数据结构——B + 树 - CSDN 博客

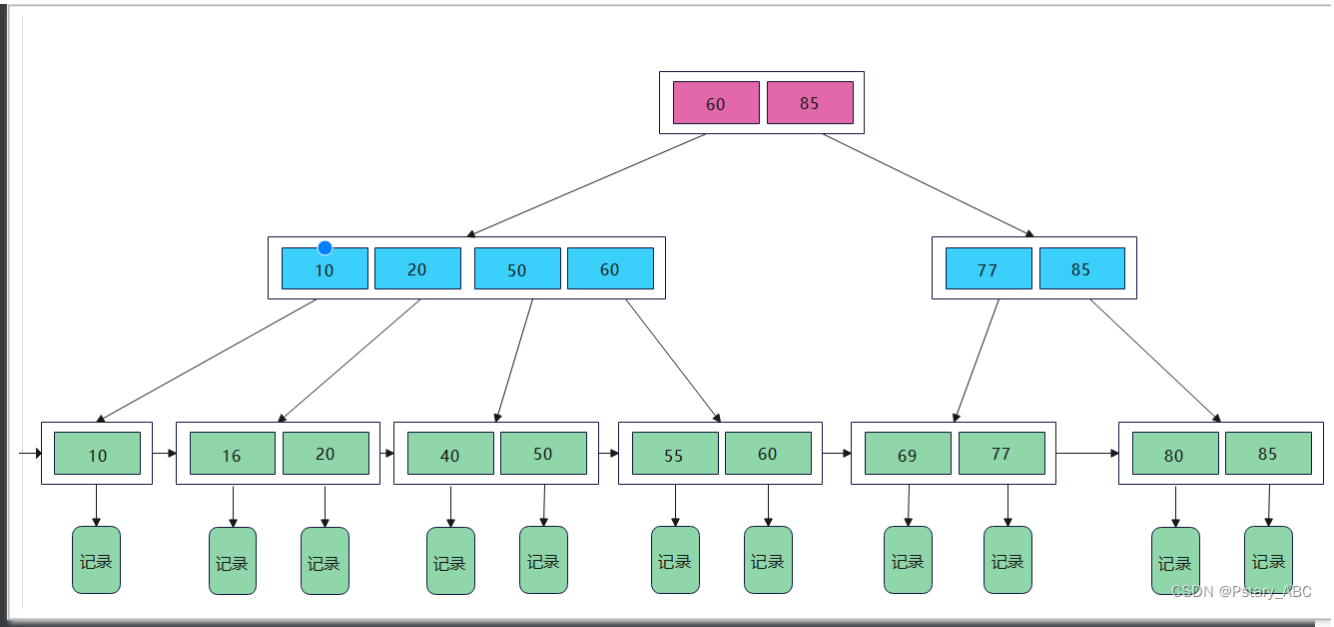
一、B + 树的概念

B + 是应数据所 出 B 变形 m B + 下列条件:

1. 个分支 最多有 m 子 (孩子);
2. 叶 少有两 子 , 其他 个分支 少有[m/2] 子 ;
3. 字数个数与关 字个数 ;
4. 所有叶 包含全 关 字及指向 应 录 指 , 叶 中将关 字按大小 序排列, 并且 叶 按大小 序 互 接 来;
5. 所有分支 (可 为 引 引) 中仅包含他 各个子 (即下 引块) 中关 字 最大值及指向其子 指

二、B + 树和 B 树的差异

1. 在 B + 中, 具有 n 个关 字 只含有 n 子 , 即 个关 字对应 子 ; 在 B 中, 具有 n 个关 字 含有 n+1 子
2. 在 B + 中, 个 (内) 关 字个数 n 围为 [m/2]~m (: 1~m); 在 B 中, 个 (内) 关 字个数 n 围为 [m/2]-1~m-1 (: 1~m-1)
3. 在 B + 中, 叶 包含了全 关 字, 叶 中出 关 字也会出 在叶 中; 在 B 中, 最外层 包含 关 字和其他 包含 关 字是不 复
4. 在 B + 中, 叶 包含信息, 所有 叶 仅 引作 , 叶 个 引 只含有对应子 最大关 字和指向 子 指 , 不含有 关 字对应 录 存储地址



三、B + 树的查找

B + 中所有数据均保存在叶子，且和内均只是充当控制找录媒介，并不代数据本，所有内元同时存在于子中，是子元中是最大（或最小）元

例如在上图中 B + 中找 55 个关字，如下：

- 在中对 55 和中元 [60, 85]，发 55<60，因应在个中寻找；
- 同，55 和个中元 [10, 20, 50, 60]，发 50<55<60，因 55 应存在于四个当中；
- 对 55 和四个中元 [55, 60]，找到 55，找成功当，也有找失情况，即找元并不在 B + 中

四、B + 树的插入

B + 插入和 B 十分似，其插入则如下：

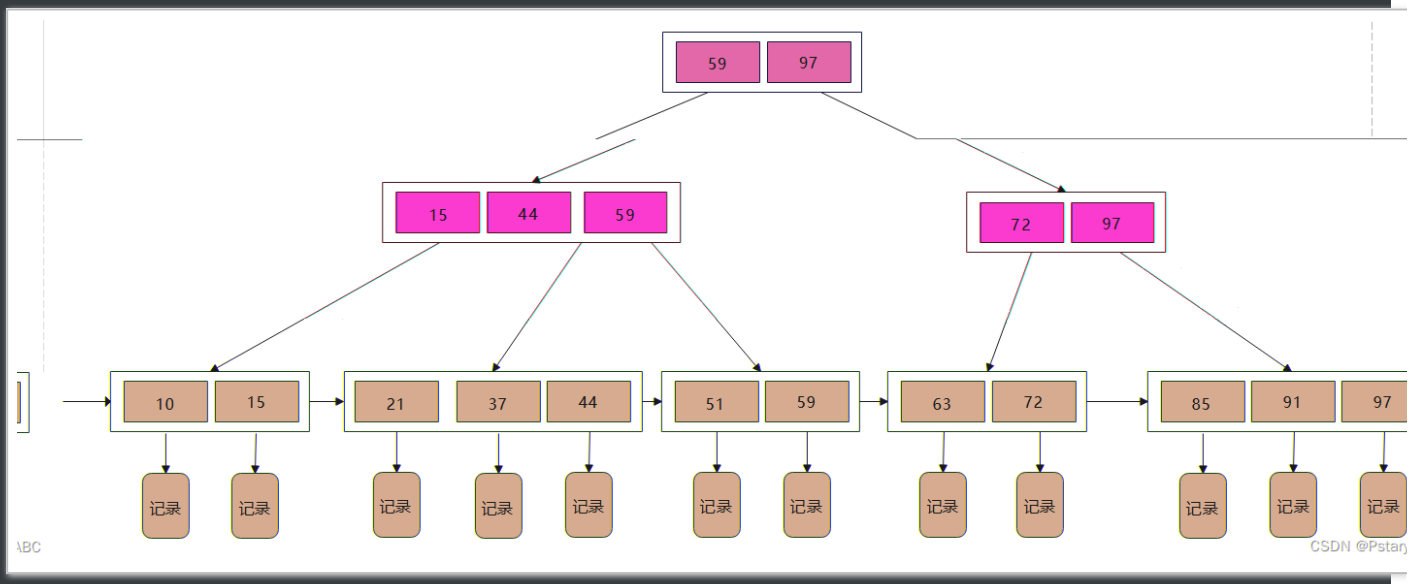
- 插入操作全在叶子上，且不坏关字小大序；
- 当插入关字后关字个数大于 m，“分”
  B + 插入有四情况：

1. 插入关字所在，其含有关字数小于 m，则接插入；
2. 插入关字所在，其含有关字数于 m，则将个分为左右两分，中放到中假其双亲中包含关字个数小于 m，则插入操作完成

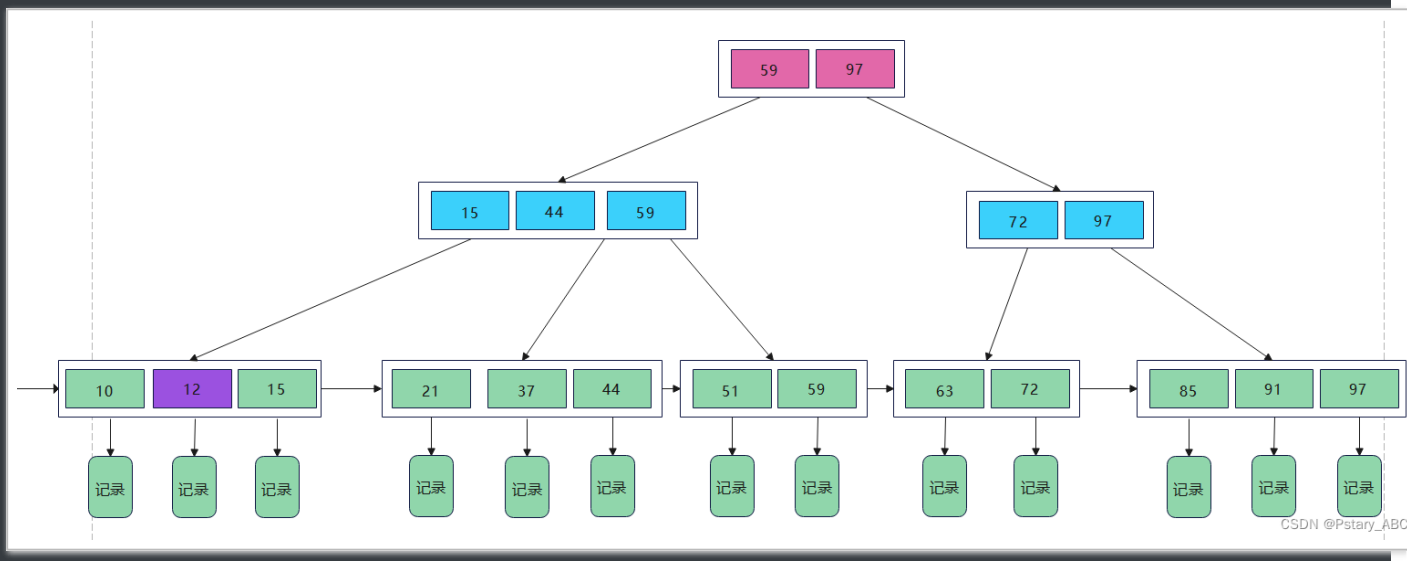
3. 在 2 情况中，如 上 操作导 其双亲 中关 字个数大于 M，则应 分 其双亲

4. 插入 关 字 当前 中 最大值 大， 坏了 B + 中从 到当前 所有 引值，时 及时修 后，再做其他操作

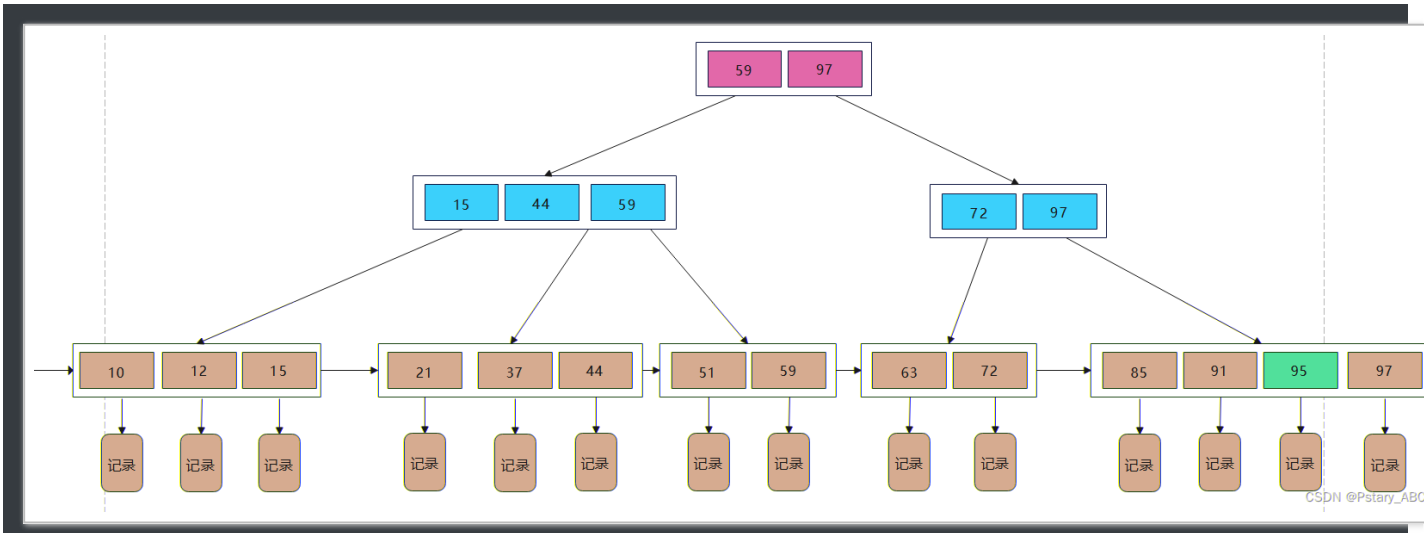
举例：



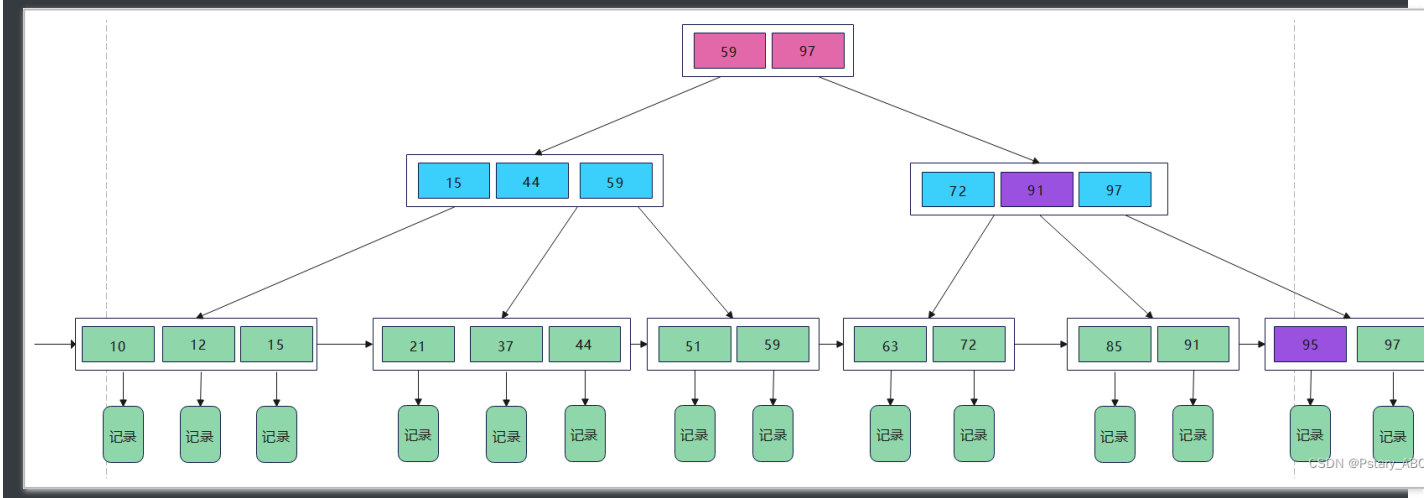
(1) 插入关 字 12， 时 个叶子 分 [10, 15] 关 字 个数 < m，可以 接插入： (代 插入 元)



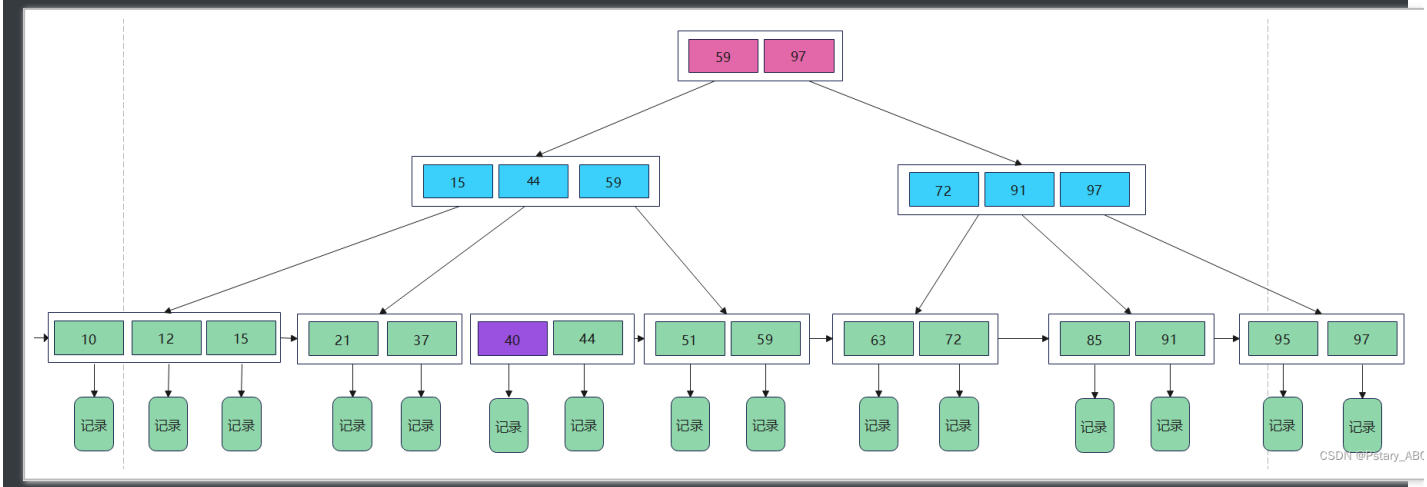
(2) 插入 95， 插入到最后 个叶子 分 [85, 91, 97]：



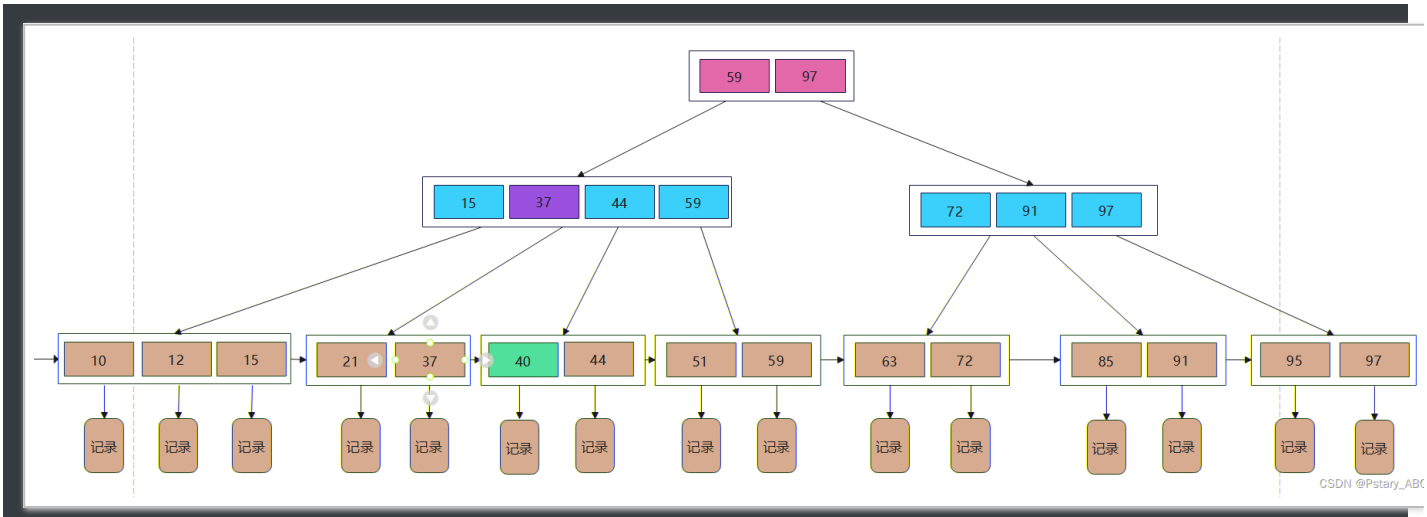
时　　关　字个数大于 m，　　分　操作，并且　　插入　个新　关　字：



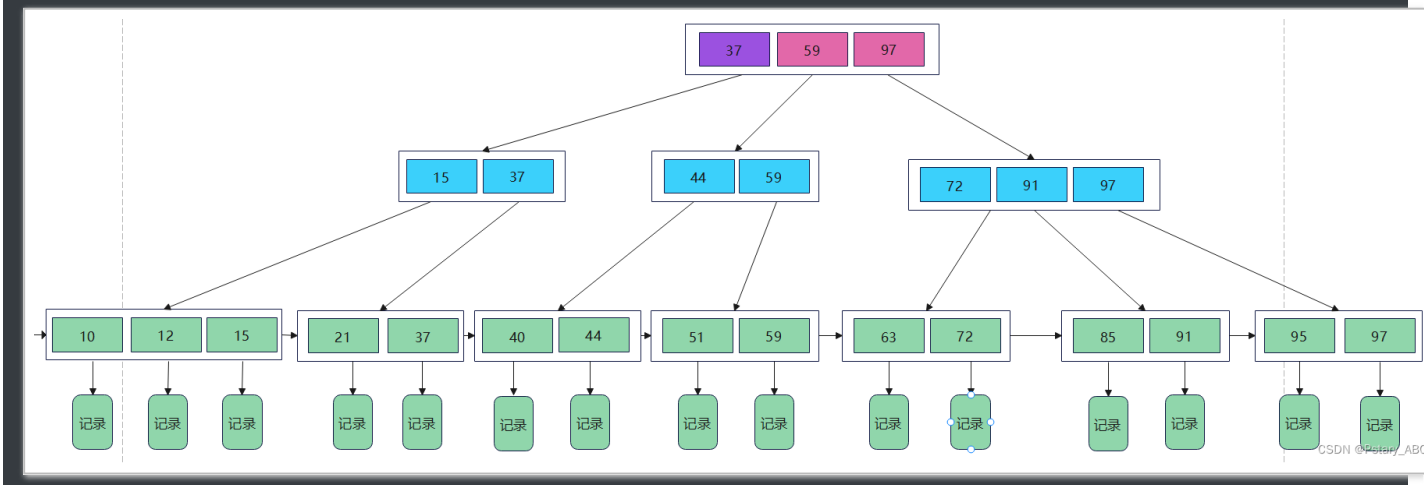
(3) 插入 40，　　插入到　二个叶子　　分 [21, 37, 44]：



时　　关　字个数大于 m，　　分　操作，并且　　插入　个新　关　字：

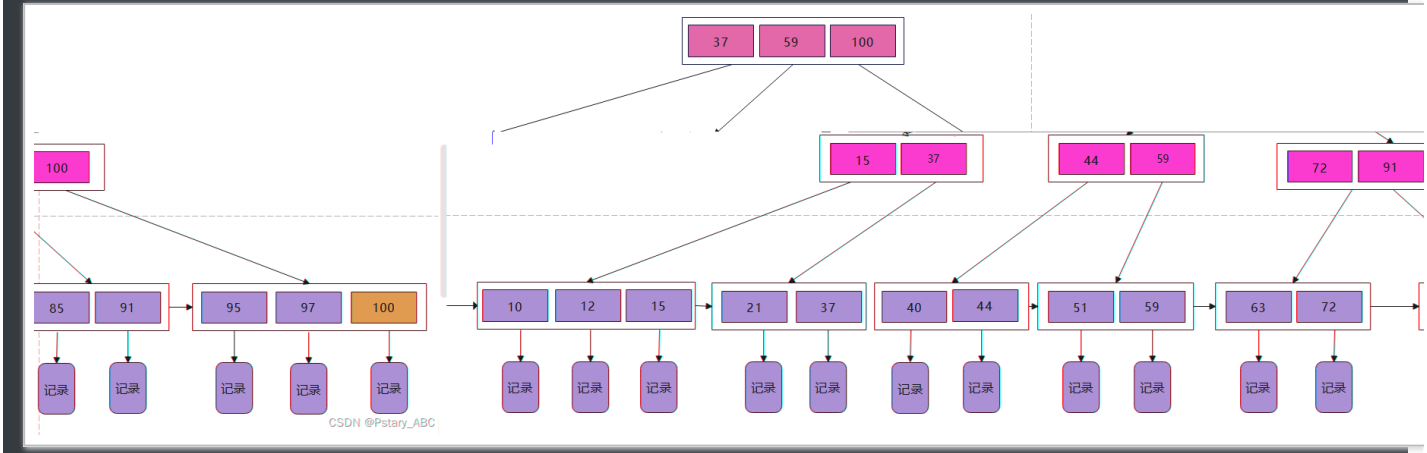


插入新　关　字之后，关　字　个数大于 m，也　　分　：



(4) 插入 100，　于其值　最大值 97　大，插入之后，从　　到　　所有　中　所有　值　　97 改为 100　（　为修改之后　）

修改完最大值之后，在最后　个　处插入 100：

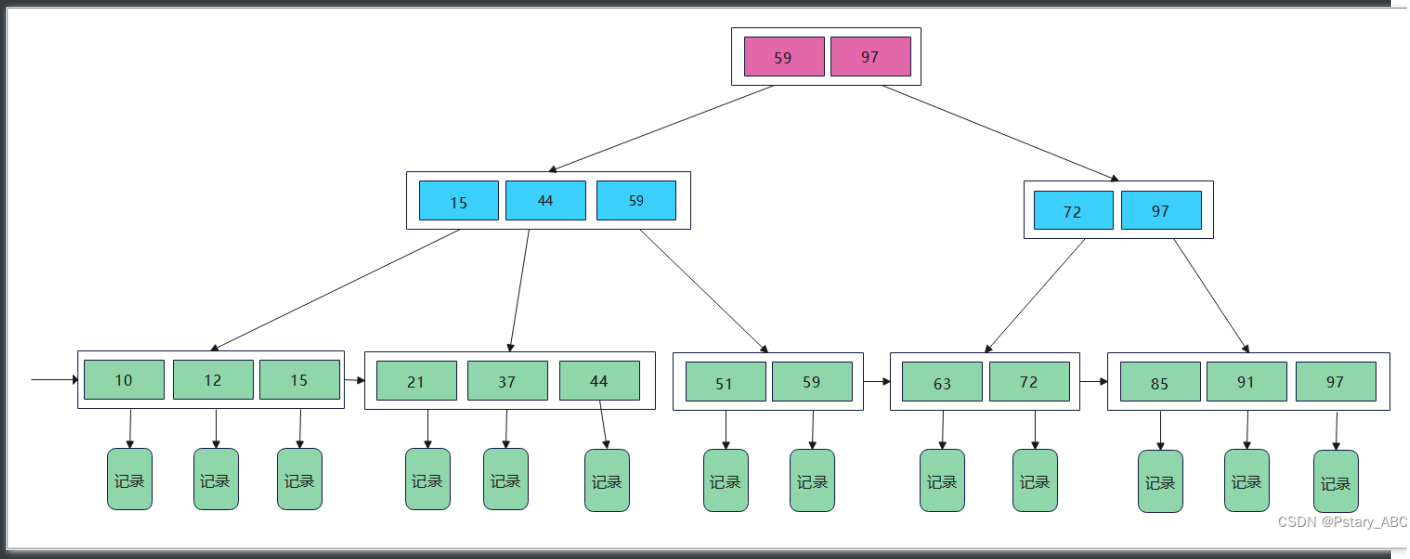


五、B + 树的删除

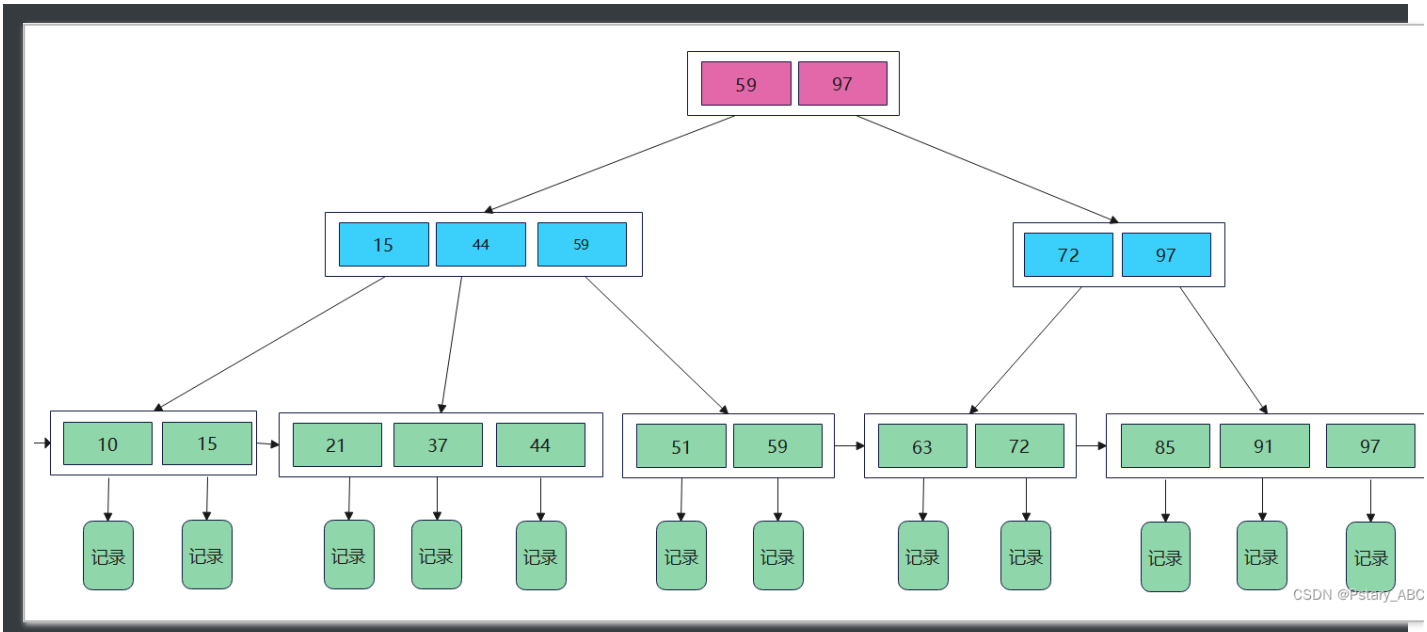
B +　　删　也和 B　十分　似，它有下　几　情况：

1. 找到存储有　关　字所在　　时，　于　　中关　字个数 >=[m/2]，做删　操作不会　坏 B +　　，则可以　接删　；
2. 当删　　中最大或　最小　关　字，就会　及到更改其双亲　　到　　中所有　引值　更改
3. 当删　关　字，导　当前　　中关　字个数小于 [m/2]，　其兄弟　　中含有多余　关　字，可以从兄弟　　中借关　字完成删　操作
4. 　3　情况中，如　其兄弟　　有多余　关　字，则　　同其兄弟　　合并
5. 当　合并时，可　会产　因合并使其双亲　　坏 B +　　，　依　以上　律处　其双亲

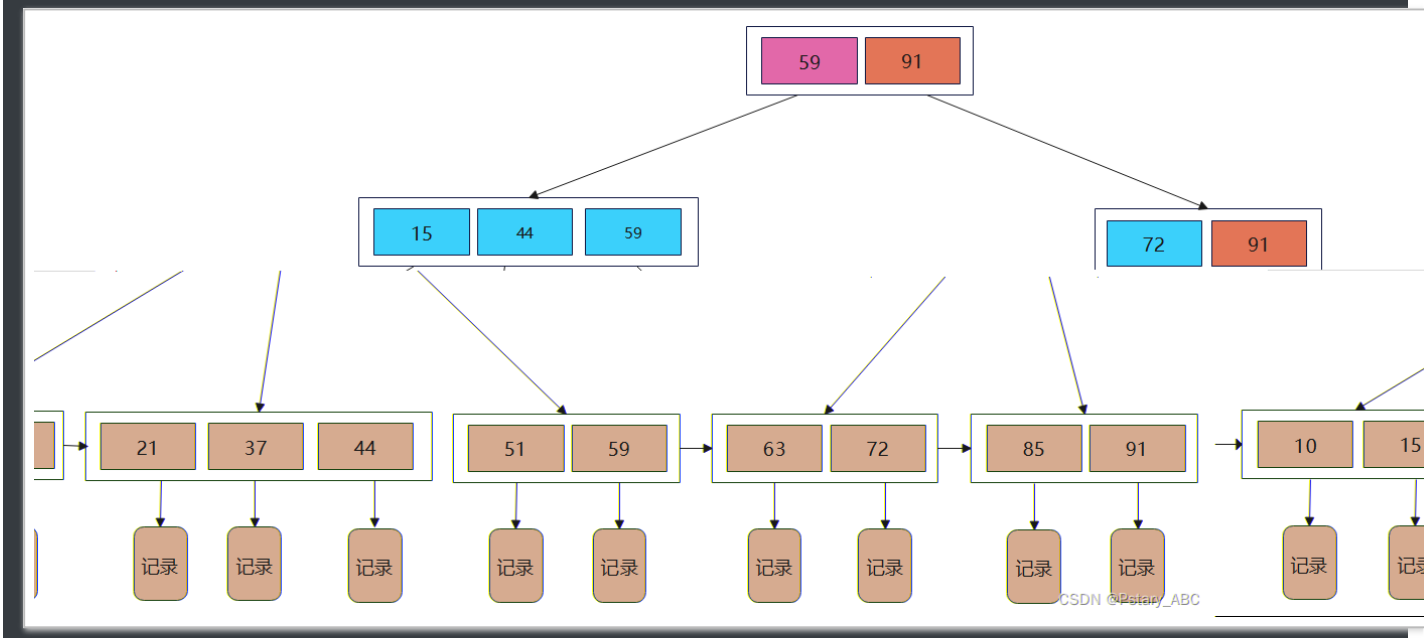
举例：



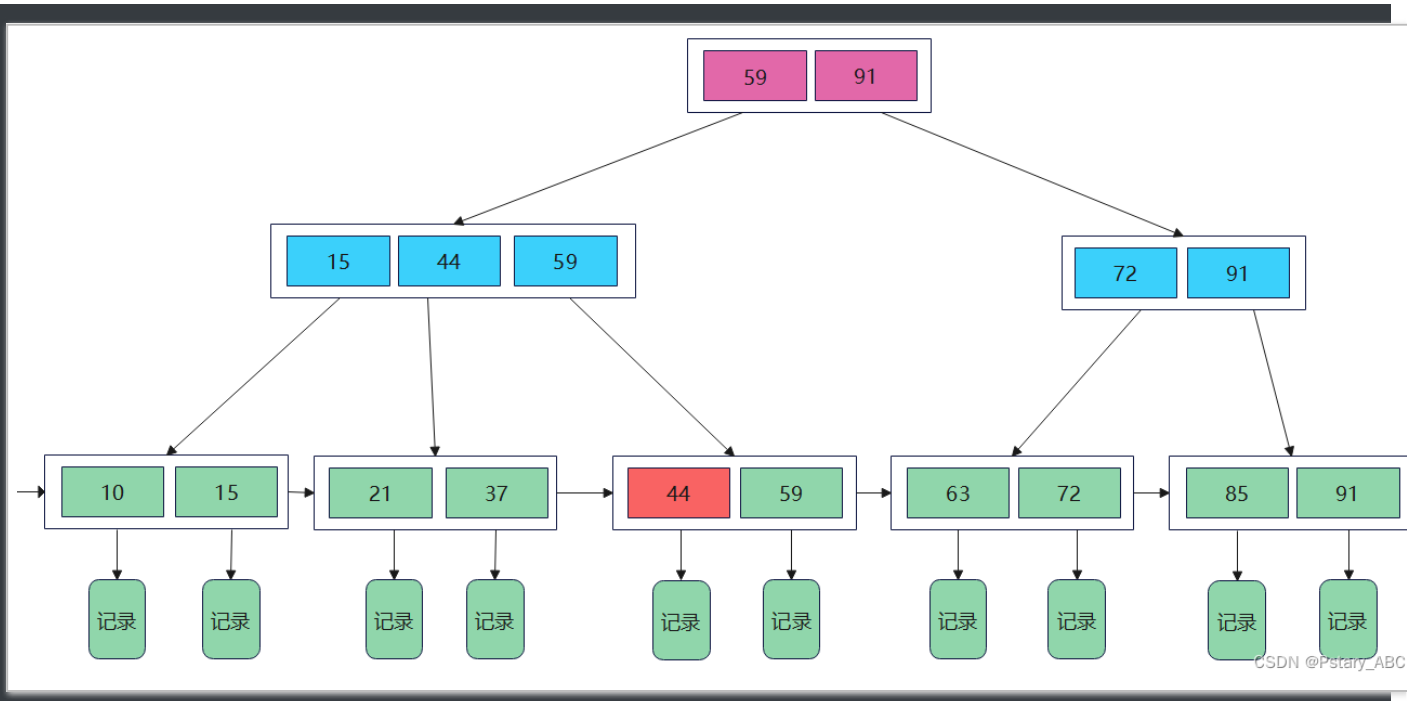
（1）删　12，包含 12　叶子　　关　字个数为 3，当删　12 时，关　字个数仍　大于　于 [m/2]，可以　接删　：



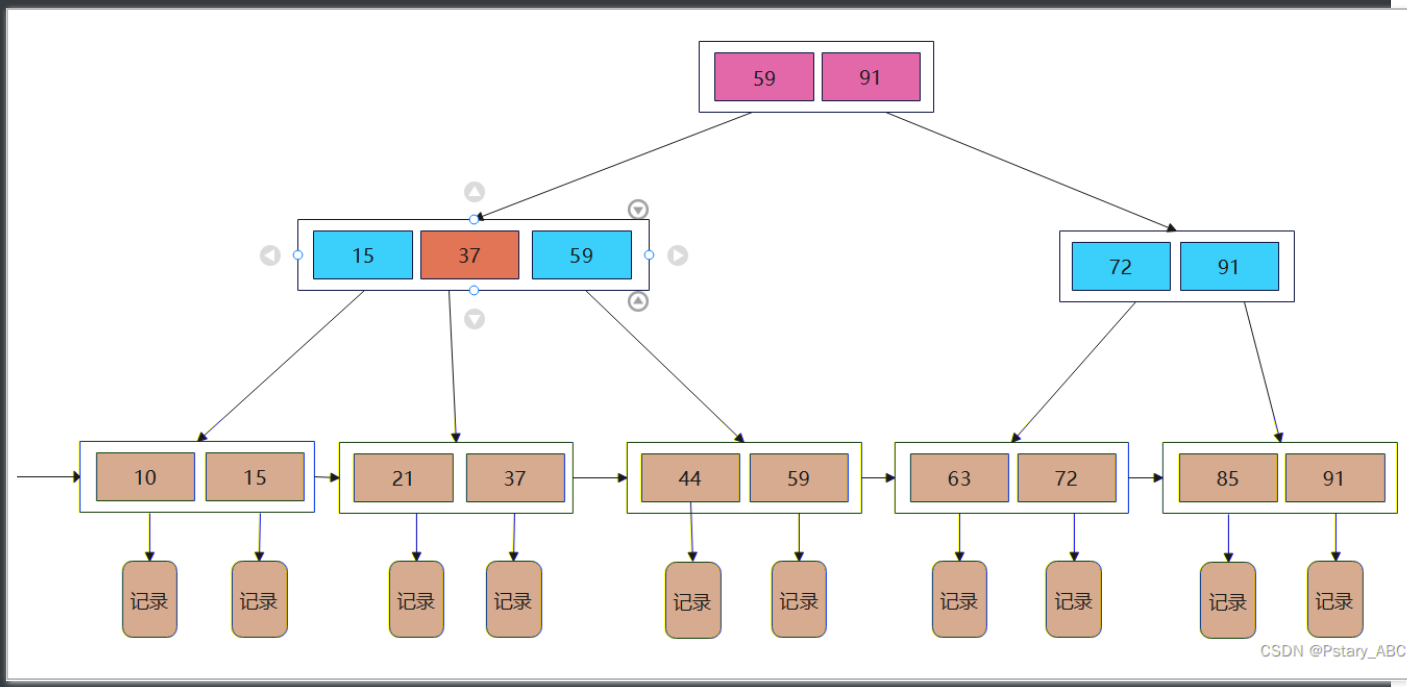
(2) 删 97，97 为整个 B + 中元 最大 值，当删 个元 时， 修改从 97 到 中所有 及到 97 值，将其修改为 三大 元 值（在 个例子中 三大 元 为 91）：



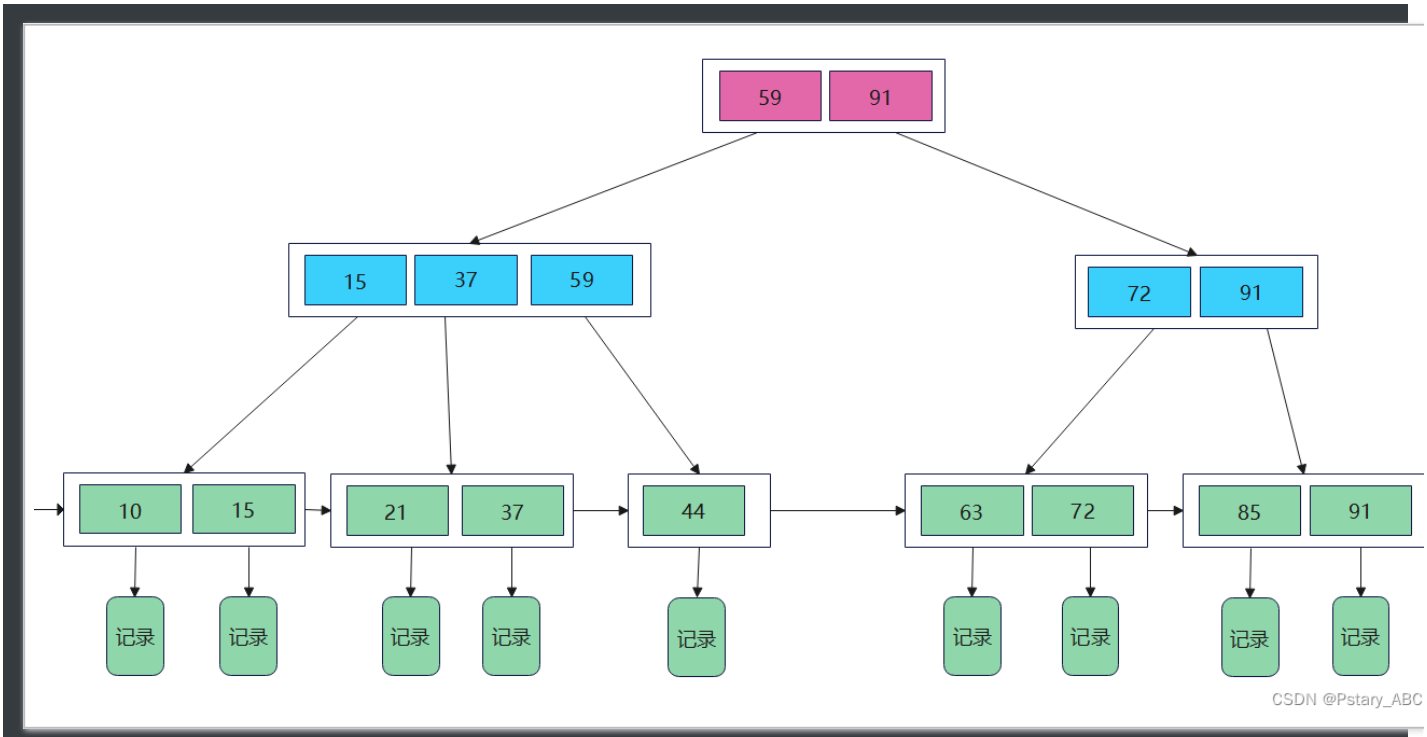
(3) 删 51， 时 51 所在 只有 59 个元 ， 关 字个数小于|m/2|， 它 兄弟 元 个数大于|m/2|，可以 借 个：



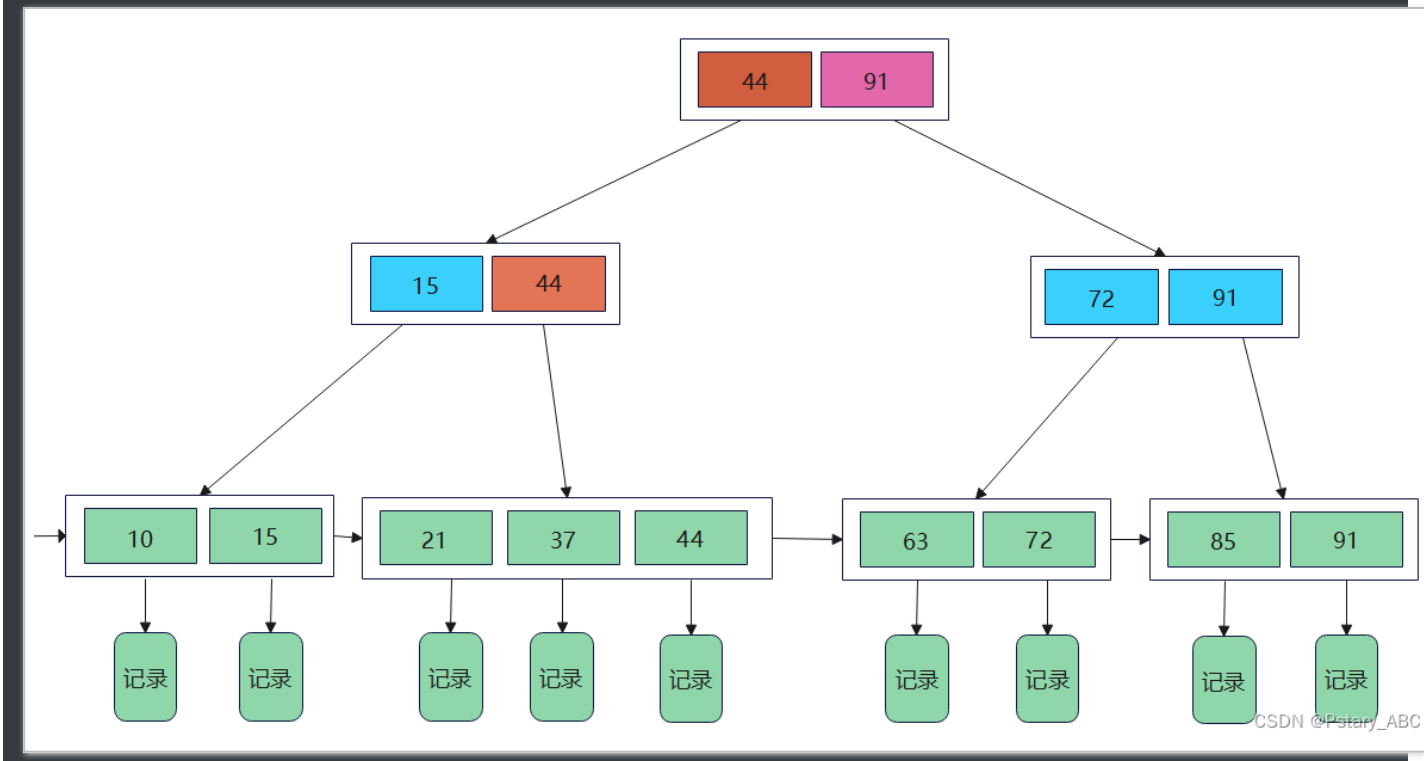
时，三个叶子节点中最大关键字为37，因此修改其父节点值：



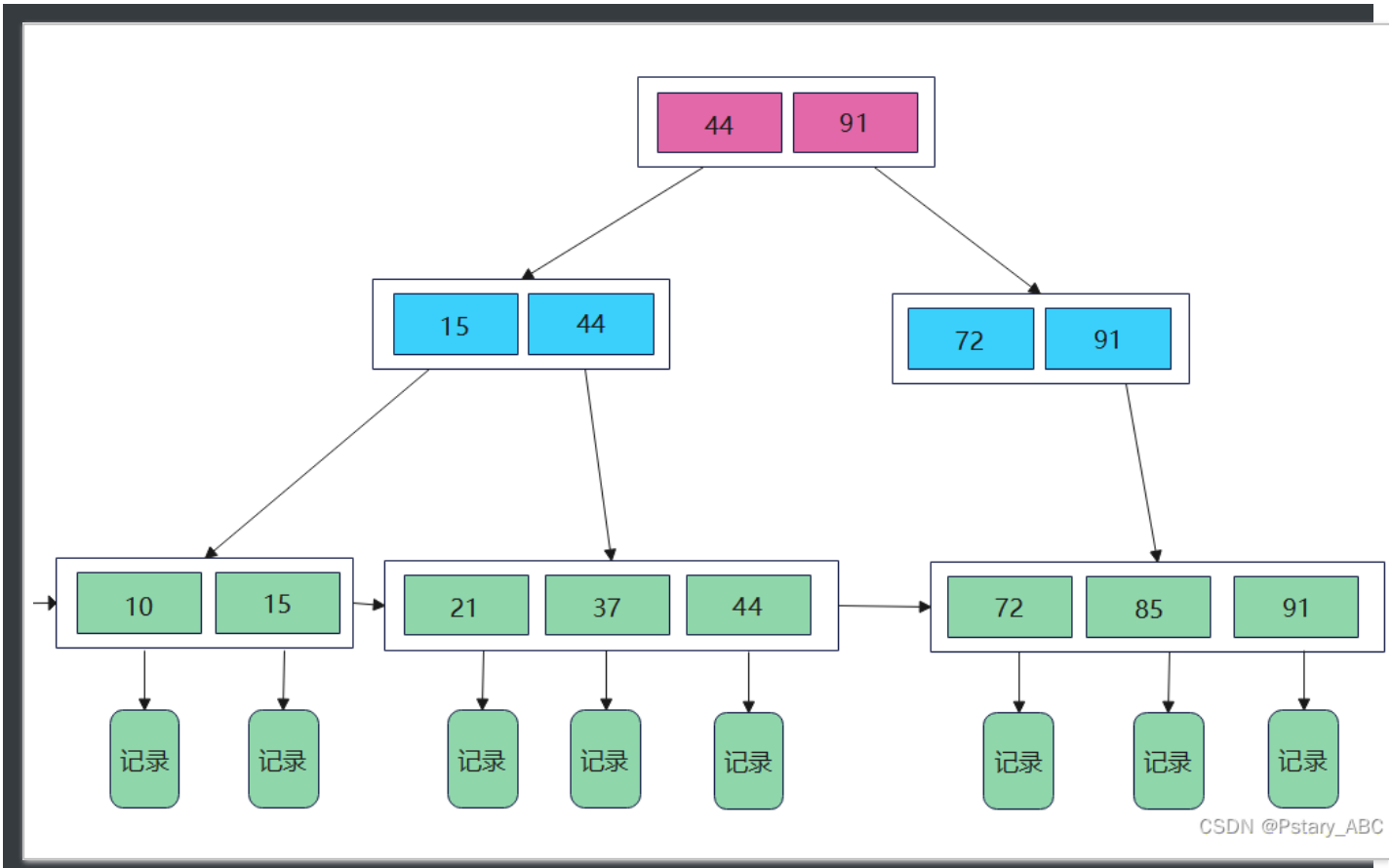
(4) 删除59，此时59所在关键字个数小于[m/2]：



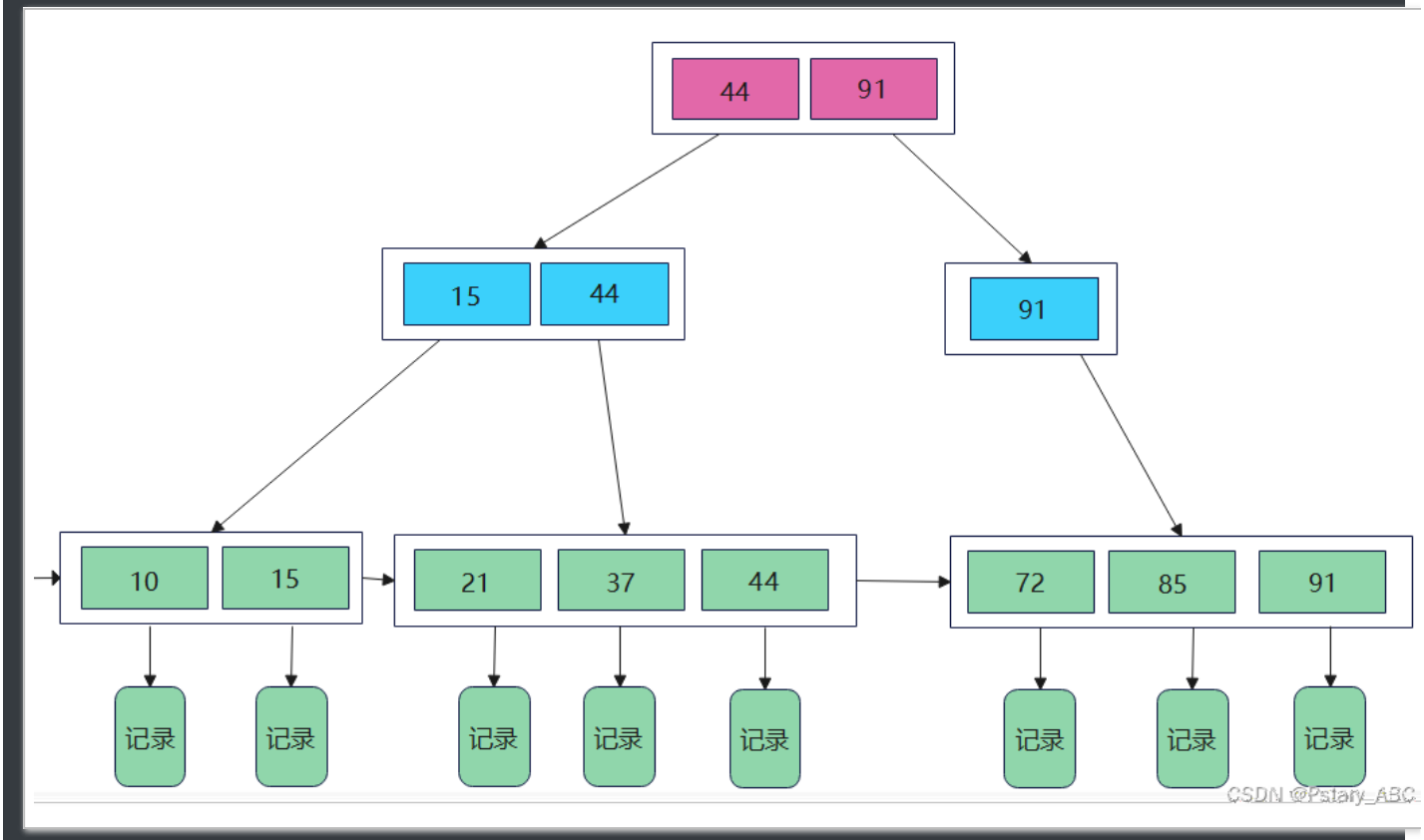
并且　　兄弟　　个数　为[m/2]，无　　它借关　字，因　将　　与兄弟　　合并，
合并之后，　　意修改　先　　关　值：



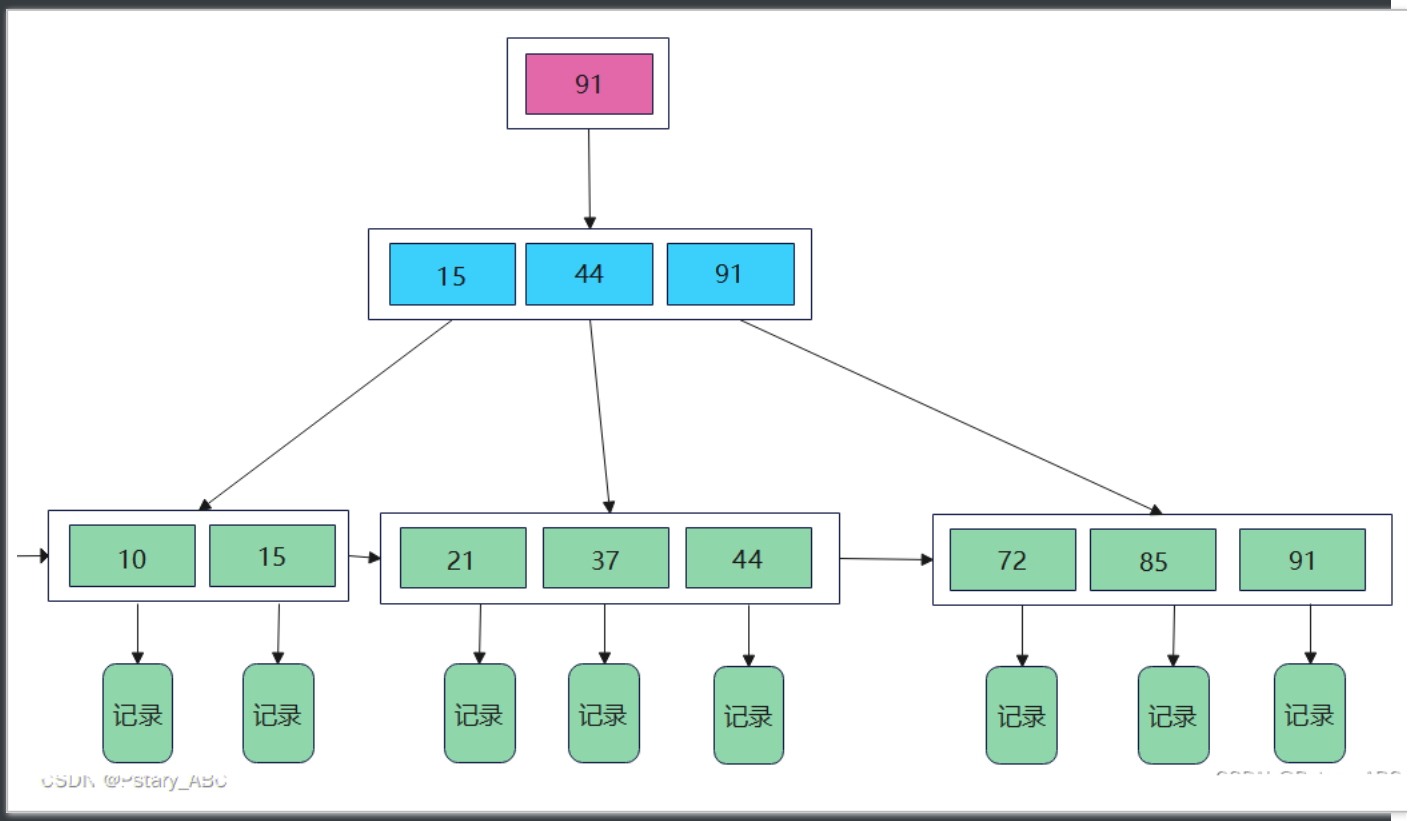
(5) 删　63，　时 63 所在　　关　字个数小于[m/2]，并且兄弟　　无　　提供关　字，
因　　和其兄弟　　合并：



时从图中可以发 ，合并后 应 删 72， 时 关 字 个数小于[m/2]：



并且 时 91 所在 兄弟 无 提供关 字，因 和兄弟 合并，并且 修



六、B + 树使用场景

- B + 是在 B 基 上改 ，它 数据 在叶子 ，同时叶子 之 加了指 形成 ，多 于 数据库索引
- 在数据库中 常不只是 (select) 条 录，如 是多条 录 ，B 做中序 历，可 层 ， B + 于所有 数据 在叶子 ，不 层，同时 于有 只 找到 尾，就 把数据 出来