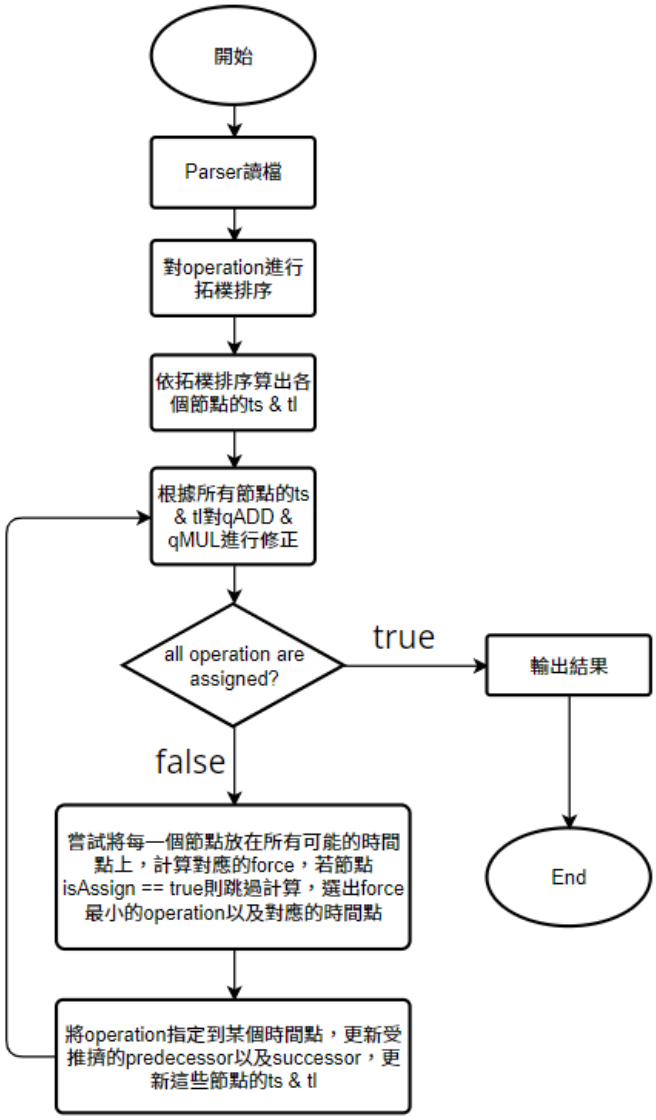
110521167_CAD_PA2_Report

一、流程圖 & main( )




```cpp
int main(int argc, char **argv)
{
    int assignTime;
    Node *minForceOp;
    Parser ps(argc, argv);
    DAG *graph = ps.readInput();
    ForceDirected fd(graph);

    // Scheduling
    fd.ASAP_ALAP();
    while(fd.get_nOpNotAssigned() > 0)
    {
        tie(minForceOp, assignTime) = fd.selectOp();
        fd.assignOp(minForceOp, assignTime);
    }
    ps.outputResult(fd.opExeTime);

    delete graph;
    return EXIT_SUCCESS;
}
```

二、資料結構

1. Node: 儲存 operation information

```
struct Node
{
    enum operation_type
    {
        NOP, // non of operation
        ADD, // 1 cycle operation
        MUL  // 3 cycle operation
    };
    bool flag, isAssigned;   // flag: for DFS, isAssigned: for ForceDirected
    int ID, opType,          // ID: number of this operation, opType: operation type
        ts, tl;              // ts: time in ASAP, tl: time in ALAP
    vector<Node *> pre, suc; // pre: predecessor of this node, suc: successor of this node
    double prob, force;      // probability & force

    int mobility() { return tl - ts; }
    double getTimeProb(int time) { return ts <= time && time <= tl ? prob : 0; }
    Node(int _ID, char symbol) : ID(_ID) // constructor
    {…
    }
};
```

2. DAG: 儲存 operation 之間的相依關係

```
struct DAG
{
    int max_latency, nOperation; // max_latency & number of operations in this DAG
    Node *SNOP, *TNOP;           // SNOP: source of this DAG, TNOP: tail of this DAG
    vector<Node *> operation;

    DAG() // constructor
    {…
    }
    ~DAG() // destructor
    {…
    }
};
```

3. TimeSlot: 儲存每一個 time level 中佔據 resource 的 operation

```
struct TimeSlot
{
    int nADD, nMUL;     // nADD: number of adders, nMUL: number of multiplier
    vector<Node *> op; // operations

    void insert(Node *n) // insert a operation into this time slot
    {…
    }
};
```

4. ForceDirected: 實現 scheduling 中包含的 function

```
class ForceDirected
{
private:
    int max_latency, nOpNotAssigned;
    DAG *graph;
    vector<double> qADD, qMUL;
    vector<int> bestTime; // for selectOp()

    void topoDFS(Node *, vector<Node *> &);
    vector<Node *> topologicalSort();
    void initialNodeTsTl(vector<Node *> &);
    void initialQk();

    double calcForce(Node *, int);
    double calcSelfForce(Node *, int);
    double calcSucForce(Node *, int);
    double calcPreForce(Node *, int);

    void fixPreTl(Node *, int);
    void fixSucTs(Node *, int);
    void fixProb(Node *, int);

    void insertOpExeTime(Node *);
public:
    vector<TimeSlot> opExeTime; // for parser output()
    ForceDirected(DAG *);
    void ASAP_ALAP();
    pair<Node *, int> selectOp();
    void assignOp(Node *, int);
    int get_nOpNotAssigned() const;
    // void scheduling();
};
```

ASAP_ALAP( )

呼叫 topologicalSort( )對 DAG 進行排序，接著呼叫 initialNodeTsTl( )進行 ASAP & ALAP，計算出每個節點的 ts 以及 tl，接著呼叫 initialQk( )，根據節點的 operation type 以及對應的 ts & tl 計算出 qk，分別存儲在 qADD & qMUL 這兩個向量中。

selectOp(Node *minForceOp, int assignTime)

用迴圈對還沒被 schedule 的節點呼叫 calcForce( )去計算將節點指定在區間 [ts, tl] 中任一個時間點所產生的 force，在迴圈跑完後回傳 force 最小的節點，以及該節點能產生最小的 force 的時間點。

assignOp(Node *minForceOp, int assignTime)

根據 selectOp( )得到的結果，將節點 schedule 到最佳的時間點，將節點標記為 isAssigned，接著更新與之相聯的節點的 ts、tl、probability，以及更新 qADD 或 qMUL 受影響的區間，供下一個 iteration 使用。

三、執行方式

$make all

```
[110521167@eda359_forclass ~/PA2]$ make all
```

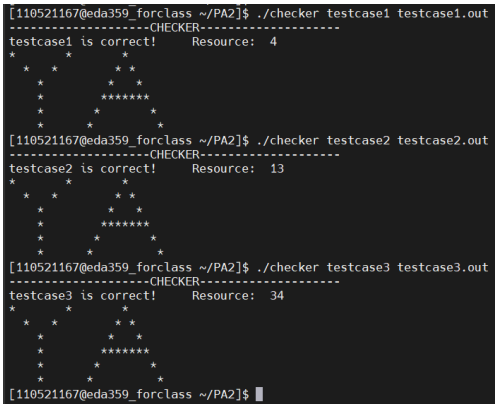
$make run input={input file} output={output file}

```
[110521167@eda359_forclass ~/PA2]$ make run input=testcase3 output=testcase3.out
```

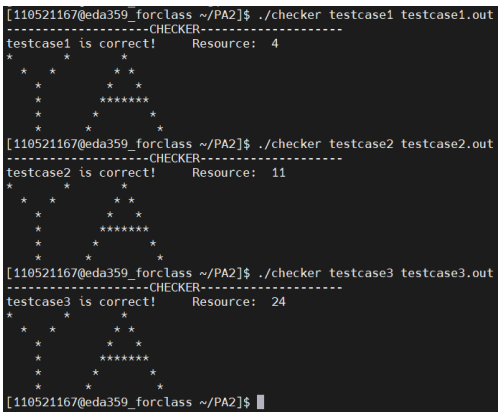
$make clean

```
[110521167@eda359_forclass ~/PA2]$ make clean
rm: cannot remove `rm': No such file or directory
make: *** [clean] Error 1
```

四、實驗數據



(圖一) ASAP　　(圖二) Force-directed one-layer



(圖三) Force-directed multilayer

(圖四) Force-directed one-layer multiplier first

ASAP	Testcase1	Testcase2	Testcase3
Adder	3	5	9
Multiplier	1	8	25

Force-directed-1-layer	Testcase1	Testcase2	Testcase3
Adder	3	3	5
Multiplier	1	8	19

Force-directed-multilayer	Testcase1	Testcase2	Testcase3
Adder	3	2	5
Multiplier	1	9	19

Force-directed-1-layer MUL first	Testcase1	Testcase2	Testcase3
Adder	3	2	4
Multiplier	1	8	19

五、實驗結論

在這 testcase2 以及 testcase3 中 Force-directed 明顯優於 ASAP，這是因為計算 ps-force 時不管是只考慮前後一級或是遞推到多層有關連的 operation，total force 都會引導我們得到一個較佳的解，但有趣的是在 testcase2 中雖然使用的 resource 總數一樣，但是使用的 adder 和 multiplier 數量卻不相同，這主要是因為會有多個 operation 的 total force 相同，且兩種方法將 operation 指定到某個時間點的順序不同導致兩個方法結果不一樣，若是以硬體複雜度來說 Force-directed one-layer 會得到最佳解。

受到前一個結論的啟發，我在 total force 相同的節點上總是優先選擇乘法運算，在實踐到 one-layer 和 multilayer 後發現 one-layer 不只效能較好，得出的結果也能再將 testcase2 及 testcase3 使用的 resource 數量降低 1，在兩個 testcase 中乘法和加法運算數量並沒有明顯的差異，因此我推測在運算數量差距不大的情況下，對於 cycle 較長的運算優先指定 time slot 能得到較好的解。

或許之後若有包含多種 cycle 運算的測資，就能夠透過在 force 上加權，或是在選擇時通過 if else 來實現並驗證我的想法是否正確。