115年公務人員高等考試三級考試試題

類　　科：資訊處理
科　　目：資料結構
考試時間：2小時　　座號：_____

※注意：㈠禁止使用電子計算器。
㈡不必抄題，作答時請將試題題號及答案依照順序寫在試卷上，於本試題上作答者，不予計分。
㈢本科目除專門名詞或數理公式外，應使用本國文字作答。

一、某系統A使用雜湊表（hash table）儲存不同的正整數鍵值，亦即不允許重複鍵值，雜湊表有11個儲存格，索引從0開始，雜湊函數（hash function）為$h^A(k)=k\ mod\ 11$，其用平方探查法（quadratic probing）處理碰撞（collision）問題，探查序列為$h_i^A(k)=(h^A(k)+i^2)\ mod\ 11\ (i=0,1,2,...)$，刪除資料時，被刪除資料的位置標記為特殊符號DELETED。請回答下列問題：

㈠給定一個空的雜湊表，依序插入下列鍵值22、1、13、24、35、46、7、18，請畫出所有鍵值插入完成後的雜湊表狀態，並列出插入鍵值46時的完整探查過程。（10分）

㈡承上題，依序刪除鍵值24、13，畫出刪除後的雜湊表狀態。並說明為什麼刪除鍵值時需用DELETED標記，而不能將該鍵值所在的儲存格恢復成「從未存放過鍵值」的空狀態。（5分）

㈢承上題，執行插入鍵值12，請列出插入時的探查過程、操作停止的理由，並寫出12最後插入那一個儲存格。插入時，DELETED標記視為可放入新鍵值的儲存格。請注意鍵值不能重覆。（5分）

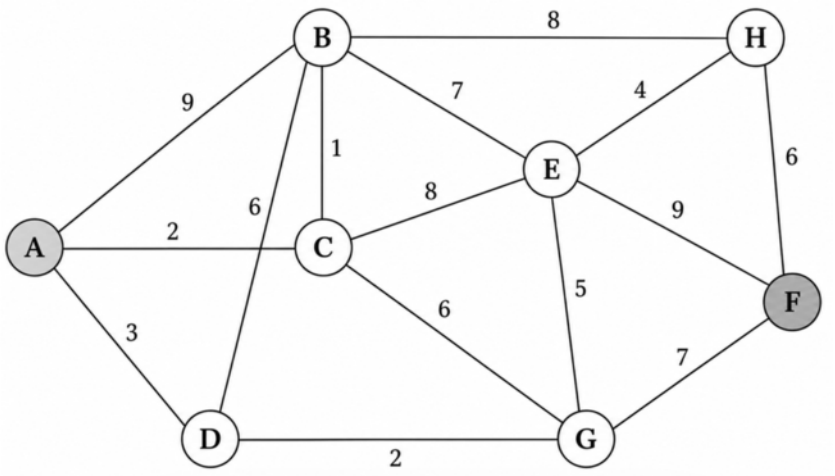
㈣相較於系統A，考慮另一個採用平方探查法之系統，系統B的表格大小為8，索引亦從0開始，雜湊函數$h^B(k)$及探查序列$h_i^B(k)$分別定義為$h^B(k)=k\ mod\ 8$、$h_i^B(k)=(h^B(k)+i+2\cdot i^2)\ mod\ 8\ (i=0,1,2,...)$。在非均勻雜湊（non-uniform hashing）的情況下，也就是許多鍵值可能被分配到相同或少數幾個初始雜湊位置時，那一個系統的雜湊表儲存格利用率可能較高？並說明理由。（5分）

二、給定一個無向圖 $G=(V, E)$，每個頂點代表一個地點，每條邊 e $(e \in E)$ 代表一條道路，邊的正整數權重 $\omega(e)$ 表示該道路的塞車程度，數值越大越壅塞。對於一條從起點 s 到終點 $t(s, t \in V)$ 的路徑 P，其最大塞車程度 C(P) 定義為路徑上所有邊權重的最大值：

$$C(P)=\max_{e \in P} \omega(e)$$

本題透過修改 Dijkstra 最短路徑演算法中陣列 d 的定義與更新方式，求出從 s 到 t 可行路徑所能達到的「最大塞車程度的最小值」。修改後的演算法流程與 Dijkstra 最短路徑演算法相同，差異僅在於 $d[v](v \in V)$ 的定義與更新規則，其中，新的 $d[v]$ 表示目前已知從 s 到 v 的路徑中，最大邊權重的最小值。初始時令 $d[s]=0$，其他頂點 v 的 $d[v]=\infty(v \neq s)$。之後依照 Dijkstra 演算法，每一輪選出尚未被選定且 d 值最小的頂點 u，並將原本的更新方式 $d[v]=\min(d[v], d[u]+\omega(u,v))$ 改為 $d[v]=\min(d[v], \max(d[u], \omega(u,v)))$，其中 $\omega(u,v)$ 為邊 (u,v) $(u, v \in V)$ 的權重。重複進行，直到終點 t 被選定為止。

㈠以下列無向圖為例，令起點 s 為 A，終點 t 為 F，依照修改後的演算法，逐步列出每次選定一個頂點後陣列 d 的變化過程。陣列中的頂點順序請依字母順序排列。（15 分）



㈡說明修改後演算法之正確性，是基於 $d[v]$ 更新規則具有何種性質。（5 分）

㈢假設圖以相鄰串列（adjacency list）表示。若要在尚未選定的頂點中找出 d 值最小者，可使用以下兩種方法：

方法一：每次以線性方式掃描所有尚未選定的頂點找出最小 d 值。

方法二：使用最小堆積（min-heap）維護目前 d 值最小的頂點。

分別就這兩種方法，分析修改後演算法最壞情況的時間複雜度。（5 分）

三、給定一棵二元搜尋樹（binary search tree），且該樹同時也是一棵 AVL 樹。樹的節點在 C 語言中宣告如下：

```
typedef struct Node {
    int key;       // 節點的鍵值，所有節點的鍵值皆互不相同
    int size;     // 以該節點為根的子樹節點總數 (包含自己)
    struct Node *left;    // 指向左子節點
    struct Node *right;  // 指向右子節點
} Node;
```

並定義以下函式：

int size (Node **node*)：

若傳入的 *node* 為 NULL，則回傳 0；否則回傳 *node* -> size。

int count_less_equal (Node **node*, int *val*)：

回傳以 *node* 為根的子樹中，所有鍵值小於等於 *val* 的節點總數。

Node* select (Node **node*, int *r*)：

回傳以 *node* 為根的子樹中，第 *r* 小的節點指標，*r* 從 1 開始算。

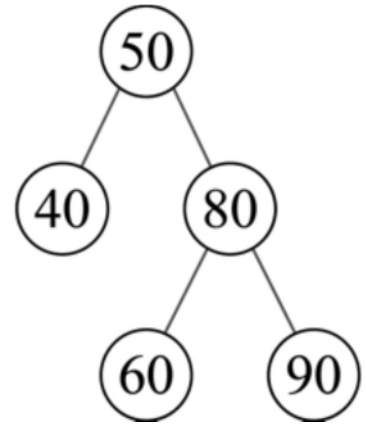
Node* greater_k_smallest (Node **root*, int *val*, int *k*)：

找出以 *root* 為根的整棵樹中，所有鍵值大於 *val* 的節點裡，第 *k* 小的節點，*k* 從 1 開始算。若第 *k* 小的節點不存在，則回傳 NULL。

㈠完成下列程式碼的空格。（20 分）

```
int count_less_equal(Node *node, int val) {
   if (node == NULL) return 0;
   if (node->key > val)
        return count_less_equal(node->left, val);
   else return size(node->left)+ ____(1)____;
}
 Node* select(Node *node, int r) {
   int left_size = size(node->left);
   if (r == ____(2)____) return node;
   else if (r <= left_size)
        return select(node->left, r);
   else return _____(3)_____ ;
}
 Node* greater_k_smallest(Node *root, int val, int k){
   int x = count_less_equal(root, val);
   int y = ____(4)_____ ;
   if (y > size(root)) return NULL;
   return select(root, y);
}
```

㈡下圖為一棵包含 5 個節點且滿足 AVL 平衡特性的二元搜尋樹，圖中顯示每個節點的鍵值。若將鍵值為 70 的新節點插入此樹，為保持 AVL 樹的平衡，會觸發旋轉。請畫出旋轉後的樹狀結構圖。除新插入的節點 70 外，若原有節點的 size 欄位值在旋轉後發生改變，請在旋轉後的圖中，於該節點旁標示其新的 size 欄位值。（5 分）



四、考慮以下兩個互相呼叫（mutually recursive）的 C 語言函式：

```
int foo(int n) {
    if (n <= 1) return 1;
    return foo(n - 1) + bar(n - 1) + 2;
}
int bar(int n) {
    if (n <= 1) return 1;
    return foo(n - 1) + bar(n - 1);
}
```

請回答下列問題：（每一小題請寫出推導過程，無推導過程不予計分。）

㈠當執行 foo(10)時，一共會呼叫 foo()函式幾次（包含最外層 foo(10)的這一次呼叫）？（10 分）

㈡執行 foo(10)的最終回傳結果為何？（10 分）

㈢以 Big-O 表示 foo(*n*)的時間複雜度（time complexity）。本題若同一個「函式與參數」組合被呼叫多次，每次都重新計算，不會儲存先前的計算結果供之後使用。（5 分）