Scheduling

2017/5/2

范真瑋

List Scheduling

實驗內容：

撰寫List Scheduling的C/C++程式

程式碼：

```
for(i = 0; i < end; ++i) {
    int j;
    for(j = i-1; j >= 0; --j) {
        //尋找運算元出現的位置
        if(sample1[list1[j].num].result==sample1[i].op1 || sample1[list1[j].num].result==sample1[i].op2) {
            state = list1[j].state + 1;  //運算元出現state+1
            break;
        }
    }

    while(1) {
        //state是否有足夠的加法器、乘法器
        if(list1[i].op == 1) {
            if(alulist[state].add > 0) {
                --alulist[state].add;
                break;
            }
            else {
                ++state;
            }
        }
        else {
            if(alulist[state].mult > 0) {
                --alulist[state].mult;
                break;
            }
            else {
                ++state;
            }
        }
    }
    list1[i].state = state;  //設定state
    //以state值為目標執行sorting由小到大
    qsort(list1, i, sizeof(struct readylist), compare);
}
```

程式說明：

Input 方式：

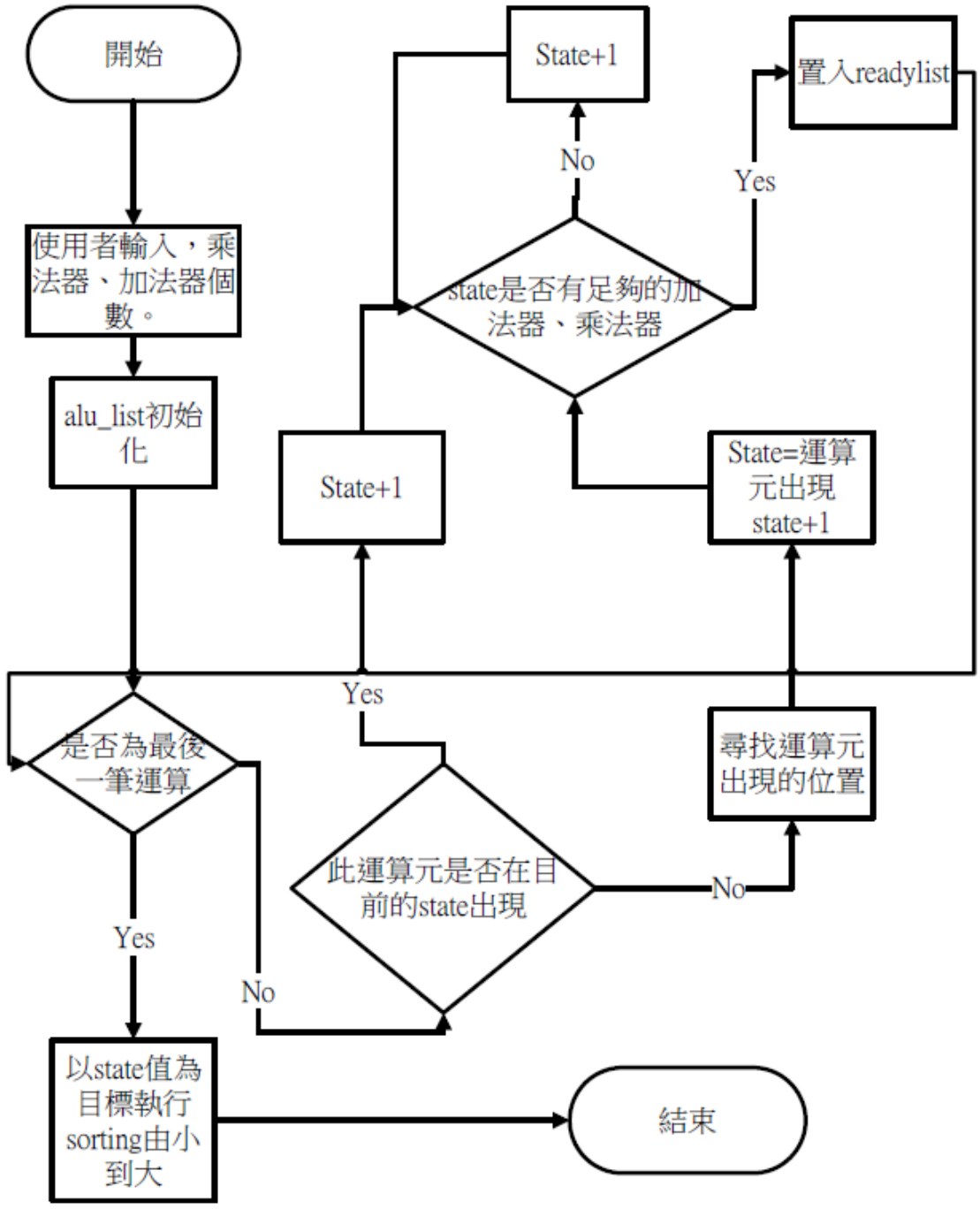
兩組 Data Flow Graph 分別以.txt 的方式做資料建構，分別為 p1.txt 及 p2.txt。

Output 方式：

產生一個 Scheduling_outcome.txt，裡面紀錄了每次運算的結果。

使用說明：

1. 選擇要載入哪一組 Data Flow Graph。欲選擇 p1 則輸入 p1.txt，選擇 p2 則輸入 p2.txt。
2. 輸入 resource constraint 乘法器跟加法器數目。
3. 程式會將排程結果顯示在銀幕上，同時也會寫入 Scheduling_outcome.txt。
4. 如果需要其他數目的 resource constraint 則輸入 1 繼續，輸入 2 結束。



實驗結果及分析：

```
*********************************
*       Resource Constraint     *
*                               *
*       Mult Constraint:2       *
*       Add Constraint:2        *
*                               *
*********************************
State: 1        v 10 = v  1 + v  2
State: 1        v 11 = v  6 + v  7
State: 2        v 12 = v 10 + v  3
State: 2        v 13 = v 11 + v  8
State: 3        v 15 = v 13 + v  9
State: 3        v 14 = v 12 + v  4
State: 4        v 16 = v 14 + v 15
State: 5        v 18 = v 16 * v 16
State: 5        v 17 = v 16 * v 16
State: 6        v 19 = v 17 + v 12
State: 6        v 23 = v 18 + v 15
State: 7        v 22 = v 19 + v 16
State: 7        v 20 = v 12 + v 19
State: 8        v 25 = v 23 + v 15
State: 8        v 24 = v 22 + v 23
State: 8        v 21 = v 20 * v 20
State: 9        v 26 = v 10 + v 21
State: 9        v 27 = v 25 * v 25
State: 10       v 29 = v 26 + v 19
State: 10       v 28 = v 10 + v 26
State: 11       v 30 = v 28 * v 28
State: 11       v 32 = v 27 + v  9
State: 11       v 31 = v 29 + v  5
State: 12       v 33 = v 31 * v 31
State: 12       v 35 = v 32 + v  9
State: 12       v 34 = v 23 + v 32
State: 13       v 38 = v 35 * v 35
State: 13       v 37 = v 11 + v 34
State: 13       v 36 = v  1 + v 30
State: 14       v 41 = v 37 * v 37
State: 14       v 40 = v 33 + v  5
State: 14       v 39 = v 36 + v 26
State: 15       v 42 = v 31 + v 40
State: 15       v 43 = v 32 + v 38
```

每個 state 皆無資料相依，且運算元一定在被計算出來後才使用。