PA2_report

108062133 劉廷宜

I. Execution

執行方法符合SPEC所述

```
cd CS3130_PA2
make
./pa2 <.in file> <.out file>
```

以下是flow chart

II. Data Structure & Implementation

- **class Node**
  主要用來建立cell gain的bucket list

```
class Node{
    friend class Cell;
public:
    Node(const int& id) :
        _id(id), _prev(NULL), _next(NULL) { }
    ~Node() { }

    ...

private:
    int         _id;
    Node*       _prev;
    Node*       _next;
};
```

- **class Cell**
  包含一個Cell的所有資訊
  private members
  _id：這個cell的編號
  _size：這個cell的大小
  _group：這個cell現在所屬的group（會因為FM-Heuristic改動）
  _gain：這個cell現在的gain（會因為FM-Heuristic改動）
  _lock：代表這個cell是否被選取過
  _node：這個cell的Node*，來做出cell gain的bucket list
  _netList：這個cell有連接哪些net

```
class Cell{
public:
    Cell(int id, int size) :
        _id(id), _size(size), _gain(0), _lock(false), _group(-1){
        _node = new Node(id);
    }
    ~Cell() {}
    ...
private:
    int         _id;
    int         _size;
    int         _group;
    int         _gain;
    bool        _lock;
    Node*       _node;
    vector<int> _netList;
};
```

- **class Net**
  包含一個net的所有資訊
  private members
  _netId：這個net的編號
  _groupCount：紀錄這個net在任個group中的cell的數目
  _lockGroup：紀錄這個net在任個group中是否有lock的cell
  _cellList：這個net有連接哪些cell

```
class Net{
public:
    Net(int netId) : _netId(netId), _netSize(0) {}
    ~Net() {}
    ...
private:
    int         _netId;
    int         _netSize;
    int*        _groupCount;
    bool*       _lockGroup;
    vector<int> _cellList;
};
```

- **class Partitioner**
  實作FM-heuristic的資料結構
  後面會詳述函式的實作

```
class Partitioner {
public:
    Partitioner() : _accGain(0), _maxAccGain(0), _totalSize(0),
_maxCellSize(-1){
    }
    ~Partitioner() {}

    void init();            // 初始化
    void initPartition();   // 剛開始先隨便分成兩組
    int costCal();          // 計算cost
    int findMaxGainCell(); // 找符合大小限制且gain最大的cell
    int twoWayPass();       // 進行一輪two-way的交換
    void Partition();       // 決定進行two-way或k-way
    void initBuckets();
    /* 在跑交換前先依據各個cell所在的group計算出這個cell的gain並初始化cell
gain的bucket list*/
    void updateCellGains(Cell*, unordered_map<int, int>&);
    // 選擇maxGainCell後更新其他被牽連到的cell的gain
    void updateBuckets(Cell*, unordered_map<int, int>&);
    // 利用剛才更新過的cell gain結果來更新bucket list
    void rollBackToBest();
    // 一輪交換結束後回到交換過程中的最佳狀態
```

```
private:
    int                         _cost;         // output的cost
    int                         _areaLimit;    // 每個group的大小限制
    int                         _groupNum;     // group的總數
    int                         _netNum;       // net的總數
    int                         _cellNum;      // cell的總數
    int                         _totalSize;    // 全部cell加總的空間大小
    int                         _maxCellSize; // 最大的cell的大小
    int*                        _groupSize;    // 紀錄各個group的大小
    Node*                       _maxGainCell; // 選出的maxGainCell
    vector<Net*>                _netArray;
    vector<Cell*>               _cellArray;
    map<int, Node*>             _bList;
    /* cell gain的bucket list (key即為gain)
    而Node*是由double linked list方式連接
    因為map的key是由小排到大
    所以如果要找maxGainCell只要利用reverse_iterator即可*/

    int                         _bestMoveNum;
    vector<int>                 _moveStack;

    void cleanUp();
};
```

2-Way Partition Implementation

- **void Partitioner::Partition()**
  在得到的maxAccGain>0的情況下不斷進行twoWayPass()

```
if(_groupNum == 2){
    int maxAccGain;
    do{
        maxAccGain = twoWayPass();
        cleanUp();
    }while(maxAccGain > 0);
}
```

- **int Partitioner::twoWayPass()**
  主要是進行一輪FM-Heuristic中一次完全交換的過程

```
int Partitioner::twoWayPass(){
    initBuckets();
    int moveNum = 0, accGain = 0, maxAccGain = 0;
    while(moveNum < _cellNum){
        int cellId = findMaxGainCell();
        if(cellId == -1) break;
        _moveStack.__emplace_back(cellId);
        Cell* c = _cellArray[cellId];
        accGain += c->getGain();
```

```
        if(accGain > maxAccGain){
            maxAccGain = accGain;
            _bestMoveNum = moveNum;
        }
        unordered_map<int, int> potentialCell;
        updateCellGains(c, potentialCell);
        updateBuckets(c, potentialCell);
        moveNum++;
    }
    rollBackToBest();
    return maxAccGain;
}
```

- **void Partitioner::initBuckets()**
  初始化cell gain的bucket list
  bucket list由map<int, Node*>實現

```
void Partitioner::initBuckets(){
    for(int i = 0; i < _cellNum; i++){
        Cell* c = _cellArray[i];
        Node* cNode = c->getNode();
        int fromBlock = c->getGroup();
        int toBlock = !fromBlock;
        vector<int> n;
        c->getNetList(n);
        // 初始化cell的gain
        if(n.empty()){
            c->setGain(0);
        }else{
            for(int j = 0; j < n.size(); j++){
                if(_netArray[n[j]]->getGroupCount(fromBlock) == 1)
                    c->incGain();
                if(_netArray[n[j]]->getGroupCount(toBlock) == 0)
                    c->decGain();
            }
        }
        // 將cell的Node*放入bucket list中
        if(_bList.find(c->getGain()) == _bList.end()){
            _bList[c->getGain()] = c->getNode();
        }else{
            Node* curHead = _bList[c->getGain()];
            _bList[c->getGain()] = cNode;
            curHead->setPrev(cNode);
            cNode->setNext(curHead);
        }
    }
}
```

- **int Partitioner::findMaxGainCell()**
  找可以符合條件限制且gain最大的cell

```
int Partitioner::findMaxGainCell(){
    /*由於map的key是由小排到大所以這邊使用reverse_iterator來找maxGainCell*/
    map<int, Node*>::reverse_iterator it;
    for(it = _bList.rbegin(); it != _bList.rend(); it++){
        Node* nd = it->second;
        while(nd != NULL){
            Cell* c = _cellArray[nd->getId()];
            int cGroup = c->getGroup();
            if(_groupSize[1 - cGroup] + c->getSize() < _totalSize / 2
+ _maxCellSize){
                _groupSize[cGroup] -= c->getSize();
                _groupSize[1 - cGroup] += c->getSize();
                return c->getId();
            }
            nd = nd->getNext();
        }
    }
    return -1;
}
```

- **void Partitioner::updateCellGains(Cell*, unordered_map<int, int>&)**
  依照課本方式在選定maxGainCell後更新其他有連到的cell的gain
  hashTable是將有可能會變動gain的cell紀錄進去
  first為cell ID、second為cell變動的gain

```
void Partitioner::updateCellGains(Cell* c, unordered_map<int, int>&
hashTable){
    vector<int> n;
    c->getNetList(n);
    int fromBlock = c->getGroup();
    int toBlock = 1 - fromBlock;
    c->move(toBlock);
    c->lock();
    for(int i = 0; i < n.size(); i++){
        Net* net = _netArray[n[i]];
        int dif = 0;
        if(!net->getLock(_groupNum)){
            if(net->getGroupCount(toBlock) == 0){
                vector<int> cList;
                net->getCellList(cList);
                for(int j = 0; j < cList.size(); j++){
                    if(_cellArray[cList[j]]->getLock() == false){
                        _cellArray[cList[j]]->incGain();
                        hashTable[cList[j]]++;
                    }
                }
            }else if(net->getGroupCount(toBlock) == 1){
```

```
                vector<int> cList;
                net->getCellList(cList);
                for(int j = 0; j < cList.size(); j++){
                    if(_cellArray[cList[j]]->getLock() == false &&
_cellArray[cList[j]]->getGroup() == toBlock){
                        _cellArray[cList[j]]->decGain();
                        hashTable[cList[j]]--;
                    }
                }
            }
        }
        net->decGroupCount(fromBlock);
        net->incGroupCount(toBlock);
        if(!net->getLock(_groupNum)){
            if(net->getGroupCount(fromBlock) == 0){
                vector<int> cList;
                net->getCellList(cList);
                for(int j = 0; j < cList.size(); j++){
                    if(_cellArray[cList[j]]->getLock() == false){
                        _cellArray[cList[j]]->decGain();
                        hashTable[cList[j]]--;
                    }
                }
            }else if(net->getGroupCount(fromBlock) == 1){
                vector<int> cList;
                net->getCellList(cList);
                for(int j = 0; j < cList.size(); j++){
                    if(_cellArray[cList[j]]->getLock() == false &&
_cellArray[cList[j]]->getGroup() == fromBlock){
                        _cellArray[cList[j]]->incGain();
                        hashTable[cList[j]]++;
                    }
                }
            }
        }
        net->lockGroup(toBlock);
    }
}
```

- **void Partitioner::updateBuckets(Cell*, unordered_map<int, int>&)**
  首先將選定的MaxGainCell移出bucket list外
  再來將剛剛的hashTable中的cell從舊的gain上拔掉接到新的gain上

```
void Partitioner::updateBuckets(Cell*c, unordered_map<int, int>&
hashTable){
    Node* cNode = c->getNode();
    int cGain = c->getGain();
    if(_bList[cGain] == cNode){
        if(cNode->getNext() == NULL){
            _bList.erase(cGain);
        }else{
```

```
            Node* cNext = cNode->getNext();
            _bList[cGain] = cNext;
            cNext->setPrev(NULL);
        }
        cNode->setPrev(NULL);
        cNode->setNext(NULL);
    }else{
        if(cNode->getNext() == NULL){
            Node* cPrev = cNode->getPrev();
            cPrev->setNext(NULL);
        }else{
            Node* cPrev = cNode->getPrev();
            Node* cNext = cNode->getNext();
            cPrev->setNext(cNext);
            cNext->setPrev(cPrev);
        }
        cNode->setPrev(NULL);
        cNode->setNext(NULL);
    }
    for(unordered_map<int, int>::iterator it = hashTable.begin(); it
!= hashTable.end(); it++){
        if(it->second != 0){ // 代表這個cell的總變動值大於0
            Cell* cell = _cellArray[it->first];
            cNode = cell->getNode();
            cGain = cell->getGain();
            if(_bList[cGain - it->second] == cNode){
                if(cNode->getNext() == NULL){
                    _bList.erase(cGain - it->second);
                }else{
                    Node* cNext = cNode->getNext();
                    _bList[cGain - it->second] = cNext;
                    cNext->setPrev(NULL);
                }
            }else{
                if(cNode->getNext() == NULL){
                    Node* cPrev = cNode->getPrev();
                    cPrev->setNext(NULL);
                }else{
                    Node* cPrev = cNode->getPrev();
                    Node* cNext = cNode->getNext();
                    cPrev->setNext(cNext);
                    cNext->setPrev(cPrev);
                }
            }
            cNode->setPrev(NULL);
            cNode->setNext(NULL);
            if(_bList.find(cGain) == _bList.end()){
                _bList[cGain] = cNode;
            }else{
                Node* curHead = _bList[cGain];
                _bList[cGain] = cNode;
                curHead->setPrev(cNode);
                cNode->setNext(curHead);
            }
```

```
        }
    }
}
```

- **void Partitioner::rollBackToBest()**
  在TwoWayPass()中有紀錄跑完交換過程中的最好狀態
  將一切恢復到那個狀態

```
void Partitioner::rollBackToBest(){
    for(int i = _bestMoveNum + 1; i < _moveStack.size(); i++){
        Cell* c = _cellArray[_moveStack[i]];
        int cGroup = c->getGroup();
        vector<int> netList;
        c->getNetList(netList);
        for(int j = 0; j < netList.size(); j++){
            _netArray[netList[j]]->decGroupCount(cGroup);
            _netArray[netList[j]]->incGroupCount(!cGroup);
        }
        c->setGroup(!cGroup);
        _groupSize[cGroup] -= c->getSize();
        _groupSize[!cGroup] += c->getSize();
    }
}
```

- **void Partitioner::cleanUp()**
  在結束twoWayPass()後將一切回復原狀準備下次的twoWayPass()

```
void Partitioner::cleanUp(){
    _bList.clear();
    for(int i = 0; i < _cellNum; i++){
        _cellArray[i]->setGain(0);
        _cellArray[i]->unlock();
        Node* cNode = _cellArray[i]->getNode();
        cNode->setPrev(NULL);
        cNode->setNext(NULL);
    }
    for(int i = 0; i < _netNum; i++){
        _netArray[i]->unlockGroup(_groupNum);
    }
    _moveStack.clear();
    _bestMoveNum = 0;
}
```

k-Way Partition Implementation

- **void Partitioner::Partition()**
  因為code冗長、也沒有使用到特別的資料結構所以用打字說明
  我本來有用2-way去完成k-way（就是先分成兩個再分成四個以此類推）
  但是出來的效果奇差無比（我也很疑惑為什麼會這樣）
  所以改成直接用net擺放cell，儘量讓net不要span太多group
  如果這個group擺滿了就放下一個group、以此類推

Output File

- **void Partitioner::Partition()**
  用costCal()算出cost
  將SPEC規定的內容以規定格式輸出

```
if(_groupNum == 2) // 2-way partition
else // k-way partition
_cost = costCal();
cout << "cost: " << _cost << endl;
out_file << _cost << endl;
out_file << _groupNum << endl;
for(int i = 0; i < _cellNum; i++){
    out_file << _cellArray[i]->getGroup() << endl;
}
```

- **int main(int argc, char* argv[])**
  讀取指令、初始化、Partition

```
int main(int argc, char* argv[]){
    in_file.open(argv[1]);
    out_file.open(argv[2]);
    Partitioner* pa = new Partitioner();
    pa->init();
    pa->Partition();
    return 0;
}
```