CS 700-34 **Scheduling**

Student: Shuhua Song

Due Date:
Soft Copy: 04/17/2020
Hard Copy: 04/17/2020

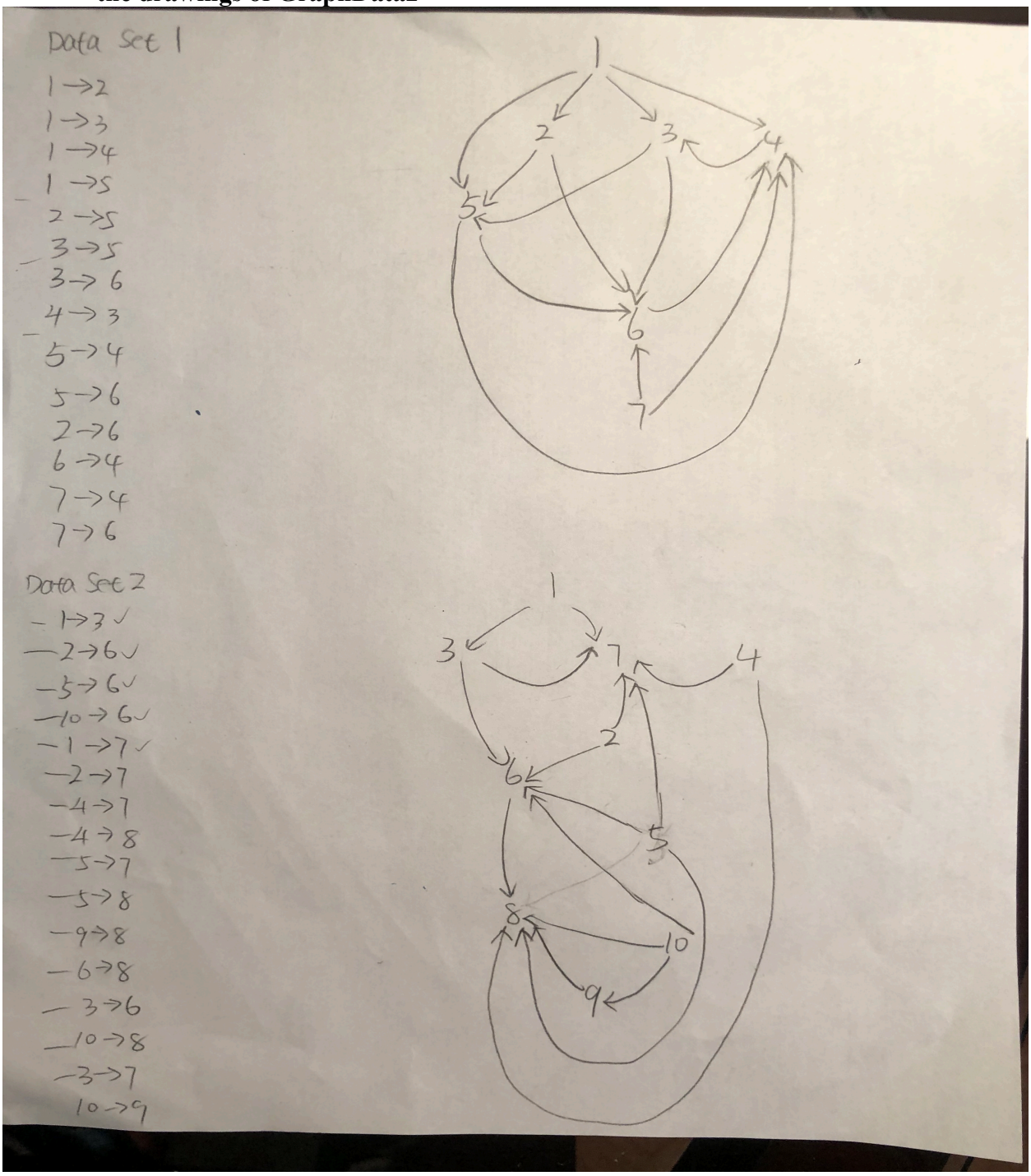
Submission Data:
Soft Copy: 04/13/2020
Hard Copy: 04/13/2020

Algorithm Steps:

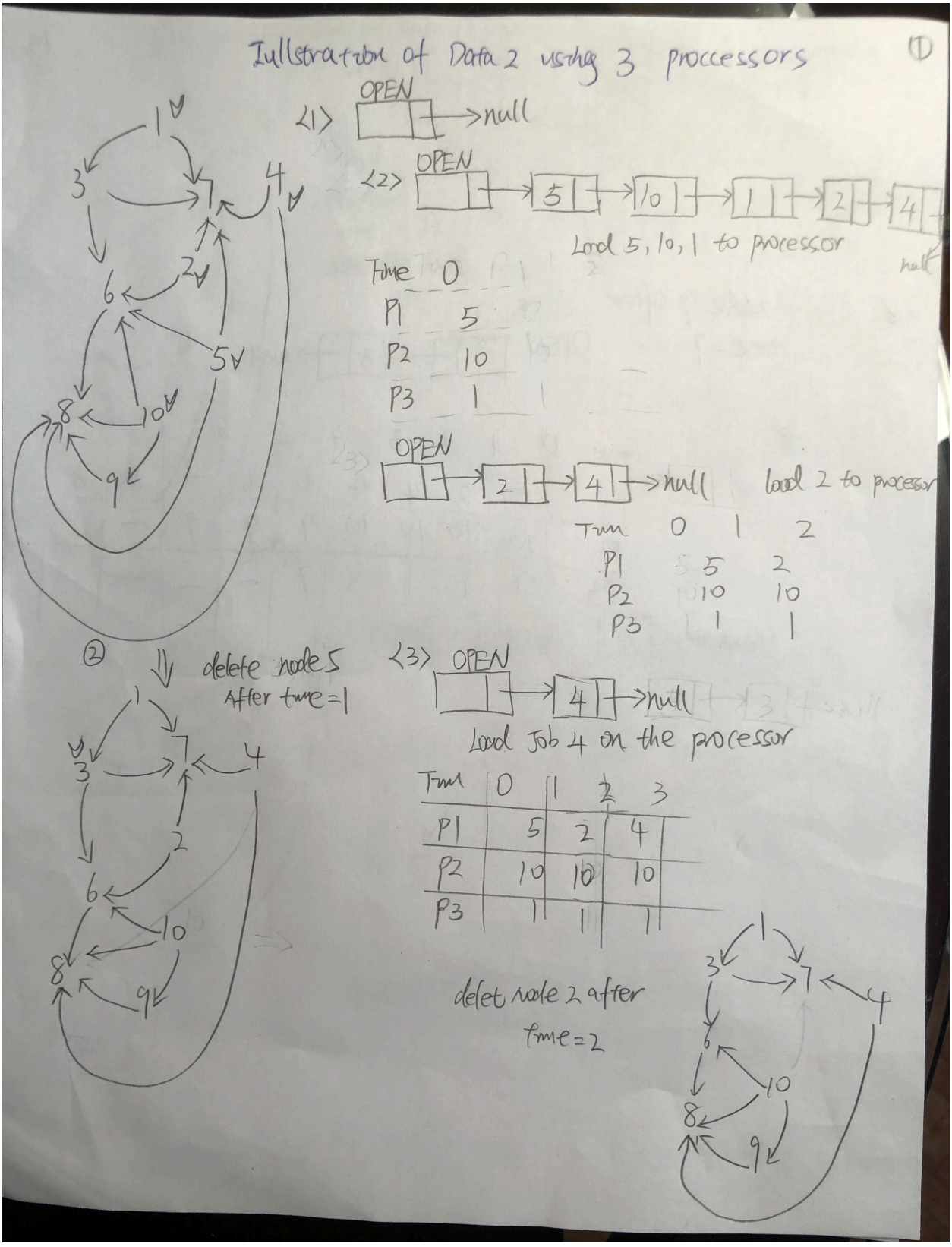
I. main(inFile, pointSet)

Step 0: inFile1, inFile1, outFile1, outFile2 ←open
Step 1: initialization (…) // see algorithm below.
Step 2: loadOpen(…) // see algorithm below.
Step 3: printList(Open, outFile2) // debug print
Step 4 loadProcAry(…) // see algorithm below.
Step 5: hasCycle ← checkCycle (…) // on your own, see the description in the above.
if hasCycle == true
output error message to console: "there is cycle in the graph!!!"
and exit the program
step 6: printScheduleTable (outFile1) // print intermediate schedule table to outFile1
step 7: currentTime++
step 8: updateProcTime (…) // on your own, see the description in the above.
step 9: deleteFinishedNodes (…)
step 10: repeat step 2 to step 11 until graphIsEmpty (…)
step 11: printScheduleTable (outFile1) // The final schedule table to outFile1
step 12: close all files

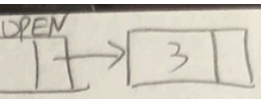
- the drawings of GraphData1
- the drawings of GraphData2

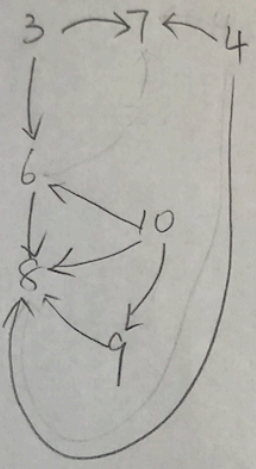


- illustrations of GraphData2 using 3 processors, if you draw it for +1 pt



(4) delete node 1 after time=3

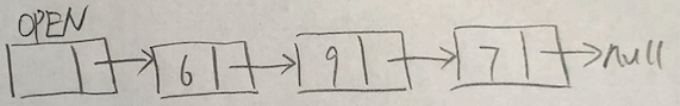




Time	0	1	2	3	4
P1	5	2	4	4	
P2	10	10	10	10	
P3	1	1	1	3	

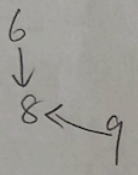
load Job 3 on processor P3

(5) delete node 4, 10, 3 after time=4



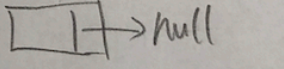
Load Job 6, 9, 7 on processor P_1, P_2, P_3

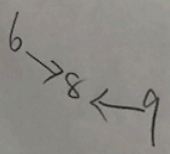
7



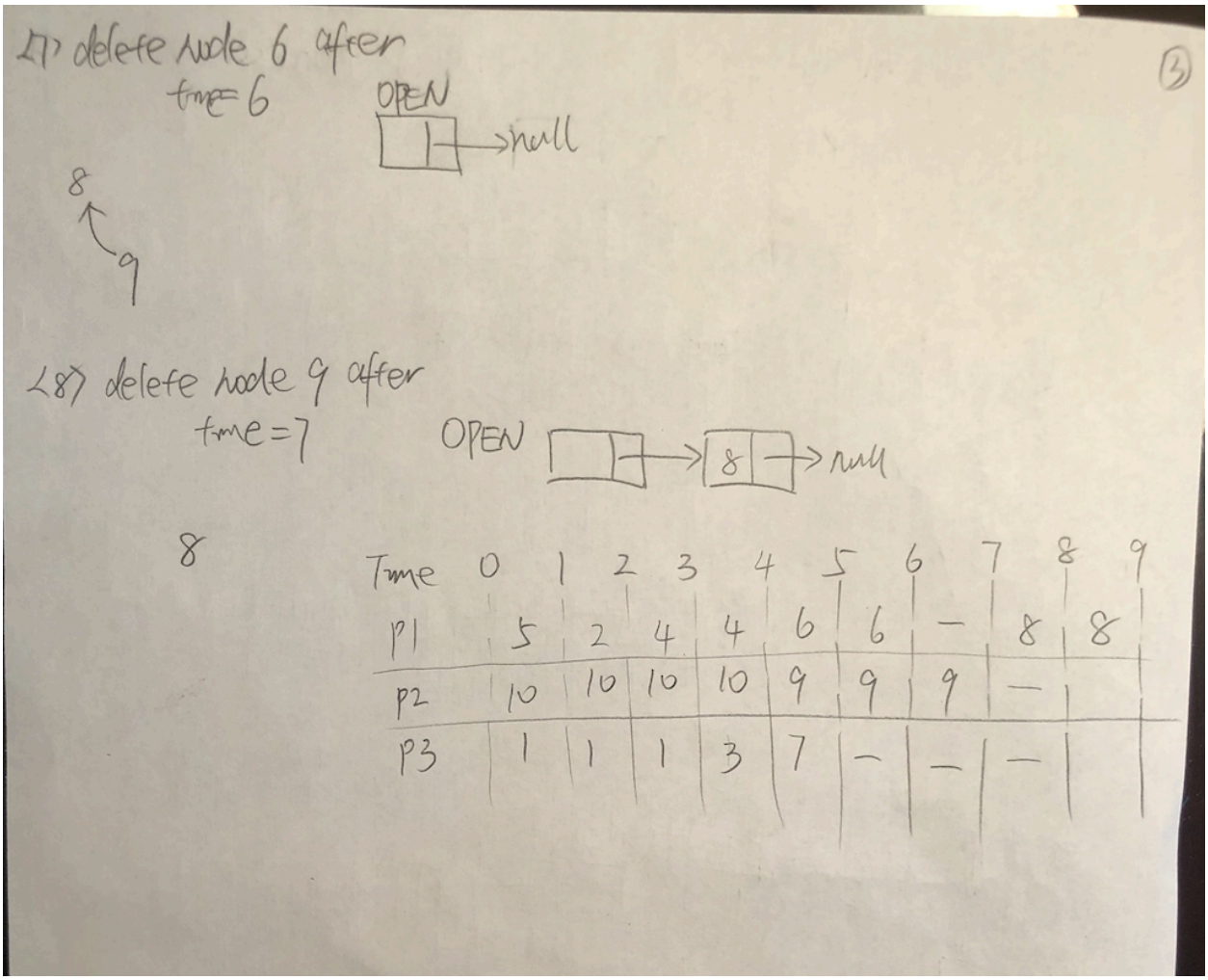
Time	0	1	2	3	4	5
P1	5	2	4	4	6	
P2	10	10	10	10	9	
P3	1	1	1	3	7	

(6) delet node 7 after time=5





Time	0	1	2	3	4	5	6	7
P1	5	2	4	4	6	6	–	
P2	10	10	10	10	9	9	9	
P3	1	1	1	3	7	–	–	



source code

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

//int  procUsed = 0;
class Scheduling {
public:
    class Node {
    public:
        Node(){
           jobId = 0;
           jobTime = 0;
           dependentCount = 0;
           next = nullptr;
        }
```

```
        Node(int jobId, int jobTime, int dependentCount) {
            this->jobId = jobId;
            this->jobTime = jobTime;
            this->dependentCount = dependentCount;
        }
        void printNode(ofstream& outFile){
            outFile <<"->";
            outFile <<"(" << this->jobId << "," << this->jobTime << "," <<
dependentCount << ","
                    << this->next->jobId << ")-> ";
        }


        Node* next;
        int dependentCount;
        int jobId;
        int jobTime;
    };

    class Jobs {
    public:
        int jobTime;
        int onWhichProc;
        int onOpen;
        int parentCount;
        int dependentCount;

        Jobs(){
            jobTime = 0; // processing time required for the job, provided in inFile2.
            onWhichProc = -1; // initialized to 0; means not on any processor
            onOpen = 0;// initialized to 0; means not on OPEN linkedlist
            parentCount = 0;
            dependentCount = 0;
        }
        Jobs(int jobTime, int onWhichProc, int onOpen, int parentCount, int
dependentCount){
            this->jobTime = jobTime;
            this->onWhichProc = onWhichProc;
            this->onOpen = onOpen;
            this->parentCount = parentCount;
            this->dependentCount = dependentCount;
        }

    };

    class Proc {
    public:
        int doWhichJob; //which job it is processing, initialize -1, meaning -
available
        int timeRemain; //time remain on a job; <= 0 means it is available

        Proc(){
            doWhichJob = -1;
            timeRemain = 0;
        }
        Proc(int doWhichJob, int timeRemain){
            this->doWhichJob = doWhichJob;
            this->timeRemain = timeRemain;
        }
    };
```

```
    int numNodes;// the total number of nodes in the input graph.
    int numProcs; //the number of processors is avaiable be used.
    int procUsed;// number of processors are used so far; initialized to 0.
    int totalJobTimes; //
    Jobs* jobAry; //a 1-D struct JOBS array of size of numNodes +1,
                  //the array index served as job id, to be dynamically allocated.
    Proc* procAry;//1-D Proc array of size of numProcs +1,the array index
                  //serve as Proc id, to be dynamically allocated
    Node* OPEN; // Nodes in OPEN are sorted in descending order by # of dependents of
orphan nodes
    int** adjMatrix; // representing the input dependency graph, initialized to zero
                     // adjMatrix[i][j] == 1, means node i is a parent of j, or, node
j is a dependent of node i.
    int* parentCountAry; // 1-D integer array to hold nodes' parent counts
    int* dependentCountAry; // 1-D integer array to hold nodes' dependent counts,
    int* onGraphAry; // 1-D integer array to indicate whether a node has been remove
from the graph, initialized to 1
                     // onGraphAry[i] == 1 means node i is on the graph, 0 means it
had been deleted

    int** scheduleTable; //rows are processors' ID and columns are the time slices,
initialized to 0

    Scheduling(int numNodes, int numProcs, int procUsed, int totalJobTime){

        this->numNodes = numNodes;
        this->numProcs = numProcs;
        this->procUsed = procUsed;
        this->totalJobTimes = totalJobTime;
        //Node* OPEN = new Node(-1,-1,-1);


//        if(numProcs > numNodes){ //???
//            numProcs = numNodes;
//        }
        adjMatrix = new int*[numNodes+1];//representing the input dependency graph
        for(int i=0; i<numNodes+1; i++){
            adjMatrix[i] = new int[numNodes+1]{0};
        }
        jobAry = new Jobs[numNodes+1];
        for(int i=0; i<=numNodes; i++){
            Jobs *job = new Jobs();  //It is Correct???
            jobAry[i] = *job;
        }

        procAry = new Proc[numProcs+1];
        for(int i=0; i<=numProcs; i++){
            Proc *newProc = new Proc();
            procAry[i] = *newProc;
        }
        parentCountAry = new int[numNodes+1]; //1-D integer array to hold nodes'
parent counts
        dependentCountAry = new int[numNodes+1];//1-D integer array to hold nodes'
dependent counts
        onGraphAry = new int[numNodes+1]{0};//1-D integer array to indicate whether a
node has been remove from the graph
        //onGraphAry[i] == 1 means node i is on the graph, 0 means it had been deleted

    }
```

```
    //load inFile1's date to adjMatrix
    void loadMatrix(ifstream& inFile, int* onGraphAry){
        int x=0;
        int y=0;
        while(!inFile.eof()){
            inFile >> x;
            inFile >> y;
            adjMatrix[x][y] = 1;
             onGraphAry[x] = 1;   //         ??????? Does it need update
             onGraphAry[y] = 1;
           // cout << "x: " << x << " y: " << y << endl;
        }
      //  cout << "Matrix: " << endl;
//         for(int i=0; i<=numNodes; i++){     //<<<<<<<<<<
//             for(int j=0; j<=numNodes; j++){
//                if( adjMatrix[i][j] != 1)
//                    cout << "." ;
//                else
//                    cout << adjMatrix[i][j];
//             }
//             cout << endl;
//         }

//         cout << "OnGraph: " << endl;
//         for(int i=0; i<=numNodes; i++){
//             cout << onGraphAry[i] << endl;
//         }
    }
    //Compute totalJobtime
    int constructJobAry(ifstream& inFile, int** adjMatrix){ //it returns the
totalJobTIme
       int totalTime = 0;
        int totalNodes = 0;
        inFile >> totalNodes;
       while(!inFile.eof()){
           int NodeID ;
           int jobOfTime;
           inFile >> NodeID;
           inFile >> jobOfTime;
           //cout << NodeID << " " << jobOfTime << endl; //>>>>>>>>>>
           totalTime += jobOfTime;
           jobAry[NodeID].jobTime = jobOfTime;
           jobAry[NodeID].onWhichProc = -1;
           jobAry[NodeID].onOpen = 0;
           jobAry[NodeID].parentCount = parentCountAry[NodeID];
           jobAry[NodeID].dependentCount = dependentCountAry[NodeID];
           //cout << NodeID << " " <<  jobAry[NodeID].jobTime << endl;
       }
 //        cout << "ConstructJobAry NodeID: " << "jobTime" << endl;
        for(int i=0; i<=numNodes; i++){      //<<<<<<<<<<<<
            cout << i << " " <<  jobAry[i].jobTime << endl;
        }
       return totalTime;
    }
    void computeParentCount(int** adjMatrix, int* parentCountAry){
        for(int j=1; j<=numNodes; j++){ // j = nodeId
            int sumParent = 0;
            for(int i=1; i<=numNodes; i++){
                sumParent += adjMatrix[i][j];
            }
```

```
            parentCountAry[j]=sumParent;
        }
//        cout << "Parent Count: " << endl;  //<<<<<<<<<<
//        cout << "NodeID: " << "numParen: " << endl;
        for(int i=0; i<=numNodes; i++){
//            cout << "parentCountAry[" <<  i << "] =  " << parentCountAry[i] << endl;
        }
    }
    void computeDependentCount(int** adjMatrix, int* dependentCountAry){
        for(int i=1; i<=numNodes; i++){ //i = nodeId
            int sumDepen = 0;
            for(int j=1; j<=numNodes; j++){
                sumDepen += adjMatrix[i][j];
            }
            dependentCountAry[i]=sumDepen;
        }
//        cout << "Dependent Count: " << endl;  //<<<<<<<<<<
//        cout << "NodeID: " << "numDepen: " << endl;
        for(int i=0; i<=numNodes; i++){
            cout << i << " " << dependentCountAry[i] << endl;
        }
    }

    void initialization(ifstream& inFile1, ifstream&  inFile2){

        loadMatrix(inFile1, onGraphAry);
        computeParentCount(adjMatrix, parentCountAry);
        computeDependentCount(adjMatrix, dependentCountAry);
        totalJobTimes = constructJobAry(inFile2,adjMatrix);

        scheduleTable = new int*[numProcs+1];
        for(int j=0; j<numProcs+1; j++){
            scheduleTable[j] = new int[totalJobTimes+1]{0};
        }
    }

    int findOrphan(){ //How to mark selected Orhpan???
        for(int i=1; i<=numNodes; i++){
            if(jobAry[i].parentCount<=0 && jobAry[i].onOpen==0 &&
jobAry[i].onWhichProc==-1){
                jobAry[i].onOpen = 1;
                return i;
            }
        }
        return -1; //there is none orphan can be found
    }
    void listInsert(Node* OPEN, Node* newNode){
        Node *spot = findSpot(OPEN, newNode);
        if(spot!=nullptr){
            newNode->next = spot->next;
            spot->next = newNode;
        }
    }
    Node* findSpot(Node* OPEN, Node* newNode){
        //cout << "OPEN11: " << OPEN->jobId << endl;
        Node* spot = OPEN;
        while(spot->next != nullptr && spot->next->dependentCount >=
newNode->dependentCount){
            spot=spot->next;
           // continue;
        }
```

```
        //cout << "JOBID: " << spot->jobId;
        return spot;
    }

    Node* removeFront(Node* OPEN){
        Node* front = OPEN->next;
        OPEN->next = OPEN->next->next;
        return front;
    }

    void printList(Node* node, ofstream& outFile){
        outFile << "OPEN LinkedList: " << endl;
        while(node->next != nullptr){
            node->printNode(outFile);
            node = node->next;
        }
        outFile << "->( "<< node->jobId << "," << node->jobTime << "," <<
node->dependentCount << "," << "NULL" << ")";
        outFile << "->NULL\n";
    }

    void loadOpen(Node* OPEN){ //find orphan nodes and put them on Open.
        int orphanNode = findOrphan();
        while(orphanNode > 0){
            if(orphanNode > 0){
                cout << "orphanNodeID: " << orphanNode << endl;  // <<<<<<<<<<<<<<<
                int jobid = orphanNode;
                int jobtime = jobAry[jobid].jobTime;
                Node* newNode = new Node(jobid, jobtime, dependentCountAry[jobid]);
                cout << "jobid: " << newNode->jobId << endl;
                newNode->next = nullptr;
                listInsert(OPEN, newNode);
                //jobAry[jobid].onOpen = 1;
            }
            orphanNode = findOrphan();
        }
    }

    void printScheduleTable(ofstream& outFile, int currentTime){
        outFile << "--------";
        for(int i=0; i<=currentTime; i++){
            outFile <<  i << "---" ;
        }
        outFile << endl;
        for(int i=1; i<= numProcs; i++){
            outFile<<"P(" << i << ") " ;
           for(int j=0 ;j<= currentTime; j++){
                if(scheduleTable[i][j]>0){
                    outFile << "|" <<  scheduleTable[i][j];
                    outFile << "  ";
                }
                else
                    outFile <<  "|"<< "-- ";
           }
           outFile << endl;
           outFile << "     ";
           outFile <<"-----------------------------------" << endl;
        }
        outFile << endl;
    }
```

```
    int findProcessor(){
        for(int i=1; i<=numProcs; i++){
            if(procAry[i].timeRemain <= 0){
                return i;
            }
        }
        return -1;
    }

    //find available processor to process nodes on the OPEN
    void loadProcAry(Node* OPEN, int currentTime){
       // cout << " Test LoadProcAry: " << endl;
        int availProc = findProcessor(); //check availiable processor
        while(availProc > 0 && OPEN->next!=nullptr && procUsed < numProcs){
            if(availProc > 0){//there is a processor availilable
                procUsed++;
                Node* newJob = OPEN->next;
                OPEN->next = OPEN->next->next;
                int jobid = newJob->jobId;
                int jobtime = newJob->jobTime;
                procAry[availProc].doWhichJob = jobid;
                procAry[availProc].timeRemain = jobtime;
                putJobOnTable(availProc, currentTime, jobid, jobtime);
            }
             availProc = findProcessor();
        }
    }

    void putJobOnTable(int availProc, int currentTime, int jobId, int jobTime) {
        int time = currentTime;
        int endTime = time + jobTime;
        cout << endl;
        cout << "availProc:" << " " << "time:" << " " << "jobId:" << endl;
        while(time < endTime){
            scheduleTable[availProc][time]=jobId;
            cout << availProc << " " << time << " " << jobId;
            time++;
        }
    }
    bool checkCycle(Node* OPEN){

        if(OPEN->next==nullptr && !graphIsEmpty() && checkProsAlldone()){
            return true;
        }
        return false;
    }

    bool checkProsAlldone(){
        for(int i=1; i<=numProcs; i++){   //???? is checking procAry or checing
scheduleTable
            if(procAry[i].timeRemain > 0 && procAry[i].doWhichJob > 0){
                return false;
            }
        }
        return true;
    }
    bool graphIsEmpty(){
       // cout << "Test GraphIsEmpty" << endl;
        for(int i=1; i<=numNodes; i++){
           // cout << "onGraphAry: " << onGraphAry[i] << endl;
            if(onGraphAry[i] == 1){
```

```
                return false;
            }
        }
        return true;
    }

    //decrease procAry[i].timeRemain by 1 for all used processors
    void updateProcTime(){
        for(int i=1; i<=numProcs; i++){
            if(procAry[i].timeRemain > 0 && procAry[i].doWhichJob >0){
                procAry[i].timeRemain--;
            }
        }
    }
    int findDoneProc(){
        int jobid = 0;
        for(int i=1; i<=numProcs; i++){
            if(procAry[i].doWhichJob > 0 && procAry[i].timeRemain <=0){
                jobid = procAry[i].doWhichJob;
                procAry[i].doWhichJob = -1; //ProcAry[i] now is not busy
                procUsed--;
                return jobid;
                //onGraphAry[jobid] = 0;
            }
        }
        return -1;
    }


    void deleteFinishedNodes(){
        int jobid = findDoneProc();
        while(jobid > 0){
            //cout << "delete jobID: " << jobid << endl;
            if(jobid > 0){
                onGraphAry[jobid] = 0;
                deleteEdge(jobid);
            }
            jobid = findDoneProc();
        }
    }

    void deleteEdge(int jobId) {
      for(int i=1;i<=numNodes; i++){
          if(adjMatrix[jobId][i] ==1 ){
              adjMatrix[jobId][i] = 0;
              parentCountAry[i]--;
              jobAry[i].parentCount--;
              //cout << "ParentCount[" << i << "]= " << jobAry[i].parentCount << endl;
          }
       }
    }

};


int main(int argc, char *argv[]) {
    ifstream inFile1(argv[1]);
    ifstream inFile2(argv[2]);

    int numProc = stoi(argv[3]);
```

```
    ofstream outFile1(argv[4]);
    ofstream outFile2(argv[5]);

    int currentTime = 0;
    bool isCycle = false;
    int procUsed = 0;
    int totalTime = 0;

    int numNodes;
    inFile1 >> numNodes;



    Scheduling::Node* OPEN = new Scheduling::Node();

    if(numProc > numNodes){ //???
        numProc = numNodes;
    }

    Scheduling *scheduling = new Scheduling(numNodes, numProc, procUsed,totalTime);

    scheduling->initialization(inFile1, inFile2);

    while(!scheduling->graphIsEmpty()){
          scheduling->loadOpen(OPEN);
          scheduling->printList(OPEN, outFile2);
          scheduling->loadProcAry(OPEN, currentTime);
          isCycle = scheduling->checkCycle(OPEN);
        if(isCycle){
            outFile1 << " There is a cycle in the graph!!!" << endl;
            exit(0);
        }

         scheduling->printScheduleTable(outFile1, currentTime);
         currentTime++;

        scheduling->updateProcTime();
        scheduling->deleteFinishedNodes();
    }
  //  scheduling->printScheduleTable(outFile1);

    inFile1.close();
    inFile2.close();
    outFile1.close();
    outFile2.close();

    return 0;
}
```

- outFile1 // result of data1 using 3 processors

-------0---
P(1) |1

P(2) |7

P(3) |--

-------0---1---
P(1) |1 |2

P(2) |7 |--

P(3) |--|--

There is a cycle in the graph!!!

- outFile2 // result of data1 using 3 processors

OPEN LinkedList:
->(0,0,0,1)-> ->(1,1,4,7)-> ->(7,1,2,NULL)->NULL
OPEN LinkedList:
->(0,0,0,2)-> ->(2,1,2,NULL)->NULL
OPEN LinkedList:

- outFile1 // result of data2 using 2 processors

-------0---
P(1) |5

P(2) |10

-------0---1---
P(1) |5 |1

P(2) |10 |10

-------0---1---2---
P(1) |5 |1 |1

```
P(2) |10  |10  |10
     ---------------------------------

-------0---1---2---3---
P(1) |5  |1  |1  |1
     ---------------------------------
P(2) |10  |10  |10  |10
     ---------------------------------

-------0---1---2---3---4---
P(1) |5  |1  |1  |1  |2
     ---------------------------------
P(2) |10  |10  |10  |10  |4
     ---------------------------------

-------0---1---2---3---4---5---
P(1) |5  |1  |1  |1  |2  |3
     ---------------------------------
P(2) |10  |10  |10  |10  |4  |4
     ---------------------------------

-------0---1---2---3---4---5---6---
P(1) |5  |1  |1  |1  |2  |3  |9
     ---------------------------------
P(2) |10  |10  |10  |10  |4  |4  |6
     ---------------------------------

-------0---1---2---3---4---5---6---7---
P(1) |5  |1  |1  |1  |2  |3  |9  |9
     ---------------------------------
P(2) |10  |10  |10  |10  |4  |4  |6  |6
     ---------------------------------

-------0---1---2---3---4---5---6---7---8---
P(1) |5  |1  |1  |1  |2  |3  |9  |9  |9
     ---------------------------------
P(2) |10  |10  |10  |10  |4  |4  |6  |6  |7
     ---------------------------------

-------0---1---2---3---4---5---6---7---8---9---
P(1) |5  |1  |1  |1  |2  |3  |9  |9  |9  |8
     ---------------------------------
P(2) |10  |10  |10  |10  |4  |4  |6  |6  |7  |--
     ---------------------------------

-------0---1---2---3---4---5---6---7---8---9---10---
```

P(1) |5 |1 |1 |1 |2 |3 |9 |9 |9 |8 |8

P(2) |10 |10 |10 |10 |4 |4 |6 |6 |7 |--|--

- outFile2 // result of data2 using 2 processors

OPEN LinkedList:
->(0,0,0,5)-> ->(5,1,3,10)-> ->(10,4,3,1)-> ->(1,3,2,2)-> ->(2,1,2,4)->
->(4,2,2,NULL)->NULL
OPEN LinkedList:
->(0,0,0,1)-> ->(1,3,2,2)-> ->(2,1,2,4)-> ->(4,2,2,NULL)->NULL
OPEN LinkedList:
->(0,0,0,2)-> ->(2,1,2,4)-> ->(4,2,2,NULL)->NULL
OPEN LinkedList:
->(0,0,0,2)-> ->(2,1,2,4)-> ->(4,2,2,NULL)->NULL
OPEN LinkedList:
->(0,0,0,2)-> ->(2,1,2,4)-> ->(4,2,2,3)-> ->(3,1,2,9)-> ->(9,3,1,NULL)->NULL
OPEN LinkedList:
->(0,0,0,3)-> ->(3,1,2,9)-> ->(9,3,1,NULL)->NULL
OPEN LinkedList:
->(0,0,0,9)-> ->(9,3,1,6)-> ->(6,2,1,7)-> ->(7,1,0,NULL)->NULL
OPEN LinkedList:
->(0,0,0,7)-> ->(7,1,0,NULL)->NULL
OPEN LinkedList:
->(0,0,0,7)-> ->(7,1,0,NULL)->NULL
OPEN LinkedList:
->(0,0,0,8)-> ->(8,2,0,NULL)->NULL
OPEN LinkedList:
->(0,0,0,NULL)->NULL

- outFile1 // result of data2 using 3 processors

-------0---
P(1) |5

P(2) |10

P(3) |1

-------0---1---
P(1) |5 |2

P(2) |10 |10

```
    ---------------------------------
P(3) |1  |1
    ---------------------------------

-------0---1---2---
P(1) |5  |2  |4
    ---------------------------------
P(2) |10  |10  |10
    ---------------------------------
P(3) |1  |1  |1
    ---------------------------------

-------0---1---2---3---
P(1) |5  |2  |4  |4
    ---------------------------------
P(2) |10  |10  |10  |10
    ---------------------------------
P(3) |1  |1  |1  |3
    ---------------------------------

-------0---1---2---3---4---
P(1) |5  |2  |4  |4  |6
    ---------------------------------
P(2) |10  |10  |10  |10  |9
    ---------------------------------
P(3) |1  |1  |1  |3  |7
    ---------------------------------

-------0---1---2---3---4---5---
P(1) |5  |2  |4  |4  |6  |6
    ---------------------------------
P(2) |10  |10  |10  |10  |9  |9
    ---------------------------------
P(3) |1  |1  |1  |3  |7  |--
    ---------------------------------

-------0---1---2---3---4---5---6---
P(1) |5  |2  |4  |4  |6  |6  |--
    ---------------------------------
P(2) |10  |10  |10  |10  |9  |9  |9
    ---------------------------------
P(3) |1  |1  |1  |3  |7  |--|--
    ---------------------------------

-------0---1---2---3---4---5---6---7---
P(1) |5  |2  |4  |4  |6  |6  |--|8
```

```
    ---------------------------------
P(2) |10  |10  |10  |10  |9  |9  |9  |--
    ---------------------------------
P(3) |1  |1  |1  |3  |7  |--|--|--
    ---------------------------------

-------0---1---2---3---4---5---6---7---8---
P(1) |5  |2  |4  |4  |6  |6  |--|8  |8
    ---------------------------------
P(2) |10  |10  |10  |10  |9  |9  |9  |--|--
    ---------------------------------
P(3) |1  |1  |1  |3  |7  |--|--|--|--
    ---------------------------------
```

- outFile2 // result of data2 using 3 processors

```
OPEN LinkedList:
->(0,0,0,5)-> ->(5,1,3,10)-> ->(10,4,3,1)-> ->(1,3,2,2)-> ->(2,1,2,4)->
->( 4,2,2,NULL)->NULL
OPEN LinkedList:
->(0,0,0,2)-> ->(2,1,2,4)-> ->( 4,2,2,NULL)->NULL
OPEN LinkedList:
->(0,0,0,4)-> ->( 4,2,2,NULL)->NULL
OPEN LinkedList:
->(0,0,0,3)-> ->( 3,1,2,NULL)->NULL
OPEN LinkedList:
->(0,0,0,6)-> ->(6,2,1,9)-> ->(9,3,1,7)-> ->( 7,1,0,NULL)->NULL
OPEN LinkedList:
->( 0,0,0,NULL)->NULL
OPEN LinkedList:
->( 0,0,0,NULL)->NULL
OPEN LinkedList:
->(0,0,0,8)-> ->( 8,2,0,NULL)->NULL
OPEN LinkedList:
->( 0,0,0,NULL)->NULL
```

- outFile1 // result of data2 using numNodes + 3 processors

```
-------0---
P(1) |5
    ---------------------------------
P(2) |10
    ---------------------------------
P(3) |1
    ---------------------------------
```

P(4) |2

P(5) |4

P(6) |--

P(7) |--

P(8) |--

P(9) |--

P(10) |--

-------0---1---
P(1) |5 |--

P(2) |10 |10

P(3) |1 |1

P(4) |2 |--

P(5) |4 |4

P(6) |-- |--

P(7) |-- |--

P(8) |-- |--

P(9) |-- |--

P(10) |-- |--

-------0---1---2---
P(1) |5 |-- |--

P(2) |10 |10 |10

P(3) |1 |1 |1

P(4) |2 |-- |--

P(5) |4 |4 |--

P(6) |-- |-- |--

P(7) |-- |-- |--

P(8) |-- |-- |--

P(9) |-- |-- |--

P(10) |-- |-- |--

-------0---1---2---3---
P(1) |5 |-- |-- |3

P(2) |10 |10 |10 |10 |10

P(3) |1 |1 |1 |--

P(4) |2 |-- |-- |--

P(5) |4 |4 |-- |--

P(6) |-- |-- |-- |--

P(7) |-- |-- |-- |--

P(8) |-- |-- |-- |--

P(9) |-- |-- |-- |--

P(10) |-- |-- |-- |--

-------0---1---2---3---4---
P(1) |5 |-- |-- |3 |6

P(2) |10 |10 |10 |10 |10 |9

P(3) |1 |1 |1 |-- |7

P(4) |2 |-- |-- |-- |--

P(5) |4 |4 |-- |-- |--

P(6) |-- |-- |-- |-- |--

P(7) |-- |-- |-- |-- |--

P(8) |-- |-- |-- |-- |--

P(9) |-- |-- |-- |-- |--

P(10) |-- |-- |-- |-- |--

-------0---1---2---3---4---5---
P(1) |5 |-- |-- |3 |6 |6

P(2) |10 |10 |10 |10 |9 |9

P(3) |1 |1 |1 |-- |7 |--

P(4) |2 |-- |-- |-- |-- |--

P(5) |4 |4 |-- |-- |-- |--

P(6) |-- |-- |-- |-- |-- |--

P(7) |-- |-- |-- |-- |-- |--

P(8) |-- |-- |-- |-- |-- |--

P(9) |-- |-- |-- |-- |-- |--

P(10) |-- |-- |-- |-- |-- |--

-------0---1---2---3---4---5---6---
P(1) |5 |-- |-- |3 |6 |6 |--

P(2) |10 |10 |10 |10 |9 |9 |9

P(3) |1 |1 |1 |-- |7 |-- |--

P(4) |2 |-- |-- |-- |-- |-- |--

P(5) |4 |4 |-- |-- |-- |-- |--

P(6) |-- |-- |-- |-- |-- |-- |--

```
P(7) |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
P(8) |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
P(9) |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
P(10) |-- |-- |-- |-- |-- |-- |--
     ---------------------------------

-------0---1---2---3---4---5---6---7---
P(1) |5  |-- |-- |3  |6  |6  |-- |8
     ---------------------------------
P(2) |10  |10  |10  |10  |9  |9  |9  |--
     ---------------------------------
P(3) |1  |1  |1  |-- |7  |-- |-- |--
     ---------------------------------
P(4) |2  |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
P(5) |4  |4  |-- |-- |-- |-- |-- |--
     ---------------------------------
P(6) |-- |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
P(7) |-- |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
P(8) |-- |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
P(9) |-- |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
P(10) |-- |-- |-- |-- |-- |-- |-- |--
     ---------------------------------

-------0---1---2---3---4---5---6---7---8---
P(1) |5  |-- |-- |3  |6  |6  |-- |8  |8
     ---------------------------------
P(2) |10  |10  |10  |10  |9  |9  |9  |-- |--
     ---------------------------------
P(3) |1  |1  |1  |-- |7  |-- |-- |-- |--
     ---------------------------------
P(4) |2  |-- |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
P(5) |4  |4  |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
P(6) |-- |-- |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
P(7) |-- |-- |-- |-- |-- |-- |-- |-- |--
     ---------------------------------
```

P(8) |-- |-- |-- |-- |-- |-- |-- |-- |--

P(9) |-- |-- |-- |-- |-- |-- |-- |-- |--

P(10) |-- |-- |-- |-- |-- |-- |-- |-- |--

- outFile2 // result of data2 using numNodes + 3 processors

OPEN LinkedList:
->(0,0,0,5)-> ->(5,1,3,10)-> ->(10,4,3,1)-> ->(1,3,2,2)-> ->(2,1,2,4)->
->(4,2,2,NULL)->NULL
OPEN LinkedList:
->(0,0,0,NULL)->NULL
OPEN LinkedList:
->(0,0,0,NULL)->NULL
OPEN LinkedList:
->(0,0,0,3)-> ->(3,1,2,NULL)->NULL
OPEN LinkedList:
->(0,0,0,6)-> ->(6,2,1,9)-> ->(9,3,1,7)-> ->(7,1,0,NULL)->NULL
OPEN LinkedList:
->(0,0,0,NULL)->NULL
OPEN LinkedList:
->(0,0,0,NULL)->NULL
OPEN LinkedList:
->(0,0,0,8)-> ->(8,2,0,NULL)->NULL
OPEN LinkedList:
->(0,0,0,NULL)->NULL