B+Tree 구현 프로젝트 보고서

Computer software

2020001658 이유민

1. Summary of my Algorithm

- Class

(1) bpNode 클래스:

index-value pair의 개수를 의미하는 m, 그 pair의 목록을 의미하는 p, 오른쪽 자식 또는 형제를 가리킬 rightChild, 부모노드를 가리킬 Parent를 가지고 있다.

내부 클래스로 index 클래스를 포함한다.

(2) index 클래스:

index, value, leftChild를 가지고 있다.

(3) bpTree 클래스:

트리를 구성하는 클래스이자 main클래스이다.

- 전체적인 알고리즘

➔ Data -> Tree(index.dat파일구성)

```
degree: 8
| 0 1
68 97321 /
| 1 2
9 87632 / 10 84382 / 20 57455 / 26 1290832 / 37 2132 /
| 1 3
68 97321 / 84 431142 / 86 67945 / 87 984796 /
* 2 3
```

Degree를 입력 받아 creation을 통해 생성된 index.dat파일에서 한 줄씩 읽어온다.

- degree: 뒤에 오는 숫자는 degree에 저장
- | 뒤에 오는 숫자는 순서대로 부모 노드 번호, 내 노드 번호로 저장
- 그 다음줄에는 내 노드 안에 존재하는 index-value pair를 /로 구분하여 저장

- * 뒤에 오는 숫자는 leaf 노드이므로, leaf리스트에 순서대로 저장. (sibling끼리 rightChild로 연결)

➔ Key Insert

1. Input.csv파일을 한 줄 한 줄 inputReadLine에 대입한다.
2. inputReadLine을 띄어쓰기로 구분하여 index와 value를 IndexValuePair배열에 각각 저장한다.
3. 순서대로 하나하나 index.dat파일에 삽입한다.
4. findCorrectNode를 통해 해당 index가 들어갈 리프노드를 찾는다.
5. findCorrectindex를 통해 해당 노드 안에서 index가 들어갈 자리를 찾는다.
6. 그 자리에 일단 삽입하고 해당 노드의 m을 1 늘려준다.
7. 그 때 m이 degree보다 크거나 같아지면 overflow이기에 split해준다.
8. split되는 노드가 root인 경우에는 새로운 left노드와 right노드를 만들어 쪼갠 내용을 넣어주고, root를 새롭게 재구성한다.
9. split되는 노드가 root가 아닌 경우에는 새로운 right노드만 만들어 내용을 옮겨주고 현재노드는 뒷부분을 날려준다. 그리고, right노드의 처음 index값을 연결된 부모노드로 올려준다.
10. 9번에서 부모노드로 값 올렸는데 overflow난 경우 부모노드를 split해준다.

➔ Key Search

1. checkNode는 root부터 시작해서 내려온다.
2. checkNode가 leaf라면 종료한다.
3. 아니라면 대소비교를 통해 index가 포함된(or 범위가 시작되는) leaf노드까지 내려간다.
4. 그곳에서 해당 index(해당 범위의 인덱스들)를 찾는다.
5. 없다면 'NOT FOUND'를 출력한다.

2. Detailed description of my codes. (for each function)

(1) Creation

- degree를 입력 받아 index.dat파일에 "degree : 8"의 형태로 출력한다.

(2) read

- 저장된 index.dat파일을 읽어오는 함수이다.
- 해당 파일을 buffer를 이용해 한 줄씩 indexLine에 넣어두고, 띄어쓰기를 기준으로 정보를 나눠 content배열에 넣어둔다.
- Index.dat파일 구성에서 "degree: ", "|", "*"로 내용을 구분하였기에, content배열의 첫 번째 값인 content[0]을 type변수로 지정하고 내용을 구분한다.
- Type == "degree: "인 경우 뒤따라오는 수를 전역변수인 degree에 저장한다.
- Type == "|"인 경우 content[1]을 parentNum, content[2]를 mineNum에 저장한다. 그리고 다음줄에 mineNum 노드에 들어있는 index-value pair를 읽어와서 index클래스 객체를 만들어 tempNode에 대입한다. (tempNode의 m 1올려주고)
- 생성한 tempNode를 Tree 리스트 mineNum 인덱스에 대입한다.
- tempNode의 Parent를 연결해줄 때는 parentNum과 mineNum을 이용한다. 여기서 parentNum이 0인 경우에는 해당 mineNum노드가 root이다.
- 반복
- 마지막에 type == "*"이 되는 경우, 해당 인덱스의 노드들을 leaf 리스트에 추가하고, leafNode끼리 rightChild로 연결한다.

(3) Insertion

- Read()를 통해 index.dat파일 안의 내용을 읽어온다.
- Input.csv파일을 읽어와 한줄씩 inputReadLine에 넣는다.
- inputReadLine을 ','로 구분해 indexValuePair배열에 저장하여 추가할 index, value값을 하나씩 가져온다.
- Insert(index, value)를 호출한다.

(4) Insert

- findCorrectNode()를 통해 값이 들어갈 리프노드를 찾는다.
- findCorrectIndex()를 통해 리프노드에서 값이 들어갈 index를 찾는다.
- 입력받은 index와 value값을 알맞은 자리에 삽입하고, 삽입한 노드의 m을 1 올려준다.
- 삽입했는데, 해당 correctNode의 m이 degree보다 작으면 종료, 크거나 같으면 splitNode를 호출하여 해당 노드를 split한다.
- splitNode()에서 노드를 두 개로 쪼개야 하니까 오른쪽부분을 담을 rightNewNode를 만든다. 그리고 rightNewNode 속을 채워준다.
- MidPoint를 정하고(degree/2), Midepint~degree까지의 p를 rightNewNode에 옮기고 기존노드에서 삭제한다.
- 기존노드의 rightChild를 rightNewNode의 rightChild로 넣어준다.
- Split한 노드가 root가 아닌 경우에는 putinParent()를 통해 기존노드의 부모노드에 MidPoint에 해당하는 MidIndex와 MidValue값을 알맞을 자리에 넣어주고, 그때 부모노드에서 m 이 degree를 넘어가는 순간 splitParent()로 부모노드를 split해준다. 넘어가지 않으면 종료한다. 하지만 만약에 split한 노드가 root인 경우에는 새로운 left노드를 만들어 쪼갠 앞 내용을 넣어주고, root를 새롭게 재구성하여 MidPoint에 해당하는 MidIndex와 MidValue값을 넣어준다.
- 업데이트된 내용을 save()를 통해 index.dat파일에 올려준다.

(5) Deletion

- 입력받은 index에 해당하는 값들을 삭제하고, Tree를 재구성한다.

(6) searchSingleKey

- root를 checkNode에 넣어 root부터 아래로 내려간다.
- 해당 index가 들어있는 리프노드를 찾아야 하기에 리프노드가 아닌 경우에만 while문에 들어가고 리프노드인 경우 해당 노드에서 index를 찾아 반환한다.
- 리프노드가 아닌 경우, 대소비교(크면 rightChild, 작으면 leftChild)를 통해 알맞은 리프노드로 내려가고, 내려가는 동안 만나는 노드들의 index-value pair값들을 모두 출력한다.

- 찾지 못 한다면, "NOT FOUND"를 출력한다.

(7) rangedSearch

- startKey와 endKey를 입력받고, read()함수로 Tree를 읽어온다.
- startKey보다 크거나 같아지는 순간의 "리프노드"를 찾는다.
- 노드를 찾으면 startKey인덱스부터 endKey보다 작거나 같은 index까지의 내용을 모두 출력한다.

(8) save

- Tree에 저장된 노드들, leaf들을 index.dat파일에 불러와 저장하는 함수이다.
- Degree를 먼저 출력하고, 순서대로 노드들을 출력하는데, 노드 출력은 save_in()함수가 담당한다.
- root부터 순서대로 출력할 것이기 때문에 save_in()함수를 처음 호출할 땐, root노드와 '0'을 넘겨준다.
- save_in()함수는 입력받은 노드의 parentNum과 MineNum을 출력하고, 다음줄에서 노드안의 값들을 /로 구분하여 모두 출력한다.
- 만약 입력받은 노드의 가장 첫번째 인덱스의 leftChild가 null이 아니라면 해당 자식노드로 내려가서 save_in()함수를 호출한다.
- 이 때 노드가 리프노드가 아니라면 자식노드로 이동해 save_in()함수를 호출한다.
- 노드가 리프라면 index값을 leaf 리스트에 add한다.
- 마지막에 * 표시와 함께 leaf 리스트에 있는 번호들을 순서대로 출력한다.

(9) is_leaf

- 입력받은 노드가 leaf인지 non-leaf인지 확인하는 함수이다.
- 만약 아예 값이 없거나(노드의 m == 0) 첫번째 index-value pair의 leftChild가 없는 상태라면 이는 leaf노드임을 의미하므로 true를 return한다.
- 아닌 경우엔 false return.

(10) find_leaf

- 입력받은 index가 들어갈 리프노드를 찾는 함수이다.
- While문은 리프노드가 아닐 경우에만 들어가며, 들어가서 돌다가 리프노드로 이동하게 되면 while문을 빠져나와 이를 return하게 된다.
- while문 안에서는 index와의 대소비교를 통해 작다면 leftChild로 크다면 rightChild로 이동하며 leaf노드에 도달하게 한다.

3. Instructions for compiling my source codes

(1) 해당프로젝트 폴더로 가서 CMD창 실행

(2) "bpTree.java"파일이 있는 주소로 이동하여 아래 명령어를 실행합니다.

```
C:₩Users₩JEONG₩bptree_>cd src

C:₩Users₩JEONG₩bptree_₩src>dir/w
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: F64C-EBA7

 C:₩Users₩JEONG₩bptree_₩src 디렉터리

[.]             [..]            bpNode.java    bpTree.java
               2개 파일                  24,111 바이트
               2개 디렉터리  181,332,545,536 바이트 남음
```

(3) javac bpTree.java -encoding UTF8 명령어 실행.

```
C:₩Users₩JEONG₩bptree_₩src>javac bpTree.java -encoding UTF8
```

(4) create, insert, single search, ranged search 명령어는 아래와 같습니다.

```
C:₩Users₩JEONG₩bptree_₩src>java bpTree -c index.dat 8
```

```
C:₩Users₩JEONG₩bptree_₩src>java bpTree -i index.dat input.csv

C:₩Users₩JEONG₩bptree_₩src>java bpTree -s index.dat 86
68,
67945

C:₩Users₩JEONG₩bptree_₩src>java bpTree -s index.dat 10
68,
84382

C:₩Users₩JEONG₩bptree_₩src>java bpTree -r index.dat 20 85
20, 57455
26, 1290832
37, 2132
68, 97321
84, 431142

C:₩Users₩JEONG₩bptree_₩src>
```

4. Other specification

(1) Java class에 있는 한글주석으로 인해 오류가 생기는 점을 확인했습니다.

"javac bpTree.java" 명령어가 아닌 "javac bpTree.java -encoding UTF8"로 실행해야 합니다.

(2) 컴파일을 통해 class파일이 생성되는 그 폴더에 input.csv 파일이 함께 들어있어야 합니다.

(3) 모든 명령어 실행은 (2)의 위치에서 진행되어야 합니다.