COP 5536 Spring 2019

Project: B+ Tree

Author: Sagar Prabhu

B+ Tree description

A B+ tree is an m-way search tree which is used to store key-value pairs. A large degree B+ tree is used to store a dictionary which resides on disk, whereas a small degree B+ tree is used to store an in-memory dictionary. In a B+ tree, the key-value pairs are stored only in the leaf nodes. The internal nodes store only the keys and are used to direct the search to the correct leaf node. The number of pairs in the node can be between ceil(m/2)-1 to m-1. Also, the leaf nodes form a doubly linked list.

Program Structure

The following 4 classes are used to implement B+ tree.

1. **Pair**:

   This class defines the data structure of a key-value pair. A list of values is used for handling insertion of duplicate keys.

Field	*Description*
key	Key of the pair
values	List of values associated with the key. Set only for Leaf nodes and empty for internal nodes.

.

2. **TreeNode**:

   This class defines the structure of a B+ Tree node. The fields in this class are:

Field	*Description*
pairs	List of key-value pairs in the node.
children	List of children of an internal node. Empty for a leaf node.
next	Pointer to next TreeNode. Set only for leaf nodes.

prev	Pointer to previous TreeNode. Set only for leaf nodes.
parent	Pointer to parent TreeNode. Set to null for root.

3. **BPlusSearchTree**:

This class implements the key search, range search, insert key, and delete key operations of the B+ tree. The fields defined in this class are:

Field	*Description*
m	The order of the B+ tree.
root	The reference of the root node of the B+ tree.

The prototypes of important functions in this class are as follows:

a. **BPlusSearchTree(int m):** This constructor initializes a B+ tree of order m, creating an empty root node and sets its parent to null.
b. **List<String> search(int key):** This method returns the values associated with the given key. It makes use of a helper function binarySearch to traverse to the correct leaf node. Then it linearly searches the leaf node for the key. If the key is not found it returns null.
c. **List<String> search(int key1, int key2):** This method returns the list of values of key such that key1<=key<=key2. It performs a binary search on the internal nodes to traverse to the leaf using key1. Then it linearly adds the values of keys which are between key1 and key2 and returns it.
d. **int binarySearch(List<Pair> pairs, int target)**: Performs a binary search on the list of pairs to find index i, such that key_{i-1}<=target <key_i .
e. **void insert(int key, String value):** Inserts the key-value pair in the tree. It checks 3 cases. If the root is null, then creates a new TreeNode with the key-value pair. If the root exists and is not full, then its add to root. Otherwise, it traverses to the leaf node using binarySearch helper function and inserts in the leaf. If the leaf overflows then it splits the node using splitLeafNode helper function.
f. **void splitLeafNode(TreeNode node):** This function splits an overfull leaf node by creating a new rightNode, with half of the pairs of node, and middleNode which has the leftmost key of rightNode and propagates the split upwards by calling splitInternalNode.
g. **void splitInternalNode(TreeNode currNode, TreeNode childNode, TreeNode newNode, boolean updatePointers):** Merges the currNode and newNode using the mergeInternal helper function. After merging if currNode becomes overfull then an internal node split operation is

performed and again splitInternalNode is called recursively until the root is reached or there is no internal node overflow.

h. **void delete(int key):** This method deletes the dictionary pair with the given key using helper function deleteUtil. After deletion, if the number of pairs in root becomes zero then it sets root to null.

i. **int deleteUtil(TreeNode parent, TreeNode node,int key):** This method recursively traverses to the left node using method binarySearch and deletes the pair with given key in the node. If there is an underflow, it uses handleLeafUnderflow to merge or redistribute pairs and returns the index of the pair to be deleted in the parent up the stack. If an internal node underflows after deletion, it uses handleInternalUnderflow method to merge or redistribute pairs.

4. bplustree

This class contains the main function of the program which reads the input file line by line, initializes a B+ tree and performs operations on it and writes the result of the search into new file ‘output_file.txt’.