# Chapter 11: Indexing and Hashing


Revision by Gun-Woo Kim

Dept. of Computer Science and Engineering

Hanyang University


**Database System Concepts, 6th Ed.**

# Chapter 11:  Indexing and Hashing

11.1 Basic Concepts

11.2 Ordered Indices

11.3 $B^+$-Tree Index (Building and Insert)

# 11.1 Basic Concepts

Indexing mechanisms used to speed up access to desired data.

E.g., 찾아보기 (교제)

**Search Key** - attribute to set of attributes used to look up records in a file.

An **index file** consists of records (called **index entries**) of the form

| | |
|---|---|
| search-key | pointer |

Index files are typically much smaller than the original file

Two basic kinds of indices:

**Ordered indices:** search keys are stored in sorted order

(e.g. B+-Tree)

**Hash indices:** search keys are distributed uniformly across "buckets" using a "hash function". (생략함)

# 11.2 Ordered Indices

In an **ordered index,** index entries are stored sorted on the search key value. E.g., author catalog in library.

**Primary index:** in a sequentially ordered file, the index whose search key specifies the sequential order of the file.

뭔가 변동이 일어난후에 재정렬함

- Also called **clustering index**

secondary index보다 비용이 더 많이 들음

- The search key of a primary index is usually but not necessarily the primary key.

**Secondary index**: an index whose search key specifies an order different from the sequential order of the file. Also called non-clustering index.

**cluster: 그룹**

# Dense Index Files

**Dense index** — Index record appears for every search-key value in the file. 밀집인덱스는 주인덱스의 하나의 종류

E.g. index on *ID* attribute of *instructor* relation

| | |
|---|---|
| 10101 | |
| 12121 | |
| 15151 | |
| 22222 | |
| 32343 | |
| 33456 | |
| 45565 | |
| 58583 | |
| 76543 | |
| 76766 | |
| 83821 | |
| 98345 | |

| | | | | |
|---|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 | |
| 12121 | Wu | Finance | 90000 | |
| 15151 | Mozart | Music | 40000 | |
| 22222 | Einstein | Physics | 95000 | |
| 32343 | El Said | History | 60000 | |
| 33456 | Gold | Physics | 87000 | |
| 45565 | Katz | Comp. Sci. | 75000 | |
| 58583 | Califieri | History | 62000 | |
| 76543 | Singh | Finance | 80000 | |
| 76766 | Crick | Biology | 72000 | |
| 83821 | Brandt | Comp. Sci. | 92000 | |
| 98345 | Kim | Elec. Eng. | 80000 | |

# Dense Index Files (Cont.)

Dense index on *dept_name*, with *instructor* file sorted on *dept_name*

| | |
|---|---|
| Biology | |
| Comp. Sci. | |
| Elec. Eng. | |
| Finance | |
| History | |
| Music | |
| Physics | |

| | | | | |
|---|---|---|---|---|
| 76766 | Crick | Biology | 72000 | |
| 10101 | Srinivasan | Comp. Sci. | 65000 | |
| 45565 | Katz | Comp. Sci. | 75000 | |
| 83821 | Brandt | Comp. Sci. | 92000 | |
| 98345 | Kim | Elec. Eng. | 80000 | |
| 12121 | Wu | Finance | 90000 | |
| 76543 | Singh | Finance | 80000 | |
| 32343 | El Said | History | 60000 | |
| 58583 | Califieri | History | 62000 | |
| 15151 | Mozart | Music | 40000 | |
| 22222 | Einstein | Physics | 95000 | |
| 33465 | Gold | Physics | 87000 | |

# Sparse Index Files

**Sparse Index**:  contains index records for only some search-key values.          주인덱스의 방법중 하나

- Applicable when records are sequentially ordered on search-key

To locate a record with search-key value *K* we:

- Find index record with largest search-key value < *K*
- Search file sequentially starting at the record to which the index record points

| | |
|---|---|
| 10101 | |
| 32343 | |
| 76766 | |

| | | | | |
|---|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 | |
| 12121 | Wu | Finance | 90000 | |
| 15151 | Mozart | Music | 40000 | |
| 22222 | Einstein | Physics | 95000 | |
| 32343 | El Said | History | 60000 | |
| 33456 | Gold | Physics | 87000 | |
| 45565 | Katz | Comp. Sci. | 75000 | |
| 58583 | Califieri | History | 62000 | |
| 76543 | Singh | Finance | 80000 | |
| 76766 | Crick | Biology | 72000 | |
| 83821 | Brandt | Comp. Sci. | 92000 | |
| 98345 | Kim | Elec. Eng. | 80000 | |

즉 정렬된 튜플에
대해 위치 범위 포인터를 가져와
그사이에서 추가검색함

**Find 45565 !**

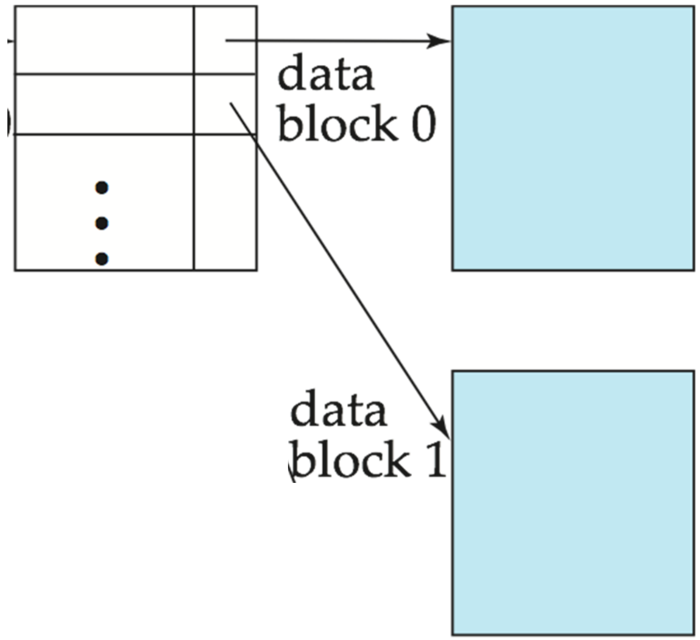
1. 45565보다 적은
가장 큰 색인키 → 32343
2. 32343부터 순차 검색

# Sparse Index Files (Cont.)

Compared to dense indices: **dense인덱스보다 공간(램)을 더 적게씀**

- Less space and less maintenance overhead for insertions and deletions.
- Generally slower than dense index for locating records.

**Good tradeoff**: sparse index with an index entry for every block in file, corresponding to least search-key value in the block.



# Multilevel Index

If primary index does not fit in memory, access becomes expensive.

Solution: treat primary index kept on disk as a sequential file and construct a sparse index on it.

- outer index – a sparse index of primary index
- inner index – the primary index file

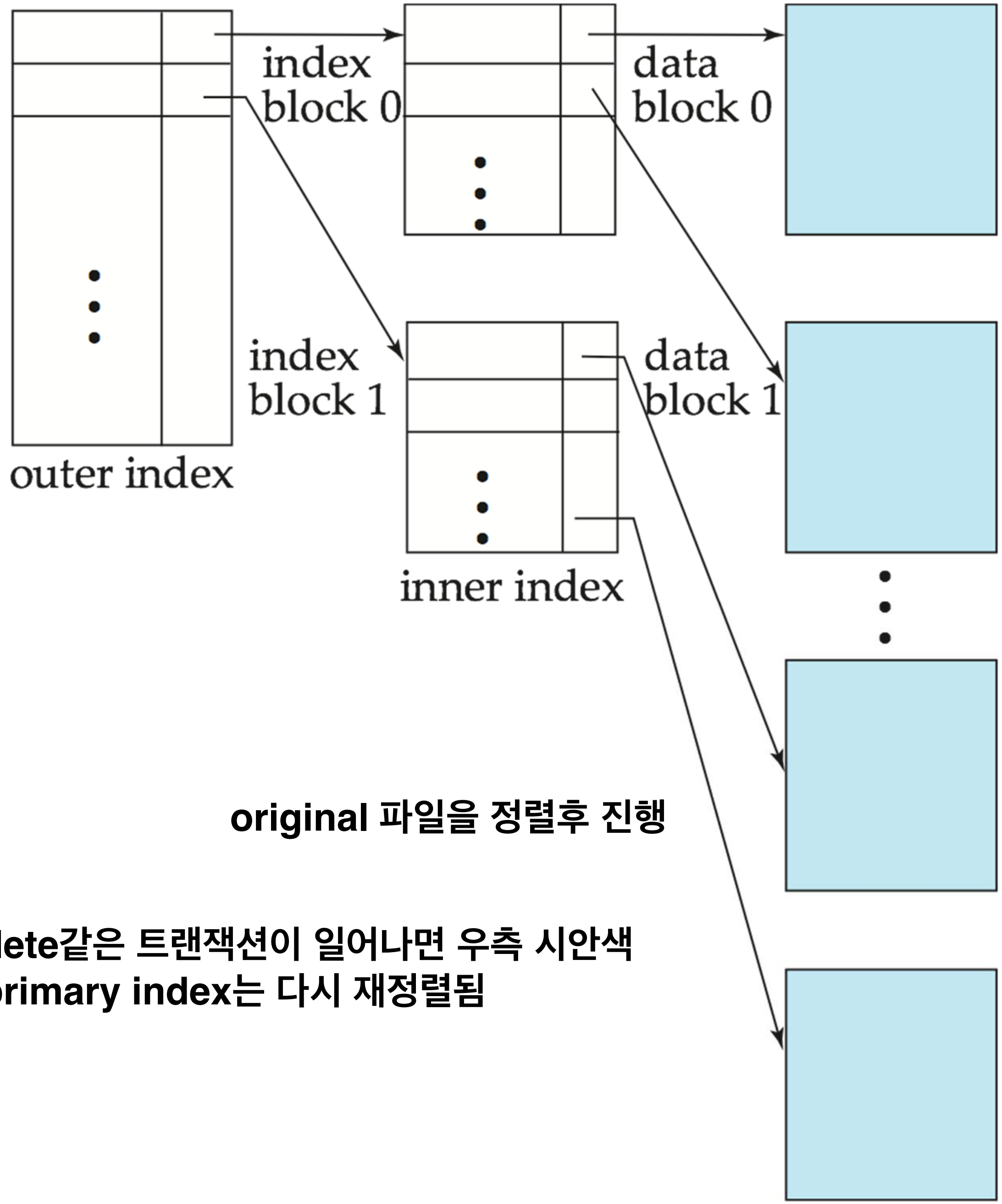
If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.

Indices at all levels must be updated on insertion or deletion from the file.

A B-tree is a particular type of tree-structured index and was introduced in 1972.
There are many variations of B-trees; we shall present $B^+$-tree introduced by Knuth.

# Multilevel Index (Cont.)




**original 파일을 정렬후 진행**

**insert,delete같은 트랜잭션이 일어나면 우측 시안색 primary index는 다시 재정렬됨**

# 11.3 B$^+$-Tree Index Files

## B+-Tree

굳이 재정렬필요없음

- B$^+$-tree is a multilevel index with a tree structure **B-tree에서 확장됨**
- When used as primary index (i.e. on sorted sequential file) maintain efficiency against insertion and deletion of records avoiding file organization
  - → Main disadvantage of index on sequential file
- Also used to index very-large relations when single-level indices don't fit in main memory
- Commerical systems (e.g. ORACLE) implemented index with B-Tree
- In the following we will present the structure of so called B$^+$-tree

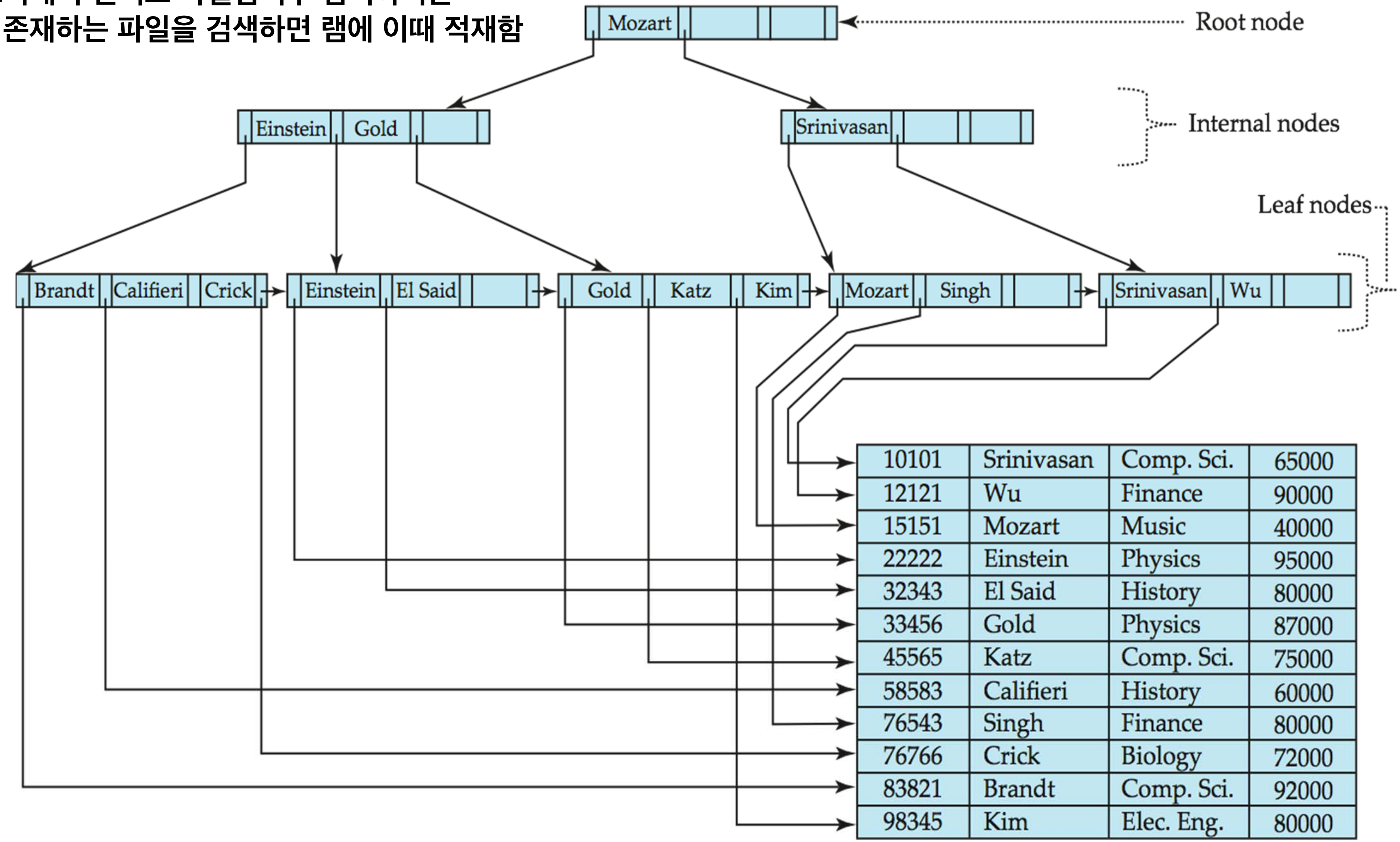
(B stands for **Balanced** **Tree**)

# 11.3 B⁺-Tree Index Files

내부검색을 왼쪽순으로할지 오른쪽 순으로 볼지 결정하고 탐색

## Example of B+-Tree (Instrcutor relation)

b트리에서 인덱스 파일검색후 검색하려는
인덱스가 존재하는 파일을 검색하면 램에 이때 적재함




| | | | |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 80000 |
| 33456 | Gold | Physics | 87000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 58583 | Califieri | History | 60000 |
| 76543 | Singh | Finance | 80000 |
| 76766 | Crick | Biology | 72000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |

# 11.3 B+-Tree Index Files

## B+-Tree node structure

| | | | | | | |
|---|---|---|---|---|---|---|
| $P_1$ | $K_1$ | $P_2$ | … | $P_{n-1}$ | $K_{n-1}$ | $P_n$ |

$K_i$ are the search-key values

$P_i$ are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).

The search-keys in a node are ordered

$$K_1 < K_2 < K_3 < \ldots < K_{n-1}$$

리프노드가 아닐때 ,
포인터부분에 자식을 갖는다

한노드에서 n(p) = n 일때, n(k)=n-1개를 갖는다

만약 리프노드이면 해당 리프노드의 끝 포인터는 다른 인접한 리프노드의 앞을 가리킨다

# B+-Tree Index Files (Cont.)

**A B+-tree is a rooted tree satisfying the following properties:**

밸런스있게 같은 레벨의 노드들을 가짐

- All paths from root to leaf are of the same length

즉, 값을 적어도 2개 가져야함

- Internal nodes (Each node that is not a root or a leaf) has between $\lceil n/2 \rceil$ and $n$ children.

이때 value는 b+의 search key(content)값

- A leaf node has between $\lceil (n-1)/2 \rceil$ and $n-1$ values
- Special cases:
  - If the root is not a leaf, it has at least 2 children.
  - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and $(n-1)$ values.

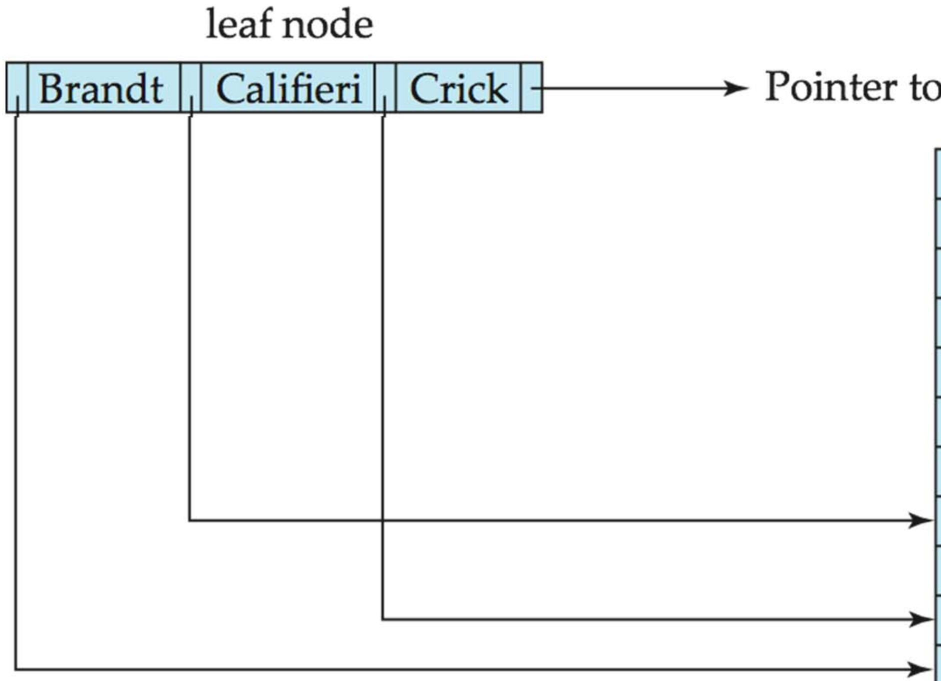
# Leaf Nodes in B+-Trees

Properties of a leaf node:

- For $i = 1, 2, . . ., n–1$, pointer $P_i$ points to a file record with search-key value $K_i$,

즉, Li는 상대적으로 Lj보다 왼쪽에 위치하고 값의 크기가 상대적으로 적음

- If $L_i$, $L_j$ are leaf nodes and $i < j$, $L_i$'s search-key values are less than or equal to $L_j$'s search-key values
- $P_n$ points to next leaf node in search-key order

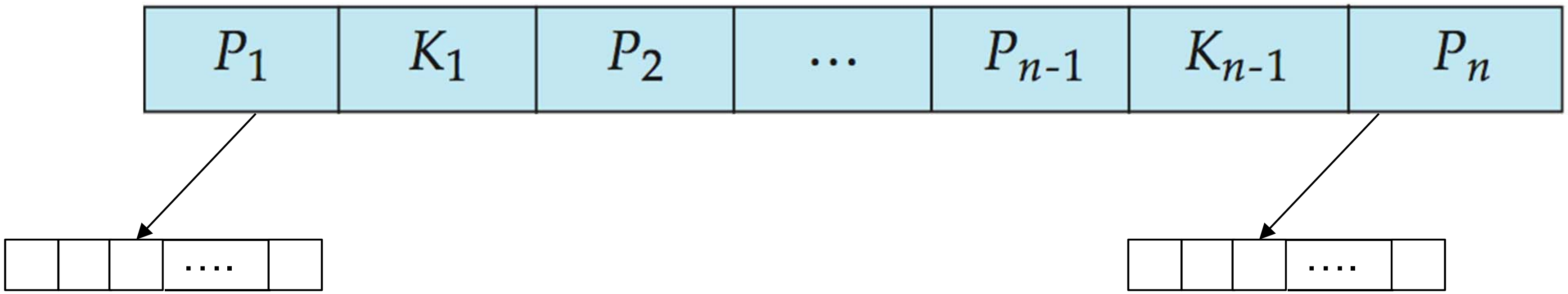



| | | | |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 80000 |
| 33456 | Gold | Physics | 87000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 58583 | Califieri | History | 60000 |
| 76543 | Singh | Finance | 80000 |
| 76766 | Crick | Biology | 72000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |

# Non-Leaf Nodes in B+-Trees

For a non-leaf node with ***n*** pointers:

- All the search-keys in the subtree to which $P_1$ points are less than $K_1$
- For $2 \leq i \leq n - 1$, all the search-keys in the subtree to which $P_i$ points have values greater than or equal to $K_{i-1}$ and less than $K_i$
- All the search-keys in the subtree to which $P_n$ points have values greater than or equal to $K_{n-1}$

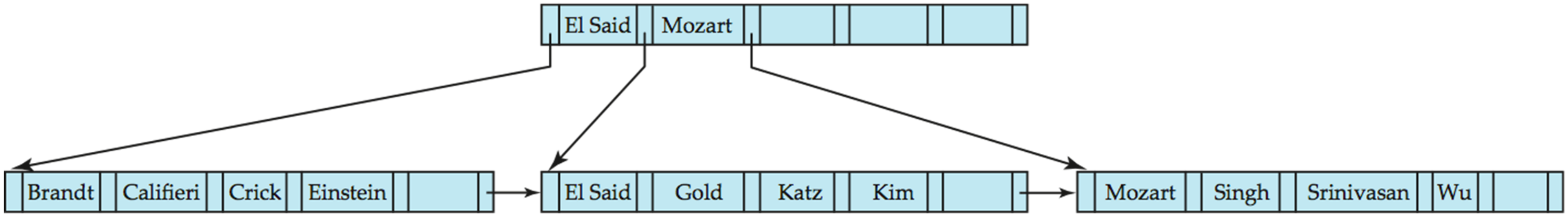



All key values in this subtree ≤ $K_1$

All key values in this subtree ≥ $K_{n-1}$

# Example of B+-tree

특정값을 주고 B+트리를 만들라는 문제 나옴




B+-tree for *instructor* file ($n$ = 6)

- Leaf nodes must have between 3 and 5 values ($\lceil (n-1)/2 \rceil$ and $n-1$, with $n = 6$).
- Non-leaf nodes other than root must have between 3 and 6 children ($\lceil n/2 \rceil$ and $n$ with $n = 6$).
- Root must have at least 2 children.

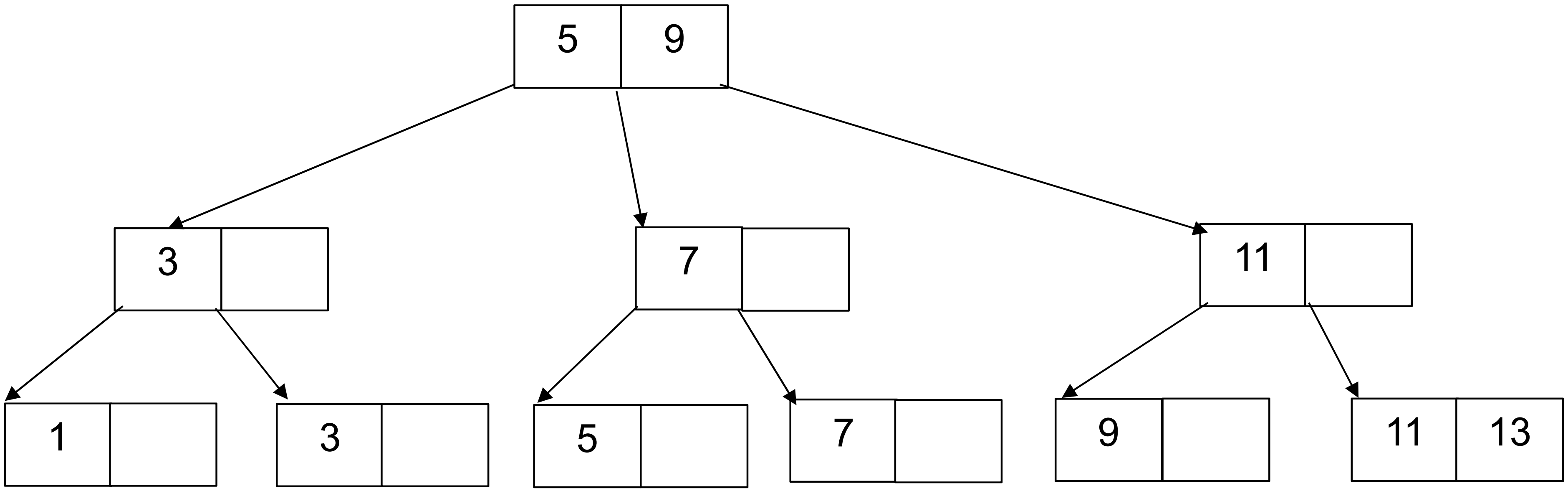
# Building $B^+$-tree

**B트리 어떻게 만들고 바뀔건지 알고 그리는 문제 나올수 있음**

Buidling B+ Tree

Construct a $B^+$- tree that contains the following entries: 1,3,5,7,9,11 and 13. (Assume that the tree is initially empty, and insert the records in the above order.)

Pointer의 개수 3 (degree)

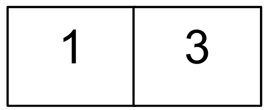


# Building B$^{+}$-tree

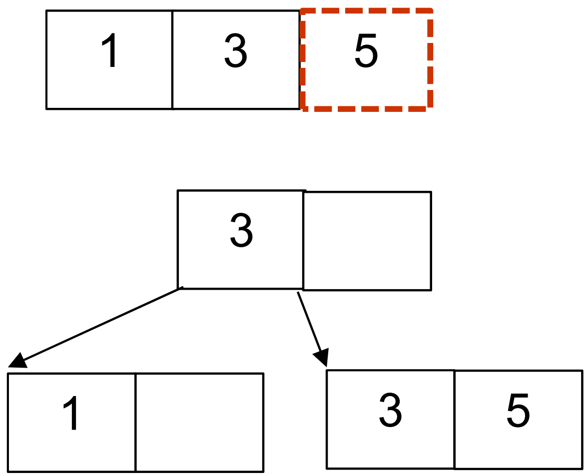
Buidling B+ Tree

1,3,5,7,9,11 and 13.  Pointer의 개수 3 (degree)

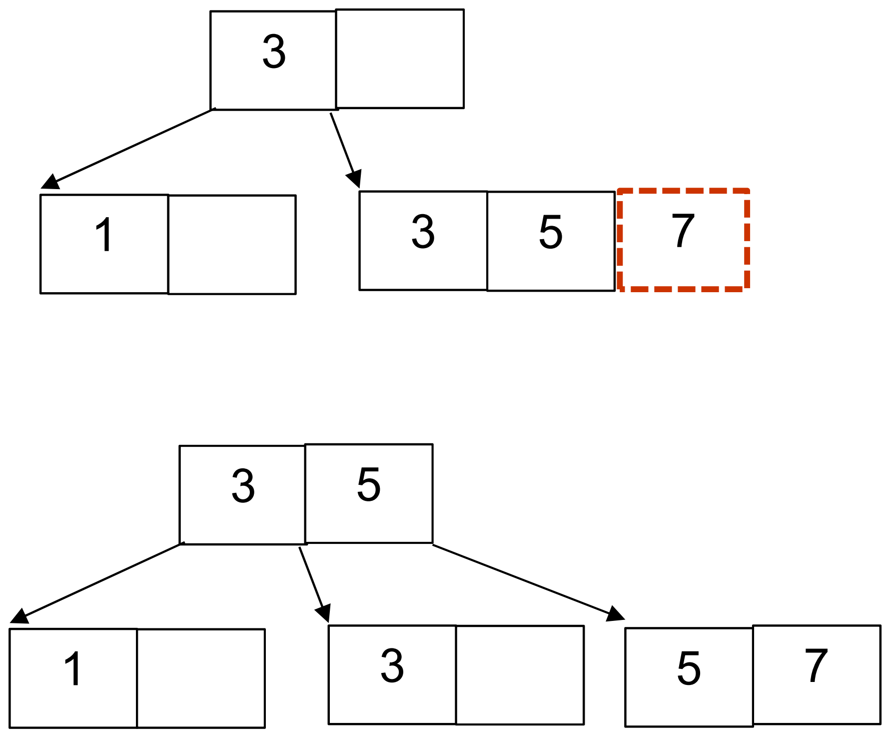
Insert 1 and 3




Insert 5 (overflow - Split)




Insert 7 (overflow - Split)



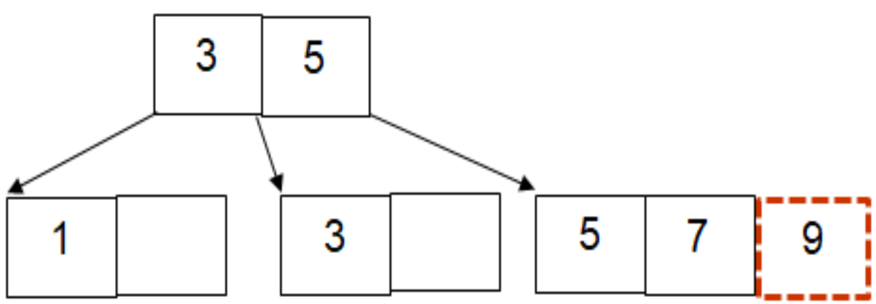

N개의 검색키 값을 가질 때
Leaf node에서 Split이 발생될 때는 [n/2]개는 원래존재하는 노드에 두고
나머지는 새로운 노드에 놓는다.

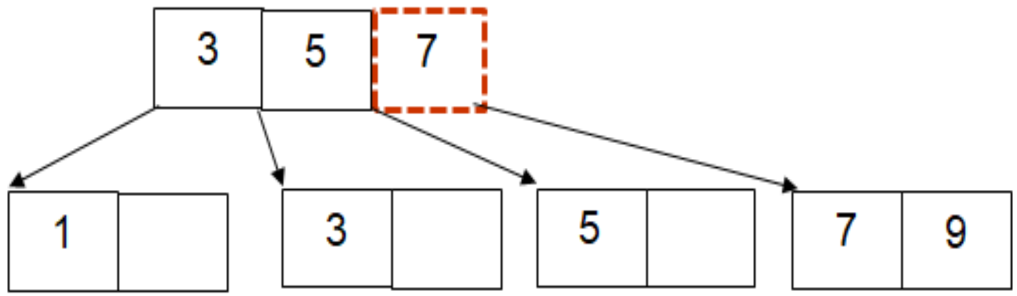
# Building B+-tree

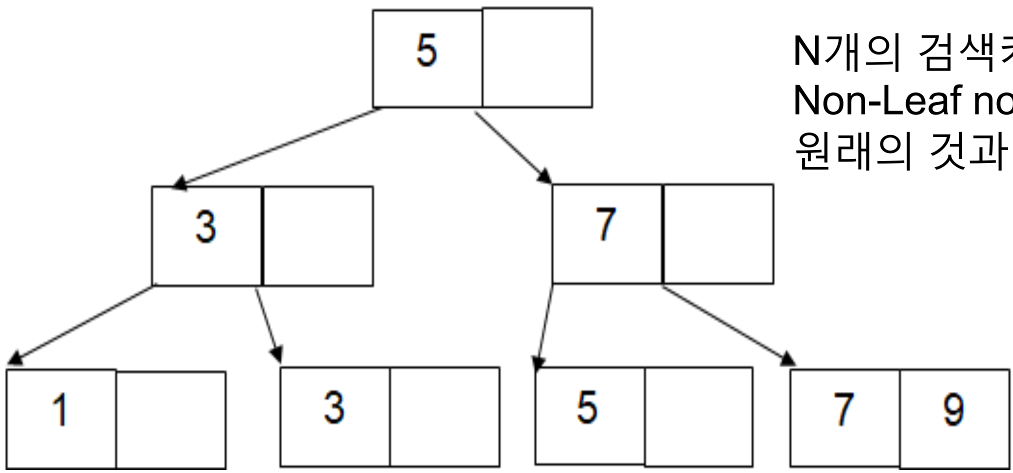
Buidling B+ Tree

1,3,5,7,9,11 and 13. Pointer의 개수 3 (degree)

Insert 9 (overflow - Split)



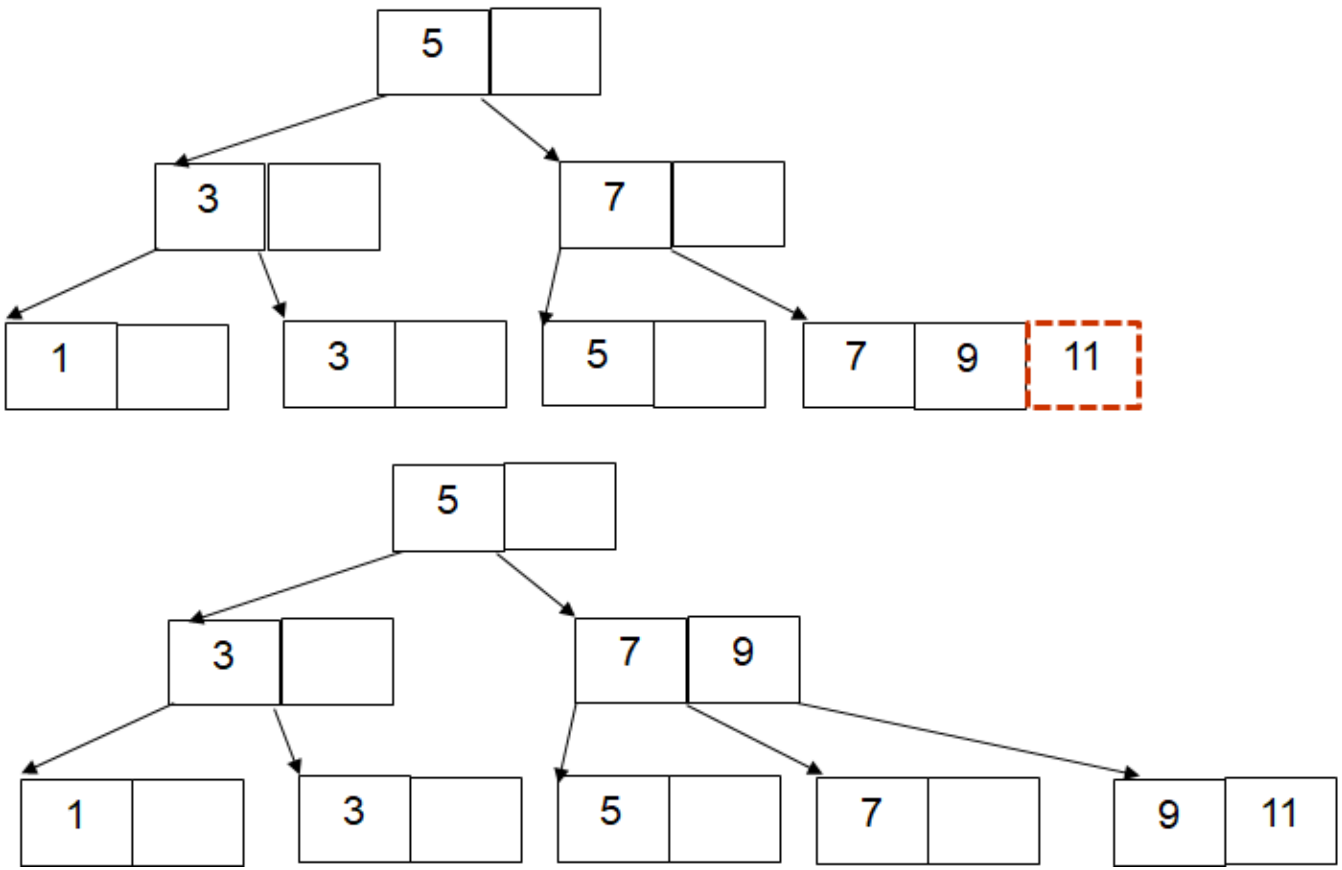

N개의 검색키 값을 가질 때
Non-Leaf node에서 Split이 발생될 때는
원래의 것과 새롭게 생성된 노드로 나누어진다.

# Building B⁺-tree

Buidling B+ Tree

1,3,5,7,9,11 and 13.    Pointer의 개수 3 (degree)

Insert 11 (overflow - Split)

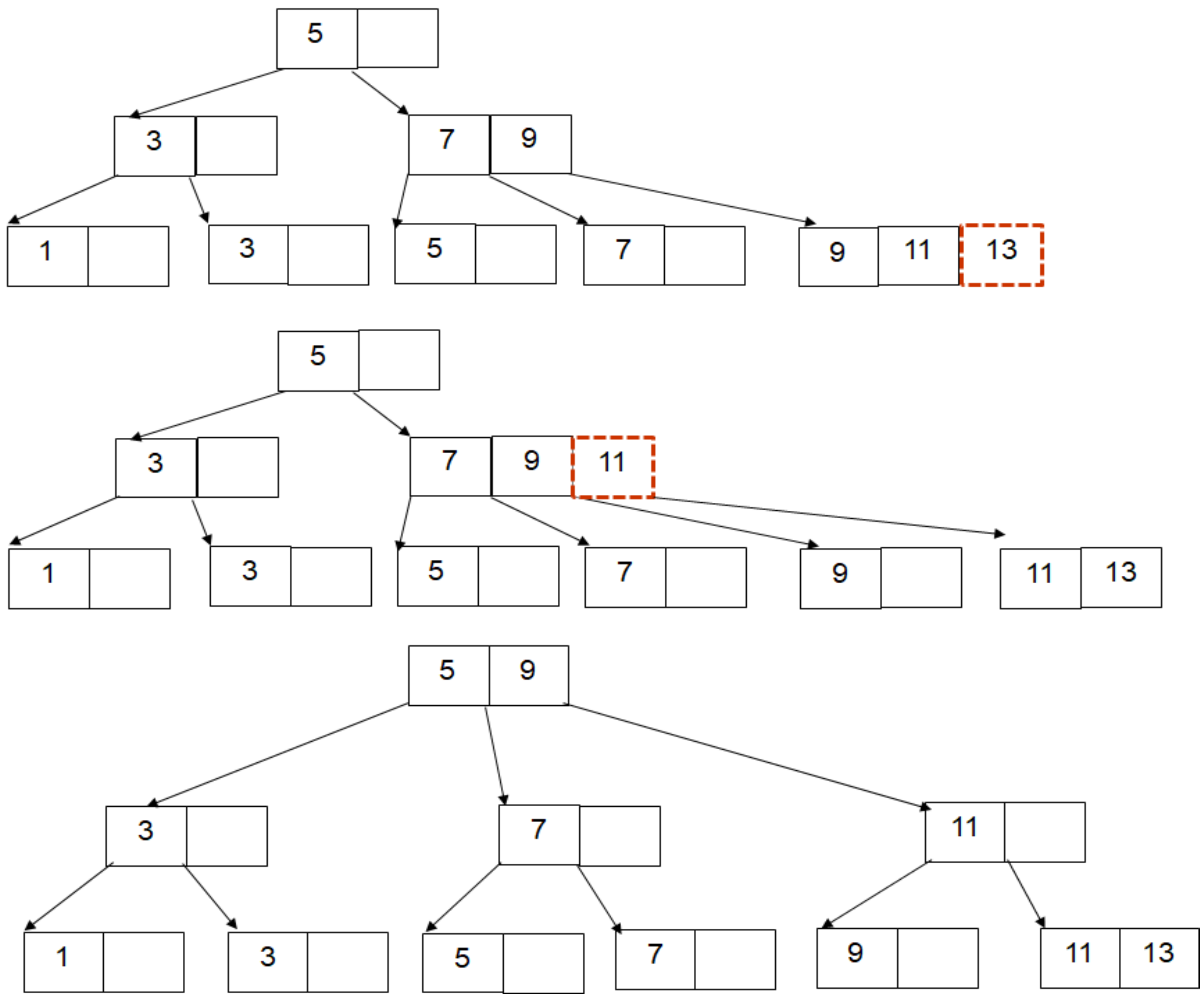


# Building B$^+$-tree

Buidling B+ Tree

1,3,5,7,9,11 and 13. Pointer의 개수 3 (degree)

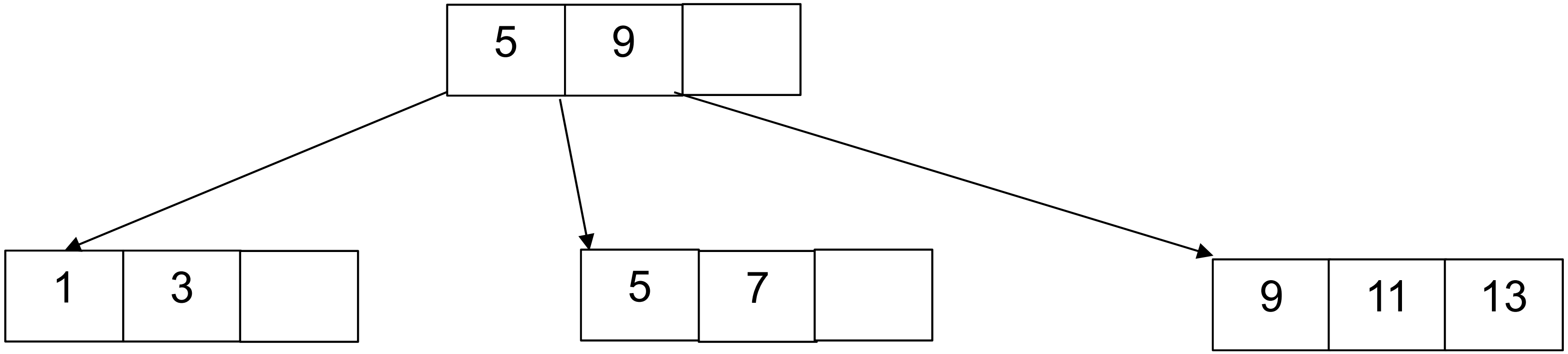
Insert 13 (overflow - Split)



# Building B$^{+}$-tree

Buidling B+ Tree (Different Pointer)

Construct a B$^{+}$- tree that contains the following entries: 1,3,5,7,9,11 and 13. Pointer의 개수 4 (degree)



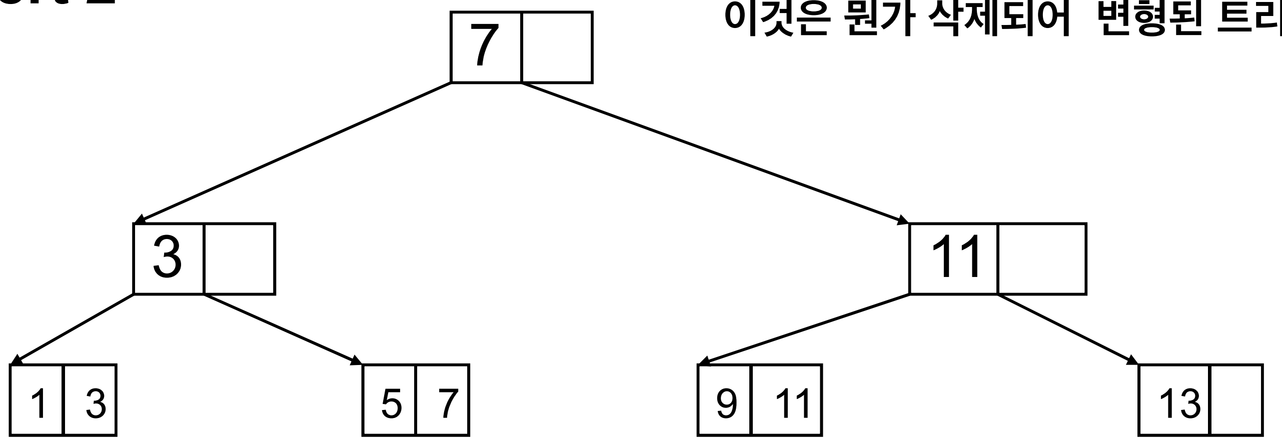
# Insert B+-tree

Insert B+ Tree

- Insert at bottom level (leaf node)
- If leaf node overflows, split node and copy middle element to next index node (Internal node or root node)
- If index node overflows, split node and move middle element to next index node (Internal node or root node)

**Insert 2**

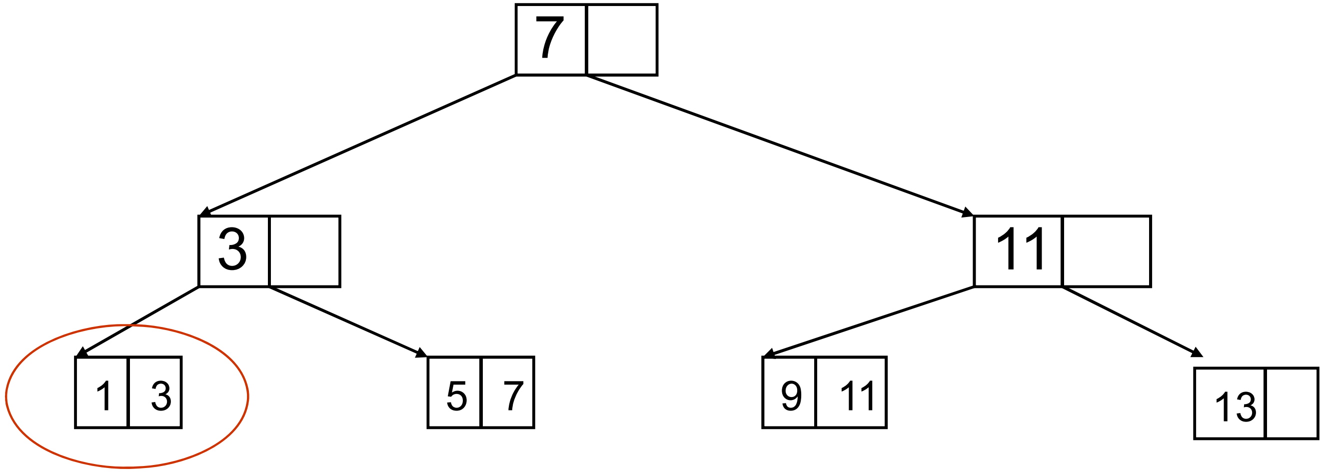
이것은 뭔가 삭제되어  변형된 트리

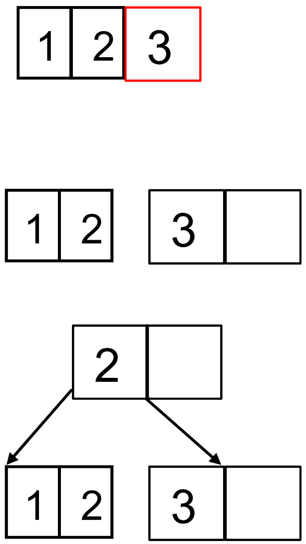


# Insert B+-tree

Buidling B+ Tree

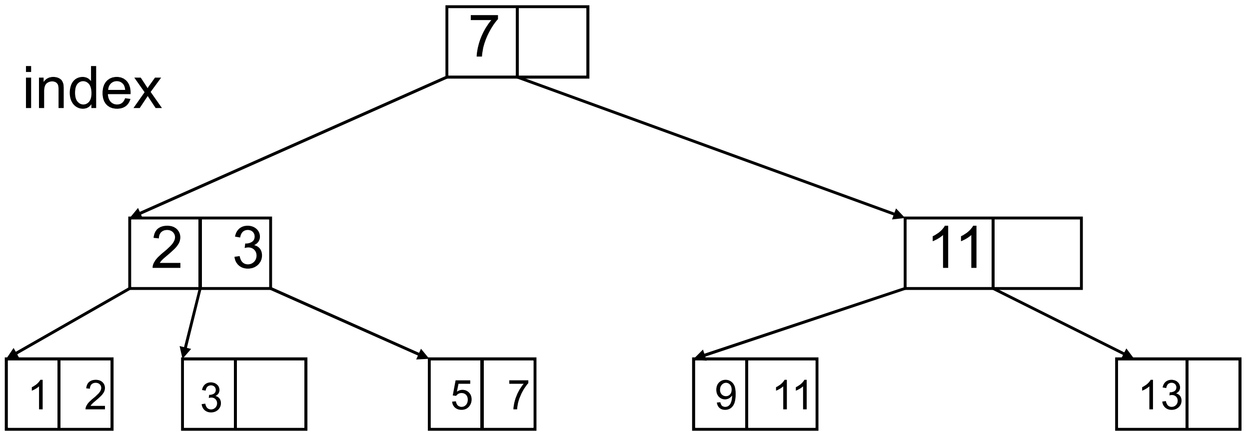
**Insert 2**




Split node

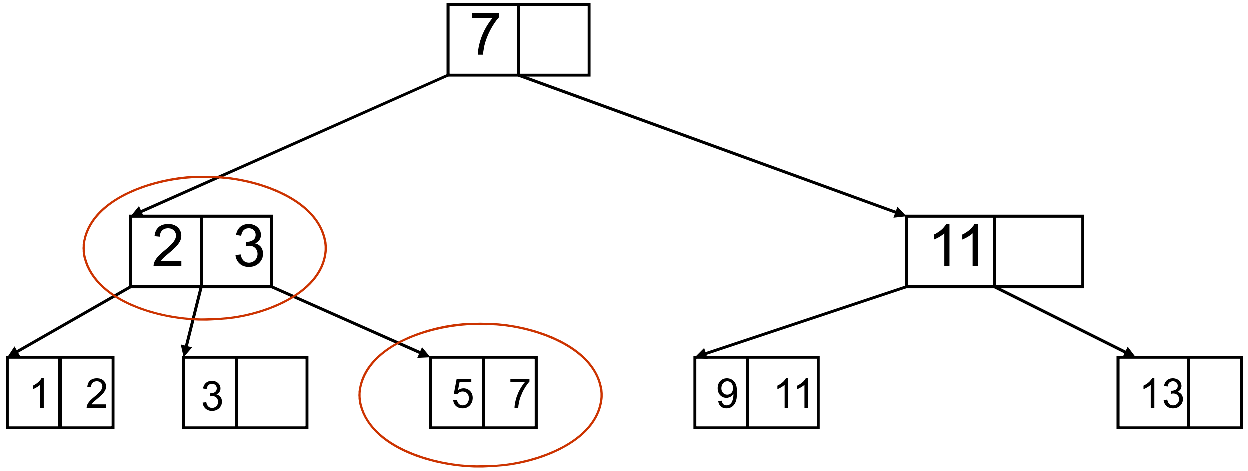
Copy 2 to index node



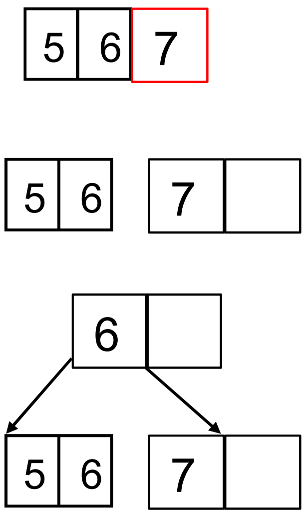
# Insert B$^+$-tree

Buidling B+ Tree

**Insert 6**



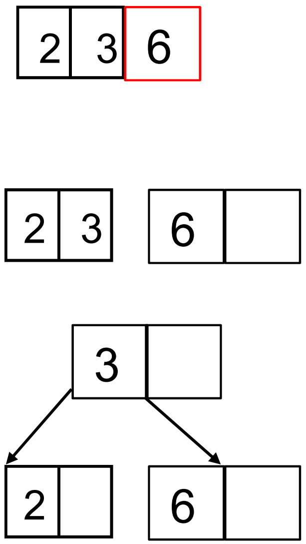

Leaf node




Split node

Copy 6 to index node

Non-leaf node




Split node

Copy 3 to index node

# Insert B$^+$-tree

Buidling B+ Tree

**Insert 6**