Patrick Fulps

George Ward

3/26/18

CS2300 B+ Tree Project Report

**Abstract:**

…

**Introduction:**

…

**Approach:**

…

**Pseudocode:**

void insert(int input)

{

Node\* node = root (First node)

while(node is not leaf)

node = node->pointer based on comparing input to node->keys

Look for input in node

if(input already in node)

output error

return;

if(node is not full)

Move all values in node after position of input right

Insert input in space created

If(node is full)

splitLeaf(node, input);

print out tree

return

}

void splitLeaf(Node\* node, int input)

{

Create newNode

put right 2 values in newNode, keep left 2 in node

If(parent of node is NULL)

Create new node

First key is last value of node

First 2 pointers are node and newNode

Root is this node

Else

Find position of node in pointers of parent

If parent is full

splitInternalNode(node->parent, input)

else

Move parent pointers after child index to the right

Move parent keys after child index to the right

Insert left value of node in created key space

Insert pointer to newNode in created pointer space

return

}

void splitInternalNode(Node\* node, int input)

{

Create newNode

Put right 2 values and right 3 pointers in newNode

If(parent of node is NULL)

Create new node

First key is last value of node

First 2 pointers are node and newNode

Root is this node

Else

Find position of node in pointers of parent

If parent is full

splitInternalNode(node->parent, input)

else

Move parent pointers after child index to the right

Move parent keys after child index to the right

Insert left value of node in created key space

Insert pointer to newNode in created pointer space

return

}

void remove(int input)

{

Node\* node = root (First node)

while(node is not leaf)

node = node->pointer based on comparing input to node->keys

Look for input in node

if(input not in node)

output error

return;

if(node has more than 1 value)

Remove input from node

Move all values to the right left 1 space to close gap

If(node has 1 value)

If(parent of node is NULL)

Remove last value, to create empty root

Else if(node is rightmost leaf)

leafMerge(left leaf, node, first node key, nodeIndex)

Else

leafMerge(node, next leaf, first node key, nodeIndex)

print out tree

return

}

void mergeLeaf(Node\* left, Node\* right, int input, int indexToNode)

{

Node\* temp = whichever node contains input, left or right

Loop through all siblings to the right

If(sibling has more than 1 value)

Use that sibling

break loop;

Loop through all siblings to the left

If(sibling has more than 1 value)

Use that sibling

break loop;

If(sibling was found to the right)

Loop through # of nodes between temp and sibling

Move first value of sibling to node left of sibling

Update respective parent key to that value

If(sibling was found to the left)

Loop through # of nodes between temp and sibling

Move last value of sibling to node right of sibling

Update respective parent key to new sibling last value

If(no sibling found)

Move values in right node to left node

Remove parent key for right node, shift keys left

Remove parent pointer to right node, shift pointers left

If(parent is empty)

internalMerge(parent)

delete right

return

}

void mergeInternalNode(Node\* node)

{

If(parent is NULL)

root = first pointer of node

delete node

return;

Find indexToNode

Loop through all siblings to the right

If(sibling has more than 1 value)

Use that sibling

break loop;

Loop through all siblings to the left

If(sibling has more than 1 value)

Use that sibling

break loop;

If(sibling was found to the right)

Loop through # of nodes between temp and sibling

Move first value of sibling to node left of sibling

Move first pointer of sibling to node left of sibling

Update respective parent key to that value

If(sibling was found to the left)

Loop through # of nodes between temp and sibling

Move last value of sibling to node right of sibling

Move last pointer of sibling to node right of sibling

Update respective parent key to new sibling last value

If(no sibling found)

Find right or left node to be merged with

Left/right = node, whichever is not already assigned

Move all values and pointers into left node

Remove respective parent key and pointer to right

Shift all other keys and pointers to the left

If(parent is empty)

mergeInternalNode(parent)

delete right

return

}

**Output:**

…

**Full Code:**

…