의 수)보다 크다면 예외 처리한다.
c. curNode 노드 포인터를 rankRoot에 포인팅하고, prevNode 널 포인터를 생성한다.
d. rankNum 변수를 1 감소시키고, curRank 변수를 1로 초기화한다.
e. curRank를 1씩 증가시키고, prevNode를 curNode를 따라 이동시키면서 데이터를 순회한다.
f. 현재 탐색하는 순위와 입력한 순위가 같으면(curRank == input) 삭제 절차를 수행한다.
g. 데이터가 1등이라면(prevNode == NULL) rankRoot를 curNode->next로 포인팅한다.
h. 그렇지 않다면, prevNode->next를 curNode->next로 포인팅한다.
i. curNode에 동적 할당된 메모리를 해제하고, break로 반복문을 탈출한다.

기능 1), 기능 2), 기능 3)이 전부 데이터를 선형적으로 탐색하므로 선형 탐색을 위해 O(n)의 시간 복잡도를 가지며, 기능 3)의 경우 데이터 삭제 자체에는 O(1)의 시간 복잡도가 발생한다. 공간 복잡도는 랭킹 목록의 데이터 수에 비례하여 O(n)이다.

void newRank(int score)
게임이 종료되었을 때 현재 score와 이름을 랭킹 목록의 적절한 위치에 저장한다. 1) 이름을 char str[NAMELEN + 1] 변수에 입력받는다. 2) 노드 newNode를 메모리 동적 할당으로 생성해 데이터를 복사한다. 3) 노드 포인터 curNode를 rankRoot에 포인팅하고, prevNode 널 포인터를 생성한다. 4) rankNum을 1 증가시키고, found_rank(랭크 삽입 여부에 대한 flag)를 0으로 reset한다. 5) prevNode를 curNode를 따라 이동시키면서, curNode가 NULL일 때까지 데이터를 순회한다. 6) 적절한 위치를 찾았다면(newNode->score >= curNode->score), found_rank flag를 set하고, 데이터를 삽입한다. 현재 노드(curNode)가 가장 앞에 있는 데이터인지, 아닌지에 따라 삽입 절차를 다르게 수행한다. 삽입이 완료되었다면 break로 반복문을 종료한다. 7) 반복문을 종료했는데 found_rank flag가 reset되어 있다면, rank에 저장된 데이터가 없거나 입력된 데이터의 점수가 가장 낮으므로, 이에 따라 데이터를 다르게 삽입한다. 링크드 리스트 데이터를 순회하면서 적절한 위치를 찾는 과정이 O(n)의 시간 복잡도를 갖고, 데이터 삽입 자체는 O(1)의 시간 복잡도를 갖는다. 공간 복잡도는 링크드 리스트의 길이에 비례하여 O(n)이다.

a. 결과보고서 작성 1

효율성을 평가하고 왜 그렇게 생각하는지 상세하게 기술하라.

테트리스 랭킹 데이터에 요구되는 연산은 다음과 같다.

1) X위부터 Y위까지의 이름, 점수 데이터 출력

2) 특정한 이름의 데이터 출력

3) 특정한 순위의 데이터 삭제

4) rank.txt 파일을 통한 입출력

5) 게임 종료 시 데이터를 적절한 순위로 입력

위의 다섯 가지 연산을 압축하면 삽입, 삭제, 탐색 연산으로 요약할 수 있다. 해당 연산들에 대해 모두 worst-case에서 O(logn)의 시간 복잡도를 가지도록 대응할 수 있는 것은 이진 탐색 트리를 개선한 RB 트리, AVL 트리가 존재한다. 이진 탐색 트리의 경우 O(h)의 시간 복잡도를 가지므로, 경사 이진 탐색 트리 형태의 데이터가 생성되면 최악의 경우 O(n)의 시간 복잡도를 가질 수 있다.

다만 RB 트리, AVL 트리 등의 자료구조에 대해서 충분한 이해를 갖지 못한

상태였기 때문에 부득이하게 linked list 자료구조를 사용했다.

링크드 리스트 자료구조가 메모리 측면에서 갖는 장점은 배열과 달리 메모리를 얼마든지 동적 할당하여 데이터를 이어붙일 수 있다는 점이다. 배열은 최대 길이가 고정되어 있기 때문에, 배열의 길이를 너무 크게 선언하면 메모리가 낭비될 수 있고, 너무 작게 선언하면 배열의 경계를 넘은 메모리에 접근하는 세그먼테이션 오류 등이 발생할 수 있다.

따라서 동적 할당을 통해 데이터를 저장하고, 배열의 현재 길이를 따로 변수로 저장하여 배열의 현재 길이보다 더 많은 양의 데이터가 저장될 경우 이에 대응하는 코드를 작성해야 한다. 구체적으로 더 큰 메모리를 동적 할당하고, 기존의 데이터를 새 메모리 공간에 복사하며, 기존 메모리는 해제해야 한다.

링크드 리스트는 따로 메모리에 관련된 대응이 필요하지 않기 때문에 구현상의 용이점이 있다. 뿐만 아니라, 데이터의 삽입/삭제를 위해 O(1)의 시간 복잡도가 발생한다는 점 역시 배열에 비한 메리트이다.

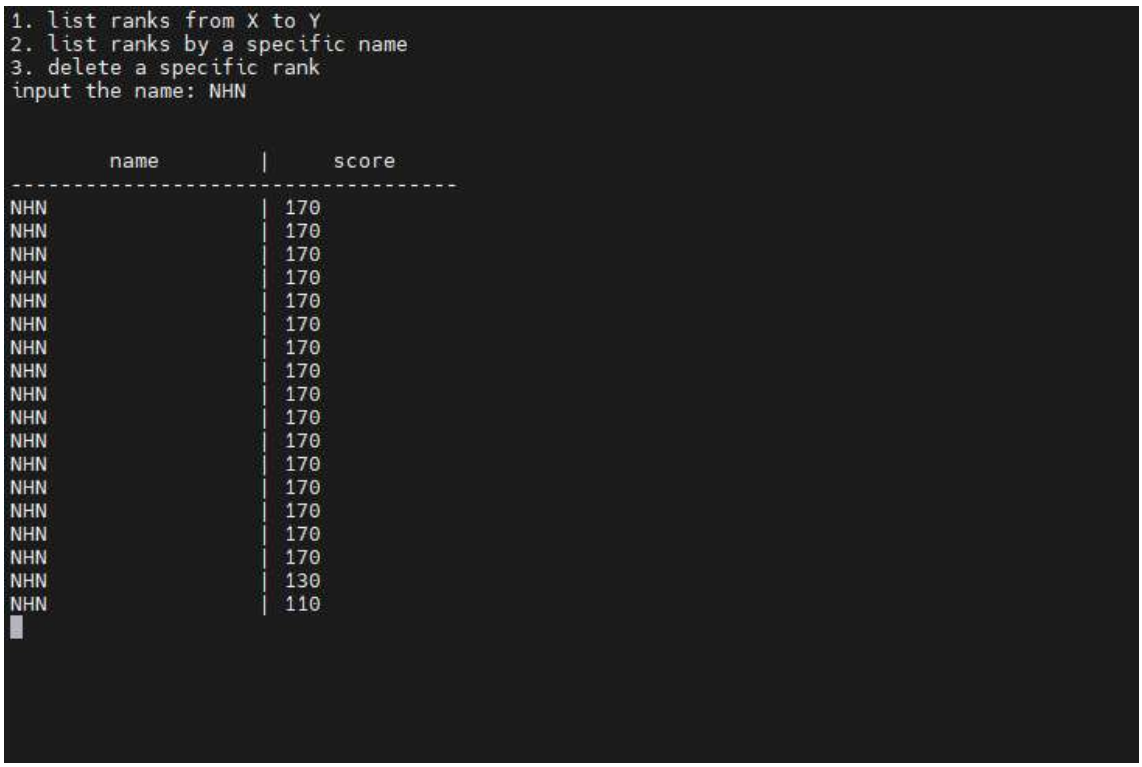
하지만, 배열을 사용할 경우 2번 연산(입력된 이름을 찾는 연산), 5번 연산(적절한 점수를 찾는 연산)에 대해 데이터 탐색을 O(logn)으로 줄여서 구현할 수 있고, 1번 연산과 3번 연산에 대해서도 O(1) 방식으로 원하는 데이터를 찾을 수 있다는 장점이 있다. 이에 비해 링크드 리스트로 해당 탐색을 수행하려면 모두 O(n)의 시간 복잡도가 발생한다.

단, 1번 연산의 경우 전체 데이터를 탐색하면 어차피 최악의 경우 O(n)의 시간 복잡도가 발생한다는 점, 3번 연산에서 삭제할 데이터를 O(1)만에 찾았더라도 결국 실제 삭제 과정에서 O(n)의 시간 복잡도가 발생한다는 점을 고려하면 실질적으로 이득이 있는 연산은 2번, 5번 연산이다.

정리하자면, 전반적인 시간 복잡도를 고려했을 때 최선의 선택은 RB 트리, AVL 트리이며, 그 다음으로 효율적인 것은 배열 자료구조이지만 메모리의 최대 크기를 항상 관리해 주어야 한다. 이에 비해 링크드 리스트는 탐색 시 시간 복잡도가 O(n)이므로 O(1)만에 삽입, 삭제가 이뤄지더라도 시간 복잡도 면에서 우월하지는 못하지만, 배열에 비해 메모리 관리가 효과적이라는 장점이 있다. 이를 바탕으로 링크드 리스트는 랭킹 목록의 특성을 보았을 때 최선의 자료구조는 아니더라도 납득할 만한 성능을 가진 자료구조라고 결론지을 수 있겠다.

b. 결과보고서 작성 2

사용자의 이름을 입력하고, 해당하는 랭킹 정보 출력 화면 첨부



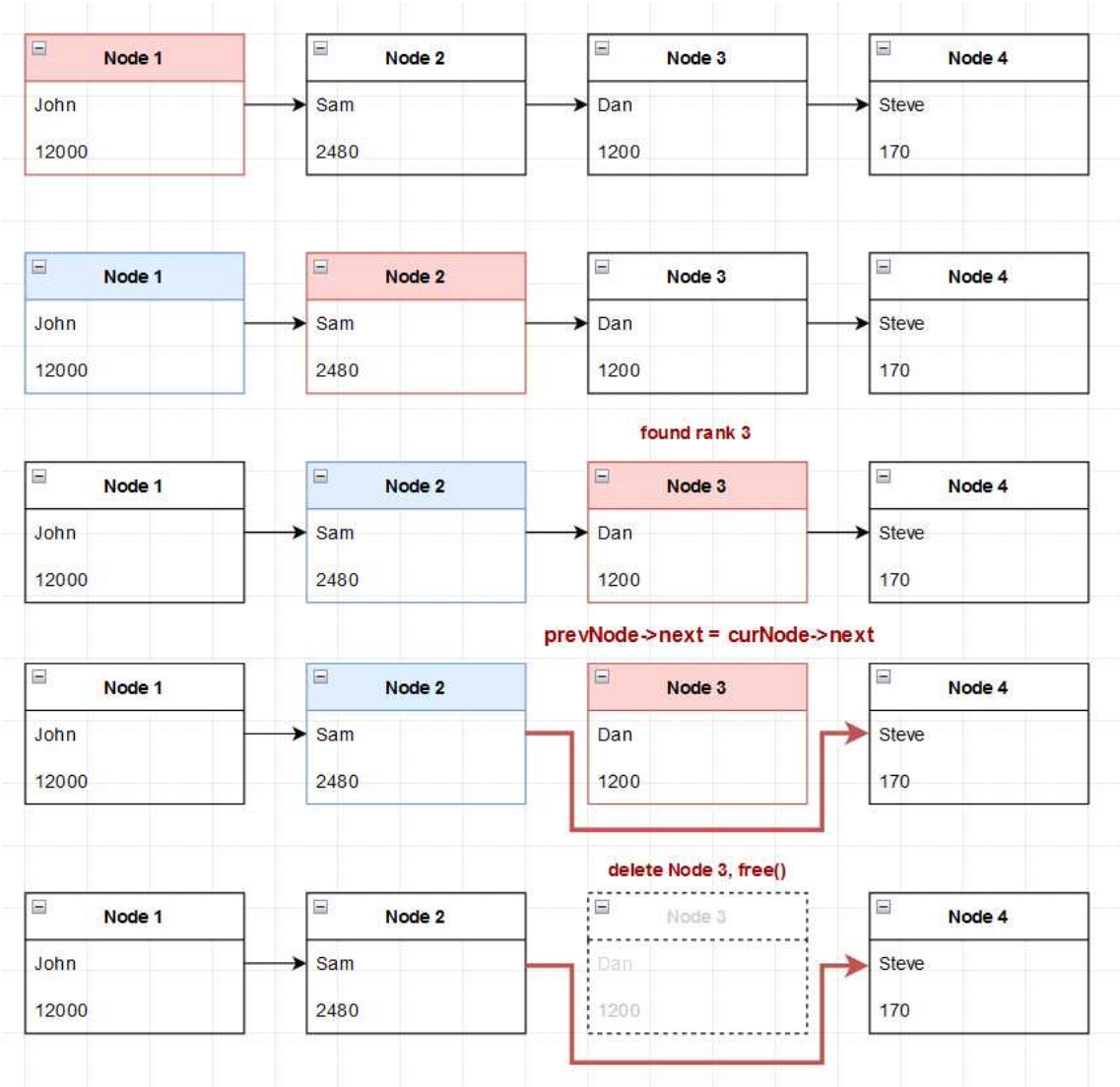
c. 결과보고서 작성 3

사용자 이름으로 원하는 사용자 이름을 검색할 때의 시간 및 공간 복잡도

상기한 rank() 함수의 설명 박스를 참고.

d. 결과보고서 작성 4

선택한 자료구조에 맞춰 삭제 알고리즘을 그림으로 표현한다.



위 그림은 **랭킹 리스트에 4개의 정보가 입력되어 있을 때, 사용자가 입력값으로 3을 입력할 경우를 표현**한다. **붉은색 노드는** 현재 링크드 리스트를 순회하는 노드 **curNode**를 표현한 것이고, **푸른색 노드**는 이전 링크드 리스트의 노드를 가리키는 **prevNode**를 표현한 것이다.

curNode와 prevNode를 다음 노드로 계속 이동시키면서 변수 curRank를 증가시키다가, 목표했던 순위인 3위에 도달하면, (curRank == 3), prevNode ->next = curNode->next로 이전 노드를 다음 노드에 연결한 뒤 노드를 삭제한다. free(curNode) 함수로 메모리를 해제한 뒤 break로 종료한다.

2. 본 실험 및 숙제를 통해 습득한 내용을 기술하시오.

프로그램의 기능을 구현하기 위해 다양한 자료구조에 대한 폭넓은 이해가 필요함을 알 수 있었고, 시간 복잡도, 공간 복잡도를 근거로 하여 적절한 자료구조를 선택하는 과정을 살펴볼 수 있었다. 특히 이진 탐색 트리와 개선된 이진 탐색 트리들에 대한 이해가 부족함을 깨달을 수 있었다.