B^+ Tree구현

(디스크 기반으로)

교재 482쪽 부록 A.4

개요

- 디스크기반 구현
  - 작업의 결과 발생한 트리구조가 디스크에 저장됨
  - 비휘발성 확보
- 블록 단위
  - 디스크 내의 페이지를 하나의 블록으로
    - 페이지에 하나의 노드를 할당, 반납
    - 내부 노드, 리프 노드 모두 하나의 페이지 단위로 처리함
  - 페이지란?
    - 화일 내의 일정한 크기를 가지는 연속된 공간
    - 보통 4~10Kbyte
    - 여기에서는 트리구조의 쉬운 식별을 위하여 64byte로 설정하고 수행함
- 데이터 레코드의 설정
  - (번호, 이름)이 간단한 구조를 사용
  - (Key, Value)의 필드명을 사용하며 자료형은 각각 int, char[12]
  - BTreePage.h 에서 정의 됨

페이지의 구성(1)

- 헤더 페이지
  - 항상 0페이지에 위치
  - BTreePage.h 에서 구조 정의 됨

```
struct _BTreeHeader {
int rootPage;              //루트페이지 번호
int firstSequencePage;     //시퀀스세트(리프노드)시작페이지
int order;                 //트리의 차수
int minKey;                // 내부노드의 최소 키 개수
int minRecord;             // 리프노드의 최소 레코드개수
int maxRecord;             // 리프노드의 최대 레코드개수
STACK *stack;
int stackTop, stackPtr;
```

페이지의 구성(2)

- 리프노드 페이지
  - 순차세트 노드 1개를 저장
  - 항상 1페이지 이후에 위치
  - BTreePage.h 에서 구조 정의 됨

최대 저장가능 레코드 수 의 계산

3*(int) + max* (Record) ≤ PageSize

max* (Record) ≤ PageSize – 3*(int)

max = {PageSize – 3*(int)} / (Record)

BTreePage.c

```
bTreeHeader->maxRecord=
(bufferManager->pageSize-sizeof(int) *3) /(sizeof(BTreeRecord));
```

페이지의 구성(3)

- 내부노드 페이지
  - 인덱스 노드 1개를 저장
  - 항상 1페이지 이후에 위치
  - BTreePage.h 에서 구조 정의 됨

PAGENO	NEXT	KEYCNT	CHILD(0)	KEY(0)	•••	CHILD(m-2)	KEY(m-2)	CHILD(m-1)

다음페이지 번호.
내부인지, 리프인지 구분
내부노드 =-1
리프노드 ≥ 0

트리차수의 계산

3*(int) + 2*(m-1) (int) + 1* (int) ≤ PageSize

2*(m-1) (int) ≤ PageSize - 4*(int)

최대 m = {PageSize - 4*(int)} / {2*(int)} + 1

BTreePage.c

```
bTreeHeader->order=
(int) ((bufferManager->pageSize-sizeof(int)*4)/(sizeof(int) *2))+1;
```

프로그램의 구성

- **헤더 파일**
  - –BTree.h : B^+-트리 연산을 정의한 파일
  - –BTreePage.h : B^+-트리 페이지 구조와 읽기, 쓰기 연산을 정의한 파일
  - –BaseHeader.h :기본적인 헤더 파일을 include, 기본 타입을 정의한 파일
  - –BufferManager.h : 디스— 버퍼를 관리하는 함수들을 정의한 파일
- **소스파일**
  - –BTree.c : 초기화 함수와 스택 연산을 구현한 파일
  - –BTreePage.c : B^+-트리 페이지 관리를 구현한 파일
  - –BTreeInsert.c : 삽입 연산을 구현한 파일
  - –BTreeDelete.c : 삭제 연산을 구현한 파일
  - –BTreeRetrieve.c : 검색 연산을 구현한 파일
  - –BTreeMain.c : B^+-트리를 구동하는 함수
  - –BufferManager.c : 디스크 버퍼 관리 함수를 구현한 파일

프로그램의 설명

레코드 탐색 (BTreeRetrieve.c)

```
BOOL findRecord(Key key, BTreePagePtr page) {
int i=0, targetPage=bTreeHeader->rootPage;
bTreeHeader->stackPtr=0;
readBTreePage(targetPage, page);
while(ISLEAF(page) == FALSE) {
for (i = 0; (i < KEYCNT(page)) && (KEY(page, i) < key); i++) {
;
}
push(PAGENO(page), i);
targetPage=CHILD(page, i);
readBTreePage(targetPage, page);
}
```

```
for (i = 0; (i < KEYCNT(page)) && (RECORD(page, i).key < key);
          i++) {
;
}


push(PAGENO(page), i);

if((i < KEYCNT(page)) && (key == RECORD(page, i).key)) {
return TRUE;
}
else {
return FALSE;
}
}
```

예를 들어 노드의 키값이
(5, 8, 10)이고
탐색 키값이 9라면
9보다 크거나 같아지는 최초의 키값
10을 찾을 때까지 반복

프로그램의 실행

- 명령어
  - h: 헤더페이지 보기
  - a: 모든 페이지 보기
  - i: 레코드 input(추가) (형식 : i 1 lee)
  - r: 레코드 read(검색) (형식 : r 1)
  - d: 레코드 delete(삭제) (형식 : d 1)
  - s: //순차검색
  - b: //순차택스트화일(backup.txt)로 백업하기
  - n: //데이터화일(data.txt) 초기화
  - o: //백업화일 가져오기

과제 수행

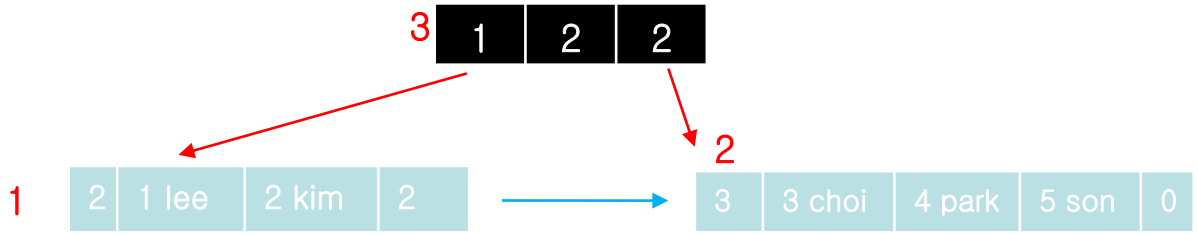
프로그램수행 시각화

프로그램의 실행 예

```
최초 실행 시 data.txt를 만들어 주고 초기화함
(페이지 크기는 64바이트로 가정)
헤더페이지
h
--------------------------------
루트노드 페이지번호   : 1
리프노드 첫페이지번호 : 1
B+ 트리의 차수        : 7
내부노드 최소키 개수  : 3
리프의 레코드 최대수  : 3
리프노드 최소키 개수  : 1
--------------------------------
a
페이지번호 : 1 (리프노드)
레코드개수 : 0
다음페이지 : 0
--------------------------------
삽입하기
i 1 lee
insert (1, lee) : success
i 2 kim
insert (2, kim) : success
i 3 choi
insert (3, choi) : success
i 4 park
insert (4, park) : success
i 5 son
insert (5, son) : success
```

```
모든 페이지 보기
a
페이지번호 : 1 (리프노드)
레코드개수 : 2
1 lee
2 kim
다음페이지 : 2

페이지번호 : 2 (리프노드)
레코드개수 : 3
3 choi
4 park
5 son
다음페이지 : 0

페이지번호   : 3 (내부노드)
다음페이지   : -1
키의개수     : 1
키리스트     : 1, (2), 2
```

```
직접접근
r 5
Retrive (5, son) : success
r 2
Retrive (2, kim) : success

순차접근
s
페이지번호 : 1   레코드개수 : 2
1 lee
2 kim
다음페이지 : 2

페이지번호 : 2   레코드개수 : 3
3 choi
4 park
5 son
다음페이지 : 0
```



프로그램의 실행 예

```
r 3
Retrive (3, choi) : success
d 8
Delete (8) : success
r 8
Retrive (8) : fail
d 3
Delete (3) : success
r 3
Retrive (3) : fail
d 4
Delete (4) : success


a
페이지번호 : 1 (리프노드)
레코드개수 : 2
1 lee
2 kim
다음페이지 : 2
```

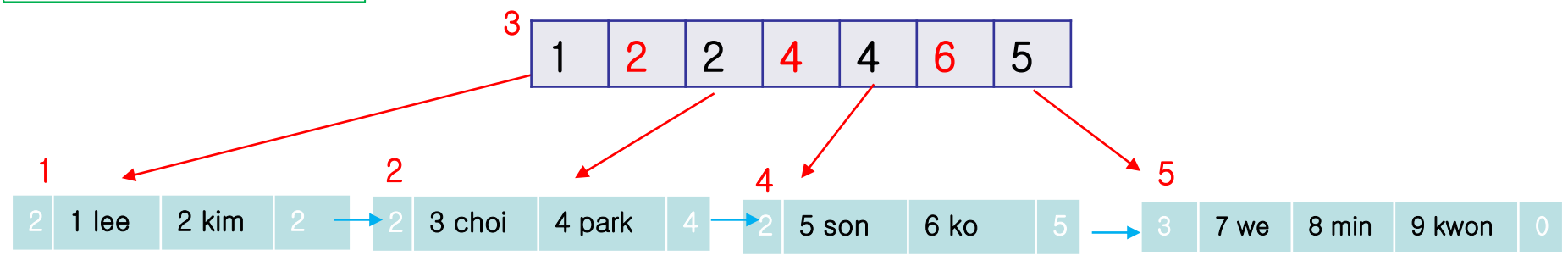
```
페이지번호 : 2 (리프노드)
레코드개수 : 1
5 son
다음페이지 : 4

페이지번호   : 3 (내부노드)
다음페이지   : -1
키의개수     : 3
키리스트     : 1, (2), 2, (5), 4, (6), 5

페이지번호 : 4 (리프노드)
레코드개수 : 1
6 ko
다음페이지 : 5

페이지번호 : 5 (리프노드)
레코드개수 : 2
7 we
9 kwon
다음페이지 : 0

x

이 창을 닫으려면 아무 키나 누르세요.
```

프로그램의 실행 예

```
i 6 ko
insert (6, ko) : success
i 7 we
insert (7, we) : success
i 8 min
insert (8, min) : success
i 9 kwon
insert (9, kwon) : success

a
페이지번호 : 1 (리프노드)
레코드개수 : 2
1 lee
2 kim
다음페이지 : 2
```

```
페이지번호 : 2 (리프노드)
레코드개수 : 2
3 choi
4 park
다음페이지 : 4

페이지번호  : 3 (내부노드)
다음페이지  : -1
키의개수    : 3
키리스트  : 1, (2), 2, (4), 4, (6), 5
```

```
페이지번호 : 4 (리프노드)
레코드개수 : 2
5 son
6 ko
다음페이지 : 5
페이지번호 : 5 (리프노드)
레코드개수 : 3
7 we
8 min
9 kwon
다음페이지 : 0
```



프로젝트 수행

- s: //순차검색
- b: //순차택스트화일(backup.txt)로 백업하기
- n: //데이터화일(data.txt) 초기화
- o: //백업화일 가져오기

추가 : 프로그램수행 시각화