데이터베이스시스템및응용 B+ Tree 구현 프로젝트

2018062733 윤동빈

1. Summary of your algorithm

Balanced Tree의 특징을 갖고 있어 삽입, 삭제에서 각각 overflow, underflow가 날 때마다 양쪽 sibling의 key pair 개수에 따라 redistribute, merge가 일어나면서 balance를 맞춥니다.

트리에 설정된 degree를 b라 하면, 가질 수 있는 최대 자식 수가 b가 됩니다. 따라서 루트 노드를 제외한 인덱스 노드에서 가질 수 있는 key의 개수는 최소 $\lceil b \rceil - 1$개, 최대 $b-1$개입니다. 루트 노드는 예외로 최소 1개에서 최대 $b-1$개까지 가집니다.

인덱스 노드와 리프 노드는 같은 Node 클래스에서 필드 isLeaf의 true/false에 따라 구분되도록 했으며 key pair는 각각 p, p_에 저장하고 취급하는 key pair 클래스는 Pair, Pair_로 구분했습니다. 설명대로 Pair는 <key, left_child_node>로, Pair_는 <key, valuePointer>로 구성했습니다.

제가 구현한 **insertion**에서 따르는 규칙은 다음과 같습니다.
(1) 삽입 후 overflow가 발생하지 않으면 종료합니다.
(2) 삽입 후 overflow가 발생하면 다음 조건을 따라 수행됩니다.
   1. left sibling이 존재할 때
   left sibling의 key pair 개수가 $b-1$미만이면 현재 노드에서 가장 작은 key pair를 left sibling으로 넘깁니다.
   2. right sibling이 존재할 때
   right sibling의 ker pair 개수가 $b-1$미만이면 현재 노드에서 가장 작은 key pair를 right sibling으로 넘깁니다.
   3. split 후 중간 key pair를 부모 노드에 삽입합니다.

제가 구현한 **Deletion**에서 따르는 규칙은 다음과 같습니다.
(1) 삭제 후 underflow가 발생하지 않으면 종료합니다.
(2) 삭제 후 underflow가 발생하면 다음 조건을 따라 수행됩니다.
   1. left sibling의 key pair 개수가 $\lceil b \rceil - 1$보다 많으면 하나를 빌려옵니다.
   2. right sibling의 key pair 개수가 $\lceil b \rceil - 1$보다 많으면 하나를 빌려옵니다.
   3. (left sibling이 존재한다면) left sibling으로 merge합니다.
   4. right sibling이 존재한다면) right sibling으로 merge합니다.

Single Key/Ranged Search 여부는 search()에서 받는 인자에 따라 처리되도록 구현했습니다.

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

<Tree 클래스>

- void search(boolean mode, long key, long key2, Node cur)
  - mode == true : Single Key Search 수행
    mode == false : Ranged Search 수행
  - key : Single Key Search에서 검색 대상
    Ranged Search에서 검색 대상의 최솟값
  - key2 : Ranged Search에서 검색 대상의 최댓값
  - cur : 현재 노드
  - 인덱스 노드의 key들을 모두 출력
    리프 노드의 <key, value> 출력

- SplitChildren insert(Node parent, Node cur, Pair_ pair, int childIndex, int depth)
  - parent : 부모 노드
  - node : 현재 노드
  - pair : 삽입할 <key, valuePointer>
  - childIndex : 현재 노드의 자식 번호 (0부터 시작)
  - depth : 트리에서 현재 노드의 depth
  - SplitChildren 리턴 : overflow 시 분리된 자식 노드와 중간 key값을 반환
    main에서 리턴된다면 중간 key값을 가진 새로운 루트 노드 생성

- boolean delete(Node cur, long target, int depth)
  - cur : 현재 노드
  - target : 삭제할 key 값
  - depth : 트리에서 현재 노드의 depth
  - true 리턴 : 자식노드가 삭제 됨(borrow, merge 필요)
    false 리턴 : 자식노드가 삭제되지 않았음

- void writeIndexFile(BufferedWriter bw)
  index_file 작성하는 함수

<Node 클래스>

- void insertPair() : Pair 삽입
  void insertPair_() : Pair_ 삽입
  void getPair(int index) : index번째 Pair 반환
  void getPair_(int index) : index번째 Pair_ 반환
  void setChild(int index, Node Child) : index번째 left_child_node를 Child로 설정
  void setParent(Node parent) : parent 설정
  void setChildIndex(int childIndex) : childIndex 설정

int size() : ArrayList<Pair> p의 크기
int size_() : ArrayList<Pair_> p_의 크기

- SplitChildren split(int dividedIndex) : 현재 노드의 중간값과 분리된 두 노드 반환

- Node searchLeftSibling(boolean turn, Node cur, int curDepth, int goalDepth)
: left sibling을 찾는 함수. root 노드를 향해 올라가기 때문에 찾는 과정에서 공통 부모 노드를 거친 여부를 turn으로 설정
- turn : 공통 부모 노드를 거쳤는지 여부
- cur : 현재 노드
- curDepth : 현재 노드의 depth
- goalDepth : 목표 노드의 depth
Node searchRightSibling(boolean turn, Node cur, int curDepth, int goalDepth)
: right sibling을 찾는 함수

- boolean leftSiblingHasSpace(boolean isLeaf, Node leftSibling)
: left sibling에 한 key pair를 넣을 공간이 있는지 체크
boolean rightSiblingHasSpace(boolean isLeaf, Node rightSibling)
: right sibling에 한 key pair를 넣을 공간이 있는지 체크

- void giveToLeft(boolean isLeaf, boolean isInsert, boolean isEnd, Node leftSibling)
: 현재 노드의 한 key pair를 left sibling에 삽입
- isLeaf : 현재 노드가 리프 노드인지 여부
- isInsert : insertion에서 호출됐는지 여부
- isEnd : (deletion 용) 삭제된 자식이 마지막 자식인지 여부
- leftSibling : 왼쪽 sibling
void giveToRight(boolean isLeaf, boolean isInsert, boolean isEnd, Node rightSibling)
: 현재 노드의 한 key pair를 right sibling에 삽입

- long updateKey(int index, Node cur, boolean start)
: 현재 노드의 index번째 key 값을 업데이트
- index : 현재 노드의 key pair 순번
- cur : 현재 노드
- start : (구현용) updateKey() 호출 시작 여부

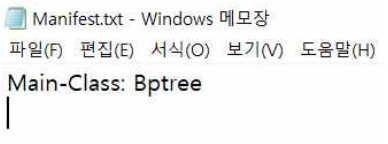
- boolean leftHasEnough(Node leftSibling)
: left sibling의 key pair 개수가 최소 개수보다 많은지 여부 (borrow 가능한지)
boolean rightHasEnough(Node rightSibling)
: right sibling의 key pair 개수가 최소 개수보다 많은지 여부 (borrow 가능한지)

- void leftMerge(Node left, int except)
  : 현재 노드에서 except번째 자식을 제외하고 모든 자식을 left에 merge
  - left : merge할 대상
  - except : 현재 노드에서 제외할 자식 순번
  void rightMerge(Node right, int except)
  : 현재 노드에서 except번째 자식을 제외하고 모든 자식을 right에 merge

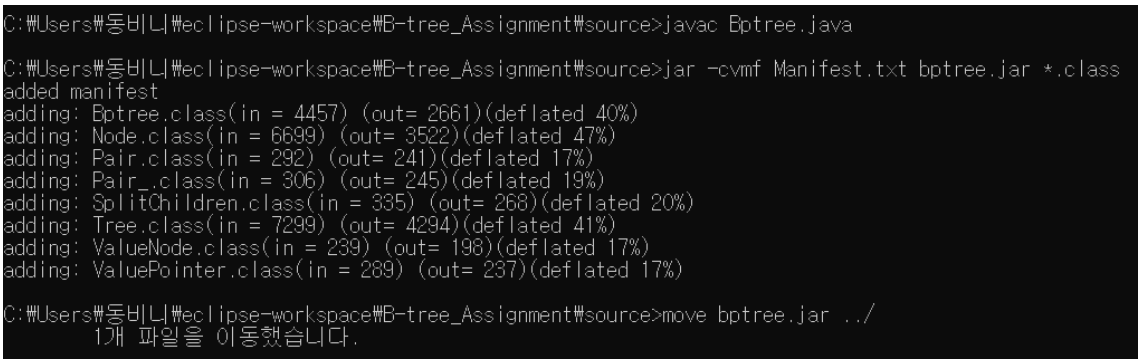
- void connectFromDeletedNode() : 삭제되는 단말 노드에서 양쪽 sibling을 연결

3. Instructions for compiling your source codes at TA's computer (e.g. screenshot)

Manifest.txt 작성 및 저장 (B-tree_Assignment/src)



컴파일 및 jar 파일 생성, src 부모 디렉토리(B-tree_Assignment)로 이동



4. Any other specification of your implementation and testing

index.dat은 다음과 같이 구성됩니다.

b {degree 수}
/ {key1} {key2} {key3} ... / {key1} {key2} ... : '/'는 인덱스 노드임을 의미
...
{key1} {value1} {key2} {value2} ... # {key1} {value1} ... : '#' 리프 노드임을 의미
...

각 line은 **각 depth에 속한 노드들의 정보**를 의미합니다.
key와 value는 서로 공백 한 칸(' ')으로 구분됩니다.

<개인 테스트>

- Creation (b=3)

```
C:₩Users₩동비니₩eclipse-workspace₩B-tree_Assignment>java -jar bptree.jar -c index.dat 3
```

index.dat - Windows 메모장

파일(F) 편집(E) 서식(O) 보기

```
b 3
```

- Insertion

input.csv

input.csv - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
26,1290832
10,84382
87,984796
86,67945
20,57455
9,87632
68,97321
84,431142
37,2132
105,4582
1,996542
754,41395
9625,7542
8888,12
521258,42158
```

insertion 실행 및 결과

```
C:₩Users₩동비니₩eclipse-workspace₩B-tree_Assignment>java -jar bptree.jar -i index.dat input.csv
```

index.dat - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
b 3
/ 68 105
/ 10 26 / 86 / 8888 9625
# 1 996542 9 87632 # 10 84382 20 57455 # 26 1290832 37 2132 # 68 97321 84 431142 # 86 67945 87 984796 # 105 4582 754 41395 # 8888 12 # 9625 7542 521258 42158
```

- Single Key Search (key=84)

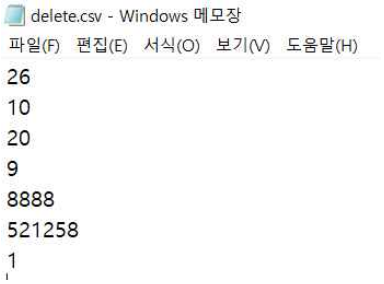
```
C:₩Users₩동비니₩eclipse-workspace₩B-tree_Assignment>java -jar bptree.jar -s index.dat 84
68,105
86
431142
```

- Ranged Search (start_key=50, end_key=1000)

```
C:₩Users₩동비니₩eclipse-workspace₩B-tree_Assignment>java -jar bptree.jar -r index.dat 50 1000
68,97321
84,431142
86,67945
87,984796
105,4582
754,41395
```

- Deletion

delete.csv



deletion 실행 및 결과