❖ 書號：AEE004000　　❖ 書名：資料結構--使用 C++(第五版)　　❖ 更新日期：2023.09.05

❖ 勘誤內容

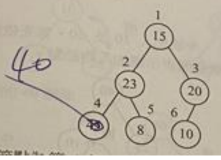
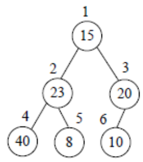
頁碼	原內容	更正後內容

3-2	C++ 片段程式》
	void Stack::push_f()
	{
	if (top >= MAX-1)
	cout << "\n\nStack is full !\n";
	else {
	top++;
	cout << "\n\n Please enter ~~au~~ an item to stack: ";
	cin >> stack[top];
	}
	}
		C++ 片段程式》
		void Stack::push_f()
		{
		if (top >= MAX-1)
		cout << "\n\nStack is full !\n";
		else {
		top++;
		cout << "\n\n Please enter an item to stack: ";
		cin >> stack[top];
		}
		}

4-15	程式解說 片段程式和隨機刪除單向鏈結串列的某一節點大同小異，其差別在於迴圈的判斷 為 while ((current != head) && (strcmp(delName,current->name) ~~==~~ != 0)) 的部份和單向鏈結串列相同，在此不再贅述。	》程式解說 此片段程式和隨機刪除單向鏈結串列的某一節點大同小異，其差別在於迴圈的判斷式為 while ((current != head) && (strcmp(delName,current->name) != 0)) 其餘的部份和單向鏈結串列相同，在此不再贅述。

4-28	class Single_link_list {
	private:
	Node_type *ptr;
	Node_type *head;
	Node_type *tail;
	~~Node_type *current;~~ 去掉
	Node_type *prev;
	
	public:
	Single_link_list();
	void insert_f(void);
	void delete_f(void);
	void display_f(void);
	void modify_f(void);
	};
	
	Single_link_list::Single_link_list()
	{
	head = new Node_type;
	head->next = NULL;
	~~tail = NULL;~~ 去掉
	}
	
	void Single_link_list::insert_f(void)
	{
	char s_temp[4];
	ptr = new Node_type;
	cout << " Student name : ";
	cin>>ptr->name;
		class Single_link_list {
		private:
		Node_type *ptr;
		Node_type *head;
		Node_type *tail;
		Node_type *prev;
		
		public:
		Single_link_list();
		void insert_f(void);
		void delete_f(void);
		void display_f(void);
		void modify_f(void);
		};
		
		Single_link_list::Single_link_list()
		{
		head = new Node_type;
		head->next = NULL;
		}
		
		void Single_link_list::insert_f(void)
		{
		char s_temp[4];
		ptr = new Node_type;
		cout << " Student name : ";
		cin>>ptr->name;
		cout << " Student score: ";

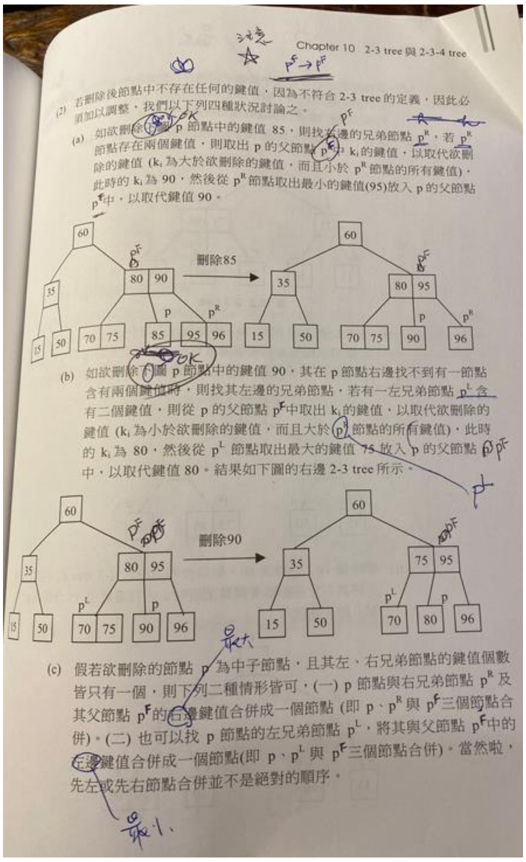
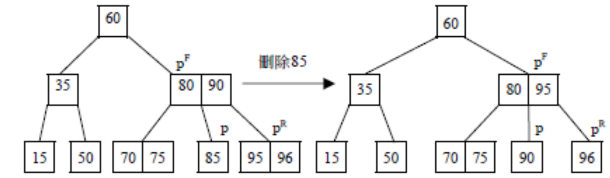
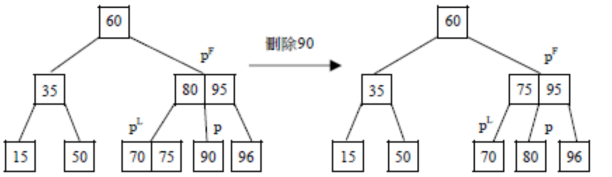
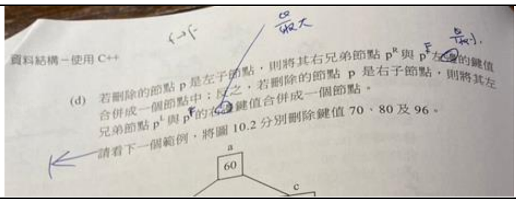
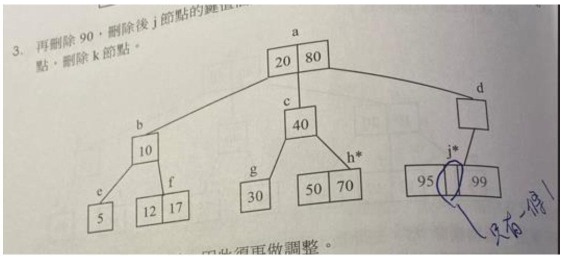
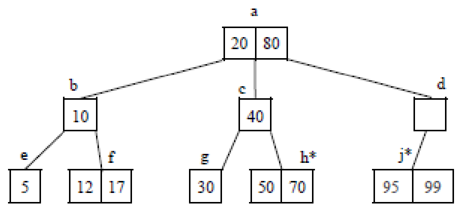
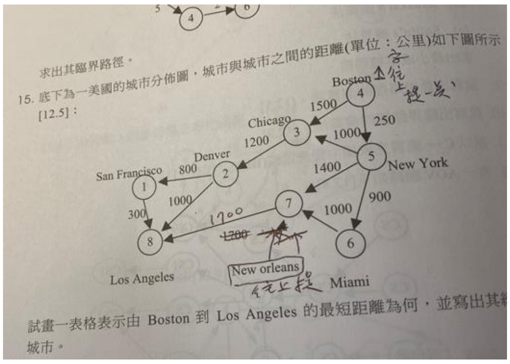
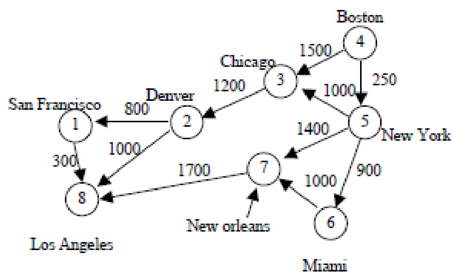
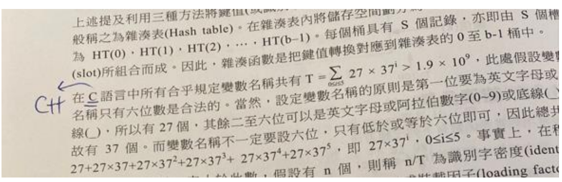
4-29	if (head->next == NULL)
	cout << " No student recond\n"; // 無資料顯示錯誤
	else {
	cout << " Want to modif student name: ";
	cin>>n_temp; 加 y
	prev = head;
	current = head->next;
	while ((current != NULL) && (strcmp(current->name, n_temp) != 0)) {
		if (head->next == NULL)
		cout << " No student recond\n"; // 無資料顯示錯誤
		else {
		cout << " Want to modify student name: ";
		cin>>n_temp;
		prev = head;
		current = head->next;
		while ((current != NULL) && (strcmp(current->name, n_temp) != 0)) {

7-11	其中 20->10->30 和 20->30->10 所建立的二元搜尋樹是一樣的，所以有五種不同的二元搜尋樹。若將鍵值以內部成功節點表示，而失敗節點以外部節點方式表示的話，這五種二元搜尋樹所對應的內部與外部之完整圖形，我們以 a'、b'、c'、d'，以及 f' 表示之，如圖 7.5 所示： a', b', c', d', e f' 其中，與下圖之相同 字型 7-11	其中 20->10->30 和 20->30->10 所建立的二元搜尋樹是一樣的，所以有五種不同的二元搜尋樹。若將鍵值以內部成功節點表示，而失敗節點以外部節點方式表示的話，這五種二元搜尋樹所對應的內部與外部之完整圖形，我們以 a'、b'、c'、d'，以及 f' 表示之，如圖 7.5 所示：
7-13	cost(d) = ((1/7)*1+(1/7)*2+(1/7)*3+(1/7)*1+(1/7)*2+(1/7)*3+(1/7)*3) = 15/7 字型，請看 P7-12 圖7.5 由以上得知，b' 的成本是最少的，也就是此棵的二元搜尋樹是最佳的。 但如果條件改變的話，如 p_1=0.4, p_2=0.2, p_3=0.1, q_0=0.1, q_1=0.1, q_2= 0.05, q_3=0.05，則 cost(a) = 0.4*1+0.2*2+0.1*3+0.1*1+0.1*2+0.05*3+0.05*3 = 1.8， cost(b) = 0.2*1+0.4*2+0.1*3+0.1*2+0.1*2+0.05*2+0.05*2 = 1.9， cost(c) = 0.1*1+0.2*2+0.4*3+0.1*3+0.1*3+0.05*2+0.05*1 = 2.45， cost(d) = 0.4*1+0.1o*2+0.2*3+0.1*1+0.1*3+0.05*3+0.05*2 = 1.85， cost(e) = 0.1*1+0.4*2+0.2*3+0.1*2+0.1*3+0.05*3+0.05*1 = 2.0 字型， 由上述所計算的成本得知，a' 的成本最少，所以它是在上述條件下最佳的二元搜尋樹。	由以上得知，b' 的成本是最少的，也就是此棵的二元搜尋樹是最佳的。 但如果條件改變的話，如 p_1=0.4, p_2=0.2, p_3=0.1, q_0=0.1, q_1=0.1, q_2= 0.05, q_3=0.05，則 cost(a) = 0.4*1+0.2*2+0.1*3+0.1*1+0.1*2+0.05*3+0.05*3 = 1.8， cost(b) = 0.2*1+0.4*2+0.1*3+0.1*2+0.1*2+0.05*2+0.05*2 = 1.9， cost(c) = 0.1*1+0.2*2+0.4*3+0.1*3+0.1*3+0.05*2+0.05*1 = 2.45， cost(d) = 0.4*1+0.1o*2+0.2*3+0.1*1+0.1*3+0.05*3+0.05*2 = 1.85， cost(e) = 0.1*1+0.4*2+0.2*3+0.1*2+0.1*3+0.05*3+0.05*1 = 2.0 由上述所計算的成本得知，a' 的成本最少，所以它是在上述條件下最佳的二元搜尋樹。
8-3	共有 6 個節點，節點上方分別標記第 1 個，第 2 個，…，等等的記號，以由下而上處理的方法，首先 $\lfloor 6/2 \rfloor$ 為 3，故由第 3 個節點開始，第 3 個節點的子節點分別為第 6 個節點和第 7 個節點(此題沒有第 7 個節點)，故以第 6 節點的 20 和第 3 節點的 10 相比，20 大於 10，故要交換，情形如下： 	共有 6 個節點，節點上分別標記第 1 個，第 2 個，…，等等的記號，以由下而上處理的方法，首先 $\lfloor 6/2 \rfloor$ 為 3，故由第 3 個節點開始，第 3 個節點的子節點分別為第 6 個節點和第 7 個節點(此題沒有第 7 個節點)，故以第 6 節點的 20 和第 3 節點的 10 相比，20 大於 10，故要交換，情形如下： 

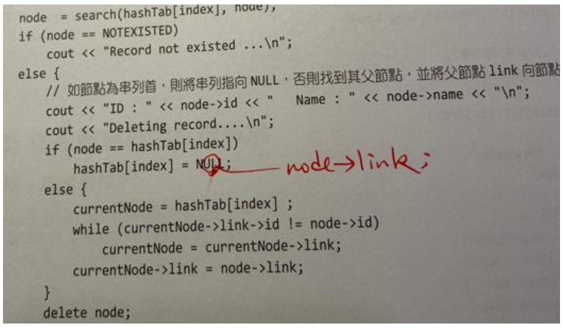
9-18	`pivot = pivot_find();`
	`if (pivot != NULL) {      // PIVOT 存在，則須改善為 AVL-TREE`
	`    op = type_find();`
	`    switch(op) {`
	`        case 11: type_ll();`
	△空一格
		`if (pivot != NULL) {      // PIVOT 存在，則須改善為 AVL-TREE`
		`    op = type_find();`
		`    switch (op) {`
		`        case 11: type_ll();`

9-20	`}`
	`bf_count(root);`
	`if (root != NULL) {    // 若根不存在，則無需作平衡改善`
	`    pivot = pivot_find();     // 尋找 PIVOT 所在節點`
	`    while (pivot != NULL) {`
	`        op = type_find();`
	`        switch(op) {`
	`            case 11: type_ll();`
	△空一格
		`}`
		`bf_count(root);`
		`if (root != NULL) {    // 若根不存在，則無需作平衡改善`
		`    pivot = pivot_find();     // 尋找 PIVOT 所在節點`
		`    while (pivot != NULL) {`
		`        op = type_find();`
		`        switch (op) {`
		`            case 11: type_ll();`

9-22	`int Avltree::height_count(Node_type *trees)`
	`{`
	`    if (trees == NULL)`
	`        return 0;`
	`    else if(trees->llink == NULL && trees->rlink == NULL)`
	`        return 1;`
	`    else if (height_count(trees->llink) > height_count(trees->rlink))`
	`        return (1 + height_count(trees->llink));`
	`    else`
	`        return (1 + height_count(trees->rlink));`
	△空一格
		`int Avltree::height_count(Node_type *trees)`
		`{`
		`    if (trees == NULL)`
		`        return 0;`
		`    else if (trees->llink == NULL && trees->rlink == NULL)`
		`        return 1;`
		`    else if (height_count(trees->llink) > height_count(trees->rlink))`
		`        return (1 + height_count(trees->llink));`
		`    else`
		`        return (1 + height_count(trees->rlink));`

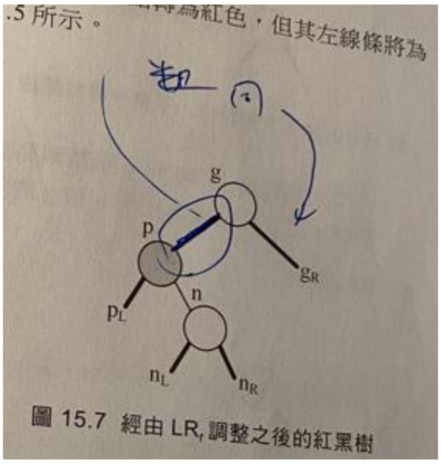
10-5		(2) 若刪除後節點中不存在任何的鍵值，因為不符合 2-3 tree 的定義，因此必須加以調整，我們以下列四種狀況討論之。 (a) 如欲刪除下圖 p 節點中的鍵值 85，則找右邊的兄弟節點 p^R，若 p^R 節點存在兩個鍵值，則取出 p 的父節點 p^F 中 k_i 的鍵值，以取代欲刪除的鍵值 (k_i 為大於欲刪除的鍵值，而且小於 p^R 節點的所有鍵值)，此時的 k_i 為 90，然後從 p^R 節點取出最小的鍵值(95)放入 p 的父節點 p^F 中，以取代鍵值 90。  (b) 如欲刪除下圖 p 節點中的鍵值 90，其在 p 節點右邊找不到有一節點含有兩個鍵值時，則找其左邊的兄弟節點，若有一左兄弟節點 p^L 含有二個鍵值，則從 p 的父節點 p^F 中取出 k_i 的鍵值，以取代欲刪除的鍵值 (k_i 為小於欲刪除的鍵值，而且大於 p^L 節點的所有鍵值)，此時的 k_i 為 80，然後從 p^L 節點取出最大的鍵值 75 放入 p 的父節點 p^F 中，以取代鍵值 80。結果如下圖的右邊 2-3 tree 所示。  (c) 假若欲刪除的節點 p 為中子節點，且其左、右兄弟節點的鍵值個數皆只有一個，則下列二種情形皆可，(一) p 節點與右兄弟節點 p^R 及其父節點 p^F 的最大邊鍵值合併成一個節點 (即 p、p^R 與 p^F 三個節點合併)。(二) 也可以找 p 節點的左兄弟節點 p^L，將其與父節點 p^F 中的最小鍵值合併成一個節點(即 p、p^L 與 p^F 三個節點合併)。當然啦，先左或先右節點合併並不是絕對的順序。
10-6		(d) 若刪除的節點 p 是左子節點，則將其右兄弟節點 p^R 與 p^F 最小的鍵值合併成一個節點中；反之，若刪除的節點 p 是右子節點，則將其左兄弟節點 p^L 與 p^F 的最大鍵值合併成一個節點。 請看下一個範例，將圖 10.2 分別刪除鍵值 70、80 及 96。
10-10		3. 再刪除 90，刪除後 j 節點的鍵值個數為 0，將 k 節點與其父節點 d 合併成 j*節點，刪除 k 節點。 
12-64		13. 底下為一美國的城市分佈圖，城市與城市之間的距離(單位：公里)如下圖所示 [12.5]： 
14-4		在 C++ 語言中所有合乎規定變數名稱共有 $T=\sum_{0\le i\le 5} 27\times 37^i > 1.9\times 10^9$，此處假設變數名稱只有六位數是合法的。當然，設定變數名稱的原則是第一位要為英文字母或底線(_)，所以有 27 個，其餘二至六位可以是英文字母或阿拉伯數字(0~9)或底線

14-22




```
node  = search(hashTab[index], node);
if (node == NOTEXISTED)
    cout << "Record not existed ...\n";
else {
    // 如節點為串列首，則將串列指向 NULL，否則找到其父節點，並將父節點 link 向節點後端
    cout << "ID : " << node->id << "    Name : " << node->name << "\n";
    cout << "Deleting record....\n";
    if (node == hashTab[index])
        hashTab[index] = node->link;
    else {
        currentNode = hashTab[index] ;
        while (currentNode->link->id != node->id)
            currentNode = currentNode->link;
        currentNode->link = node->link;
    }
    delete node;
```

15-3



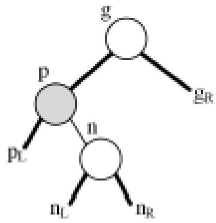
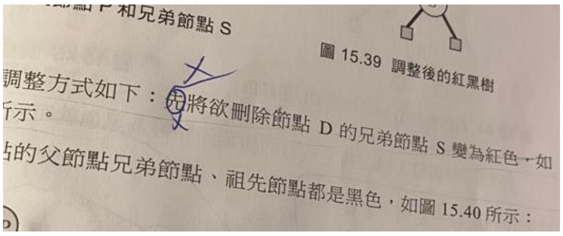


圖 15.7 經由 LR_r 調整之後的紅黑樹

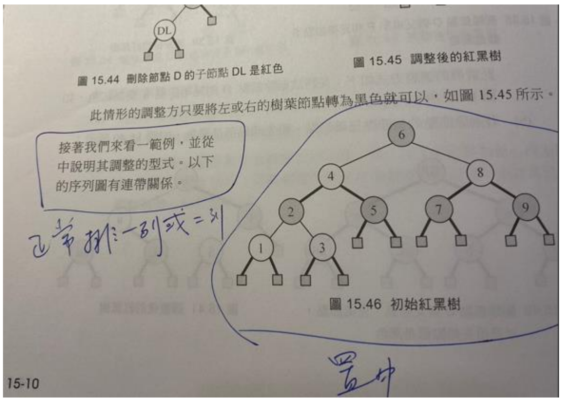
15-9



此情形的調整方式如下：將欲刪除節點 D 的兄弟節點 S 變為紅色，如圖 15.39 所示。

(b) 若刪除節點的父節點兄弟節點、祖先節點都是黑色，如圖 15.40 所示：

15-10



此情形的調整方只要將左或右的樹葉節點轉為黑色就可以，如圖 15.45 所示。

接著我們來看一範例，並從中說明其調整的型式。以下的序列圖有連帶關係。

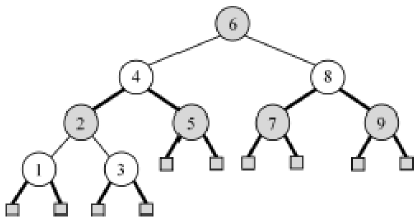
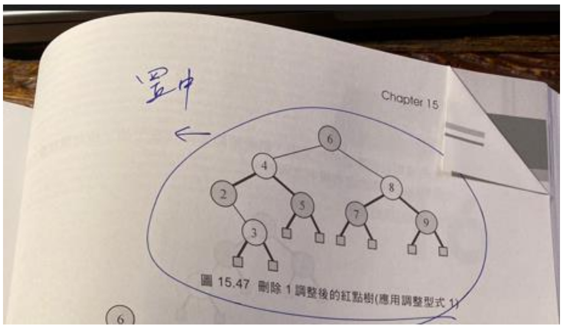


圖 15.46 初始紅黑樹

15-11



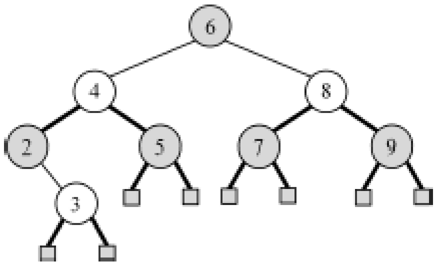
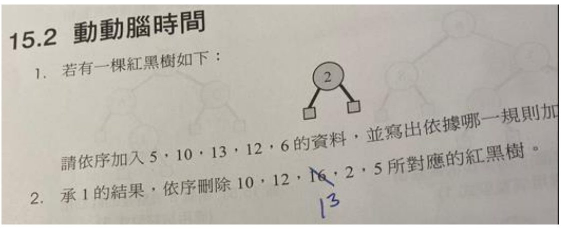
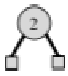
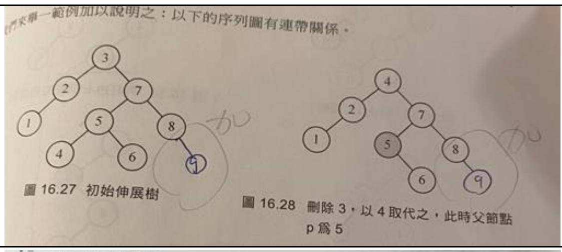
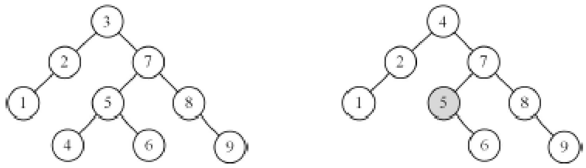
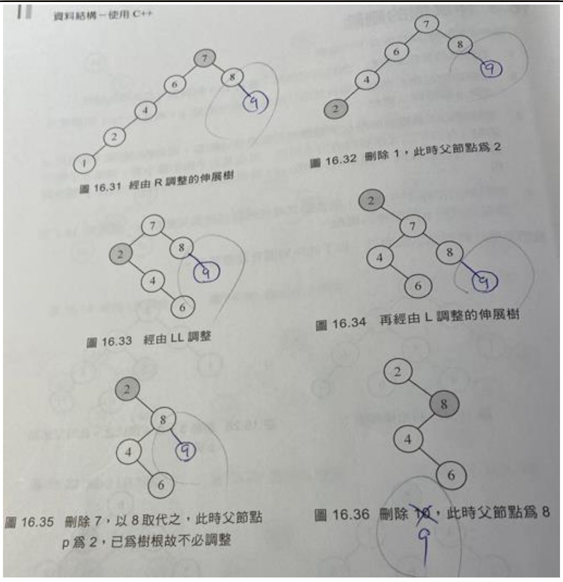
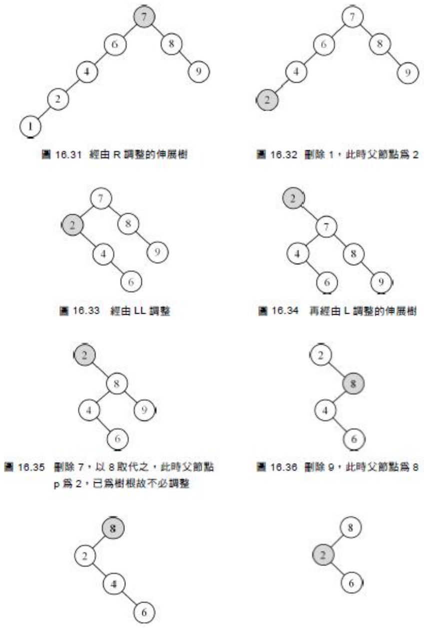
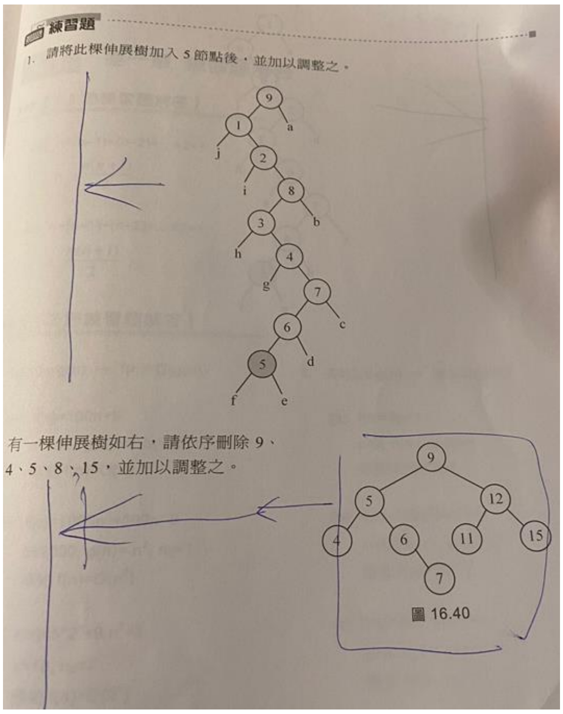
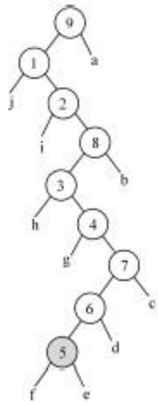
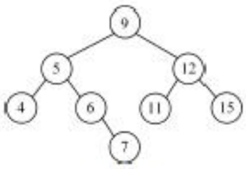
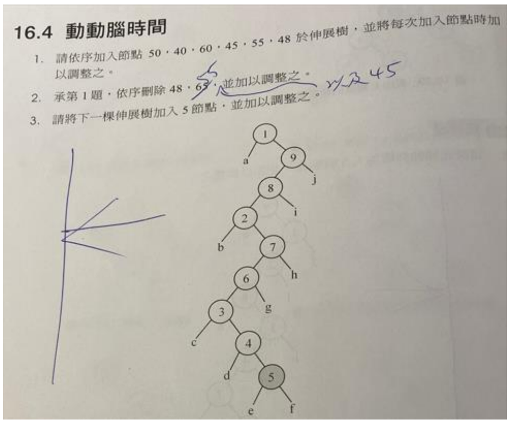


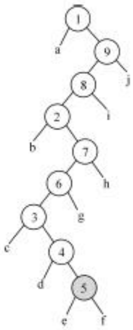
圖 15.47 刪除 1 調整後的紅點樹(應用調整型式 1)

15-12	 15.2 動動腦時間 1. 若有一棵紅黑樹如下： 請依序加入 5，10，13，12，6 的資料，並寫出依據哪一規則加 2. 承 1 的結果，依序刪除 10，12，~~16~~ 13，2，5 所對應的紅黑樹。	 請依序加入 5，10，13，12，6 的資料，並寫出依據哪一規則加以調整的。 2. 承 1 的結果，依序刪除 10，12，13，2，5 所對應的紅黑樹。
16-5	 以下的序列圖有連帶關係。 圖 16.27 初始伸展樹 圖 16.28 刪除 3，以 4 取代之，此時父節點 p 為 5	 圖 16.27 初始伸展樹 圖 16.28 刪除 3，以 4 取代之，此時父節點 p 為 5
16-6	 資料結構－使用 C++ 圖 16.31 經由 R 調整的伸展樹 圖 16.32 刪除 1，此時父節點為 2 圖 16.33 經由 LL 調整 圖 16.34 再經由 L 調整的伸展樹 圖 16.35 刪除 7，以 8 取代之，此時父節點 p 為 2，已為樹根故不必調整 圖 16.36 刪除 ~~10~~ 9，此時父節點為 8	 圖 16.31 經由 R 調整的伸展樹 圖 16.32 刪除 1，此時父節點為 2 圖 16.33 經由 LL 調整 圖 16.34 再經由 L 調整的伸展樹 圖 16.35 刪除 7，以 8 取代之，此時父節點 p 為 2，已為樹根故不必調整 圖 16.36 刪除 9，此時父節點為 8
16-7	 練習題 1. 請將此棵伸展樹加入 5 節點後，並加以調整之。 有一棵伸展樹如右，請依序刪除 9、4、5、8、15，並加以調整之。 圖 16.40	1. 請將此棵伸展樹加入 5 節點後，並加以調整之。  2. 有一棵伸展樹如右，請依序刪除 9、4、5、8、15，並加以調整之。  圖 16.40

<table>
<tr>
<td>16-8</td>
<td>
16.4 動動腦時間
1. 請依序加入節點 50、40、60、45、55、48 於伸展樹，並將每次加入節點時加以調整之。
2. 承第 1 題，依序刪除 48、65 55，並加以調整之。 以及45
3. 請將下一棵伸展樹加入 5 節點，並加以調整之。</td>
<td>

16.4 動動腦時間

1. 請依序加入節點 50、40、60、45、55、48 於伸展樹，並將每次加入節點時加以調整之。
2. 承第 1 題，依序刪除 48、55 以及 45，並加以調整之。
3. 請將下一棵伸展樹加入 5 節點，並加以調整之。



</td>
</tr>
</table>