PA2 Report

B06502164 機械三 楊淳富

在本次的作業中，我寫了兩個 class，分別是 Data 與 Solution。Data 是用來記錄 input 的資訊，內容是一個 int 跟一個 int array。Int N 記錄給定資料的弦數，array 記錄了各點上弦的另一個端點代號，以便在之後的 Solution 中記錄端點。

```
class Data {
    public:
        int N;                  //number of chords
        int* chords;            //record the points that are bonded together
        Data();                 //constructor
        Data(int num);          //constructor with number of chords N
};
```

另一個 class Solution 是這次作業的解，private 中記錄的是 DP 用的表格。m 對應的是上課講的 m[i][j]；find_case 記錄各種 m[i][j]是怎麼被得到的，有三種 case；passed 是由於演算法是使用 top-down 的 DP，用一個陣列記錄 m[i][j]是否曾經被填過值。Member function MPSC 是一個 top-down 的 DP 演算法，跑完 MPSC，Solution 中的 m、find_case、passed 會被用到的格子會被填完。在之後的 find_chords 找弦的時候會被用到。傳入 Data 作為 attribute 的 constructor 是方便在傳入 data 時直接把我所需的動態陣列直接開好。

```
class Solution {
    private:
        int** m;
        int** find_case;
        int** passed;
        int arraysize;
    public:
        Solution();
        Solution(Data& data);   //constructor
        int MPSC(Data& data, int start, int end);
                                //the number of MPSC
        void find_chords(Data& data, int start, int end, char* filename);
};
```

找弦是讓我決定寫成 class 最重要的原因，在一開始我原想單純用寫 function 的方式來完成這次的作業，但如果不用 class，在每次 function 間資料傳輸我必須一次傳入三個 int**的 attribute。在我的習慣中，我不太喜歡讓代號太過簡短，所以會讓代號名稱設成如 find_case、passed，這樣很直觀的代號，否則對我而言代號容易搞混。因此如果每個 function 都要傳入這樣的 attribute，會讓下方的 recursive call 變得非常龐雜，因此我寫成一個 class 之後，我需要使用的表格可以直接在 member function 呼叫，讓程式的撰寫變得比較不冗長。

```cpp
void Solution::find_chords(Data& data, int start, int end, char* filename) {
    //cout << start << " " << end << endl;
    fstream fout;
    if (find_case[start][end] == 1) {
        fout.open(filename, ios::app);
        fout << start << " " << end << endl;
        find_chords(data, start + 1, end - 1, filename);
    }
    else if (find_case[start][end] == 2) {
        int k = data.chords[end];
        find_chords(data, start, k - 1, filename);
        fout.open(filename, ios::app);
        fout << k << " " << end << endl;
        find_chords(data, k + 1, end - 1, filename);
    }
    else if (find_case[start][end] == 3) {
        find_chords(data, start, end - 1, filename);
    }
    else {
        return;
    }
}
```

找弦的過程中，根據不同的 case 用不同的方式找弦。在本次作業中，由於輸出的弦需要由小到大排好，因此在找弦的方式，我使用了類似 print binary search tree 的方式。以上圖的 case 2 為例，case 2 為 k ∈ (i, j)，弦 kj 要取的狀況，因為 i ~ k-1 找出來的弦，i(也就是圖中的 start)必定小於 k，所以使用 in-order 的方式去寫入 kj 這條弦，可以讓 print 出來的結果會自動被由小到大排列，依據不同的 case 也可能用 pre-order 的方式(如 case 1)。