Algorithm Programming Assignment #2 **Report**

B07901021 潘世軒

1. **Data structure**
2. **class** ChordSet{
3. **friend** **class** Node;
4. **public**:
5. ChordSet();
6. ChordSet(**int**);
7. **bool** connection\_assign(**int**, **int**);
8. **int** find\_connection(**int**);
9. **bool** chord\_existence(**int**, **int**);
10. **void** test();
11. **private**:
12. vector<**int**> data;
14. };

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\_existance)。

1. **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，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. **int** \*\*M;
2. M = (**int** \*\*)malloc(**sizeof**(**int**\*) \* N);
3. **for**(**int** i = 0; i < N; i++){
4. M[i] = (**int** \*)malloc(**sizeof**(**int**) \* N);
5. }
7. **for**(**int** l = 0; l < N; l++){
8. **for**(**int** i = 0; i < N - l; i++){
9. **int** j = i + l;
10. **if**(i == j){
11. M[i][j] = 0;
12. **continue**;
13. }
14. **int** k = C.find\_connection(j);
16. **if**((k > j) || (k < i)){
17. M[i][j] = M[i][j-1];
18. }
20. **else** **if**(k == i){
21. M[i][j] = M[i+1][j-1] + 1;
22. }
24. **else** **if**(k < j && k > i){
25. **int** A = M[i][j-1];
26. **int** B = M[i][k-1] + 1 + M[k+1][j-1];
27. **if**(A >= B){
28. M[i][j] = A;
29. }
30. **else**{
31. M[i][j] = B;
32. }
33. }
34. }
35. }

(Main.cpp片段->填表格)

1. **void** traverse(**int** i, **int** j, ChordSet\* C, vector<**int**>\* list, **int**\*\* M){
2. **int** k = C->find\_connection(j);
3. **if**(!M[i][j]) **return**;
4. **else** **if**((k > j) || (k < i)){
5. traverse(i, j-1, C, list, M);
6. }
8. **else** **if**(k == i){
9. list->push\_back(i);
10. traverse(i+1, j-1, C, list, M);
11. }
13. **else** **if**(k < j && k > i){
14. **int** A = M[i][j-1];
15. **int** B = M[i][k-1] + 1 + M[k+1][j-1];
16. **if**(A >= B){
17. traverse(i, j-1, C, list, M);
18. }
19. **else**{
20. traverse(i, k-1, C, list, M);
21. list->push\_back(k);
22. traverse(k+1, j-1, C, list, M);
23. }
24. }
25. }

(函式traverse)

1. **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(n2)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….。  
另外vector雖然很好用，但是他在自己分配空間的時候是直接無腦兩倍的樣子，因此用比較古老的方法(如自己malloc)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，自己malloc可能比較快(？