CS6135 VLSI Physical Design Automation
Homework 2: Two-way Min-cut Partitioning
Student ID: 107065507 Name: Yun-Fan Lu

1. Goal

使用 C++實作 Fiduccia-Mattheyses (FM) Algorithm。

2. Design Concept

Define Data Structure:

```
struct Cell
{
    //string name;
    int size = 0;
    int gain = 0;
    int set = A;
    int pin = 0;
    bool isLocked = false;

    vector<string> this_net_list;
    list<string>::iterator  ptr;  // ptr to the bucket list
};

struct Net
{
    //string name;
    vector<string> this_cell_list;
};
```

struct Cell 中有 this_net_list 紀錄連接此 Cell 的 Net，但只記錄 Net 的 name。反之 struct Net 亦然。

```
unordered_map<string, Cell*> cellList;  // global cellList and netList
unordered_map<string, Net*> netList;
```

cellList 和 netList 則是使用 unordered_map 做為容器。
原因如下：因為 STL map 的查詢很方便，但是 operator time (=, []) 是 O(logn)，後來查到有個東西叫做 unordered_map，不會照著鍵值排序，但是可以優化 operator = time 為 O(n)，operator [] time 為 O(1)。而 cellList 或 netList 確實也不需要排序。實際將原本的 map 改成 unorder_map 後，發現速度的確有提升，故使用 unorder_map 做為容器。

```
map<int, list<string>> bucket_A;
map<int, list<string>> bucket_B;
```

Bucket list 為一個 map。Key 為 gain，value 為一 list，內容是 Cell name。且必須使用 map，因為要排序 gain 值。

FM Algorithm:

```
void FM()
{
    int _Gk = 999999;

    while(true){
        _Gk = Gk();
        if(_Gk <= 0)
            break;

        init_all();
        init_gain();
        make_bucket_list();
    }
}
```

FM()的主體是計算出此 iteration 的 maximum partial sum Gk，然後根據此 partition 重新計算 gain (init_gain())和重做 bucket list (make_bucket_list())。

```
int Gk()
{
    int lockedNum = 0;
    string Max_cell;
    stack<string> st;  // to restore cell
    vector<int> Gk;  // store partial Gk sum

    while(lockedNum != C_NUM){
        Max_cell = get_max_gain();
        remove_cell(Max_cell);
        update_gain(Max_cell);
        st.push(Max_cell);
        Gk.push_back(cellList[Max_cell]->gain);

        lockedNum++;
        //cout << lockedNum << endl;

    }
```

Gk()為計算 partial sum Gk 的函式：先拿到 max gain 的 cell name，再去 remove_cell 和 update_gain 等動作，直到全部 cell 都為 locked。

get_max_gain()和 remove_gain()的複雜度在 discussion 有詳細分析。

Find a cell with max gain: (get_max_gain())

Time complexity = $O(P_{max} \log P_{max})$

Remove a Cell i from the structure: (remove_gain())

在 map 上做 operator []和 erase 的操作。

Time complexity = $O(\log P_{max})$

update_gain()的分析如下：

根據講義上 update cell gains 的方法實作。只有跟 base cell 有連接的 net 的 cell 的 gain 值會更動。更動 bucket list 的複雜度為 $O(\log P_{max})$。

Time complexity = $O(C*P*\log P_{max})$

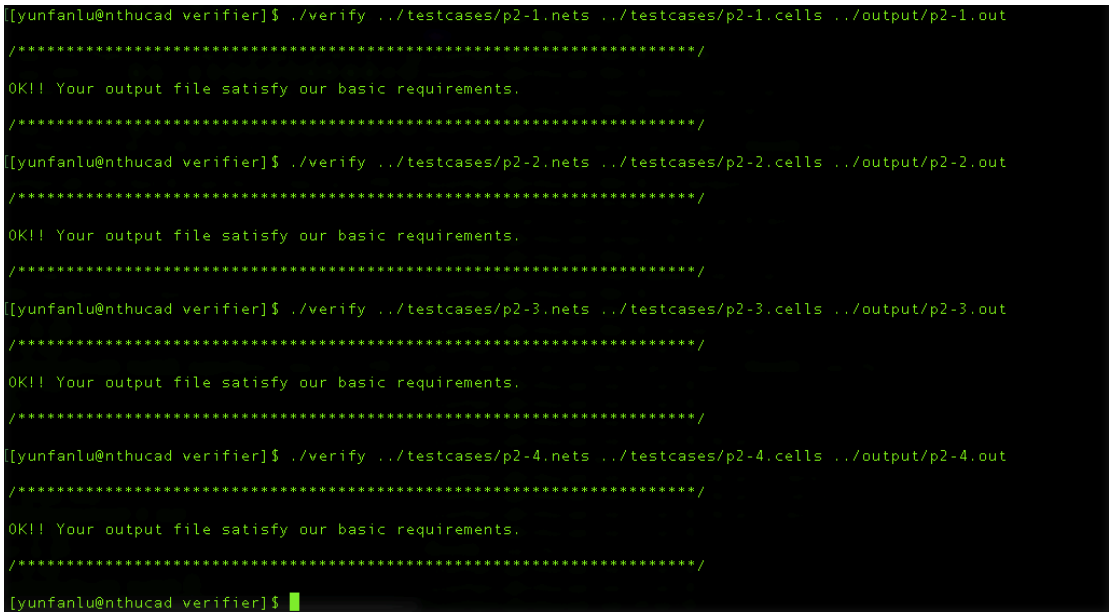
這個部分是我程式的 bottle neck。

所以整個計算 Gk()函式的複雜度為：

Time complexity = $O(C^2*P*\log P_{max})$

這也是整個 FM Algorithm 的時間複雜度。

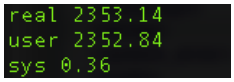
3. Results



verify 結果皆符合規則。

	p2-1	p2-2	p2-3	p2-4
cutsize	21	494	2957	39792
Runtime	0.04	1.17	775.20	2353.14
I/O time	0.00	0.00	0.18	0.36

I/O time 計算說明：使用指令**$ /usr/bin/time -p** 可計算時間，並且有三個數字如下圖，因為 I/O 是 system 層面的動作，所以 sys time 可視為 I/O time。



4. Discussion

Where is the difference between your algorithm and FM Algorithm described in class? Are they exactly the same?

基本上是照著講義的方式去做實作，但複雜度可能不會像講義上那麼理想，以下問題會詳細分析我的實作的 complexity。

Did you implement the bucket list data structure?

```
map<int, list<string>> bucket_A;
map<int, list<string>> bucket_B;
```

Bucket list 為一個 map。Key 為 gain，value 為一 STL list，內容是 Cell name。找 max gain 的方式是從 key 最大開始往下找(O(lgP_{max}))，如果此 list 為 empty 則繼續找下一個。找到之後拿到 Cell name 並從 cellList 拿到此 Cell (O(1))並測試 balance constrain，如果都符合就 return max_cell_gain 的 Cell name。

以下是一些基本操作的時間複雜度：

Find a cell with max gain:

Time complexity = O(P_{max} logP_{max})

Remove a Cell i from the structure:

在 map 上做 operator []和 erase 的操作。

Time complexity = O(logP_{max})

How did you find the maximum partial sum and restore the result?

在移動 max gain cell 時會將此 Cell name push 進一個 stack，最後找 partial max Gk sum 的 turn 時跟總數相減，代表要 pop 的次數。而後只做 move cell 動作，不更改 bucket list 和 gain 值，因為如果此次 Gk>0，會再重新做一次 FM，待那時再重新計算即可。

Please compare your results with the top 5 students' results from last year and show your advantage either in runtime or in solution quality. Are your results better than them?

跟去年 top 5 相比，前兩個小測資明顯差於他們，但到 p2-4 的 cutsize 竟然比去年前五名都還要好。而我程式時間方面，真的非常差，這個是我日後需要解決的問題。

What else did you do to enhance your solution quality or to speed up your program?

使用 unorder_map 而不是 map 當作 cellList/netList。因為 FM 演算法很常需要查詢(operator [])，如果是一般 map 需要 O(1)，而 unordered_map 只需要 O(1)時間。

另外，initial partition 的方式是照照 input 順序將 Cell 均分為兩堆。使用 map 而不是 unordered_map 的話會照字典序排列，heuristic algorithm 的結果可能會不一樣。舉例來說，我使用前者的話，p2-1 得到的 cutsize 會是 15，比目前繳交的這份程式還要好，但是考慮到 run time，還是使用

unordered_map，雖然也沒有提升很多就是了。

What have you learned from this homework? What problem(s) have you encountered in this homework?

資料結構的選擇很重要，而寫程式的方便性和時間複雜度也是需要做 trade-off 的。例如使用 map 容器，但犧牲 O(logn)時間做查詢。

迴圈內的動作順序，以及 break 的時機很重要，可以避免許多其實不需要的動作。

學到了一些以前沒什麼在用的 STL，例如說 STL list 是一個 double linked list，erase()裡面是填入 iterator type。所以把 struct Cell 中單純的 pointer to string 改成 list<string>::iterator。

當測資變大時，我的程式跑得非常慢，今後要克服這個大問題，尤其 problem size 在 EDA 領域又是特別重要。