CSE3207 Project #2

002분반

12130919 강 혁

1. 구현 사항

- Index creation
- Insertion
- Point (exact) search
- Range search
- Print B+-Tree Structure (Level 0, Level 1)

모두 구현 했습니다.

2. 구현 설명

Leaf node에 Data record가 저장되고, Non leaf node에 index block들을 저장할 수 있게 만들었습니다.

Key와 value, 그리고 block을 가리키는 BID (ptr) 모두 4byte int type으로 입력을 받아 Leaf node와 Non leaf node의 entry 개수가 같아 하나의 클래스 node에 멤버로 int vector를 이용해서 entry를 구성했습니다.

그 후 각 leaf node와 non leaf node를 in memory에 로딩한 후 search시 구분할 때에는 header의 depth를 이용해 체크 후 entry를 입력하는 과정에서는 key가 저장되는 entry list index가 홀수인지 짝수인지를 체크해 구분했습니다. Split도 leaf node와 non leaf node 두 개에 대한 각각의 메소드로 나누어 구현했습니다.

In memory에 각 블록을 loading했을 때 헤더의 정보나 leaf node 도달시 parent node에 대한 정보를 다시 external memory에 접근해 찾는 것이 아니라 BPTree class에 멤버로 node vector를 nodeBuffer로 정의해 사용했습니다.

새로운 BID를 할당해 새 node를 file에 write하는 것이 아닌 update operation의 경우에는 중간단계에 update하는 것이 아니라 commitBuffer에 저장해 두었다가 모든 과정을 수행 후 마지막에 commitBuffer의 node들을 pop하며 write해주었습니다. 이 연산은 따로 함수로 분리하지 않고 insert 메소드 내에서 if문으로 분기했습니다.

print의 경우 BPTree의 멤버함수로 포함시키지 않고 root node를 in memory로 로딩하고 바로 fstream을 이용해 output file에 출력합니다.

문제에서 주어진 내용에 따라 레벨0과 레벨1만 출력할 수 있도록 선형적으로 출력합니다. 또한 레벨 1이 leaf node일경우 <key, value, key, value...> 형태인데, 문제에 따라 key를 출력하도록 했습니다.

가장 마지막의 comma를 지우기 위해 " "를 넣었습니다.

그 외 input text file을 memory로 불러올 때에는 getline함수를 이용하고 strtok을 이용해 ,와 ₩n 단위로 파싱해 값들을 이용합니다.

3. How to compile and run

- Compile : g++ -o [exe file name] bptree.cpp
- Index creation : [exe file name] c [index file name] [block size(int)]
- Insertion : [exe file name] i [index file name] [input text file name]
- Point search : [exe file name] s [index file name] [input text file name] [output text file name]
- Range search : [exe file name] r [index file name] [input text file name] [output text file name]
- Print : [exe file name] p [index file name] [output text file name]
-

4. Project를 수행하며 경험한 것

Search 연산을 재귀적으로 수행하려다 보니 첫 search시 root block이 초기화 되어 있지 않으면 에러가 계속해서 발생해 그 부분은 첫 insertion call이 발생했을 때 직접 예외처리를 통해 빈 root block(entry가 모두 0인)을 초기화 해 입력하고나서 잡을 수 있었습니다. 또 insert 중 random access가 가능한 구조이다 보니 record가 위치를 잡아 insert가 되었을 때 split이 발생하면 보통은 한쪽 방향으로 트리가 커지지만 초기 루트의 가장 왼쪽 포인터는 따로 방향을 잡아 예외처리를 해주어야 했기 때문에 따로 insertion direction 변수를 두어 예외처리를 했습니다. 그리고 node가 split되어 위쪽으로 엔트리가 올라갈 경우 split propagtion이 발생할 여지가 있기에 새로운 노드가 할당되고 그 노드에 밑의 엔트리가 들어갈지 아니면 기존에 존재하던 노드에 단순히 엔트리만 추가될지에 따라 수행할 연산이 조금씩 달랐기에 그 부분에 대해서도 예외처리를 해주었습니다.

또한 제가 구현에 사용한 node buffer는 단순히 split propagation 처리를 위해 parent node를 tracking하고 pop해 버리는 용도로만 사용했지만 추후 update연산을 추가하게 된다면 대기 버퍼에 따로 저장해 두어 external memory I/O를 줄이는데 이용할 수 있을 것 같습니다.

그 외에도 여러 메소드에서 file stream을 다루다 보니 프로젝트 초반에는 이미 open 되어있는 file stream을 또 열고 그 과정에서 여러 스레드가 하나의 file stream에 접근을 해 data가 깨지거나 업데이트가 제대로 수행되지 않는 등의 경험을 했습니다. 그래서 하나의 메소드 내에서도 file stream의 open과 close를 atomic한 operation 단위로 묶어 수행하려 했습니다.

5. Contact information

핸드폰 : 010 9690 9410

Email : k941026h@naver.com