Documentation for Project: B+ Tree

Assumption made:

1. We assume that there will be no replicate key in the pages.
2. We assume to preferentially use the left sibling for redistribution than right and merge from right sibling into left in Delete.

Description for the methods:

Throughout the description, we will not discuss PIN and UNPIN except in the constructor and deconstructor as so many methods used them. Most of implemented help functions are implemented for small tasks, and the details for those methods will not be discussed. I will only mention the purpose of the functions unless it is necessary. Also, we will only focus on the normal cases and exceptional cases will not be discussed in the documentation as there are so many of them.

BTreeFile::BTreeFile

For this method, I simply follow the instruction of the coursework. If the file with the filename does not exist, it returns FAIL and it creates a new file with the filename given using the methods provided in instruction. I create a private two members for the BTreeFile in btfile.h:

```
    void setRootPid(PageID pid) { rootPid = pid; }
    void setFileName(const char* filename){fname=filename;}
```

rootPid records the current root page id for the file. fname stores the current fname of the btfile for later usage (for DestroyFile). Also, two private method are create for setup the root page id and filename for the convenience.

If the file exist, I firstly pin the page use the pid returned by the GetFileEntry method, and setup the root page id and the filename.

```
        setRootPid(pid);
        setFileName(filename);
```

In both conditions, unpin the page at the end of the constructor.

```
MINIBASE_BM->UnpinPage(pid,DIRTY);
```

I think it is dirty because I have made the changes to the page by setting up the file name and root page id.

BTreeFile::~BTreeFile && BTreeFileScan::~BTreeFileScan

There are both empty as I do not always keep the root page id pined, and always unpin the page after usage. The default deconstructor would be enough for our task.

BTreeFile::DestroyFile

For this method, I create a help function DestroyFileHelper that has input of PageID curPid. It returns Status OK if successful, FAIL otherwise.

```
Status BTreeFile::DestroyFileHelper(PageID curPid)
```

It recursively free the pages by checking the page type of the current page. If the current page is index page, it calls DestroyFileHelper on each child page, and free each child page, return OK; If the current page is a leaf page, simply return OK. The function return FAIL if any error occurs in the process.

BTreeFile::Insert

In this method, we use a help function InsertHelper that recursively find the page to insert the key and the record, and propagate the information of the child note to the parent page.

```
Status BTreeFile::InsertHelper(const int key, const RecordID rid, PageID curPid,
bool& split, int& child_key, PageID& child_pageid)
```

It has input of key, rid and PageID of the current page, and it has output of a boolean variable “split”, a key “child_key” and PageID “child_pageid”. If the child node needs to be split, it propagates this information to the parent and “child_key” and “child_pageid” will be the key and pageID needs to be inserted into the index node. It returns OK if the insertion succeeded and FAIL otherwise.
In the InsertHelper function, if the current page is an index node, it uses an implemented method FindPageWithKey that determine which page does the key belongs to.

```
Status BTIndexPage::FindPageWithKey(const int d_key, int& key, PageID& pid,
RecordID& rid)
```

FindPageWithKey is implemented in the btindex.cpp as a private method. Note that if the aiming key is smaller than the first key in the index node, it returns the leftlink of the index page. If the child node of the current index node needs to be split, try to insert the propagated key and pageID into the current index node. If the current index node is full, create a new index node and split the current index node into two using an implemented function Split_Index. newPageKey sets up the value of child_key.

```
Status BTreeFile::Split_Index(BTIndexPage* oldPage, BTIndexPage* newPage, const int
key, PageID pid, int& newPageKey)
```

If the current page is a leaf node, it checks whether if the leaf node is full, and insert the key and record into the leaf node if it is not full. If it is full, split the leaf node into two using function Split_Leaf.

```
Status BTreeFile::Split_Leaf(BTLeafPage* oldPage, BTLeafPage* newPage, const int
key,const RecordID rid)
```

Note that if the split returned by the InsertHelper in the Insert function is true, it means that the root node needs to be split, and we need to split it.

<u>BTreeFile::Delete</u>

In Delete method, it firstly uses an implemented function FindPageWithKeys, which is a private method for BTreeIndex and implemented in btindex.cpp.

```
Status BTIndexPage::FindPageWithKeys(const int d_key, int& key,int& nextKey,
PageID& pid, PageID& prevPid, PageID& nextPid, RecordID& rid)
```

It has input of d_key which is the aiming key, and return the key and pid (PageID) where the aiming is belong. It also returns the neighbour nodes information for later use. Those returned information is used as input in an implemented method DeleteHelper, which helps to recursively find the page that has aiming key and rid and delete, similar to InsertHelper.

```
Status BTreeFile::DeleteHelper(const int key, const RecordID rid, const int
curKey,const int nextKey,PageID curPid, PageID prevPid, PageID nextPid,bool&
underflow,bool& merged, int& child_key, PageID& child_pageid,int& deletedKey)
```

It has bool “underflow” that tells whether child node is underflow and bool “merged” tells whether if the method to solve the underflow is merging. “child_key” and “child_pageid” propagate the information for the insertion and “deletedKey” for deletion to the current page.
In the DeleteHelper, if the current page is a leaf node, it firstly delete the aiming key and rid, and then check if the underflow occurs. If it occurs, it uses function IsAtLeastHalfFullAfterDelete, a function implemented in both BTLeafPage and BTIndexPage to checks if the neighbours are available for sibling redistribution (firstly check the left sibling then check the right sibling).

```
bool IsAtLeastHalfFullAfterDelete()
```

If it is available, then use implemented functions DeletedLeaf_prev, DeletedLeaf_next to do the redistribution.

```
    Status DeleteLeaf_prev(BTLeafPage* prevPage, BTLeafPage* curPage, int& newKey);
    Status DeleteLeaf_next(BTLeafPage* nextPage, BTLeafPage* curPage, int& newKey);
```

If both siblings are not available, then apply the merge method using MergeLeaf_next if the current page is not the rightmost page or MergeLeaf_prev otherwise.

```
    Status MergeLeaf_prev(BTLeafPage* prevPage, BTLeafPage* curPage);
    Status MergeLeaf_next(BTLeafPage* nextPage, BTLeafPage* curPage);
```

Similar procedures carried out if the current page is an index node. Instead of deleteing the aiming key given, it deletes the propagated child_key and child_pageid if underflow occurs in the child node. However, no help functions like DeleteLeaf_prev implemented as I felt it is more convenient.

BTreeFile::OpenScan

For this method, it creates a BTreeFileScan object which has private member int lowKey and highKey, key_scanned, Status s, PageID curPid, RecordID dataRid, BTreeFile * btfile. If the lowKey is nullpointer, use GetMinimumPid to find curPid which is the PageID of the first record in the BTreeFile and setup the lowkey to be the key of the first record. Status s is initially set to OK unless rootpid is INVALID_PAGE. btfile is set to the current BTreeFile when the scan is created.

```
PageID BTreeFile::GetMinimumPid(int & key, int & height )
```

If the highKey is nullpointer, use GetMaxKey to set up the highKey.

```
PageID BTreeFile::GetMaxKey(int & key)
```

BTreeFile::DumpStatistics

Sum_Index_nodes and Sum_leaf_nodes are used to sum the index nodes and leaf nodes information. Then print those information out as required in the coursework introduction

```
Status BTreeFile::Sum_Index_nodes(PageID curPid, int& nodes, int& num_entries,
float& sum_fill, float& max_fill,float& min_fill
Status BTreeFile::Sum_leaf_nodes(PageID pid, int& nodes, int& num_of_records,
float& sum_fill, float& max_fill,float& min_fill )
```

BTreeFileScan::GetNext

It uses an implemented method GetNextHelper to return the next key and rid. In the GetNextHelper function, if the last key in the page returned by function GetLast is lower than the lowKey or the last key is scanned, then call GetNextHelper on the next page. lowKey, key_scanned and dataRid set to the current key and rid just scanned. curPid is updated if moves onto the next page.

```
Status BTreeFileScan::GetNextHelper(PageID pid, RecordID & rid, int& key)
Status BTLeafPage::GetLast(int& key, RecordID& dataRid, RecordID& rid)
Status BTIndexPage::GetLast(int& key, PageID& pid, RecordID& rid)
```

GetLast is also used in other places to retrieve the last record as needed for my implementation.

BTreeFileScan::DeleteCurrent

Simply use the Delete method from BTreeFile* btfile that is set initially when the scan is created. The input for the Delete method is key_scanned and dataRid in the private member.

Tests:

1:The code succeeded for running the sample test.txt.

2:I have tried to insert and delete records with changing to the height of the tree. This is important because the tree in test.txt has height 0 (only one leaf node) and the code could fail for deeper tree.

```
insert 10 130
scan -1 -1
print
delete 15 100
scan
print
stats
quit
```

3:I have tried to insert and delete with more records where the height increased to 2. This is important test because it tests whether the insertion and deletion in leaf nodes correct propogate to all the index nodes including the root node.

```
insert 10 4000
scan -1 -1
print
delete 10 4000
scan
print
stats
quit
```

4:I have also tested null and non-null pointer of lowKey and highKey and they all worked.

5:Insertion of so many records would fail, which is normal as discussed on piazza.

```
insert 10 100000
```