Chapter 10 Index

Content

10.1 Some Basic Concept

Indexing Goals

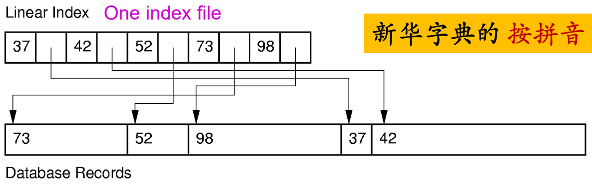
1. Organizing large databases (files)
2. Support multiple keys search
3. Support efficient insert, delete, and range queries

Basic terminology

- **Index file**: storing **key/pointer pairs.**
  - ✓ pointer point to actual records .
  - ✓ Could be organized with a linear data structure — Linear Indexing
  - ✓ Could be organized with a non-linear data structure such as a tree. — Tree Indexing
- **Primary Key**: A **unique** identifier for records.
- **Secondary Key**: An alternate search key, often **not unique** for each record. Often used for **search key**.

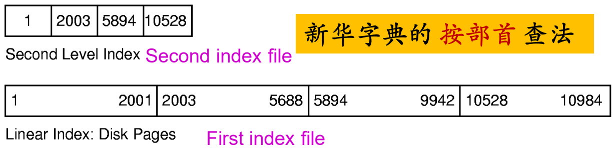
10.2 Linear Indexing

Linear index: Index file organized as a simple sequence of key/record pointer pairs with key values are in sorted order.



新华字典的 按拼音 查法

If the index is too large to fit in main memory, a second-level index might be used. Two index files



新华字典的 按部首 查法

5

Linear indexing

- **Good for indexing an entry sequenced database.**
- **Good for searching variable-length records**
- **Efficient when the database is static**
- **Poor for frequently insertion/deletion**

6

10.3 Tree Indexing

- **Tree indexing can efficiently support all desired operations:**
  - **Frequently Insert/delete**
  - **Search by one or combination of several keys**
  - **Key range search**
- **BST** 15,80, 23, 45, 30
  - **may be unbalanced**
  - **storing tree on disk based BFS, path from root to leaf would cover many disk page**
- **2-3 tree**
- **B-tree/B$^+$ tree**

子树的高度之差的绝对值不超过1

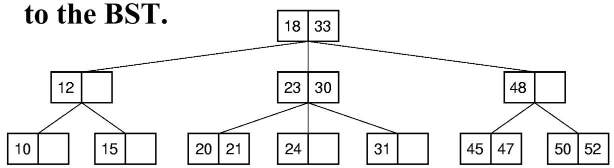
Balanced, Each path from root to leaf would cover few disk pages

7

10.3.1 2-3 Tree

1) **The 2-3 Tree has the following shape properties:**
   a) **A node contains one or two key(/pointer pairs)**
   b) **Every internal node has either two children (if it contains one key) or three children (if it contains two key).**
   c) **All leaves are at the same level in the tree, so the tree is always height balanced.**

2) **The 2-3 Tree has a search tree property analogous to the BST.**

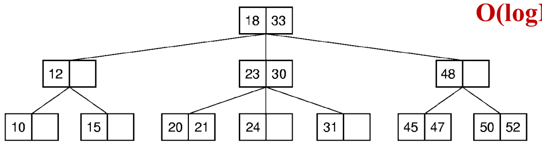


8

10.3.1 2-3 Tree Search

1. Start from the root, search the keys in current node. If search key is found, then return key/record pointer. If current node is a leaf node and key is not found, then report an unsuccessful search.
2. Otherwise, follow the proper branch and repeat the search process.

O(logN)



9

9

10.3.1 2-3 Tree Insertion

插入原则：插入新记录后依然为**2-3 Tree**

- Split-and-promote
- Unlike BST, the 2-3 tree does not grow downward, but grows upward

Root is splited, a new level added

1. find the proper *leaf* node L
2. if L contain only one value, insert the new key into L

else

1) split L into two nodes L and L', L contain the least of the three keys, L' contain the greatest of the three, Split(分裂)

2个是L中原有的，1个是待插入的

2) the middle key is passed up to the parent node alone with a pointer to L' Promotion(晋级)

3) the promoted key is then inserted into the parent.

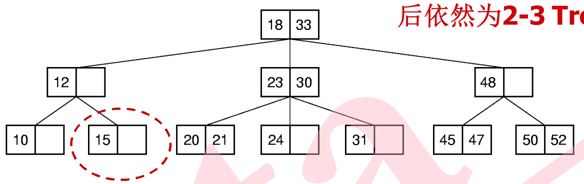
a) if the parent contain only one value, then the promoted key and the pointer to L' are simply added to the parent node,

b) if the parent is full, then the split and promote process is repeated

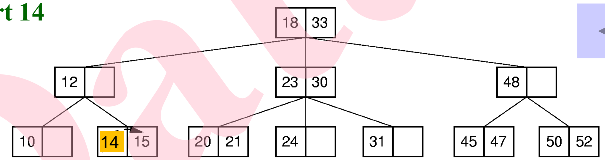
10

10.3.1 2-3 Tree Insertion example（1）

插入原则：插入新记录后依然为**2-3 Tree**



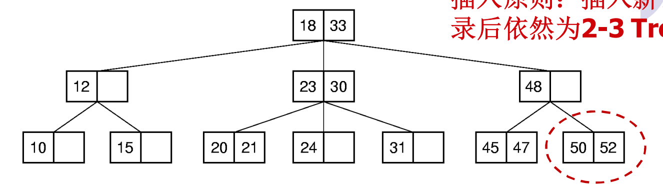
Insert 14



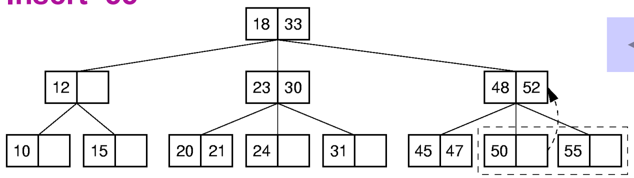
11

10.3.1 2-3 Tree Insertion example (2)

插入原则：插入新记录后依然为**2-3 Tree**

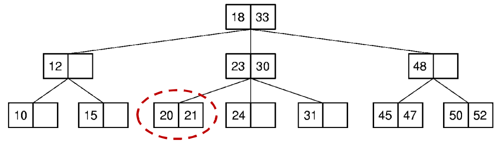


Insert 55

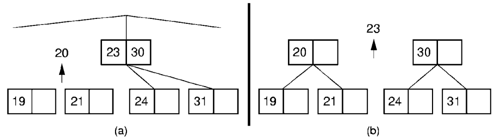


12

10.3.1 2-3 Tree Insertion example (3)

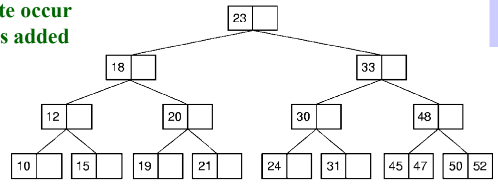


Insert 19



(a) (b)

If split and promote occur at root, new level is added



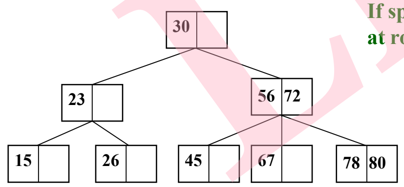
(c)

13

10.3.1 2-3 Tree Insertion example (4)

给定序列： **15,80, 23, 45, 56, 67,30,26,72,78**

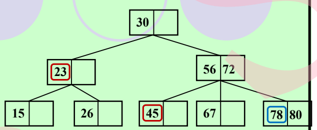
按顺序插入到 **2-3 tree**



If split and promote occur at root, new level is added

14

Delete a key from 2-3 Tree



Locate the node that contains the key

Case1

Delete a key from a leaf node containing two value

remove the key

Case 2

Delete a key from a leaf node containing only one value

Case 3

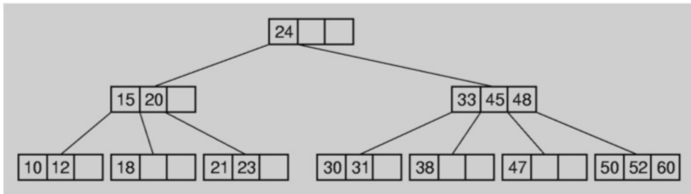
Delete a key from a internal node

Deleted key is replaced with another that can take its place while maintain the correct order

15

10.3.2 B-Trees

The B-Tree is an extension of the 2-3 Tree.



The B-Tree is attributed to R Bayer and E. McCreight in 1971, now is the standard file organization for applications requiring insertion, deletion, and key range searches

16

B-Trees Definition

1) A B-Tree of order *m(m 阶)* has following shape properties:

- The root is either a leaf or has at least two children（one key/pointer pair).
- Each internal node, except for the root, has ⌈*m*/2⌉~ *m* children; has ⌈*m*/2⌉-1 ~ *m-1* key/pointer pairs
- All leaves are at the same level in the tree, so the tree is always height balanced.

2) A B-Tree has search tree(BST) property

3) A B-Tree node size (m-1) is usually selected to match the size of a disk block.

- A B-Tree node could have hundreds of children.

2-3树实际就是3阶B树

17

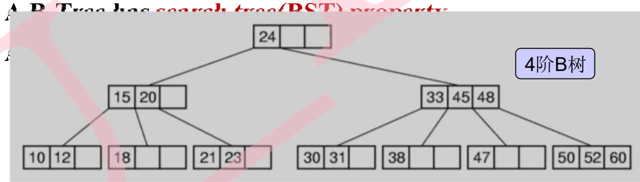
B-Trees Definition

1) A B-Tree of order *m(m 阶)* has following shape properties:

- The root is either a leaf or has at least two children (one key/pointer pair)
- Each internal node, except for the root, has ⌈*m*/2⌉~ *m* children; has ⌈*m*/2⌉-1 ~ *m-1* key values/pointer pairs
- All leaves are at the same level in the tree, so the tree is always height balanced.

2) A B-Tree has search tree(BST) property

3)



18

B-Trees property

1. B-Trees are always balanced.
2. B-Trees keep records with similar-key together on a disk page, which takes advantage of locality of reference.
3. B-Trees guarantee that every node(except root) in the tree will be almost half-full(50%). This improves space efficiency while reducing the typical number of disk access necessary during a search or update operation.

⌈*m*/2⌉-1 ~ *m-1*

19

B树是一种平衡的 多路搜索 树

在 *m* 阶的B树上，

- 根结点有2~ *m* 个子树，有 1~ *m-1* 个关键字，
- 其余内部结点有⌈m/2⌉ ~ *m* 个子树，有⌈m/2⌉-1 ~ *m-1*个关键字；
- 叶子结点有⌈m/2⌉-1 ~ *m-1*个关键字

多叉树的特性

- 结点中的多个关键字均自小至大有序排列，

  即： $K_1 < K_2 < \ldots < K_l$

- A_{i-1} 子树上所有关键字均小于K_i
- A_i 子树上所有关键字均大于等于K_i

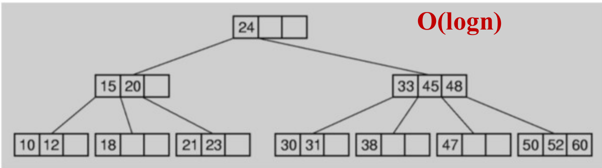
搜索树的特性

20

B-Trees Search

Search in a B-Tree is a generalization of search in a 2-3 Tree.

1. Start from root, do searching on keys in current node. If search key is found, then return record. If current node is a leaf node and key is not found, then report an unsuccessful search.
2. Otherwise, follow the proper branch and repeat the process.



21

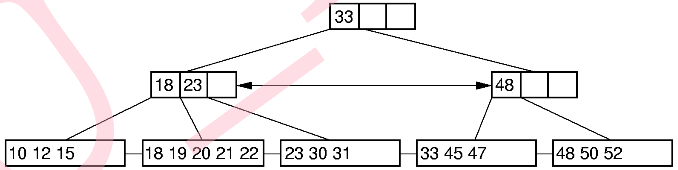
10.3.3 B^+ Trees

The most commonly implemented form of the B-Tree is the B^+ Tree

1. Internal nodes of the B^+ Tree do not store pointers -- only keys to guild the search; Leaf nodes store keys/pointers to records.

   placeholders

2. A leaf node may store more or less values than internal node.



22

B^+树是B树的一种实现变型

m阶B^+树

- ◆ 根节点有2 ~ *m*个孩子，有 1 ~ *m-1* 个关键字；
- ◆ 除根以外的内部结点有⌈m/2⌉ ~ *m* 个孩子，有⌈m/2⌉-1 ~ *m-1* 个关键字；
- ◆ 叶子结点有⌈*n*/2⌉~*n* (*n*与 *m*可等可不等) 个关键字/记录指针对；
- ◆ 叶子结点彼此相链接构成一个有序链表，其头指针指向含最小关键字的结点
- ◆ 内部结点中只存关键字，记为K_1，K_2，…，其子树记为A_0，A_1,…，有下列关系：$Min(A_i) \geq K_i > \max(A_{i-1})$

B^+树需要两个参数m和n来初始化

23

B^+树查找

※ 在 B^+ 树上，既可以进行缩小范围的查找，也可以进行顺序查找(在叶子结点层查找)

※ 在进行缩小范围的查找时，给定值<K_i，则应继续在 A_{i-1} 子树中进行查找，给定值>=K_i，则应继续在 A_i 子树中进行查找，一直查到叶子结点

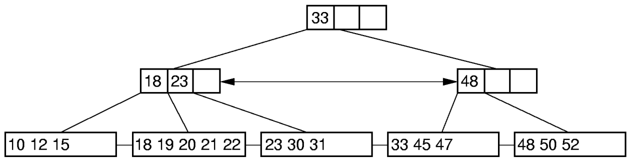
※ 在进行缩小范围的查找时，不管成功与否，都必须查到叶子结点才能结束

24

B+树查找

For example: 查找 30，23 和 68

O(log n)



25

B+ Tree Insertion

插入原则：插入新记录后依然为B+ Tree

23, 48 12 10 33 **50** 20 18 15 21 45 47 52 31 **30**

n=5, m=4

1. find the proper leaf node L
2. if L isn't full,
   insert the new key into L

   else
   1) split L into two (dividing the records evenly among the two nodes)
   2) promote a copy of the least-valued key in the newly formed right leaf node to the parent
   3) the promoted key is then inserted into the parent.
      a) if the parent contain isn't full, then the promoted key and the newly right leaf node are simply added to the parent node,
      b) if the parent is full, then the split the parent into two nodes and promote the least-valued key in the right node to the parent.

Why?

Split(分裂) and Promotion(晋级) process may repeated upward, perhaps eventually leading to splitting the root and causing a new level

26

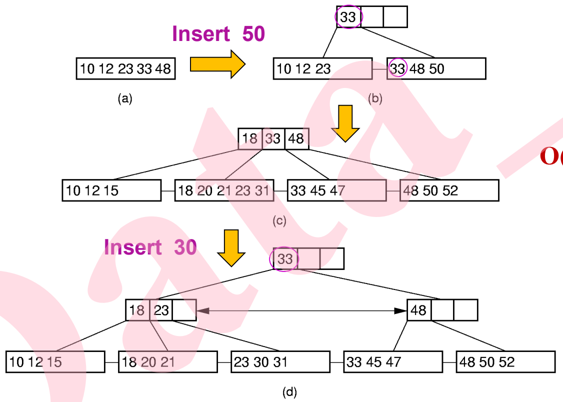
B+ Tree Insertion example

插入原则：插入新记录后依然为B+ Tree

23, 48 12 10 33 **50** 20 18 15 21 45 47 52 31 **30**

n=5
m=4

O(logn)



27

Deletion a record from B+ Tree

删除原则：删除某个记录后依然为B+ Tree

Locate the leaf L that contains the record, remove the record from L

Case1: L is more than half full ($k >= \lceil n/2 \rceil$)

do nothing

underflow

Case 2: L is less than half full ($k < \lceil n/2 \rceil$)

look at L's adjacent siblings to determine if they have a spare record to fill the gap to keep L half full at least

1) if so, transfer enough records from the sibling to L so that both nodes have the same number of records and modify the parent key;

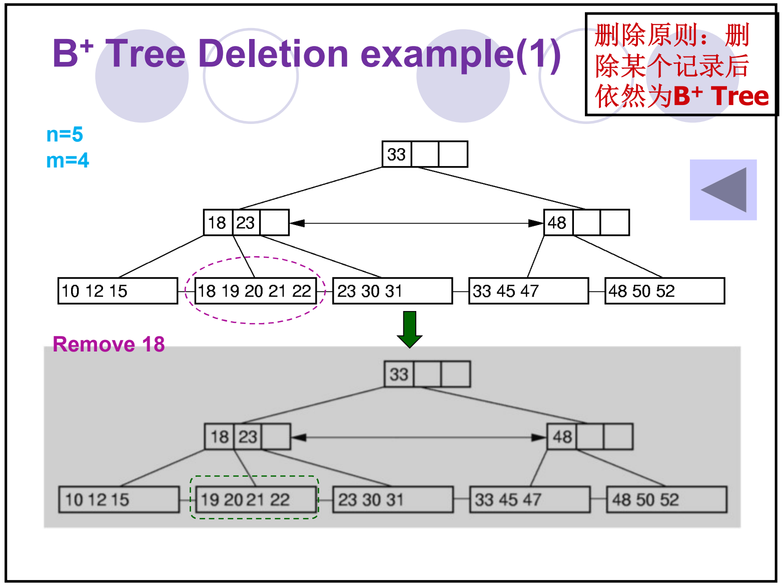
merge

2) else, L must give its records to a sibling and be removed from the tree, at the same time, delete the proper parent key. This merge process may cause the parent underflow($k < \lceil m/2 \rceil - 1$) in turn, if so, the transfer or merge process is repeated.

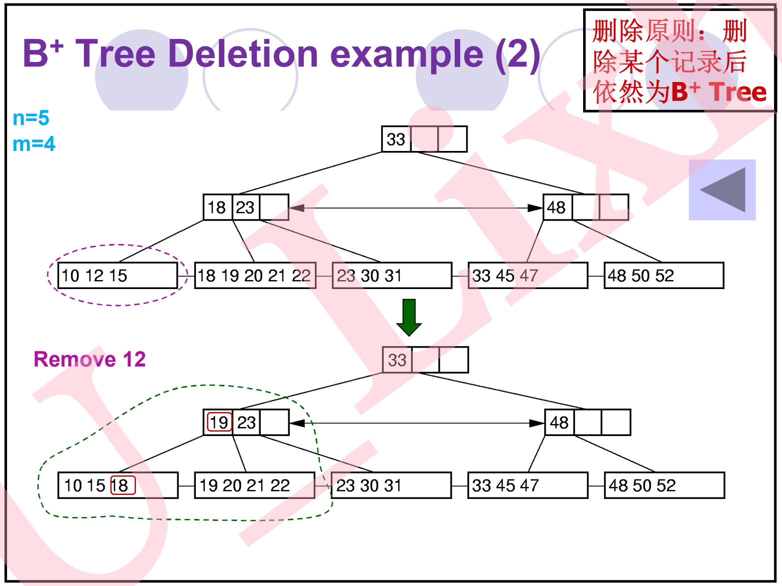
If the last two children of the root merge together, then the tree loses a level

Are there room to do this in sibling ???

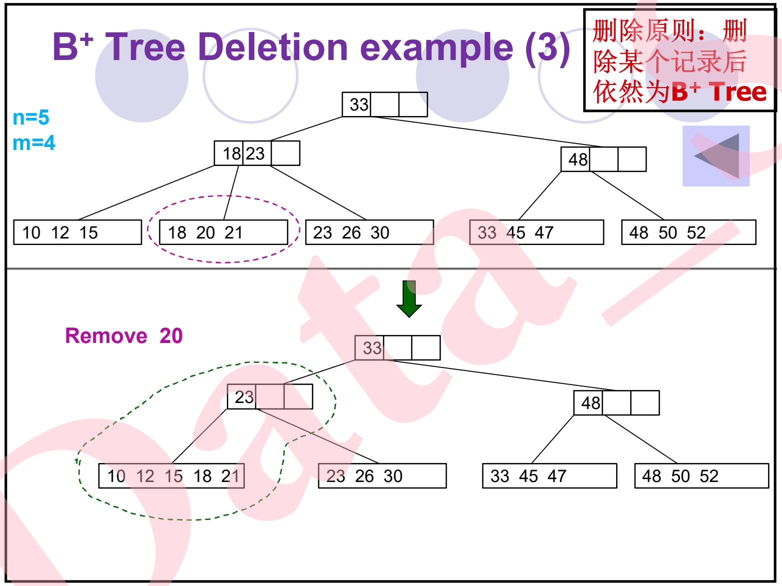
28


B+ Tree Deletion example(1)
删除原则：删除某个记录后依然为B+ Tree
n=5
m=4
33
18 23
48
10 12 15
18 19 20 21 22
23 30 31
33 45 47
48 50 52
Remove 18
33
18 23
48
10 12 15
19 20 21 22
23 30 31
33 45 47
48 50 52

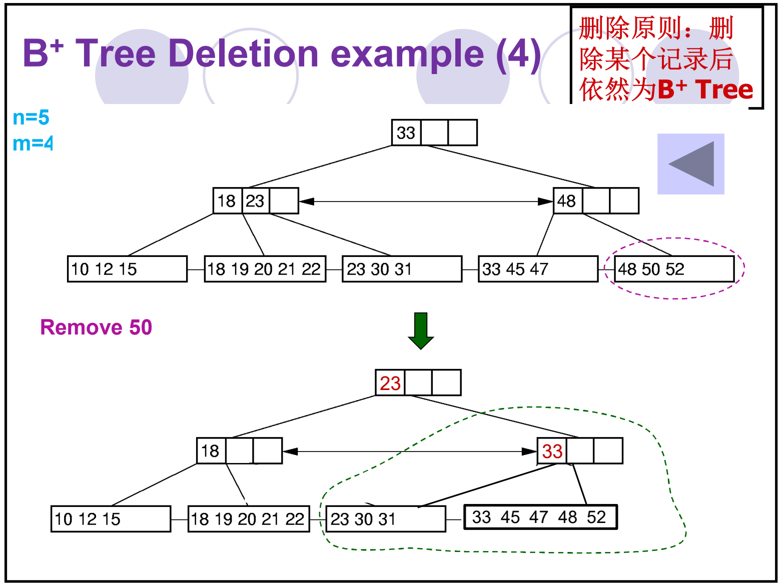
29


B+ Tree Deletion example (2)
删除原则：删除某个记录后依然为B+ Tree
n=5
m=4
33
18 23
48
10 12 15
18 19 20 21 22
23 30 31
33 45 47
48 50 52
Remove 12
33
19 23
48
10 15 18
19 20 21 22
23 30 31
33 45 47
48 50 52

30


B+ Tree Deletion example (3)
删除原则：删除某个记录后依然为B+ Tree
n=5
m=4
33
18 23
48
10 12 15
18 20 21
23 26 30
33 45 47
48 50 52
Remove 20
33
23
48
10 12 15 18 21
23 26 30
33 45 47
48 50 52

31

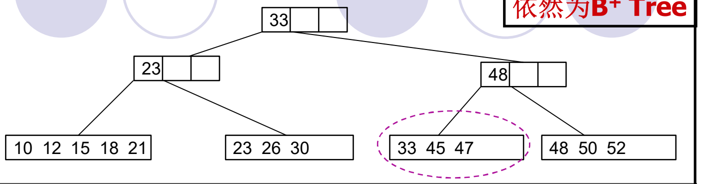

B+ Tree Deletion example (4)
删除原则：删除某个记录后依然为B+ Tree
n=5
m=4
33
18 23
48
10 12 15
18 19 20 21 22
23 30 31
33 45 47
48 50 52
Remove 50
23
18
33
10 12 15
18 19 20 21 22
23 30 31
33 45 47 48 52

32

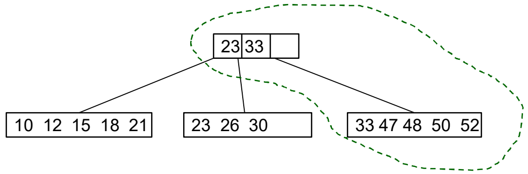
B+ Tree Deletion example (5)

删除原则：删除某个记录后依然为B+ Tree

n=5
m=4



Remove 45



33

10.3.4 B-Tree Time/Space Analysis

1. Asymptotic **time cost** of search, insertion, and deletion of records from B-tree, B+ Tree is $O(\log N)$.
   N: 结点个数

2. B-Trees and B+ Tree nodes(except root) are always **at least one half full**.

34

In this chapter , we study….

- **Linear index**
- **2-3 tree**
  - 定义，特点
  - **searching， insert**
- **B tree**
  - 定义，特点
  - **Searching**
- **B+树**
  - 定义，特点
  - **Seaching， insert， delete**

35

CHAPTER10 END

36