

# Algorithm Programming Assignment #2 Report

B07901021 潘世軒

## (1) Data structure

```
1. class ChordSet{
2.     friend class Node;
3.     public:
4.         ChordSet();
5.         ChordSet(int);
6.         bool connection_assign(int, int);
7.         int find_connection(int);
8.         bool chord_existence(int, int);
9.         void test();
10.    private:
11.        vector<int> data;
12.
13. };
```

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 為連線的點，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\_existence)。

## (2) Algorithm

根據輸入建立好 ChordSet 後，先建立二維陣列 M 和 M\_aux，其中 M[i][j]用來表示 M(i, j)是藉由 case 幾來完成的，M\_aux 則是用來表示 M(i, j)裡的最大 chord 數，在填完這兩個表格後，由 M[0][N-1]開始跑遞迴，將範圍漸漸切小之後把範圍內的 chord 的起點(擇一)慢慢加入 vector Chordlist 中，最後將 Chordlist 依照輸出格式寫進.out 檔裡面，就完成本次計算了

```
1. for(int i = 0; i < M.size(); i++){
2.     vector<short> temp1(N);
3.     vector<int> temp2(N);
4.     M[i] = temp1;
5.     M_aux[i] = temp2;
6. }
7.
8. for(int l = 1; l < N; l++){
9.     for(int i = 0; i < N - l; i++){
10.         int j = i + l;
11.         int k = C.find_connection(j);
12.
13.         if((k > j) || (k < i)){
14.             M[i][j] = 1;
15.             M_aux[i][j] = M_aux[i][j-1];
16.         }
17.
18.         else if(k == i){
19.             M[i][j] = 2;
20.             M_aux[i][j] = M_aux[i+1][j-1] + 1;
21.         }
22.
23.         else if(k < j && k > i){
```

```

24.         int A = M_aux[i][j-1];
25.         int B = M_aux[i][k-1] + 1 + M_aux[k+1][j-1];
26.         if(A >= B){
27.             M[i][j] = 3;
28.             M_aux[i][j] = A;
29.         }
30.         else{
31.             M[i][j] = 4;
32.             M_aux[i][j] = B;
33.         }
34.     }
35. }
36. }

```

(Main.cpp 片段->填表格)

```

1. void traverse(int i, int j, ChordSet* C, vector<int>* list, vector<vector<short> >* M){
2.     switch((*M)[i][j]){
3.         case 1:
4.             traverse(i, j-1, C, list, M);
5.             break;
6.
7.         case 2:
8.             list->push_back(i);
9.             traverse(i+1, j-1, C, list, M);
10.            break;
11.
12.        case 3:
13.            traverse(i, j-1, C, list, M);
14.            break;
15.
16.        case 4:
17.            traverse(i, C->find_connection(j)-1, C, list, M);
18.            list->push_back(C->find_connection(j));
19.            traverse(C->find_connection(j)+1, j-1, C, list, M);
20.            break;
21.
22.        default:
23.            return;
24.    }
25.
26. }

```

(函式 traverse)

### (3) Findings

這次作業比較困難的部分是控制記憶體用量的部分，因為 bottom up 的 dynamic programming 很多都需要填完  $O(n^2)$  的表格，因此如果不好好控管記憶體的話很容易就會炸開，有些能夠使用比較小的整數型態(如 short)的 vector，我就會盡量使用之，但是最後在做魔王測資 100000.in 的時候我在工作站還是要跑一陣子，甚至有時候還會被 killed 掉，只能祈禱沒有太多人同時在用同一台主機....。