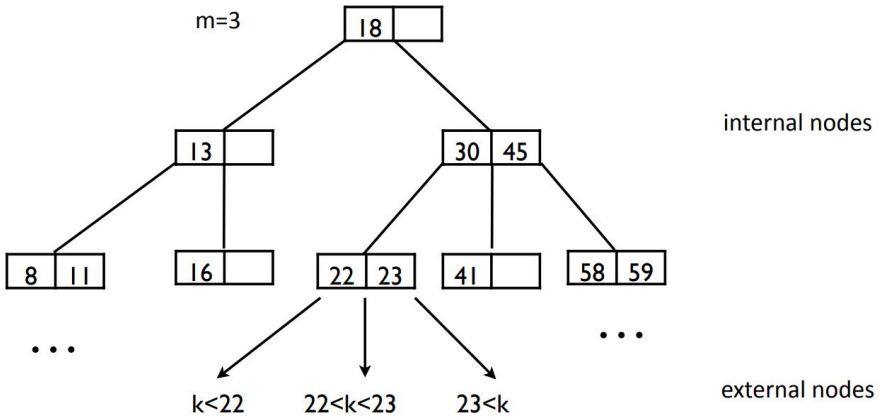
Lab 09: B-Tree

Data Structure

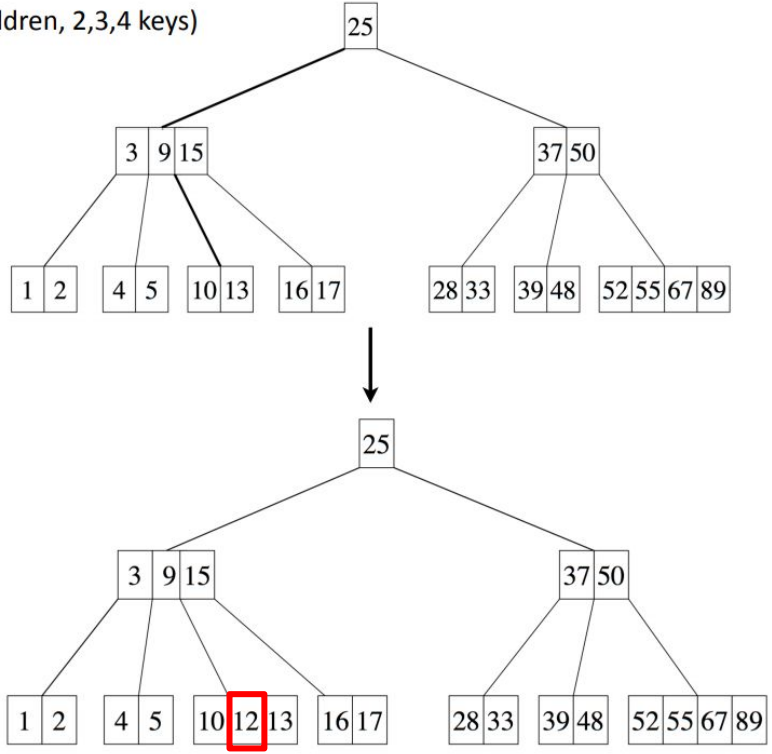
B-Tree

- A B-tree of **order m** is an m-way search tree with the following properties
  - the root has **at least 2** children
  - each node has **upto m-1** keys
  - all external nodes are at the same level (perfectly balanced)
  - all internal nodes (except the root) have **between ⌈m/2⌉and m** children
    - when m=3, all internal nodes of B-tree have a degree of either 2 or 3 (2-3 tree)
    - when m=4, all internal nodes



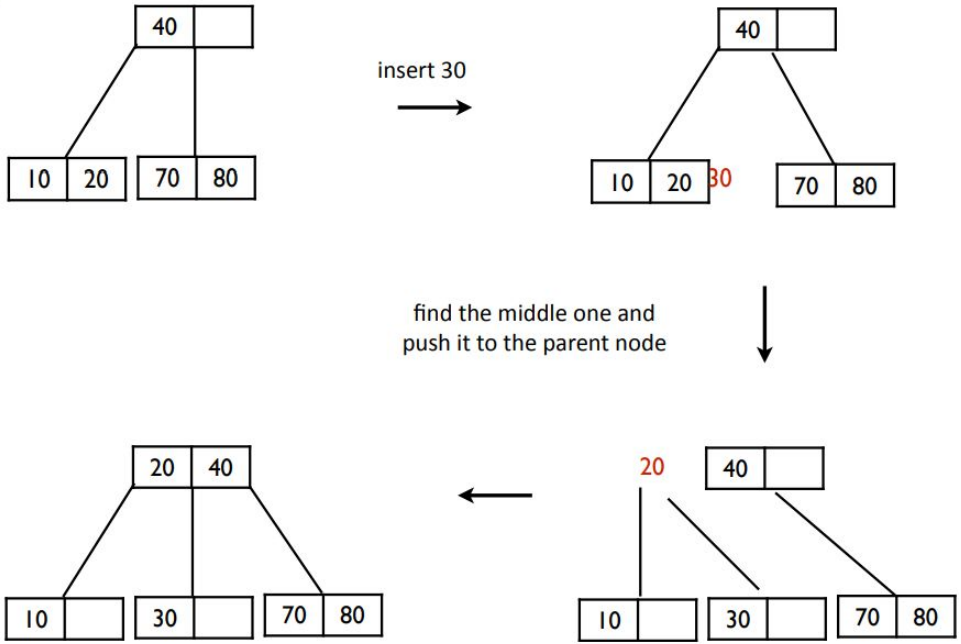
B-Tree - Insertion

m= 5 (3,4,5 children, 2,3,4 keys)
insert 12

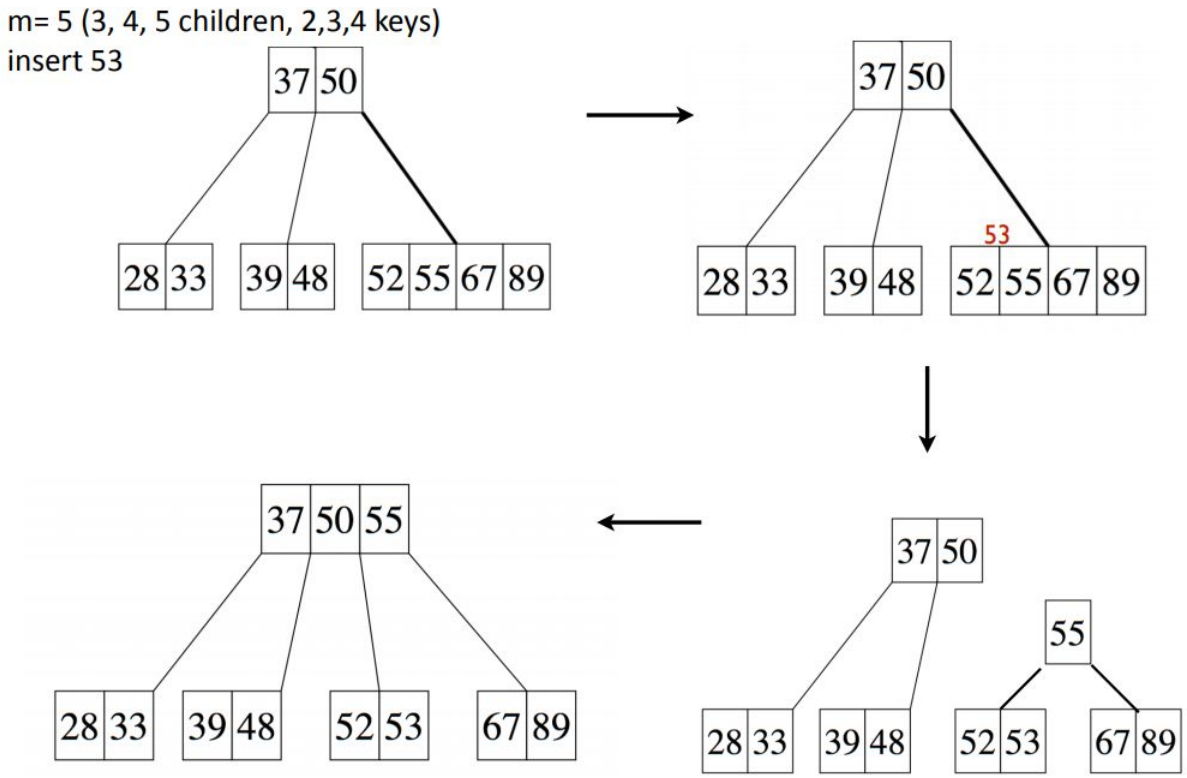


B-Tree – Node Split

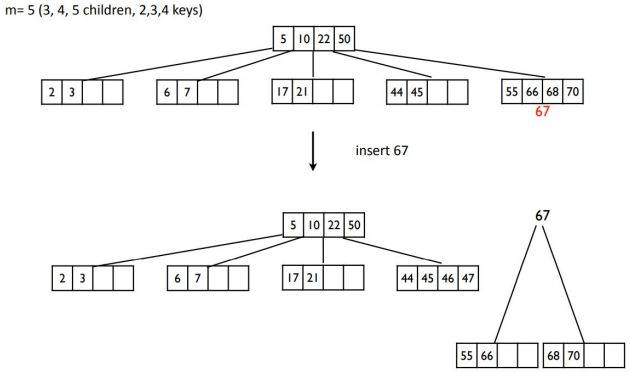
m= 3 (2,3 children, 1,2 keys)
insert 30

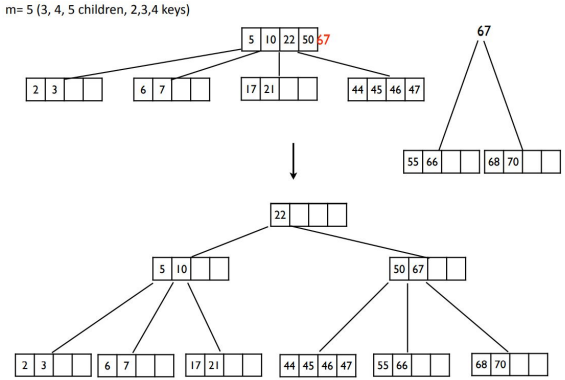


B-Tree – Node Split



B-Tree – Node Split





B-Tree ADT

- **BNodePtr CreateTree(int order)**
  - Create a B-Tree with the given order.
- **void Insert(BNodePtr root, int key)**
  - Insert a new key to the b-tree. You should find the right position for the new key to maintain the B-Tree. Print what key you inserted. You can split the node when node is full.
- **int Find(BNodePtr root, int key)**
  - Find the key in the B-Tree. Return 1 if the value exists. Otherwise, return 0.
- **void PrintTree(BNodePtr root)**
  - Print the entire tree by inorder traversal.
- **void DeleteTree(BNodePtr root)**
  - Delete the entire tree.

B-Tree ADT

- **i x**
  - Insert a new key “x” into the B-Tree. **Print what key you inserted.**
- **f x**
  - Find the given key to check whether the key exists in the B-Tree, and **print whether the key exists or not.**
- **p**
  - **Print the entire B-Tree in inorder traversal.**

B-Tree ADT

Structure

```
typedef struct BNode* BNodePtr;
struct BNode{
    int order;
    int size        /* number of children */
    BNodePtr *child; /* children pointers */
    int *key;       /* keys */
    int is_leaf;
}BNode;
```

Function

```
BNodePtr CreateTree(int order);
void Insert(BNodePtr root, int key);
int Find(BNodePtr root, int key);
void PrintTree(BNodePtr root);
void DeleteTree(BNodePtr root);
```

B-Tree ADT – Skeleton Code

```c
#include<stdio.h>
#include<stdlib.h>

FILE *fin;
FILE *fout;

typedef struct BNode* BNodePtr;

struct BNode{
    int order;
    int size;           /* number of children */
    BNodePtr *child;    /* children pointers */
    int *key;           /* keys */
    int is_leaf;
}BNode;

BNodePtr CreateTree(int order);
void Insert(BNodePtr *root, int key);
int Find(BNodePtr root, int key);
void PrintTree(BNodePtr root);
void DeleteTree(BNodePtr root);

int main(int argc, char* argv[]){
    fin = fopen(argv[1], "r");
    fout = fopen(argv[2], "w");

    int order;
    fscanf(fin, "%d", &order);
    BNodePtr root = CreateTree(order);

    char cv;
    int key;
    while(!feof(fin)){
        fscanf(fin, "%c", &cv);
        switch(cv){
            case 'i':
                fscanf(fin, "%d", &key);
                if(Find(root, key))
                    fprintf(fout, "insert error: key %d is already in the tree!\n", key);
                else
                    Insert(&root, key);
                break;
            case 'f':
                fscanf(fin, "%d", &key);
                if(Find(root, key))
                    fprintf(fout, "key %d found\n", key);
                else
                    fprintf(fout, "finding error: key %d is not in the tree!\n", key);
                break;
            case 'p':
                if (root->size == 1)
                    fprintf(fout, "print error: tree is empty!");
                else
                    PrintTree(root);
                fprintf(fout, "\n");
                break;
        }
    }

    DeleteTree(root);
    fclose(fin);
    fclose(fout);

    return 0;
}
```

Input & Output Example

input1.txt

```
3
i 10
i 40
i 70
i 4
p
i 92
i 54
i 32
f 5
f 32
p
i 29
f 10
p
```

output1....

```
4 10 40 70
finding error : key 5 is not in the tree!
key 32 found
4 10 32 40 54 70 92
key 10 found
4 10 29 32 40 54 70 92
```

Assignment

- Due
  - **~ 2023.05.17(수) 23:59**
  - Last Commit 기준

- 자세한 내용은 과제 명세 PDF 파일 참고