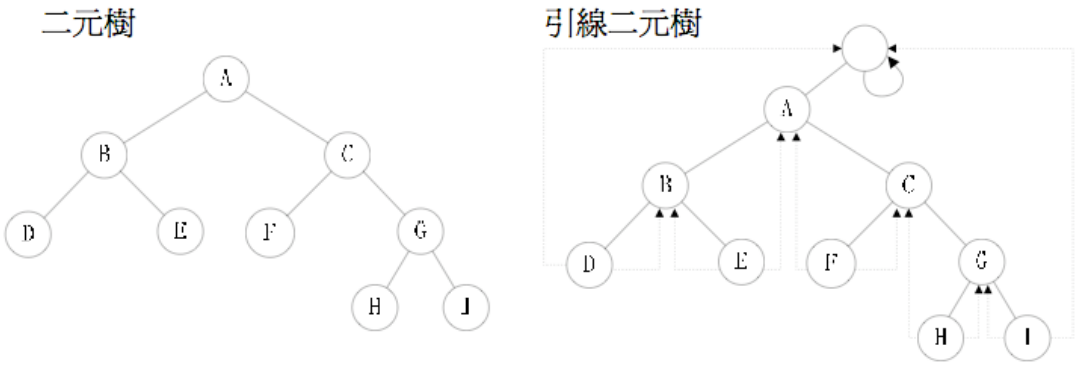
4-49：

A. 區別



把二元樹以中序追蹤法將所有節點排列好，再將節點的空指標指向其左、右相鄰節點。即二元樹鏈結結構表示法中左、右指標指向NULL的指標使用引線方式取代。

將原本浪費掉的指標廢物利用，讓 In-Order Traversal 更迅速
Left-Child -> In-Order Predecessor
Right-Child -> In-Order Successor

用一個額外的 Bit 判斷指標內容是 Link 還是 Thread
引線二元樹的資料結構如下：

LBit	LChild	Data	RChild	RBit

當LBit=1, LChild是正常指標
當LBit=0, LChild是引線
當RBit=1, RChild是正常指標
當RBit=0, RChild是引線

B. 實作

資料宣告

```
//定義資料結構
//rbit = 1 ,rchild為正常指標,rbit =0 ,rchild為引線
//lbit = 1 ,lchild為正常指標,lbit =0 ,lchild為引線

struct tbintree {
    int number;
    int rbit;
    int lbit;
    struct tbintree *lchild;
```

```
        struct tbintree *rchild;
} *root, *ptr, *newnode;

typedef struct t_tree treenode;
typedef treenode *t_btree;
```

//建立方式

- 引線二元樹的建立原則是中序走訪的順序。中序走訪的原則，如下所示：
  - 如果任何節點的右指標不是引線，就需要從右子節點為起點，沿著左子樹的路徑往下走直到左指標是引線為止。
  - 如果任何節點的右指標是引線，引線所指的節點就是下一個中序走訪的節點。

Insertion：

```
/*插入節點函數*/
void insert()
{
        newnode = newtbintree();
        printf("Enter a number to insert : ");
        scanf("%d", &newnode->number);
        if (root->lchild == root) {
                insert_left(root, newnode);
                puts("node insert ok!");
        } else {
                ptr = root->lchild;
                while (ptr->number != newnode->number) {
                        /*    如新節點小於目前節點且lbit為0 (lchild為引線)
                              則插入目前節點左方,否則ptr往左搜尋    */
                        if (newnode->number < ptr->number) {
                                if (ptr->lbit == 0) {
                                        insert_left(ptr, newnode);
                                        break;
                                } else
                                        ptr = ptr->lchild;
                        }
                        /*    如新節點大於目前節點且rbit為0 (rchild為引線)
                              則插入目前節點右方,否則ptr往右搜尋    */
                        else if (newnode->number > ptr->number) {
```

```
                    if (ptr->rbit == 0) {
                        insert_right(ptr, newnode);
                        break;
                    } else
                        ptr = ptr->rchild;
                }
            }
            if (ptr->number == newnode->number) {
                puts("Number existed ...!");
                return;
            } else
                puts("node insert ok!");
        }
}


/*  加入節點於右方函數  */
/*  傳入參數: */
/* 1. node_parent  為新節點之父節點  */
/* 2. node  為欲新增之節點  */
void insert_right(struct tbintree *node_parent, struct tbintree *node)
{
        struct tbintree *w;
        node->rchild = node_parent->rchild;
        node->rbit = node_parent->rbit;
        node->lchild = node_parent;
        node->lbit = 0;
        node_parent->rchild = node;
        node_parent->rbit = 1;

        if (node->rbit == 1) {          /*node底下還有tree */
            w = insucc(node);
            w->lchild = node;
        }
}

/*  加入節點於左方函數  */
/*  傳入參數: */
```

```
/* 1. node_parent  為新節點之父節點  */
/* 2. node  為欲新增之節點  */

void insert_left(struct tbintree *node_parent, struct tbintree *node)
{
        struct tbintree *w;

        node->lchild = node_parent->lchild;
        node->lbit = node_parent->lbit;
        node->rchild = node_parent;
        node->rbit = 0;
        node_parent->lchild = node;
        node_parent->lbit = 1;

        if (node->lbit == 1) {          /*node  底下還有tree */
                w = inpred(node);
                w->rchild = node;
        }
}
```

Deletion：

```
/*刪除節點函數*/
void delete_node()
{
        struct tbintree *ptr_parent;
        struct tbintree *ptr_pred, *ptr_succ;
        int num;

        /*引線二元樹從root->lchild開始放資料  */
        if (root->lchild == root) {
                puts("No Data!");
                return;
        }
        printf("Enter a number u want to delete : ");
        scanf("%d", &num);

        ptr_parent = root;
        ptr = root->lchild;
```

```
while (ptr->number != num) {              /*搜尋二元樹直到找到節點  */
      ptr_parent = ptr;

      /*如num值小於目前節點且lbit為1 */
      /*(lchild為正常指標)則往左搜尋  */
      if (num < ptr->number) {
            if (ptr->lbit == 1) /*否則(lchild為引線)即找不到節點  */
                  ptr = ptr->lchild;
            else
                  break;
      } else if (num > ptr->number) {
            if (ptr->rbit == 1)
                  ptr = ptr->rchild;
            else
                  break;
      }
}
if (ptr->number != num) {
      puts("Not found number!");
      return;
}
printf("Deleting number %d...\n", ptr->number);

/*  刪除樹葉節點  */
if (ptr->lbit == 0 && ptr->rbit == 0) {
      if (ptr_parent == root) {           /*刪除第一個節點  */
            ptr_parent->lchild = root;
            ptr_parent->lbit = 0;
      } /*刪除左節點  */
      else if (ptr->number < ptr_parent->number) {
            ptr_parent->lchild = ptr->lchild;
            ptr_parent->lbit = 0;
      } else { /*刪除右節點  */

            ptr_parent->rchild = ptr->rchild;
            ptr_parent->rbit = 0;
      }
      free(ptr);
```

```
        }
        /*  刪除有兩分支度節點  */
        else if (ptr->lbit == 1 && ptr->rbit == 1) {
            /*求ptr的前行者節點,將右子樹插入前行者右方  */
            ptr_pred = inpred(ptr);

            ptr_pred->rchild = ptr->rchild;
            ptr_pred->rbit = ptr->rbit;
            ptr_parent->lchild = ptr->lchild;
            free(ptr);
        } else {    /*刪除一分支度節點  */

            if (ptr_parent == root) {    /*刪除第一節點  */
                if (ptr->lbit == 1) {
                    ptr_pred = inpred(ptr);
                    root->lchild = ptr->lchild;
                    ptr_pred->rchild = root;
                } else {
                    ptr_succ = insucc(ptr);
                    root->lchild = ptr->rchild;
                    ptr_succ->lchild = root;
                }
            } else {
                if (ptr->number < ptr_parent->number)
                    ptr_parent->lchild = ptr->lchild;
                else
                    ptr_parent->rchild = ptr->rchild;
            }
        }
    }
```

C.　好處

引線二元樹的追蹤，可以由某一節點知道其前行者與後繼者是那一個節點，因此可以不需使用到遞迴呼叫，也因此不需要使用堆疊，加快搜尋速度。