광운대학교

**데이터구조설계**

Project 2

제출일자: 2017년 11월 10일 (금)

학 과: 컴퓨터공학과

담당교수: 이기훈 교수님

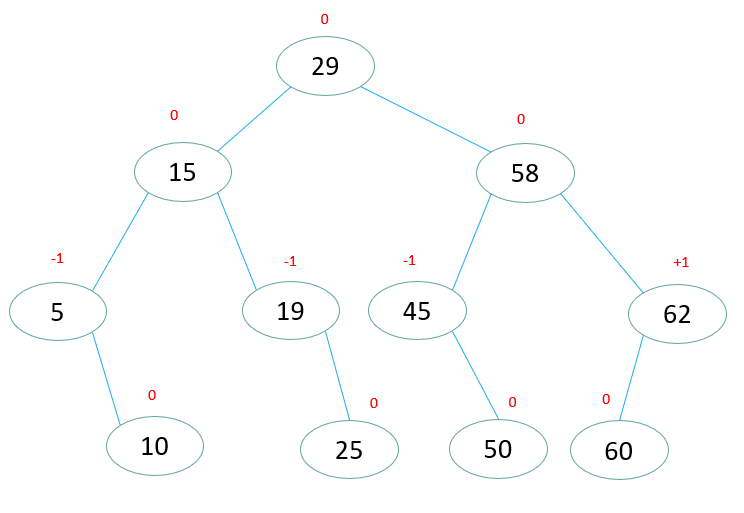
학 번: 2014722075

성 명: 이동준

O Introduction

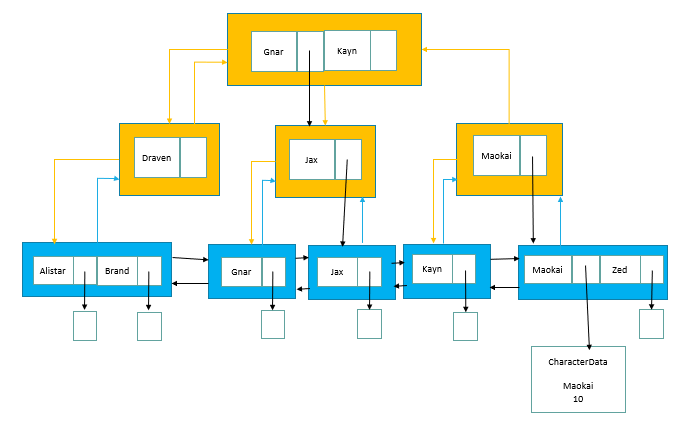
위 프로그램은 NoL게임 정보 관리 시스템으로 각 사용자(id)들의 게임 정보를 저장하고, 효율적으로 관리할 수 있다. 게임 정보에는 Game ID, Duration, Winner, Characters of team A, team B가 있다. Winner가 0이면 A팀의 승리, 1이면 B팀의 승리이다. 각 사용자들의 게임 정보는 AVL Tree를 이용하고, 각 게임에 사용된 캐릭터 정보는 B+-Tree를 통해 관리한다. AVL Tree는 Game ID를 기준으로 설계하였고, B+-Tree는 캐릭터의 이름을 기준으로 설계하였다.

\* AVL Tree



- AVL Tree는 BST의 단점을 보충하기 위해 설계되었으며, 한 노드의 오른쪽 subtree와 왼쪽 subtree의 level의 차이가 -1이상 1이하로 되도록 설정된 Tree이다. 기본 조건은 BST와 똑같이 노드의 data에 의해 구성된다. 각 노드들은 balance factor를 가지고 있으며 그 balance가 2이상 -2이하가 되면 LL,LR 등의 방법으로 새롭게 balance를 맞춘다. 위 프로그램에서는 한 node안에 각 사용자가 play한 하나의 게임 정보가 들어있다.

\* B+ Tree



- B+ Tree는 인덱스 노드와 데이터 노드로 구성되어 있으며, 각 노드 클래스는 B+ Tree 노드 클래스(BpTreeNode)를 상속받는다. 데이터 노드는 각 캐릭터들의 객체 정보를 map 컨테이너 형태로 저장되고, 인덱스 노드는 자식을 가리킬 포인터가 있는 map형태로 저장된다. B+ Tree는 order(한 노드에 들어갈 수 있는 맵의 개수를 구할 수 있음)를 가지고 있으며 그 수는 고정되어 있지 않다. B+ Tree는 캐릭터 이름을 기준으로 캐릭터 정보를 저장한다. 캐릭터 data를 입력하는 과정에서 처음 입력된 캐릭터는 새로운 객체를 생성하여 삽입하며, 이전에 참여했던 캐릭터가 다시 참여하는 경우는, 해당 객체의 Cpick을 업데이트하고 상황에따라 Cwin도 업데이트 한다.

\* Heap

- Max Heap – 캐릭터 선택 횟수를 기준으로 캐릭터 정보를 삽입하고 출력한다.(내림차순)

- Min Heap – 캐릭터의 승률을 기준으로 캐릭터 정보를 삽입하고 출력한다.(오름차순)

\* Command(command\_list.txt) – 각 명령어가 입력되어 있다.

- LOAD

: NoL 게임 정보를 저장하는 명령어로, game\_load.txt를 읽어 각 Tree를 구성한다.

Game ID Duration Winner Team A(5) Team B(5)순으로 입력되어 있으며 각 사용자들의 게임정보를 가지고 있다. ID의 범위는 1~1000, Duration의 범위는 1~10000로 한정하며 winner는 0또는 1뿐이다. 텍스트 파일이 존재하지 않거나 이미 LOAD가 완료되어 있는 경우에는 에러 코드(100)을 출력한다.

- UPDATE

: 게임 정보를 수정하는 명령어로, game\_update.txt.를 읽어 각 Tree를 수정한다.

Duration과 win, 게임에 참여한 캐릭터 이름만 수정하며, ID는 수정되지 않는다. LOAD와 마찬가지로 범위 밖의 입력은 없다고 가정한다. 만약 게임에 참여한 횟수가 0이 되는 캐릭터가 있더라고 B+ Tree에서 삭제하지 않는다. 텍스트 파일이 존재하지 않거나 입력되지 않은 ID를 UPDATE하는 경우, 에러코드(200)을 출력한다.

- SEARCH

: GAME이나 CHARACTER를 기반으로 정보를 검색하는 명령어.

Search\_game같은 경우에는 입력한 ID를 기반으로 AVL Tree에서 검색하여 반환한다.

Search\_character같은 경우에는 두 개의 캐릭터이름을 입력 받아 두 캐릭터 사이의 모든 캐릭터를 오름차순으로 출력한다.

만약 입력한 ID 정보가 존재하지 않는 경우 에러코드(300)을 출력한다.

입력한 캐릭터 중 하나라도 정보가 존재하지 않는 경우 에러코드(400)을 출력한다.

- PRINT

: game, character, pick, winratio를 기준으로 정보들을 출력하는 명령어

print\_game은 AVL Tree에 구성되어 있는 ID기준의 모든 정보들을 오름차순(ID)으로 출력한다.

만약 AVL Tree가 구성되어 있지 않은 경우, 에러코드(500)을 출력한다.

print\_character는 B+ Tree에 구성되어 있는 character기준의 모든 정보들을 오름차순(캐릭터이름)으로 출력한다.

만약 B+ Tree가 구성되어 있지 않은 경우, 에러코드(600)을 출력한다.

print\_pick은 B+ Tree를 탐색하여 캐릭터의 게임참여 횟수를 기준으로 내림차순(Max-heap)으로 출력한다.

만약 Heap에 저장되어 있는 정보가 없을 경우, 에러코드(700)을 출력한다.

print\_winratio는 B+ Tree를 탐색하여 캐릭터의 승률을 기준으로 오름차순(Min-heap)으로 출력한다.

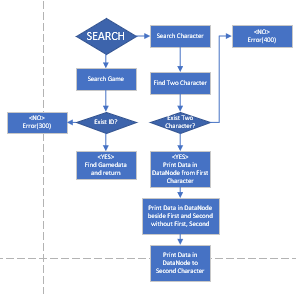
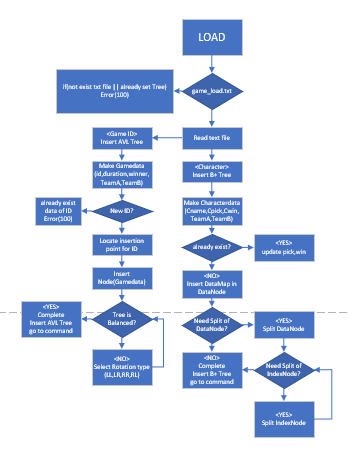
만약 Heap에 저장되어 있는 정보가 없을 경우, 에러코드(800)을 출력한다.

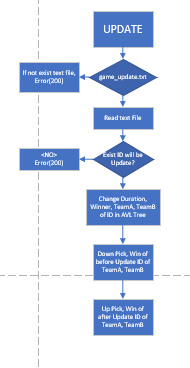
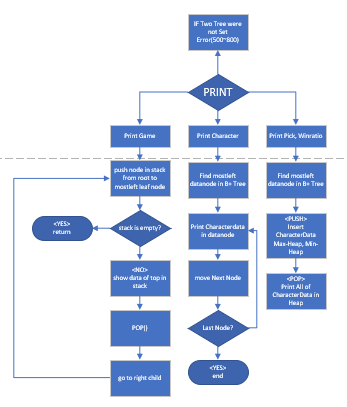
- EXIT

: 프로그램을 종료하는 명령어

# 각 명령어들이 완료됨에 따라 결과(+에러)를 log.txt.에 출력한다.

O Flowchart





O Algorithm

#Manager(run)

file.open(command text file)

if(LOAD)

file.open(load text file)

if(not exist text file)

error(100)

else

AVL Tree = GameData

if(already exist GameData)

error(100)

B+ Tree = CharacterData

load complete

else if(SEARCH)

if(GAME)

find data of ID

if(not exist ID)

error(300)

else

fout<<GameData of ID<<;

else if(CHARACTER)

find data of Characters

if(not exist first || not exist second)

error(400)

else

fout<<CharacterData from first character to second<<;

else if(PRINT)

if(GAME)

if(not set AVL Tree)

eroor(500)

else

fout<<all of GameData<<;

else if(PICK)

if(not set Heap)

error(700)

else

fout<<Pick Rank of CharacterData<<;

else if(CHARACTER)

if(not set B+ Tree)

error(600)

else

fout<<all of CharacterData<<:

else if(WINRATIO)

if(not set Heap)

error(800)

else

fout<<Winratio Rank of CharacterData<<;

else if(UPDATE)

file.open(update text file

if(not exist update text file)

error(200)

else

if(ID will be updated is not exist)

error(200)

else

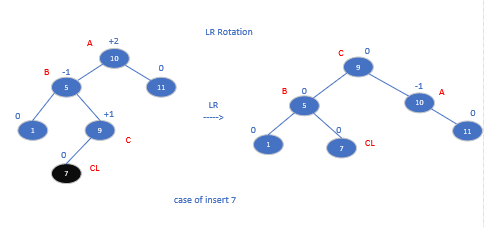
update GameData of ID in AVL Tree

update Character of GameData in B+ Tree

else if(EXIT)

end Program

# AVL Tree



- Insert(pGame)

if(root == NULL)

root = pGame

else

find insertion point for ID of pGame

make new AVLNode(pGame)

insert node in insertion point for ID

reset balance factor from root

while(Tree is unbalanced)

find position not unbalanced

decide rotation type

if(type is ‘LL’)

do LL rotation

if(type is ‘LR’)

do LR rotation

if(type is ‘RR’)

do RR rotation

if(type is ‘RL’)

do RL rotation

if(Tree is balanced)

break;

insert complete

- Print

d = root;

while(d != NULL)

push d in stack

d = Leftchild of d

if(stack == empty)

return;

d = top data of stack

delete top data of stack(pop)

print Gamedata of d

d = Rightchild of d

- Search(ID)

if(root == gamedata of ID)

return gamedata of root

find node want to search

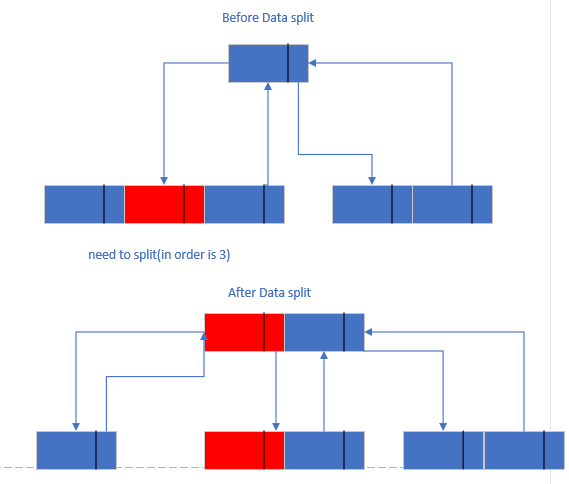
if (find)

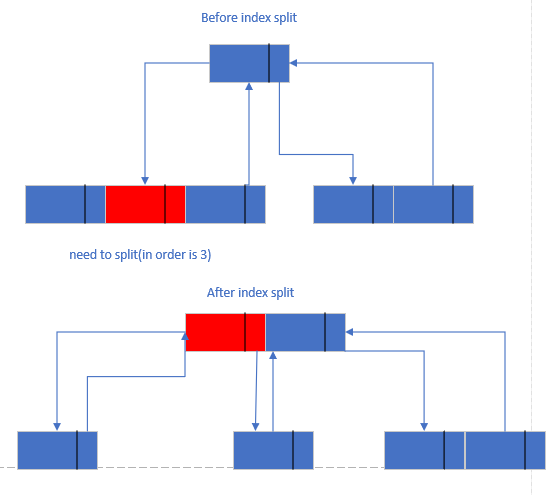
return gamedata of ID found

if(cant find)

return NULL;

# B+ Tree





- Insert(CharacterData)

if(exist CharacterData)

update pick,win of character

return false;

if(root == NULL)

make new BpTreeNode

insert data map of Character in new Node

root = new Node

return true;

find node can insert data map of character

after found insert data map

if(found node need to split)

splitDataNode(found node)

else

return true;

- SplitDataNode(pNode)

find map of pNode will be split

make new node(after split, it will be left node)

if(no parent, only root)

make index node

insert map

set new link

root = new index node

if(have parent)

if(pNode is mostleft of parent)

insert map in parent node

set new link(must be change mostleft)

else(the others case )

insert map in parent node

set new link

if(parent node need to split)

splitIndexNode(parent)

- SplitIndexNode(pNode)

find index map will be split

make new node

set new link for new node

if(parent == NULL)

make new parent node

insert index map in parent

set new link

root = parent

else if(pNode is mostleft of parent)

set new link(mostleft child will be changed)

insert map in parent

delete map in pNode

else if(pNode is not mostleft of parent)

set new link

insert map in parent

delete map in pNode

if(parent node need to split)

splitIndexNode(parent)

- SearchRange(start, end)

find dataNode1 had map of start

find dataNode2 had map of end

fout<<CharacterData in dataNode 1 from start

fout<<CharacterData in dataNode between start and end

fout<<CharacterData in dataNode2 to end

- Print()

Cur = find mostleft DataNode in B+ Tree

fout<<CharacterData in B+ Tree from Cur to mostright DataNode in B+ Tree

- printPickRank() – using max-Heap

while(visit all of dataNode)

push data in heap (data will be sorted according to Pick)

while(heap == empty)

fout<<top data of heap<<;

pop();

- printRatioRank() – using min-Heap

while(visit all of dataNode)

push data in heap(data will be sorted according to winratio)

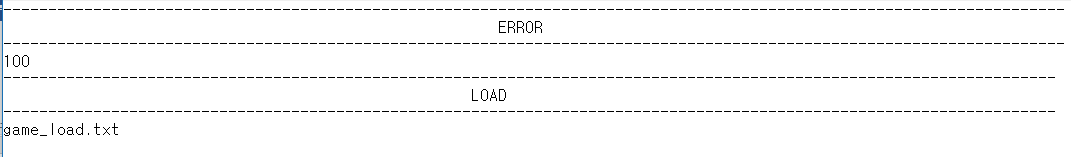
while(heap == empty)

fout<<top data of heap<<;

pop()

O Result Screen

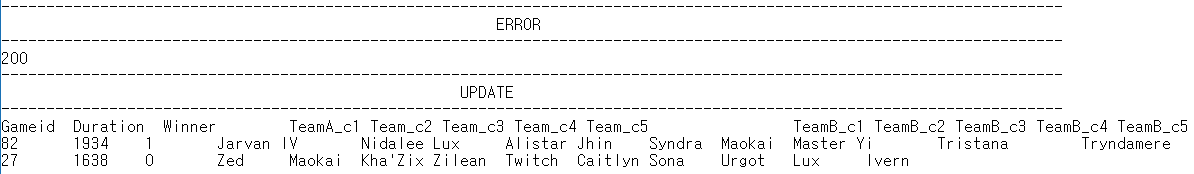
\* LOAD



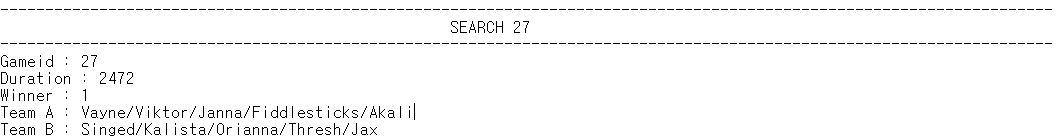
LOAD실행 후 결과화면이다. 만약 텍스트 파일이 존재하지 않거나 이미 트리가 구성되어 있는 경우 위의 그림처럼 ERROR를 출력하게 되고 정상적으로 LOAD가 완료되면 LOAD를 실행한 text파일 이름과 같이 LOAD를 출력하게 된다.

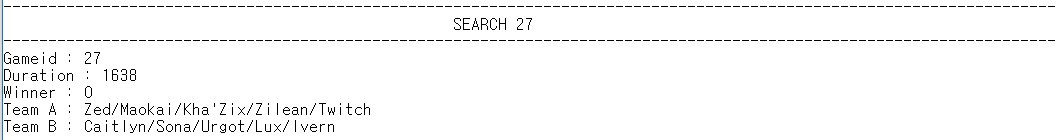
LOAD명령어는 text파일에 있는 게임 정보들을 받아와 AVL Tree와 B+ Tree를 구성한다.

\* UPDATE



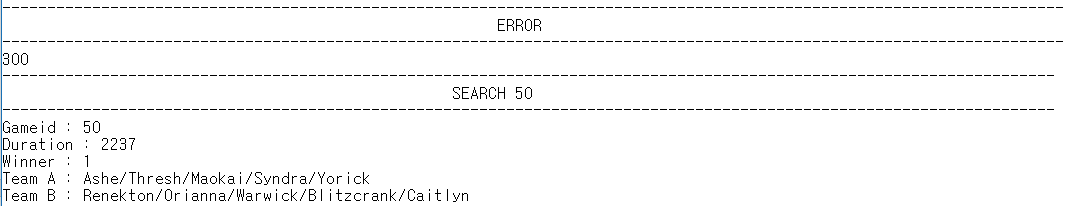
UPDATE 실행 후 결과화면이다. 만약 텍스트 파일이 존재하지 않거나 입력한 게임 ID가 존재하지 않을 경우 위와 같이 ERROR를 출력하게 되고 정상적으로 UPDATE가 완료된 게임데이터들을 출력한다. 만약 UPDATE과정에서 하나라도 잘못된 ID가 입력되었을 경우에는 UPDATE를 종료한다.





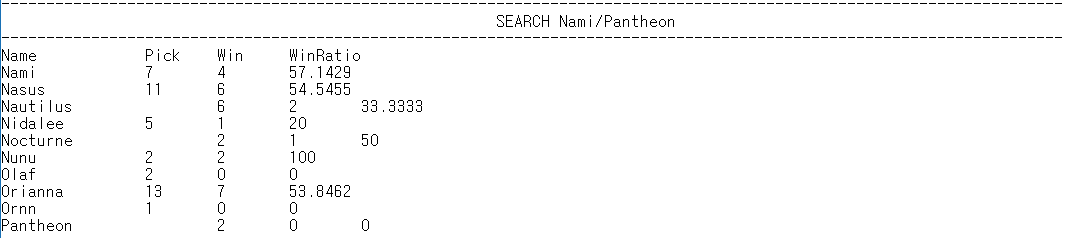
위 두 그림은 ID 27에 대한 업데이트 전 후의 그림이다 다음과 같이 Duration, Winner, TeamA, TeamB에 있어서 Update를 진행한다.

\* SEARCH\_GAME



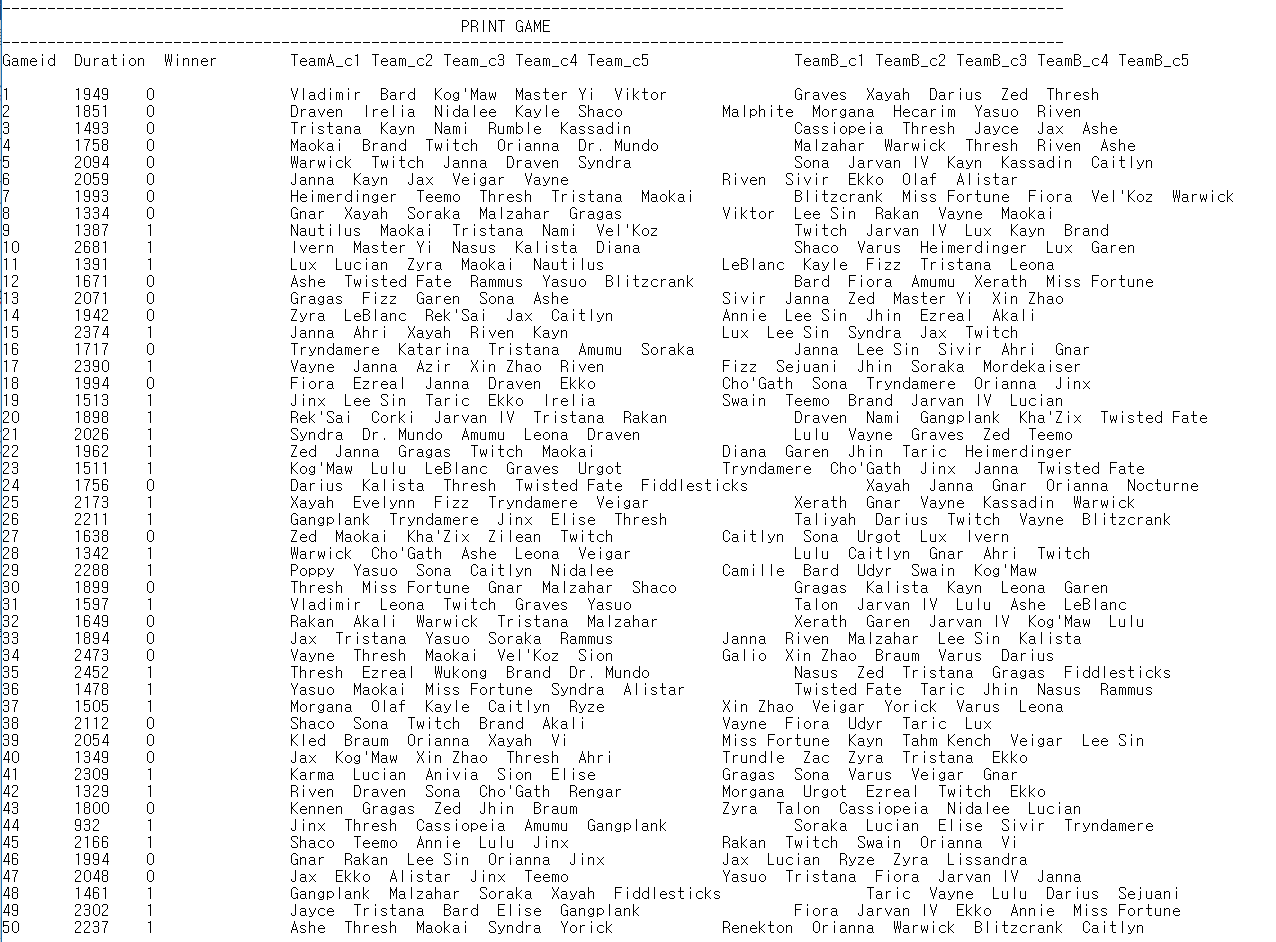
ID를 기반으로 AVL Tree에서 탐색하여 그 ID의 GameData를 출력한다. 만약 입력한 게임 ID가 존재하지 않을 경우에는 ERROR를 출력하게 된다.

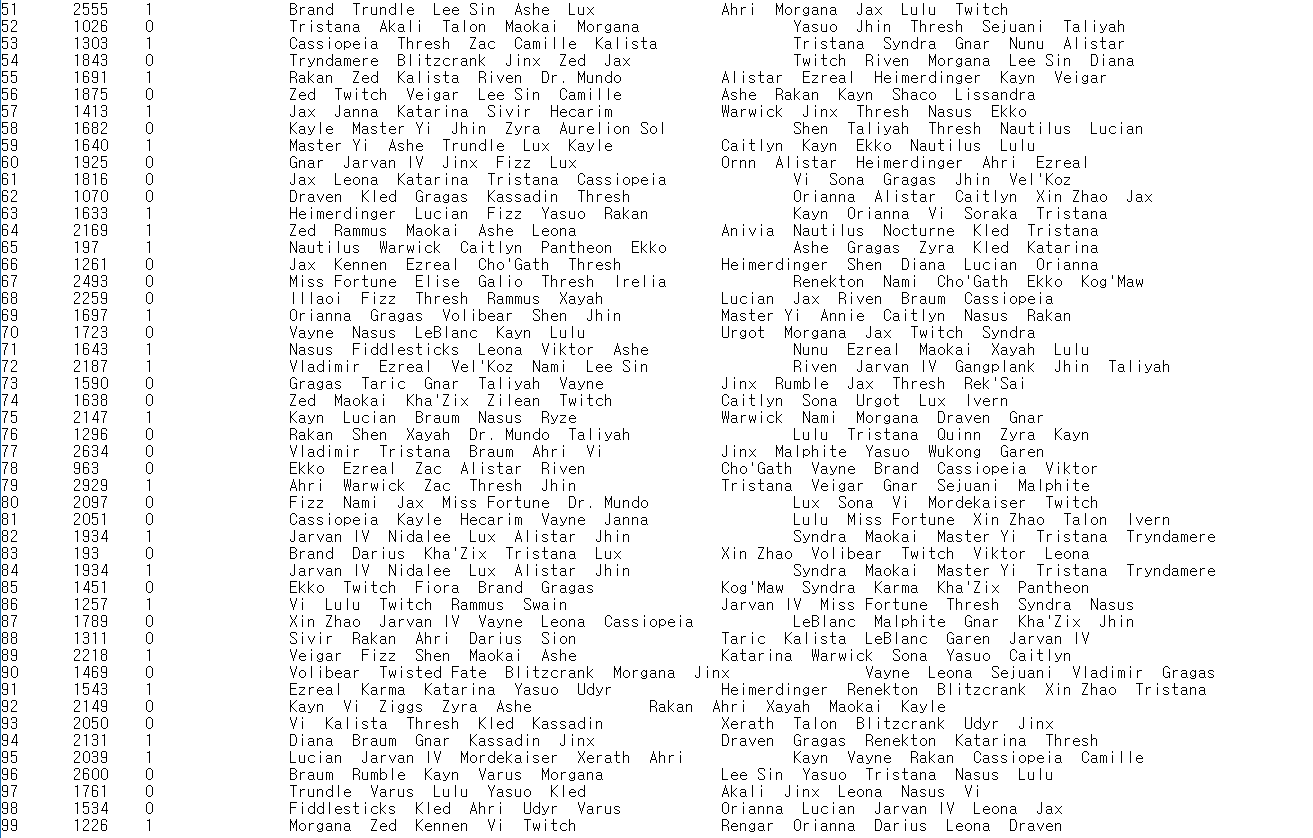
\* SEARCH\_CHARACTER



두 개의 캐릭터를 입력 받아 B+ Tree를 탐색하여 Start 캐릭터와 end 캐릭터사이에 characterData를 출력한다. 만약 입력한 두 캐릭터 중 하나라도 data가 존재하지 않는 경우, ERROR를 출력한다.

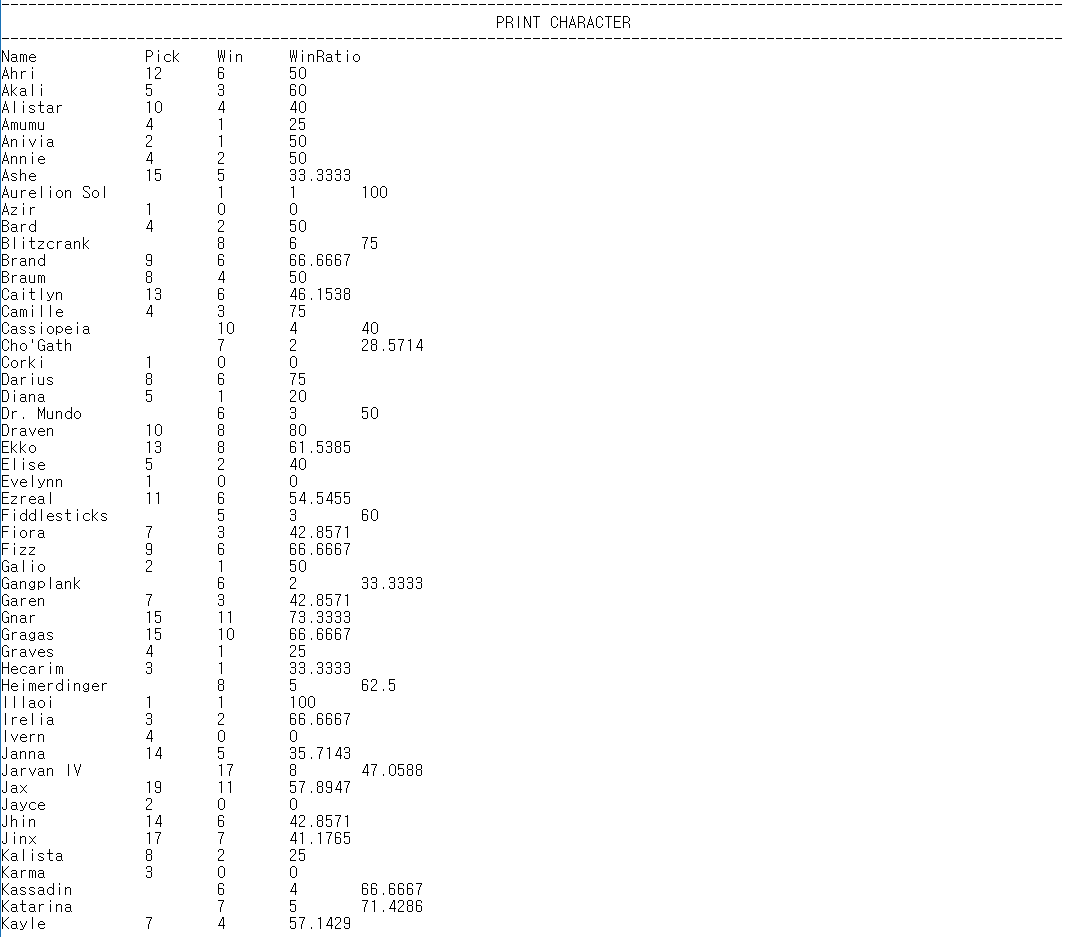
\* PRINT\_GAME



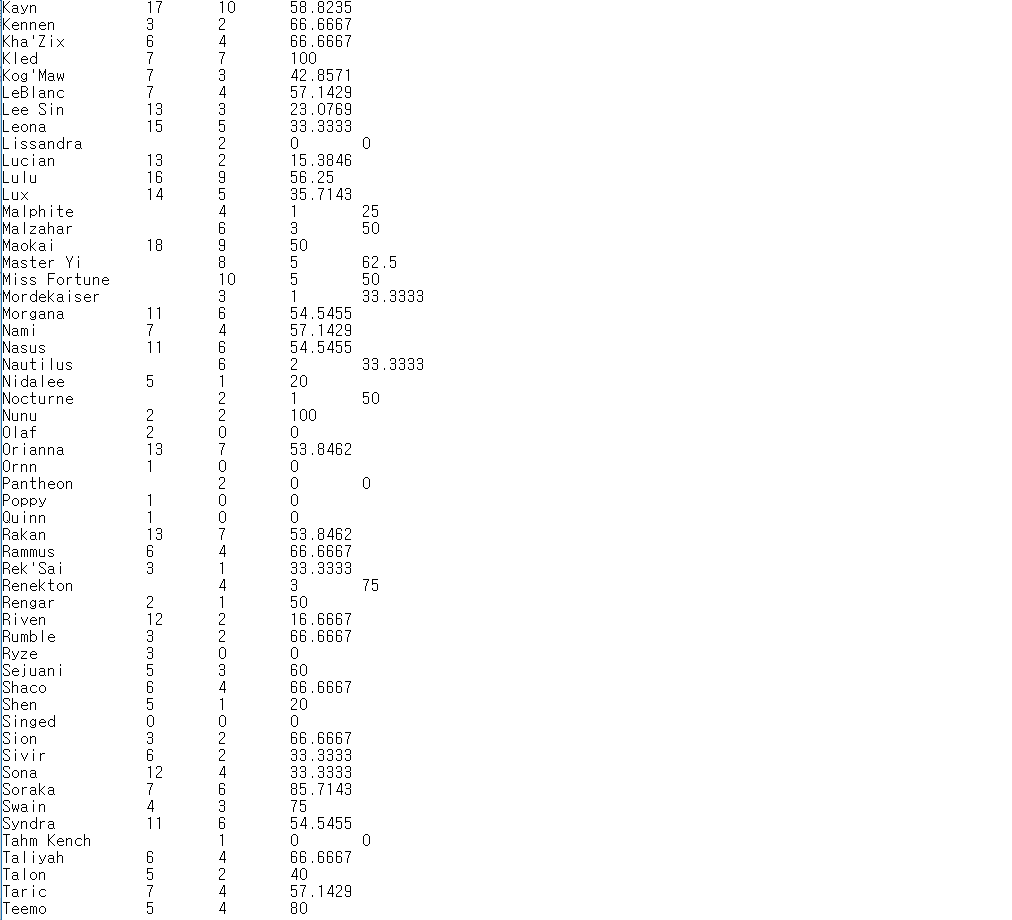


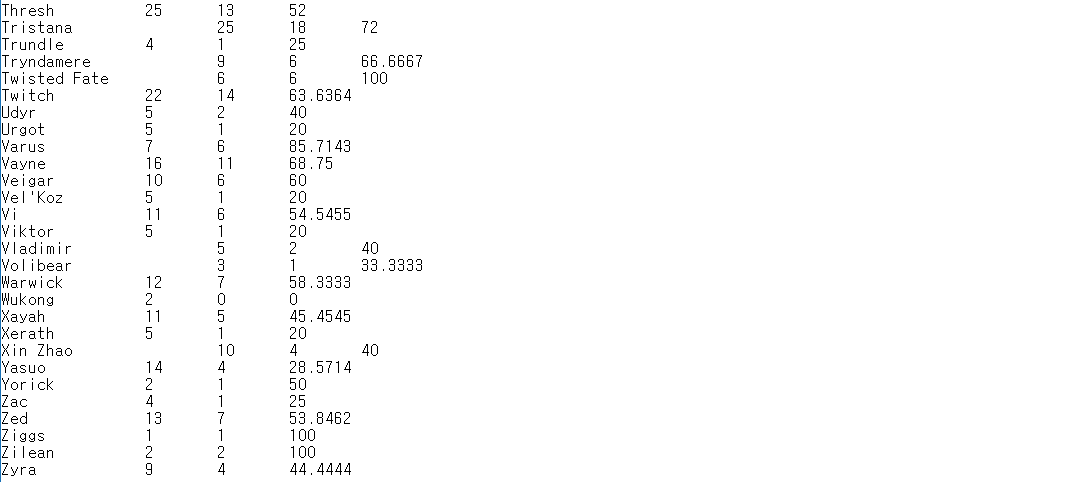
AVL Tree에 있는 모든 GameData를 ID의 오름차순으로 출력을 한다. 만약 AVL Tree가 구성이 되어있지 않은 경우 ERROR를 출력한다.

\* PRINT\_CHARACTER

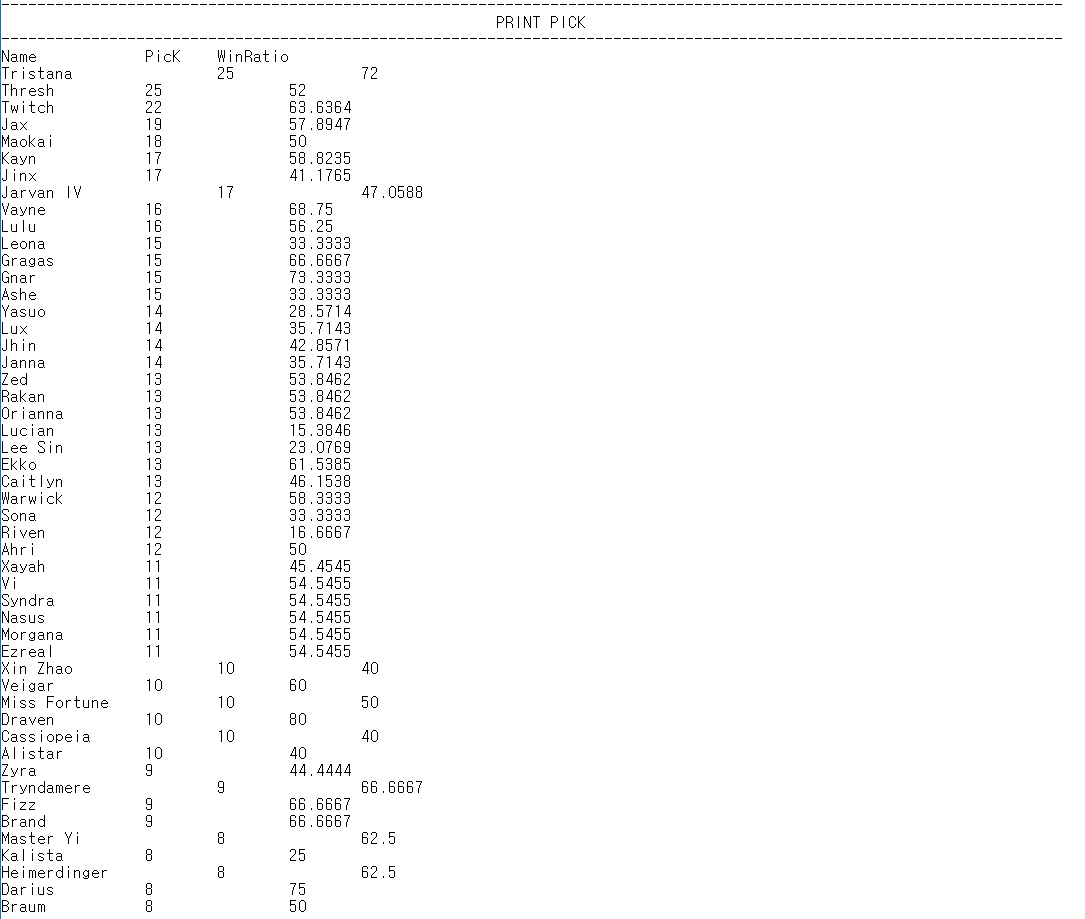


B+ Tree에 있는 모든 CharacterData를 캐릭터 이름의 오름차순으로 출력한다. 만약 B+ Tree가 구성되어 있지 않은 경우 ERROR를 출력한다.

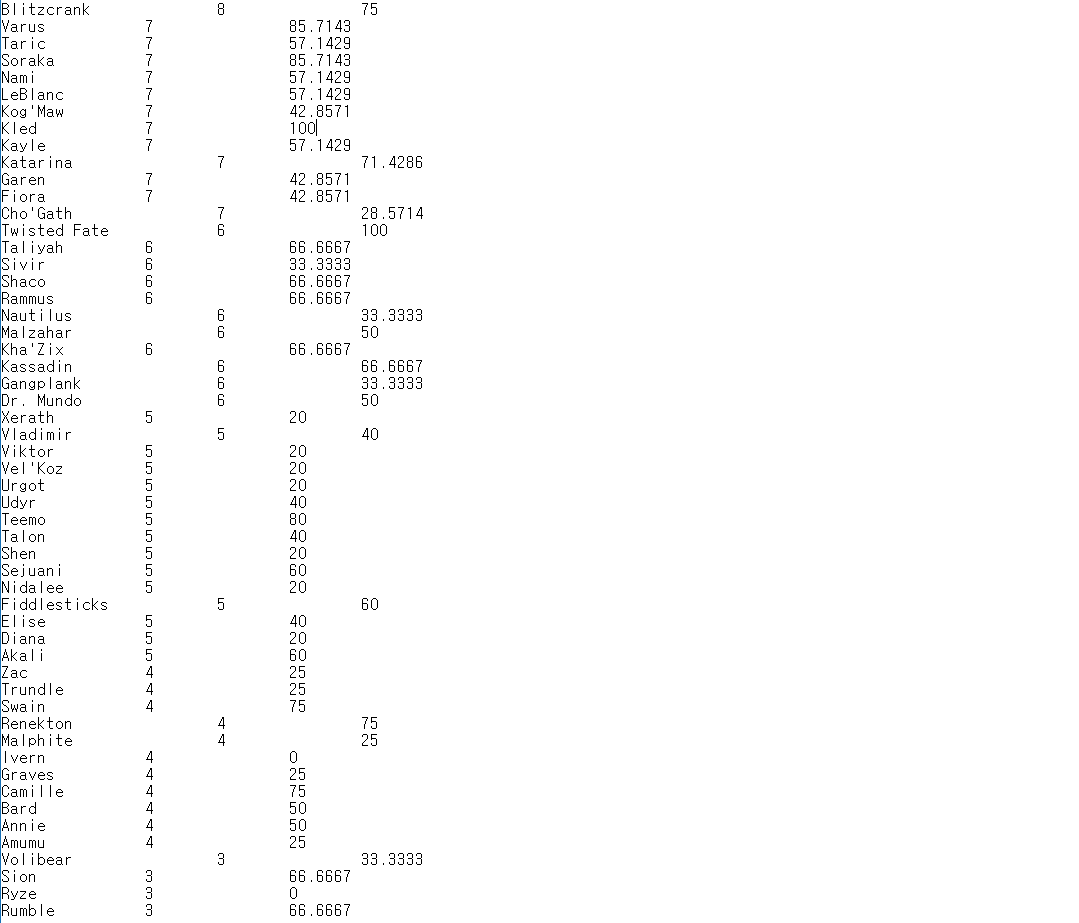


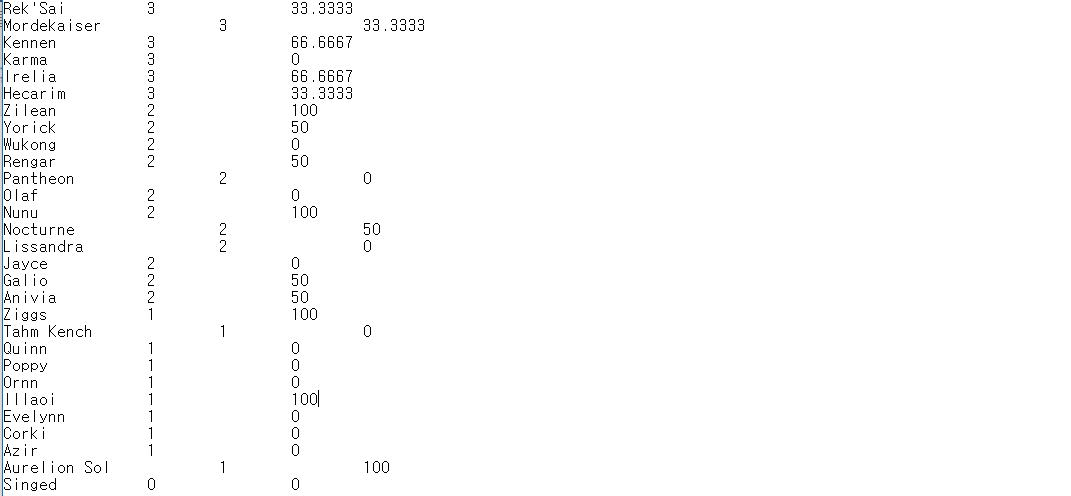


\* PRINT\_PICK

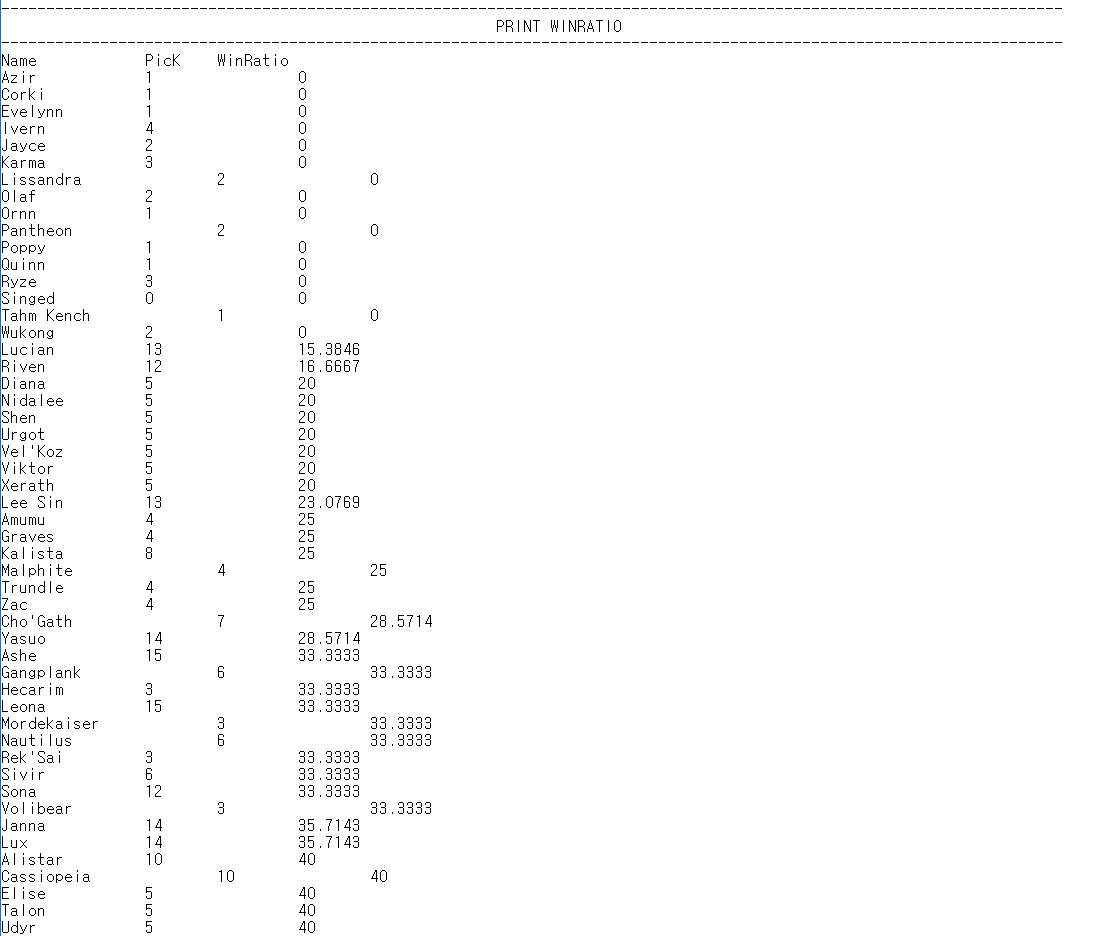


B+ Tree에 있는 모든 CharacterData를 Pick의 데이터로 max-heap에 push하여 가장 높은 pick의 data부터 pop을 하여 출력한다. 만약 heap이 구성되어 있지 않은 경우 ERROR를 출력한다.

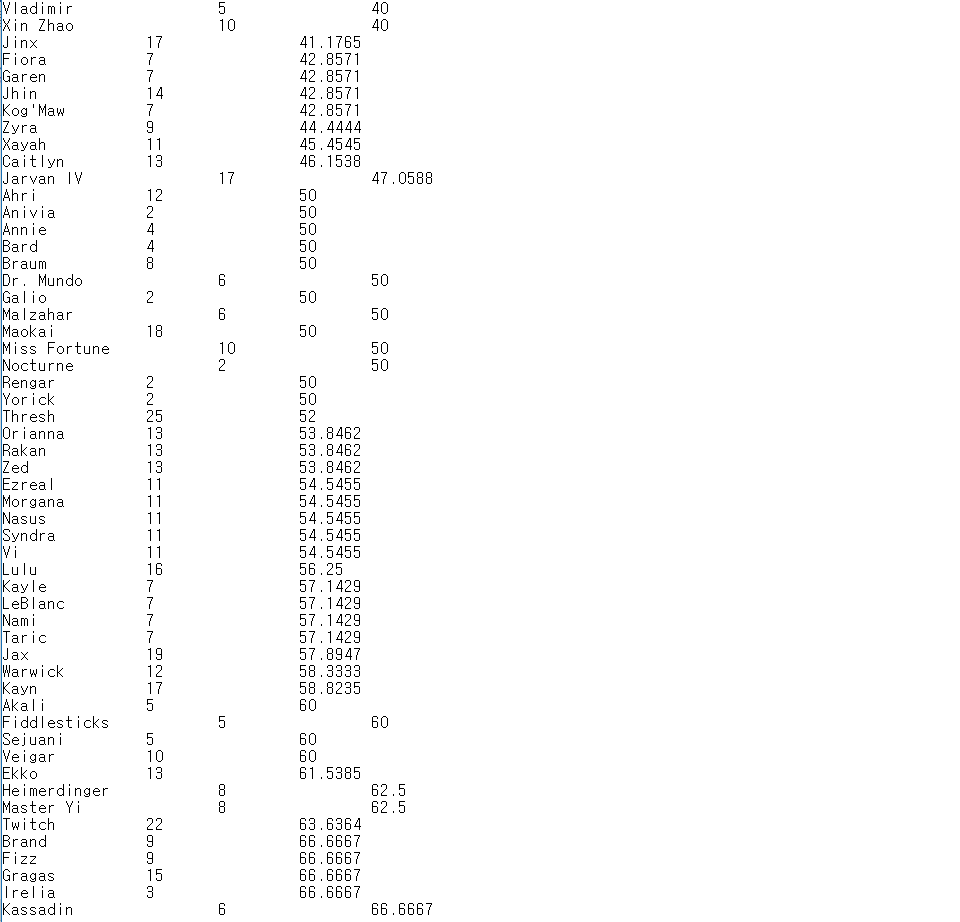


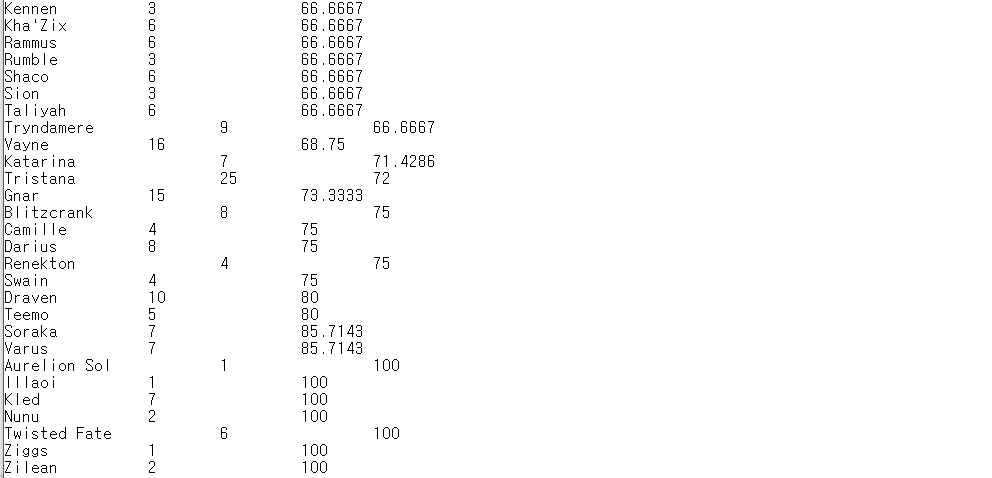


\* PRINT\_WINRATIO



B+ Tree에 있는 모든 CharacterData를 winratio의 데이터로 min-heap에 push하여 가장 낮은 승률의 data부터 pop을 하여 출력한다. 만약 heap이 구성되어 있지 않은 경우 ERROR를 출력한다.





O Consideration

이번 프로젝트에서는 새로운 Tree를 이용하여 설계를 하였다. AVL Tree와 B+ Tree를 이용하였는데, 처음에는 AVL Tree의 rotation방법과 어떤 경우에 어떤 rotation을 해야 하는지 이해가 가지 않았다. 마찬가지로 B+ Tree에서 data split과 index split에 대한 이해도도 떨어졌다. 프로젝트를 준비하면서 공책에 rotation과 split을 직접 그리면서 다양한 케이스를 따져보니 명확한 해법을 얻어냈고, 그 결과 프로젝트를 완성할 수 있었다. B+ Tree에서 map이라는 새로운 개념을 이용하였는데 처음에는 왜 굳이 이런 걸 쓰나 싶었지만, 설계를 하면 할수록 효율성이 뛰어나다는 것을 알게 되었다. 이번 프로젝트를 진행 하면서 상속에 대해서도 알게 되었다.

다양한 시도 끝에 프로젝트를 완성할 수 있어서 뿌듯하고 3차 프로젝트로 성공적으로 완성 해야 겠다.