Data Structure

grayson

2022-07-13

1

0.0 B

1 B B+

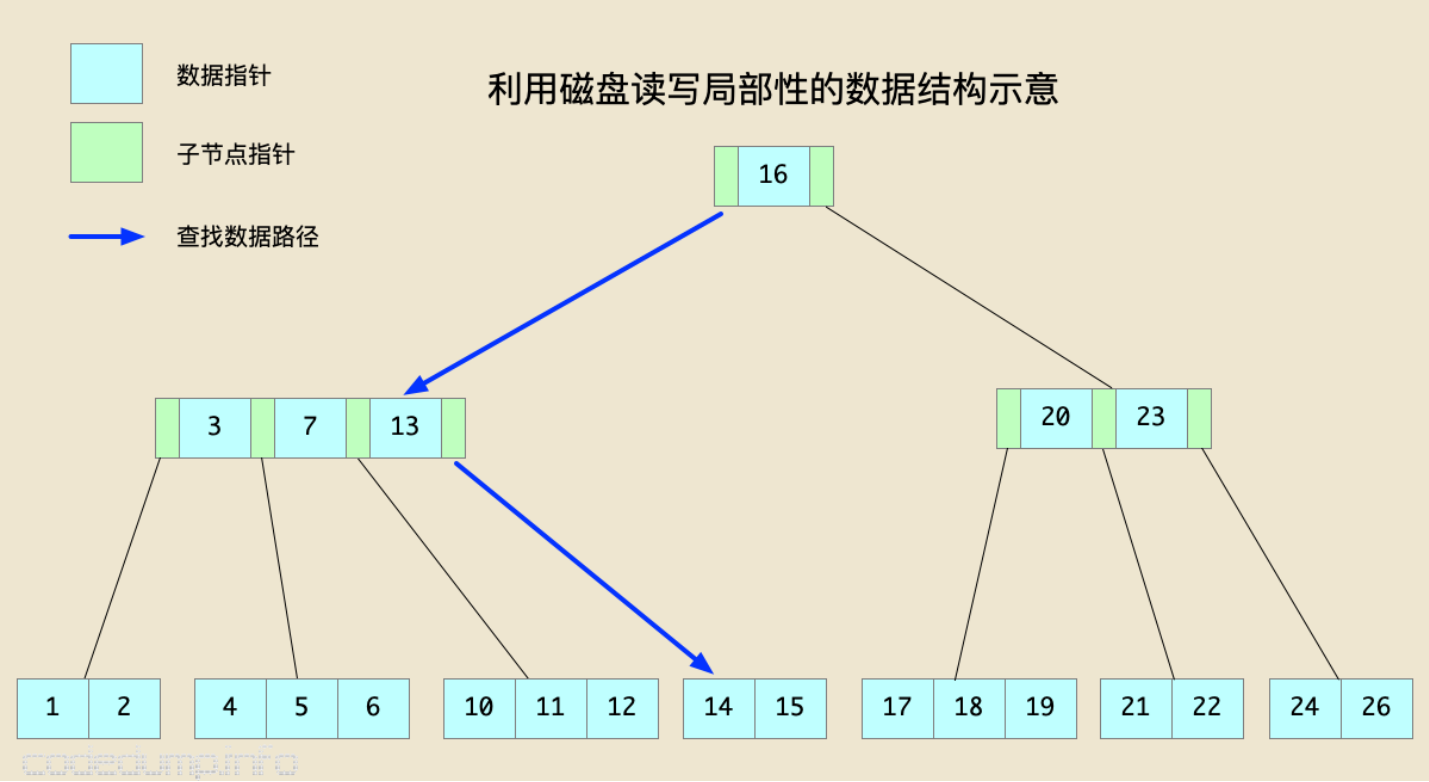
1. Binary Search Tree, BST

2.

3. BST

4. 14 [3,7,13]

   1.
   2.



5. B B+

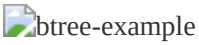
> MySQL
>
> 1.
>
> 1.
> 2. **IO** **IO**

2 B

2.1 B

1. B
    1. Internal Node
    2. Leaf Node
2. B+ **B+**
3. B degree t

4. B
    1.
    2.
    3. **+1**
    4.
    5. **$[t-1,2t-1]$**
        1. **$ t - 1 $**
        2. **$ 2t - 1 $**
5. B t 2 key value
    1. $t=2$ $[1,3]$
    2.
    3. 3 $[1,2]$ 3 $[4,5,6]$ 3



2.2 B

B B B
B

2.2.1

1. B
   B
2. **y** **$ 2t-1 $** **$t-1$**
   y **y**

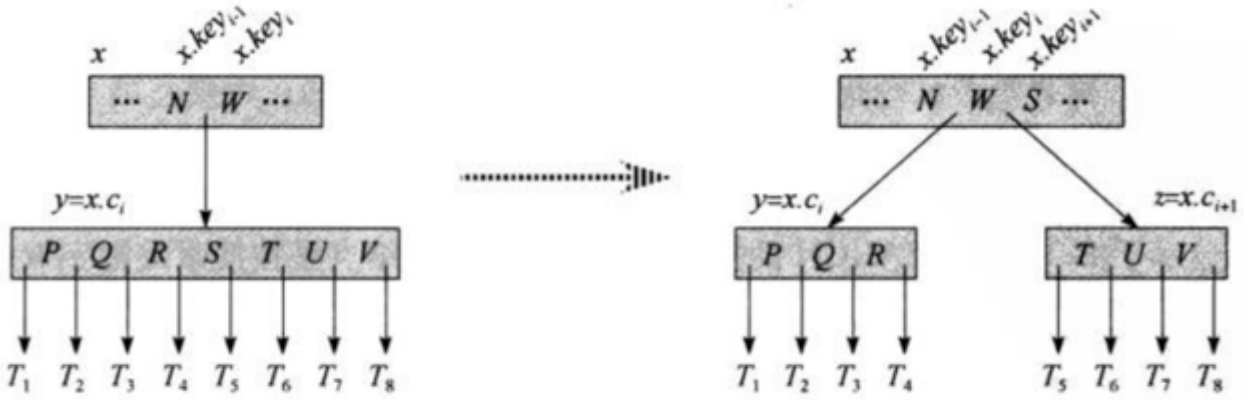


图 18-5　分裂一个 $t=4$ 的结点。结点 $y=x.c_i$ 分为两个结点 y 和 z，y 的中间关键字 S 被提升到 y 的父结点中

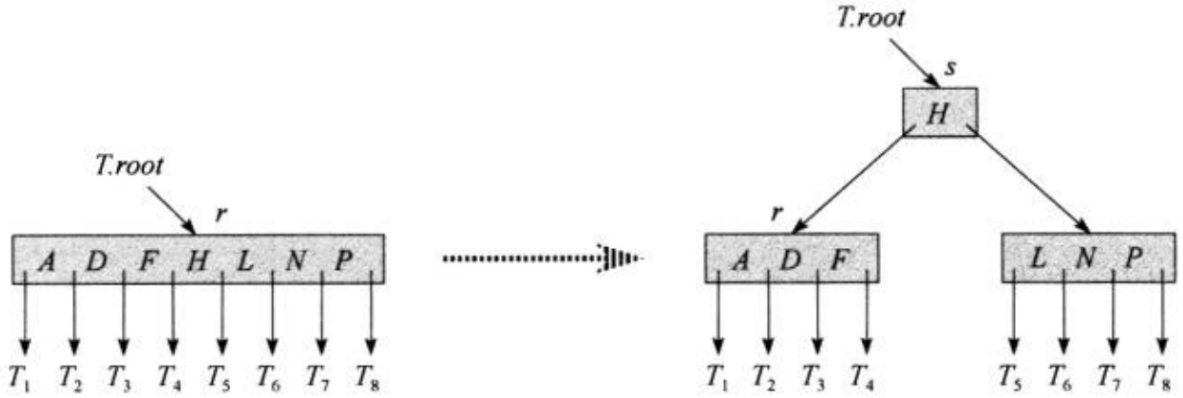


图 18-6　分裂 $t=4$ 的根。根结点 r 一分为二，并创建了一个新结点 s。新的根包含了 r 的中间关键字，且以 r 的两半作为孩子。当根被分裂时，B 树的高度增加 1

3.

y

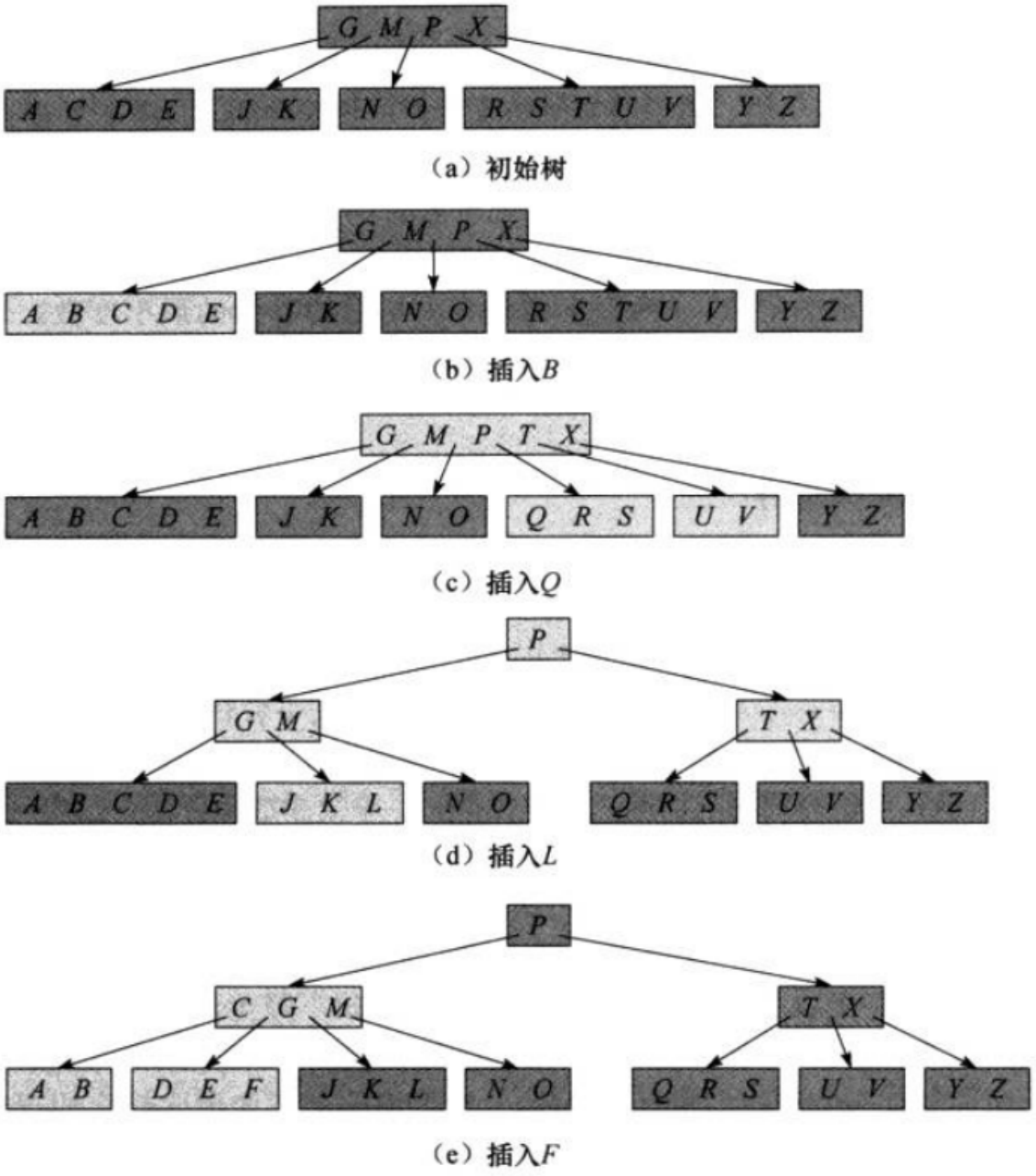


图 18-7　向 B 树中插入关键字。这棵 B 树的最小度数 t 为 3，所以一个结点至多可包含 5 个关键字。在插入过程中被修改的结点由浅阴影标记。(a)这个例子初始时的树。(b)向初始树中插入 B 后的结果；这是一个对叶结点的简单插入。(c)将 Q 插入前一棵树中的结果。结点 $RSTUV$ 被分裂为两个分别包含 RS 和 UV 的结点，关键字 T 被提升到根中，Q 被插入两半的最左边(RS 结点)。(d)将 L 插入前一棵树中的结果。由于根结点是满的，所以它立即被分裂，同时 B 树的高度增加 1。然后 L 被插入包含 JK 的叶结点中。(e)将 F 插入前一棵树中的结果。在将 F 插入两半的最右边(DE 结点)之前，结点 $ABCDE$ 会进行分裂

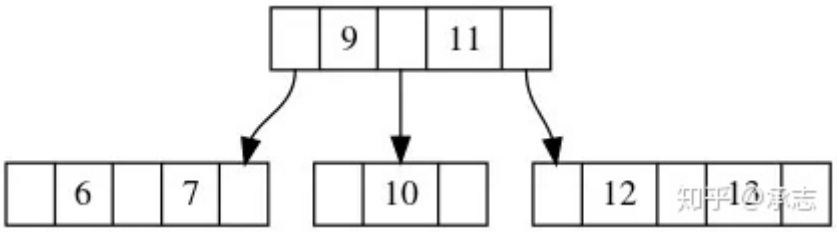
2.2.1

2.2.1.1

1.

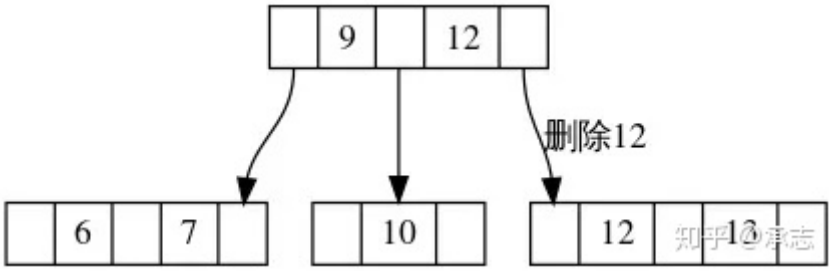
2.

1. 11 11



2. 11

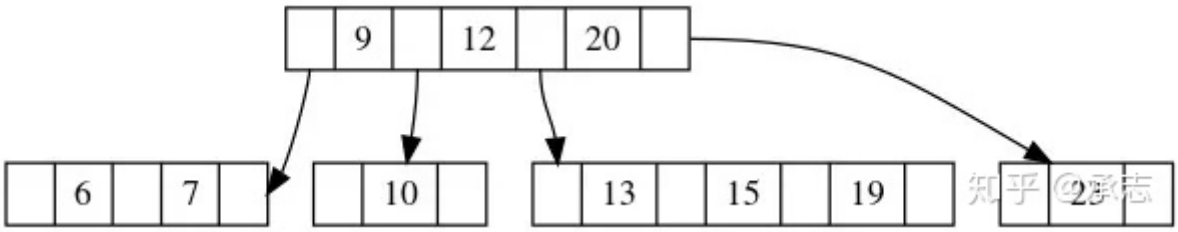
11 12 12 11 12



3.

2.2.1.2

1.
2.
   1. $t=2$ $[1,3]$ 4
   2. 19 3
   3. 10 10

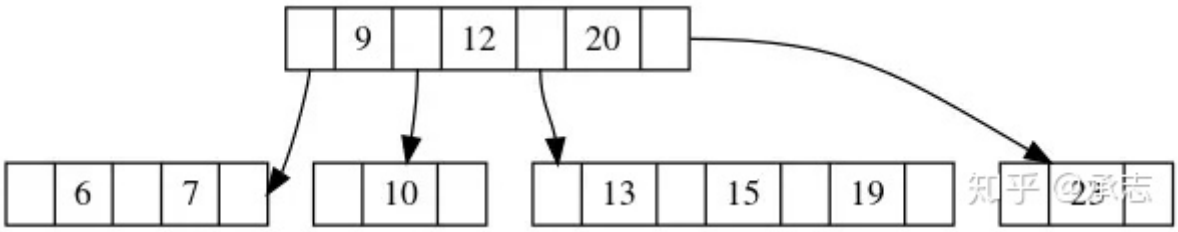


2.2.1.3

1.
2. B

3.
   1. 10 10

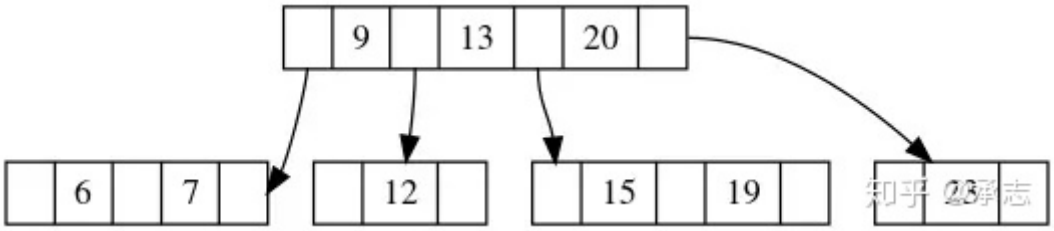
2. 3



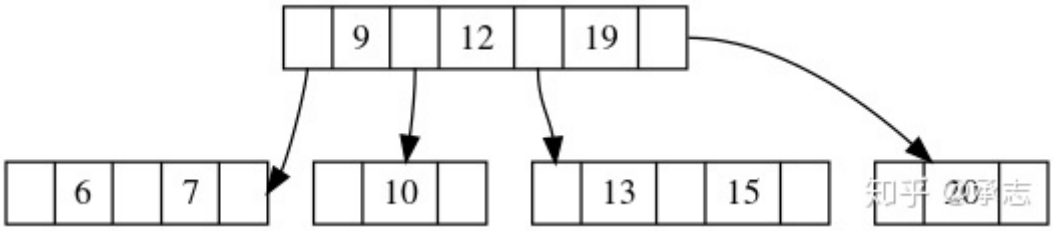
3. 13 13

12 13 10 **12** **10** **13** **12**

4.



5. 23



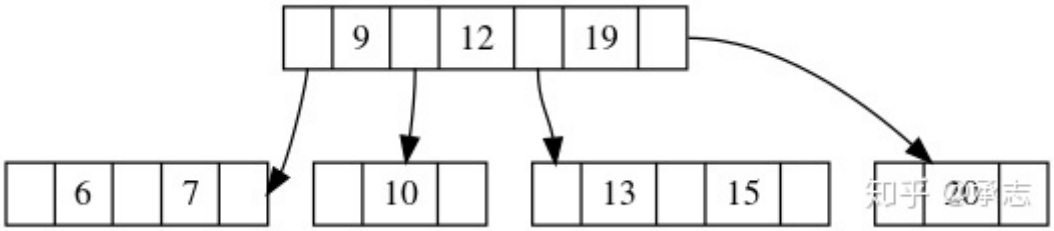
2.2.1.4

1.

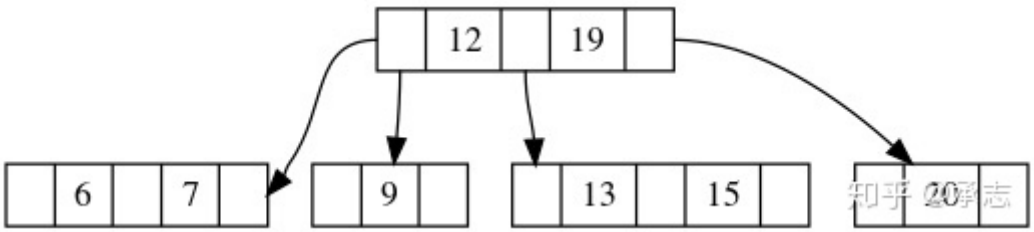
2. **B**

3.

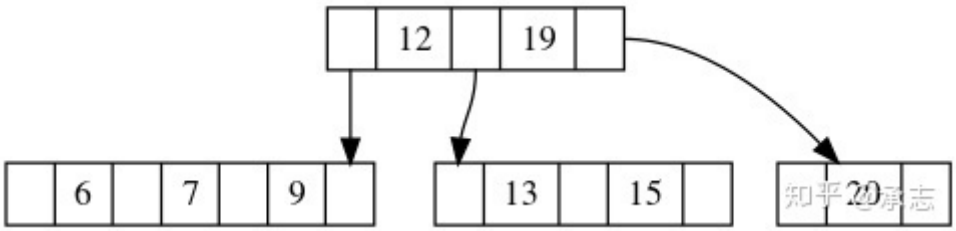
1. 10



2.



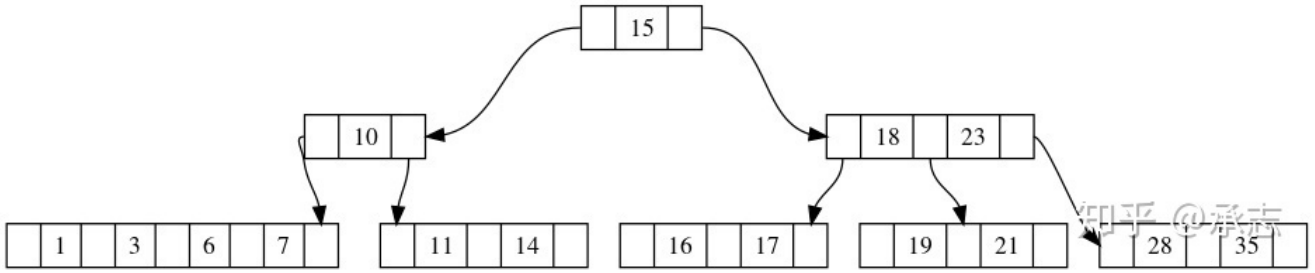
3. **B** **9**
4. $[6,7]$ $[9]$



5.

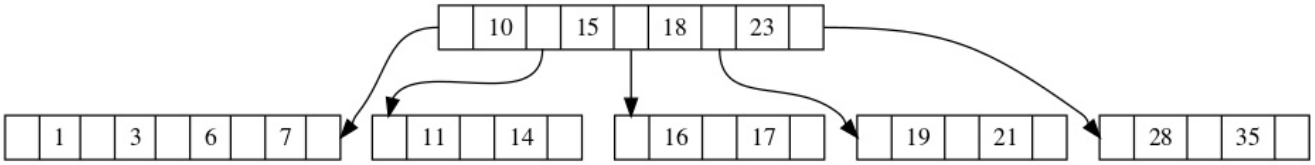
1

1. $t=3$ 9 6 $[1,3]$



2. B

$[10]$



3.

3 B+

3.1 B+

1. B+ **1**
   B
2. B+
   1
3. B+ B
4. m B+ ** $[m/2,m-1]$**
5.
6. **key** **key** **key** **key**
7. **key**
8.

clip_image039

2.2 B+

2.2.1

5 B+ $[2,4]$

1.

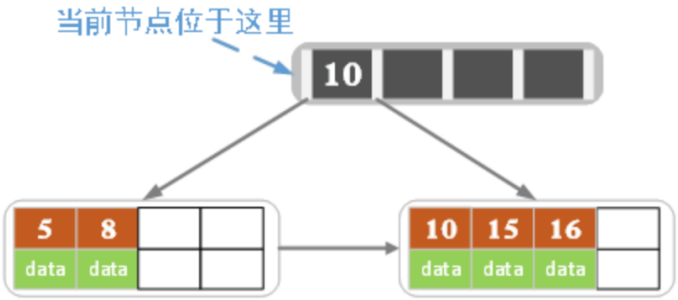
2.
    1. **key**
    2. **key** **$m-1$**
    3. **$m/2$** **$m/2+1$** **key** key

b）依次插入8，10，15

---

c）插入16

插入16后超过了关键字的个数限制，所以要进行分裂。在叶子结点分裂时，分裂出来的左结点2个记录，右边3个记录，中间key成为索引结点中的key，分裂后当前结点指向了父结点（根结点）。结果如下图所示。



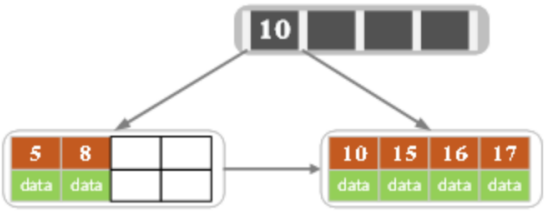
3.

1. **key** **$m-1$**
2. **$m/2$** **key**

   key

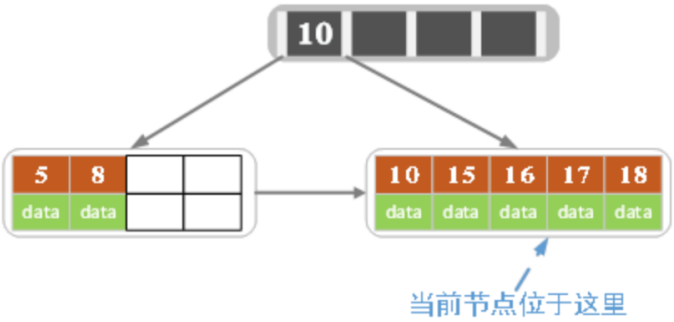
   3

d）插入17

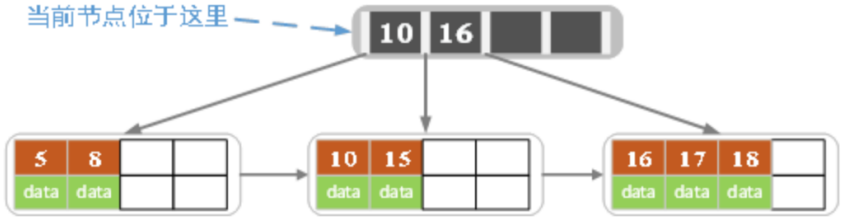


---

e）插入18，插入后如下图所示



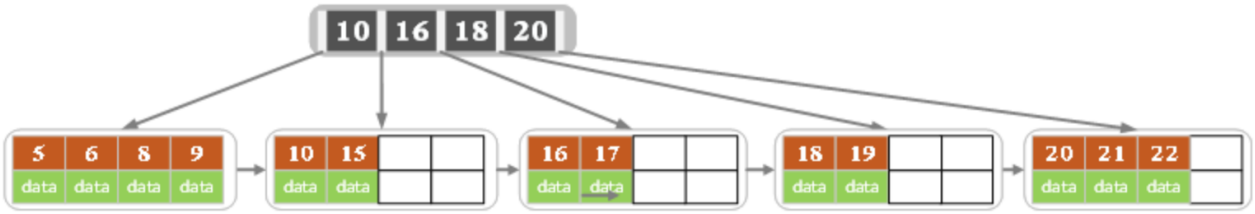
当前结点的关键字个数大于5，进行分裂。分裂成两个结点，左结点2个记录，右结点3个记录，关键字16进位到父结点（索引类型）中，将当前结点的指针指向父结点。



当前结点的关键字个数满足条件，插入结束。

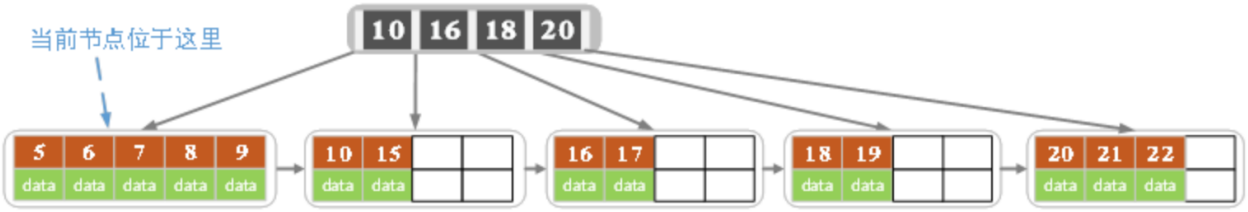
---

f）插入若干数据后

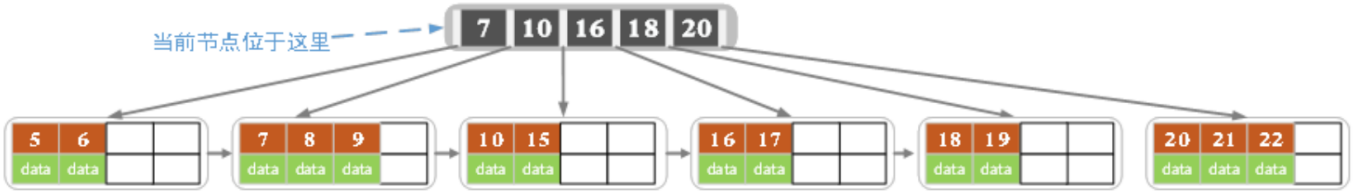


---

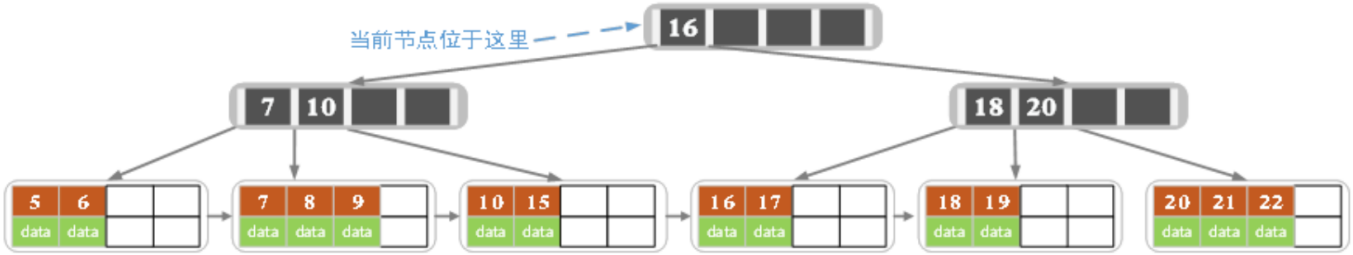
g）在上图中插入7，结果如下图所示



当前结点的关键字个数超过4，需要分裂。左结点2个记录，右结点3个记录。分裂后关键字7进入到父结点中，将当前结点的指针指向父结点，结果如下图所示。



当前结点的关键字个数超过4，需要继续分裂。左结点2个关键字，右结点2个关键字，关键字16进入到父结点中，将当前结点指向父结点，结果如下图所示。



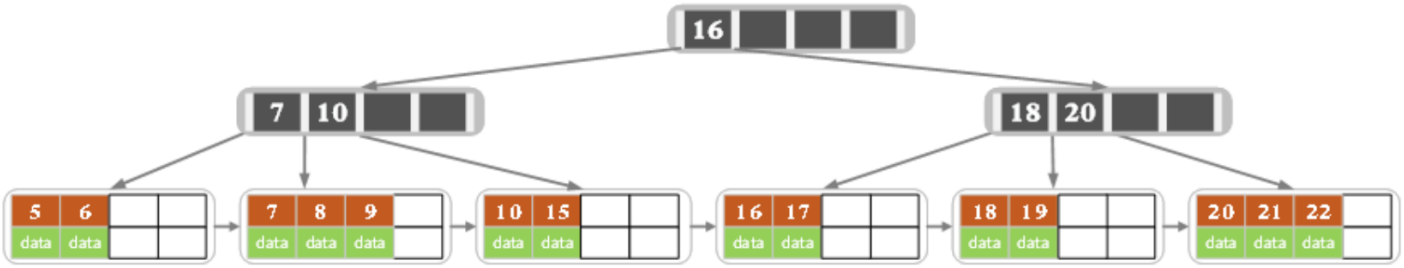
当前结点的关键字个数满足条件，插入结束。

2.2.2

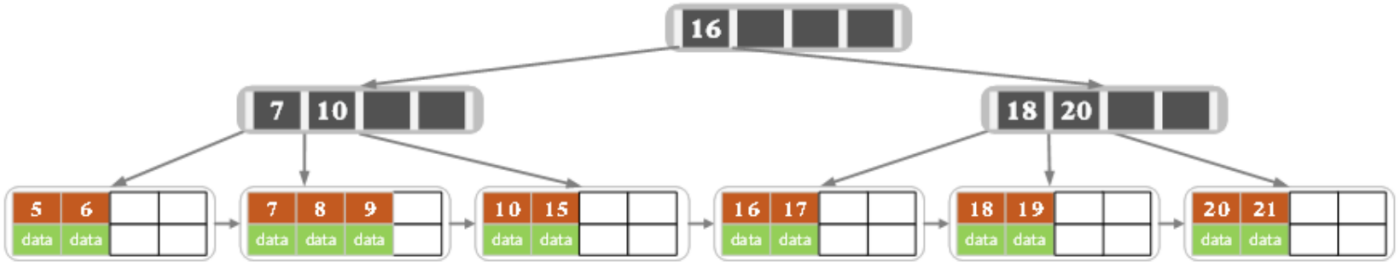
5 B+ $[2,4]$

1. **key**
2. key **key** **$m/2$**

a）初始状态



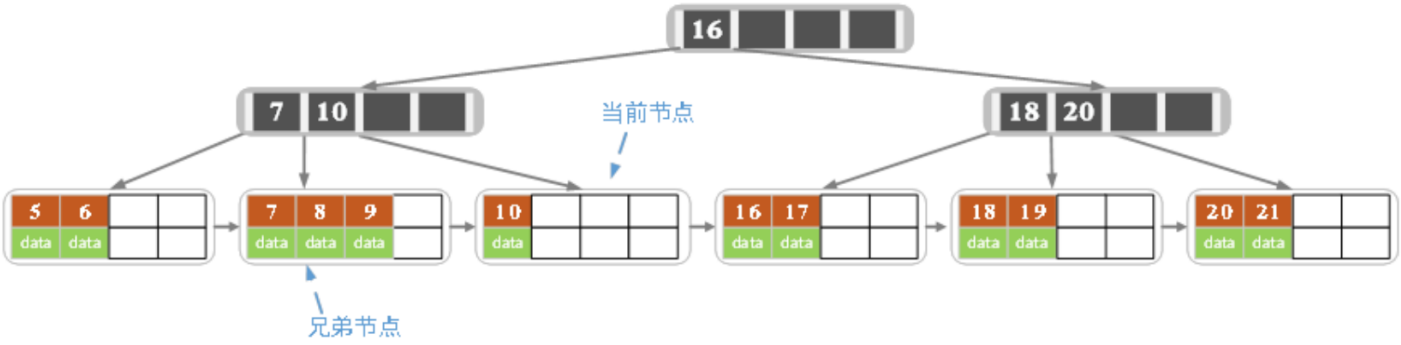
b）删除22,删除后结果如下图



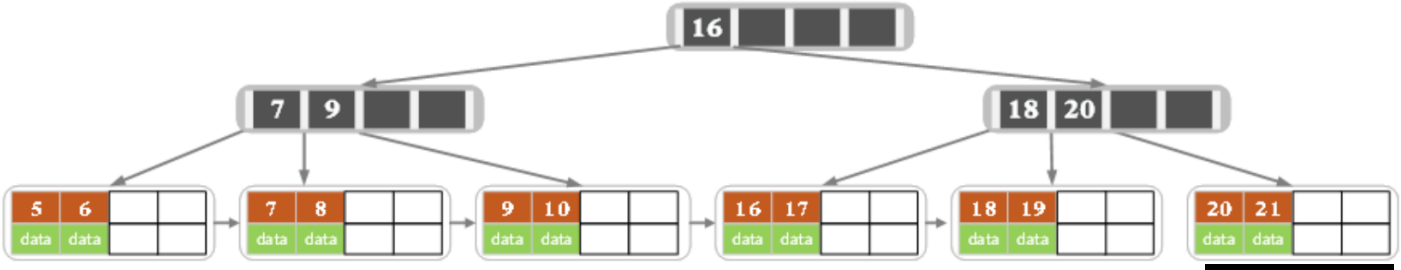
删除后叶子结点中key的个数大于等于2，删除结束

3. **key** B+ **key** **key** **key**

c）删除15，删除后的结果如下图所示



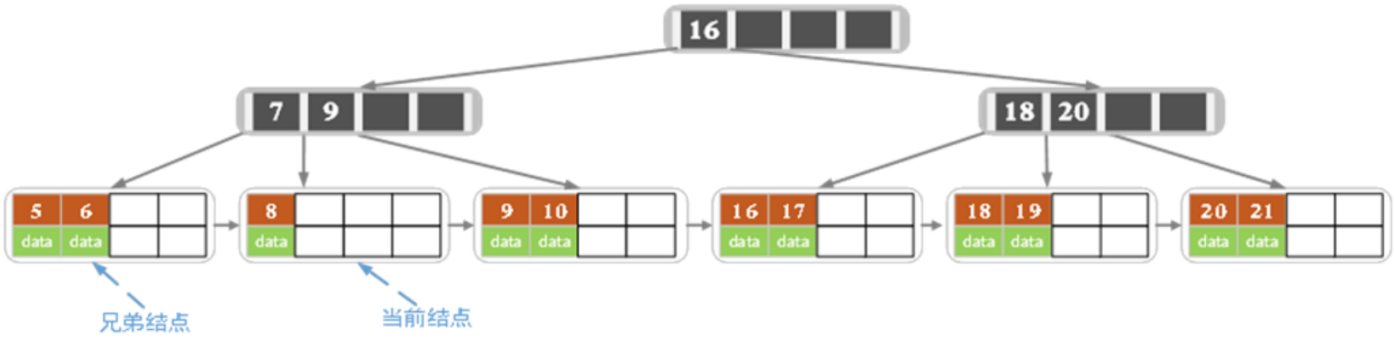
删除后当前结点只有一个key,不满足条件，而兄弟结点有三个key，可以从兄弟结点借一个关键字为9的记录,同时更新将父结点中的关键字由10也变为9，删除结束。



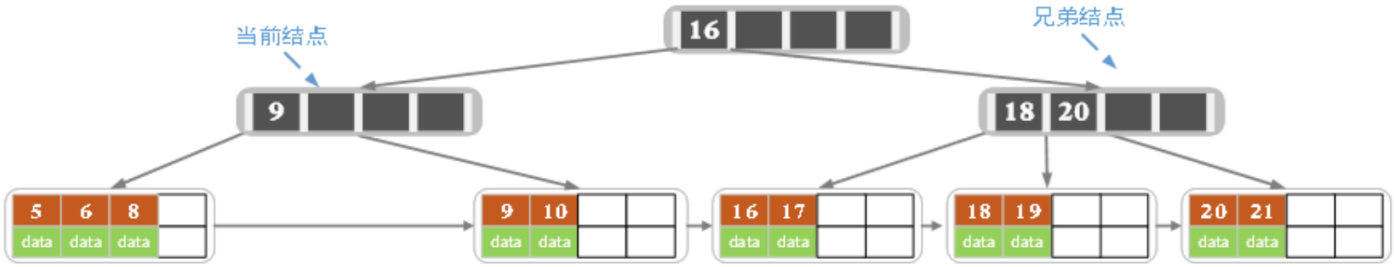
4. **key** **key** **key**

   1. **key** **$m/2$**
   2. **key** **key**
   3. **key** 4

   d）删除7，删除后的结果如下图所示



当前结点关键字个数小于2，（左）兄弟结点中的也没有富余的关键字（当前结点还有个右兄弟，不过选择任意一个进行分析就可以了，这里我们选择了左边的），所以当前结点和兄弟结点合并，并删除父结点中的key，当前结点指向父结点。



此时当前结点的关键字个数小于2，兄弟结点的关键字也没有富余，所以父结点中的关键字下移，和两个孩子结点合并，结果如下图所示。

5. **B+** **key**

4 B+

> InnoDB B+

1. B+ **B**
2.

1. SQL `select * from tbl where t > 10` **B+** **10**
   $next$
2. **B**

---

1. B B+
2.
3. —— B-
4. B B+
5. B B+
6. B+
7. MySQL B+

0.1

h

$O(h)$

“ ” **

$O(lgn)$**

1

1. **RED BLACK**

   2

2. **5** $color$ key $left$ $right$ p

   NIL **NIL**

3.

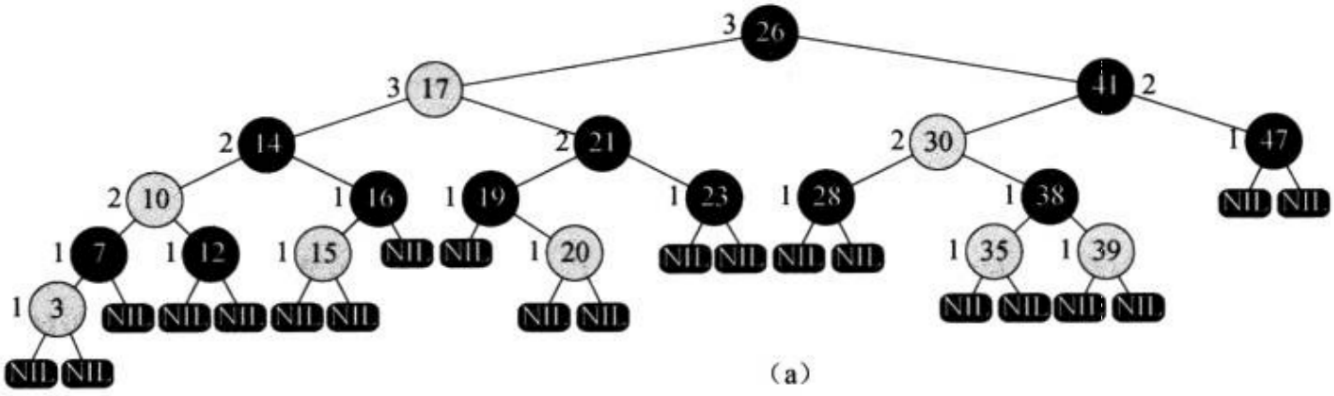
   1.

   2.

   3. NIL

   4.

   5.

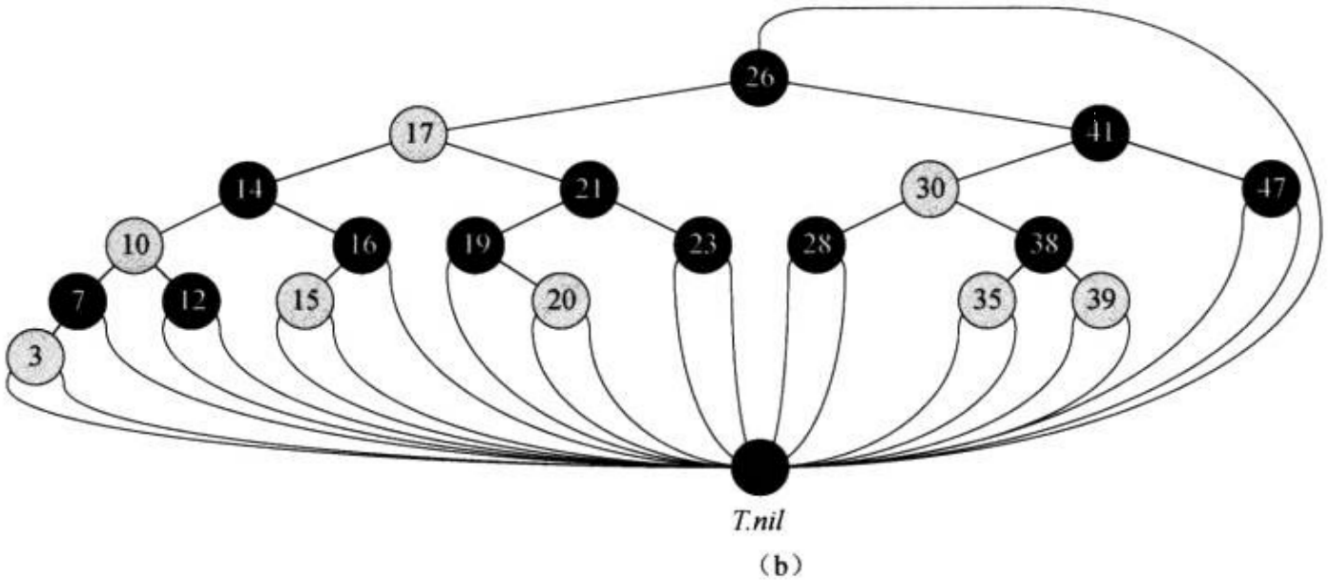


> NIL

4. ** NIL**

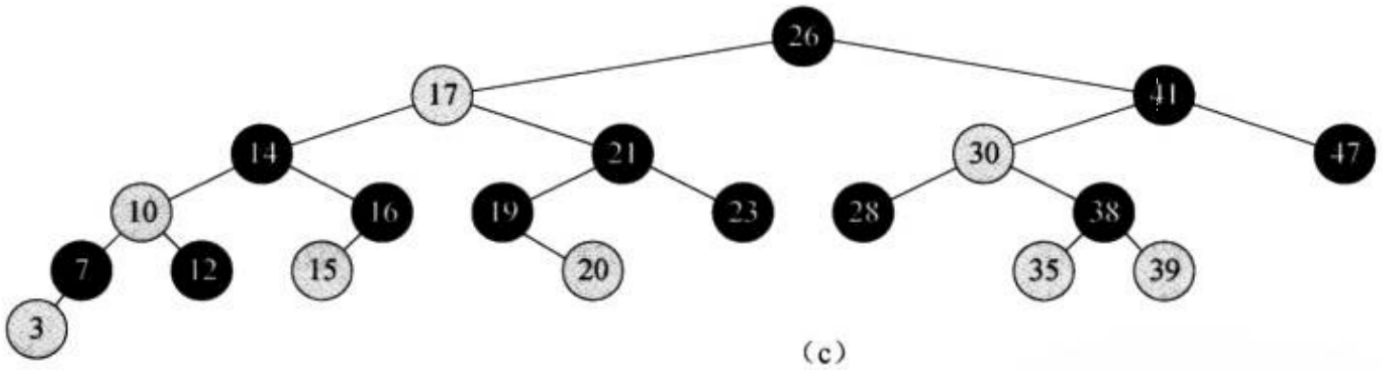
   1. **T** **$T.nil$** **$color$** **BLACK** **p** **$left$** **$right$** **key** **NIL** **$T.nil$**

2. **x NIL x NIL NIL $T.nil$ NIL p $left$ $right$ key**



(b)

5.



(c)

6. **x** $bh(x)$ 5

7. ** n $ 2lg(n + 1) $**

2

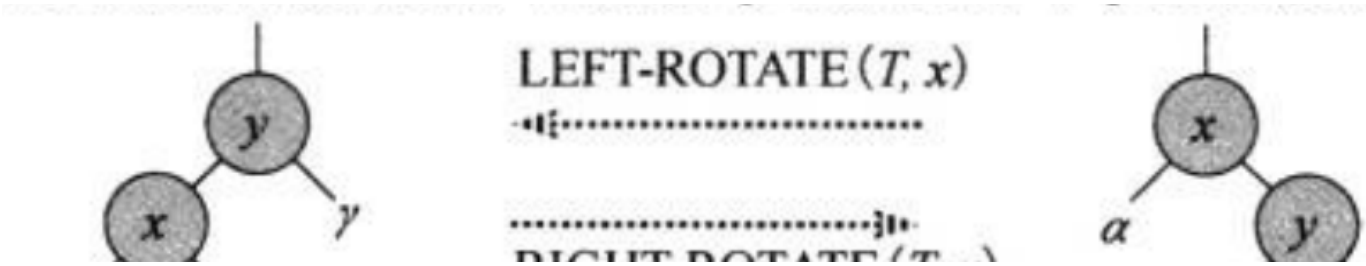
2.1

1. **n** **$O(lgn)$**

2.

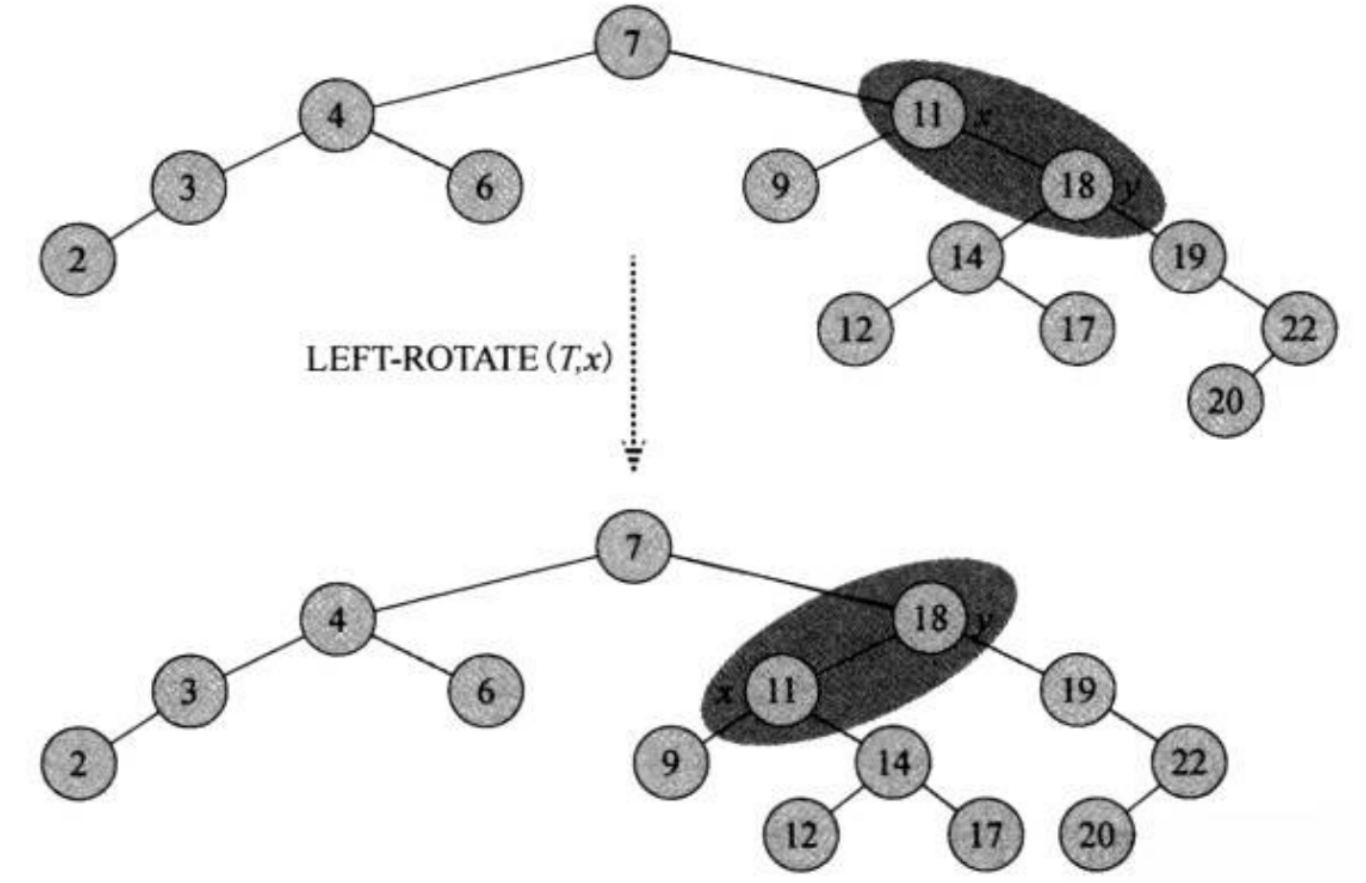
1. **x** ** y $T.nil$** x **$T.nil$**

2. **x** **y** **“ ”** **y** x **y** **y x **



3.

1. $O(1)$



2.2

1. **$O(lgn)$** **n**

2. **z** **z**

1. **1** **3** ** $T.nil$**

2. **5** **z** **z**

3. **2** **4** z

    1. **z** **2**
    2. **z** **4**

3. **6**

z **$z.p$** **z** **$z.p.p$** **$z.p$**

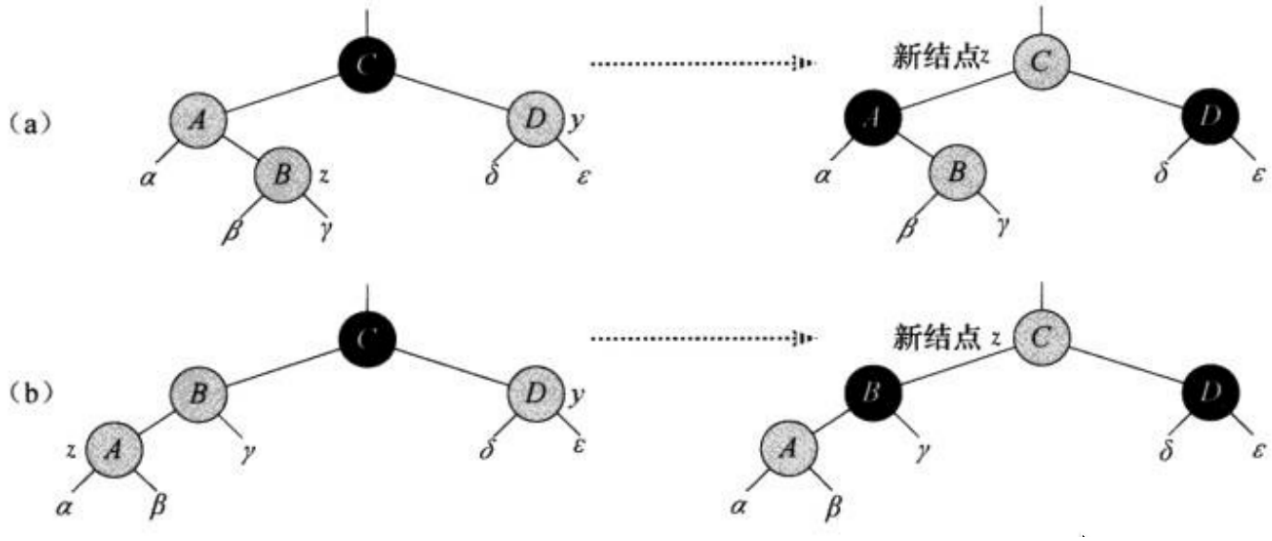
1. **z** “ ” y

    1. **$z.p$** **y** **4**

    2. **$z.p.p$** **$z.p$** **y** **z** **$z.p$** **$z.p.p$**
    5

    3. **z** **$z.p.p$** **z** **z**

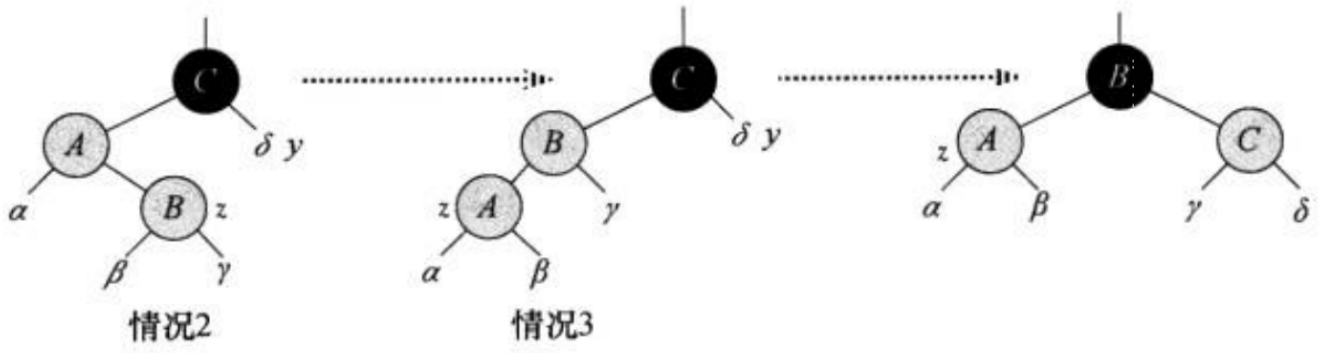
    4. **4** **z**



2. **z** **y** **z**

3. **z** **y** **z**

    1. **2** **3** **z** y z $z.p$ **
    2. **2** **z** **3** **z** **z**
    $z.p$ **5**
    3. **3** **5**



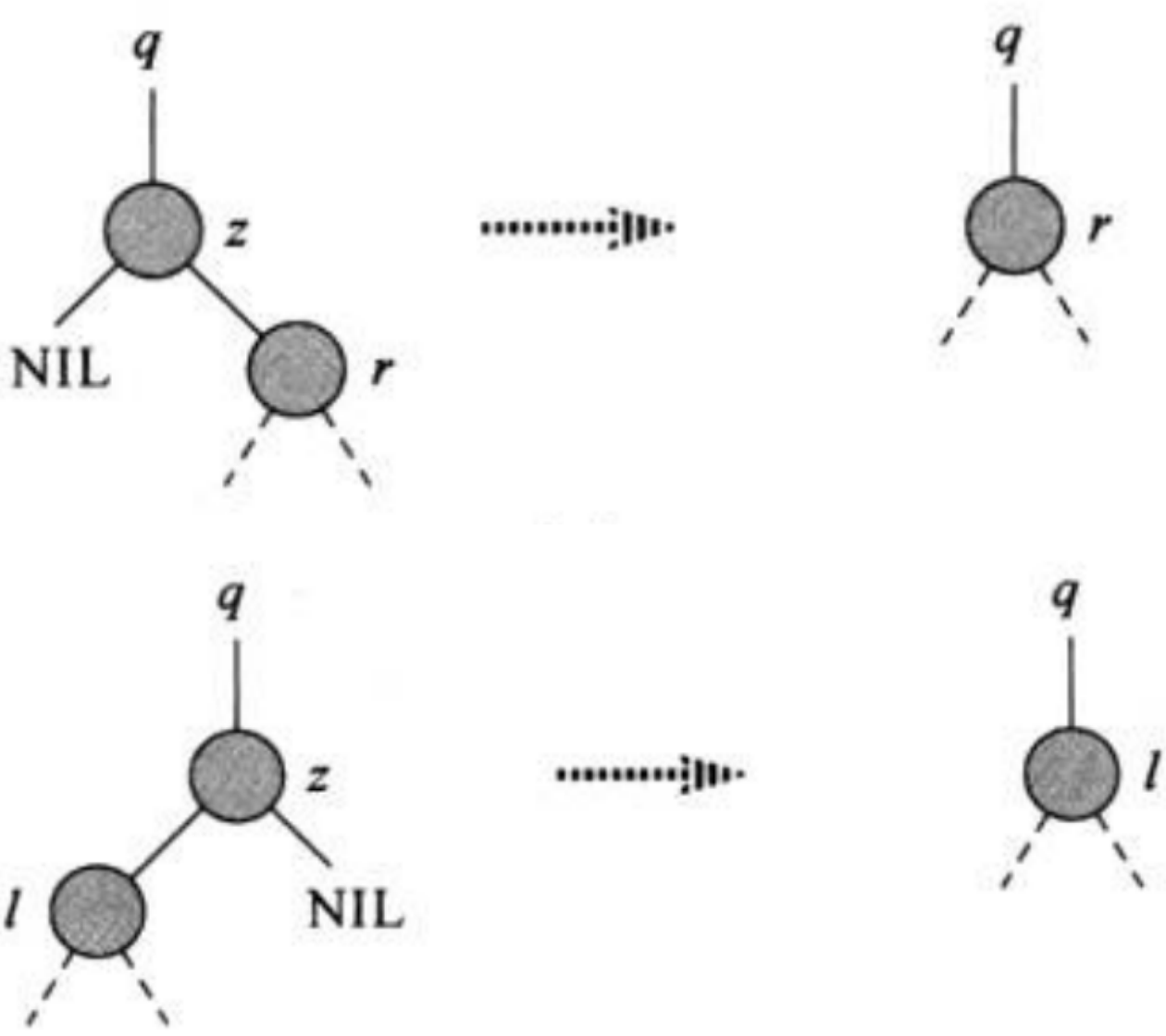
2.3

1. **5** 4
2.

2.3.1

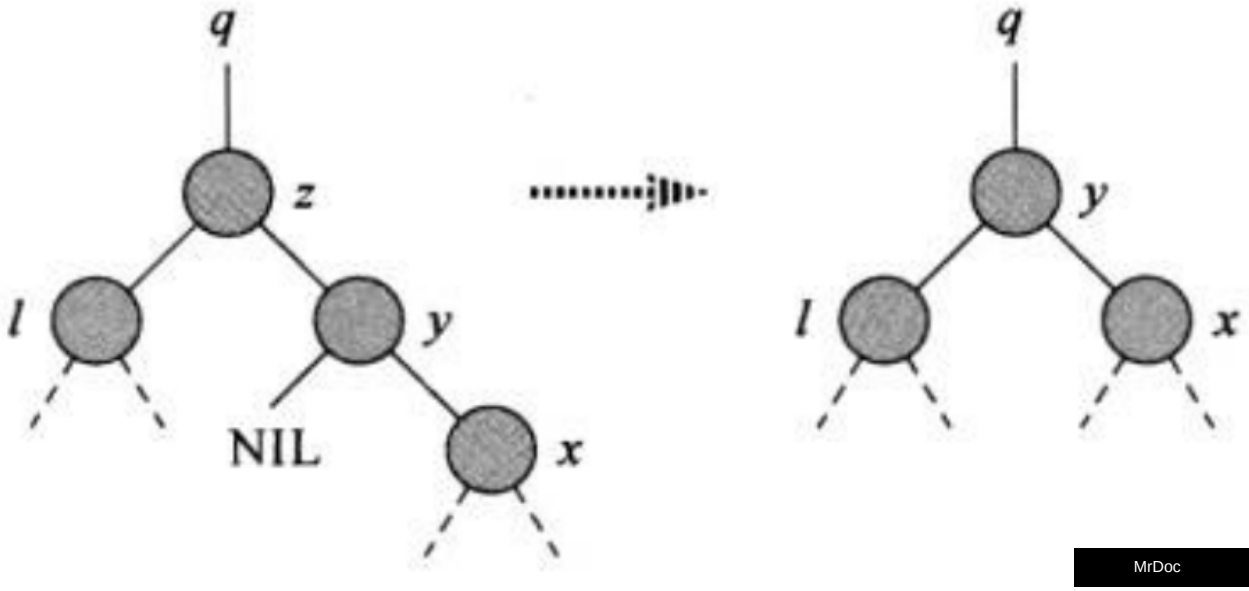
1.

    1. **z** ** NIL z**
    2. **z** **z** **z** ** z z**

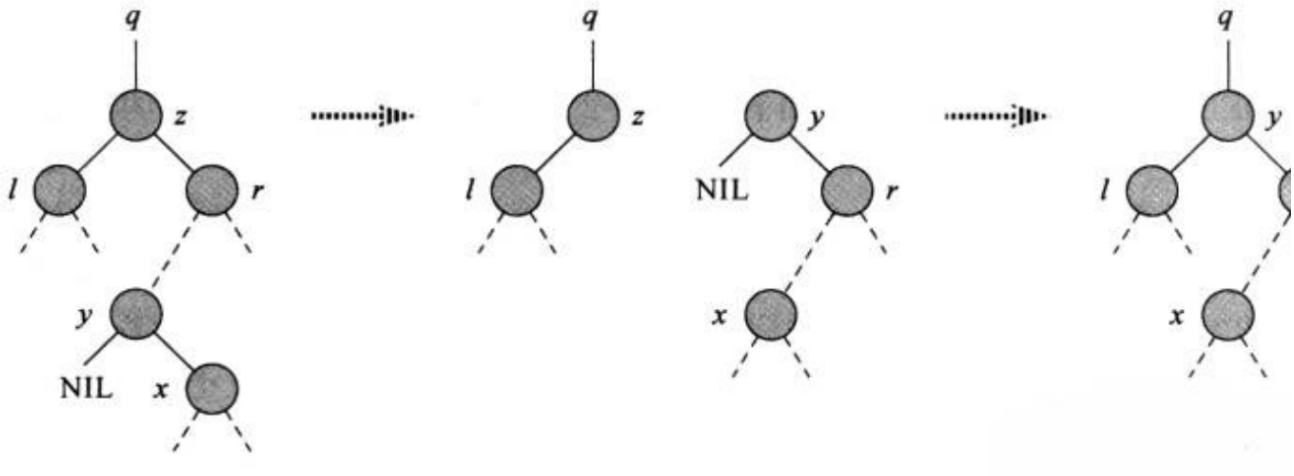


3. **z** ** z y** z

    1. **y z** **y z** **y**



2. **y** **z** **z** ** y y y z**



2.3.2

1. x x
   x x **x**
   x

   1. **x** **w** delfixup-case1

      1. x **$x.p$** **x** **$x.p$**
      2. **x** **B** **B** **1** **B**
         $bh_a = bh_b = bh_r - 1$ $bh_r = bh_d = bh_e = bh_f$
      3. **B** **B** **D** **2 3 4** x
         A B

   2. **x** **w** **w** delfixup-case2

      1. **1** **B** ** $bh_a = bh_b = bh_r = bh_d = bh_e = bh_f$** B
         $bh_a + 1$ **$bh_r + 2$**
      2. **D** **B** **B** **1** **B**

         1. **B** Case 2.1 **B** **B**

         2. **B** **B** **x**

   3. **x** **w** **w** **w** delfixup-case3

      1. **2** **4**
         1. **$x.p$**
         2. **c$x.p$**
      2. **w** **w**
         1. **w** ** $bh_r = bh_d = bh_e + 1 = bh_f + 1$**
         2. **w** **C**

1. **w** **bh_r** **$bh_e + 1 = bh_r$** **w**
2. **D** **bh_d** **$bh_e + 1 = br_d$** **D**
3. **B** **4**

4. x **w** **w** 4

1. $x.p$ **w** 

1. **B** $bh_a = bh_b = bh_r = bh_d$ $bh_e = bh_f = bh_a + 1$
2. **B** **D** **B** **B**
   B B **E** **B**
   D
   1

2. $x.p$ **w** 

1. $bh_a = bh_b$ $bh_r = bh_d = bh_e = bh_f = bh_a + 1 = bh_b + 1$
2. **B** **D** **B** **B** **E**
   B **D**
   2

3. $x.p$ **w** 

1. $bh_a = bh_b = bh_r = bh_d$ $bh_e = bh_f = bh_a + 1$
2. **B** **D** **B** **B** **E**
   B **D**
   x **2** **x**
   x

4. $x.p$ **w** 

1. $bh_r = bh_d = bh_e = bh_f$ $bh_a = bh_b = bh_r - 1$
2. **B** **D** **B** **B** **E**
   B **D**
   x **2** **x**
   x

> 1 2 3 4 x x

3

1. **$O(lgn)$** **2** 2
2. **$O(lgn)$** **3** 3

1.
2.
3. (R-B tree)