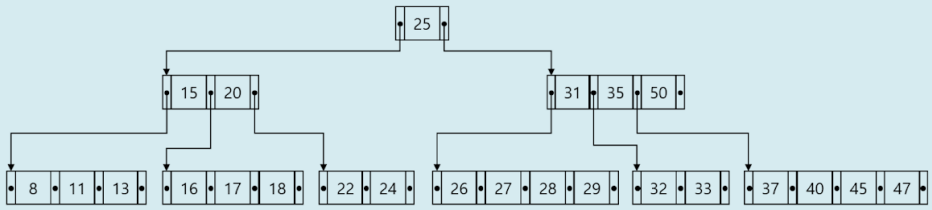
다음과 같은 B-tree가 있다. (capacity order d=2, m=2d+1=5)




```
nodeptr retrieve(char *skey, int *idx_found) {
  nodeptr curr = root;
  nodeptr P;
  int level = 1;
  int cnt = 0;
  int i;

  do {
    for (i = 0; i < curr->fill_cnt; i++) {
      cnt++;
      if (strcmp(skey, curr->rec[i].name) < 0) {
        break;
      }
      else if (strcmp(skey, curr->rec[i].name) == 0) {
        printf("\t레벨%d에 위치, 이 노드의 %d번째 레코드에 존재\n", level, i+1);
        *idx_found = i;
        return curr;
      else;
      }
    }

    P = curr->ptr[i];
    if (P) { curr = P; level++; }
  } while (P);

  return NULL;
}
```

스크린샷

(1) 위 B-tree에서 key 값이 36, 30인 레코드를 차례로 삽입하려고 한다.

(1-1) 삽입하는 과정에서 발행한 split의 총 횟수를 쓰시오. [2점]

split 총 횟수: 3 회

(1-2) 36, 30이 삽입된 후, B-tree 의 높이를 쓰시오. [2점]

B-tree 높이: 3

(1-3) 36, 30이 삽입된 후, 레벨 2의 맨 우측 노드 안의 키 값들을 쓰시오. [2점]

Answer: 40, 50

(1-4) 36, 30이 삽입된 후, key 값이 30인 레코드를 탐색하고자 할 때, 위 retrieval( ) 함수를 이용한다고 하자. 탐색이 완료되고 난 후 cnt의 값과 retrieve( ) 함수의 실행 결과를 쓰시오. [4점]

cnt: 2

retrieve( ) 함수 실행결과: 레벨 3 에 위치, 이 노드의 2 번째 레코드에 존재

(2) 위 B-tree에서 key 값이 17, 15, 13인 레코드를 차례로 삭제하려고 한다.

(2-1) 삭제 시 트리를 재조정하는 과정에서 발생한 redistribution의 총 횟수와 merge의 총 횟수를 쓰시오. redistribution을 할 때는 왼쪽 형제 노드를 먼저 검사해서 처리하도록 한다. merge를 할 때는 오른쪽 형제 노드를 먼저 검사해서 처리하도록 한다. [4점]

redistribution 횟수: 2 회

merge 횟수: 0 회

(2-2) 17, 15, 13이 삭제된 후, 레벨 2의 맨 좌측 노드 안의 키 값들을 쓰시오. [2점]

Answer: 8

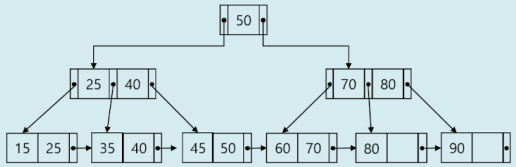
(2-3) 17, 15, 13이 삭제된 후, key 값이 20인 레코드를 탐색하고자 할 때, 위 retrieval( ) 함수를 이용한다고 하자. 탐색이 완료되고 난 후 cnt의 값과 retrieve( ) 함수의 실행 결과를 쓰시오. [4점]

스크린샷

cnt: 2

retrieve( ) 함수 실행결과: 레벨 2 에 위치, 이 노드의 2 번째 레코드에 존재

다음과 같이 인덱스 구조의 d=2(m=2d+1=5), 데이터 노드의 d=1(m=2d+1=3)인 B+ tree가 있다. key 값이 85, 75, 43인 레코드를 차례로 삽입하고자 한다.




```
void retrieve(char* name) {
  int i, j cnt=0;
  type_ptr_idxnode parent = NULL, curr = NULL;
  type_ptr_datanode curr_d = NULL;
  type_key in_key;
  strcpy(in_key, name);
  curr = ROOT;

  top = -1;
  if (ROOT->ptri[0] != NULL) {
    do {
      for (i = 0; i < curr->fill_cnt; i++) {
        cnt++;
        if (strcmp(in_key, curr->key[i]) <= 0)
          break;
      }
      push(curr);
      curr = curr->ptri[i];
      if (curr->ptri[0] == NULL)
        break;
    } while (1);
  }

  for (i = 0; i < curr->fill_cnt; i++) {
    cnt++;
    if (strcmp(in_key, curr->key[i]) <= 0)
      break;
  }

  parent = curr;
  curr_d = curr->ptrd[i];

  for (i = 0; i < curr_d->fill_cnt; i++) {
    cnt++;
    if (strcmp(in_key, curr_d->rec[i].name) < 0) {
      break;
    }
    else if (strcmp(in_key, curr_d->rec[i].name) == 0) {
      printf("cnt: %d\n", cnt);
      break;
    }
    else;
  }
  return;
}
```

스크린샷

(1) key 값이 85, 75, 43인 레코드를 삽입하는 과정에서 인덱스 구조에서 발생한 split의 총 횟수와 시퀀스 셋에서 발생한 split의 총 횟수를 쓰시오. [4점]

인덱스 구조에서 발생한 split의 총 횟수: 1 회

스크린샷

시퀀스 셋에서 발생한 split의 총 횟수: 1 회

(2) key 값이 85, 75, 43인 레코드를 삽입 후, B+ tree의 높이를 쓰시오. [2점]

B+ tree 높이: 3

(3) key 값이 85, 75, 43인 레코드를 삽입 후, key 값이 50인 레코드를 탐색하고자 할 때, 위 retrieve( ) 함수를 이용한다고 하자. retrieve( ) 함수의 실행 결과를 쓰시오. [2점]

cnt: 3

(4) key 값이 85, 75, 43인 레코드를 삽입 후, 레벨 3의 맨 우측 노드 안의 키 값들을 쓰시오. [2점]

Answer: 85,90