B+ tree 구현 보고서

컴퓨터소프트웨어학부 2022045723 김세연

1. Summary of algorithm

Main.java

-c, -i, -d, -s, -r 각 케이스에 대한 함수 생성

-creation: sieOfNode를 받아 BPlusTree 생성 후 saveTree()로 indexFile에 tree 저장
-insertion: loadTree()로 BPlusTree 불러오기+ input.csv 한 줄 씩 읽으면서 key, value 가져와 tree에 insert()에 전달. Insert 끝났으면 saveTree함수로 indexFile에 저장
-deletion: loadTree()로 BPlusTree 불러오기 + delete.csv 한 줄 씩 읽으면서 key값 가져와 delete()에 전달
-singleKeySearch:loadTree()로 BPlusTree 불러오기 + findSinglekey()가 false이면 “NOT FOUND” 출력
-rangedSearch: loadTree()로 BPlusTree 불러오기 + startKey와 endKey를 findRangedKeys()에 전달하여 그 값이 false이면 “NOT FOUND” 출력

-creation 제외한 함수에서 bptree= new BPTree(0) 인 이유는 loadTree에서 sizeOfNode를 읽어오기 때문에 처음 노드 사이즈는 0으로 해도 무방
-arg의 length가 0이거나 명령어가 -c, -i, -d, -s, -r이 아닌 경우는 각각 에러 메시지 출력하도록 했음

2. Detailed description of codes

BPTNode.java

class BPTNode: BPlusTree의 노드. leafNode와 interNode구분 없이 사용

<멤버변수>

List<Integer>keys: 노드 내부에 저장하는 key들 리스트

List<BPTNode>children: internalNode일 때 자식들 저장하는 리스트

boolean isLeafNode: leafNode인지 아닌지 표시. leafNode이면 true, 아니면 false

Map<Integer, Integer> keyAndValue; leafNode일때 key-value쌍 저장

BPTNode right; leafNode일 때 포인터로 오른쪽 노드 연결

BPTNode parent: root가 아닐 때 부모 저장

<BPTNode생성자>

isleafNode를 전달받아 저장

List와 Map은 new 객체 생성, right와 parent는 null로 초기화

BPTree.java

class BPTree: BPlusTree

<멤버 변수>

BPTNode root: 트리의 루트노드

Int sizeOfNOde: 트리의 노드의 크기

BPTNode previousLeaf: loadTree할 때 리프노드들 잇기 위한 변수

Int minKey: 최소 키 개수

<BPTree생성자>

sizeOfNode를 전달받아 저장

root는 leafNode이므로 BPTNode에 true를 전달

minKey는 ceil ((노드크기 + 1 /2)) -1

public BPTNode findLeafNode(**int** key): key가 존재하는 leafNode를 찾는 함수

루트에서 리프까지 타고 내려갈 BPTNode형 변수 p 선언

While문 조건은 leafNode가 아닐 때

내부 노드에서 탐색할 index 0으로 초기화

노드의 keys의 크기 보다 작고 찾고자 하는 key가 해당 노드의 index번째 key보다 크거나 같은 동안 index 1씩 증가

찾은 leafNode 반환

<insertion에 필요한 함수들>

public void insert(**int** key, **int** value)

findLeafNode() 호출하여 key를 insert할 leafNode 탐색

leafNode의 keys와 keyAndValue에 각각 key, key-value add

keys 오름차순으로 정렬

leafNode의 key 개수가 sizeOfNode보다 크면 오버플로우이므로 이를 처리해주는 splitAndReorganize() 호출

public void splitAndReorganizeLeafNode(BPTNode leafNode): leafNode를 split하고 재구성하는 함수

새로운 리프노드(newNode) 생성

기존 노드의 중간 지점을 구하기 위해 keys의 size를 2로 나눈 몫 x로 저장

기존 노드의 중간지점 부터의 key, key-value를 newNode에 추가

기존 노드에서는 key, key-value삭제

newNode의 parent는 기존 노드의 parent, right은 기존 노드의 right, 기존노드의 right은 newNode로 설정하여 리프 노드끼리 연결되도록 함

1. split해야되는 노드가 root였을 때는 root를 새로 만들어 newNode의 첫번째 키를 넣어주고, leafNode와 newNode를 children에 추가, leafNode와 newNode의 parent는 root로 설정
2. split해야되는 노드가 root가 아닐 때는 부모 노드에 업데이트

public void updateParent(BPTNode node, **int** key, BPTNode newNode)

parent=node.parent

node가 루트일 때는 위의 1번과 같이 동작하고 return

parent에 update해야 할 인덱스(i) while문으로 탐색

parent의 keys에 i번째 위치에 key 추가, children의 i+1번째 위치에 newNode 추가, newNode의 parent는 parent

그때 parent가 오버플로우 나면 splitAndReorganizeInternalNode 호출하여 처리

public void splitAndReorganizeInternalNode(BPTNode node): Internal node를 split하고 재구성하는 함수

splitAndReorganizeLeafNode와 비슷하나 기존 노드의 중간 지점 이후부터만 key와 children 새로운 노드로 이동하면 된다 (i=x+1부터)

똑같이 기존 노드의 key와 children은 삭제

기존노드와 새로운 노드의 children 존재할 때 기존 노드의 가장 오른쪽 자식(leftChild)과 새로운 노드의 가작 왼쪽 자식(rightChild)이 모두 leafNode라면 leftChild의 right에 rightChild 연결

newNode의 parent는 node의 parent

1. node가 root였을 때 새로운 root생성
2. node가 root가 아니었을 때 updateParent 호출하여 parent에 새로운 key 넣어줌

<deletion에 필요한 함수들>

public void delete(**int** key)

삭제할 key가 있는 leafNode findLeafNode()로 탐색

몇번째 index인지 확인해서 index가 0이라면 노드의 맨 왼쪽 키라는 것이므로 boolean 형 변수에 따로 저장(이 경우는 따로 처리해줘야하기 때문)

leafNode에서 key와 key-value 삭제

minKey 재설정

1. leafNode가 root였을 때 마지막 남은 key도 삭제했다면 트리도 삭제된 것

2. leafNode가 root가 아니었을 때
    1. underflow+맨왼쪽키일 때
       deleteAndReorganize 호출
       + 가장 오른쪽 리프노드가 아니면 goUpandUpdate호출
    2. underflow+맨왼쪽키 아닐 때
       deleteAndReorganize호출
    3. underflow는 아니고 맨왼쪽키일 때
    - goUpandUpdate호출

public void deleteAndReorganize(BPTNode node)

1. 삭제한 노드가 parent의 첫번째 노드였을 때
   오른쪽 노드에서 빌려올 수 있으면 borrowFromRight
   빌려올 수 없으면 merge
2. 삭제한 노드가 parent의 마지막 노드였을 때
   왼쪽 노드에서 빌려올 수 있으면 borrowFromLeft
   빌려올 수 없으면 merge
3. 삭제한 노드가 parent의 첫번째나 마지막 노드가 아니였을 때
   왼쪽/오른쪽에서 빌려올 수 없으면 merge
   왼쪽에서만 빌려올 수 없으면 borrowFromRight
   오른쪽에서만 빌려올 수 없으면 borrowFromLeft

public void borrowFromRight(BPTNode parent, BPTNode node, BPTNode rightNode, **int** index) : 노드의 오른쪽 노드에서 key를 빌려오는 함수

1. leafNode일 때
   오른쪽 노드에서 key, key-value쌍 가져오고 node에 추가
   Parent의 기존 index번째 key는 지우고 빌려온 key 추가
   오른쪽 노드에서는 삭제
2. internalNode일때
   오른쪽 노드의 첫번째 자식의 parent를 node로 바꿈
   기존 노드에 parent의 Index번째 key 추가, 자식 추가
   Parent의 index번째 key는 오른쪽 노드의 맨 왼쪽 키로 대체
   오른쪽 노드에서는 삭제

public void borrowFromLeft(BPTNode parent, BPTNode node, BPTNode leftNode, **int** index): 노드의 왼쪽 노드에서 key를 빌려오는 함수, borrowFromRight과 symmetry함

public void goUpandUpdate(BPTNode leafNode, **int** key, **int** nextIndex): leafNode key삭제 시 internal node의 key도 삭제하는 함수

삭제한 key를 internal 에서 찾았으면 빠르게 반복문 탈출하기 위한 flag 설정

1. leafNode에 남아있는 key가 없다면 그 leafNode의 오른쪽 노드의 맨 왼쪽 key를 가져온다
2. leafNode에 남아있는 key가 있다면 그 leafNode의 삭제한 key 다음 key를 가져온다

while문으로 root까지 타고 올라가면서 삭제한 key 찾아서 internal node에서도 삭제

public void merge(BPTNode leftNode, BPTNode rightNode, BPTNode parent, **int** index):
어느쪽에서도 빌려올 수 없어서 합치는 함수

1. 합칠 노드가 leafNode일 때
   오른쪽 노드 key, key-value 모두 왼쪽 노드에 추가, leftNode의 right은 rightNode의 right으로 바꿔줌
2. 합칠 노드가 internalNode일 때
   기존 노드에 부모 노드의 key와 오른쪽 노드의 key, children 추가

부모의 몇번째 key를 삭제할 것인가

부모의 가장 마지막 노드였다면 index-1번째 Key를 삭제
아니라면 index번째 key 삭제

Parent의 children에서 rightNode는 삭제

1. Parent가 root이고 key가 없다면 lefNode가 root가 됨
2. Parent가 root아닌데 underflowskTdmaus deleteAndReorganize호출

<singleKey 찾는 함수>

public boolean findSingleKey(**int** key)

찾았는지 여부를 알려주는 ifFound 반환

Root부터 타고 내려가면서 해당 노드의 key 모두 출력

찾았으면 value 출력

<rangedKeys찾는 함수>

public boolean findRangedKeys(**int** startKey, **int** endKey): 범위 내의 Key 찾는 함수

찾았는지 여부를 알려주는 flag 반환

startKey가 있는 leafNode findLeafNode 통해 찾기

그 노드에서 시작하여 right로 오른쪽으로 이동하며 startKey보다 크거나 같고 endKey보다 작거나 같은 Key-value쌍 출력

<BPlusTree를 저장하고 불러오는 함수들

public void saveTree(String indexFile) : tree 저장 함수

맨 첫번째 줄에는 sizeOfNode 출력

노드 하나씩 저장하는 saveNode에 root부터 전달

public void saveNode(BufferedWriter bw, BPTNode node)**throws** IOException : node 저장 함수

saveTree()이후 줄 부터 작성

leafNode이면 1, 아니면 0 작성

그 다음 줄에는 node의 key를 ‘,’로 구분하여 출력

그 다음 줄 부터는 key-value쌍 ‘,’로 구분하여 한 줄 씩 출력

재귀적으로 saveNode호출

public void loadTree(String indexFile): tree 로드하는 함수

previosLeaf null로 초기화

indexFile의 첫째 줄 에서 sizeOfNode 가져옴

loadNode로 root부터 불러옴

public BPTNode loadNode(BufferedReader br)**throws** IOException

loadTree()이후 줄 부터 읽기

1이면 leafNode, 0이면 internalNode

그 다음줄 부터 ‘,’로 구분된 key들 읽어서 node에 add

leafNode라면 key-value도 있으므로 ‘,’로 구분된 Key-value 쌍 한 줄 씩 읽으며 node에 Put

previousLeaf가 null이 아니면 right에 node 넣어주고 previousLeaf는 node로 변경

internalNode라면 child와 parent관계 처리

불러온 node 반환

3. Instructions for compiling source