Cheewin Thawornjaroenpong

CSCI323.25 Designs and Analysis of Algorithms (Spring 2023)

Project5

Kruskal’s algorithm

03/27/2023

Algorithm Steps:

Step 0: inFile, outFile1, deBugFile open via argv[] as given in the above N inFile numSets N whichSet allocate space, size of numNodes + 1, set whichSet[i] to i, i from 1 to numNodes+1 edgeList get a dummy edge <0,0,0> mstList get a dummy edge<0,0,0> totalMSTCost 0 deBugFile "*** Printing the input graph ***"

Step 1: u, w, cost read from inFile newEdge get a new edge (u, w, cost) deBugFile "newEdge from inFile is" printEdge (newEdge)

Step 2: insertEdge (newEdge, edgeList) // insert newEdge into linked list, pointed by edgeList. deBugFile "Printing edgeList after insert the new edge:"

Step 3: printList (edgeList, deBugFile) // print edgeList after each insertion.

Step 4: repeat step 1 to step 3 while inFile is not empty deBugFile "*** At the end of printing all edges of the input graph"

Step 5: nextEdge removeEdge (edgeList) // Note: nextEdge is a edge pointer and points to //the node after dummy node and it is the min cost of all edges in edgeList.

Step 6: repeat Step 5 while whichSet [nextEdge->nU] == whichSet [nextEdge->nW] // since nU and nW cannot be in the same set, continue get nextEdge //until they are not in the same set.

Step 7: deBugFile "the nextEdge is" pintEdge (nextEdge) insertEdge (nextEdge, mstList) totalMSTCost += nextEdge->cost merge2Sets (nextEdge->nU, nextEdge->nW) numSets -- deBugFile "numSets is" print numSets here

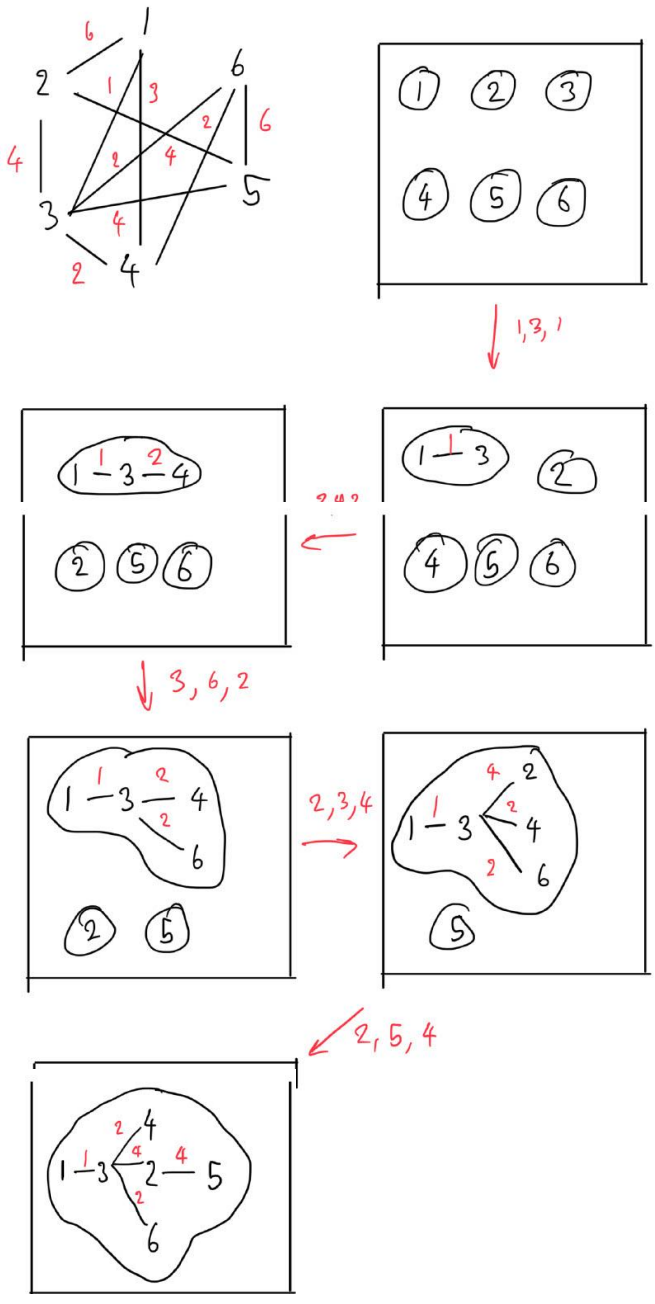
Step 8: deBugFile "Printing whichSet array" printAry(whichSet) deBugFile "Printing the remaining of edgeList" printList (edgeList, deBugFile) deBugFile "Print the growing MST list" printList (mstList, deBugFile)

Step 9: repeat step 5 – step 8 while numSets > 1

Step 10: printList (mstList, outFile1)

Step 11: close all files

Illustrations:



Source code:

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

using namespace std;

class edge
{
public:
    int nU;
    int nW;
    int cost;
    edge* next;

    edge()
    {
        this->nU = 0;
        this->nW = 0;
        this->cost = 0;
        this->next = NULL;
    }

    edge(int n1, int n2, int cost)
    {
        this->nU = n1;
        this->nW = n2;
        this->cost = cost;
        this->next = NULL;
    }

public:
    void printEdge(ofstream &File)
    {
        if (File.is_open() && this != NULL)
        {
            File << "(" << this->nU << ", " << this->nW << ", "
                << this->cost << ")" << " -> ";
        }
    }
};

class KruskalMST
{
```

```
public:
   int N;
   int* whichSet;
   int numSets;
   int totalMSTcost;
   edge* edgeList;
   edge* msList;

   KruskalMST()
   {
      this->N = 0;
      this->whichSet = NULL;
      this->numSets = 0;
      this->totalMSTcost = NULL;
      this->edgeList = NULL;
      this->msList = NULL;
   }

   KruskalMST(int N, int* whichSet, int numSets, int totalMSTcost, edge* edgeList, edge*
msList)
   {
      this->N = N;
      this->whichSet = whichSet;
      this->numSets = numSets;
      this->totalMSTcost = totalMSTcost;
      this->edgeList = edgeList;
      this->msList = msList;
   }

public:

   edge* findSpot(edge* listHead, edge* newEdge, ofstream &File)
   {
      edge* Spot = listHead;

      if (File.is_open())
      {
         File << "***In findSpot()*****" << endl;
         File << "newEdge Cost: " << newEdge->cost << " ";
         newEdge->printEdge(File);
         File << endl;
      }

      while (Spot->next != NULL)
```

```
    {
        if (newEdge->cost < Spot->next->cost)
        {
            break;
        }
        else
        {
            Spot = Spot->next;
        }

    }

    if (File.is_open())
    {
        File << "Spot Cost: " << Spot->cost << " ";
        Spot->printEdge(File);
        File << endl;
    }

    if (File.is_open())
    {
        File << "***Leaving findSpot()*****" << endl;
    }


    return Spot;
}

void insertEdgeList(edge* newEdge, ofstream &File)
{

    if (File.is_open())
    {
        File << "***In insertEdgeList()*****" << endl;
    }
    edge* Spot = this->findSpot(this->edgeList, newEdge, File);
    if (Spot != NULL)
    {
        newEdge->next = Spot->next;
        Spot->next = newEdge;
    }
    else
    {
        if (File.is_open())
```

```
        {
            File << "Spot = NULL" << endl;
        }
    }

    if (File.is_open())
    {
        File << "***Leaving insertEdgeList()*****" << endl;
    }

}

void insertMSTList(edge* newEdge, ofstream& File)
{
    if (File.is_open())
    {
        File << "***In insertMSTList()*****" << endl;
    }
    edge* Spot = this->findSpot(this->msList, newEdge, File);
    if (Spot != NULL)
    {
        newEdge->next = Spot->next;
        Spot->next = newEdge;
    }
    else
    {
        if (File.is_open())
        {
            File << "Spot = NULL" << endl;
        }
    }

    if (File.is_open())
    {
        File << "***Leaving insertMSTList()*****" << endl;
    }
}

edge* removeEdge(ofstream &File)
{
    if (File.is_open())
    {
        File << "*****In removeEdge()*****" << endl;
    }
```

```
    edge* tmp = this->edgeList->next;
    this->edgeList->next = this->edgeList->next->next;
    tmp->next = NULL;


    if (File.is_open() && tmp != NULL)
    {
        File << "tmp: " << tmp->nU << ", " << tmp->nW << ", " << tmp->cost << endl;
        if (tmp->next == NULL)
        {
            File << "tmp.next is Null" << endl;
        }
        else
        {
            File << "Something wrong in removeEdge()" << endl;
        }
    }

    if (File.is_open())
    {
        File << "*****Leaving removeEdge()*****" << endl;
    }

    return tmp;
}


void merge2Sets(int Ni, int Nj)
{
    if (this->whichSet[Ni] < this->whichSet[Nj])
    {
        this->updateWhichSet(this->whichSet[Nj], this->whichSet[Ni]);
    }
    else
    {
        this->updateWhichSet(this->whichSet[Ni], this->whichSet[Nj]);
    }
}

void updateWhichSet(int a, int b)
{
    for (int i = 1; i <= this->N; i++)
    {
        if (this->whichSet[i] == a)
```

```
        {
            this->whichSet[i] = b;
        }
    }
}

void printAry(ofstream &File)
{
    int data = 0;
    for (int i = 1; i <= N; i++)
    {
        data = this->whichSet[i];
        File << "whichSet[" << i << "]: " << data << endl;
    }

}

void printEdgeList(ofstream &File)
{
    edge* temp = this->edgeList;

    if (File.is_open())
    {
        File << "listHead -> ";
    }
    if (temp == NULL)
    {
        if (File.is_open())
        {
            File << "The list is Empty." << endl;
        }
    }
    while (temp != NULL)
    {
        File << " < " << temp->nU << ", " << temp->nW << ", " << temp->cost << " > " << " -> ";
        temp = temp->next;
    }
    if (temp == NULL)
    {
        File << "NULL" << endl;
    }

}
```

```
    void printMSTList(ofstream& File)
    {
        edge* temp = this->msList;

        if (File.is_open())
        {
            File << "listHead -> ";
        }
        if (temp == NULL)
        {
            if (File.is_open())
            {
                File << "The list is Empty." << endl;
            }
        }
        while (temp != NULL)
        {
            File << " < " << temp->nU << ", " << temp->nW << ", " << temp->cost << " > " << " -> ";
            temp = temp->next;
        }
        if (temp == NULL)
        {
            File << "NULL" << endl;
        }
    }
};

int main(int argc, char** argv)
{
    ifstream inFile(argv[1]);
    ofstream outFile(argv[2]);
    ofstream deBugFile(argv[3]);

    int N = 0;
    int numSets = 0;
    int* whichSet = NULL;
    edge* edgeList;
    edge* msList;
    int totalMSTCost = 0;

    inFile >> N;
    numSets = N;

    whichSet = new int[N + 1];
```

```
for (int i = 1; i < N + 1; i++)
{
   whichSet[i] = i;
}
edgeList = new edge(0, 0, 0);
msList = new edge(0, 0, 0);

KruskalMST KKMST(N, whichSet, numSets, totalMSTCost, edgeList, msList);

int u; int w;  int cost;
u = w = cost = 0;
edge* newEdge = NULL;
while (inFile >> u && inFile >> w && inFile >> cost)
{
   newEdge = new edge(u, w, cost);
   if (deBugFile.is_open() && newEdge != NULL)
   {
      deBugFile << "newEdge from inFile is: ";
      newEdge->printEdge(deBugFile);
      deBugFile << endl;
   }
   KKMST.insertEdgeList(newEdge, deBugFile);
   KKMST.printEdgeList(deBugFile);

}
if (deBugFile.is_open())
   deBugFile << "*** At the end of printing all edges of the input graph.***" << endl;


while (KKMST.numSets > 1)
{
   edge* nextEdge = KKMST.removeEdge(deBugFile);
   //ME
   if (deBugFile.is_open())
   {
      deBugFile << "nU: " << nextEdge->nU << " nW: " << nextEdge->nW << endl;
   }
   while (KKMST.whichSet[nextEdge->nU] == KKMST.whichSet[nextEdge->nW])
   {
      nextEdge = KKMST.removeEdge(deBugFile);
      //ME
      if(KKMST.whichSet[nextEdge->nU] == KKMST.whichSet[nextEdge->nW])
      if (deBugFile.is_open())
      {
```

```
            deBugFile << "*****whichSet[U] == whichSet[W]******" << endl;
            deBugFile << "nU: " << nextEdge->nU << " nW: " << nextEdge->nW << endl;
         }
      }
      if (deBugFile.is_open())
      {
         deBugFile << "The nextEdge is ";
         nextEdge->printEdge(deBugFile);
         deBugFile << endl;
      }
      KKMST.insertMSTList(nextEdge, deBugFile);
      KKMST.totalMSTcost += nextEdge->cost;
      KKMST.merge2Sets(nextEdge->nU, nextEdge->nW);
      KKMST.numSets--;
      if (deBugFile.is_open())
      {
         deBugFile << "numSets is " << KKMST.numSets << endl;
      }

      if (deBugFile.is_open())
      {
         deBugFile << "*****Printing whichSet Array*****" << endl;
         KKMST.printAry(deBugFile);
         deBugFile << "***Printing the remaining of edgeList***";
         KKMST.printEdgeList(deBugFile);
         deBugFile << "***Printing the growing MST List***" << endl;
         KKMST.printMSTList(deBugFile);
      }
   }

   KKMST.printMSTList(outFile);


}
```

Program output:

outFile from Data1 :

listHead -> < 0, 0, 0 > -> < 1, 3, 1 > -> < 3, 4, 2 > -> < 3, 6, 2 > -> < 2, 3, 4 > -> < 2, 5, 4 > -> NULL

outFile from Data2 :

listHead -> < 0, 0, 0 > -> < 2, 4, 1 > -> < 8, 10, 1 > -> < 8, 6, 2 > -> < 9, 8, 2 > -> < 5, 4, 2 > -> < 6, 7, 2 > -> < 6, 4, 3 > -> < 1, 6, 3 > -> < 4, 3, 3 > -> < 12, 7, 4 > -> < 9, 11, 5 > -> NULL

debugFile from Data1 :

```
newEdge from inFile is: (1, 2, 6) ->
***In insertEdgeList()*****
***In findSpot()*****
newEdge Cost: 6 (1, 2, 6) ->
Spot Cost: 0 (0, 0, 0) ->
***Leaving findSpot()*****
***Leaving insertEdgeList()*****
listHead ->  < 0, 0, 0 >  ->  < 1, 2, 6 >  -> NULL
newEdge from inFile is: (1, 3, 1) ->
***In insertEdgeList()*****
***In findSpot()*****
newEdge Cost: 1 (1, 3, 1) ->
Spot Cost: 0 (0, 0, 0) ->
***Leaving findSpot()*****
***Leaving insertEdgeList()*****
listHead ->  < 0, 0, 0 >  ->  < 1, 3, 1 >  ->  < 1, 2, 6 >  -> NULL
newEdge from inFile is: (1, 4, 3) ->
***In insertEdgeList()*****
***In findSpot()*****
newEdge Cost: 3 (1, 4, 3) ->
Spot Cost: 1 (1, 3, 1) ->
***Leaving findSpot()*****
***Leaving insertEdgeList()*****
listHead ->  < 0, 0, 0 >  ->  < 1, 3, 1 >  ->  < 1, 4, 3 >  ->  < 1, 2, 6 >  -> NULL
newEdge from inFile is: (2, 3, 4) ->
***In insertEdgeList()*****
***In findSpot()*****
newEdge Cost: 4 (2, 3, 4) ->
Spot Cost: 3 (1, 4, 3) ->
***Leaving findSpot()*****
***Leaving insertEdgeList()*****
listHead ->  < 0, 0, 0 >  ->  < 1, 3, 1 >  ->  < 1, 4, 3 >  ->  < 2, 3, 4 >  ->  < 1, 2, 6 >  -> NULL
newEdge from inFile is: (3, 4, 2) ->
***In insertEdgeList()*****
***In findSpot()*****
newEdge Cost: 2 (3, 4, 2) ->
Spot Cost: 1 (1, 3, 1) ->
***Leaving findSpot()*****
***Leaving insertEdgeList()*****
listHead ->  < 0, 0, 0 >  ->  < 1, 3, 1 >  ->  < 3, 4, 2 >  ->  < 1, 4, 3 >  ->  < 2, 3, 4 >  ->  < 1, 2, 6 >
-> NULL
newEdge from inFile is: (2, 5, 4) ->
***In insertEdgeList()*****
```

In findSpot()**
newEdge Cost: 4 (2, 5, 4) ->
Spot Cost: 4 (2, 3, 4) ->
Leaving findSpot()**
Leaving insertEdgeList()**
listHead -> < 0, 0, 0 > -> < 1, 3, 1 > -> < 3, 4, 2 > -> < 1, 4, 3 > -> < 2, 3, 4 > -> < 2, 5, 4 > -> < 1, 2, 6 > -> NULL
newEdge from inFile is: (3, 5, 4) ->
In insertEdgeList()**
In findSpot()**
newEdge Cost: 4 (3, 5, 4) ->
Spot Cost: 4 (2, 5, 4) ->
Leaving findSpot()**
Leaving insertEdgeList()**
listHead -> < 0, 0, 0 > -> < 1, 3, 1 > -> < 3, 4, 2 > -> < 1, 4, 3 > -> < 2, 3, 4 > -> < 2, 5, 4 > -> < 3, 5, 4 > -> < 1, 2, 6 > -> NULL
newEdge from inFile is: (3, 6, 2) ->
In insertEdgeList()**
In findSpot()**
newEdge Cost: 2 (3, 6, 2) ->
Spot Cost: 2 (3, 4, 2) ->
Leaving findSpot()**
Leaving insertEdgeList()**
listHead -> < 0, 0, 0 > -> < 1, 3, 1 > -> < 3, 4, 2 > -> < 3, 6, 2 > -> < 1, 4, 3 > -> < 2, 3, 4 > -> < 2, 5, 4 > -> < 3, 5, 4 > -> < 1, 2, 6 > -> NULL
newEdge from inFile is: (4, 6, 2) ->
In insertEdgeList()**
In findSpot()**
newEdge Cost: 2 (4, 6, 2) ->
Spot Cost: 2 (3, 6, 2) ->
Leaving findSpot()**
Leaving insertEdgeList()**
listHead -> < 0, 0, 0 > -> < 1, 3, 1 > -> < 3, 4, 2 > -> < 3, 6, 2 > -> < 4, 6, 2 > -> < 1, 4, 3 > -> < 2, 3, 4 > -> < 2, 5, 4 > -> < 3, 5, 4 > -> < 1, 2, 6 > -> NULL
newEdge from inFile is: (5, 6, 6) ->
In insertEdgeList()**
In findSpot()**
newEdge Cost: 6 (5, 6, 6) ->
Spot Cost: 6 (1, 2, 6) ->
Leaving findSpot()**
Leaving insertEdgeList()**
listHead -> < 0, 0, 0 > -> < 1, 3, 1 > -> < 3, 4, 2 > -> < 3, 6, 2 > -> < 4, 6, 2 > -> < 1, 4, 3 > -> < 2, 3, 4 > -> < 2, 5, 4 > -> < 3, 5, 4 > -> < 1, 2, 6 > -> < 5, 6, 6 > -> NULL
*** At the end of printing all edges of the input graph.***

*****In removeEdge()*****
tmp: 1, 3, 1
tmp.next is Null
*****Leaving removeEdge()*****
nU: 1 nW: 3
The nextEdge is (1, 3, 1) ->
In insertMSTList()**
In findSpot()**
newEdge Cost: 1 (1, 3, 1) ->
Spot Cost: 0 (0, 0, 0) ->
Leaving findSpot()**
Leaving insertMSTList()**
numSets is 5
*****Printing whichSet Array*****
whichSet[1]: 1
whichSet[2]: 2
whichSet[3]: 1
whichSet[4]: 4
whichSet[5]: 5
whichSet[6]: 6
Printing the remaining of edgeListlistHead -> < 0, 0, 0 > -> < 3, 4, 2 > -> < 3, 6, 2 > -> < 4, 6, 2 > -> < 1, 4, 3 > -> < 2, 3, 4 > -> < 2, 5, 4 > -> < 3, 5, 4 > -> < 1, 2, 6 > -> < 5, 6, 6 > -> NULL
Printing the growing MST List
listHead -> < 0, 0, 0 > -> < 1, 3, 1 > -> NULL
*****In removeEdge()*****
tmp: 3, 4, 2
tmp.next is Null
*****Leaving removeEdge()*****
nU: 3 nW: 4
The nextEdge is (3, 4, 2) ->
In insertMSTList()**
In findSpot()**
newEdge Cost: 2 (3, 4, 2) ->
Spot Cost: 1 (1, 3, 1) ->
Leaving findSpot()**
Leaving insertMSTList()**
numSets is 4
*****Printing whichSet Array*****
whichSet[1]: 1
whichSet[2]: 2
whichSet[3]: 1
whichSet[4]: 1
whichSet[5]: 5

```
whichSet[6]: 6
***Printing the remaining of edgeList***listHead ->  < 0, 0, 0 >  ->  < 3, 6, 2 >  ->  < 4, 6, 2 >  ->
< 1, 4, 3 >  ->  < 2, 3, 4 >  ->  < 2, 5, 4 >  ->  < 3, 5, 4 >  ->  < 1, 2, 6 >  ->  < 5, 6, 6 >  -> NULL
***Printing the growing MST List***
listHead ->  < 0, 0, 0 >  ->  < 1, 3, 1 >  ->  < 3, 4, 2 >  -> NULL
*****In removeEdge()*****
tmp: 3, 6, 2
tmp.next is Null
*****Leaving removeEdge()*****
nU: 3 nW: 6
The nextEdge is (3, 6, 2) ->
***In insertMSTList()*****
***In findSpot()*****
newEdge Cost: 2 (3, 6, 2) ->
Spot Cost: 2 (3, 4, 2) ->
***Leaving findSpot()*****
***Leaving insertMSTList()*****
numSets is 3
*****Printing whichSet Array*****
whichSet[1]: 1
whichSet[2]: 2
whichSet[3]: 1
whichSet[4]: 1
whichSet[5]: 5
whichSet[6]: 1
***Printing the remaining of edgeList***listHead ->  < 0, 0, 0 >  ->  < 4, 6, 2 >  ->  < 1, 4, 3 >  ->
< 2, 3, 4 >  ->  < 2, 5, 4 >  ->  < 3, 5, 4 >  ->  < 1, 2, 6 >  ->  < 5, 6, 6 >  -> NULL
***Printing the growing MST List***
listHead ->  < 0, 0, 0 >  ->  < 1, 3, 1 >  ->  < 3, 4, 2 >  ->  < 3, 6, 2 >  -> NULL
*****In removeEdge()*****
tmp: 4, 6, 2
tmp.next is Null
*****Leaving removeEdge()*****
nU: 4 nW: 6
*****In removeEdge()*****
tmp: 1, 4, 3
tmp.next is Null
*****Leaving removeEdge()*****
*****whichSet[U] == whichSet[W]******
nU: 1 nW: 4
*****In removeEdge()*****
tmp: 2, 3, 4
tmp.next is Null
*****Leaving removeEdge()*****
```

```
The nextEdge is (2, 3, 4) ->
***In insertMSTList()*****
***In findSpot()*****
newEdge Cost: 4 (2, 3, 4) ->
Spot Cost: 2 (3, 6, 2) ->
***Leaving findSpot()*****
***Leaving insertMSTList()*****
numSets is 2
*****Printing whichSet Array*****
whichSet[1]: 1
whichSet[2]: 1
whichSet[3]: 1
whichSet[4]: 1
whichSet[5]: 5
whichSet[6]: 1
***Printing the remaining of edgeList***listHead ->  < 0, 0, 0 >  ->  < 2, 5, 4 >  ->  < 3, 5, 4 >  ->
< 1, 2, 6 >  ->  < 5, 6, 6 >  -> NULL
***Printing the growing MST List***
listHead ->  < 0, 0, 0 >  ->  < 1, 3, 1 >  ->  < 3, 4, 2 >  ->  < 3, 6, 2 >  ->  < 2, 3, 4 >  -> NULL
*****In removeEdge()*****
tmp: 2, 5, 4
tmp.next is Null
*****Leaving removeEdge()*****
nU: 2 nW: 5
The nextEdge is (2, 5, 4) ->
***In insertMSTList()*****
***In findSpot()*****
newEdge Cost: 4 (2, 5, 4) ->
Spot Cost: 4 (2, 3, 4) ->
***Leaving findSpot()*****
***Leaving insertMSTList()*****
numSets is 1
*****Printing whichSet Array*****
whichSet[1]: 1
whichSet[2]: 1
whichSet[3]: 1
whichSet[4]: 1
whichSet[5]: 1
whichSet[6]: 1
***Printing the remaining of edgeList***listHead ->  < 0, 0, 0 >  ->  < 3, 5, 4 >  ->  < 1, 2, 6 >  ->
< 5, 6, 6 >  -> NULL
***Printing the growing MST List***
listHead ->  < 0, 0, 0 >  ->  < 1, 3, 1 >  ->  < 3, 4, 2 >  ->  < 3, 6, 2 >  ->  < 2, 3, 4 >  ->  < 2, 5, 4 >
-> NULL
```

debugFile from Data2 :

The debugFile for Data2 contains more than 4 pages.