

Chapter 10

Multiway Trees

Objectives

Upon completion you will be able to:

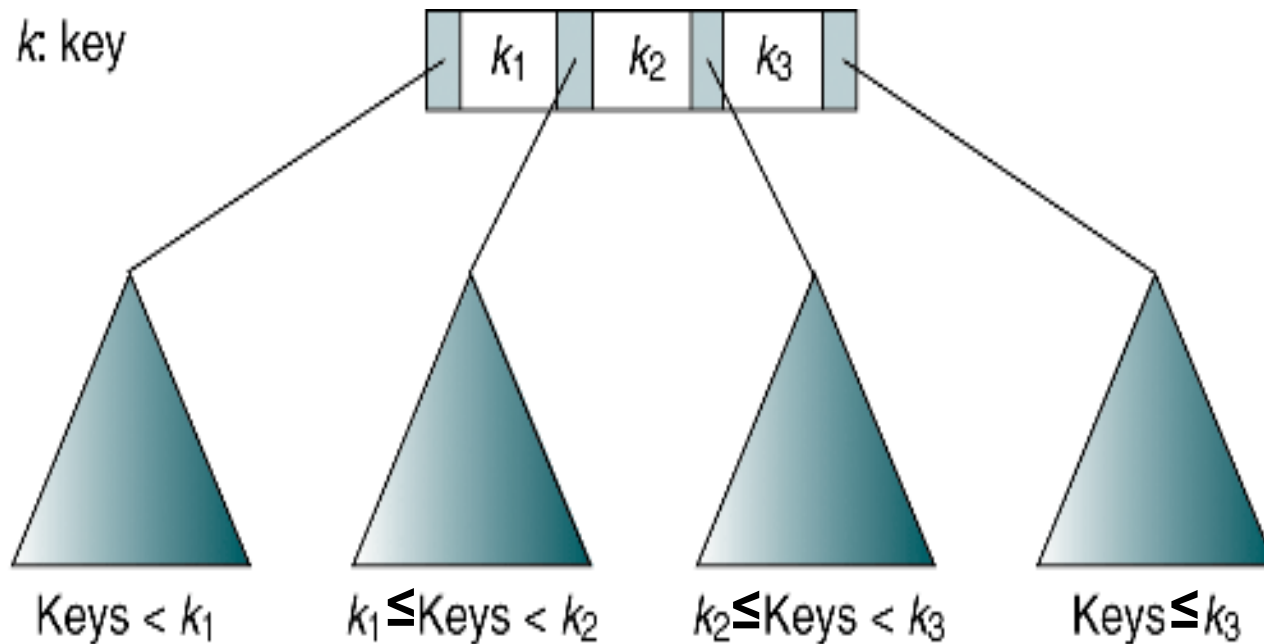
- Define and discuss multiway tree structures
- Understand the operation and use of the B-tree ADT
- Discuss the use and implementation of simplified B-trees
- Compare and contrast B-trees, B*trees, and T+trees
- Discuss the design and use a lexical search trees (Tries)

10-1 M-way Search Tree

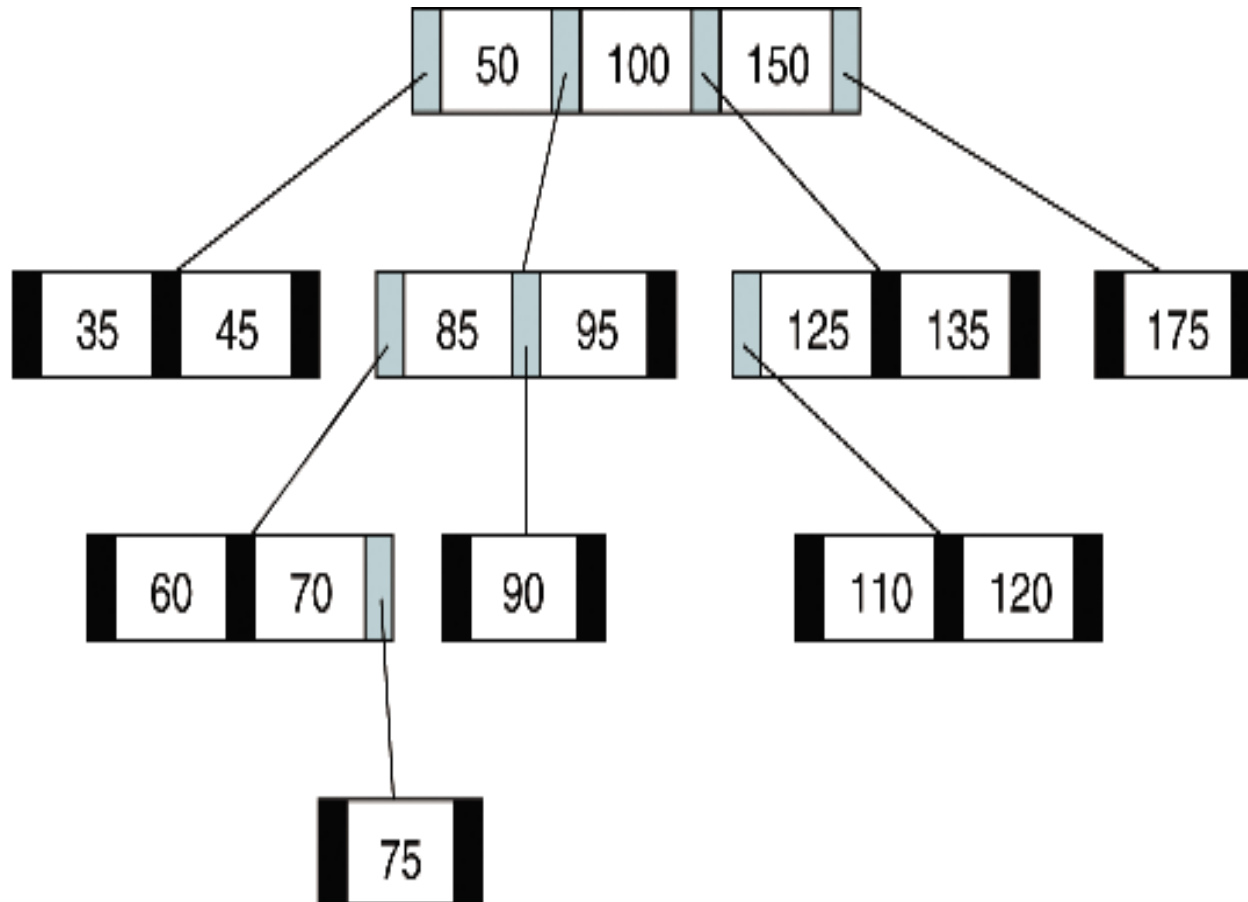
This section explore trees whose outdegree is not restricted to 2 but that retain the general properties of binary search trees; Multiway trees have multiway entries in each node and thus may have multiple subtrees.

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

M-way Tree

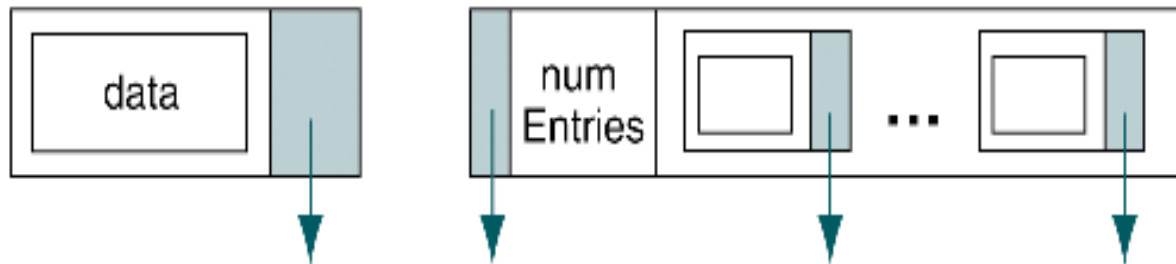


4-way Tree



M-way Node Structure

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



(a) Entry

(b) Node

10-2 B-trees

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

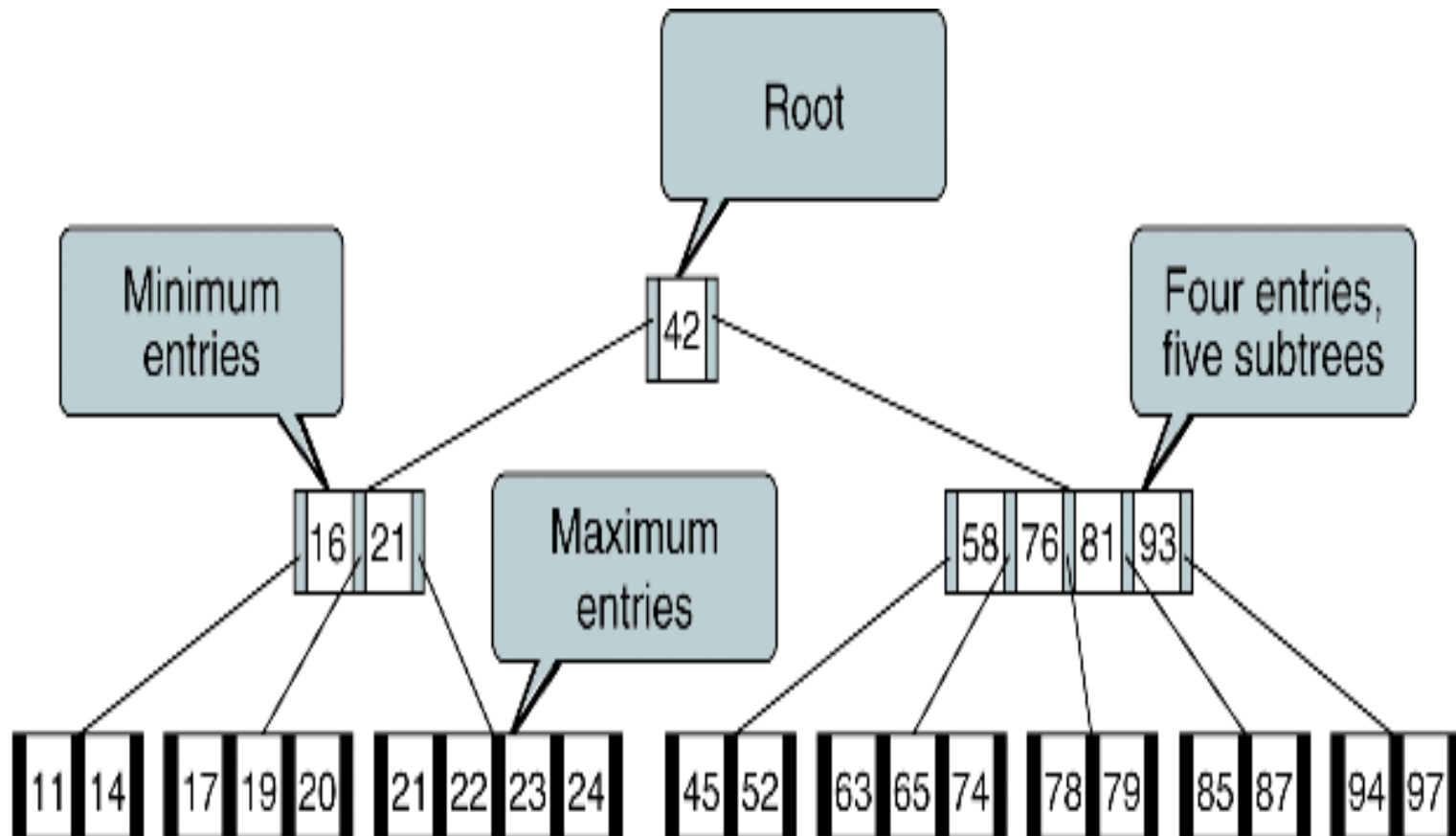
- Insertion
- Deletion
- Traversal
- Search

Entries in B-trees of Various Orders

- Root: a leaf or it has 2...m subtrees
- Internal node: $\lceil m/2 \rceil \dots m$ subtrees ($\lceil m/2 \rceil - 1 \dots m - 1$ entries)
- Leaf node: $\lceil m/2 \rceil - 1 \dots m - 1$ 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

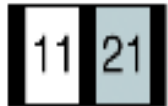
A B-tree of Order 5



B-tree Insert Overview



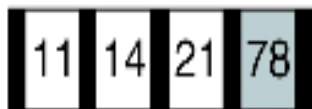
(a) Insert 11



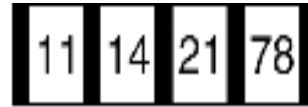
(b) Insert 21



(c) Insert 14

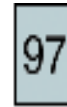


(d) Insert 78

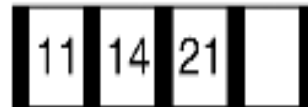


Full node

(e) Ready to insert 97



New data

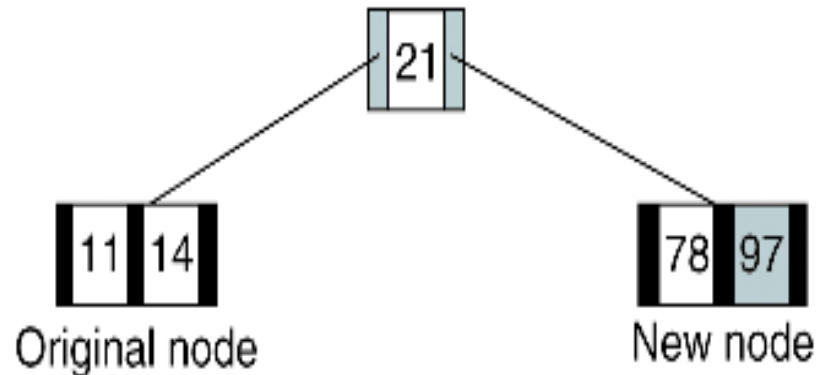


Original node



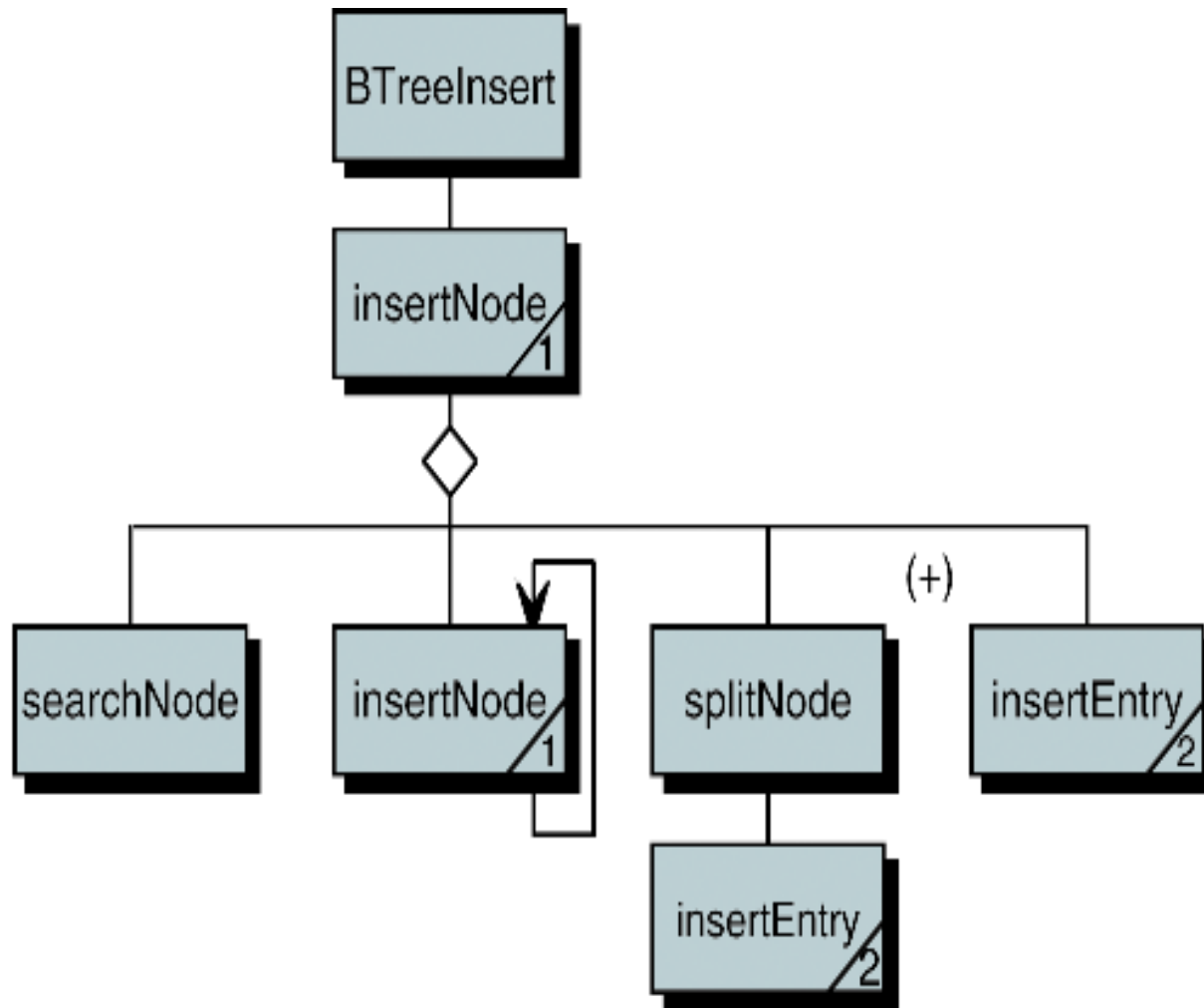
New node

(f) Create new right subtree

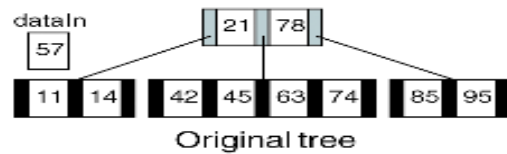


(g) Insert median into parent (new root)

B-tree Insert Design



Build B-tree with Overflow



```

BTreeInsert
3 insertNode (tree, data, upEntry)
4 if (tree Higher)
  Tree has grown. Create new root.
  1 create new node
  2 move upEntry to first entry in new node
  3 set left subtree of node to tree
  4 set tree root to new node
  5 set number of entries to 1
  6 end if
6 return
    
```

```

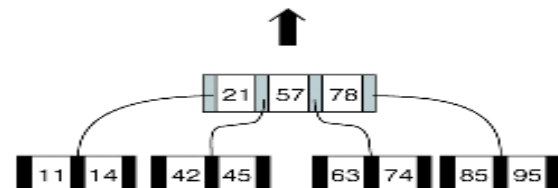
insertNode
1 if (root null)
  ...
2 end if
3 set entryNdx = searchNode (root, key)
  Determine subtree ... left or right
4 if (entryNdx equal 0)
  ...
6 end if
7 set taller to insertNode (subTree, dataIn, ...)
8 if (taller)
  1 if (node full)
    ...
  3 end if
9 end if
10 return taller
    
```

```

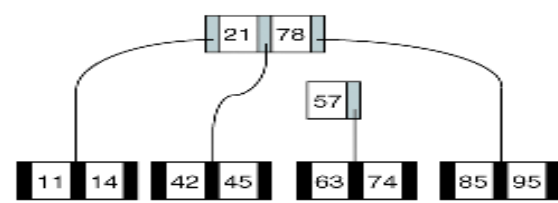
insertNode
1 if (root null)
  ...
2 end if
3 set entryNdx = searchNode (root, key)
  Determine subtree ... left or right
4 if (entryNdx equal 0)
  ...
6 end if
7 set taller to insertNode (subTree, dataIn, ...)
8 if (taller)
  1 if (node full)
    ...
  3 end if
9 end if
10 return taller
    
```

```

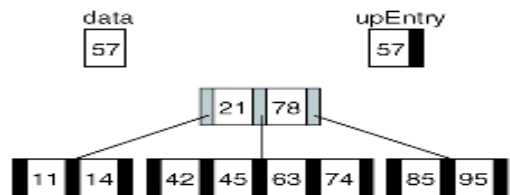
insertNode
1 if (root null)
  Leaf found -- build new entry
  1 move dataIn to upEntry
  2 set upEntry subtree to null
  3 return taller true
2 end if
3 set entryNdx = searchNode (root, key)
8 if (taller)
  ...
9 end if
10 return taller
    
```



1
entryNdx

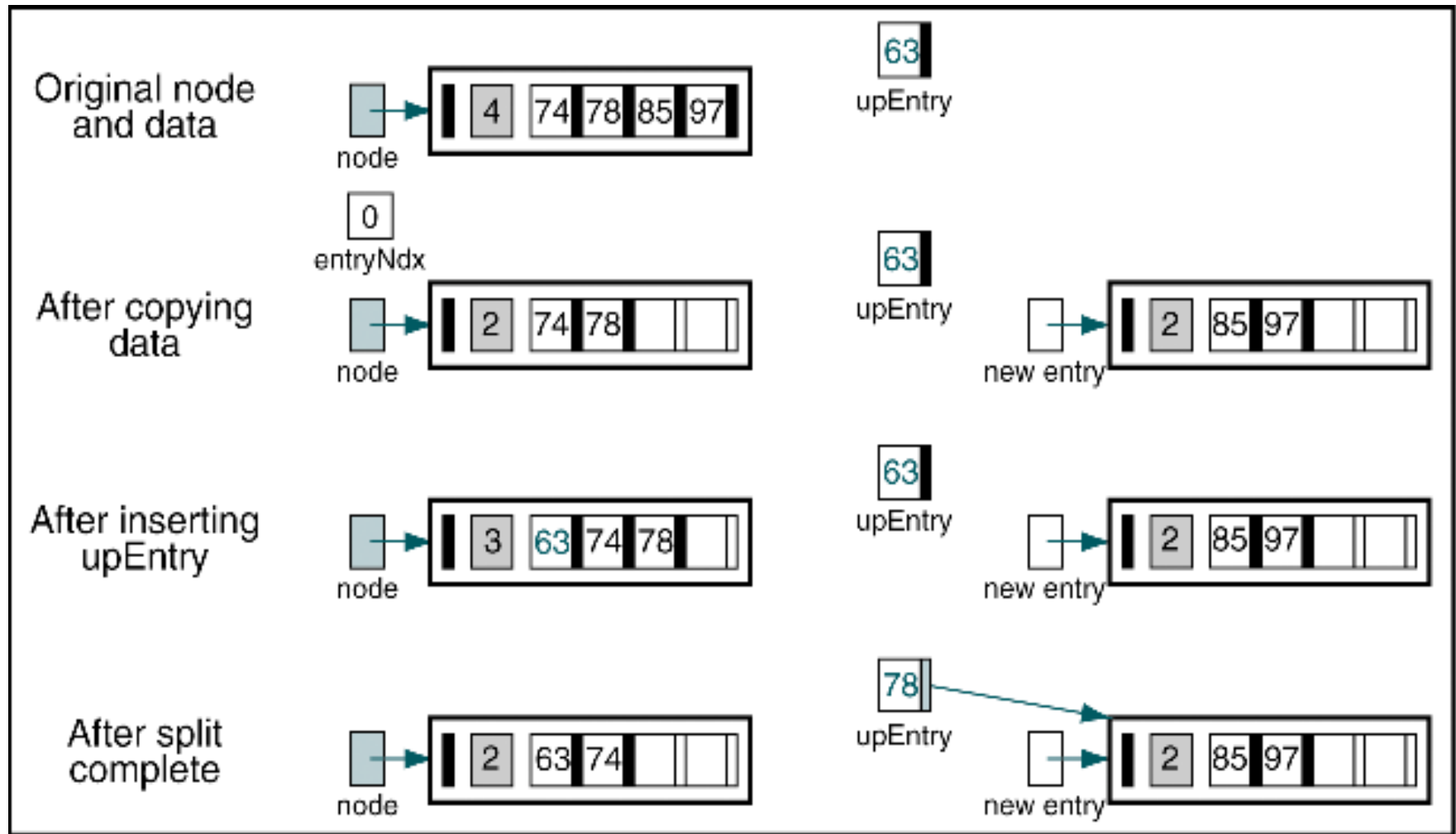


2
entryNdx



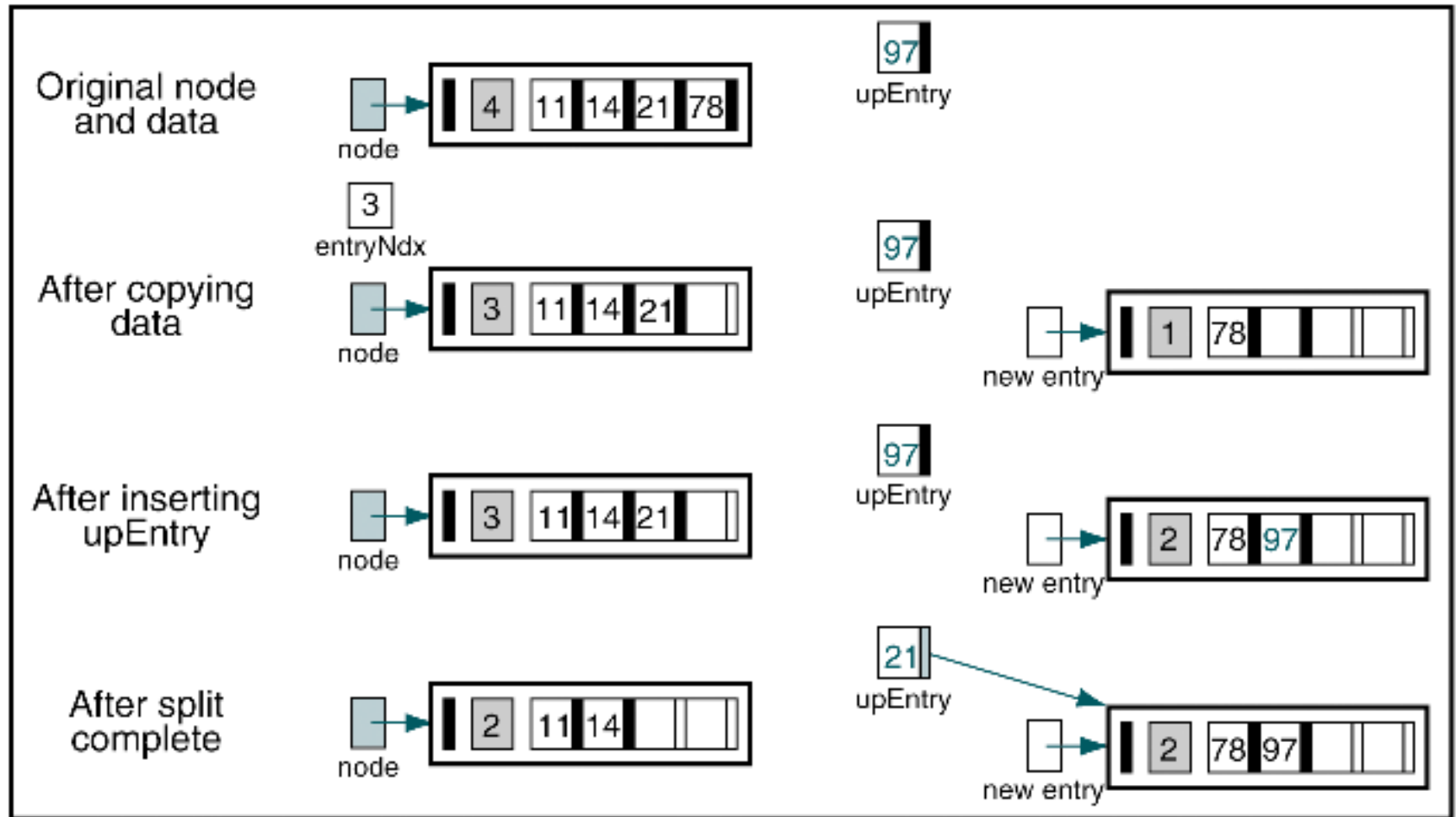
2
entryNdx

Split Node B-tree Order of 5



(a) New entry \leq median

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

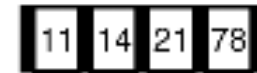


(b) New entry > median

Building a B-tree of Order 5

(a) Insert 78, 21, 14, 11

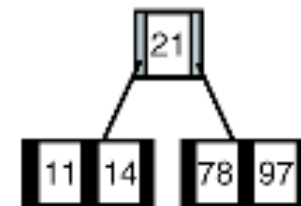
Trees after insert



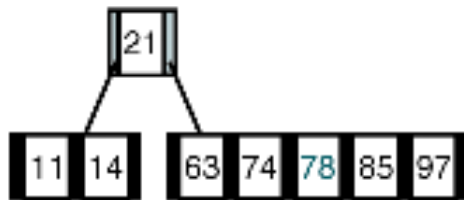
(b) Insert 97



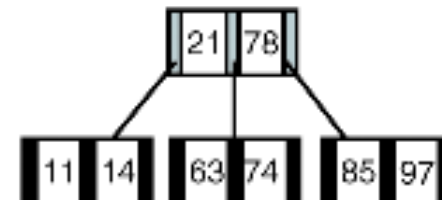
Overflow



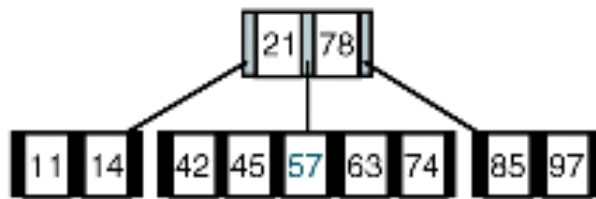
(c) Insert 85, 74, 63



Overflow



(d) Insert 45, 42, 57

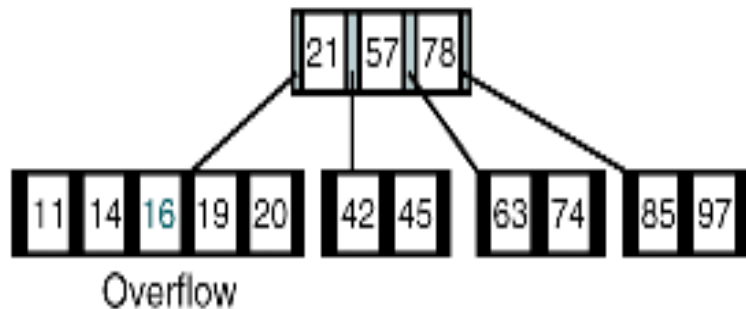


Overflow

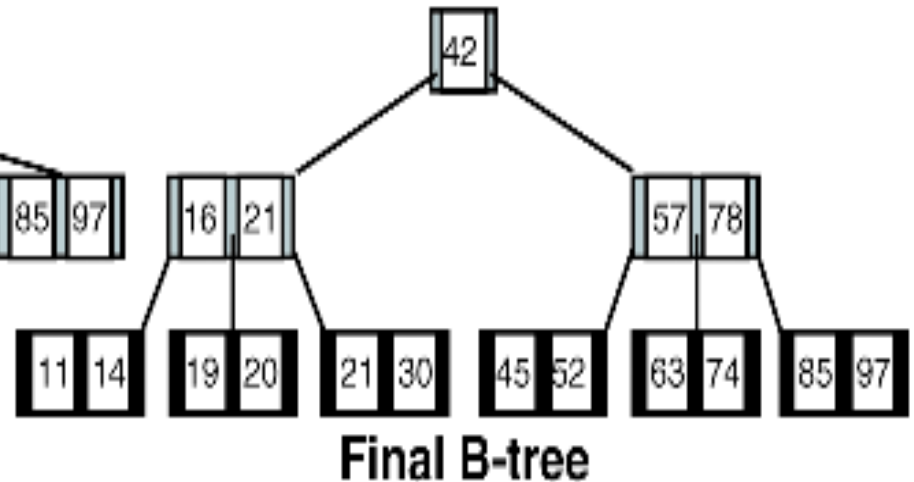
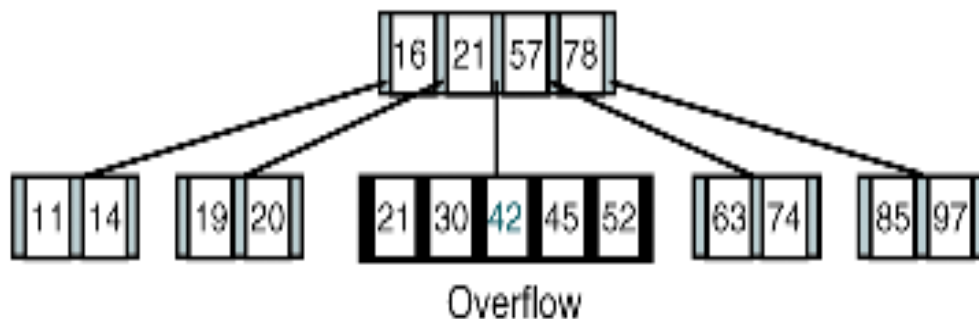


Building a B-tree of Order 5 (cont.)

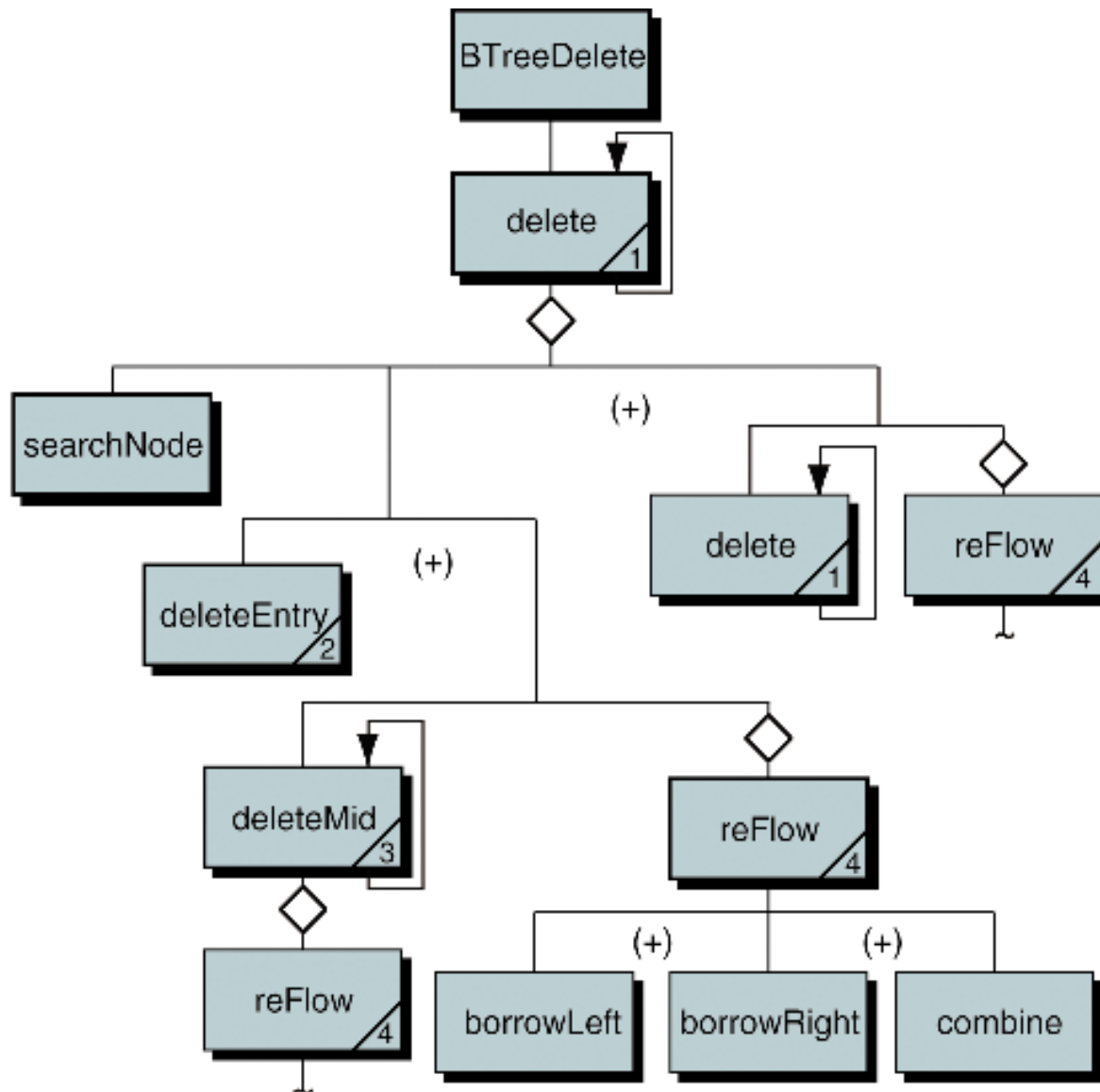
(e) Insert 20, 16, 19



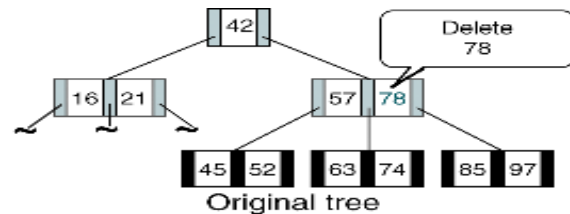
(f) Insert 52, 30, 21



B-tree Delete Design



B-tree Deletions



```

BTreeDelete
1 if tree empty)
1 return false
2 delete (root, target, success)
3 if (success)
1 if (tree->numEntries is zero)
Tree is shorter--delete root
1 dltPtr = root
2 root = root->firstPtr
3 recycle (dltPtr)
4 return success

```

```

1 if (root null)
...
3 set entryNdx to searchNode (root, deleteKey)
4 if (deleteKey found)
...
5 else
1 if (deleteKey less key in first entry)
1 set subtree to root firstPtr
2 else
deleteKey is in right subtree
1 set subtree to entryNdx rightPtr
3 end if
4 set underflow to delete(subtree, deleteKey, success)
6 end if
7 return underflow

```

```

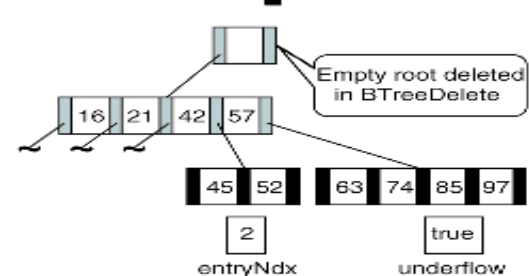
delete
1 if (root null)
...
3 set entryNdx to searchNode (root, deleteKey)
4 if (deleteKey found)
Found entry to be deleted
1 set success to true
2 if (leaf node)
entry is a leaf node
1 set underflow to deleteEntry (root, entryNdx)
3 else
Entry is in internal node
1 if (entryNdx > 0)
...
3 end if
4 set underflow to deleteMid (root, entryNdx, ...)
6 end if
7 return underflow

```

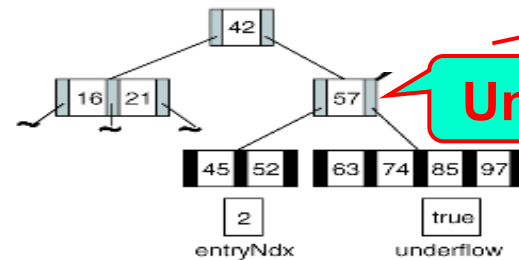
```

deleteMid
1 if (no rightmost subtree)
Leaf located. Replace data and delete leaf entry.
1 move predecessor's data to delete entry
2 set underflow if node entries less minimum
2 else
Not located. Traverse right to locate predecessor.
1 set underflow to deleteMid (node, entryNdx, ...)
...
3 end if
4 return underflow

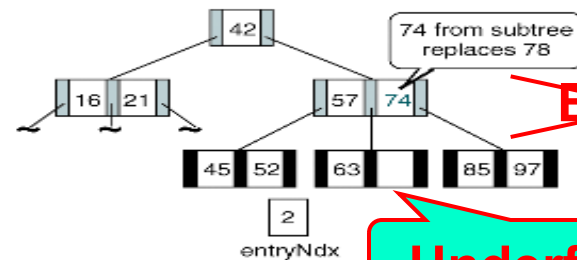
```



Combine
~~Borrow?~~



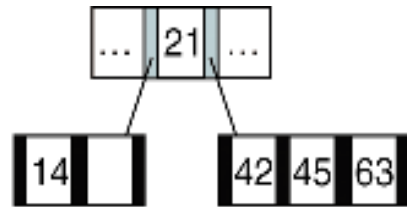
Combine



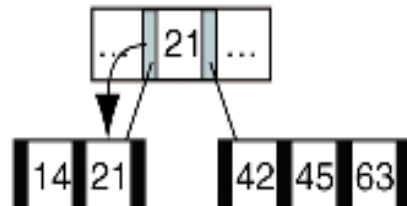
~~Borrow?~~

Underflow

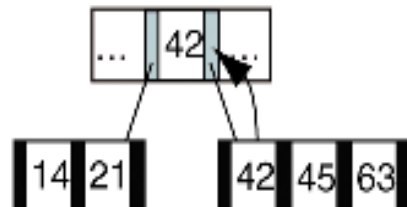
Restoring Order by Borrowing



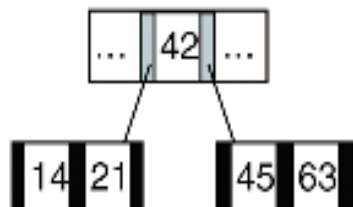
Original node



First rotate parent down

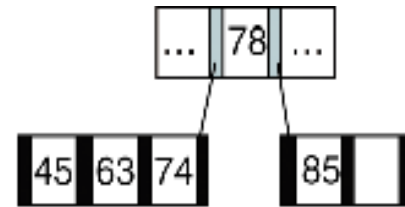


Then rotate data to parent

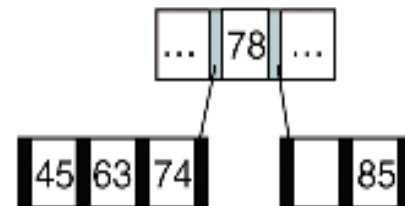


Finally, shift entries left

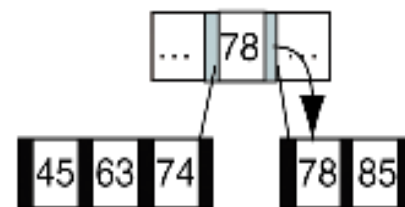
(a) Borrow from right



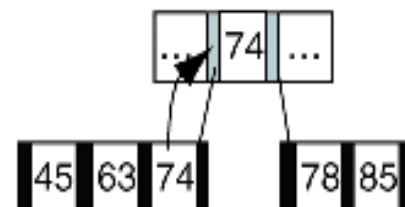
Original node



First shift data right



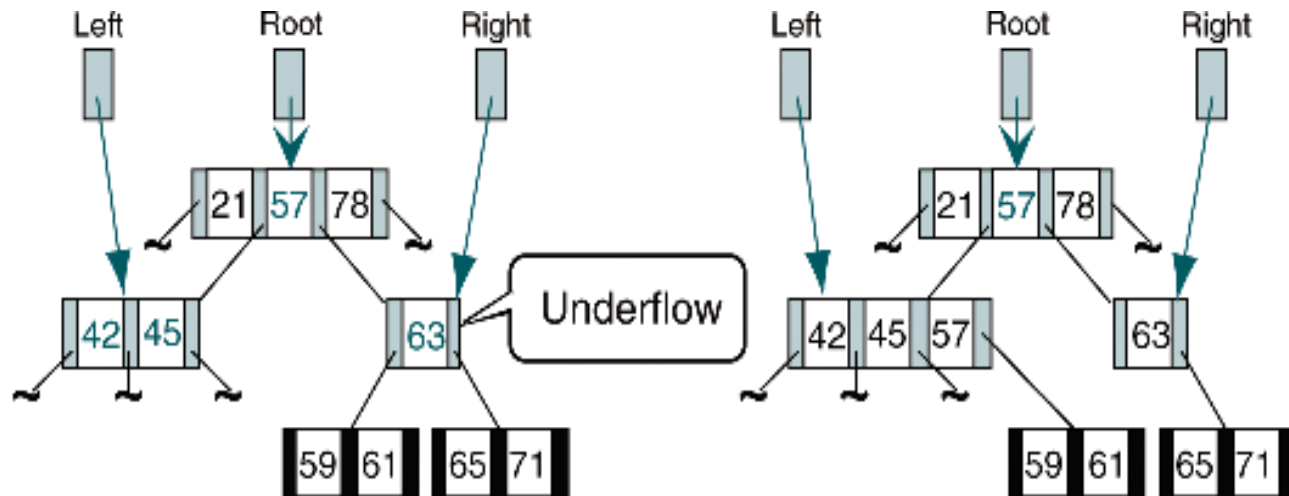
Then rotate parent data down



Then rotate data up

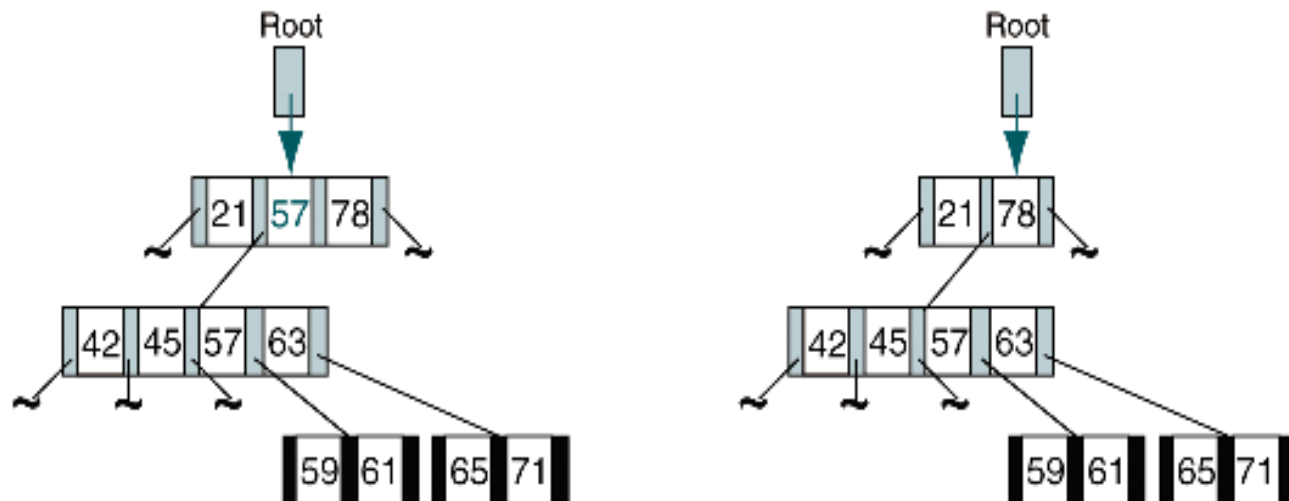
(b) Borrow from left

B-tree Combine



(a) After underflow

(b) After moving root to subtree

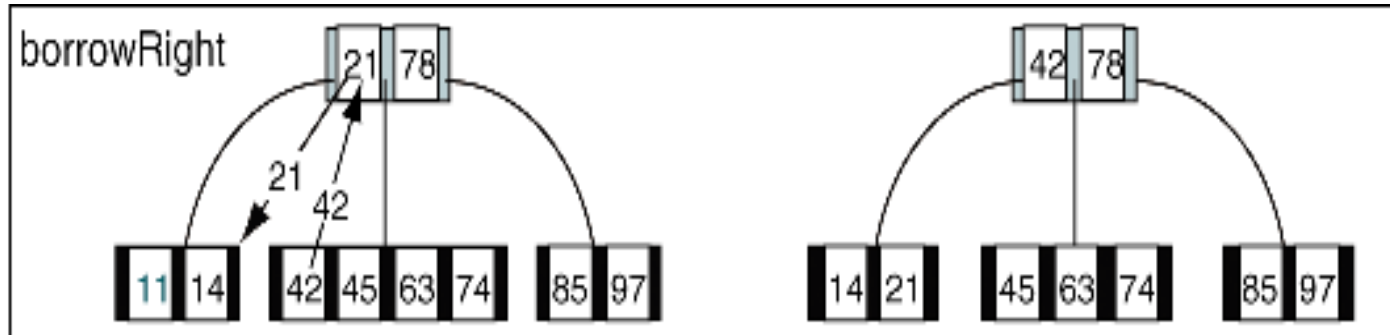


(c) After moving right entries

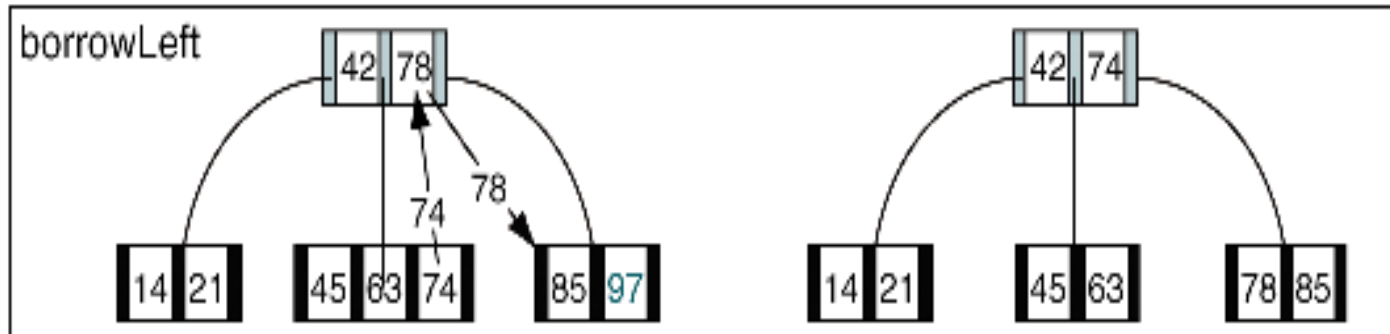
(d) After shifting root

B-tree Deletion Summary

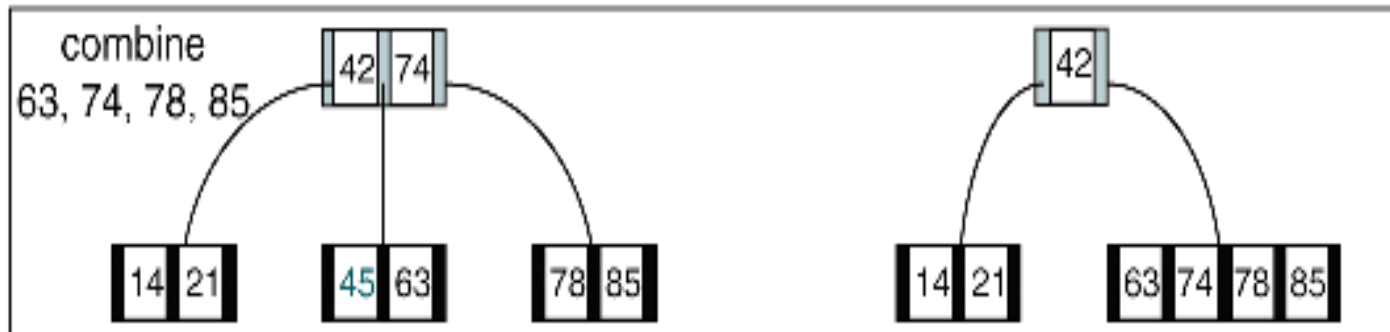
(a) Delete 11



(b) Delete 97



(c) Delete 45

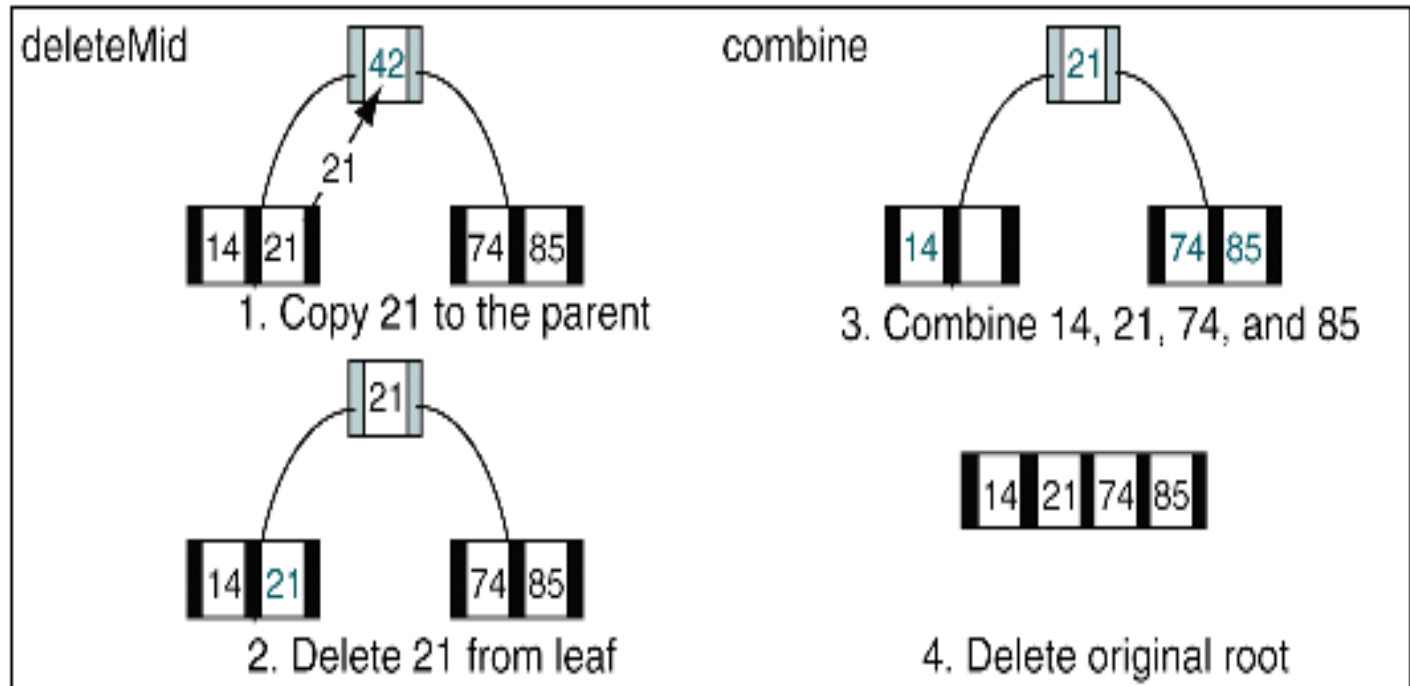


B-tree Deletion Summary (cont.)

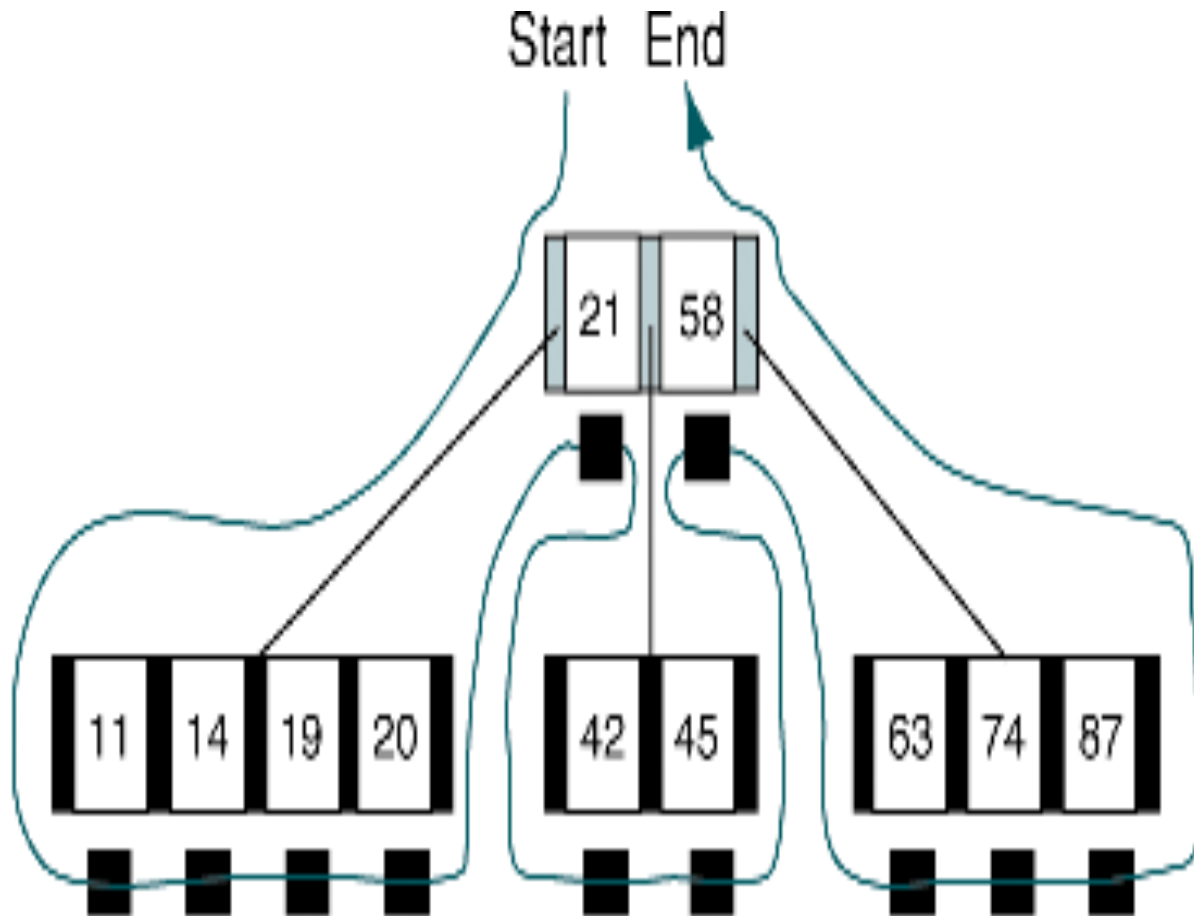
(d) Delete 63 and 78



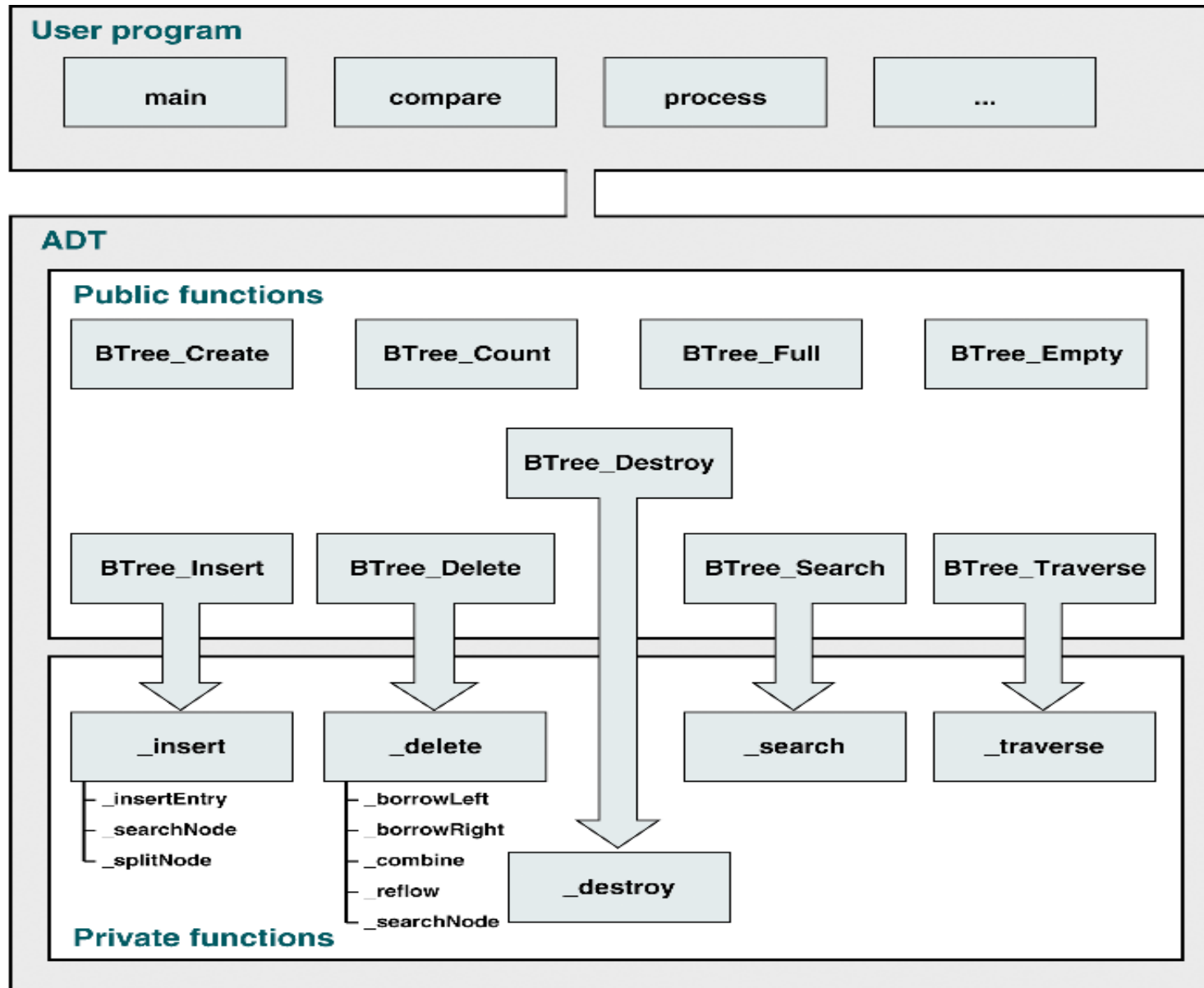
(e) Delete 42



Basic B-tree Traversal



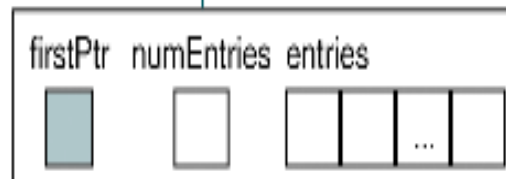
10-3 B-tree ADT



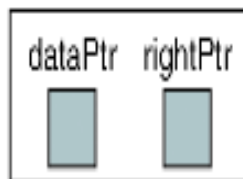
B-tree Data Structure



NODE



ENTRY



```
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* arg1, void* arg2);
} BTREE;
```


B-tree Declaration

```
1  /* ===== B-Tree.h =====
2      This header file contains the functions for the AVL
3      Tree abstract data type.
4          Written by:
5          Date:
6  */
7  #include <stdlib.h>
8  #include <stdbool.h>
9
10 // ===== CONSTANTS & MACROS =====
11 const int ORDER = 5;
12 const int MIN_ENTRIES = (((ORDER + 1) / 2) - 1);
13
14 // ===== STRUCTURES =====
15 struct node;
16
17 typedef struct
18 {
19     void*          dataPtr;
20     struct node*   rightPtr;
21 } ENTRY;
22
23 typedef struct node
```

B-tree Declaration (cont.)

```
24     {
25         struct node* firstPtr;
26         int          numEntries;
27         ENTRY        entries[ORDER - 1];
28     } NODE;
29
30 typedef struct
31 {
32     int    count;
33     NODE* root;
34     int    (*compare) (void* argu1, void* argu2);
35 } BTREE;
36
37 // ===== Prototype Declarations =====
38
39 // User interfaces
40 BTREE* BTree_Create
41     (int    (*compare) (void* argu1, void* argu2));
42 void    BTree_Traverse
43     (BTREE* tree, void (*process) (void* dataPtr));
44 BTREE* BTree_Destroy (BTREE* tree);
45 void    BTree_Insert (BTREE* tree, void* dataInPtr);
46 bool    BTree_Delete (BTREE* tree, void* dltKey);
47 void*    BTree_Search (BTREE* tree, void* dataPtr);
48 bool    BTree_Empty (BTREE* tree);
49 bool    BTree_Full (BTREE* tree);
50 int      BTree_Count (BTREE* tree);
```

B-tree Declaration (cont.)

```
51
52 // Internal BTree functions
53 static void* _search
54         (BTREE* tree, void* targetPtr,
55         NODE* root);
56 static int _searchNode
57         (BTREE* tree, NODE* nodePtr,
58         void* target);
59 static bool _delete
60         (BTREE* tree, NODE* root,
61         void* dltKeyPtr, bool* success);
62 static bool _insert
63         (BTREE* tree, NODE* root,
64         void* dataInPtr, ENTRY* upEntry);
65 static void _traverse
66         (NODE* root,
67         void (*process)(void* dataPtr));
68 static void _splitNode
69         (NODE* root, int entryNdx,
70         int compResult, ENTRY* upEntry);
71 static void _insertEntry
```

B-tree Declaration (cont.)

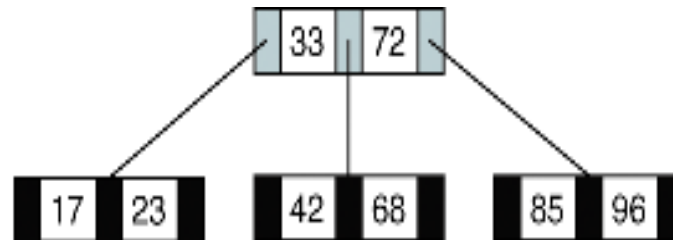
```
72         (NODE*  root, int entryNdx,  
73         ENTRY  upEntry);  
74 static bool _deleteEntry  
75         (NODE*  node, int entryNdx);  
76 static bool _deleteMid  
77         (NODE*  root, int entryNdx,  
78         NODE*  leftPtr);  
79 static bool _reFlow  
80         (NODE*  root, int entryNdx);  
81 static void _borrowLeft  
82         (NODE*  root,      int  entryNdx,  
83         NODE*  leftTree, NODE* rightTree);  
84 static void _borrowRight  
85         (NODE*  root,      int  entryNdx,  
86         NODE*  leftTree, NODE* rightTree);  
87 static void _combine  
88         (NODE*  root,      int  entryNdx,  
89         NODE*  leftTree,  NODE* rightTree);  
90 static void _destroy (NODE* root);
```

10-4 Simplified B-tree

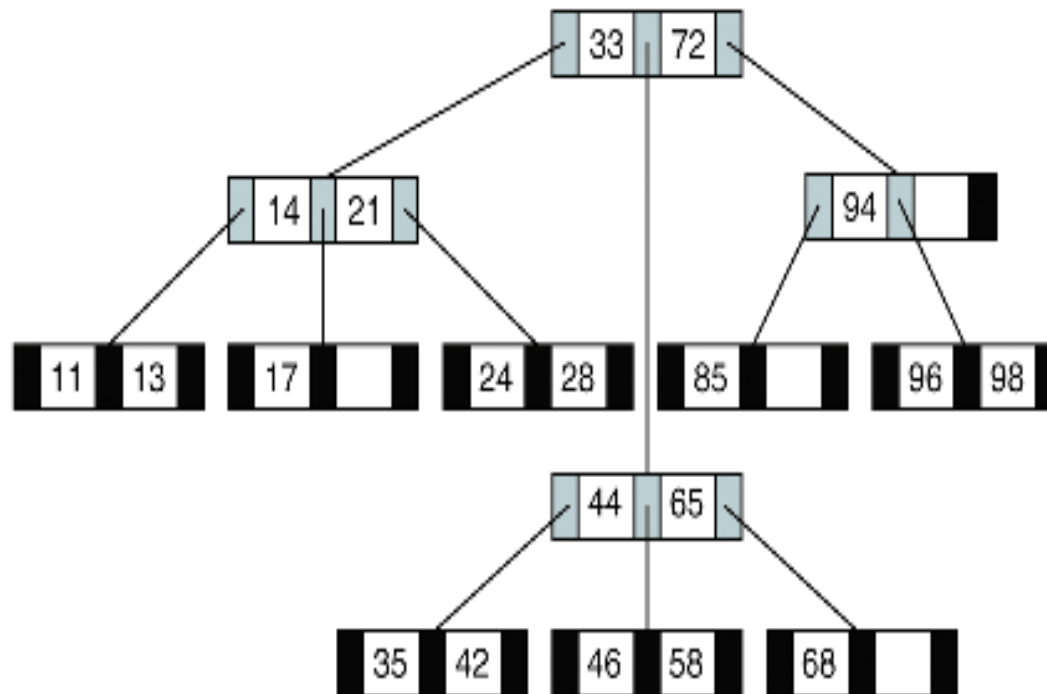
This section discusses 2 specialized B-tree which have been assigned unique names by computer scientists

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

2-3 Trees

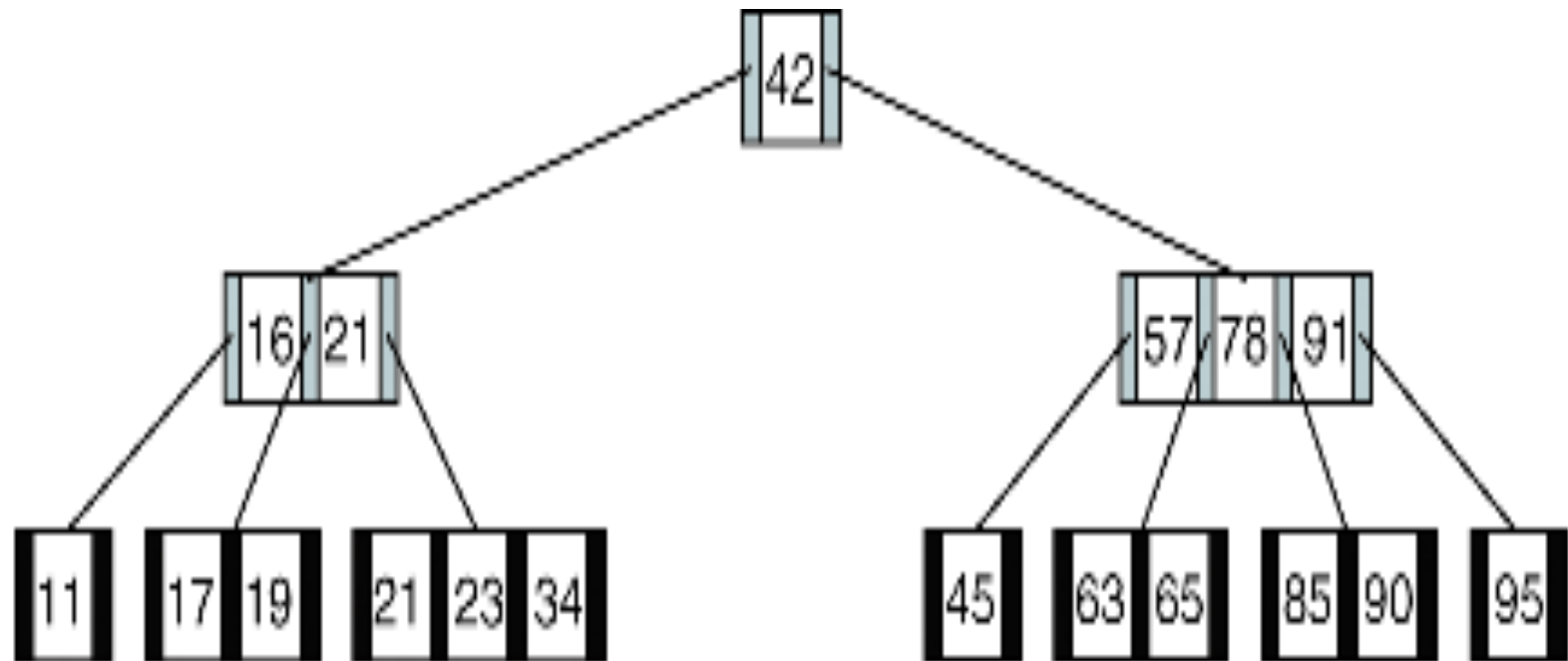


(a) Complete 2-3 tree



(b) 2-3 tree with empty entries

2-3-4 Tree

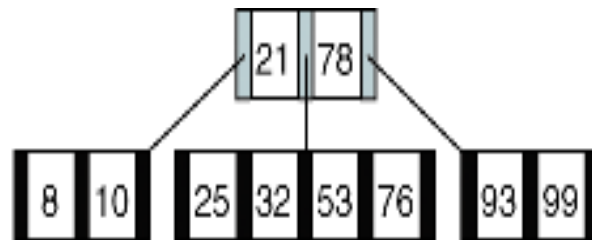


10-5 B-tree Variations

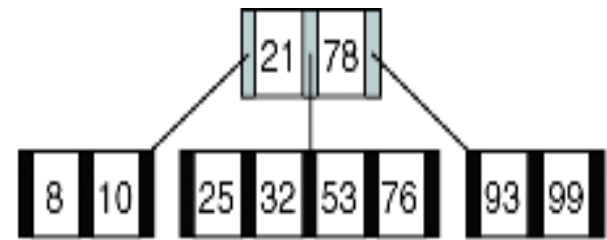
This section discusses two popular variations on the B-tree

- B* tree
- B+ tree

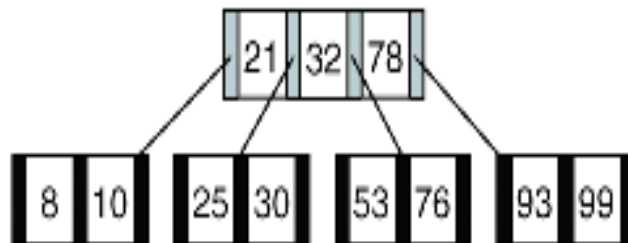
B* tree Insertion



Before adding 30

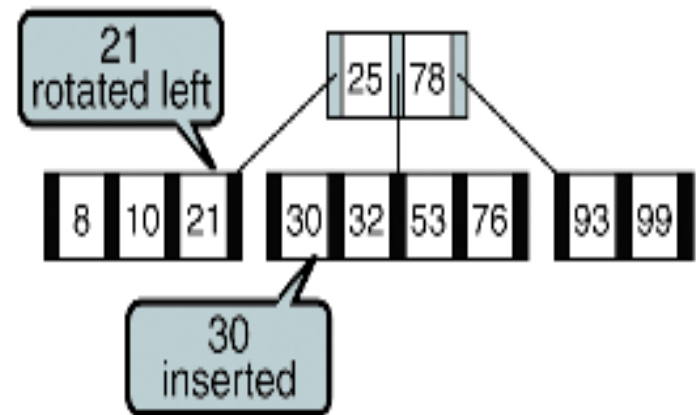


Before adding 30



Result after overflow

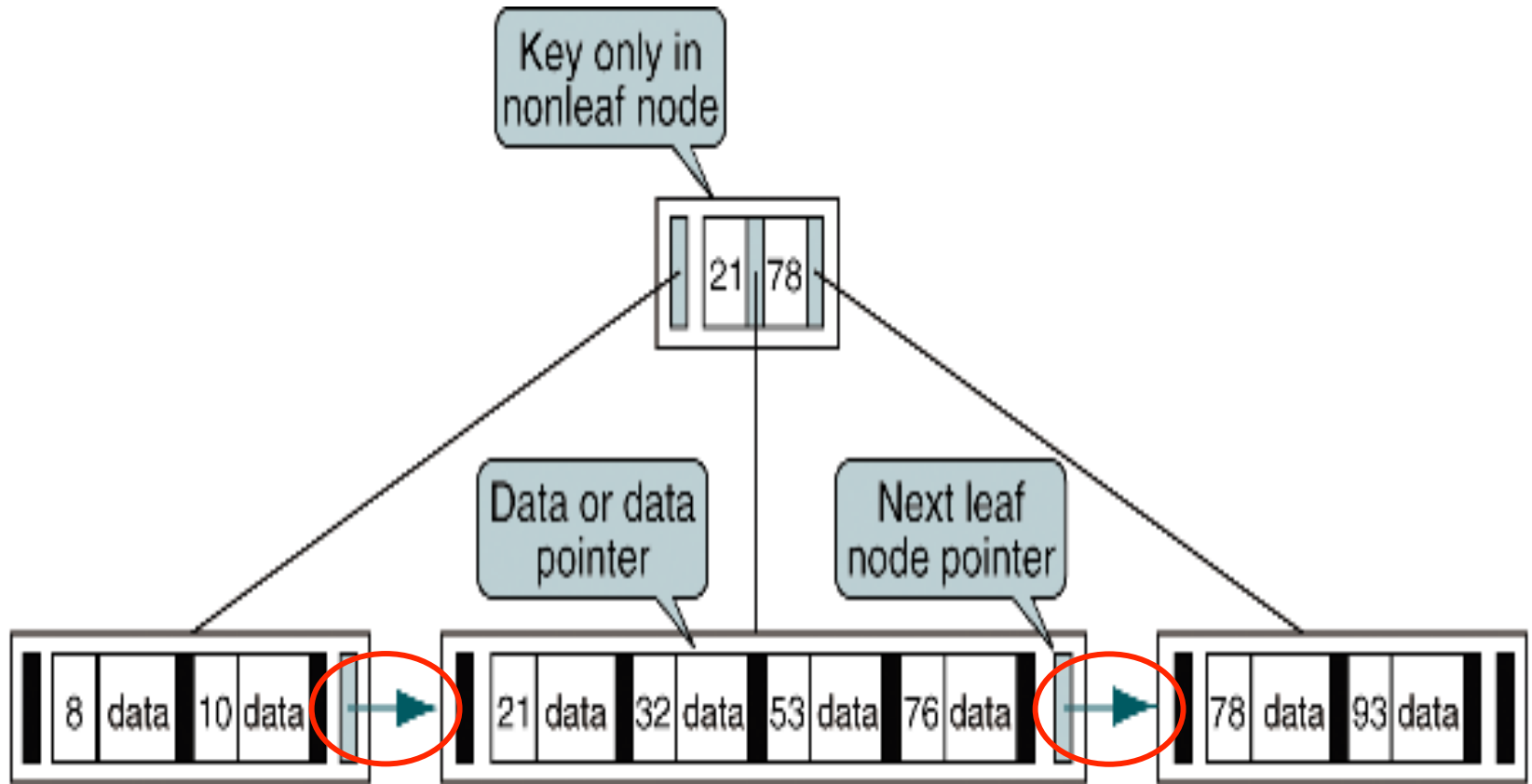
(a) Insertion into B-tree of order 5



Result after redistribution

(b) Insertion into B*-tree of order 5

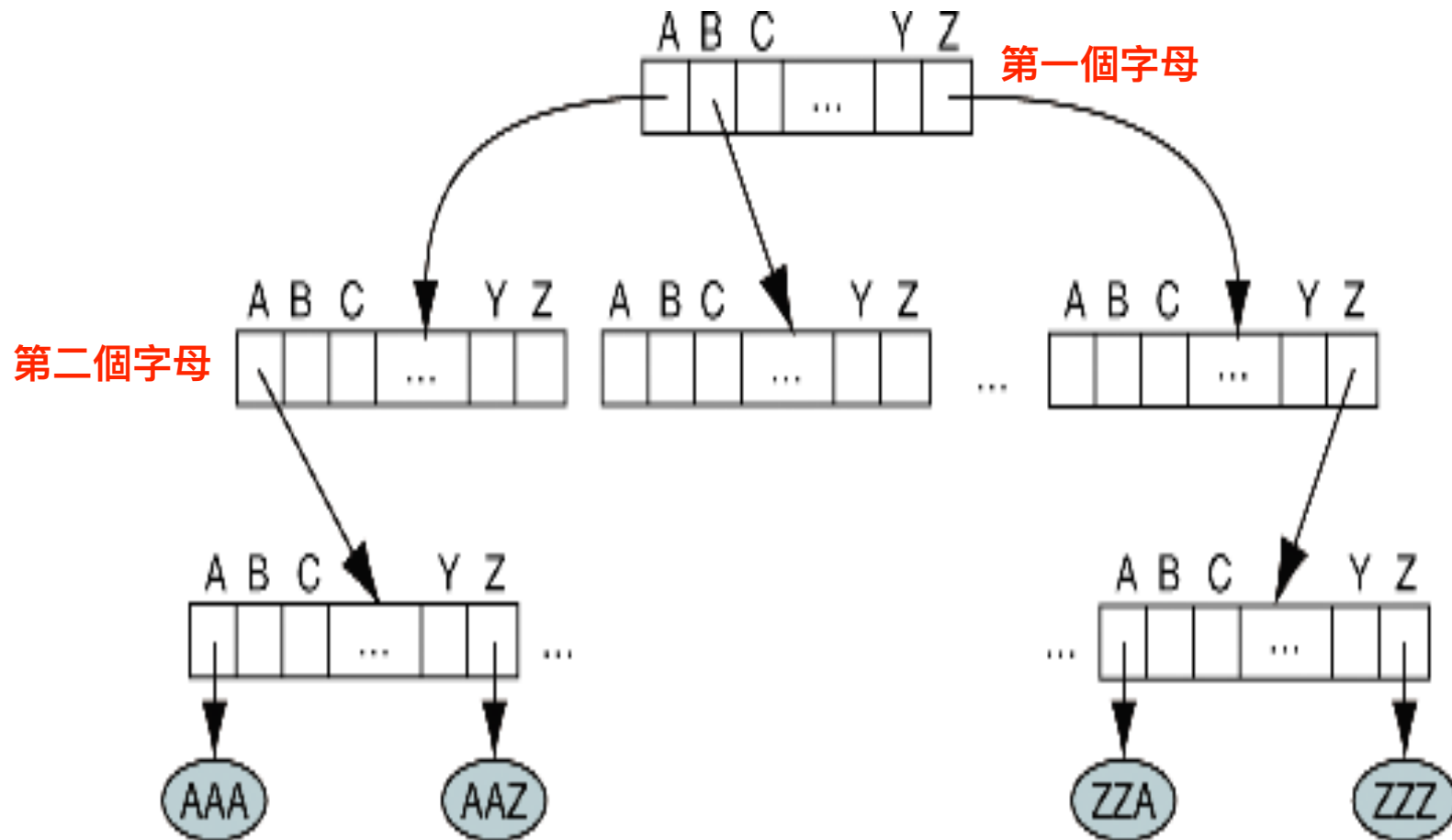
B+ tree 結合了linear list跟tree的優點



leaf node才有data

10-6 Lexical Search Tree

字典可能會用這種結構



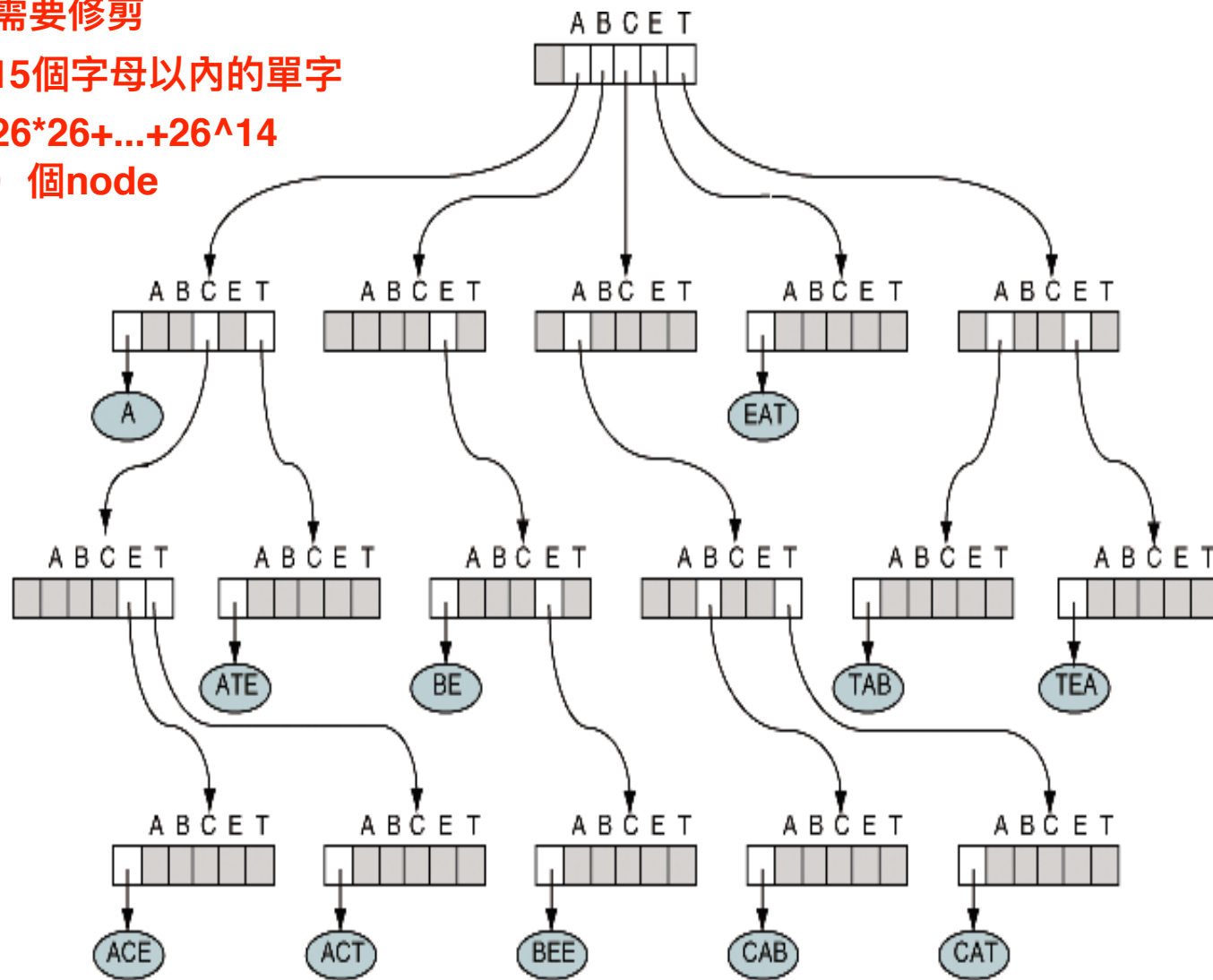
Spell Checker Trie

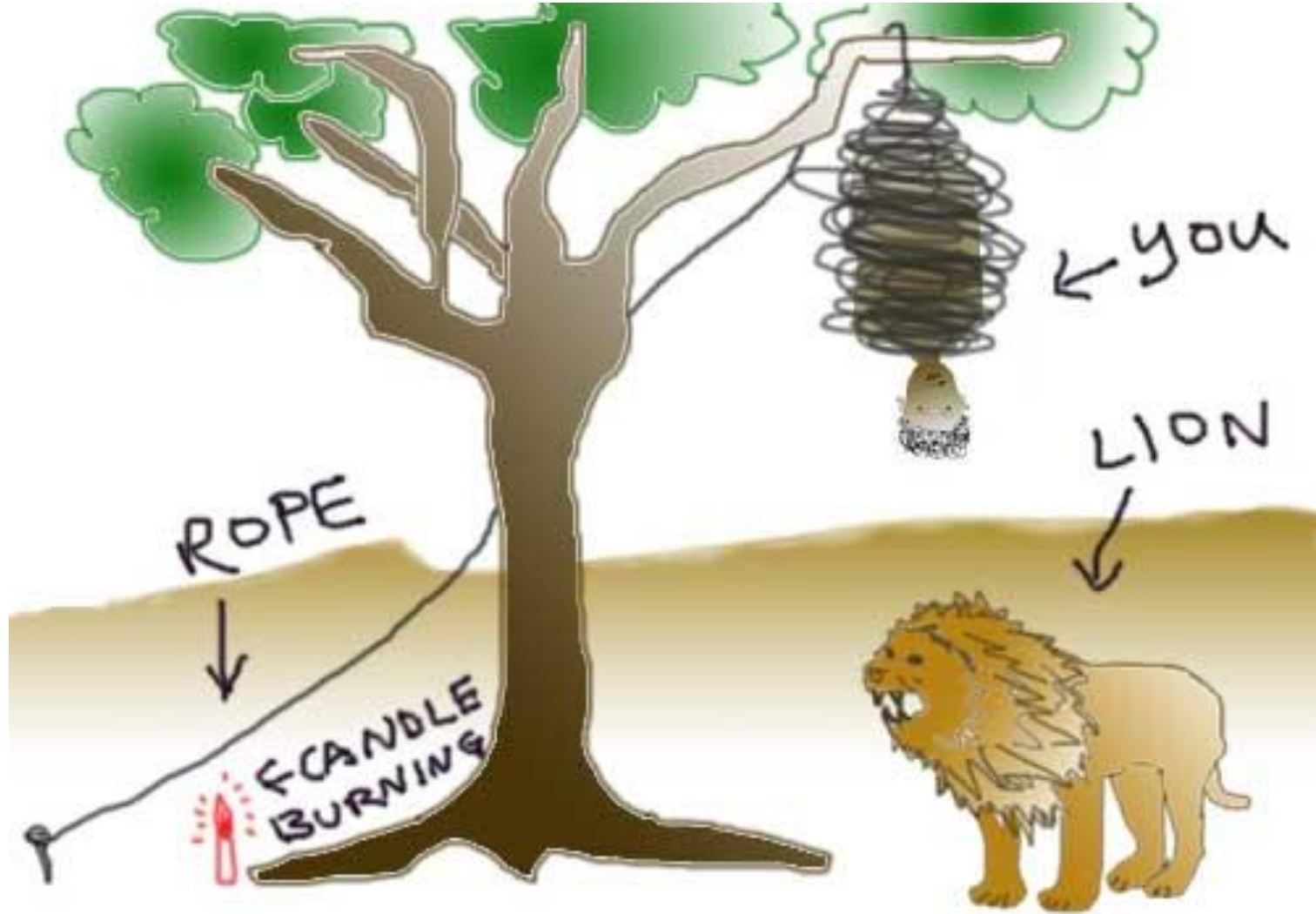
實務上可能需要修剪

要儲存所有15個字母以內的單字

需要 $1+26+26*26+\dots+26^{14}$

(約 10^{19}) 個node









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.



