Data Structure Project2

Report

학과 : 컴퓨터 공학과

팀장 : 2013722095 최재은

팀원 : 2013722021 정무영

2013722079 김영근

1. Introduction

이번 프로젝트에서 구현한 프로그램은 학생들의 기본 정보와 성적을 저장해서 각종 정보를 기준으로 학생정보를 관리해주는 프로그램이다. 학생 정보가 저장된 txt파일을 LOAD명령으로 읽어 들여서 정보를 자동으로 저장한다. 학생 정보를 저장하는 데는 총 3가지의 자료구조가 사용됐다.

원하는 범위의 성적을 가진 학생을 검색해주는 기능을 위해 B+ Tree가 사용됐으며 B+ Tree는 성적을 기준으로 정렬된다. 성적이 같은 경우는 학번이 빠른 순서대로 정렬한다. PRINT\_BP명령어를 사용하면 성적을 기준으로 모든 학생의 정보를 출력할 수 있다. SEARCH\_BP 명령어를 사용하면 0부터 4.5까지의 범위에서 원하는 범위의 성적과 학년을 가진 학생들의 정보를 출력할 수 있다.

학생의 이름을 기준으로 학생정보를 관리하기 위해서 AVL Tree가 사용됐으며 학생의 정보는 이름의 오름차순으로 정렬된다. PRINT\_AVL명령어를 사용하면 학생의 이름을 오름차순으로 정렬해서 출력 해 준다. SEARCH\_AVL명령어를 사용하면 찾고자 하는 학생의 이름을 입력해서 찾을 수 있다. 여기서 찾은 학생의 정보는 max heap에 따로 저장돼서 RANK명령어에 이용된다.

원하는 학생들만 뽑아서 순위를 매기는 RANK기능을 위해 max heap이 사용 되었다. AVL Tree에서 찾고자 하는 학생의 이름을 입력해서 알맞은 정보가 존재 하면 그 학생의 정보를 복사해서 max heap에 저장하고 성적을 기준으로 heap내부에서 정렬이 된다. RANK를 한번 호출하면 이전에 heap에 들어있던 정보들이 전부 삭제 되므로 RANK를 사용하고 싶다면 다시 정보를 넣어 주어야 한다.

명령어 입력 시 조건에 맞지않는 값을 입력하거나 정보가 없으면 해당 명령어에 맞는 에러 코드를 출력해 준다.

1. Flowchart

프로그램 내 data가 없고 명령어가 정상인가?

No

BF Tree가 비어있나?

AVL이 비어있나?

LOAD

No

시작

No

Yes

“grade\_list.txt”불러 오기

Yes

Yes

root에 정보 저장

root에 정보 저장

No

false반환

성적을 비교해서 작으면 왼쪽 크면 해당 map의 pointer가 가리키는 곳으로 이동 후 더 이상 index node가 아니면 저장

이름을 비교해서 작으면 왼쪽 크면 오른쪽 자식으로 이동 후 NULL일 때 그 자리에 저장

파일이 제대로 열렸나?

Yes

txt파일에서 한 줄 읽는다.

node마다 BF를 설정해준다.

No

저장 한 곳의 map이 Bp order를 넘어가나?

학년을 분류해 알맞은 학년의 클래스로 동적할당 한다.

Yes

No

BF가 2를 넘어가는 node가 존재하나?

No

그 곳이 DataNode인가?

Yes

나머지 정보들도 동적할당 한 클래스 변수에 저장한다.

Yes

map의 가운데 요소를 지우지 않고 split 한다.

LL, LR, RR, RL 경우를 나눠서 Tree를 재구성 해준다.

AVL의 Insert함수로 학생의 정보를 전달한다.

map의 가운데 요소를 지우고 split 한다.

BP Tree의 Insert함수로 학생의 정보를 전달한다.

No

insert끝

부모 노드가 존재 하는가?

map의 가운데 요소를 부모 노드 map에 저장 후 노드끼리 재 연결

Yes

insert끝

파일을 닫는다

가운데 요소를 root로 설정 한다.

에러 코드 출력

split 하면서 새로 생긴 node나 root를 알맞게 연결한다

끝

SEARCH\_BP SEARCH\_AVL RANK

시작

시작

시작

명령어의 인자가 조건에 맞고 max heap에 data가 존재하는가

명령어의 인자가 조건에 맞고 AVL Tree에 data가 존재하는가

명령어의 인자가 조건에 맞고 BP Tree에 data가 존재하는가

No

No

No

Yes

Yes

Yes

범위 에 속하는 성적 중 가장 높은 성적을 가진 Data Node를 찾아 간다.

heap에 아무 데이터도 없을 때까지 root를 계속 pop하면서 출력한다.

같은 이름이 나올 때까지 입력 받은 이름이 작으면 왼쪽 자식 크면 오른쪽 자식으로 이동한다.

범위 내에서 제일 높은 성적부터 제일 작은 성적까지 입력한 학년과 같은 Data만 출력한다.

에러코드 출력

No

같은 이름을 발견 했는가?

끝

Yes

Yes

출력한 데이터가 하나라도 있는가?

찾은 학생의 정보를 출력하고 max\_heap에 정보를 저장한다.

에러코드 출력

No

에러코드 출력

끝

끝

PRINT\_BP PRINT\_AVL EXIT

시작

시작

시작

할당된 메모리를 전부 해제 한다.

명령어의 인자가 조건에 맞고 AVL Tree에 data가 존재하는가

No

명령어의 인자가 조건에 맞고 BP Tree에 data가 존재하는가

프로그램을 종료한다.

Yes

root부터 중위 순회로 출력 한다.

가장 높은 성적을 가진 Data Node를 찾아 간다.

끝

에러 코드 출력

가장 높은 성적부터 가장 작은 성적 까지 NULL값이 나올 때 까지 출력한다.

끝

No

출력하려는 정보 중 성적이 중복 되는 것이 있는가?

Yes

중복되는 정보만 학번이 빠른 순으로 출력한다.

에러 코드 출력

끝

1. Algorithm

B+ Tree

Insert

이 프로젝트에서 사용한 B+ Tree에는 Index node와 Data node가 있다. Index node는 map으로 구성되었으며 map의 key값은 학생의 학점 평균이 들어있고 value에는 왼쪽 자식 node를 제외한 자식 노드들의 주소 값이 들어있다.

Index node에는 학생의 실질 정보는 없고 Tree의 정렬 기준인 학생의 평점 정도만 들어있다. Data node는 map이 2개가 중첩 돼서 구성되어 있는데 바깥쪽 map의 key값은 Index node와 마찬가지로 학생의 평점이 들어있고 value에는 map이 들어있다. 내부 map의 key값은 평점이 같을 때 정렬 기준으로 사용할 학생의 학번이 들어있으며 value값에는 학생의 정보를 가리키는 주소 값이 들어있다.

B+ Tree가 비어 있는 상태에서 처음 data를 넣을 때는 Index node는 생성되지 않고 data node하나가 생성되며 그 data node가 root로 설정된다. B+ Tree가 비어 있지 않은 상태에서 data를 넣을 때는 data속 학생의 성적을 기존 Tree의 node속 key값인 학생의 평점과 비교하면서 들어간다. 삽입된 평점이 더 낮으면 왼쪽으로 그렇지 않으면 map속 다음 인자와 비교해서 낮으면 현재 인자가 가리키는 자식node로 들어가고 높으면 그 다음 인자로 넘어가서 그 다음 인자와 비교한다. 이렇게 계속 넘어가다가 마지막 인자와 비교했을 때도 높으면 마지막 인자가 가리키는 자식 노드로 들어간다. 위 동작을 반복하다가 Data Node를 만나면 해당 Data Node map에 정보를 넣어준다.

map은 삽입된 정보를 자동으로 오름차순으로 정렬해 주기 때문에 자동으로 정렬이 된다. Data node에 정보가 삽입될 때 map속 인자 개수가 B+ Tree order값인 3을 넘어가게 되면 split을 시켜야 한다.

Data node를 split시킬 때는 order값을 반으로 나눈 후 소수점을 버리고 1을 더하면 split할 위치를 나타내는 수가 나온다. 이번 프로젝트에선 2가 split할 map의 인자의 위치를 가리킨다. 2번째 인자의 이전 인자인 1번 인자와 2, 3번 인자를 나눠야 한다. 그러기 위해 새로운 Data node를 생성하고 그 곳에 split할 map전체를 복사한 후 1번 인자를 지운다. 그리고 split할 map에선 2, 3번째 인자를 지운다. 그 후 2번 인자를 부모 노드가 없다면 Index node를 새로 생성해서 그 곳에 넣어 주고 부모 노드가 있다면 부모 노드의 map에 2번 인자의 정보를 넣어준다. 이 때 split된 정보를 받는 부모 노드는 항상 Index node이므로 Index node의 서식에 맞게 정보를 넣어 주어야 한다.

부모에게 값 전달까지 무사히 끝났다면 새 Data node의 다음 node를 가리키는 포인터는 이전 Data node가 가리키던 node로 설정 해 주고 이전 node를 가리키는 포인터는 split된 Data node로 설정 해준다. 부모 노드가 새로운 Data node를 가리키게 하고 새 Data node도 자신의 부모 node를 가리키게 한다.

좀 전에 부모node에게 새 data를 전해 줄 때 data를 받은 Index node도 split 여부를 확인 해야 하는데 node 속 map의 내부 인자 수가 3미만이면 상관 없지만 3이상이면 split을 해줘야 한다. Index node의 split 방식은 Data node의 split방식에서 split 위치에 있는 인자가 복사되지 않고 삭제 되면서 부모 노드로 전달된다는 점만 다르다. 따라서 Index node의 split까지도 위와 같이 구현하면 된다.

Print

이번 프로젝트에선 가장 높은 성적부터 출력하되 중복된 성적이 나오면 빠른 학번부터 출력하라는 조건이 있다. 그래서 B+ Tree에서 가장 오른쪽 정보를 찾아가야 한다. 그러기 위해선 root부터 시작해서 Data node에 도착할 때까지 node속 map의 가장 오른쪽 인자까지 간 후 그 인자가 가리키는 자식 노드로 들어가서 위 방식을 계속 반복하면 된다.

목적지에 도착하면 현재 Data node의 가장 오른쪽 인자부터 출력하다가 중복된 성적이 나오면 해당 인자의 value인 내부 map에 들어가서 가장 왼쪽 인자부터 출력하면 된다. 이런 식으로 반복 하다가 평점을 key값으로 가지고 있는 map이 가장 첫번째 인자까지 출력 했다면 Data node의 Prev node로 가서 출력을 이어서 해준다. 그러다 NULL이 나오면 출력을 종료 한다.

Search

print와 동작이 꽤 비슷하다. print에선 무조건 가장 높은 값을 찾아서 오른쪽으로만 갔다면 Search에서는 범위의 최소값과 최대값 중에서 최대값을 가장 높은 값이라 생각하고 찾아 들어간 후 print에서 NULL로 설정한 ‘출력을 멈추는 위치’를 범위의 최소값보다 작은 값으로 설정해서 출력하되 입력한 학년과 다르면 제외하고 출력한다.

AVL tree

Insert

정렬 기준은 이진 트리와 같다. 하지만 트리 모양을 한 쪽에 치우치지 않게 하기 위해 값을 삽입 할 때마다 정렬 해주는 특징을 가지고 있다. 일단 이진 트리처럼 이전 node속 이름과 삽입된 data속 이름과 비교해서 삽입된 이름이 더 작으면 왼쪽자식으로 그렇지 않으면 오른쪽으로 이동시키는 동작을 NULL을 만날 때까지 반복한다.

위에 삽입하는 과정에서root부터 내려갈 때 각 node별로 balance factor값을 봐서 0이 아닌 node를 ‘a’가 가리키게 한다. 이렇게 하면 BF가1, -1인 node중 가장 최근 node를 ‘a’가 가리킬 수 있다. BF가 1, -1인 가장 최근 노드를 찾는 이유는 새 node가 삽입되면서 BF가 2, -2 둘 중 하나가 되어서 균형이 무너지는 지점이 될 수 있기 때문이다. 그 후 삽입된 노드가 ‘a’와 비교해서 왼쪽인지 오른쪽인지 확인 한 후 왼쪽이면 정수 형 변수 d를 1로 하고 a의 왼쪽 자식을 b와 p가 가리키게 한다. 오른쪽이면 정수형 변수 d를 -1로 설정하고 a의 오른쪽 자식을 b와 p가 가리키게 한다.

돌아가서, p부터 시작해서 삽입된 node까지 찾아 내려가면서 p가 현재 방문하고 있는 node의 위치를 기준으로 삽입된 node가 왼쪽이라면 1로 오른쪽이라면 -1로 설정 한다. 그 후 a가 가리키는 node의 BF에 d를 더했을 때 절대값이 2이상이 아니라면 아직 Tree가 균형을 유지하는 것이기 때문에 Tree를 재구성 하지 않고 insert를 종료한다. 만약 a의 BF가 2나 -2라면 균형이 무너졌기 때문에 Tree를 재구성 해야 하는데, 이 때 경우의 수를 LL, LR, RR, RL로 구분해서 재구성 해줘야 한다.

LL인 경우(a의 왼쪽 🡪 왼쪽 🡪 에 새 node가 삽입된 경우) 위에서 설정했던 a, b의 위치를 적절히 조절 해 주면 된다. b를 a의 위치로 보내고 b의 오른쪽자식을 a로 설정한다. b의 이전에 있던 오른쪽 자식은 a의 왼쪽 자식으로 설정한다. 그 후 a, b의 BF를 0으로 설정 해주고 b와 a의 부모였던 node와 연결해준다.

LR인 경우a, b에 추가로 c node가 필요한데 이 때 c는 b의 오른쪽 자식으로 설정해 준다. c를 a의 위치로 보내고 c의 왼쪽 자식을 b로 오른쪽 자식을 a로 설정 해준다. 그 후 원래 c의 자리였던 b의 오른쪽 자식을 c의 왼쪽 자식으로 설정 해 주고 원래 b의 자리였던 a의 왼쪽자식은 c의 이전 오른쪽 자식으로 설정 해 준다. 여기서는 a, b, c의 BF를 설정해 줄 때 c의 이전 BF에 따라서 선택 해야 한다. c의 이전 BF가 1이였다면 b의 BF는 0 a의 BF는 -1이고 c의 이전 BF가 -1이였다면 a의 BF는 0, b의 BF는 1이 된다.

c의 BF가 0인 경우도 있는데 (이 때 삽입 후 c의 BF가 0이 되려면 c의 BF가 이전에 1, -1이여야 된다는 전재가 있어서 a가 가장 아래쪽에 위치한 BF가 -1, 1인 node라는 전재에 모순이 된다 생각할 수 있다. 이 때는 c가 원래 존재하던 node가 아닌 새로 삽입된 node라고 생각하면 의문이 해결 된다. ) 이 때는 a, b모두의 BF를 0으로 설정해준다. 마지막으로 a의 부모였던 node와 c를 연결시켜 주면 Tree 재구성이 끝난다.

RR, RL은 LL, LR알고리즘에서 왼쪽 오른쪽만 적절히 바꿔서 적용 해 주면 된다.

Print

출력은 사전 편찬 순서로 해야 하므로 중위 순회를 사용해서 하면 된다.

Search

root부터 시작해서 같은 이름이 나올 때 까지 이름 비교해서 찾는 이름이 node내 이름 보다 작으면 왼쪽으로 크면 오른쪽으로 가다가 같은 이름을 발견하지 못하고 NULL을 만나면 false를 반환하고 찾았으면 찾은 이름을 출력해주고 heap에 해당 학생 정보를 push해 준다.

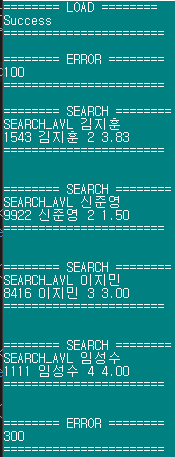
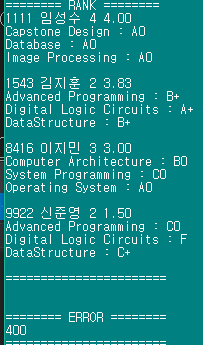
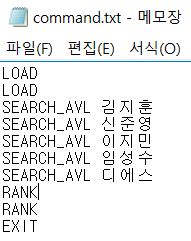
Heap

Push

heap의 가장 마지막에 위치한 node에 삽입할 data를 넣어준다. 그 후 그 값을 자신의 부모와 비교한다. 이번 프로젝트에서 사용한 heap은 max heap으로 가장 높은 값이 root에 위치한다. 그러므로 자신의 부모와 비교해서 부모 data가 더 작으면 자식 data와 위치를 바꿔준다. 이 동작을 root에 도착하거나 자신보다 큰 data를 가진 부모를 만날 때까지 반복한다. 자신보다 큰 data의 부모를 만났을 때 멈추는 이유는 더 위쪽의 node들은 부모 node보다 무조건 큰 값을 가지고 있을 것이기 때문에 비교하나 마나 이기 때문이다.

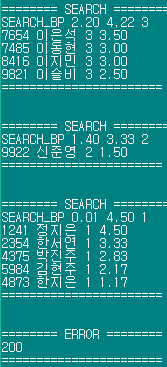
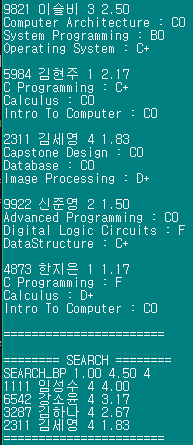
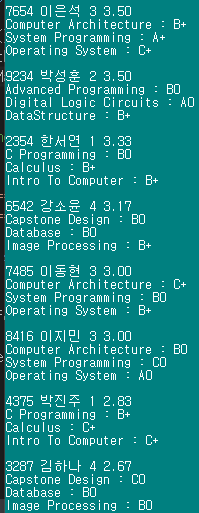
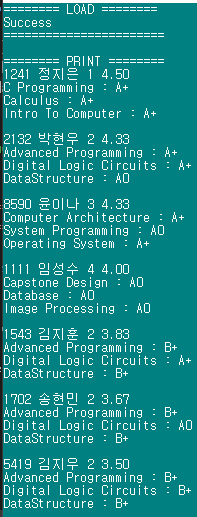
Pop

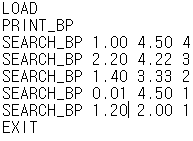
heap의 root에 위치한 값을 삭제 후 heap을 재구성 해야 한다. heap을 이루던 node개수가 하나 줄었는데 그렇다고 root node가 없어진 채로 둘 순 없으니 가장 마지막에 위치한 node를 지워야 한다. 그러기 위해서 가장 마지막 노드에 있던 data를 따로 저장해 놓고(임의로 last node라고 하겠습니다. ) root부터 자신의 자식 중 가장 큰 값을 가진 자식과 last node와 비교해서 last node가 더 크면 last node를 비어 있는 그 자리에 저장 하면 된다. 그렇지 않고 비어 있는 노드의 자식이 더 크면 그 자식을 비어 있는 node로 옮기고 그 자식이 있던 자리를 비어 있는 node라 생각하고 또 자식들과 last node와 비교를 진행한다. 계속 비교하다 last node가 크면 그 자리에 넣고 동작을 중지 한다.

1. Result Screen
2. LOAD, SEARCH\_AVL, RANK 명령어 검증

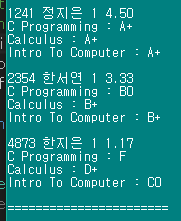
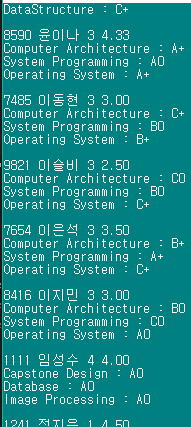
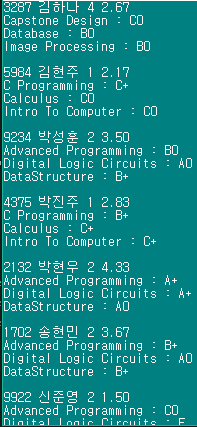
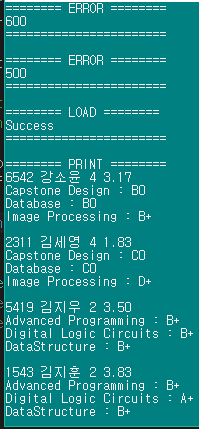
grade\_list는 홈페이지에 올라온 그대로 사용 했으며 가장 오른쪽 이미지가 이번에 사용한 command.txt이다. 일단 SEARCH\_AVL 명령어로 학생들을 검색하면 결과값이 제대로 나오는 걸 확인 할 수 있고 존재하지 않는 이름(Ex. 디에스)를 입력하면 SEARCH\_AVL의 ERROR CODE인 300을 출력 해 주는 걸 알 수 있다. RANK 명령어를 보면 방금 전 SEARCH\_AVL로 검색했던 학생들 중 데이터가 존재하는 학생들의 성적을 순위를 매겨서 순서대로 출력 해 주는 걸 알 수 있다. RANK 명령어 동작 중 하나인 heap에 있던 data들을 모두 제거 하는 기능을 두번째 RANK명령어를 보면 확인 할 수 있다. 출력 할 데이터가 없어서 RANK의 ERROR CODE인 400을 출력 해 주는 걸 확인 할 수 있다. 또 한 AVL쪽에서 출력이 잘 되는 것으로 보아 LOAD의 전체적인 동작까진 아니지만 부분적으로 잘 동작하는 것을 확인 할 수 있고 LOAD가 한번 되고 두번째 진행 될 때 LOAD의 ERROR CODE를 출력 해 주는 것을 확인 할 수 있다.

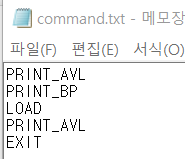
1. LOAD, PRINT\_BP, SEARCH\_BP 명령어 검증



LOAD후 PRINT\_BP를 통해서 B+ Tree에 저장된 학생들의 정보가 성적 순으로 출력 되는 걸 확인 할 수 있다. SEARCH\_BP 명령어를 통해서 찾고자 하는 범위와 학년 조건에 맞는 정보가 출력 되는 걸 확인 할 수 있다. 또한 조건에 맞는 데이터가 없으면 ERROR CODE를 출력 하는 것을 확인 할 수 있다.

1. LOAD, PPRINT\_AVL, PRINT\_BP 명령어 검증



먼저 LOAD명령어가 실행 되지 않아서 아무 데이터도 없을 때 PRINT\_AVL, BP 명령어를 실행했을 때 ERROR CODE를 출력하는 것을 확인 할 수 있다. 그 후 PRINT\_AVL을 입력 했을 때 학생의 정보를 사전 편찬 순으로 출력 해 주는 것을 확인 할 수 있다.

1. Consideration

1) 최재은

이번 프로젝트를 하면서 가장 큰 난관은 map에 대해서 이해하는 것이었다. 실습시간에 STL에 대해 조금씩 배우면서 map을 살짝 핥아만 보았던 기억이 나는데, 이번 과제에서는 그를 실질적으로 이용해서 구현하는 것이 목표였기 때문에 더 깊은 이해가 필요했었다. 특히나 상속받은 node의 함수를 사용하기 위해 부모노드에서 순수가상함수를 적어주는 것이 은근히 많았던 것으로 기억한다. map을 이해할 때, 그 내부 component를 확인하기 위해서 중복자를 사용해야 한다는 것이 매우 인상적이었다. 지금까지의 구현에서는 내부적으로 포인터를 갖고 있고 그 포인터가 하위 노드를 가리키게 하여 찾아가는 방식이었는데, map을 사용하니 전체적인 크기가 작아지는 듯한 느낌을 받았다.

정무영학생의 insert코드를 보고 무었이 잘못되었는지를 찾았다. 나는 저장할 datanode를 찾고 그 데이터 노드의 map에서 같은 key값을 갖는 component가 있는지 확인하고 있으면 그 component의 second에 학번과 학생정보를 넣는 방식으로 했는데, 굳이 그렇게 할 필요없이 그냥 해당되는 datanode를 찾아가서 insert하기만 하면 되는 것이었다. 이번 과제때 이런 알고리즘 적인 문제를 하나 잡지 못하고 제출했는데 이 부분이 문제였나보다. 그 이외에 데이터를 insert했을 때 split하는 것과 split하여 부모 indexnode에 index를 추가했을때 그 부모를 다시 split하는 것등은

내 것과 비슷했다. 이 부분에 있어서 대부분의 학생들의 알고리즘이 비슷하지 않을까 싶다. 가독성을 위해서 학생 정보를 print하는 부분에서 나는 출력하는 함수를 하나 따로 구현해서 사용했는데, 무영학생은 그냥 풀어서 반복문에서 조건에 따라 출력되도록 하였던 것이 달랐다. 이는 그렇게 중요한 부분이 아니기 때문에 상관없다고 생각한다. 그리고 하나 배운것은 중복자를 map의 end()로 하면 마지막 component의 다음 중복자 위치를 호출하는데, 이 때문에 마지막 component를 사용하려면 --를 해줘야 한다. 나는 이를 it = m->end();it--;라고 했는데 이를 그냥 it = --(m->end); 라고 하면 되는것이었다는 걸 왜 이제야 알았나 한다.

김영근 학생의 경우 insert에서 같은 key를 갖는 component가 있는지를 찾는 것을 m->find(key) == m->end()라고 조건문을 걸어서 사용했는데 이것을 보니 왜 조교님께서 저렇게 힌트를 주셨는지 이제야 이해가 간다. 저렇게 간단한 방법을 두고서 왜 반복문을 통해서 같은 key값을 갖는 component가 있는지를 확인했는지 나 스스로가 이해가 안간다..... 그 다음에 같은 key값을 갖는 component가 있는 경우에서 왜 잘 insert해놓고 key값을 갖는 component를 삭제했다가 다시 insert했는지는 이해가 가지 않는다. 그 이외에 split을 하는 조건이나 방식에 대한 알고리즘은 비슷한듯하다. 그 외에 SEARCH를 하는 방식이 내것과 조금 다른듯 해서 이런 방법도 있구나 싶었다. 코드 길이가 훨씬 짧고 조건문을 다중으로 사용하여 검색에 사용했다. 다른 사람의 코드를 보면서 고찰하는 것은 프로그래밍에 있어서 생각을 넓혀주고 또한 실력을 키울 수 있는 좋은 기회라고 느꼈다.

2) 정무영

정무영의 고찰

처음 프로젝트를 시작할 때 가상함수와 상속을 두루뭉술하게 알고 있어서 제공받은 코드를 그에 맞게 수정할 때 확신이 서지 않았다. 상속부분을 다시 공부하고 코드를 살펴보니 파해법이 보여서 학생 클래스의 작성을 완성 할 수 있었다. AVL tree는 수업만 들었을 뿐 중간고사 공부도 제대로 하지 않아서 코드 구현하려면 많은 시간이 걸릴 듯 했으나 강의자료에 AVL tree의 코드를 보니 90%이상이 완성되어 있어서 프로젝트에 맞게 약간의 수정만 했더니 잘 돌아가서 그대로 사용했다. 그래도 왠지 모를 불안감에 프로그램을 완성하고 다시 한번 짜봤다. 이번 프로젝트의 실질적인 부분은 B+ Tree였는데 처음에 map의 사용법도 생소하고 iterator가 도대체 무슨 역할인지도 몰라서 엄청 막막했으나 강의자료를 보면서 차근차근 공부하니 map의 사용법에 익숙해 질 수 있었다. map에 익숙해지니 프로젝트에 사용된 B+ Tree의 구조도 갑자기 눈에 들어오기 시작했다. Data split부분부터 그림을 그려가면서 차근차근 구현하다 보니 어느새 완성되었다. 완성 후 몇가지 테스트 도중 10개가 넘는 데이터를 넣었는데 B+ Tree부분에서 2개의 데이터 밖에 출력되지 않아서 당황했으나 다시 코드 그대로 그려가며 잘못된 부분을 찾아보니 split때 새로 할당된 node의 연결을 제대로 해주지 않아서 그런 것이었고 그 부분을 수정하니 잘 돌아갔다.

나중에 고찰 작성을 위해 조원의 코드를 보다가 내가 map의 find를 좀 더 다방면으로 사용하지 않았음을 알게 되었다. 그걸 느낀 부분이 내 코드의 datamap의 insert함수 였다. insert함수에서 중복 학점을 판별할 때 map의 begin부터 end까지 일일이 비교하는 코드를 사용했는데 그러지 않고 그냥 find로 검사해서 좀 더 코드를 간결하게 만들었어야 했다.

이번 프로젝트를 하면서 다시한번 머리속으로 생각하는 것보단 직접 그려가면서 하는게 구현과 이해에 도움된다는 것을 느낄 수 있었다.

조원 최재은에 대한 고찰

일단 AVL부분은 거의 동일해서 따로 비교할 만 한 게 없었다. Manager.cc에서 if else구문으로 명령어들을 구분해 내는 코드에서 명령어를 수행하고 continue로 바로 반복문의 시작부분으로 보내는 코드가 있었는데 어차피 if else구문이 한 가지가 조건 충족하면 나머지 구문을 무시하기 때문에 굳이 continue를 쓸 필요가 있을까 싶었다. B+ Tree에서 동일 학점을 찾을 때 나는 mapdata insert함수에서 찾았으나 이 코드에선 B+ Tree insert함수에서 찾고 있었다. 둘 다 별 차이가 없을 것 같다. 이 코드에서 중복학점을 찾을 때 map의 find를 사용하는 것을 보고 위에서 언급헀던 생각이 들었었다.

조원 김영근에 대한 고찰

주석이 엄청 자세하게 달려있어서 정성에 놀랐다. Manager.cc에서 Error Code 출력용 함수가 따로 주어져서 그걸 사용해서 Error Code를 처리했을 줄 알았는데 각 함수의 내부에 직접 Error code 출력 부분을 구현해 놔서 약간 비효율 적이라고 느껴졌다. 하지만 위에서 언급했던 find를 사용해서 중복 값을 찾는 코드가 생각했던 것과 거의 비슷하게 구현돼 있어서 다시 보게 됐다. 하지만 split을 insert함수에서 exceed의 return값에 따라 호출여부를 결정하지 않고 exceed내부에서 split을 호출 하고 있었다. 출제자가 exceed함수의 return형식을 bool로 한 것에 대해서 별로 신경을 쓰지 않은 것 같다.

3) 김영근

이번 프로젝트는 저번 프로젝트와 다르게 혼자 다 구현해야 했기 때문에 따로 파트를 나누거나 하지 않았다. 그래도 다행이었던 점은 저번 프로젝트에 비해 주어진 코드가 많았다는 것이다. 프로젝트는 크게 AVL과 B+Tree로 나눌 수 있었는데 AVL같은 경우 거의 제공이 많이 되어있었기 때문에 어렵지 않게 구현할 수 있었다. 특히 수업시간에 AVL insert를 제공해주었기 때문에 조금만 변형해서 그대로 넣으면 insert를 만들 수 있었고 SEARCH를 통한 RANK를 구현은 코드가 다 주어져 있었기 때문에 그대로 fout하면 되었다. 문제는 B+Tree에 있었는데 B+Tree는 insert가 주어져 있지 않았기 때문에 스스로 짜야 했다. 먼저 pdf를 통해 B+Tree가 어떤 규칙을 갖고 나눠지고 트리가 data와 index로 나눠져 있다는 것을 인지해야 했다. 다행히 insert와 searchDataNode는 주어져서 이것을 토대로 split를 짰는데 이 split을 구현하는데 정말 오랜 시간이 걸렸던 것 같다. 자잘한 오류가 계속 적으로 발생했고 최종적으로 다하기까지 많은 시간과 노력이 필요했던 것 같다. Split은 data와 index로 나눌 수 있는데 data를 구현하고 나면 index는 조금만 변형하면 되었기 때문에 자잘한 오류를 잡는데 집중 했다.

정무영 조원의 고찰

전체적으로 매우 잘 정돈된 느낌이 들었다. 깔끔했고 한눈에 봐도 이게 무엇을 실행하는지 알 수 있었다. 뭔가 조잡한 느낌이 나는 내 코드와는 달랐으며 주어진 코드를 하나도 변형안하고 그대로 쓴 것이 인상깊었다. 특히 B+Tree insert부분에서 만약에 같은 평균이면 따로 노드를 주는 것이 아닌 평균에 하나의 노드를 주도록 되어 있었는데 선언된 iterator에서 바로 second.insert를 사용하여 값을 넣어 주도록 코드를 구성하였는데 이런 식으로 map에 값을 주입할 수 있는지 몰랐는데 새롭게 배운 것 같다.

최재은 조원의 고찰

정무영 조원과 같이 주어진 코드 변형을 거의 안하였고 B+Tree를 제외한 부분은 구현하는게 비슷하였다. 최재은 조원도 insert 부분이 달랐는데 datamap을 insert할 때 선언된 map에 []를 통해 first를 지정하고 바로 뒤에 = second 값을 넣어 map에 data를 주입했는데 insert를 통해 값을 주입한 나로서는 또 다시 새로운 점을 배울 수 있었다.

내가 짠 코드와 다른 조원이 짠 코드를 비교해 보니 대체로 비슷하게 짠 것 같았다. AVL구현의 경우 다 주어져 있기 때문에 거의 똑같이 구현 되었다고 봐도 될 것 같다. 반면 B+Tree 다 달랐는데 먼저 exceed 조건문을 확인하여 split으로 넘기는 부분을 다들 inset에서 행해졌는데 나는 별다른 생각없이 exceed 조건에 그냥 바로 split문으로 넘겨 버렸는데 이렇게 하면 bool을 사용한 의미가 없어지는데 별 다른 생각 없이 쓴 것 같았다. 이것을 보고 manager에 구현된 bool들을 보니 똑같이 의미없이 bool문이 사용되고 있었다. 또한 manager에서 error 메시지가 출력되는 것을 그냥 지우고 각각 넣어줬는데 역시 다른 조원들은 그대로 활용 하여 error문을 출력하고 있었다. 다음부터는 좀더 주어진 함수들을 잘 활용해야 할 것 같다. 그 후 나머지 조건들은 조금씩 달랐지만 최종 구현되는 것은 같았던 것 같다. 확실히 다른 조원이 짠 코드를 비교하는 것으로 배울 것들을 많이 찾을 수 있었던 것 같다.