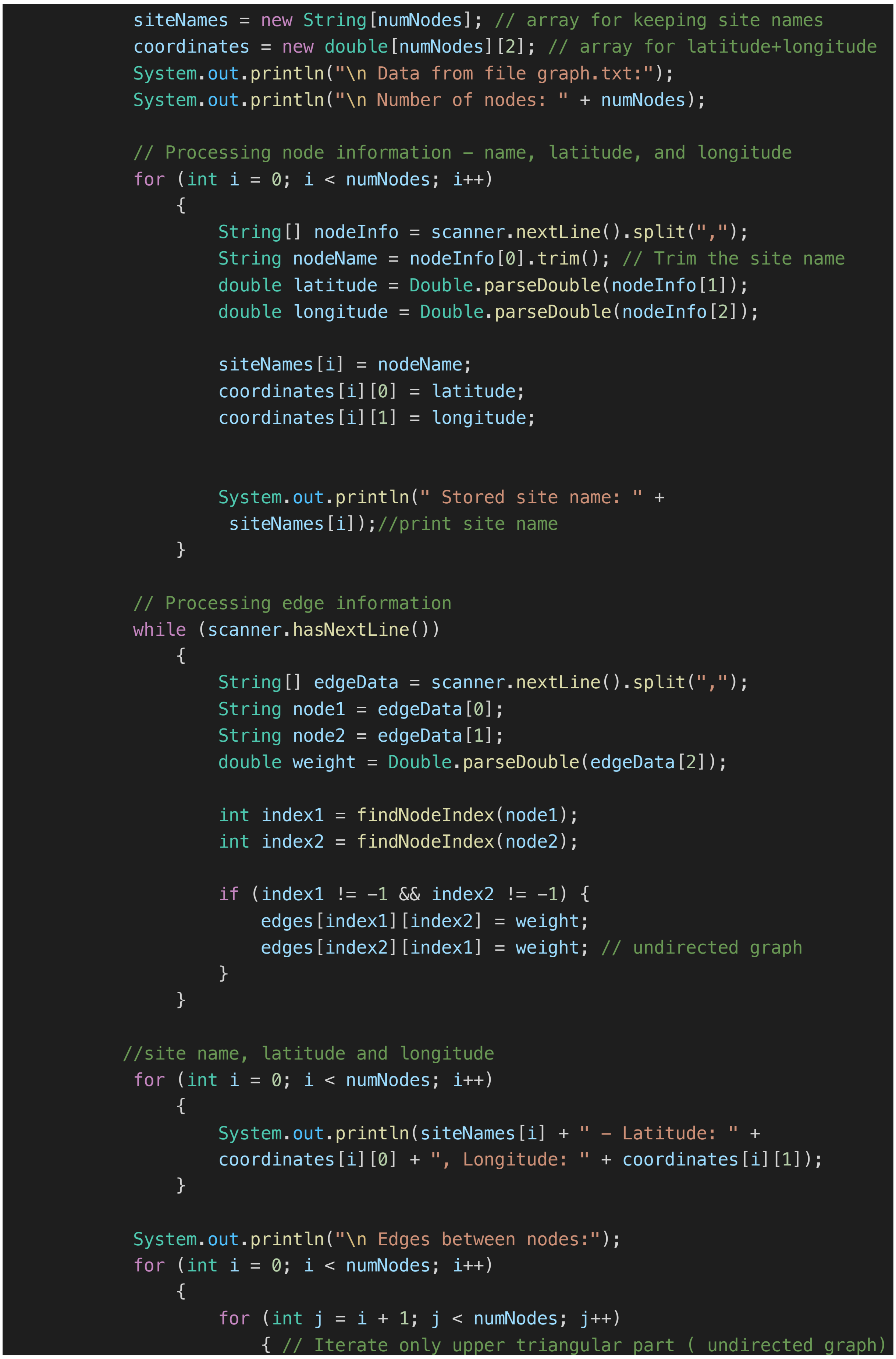
```java
/*Walking tour of Paris which gives the user the options to:
 open +input a graph from an external file(graph.txt),
 search for a site, insert and search for edges,
 find all sites connected to given site and find closest site to given site
 */

/*Group: Abigail Murray (C00260073), Ryan Dunne (C00263405) */

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.InputMismatchException;
import java.util.NoSuchElementException;



public class ParisWalkingTour
{
    private static double[][] edges; // Adjacency matrix to store edges
    private static String[] siteNames; // array to store site names
    private static double[][] coordinates; //to store latitude and longitude

    public static void main(String[] args)
    {
        readDataFromFile(); // Initialize the data from the file

        displayMenu(); // method to display menu options
    }
    private static void displayMenu() {
        Scanner scanner = new Scanner(System.in);

        while (true)//loop to display menu options
        {
            System.out.println("");
            System.out.println("-------------------------------------------");
            System.out.println(" Welcome to the Walking Tour of Paris!");
            System.out.println("-------------------------------------------");
            System.out.println("Select an option (1-7):");
            System.out.println("1. Open and Input a graph from a file :");
            System.out.println("2. Search for a site ");
            System.out.println("3. Insert an edge");
            System.out.println("4. Search for an edge");
            System.out.println("5. Enter a site name to search
                                  for sites connected to it");
            System.out.println("6. Enter a site and find the closest site to it");
            System.out.println("7. Exit");
            System.out.println(" ");

            int choice;
            try
            {
                System.out.print("Enter your choice: ");
                choice = Integer.parseInt(scanner.nextLine());
            }
            catch (NumberFormatException e) // If the input is not an integer
            {
                System.out.println("Invalid input. Please enter a valid integer.");
                continue; // Restart the loop to prompt for input again
            }


            switch (choice)//switch statement to execute the menu options
            {
                case 1: System.out.println("Option 1: Open and input a graph from
                        an external file");
                        System.out.println("Enter the file name: ");
                        String fileName = scanner.nextLine();
                        // Read the file name as a string

                       if (fileName.equalsIgnoreCase("graph.txt"))
                       {
                            System.out.println("");
                            System.out.println("File loaded successfully!");
                            System.out.println("");
                            readDataFromFile();
                       }
                       else
                       {
                            System.out.println("");
                            System.out.println("File not found.
                            Hint: Try the file name graph.txt");
                       }
                       break;

                case 2: System.out.println("Option 2: Search for a site ");
                        System.out.println("Enter a site name: ");
                        String siteName = scanner.nextLine().trim();
                        int siteIndex = findNodeIndex(siteName);

                        if (siteIndex != -1)//if site found
                        {
                            System.out.println("Site found!");
                            System.out.println(siteNames[siteIndex] + " - Latitude:
"                           + coordinates[siteIndex][0] +
                            ", Longitude: " + coordinates[siteIndex][1]);
                        }
                        else
                        {
                             System.out.println("The site you entered is not

                             included in this walking tour");
                        }
```

```
                    break;

            case 3:
                System.out.println("Option 3: Insert an Edge ");
                // Enter the first site name
                System.out.println("Enter first site name: ");
                String siteName1 = scanner.nextLine().trim();//Trim removes
                spaces before and after the string to avoid errors
                int siteIndex1 = findNodeIndex(siteName1);

                // Enter the second site name
                System.out.println("Enter second site name: ");
                String siteName2 = scanner.nextLine().trim();
                int siteIndex2 = findNodeIndex(siteName2);

                if (siteIndex1 == -1)//if site not found
                {
                    System.out.println("First site not found.");
                    break;
                }
                else if (siteIndex2 == -1)
                {
                    System.out.println("Second site not found.");
                    break;
                }

                // Check if the edge already exists
                if (edges[siteIndex1][siteIndex2] > 0)
                {
                     System.out.println("There is already an edge between "
                     + siteName1 + " and " + siteName2 + " with distance " +
                     edges[siteIndex1][siteIndex2]);
                     break;
                }
                Else
                {
                    System.out.println("There is no existing edge between " +
                    siteName1 + " and " + siteName2);
                    // Enter the distance
                    double newDistance;
                    try
                    {
                        System.out.println("Enter the distance: ");
                        newDistance = scanner.nextDouble();
                    }
                    catch (InputMismatchException e)//if input is not a double
                    {
                        System.out.println("Invalid input for distance.
                         Please enter a valid number.");
                        scanner.nextLine();// Consume the newline character
                        left by the previous nextDouble()
```

```java
                break;
            }

        edges[siteIndex1][siteIndex2] = newDistance;
        edges[siteIndex2][siteIndex1] = newDistance; //an undirected
                                                    //graph

        System.out.println("Edge between " + siteName1 + " and " +
        siteName2 + " with distance " + newDistance + " inserted
        successfully.");
        scanner.nextLine(); // Consume the newline character
        }
        break;

    case 4:
        System.out.println("Option 4: Search for an Edge");
        System.out.println("Enter first site name: ");
        String searchSite1 = scanner.nextLine(); // Read the first site
                                                 //name as a string
        int searchSiteIndex1 = findNodeIndex(searchSite1);
        //Consume the newline character left by the previous nextLine()
        scanner.nextLine();

        System.out.println("Enter second site name: ");
        String searchSite2 = scanner.nextLine();
        int searchSiteIndex2 = findNodeIndex(searchSite2);

        if (searchSiteIndex1 != -1 && searchSiteIndex2 != -1)
        {
           if (edges[searchSiteIndex1][searchSiteIndex2] > 0)
           {
              System.out.println("There is an edge between " +
              searchSite1 + " and " + searchSite2 + " with distance " +
              edges[searchSiteIndex1][searchSiteIndex2]);
           }
           else
           {
              System.out.println("There is no edge between " +
              searchSite1 + " and " + searchSite2);
            }
         }

        else
        {
            System.out.println("One or both of the specified sites
            not found.");
       }
        break;

    case 5:
        System.out.println("Option 5: Enter a site name to display all
                    connected sites");
                    System.out.println("Enter site name: ");
                    siteName = scanner.nextLine(); //Read the site name as a string
                    siteIndex = findNodeIndex(siteName);
                    if (siteIndex == -1) //error handling: if site is not found
                    {   System.out.println("Error: Site not found.");
                        break;
                    }
                    int j = 0;
                    for(int i = 0; i < siteNames.length; i++)
                    {
                        if(edges[siteIndex][i] != 0) // If there is an edge...
                        {
                            System.out.print(edges[siteIndex][i] + " : " +
                            siteNames[i]); //Print Edge & Site Name
                            while(j < 2)
                            {
                                System.out.print(" " + coordinates[i][j] + " ");
                                //Prints the coords for the site
                                j++;
                            }
                            j = 0;
                        }

                    }
                    break;

                case 6:
                    System.out.println("Option 6: Enter a site + Display closest
                    site ");
                    System.out.println("Enter site name: ");
                    siteName = scanner.nextLine(); // Read site name as a string
                    siteIndex = findNodeIndex(siteName);
                    double closestEdge = Double.MAX_VALUE;
                    String closestSite = "";
                    int closestSiteIndex = 0;
                    j = 0;
                    siteIndex = findNodeIndex(siteName);

                    if (siteIndex == -1) {//error handling: site is not found
                        System.out.println("Error: Site not found.");
                        break;
                    }
                    for(int i = 0; i < siteNames.length; i++)
                    {
                        if(edges[siteIndex][i] != 0) //If there is an edge...
                        {


     if(edges[siteIndex][i] < closestEdge) // The current edge is less than the
```

```
                                             closest edge
    {
            closestEdge = edges[siteIndex][i]; //Saves closest edge
            closestSite = siteNames[i]; //Saves corresponding site name
            closestSiteIndex = i; //Saves the current index for use outside of
                                    loop
     }

   }
}


            System.out.print(closestEdge + " : " + closestSite + " - ");
            //Prints closest edge & Site name
                  while(j < 2)
                  {
                      System.out.print(coordinates[closestSiteIndex][j] + " " );
                      //Prints coordinates
                      j++;
                  }

                  break;


     case 7: System.out.println("Exiting...");

             scanner.close();

             return;

     default:
              System.out.println("Invalid choice!");
          }
      }
  }
  //method to read data from file
  private static void readDataFromFile() {
      try {
          File file = new File("graph.txt");
          Scanner scanner = new Scanner(file);
          int numNodes = Integer.parseInt(scanner.nextLine());
          edges = new double[numNodes][numNodes];
         // Initialize the adjacency matrix

          // Initialize the edges array with 0 weights
          for (int i = 0; i < numNodes; i++)
              {
                  for (int j = 0; j < numNodes; j++)
                      {
                          edges[i][j] = 0.0;
                      }
              }
```



```
 siteNames = new String[numNodes]; // array for keeping site names
 coordinates = new double[numNodes][2]; // array for latitude+longitude
 System.out.println("\n Data from file graph.txt:");
 System.out.println("\n Number of nodes: " + numNodes);

 // Processing node information – name, latitude, and longitude
 for (int i = 0; i < numNodes; i++)
   {
     String[] nodeInfo = scanner.nextLine().split(",");
     String nodeName = nodeInfo[0].trim(); // Trim the site name
     double latitude = Double.parseDouble(nodeInfo[1]);
     double longitude = Double.parseDouble(nodeInfo[2]);

     siteNames[i] = nodeName;
     coordinates[i][0] = latitude;
     coordinates[i][1] = longitude;


     System.out.println(" Stored site name: " +
      siteNames[i]);//print site name
   }

 // Processing edge information
 while (scanner.hasNextLine())
   {
     String[] edgeData = scanner.nextLine().split(",");
     String node1 = edgeData[0];
     String node2 = edgeData[1];
     double weight = Double.parseDouble(edgeData[2]);

     int index1 = findNodeIndex(node1);
     int index2 = findNodeIndex(node2);

     if (index1 != -1 && index2 != -1) {
       edges[index1][index2] = weight;
       edges[index2][index1] = weight; // undirected graph
     }
   }

//site name, latitude and longitude
 for (int i = 0; i < numNodes; i++)
   {
     System.out.println(siteNames[i] + " – Latitude: " +
     coordinates[i][0] + ", Longitude: " + coordinates[i][1]);
   }

 System.out.println("\n Edges between nodes:");
 for (int i = 0; i < numNodes; i++)
   {
     for (int j = i + 1; j < numNodes; j++)
       { // Iterate only upper triangular part ( undirected graph)
```

```
                         if (edges[i][j] != 0.0)
                             {
                                 System.out.println(siteNames[i] + " <-> " +
                                      siteNames[j] + " : " + edges[i][j]);
                             }
                     }
             }

         System.out.println("\n Remaining lines in the file/ unprocessed
                            data:");
         while (scanner.hasNextLine())
             {
                 System.out.println(scanner.nextLine());
             }

         scanner.close();
         //error handling
     } catch (FileNotFoundException e) {
         System.out.println("File not found: " + e.getMessage());
     } catch (NoSuchElementException e) {
         System.out.println("Error reading the file: " + e.getMessage());
     } catch (NumberFormatException e) {
         System.out.println("Invalid number format in the file: " +
         e.getMessage());
     }
 }



 //method to find the index of a node
 private static int findNodeIndex(String nodeName)
     {
         for (int i = 0; i < siteNames.length; i++)
       // Iterate over the site names array
             {
                 String storedName = siteNames[i].trim();
                // Trim the stored name to remove leading and trailing spaces

                 if (storedName.equalsIgnoreCase(nodeName.trim()))
                // Compare the stored name with the input name
                     {
                         return i;
                     }
             }
         return -1; // Node not found
     }
}
```