B+ Tree Assignment

Implementation of a B+ Tree Index

과목명 | 데이터베이스시스템및응용

담당교수 | 김상욱 교수님

학과 | 컴퓨터소프트웨어학부

학년 | 2학년

학번 | 2019014320

이름 | 최지석

제출일 | 2022-09-23

1. Summary of Algorithm

B+Tree Operation

1. Insertion

- key-value 쌍의 데이터가 주어지면, B+트리에서 해당 key 값이 들어갈 수 있는 위치의 Leaf 노드를 찾는다. (B+트리에서 데이터는 반드시 Leaf 노드에 저장된다.)
- Leaf 노드를 찾은 다음, 삽입 이후 키의 개수가 최대 키의 개수를 넘지 않는다면 바로 해당 데이터를 삽입한다.
- 만약 삽입 이후 키의 개수가 최대 키의 개수를 넘는다면, 데이터 삽입 이후 split이 일어나 현재 키의 개수가 최대 키의 개수를 넘지 않는 노드들로 분할된다.

2. Single key Search

- key 값이 주어지면, B+트리에서 해당 key 값이 존재하는 Leaf 노드를 찾는다.
- 해당 key 값을 갖는 데이터를 보유한 Leaf 노드가 존재하면, 해당 데이터의 value 값을 출력한다.
- 만약 key 값을 갖는 데이터를 보유한 Leaf 노드가 없을 경우, "NOT FOUND"를 출력한다.

3. Ranged Search

- key 값의 상한(Upper Bound)과 하한(Lower Bound)이 주어지면, 하한보다 크거나 같은 key 값을 갖는 데이터 중에서 가장 작은 데이터를 보유한 Leaf 노드를 찾는다.
- 해당 Leaf 노드에서부터 오른쪽 Leaf 노드들로 이동하면서, 하한보다 크거나 같고, 상한보다 작거나 같은 key 값을 갖는 데이터들을 모두 출력한다.
- 이때, B+트리의 Leaf 노드들은 자신의 오른쪽 노드를 가리키는 포인터를 갖고 있으므로, 해당 포인터를 따라가며 데이터를 탐색한다.
- 만약 주어진 범위 내의 key 값을 갖는 데이터를 보유한 Leaf 노드가 없을 경우, "NOT FOUND"를 출력한다.

4. Deletion

- key 값이 주어지면, B+트리에서 해당 key 값이 존재하는 Leaf 노드를 찾는다.
- Leaf 노드를 찾은 다음, 삭제 이후의 키의 개수가 최소 키의 개수보다 크거나 같다면 바로 해당 데이터를 삭제한다.
- 만약 삭제 이후의 키의 개수가 최소 키의 개수보다 작다면, 데이터 삭제 이후 우선 왼쪽 또는 오른쪽 형제 노드에서 데이터를 하나 빌려오는 Redistribution 을 시도한다.
- 이때, 양쪽 형제 노드 모두 삭제 이후의 키의 개수가 최소 키의 개수보다 작다면, Redistribution 대신 두 형제 노드 중 하나의 노드와 Merge 를 수행한다.

(※ 자세한 내용은 2장 'Program Description' 에 설명되어 있습니다.)

2. Program Description

DBProgram.java

1. Class Introduction

- 사용자로부터 명령어를 입력받아 인덱스 파일 조작하는 클래스.
- Command-Line 에서 명령어를 입력받아 다음과 같은 작업을 수행할 수 있다.

① Data File Creation

- B+트리 노드의 Degree가 저장되어있는 새로운 index_file 을 생성한다.
  (이미 같은 이름의 파일이 존재할 경우, 해당 파일에 덮어쓴다.)

✓ Command : *DBProgram* -c *index_file b*

· *DBProgram* : 데이터베이스 조작에 사용할 프로그램.

· *index_file* : key-value 데이터를 저장할 새로운 인덱스 파일.

· *b* : B+트리의 하나의 노드가 가질 수 있는 최대 자식 노드의 개수(Degree).

ex) **java DBProgram –c index.dat 8**

② Insertion

- data_file (CSV 파일) 에 저장되어있는 key-value 데이터를 B+트리에 삽입한다.
- 삽입이 끝나면, 트리에 저장된 모든 데이터를 다시 index_file 에 저장한다.

✓ Command : *DBProgram* -i *index_file data_file*

· *data_file* : index_file 에 새로 저장될 key-value 데이터가 저장된 입력 데이터 파일.

ex) **java DBProgram –i index.dat input.csv**

③ Deletion

- data_file (CSV 파일) 에 저장되어있는 key 값을 갖는 key-value 데이터를 B+트리에서 삭제한다.
- 삭제가 끝나면, 트리에 저장된 모든 데이터를 다시 index_file 에 저장한다.

✓ Command : *DBProgram* -d *index_file data_file*

· *data_file* : index_file 에서 삭제될 key-value 데이터의 key 값이 저장된 입력 데이터 파일.

ex) **java DBProgram –d index.dat delete.csv**

④ Single Key Search

- B+트리에서 주어진 key 값을 갖는 데이터의 value 값을 출력한다.
- 탐색 과정에서 방문한 모든 Non-leaf 노드의 key 값들도 차례대로 출력한다.
- 만약 해당 key 값을 갖는 데이터가 없는 경우, "NOT FOUND" 를 출력한다.

✓ Command : *DBProgram* -s *index_file key*

· *key* : B+트리에서 찾을 key-value 데이터의 key 값.

ex) **java DBProgram –s index.dat 125**

⑤ Ranged Search

- B+트리에서 주어진 범위 내의 key 값을 갖는 데이터들의 key-value 값을 출력한다.
- 만약 해당 범위 내의 key 값을 갖는 데이터가 없는 경우, "NOT FOUND" 를 출력한다.

✓ Command : *DBProgram* -r *index_file start_key end_key*

· *start_key* : 범위 탐색에서 찾을 데이터 key 값의 lower bound (하한).

· *end_key* : 범위 탐색에서 찾을 데이터 key 값의 upper bound (상한).

ex) **java DBProgram –r index.dat 100 200**

2. Method Introduction

1) public static void main (String[] args)

▪ Command-line에서 argument를 입력받아 Data File Creation, Insertion, Deletion, Single Key Search, Ranged Search 등의 기능을 수행하는 메소드.

(1) Data File Creation

- B+트리 노드의 Degree가 저장되어있는 index_file을 생성한다.

(2) Insertion

- readBPTree 메소드를 호출하여 index_file에 저장된 데이터들로 B+트리를 만든다.
- data_file에 저장되어있는 key-value 데이터를 하나씩(한 줄) 읽어 들이면서, insert 메소드를 호출하여 해당 데이터들을 B+트리에 삽입한다.
- 삽입이 끝나면 saveBPTree 메소드를 호출하여 B+트리에 저장된 데이터들을 index_file 에 다시 저장한다.

(3) Deletion

- readBPTree 메소드를 호출하여 index_file에 저장된 데이터들로 B+트리를 만든다.
- data_file에 저장되어있는 key 값들을 하나씩(한 줄) 읽어 들이면서, delete 메소드를 호출하여 B+트리에서 해당 key 값을 갖는 데이터들을 삭제한다.
- 삭제가 끝나면 saveBPTree 메소드를 호출하여 B+트리에 저장된 데이터들을 index_file 에 다시 저장한다.

(4) Single Key Search

- readBPTree 메소드를 호출하여 index_file에 저장된 데이터들로 B+트리를 만든다.
- singleSearch 메소드를 호출하여 B+트리에서 주어진 key 값을 갖는 데이터를 탐색하여 출력한다.

(5) Ranged Search

- readBPTree 메소드를 호출하여 index_file에 저장된 데이터들로 B+트리를 만든다.
- rangeSearch 메소드를 호출하여 B+트리에서 주어진 범위 내의 key 값을 갖는 데이터들을 모두 탐색하여 출력한다.

2) public static BPlusTree readBPTree (String indexFile)

- indexFile에 저장된 데이터들로 B+트리를 만드는 메소드.
  - 우선 argument로 입력받은 index_file을 연 다음, 맨 첫 줄에 있는 Degree를 읽어 들여 해당 Degree를 갖는 B+트리를 만든다.
  - 그다음, 한 줄씩 저장되어있는 key-value 데이터를 읽어 들여 B+Tree에 삽입한다.
  - 삽입이 끝나면 해당 B+트리를 반환한다.

3) public static void readBPTree (String indexFile, BPlusTree bpTree)

- bpTree에 저장된 데이터들을 indexFile에 저장하는 메소드.
  - 우선 argument로 입력받은 bpTree의 Degree를 가져온다.
  - 그다음, argument로 입력받은 index_file을 열어서 맨 첫 줄에 Degree를 작성한다.
  - bpTree가 비어 있지 않다면, bpTree의 맨 왼쪽 Leaf 노드에 저장된 key-value 데이터부터 차례대로 index_file에 한 줄씩 작성한다.
  - 더 이상 작성할 데이터가 없는 경우, 함수를 종료한다.

DataPair.java

1. Class Introduction

- B+트리의 Leaf 노드에 저장되는 key-value 데이터를 구현한 클래스.
- DataPair 클래스는 다음과 같은 필드(Field)를 갖는다.
  - int key : Leaf 노드에 저장되는 키값.
    (Non-leaf 노드의 경우, key 값이 Leaf 노드에 저장된 데이터를 탐색하는 데 사용된다.)
  - int value : Leaf 노드에 저장되는 실제 데이터 값.
  - Node leftChild : DataPair의 키값보다 작은 키값들을 갖는 왼쪽 자식 노드를 가리키는 포인터.

2. Method Introduction

1) public DataPair (int key, int value) (Constructor)

- DataPair를 초기화하는 생성자.
  - key, value 값은 argument로 받아 저장하고, leftChild에는 null을 저장한다.

2) public int getKey ( )

- 현재 DataPair에 저장되어있는 key 값을 반환하는 메소드.

3) public int getValue ( )

- 현재 DataPair에 저장되어있는 value 값을 반환하는 메소드.

4) public Node getLeftChild()
- 현재 DataPair에 저장되어있는 왼쪽 자식 노드인 leftChild를 반환하는 메소드.

5) public void setKey(int key)
- 현재 DataPair에 저장되어있는 키값을 주어진 key 값으로 설정하는 메소드.

6) public void setValue(int value)
- 현재 DataPair에 저장되어있는 실제 데이터 값을 주어진 value 값으로 설정하는 메소드.

7) public void setLeftChild(Node leftChild)
- 현재 DataPair에 저장되어있는 왼쪽 자식 노드를 주어진 leftChild로 설정하는 메소드.

8) public int compareTo(DataPair o)
- key 값을 기준으로 DataPair를 정렬하기 위해 사용되는 비교 메소드.

Node.java

1. Class Introduction

- B+트리의 노드를 구현한 클래스.
- Node 클래스는 다음과 같은 필드(Field)를 갖는다.
  - int maxNumOfKey : 노드가 가질 수 있는 최대 키의 개수. (Degree − 1)
  - int minNumOfKey : 노드가 가질 수 있는 최소 키의 개수. (⌊Degree/2⌋ - 1)
  - int currentNumOfKey : 현재 노드에 저장되어있는 키의 개수 (m).
  - boolean isLeaf : 현재 노드가 Leaf 노드인지를 나타내는 논리값.
  - DataPair[] keyPairArray : DataPair를 저장하는 배열.
  - Node parent : 현재 노드의 부모 노드를 가리키는 포인터.
  - Node rightSibling : 현재 노드의 오른쪽 형제 노드를 가리키는 포인터.
  - Node rightmostChild : Non-leaf 노드에서 가장 오른쪽 자식 노드를 가리키는 포인터.

2. Method Introduction

1) public Node(int maxNumOfKey) (Constructor)
- Node를 초기화하는 생성자.
  - maxNumOfKey는 argument로 받아 저장한다.
  - minNumOfKey는 '⌊((maxNumOfKey + 1) / 2)⌋ - 1'을 계산한 값을 저장한다.
  - currentNumOfKey는 0으로 설정하고, isLeaf는 true로 설정한다.

- keyPairArray는 ‘maxNumOfKey + 1’ 만큼의 크기를 갖도록 설정한다.
  (노드의 Split을 용이하게 하기 위함이다.)
- parent, rightSibling, rightmostChild는 모두 null로 설정한다.

2) public Node push (int key, int value, DataPair fromChild)

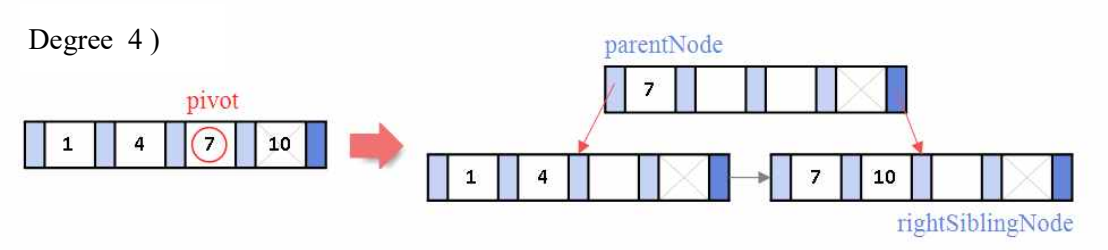
- B+트리의 노드에 주어진 key-value 값을 갖는 DataPair를 push 하는 메소드.
- push 이후 B+트리의 루트 노드를 찾아 반환한다.

(1) Leaf 노드에 push 하는 경우

- 해당 DataPair를 keyPairArray에 추가한 뒤 정렬한다.
- push로 인해 노드가 가득 찼고,

① 부모 노드가 없는 경우,

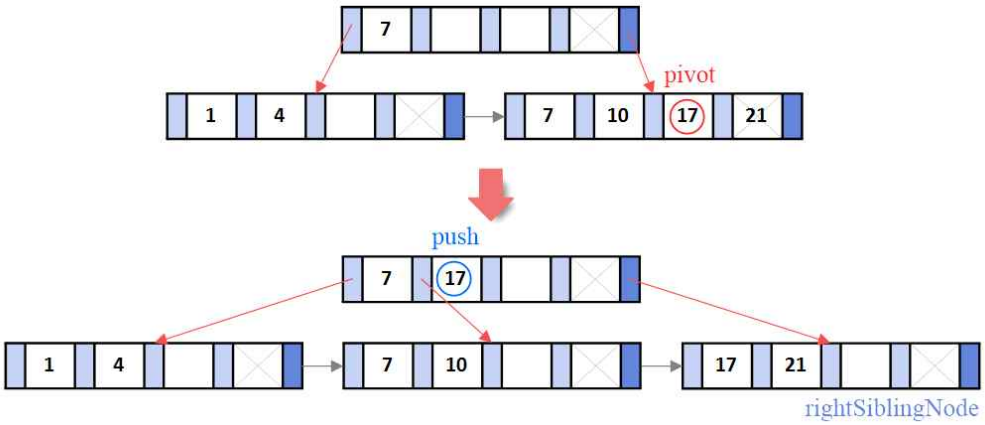
- 우선 parentNode와 rightSiblingNode 노드를 만든다.
- keyPairArray 배열의 중간에 위치한 DataPair를 가리키는 pivot 인덱스를 설정한 다음, pivot 인덱스에 위치한 DataPair를 parentNode에 추가한다.
- parentNode에 새로 저장된 DataPair의 leftChild를 현재 노드로,
  parentNode의 rightmostChild를 현재 노드의 오른쪽 자식 노드로 설정한다.
- rightSiblingNode에 pivot DataPair를 포함한 오른쪽 절반만큼의 DataPair를 복사한다. (Shallow Copy)
- 이때, rightSiblingNode에 복사한 DataPair들은 현재 노드에서 제거한다.
- 현재 노드와 rightSiblingNode의 rightSibling, parent를 재설정한다.



,

② 부모 노드가 있는 경우,

- 우선 rightSiblingNode 노드를 만든다.
- keyPairArray 배열의 중간에 위치한 DataPair를 가리키는 pivot 인덱스를 설정한다.
- pivot 인덱스에 위치한 DataPair를 parentDataPair 변수에 저장한다.
- rightSiblingNode에 pivot DataPair를 포함한 오른쪽 절반만큼의 DataPair를 복사한다.
  (rightSiblingNode에 복사한 DataPair들은 현재 노드에서 제거한다.)
- 현재 노드와 rightSiblingNode의 rightSibling을 재설정한다.
- rightSiblingNode의 parent를 재설정한다.
- parentDataPair에 저장된 왼쪽 자식 노드를 현재 노드로 설정한다.
- parent에 (key, value, parentDataPair)를 argument로 넘겨 push를 호출한다.

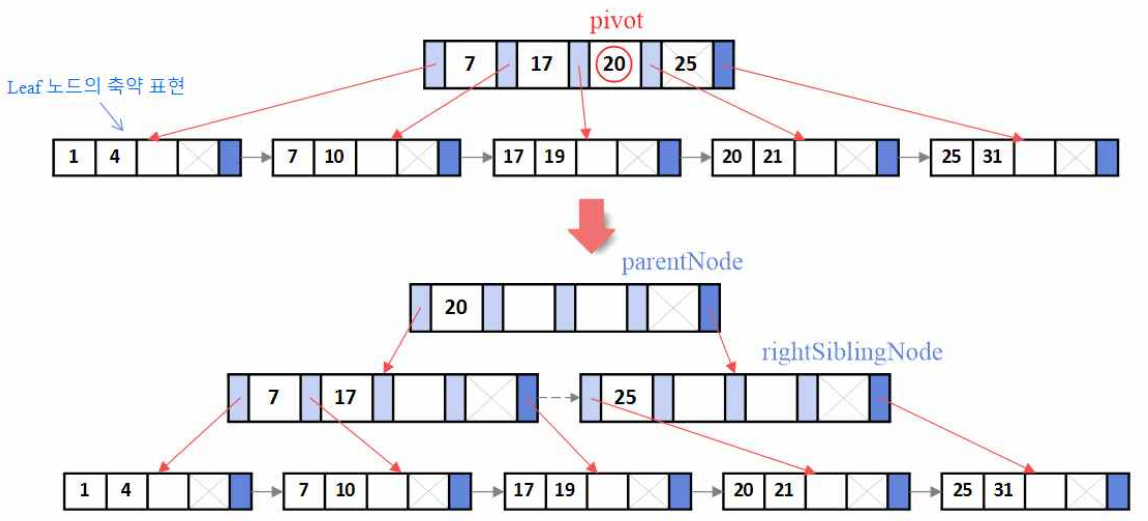


(2) Non-leaf 노드에 push 하는 경우

- fromChild argument를 통해 전달받은 DataPair를 keyPairArray에 추가한 뒤 정렬한다.
- keyPairArray 안에 저장된 DataPair들의 왼쪽 자식 노드와 rightmostChild를 업데이트 한다.
- push로 인해 노드가 가득 찼고,

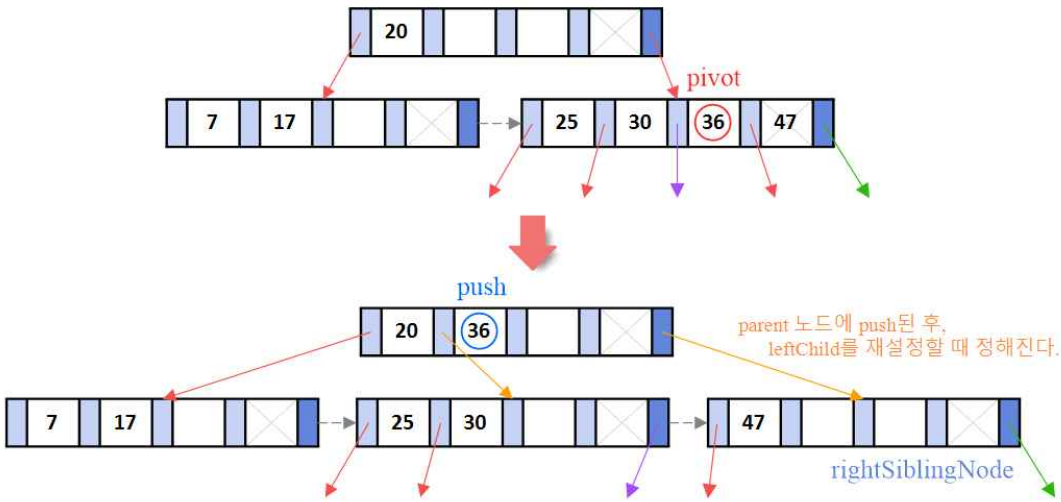
① 부모 노드가 없는 경우,

- 우선 parentNode와 rightSiblingNode 노드를 만든다.
- keyPairArray 배열의 중간에 위치한 DataPair를 가리키는 pivot 인덱스를 설정한 다음, pivot 인덱스에 위치한 DataPair를 parentNode에 추가한다.
- parentNode에 새로 저장된 DataPair의 leftChild를 현재 노드로, parentNode의 rightmostChild를 현재 노드의 오른쪽 자식 노드로 설정한다.
- 현재 노드와 rightSiblingNode의 rightSibling, rightmostChild를 재설정한다.
- 현재 노드에서 pivot DataPair를 제거한다.
- rightSiblingNode에 pivot DataPair를 포함하지 않은 오른쪽 절반만큼의 DataPair를 복사한다. (Shallow Copy)
- 이때, rightSiblingNode에 복사한 DataPair들은 현재 노드에서 제거한다.
- 현재 노드와 rightSiblingNode의 parent를 재설정한다.



② 부모 노드가 없는 경우,

- 우선 rightSiblingNode 노드를 만든다.
- keyPairArray 배열의 중간에 위치한 DataPair를 가리키는 pivot 인덱스를 설정한다.
- pivot 인덱스에 위치한 DataPair를 parentDataPair 변수에 저장한다.
- 현재 노드와 rightSiblingNode의 rightSibling, rightmostChild를 재설정한다.
- 현재 노드에서 pivot DataPair를 제거한다.
- rightSiblingNode에 pivot DataPair를 포함하지 않은 오른쪽 절반만큼의 DataPair를 복사한다. (Shallow Copy)
- rightSiblingNode의 parent를 재설정한다.
- parentDataPair에 저장된 왼쪽 자식 노드를 현재 노드로 설정한다.
- parent에 (key, value, parentDataPair)를 argument로 넘겨 push를 호출한다.



2) public Node remove (int key)

- B+트리에서 주어진 key 값을 갖는 DataPair를 찾아 삭제하는 메소드.
- 삭제 이후 B+트리의 루트 노드를 찾아 반환한다.

- 먼저 현재 노드에서 입력받은 key 값을 갖는 DataPair가 존재하는지 확인한다.
- DataPair가 존재하지 않으면 별도의 행동 없이 루트 노드를 찾아 반환한다.
- DataPair가 존재할 때,

(1) Leaf 노드인 동시에 루트 노드인 경우

- 해당 DataPair만 삭제하고 정렬한다.

(2) (루트 노드가 아닌) Leaf 노드인 경우

① 삭제 이후 키의 개수가 최소 키 개수보다 크거나 같은 경우

- 해당 DataPair를 삭제하고 정렬한다.
- 삭제할 데이터의 key 값이 부모 노드의 key 값으로도 사용되는 경우, 부모 노드와 루트 노드의 자식 노드를 업데이트한다.

② 삭제 이후 키의 개수가 최소 키 개수보다 작은 경우

(ⅰ) 현재 노드가 부모 노드의 rightmostChild 가 아니고, 오른쪽 형제 노드에 저장된 키의 개수가 충분한 경우

- 삭제 후 Redistribution을 통해 오른쪽 형제 노드에서 DataPair 하나를 가져온다.

(ⅱ) 현재 노드가 부모 노드의 leftmostChild가 아니고, 왼쪽 형제 노드에 저장된 키의 개수가 충분한 경우

- 삭제 후 Redistribution을 통해 왼쪽 형제 노드에서 DataPair 하나를 가져온다.

(ⅲ) 양쪽 형제 노드 모두 키의 개수가 충분하지 않고, 현재 노드가 rightmostChild가 아닌 경우

- 삭제 후 Merge를 통해 오른쪽 형제 노드와 병합한다.

(ⅳ) 왼쪽 형제 노드의 키의 개수가 충분하지 않은 rightmostChild인 경우

- 삭제 후 Merge를 통해 왼쪽 형제 노드와 병합한다.

(3) (루트 노드가 아닌) Non-leaf 노드인 경우

① 삭제 이후 키의 개수가 최소 키 개수보다 크거나 같은 경우

- 해당 DataPair를 삭제하고 정렬한다.

② 삭제 이후 키의 개수가 최소 키 개수보다 작은 경우

- Leaf 노드인 경우의 ②번 과정과 동일하다.

(4) (Leaf 노드가 아닌) 루트 노드인 경우

- 해당 DataPair를 삭제하고 정렬한다.
- 삭제 이후 루트 노드에 더 이상 DataPair가 남아 있지 않은 경우,
  삭제 이전에 남아 있던 DataPair의 leftChild가 가리키던 노드를 새 루트 노드로 만든다.
  (기존의 루트 노드는 삭제된다.)

3) public void redistribute (int deleteDataIndex, String lr)

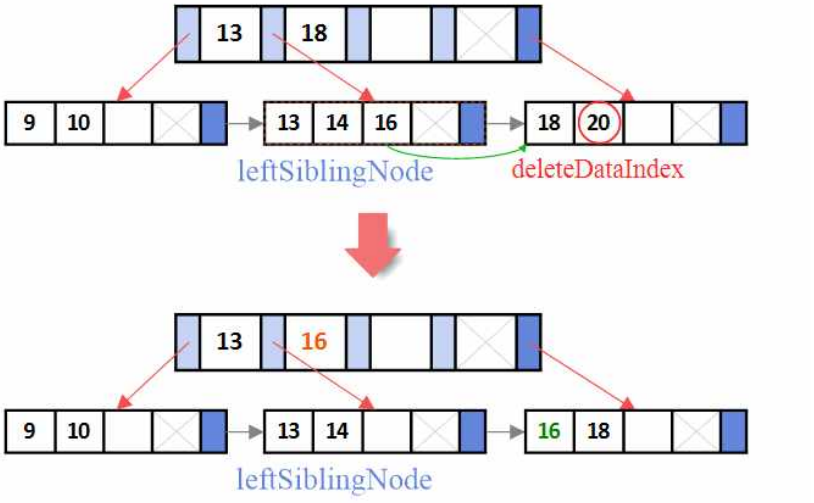
- 노드의 keyPairArray에서 deleteDataIndex 위치에 있는 DataPair를 찾아 삭제하고, 형제 노드에서 DataPair를 하나 가져오는 메소드.
- deleteDataIndex는 현재 노드의 keyPairArray에서 삭제할 DataPair의 위치를, lr은 어느 형제 노드에서 DataPair를 가져올 것인지를 나타내는 argument이다.

(1) Leaf 노드에서 Redistribution을 하는 경우

- 우선 deleteDataIndex 위치에 있는 DataPair를 keyPairArray에서 제거하고, 정렬한다.

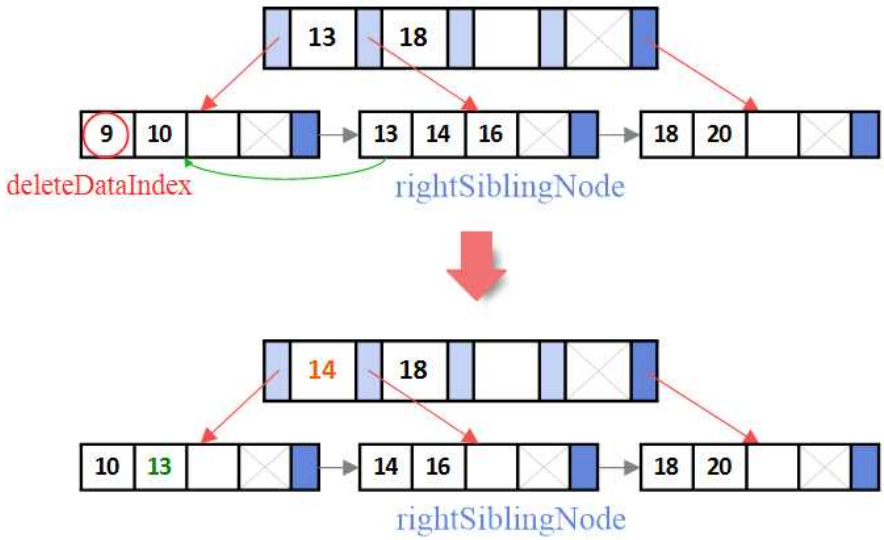
① 왼쪽 형제 노드에서 DataPair를 가져올 경우

- 왼쪽 형제 노드에서 가장 큰 key 값을 갖는 DataPair를 가져와 현재 노드에 추가한 다음, 왼쪽 형제 노드와 현재 노드를 정렬한다.



② 오른쪽 형제 노드에서 DataPair를 가져올 경우

- 오른쪽 형제 노드에서 가장 작은 key 값을 갖는 DataPair를 가져와 현재 노드에 추가한 다음, 오른쪽 형제 노드와 현재 노드를 정렬한다.



- 부모 노드와 루트 노드의 자식 노드들을 업데이트한다.

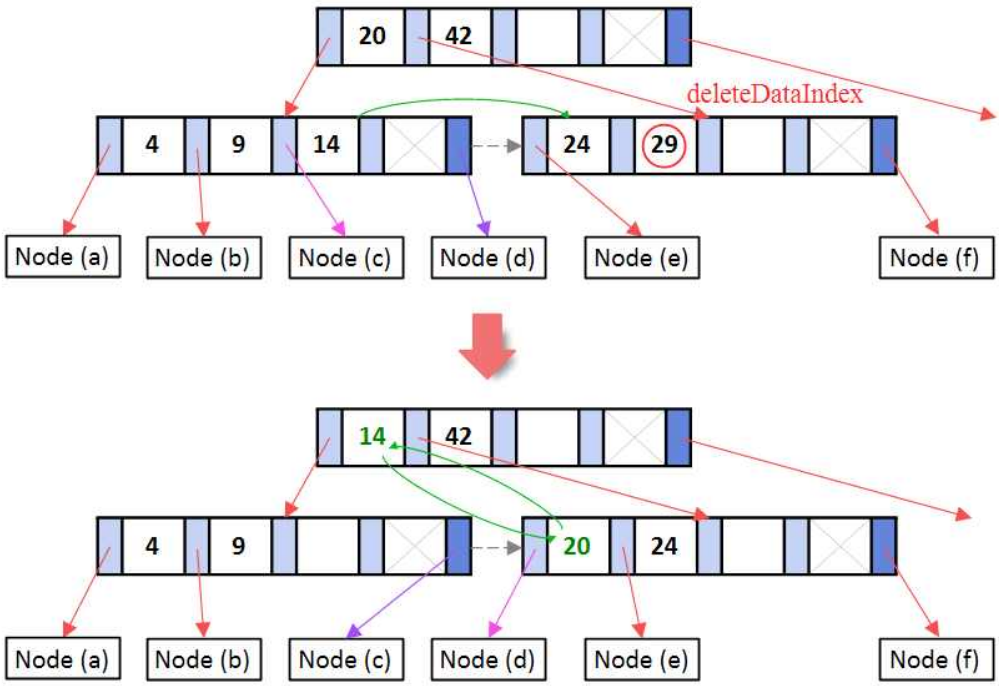
(2) Non-Leaf 노드에서 Redistribution을 하는 경우

- 우선 deleteDataIndex 위치에 있는 DataPair를 keyPairArray에서 제거하고, 정렬한다.

① 왼쪽 형제 노드에서 DataPair를 가져올 경우

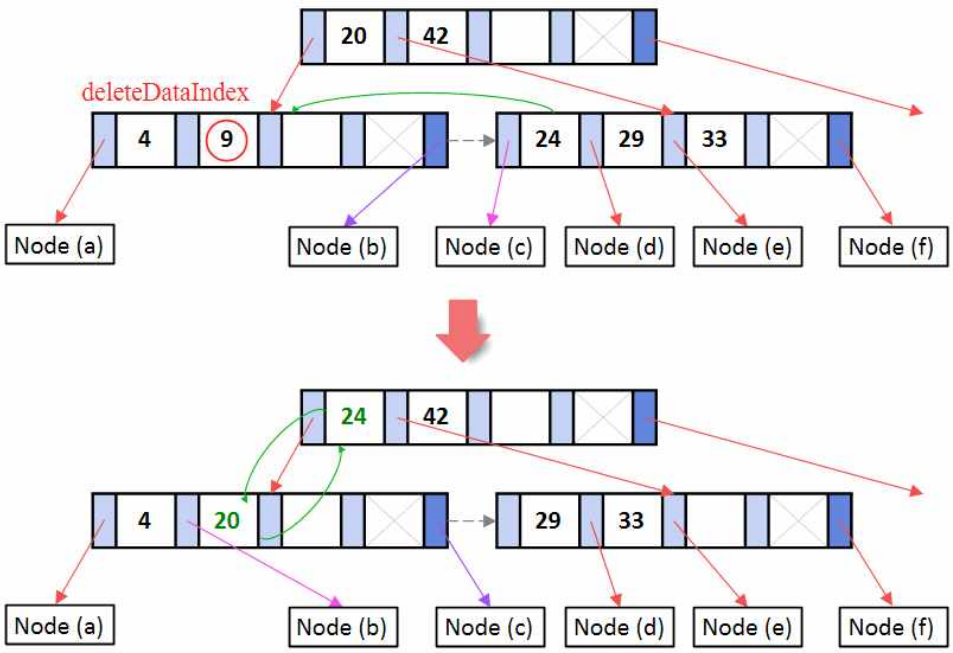
- 왼쪽 형제 노드의 rightmostChild와 왼쪽 형제 노드 keyPairArray의 맨 끝에 있는 DataPair의 leftChild가 가리키는 노드를 서로 교체한다.
- 왼쪽 형제 노드 keyPairArray의 맨 끝에 있는 DataPair를 가져와 현재 노드 keyPairArray의 맨 앞에 추가한다.

- 왼쪽 형제 노드에서 옮긴 DataPair의 (key, value)와 왼쪽 형제 노드를 leftChild로 갖는 부모 노드의 DataPair의 (key, value)를 서로 바꿔준다.



② 오른쪽 형제 노드에서 DataPair를 가져올 경우

- 오른쪽 형제 노드에서 가장 작은 key 값을 갖는 DataPair를 가져와 현재 노드 keyPairArray의 맨 끝에 추가한다.
- 오른쪽 형제 노드에서 옮긴 DataPair의 leftChild가 가리키는 노드와 현재 노드의 rightmostChild가 가리키는 노드를 서로 교체한다.
- 오른쪽 형제 노드에서 옮긴 DataPair의 (key, value)와 현재 노드를 leftChild로 갖는 부모 노드의 dataPair의 (key, value)를 서로 바꿔준다.



4) public Node merge (int deleteDataIndex, String lr)

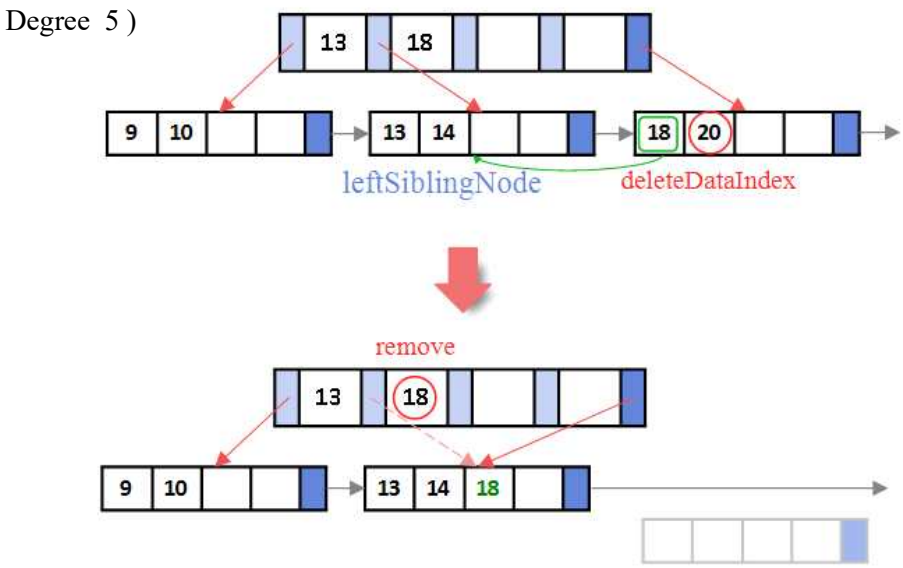
- 노드의 keyPairArray에서 deleteDataIndex 위치에 있는 DataPair를 찾아 삭제하고, 형제 노드와 현재 노드를 병합하여 하나의 노드로 만드는 메소드.
- deleteDataIndex는 현재 노드의 keyPairArray에서 삭제할 DataPair의 위치를, lr은 어느 형제 노드와 병합할지를 나타내는 argument이다.
- Leaf 노드의 병합이 끝나면, B+트리의 루트 노드를 찾아 반환한다.

- 우선 deleteDataIndex 위치에 있는 DataPair를 keyPairArray에서 제거하고, 정렬한다.

(1) Leaf 노드에서 Merge 하는 경우

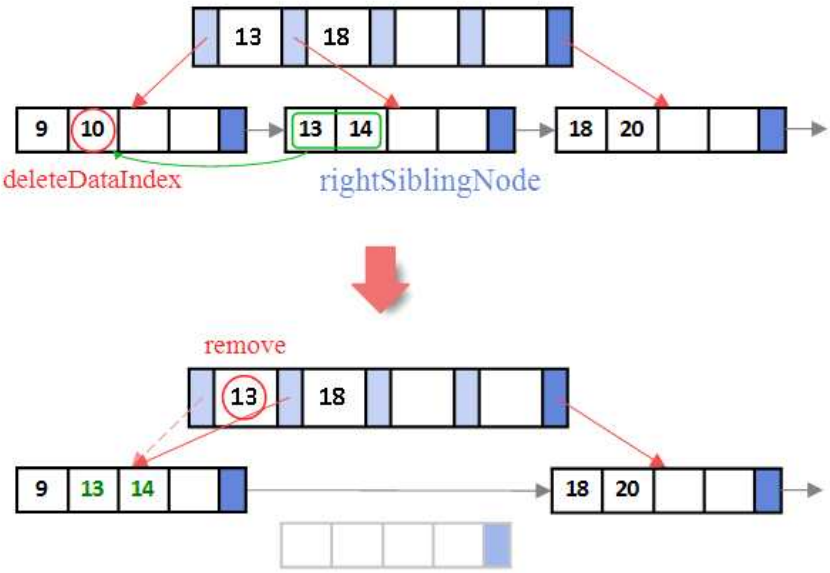
① 왼쪽 형제 노드와 병합하는 경우 (현재 노드가 parent의 rightmostChild인 경우)

- 현재 노드의 모든 DataPair를 왼쪽 형제 노드에 복사한다.
- 왼쪽 형제 노드의 rightSibling을 현재 노드의 rightSibling으로 설정한다.
- 부모 노드의 rightmostChild를 왼쪽 형제 노드로 설정한다.
- 부모 노드에서 leftSiblingNode를 leftChild로 갖는 DataPair를 remove 한다.
- leftSiblingNode의 parent가 null이 아닐 경우, leftSiblingNode를 기준으로 한 부모 노드와 루트 노드의 자식 노드를 업데이트한다.



② 오른쪽 형제 노드와 병합하는 경우

- 오른쪽 형제 노드의 모든 DataPair를 현재 노드에 복사한다.
- 현재 노드의 rightSibling를 오른쪽 형제 노드의 rightSibling으로 설정한다.
- 부모 노드에서 오른쪽 형제 노드가 leftChild로 저장된 DataPair의 leftChild를 현재 노드로 설정한다.
- 부모 노드에서 현재 노드를 leftChild로 갖는 DataPair를 remove 한다.
- 현재 노드의 parent가 null이 아닐 경우, 현재 노드를 기준으로 한 부모 노드와 루트 노드의 자식 노드를 업데이트한다.

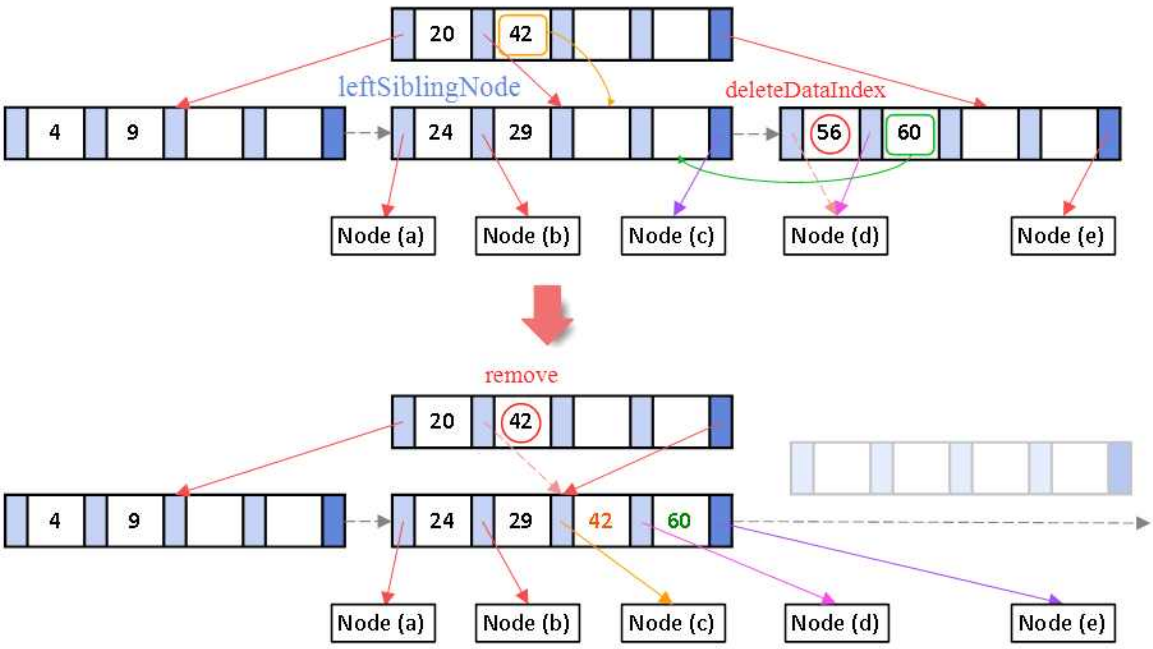


(2) Non-Leaf 노드에서 Merge 하는 경우

- 우선 deleteDataIndex 위치에 있는 DataPair를 keyPairArray에서 제거하고, 정렬한다.

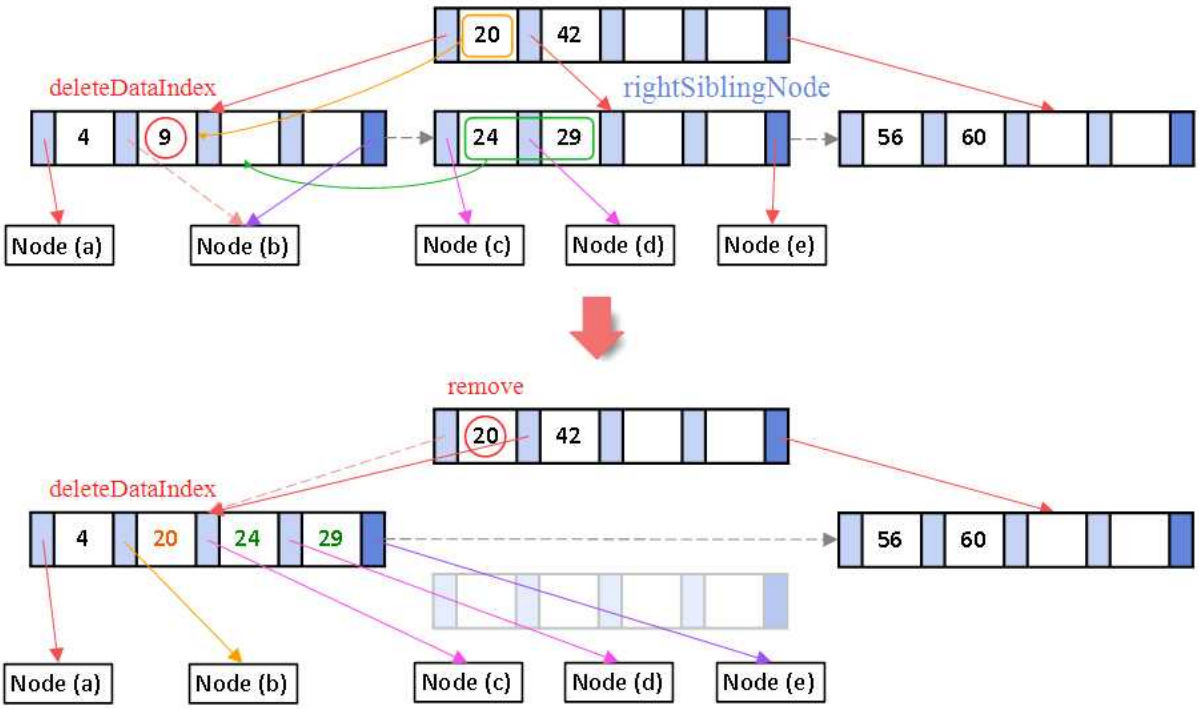
① 왼쪽 형제 노드와 병합하는 경우 (현재 노드가 parent의 rightmostChild인 경우)

- 부모 노드에서 왼쪽 형제 노드를 leftChild로 갖는 DataPair의 (key, value)를 갖는 새로운 DataPair를 만들어 leftSiblingNode keyPairArray의 맨 뒤에 추가한다.
- 새로운 DataPair가 가리키는 leftChild는 왼쪽 형제 노드의 rightmostChild이다.
- 현재 노드의 모든 DataPair를 왼쪽 형제 노드에 복사한다.
- 왼쪽 형제 노드의 rightSibling을 현재 노드의 rightSibling으로 설정한다.
- 부모 노드의 rightmostChild를 왼쪽 형제 노드로 설정한다.
- 부모 노드에서 leftSiblingNode를 leftChild로 갖는 DataPair를 remove 한다.



② 오른쪽 형제 노드와 병합하는 경우

- 부모 노드에서 현재 노드를 leftChild로 갖는 DataPair의 (key, value)를 갖는 새로운 DataPair를 만들어 현재 노드 keyPairArray의 맨 뒤에 추가한다.
- 새로운 DataPair가 가리키는 leftChild는 현재 노드의 rightmostChild이다.
- 오른쪽 형제 노드의 모든 DataPair를 현재 노드에 복사한다.
- 현재 노드의 rightSibling를 오른쪽 형제 노드의 rightSibling으로 설정한다.
- 부모 노드에서 오른쪽 형제 노드가 leftChild로 저장된 DataPair의 leftChild를 현재 노드로 설정한다.
- 부모 노드에서 현재 노드를 leftChild로 갖는 DataPair를 remove 한다.



5) public int getCurrentNumOfKey ( )

- 현재 노드에 저장되어있는 키의 개수(currentNumOfKey)를 반환하는 메소드.

6) public boolean isLeafNode ( )

- 현재 노드가 리프 노드인지에 대한 논리값(isLeaf)을 반환하는 메소드.

7) public Node getRightmostChild ( )

- 현재 노드의 rightmostChild를 반환하는 메소드.

8) public Node getRightSibling ( )

- 현재 노드의 rightSibling을 반환하는 메소드.

9) public DataPair[] getkeyPairArray ( )

- 현재 노드의 keyPairArray를 반환하는 메소드.

10) private void swapData (DataPair nodeA, DataPair nodeB)

- argument로 입력받은 두 DataPair의 key-value 데이터를 서로 바꾸는 메소드.

11) private int findParentDataPairIndex ( )

- 현재 노드가 부모 노드에 저장된 몇 번째 DataPair의 leftChild인지를 반환하는 메소드.
- 현재 노드가 부모 노드의 rightmostChild인 경우에는 -1 을 반환한다.

12) private Node getRightSiblingNode ( )

- 현재 노드의 오른쪽 형제 노드를 반환하는 메소드.
- getRightSibling 메소드와 달리, 현재 노드와 같은 부모 노드를 갖는 형제 노드들만 반환할 수 있다. (rightSilbing이 다른 부모 노드의 자식 노드일 경우 null을 반환한다.)

13) private Node getLeftSiblingNode ( )

- 현재 노드의 왼쪽 형제 노드를 반환하는 메소드.

14) private void updateParentKeys ( )

- 현재 노드(Non-leaf)의 key와 value를 업데이트하는 메소드.

15) private void updateRootKeys ( )

- 현재 노드를 기준으로, 루트 노드(Non-leaf)의 key와 value를 업데이트하는 메소드.

16) private DataPair getMinimumDataPair ( )

- 현재 노드를 루트로 하는 트리의 Leaf 노드들 중에서, 가장 작은 key 값을 갖는 DataPair를 찾아 반환하는 메소드

17) private Node findRoot()

- 현재 노드를 기준으로, 루트 노드를 찾아 반환하는 메소드.

18) private void pullForward (int i)

- 현재 노드의 keyPairArray에서 i 번째에 있는 삭제될 DataPair를 기준으로, 뒤에 있는 DataPair들을 한 칸씩 앞으로 옮기는 메소드.
- 결과적으로 i 번째에 있는 DataPair는 삭제된다.

19) private void pullBack (int i)

- 현재 노드의 keyPairArray에서 i 번째에 있는 DataPair를 포함하여, 뒤에 있는 모든 DataPair들을 한 칸씩 뒤로 옮기는 메소드.

20) private void addCurrentNumOfKey (int increment)

- 현재 노드의 currentNumOfKey에 argument로 입력받은 increment만큼 더하는 메소드.

BPlusTree.java

1. Class Introduction

- B+트리를 구현한 클래스.
- BPlusTree 클래스는 다음과 같은 필드(Field)를 갖는다.
  - · Node root : B+트리의 루트 노드
  - · int maxNumOfChild : B+트리의 노드가 가질 수 있는 최대 자식 노드의 개수

2. Method Introduction

1) public BPlusTree (int maxNumOfChild) (Constructor)

- BPlusTree를 초기화하는 생성자.
  - maxNumOfChild는 argument로 받아 저장한다.
  - root에는 최대 'maxNumOfChild - 1' 개의 key 값을 가질 수 있는 새 노드를 저장한다.

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

- B+트리에 key-value 데이터를 삽입하는 메소드.
  - 루트 노드를 currentNode로 설정한다.
  - currentNode가 Leaf 노드가 될 때까지 트리를 탐색하여 아래로 내려간다.
    ① 입력받은 key 값이 currentNode keyPairArray에 있는 어떤 DataPair의 키값보다 작을 경우, 해당 DataPair의 leftChild가 가리키는 노드로 이동한다.
    ② 입력받은 key 값이 currentNode keyPairArray에 있는 모든 DataPair의 키값보다 클 경우, currentNode의 rightmostChild가 가리키는 노드로 이동한다.
  - currentNode가 Leaf 노드가 됐을 때, 해당 노드에 key-value 데이터를 push 한다.
  - 이때 push 메소드가 반환한 노드를 새로운 루트 노드로 설정한다.

3) public void singleSearch (int key)

- B+트리에서 주어진 키값을 갖는 데이터가 존재할 경우, 이를 출력하는 메소드.
  - 루트 노드를 currentNode로 설정한다.
  - currentNode가 Leaf 노드가 될 때까지 트리를 탐색하여 아래로 내려간다.
  - 이때 Leaf 노드까지 내려가면서 탐색한 모든 DataPair의 키값들을 prevKeys에 순서대로 concatenate 한다. (각 키값은 공백으로 구분한다.)
  - currentNode가 Leaf 노드가 됐을 때, 우선 prevKeys에 저장되어있는 문자열을 출력한다. (prevKeys에 저장된 문자열의 길이가 0이면 출력하지 않는다.)
  - 그다음, 해당 Leaf 노드에 주어진 key 값을 갖는 DataPair가 있을 경우, 해당 DataPair의 value 값을 다음 줄에 출력한다.
    (주어진 key 값을 갖는 DataPair가 없을 경우, "NOT FOUND"를 출력한다.)

4) public void rangeSearch (int startKey, int endKey)

- B+트리에서 주어진 범위 내의 키값을 갖는 데이터가 존재할 경우, 이를 모두 출력하는 메소드.
  - 루트 노드를 currentNode로 설정한다.
  - currentNode가 Leaf 노드가 될 때까지 트리를 탐색하여 아래로 내려간다.
  - currentNode가 Leaf 노드가 됐을 때, 해당 노드 keyPairArray에 있는 DataPair 중에서 key 값이 startKey 값보다 크거나 같고, endKey 값보다 작거나 같은 DataPair들의 key-value 데이터를 콤마로 구분하여 한 줄씩 출력한다.
    (해당 범위 내의 키값을 갖는 DataPair가 없을 경우, "NOT FOUND"를 출력하고 함수를 종료한다.)
  - 만약 해당 노드의 모든 DataPair의 키값이 startKey 값보다 크거나 같고, endKey 값보다 작거나 같을 경우, rightSibling이 가리키는 노드로 이동하여 Leaf 노드에서 수행한 작업을 반복한다.
  - endKey 값보다 큰 DataPair를 만나거나, rightSibling이 null인 경우 함수를 종료한다.

5) public void delete (int key)

- B+트리에서 주어진 key 값을 갖는 데이터가 존재할 경우, 이를 삭제하는 메소드.
  - 루트 노드를 currentNode로 설정한다.
  - currentNode가 Leaf 노드가 될 때까지 트리를 탐색하여 아래로 내려간다.
  - currentNode가 Leaf 노드가 됐을 때, 해당 노드에서 주어진 key 값을 갖는 key-value 데이터를 remove 한다.
  - 이때 remove 메소드가 반환한 노드를 새로운 루트 노드로 설정한다.

6) private int getMaxNumOfChild ( )

- B+트리의 노드가 가질 수 있는 최대 키의 개수를 반환하는 메소드.

7) private Node getLeftmostNode ( )

- B+트리에서 가장 왼쪽에 있는 Leaf 노드를 반환하는 메소드.

8) private boolean isEmpty ( )

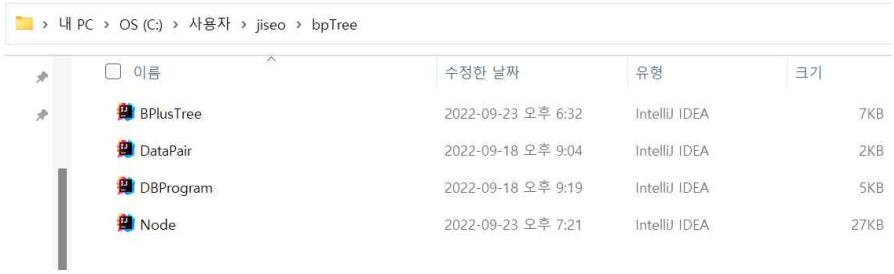
- B+트리가 비어 있는지에 대한 논리값을 반환하는 메소드.

3. Compilation and Execution

(1) Compilation

1. Source 폴더 안의 모든 코드를 하나의 폴더에 저장한 뒤, 해당 폴더에서 CMD 열기

- ① Source 폴더 안의 모든 코드를 하나의 폴더(ex - bpTree 폴더)에 저장한다.



- ② 해당 폴더에서 CMD(명령 프롬프트)를 연다.

2. 해당 폴더 내부의 모든 코드 컴파일

- ① CMD에서 "javac *.java"라고 입력하여 4개의 코드를 모두 컴파일한다.

- ② 그러면 다음과 같이 BPlusTree.class, DataPair.class, DBProgram.class, Node.class 파일이 새로 생성된 것을 확인할 수 있다.

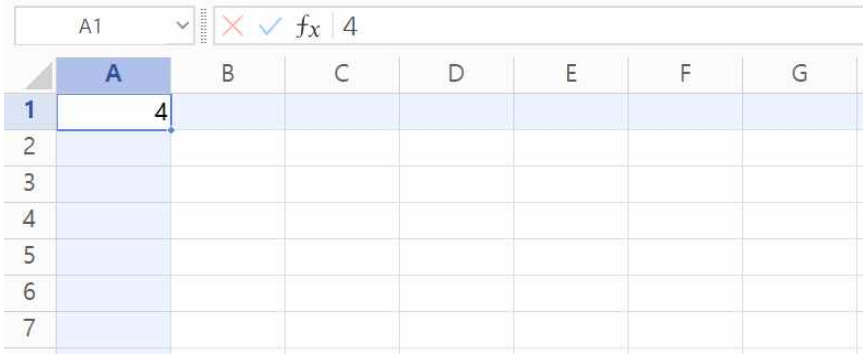
(2) Execution

1. Data File Creation

- ① 컴파일이 완료되었다면, "java DBProgram -c index_file b" 명령어를 사용하여 index_file(인덱스 파일)을 생성한다.
- ② index_file은 생성할 인덱스 파일을, b는 B+트리의 Degree를 의미한다.

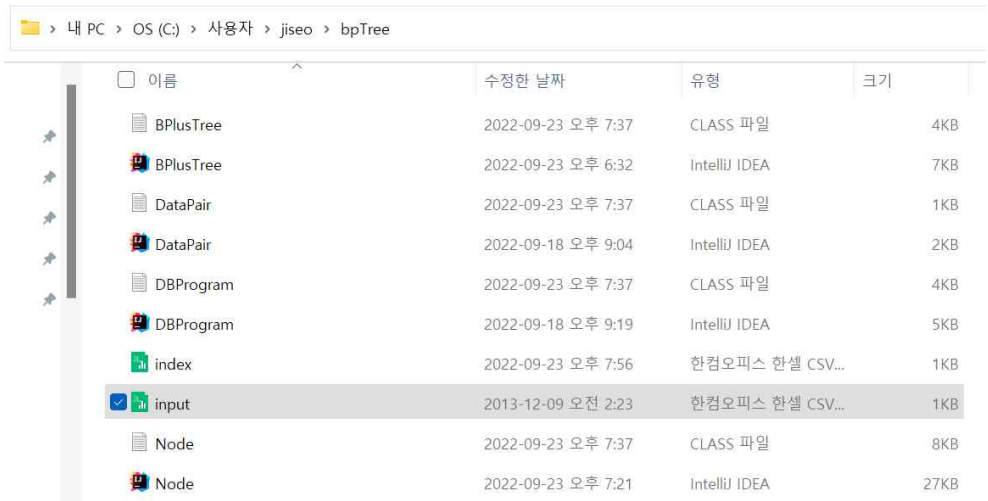
```
C:₩Users₩jiseo₩bpTree>java DBProgram -c index.csv 4
```

- ③ 생성된 index_file을 열어보면 다음과 같이 B+트리의 Degree가 첫 줄에 적혀 있는 것을 확인할 수 있다.



2. Insertion

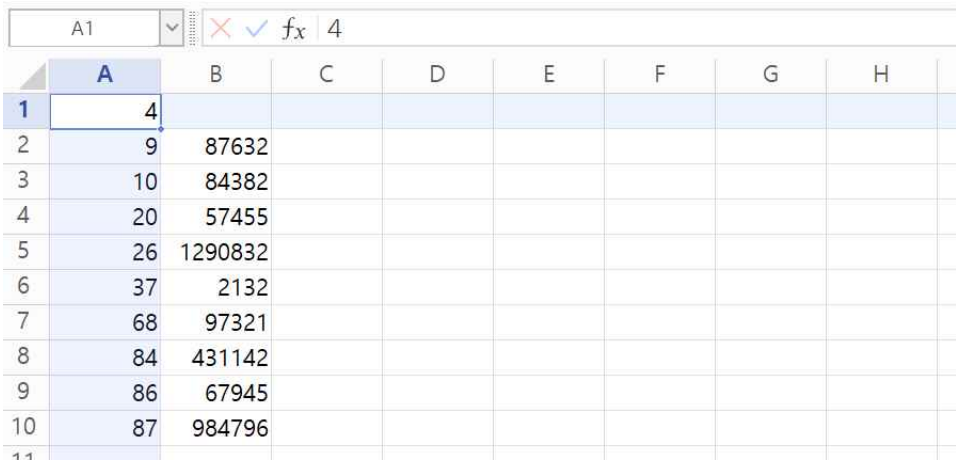
- ① index_file이 생성된 후, 컴파일된 코드와 index_file이 존재하는 폴더에 Input file을 위치시킨다.



- ② "java DBProgram -i index_file input_file" 명령어를 사용하여 Insertion을 수행한다.
- ③ input_file은 index_file에 저장할 데이터가 저장되어있는 Input file을 의미한다.

```
C:₩Users₩jiseo₩bpTree>java DBProgram -i index.csv input.csv
```

- ④ index_file을 열어보면 다음과 같이 key 값을 기준으로 데이터가 오름차순으로 정렬되어 저장된 것을 확인할 수 있다.



	A	B	C	D	E	F	G	H
1	4							
2	9	87632						
3	10	84382						
4	20	57455						
5	26	1290832						
6	37	2132						
7	68	97321						
8	84	431142						
9	86	67945						
10	87	984796						

3. Single Key Search

- ① index_file이 생성된 후, "java DBProgram -s index_file key" 명령어를 사용하여 Single Key Search를 수행한다.
- ② key는 B+트리에서 탐색할 key-value 데이터의 키값을 의미한다.
- ③ 먼저 해당 key 값을 갖는 Leaf 노드를 찾아 내려가면서, 방문한 Non-Leaf 노드에 저장된 DataPair의 key 값들이 (한 줄에) 순서대로 출력된다.
- ④ Leaf 노드에 해당 key 값을 갖는 데이터가 존재할 경우, 해당 데이터의 value 값이 다음 줄에 출력되고, 존재하지 않으면 "NOT FOUND"가 다음 줄에 출력된다.

```
C:₩Users₩jiseo₩bpTree>java DBProgram -s index.csv 37
84
2132

C:₩Users₩jiseo₩bpTree>java DBProgram -s index.csv 3
20
NOT FOUND
```

4. Ranged Search

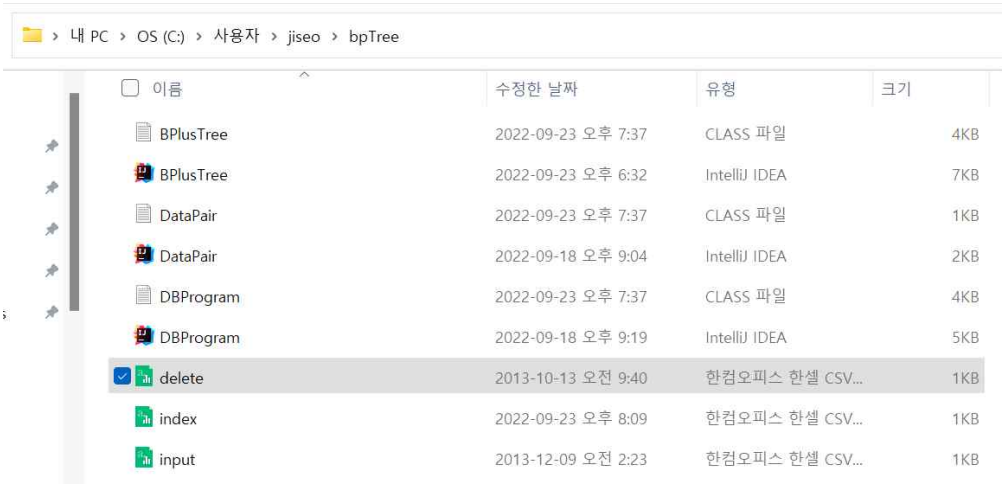
- ① index_file이 생성된 후, "java DBProgram -r index_file start_key end_key" 명령어를 사용하여 Ranged Search를 수행한다.
- ② start_key는 탐색할 key 값 범위의 하한(Lower Bound)을, end_key는 탐색할 key 값 범위의 상한(Upper Bound)를 의미한다.
- ③ 해당 범위 내의 key 값을 갖는 데이터들이 B+트리 내에 존재할 경우, 해당 데이터들의 key-value 쌍이 콤마로 구분되어 키값을 기준으로 (한 줄씩) 오름차순으로 출력된다.
- ④ 해당 범위 내의 key 값을 갖는 데이터가 B+트리 내에 존재하지 않는다면 "NOT FOUND"가 출력된다.

```
C:₩Users₩jiseo₩bpTree>java DBProgram -r index.csv 1 30
9,87632
10,84382
20,57455
26,1290832

C:₩Users₩jiseo₩bpTree>java DBProgram -r index.csv 100 200
NOT FOUND
```

5. Deletion

- ① index_file이 생성된 후, 컴파일된 코드와 index_file이 존재하는 폴더에 Input file을 위치시킨다.



- ② "java DBProgram -d index_file delete_file" 명령어를 사용하여 Deletion을 수행한다.
- ③ delete_file은 index_file에서 삭제할 데이터의 key 값들이 저장되어있는 Input File을 의미한다.

```
C:₩Users₩jiseo₩bpTree>java DBProgram -d index.csv delete.csv
```

- ③ index_file을 열어보면, 다음과 같이 delete_file에 있는 key 값을 갖는 데이터들이 삭제되고, 남은 데이터들이 key 값을 기준으로 오름차순으로 정렬되어 저장된 것을 확인할 수 있다.