BSc. Software Development
Year II

Project Report-CA3

Aaron Tierney & Ethan Payne
C00312018 & C00309151
7 December 2025

Contents

Project Introduction

This project serves as the end of semester submission for our Data Structures and Algorithms course. It presents our custom implementations of both hash tables and graph data structures, demonstrating a practical application of the concepts covered throughout the semester. This project was produced by Aaron Tierney and Ethan Payne.

Part 1: Hash Table Application

What is a Hash Table?

A hash table is a data structure that uses keys to store data in a specific location in an array. The keys are produced by using a hash function to store the data at a specific index.

Description of your Hash Table application

Our hash table takes a username of max 6 characters and converts them to a number by adding the ASCII values of each character. Using this number, we perform a mod 20 as our hash function on it and store it in a hash table of size 20. In the case of a collision occurring, we will be using linear probing.

Table of the Data used under the following headlines:

Username	ASCII Integer	Index after Hash function applied
LumiNo	76+117+109+105+78+111	596 % 20 = 16
Aer0Ny	65+101+114+48+78+121	527 % 20 = 7
S0lar1	83+48+108+97+114+49	499 % 20 = 19
K1nDer	75+49+110+68+101+114	517 % 20 = 17
Rave3n	82+97+118+101+51+110	559 % 20 = 19
V8xurA	86+56+120+117+114+65	558 % 20 = 18
NoXe1l	78+111+88+101+49+108	535 % 20 = 15
E1v3ri	69+49+118+51+114+105	506 % 20 = 6
CYm4ra	67+89+109+52+114+97	528 % 20 = 8
5yp4er	53+121+112+52+101+114	553 % 20 = 13

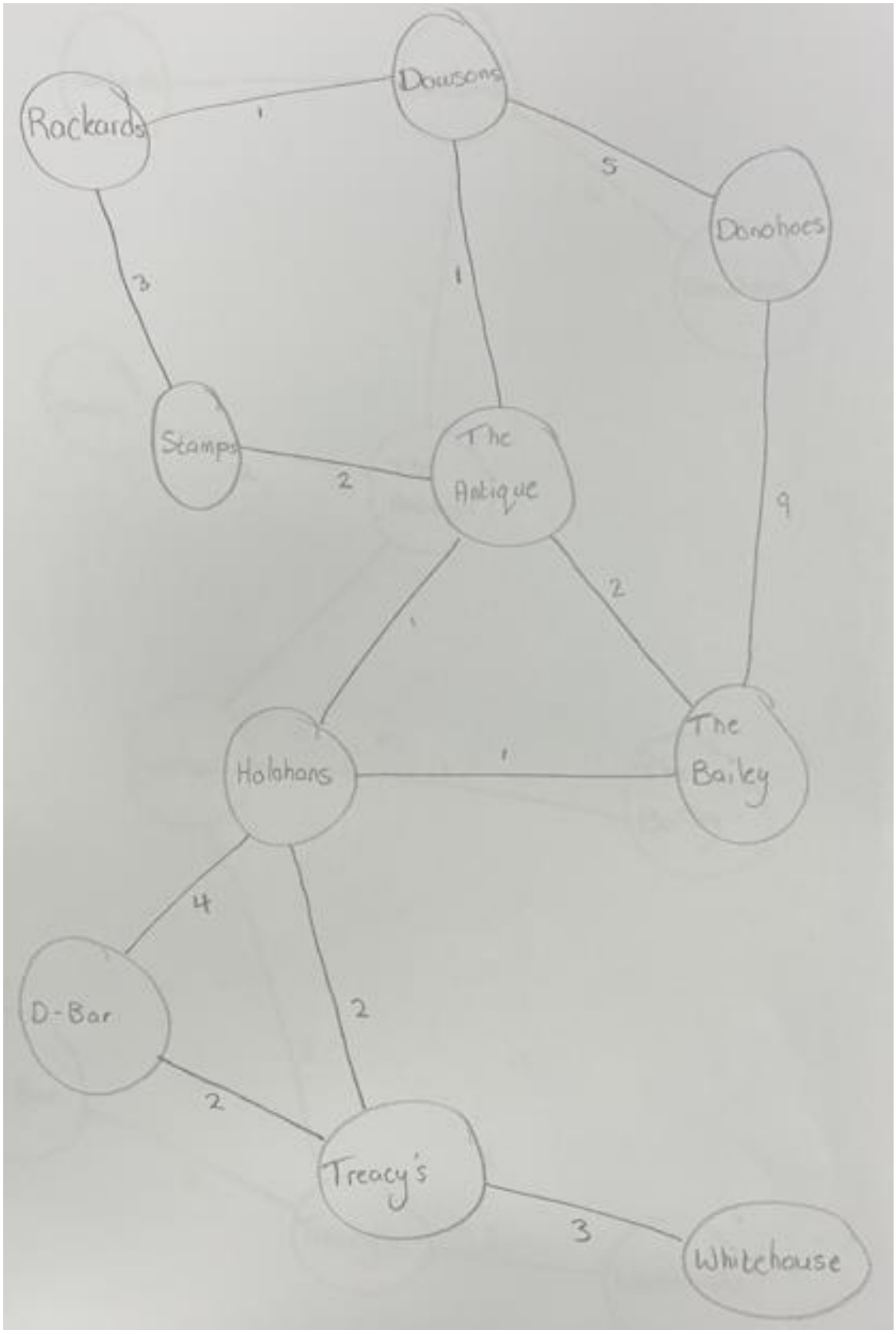
Diagram of the Hash Table produced- with collisions highlighted

Index	Usernames	Index	Usernames
0	Rave3n	10	
1		11	
2		12	
3		13	5yp4er
4		14	
5		15	NoXe1l
6	E1v3ri	16	LumiNo
7	Aer0Ny	17	K1nDer
8	Cym4ra	18	V8xurA
9		19	S0lar1

Usernames highlighted are to show that a collision has occurred.

Part 2: Graph Application

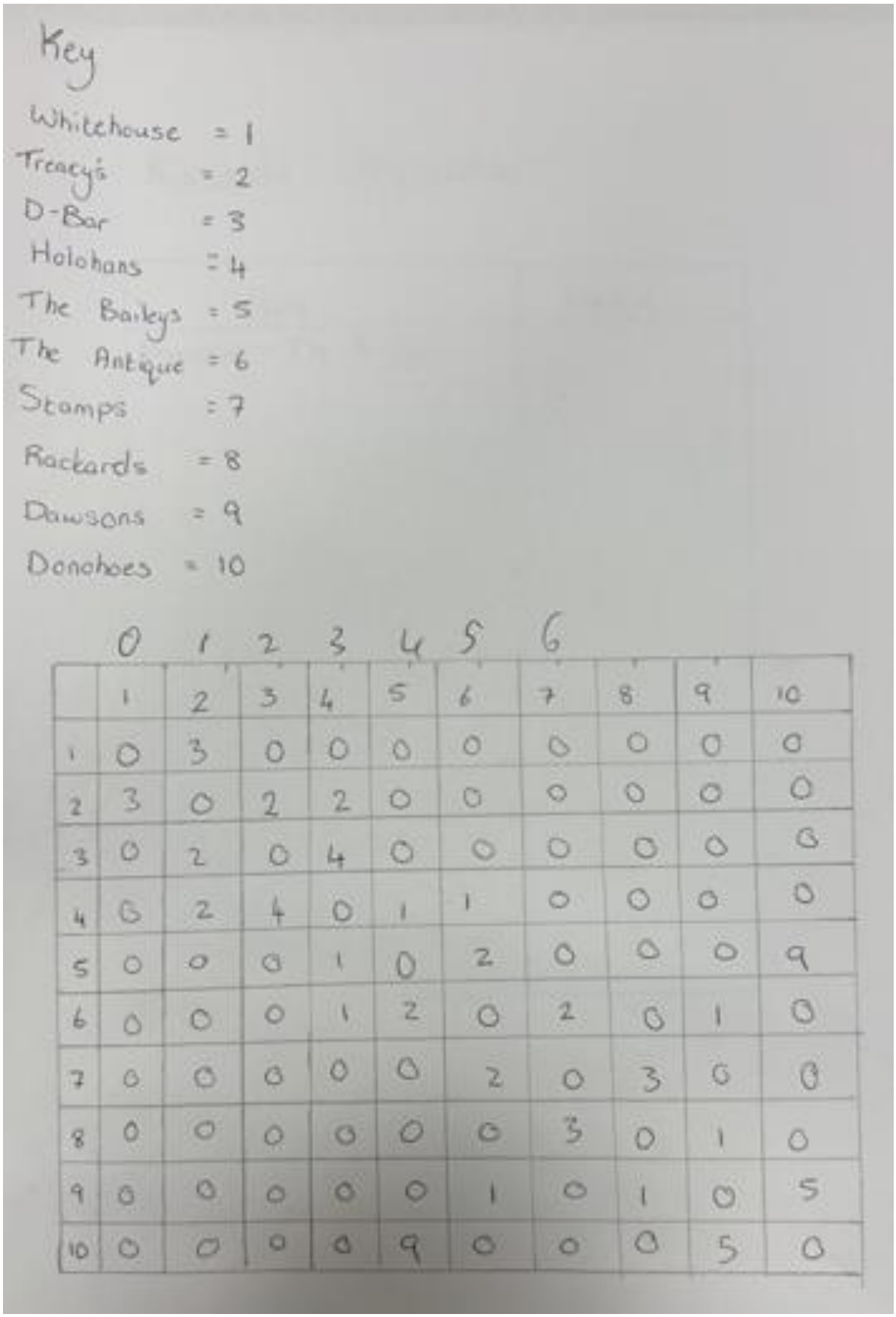
Diagram of Map of Tourist Sites



Description of Data Structure used to store your Map

To represent our graph, we used an adjacency matrix to store the edges along with their associated weights, and an array of objects to maintain information about each node. In the accompanying diagram, we introduced a key to minimise the space required in the 2D array. This key also clarifies how nodes correspond to their positions in the adjacency matrix, reflecting the way index values are used to reference nodes in our implementation

Diagram of the Map stored in an adjacency matrix

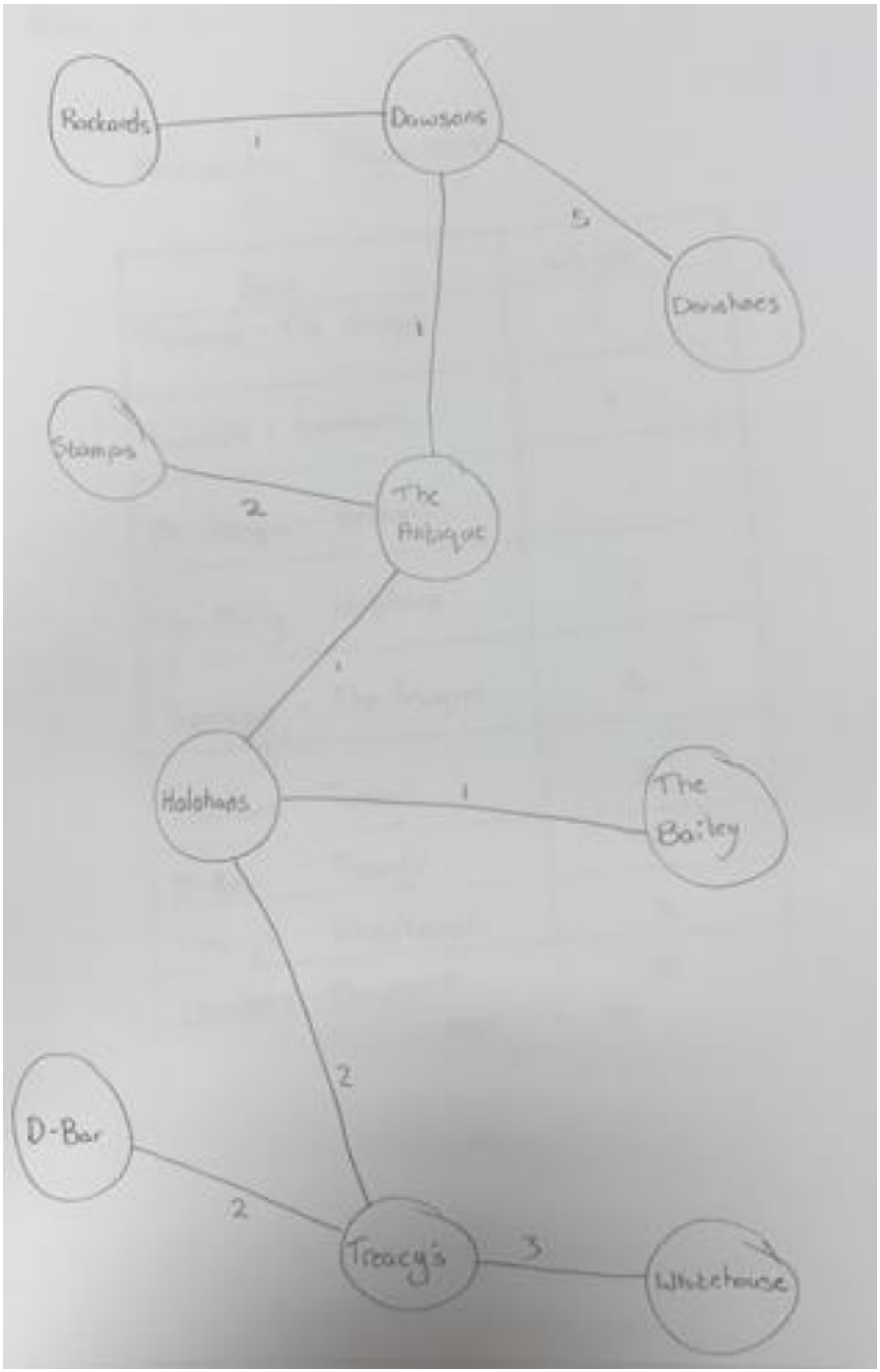


Key

Whitehouse = 1
Treacy's = 2
D-Bar = 3
Holohans = 4
The Baileys = 5
The Antique = 6
Stamps = 7
Rackards = 8
Dawsons = 9
Donohoes = 10

	0	1	2	3	4	5	6			
	1	2	3	4	5	6	7	8	9	10
1	0	3	0	0	0	0	0	0	0	0
2	3	0	2	2	0	0	0	0	0	0
3	0	2	0	4	0	0	0	0	0	0
4	0	2	4	0	1	1	0	0	0	0
5	0	0	0	1	0	2	0	0	0	9
6	0	0	0	1	2	0	2	0	1	0
7	0	0	0	0	0	2	0	3	0	0
8	0	0	0	0	0	0	3	0	1	0
9	0	0	0	0	0	1	0	1	0	5
10	0	0	0	0	9	0	0	0	5	0

Minimum Spanning Tree of your Graph



Showing the use of Kruskal's Algorithm

Kruskal's Algorithm

Edges	Weight
Dawsons - The Antique	1
Rackards - Dawsons	1
The Antique - Holohans	1
The Bailey - Holohans	1
Stamps - The Antique	2
Holohans - Treacy's	2
D-Bar - Treacy's	2
Treacy's - Whitehouse	3
Donohoes - Dawsons	5

MST = 18

Pseudocode of Algorithms

```
Search(site){
        for( i = 0; i < siteAmount; i ++){
                if( sitesObject[i].getName() == site){
                        return sitesObject[i].toString
                }
                Else{
                        Return “Site not found”
                }
        }
}
```

```
Insert(site1, site2, weight){
        Int site1Index = -1
        Int site2Index = -1
        for( i = 0; i < siteAmount; i ++){
                if( sitesObject[i].getName() == site1){
                        site1Index = i
                }
                Else if( sitesObject[i].getName() == site2){
                        Site2Index = i
                }
        }

        If(site1Index != -1 && site2Index != -1){
                If(site1Index != site2Index){
                        If( Edges[site1Index][site2Index] != 0){
                                Edges[site1Index][site2Index]
                                Edges[site2Index][site1Index]
                        }
                        Else{
                                Print “There is a connection already”
                        }
                Else{
                        Print “cant create a connection to same site”
                }
        }
        Else{
                Print “Sites not found”
        }
}
```

```
AllCons(site){
	Int rowIndex

	For( i = 0; i < siteAmount; i++){
		if( site == siteObjects[i].getName){
			rowIndex = i
		}
	}

	For(col = 0; col < siteAmount; col++){
		If( col != rowIndex && edges[rowIndex][col] != 0){
			Print siteObject[col].toString
		}
	}
}

Closest(site){
	Int smallestEdge = 10  // Set to 10 as our current highest edge is at 9
	Int smallestEdgeIndex;
	Int rowIndex

	For( i = 0; i < siteAmount; i++){
		if( site == siteObjects[i].getName){
			rowIndex = i
		}
	}

	For(col = 0; col < siteAmount; col++){
		If( edges[rowIndex][col] < smallestEdge){
			smallestEdgeIndex = col
		}
	}
	Return siteObject[smallestEdgeIndex]
}
```

Code

Menu()

DSA-Project > JavaCode > GraphDriver.java > GraphDriver

```java
import java.util.Scanner;

public class GraphDriver
{

    /**
     * Displays a menu of options and handles user input to perform graph operations
     * @param input Scanner for reading user input
     * @param siteAmount Number of sites in the graph
     * @param SitesArray Array containing all site objects
     * @param edges 2D array representing connections between sites with weights
     */
    public static void Menu(Scanner input, int siteAmount, Sites[] SitesArray, int[][] edges)
    {
        int menuChoice = 0;
        String site;
        String site1;
        String site2;
        int weight;

        System.out.println(x: "Menu:");
        System.out.println(x: "1. Search for a site");
        System.out.println(x: "2. Insert a connection between two sites");
        System.out.println(x: "3. Display all connections for a site");
        System.out.println(x: "4. Find the closest site to a given site");
        System.out.println(x: "5. Exit");

        menuChoice = input.nextInt();
        input.nextLine(); // Consume newline
```

```
30
31          switch(menuChoice)
32          {
33          case 1:
34              System.out.println(x: "Enter site name to search:");
35              site = input.nextLine();
36              System.out.println(Search(site, siteAmount, SitesArray));
37              break;
38          case 2:
39              System.out.println(x: "Enter site1 name:");
40              site1 = input.nextLine();
41              System.out.println(x: "Enter site2 name:");
42              site2 = input.nextLine();
43              System.out.println(x: "Enter weight:");
44              weight = input.nextInt();
45              Insert(site1, site2, weight, siteAmount, SitesArray, edges);
46              break;
47          case 3:
48              System.out.println(x: "Enter site name to display all connections:");
49              site = input.nextLine();
50              Allcons(site, siteAmount, SitesArray, edges);
51              break;
52          case 4:
53              System.out.println(x: "Enter site name to find the closest site:");
54              site = input.nextLine();
55              Closest(site, siteAmount, SitesArray, edges);
56              break;
57          case 5:
58              System.out.println(x: "Exiting the program.");
59              break;
60          default:
61              System.out.println(x: "Invalid choice. Please try again.");
62              break;
63          }
64      }
65
```

Search()

```
66
67      /**
68       * Searches for a specific site by name and returns its details
69       * @param site Name of the site to search for
70       * @param siteAmount Total number of sites in the array
71       * @param SitesArray Array containing all site objects
72       * @return String containing site details or "Site not found"
73       */
74      public static String Search(String site, int siteAmount, Sites[] SitesArray)
75      {
76          for(int i = 0; i < siteAmount; i++)
77          {
78              // Case-insensitive comparison to find the site
79              if(site.equalsIgnoreCase(SitesArray[i].getName()))
80              {
81                  return SitesArray[i].toString();
82              }
83          }
84          return "Site not found";
85      }
86
```

Insert()

```java
/**
 * Creates a connection between two sites with a specified weight
 * @param site1 Name of the first site
 * @param site2 Name of the second site
 * @param weight The weight/distance of the connection
 * @param siteAmount Total number of sites
 * @param SitesArray Array containing all site objects
 * @param edges 2D array to store the connection weights
 */
public static void Insert(String site1, String site2, int weight, int siteAmount, Sites[] SitesArray, int[][] edges)
{
    int site1Index = -1;
    int site2Index = -1;

    // Find the indices of both sites in the array
    for (int i = 0; i < siteAmount; i++)
    {

        if(SitesArray[i].getName().equalsIgnoreCase(site1))
        {
            site1Index = i;
        }
        else if(SitesArray[i].getName().equalsIgnoreCase(site2))
        {
            site2Index = i;
        }
    }

    // Check if sites are different and both exist
    if(site1Index != site2Index)
    {
        if(site1Index != -1 && site2Index != -1)
        {
            // Create the connection with the specified weight
            edges[site1Index][site2Index] = weight;
        }
        else
        {
            System.out.println(x: "Sites not found ");
        }
    }
    else
    {
        System.out.println(x: "Cant create a connection to same site");
    }
}
```

Allcons()

```java
    /**
     * Displays all connections for a specific site
     * @param site Name of the site to show connections for
     * @param siteAmount Total number of sites
     * @param SitesArray Array containing all site objects
     * @param edges 2D array containing connection weights
     */
    public static void Allcons(String site, int siteAmount, Sites[] SitesArray, int[][] edges)
    {
        int rowIndex = 0;

        for(int i = 0; i < siteAmount; i++)
        {
            if(site.equals(SitesArray[i].getName()))
            {
                rowIndex = i;
                break;
            }
        }

        // Check all possible connections from this site
        for(int col = 0; col < siteAmount; col++)
        {
            if(col != rowIndex && edges[rowIndex][col] != 0)
            {
                System.out.println(SitesArray[col].toString());
            }
        }
    }

```

Closest()

```java
/**
 * Finds and displays the closest connected site to a given site
 * @param site Name of the site to find closest neighbor for
 * @param siteAmount Total number of sites
 * @param SitesArray Array containing all site objects
 * @param edges 2D array containing connection weights
 */
public static void Closest(String site, int siteAmount, Sites[] SitesArray, int[][] edges)
{
    int rowIndex = 0;
    int minDist = Integer.MAX_VALUE;
    int closestIndex = -1;

    for(int i = 0; i < siteAmount; i++)
    {
        if(site.equals(SitesArray[i].getName()))
        {
            rowIndex = i;
            break;
        }
    }

    // Find the connection with minimum weight
    for(int col = 0; col < siteAmount; col++)
    {
        if(col != rowIndex && edges[rowIndex][col] != 0)
        {
            if(col != rowIndex && edges[rowIndex][col] != 0 && edges[rowIndex][col] < minDist)
            {
                minDist = edges[rowIndex][col];
                closestIndex = col;
            }
        }
    }
    System.out.println(SitesArray[closestIndex].toString());

}
```

Main driver

```
202     public static void main(String[] args)
203     {
204         final int siteAmount = 10;
205 
206         // Initialize data structures
207         Sites[] sitesArray = new Sites[siteAmount];
208         int[][] edges = new int[siteAmount][siteAmount]; // Adjacency matrix for graph
209         Scanner input = new Scanner(System.in);
210 
211         // Initialize all edges to 0 (no connection)
212         for(int row = 0; row < siteAmount; row++)
213         {
214             for(int col = 0; col < siteAmount; col++)
215             {
216                 edges[row][col] = 0;
217             }
218         }
219 
220         // Set up predefined connections between sites
221         edges[0][1] = 3;
222         edges[1][0] = 3;
223         edges[1][2] = 2;
224         edges[2][1] = 2;
225         edges[1][3] = 2;
226         edges[3][1] = 2;
227         edges[2][3] = 4;
228         edges[3][2] = 4;
229         edges[3][4] = 1;
230         edges[4][3] = 1;
231         edges[3][5] = 1;
232         edges[5][3] = 1;
233         edges[4][5] = 2;
234         edges[4][9] = 9;
235         edges[5][4] = 2;
236         edges[5][6] = 2;
237         edges[5][8] = 2;
238         edges[6][5] = 2;
239         edges[6][7] = 3;
240         edges[7][6] = 3;
241         edges[7][8] = 1;
242         edges[8][5] = 1;
243         edges[8][7] = 1;
244         edges[8][9] = 5;
245         edges[9][4] = 9;
246         edges[9][8] = 5;
```

```
248             // Create site objects with names and coordinates
249             sitesArray[0] = new Sites(name: "Whitehouse", x: 167, y: 98);
250             sitesArray[1] = new Sites(name: "Treacys", x: 201, y: 134);
251             sitesArray[2] = new Sites(name: "D-Bar", x: 89, y: 123);
252             sitesArray[3] = new Sites(name: "Holohans", x: 156, y: 78);
253             sitesArray[4] = new Sites(name: "The Baileys", x: 234, y: 167);
254             sitesArray[5] = new Sites(name: "The Antique", x: 189, y: 45);
255             sitesArray[6] = new Sites(name: "Stamps", x: 67, y: 234);
256             sitesArray[7] = new Sites(name: "Rackards", x: 45, y: 23);
257             sitesArray[8] = new Sites(name: "Dawsons", x: 78, y: 156);
258             sitesArray[9] = new Sites(name: "Donohoes", x: 123, y: 89);
259 
260             // Start the interactive menu
261             Menu(input, siteAmount, sitesArray, edges);
262 
263         }
264     }
```

Object to store nodes

```java
public class Sites
{
    String name;
    int x;
    int y;

    public Sites()
    {
        this.name = "";
        this.x = 0;
        this.y = 0;
    }

    public Sites(String name, int x, int y)
    {
        this.name = name;
        this.x = x;
        this.y = y;
    }

    public String getName()
    {
        return name;
    }

    public int getX()
    {
        return x;
    }

    public int getY()
    {
        return y;
    }

    @Override
    public String toString()
    {
        return name + " " + x + " " + y;
    }
}

```

Outputs for each method

Search()

```
Menu:
1. Search for a site
2. Insert a connection between two sites
3. Display all connections for a site
4. Find the closest site to a given site
5. Exit
1
Enter site name to search:
The Baileys
The Baileys 234 167
```

Insert()

```
Menu:
1. Search for a site
2. Insert a connection between two sites
3. Display all connections for a site
4. Find the closest site to a given site
5. Exit
2
Enter site1 name:
D-Bar
Enter site2 name:
Dawsons
Enter weight:
6
```

Allcons()

```
Menu:
1. Search for a site
2. Insert a connection between two sites
3. Display all connections for a site
4. Find the closest site to a given site
5. Exit
3
Enter site name to display all connections:
Holohans
Treacys 201 134
D-Bar 89 123
The Baileys 234 167
The Antique 189 45
```

Closest()

```
Menu:
1. Search for a site
2. Insert a connection between two sites
3. Display all connections for a site
4. Find the closest site to a given site
5. Exit
4
Enter site name to find the closest site:
Holohans
The Baileys 234 167
```

References

OpenAI (2025). *ChatGPT* (GPT-4o) [Large Language Model]. Available at: https://chat.openai.com/ [Accessed: 2 December 2025].

Microsoft. (2025) *Copilot*. [Generative AI]. Available at: https://copilot.microsoft.com/ (Accessed: 2 December 2025).