**B+ Tree 검색엔진 구현**

컴퓨터공학과

20153055 박태형

목차

1. 개발 목적
2. 프로그램 알고리즘 및 구조도
3. 결과
4. 개발목적

B+트리의 구현 및 효율적인 자료구조 학습

1. 프로그램 알고리즘 및 구조도

전제: 트리의 리프노드에 실질적인 정보가 모두 담겨있어야 하며, 리프노드가 아닌 노드는 인덱스 역할만 한다.

차수가 3이고 리프노드는 서로 연결(이중연결리스트)되어 정보를 선형탐색 할 수 있다.

트리에 단어가 들어온다고 가정하고 모든 경우의 수를 살펴본다.

**① 단어 삽입 부분**

**1-1. 트리가 없을 경우**

- 단어 값을 지니는 노드를 생성하고 루트로 지정한다.

**1-2. 트리가 있을 경우**

**1) 입력 단어가 단어1과 같은 경우**

가. 노드가 리프노드의 경우

가-1. 노드가 가진 단어 정보와 일치하는 문서가 있을 경우

가-2. 노드가 가진 단어 정보와 다른 문서일 경우

나. 노드가 리프노드가 아닐 경우

나-1. 해당 노드에 단어가 1개만 있을 경우

나-2. 해당 노드에 단어가 2개일 경우

**2) 입력 단어가 단어 2와 같은 경우**

가. 노드가 리프노드의 경우

가-1. 노드가 가진 단어 정보와 일치하는 문서가 있을 경우

가-2. 노드가 가진 단어 정보와 다른 문서일 경우

나. 노드가 리프노드가 아닐 경우

**3) 노드가 리프노드의 경우**

3.1 노드에 단어가 하나만 있을 경우

3.1.1 입력 < 단어1 경우

3.1.2 단어1 < 입력 경우

3.2 노드에 단어가 두개 있을 경우

3.2.1 입력 < 단어1 < 단어2 경우

- 임시노드에 저장 후 리프노드분해(leafSplit)

3.2.2 단어1 < 입력 < 단어2 경우

- 임시노드에 저장 후 리프노드분해(leafSplit)

3.2.3 단어1 < 단어2 < 입력 경우

- 임시노드에 저장 후 리프노드분해(leafSplit)

**4) 노드가 리프노드가 아닐 경우**

4.1 단어2가 없을 경우

4.1.1 입력<단어1 경우

노드의 왼쪽으로 이동

4.1.2 단어1<입력 경우

노드의 오른쪽으로 이동

4.2 단어2가 있을 경우

4.2.1 입력 < 단어1 < 단어2 경우

노드의 왼쪽 이동

4.2.2 단어1 < 입력 < 단어2 경우

노드의 중간으로 이동

4.2.3 단어1 < 단어2 < 입력 경우

노드의 오른쪽으로 이동

**② 리프노드 분해부분**

**1. 부모노드가 없을 경우**

- 분해 후 새로운 부모를 만들어 루트 지정

**2. 부모노드가 있을 경우**

2.1 부모노드에 단어가 1개일 경우

2.1.1 부모노드의 왼쪽에서 분해 경우

2.1.2 부모노드의 오른쪽에서 분해 경우

2.2 부모노드에 단어가 2개일 경우

2.2.1 부모노드의 왼쪽에서 분해 경우 + 추가 비리프노드 분해

2.2.2 부모노드의 중간에서 분해 경우 + 추가 비리프노드 분해

2.2.3 부모노드의 오른쪽에서 분해 경우 + 추가 비리프노드 분해

**③ 비(non)리프노드 분해 부분**

**1. 부모노드가 없을 경우**

- 분해 후 새로운 부모를 만들어 루트 지정

**2. 부모노드 있을 경우**

2.1 부모노드에 단어가 1개일 경우

2.1.1 부모노드의 왼쪽에서 분해 경우

2.1.2 부모노드의 오른쪽에서 분해 경우

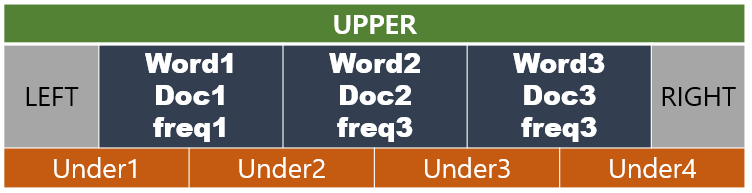
2.2 부모노드에 단어가 2개일 경우

2.2.1 부모노드의 왼쪽에서 분해 경우 + 추가 비리프노드 분해

2.2.2 부모노드의 중간에서 분해 경우 + 추가 비리프노드 분해

2.2.3 부모노드의 오른쪽에서 분해 경우 + 추가 비리프노드 분해

**B+Tree에 사용된 노드의 구조는 다음과 같다.**



**차수가 3인 B+Tree**는 노드안에 **최대 3개의 서브트리, 2개의 단어**를 담을 수 있다. 그러나 추가적인 **노드분열를 고려하여** 각각 1개씩 여유를 두어, **최대 4개의 서브트리, 3개의 단어를** 담고 분열을 할 수 있도록 설계하였다.

**UPPER:** 부모노드와의 접근을 위한 포인터

**LEFT, RIGHT**: 리프노드에서 리프노드간의 접근을 위한 포인터

**Under1, Under2, Under3, Under4:** 하위 노드들에 대한 접근을 위한 포인터

B+Tree 에 사용된 코드들은 첨부파일을 참조

아래는 주요 함수들이다.

**BPlusTree.java**

boolean isLeaf(Node node): 해당 노드가 리프노드인지 판별한다.

Node returnRoot(Node node): 해당 노드의 루트를 반환한다.

Node leafSplit(Node node): 해당 리프노드를 분열시킨다.

Node nonLeafSplit(Node node): 리프노드 분리 후, 추가적인 비리프노드를 분리할 때 쓰이는 함수이다.

Node insertKey(Node node, String input, String doc): 실질적인 단어와 관련 정보를 트리에 추가하는 함수이다.

Void height(Node node): 트리의 키를 반환하는 함수이다. 이 구현에서, 약 76000개의 문서를 색인 한 결과 **트리의 키는 17**이다.

Void search(Node node, String input): 검색을 위한 함수이다. 입력 단어를 트리에서 찾기위해 재귀함수를 사용하였다.

**Main.java**

String Remove\_SChar(String str): 한 문자열에서 특수문자를 제거하는 함수이다.

BPlusTree make\_tree(): 실질적으로 트리를 만드는 함수이다. 이는 AVL트리에서 구현한 부분을 BPlusTree에 맞게 수정하였다.

Void show\_dir(): 현재 디렉토리를 알려주는 함수이다.

Void search(BPlusTree BPT): 메인함수에서 실행되는 트리 단어 검색 함수이다.

Void sw(BPlusTree tree): 검색에서 사용하기 위한 함수로, 사용자에게 문자열을 입력 받는 함수이다.

Void ui(BPlusTree BPT): 사용자 인터페이스를 위한 함수이다.

Void show\_title(String file\_name): 파일 이름을 입력하면 제목을 보여주는 기능을 가진 함수이다.

Void main(): 메인 함수이다.

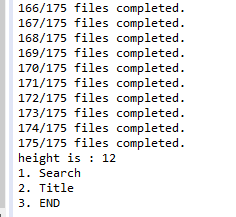
1. **결과**

약 76000개를 실행시켰을 경우, 색인 시간은 i5-4590, 4G RAM 기준 10~15분이 걸렸다.

처음에는 OutOfMemoryExecption 오류로 실행이 되지 않다가, 임시 객체들의 메모리를 반환하는 코드를 삽입하여 실행시간을 줄였다. 참고로, AMD R5 2600, 16G RAM에서는 5분~7분 사이를 기록했다.

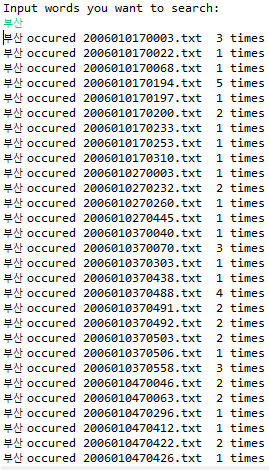
실행시간이 길어 175개의 파일만 넣고 실행하였다.

1. 실행



실행 시 파일 색인 완료정도가 출력되고 색인이 끝나면 트리의 높이, 간단한 조작을 할 수 있는 UI를 출력한다.

1. 검색



단어를 검색하면 해당 단어의 노드의 정보들이 출력된다.

1. 제목 기능

