Homework2

109550116 楊傑宇

Part 1 Sorting

- **Introduction**:

  兩者都是 Divide-and-Conquer 的演算法，只是 quick sort 的 partition 比較麻煩而 merge 是遞迴後的 merge 比較麻煩。

- **Implement Details**:

  Quick Sort: 這個演算法是先找出一個 pivot（通常是最後一個），並從頭開始比較，若數字小於 pivot 則與第 i 位換位子（當有比 pivot 小的號碼 i 才會增加），若數字比 pivot 大則不動，最後 i+1 再與 pivot 換，最後便會形成左邊小於右邊大於的情況(程式碼第 15~30 行)，一路分下去便能做好排序。

  Merge Sort: 這個演算法是以遞迴方式先分成一半又一半，當都分成一個一個時，在進行比較來裝箱（從小快裝到大塊），有一個特色辨是，因為是前一個小塊已經分好大小了，所以只需要依序拿兩邊的數字來比較，一路下去便能做好排序。

- **Results:**

  Quick Sort: Average time: (NlogN), fastest time: (NlogN), slowest time: (N^2)

```
(base) jerrys_macbook_
(base) jerrys_macbook_
5
7 -1 -2 4 9
-2 -1 4 7 9
ç^C
(base) jerrys_macbook_
(base) jerrys_macbook_
5
7 -1 -2 4 9
0.00001ms-2 -1 4 7 9
```

  Merge Sort: Average time: (NlogN),fastest time: (NlogN), slowest time: (NlogN)

```
1 error generated.
(base) jerrys_macbook_
(base) jerrys_macbook_
5
7 -1 -2 4 9
0.00001ms-2 -1 4 7 9
7
8 4 5 -10 3 4 -6
0.00001ms-10 -6 3 4 4
```

- **Discussion:**

  經過這個功課，我發現原來簡單的從小排到大居然也有很多不同的方法，更有趣的是當我們把一個數列分成一塊一塊可以省下那麼多時間。在一般情況下 Quick Sort 比較好因為 Merge Sort 是用外部的記憶體，但由時間複雜度來看，在大數據是 Merge Sort 比較好。在這個功課裡沒遇到什麼太大的困難。

Part 2 Minimum Spanning Tree

- **Introduction:**

  Prim Algorithm 是找沒去過但最近的點，Kruskal Algorithm 是由最小的邊以不 loop 的方式往大的邊找。簡單來說 Prim Algorithm 比較注重點，Kruskal Algorithm 比較注重邊。

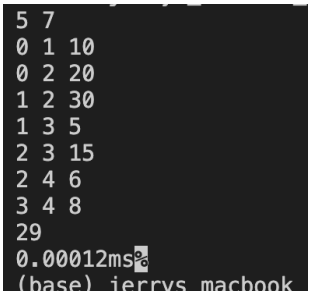
- **Implement Details**:

  Prim Algorithm:我們首先先把點分成“到過”與“沒到過”，第一步先把起始點設為到過，並且由所有“到過的點”去找剩下的點，若剩下的點符合沒到過且重量不等於 0（有連線）(程式碼第 47 行)，則將這個邊列入最小邊的候選人，當考慮完所有“到過的點”後，在候選人裡選出最小的並連接他們，連接的方式就是將“沒到過的點”設定成“有到過的點”(程式碼第 62 行)。這樣的步驟進行(所有點-1)次便可以找出最小生成樹。

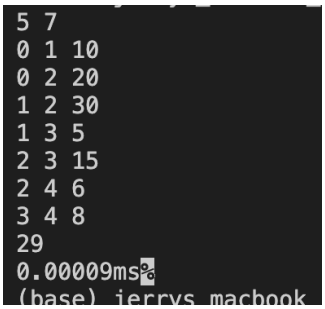
  Kruskal Algorithm:這個演算法是找所有最小的邊以不 loop 的形況連線。在分辯是否 loop 時我用“能走到最大點”來判斷，舉例：當連線(1,3)那 1 能走到的最大點便是 3，當我們把(3,4)、(2,4)連線，如果我們這時候再連接(1,2)，因為 1 跟 2 能走到的最大點都是 4 所以會形成 loop。因此我們只要以避免形成 loop(程式碼第 64 行)的方式不斷找最小的邊連接便可以找出最小生成樹。

- **Results:**

  Prim Algorithm: O(E logV)



Kruskal Algorithm: O(ElogE)



- **Discussion:**

  在了解這兩個演算法以前，如果要我想 MST 我可能會覺得只有 Kruskal Algorithm 是可行的，沒想到以點的角度去找也可以找到 MST。當然兩者著手的方面不同，對於不同圖形的時間複雜度也會有差，因此 Kruskal Algorithm 適合點比邊多的圖形，而 Prim Algorithm 適合邊比點多的圖形。在過程中有遇到一個困難，但是到現在都還沒解決，我在做 online judge 的時候有一個測資 timeout error 然而我用助教給的比較大的測資卻可以跑。

Part 3 Shortest Path

- **Introduction:**

  兩者都是由一個點來尋找最短路徑，然而 Bellman-Ford Algorithm 可以有負的邊也可以檢查 negative loop，Dijkstra's Algorithm 不行。

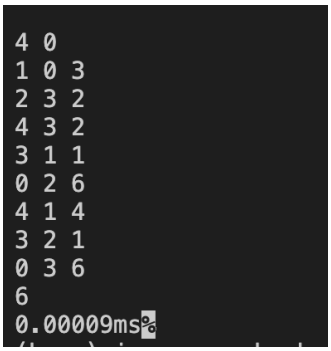
- **Implement Details**:

  Dijkstra's Algorithm:我們先給定起點，並設定其他的點距離起點無限遠，(程式碼第 80、81 行)，之後進行距離更新(程式碼第 94~97 行)，若距離小於原本的距離就更新沒有則不動，在距離更新後我們選擇距離最小的走，在進行(所有點-1)次後就可以找到最短路徑。

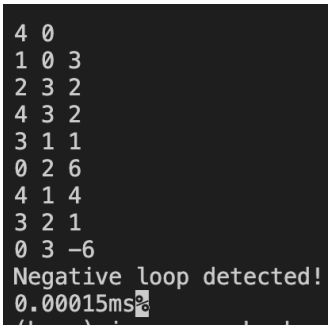
  Bellman-Ford Algorithm:我們一樣先給定一個起點，然而尋找最短路徑的方法與 Dijkstra's Algorithm 差不多，然而此演算法多了一個判斷 negative loop 的步驟，判斷 negative loop 的方法是檢查每對(u->v)，若起點到 u 再到 v(+weight)還比起點到 v 都還要近那就是有 negative loop。

- **Results:**

  Dijkstra's Algorithm: O(V^2):



  Bellman-Ford Algorithm: O(VE)



- **Discussion:**

  如果檢測 negative loop 我已經寫在上面了，至於如果我們要紀錄路徑過程的話我們在做距離更新時，同時也要做路徑的更新，要記錄上一個點是誰(parent)，由時間複雜度來看，Dijkstra's Algorithm 適合邊比較多的圖形，Bellman-Ford Algorithm 適合點比較多的圖形，然而比起時間來說或許有無負邊才是主要考量的，若圖形有負的邊我們就要採用 Bellman-Ford Algorithm 來做。

Conclusion

- 我認為這個作業幫助我很多，在上學期物件導向我也是在寫完 dungeon 的過程中才有所成長，這次的作業也給我了同樣的感覺，比起上課講義真正打開 VScode 打程式更能讓我有所吸收。在過程中當然也遇到很多困難，像是在 Kruskal Algorithm 裡我要怎麼去判斷 loop 就困擾我很久，經過上網查閱後，才知道有"走到最大點"的方法讓我印象深刻，當然我也更了解每個演算法的操作方法，時間複雜度，限制，適合處理的問題……。雖然因為疫情取消了上機考但我相信這份作業教給我的不亞於上機考。最後我沒什麼 feedback 要給 TA 的就謝謝你們的付出。

花費時間：

Quick Sort (2 hours)

Merge Sort(1.5 hours)

Prim Algorithm(3 hours)

Kruskal Algorithm(4 hours)

Dijkstra’s Algorithm(3 hours)

Bellman-Ford Algorithm(2.5 hours)

Report(5 hours)