Algorithm PA2

B08901041 電機三 王昱翔

Data Strucure

1. 一開始先創一個 int 變數存取 # of node，再把他除以2變成 `length_num`，並創一個 [`length_num`] [2] 的 array `data` 來存取input data的chords，並把每一個chord中index比較小的放在前面
2. 將 `data` 的內容建成一個新的一維 `2*length_num` 大小的array `chord` 來存放chord另一端的node的index（i.e.假如 (1,4) 是chord，`chord[1]` = 4、`chord[4]` = 1），接著把 `data` array給delete
3. 創造兩個新的int變數 `start` 跟 `end` ，裡面存 `0` 跟 `2*length_num-1` (不必要、只不過為了程式碼的易讀性)
4. 創造一個二維的array `Max` ，藉由動態宣告的方式，創造一個長得像三角形的矩陣，以此省下一半空間，並用其紀錄每一個算過的最大值
5. 創造一個vector `final_start` ，用以紀錄最後要輸出的chords的每一個start node

檔案架構

```
bin/
  - mps        //executable file
src/
  - map.cpp    //main function
  - maxLine.h  //function to find the max # of chord and the index of chords
doc/
  - report.pdf //the report of PA2
- makefile
- README.md
```

Algorithm

- 找最大的chord數：

以Top-down的架構，不斷recursive的call `Max_Planar_Subset` ，來填 `Max` array，也因為是Top-down的關係， `Max` array並不會被全部被填滿，只會算出並填上需要的值，當每個recursive最後都遇到base case後，最終答案就出來了

- 找出有哪些的chords：

一樣以Top-down的架構，並且判斷式都跟 `Max_Planar_Subset` 這個function的一樣，差別是中間的判斷可以用已經填好的 `Max` array，依序把有用到的chord的start node加進 `final_start` vector，全部都遇到base case後，最終答案就出來了

遇到的問題

1. 起初用buttom up的方式來寫，runtime會是 N^2 ，理論上已經是最好了，不過如果data其實很sparse的話，那用buttom up的方式會多花很多時間算不必要的值，因此後來改成top down寫法，除了速度更快之外，程式碼也更加簡潔
2. 在Space complexity的部分，一開始我使用的是6個N^2的矩陣，因為當初想說都是$O(N^2)$的space complexity，但其實前面的係數也差很多。因此後來又想辦法縮到變成一個三角形、$N^2/2$ 的大小，結果速度真的就改善了很多，而且也不會因為開了太多記憶體而出現segmentation fault或是被killed的情形
3. 因為是用Top down的方式，所以需要recursive的call function，但起初我在function中有宣告一個變數，當初想說只是一個 int 變數應該差別不大，但是假如我recursive call了function N次，那其實就是會開了N個變數，所以我後來也修改了一下變成不需要需告變數
4. 起初都是用靜態矩陣的方式，但這次作業很明顯的就是在考驗我們怎麼壓縮我們的記憶體空間，所以就把所有矩陣都換成靜態陣列了，並且一旦用完就馬上delete，不會佔用多餘記憶體空間
5. 在最後要輸出答案時，規定要依照starting node進行sort過，起初為了方便就自己刻了一個insertion sort，後來為了改善，就用了algorithm library裡面的sort()，除了從$O(N^2)$降為$O(NlgN)$ 之外，程式也簡潔很多