Cheewin Thawornjaroenpong

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

Project6

Dijkstra’s algorithm

04/20/2023

Algorithm Steps:

```
step 0: inFile, SSSfile, deBugFile  open with args[] numNodes  get from inFile Allocate and
initialize all members in the DijktraSSS class accordingly
step 1: loadCostMatrix (inFile) sourceNode  1
step 2: setBestAry (sourceNode) setFatherAry (sourceNode) setToDoAry (sourceNode)
step 3: minNode  findMinNode (…) ToDoAry[minNode]  0 debugPrint (…)
step 4: // expanding the minNode childNode  1
step 5: if ToDoAry[childNode] == 1 { newCost  computeCost (minNode, childNode) if newCost <
BestAry [childNode] BestAry[childNode]  newCost fatherAry[childNode]  minNode debugPrint
(…) } step 6: childNode ++ step 7: repeat step 5 to
step 6 while childNode <= numNodes
step 8: repeat step 3 to
step 7 until checkToDoAry (…) == true
step 9: currentNode  1
step 10: printShortestPath (currentNode, sourceNode, SSSfile)
 step 11: currentNode ++ 4
step 12: repeat 10 and step 11 while currentNode <= numNodes
step 13: sourceNode ++
step 14: repeat step 2 to step 13 while sourceNode <= numNodes
step 15: close all files
```

Source code:
```
import java.io.*;
import java.util.Scanner;

public class ThawornjaroenpongC_Project6_Main {
    public static class DijktraSSS {
        public int numNodes = 0;
        public int sourceNode = 0;
        public int minNode = 0;
        public int currentNode = 0;
        public int newCost = 0;
        public int[][] costMatrix;
        public int[] fatherAry;
        public int[] ToDoAry;
        public int[] BestAry;

        public DijktraSSS(int numNodes) {
            this.numNodes = numNodes;
            this.costMatrix = new int [numNodes + 1][numNodes + 1];
            for(int i = 0; i < numNodes + 1; i++)
            {
                    for(int j = 0; j < numNodes + 1; j++)
                    {
                            this.costMatrix[i][j] = 9999;
                    }
            }

            this.ToDoAry = new int [numNodes + 1];
            for(int i = 0; i < numNodes + 1; i++)
            {
                this.ToDoAry[i] = 9999;
            }

            this.BestAry = new int[numNodes + 1];
            for(int i = 0; i < numNodes + 1; i++)
            {
                this.BestAry[i] = 9999;
            }

            this.fatherAry = new int[numNodes + 1];
            for(int i = 0; i < numNodes + 1; i++)
            {
                this.fatherAry[i] = 9999;
            }
```

```
}

public void loadCostMatrix(Scanner inFile, BufferedWriter deBugFile) throws IOException {
   inFile.nextLine();
   while(inFile.hasNextInt())
   {
            int row = inFile.nextInt();
            int col = inFile.nextInt();
            int cost = inFile.nextInt();
            this.costMatrix[row][col] = cost;
   }

   try {
      this.debugPrintCostMatrix(deBugFile);
   } catch (IOException e) {
      throw new RuntimeException(e);
   }
}

public void setBestAry(int sourceNode , BufferedWriter deBugFile) {
   this.sourceNode = sourceNode;
   for (int i = 1; i < numNodes + 1; i++)
   {
            this.BestAry[i] = this.costMatrix[this.sourceNode][i];
   }
   this.deBugPrintBestAry(deBugFile);
}

public void setFatherAry(int sourceNode, BufferedWriter deBugFile) {
   this.sourceNode = sourceNode;
   for (int i = 1; i < numNodes + 1; i++)
   {
            this.fatherAry[i] = this.sourceNode;
   }
   this.deBugPrintFatherAry(deBugFile);
}

public void setToDoAry(int sourceNode, BufferedWriter deBugFile) {
   this.sourceNode = sourceNode;
   for (int i = 1; i < numNodes + 1; i++)
   {
            if(i == this.sourceNode)
            {
                     this.ToDoAry[i] = 0;
            }
            else if(this.ToDoAry[i] == 0)
            {
                    this.ToDoAry[i] = 0;
            }
            else
            {
                    this.ToDoAry[i] = 1;
            }
    }
    this.deBugPrintToDoAry(deBugFile);

}

public int findMinNode() {
    int minCost = 9999;
    int minNode = 0;
    int index = 1;

    while(index <= this.numNodes)
    {
            if(this.ToDoAry[index] == 1 && this.BestAry[index] < minCost)
            {
                    minCost = this.BestAry[index];
                    minNode = index;
            }
            index++;
    }

    return minNode;
}

public int computeCost(int minNode, int Node)
{
    return this.BestAry[minNode] + this.costMatrix[minNode][Node];
}

public boolean checkToDoAry()
{
    for(int i = 1; i < numNodes + 1; i++)
    {
                    if(this.ToDoAry[i] == 1)
                            return false;
    }
    return true;
}

public void debugPrintCostMatrix(BufferedWriter deBugFile) throws IOException {
    try {
        deBugFile.write("***********2D costMatrix************ \n");
        for(int i = 1; i < this.numNodes + 1; i++)
        {
            for (int j = 1; j < this.numNodes + 1; j++)
            {
                deBugFile.write(this.costMatrix[i][j] + " ");
            }
            deBugFile.write("\n");
        }

    } catch (IOException e) {
        throw new RuntimeException(e);
    }

}

public void deBugPrintBestAry(BufferedWriter deBugFile)
{
    try {
                                        deBugFile.write("************ Best Array ********** \n");
                                        for(int i = 1; i < numNodes + 1; i++)
                {
                                                    deBugFile.write(this.BestAry[i] + " ");
                }
                                        deBugFile.write("\n");
                            } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                            }

}

public void deBugPrintToDoAry(BufferedWriter deBugFile)
{
    try {
                                        deBugFile.write("************ ToDo Array ********** \n");
                                        for(int i = 1; i < numNodes + 1; i++)
                {
                        deBugFile.write(this.ToDoAry[i] + " ");
            }
                    deBugFile.write("\n");
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
    }

    public void deBugPrintFatherAry(BufferedWriter deBugFile)
    {
       try {
                    deBugFile.write("************ Father Array ********** \n");
                    for(int i = 1; i < numNodes + 1; i++)
            {
                        deBugFile.write(this.fatherAry[i] + " ");
            }
                    deBugFile.write("\n");
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
    }

    public void printShortestPath(int currentNode, int sourceNode, BufferedWriter SSSfile)
    {
       this.sourceNode = sourceNode;
       this.currentNode = currentNode;
       int totalCost = 0;
       try {

                    SSSfile.write("The path from " + this.sourceNode + " to " +
this.currentNode + " : " + this.currentNode);
                    while(this.fatherAry[currentNode] != sourceNode)
                    {
                        SSSfile.write(" <- " + this.fatherAry[currentNode]);
                        currentNode = this.fatherAry[currentNode];
                    }
                    SSSfile.write(" <- " + this.sourceNode +  " : cost = " +
this.BestAry[this.currentNode] +  "\n");


       } catch (IOException e) {
                    // TODO Auto-generated catch block
```

```
                                e.printStackTrace();
                        }

    }


}
public static void main(String[] args) throws IOException {


    try {
        Scanner inFile = new Scanner(new FileReader(args[0]));
        BufferedWriter SSSfile = new BufferedWriter(new FileWriter(args[1]));
        BufferedWriter deBugFile = new BufferedWriter(new FileWriter(args[2]));

        int numNodes = 0;
        numNodes = inFile.nextInt();



        DijktraSSS DSSS = new DijktraSSS(numNodes);
        DSSS.debugPrintCostMatrix(deBugFile);
        DSSS.loadCostMatrix(inFile, deBugFile);

        int sourceNode = 1;
        while(sourceNode <= DSSS.numNodes)
        {
                DSSS.setBestAry(sourceNode, deBugFile);
            DSSS.setFatherAry(sourceNode, deBugFile);
            DSSS.setToDoAry(sourceNode, deBugFile);

            while(DSSS.checkToDoAry() == false)
            {
                DSSS.minNode = DSSS.findMinNode();
                DSSS.ToDoAry[DSSS.minNode] = 0;
                DSSS.deBugPrintToDoAry(deBugFile);

                int childNode = 1;
                while (childNode <= DSSS.numNodes)
                {
                    if(DSSS.ToDoAry[childNode] == 1)
                    {
                            int newCost = DSSS.computeCost(DSSS.minNode, childNode);
                            if(newCost < DSSS.BestAry[childNode])
                            {
```

```
                                        DSSS.BestAry[childNode] = newCost;
                                        DSSS.fatherAry[childNode] = DSSS.minNode;
                                        DSSS.deBugPrintBestAry(deBugFile);
                                        DSSS.deBugPrintFatherAry(deBugFile);
                                  }
                           }

                           childNode++;
                       }
                   }

                   int currentNode = 1;
                   while(currentNode <= DSSS.numNodes)
                   {
                           SSSfile.write("Source node = " + sourceNode + "\n");
                       DSSS.printShortestPath(currentNode, sourceNode, SSSfile);
                       currentNode++;
                   }
                   SSSfile.write("\n");
                   sourceNode++;
               }


               inFile.close();
                       SSSfile.close();
                       deBugFile.close();

             } catch (IOException e) {
               throw new RuntimeException(e);
           }


       }
   }
```

Program output:

outFile from Data1 :

```
Source node = 1
The path from 1 to 1 : 1 <- 1 : cost = 9999
Source node = 1
The path from 1 to 2 : 2 <- 1 : cost = 10
Source node = 1
The path from 1 to 3 : 3 <- 4 <- 1 : cost = 50
Source node = 1
The path from 1 to 4 : 4 <- 1 : cost = 30
Source node = 1
The path from 1 to 5 : 5 <- 3 <- 4 <- 1 : cost = 60

Source node = 2
The path from 2 to 1 : 1 <- 2 : cost = 9999
Source node = 2
The path from 2 to 2 : 2 <- 2 : cost = 9999
Source node = 2
The path from 2 to 3 : 3 <- 2 : cost = 50
Source node = 2
The path from 2 to 4 : 4 <- 2 : cost = 9999
Source node = 2
The path from 2 to 5 : 5 <- 2 : cost = 9999

Source node = 3
The path from 3 to 1 : 1 <- 3 : cost = 9999
Source node = 3
The path from 3 to 2 : 2 <- 3 : cost = 9999
Source node = 3
The path from 3 to 3 : 3 <- 3 : cost = 9999
Source node = 3
The path from 3 to 4 : 4 <- 3 : cost = 9999
Source node = 3
The path from 3 to 5 : 5 <- 3 : cost = 10

Source node = 4
The path from 4 to 1 : 1 <- 4 : cost = 9999
Source node = 4
The path from 4 to 2 : 2 <- 4 : cost = 9999
Source node = 4
The path from 4 to 3 : 3 <- 4 : cost = 20
Source node = 4
```

```
The path from 4 to 4 : 4 <- 4 : cost = 9999
Source node = 4
The path from 4 to 5 : 5 <- 4 : cost = 60

Source node = 5
The path from 5 to 1 : 1 <- 5 : cost = 40
Source node = 5
The path from 5 to 2 : 2 <- 5 : cost = 9999
Source node = 5
The path from 5 to 3 : 3 <- 5 : cost = 9999
Source node = 5
The path from 5 to 4 : 4 <- 5 : cost = 9999
Source node = 5
The path from 5 to 5 : 5 <- 5 : cost = 9999
```

outFile from Data2 :

```
Source node = 1
The path from 1 to 1 : 1 <- 1 : cost = 9999
Source node = 1
The path from 1 to 2 : 2 <- 3 <- 1 : cost = 7
Source node = 1
The path from 1 to 3 : 3 <- 1 : cost = 5
Source node = 1
The path from 1 to 4 : 4 <- 3 <- 1 : cost = 10
Source node = 1
The path from 1 to 5 : 5 <- 4 <- 3 <- 1 : cost = 15
Source node = 1
The path from 1 to 6 : 6 <- 4 <- 3 <- 1 : cost = 13
Source node = 1
The path from 1 to 7 : 7 <- 8 <- 2 <- 3 <- 1 : cost = 11
Source node = 1
The path from 1 to 8 : 8 <- 2 <- 3 <- 1 : cost = 9
Source node = 1
The path from 1 to 9 : 9 <- 7 <- 8 <- 2 <- 3 <- 1 : cost = 21

Source node = 2
The path from 2 to 1 : 1 <- 2 : cost = 25
Source node = 2
The path from 2 to 2 : 2 <- 2 : cost = 9999
Source node = 2
The path from 2 to 3 : 3 <- 2 : cost = 35
Source node = 2
```

```
The path from 2 to 4 : 4 <- 2 : cost = 9999
Source node = 2
The path from 2 to 5 : 5 <- 2 : cost = 9999
Source node = 2
The path from 2 to 6 : 6 <- 2 : cost = 9999
Source node = 2
The path from 2 to 7 : 7 <- 2 : cost = 40
Source node = 2
The path from 2 to 8 : 8 <- 2 : cost = 2
Source node = 2
The path from 2 to 9 : 9 <- 2 : cost = 9999

Source node = 3
The path from 3 to 1 : 1 <- 3 : cost = 9999
Source node = 3
The path from 3 to 2 : 2 <- 3 : cost = 2
Source node = 3
The path from 3 to 3 : 3 <- 3 : cost = 9999
Source node = 3
The path from 3 to 4 : 4 <- 3 : cost = 5
Source node = 3
The path from 3 to 5 : 5 <- 3 : cost = 9999
Source node = 3
The path from 3 to 6 : 6 <- 3 : cost = 9999
Source node = 3
The path from 3 to 7 : 7 <- 3 : cost = 30
Source node = 3
The path from 3 to 8 : 8 <- 3 : cost = 10
Source node = 3
The path from 3 to 9 : 9 <- 3 : cost = 9999

Source node = 4
The path from 4 to 1 : 1 <- 4 : cost = 9999
Source node = 4
The path from 4 to 2 : 2 <- 4 : cost = 9999
Source node = 4
The path from 4 to 3 : 3 <- 4 : cost = 9999
Source node = 4
The path from 4 to 4 : 4 <- 4 : cost = 9999
Source node = 4
The path from 4 to 5 : 5 <- 4 : cost = 5
Source node = 4
The path from 4 to 6 : 6 <- 4 : cost = 3
Source node = 4
```

```
The path from 4 to 7 : 7 <- 4 : cost = 25
Source node = 4
The path from 4 to 8 : 8 <- 4 : cost = 20
Source node = 4
The path from 4 to 9 : 9 <- 4 : cost = 9999

Source node = 5
The path from 5 to 1 : 1 <- 5 : cost = 9999
Source node = 5
The path from 5 to 2 : 2 <- 5 : cost = 10
Source node = 5
The path from 5 to 3 : 3 <- 5 : cost = 9999
Source node = 5
The path from 5 to 4 : 4 <- 5 : cost = 9999
Source node = 5
The path from 5 to 5 : 5 <- 5 : cost = 9999
Source node = 5
The path from 5 to 6 : 6 <- 5 : cost = 15
Source node = 5
The path from 5 to 7 : 7 <- 5 : cost = 9999
Source node = 5
The path from 5 to 8 : 8 <- 5 : cost = 3
Source node = 5
The path from 5 to 9 : 9 <- 5 : cost = 9999

Source node = 6
The path from 6 to 1 : 1 <- 6 : cost = 5
Source node = 6
The path from 6 to 2 : 2 <- 6 : cost = 5
Source node = 6
The path from 6 to 3 : 3 <- 6 : cost = 20
Source node = 6
The path from 6 to 4 : 4 <- 6 : cost = 9999
Source node = 6
The path from 6 to 5 : 5 <- 6 : cost = 9999
Source node = 6
The path from 6 to 6 : 6 <- 6 : cost = 9999
Source node = 6
The path from 6 to 7 : 7 <- 6 : cost = 2
Source node = 6
The path from 6 to 8 : 8 <- 6 : cost = 9999
Source node = 6
The path from 6 to 9 : 9 <- 6 : cost = 9999
```

```
Source node = 7
The path from 7 to 1 : 1 <- 7 : cost = 40
Source node = 7
The path from 7 to 2 : 2 <- 7 : cost = 9999
Source node = 7
The path from 7 to 3 : 3 <- 7 : cost = 9999
Source node = 7
The path from 7 to 4 : 4 <- 7 : cost = 4
Source node = 7
The path from 7 to 5 : 5 <- 7 : cost = 30
Source node = 7
The path from 7 to 6 : 6 <- 7 : cost = 3
Source node = 7
The path from 7 to 7 : 7 <- 7 : cost = 9999
Source node = 7
The path from 7 to 8 : 8 <- 7 : cost = 9999
Source node = 7
The path from 7 to 9 : 9 <- 7 : cost = 10

Source node = 8
The path from 8 to 1 : 1 <- 8 : cost = 6
Source node = 8
The path from 8 to 2 : 2 <- 8 : cost = 9999
Source node = 8
The path from 8 to 3 : 3 <- 8 : cost = 9999
Source node = 8
The path from 8 to 4 : 4 <- 8 : cost = 9999
Source node = 8
The path from 8 to 5 : 5 <- 8 : cost = 9999
Source node = 8
The path from 8 to 6 : 6 <- 8 : cost = 7
Source node = 8
The path from 8 to 7 : 7 <- 8 : cost = 2
Source node = 8
The path from 8 to 8 : 8 <- 8 : cost = 9999
Source node = 8
The path from 8 to 9 : 9 <- 8 : cost = 9999

Source node = 9
The path from 9 to 1 : 1 <- 9 : cost = 9999
Source node = 9
The path from 9 to 2 : 2 <- 9 : cost = 18
Source node = 9
The path from 9 to 3 : 3 <- 9 : cost = 9999
```

```
Source node = 9
The path from 9 to 4 : 4 <- 9 : cost = 3
Source node = 9
The path from 9 to 5 : 5 <- 9 : cost = 6
Source node = 9
The path from 9 to 6 : 6 <- 9 : cost = 22
Source node = 9
The path from 9 to 7 : 7 <- 9 : cost = 9999
Source node = 9
The path from 9 to 8 : 8 <- 9 : cost = 9999
Source node = 9
The path from 9 to 9 : 9 <- 9 : cost = 9999
```

debugFile from Data1 :

```
***********2D costMatrix************
9999 9999 9999 9999 9999
9999 9999 9999 9999 9999
9999 9999 9999 9999 9999
9999 9999 9999 9999 9999
9999 9999 9999 9999 9999
***********2D costMatrix************
9999 10 70 30 9999
9999 9999 50 9999 9999
9999 9999 9999 9999 10
9999 9999 20 9999 60
40 9999 9999 9999 9999
************ Best Array **********
9999 10 70 30 9999
************ Father Array **********
1 1 1 1 1
************ ToDo Array **********
0 1 1 1 1
************ ToDo Array **********
0 0 1 1 1
************ Best Array **********
9999 10 60 30 9999
************ Father Array **********
1 1 2 1 1
************ ToDo Array **********
0 0 1 0 1
************ Best Array **********
9999 10 50 30 9999
************ Father Array **********
1 1 4 1 1
************ Best Array **********
9999 10 50 30 90
************ Father Array **********
1 1 4 1 4
************ ToDo Array **********
0 0 0 0 1
************ Best Array **********
9999 10 50 30 60
************ Father Array **********
1 1 4 1 3
************ ToDo Array **********
0 0 0 0 0
```

```
************ Best Array **********
9999 9999 50 9999 9999
************ Father Array **********
2 2 2 2 2
************ ToDo Array **********
0 0 0 0 0
************ Best Array **********
9999 9999 9999 9999 10
************ Father Array **********
3 3 3 3 3
************ ToDo Array **********
0 0 0 0 0
************ Best Array **********
9999 9999 20 9999 60
************ Father Array **********
4 4 4 4 4
************ ToDo Array **********
0 0 0 0 0
************ Best Array **********
40 9999 9999 9999 9999
************ Father Array **********
5 5 5 5 5
************ ToDo Array **********
0 0 0 0 0
```

debugFile from Data2 :

The debugFile for Data2 contains more than 4 pages.