一、 程式摘要

1. 邏輯/原理

「給定一字串，找出其最長迴文長度及迴文子序列」，我們分別以下列方式實作。

✓ 最長迴文長度

利用 Dynamic Programming 的概念，我們先建立表格（建表的過程稍後將以圖解表示），並逐一找出各個字元長度下是否存在迴文，若存在，則將目前迴文長度儲存於表格中。當發現更長的字元長度下存在迴文時，則可利用過去表格所涵蓋（overlap）到的結果，加上新對稱的 2 個字元，即為當前字元長度下的迴文長度，如此反覆地計算，最後即能得出最長的迴文長度。而我們可用下列關係式表示之：

$$LPS[i,j] = \begin{cases} 1 & if\ i = j \\ LPS[i+1, j-1] + 2 & if\ S[i] = S[j] \\ \max(LPS[i, j-1], LPS[i+1, j]) & if\ S[i] \neq S[j] \end{cases}$$

（最長迴文長度定義為LPS[i,j]；S 為給定的字元陣列（字串）；i 為該字元陣列的起始索引；j 為結尾索引）

✓ 印出最長迴文子序列

前述邏輯的反向操作，利用 Iteration 循序判斷字元陣列左右兩端字元是否對稱，若對稱，則儲存該字元，並分別將字元陣列的起始索引及結尾索引往中間逼近，如此漸漸縮小範圍後，最後即可蒐集完整的子序列。

2. 語言

以 C 語言實作。

二、 程式內容說明

1. 程式註解

```c
#include <stdio.h>
#include <string.h>

// 字串長度最大值常數化
#define MAX_SIZE 1000

// 具有長度、字串屬性的結構
struct lps_data {
    int size;
```

```
        char data[MAX_SIZE];
};

// 比較 int 大小的 funciton
int find_max(int a, int b) {
        return (a > b) ? a : b;
}

// 計算 longest palindrome subsequence 的 function
struct lps_data find_lps(char data[], int data_size){
    // 宣告二維陣列，用來儲存計算過程 (DP)
        int lps[data_size][data_size];
    // 初始化陣列
        memset (lps, 0, sizeof lps);
    // 計算最長迴文長度
        for(int l = 1; l < data_size + 1; l++) {
                for (int i = 0; i < data_size - l + 1; i++) {
                        int j = i + l - 1;
                        if(data[i] == data[j]) {
                                lps[i][j] = (l == 1) ? 1 : lps[i + 1][j - 1] + 2;
                        } else {
                                lps[i][j] = find_max(lps[i][j - 1], lps[i + 1][j]);
                        }
                }
        }

        // 取得最長迴文長度
        int subseq_size = lps[0][data_size - 1];

        // 宣告最長迴文子序列
        char subseq[MAX_SIZE] = "";
        // 初始化起始索引
        int i = 0;
        // 初始化結尾索引
        int j = data_size - 1;
        int index = 0;

        // 印出最長迴文子序列
        while(i < data_size && j >= 0) {
                if (data[i] == data[j]) {
                        if (lps[i][j] == 1) {
                                subseq[index] = data[i];
```

```
                    break;
                }
                subseq[index] = data[i];
                subseq[subseq_size - index - 1] = data[i];
                i++;
                j--;
                index++;
            } else if (lps[i][j - 1] > lps[i + 1][j]) {
                j--;
            } else {
                i++;
            }
        }

        // 裝填結果
        struct lps_data result;
        result.size = subseq_size;
        strcpy(result.data, subseq);
        // 回傳結果
        return result;
}

// main funciton
int main() {
        int input_num;
        // 輸入測資個數
        scanf("%i", &input_num);
        struct lps_data input_list[input_num];

        // 裝填每一筆測資
        for (int i = -1; i < input_num; i++) {
                char input[MAX_SIZE];
                fgets(input, sizeof(input), stdin);

                if (i >= 0) {
                        // 計算長度
                        input_list[i].size = -1;
                        for(int j = 0; input[j] != '\0'; j++) {
                                input_list[i].size++;
                        }
                        strcpy(input_list[i].data, input);
                }
```

```
    }

    // 將裝填的測資拿出來一筆一筆執行
    for (int i = 0; i < input_num; i++) {
        // 呼叫運算 funciton，將測資及長度作為參數傳入
        struct lps_data result = find_lps(input_list[i].data, input_list[i].size);
        // 印出結果
        printf("%d\n%s\n", result.size, result.data);
    }
}
```

2. 圖解

✓ 最長迴文長度

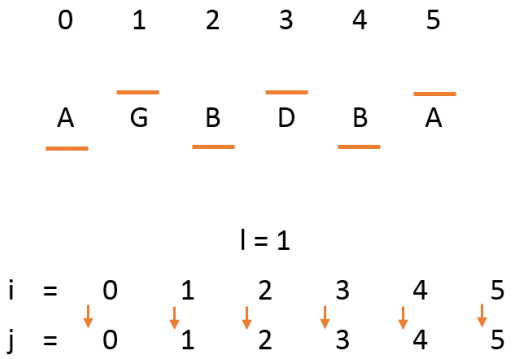
我們以字串 S = [AGBDBA]為例，依照前述的關係式，以下圖說明計算最長迴文長度的過程。

$$LPS[i,j] = \begin{cases} 1 & if\ i = j \\ LPS[i+1, j-1] + 2 & if\ S[i] = S[j] \\ \max(LPS[i, j-1], LPS[i+1, j]) & if\ S[i] \neq S[j] \end{cases}$$

（最長迴文長度定義為LPS[i,j]；S 為給定的字元陣列（字串）；i 為該字元陣列的起始索引；j 為結尾索引；l 為欲檢查的字元長度）

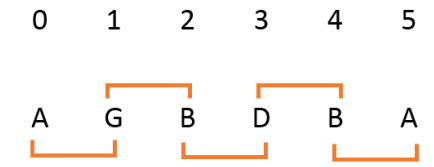
(a)

i \ j	0	1	2	3	4	5
0	1					
1		1				
2			1			
3				1		
4					1	
5						1



(b)

i \ j	0	1	2	3	4	5
0	1	1				
1		1	1			
2			1	1		
3				1	1	
4					1	1
5						1



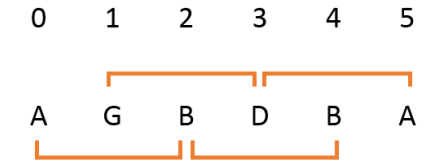
l = 2

i = 0 1 2 3 4

j = 1 2 3 4 5

(c)

i \ j	0	1	2	3	4	5
0	1	1	1			
1		1	1	1		
2			1	1	3	
3				1	1	1
4					1	1
5						1



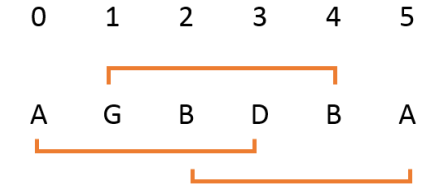
l = 3

i = 0 1 2 3

j = 2 3 4 5

(d)

i \ j	0	1	2	3	4	5
0	1	1	1	1		
1		1	1	1	3	
2			1	1	3	3
3				1	1	1
4					1	1
5						1



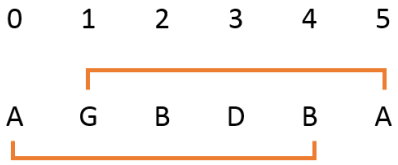
l = 4

i = 0 1 2

j = 3 4 5

(e)

i \ j	0	1	2	3	4	5
0	1	1	1	1	3	
1		1	1	1	3	3
2			1	1	3	3
3				1	1	1
4					1	1
5						1



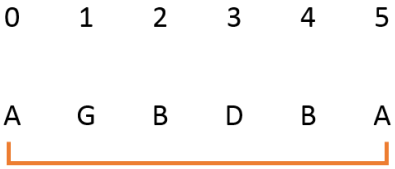
l = 5

i = 0 1

j = 4 5

(f)

i \ j	0	1	2	3	4	5
0	1	1	1	1	3	5
1		1	1	1	3	3
2			1	1	3	3
3				1	1	1
4					1	1
5						1



l = 6

i = 0

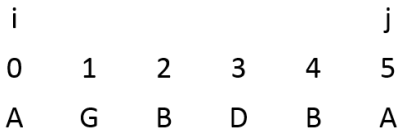
j = 5

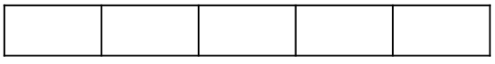
✓ **印出最長迴文子序列**

我們依然以字串 S = [AGBDBA]為例，以下圖說明找出最長迴文子序列的過程。

(a)

i \ j	0	1	2	3	4	5
0	1	1	1	1	3	5
1		1	1	1	3	3
2			1	1	3	3
3				1	1	1
4					1	1
5						1





(b)

i \ j	0	1	2	3	4	5
0	1	1	1	1	3	5
1		1	1	1	3	3
2			1	1	3	3
3				1	1	1
4					1	1
5						1

		i			j
0	1	2	3	4	5
A	G	B	D	B	A

i = 0 → 1
j = 5 → 4

A				A

(c)

i \ j	0	1	2	3	4	5
0	1	1	1	1	3	5
1		1	1	1	3	3
2			1	1	3	3
3				1	1	1
4					1	1
5						1

		i		j	
0	1	2	3	4	5
A	G	B	D	B	A

i = 1 → 2
j = 4

A				A

(d)

i \ j	0	1	2	3	4	5
0	1	1	1	1	3	5
1		1	1	1	3	3
2			1	1	3	3
3				1	1	1
4					1	1
5						1

			i j		
0	1	2	3	4	5
A	G	B	D	B	A

i = 2 → 3
j = 4 → 3

A	B		B	A

(e)

i \ j	0	1	2	3	4	5
0	1	1	1	1	3	5
1		1	1	1	3	3
2			1	1	3	3
3				1	1	1
4					1	1
5						1

		i j			
0	1	2	3	4	5
A	G	B	D	B	A

A	B	D	B	A

3. 虛擬碼

```
FIND_LPS(data, size)
    // 以二維陣列 lps 儲存各個字元長度下的最長迴文長度
    // 初始化 lps
    for m = 0 to size
        lps[m,0] = 0
    for n = 0 to size
        lps[0,n] = 0;

    // 計算最長迴文長度
    // l 為欲檢查的字元長度
    for l = 1 to size + 1
        // i 為第一個字元
        for i = 0 to size - l + 1
            // j 為最後一個字元
            j = i + l - 1
            // 如果發現對稱字元
            if data[i] == data[j]
                // 長度為 1 的情況，代表自己跟自己比
                if l = 1
                    // 最長迴文長度為 1
                    lps[i,j] = 1
                else
                    // 取出重疊的結果，並加上當前的 2 個字元，即為當前的最長迴文長度
                    lps[i,j] = lps[i+1,j-1] + 2
            // 如果沒有發現對稱字元
            else
                // 則最長迴文長度仍為過去重疊部分的結果，找出長度較長的
```

```
                    lps[i,j] = FIND_MAX(lps[i,j-1], lps[i+1,j])

// 取得最長迴文長度
subseq_size = lps[0,size - 1]

// 印出最長迴文子序列
// 初始化起始索引
i = 0
// 初始化起始索引
j = size - 1
// 初始化最長迴文子序列的索引
index = 0

// 從字串的頭跟尾開始逼近
// 判斷邏輯為裝填 lps 時的反向路徑
while i < size and j >= 0
    // 如果發現對稱字元
    if data[i] = data[j]
        // 判斷是否已找到中間值
        if lps[i,j] = 1
            // 賦予中間值 (子序列都裝填完成)
            subseq[index] = data[i]
            // 跳出迴圈
            break;
        // 因為是迴文，頭跟尾都賦予值
        subseq[index] = data[i]
        subseq[subseq_size - index - 1] = data[i]
        // 頭的索引往右逼近
        i++;
        // 尾的索引往左逼近
        j--;
        // 子序列索引累加
        index++;
    // 如果沒發現對稱字元，則判斷過去最長的迴文長度為何
    // 往長度較長的走
    else if lps[i,j - 1] > lps[i + 1,j]
        // 尾的索引往左逼近
        j--;
    // 反之，往另一端走
    else
        // 頭的索引往右逼近
        i++;
```

```
// 回傳最長迴文子序列及最長迴文長度
return subseq, subseq_size
```