Chapter 10

Multiway Trees

Objectives

Upon completion you will be able to:

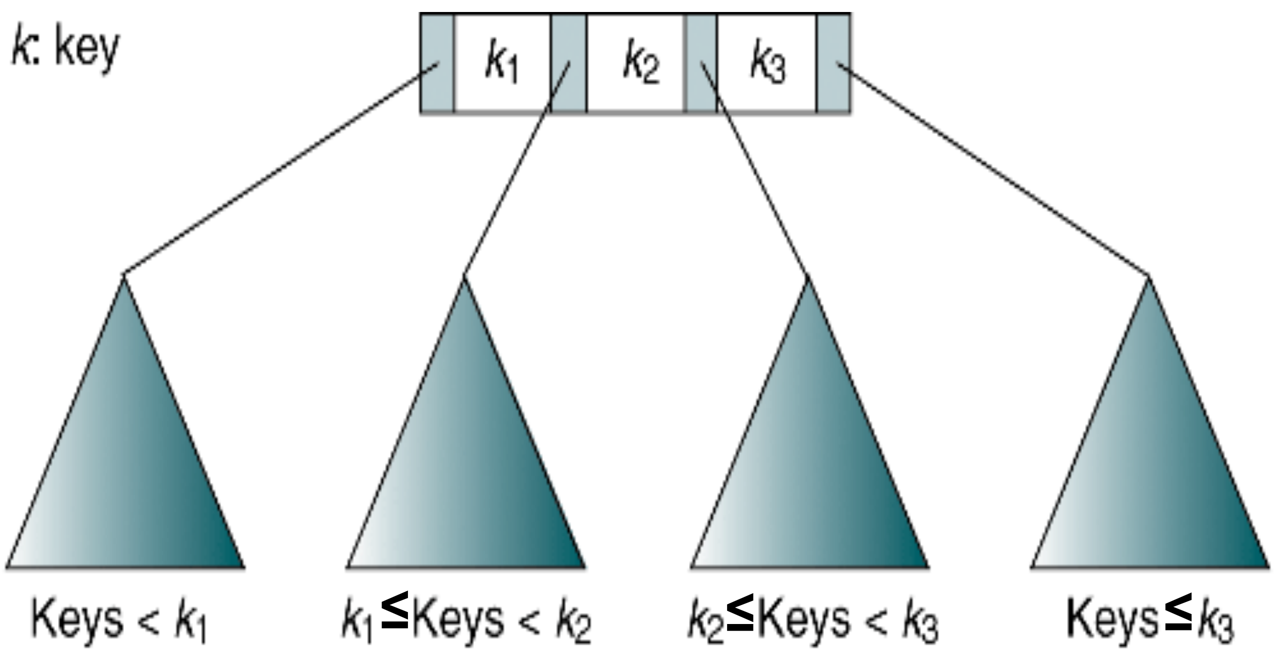
- Define and discuss multiway tree structures
- Understand the operation and use of the B-tree ADT
- Discuss the use and implementation of simplified B-trees
- Compare and contrast B-trees, B*trees, and T+trees
- Discuss the design and use a lexical search trees (Tries)

10-1 M-way Search Tree

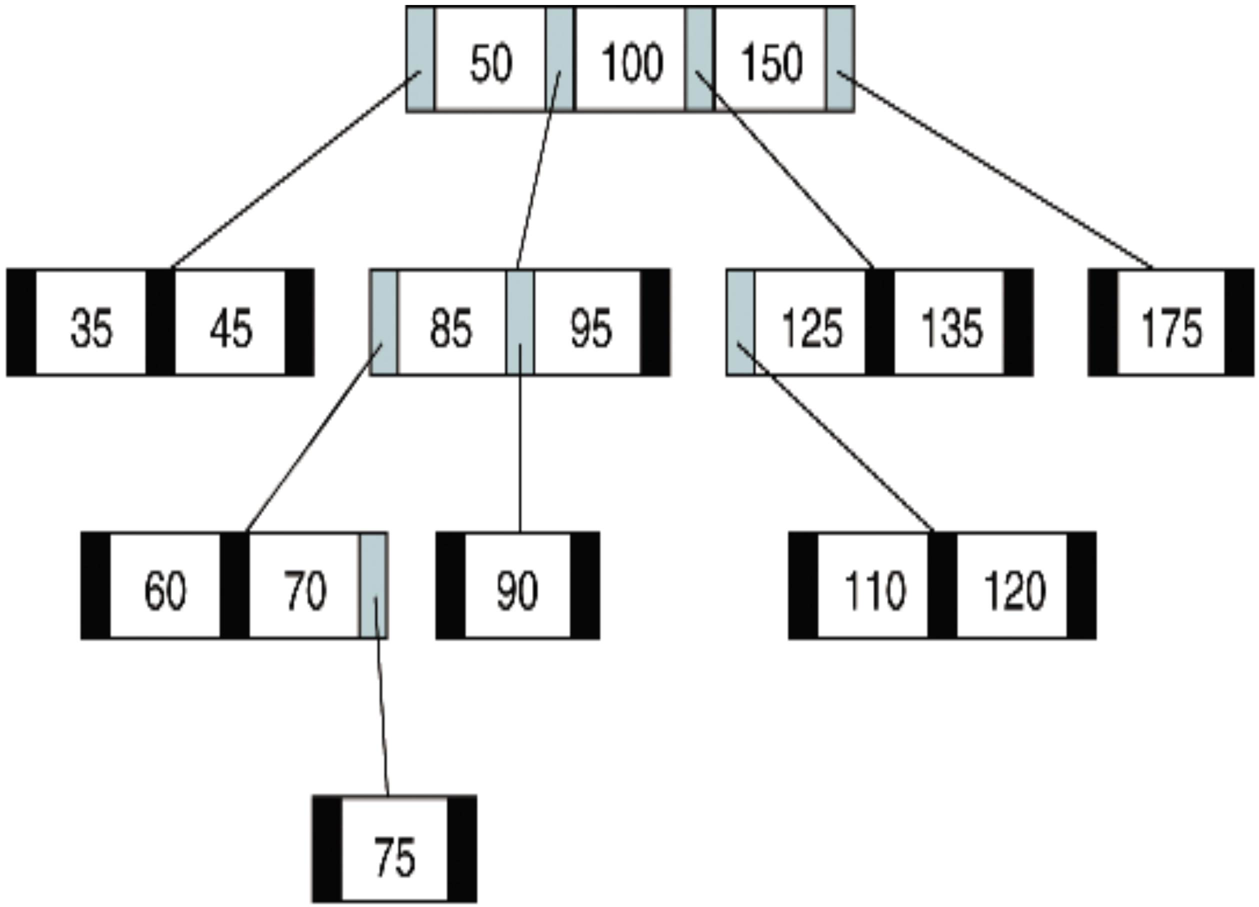
This section explore trees whose outdegree is not restricted to 2 but that retain the general properties of binary search trees; Multiway trees have multiway entries in each node and thus may have multiple subtrees.

- **M-way tree**
- **B-trees**
- **B*trees**
- **T+trees**

M-way Tree

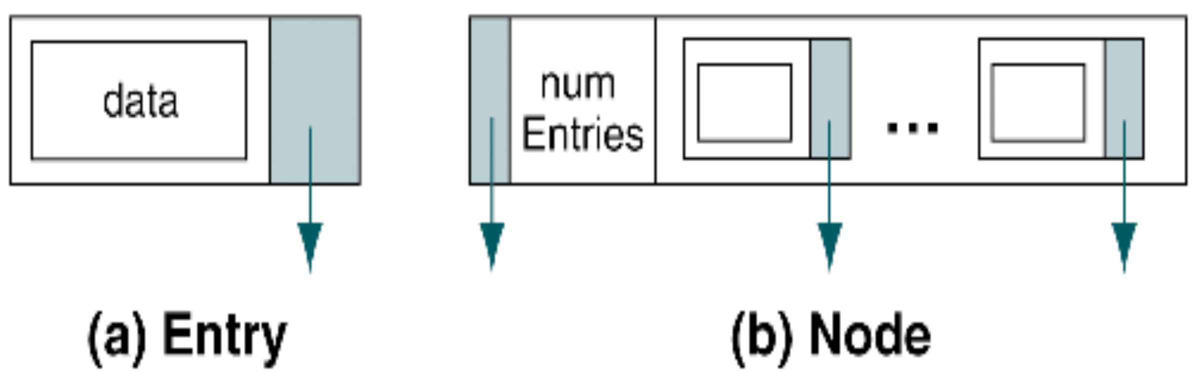


4-way Tree



M-way Node Structure

```
entry
    data
    rightPtr
end entry
node
    firstPtr
    numEntries
    entries   array of entry
end node
```



10-2 B-trees

This section discusses the B-tree structure which is a balanced version of m-way tree.

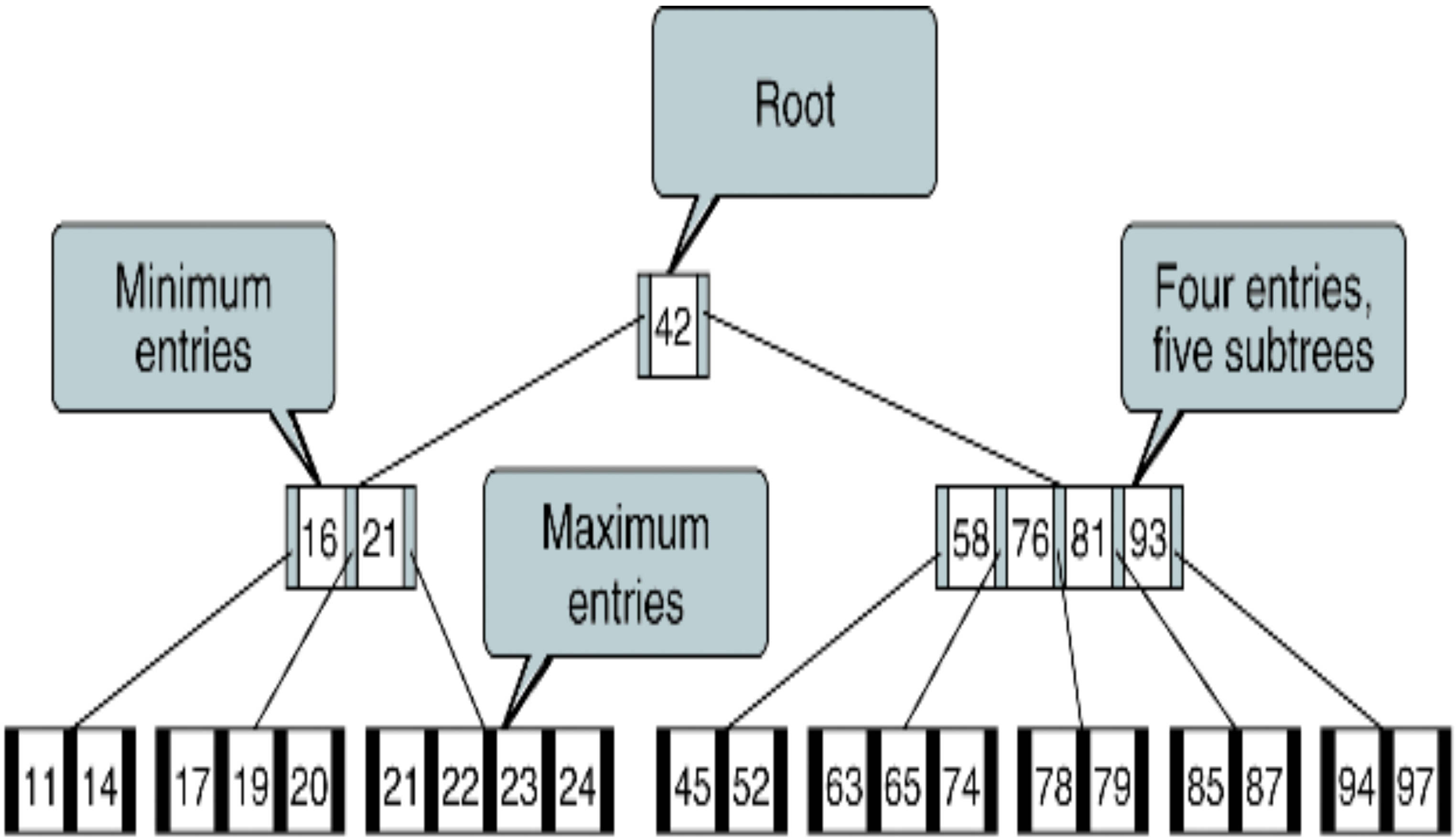
- **Insertion**
- **Deletion**
- **Traversal**
- **Search**

Entries in B-trees of Various Orders

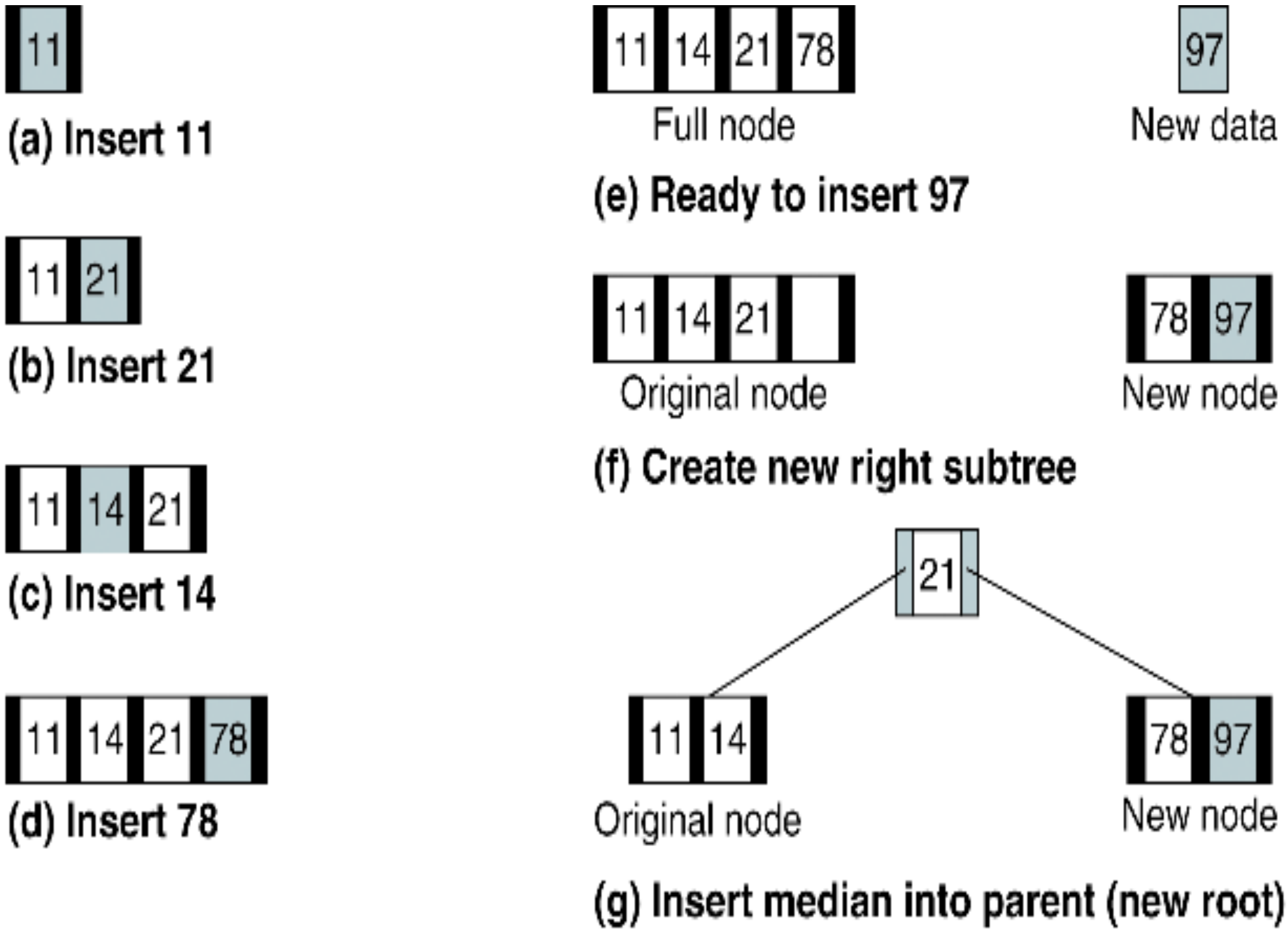
- **Root: a leaf or it has 2…m subtrees**
- **Internal node:** $\lceil m/2 \rceil$ **… m subtrees (** $\lceil m/2 \rceil$ **-1 … m-1 entries)**
- **Leaf node:** $\lceil m/2 \rceil$ **-1 … m-1 entries**
- **All leaf nodes are at the same level (perfectly balanced)**

Order	Number of subtrees		Number of entries	
	Minimum	Maximum	Minimum	Maximum
3	2	3	1	2
4	2	4	1	3
5	3	5	2	4
6	3	6	2	5
…	…	…	…	…
m	$\lceil m/2 \rceil$	m	$\lceil m/2 \rceil - 1$	$m-1$

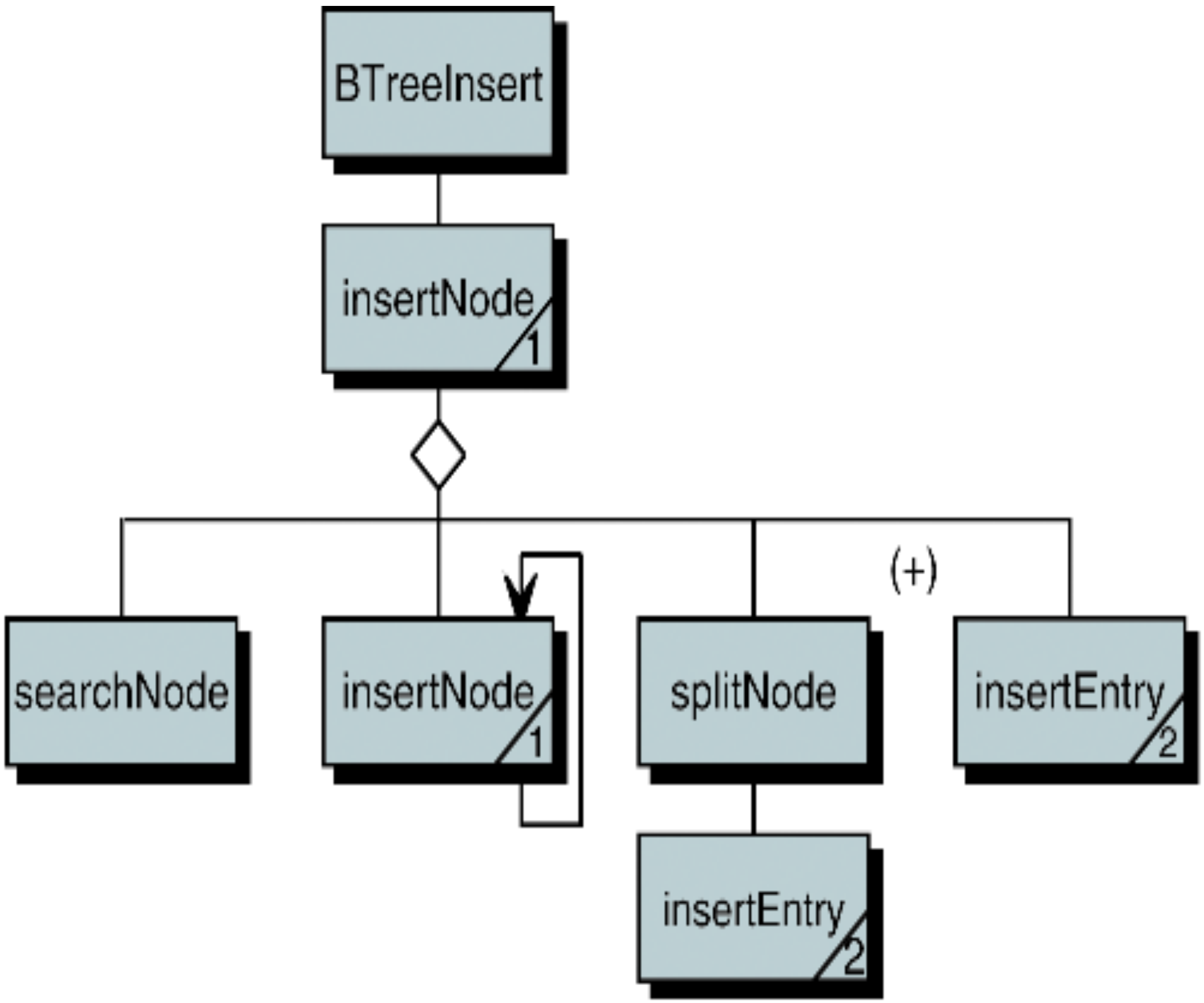
A B-tree of Order 5



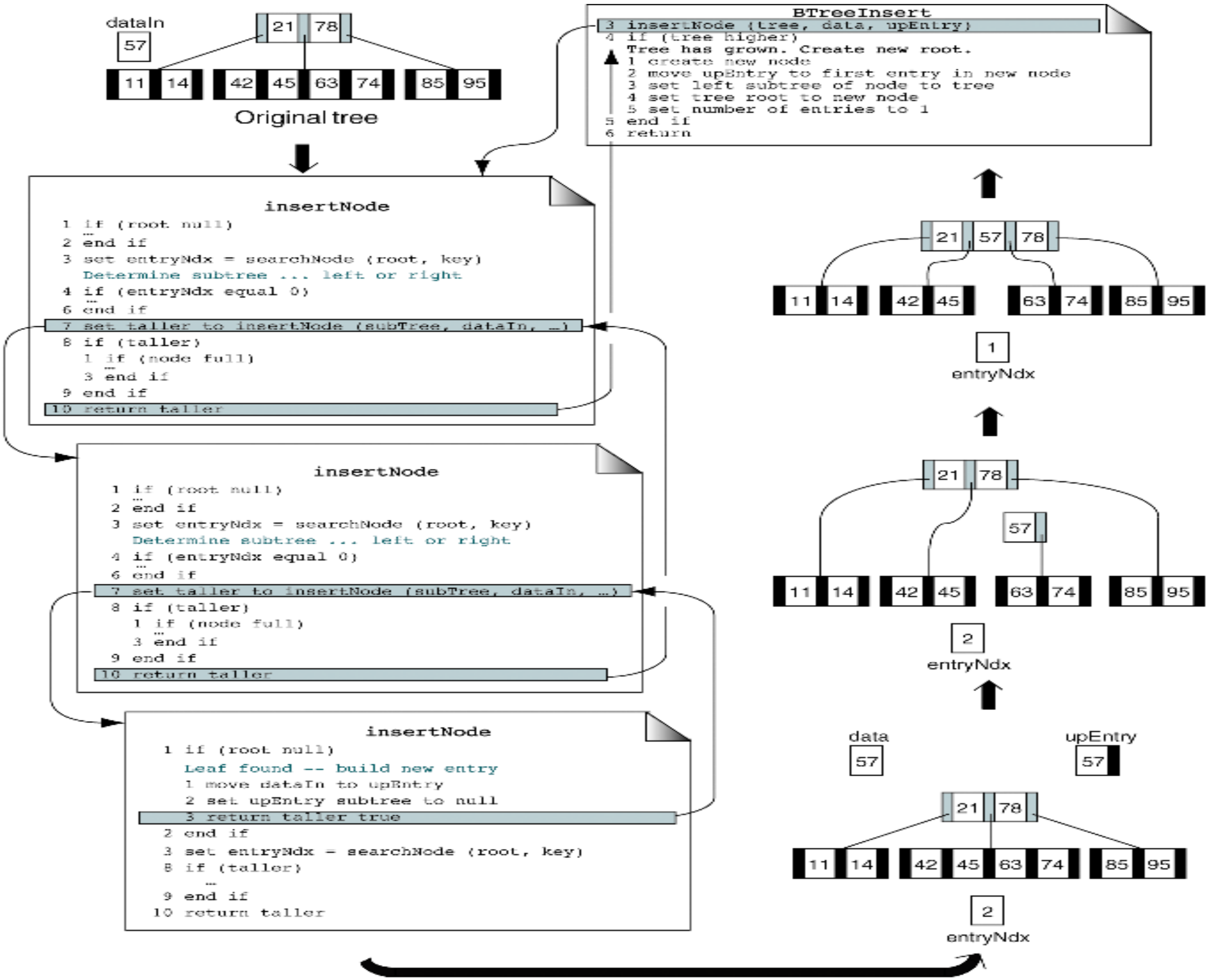
B-tree Insert Overview



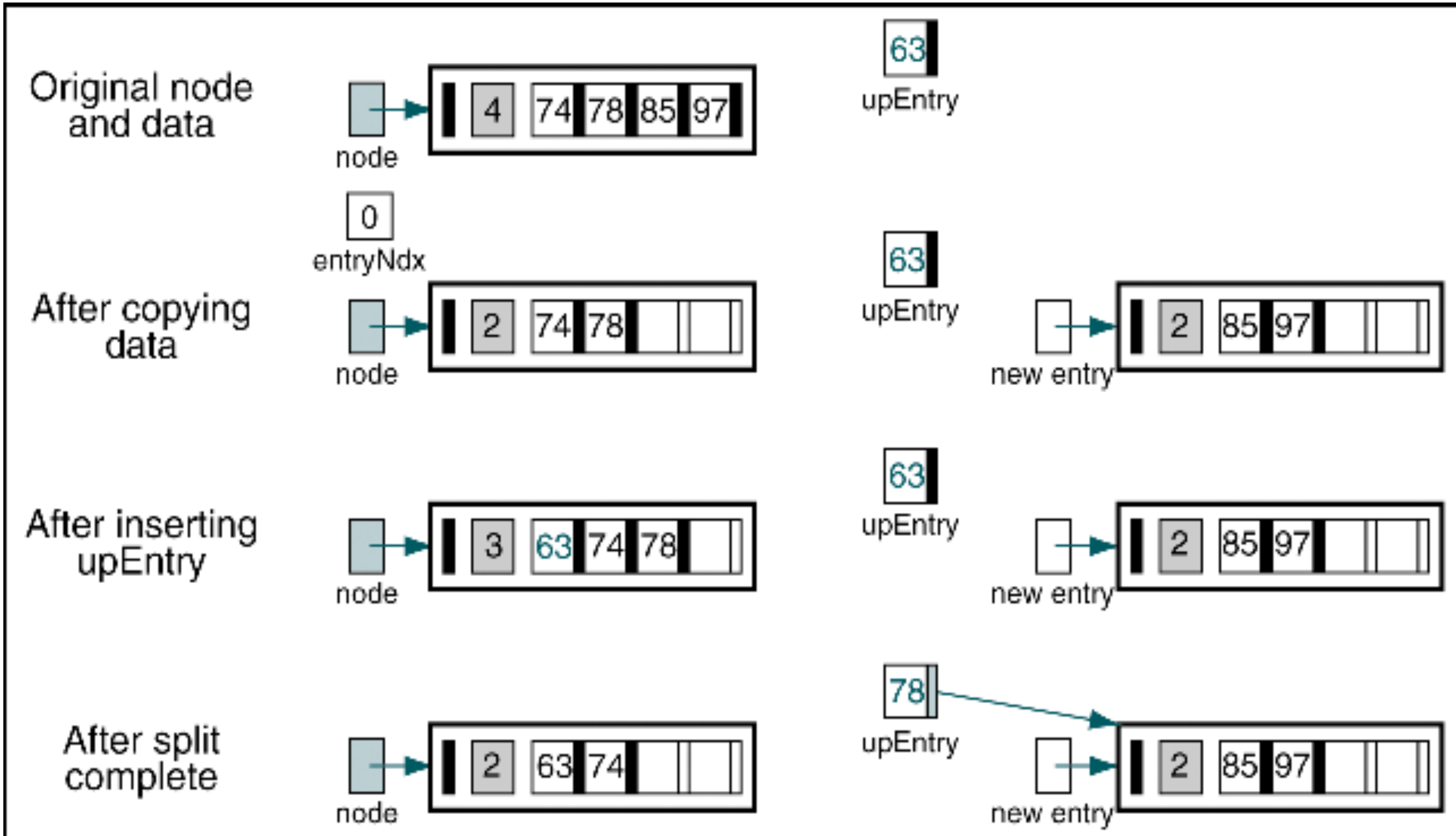
B-tree Insert Design



Build B-tree with Overflow

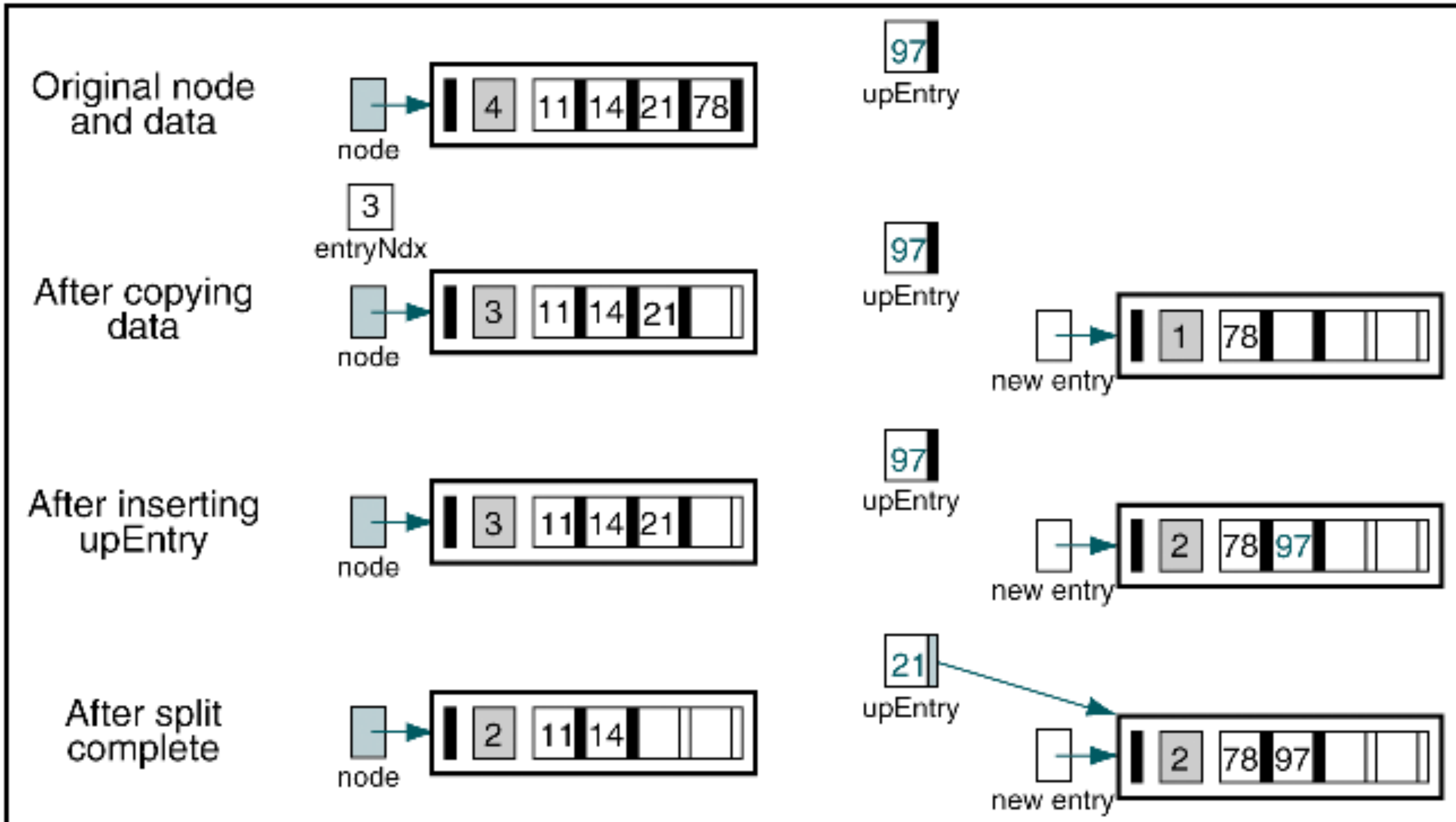


Split Node B-tree Order of 5



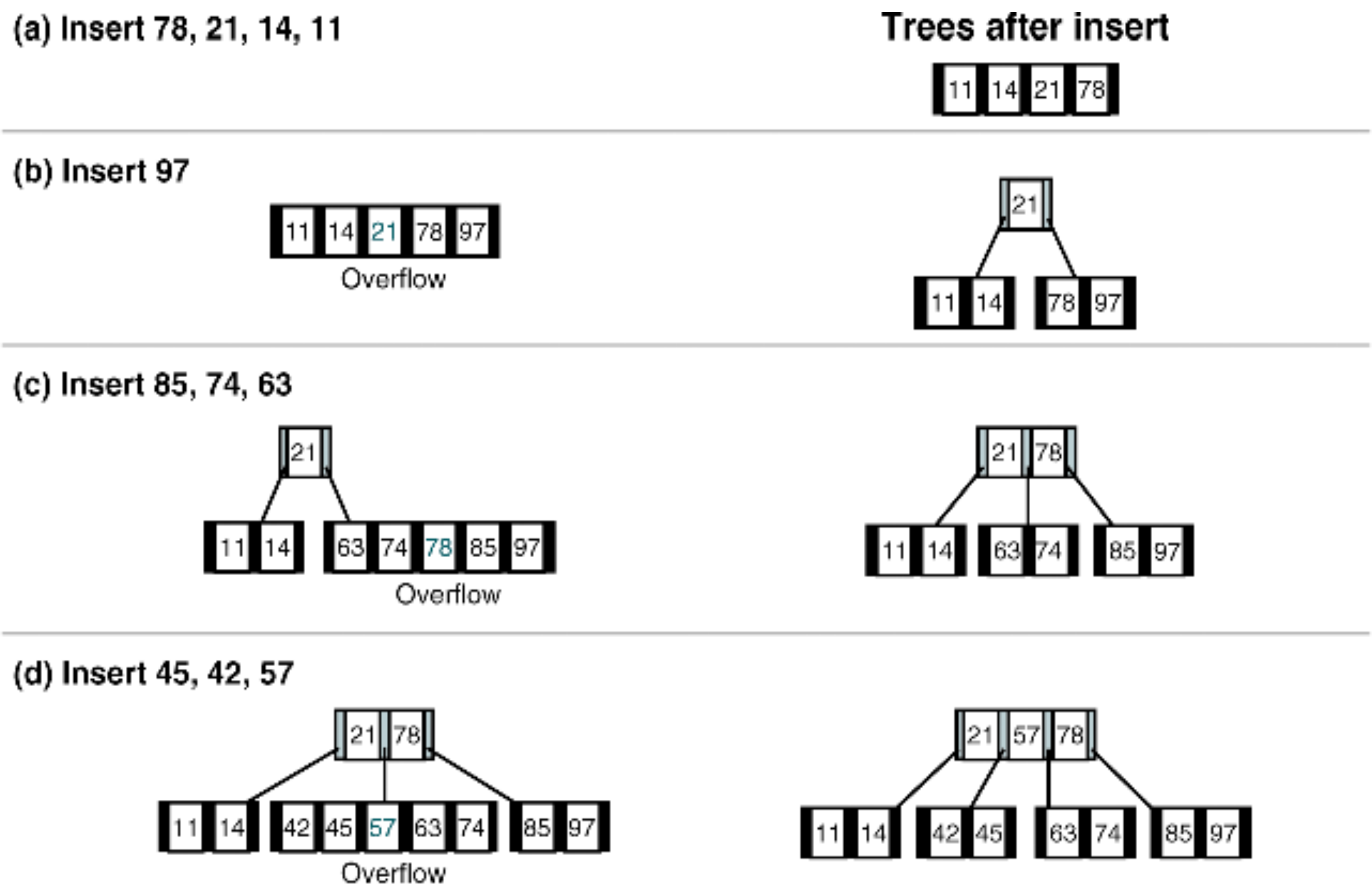
(a) New entry ≤ median

Split Node B-tree Order of 5 (cont.)



(b) New entry > median

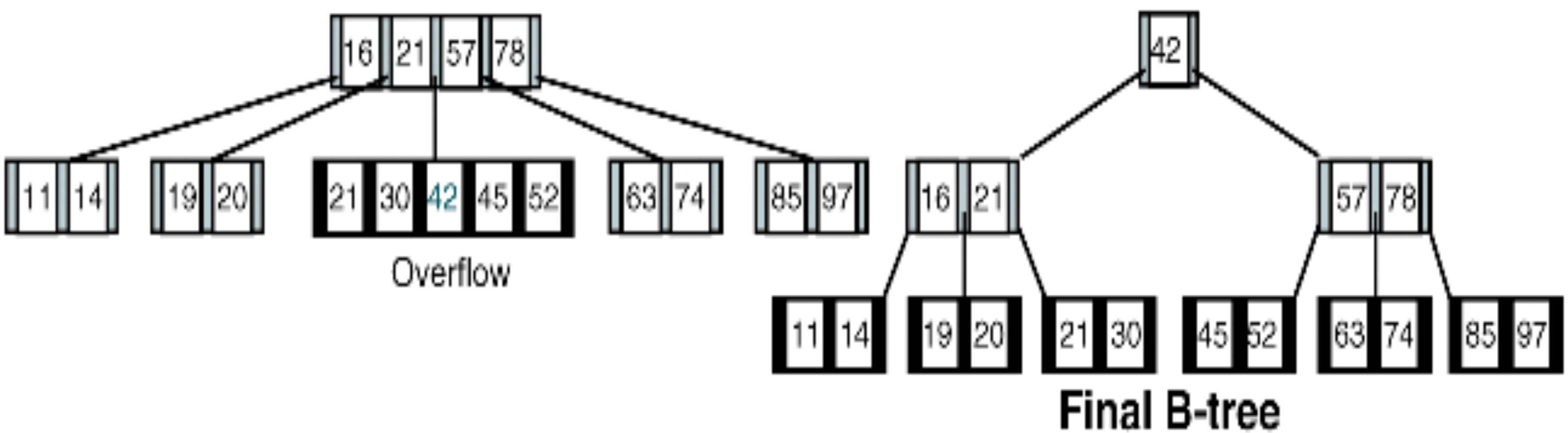
Building a B-tree of Order 5



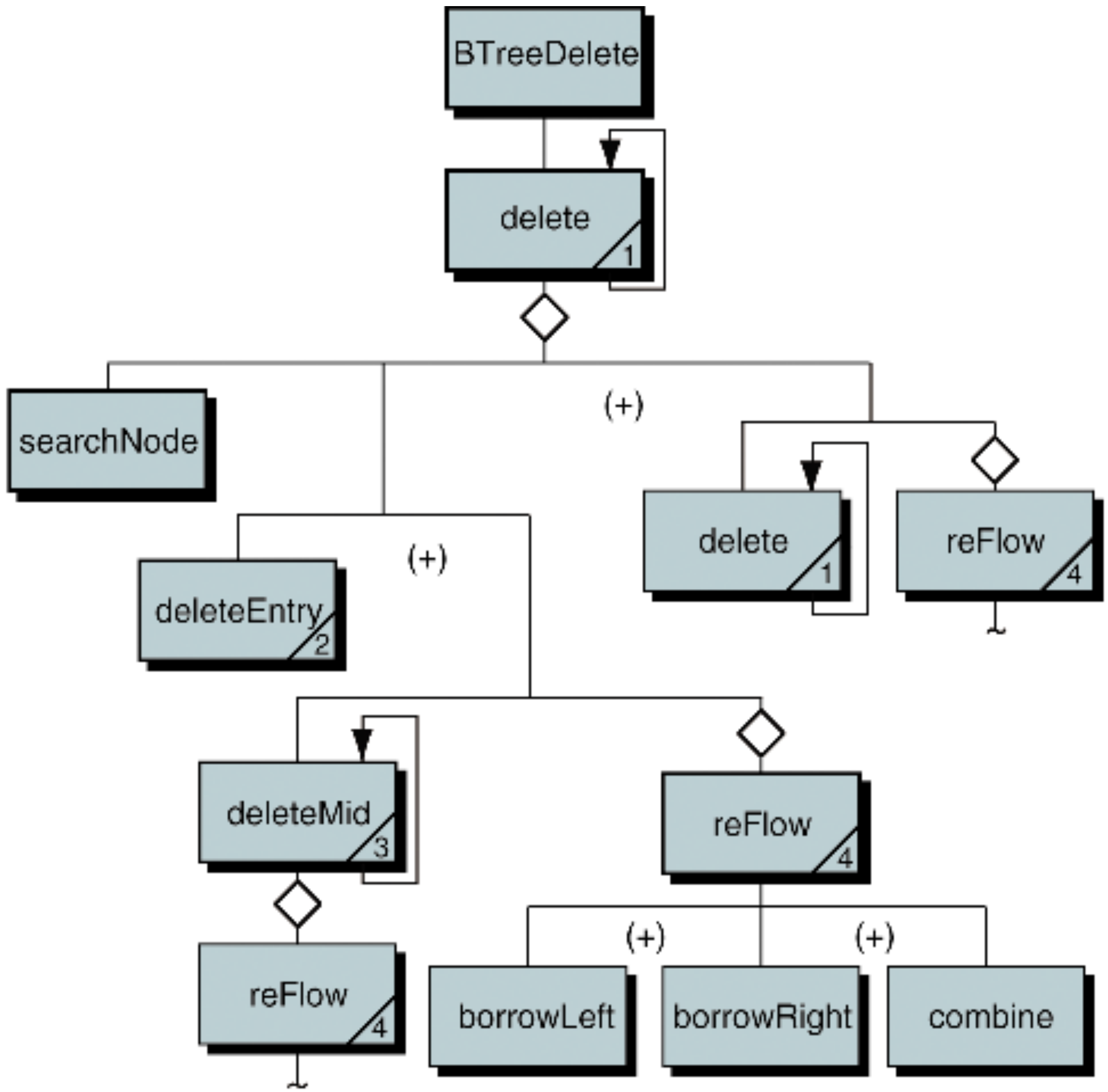
Building a B-tree of Order 5 (cont.)

(e) Insert 20, 16, 19

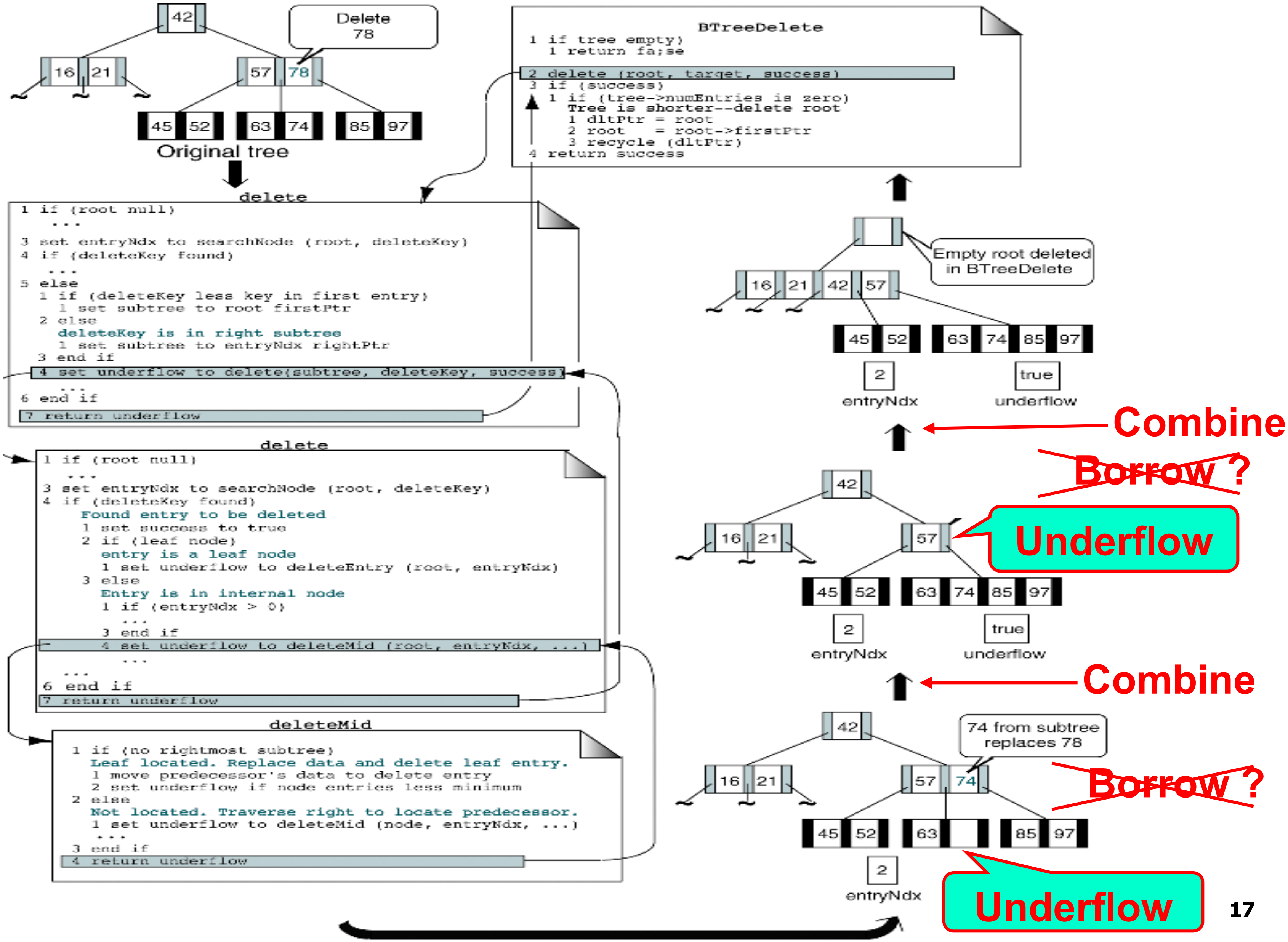
(f) Insert 52, 30, 21



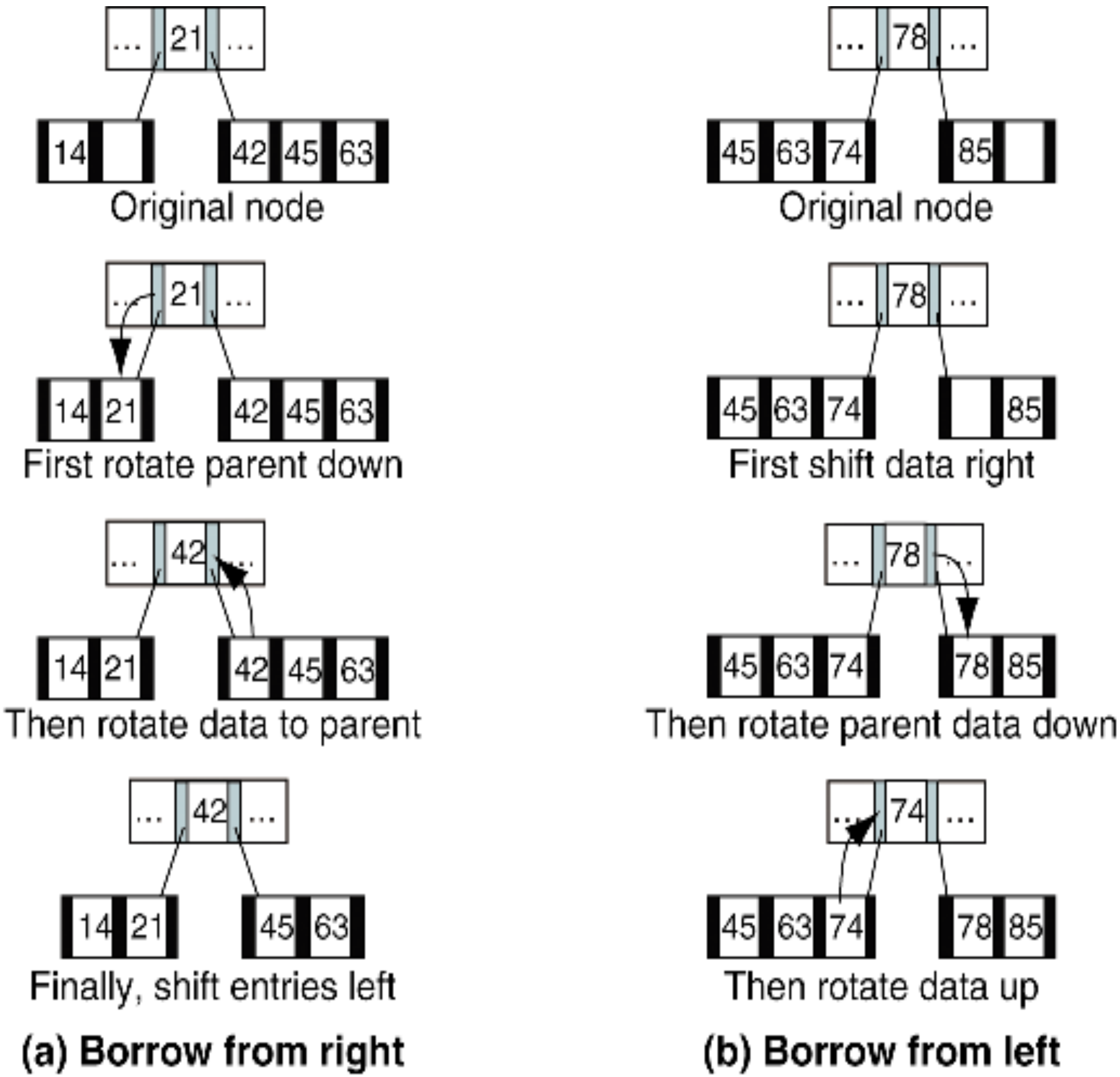
B-tree Delete Design



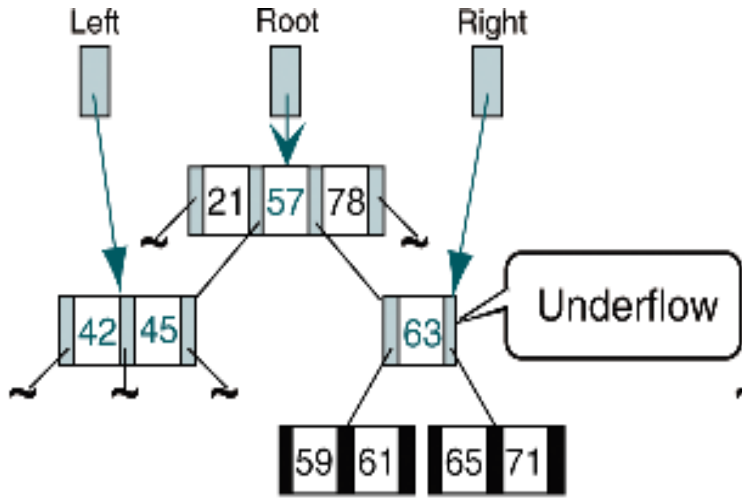
B-tree Deletions



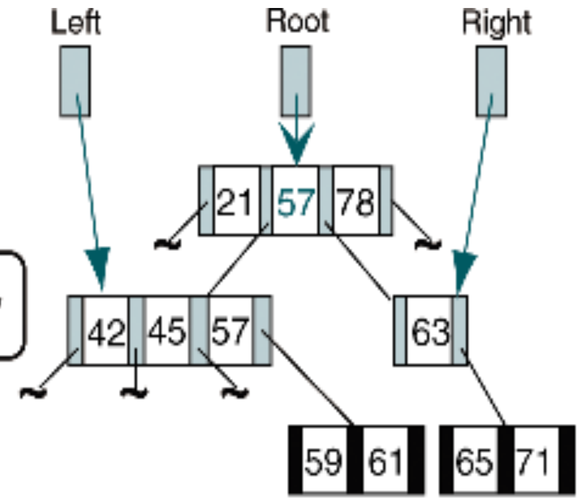
Restoring Order by Borrowing



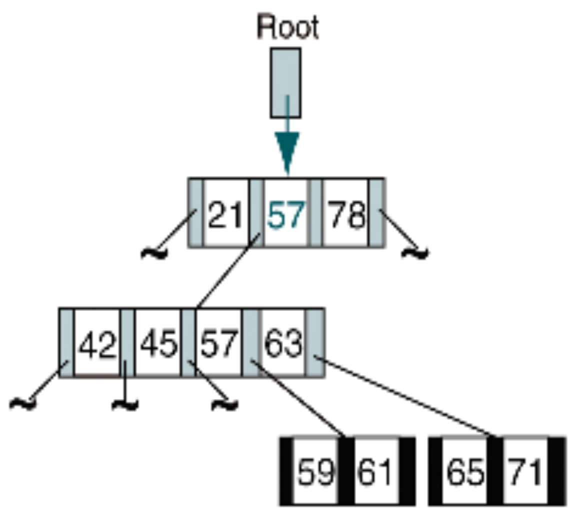
B-tree Combine



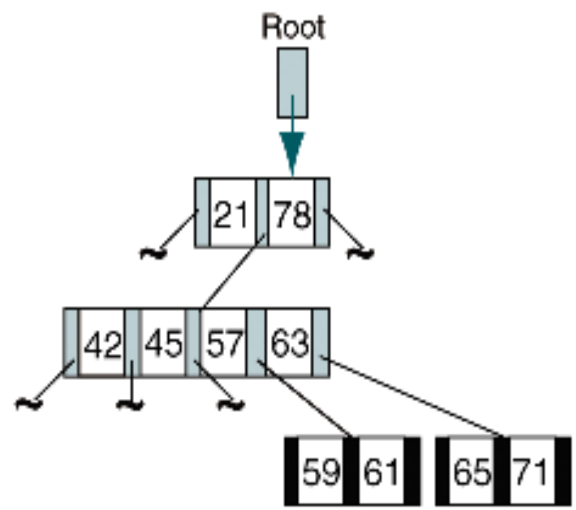
(a) After underflow



(b) After moving root to subtree



(c) After moving right entries



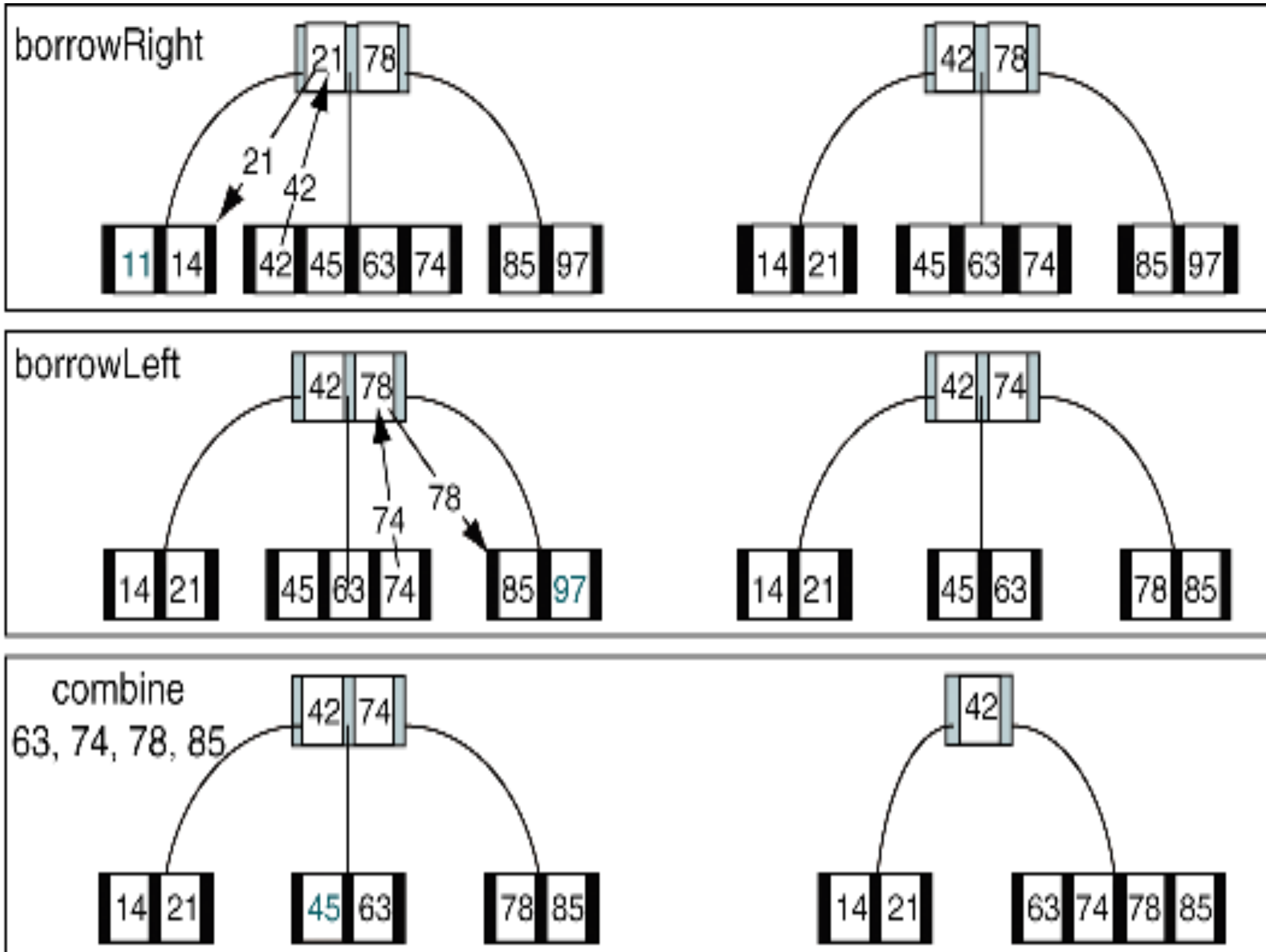
(d) After shifting root

B-tree Deletion Summary

(a) Delete 11

(b) Delete 97

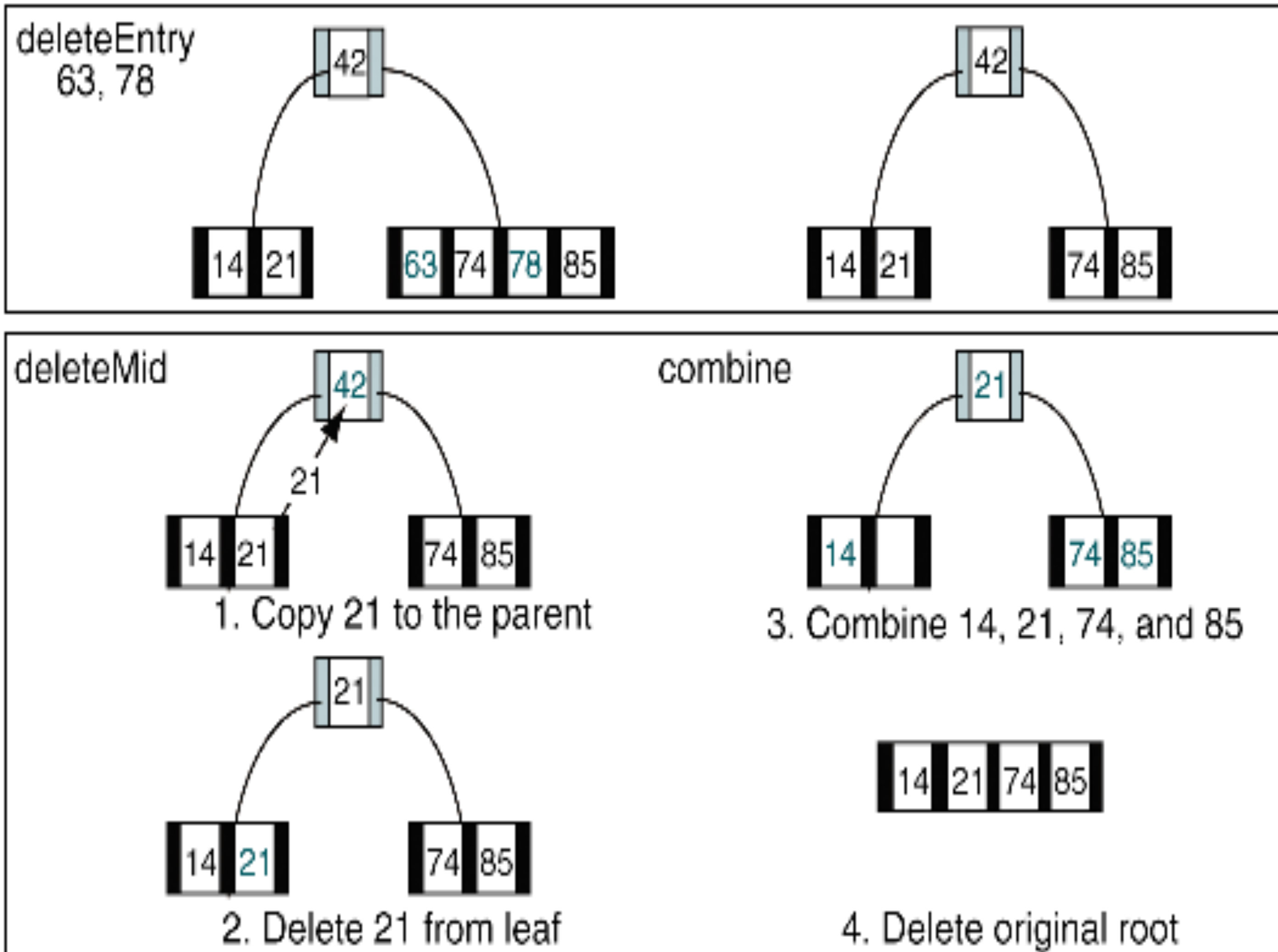
(c) Delete 45



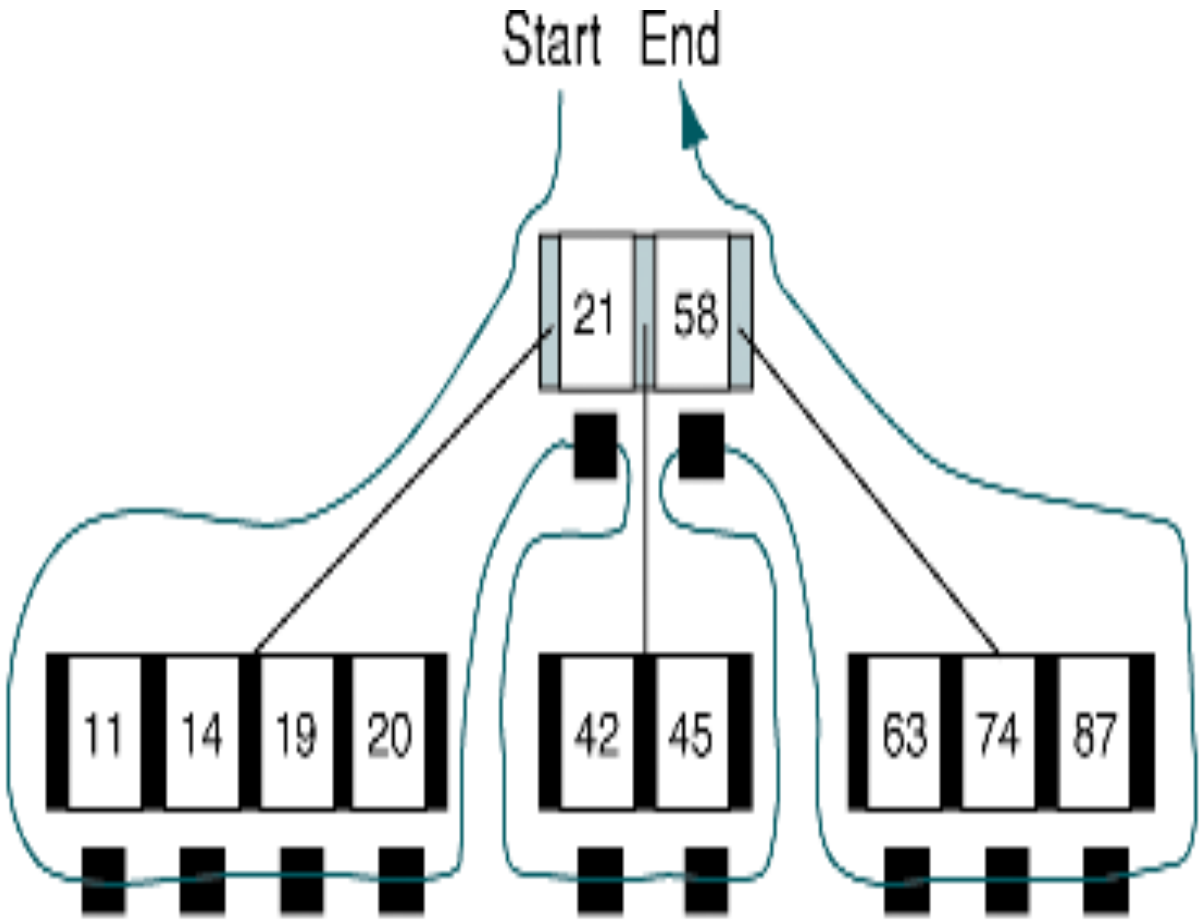
B-tree Deletion Summary (cont.)

(d) Delete 63 and 78

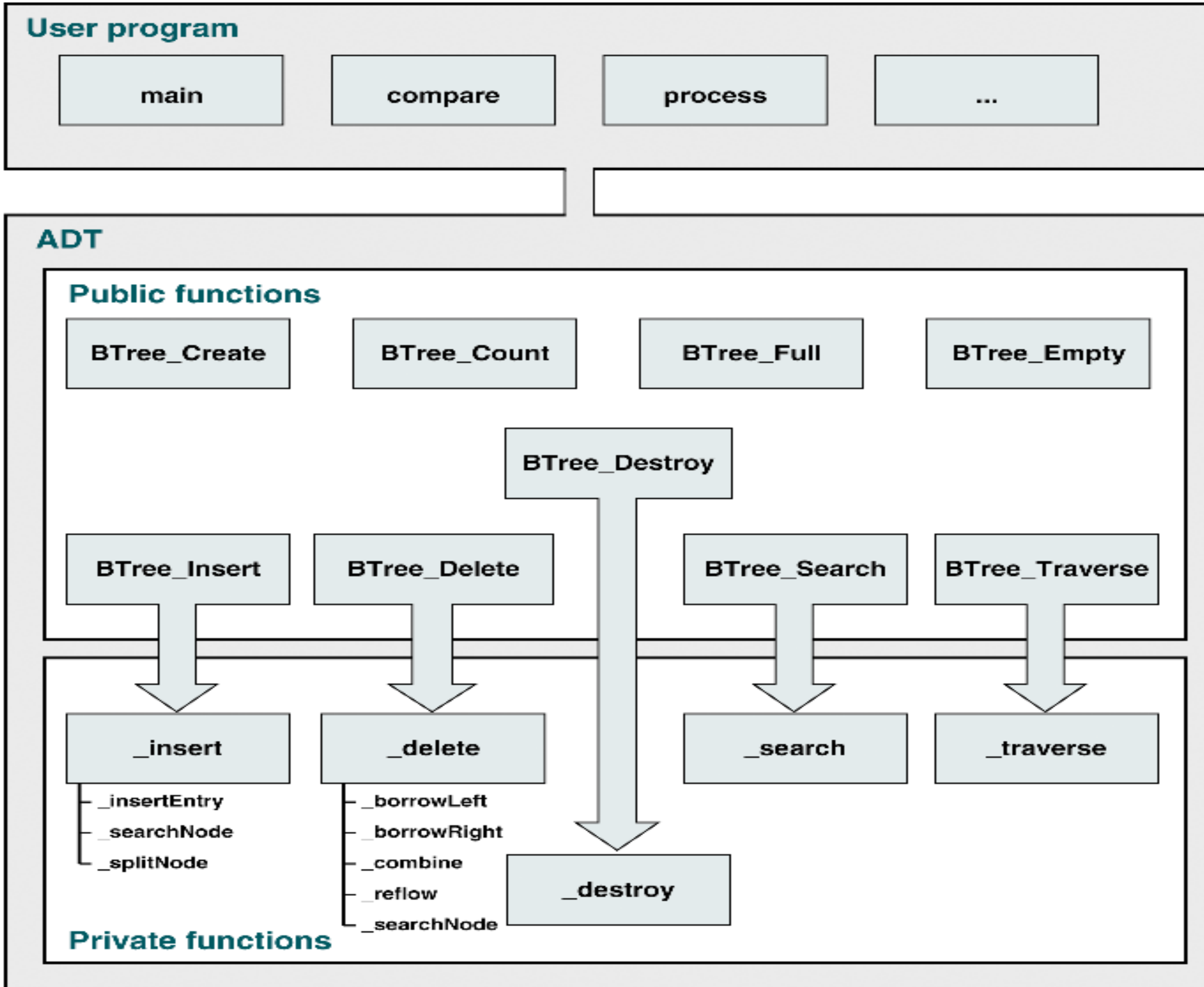
(e) Delete 42



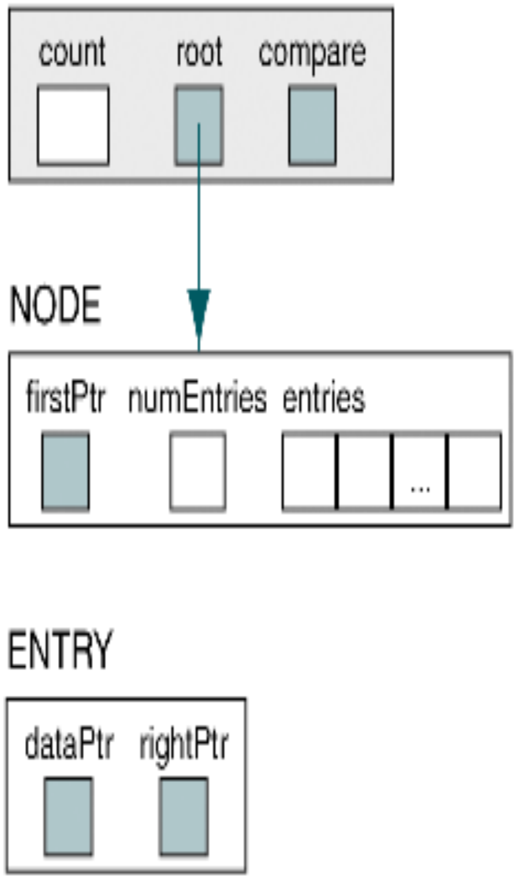
Basic B-tree Traversal



10-3 B-tree ADT



B-tree Data Structure




```
typedef struct
 {
  void*        dataPtr;
  struct node* rightPtr;
 } ENTRY;

typedef struct node
 {
  struct node* firstPtr;
  int          numEntries;
  ENTRY        entries[ORDER - 1];
 } NODE;

typedef struct
   {
    int   count;
    NODE* root;
    int  (*compare) (void* argu1, void* argu2);
   } BTREE;
```

B-tree Declaration

```
/* ==================== B-Tree.h ====================
   This header file contains the functions for the AVL
   Tree abstract data type.
      Written by:
      Date:
*/
#include <stdlib.h>
#include <stdbool.h>

// =============== CONSTANTS & MACROS ==============
const int ORDER = 5;
const int MIN_ENTRIES = (((ORDER + 1) / 2) - 1);

// ================== STRUCTURES ==================
struct node;

typedef struct
   {
    void*        dataPtr;
    struct node* rightPtr;
   } ENTRY;

typedef struct node
```

B-tree Declaration (cont.)

```
    {
     struct node*  firstPtr;
     int           numEntries;
     ENTRY         entries[ORDER - 1];
    } NODE;

typedef struct
    {
     int    count;
     NODE*  root;
     int   (*compare) (void* argu1, void* argu2);
    } BTREE;

// =========== Prototype Declarations ===========

// User interfaces
BTREE* BTree_Create
        (int  (*compare) (void* argu1, void* argu2));
void   BTree_Traverse
        (BTREE* tree, void (*process) (void* dataPtr));
BTREE* BTree_Destroy (BTREE* tree);
void   BTree_Insert  (BTREE* tree, void* dataInPtr);
bool   BTree_Delete  (BTREE* tree, void* dltKey);
void*  BTree_Search  (BTREE* tree, void* dataPtr);
bool   BTree_Empty   (BTREE* tree);
bool   BTree_Full    (BTREE* tree);
int    BTree_Count   (BTREE* tree);
```

B-tree Declaration (cont.)

```

// Internal BTree functions
static void* _search
                (BTREE* tree, void*  targetPtr,
                 NODE*  root);
static int   _searchNode
                (BTREE* tree, NODE* nodePtr,
                 void*  target);
static bool  _delete
                (BTREE* tree,       NODE* root,
                 void*  dltKeyPtr, bool* success);
static bool _insert
                (BTREE* tree,       NODE*  root,
                 void*  dataInPtr, ENTRY* upEntry);
static void _traverse
                (NODE* root,
                 void  (*process)(void* dataPtr));
static void _splitNode
                (NODE*  root,     int    entryNdx,
                 int compResult, ENTRY* upEntry);
static void _insertEntry
```

B-tree Declaration (cont.)

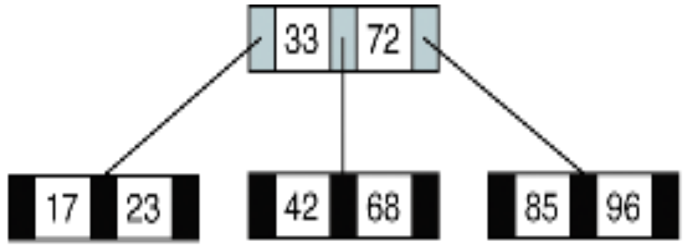
```
                      (NODE*  root, int entryNdx,
                       ENTRY  upEntry);
static bool _deleteEntry
                      (NODE*  node, int entryNdx);
static bool _deleteMid
                      (NODE*  root, int entryNdx,
                       NODE*  leftPtr);
static bool _reFlow
                      (NODE*  root, int entryNdx);
static void _borrowLeft
                      (NODE*  root,     int   entryNdx,
                       NODE*  leftTree, NODE* rightTree);
static void _borrowRight
                      (NODE*  root,     int   entryNdx,
                       NODE*  leftTree, NODE* rightTree);
static void _combine
                      (NODE*  root,       int   entryNdx,
                       NODE*  leftTree,   NODE* rightTree);
static void _destroy (NODE* root);
```

10-4 Simplified B-tree

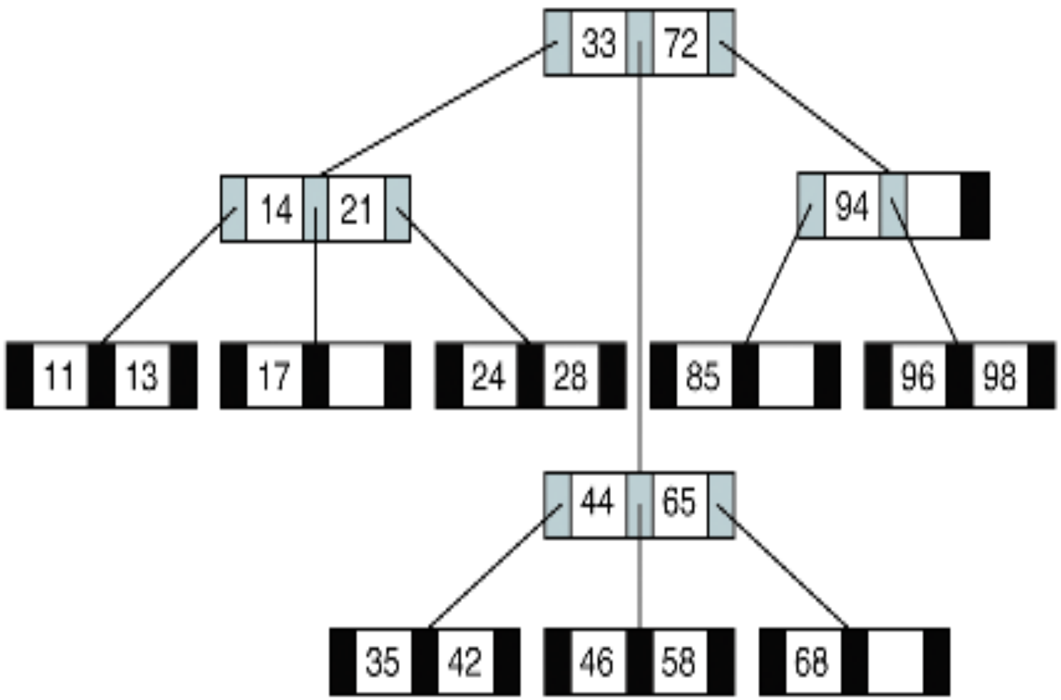
This section discusses 2 specialized B-tree which have been assigned unique names by computer scientists

- **2-3 tree**
- **2-3-4 tree**

2-3 Trees

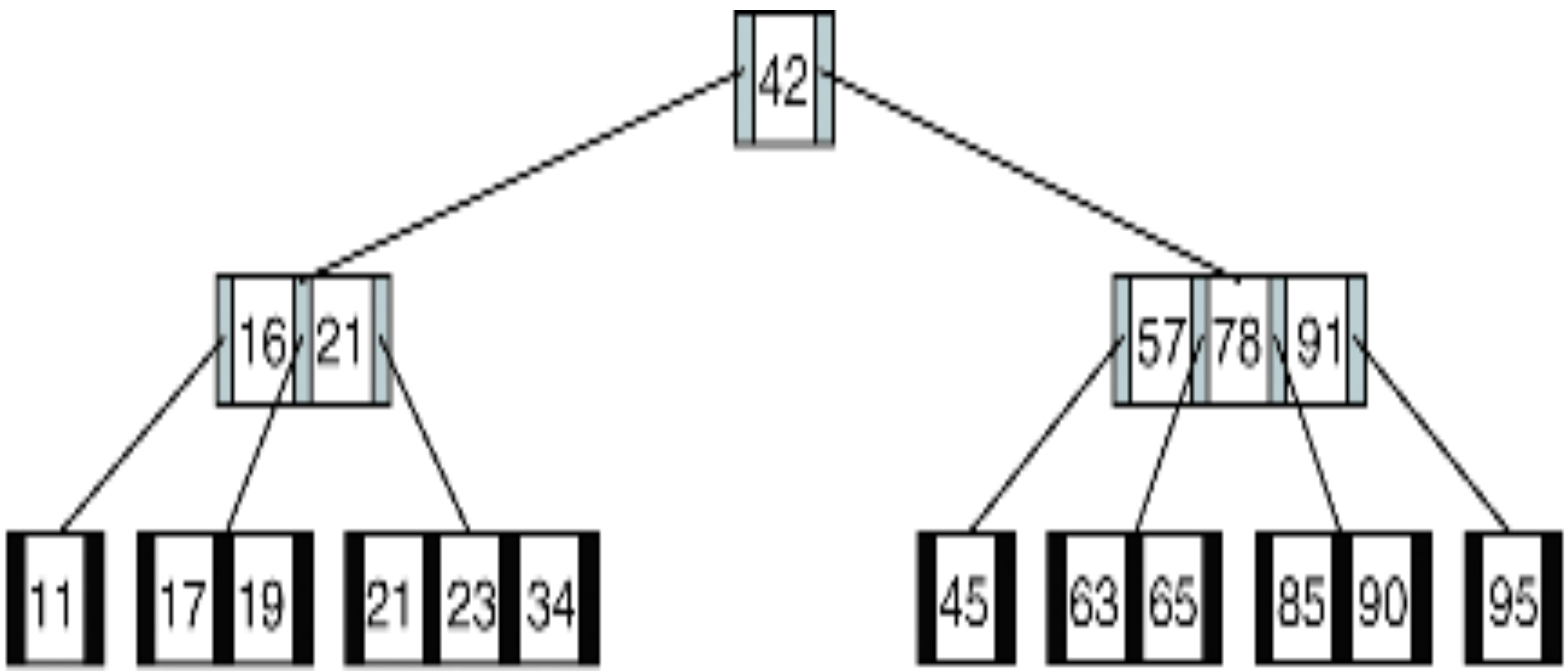


(a) Complete 2-3 tree



(b) 2-3 tree with empty entries

2-3-4 Tree

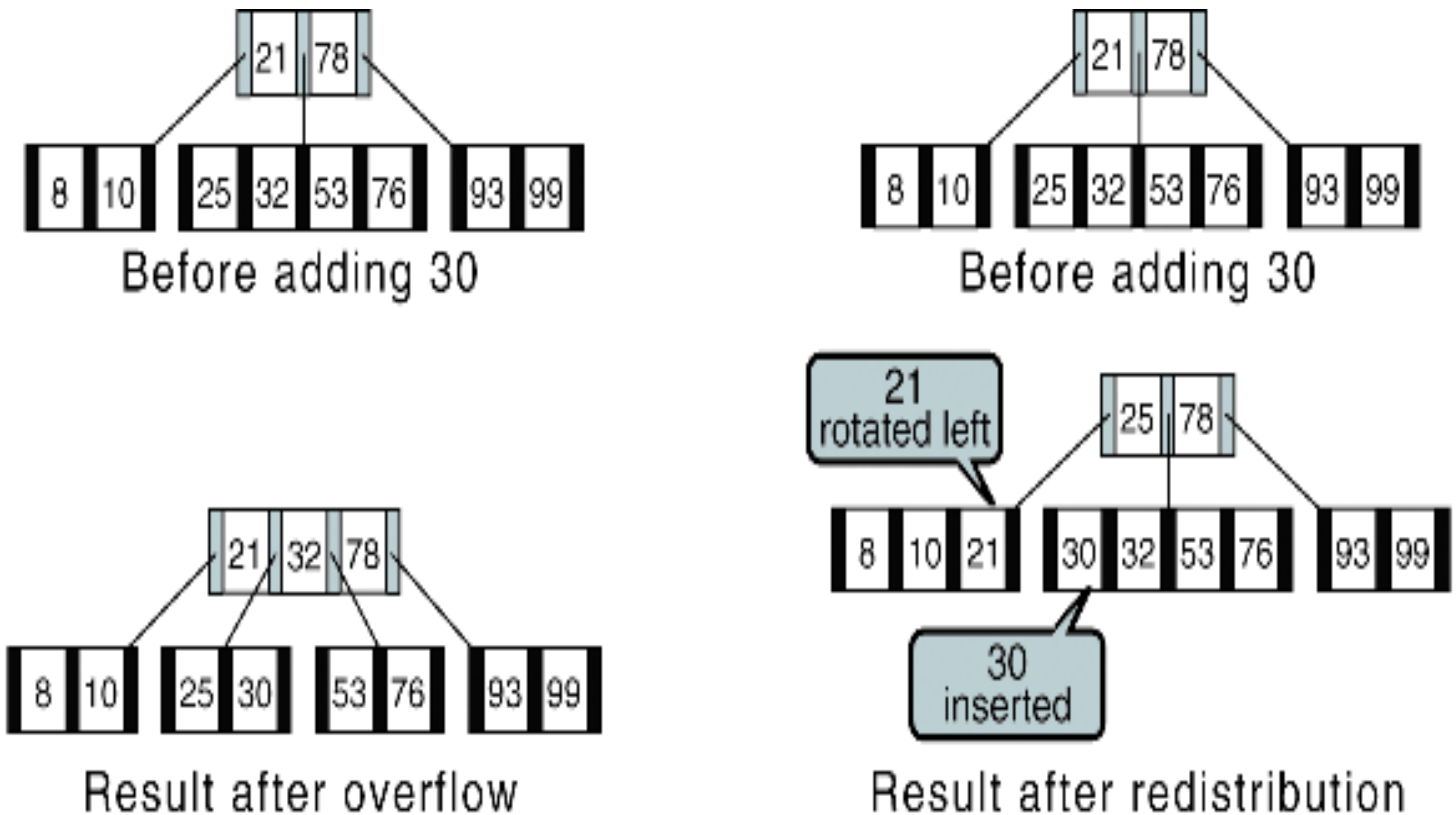


10-5 B-tree Variations

This section discusses two popular variations on the B-tree

- **B* tree**
- **B+ tree**

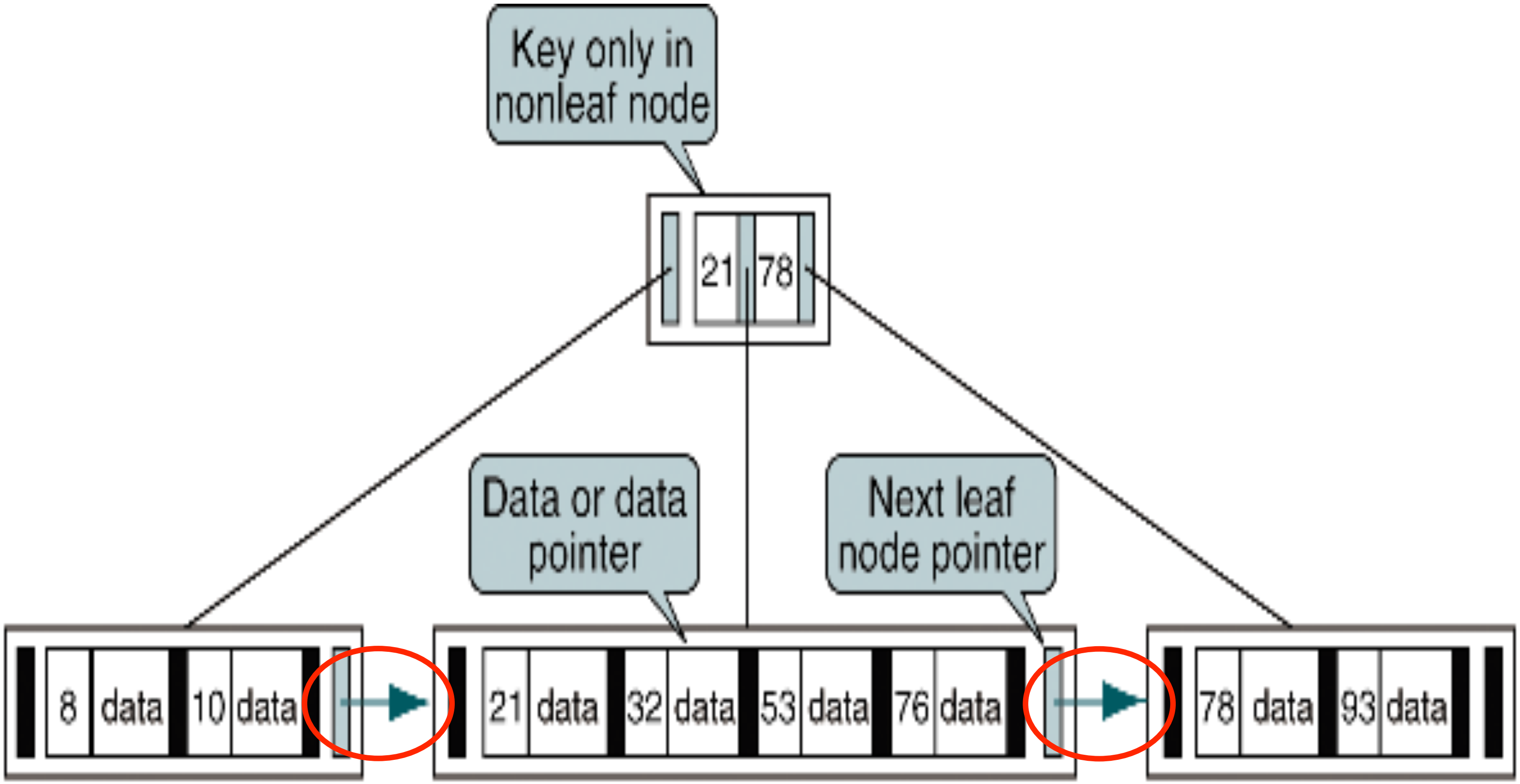
B* tree Insertion



(a) Insertion into B-tree of order 5

(b) Insertion into B*tree of order 5

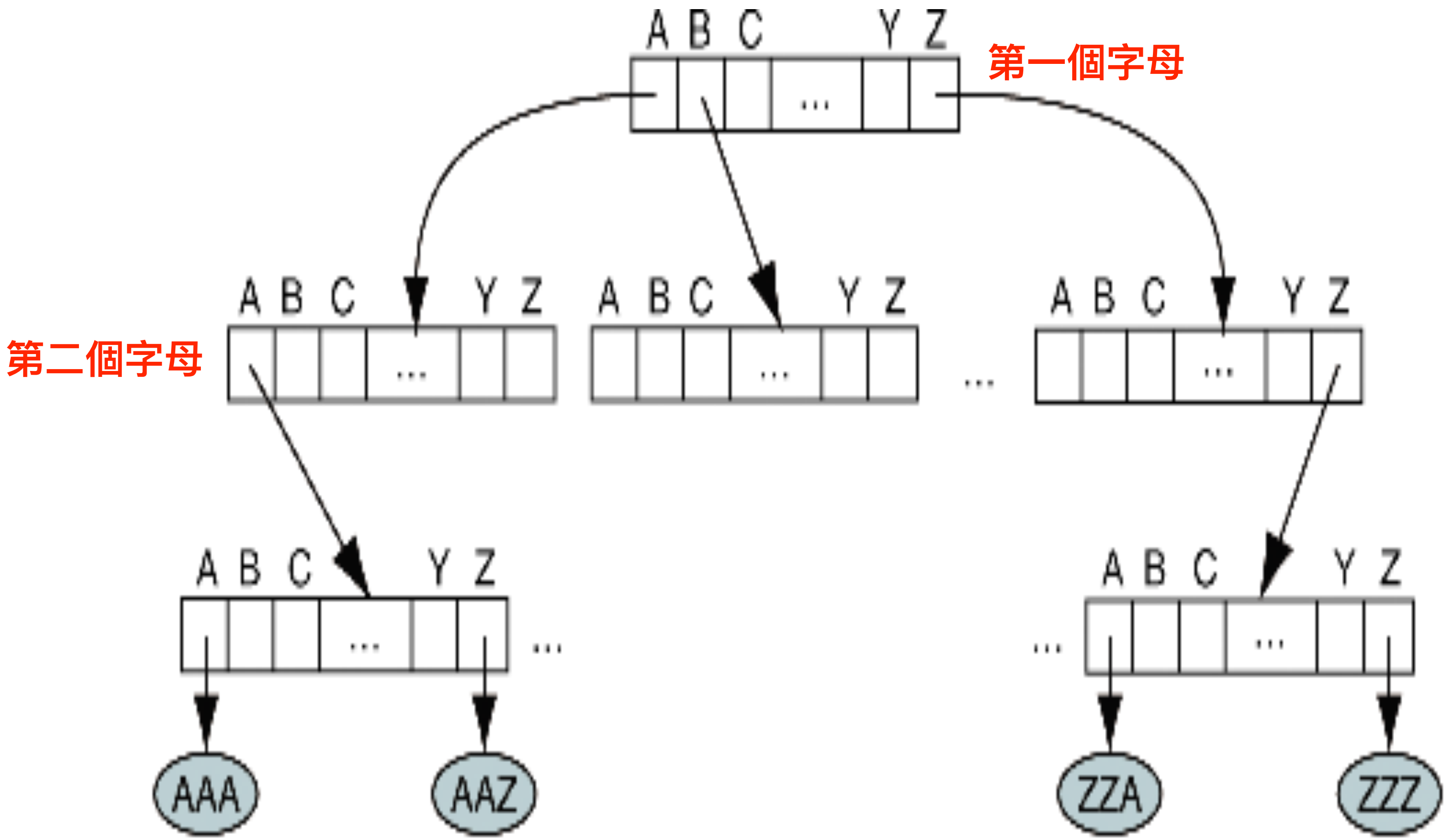
B+ tree 結合了linear list跟tree的優點



leaf node才有data

10-6 Lexical Search Tree

字典可能會用這種結構

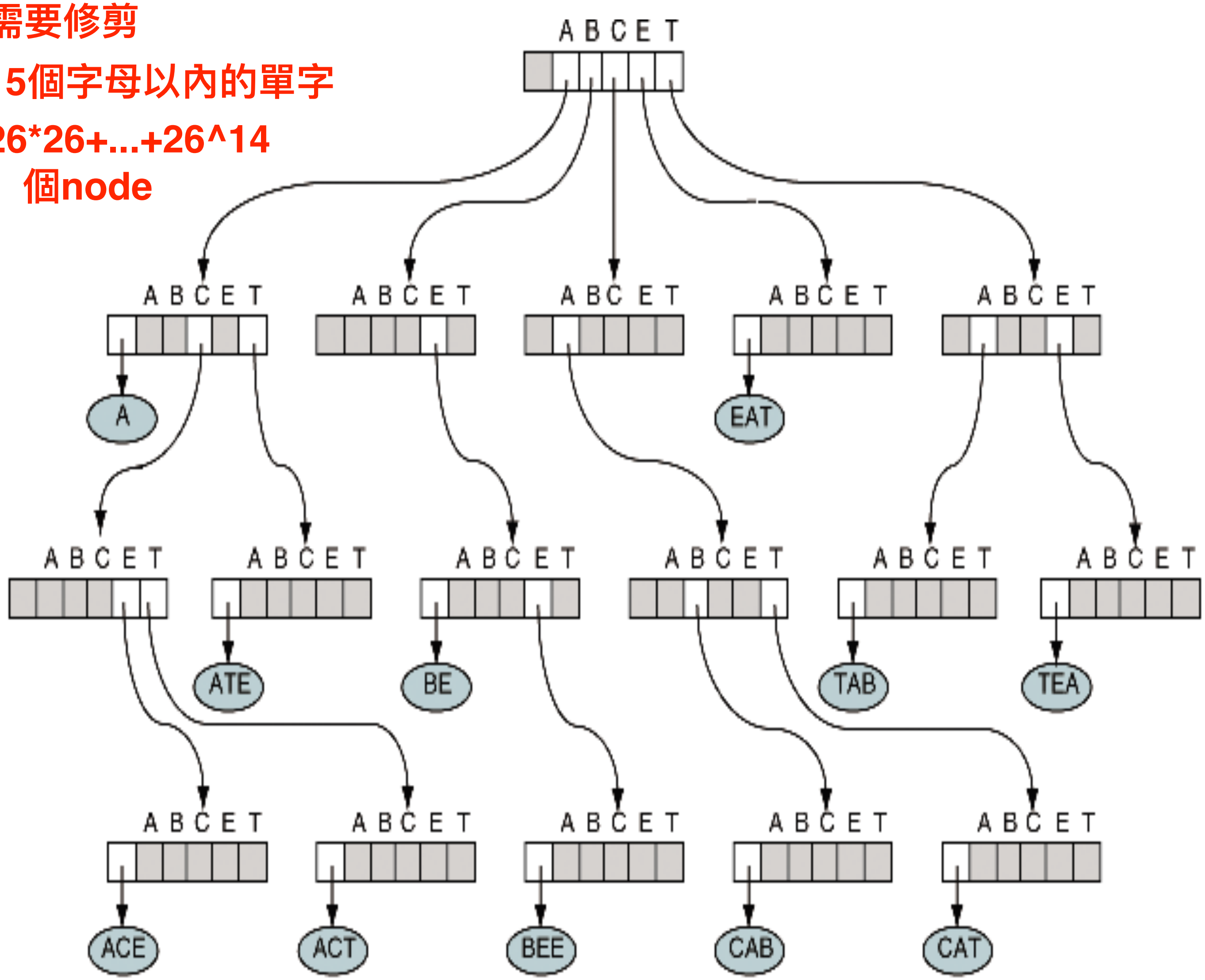


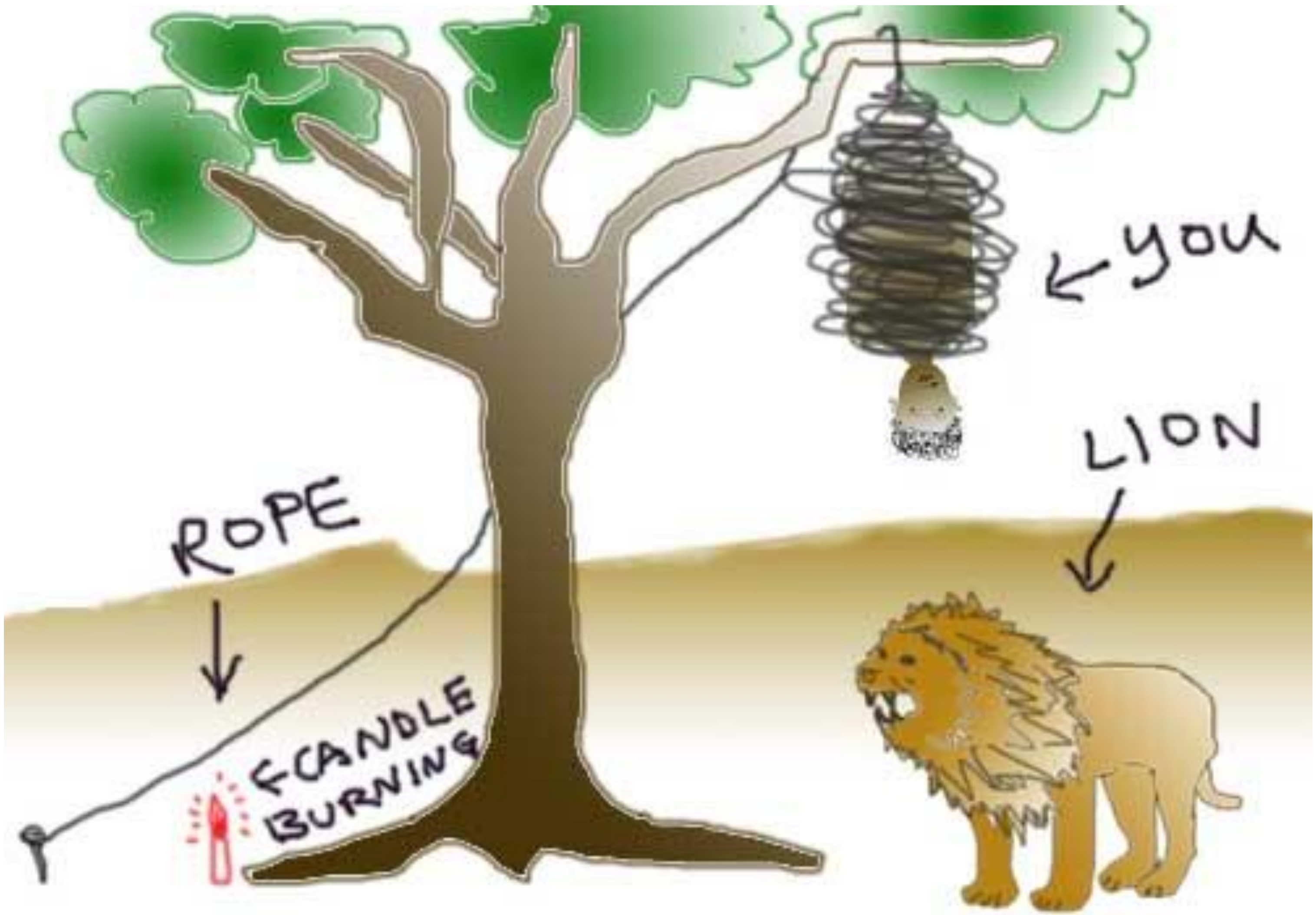
Spell Checker Trie

實務上可能需要修剪

要儲存所有15個字母以內的單字

需要1+26+26*26+...+26^14（約10^19）個node





← YOU
LION
ROPE
CANDLE
BURNING


HAPPY BIRTHDAY TO YOU
CANDLE NO FIRE!
←HAPPY LION


NEED CASH
FOR ALCOHOL
RESEARCH

With the high rate of attacks on women in secluded parking lots, especially during evening hours, the Minneapolis City Council has established a "Women Only" parking lot at the Mall of America. Even the parking lot attendants are exclusively female so that a comfortable and safe environment is created for patrons.

Below is the first picture available of this world-first women-only parking lot in Minnesota.