Algorithm Programming Assignment #2 **Report**

B07901021 潘世軒

(1) Data structure

```
class ChordSet{
    friend class Node;
    public:
        ChordSet();
        ChordSet(int);
        bool connection_assign(int, int);
        int find_connection(int);
        bool chord_existence(int, int);
        void test();
    private:
        vector<int> data;

};
```

使用一個叫做 ChordSet 的資料，將讀入的.in 檔轉成可使用的集合，private member 中 data[i]存放和 i 爲連線的點，舉例來說，將作業提供的 12.in 讀入之後，data 會變成[4, 9, 6, 10, 0, 7, 2, 5, 11, 1, 3, 8]，將資料結構這樣設計的原因是，我們可以透過端點 i 直接去找到他連線的另外一端點(函式 find_connection)，不需要檢查全部的點。另外，也支援判斷某兩點是否存在連線(函式 chord_existance)。

(2) Algorithm

根據輸入建立好 ChordSet 後，先建立二維陣列 M，其中 M[i][j]用來表示 M(i, j)裡的最大 chord 數，先將每一個數字設爲-1，代表未尋訪過，然後用 top-down 的方式來把需要的點填滿。

接下來，由 M[0][N-1]開始跑遞迴，有點像上方的走法，但是這次是用來輸出範圍內的弦，將範圍漸漸切小之後把範圍內的 chord 的起點慢慢寫進 output 檔案中。

```
    //表格的初始化和建立
    int **M = new int* [N];
    for(int i = 0; i < N; i++){
        M[i] = new int[N];
    }
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            M[i][j] = -1;
        }
    }

    M[0][N-1] = MPS(0, N-1, &C, M);
```

```
// Topdown dynamic programming
int MPS(int i, int j, ChordSet* C, int** M){
    // cout << "fill in " << "M[" << i << "][" << j << "]" << endl;
    if(M[i][j] != -1) return M[i][j];
    if(i >= j){
        M[i][j] = 0;
        return 0;
    }
```

```
    int k = C->find_connection(j);

    if((k > j) || (k < i)){
        M[i][j] = MPS(i, j-1, C, M);
    }

    else if(k == i){
        M[i][j] = MPS(i+1, j-1, C, M) + 1;
    }

    else if(k < j && k > i){
        int A = MPS(i, j-1, C, M);
        int B = MPS(i, k-1, C, M) + 1 + MPS(k+1, j-1, C, M);
        M[i][j] = (A >= B) ? A : B;
    }
    return M[i][j];
}
```

```
// 跑遞迴，把在範圍內的點輸出
void traverse(int i, int j, ChordSet* C, fstream& fout, int** M){
    if(!M[i][j]) return;
    int k = C->find_connection(j);

    if((k > j) || (k < i)){
        traverse(i, j-1, C, fout, M);
    }

    else if(k == i){
        fout << i << " " << j << endl;
        traverse(i+1, j-1, C, fout, M);
    }

    else if(k < j && k > i){
        int A = M[i][j-1];
        int B = M[i][k-1] + 1 + M[k+1][j-1];
        if(A >= B){
            traverse(i, j-1, C, fout, M);
        }
        else{
            traverse(i, k-1, C, fout, M);
            fout << k << " " << j << endl;
            traverse(k+1, j-1, C, fout, M);
        }
    }
}
```

(3) Findings

這次作業比較困難的部分是控制記憶體用量的部分，需要填完 $O(n^2)$的表格，因此如果不好好控管記憶體的話很容易就會炸開，再來在填表格的時候後來選用 Top down 的方法，因爲這次作業要求的只有找出 0~N-1 範圍內的點，不需要多次計算，因此選用 top down 的方法可以省時間，最後 100000 筆的測資可以壓在大約兩分鐘之內跑完，算是還蠻 OK 的。

另外 vector 雖然很好用，但是他在自己分配空間(一個一個 pushback)的時候是直接無腦兩倍的樣子，因此用比較古老的方法(用指標陣列，自己 new 空間出來)可能會對記憶體用量有比較多的掌控度，另外根據有修計算機結構的同學說法，這樣似乎也會比較快