(1) How to compile and execute your program

In the data flow graph, nodes stand for operations. An operation can be executed only when it does not have parents. In other words, the highest nodes (top most in a DFG diagram) are the operations available to be executed at the moment. If an operation is ready for running, assign time step to it as soon as possible. Then increase the time step and assign the currently highest nodes. That means the ASAP scheduling technique will assign all the available operations in each time step.

(2) The completion of the assignment

I have completed asap and alap and now I almost have to complete the forced direction。

First I determine a struct name Node, Edge inside the node have some information

```
struct Node {
    int number;
    string type;
    vector<struct Node*> to;
    vector<struct Node*> from;
};

struct Edge {
    int from;
    int to;
};

struct ScheduledNode {
    struct Node *node;
    int scheduledTime;
};
```

OperationOfTime is enter a schedule and time , to calculate all of schedule time node and put in the vector.

```
int getScheduledTime(vector<struct ScheduledNode> *schedule, struct Node *node) {
    for (int i=0; i<schedule->size(); i++) {
        if ((*schedule)[i].node == node) return (*schedule)[i].scheduledTime;
    }
    return -1;
}

void operationsOfTime(vector<struct ScheduledNode> *schedule, int time, vector<struct Node *> *nodes) {
    for (int i=0; i<schedule->size(); i++) {
        struct ScheduledNode *vi = &(*schedule)[i];
        int runningTime = cost(vi->node);
        int startTime = vi->scheduledTime;
        int endTime = startTime + runningTime;
        if ((startTime <= time) && (time < endTime)) {
            nodes->push_back(vi->node);
        }
    }
}
```

To count all the adder

```
int countAdder(vector<struct ScheduledNode> *schedule, int latency) {
    int max = -1;
    for (int i=0; i<=latency; i++) {
        vector<struct Node *> nodes;
        operationsOfTime(schedule, i, &nodes);
        int count = 0;
        for (int j=0; j<nodes.size(); j++) {
            if (nodes[j]->type == "+") count++;
        }
        if (count > max) max = count;
    }
    return max;
}
```

To count all the multipier

```
int countMultiplier(vector<struct ScheduledNode> *schedule, int latency) {
    int max = -1;
    for (int i=0; i<=latency; i++) {
        vector<struct Node *> nodes;
        operationsOfTime(schedule, i, &nodes);
        int count = 0;
        for (int j=0; j<nodes.size(); j++) {
            if (nodes[j]->type == "*") count++;
        }
        if (count > max) max = count;
    }
    return max;
}
```

To schedule all scheduleNode time max.

```
int totalTime(vector<struct ScheduledNode> *schedule) {
    int max = -1;
    for(int i=0; i<schedule->size(); i++){
        int ti = (*schedule)[i].scheduledTime;
        if (ti > max) max = ti;
    }
    return max;
    // 算出schedule裡面所有ScheduledNode的scheduledTime的max
}
```

Asap:

```
void asap(vector<struct ScheduledNode> *asapSchedule, vector<struct Node> *nodes) {
    // 把type為i的node都排入schedule
    for (int i=0; i<nodes->size(); i++) {
        if ((*nodes)[i].type == "i") {
            struct Node *v0 = &(*nodes)[i];
            doSchedule(asapSchedule, v0, 0);
        }
    }
    while (asapSchedule->size() < nodes->size()) {                      // 還有nodes沒排進schedule
        for (int i=0; i<nodes->size(); i++) {
            struct Node *vi = &(*nodes)[i];
            if (isScheduled(asapSchedule, vi) == false) {               // vi還沒被排進schedule的話
                                                                        // 判斷vi前面的node是否都排進schedule了
                bool allVjAreScheduled = true;                          // 先假設vi前面全部的vj都排進去了
                for (int j=0; j<vi->from.size(); j++) {
                    struct Node *vj = vi->from[j];
                    if (isScheduled(asapSchedule, vj) == false) {       // 只要有一個vj其實還沒排進去
                        allVjAreScheduled = false;                      // 就代表這個假設是錯的
                        break;                                          // 就可以不用再檢查下去了
                    }
                }
                // 如果都排進去了，才去算vi應該被排到的時間
                if (allVjAreScheduled == true) {
                    int max = -1;
                    for (int j=0; j<vi->from.size(); j++) {             // 看vi前面得所有vj
                        struct Node *vj = vi->from[j];
                        int tj = getScheduledTime(asapSchedule, vj);
                        if (tj + cost(vj) > max) {                      // 如果這個vj的tj+dj > max
                            max = tj + cost(vj);                        // 就把這個tj+dj設為新的max
                        }
                    }
                    // 這時候max就是所有vi前面的vj中，dj+tj的最大值
                    int ti = max;                                       // 把max作為vi要排的時間
                    doSchedule(asapSchedule, vi, ti);                   // schedule vi at ti
                }

            }
        }
    }
}
```

alap

```
void alap(vector<struct ScheduledNode> *alapSchedule, int latency ,vector<struct Node> *nodes) {
    // 把type為o的node都排入schedule

    for (int i=0; i<nodes->size(); i++) {
        if ((*nodes)[i].type == "o") {
            struct Node *vi = &(*nodes)[i];
            doSchedule(alapSchedule, vi, latency + 1);
        }
    }

    while (alapSchedule->size() < nodes->size()) {                      // 還有nodes沒排進schedule
        for (int i=0; i<nodes->size(); i++) {
            struct Node *vi = &(*nodes)[i];

            if (isScheduled(alapSchedule, vi) == false) {               // vi還沒被排進schedule的話
                                                                        // 判斷vi後面的node是否都排進schedule了
                bool allVjAreScheduled = true;                          // 先假設vi後面全部的vj都排進去了
                for (int j=0; j<vi->to.size(); j++) {
                    struct Node *vj = vi->to[j];
                    if (isScheduled(alapSchedule, vj) == false) {       // 只要有一個vj其實還沒排進去
                        allVjAreScheduled = false;                      // 就代表這個假設是錯的
                        break;                                          // 就可以不用再檢查下去了
                    }
                }
                // 如果都排進去了，才去算vi應該被排到的時間
                if (allVjAreScheduled == true) {
                    int min = latency + 2;
                    for (int j=0; j<vi->to.size(); j++) {               // 看vi後面得所有vj
                        struct Node *vj = vi->to[j];
                        int tj = getScheduledTime(alapSchedule, vj);
                        if (tj - cost(vi) < min) {                      // 如果這個vj的tj-di > min
                            min = tj - cost(vi);                        // 就把這個tj-di設為新的,om
                        }
                    }
                    // 這時候max就是所有vi前面的vj中，dj+tj的最大值
                    int ti = min;                                       // 把min作為vi要排的時間
                    doSchedule(alapSchedule, vi, ti);                   // schedule vi at ti
                }

            }
        }
    }
}
```

```
struct NodeTimeFrame {
    struct Node* node;
    int left;
    int right;  // inclusive
};

int findScheduledTimeByNode(vector<struct ScheduledNode> *schedule, struct Node *node) {
    for (int j=0; j<schedule->size(); j++) {
        if ((*schedule)[j].node == node) {
            return (*schedule)[j].scheduledTime;
        }
    }
    return -1;
}

void computeTimeFrame(vector<struct ScheduledNode> *asapSchedule,
                      vector<struct ScheduledNode> *alapSchedule,
                      vector<struct NodeTimeFrame> *timeFrames) {
    for (int i=0; i<asapSchedule->size(); i++) {
        struct NodeTimeFrame frame;
        frame.node = (*asapSchedule)[i].node;
        frame.left = (*asapSchedule)[i].scheduledTime;
        frame.right = findScheduledTimeByNode(alapSchedule, frame.node);
        timeFrames->push_back(frame);
    }
}
```

```
void timeFrameToSchedule(vector<struct NodeTimeFrame> *timeFrames,
                         vector<struct ScheduledNode> *schedule) {
    for (int i=0; i<timeFrames->size(); i++) {
        struct NodeTimeFrame frame = (*timeFrames)[i];
        struct ScheduledNode scheduledNode;
        scheduledNode.node = frame.node;
        scheduledNode.scheduledTime = frame.left;
        schedule->push_back(scheduledNode);
    }
}

bool schedulingIsFinished(vector<struct NodeTimeFrame> *timeFrames) {
    for (int i=0; i<timeFrames->size(); i++) {
        struct NodeTimeFrame* vi = &(*timeFrames)[i];
        if (vi->left != vi->right) return false;
    }
    return true;
}
```

```
void scheduleOperation(vector<struct NodeTimeFrame> *oldTimeFrames,
                       struct Node* node, int time,
                       vector<struct NodeTimeFrame> *newTimeFrames) {
    for (int i=0; i<oldTimeFrames->size(); i++) {
        struct NodeTimeFrame* oldVi = &(*oldTimeFrames)[i];
        struct NodeTimeFrame newVi;
        newVi.node = node;
        if (oldVi->node == node) {
            newVi.left = time;
            newVi.right = time;
        } else {
            newVi.left = oldVi->left;
            newVi.right = oldVi->right;
        }
        newTimeFrames->push_back(newVi);
    }
}

// double totalForce(vector<struct NodeTimeFrame> *timeFrames) {

// }

bool isGreaterThan(double a, double b) {
    int scale = 1000;
    int newA = (int) (a * scale);
    int newB = (int) (b * scale);
    return (newA > newB);
}
```

This is my alg pseudo code

- Algorithm (Paulin)

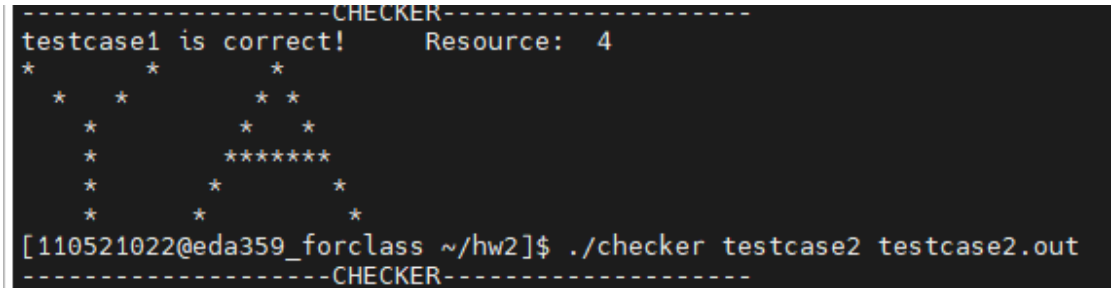
```
Call ASAP (V);
Call ALAP (V);
while there exists o_i such that E_i ≠ L_i do
   MaxGain = −∞;
   /* Try scheduling all unscheduled operations to every */
   /* state in its range */
   for each o_i, E_i ≠ L_i do
      for each j, E_i ≤ j ≤ L_i do
         S_work = SCHEDULE_OP(S_current, o_i, s_j);
         ADJUST_DISTRIBUTION(S_work, o_i, s_j);
         if COST(S_current) - COST(S_work) > MaxGain then
            MaxGain = COST(S_current) - COST(S_work);
            BestOp = o_i; BestStep = s_j;
         endif
      endfor
   endfor
   S_current = SCHEDULE_OP(S_current, BestOp, BestStep);
   ADJUST_DISTRIBUTION(S_current, BestOp, BestStep);
endwhile
```

This is my fored directed alg architecture ◦

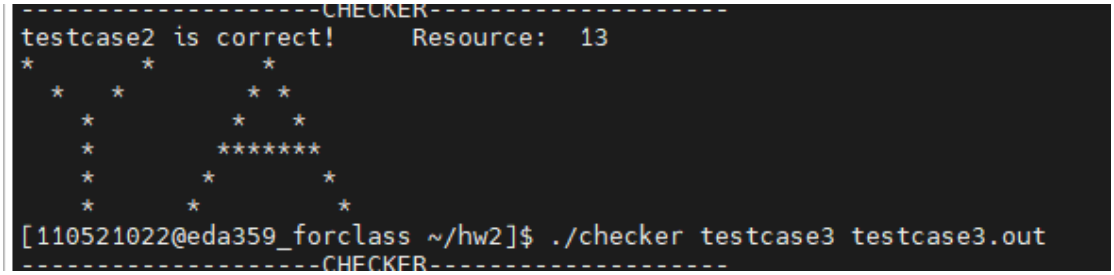
```
void forceDirectedScheduling(vector<struct ScheduledNode> *asapSchedule,
                vector<struct ScheduledNode> *alapSchedule,
                int latency,
                vector<struct ScheduledNode> *forceDirectedSchedule) {

    vector<struct NodeTimeFrame> timeFrames;
    computeTimeFrame(asapSchedule, alapSchedule, &timeFrames);
    while (schedulingIsFinished(&timeFrames) == true) {
        double maxGain = -4147483648;    // 檢查一下double的下限是不是這個值
        struct Node *bestOperation;
        int bestStep;
        for (int i=0; i<timeFrames.size(); i++) {
            struct NodeTimeFrame *vi = &(timeFrames[i]);
            if (vi->left == vi->right) continue;
            for (int j=vi->left; j<=vi->right; j++) {
                vector<struct NodeTimeFrame> workingFrames;
                scheduleOperation(&timeFrames, vi->node, j, &workingFrames);
                double gain = totalForce(&timeFrames) - totalForce(&workingFrames);
                if (isGreaterThan(gain, maxGain)) {
                    maxGain = gain;
                    bestOperation = vi->node;
                    bestStep = j;
                }
            }
        }
        vector<struct NodeTimeFrame> newFrames;
        scheduleOperation(&timeFrames, bestOperation, bestStep, &newFrames);
        timeFrames = newFrames;
    }
```

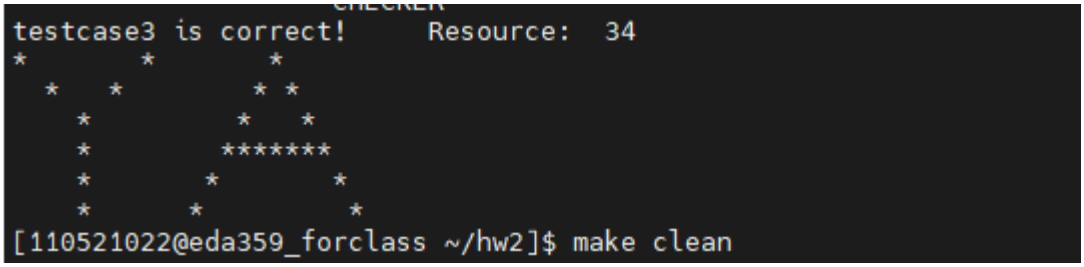
Testcase1



Testcase2



Testcase3



(3) The hardness of this assignment and how you overcome it

I feel that there is no enough time to complete this whole project.

The hardest part is probably ,I think how to find his grand and children when going to a node

Because of this, I took another course called pointer.

(4) Any suggestions about this programming assignment?

老師這學期能少做一次作業嗎?

這次作業做完還有 final project 還要做。做這次作業好幾天都沒有睡個好覺了，想到期末有 final project 要做又有 pa4 要做還要準備期末考，三個東西一起來真的會受不了。真的會沒時間，這樣沒有一的東西會做得好。懇求老師的法外開恩。

(助教 INDEX 有話想對老師說)