CS542 Project Two

Xiaoyan Sun & Shi Wang

1. Design

1.1 Assumptions

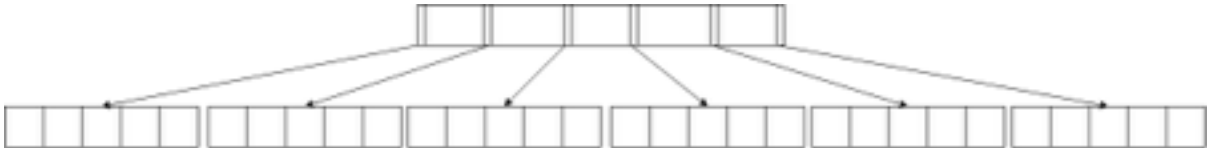
In this project, we use B+ tree as indexing mechanism for the relation table movie. The index shall be built on the attributes *year* and *year|format*. The indexes shall provide a way of find the keys, that is movie title, of the given table. Here are some of our key assumptions.

- ➢ The keys are distinctly stored in the index. For example, all movies from the same year would be stored under the same year index.
- ➢ The maximum number of value can be stored in a node, leaf and inner node, is 5.
- ➢ The construction of the B+ tree is sorting based. When inserting a value into the B+ tree, it first searches for its position, and then adds the value. When maximum number of values stored is reached, the node splits into two.
- ➢ In each node of the tree, values do not have to be stored continuously. Our search method will ignore null values in a node. Our insert method also can find the right spot for the inserted value, and rearrange the existing values in a node if necessary.

2. Algorithm

2.1 B+tree Model

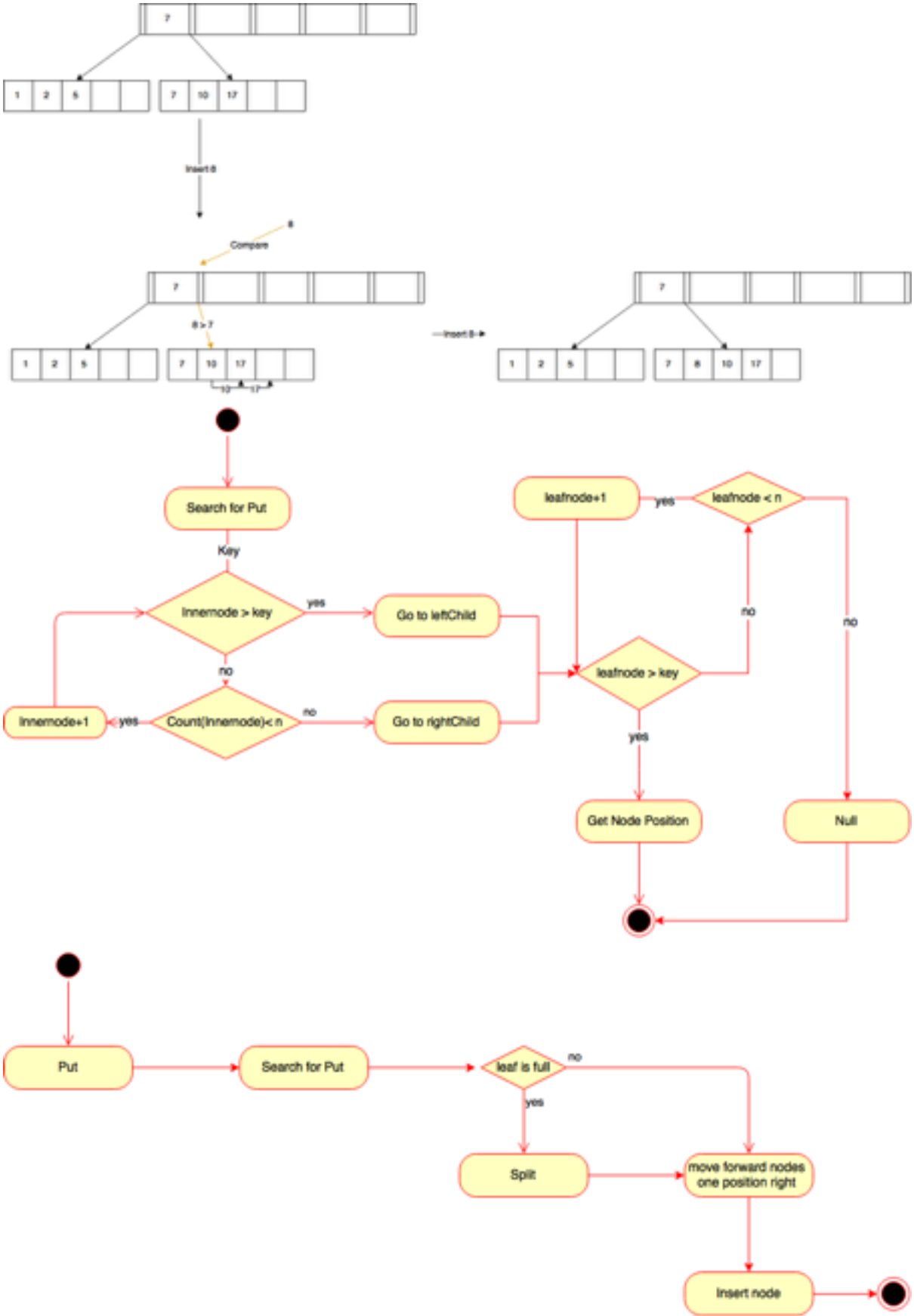
- We built a B+ tree index model. The height of the tree can be unlimited. Fanout of each inner node (including the root) is set to be 6. Construction, searching and removing indexes methods are thoroughly tested.



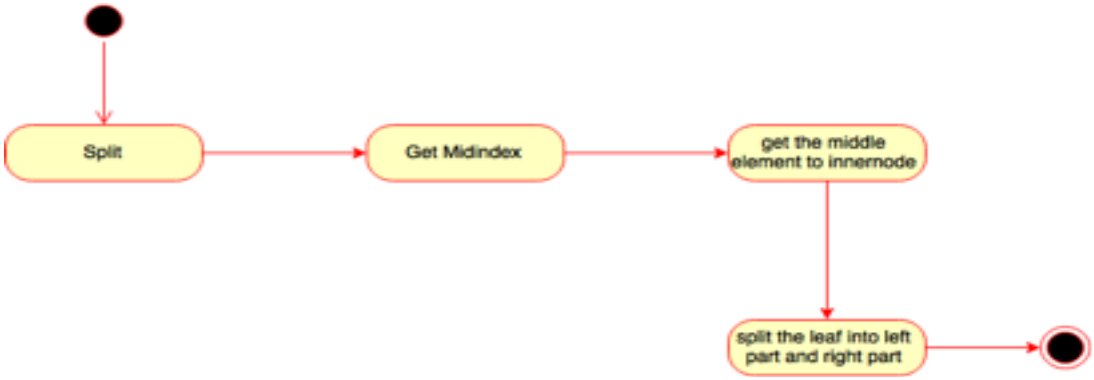
2. B+tree Put

The Put method inserts a new value into the index.

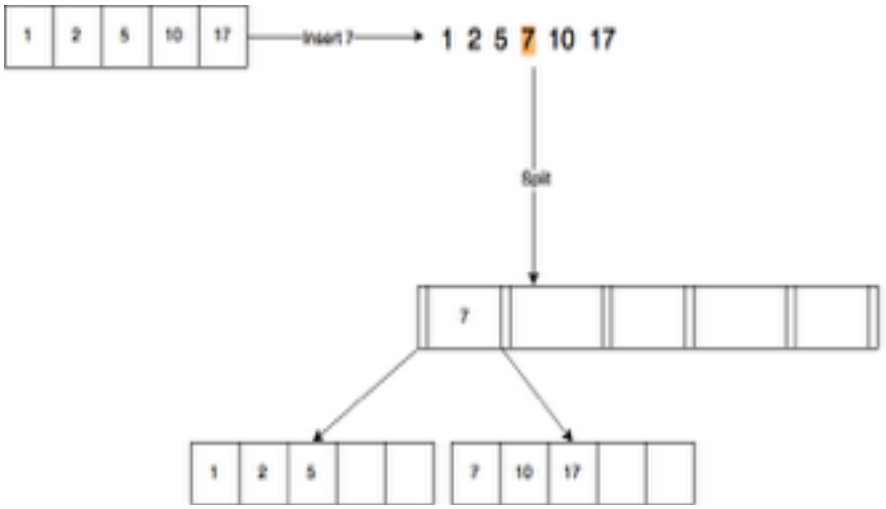
- ➢ Search for the leaf that the value should be inserted in. Our search method returns the leaf and the index of the position where new values should be inserted in.
- ➢ Insert the value in the leaf node. If the spot is empty itself, insert it there. If the spot contains value, rearrange the values in the leaf node and insert the new value. The values shall always be kept in sorted order.
- ➢ After a new value is inserted, check the leaf node to see whether it is overflowed. Of the number of values stored in the node is larger than 5. Call the split function.



2.3 B+tree Node Split



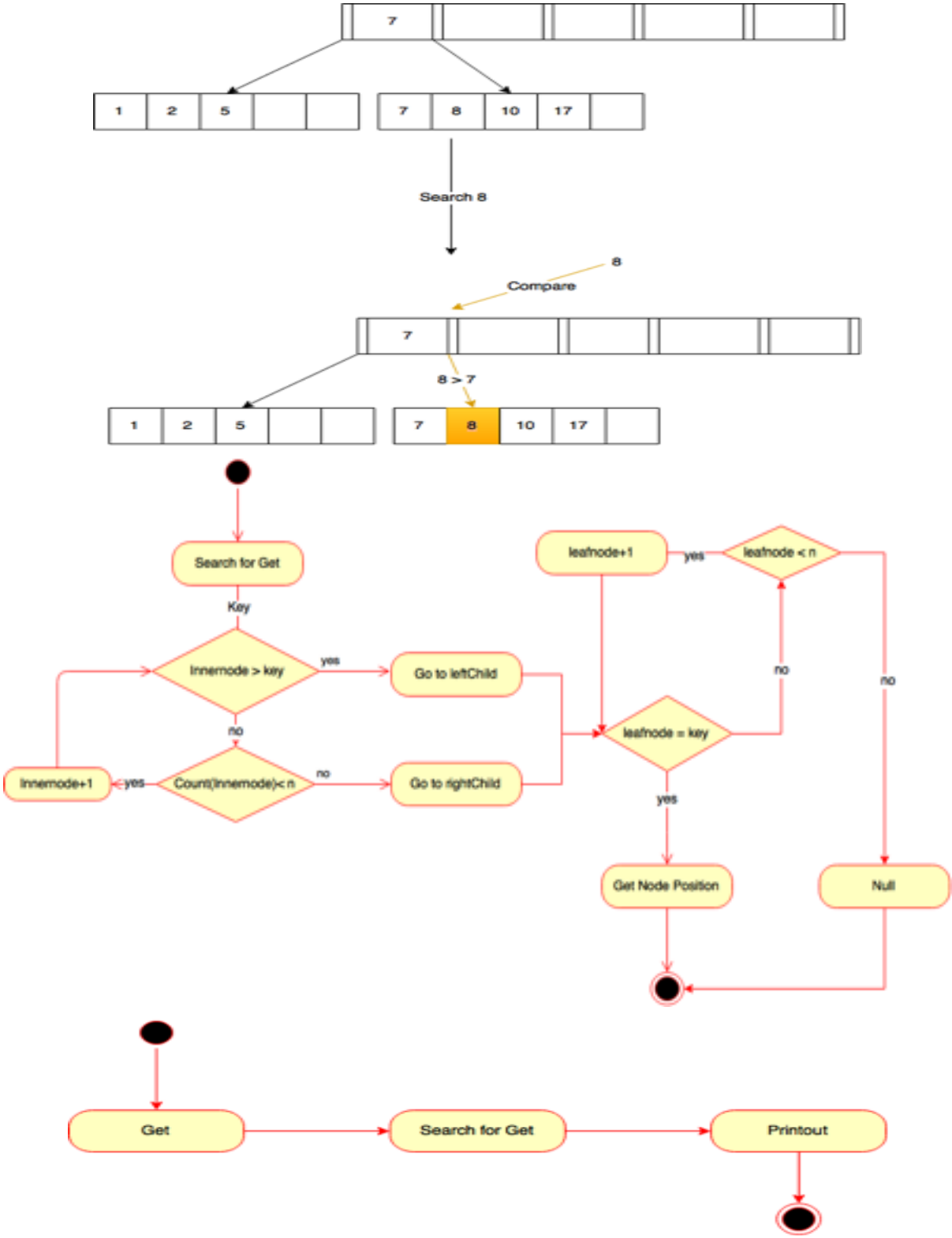
- Split the leaf node. Push up the middle value to upper level node, and keep the left hand half and right hand half as two leaf nodes.
- Check whether the upper level inner node, which has just been added a new value, is overflowed. If so, further split the inner node. The procedure shall continue until the parent of a split node is not overflowed. Recursive functions are used for the purpose of handling multiple level overflows.



2.4 B+tree Get

`string Get(Number data_value);` or `string Get(string data_value);` retrieves the key given the index and

- When we are searching the specific element in the B+tree, we should finnd the position of the element and print out the value.
- First, we compare the element with the innernodes start from root. If bigger than the innernode, go to the right child. If smaller, go to the left child.
- Second, we search the element in the leafnodes. If the element equles the leafnode, then print out the searching value.



2.5 B+tree Remove

`void Remove(string key);` deletes the index.

3. test and result

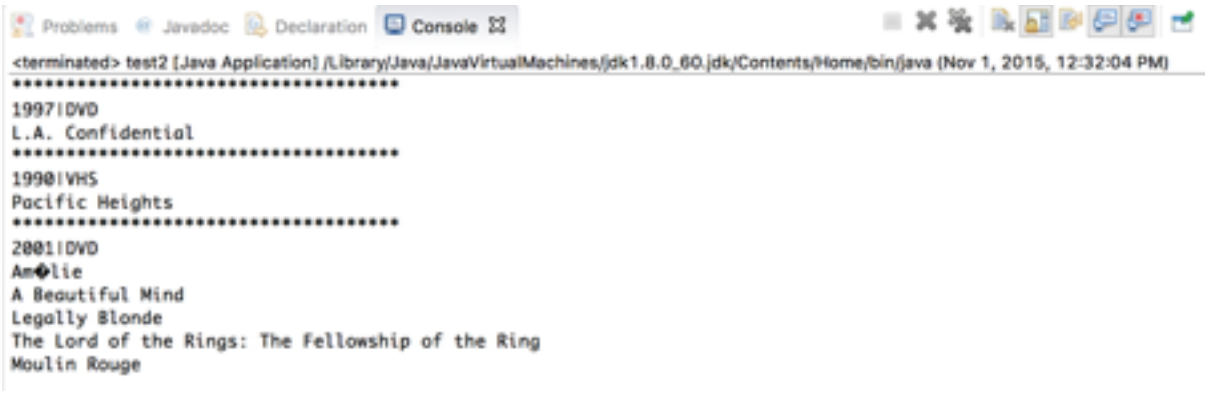
3.1 The index in this case is defined as CREATE INDEX movieInd ON Movie(year, format). The data value could be a composite string. For the movie Ghost for example, it could be "1990|laserdisk". Using the index you created,

3.1.1 Find all DVD movies made in 1977.
3.1.2 Find all VHS movies made in 1990.
3.1.3 Find all DVD movies made in 2001.

(1)method and process

- We insert the key and value by reading the file with split function, after insert all the elements, we search the DVD movies made in 1997, VHS movies made in 1990 and DVD movies made in 2001.

(2)result



3.2 In the second test, assume the same table and index yrInd ON Movie(year).

3.2.1 Find all movies made in 2000.
3.2.2 Find all movies made in 2005.
3.2.3 Find all movies made in 2010.

(1)method and process

- We insert the key and value by reading the file with split function, after insert all the elements, we search the movies made in 2000, 2005 and 2010.

(2)result

4. Some of the codes and key functions

We constructed the leaf and inner node structure, and attached a linked list to each of the leaf pointer to contain values. For example, multiple movies are made in the year 1994. These movies will all be store under the index 1994 in a linked list. Both leafNode and innerNode classes extend the abstract btreeNode class. Some of the print methods are used for unit tests purposes.

Class leafNode

```
//Search for a value in a leaf. Return the index where the value is at
public int search(TKey key)
//insert a key value pair into the leaf. Calls search() and insertAt()
public void insertKey(TKey key, TValue value)
//insert a pair at the index position
private void insertAt(int index, TKey key, TValue value)
//calls split function and furtherhandle functions, returns the highest node that
is //affected by the split.
public btreeNode<TKey> handleOverflow(TKey key, TValue value)
// Param: the value to be pushed up the upper node and the index from which the
current //node shall be split. Return: the parent of the split nodes
public btreeNode<TKey> split(TKey key, int midIndex)
//delete a value in the leaf node at position index.
public int deleteNode(int index, TKey key)
```

Class btree

```
//param: value to be searched in the tree; return: reference to the list where the
value //should be contained
public LinkedList<TValue> search(TKey key)
//calls the search function, and prints out the list
public void searchPrint(TKey key)
//param: value to be searched in the tree; return: the leaf that contains the value
private leafNode<TKey, TValue> findTheLeaf(TKey key)
//insert a key value pair
public void insert(TKey key, TValue value)
//delete a key value pair
public void delete(TKey key)
//for testing purposrs. Prints out the root and it children.
public void printTree()
```

Class btreeNode

```
//Recursive function that handles when multiple levels of inner node need to be
split
public btreeNode<TKey> furtherHandle()
```