CSC2103: Data Structures and Algorithms

Group Members

No.	Name	StudentID
1.	Chua Yao Xian	20042552
2.	Lee Jia Qian	19117613
3.	Mohamed Rafshan Nashid	20076162
4.	Tan Boon Wee	19122381

Table of Content

Introduction

For starters, the problems that we choose to discuss are the shortest path finding and binary search tree. Discovery of the shortest path from the source node to each of the other nodes in the issue of shortest path finding. We had chosen Dijkstra's algorithm to find the shortest path as our preferred method which involves each edge with a length, which is the distance between the node at one end of the edge and the node at the other end. The algorithm's basic purpose is to find the shortest path between a beginning node and the rest of the graph. The choice of data structure that we choose to use is graphs, an adjacency list is an array of linked lists that depicts a graph, because we only need to keep the values for the edges, an adjacency list is efficient in terms of storage when using the Dijkstra Algorithm. The purpose of choosing the Dijkstra algorithm which is the Dijkstra algorithm's main premise is to continuously eliminate longer pathways between the beginning node and all feasible destinations.

For our program, we have used two unique sets of nodes which are settled and unsettled to maintain track of the process. Furthermore, settled nodes have a known minimum distance from the source. The unsettled nodes collection contains nodes that can be reached from the source but whose minimum distance from the starting node is unknown. Besides, the steps of how the algorithm works. Firstly, the start node distance was set to zero, and all the other distances should be set to infinity. Second, we add the start node to the list of unsettled nodes, while the set of unsettled nodes is not empty, one of the nodes will be chosen as an evaluation node from the set. The evaluation node should have the shortest distance to the source. Moreover, it will calculate new distances to direct neighbors while keeping the shortest distance in mind at all times, then add unresolved neighbors to the set of unsettled nodes and keep looping until the algorithm found the shortest path and print the result to user from the source node to other each node (Baeldung, 2022).

Moving on to the second problem is the binary search tree. A binary Search Tree is known as BST which is used to store keys in nodes in a way that allows for efficient searching, insertion, and deletion in an ordered array. Binary search is used when we need to access or edit a collection while keeping the order of its members intact. It is useful in any case where a node may be compared in a less than or greater than method. BST is a tree structure with a maximum of 2 children for each node, in which any child node to the left is less than the parent node and any child node to the right

is more than the parent node. As a result, the complexity of lookup, insertion and removal operations is O(log n). This is because, as we traverse the tree from root to leaf, we can reject half of the tree at each step based on whether the input value is higher or less than the one in the current node. The choice of data structure that we chose to use is the tree. This is because we wanted to store keys in a natural form of a hierarchy that allows more efficient searching. (Baeldung, 2021).

The tool that we will be using for the two problems is Replit website as it allows all the groupmates to code together. The programming language that we will be using is Java.

Problem 1 implementation (Shortest Path Finding – Dijkstra's Algorithm)

The first problem that we have chosen is Dijkstra Algorithm. Dijkstra Algorithm is used to find the shortest pathway

```
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Map.Entry;
import java.util.Set;
import java.util.stream.Collectors;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
```

(Import necessary Java Libraries.)

```
public class Dijkstra {
  static Scanner input = new Scanner(System.in);

  public static void calculateShortestPathFromSource(Node source) {
    // to calculate the shortest path from the source node
    source.setDistance(0); // to set distance

    Set<Node> settledNodes = new HashSet<>();
    Set<Node> unsettledNodes = new HashSet<>();
    unsettledNodes.add(source);// unsettled node will be added to unsettled nodes hash set

    while (unsettledNodes.size() != 0) { // loop while the size for unsettle nodes is not 0
      Node currentNode = getLowestDistanceNode(unsettledNodes);
      // get lowest distance node from unsettled nodes hash set
      unsettledNodes.remove(currentNode); // remove node
      for (Entry<Node, Integer> adjacencyPair : currentNode.getAdjacentNodes().entrySet()) {
        // get adjacent nodes and input it to Node and
        // get entry set and input it to integer
        Node adjacentNode = adjacencyPair.getKey(); // adjacent pair will get key and set it adjacent node
        Integer edgeWeight = adjacencyPair.getValue(); // adjacent pair will get value and set edge weight

        if (!settledNodes.contains(adjacentNode)) { // if settle node does not have adjacent node
          // calculate the minimun distance by using adjacent node, edge weight,and current node
          CalculateMinimumDistance(adjacentNode, edgeWeight, currentNode);
          unsettledNodes.add(adjacentNode); // add adjacent node to unsettled nodes hash set
        }
      }
      settledNodes.add(currentNode); // add current node to settled nodes hash set
    }
  }
```

(This code is used to calculate the shortest path.)

```
public static void welcome() {// print welcome
  System.out.println();
  System.out.println("Welcome To Dijkstra Algorithm!");
  System.out.println();
  System.out.println("Please Enter a MINIMUM of 5 and a MAXIMUM of 10 Different Node Name: ");
  System.out.println("For Empty Node Please Enter '-'");
}
```

(This code is used to print a welcome message when the user runs the Dijkstra Algorithm.)

```
public static void select() { // print menu
  System.out.println();
  System.out.println("Please Select:");
  System.out.println("1 - Continue");
  System.out.println("2 - Result");
  System.out.println("0 - Exit");
}
```

(This code is used to print a menu.)

```
public static void CalculateMinimumDistance(Node evaluationNode, Integer edgeWeight, Node sourceNode) {
  Integer sourceDistance = sourceNode.getDistance(); // get distance from source node and assigned to source distance
  // if distance from evaluation node is greater than the total amount of source distance and edge weight
  if (sourceDistance + edgeWeight < evaluationNode.getDistance()) {
    // set distance from evaluation node with the total amount of source distance
    // and edge weight
    evaluationNode.setDistance(sourceDistance + edgeWeight);
    // insert shortest path from source node and insert it to the shortest path
    // linked list
    LinkedList<Node> shortestPath = new LinkedList<>(sourceNode.getShortestPath());
    shortestPath.add(sourceNode); // source node is added to shortest path
    evaluationNode.setShortestPath(shortestPath); // shortest path is set to evaluation node
  }
}
```

(This code is used to calculate the minimum distance by obtaining data from the evaluationNode, edgeWeight and sourceNode.)

```
public static Node getLowestDistanceNode(Set<Node> unsettledNodes) {
  Node lowestDistanceNode = null; // lowest distance node is set to null
  int lowestDistance = Integer.MAX_VALUE; // Integer.MAX_VALUE is used to initialise all node distances, to create an
nfinite distance.
  for (Node node : unsettledNodes) { // for loop for unsettle node
    int nodeDistance = node.getDistance(); // get distance from node, set it to nodeDistance
    if (nodeDistance < lowestDistance) { // if node distance lesser than lowest distance
      lowestDistance = nodeDistance; // lowest distance equal to node distance
      lowestDistanceNode = node; // lowest distance node equal to node
    }
  }
  return lowestDistanceNode;
}
```

(This code is used to get the lowest distance node from unsettledNode that is set into the Node Linked List.)

```
static class Graph {
  public Set<Node> nodes = new HashSet<>();

  public void addNode(Node nodeA) {
    // add node A to node
    nodes.add(nodeA);
  }

  public Set<Node> getNodes() { // get node from Node set
    return nodes;
  }

  public void setNodes(Set<Node> nodes) { // set nodes from Node set
    this.nodes = nodes;
  }
}
```

(This code is used to add node, get node, and set node into Node Linked List.)

```
static class Node {
  public String name;
  public List<Node> shortestPath = new LinkedList<>(); // create shortest path linked list
  public Integer distance = Integer.MAX_VALUE; // Integer.MAX_VALUE is used to initialise all node distances, to create an
infinite distance.
  public Map<Node, Integer> adjacentNodes = new HashMap<>(); // create hash map that allow Node and Integer to be inserted

  public Node(String name) { // set Node Name
    this.name = name;
  }

  public void addDestination(Node destination, int distance) {
    // obtain node destination and distance value to add destination to adjacent nodes hash map
    adjacentNodes.put(destination, distance);
  }

  public String getName() { // get node name
    return name;
  }

  public void setName(String name) { // set node name
    this.name = name;
  }

  public Map<Node, Integer> getAdjacentNodes() { // get adjacent node and input it to hash map
    return adjacentNodes;
  }

  public void setAdjacentNodes(Map<Node, Integer> adjacentNodes) {
    // get adjacent node from hash map and set adjacentNode
    this.adjacentNodes = adjacentNodes;
  }

  public Integer getDistance() { // get distance as integer
    return distance;
  }
```

```
  public void setDistance(Integer distance) { // get and set distance
    this.distance = distance;
  }

  public List<Node> getShortestPath() { // get shortest path from Node Linked List
    return shortestPath;
  }

  public void setShortestPath(LinkedList<Node> shortestPath) {
    // get shortest path from node linked list and set it to the shortest path
    this.shortestPath = shortestPath;
  }

}
```

(A node has a name, a LinkedList referencing the shortestPath, a distance from the source, and an adjacency list named adjacentNodes.）

```
public static void printPaths(List<Node> nodes) { // print path from Node Linked List
  nodes.forEach(node -> {
    String path = node.getShortestPath().stream()
        .map(Node::getName)
        .collect(Collectors.joining(" -> "));
    System.out.println((path.isBlank()
        ? "%s : %s".formatted(node.getName(), node.getDistance())
        : "%s -> %s : %s".formatted(path, node.getName(), node.getDistance())) // print data from get name, get distance
    );
  });
}
```

(This code is used to print out path that is stored in Node Linked List.)

```java
// introduction for dijkstra algorithm
public static void runD() {
  try {
    welcome();
    System.out.println("Enter Node Name Number 1:");
    String node1 = input.next(); // Allow user to input Node 1 Name
    Dijkstra.Node n1 = new Dijkstra.Node(node1);

    System.out.println("Enter Node Name Number 2:");
    String node2 = input.next();
    Dijkstra.Node n2 = new Dijkstra.Node(node2);

    System.out.println("Enter Node Name Number 3:");
    String node3 = input.next();
    Dijkstra.Node n3 = new Dijkstra.Node(node3);

    System.out.println("Enter Node Name Number 4:");
    String node4 = input.next();
    Dijkstra.Node n4 = new Dijkstra.Node(node4);

    System.out.println("Enter Node Name Number 5:");
    String node5 = input.next();
    Dijkstra.Node n5 = new Dijkstra.Node(node5);

    System.out.println("Enter Node Name Number 6:");
    String node6 = input.next();
    Dijkstra.Node n6 = new Dijkstra.Node(node6);

    System.out.println("Enter Node Name Number 7:");
    String node7 = input.next();
    Dijkstra.Node n7 = new Dijkstra.Node(node7);

    System.out.println("Enter Node Name Number 8:");
    String node8 = input.next();
    Dijkstra.Node n8 = new Dijkstra.Node(node8);
```

```java
    System.out.println("Enter Node Name Number 9:");
    String node9 = input.next();
    Dijkstra.Node n9 = new Dijkstra.Node(node9);

    System.out.println("Enter Node Name Number 10:");
    String node10 = input.next();
    Dijkstra.Node n10 = new Dijkstra.Node(node10);
```

```java
  } catch (Exception e) {
    System.out.println("Something went wrong.");
  } finally {
    System.out.println("The program is terminated.");
  }
```

(This code is used to run Dijkstra's Algorithm in the main driver to allow user to input the node name as well as a try catch is implemented to make sure user input a valid input else program will be terminated.)

```java
// while loop for error handling when user input an invalid input
main: while (true) {
  System.out.println();
  System.out.println("Select A From Node to add Destination: ");
  System.out.println("Node 1 = " + node1);
  System.out.println("Node 2 = " + node2);
  System.out.println("Node 3 = " + node3);
  System.out.println("Node 4 = " + node4);
  System.out.println("Node 5 = " + node5);
  System.out.println("Node 6 = " + node6);
  System.out.println("Node 7 = " + node7);
  System.out.println("Node 8 = " + node8);
  System.out.println("Node 9 = " + node9);
  System.out.println("Node 10 = " + node10);
  System.out.println("For Example: Enter 1 To Select Node 1");
  int sn = input.nextInt();
```

(This code is used to show the user what node name that they had input followed by an example on how to select a from node.)

```
if (sn == 1) { // if user select 1 it will continue the code
  System.out.println();
  System.out.println("Select A 'TO' Node To Add Destination To Node: " + sn);
  int add = input.nextInt();
```

(This code is used to check the user input if user input is 1 the code will be continued.)

```
if (add == 2) {
  System.out.println();
  System.out.println("Enter Distance From Node 1 to Node 2:");
  int d2 = input.nextInt();
  n1.addDestination(n2, d2);
} else if (add == 3) {
  System.out.println();
  System.out.println("Enter Distance From Node 1 to Node 3:");
  int d3 = input.nextInt();
  n1.addDestination(n3, d3);
} else if (add == 4) {
  System.out.println();
  System.out.println("Enter Distance From Node 1 to Node 4:");
  int d4 = input.nextInt();
  n1.addDestination(n4, d4);
} else if (add == 5) {
  System.out.println();
  System.out.println("Enter Distance From Node 1 to Node 5:");
  int d5 = input.nextInt();
  n1.addDestination(n5, d5);
} else if (add == 6) {
  System.out.println();
  System.out.println("Enter Distance From Node 1 to Node 6:");
  int d6 = input.nextInt();
  n1.addDestination(n6, d6);
} else if (add == 7) {
  System.out.println();
  System.out.println("Enter Distance From Node 1 to Node 7:");
  int d7 = input.nextInt();
  n1.addDestination(n7, d7);
} else if (add == 8) {
  System.out.println();
  System.out.println("Enter Distance From Node 1 to Node 8:");
  int d8 = input.nextInt();
  n1.addDestination(n8, d8);
```

```
} else if (add == 9) {
  System.out.println();
  System.out.println("Enter Distance From Node 1 to Node 9:");
  int d9 = input.nextInt();
  n1.addDestination(n9, d9);
} else if (add == 10) {
  System.out.println();
  System.out.println("Enter Distance From Node 1 to Node 10:");
  int d10 = input.nextInt();
  n1.addDestination(n10, d10);
} else {
  // if user input an invalid code it will prompt user invalid input and will loop again
  System.out.println("Invaid Input");
  System.out.println("Please Select A Valid Node!");
  continue;
}
```

(This code is used to check what number the user entered. If user input 2 the code in the if statement 2 will be prompt to the user so on and so forth.)

```
select: while (true) {
  select();
  int userRequest = input.nextInt();
  if (userRequest == 0) {
    System.exit(0);
  } else if (userRequest == 1) {
    continue main;
  } else if (userRequest == 2) {
    System.out.println("Amount of Nodes You Enter: ");
    int nodeAmount = input.nextInt();
    System.out.println("Please Enter Source Node: ");
    System.out.println("Node 1 = " + node1);
    System.out.println("Node 2 = " + node2);
    System.out.println("Node 3 = " + node3);
    System.out.println("Node 4 = " + node4);
    System.out.println("Node 5 = " + node5);
    System.out.println("Node 6 = " + node6);
    System.out.println("Node 7 = " + node7);
    System.out.println("Node 8 = " + node8);
    System.out.println("Node 9 = " + node9);
    System.out.println("Node 10 = " + node10);

    if ((nodeAmount == 5) || (nodeAmount == 6) || (nodeAmount == 7) || (nodeAmount == 8) || (nodeAmount == 9)
        || (nodeAmount == 10)) {
      switch (nodeAmount) {
        case 5:
          Dijkstra.calculateShortestPathFromSource(n1);
          Dijkstra.printPaths(Arrays.asList(n1, n2, n3, n4, n5));
          continue select;
        case 6:
          Dijkstra.calculateShortestPathFromSource(n1);
          Dijkstra.printPaths(Arrays.asList(n1, n2, n3, n4, n5, n6));
          continue select;
        case 7:
          Dijkstra.calculateShortestPathFromSource(n1);
          Dijkstra.printPaths(Arrays.asList(n1, n2, n3, n4, n5, n6, n7));
```

```
        case 8:
          Dijkstra.calculateShortestPathFromSource(n1);
          Dijkstra.printPaths(Arrays.asList(n1, n2, n3, n4, n5, n6, n7, n8));
          continue select;
        case 9:
          Dijkstra.calculateShortestPathFromSource(n1);
          Dijkstra.printPaths(Arrays.asList(n1, n2, n3, n4, n5, n6, n7, n8, n9));
          continue select;
        case 10:
          Dijkstra.calculateShortestPathFromSource(n1);
          Dijkstra.printPaths(Arrays.asList(n1, n2, n3, n4, n5, n6, n7, n8, n9, n10));
          continue select;
      }
    } else { // if user input an invalid code it will prompt user invalid input and will loop again
      System.out.println("Invaid Input!");
      System.out.println("Please Select A Valid Source Node!");
      continue;
    }
```

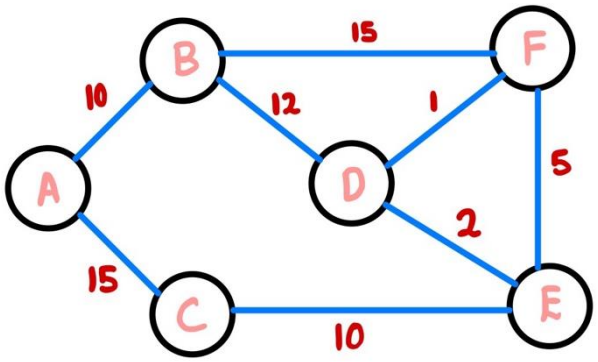
(This code allows user selection to perform a certain task. If user input 0 the system will exit and terminate the program. If user input 2 it will require user to input the number of nodes that they had entered and prompt user, the node that they had entered and print path. If user input other than 0,1 or 2 an error will be prompt to the user.)

```
// if user input 1,2,3,4,5,6,7,8,9 or 10 it will continue the code
if ((sn == 1) || (sn == 2) || (sn == 3) || (sn == 4) || (sn == 5)
    || (sn == 6) || (sn == 7) || (sn == 8) || (sn == 9) || (sn == 10)) {
  if (sn == 1) { // if user select 1 it will continue the code
    System.out.println();
    System.out.println("Select A 'TO' Node To Add Destination To Node: " + sn);
    int add = input.nextInt();
```

```
    } else { // if user input an invalid it will prompt user invalid input and will loop again
      System.out.println("Invaid Input!");
      continue select;
    }
  }
} else { // if user input an invalid it will prompt user invalid input and will loop again
  System.out.println("Invaid Input!");
  System.out.println("Please Select A Valid Node!");
  continue;
}
}
```

(This code is used to perform error handling if the user had entered an invalid input.)

Diagram 1 (6 Node)



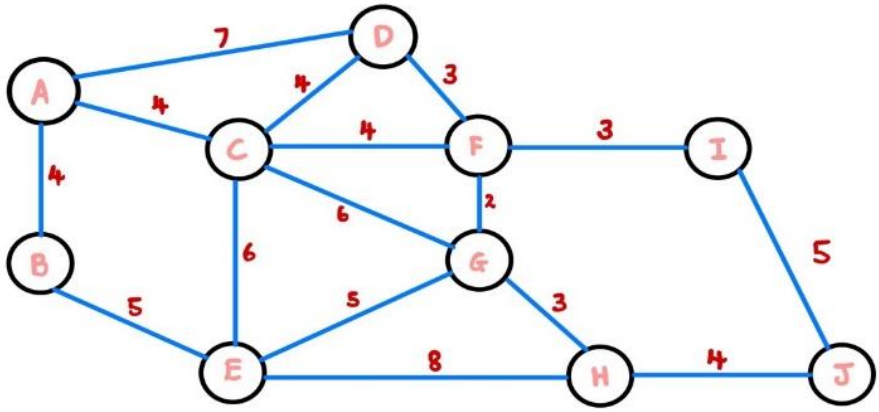
Results

```
Please Enter a MINIMUM of 5 and a MAXIMUM of 10 Different Node Name:
For Empty Node Please Enter '-'
Enter Node Name Number 1:
A
Enter Node Name Number 2:
B
Enter Node Name Number 3:
C
Enter Node Name Number 4:
D
Enter Node Name Number 5:
E
Enter Node Name Number 6:
F
Enter Node Name Number 7:
-
Enter Node Name Number 8:
-
Enter Node Name Number 9:
-
Enter Node Name Number 10:
-
```

```
Please Select:
1 - Continue
2 - Result
0 - Exit
2
Amount of Nodes You Enter:
6
Please Enter Source Node:
Node 1 = A
Node 2 = B
Node 3 = C
Node 4 = D
Node 5 = E
Node 6 = F
Node 7 = -
Node 8 = -
Node 9 = -
Node 10 = -
A : 0
A -> B : 10
A -> C : 15
A -> B -> D : 22
A -> B -> D -> E : 24
A -> B -> D -> F : 23

Please Select:
1 - Continue
2 - Result
0 - Exit
```

Diagram 2 (10 Node)



Results

```
Please Enter a MINIMUM of 5 and a MAXIMUM of 10 Different
Node Name:
For Empty Node Please Enter '-'
Enter Node Name Number 1:
A
Enter Node Name Number 2:
B
Enter Node Name Number 3:
C
Enter Node Name Number 4:
D
Enter Node Name Number 5:
E
Enter Node Name Number 6:
F
Enter Node Name Number 7:
G
Enter Node Name Number 8:
H
Enter Node Name Number 9:
I
Enter Node Name Number 10:
J
```

```
Please Select:
1 - Continue
2 - Result
0 - Exit
2
Amount of Nodes You Enter:
10
Please Enter Source Node:
Node 1 = A
Node 2 = B
Node 3 = C
Node 4 = D
Node 5 = E
Node 6 = F
Node 7 = G
Node 8 = H
Node 9 = I
Node 10 = J
A : 0
A -> B : 4
A -> C : 4
A -> D : 7
A -> B -> E : 9
A -> C -> F : 8
A -> C -> G : 10
A -> C -> G -> H : 13
A -> C -> F -> I : 11
A -> C -> F -> I -> J : 16
```

Steps On How to Enter Node:

1. Enter 5 - 10 different node name

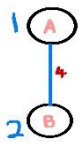
   If Enter 5 Nodes (Empty node will be entered with a "-" dash/hyphen)

```
Please Enter a MINIMUM of 5 and a MAXIMUM of 10 Different Node Name:
For Empty Node Please Enter '-'
Enter Node Name Number 1:
A
Enter Node Name Number 2:
B
Enter Node Name Number 3:
C
Enter Node Name Number 4:
D
Enter Node Name Number 5:
E
Enter Node Name Number 6:
-
Enter Node Name Number 7:
-
Enter Node Name Number 8:
-
Enter Node Name Number 9:
-
Enter Node Name Number 10:
-
```

   If Enter 10 Node

```
Please Enter a MINIMUM of 5 and a MAXIMUM of 10 Different
Node Name:
For Empty Node Please Enter '-'
Enter Node Name Number 1:
A
Enter Node Name Number 2:
B
Enter Node Name Number 3:
C
Enter Node Name Number 4:
D
Enter Node Name Number 5:
E
Enter Node Name Number 6:
F
Enter Node Name Number 7:
G
Enter Node Name Number 8:
H
Enter Node Name Number 9:
I
Enter Node Name Number 10:
J
```

2. Select a from node




```
Select A From Node to add Destination:
Node 1 = A
Node 2 = B
Node 3 = C
Node 4 = D
Node 5 = E
Node 6 = F
Node 7 = G
Node 8 = H
Node 9 = I
Node 10 = J
For Example: Enter 1 To Select Node 1
1
```

(In this case, we wanted to input the destination from Node A to Node B. Node A is known as

Node 1 and Node B is Node 2. Therefore, when selecting a from node we will enter 1 which is referring to Node A.)

3. Select a ‘TO’ Node

```
Select A 'TO' Node To Add Destination To Node: 1
2
```

A ‘TO’ node which is 2 that refers to Node B.

4. Enter the distance

```
Enter Distance From Node 1 to Node 2:
4
```

Lastly, enter the distance From Node A to Node B which is 4.

Limitations

The limitation for Dijkstra’s algorithms code is that we are unable to allow users to select which node they wanted to start. Therefore, the user is required to enter the source node in Node 1.

```
A : 0
A -> B : 4
A -> C : 4
A -> D : 7
A -> B -> E : 9
A -> C -> F : 8
A -> C -> G : 10
A -> C -> G -> H : 13
A -> C -> F -> I : 11
A -> C -> F -> I -> J : 16
```

Not to mention, the code also requires more time to run as the code is quite lengthy compared to Binary Search Tree Algorithm.

Moreover, Dijkstra’s algorithms do not work for graphs with negative weight cycles. This can the third limitation which is our code will allow negative weight cycles to show a correct answer but is the wrong answer.

Problem 2 implementation (Binary Search Tree)

The second problem that we have chosen is binary search tree. Binary search tree is used to enable efficient searching, insertion, and deletion by storing the keys in the nodes in a specific method.

```
import java.util.Scanner;
```

(Import necessary Java libraries.)

```
public void addNode(int key, String name) {
  Node newNode = new Node(key, name);
  if (root == null) {// If there is no root this becomes root
    root = newNode;
  } else {
    //Set root as the Node when traverse the tree
    Node selectedNode = root;
    Node parent;
    while (true) {
      // root is the top parent
      parent = selectedNode;
      if (key < selectedNode.key) { // the node is added to the left
        // select left child
        selectedNode = selectedNode.leftChild;
        if (selectedNode == null) {// If the left child has no children
          // then place the new node on the left
          parent.leftChild = newNode;
          return;
        }
      } else { // the node is added to the right
        selectedNode = selectedNode.rightChild;
        if (selectedNode == null) {// If the right child has no children
          // then place the new node on the right
          parent.rightChild = newNode;
          return;
        }
      }
    }
  }
}
```

(This is to initialize new node object when adding node.)

```
public static void traversal() { // print menu
  System.out.println();
  System.out.println("Please a Number to Select Method of Displaying:");
  System.out.println("1 - Inorder Traversal");
  System.out.println("2 - Preorder Traversal");
  System.out.println("3 - Postorder Traversal");
  System.out.println("4 - Search");
  System.out.println("0 - End");
}
```

(This code is used to prompt user the menu.)

```
public void inOrderTraverseTree(Node selectedNode) {
  if (selectedNode != null) {
    // Traverse the left node
    inOrderTraverseTree(selectedNode.leftChild);
    // Visit the currently focused on node
    System.out.println(selectedNode);
    // Traverse the right node
    inOrderTraverseTree(selectedNode.rightChild);
  }
}
```

(This code is used to get selected node and arrange it in Inorder Traverse Tree.)

```
public void preorderTraverseTree(Node selectedNode) {
  if (selectedNode != null) {
    System.out.println(selectedNode);
    preorderTraverseTree(selectedNode.leftChild);
    preorderTraverseTree(selectedNode.rightChild);
  }
}
```

(This code is used to get selected node and arrange it in Preorder Traverse Tree.)

```
public void postOrderTraverseTree(Node selectedNode) {
  if (selectedNode != null) {
    postOrderTraverseTree(selectedNode.leftChild);
    postOrderTraverseTree(selectedNode.rightChild);
    System.out.println(selectedNode);
  }
}
```

(This code is used to get selected node and arrange it in Postorder Traverse Tree.)

```
public Node findNode(int key) {
  // Start at the top of the tree
  Node selectedNode = root;
  while (selectedNode.key != key) {// While Node not found, keep looping
    if (key < selectedNode.key) {
      // Shift the selected Node to the left child
      selectedNode = selectedNode.leftChild;
    } else {
      // Shift the selected Node to the right child
      selectedNode = selectedNode.rightChild;
    }
    if (selectedNode == null){// The node not found
      return null;
    }
  }
  return selectedNode;
}
```

(This code is used to find node by using the key that is entered by the user and loop if the user input

null into the key. The if statement will make sure that the selected node key is greater than key and will put it to the left child. If the selected node is less than key, it will be the right to the parent node.)

```
public boolean remove(int key) {
  // Start at the top of the tree
  Node selectedNode = root;
  Node parent = root;

  // This will tell us whether to search to the right or left
  boolean isItALeftChild = true;

  while (selectedNode.key != key) {// While Node not found, keep looping
    parent = selectedNode;

    if (key < selectedNode.key) {// If less selected Node should search to the left
      isItALeftChild = true;
      // Shift the selected Node to the left child
      selectedNode = selectedNode.leftChild;
    } else {// Greater than selected Node so go to the right
      isItALeftChild = false;
      // Shift the selected Node to the right child
      selectedNode = selectedNode.rightChild;
    }
    if (selectedNode == null){// The node wasn't found
      return false;
    }
  }
  if (selectedNode.leftChild == null && selectedNode.rightChild == null) {// If Node doesn't have children delete it
    if (selectedNode == root){ // If root delete it
      root = null;
    }else if (isItALeftChild){// If it was marked as a left child of the parent delete it in its parent
      parent.leftChild = null;
    }else{// If it was marked as a right child of the parent delete it in its parent
      parent.rightChild = null;
    }
  }
```

```
  else if (selectedNode.rightChild == null) { // If no right child
    if (selectedNode == root){
      root = selectedNode.leftChild;
    }else if (isItALeftChild){// If selected Node was on the left of parent move the selected Node left child up to the parent node
      parent.leftChild = selectedNode.leftChild;
    }else{// If selected Node was on the right of parent move the selected Node left child up to the parent node
      parent.rightChild = selectedNode.leftChild;
    }
  }
  else if (selectedNode.leftChild == null) { // If no left child
    if (selectedNode == root){
      root = selectedNode.rightChild;
    }else if (isItALeftChild){// If selected Node was on the left of parent move the selected Node right child up to the parent
ode
      parent.leftChild = selectedNode.rightChild;
    }else{// If selected Node was on the right of parent move the selected Node right child up to the parent node
      parent.rightChild = selectedNode.rightChild;
    }
  }
  // Two children, find the deleted nodes replacement
  else {
    Node replacement = getReplacementNode(selectedNode);

    if (selectedNode == root){// If the selectedNode is root replace root with the replacement
      root = replacement;
    }else if (isItALeftChild){// If the deleted node was a left child make the replacement the left child
      parent.leftChild = replacement;
    }else{// If the deleted node was a right child make the replacement the right child
      parent.rightChild = replacement;
      replacement.leftChild = selectedNode.leftChild;
    }
  }
  return true;
}
```

(This code is used to remove a node. There are several processes that need to be done before a node

can be removed. The code will need to search for the node and identify the position of the child node. After identifying the position of the child node, it identifies whether there are children within the node. If there is no child node the parent node will be deleted. Not to mention, it will also verify is there a left child and right child. After verifying it will also replace the position of the node.)

```
public Node getReplacementNode(Node replacedNode) {
  Node replacementParent = replacedNode;
  Node replacement = replacedNode;
  Node selectedNode = replacedNode.rightChild;

  while (selectedNode != null) {// While there are no more left children
    replacementParent = replacement;
    replacement = selectedNode;
    selectedNode = selectedNode.leftChild;
  }
  // If the replacement isn't the right child move the replacement into the parents
  // leftChild slot and move the replaced nodes
  // right child into the replacements rightChild
  if (replacement != replacedNode.rightChild) {
    replacementParent.leftChild = replacement.rightChild;
    replacement.rightChild = replacedNode.rightChild;
  }
  return replacement;
}
```

(This code is used to get replacement node from replaced node and it will identify whether the right child had replaced the parent node. If the replacement node is not the right child but the left child, the left child will be replacing the parent, the right child will remain as the right child.)

```
public static void runBST() {
  try{
  BinarySearchTree BST = new BinarySearchTree();
  System.out.println("Welcome To Binary Search Tree!");
  System.out.println();
  System.out.println("Please Enter Number of Nodes:");
        int node = input.nextInt();
        for(int num=1; num<node+1; num++) {
          System.out.println();
          System.out.println("Please add node " + num + ":");
          System.out.println("Please enter key for node " + num + ":");
          int key = input.nextInt();
          System.out.println("Please enter name for node " + num + ":");
          String name = input.next();
          BST.addNode(key, name);
        }
```

(This code is used to prompt user welcome message and allow user input.)

```
        while(true) {
          traversal();
            int method = input.nextInt();
                if (method == 1) {
                  BST.inOrderTraverseTree(BST.root);
                }else if (method == 2) {
                  BST.preorderTraverseTree(BST.root);
                }else if (method == 3 ) {
                  BST.postOrderTraverseTree(BST.root);
                }else if(method == 4) {
                  System.out.println();
                  System.out.println("Please Enter Node Number to Search: ");
                  int searchNode = input.nextInt();
                  System.out.println(BST.findNode(searchNode));
                }else if (method == 0 ) {
                  System.out.println();
                  System.out.println("Programme Ended! Thank You!");
                  break;
                } else{
                  System.out.println("Invalid Input!");
                }
            }
}catch(Exception e){
    System.out.println("Something went wrong.");
    System.out.println("The program is terminated.");
}
}
```

(This code is used to encounter Inorder, Preorder, and Postorder traverse tree)

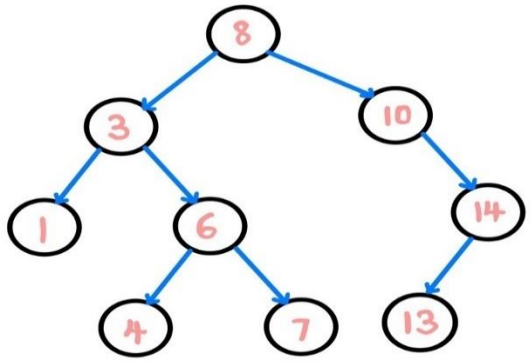
```
class Node {
  int key;
  String name;
  Node leftChild;
  Node rightChild;

  Node(int key, String name) {
    this.key = key;
    this.name = name;
  }

  public String toString( ) {
    return name + " has the key " + key;
  }
  }
}
```

(This code is used to create node that store key and name. This code also allows it to print names and keys.)

Diagram



Result

1) Inorder Traversal

```
Please a Number to Select Method of Displaying:
1 - Inorder Traversal
2 - Preorder Traversal
3 - Postorder Traversal
4 - Search
0 - End
1
C has the key 1
B has the key 3
E has the key 4
D has the key 6
F has the key 7
A has the key 8
G has the key 10
I has the key 13
H has the key 14
```

2) Preorder Traversal

```
Please a Number to Select Method of Displaying:
1 - Inorder Traversal
2 - Preorder Traversal
3 - Postorder Traversal
4 - Search
0 - End
2
A has the key 8
B has the key 3
C has the key 1
D has the key 6
E has the key 4
F has the key 7
G has the key 10
H has the key 14
I has the key 13
```

3) Postorder Traversal

```
Please a Number to Select Method of Displaying:
1 - Inorder Traversal
2 - Preorder Traversal
3 - Postorder Traversal
4 - Search
0 - End
3
C has the key 1
E has the key 4
F has the key 7
D has the key 6
B has the key 3
I has the key 13
H has the key 14
G has the key 10
A has the key 8
```

Steps On How to Enter Node:

1. Enter the number of nodes that you wish to enter

```
Welcome To Binary Search Tree!

Please Enter Number of Nodes:
9
```

2. Enter the nodes key and nodes name

```
Please enter key for node 1:
8
Please enter name for node 1:
A

Please add node 2:
Please enter key for node 2:
3
Please enter name for node 2:
B

Please add node 3:
Please enter key for node 3:
1
Please enter name for node 3:
C

Please add node 4:
Please enter key for node 4:
6
Please enter name for node 4:
D

Please add node 5:
Please enter key for node 5:
4
Please enter name for node 5:
E

Please add node 6:
Please enter key for node 6:
7
Please enter name for node 6:
F

Please add node 7:
Please enter key for node 7:
10
Please enter name for node 7:
G
```

```
Please add node 8:
Please enter key for node 8:
14
Please enter name for node 8:
H

Please add node 9:
Please enter key for node 9:
13
Please enter name for node 9:
I
```

3. Enter the method you wish the result to be displayed

```
Please a Number to Select Method of Displaying:
1 - Inorder Traversal
2 - Preorder Traversal
3 - Postorder Traversal
4 - Search
0 - End
```

4. Search

```
Please a Number to Select Method of Displaying:
1 - Inorder Traversal
2 - Preorder Traversal
3 - Postorder Traversal
4 - Search
0 - End
4

Please Enter Node Number to Search:
14
H has the key 14
```

5. End

```
Please a Number to Select Method of Displaying:
1 - Inorder Traversal
2 - Preorder Traversal
3 - Postorder Traversal
4 - Search
0 - End
0

Programme Ended! Thank You!
```

Limitation

The limitation for Binary Search Tree is that its shape might degenerate and depends on the order of insertions. The second limitation is that each node's key must be compared to the element's key when inserting or searching for an element. Long keys may cause the run time to significantly increase. The third limitation is that it retrieves elements significantly more slowly than an array. The last limitation is that it will increase the complexity when BST becomes unbalanced or degenerate.

Main Class

```
import java.util.Scanner;

public class Main {
  static Scanner input = new Scanner(System.in);

  public static void menu() {// print menu
    System.out.println("Welcome to Dijkstra Algorithm and Binary Search Tree Generator!");
    System.out.println("Please Enter a Number to Select:");
    System.out.println("1 - Dijkstra Algorithm");
    System.out.println("2 - Binary Search Tree:");
  }

  public static void main(String args[]) {
    menu();
    int x = input.nextInt();
    if(x == 1) {
      Dijkstra D = new Dijkstra();
      D.runD();
    }else if(x ==2){
      BinarySearchTree BST = new BinarySearchTree();
      BST.runBST();
    }else {
      System.out.println("Invalid input");
      System.out.println("-------------------------------------------------------------------------");
      System.out.println();
      menu();
    }
  }
}
```

(This code is used to allow the user to choose which algorithm the user wishes to run. If the user enters 1 will run the Dijkstra Algorithm code. If the user enters 2 it will run the Binary Search Tree code. If the user enters other than 1 or 2 an error will be prompted to the user.)

Contribution

Name	Contribution
Chua Yao Xian **20042552**	**Documentation and Code**
Lee Jia Qian **19117613**	**Documentation and Code**
Mohamed Rafshan Nashid **20076162**	**Documentation and Code**
Tan Boon Wee **19122381**	**Documentation and Code**

Challenges

Most of our challenges in building the code were in the algorithm structure, such as presenting the application in console to illustrate algorithm in a meaningful way. Our target was to minimize the code and present the results. This includes taking in user inputs relevant to how the algorithm works. We spent significant resources while managing the code Dijkstra's Algorithm for this reason. Since Dijkstra's Algorithm calculates the shortest path by the edge values between the nodes, extrapolating user input for nodes and edges was a problem.

While doing Dijkstra's Algorithms our initial plan was to create a user input that allows users to input how many nodes they wish to input. When we were halfway through the code we noticed the way, we loop the code will lead us to the same node being overwritten every time it loops. Therefore, we were stuck for quite some time and decided to manually code each part.

Though the Dijkstra's problem existed, we moved forward to build the Binary Search Tree Algorithm (BST). Conversely for BST, the code structure was fairly easy. It takes user inputs as a stream and asks the user how the result to be presented with options of pre-order, in-order or post-order. We have built this algorithm so that it can perform Searching and Deletion.

After building the BST we held consecutive meetings to exhaust all efforts to solve the issues for Dijkstra's Algorithm. The user inputs were crucial in building the logic for the algorithm as these values have to be allocated to the algorithm's edge and node values. We came up with two solutions to continuously fetch values from user or to take multidimension-matrix values from user. With careful consideration, we chose the latter, to take in user inputs separately for edges and nodes.

The problem we face while doing these two algorithms is that we tried using an if-else statement for error handling but to no avail as the variable located in the if statement is unreachable for the other code. Therefore, we ended up with a loop whenever a user enters an invalid input it will keep looping until the user enters a valid input.

Conclusion

This project is the result of meticulous and diligent effort among teammates. Dijkstra's Algorithm is used to discover the graph's shortest route between a given node and all other nodes. The code we created is able to allow user to enter a maximum of 10 nodes and distances between each node to find the shortest path. Moreover, Binary Search Tree Algorithm implementation of a priority queue with x amount of internal nodes, each of which stores a single element. The code we have created is able to allow the user to enter the number of nodes that they wish to enter and generate a binary search tree that shows the user 3 different traversals such as Inorder Traversal, Preorder Traversal, and Postorder Traversal. The code that we have crafted took careful consideration for a minimal code base and avoided unstructured code to produce meaningful results. It is critical to have Binary Seach Tree in a balanced approach, or else data ordering and search would not occur in a logical way, incurring additional expenses for incorporating Binary Search Tree in the data flow. Also, being cautious when inputting data because it impacts the structure of BST and hence data search. Our teammates took upon individual responsibilities for delegating tasks by our strengths. In this approach, we managed to deliver each component of the code in a timely fashion to bring the console application altogether.

Reference

Baeldung. (2022). *Dijkstra Shortest Path Algorithm in Java.* Retrieved from https://www.baeldung.com/java-dijkstra

https://www.youtube.com/watch?v=BuvKtCh0SKk

Baeldung. (2021). *A quick Guide to Binary Search Trees.* Retrieved from https://www.baeldung.com/cs/binary-search-trees#:~:text=Simply%20put%2C%20a%20binary%20search,iterate%20them%20in%20sorted%20order.

Banas, D. (2013). *Binary Trees in Java 2.* Retrieved from https://www.newthinktank.com/2013/03/binary-trees-in-java-2/

https://www.youtube.com/watch?v=UcOxGmj45AA

GeeksforGeeks. (2022). *Applications, Advantages and Disadvantages of Binary Search Tree.* Retrieved from https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-search-tree/

GeeksforGeeks. (2022). *Dijsktra's algorithm*. Retrieved from

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/