Project 2 Document

Author: Yan Wang & Zishan Qin

Project Requirement

1. An indexing mechanism:B+ tree or Hash function index..
2. Five interfaces related to: Put, Get and Remove.
3. Extra issues: multiple threadings, value-based search

Assumption

The total size of dataset will not exceed the limit of memory

Design

1. We built a class called Btree for building up our index mechanism and doing Put/Get/Remove operations. Btree is composed by nodes, each node has a list of entrys. Btree class is used for maintaining the tree size, tree height, leaves list and root information. Node is used for maintaing the usage of entries in this node. Entry is special : for internal nodes,only key and next pointer will be used; for external nodes, only key and value will be used.
2. The dataset used in this project is a list of words for indexing the bag-of-word data(Supported by UCI datasets). It's a Key-Value pair dataset and the size of tuples is more than 32000. So we can make the fanout of b+ tree as 8. So for get/put/remove operations, only 6 times look-ups will be needed.
3. We will index on Data attribute for the dataset. Although the description of the project only ask for two kinds of Get/Put/Remove Key-Value pair data types(`void Put(string key, Number data_value);` or`void Put(string key, string data_value);`). We implemented using a more general way- using Java Generics to abstract the Key-Value pair,as BTree<Key extends Comparable<Key>, Value>. Key and Value can be any object type(Integer/String/Date...) as long as the Key is a comparable object type. So we only need to implement 3 functions but the codes are applicable for the five interfaces in the project description.

   For Put/Get/Remove functions, we implemented in this way, supposed the fanout is m:

   Get: we start from root, based on comparison of keys of upper layer, we step into next layer untill we get height=0.When height=0, we are in the Leaf-layer, we locate the item according to its key. The time complexity is logm(n).

   Put:we start from root, locate the biggest leafnode which is smaller than the inserted item, we try to insert it in the entries of that leaf node. If the entries size is less than M, then we are done. If node, we need to split that node into two and insert the smaller spliited node into its parent entries.If the parent is full, split it too, repeat the split process above until a parent is found that need not split.

Remove:We implemented two remove() functions in the project.

One is based on Data. For this type, perform a search to determine which leaf node contains the key. Remove the key from the leaf node. If the leaf node is at least half-full, done! If the leaf node as L is less than half-full: Try to borrow a key from sibling node as S (adjacent node with same parent). If S is L's left sibling, then borrow S's last key, and replace their parent navigate key with this borrowed key value. If S is L's right sibling, then borrow S's first key, and replace their parent navigate key with S's second key value. If can not borrow a key from sibling node, then merge L and sibling S.

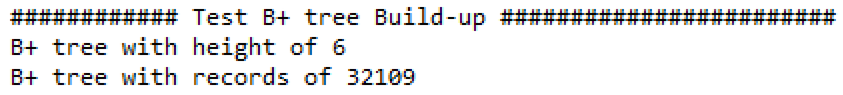
After merged L and S, delete their parent navigate key and proper child pointer. Repeat the borrow or merge operation on parent node, perhaps propagate to root node and decrease the height of the tree.

Another remove() is based on Key filed, which is required in the project description. Since we build the index on data field, we cannot use key to locate where should we delete. Our implementation is scanning the leaves node list, which is linked together in the order of Data. Find related tuples and call Remove(Data) to remove these tuples.

4. All three methods “put”, “get”, “remove” are synchronized. Every method will wait until it is notified by other threadings. After the method is implemented, it will notify other threadings.

Test

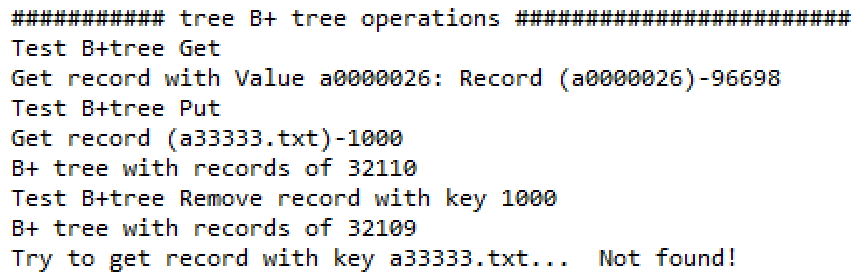
Test indexing mechanism:
we read data from the dataset line by line and simutaneously, insert these data into btree. After that we check the properties of that Btree, the height is 6 and there are 32109 records stored.



```
############ Test B+ tree Build-up ########################
B+ tree with height of 6
B+ tree with records of 32109
```

Test Five interfaces related to: Put, Get and Remove.
in this test we do folloing things:

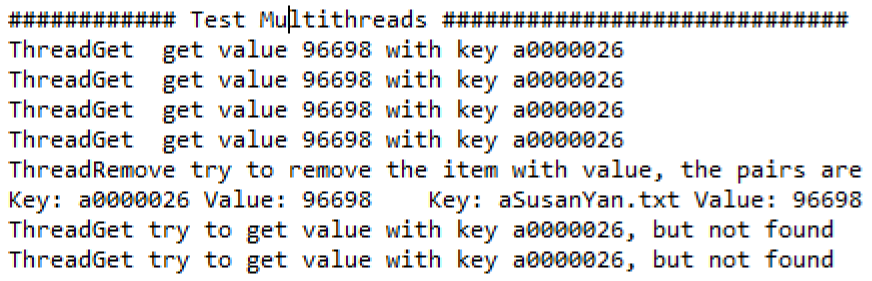
1) Get(key=a0000026)
2) Insert(a3333.txt,1000) into btree.
3) Get(key=a3333.txt);
4) Check the tree size: notice there the size of tree is increased by one.
5) Remove(Key=a3333.txt);
6) check the tree size:notice there the size of tree is decreased by one.
7) Get(key=a3333.txt)



```
########### tree B+ tree operations ########################
Test B+tree Get
Get record with Value a0000026: Record (a0000026)-96698
Test B+tree Put
Get record (a33333.txt)-1000
B+ tree with records of 32110
Test B+tree Remove record with key 1000
B+ tree with records of 32109
Try to get record with key a33333.txt...  Not found!
```

Test multiple threadings

Multiple threadings are generated to mimic multiple client. Except for the first threading, we don’t control the order of other threadings. Therefore, we generated a bunch of threadings for “get”, so as to display the “put” and “remove” go well. Result of this test shows below.



```
############ Test Multithreads #############################
ThreadGet  get value 96698 with key a0000026
ThreadGet  get value 96698 with key a0000026
ThreadGet  get value 96698 with key a0000026
ThreadGet  get value 96698 with key a0000026
ThreadRemove try to remove the item with value, the pairs are
Key: a0000026 Value: 96698    Key: aSusanYan.txt Value: 96698
ThreadGet try to get value with key a0000026, but not found
ThreadGet try to get value with key a0000026, but not found
```