<B Plus Tree 구현 과제>

2018007965 김산

[구현 방식 요약]

BPlusTree 객체는 indexFile을 가지고 생성되고 indexFile의 데이터를 블럭(기본 1KB) 단위로 관리한다. 파일의 첫 블럭 사이즈는 512byte 로 해당 영역에 트리의 헤더(MetaData)를 기록한다. 이를 위해, 메타 데이터 클래스를 정의했다. 모든 Node는 indexFile 상의 filePosition을 가진다. 이에 따라 모든 Node에 대한 참조는 Node의 filePosition이 된다. indexFile 내에서 filePosition 위치에 Node 를 기록하거나, 해당 위치에서 노드를 읽어들이기 위해 RandomAccessFile을 이용해 임의로 접근한다. MetaData나 Node 단위의 읽기/쓰기를 위해 RandomAccessFile을 상속받은 TreeRandomAccessFile을 정의하고, BPlusTree의 indexFile로 사용한다. 노드를 매번 파일로부터 읽고 쓰는 과정에서 속도 지연을 보완하기 위해 CacheMap 을 사용하였다.

[클래스 구성]

1. BPlusTree.java:

이 자바 클래스는 BPlusTree의 root를 알려주는 Node 타입의 root를 가지고 있고, degree를 가지고 있다. 또 TreeRandomAccessFile을 상속하는 indexFile과, blockSize, 그리고 노드의 정보를 기록할 BPlusTreeMetaData를 상속하는 metaData, 그리고 BPlusTree의 개수를 나타는 size를 가지고 있다.

createTree(File indexFile, int degree)

넘겨받은 indexFile에 넘겨받은 degree를 가지는 새로운 BPlusTree 객체를 생성한다.

getInstance(File file)

기존의 indexFile로부터 BPlusTree 객체를 복원할 때 사용한다. 해당 메소드가 호출되면 파일에 기록된 BPlusTreeMetaData를 읽어서 BPlusTree 객체를 복원하는 생성자가 호출된다.

close()

BPlusTree 종료시 갱신된 메타 데이터를 indexFile에 기록한다.

searchSingleKey(long key)

메인 메소드에서 Single Key Search를 하기 위해 필요한 메소드이다. recursiveSearchAtKey를 활용해서 LeafNode의 특정 key까지 도달할 때까지 지나가는 모든 노드의 값을 출력한 후, 최종적으로 LeafNode의 특정 key에 해당하는 value값을 출력한다.

recursiveSearchAtKey(long key, Node node, boolean print)

메인 메소드에서 Single Key Search를 할 때, root에서 LeafNode까지 도달하기 위해 재귀적으로 호출하는 메소드이다. boolean을 인자로 받는 이유는 Range Search를 하기 위한 메소드 내에서도 Range Search의 시작점을 찾기 위해 해당 LeafNode까지 가야하지만 그 경로 상의 모든 노드의 출력은 필요하지 않으므로 경우에 따라 출력 여부를 정하기 위함이다. indexFile의 readNode로 가져오는 Node가 LeafNode일 때 까지 해당 메소드를 계속 호출한다.

searchRange(long start, long end)

메인 메소드에서 Ranged Search를 하기 위해 필요한 메소드이다. 메소드 내에서 Entry<Integer> 타입의 List를 만들고 RangeSearch의 시작 노드와 끝 노드가 될 LeafNode 두 개를 가져온 후 searchLeafRange함수를 호출한 후, 만들었던 List를 return 값으로 받는다.

searchLeafRange(long start, long end, LeafNode startLeaf, LeafNode endLeaf, List<Entry<Integer>> list)

Ranged Search를 하기 위해 필요한 searchRange 메소드의 실제 출력에 필요한 정보를 담는 메소드이다. 인자로 받은 startLeaf에서 인자로 받은 start에 해당하는 key를 찾고, 인자로 받은 endLeaf에서 인자로 받은 end에 해당하는 key를 찾아서 시작점과 끝점을 설정한다. 그리고 시작점부터 (key, value) 쌍을 인자로 받은 List에 add해주면서 startLeaf의 끝에 도달했을 때, startLeaf를 현재 자신의 우측 노드로 변경해주면서 endLeaf의 end에 도달할 때 까지 List에 add를 반복한다.

insert(long key, int value)

splitEntry라는 Entry를 만든다. 현재 BPlusTree의 root가 NonLeafNode일 경우 LeafNode까지 내려가서 실제 insert를 진행할 수 있도록 recursiveInsert를 호출하고, root가 LeafNode일 경우, 바로 insert를 하고 return값을 splitEntry에 할당한다. 이 메소드에서 splitEntry에 값이 생기는 경우는 현재 생성된 BPlusTree의 최상위 노드인 root 노드에서 overflow가 일어나서 split할 정보가 생기는 경우이다. 따라서 그 경우에는 root 노드를 새롭게 만들어주고 splitEntry를 그 노드에 할당해준다.

recursiveInsert(Entry entry, NonLeafNode nonLeaf)

insert할 LeafNode를 찾을 때 까지 재귀적인 호출을 통해 LeafNode까지 도달하는 메소드이다. LeafNode에 도달했을 경우, 인자로 받은 entry를 LeafNode에 insert한다. LeafNode에서 overflow가 일어나서 split된 정보가 있을 때 splitEntry 변수에 그 정보를 담고 인자로 받은 nonLeaf에 splitEntry를 insert한 것을 return값으로 받는다.

delete(long key)

delete할 key를 넘겨받았을 때, 메소드 내에서 해당 key가 있을 것이라고 추정되는 LeafNode까지 root에서부터 BPlusTree를 타고 내려갈 때, 모든 경로를 nodeOnPath에 기록함과 동시에 Node 타입의 Stack에 push하면서 기록한다. LeafNode까지 도달했을 경우, LeafNode부터 다시 Stack의 경로를 거꾸로 올라가며 해당 노드에 key가 있을 경우 Node의 delete 메소드를 호출하여 delete를 진행한다. Node의 delete 메소드가 delete할 key와, 자신의 부모 노드를 인자로 받으므로, BPlusTree의 delete 메소드에서 Stack을 타고 root까지 도달했을 경우, 부모 노드에 null을 주고 Node의 delete 메소드를 호출한다.

2. BPlusTreeMetaData.java:

노드에 대한 부가정보(차수, indexFile 의 블럭 사이즈, 루트 노드의 위치, 트리의 데이터 개수)를 가진 객체이다. BPlusTree 객체 생성, 또는 indexFile 로부터 복원할 때 이 메타데이터 객체가

만들어지고, close 메소드를 통해 이 객체가 종료될 경우 이 객체를 파일에 기록한다.

3. TreeRandomAccessFile.java:

파일을 대상으로 filePointer 기반의 랜덤 액세스를 지원하는 RandomAccessFile 을 상속한다.
indexFile 을 대상으로 블럭 단위의 읽기(readNode)/쓰기(writeNode)를 지원한다.
모든 노드는 생성할 때 filePosition이 결정되어, indexFile 상의 하나의 블럭을 할당받고, 각 노드를 대상으로 insert/delete 시 해당 filePosition에 노드 객체를 다시 기록한다.

readTreeMetaData()

BPlusTree의 indexFile에 기록된 메타데이터를 읽어온다.

writeTreeMetaData(BPlusTreeMetaData metaData)

파일의 맨 처음 위치에 512바이트의 블럭을 메타데이터 영역으로 삼고, 메타데이터를 해당 블록에 기록한다.

readNode(long filePosition, int blockSize)

BPlusTree의 특정 filePosition의 노드를 읽어들이는 메소드이다.

writeNode(Node<?> node)

BPlusTree의 노드가 가진 filePosition 위치에 노드를 기록한다. 해당 위치에 노드가 이미 있을 경우, 새롭게 갱신한다.

removeNode(long filePosition)

파일에 있는 노드 정보를 삭제한다.

4. Entry.java:

BPlusTree의 각 노드의 m, p, r 중, p에 해당하는 (key, value), (key, node)에 해당하는 데이터를 담기 위해 만들어진 객체이다. long 타입의 key와 Generic 타입의 partner를 변수로 가지며 LeafNode와 NonLeafNode에서 각각 다른 타입을 가진다.

5. Node.java:

BPlusTree의 각 노드가 가지고 있어야 할 m, p, r과 degree를 가지고 있다. LeafNode와 NonLeafNode의 r에 각각 right sibling 또는 rightmost child에 해당하는 Node의 파일의 주소에 해당하는 filePosition을 long 타입으로 가지고 있으므로, TreeRandomAccessFile의 readNode를 통해 실제 노드를 가져와야 하므로 blockSize, filePosition, indexFile을 가지고 있다.

insert(Entry<T> entry)

LeafNode 또는 NonLeafNode에서 실제 insert를 한 후 return되는 값을 받아와서 다시 return 시키고 그 과정에서 변동사항을 indexFile에 기록한다.

delete(long key, NonLeafNode parent)
realDelete를 호출한 후 그 과정에서 변동사항을 indexFile에 기록한다.

6. LeafNode.java:

BPlusTree 구조의 가장 하위에 해당하는 노드로, 실제 insert 또는 delete가 해당 위치에서부터 시작하게 되며 insert또는 delete과정 수행 후 상위에 영향을 주는 경우가 생길 수도 있다.

searchAtKey(long key)
인자로 받은 key의 실제 위치를 return값으로 받는 메소드이다.

reallnsert(Entry<Integer> entry)
인자로 받은 entry를 해당 노드에 실제로 insert하는 메소드이다. 이 과정에서 overflow가 발생하면 현재 노드를 split하고 새롭게 LeafNode를 만들어서 필요한 정보를 옮겨주며, split된 key에 해당하는 entry의 정보를 담고 있는 entry를 새롭게 만들어 return 값으로 넘겨준다.

underFlowPossible(int nodeM)
BPlusTree의 각 노드에서 delete를 할 경우 underflow가 일어나면 BPlusTree의 구조가 변경되어야하므로, underflow 여부를 체크하는 메소드이다.

realDelete(long key, NonLeafNode parent)
인자로 받은 부모 노드를 이용하여 현재 노드의 왼쪽과 오른쪽 형제 노드들이 있을 경우, 그 노드를 변수를 선언해서 가져와서 실제 delete를 할 때 활용하여 delete를 하는 메소드이다. 현재 노드, 왼쪽 형제 노드, 오른쪽 형제 노드, 이 3가지 노드의 underFlow 여부를 확인하여 각각의 상황에 맞는 delete 관련 메소드를 호출한다.

keyBorrow(long key, NonLeafNode parent, LeafNode leftSib, int Case)
현재 노드가 delete를 통해 underflow이며, 왼쪽 또는 오른쪽 형제 노드가 해당 key를 현재 노드로 빌려줘도 underflow가 일어나지 않을 경우, 호출하는 메소드이다. 왼쪽에서 빌려오는 경우와 오른쪽에서 빌려오는 경우를 나누기 위하여 Case를 인자로 넘겨받았다. 왼쪽 노드에서 키가 하나 빠져도 underflow가 안 일어나면 왼쪽 노드의 마지막 index의 key, value를 가져와 현재 노드의 맨 앞에 추가하여 underflow를 막는다. 오른쪽 형제 노드도 같은 경우로 적용된다.

keyMerge(long key, NonLeafNode parent, LeafNode leftSib, int Case)
현재 노드가 delete를 통해 underflow이며, 왼쪽과 오른쪽 형제 노드가 모두 해당 key를 현재 노드로 빌려주면 underflow가 일어날 경우, 형제 노드와 해당 노드를 합병하기 위해 호출하는 메소드이다. 왼쪽 또는 오른쪽과 합병하는 경우를 나누기 위하여 Case를 인자로 넘겨받았다. 왼쪽 노드와 오른쪽 노드 모두 키가 하나 빠지면 underflow가 일어날 경우 현재 노드와 형제 노드를 병합하여 새로운 노드를 만들고 연결된 다른 노드들의 정보를 현재 트리 상태에 맞게 바꿔준다.

7. NonLeafNode.java:

LeafNode와는 다르게 추가적으로 newNonLeafNodePosition을 가지며, NonLeafNode에서 overflow가 일어날 때 split되어 생기게 될 newNonLeaf에 필요한 데이터를 용이하게 넣기 위해 추가한 변수이다. NonLeafNode에서는 하위 노드에서 insert/delete시에 상위 노드에서 구조적 변동이 필요한 경우 그 구조적 변동을 하게 된다.

searchAtKey(long key)

인자로 받은 key의 실제 위치 또는 인자의 key보다 큰 값 중 가장 적은 값을 가지고 있는 위치의 key의 왼쪽 child 노드의 파일 위치를 return값으로 받는 메소드이다. 해당 노드의 모든 값이 넘겨받은 key값보다 작은 경우 return 값은 해당 노드의 r 노드의 위치가 된다.

realInsert(Entry<Long> entry)

인자로 받은 entry를 해당 노드에 실제로 insert하는 메소드이다. 이 과정에서 overflow가 발생하면 현재 노드를 split하고 새롭게 newNonLeafNode를 만들어서 필요한 정보를 옮겨주며, split된 key에 해당하는 entry의 정보를 담고 있는 entry를 새롭게 만들어 return 값으로 넘겨준다.

printKeys()

BPlusTree의 Ranged Search에 필요한 해당 노드의 모든 key값을 출력하는 메소드이다.

underFlowPossible(int nodeM)

BPlusTree의 각 노드에서 delete를 할 경우 underflow가 일어나면 BPlusTree의 구조가 변경되어야하므로, underflow 여부를 체크하는 메소드이다.

realDelete(long key, NonLeafNode parent)

인자로 받은 부모 노드를 이용하여 현재 노드의 왼쪽과 오른쪽 동일 height의 노드들이 있을 경우, 그 노드를 변수를 선언해서 가져와서 실제 delete를 할 때 활용하여 delete를 하는 메소드이다. 현재 노드, 왼쪽 노드, 오른쪽 노드, 이 3가지 노드의 underFlow 여부를 확인하여 각각의 상황에 맞는 delete 관련 메소드를 호출한다.

internalKeyBorrow(long key, NonLeafNode parent, int Case)

인자로 받은 부모 노드의 왼쪽과 오른쪽에 동일 height의 노드들이 있을 경우, 그 노드를 변수로 선언해서 가져와 실제 delete에서 underflow가 발생했을 때 양 옆에서 key를 빌려오는 메소드이다. 왼쪽 인접 노드에서 키 하나를 빼도 underflow가 일어나지 않을 경우 왼쪽 인접 노드의 키를 현재 노드의 parent로 올리고 현재 노드의 parent의 key 하나를 현재 노드로 가져와서 현재 노드의 underflow를 막는다. 같은 경우가 오른쪽에도 적용된다.

internalMerge(long key, NonLeafNode parent, int Case)

인자로 받은 부모 노드의 왼쪽과 오른쪽에 동일 height의 노드들이 있을 경우, 그 노드를 변수로 선언해서 가져와 실제 delete에서 underflow가 발생했을 때 양 옆의 노드와 merge를 수행하는 메소드이다. 양 옆에서 key가 하나라도 빠지면 underflow가 일어날 경우 인접 형제 노드와 parent의 key 하나 그리고 현재 노드를 합쳐주고 연결된 정보들을 갱신해서 현재 노드의

underflow를 막는다.

8. LRUCacheMap.java:

indexFile 에 기록된 노드 데이터를 매번 필요할 때마다 파일로 부터 읽어들이게 되면, 파일 I/O 과정에서 속도 저하가 발생한다. 한번 파일로부터 읽어들인 노드를 캐시 데이터로 저장하여 사용하면, 속도 향상의 효과가 있다. 각 노드를 파일로 부터 읽어들일 때, 캐시로 저장하고, 이 때 저장형태는 [filePosition, Node] 와 같으며, 각 노드를 기록할 때, 캐시된 노드의 상태 역시 갱신한다. Cache 데이터의 크기는 maxSize 로 제한하며, 이로 인해, 캐시 데이터가 꽉 차는 경우, 삭제할 데이터를 결정해야 하는데, LRU(Least Recently Used) 알고리즘에 따라, 최근까지 사용한 적인 없는 캐시 노드를 먼저 삭제하기 위해 구현된 Cache Map이다.

9. ClosableLinkedBlockingQueue.java:

한꺼번에 대량의 데이터를 파일로부터 삽입하는 경우, 데이터 파일로부터 데이터를 읽는 속도와, indexFile에 데이터를 기록하는 속도에 차이가 발생한다. 다 읽고 나서 기록하는 직렬형태의 작업에서는 그 속도차로 인해 전체 성능이 저하된다. 따라서 동시에 읽고, 쓰는 작업을 진행하되, 어느 한쪽이 빨리 진행되더라도, 누락되는 데이터가 없도록, Producer-Consumer 패턴을 사용하여, Producer(데이터 파일 읽기)와 Consumer(indexFile 에 기록)가 동시에 사용할 수 있는 자료구조 객체가 필요하다. LinkedBlockingQueue 는 push/take 로 데이터를 넣고 꺼내는 과정에서 넣으려는데, size 가 full이거나, 빼려는데, size==0이면 유효데이터에 접근할 때까지 대기 상태가 된다.

10. TreeTest.java:

실제 BPlusTree의 Create, Insert, Single Key Search, Ranged Search, Delete를 수행하기 위한 main 메소드가 있는 클래스이다. private 변수로 start와 end를 가지며 실제 특정 동작을 수행하는 데에 걸린 시간을 나타내기 위해 추가했다. command로 "-c"를 입력받을 때는 BPlusTree의 createTree를 통해 BPlusTree 객체를 생성하고, 나머지 command를 입력받을 때는 getInstance로 indexFile의 BPlusTree 객체 정보를 가져와서 실제 동작에 활용한다.

insertFromCSVUseQueue(BPlusTree tree, File file)

Thread 객체에 인자로 넘겨받는 target에 해당하는 run 메소드가 실행되므로 "-i" 과정 수행 시 insert할 정보를 넘겨받은 csv파일을 insert할 데이터로 바꿔서 실제 BPlusTree에 insert 과정을 수행하는 메소드이다.

deleteFromCSVUseQueue(BPlusTree tree, File file)

Thread 객체에 인자로 넘겨받는 target에 해당하는 run 메소드가 실행되므로 "-d" 과정 수행 시 delete할 정보를 넘겨받은 csv파일을 delete할 데이터로 바꿔서 실제 BPlusTree에 delete 과정을 수행하는 메소드이다.

class ReadFromCSV -> run()

넘겨받은 csv파일을 String 배열로 한 줄 씩 읽을 때, 배열의 길이가 2이면 insert를 위한 파일로 간주해서 0번째와 1번째 index를 Integer로 파싱한다. 그 외의 경우는 배열의 길이가 1인 delete를 위한 파일이므로 0번째 index만 Integer로 파싱한다. 그 후 Queue에 기록을 한 후 Queue를 close한다.

class WriteToTree -> run()

Queue에 저장되어 있던 데이터를 한 쌍씩 int 배열로 가져와서 실제 BPlusTree에 insert를 한다. Queue가 비어있을 때 까지 위의 과정을 반복한다. 그리고 모든 과정이 끝나고 insert한 key의 개수와 걸린 시간을 출력한다.

class DeleteFromTree -> run()

Queue에 저장되어 있던 데이터를 하나씩 int 배열로 가져와서 실제 BPlusTree에서 delete를 한다. Queue가 비어있을 때 까지 위의 과정을 반복한다. 그리고 모든 과정이 끝나고 delete한 key의 개수와 걸린 시간을 출력한다.

[구현 방법]

1) BPlus/basic 폴더에서 javac basic/TreeTest.java -encoding UTF-8을 통해 컴파일을 한다.
2) 이후 Command-Line에서 과제의 설명서에서 주어진대로 java TreeTest -c index.dat 5와 같은 예시처럼 파일을 생성한다.
3) 이후
java basic/TreeTest -i index.dat input.csv
java basic/TreeTest -s index.dat 1
java basic/TreeTest -r index.dat 1 100
java basic/TreeTest -d index.dat delete.csv
등 다양한 Command-Line 입력을 줄 수 있다.

*CSV 파일은 BPlus/src 파일에 위치해 있어야 한다.