HW8. BTree_Part1

부산대학교 정보컴퓨터공학부
2020-55645
신세환

제출일: 2024-06-03

1. 문제정의

예제 소스 코드를 이해한 상태로 BTree의 Remove함수와 BTree상태를 보기 위한 중위 순회 함수를 구현하고 Insert시에 BTree의 동작을 볼 수 있도록 만든다. 그리고 삽입과 삭제 모두 잘 동작하는지 테스트코드를 짜서 동작을 살펴본다. Order는 5로 정한다.

2. 클래스 정의 설명

BTree 클래스와 BTreeNode클래스를 통해 B-Tree를 구현한다. BTreeNode의 경우 SimpleIndex를 상속받아서 구현한다.
BTree 객체에서의 Insert와 Remove에서는 split, merge연산이 필요한지 점검하고 BTreeNode객체 리스트들을 통해 필요한 경우 BTreeNode의 split, merge, updateKey함수를 호출해서 BTree를 관리한다.

3. 클래스의 주요 함수 설명

- BTree의 Remove함수를 findleaf로 삭제할 leafnode를 가져오고 여기서 해당 요소를 지운다.
이 때 half pool을 만족시키지 못하면 인접한 leafnode와 합치는 과정을 level을 올라가면서 수행한다.(Insert함수와 반대로 수행한다.)

1) int BTree<keyType>::Insert (const keyType key, const int recAddr)

함수 부분

```
template <class keyType>
int BTree<keyType>::Remove (const keyType key, const int recAddr)
{
        std::cout << "Attempting to remove key: " << key << std::endl;//
        // find the leaf node containing the key
        BTNode* leafNode = FindLeaf(key);
```

```
	// remove the key from the leaf node
	int result = leafNode->Remove(key, recAddr);
	if(result == 0)
	{
		// 키가 없음
		return 0;
	}

	std::cout << "Removed key from leaf node. Checking for underflow..." << std::endl;//

	// handle underflow
	int level = Height - 1;
	while(result == -1 && level > 0)
	{
		std::cout << "Underflow at level " << level << ". Attempting to resolve..." << std::endl;//
		// fetch parent node
		BTNode* parentNode = Nodes[level - 1];

		// find the index of the leaf node in the parent node
		int nodeIndex = parentNode->Search(leafNode->LargestKey(), -1, 0);
		if(nodeIndex < 0) nodeIndex = 0;

		// try to borrow a key from a sibling node or merge
		BTNode* leftSibling = nullptr;
		BTNode* rightSibling = nullptr;

		if(nodeIndex > 0)
		{
			leftSibling = Fetch(parentNode->RecAddrs[nodeIndex - 1]);
		}
		if(nodeIndex < parentNode->numKeys() - 1)
		{
			rightSibling = Fetch(parentNode->RecAddrs[nodeIndex + 1]);
		}
```

```
                if(leftSibling && leftSibling->numKeys() > leftSibling->MinKeys)
                {
                        std::cout << "Borrowing key from left sibling..." << std::endl;//
                        // borrow from leftSibiling
                        keyType borrowedKey = leftSibling->Keys[leftSibling->numKeys() - 1];
                        int borrowedAddr = leftSibling->RecAddrs[leftSibling->numKeys() - 1];

                        leftSibling->Remove(borrowedKey, borrowedAddr);
                        leafNode->Insert(borrowedKey, borrowedAddr);
                        parentNode->UpdateKey(leftSibling->LargestKey(), leafNode->LargestKey());
                        Store(leftSibling);
                        Store(leafNode);
                        result = 1;
                }
                else if(rightSibling && rightSibling->numKeys() > rightSibling->MinKeys)
                {
                        std::cout << "Borrowing key from right sibling..." << std::endl;//

                        keyType borrowedKey = rightSibling->Keys[0];
                        int borrowedAddr = rightSibling->RecAddrs[0];
                        rightSibling->Remove(borrowedKey, borrowedAddr);
                        leafNode->Insert(borrowedKey, borrowedAddr);
                        //parentNode->UpdateKey(borrowedKey, rightSibling->LargestKey());
                        parentNode->UpdateKey(leafNode->LargestKey(), rightSibling->Keys[0]);
                        Store(rightSibling);
                        Store(leafNode);
                        result = 1;
                }
                else if(leftSibling)
                {
                        std::cout << "Merging with left sibling..." << std::endl;//
                        // merge with leftSibling
                        leftSibling->Merge(leafNode);
```

```
                        parentNode->Remove(leafNode->LargestKey());

                        Store(leftSibling);
                        result = (parentNode->numKeys() < parentNode->MinKeys) ? -
1 : 1;
                }
                else if(rightSibling)
                {
                        std::cout << "Merging with right sibling..." << std::endl;//

                        // merge with rightSibling
                        leafNode->Merge(rightSibling);
                        parentNode->Remove(rightSibling->LargestKey());
                        Store(leafNode);
                        result = (parentNode->numKeys() < parentNode->MinKeys) ? -
1 : 1;
                }

                Store(parentNode);
                leafNode = parentNode;
                --level;
        }

        // handle the root node case
        if(Height > 1 && Root.numKeys() == 1)
        {
                std::cout << "Reducing height of the tree." << std::endl;//

                // 루트에 키가 하나만 있는 경우 -> reduce height
                BTNode* newRoot = Fetch(Root.RecAddrs[0]);
                Root = *newRoot;
                Height--;
        }

        Store(&Root);
        return 1;
}
```

Leafnode로 값을 가져와서 키에 해당하는 값을 삭제한 후 underflow, 즉 half pool을 만족하지 못하는 상황에서 같은 레벨의 왼쪽 노드, 오른쪽 노드에서 값을 가져오거나 그래도 half pool이 발생하는 경우 merge를 수행하고 parent node의 정보를 갱신해주는 과정을 level을 올라가면서 half pool을 만족시킬때까지 수행해준다.

2) void BTree<keyType>::InorderTraversal()

함수 부분

```
template <class keyType>
void BTree<keyType>::InorderTraversal(BTreeNode<keyType>* node)
{
	if(node == nullptr)
	{
		return;
	}
	cout << "IsLeaf : " << node->IsLeaf() << endl;
	if(!node->IsLeaf())
	{
		for(int i = 0; i < node->NumKeys; i++)
		{
			cout << i << " " << node->NumKeys << endl;
			InorderTraversal(Fetch(node->RecAddrs[i]));
			if(i < node->NumKeys)
			{
				cout << node->Keys[i] << " ";
			}
		}
		InorderTraversal(Fetch(node->RecAddrs[node->NumKeys -
1]));
	}

	else
	{

		for(int i = 0; i < node->NumKeys; i++)
		{
			cout << i << endl;
			cout << node->Keys[i] << " ";
		}
	}
}
```

LeafNode가 아닌 경우에 하위 노드를 계속해서 순회하고 해당 노드의 키 값도 출력해준다. 마지막 포인터에 해당하는 경우도 순회해주면서 함수를 종료시킨다.

LeafNode인 경우에는 노드의 키 값들만 출력해준다.

3) void BTree<keyType>::InorderTraversal()

함수 부분

```
template <class keyType>
bool BTreeNode<keyType>::IsLeaf() const
{
        return (this->NumKeys == 0);
}
```

leafNode인지 판단하기 위해 BTreeNode클래스에 IsLeaf()메소드를 추가해주었다.

4) Test code 추가 부분

함수 부분

```
// BTree::remove연산 테스트
        const char* keysToRemove = "DTAMPIBWNGU";
        cout << "₩nRemove test" << endl;
        for (i = 0; i < 11; i++)
        {
                cout << "Removing " << keysToRemove[i] << endl;
                result = bt.Remove(keysToRemove[i]);
                bt.Print(cout);
        }
```

Remove함수를 테스트하기 위해 추가했다.

4. 프로그램 실행

1) 프로그램 실행 환경

- WSL Ubuntu환경

g++ (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

2) 프로그램 실행 방법

- make 수행 후 실행

```
sk3456@DESKTOP-PQL1ORV:~/git_project/hw8_BTree_part1/fs24hw8-sinsehwan$ make
g++ -c -I./include -Wall  -o testbtree.o testbtree.cpp
testbtree.cpp: In function ‘int main(int, char**)’:
testbtree.cpp:26:29: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings]
   26 |         result = bt.Create ("testbt.dat",ios::in|ios::out);
      |                             ^~~~~~~~~~~~
In file included from testbtree.cpp:4:
./btindex/btree.cpp: In instantiation of ‘BTree<keyType>::BTree(int, int, int) [with keyType = char]’:
./btindex/btree.cpp:361:16:   required from here
./include/btree.h:60:28: warning: ‘BTree<char>::BTreeFile’ will be initialized after [-Wreorder]
   60 |         RecordFile<BTNode> BTreeFile;
      |                            ^~~~~~~~~
./include/btree.h:52:16: warning:   ‘BTree<char>::BTNode BTree<char>::Root’ [-Wreorder]
   52 |         BTNode Root;
      |                ^~~~
In file included from testbtree.cpp:5:
./btindex/btree.cpp:8:1: warning:   when initialized here [-Wreorder]
    8 | BTree<keyType>::BTree(int order, int keySize, int unique)
      | ^~~~~~~~~~~~~~
make[1]: Entering directory '/home/sk3456/git_project/hw8_BTree_part1/fs24hw8-sinsehwan/btindex'
gcc -fPIC -c -Wall  -I../include -o simpind.o simpind.cpp
gcc -fPIC -c -Wall  -I../include -o btnode.o btnode.cpp
gcc -fPIC -c -Wall  -I../include -o btree.o btree.cpp
In file included from btree.cpp:3:
btree.cpp: In instantiation of ‘BTree<keyType>::BTree(int, int, int) [with keyType = char]’:
btree.cpp:361:16:   required from here
../include/btree.h:60:28: warning: ‘BTree<char>::BTreeFile’ will be initialized after [-Wreorder]
   60 |         RecordFile<BTNode> BTreeFile;
```

```
sk3456@DESKTOP-PQL1ORV:~/git_project/hw8_BTree_part1/fs24hw8-sinsehwan$ ./testBtree
Insert test
Inserting C
Finding leaf for key: C
BTree of height 1 is
Simple Index max keys 6 num keys 1
        Key[0] C RecAddr 0
end of BTree
Inserting S
Finding leaf for key: S
BTree of height 1 is
Simple Index max keys 6 num keys 2
        Key[0] C RecAddr 0
        Key[1] S RecAddr 1
end of BTree
Inserting D
Finding leaf for key: D
BTree of height 1 is
Simple Index max keys 6 num keys 3
        Key[0] C RecAddr 0
        Key[1] D RecAddr 2
        Key[2] S RecAddr 1
end of BTree
Inserting T
Finding leaf for key: T
BTree of height 1 is
Simple Index max keys 6 num keys 4
        Key[0] C RecAddr 0
        Key[1] D RecAddr 2
        Key[2] S RecAddr 1
```

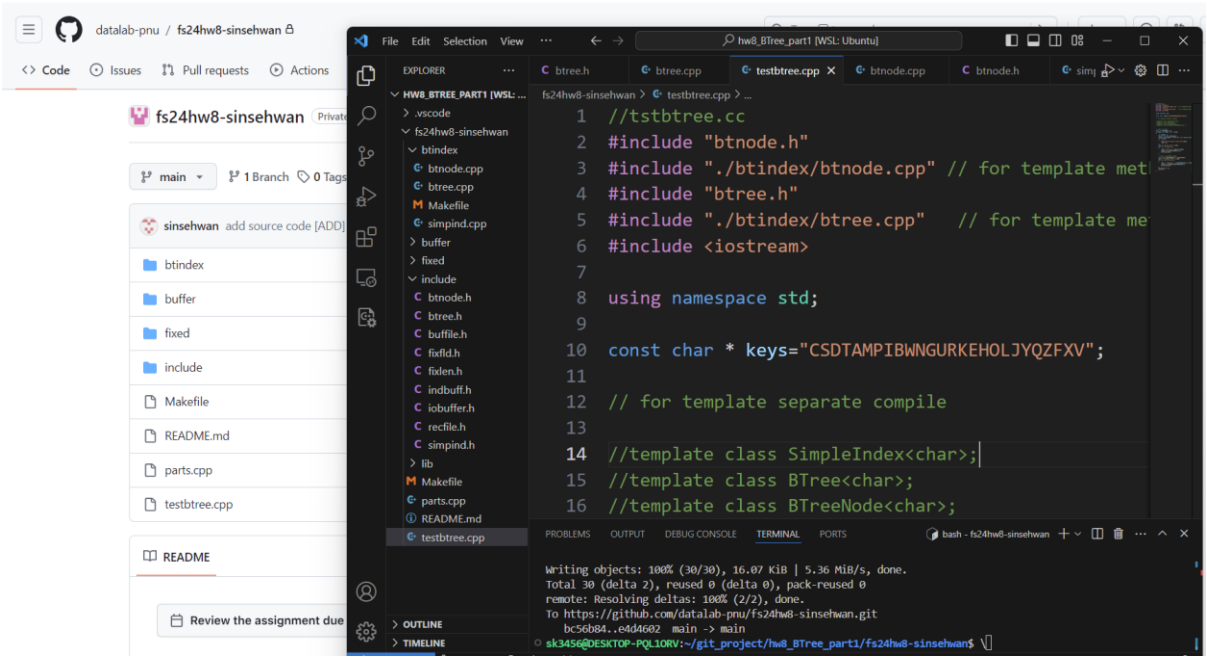
```
end of BTree
Removing G
Attempting to remove key: G
Finding leaf for key: G
0x55fc5fe4b2c0
BTree of height 2 is
Simple Index max keys 6 num keys 2
        Key[0] D RecAddr 298
        Key[1] T RecAddr 182
Node at level 2 address 298 Simple Index max keys 6 num keys 3
        Key[0] A RecAddr 4
        Key[1] C RecAddr 0
        Key[2] D RecAddr 2
Node at level 2 address 182 Simple Index max keys 6 num keys 0
end of BTree
Removing U
Attempting to remove key: U
Finding leaf for key: U
0x55fc5fe4b440
BTree of height 2 is
Simple Index max keys 6 num keys 2
        Key[0] D RecAddr 298
        Key[1] T RecAddr 182
Node at level 2 address 298 Simple Index max keys 6 num keys 3
        Key[0] A RecAddr 4
        Key[1] C RecAddr 0
        Key[2] D RecAddr 2
Node at level 2 address 182 Simple Index max keys 6 num keys 0
end of BTree
Finding leaf for key: S
0x55fc5fe4b5c0
sk3456@DESKTOP-PQL1ORV:~/git_project/hw8_BTree_part1/fs24hw8-sinsehwan$
```

5. Github 화면

(1) cloning, adding, committing, push을 위한 github 명령들을 포함

```
sk3456@DESKTOP-PQL1ORV:~/git_project/hw8_BTree_part1/fs24hw8-sinsehwan$ git add .
sk3456@DESKTOP-PQL1ORV:~/git_project/hw8_BTree_part1/fs24hw8-sinsehwan$ git commit -m "add source code [ADD]"
[main e4d4602] add source code [ADD]
 24 files changed, 1769 insertions(+)
 create mode 100644 Makefile
 create mode 100644 btindex/Makefile
 create mode 100644 btindex/btnode.cpp
 create mode 100644 btindex/btree.cpp
 create mode 100644 btindex/simpind.cpp
 create mode 100644 buffer/Makefile
 create mode 100644 buffer/buffile.cpp
 create mode 100644 buffer/iobuffer.cpp
 create mode 100644 buffer/recfile.cpp
 create mode 100644 fixed/Makefile
 create mode 100644 fixed/fixfld.cpp
 create mode 100644 fixed/fixlen.cpp
 create mode 100644 fixed/indbuff.cpp
 create mode 100644 include/btnode.h
 create mode 100644 include/btree.h
 create mode 100644 include/buffile.h
 create mode 100644 include/fixfld.h
 create mode 100644 include/fixlen.h
 create mode 100644 include/indbuff.h
 create mode 100644 include/iobuffer.h
 create mode 100644 include/recfile.h
 create mode 100644 include/simpind.h
 create mode 100644 parts.cpp
 create mode 100644 testbtree.cpp
sk3456@DESKTOP-PQL1ORV:~/git_project/hw8_BTree_part1/fs24hw8-sinsehwan$ git push
\Enumerating objects: 31, done.
Counting objects: 100% (31/31), done.
Delta compression using up to 16 threads
Compressing objects: 100% (30/30), done.
Writing objects: 100% (30/30), 16.07 KiB | 5.36 MiB/s, done.
Total 30 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/datalab-pnu/fs24hw8-sinsehwan.git
   bc56b84..e4d4602  main -> main
sk3456@DESKTOP-PQL1ORV:~/git_project/hw8_BTree_part1/fs24hw8-sinsehwan$ \
```

(2) 소스 코드와 makefile을 push한 후, 본인의 Github repository를 스크린 캡쳐하여 포함



6. 논의 사항

- Buffer를 String으로 구현한 코드를 최대한 재사용해서 구성하고자 했는데 템플릿 타입인 typeKey를 String으로 변환해서 Pack, Unpack구현하는 것이 매우 까다로웠고 억지로 변환시켜서 사용하는 경우 템플릿을 활용한 이점을 전혀 살리지 못해서 void*형식으로 인자를 받는 예제 방식을 활용해서 코드를 구성할 수밖에 없었다. Modern C++을 활용해서 Warning이 뜨지 않게 BTree를 깔끔히 설계하고 싶다.