110 年公務人員特種考試身心障礙人員考試試題

考試別：身心障礙人員考試
等　別：三等考試
類　科：資訊處理
科　目：資料結構

一、在大數據分析的應用中，許多資料集是以稀疏矩陣（Sparse matrix）的方式呈現，如：客戶的購物行為等。這些資料集共同的特性是資料維度大，但其中的資料量相對稀少， 若直接以傳統陣列結構來儲存資料， 將會造成大量的空間浪費。 請以 C 語言完成下列問題要求：

㈠設計一有效的二維稀疏矩陣資料結構， 避免儲存不存在（或其值為 0）的資料， 有效利用空間。（5 分）

㈡使用所設計的資料結構，完成矩陣的轉置（Transpose）運算函式。（10 分）

㈢使用所設計的資料結構，完成維度分別為 m×n 的 A 矩陣與 n×l 的 B 矩陣之矩陣相乘（Multiply）運算函式。（10 分）

【擬答】

㈠

```
class sparse_matrix{
    int **data;
    // dimensions of matrix
    int row, col;
    // total number of elements in matrix
    int len;
public:
    sparse_matrix(int r, int c)       {
        // initialize row
        row = r;
        // initialize col
        col = c;
        // initialize length to 0
        len = 0;
        //Array of Pointer to make a matrix
        data = new int *[MAX];

        // Array representation
        // of sparse matrix
        //[,0] represents row
```

```
            //[,1] represents col
            //[,2] represents value
            for (int i = 0; i < MAX; i++)
                data[i] = new int[3];
        }
    }
```

㈡

```
    sparse_matrix transpose()       {
        // new matrix with inversed row X col
        sparse_matrix result(col, row);
        // same number of elements
        result.len = len;
        // to count number of elements in each column
        int *count = new int[col + 1];

        // initialize all to 0
        for (int i = 1; i <= col; i++)
            count[i] = 0;
        for (int i = 0; i < len; i++)
            count[data[i][1]]++;
         int *index = new int[col + 1];
         // to count number of elements having
         // col smaller than particular i
         // as there is no col with value < 0
         index[0] = 0;
         // initialize rest of the indices
         for (int i = 1; i <= col; i++)
             index[i] = index[i - 1] + count[i - 1];
         for (int i = 0; i < len; i++)              {
             // insert a data at rpos and
             // increment its value
             int rpos = index[data[i][1]]++;
             // transpose row=col
             result.data[rpos][0] = data[i][1];

             // transpose col=row
```

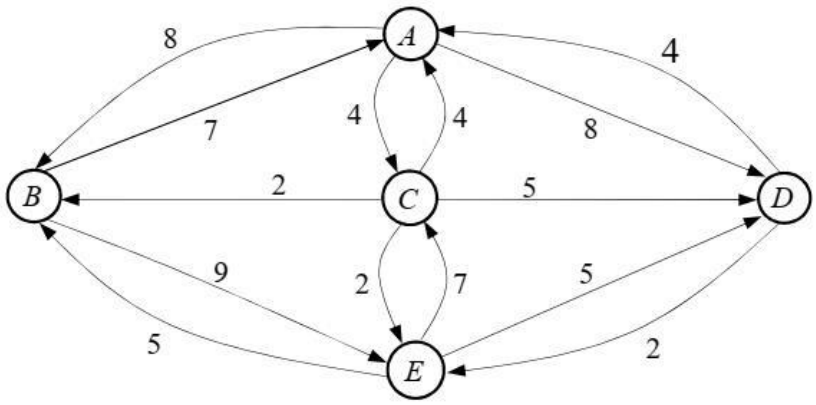
```
                result.data[rpos][1] = data[i][0];
                // same value
                result.data[rpos][2] = data[i][2];
            }
            return result;
        }
```

㈢

```
    void multiply(sparse_matrix b)      {
        if (col != b.row)             {
            // Invalid multiplication
            cout << "Can't multiply, Invalid dimensions";
            return;
        }

        // transpose b to compare row
        // and col values and to add them at the end
        b = b.transpose();
        int apos, bpos;
        // result matrix of dimension row X b.col
        // however b has been transposed,
        // hence row X b.row
        sparse_matrix result(row, b.row);
        // iterate over all elements of A
        for (apos = 0; apos < len;)           {
            // current row of result matrix
            int r = data[apos][0];

            // iterate over all elements of B
            for (bpos = 0; bpos < b.len;)                 {
                // current column of result matrix
                // data[,0] used as b is transposed
                int c = b.data[bpos][0];
                // temporary pointers created to add all
                // multiplied values to obtain current
                // element of result matrix
```

```
                int tempa = apos;
                int tempb = bpos;
                int sum = 0;
                // iterate over all elements with
                // same row and col value
                // to calculate result[r]
                while (tempa < len && data[tempa][0] == r &&
                        tempb < b.len && b.data[tempb][0] == c)      {
                    if (data[tempa][1] < b.data[tempb][1])
                        // skip a
                        tempa++;
                    else if (data[tempa][1] > b.data[tempb][1])
                        // skip b
                        tempb++;
                    else
                        // same col, so multiply and increment
                        sum += data[tempa++][2] *
                                b.data[tempb++][2];
                }
                // insert sum obtained in result[r]
                // if its not equal to 0
                if (sum != 0)
                    result.insert(r, c, sum);
                while (bpos < b.len &&
                        b.data[bpos][0] == c)
                    // jump to next column
                    bpos++;
            }
            while (apos < len && data[apos][0] == r)

                // jump to next row
                apos++;
        }
        result.print();
```

二、一有向圖形（ directed graph） G = (V, E)如下：



㈠請以相鄰矩陣（ adjacency matrix） 表達有向圖形 G。（5 分）

㈡設計一演算法找尋圖形中所有端點（ node） 對端點的最短路徑（ all-pairs shortest path），並以有向圖形 G 的相鄰矩陣為例說明所使用演算法的計算過程。（15 分）

㈢請說明在上述㈡中所使用演算法的時間複雜度（ time complexity）為何？（5 分）

【擬答】

㈠

	A	B	C	D	E
A	0	8	4	8	∞
B	7	0	∞	∞	9
C	4	2	0	5	2
D	4	∞	∞	0	2
E	∞	5	7	5	0

㈡演算法如下

```
void floydWarshall(int graph[][V]){
    int dist[V][V], i, j, k;
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];

    for (k = 0; k < V; k++) {
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++) {
            // Pick all vertices as destination for the
            // above picked source
            for (j = 0; j < V; j++) {
                if (dist[j][k] > (dist[j][i] + dist[i][k])
                   && (dist[j][i] != INF
                          && dist[i][k] != INF))
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
```

```
            }
        }
    }
```

計算過程：

1. 透過節點 A

	A	B	C	D	E
A	0	8	4	8	∞
B	7	0	11	15	9
C	4	2	0	5	2
D	4	12	8	0	2
E	∞	5	7	5	0

2. 透過節點 B

	A	B	C	D	E
A	0	8	4	8	17
B	7	0	11	15	9
C	4	2	0	5	2
D	4	12	8	0	2
E	12	5	7	5	0

3. 透過節點 C

	A	B	C	D	E
A	0	6	4	8	6
B	7	0	11	15	9
C	4	2	0	5	2
D	4	10	8	0	2
E	11	5	7	5	0

4. 透過節點 D

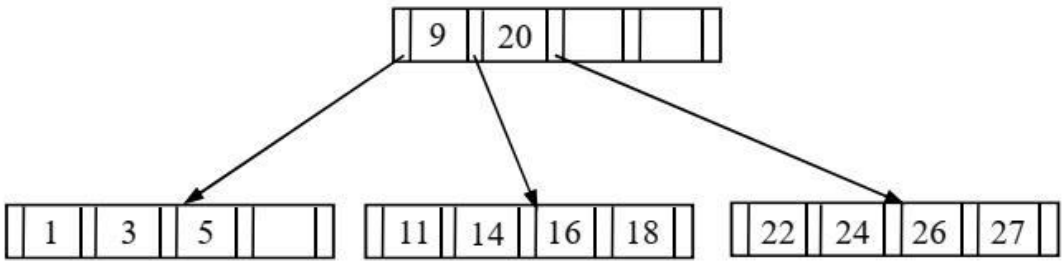
	A	B	C	D	E
A	0	6	4	8	6
B	7	0	11	15	9
C	4	2	0	5	2
D	4	10	8	0	2
E	9	5	7	5	0

5. 透過節點 E，完成如下

	A	B	C	D	E
A	0	6	4	8	6
B	7	0	11	14	9
C	4	2	0	5	2
D	4	7	8	0	2
E	9	5	7	5	0

㈢演算法中 for 迴圈有三層，因此時間複雜度為: O(V^3)

三、對資料庫系統的檔案儲存結構而言，必須能夠隨著檔案資料的增多，動態的新增儲存區塊（ block），例如： B-tree 樹狀檔案資料結構， 即可隨著資料量變大而增加葉節點區塊(leaf node block）或增加樹的高度來因應。下列為 m = 5 (5way) B-tree 的現況， 目前已存有 13 筆資料：

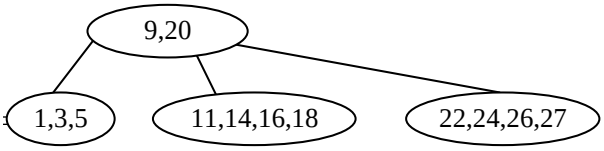


㈠請問具有 K 層以上 m = 5 結構的 B-tree 至少可以存放多少筆資料？（5分）

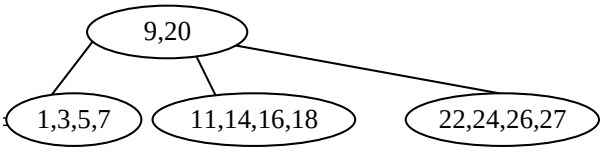
㈡請畫出完成運算 insert(7)與 insert(28)後的 B-tree 結構。（10分）

㈢完成上述㈡之後接續畫出先後完成運算 insert(15)與 insert(6)的 B-tree結構。（10分）

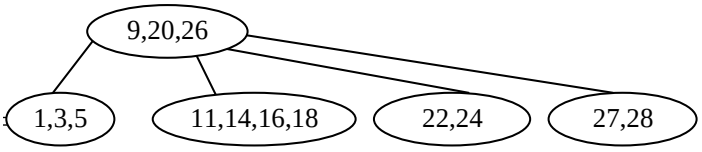
【擬答】



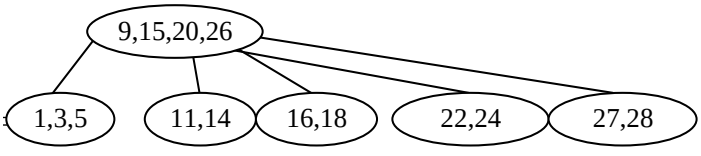
㈠高度為k且m=5 的B-樹最少擁有$1+\frac{2\lceil 5/2\rceil^{k-1}-2}{\lceil 5/2\rceil-1}$個節點與$2\lceil 5/2\rceil^{k-1}-1$個鍵值(存放資料數)。



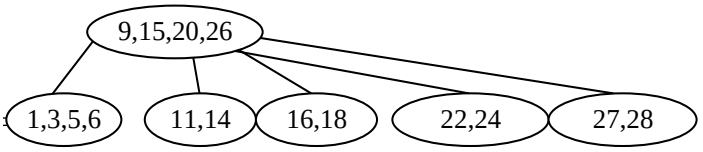
㈡ insert(7)後成為



insert(28) 後成為



㈢ insert(15) 後成為



insert(6) 後成為

四、霍夫曼碼（ Huffman code） 是具有最佳編碼的資料壓縮方法之一，今有下列的訊息欲以霍夫曼碼編碼傳遞以節省訊息量“PAPAYA_AND_BANANA_ARE_YUMMY”其中空格 ‘_’ 亦

需計算在訊息量內。

㈠請以該訊息詳述構建霍夫曼碼演算法的過程與結果。（20 分）

㈡依步驟說明所使用演算法的時間複雜度（time complexity）。

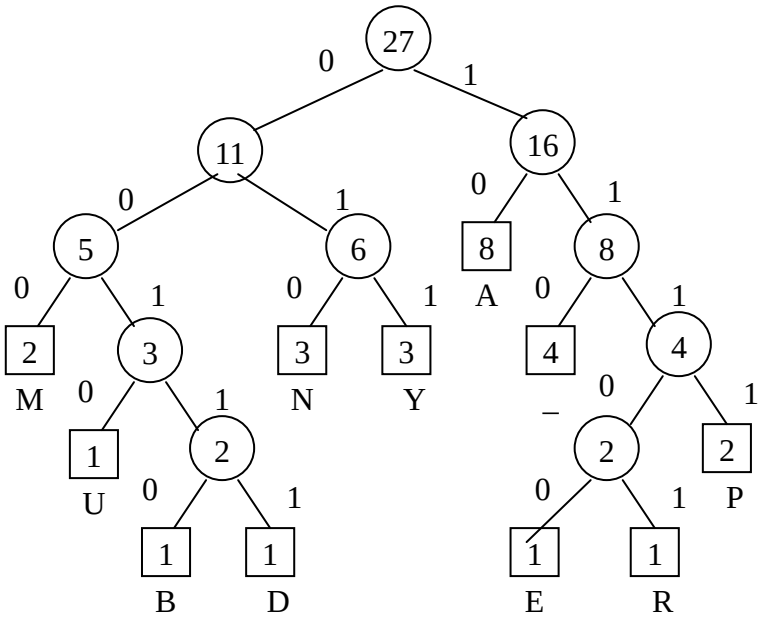
【擬答】

㈠

1. 先找出各字元出現頻率如下：

_:4，A:8，B:1，D:1，E:1，M:2，N:3，P:2，R:1， U:1，Y:3

2. 重複選取頻率最少的兩個組成樹，最後可得下列 Huffman 解碼樹



3. 因此各字元編碼如下：

_:110，A:10，B:00110，D:00111，E:11100，M:000，N:010，P:1111，R:11101， U:0010，Y:011

㈡此演算法的時間複雜度（ Time Complexity）為 O（n log n）；因為有 n 個終端節點，所以樹總共有 2n-1 個節點，使用優先佇列每個迴圈須 O（log n）。