姓名:葉冠宏 學號:r11943113

一、資料結構、演算法與發現

在程式實作中，我會有 partition.h 的檔案負責去紀錄兩邊有關 cell, net 綜觀上的資訊，例如: 所擁有的 cell 數目，以及所創建的 cell 和 net 的物件等等。而 cell.h 則負責去紀錄有關 cell 中所承載的資訊，例如: 紀錄其當時的 gain、所連接到的 net，所在的 part 是哪一個、是否已被 lock 住了等資訊。由於是二元的 partition，所以我們可以僅靠 0 或是 1 來去紀錄所在的位置。我令左邊的那一邊為 0 或是 A，右邊的那一邊為 1 或是 B。Net.h 則是去紀錄有關每個 net 的資料資訊，例如: 紀錄所連接的 0 那一邊和 1 那一邊的 cell 數目各有多少，還有就是所連接到的 cell 有哪些等等。Partition.cpp 則負責去實作演算法的部分。

在 F-M 演算法中，由於我們需要去尋找每一輪有最大的那個 gain 的 cell 是哪一個，因此為了可以更有效率的去找到，我們需要一個好的資料結構去維持紀錄每個 cell 的 gain。在本次實作中，我是用 map 與 linked list 的架構，我們稱作 bucket list 來去做紀錄。也就是說我會以所獲得的 gain 當作 map 的 key，然後相同的 gain 的 cell 或是 node 都會在對應的 value 上去進行串接。每個 cell 都會由 id 去直接取得其對應 node 的儲存位置。而每個 node 的資料結構有記載著前後是串接誰。因此我們僅需要去改變那個 node 所串接的狀態就可以達到插入和刪除的效果。這樣僅需要 O(1)的時間就可以做到了。由於現在有兩邊的 partition，所以我會去維持兩個 bucket list，分別用去紀錄兩邊的 gain 有哪些以及其對應的 node。由於在本次實作中，input 並沒有給出每個 cell 的 size 等資訊，因此我們可以僅靠一邊 cell 的數量就去判斷是否符合 balance 的條件。因此當如果我們要決定下一輪要移動誰的時候，當某一邊如果移動一個 cell 到另外一邊就會違反 balance 的條件的話，我們可以直接刪掉那一邊所有 cell 的移動的可能，因為沒有 size 的資訊，所以移動誰都會違反，所以我們就知道只能從另外一邊來去尋找最大的 gain 的 cell 來去移動。而由於 c++的 map 的架構是由 tree 來維持的，所以我們可以很快的直接知道最大的 gain 是什麼，然後再從那個 value 去取其 value node。這也就是為什麼我們需要兩邊的 bucket list。而在運作上，我們是每當 cell 的值如果有更新，才會去把原本的 gain 以及 node 的資訊刪除，然後再去 insert 一個新的 node。在 gain 的資料結構的設計上，我也曾經嘗試用 map 的 value 是 set 的架構，但發現 set 的 insert 和 erase 也需要 logN 的時間，因此還是沒有比較快。

在初始 partition 的分配上，我是採用每看到一個新的 net，其上的 cell 就都在新的一邊的方式來去做初始的 partition。我也嘗試用隨機的方式來去做分配，但發現效果只有在某些 dataset 才表現比較好，整體來說表現並不好。

每一輪要移動的 cell，我都會以 vector 的方式來去做紀錄。而由於在每一輪的實作中，我們需要等整個 cell 都 lock 住了才會知道最終和上一輪的最終態相比，我們需要怎麼移才可以達到整體的 gain 是正的，因此起初在實作上，我曾經嘗試先用暫時移動以及暫時的 part 的資訊的方式來去移動 cell，在知道要怎麼移動之後，再從上一輪的最終態直接照著所記錄的歷程去移動。但最後發現因為每一輪的 copy assignment 也會耗費時間，因此我最後是採用先直接改變整個 partition part 的狀態，cell 以及 net part 的狀態，然後最後多移的部分再去移回來的方式來去操作，這樣可以避免掉多餘 copy 的時間。

在每次計算是否要去更新每個 cell 是否要去更新 gain 時，我們可以先去檢查那個 net 是否有符合所有有可能需要去更新的條件，如果每一項都不符合，那我們就連看都不用，不需要去遍歷其所連接的每個 cell 了，這樣就可以省去大筆的運算時間。除此之外，我也曾經嘗試去紀錄每個 net 上動態的 unlock cells，這樣或許可以省去運算的時間，但發現每一次操作都要去從 set

中刪除的時間可能還是比直接去檢查每一個連接 cell 的狀態來的慢，所以我最後捨棄這個做法。

二、實驗結果

1.input_0.dat

```
[Eval] Run cmd: ./checker input_0.dat output_0.dat
[Check] Cut size = 4278 matched!
[Check] Balance passed:: 60300(min) < 89467(G1), 61283(G2) < 90450(max)
================================
Congratulations! Legal Solution!!
================================

[Eval] Your cutsize: 4278
[Eval] Your runtime: 1
[Eval] SCORE = QUALITY_SCORE*0.8 + RUNTIME_SCORE*0.2
             = 8.103678*0.8 + 8.862164*0.2
             = 8.25537
```

2. input_1.dat

```
[Eval] Run cmd: ./checker input_1.dat output_1.dat
[Check] Cut size = 1260 matched!
[Check] Balance passed:: 1485(min) < 1515(G1), 1485(G2) < 1515(max)
================================
Congratulations! Legal Solution!!
================================

[Eval] Your cutsize: 1260
[Eval] Your runtime: 1
[Eval] SCORE = QUALITY_SCORE*0.8 + RUNTIME_SCORE*0.2
             = 6.588239*0.8 + 4.3395127*0.2
             = 6.138493
```

3. input_2.dat

```
[Eval] Run cmd: ./checker input_2.dat output_2.dat
[Check] Cut size = 2187 matched!
[Check] Balance passed:: 3430(min) < 3561(G1), 3439(G2) < 3570(max)
================================
Congratulations! Legal Solution!!
================================

[Eval] Your cutsize: 2187
[Eval] Your runtime: 0.1
[Eval] SCORE = QUALITY_SCORE*0.8 + RUNTIME_SCORE*0.2
             = 8.4754915*0.8 + 7.3970615*0.2
             = 8.25980
```

4. input_3.dat

```
[Eval] Run cmd: ./checker input_3.dat output_3.dat
[Check] Cut size = 27722 matched!
[Check] Balance passed:: 29999.7(min) < 36665(G1), 30001(G2) < 36666.3(max)
================================
Congratulations! Legal Solution!!
================================

[Eval] Your cutsize: 27722
[Eval] Your runtime: 0.1
[Eval] SCORE = QUALITY_SCORE*0.8 + RUNTIME_SCORE*0.2
             = 7.3396598*0.8 + 12.00173*0.2
             = 8.27208
```

5. input_4.dat

```
[Eval] Run cmd: ./checker input_4.dat output_4.dat
[Check] Cut size = 45198 matched!
[Check] Balance passed:: 74621.2(min) < 74631(G1), 76119(G2) < 76128.8(max)
================================
Congratulations! Legal Solution!!
================================

[Eval] Your cutsize: 45198
[Eval] Your runtime: 1
[Eval] SCORE = QUALITY_SCORE*0.8 + RUNTIME_SCORE*0.2
             = 6.1883057*0.8 + 9.0514195*0.2
             = 6.76092
```

6. input_5.dat

```
[Eval] Run cmd: ./checker input_5.dat output_5.dat
[Check] Cut size = 143772 matched!
[Check] Balance passed:: 189332(min) < 193156(G1), 189333(G2) < 193157(max)
================================
Congratulations! Legal Solution!!
================================

[Eval] Your cutsize: 143772
[Eval] Your runtime: 1
[Eval] SCORE = QUALITY_SCORE*0.8 + RUNTIME_SCORE*0.2
             = 7.16432409*0.8 + 13.42264*0.2
             = 8.41599
```

三、結論

我們最後發現我所執行的程式都可以在每個 input data 去達到 balance 的效果，也盡量去縮減 cut size。然而其中還是有些 data 的 cutsize 結果並不理想，原因可能是因為 initial partition 的狀態就不好，使我們所得出的結果只是 local maximum，不是 global 的。但我也嘗試去改變 initial partition assign 的方式，例如隨機等等，或是去改變 link list 串接的順序等等，但發現這些改動效果並不顯著。所以未來如果有時間也可以嘗試看看其他 paper 是否有更好的方式去決定 initial partition。