

圖 5-39 新增資料進入二元搜尋樹中

由上面的程序以及範例 5-24 可知，若目前的二元搜尋樹高度為 h ，則新增資料得花 $O(h)$ 的時間找到其所應在的位置，遂新增資料至二元搜尋樹中的時間複雜之度為 $O(h)$ 。

5.9.3 自二元搜尋樹中刪除資料

欲自二元搜尋樹中刪除資料，可能會面臨兩種情況：

- (1) 刪除時節點面於樹葉節點上，如圖 5-40 所示。這是比較容易處理的情況。

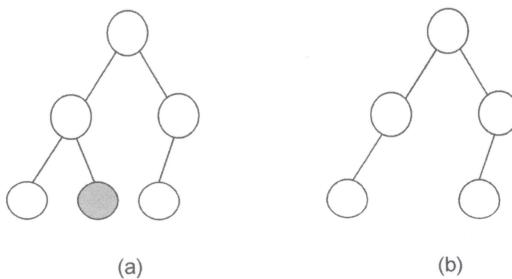


圖 5-40 刪除樹葉資料

- (2) 刪除的節點不是樹葉，如圖 5-41 (a) 所示：

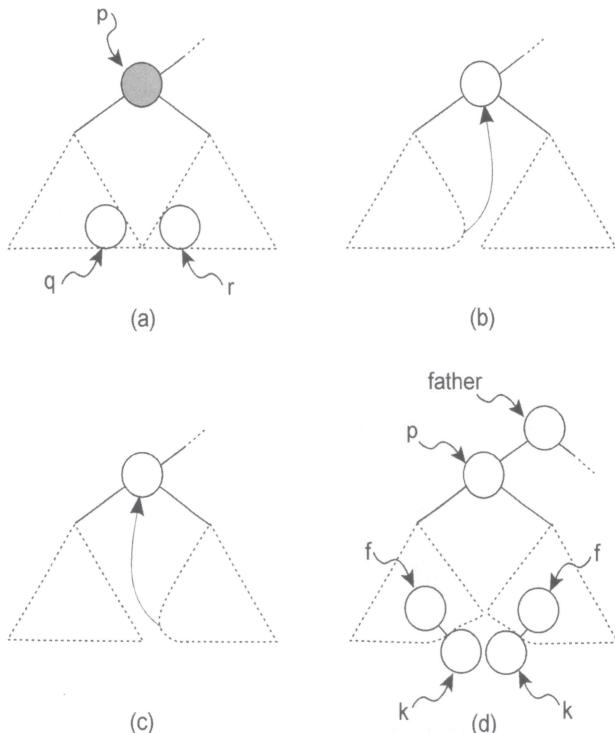


圖 5-41 刪除非樹葉節點

此時刪除節點 *p* 後，原 *p* 所指的節點可由 *q* (*q* 中包含了 *p* 左子樹中的最大節點值) 或 *r* (*r* 中包含了 *p* 右子樹中的最小節點值) 來取代，以維持二元搜尋樹中資料的大小關係；其結果分別如圖 5-41 (b) 和 (c) 所示。而圖 5-41 (d) 則標明了指標會改變的節點位置，即 *p* 會消失、*father* 的左或右子樹指標會異動、*k* (可能是 *q* 或 *r*) 會移到原來 *p* 的位置、其父節點 *f* 的左或右子樹指標會異動。其中 *father* 為 *p* 父節點，*f* 為 *k* 的父節點。

程式 5-18 解決了這個自二元搜尋樹中刪除資料的問題。

程式 5-18 自二元搜尋樹中刪除資料

```

46 int DeleteBST (int x)
47 {   Struct BSTreeNode *p , *father, *k, *f;
48     p = root; father = NULL;
49     while (p != NULL)
50     {   if (x == p->data)      //找到 x 所在的節點 p 了

```

```

51     {   if ((p->leftchild == NULL) &&
52             (p->rightchild == NULL))
53         k = NULL;           //p 為樹葉節點
54     else if (p->leftchild != NULL) //x 位於非子節點
55     {   f = p;           //找出 p 左子樹的最右樹葉
56         k = p->leftchild;
57         while (k->rightchild !=NULL)
58         {   f = k;
59             k = k->rightchild;
60         }           //k 將挪至原 p 處
61         if (p == f)
62             f->leftchild = NULL;
63         else
64             f->rightchild = NULL;
65     }
66     else           //p 無左子樹
67     {   f = p;           //找出 p 右子樹的最左樹葉
68         k = p->rightchild;
69         while (k->leftchild != NULL)
70         {   f = k;
71             k = k ->leftchild;
72         }
73         if (p == f)
74             f->rightchild = NULL;
75         else
76             f->leftchild = NULL;
77     }
78     if (father == NULL) root = k;
79     else if (k != NULL) //k 挪至原 p 處繼承 p 的左右指標
80     {   k->leftchild = p->leftchild;
81         k->rightchild = p->rightchild;
82     }
83     if (x < father->data)//決定 k 是 father 的左或右兒子
84         father->leftchild = k;
85     else
86         father->rightchild = k;
87     free(p);

```

```

88         return 1;      //成功地刪除 x，於此傳回 1 返回呼叫處
89     }
90     else           //尚未找到 x，繼續往下階層找
91     {
92         father = p;
93         if (x < p->data)
94             p = p->leftchild;
95         else
96             p = p->rightchild;
97     }
98     return 0;       //未找到 x，傳回 0
99 }
```

在程式 5-18 第 46 行中，宣告了程序 DeleteBST，傳入的參數是欲刪除的資料 x，傳回的結果是整數 0 或 1 分別表示刪除失敗或成功。第 48~49~97 行的迴圈即在找到 x 所在的節點 p、其父節點 father；而當 p 為內部節點時，再找出可取代 p 的節點 k、以及 k 的父節點 f；並完成刪除 p 的所有動作。

第 51~89 行乃找到 x 所在的節點 p 後所應處理的步驟，包括了：(1) p 為樹葉節點，執行 53~78~89 行的指令；(2) p 為內部節點，執行 54~77~78~89 行的指令。第 54~65 行是在 p 有左子樹時，取 p 左子樹的最右樹節點 k 來取代原節點 p 的位置（若 p 左子樹只有一個節點 k，在 k 挪走後，p 的左子樹指標應指向 NULL；否則 k 的父節點 f 的右子樹指標應改為指向 NULL）；若 p 沒有左子樹，則採用 67~77 行，取 p 右子樹的最左樹節點 k 來取代原節點 p 的位置（若 p 右子樹只有一個節點 k，在 k 挪走後，p 的右子樹指標應指向 NULL；否則 k 的父節點 f 的左子樹指標應改為指向 NULL）。第 90~96 行則在尚未尋覓 x 的情形下繼續尋找之。

這個程序較長，考慮的情況亦較多，請各位用心思索。