Cover Page

Name: David Chen Salas

Section: 2023 Fall Term (1) Algorithms I CSCI 700 231[25504] (Queens College)

Project#: 5

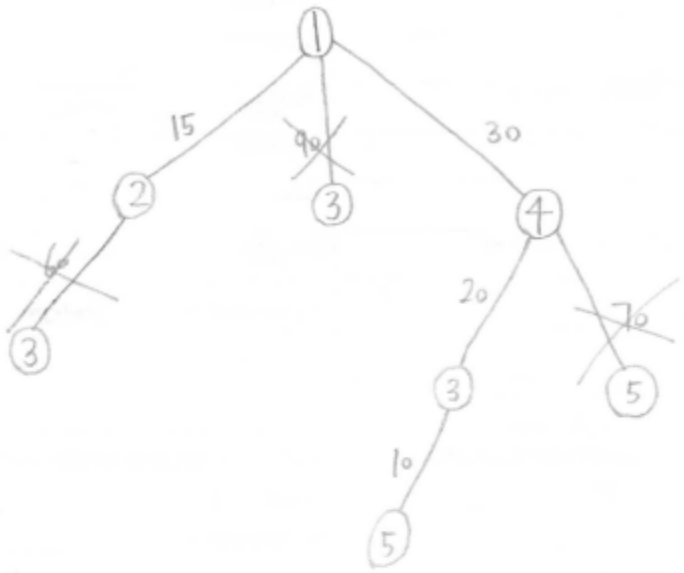
Project Name: All pairs shortest paths using Dijkstra's (SSS) algorithm

Due Date: 11/03/2023, Friday before midnight

Algorithm Steps:

step 0: inFile, SSSfile, deBugFile open via argv[]
numNodes get from inFile
Allocate and initialize all members in the DijktraSSS class accordingly
step 1: loadCostMatrix (inFile)
sourceNode 1
deBugFile “Source Node is” // write sourceNode
step 2: initBest (sourceNode)
initParent (sourceNode)
initToDo (sourceNode)
step 3: minNode findMinNode (…)
deBugFile “minNode is” // write minNode
ToDo[minNode] 0
printToDo (…)
printParent (…)
printBest (…)
Step 4: // expanding the minNode
childNode 1
deBugFile “childNode is” // write childNode
step 5: if ToDo [childNode] == 1 {
newCost Best [minNode] + costMatrix [minNode, childNode]
if newCost < Best [childNode]
deBugFile “newCost < Best [childNode]” // write both costs
Best [childNode] newCost
Parent [childNode] minNode
printParent (…)
printBest (…)
}
step 6: childNode ++
step 7: repeat step 5 to step 6 until childNode > numNodes
step 8: repeat step 3 to step 7 until checkToDo (…) == true
step 9: currentNode 1
step 10: printShortestPath (currentNode, sourceNode, SSSfile)
step 11: currentNode ++
step 12: repeat 10 and step 11 until currentNode > numNodes
step 13: sourceNode ++
step 14: repeat step 2 to step 13 while sourceNode <= numNodes
step 15: close all files

Illustration



Paths		Cost
1 to 1	1 → 1	0
1 to 2	1 → 2	15
1 to 3	1 → 4 → 3	50
1 to 4	1 → 4	30
1 to 5	1 → 4 → 3 → 5	60

Source Code

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

using namespace std;

ifstream inFile;
ofstream SSSfile, deBugFile;

int numNodes, sourceNode, minNode, currentNode, newCost;
int** costMatrix;
int* Parent;
int* ToDo;
int* Best;

void loadCostMatrix(ifstream& in_file);

void initBest(int srcNode);
void initParent(int srcNode);
void initToDo(int srcNode);

int findMinNode();

void printToDo();
void printParent();
void printBest();
void printShortestPath(int curNode, int srcNode, ofstream& SSS_file);

bool checkToDo();


int main(int argc, const char* argv[])
{
   inFile.open(argv[1]);
   SSSfile.open(argv[2]);
   deBugFile.open(argv[3]);

   inFile >> numNodes;
   costMatrix = new int* [numNodes + 1];
   for (int i = 0; i < numNodes + 1; i++) {
      costMatrix[i] = new int[numNodes];
   }
   Parent = new int[numNodes + 1];
   ToDo = new int[numNodes + 1];
   Best = new int[numNodes + 1];
   loadCostMatrix(inFile);
   sourceNode = 1;

   SSSfile << "========================================================\n";
   SSSfile << "There are " << numNodes << " nodes in the input graph. Below are all pairs of shortest paths:\n";

   while (sourceNode <= numNodes) {
      deBugFile << "Source Node is " << sourceNode << endl;
      SSSfile << "========================================================\nThe Source node = " <<
sourceNode << "\n\n";

      initBest(sourceNode);
      initParent(sourceNode);
      initToDo(sourceNode);
```

```
        int childNode;
        while (!checkToDo()) {
            minNode = findMinNode();
            ToDo[minNode] = 0;
            printToDo();
            printParent();
            printBest();

            childNode = 1;
            while (childNode <= numNodes) {
                deBugFile << "childNode is " << childNode << endl;
                if (ToDo[childNode] == 1) {
                    newCost = Best[minNode] + costMatrix[minNode][childNode];
                    if (newCost < Best[childNode]) {
                        deBugFile << "newCost:" << newCost << " < Best[" << childNode << "]:" << Best[childNode] << endl;
                        Best[childNode] = newCost;
                        Parent[childNode] = minNode;
                        deBugFile << "Update Parent Array and Best Array:\n";
                        printParent();
                        printBest();
                    }
                }
                childNode++;
            }
        }
        currentNode = 1;
        while (currentNode <= numNodes) {
            printShortestPath(currentNode, sourceNode, SSSfile);
            currentNode++;
        }
        sourceNode++;
    }


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

void loadCostMatrix(ifstream& in_file) {
    for (int i = 0; i < numNodes + 1; i++) {
        for (int j = 0; j < numNodes + 1; j++) {
            if (i == j) {
                costMatrix[i][j] = 0;
            }
            else {
                costMatrix[i][j] = 9999;
            }
        }
    }
    int i, j;
    while (!in_file.eof()) {
        in_file >> i >> j;
        in_file >> costMatrix[i][j];
    }
}

void initBest(int srcNode) {
    for (int i = 0; i <= numNodes; i++) {
        Best[i] = costMatrix[srcNode][i];
    }
}
```

```
void initParent(int srcNode) {
   for (int i = 0; i <= numNodes; i++) {
      Parent[i] = srcNode;
   }
}

void initToDo(int srcNode) {
   for (int i = 0; i <= numNodes; i++) {
      ToDo[i] = 1;
   }
   ToDo[srcNode] = 0;
}

int findMinNode() {
   deBugFile << "Entering findMinNode() method!\n";
   int minCost = 9999;
   minNode = 0;
   for (int i = 1; i <= numNodes; i++) {
      if (ToDo[i] == 1 && Best[i] < minCost) {
         minCost = Best[i];
         minNode = i;
      }
   }
   deBugFile << "Leaving findMinNode(): minNode is " << minNode << endl;
   return minNode;
}

void printToDo() {
   deBugFile << "Below is ToDo Array:\n";
   for (int i = 1; i <= numNodes; i++) {
      deBugFile << i << "\t";
   }
   deBugFile << endl;
   for (int i = 1; i <= numNodes; i++) {
      deBugFile << ToDo[i] << "\t";
   }
   deBugFile << endl;
}

void printParent() {
   deBugFile << "Below is Parent Array:\n";
   for (int i = 1; i <= numNodes; i++) {
      deBugFile << i << "\t";
   }
   deBugFile << endl;
   for (int i = 1; i <= numNodes; i++) {
      deBugFile << Parent[i] << "\t";
   }
   deBugFile << endl;
}

void printBest() {
   deBugFile << "Below is Best Array:\n";
   for (int i = 1; i <= numNodes; i++) {
      deBugFile << i << "\t";
   }
   deBugFile << endl;
   for (int i = 1; i <= numNodes; i++) {
      deBugFile << Best[i] << "\t";
   }
   deBugFile << endl;
```

```
}

bool checkToDo() {
    for (int i = 1; i <= numNodes; i++) {
        if (ToDo[i] == 1) {
            return false;
        }
    }
    return true;
}

void printShortestPath(int curNode, int srcNode, ofstream& SSS_file) {
    int totalCost = Best[curNode];
    SSS_file << "The path from " << srcNode << " to " << curNode << " : " << curNode << " <-- ";
    while (Parent[curNode] != srcNode) {
        curNode = Parent[curNode];
        SSS_file << curNode << " <-- ";
    }
    SSS_file << srcNode << " : cost = " << totalCost << endl;
}
```

Program Output

*****Data 1*****
SSSfile:

==========

There are 5 nodes in the input graph. Below are all pairs of shortest paths:

==========

The Source node = 1

The path from 1 to 1 : 1 <-- 1 : cost = 0
The path from 1 to 2 : 2 <-- 1 : cost = 15
The path from 1 to 3 : 3 <-- 4 <-- 1 : cost = 50
The path from 1 to 4 : 4 <-- 1 : cost = 30
The path from 1 to 5 : 5 <-- 3 <-- 4 <-- 1 : cost = 60

==========

The Source node = 2

The path from 2 to 1 : 1 <-- 5 <-- 3 <-- 2 : cost = 110
The path from 2 to 2 : 2 <-- 2 : cost = 0
The path from 2 to 3 : 3 <-- 2 : cost = 60
The path from 2 to 4 : 4 <-- 1 <-- 5 <-- 3 <-- 2 : cost = 140
The path from 2 to 5 : 5 <-- 3 <-- 2 : cost = 70

==========

The Source node = 3

The path from 3 to 1 : 1 <-- 5 <-- 3 : cost = 50
The path from 3 to 2 : 2 <-- 1 <-- 5 <-- 3 : cost = 65
The path from 3 to 3 : 3 <-- 3 : cost = 0
The path from 3 to 4 : 4 <-- 1 <-- 5 <-- 3 : cost = 80
The path from 3 to 5 : 5 <-- 3 : cost = 10

==========

The Source node = 4

The path from 4 to 1 : 1 <-- 5 <-- 3 <-- 4 : cost = 70
The path from 4 to 2 : 2 <-- 1 <-- 5 <-- 3 <-- 4 : cost = 85
The path from 4 to 3 : 3 <-- 4 : cost = 20
The path from 4 to 4 : 4 <-- 4 : cost = 0
The path from 4 to 5 : 5 <-- 3 <-- 4 : cost = 30

==========

The Source node = 5

The path from 5 to 1 : 1 <-- 5 : cost = 40
The path from 5 to 2 : 2 <-- 1 <-- 5 : cost = 55
The path from 5 to 3 : 3 <-- 4 <-- 1 <-- 5 : cost = 90
The path from 5 to 4 : 4 <-- 1 <-- 5 : cost = 70
The path from 5 to 5 : 5 <-- 5 : cost = 0

deBugFile:

```
Source Node is 1
Entering findMinNode() method!
Leaving findMinNode(): minNode is 2
Below is ToDo Array:
1       2       3       4       5
0       0       1       1       1
Below is Parent Array:
1       2       3       4       5
1       1       1       1       1
Below is Best Array:
1       2       3       4       5
0       15      90      30      9999
childNode is 1
childNode is 2
childNode is 3
newCost:75 < Best[3]:90
Update Parent Array and Best Array:
Below is Parent Array:
1       2       3       4       5
1       1       2       1       1
Below is Best Array:
1       2       3       4       5
0       15      75      30      9999
childNode is 4
childNode is 5
Entering findMinNode() method!
Leaving findMinNode(): minNode is 4
Below is ToDo Array:
1       2       3       4       5
0       0       1       0       1
Below is Parent Array:
1       2       3       4       5
1       1       2       1       1
Below is Best Array:
1       2       3       4       5
0       15      75      30      9999
childNode is 1
childNode is 2
childNode is 3
newCost:50 < Best[3]:75
Update Parent Array and Best Array:
Below is Parent Array:
1       2       3       4       5
1       1       4       1       1
Below is Best Array:
```

```
1       2       3       4       5
0       15      50      30      9999
childNode is 4
childNode is 5
newCost:100 < Best[5]:9999
Update Parent Array and Best Array:
Below is Parent Array:
1       2       3       4       5
1       1       4       1       4
Below is Best Array:
1       2       3       4       5
0       15      50      30      100
Entering findMinNode() method!
Leaving findMinNode(): minNode is 3
Below is ToDo Array:
1       2       3       4       5
0       0       0       0       1
Below is Parent Array:
1       2       3       4       5
1       1       4       1       4
Below is Best Array:
1       2       3       4       5
0       15      50      30      100
childNode is 1
childNode is 2
childNode is 3
childNode is 4
childNode is 5
newCost:60 < Best[5]:100
Update Parent Array and Best Array:
Below is Parent Array:
1       2       3       4       5
1       1       4       1       3
Below is Best Array:
1       2       3       4       5
0       15      50      30      60
Entering findMinNode() method!
Leaving findMinNode(): minNode is 5
Below is ToDo Array:
1       2       3       4       5
0       0       0       0       0
Below is Parent Array:
1       2       3       4       5
1       1       4       1       3
Below is Best Array:
1       2       3       4       5
```

0 15 50 30 60
childNode is 1
childNode is 2
childNode is 3
childNode is 4
childNode is 5
Source Node is 2
Entering findMinNode() method!
Leaving findMinNode(): minNode is 3
Below is ToDo Array:
1 2 3 4 5
1 0 0 1 1
Below is Parent Array:
1 2 3 4 5
2 2 2 2 2
Below is Best Array:
1 2 3 4 5
9999 0 60 9999 9999
childNode is 1
childNode is 2
childNode is 3
childNode is 4
childNode is 5
newCost:70 < Best[5]:9999
Update Parent Array and Best Array:
Below is Parent Array:
1 2 3 4 5
2 2 2 2 3
Below is Best Array:
1 2 3 4 5
9999 0 60 9999 70
Entering findMinNode() method!
Leaving findMinNode(): minNode is 5
Below is ToDo Array:
1 2 3 4 5
1 0 0 1 0
Below is Parent Array:
1 2 3 4 5
2 2 2 2 3
Below is Best Array:
1 2 3 4 5
9999 0 60 9999 70
childNode is 1
newCost:110 < Best[1]:9999
Update Parent Array and Best Array:
Below is Parent Array:

```
1       2       3       4       5
5       2       2       2       3
Below is Best Array:
1       2       3       4       5
110     0       60      9999    70
childNode is 2
childNode is 3
childNode is 4
childNode is 5
Entering findMinNode() method!
Leaving findMinNode(): minNode is 1
Below is ToDo Array:
1       2       3       4       5
0       0       0       1       0
Below is Parent Array:
1       2       3       4       5
5       2       2       2       3
Below is Best Array:
1       2       3       4       5
110     0       60      9999    70
childNode is 1
childNode is 2
childNode is 3
childNode is 4
newCost:140 < Best[4]:9999
Update Parent Array and Best Array:
Below is Parent Array:
1       2       3       4       5
5       2       2       1       3
Below is Best Array:
1       2       3       4       5
110     0       60      140     70
childNode is 5
Entering findMinNode() method!
Leaving findMinNode(): minNode is 4
Below is ToDo Array:
1       2       3       4       5
0       0       0       0       0
Below is Parent Array:
1       2       3       4       5
5       2       2       1       3
Below is Best Array:
1       2       3       4       5
110     0       60      140     70
childNode is 1
childNode is 2
```

childNode is 3
childNode is 4
childNode is 5
Source Node is 3
Entering findMinNode() method!
Leaving findMinNode(): minNode is 5
Below is ToDo Array:
1 2 3 4 5
1 1 0 1 0
Below is Parent Array:
1 2 3 4 5
3 3 3 3 3
Below is Best Array:
1 2 3 4 5
9999 9999 0 9999 10
childNode is 1
newCost:50 < Best[1]:9999
Update Parent Array and Best Array:
Below is Parent Array:
1 2 3 4 5
5 3 3 3 3
Below is Best Array:
1 2 3 4 5
50 9999 0 9999 10
childNode is 2
childNode is 3
childNode is 4
childNode is 5
Entering findMinNode() method!
Leaving findMinNode(): minNode is 1
Below is ToDo Array:
1 2 3 4 5
0 1 0 1 0
Below is Parent Array:
1 2 3 4 5
5 3 3 3 3
Below is Best Array:
1 2 3 4 5
50 9999 0 9999 10
childNode is 1
childNode is 2
newCost:65 < Best[2]:9999
Update Parent Array and Best Array:
Below is Parent Array:
1 2 3 4 5
5 1 3 3 3

Data 2
SSSfile:

==========

There are 8 nodes in the input graph. Below are all pairs of shortest paths:

==========

The Source node = 1

The path from 1 to 1 : 1 <-- 1 : cost = 0
The path from 1 to 2 : 2 <-- 3 <-- 1 : cost = 7
The path from 1 to 3 : 3 <-- 1 : cost = 5
The path from 1 to 4 : 4 <-- 3 <-- 1 : cost = 10
The path from 1 to 5 : 5 <-- 4 <-- 3 <-- 1 : cost = 16
The path from 1 to 6 : 6 <-- 4 <-- 3 <-- 1 : cost = 13
The path from 1 to 7 : 7 <-- 8 <-- 2 <-- 3 <-- 1 : cost = 11
The path from 1 to 8 : 8 <-- 2 <-- 3 <-- 1 : cost = 9

==========

The Source node = 2

The path from 2 to 1 : 1 <-- 8 <-- 2 : cost = 8
The path from 2 to 2 : 2 <-- 2 : cost = 0
The path from 2 to 3 : 3 <-- 1 <-- 8 <-- 2 : cost = 13
The path from 2 to 4 : 4 <-- 7 <-- 8 <-- 2 : cost = 12
The path from 2 to 5 : 5 <-- 4 <-- 7 <-- 8 <-- 2 : cost = 18
The path from 2 to 6 : 6 <-- 7 <-- 8 <-- 2 : cost = 8
The path from 2 to 7 : 7 <-- 8 <-- 2 : cost = 4
The path from 2 to 8 : 8 <-- 2 : cost = 2

==========

The Source node = 3

The path from 3 to 1 : 1 <-- 8 <-- 2 <-- 3 : cost = 10
The path from 3 to 2 : 2 <-- 3 : cost = 2
The path from 3 to 3 : 3 <-- 3 : cost = 0
The path from 3 to 4 : 4 <-- 3 : cost = 5
The path from 3 to 5 : 5 <-- 4 <-- 3 : cost = 11
The path from 3 to 6 : 6 <-- 4 <-- 3 : cost = 8
The path from 3 to 7 : 7 <-- 8 <-- 2 <-- 3 : cost = 6
The path from 3 to 8 : 8 <-- 2 <-- 3 : cost = 4

==========

The Source node = 4

The path from 4 to 1 : 1 <-- 6 <-- 4 : cost = 12
The path from 4 to 2 : 2 <-- 6 <-- 4 : cost = 9
The path from 4 to 3 : 3 <-- 1 <-- 6 <-- 4 : cost = 17
The path from 4 to 4 : 4 <-- 4 : cost = 0
The path from 4 to 5 : 5 <-- 4 : cost = 6
The path from 4 to 6 : 6 <-- 4 : cost = 3

The path from 4 to 7 : 7 <-- 6 <-- 4 : cost = 5
The path from 4 to 8 : 8 <-- 2 <-- 6 <-- 4 : cost = 11
==========
The Source node = 5

The path from 5 to 1 : 1 <-- 8 <-- 5 : cost = 13
The path from 5 to 2 : 2 <-- 5 : cost = 14
The path from 5 to 3 : 3 <-- 1 <-- 8 <-- 5 : cost = 18
The path from 5 to 4 : 4 <-- 7 <-- 8 <-- 5 : cost = 17
The path from 5 to 5 : 5 <-- 5 : cost = 0
The path from 5 to 6 : 6 <-- 7 <-- 8 <-- 5 : cost = 13
The path from 5 to 7 : 7 <-- 8 <-- 5 : cost = 9
The path from 5 to 8 : 8 <-- 5 : cost = 7
==========
The Source node = 6

The path from 6 to 1 : 1 <-- 6 : cost = 9
The path from 6 to 2 : 2 <-- 6 : cost = 6
The path from 6 to 3 : 3 <-- 1 <-- 6 : cost = 14
The path from 6 to 4 : 4 <-- 7 <-- 6 : cost = 10
The path from 6 to 5 : 5 <-- 4 <-- 7 <-- 6 : cost = 16
The path from 6 to 6 : 6 <-- 6 : cost = 0
The path from 6 to 7 : 7 <-- 6 : cost = 2
The path from 6 to 8 : 8 <-- 2 <-- 6 : cost = 8
==========
The Source node = 7

The path from 7 to 1 : 1 <-- 6 <-- 7 : cost = 13
The path from 7 to 2 : 2 <-- 6 <-- 7 : cost = 10
The path from 7 to 3 : 3 <-- 1 <-- 6 <-- 7 : cost = 18
The path from 7 to 4 : 4 <-- 7 : cost = 8
The path from 7 to 5 : 5 <-- 4 <-- 7 : cost = 14
The path from 7 to 6 : 6 <-- 7 : cost = 4
The path from 7 to 7 : 7 <-- 7 : cost = 0
The path from 7 to 8 : 8 <-- 2 <-- 6 <-- 7 : cost = 12
==========
The Source node = 8

The path from 8 to 1 : 1 <-- 8 : cost = 6
The path from 8 to 2 : 2 <-- 6 <-- 7 <-- 8 : cost = 12
The path from 8 to 3 : 3 <-- 1 <-- 8 : cost = 11
The path from 8 to 4 : 4 <-- 7 <-- 8 : cost = 10
The path from 8 to 5 : 5 <-- 4 <-- 7 <-- 8 : cost = 16
The path from 8 to 6 : 6 <-- 7 <-- 8 : cost = 6
The path from 8 to 7 : 7 <-- 8 : cost = 2
The path from 8 to 8 : 8 <-- 8 : cost = 0

deBugFile:

Source Node is 1
Entering findMinNode() method!
Leaving findMinNode(): minNode is 3
Below is ToDo Array:

1	2	3	4	5	6	7	8
0	1	0	1	1	1	1	1

Below is Parent Array:

1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1

Below is Best Array:

1	2	3	4	5	6	7	8
0	15	5	25	35	9999	9999	21

childNode is 1
childNode is 2
newCost:7 < Best[2]:15
Update Parent Array and Best Array:
Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	1	1	1	1	1

Below is Best Array:

1	2	3	4	5	6	7	8
0	7	5	25	35	9999	9999	21

childNode is 3
childNode is 4
newCost:10 < Best[4]:25
Update Parent Array and Best Array:
Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	3	1	1	1	1

Below is Best Array:

1	2	3	4	5	6	7	8
0	7	5	10	35	9999	9999	21

childNode is 5
childNode is 6
childNode is 7
newCost:38 < Best[7]:9999
Update Parent Array and Best Array:
Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	3	1	1	3	1

Below is Best Array:

1	2	3	4	5	6	7	8
0	7	5	10	35	9999	38	21

childNode is 8
newCost:17 < Best[8]:21

Update Parent Array and Best Array:
Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	3	1	1	3	3

Below is Best Array:

1	2	3	4	5	6	7	8
0	7	5	10	35	9999	38	17

Entering findMinNode() method!
Leaving findMinNode(): minNode is 2
Below is ToDo Array:

1	2	3	4	5	6	7	8
0	0	0	1	1	1	1	1

Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	3	1	1	3	3

Below is Best Array:

1	2	3	4	5	6	7	8
0	7	5	10	35	9999	38	17

childNode is 1
childNode is 2
childNode is 3
childNode is 4
childNode is 5
childNode is 6
childNode is 7
childNode is 8
newCost:9 < Best[8]:17
Update Parent Array and Best Array:
Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	3	1	1	3	2

Below is Best Array:

1	2	3	4	5	6	7	8
0	7	5	10	35	9999	38	9

Entering findMinNode() method!
Leaving findMinNode(): minNode is 8
Below is ToDo Array:

1	2	3	4	5	6	7	8
0	0	0	1	1	1	1	0

Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	3	1	1	3	2

Below is Best Array:

1	2	3	4	5	6	7	8
0	7	5	10	35	9999	38	9

childNode is 1

childNode is 2
childNode is 3
childNode is 4
childNode is 5
childNode is 6
newCost:19 < Best[6]:9999
Update Parent Array and Best Array:
Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	3	1	8	3	2

Below is Best Array:

1	2	3	4	5	6	7	8
0	7	5	10	35	19	38	9

childNode is 7
newCost:11 < Best[7]:38
Update Parent Array and Best Array:
Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	3	1	8	8	2

Below is Best Array:

1	2	3	4	5	6	7	8
0	7	5	10	35	19	11	9

childNode is 8
Entering findMinNode() method!
Leaving findMinNode(): minNode is 4
Below is ToDo Array:

1	2	3	4	5	6	7	8
0	0	0	0	1	1	1	0

Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	3	1	8	8	2

Below is Best Array:

1	2	3	4	5	6	7	8
0	7	5	10	35	19	11	9

childNode is 1
childNode is 2
childNode is 3
childNode is 4
childNode is 5
newCost:16 < Best[5]:35
Update Parent Array and Best Array:
Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	3	4	8	8	2

Below is Best Array:

1	2	3	4	5	6	7	8

0 7 5 10 16 19 11 9
childNode is 6
newCost:13 < Best[6]:19
Update Parent Array and Best Array:
Below is Parent Array:
1 2 3 4 5 6 7 8
1 3 1 3 4 4 8 2
Below is Best Array:
1 2 3 4 5 6 7 8
0 7 5 10 16 13 11 9
childNode is 7
childNode is 8
Entering findMinNode() method!
Leaving findMinNode(): minNode is 7
Below is ToDo Array:
1 2 3 4 5 6 7 8
0 0 0 0 1 1 0 0
Below is Parent Array:
1 2 3 4 5 6 7 8
1 3 1 3 4 4 8 2
Below is Best Array:
1 2 3 4 5 6 7 8
0 7 5 10 16 13 11 9
childNode is 1
childNode is 2
childNode is 3
childNode is 4
childNode is 5
childNode is 6
childNode is 7
childNode is 8
Entering findMinNode() method!
Leaving findMinNode(): minNode is 6
Below is ToDo Array:
1 2 3 4 5 6 7 8
0 0 0 0 1 0 0 0
Below is Parent Array:
1 2 3 4 5 6 7 8
1 3 1 3 4 4 8 2
Below is Best Array:
1 2 3 4 5 6 7 8
0 7 5 10 16 13 11 9
childNode is 1
childNode is 2
childNode is 3
childNode is 4

childNode is 5
childNode is 6
childNode is 7
childNode is 8
Entering findMinNode() method!
Leaving findMinNode(): minNode is 5
Below is ToDo Array:

1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0

Below is Parent Array:

1	2	3	4	5	6	7	8
1	3	1	3	4	4	8	2

Below is Best Array:

1	2	3	4	5	6	7	8
0	7	5	10	16	13	11	9

childNode is 1
childNode is 2
childNode is 3
childNode is 4
childNode is 5
childNode is 6
childNode is 7
childNode is 8
Source Node is 2
Entering findMinNode() method!
Leaving findMinNode(): minNode is 8
Below is ToDo Array:

1	2	3	4	5	6	7	8
1	0	1	1	1	1	1	0

Below is Parent Array:

1	2	3	4	5	6	7	8
2	2	2	2	2	2	2	2

Below is Best Array:

1	2	3	4	5	6	7	8
22	0	30	9999	9999	9999	52	2

childNode is 1
newCost:8 < Best[1]:22
Update Parent Array and Best Array:
Below is Parent Array:

1	2	3	4	5	6	7	8
8	2	2	2	2	2	2	2

Below is Best Array:

1	2	3	4	5	6	7	8
8	0	30	9999	9999	9999	52	2

childNode is 2
childNode is 3