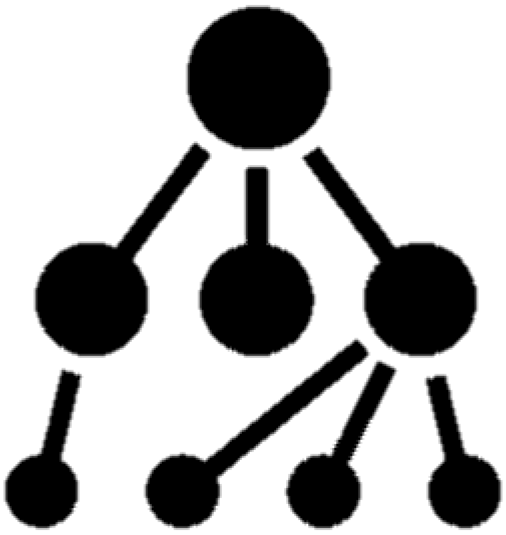
Data Structures & Algorithms – Final Report

A technical report covering the applications of hash tables & graphs.



By: Mark Lambert (C00192497), Darius Herdes (C00296548)

Course: Software Development (CW_KCSOF_B)

Submission Date: 10/12/2024

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Table of Contents Page

Introduction

This report wash commissioned by Áine Byrne as a final project for the Data Structures & Algorithms module. The purpose of the report is to display an understanding of the implementation of a hash application, a graph application including a coded protype to show said implementation. The report should also show one's ability to perform as part of a group dynamic.

Hash Table Application

What is a Hash Table?

A hash table is a data structure wherein a piece of information is stored in an encrypted manner after performing what's known as a "hash function" on some input. The hash function produces an index position for one single unit of information to be stored in the hash table, which is implemented as an array.

Application

The hash table application will take in a username (max characters: 6) and create a unique index for each username. The key is calculated by accumulating the total decimal ASCII value of the characters and performing a modulo of 20. In the case of a collision, a linear probe will be considered.

For this implementation, the final hash table will be of length 20 and the hash function will be % 20.

Example hash function equation

$$mlamb - 521 \ \% \ 20 = 1$$

Table of Data

The following table of data shows an example of 8 username inputs, some with uppercase characters.

Username	ASCII	Index (after hash function %20)
mlamb	521	1
darius	648	8
AdAm	339	19
hannah	662	2
Aaron	497	17
frank	529	9
james	528	8
daliah	611	11

Diagram of Hash Table Produced

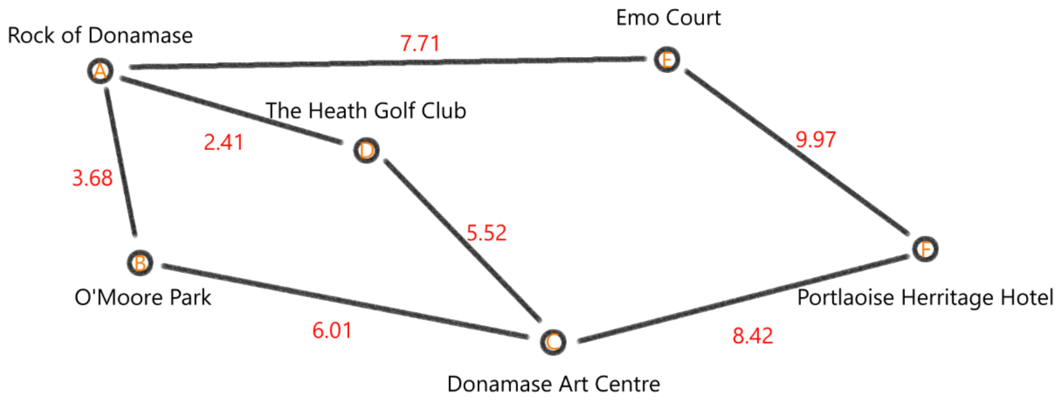
Below is a diagram visualising what the hash table will look like with each username in their respective index as well as any collisions which occurred.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Username		mlamb	hannah						darius	frank	james	daliah						Aaron		AdAm
Collision											2									

Graph Application

What is a Graph?

The image shows a graph connecting six points of interest in Portlaoise, labeled A to F, where nodes represent locations (e.g., A = Rock of Dunamase) and edges are labeled with distances in kilometers. Below the graph is an adjacency matrix, where rows and columns correspond to locations, and each cell shows the distance between two points. A value of 0 indicates no direct connection. The graph and matrix provide a clear visualization of the distances and connections between the locations.



	A	B	C	D	E	F
A	0	3.68	0	2.41	7.71	0
B	3.68	0	6.01	0	0	0
C	0	6.01	0	5.52	0	8.42
D	2.41	0	5.52	0	0	0
E	7.71	0	0	0	0	9.97
F	0	0	8.42	0	9.97	0

Note: The adjacency matrix is symmetrical via the diagonal as this graph is undirected.

Graph Implementation

What data structure? A 2D array of equal length rows as it has columns will be the chosen data structure to implement this application.

It should be noted that using a 2D array implies the use of an **adjacency matrix** versus the use of an **edge list.**

An **adjacency matrix** is a square matrix used to represent a graph, where rows and columns correspond to nodes. Each entry indicates whether an edge exists between a pair of nodes: 1 (or a weight) for an edge and 0 for no edge. It is widely used for analyzing graph structures in both directed and undirected graphs.

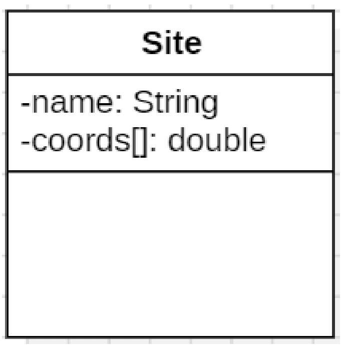
- Initially the adjacency matrix should **not** be populated. Meaning, no nodes have an associated edge. (Represented as a **0**)
- The graph is **weighted** and **undirected**. Edges are represented in weights of kilometres greater than 0.

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

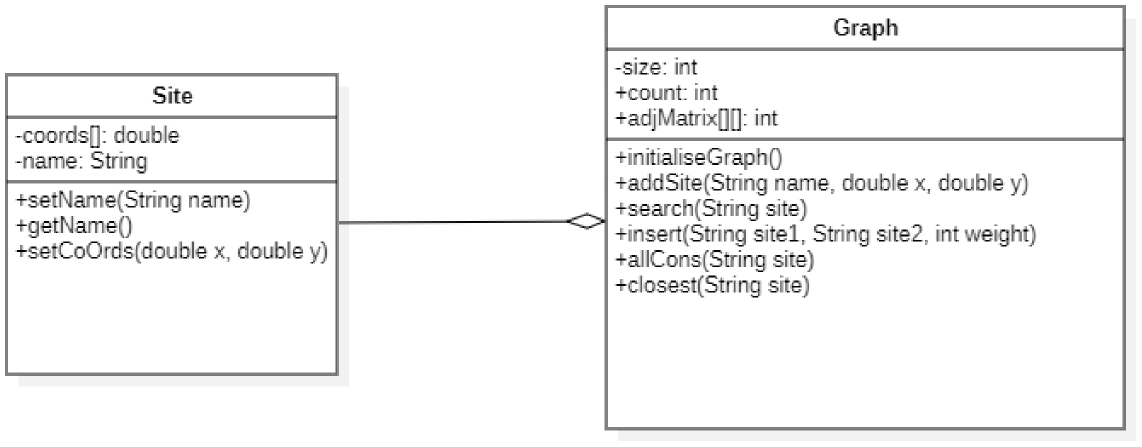
Adjacency matrix representing a graph with no edges between nodes

Defining Tourist Sites -"Each [tourist site] stored will contain the name of the site and the co-ordinates..." – as set out in the rubric.

- To store the site information each site will be represented as its own **Object.** This allows easy access to store and read site names and co-ordinates.
- To **add** a site to the graph, a name of type String and x and y coordinates of type double should be supplied.



The Graph object will communicate with the Site object via a form of basic aggregation. Each Graph object will hold the sites via an array of type Site.



Pseudocode for Algorithms

//Explanation of un-declared helper variables

count = number of current sites (nodes) in the graph

sites[] = array of site objects (nodes) in the graph

```
int isSite(String siteName)
{
        for(i = 0; i < count; i++)
        {
                if(sites[i] == siteName)
                {
                        //Returns index pos of site in array if found
                        return i
                }
        }
        //Otherwise return -1
        return -1
}
```

```
Algorithm allCons(String site)
{
        index = isSite(site)
        //Check if the index is valid
        if(index != -1)
        {
                for(col = 0; col < count; col++)
                {
                //If a weight greater than 0 is detected, there is a connection
                        if(graph[index][col] > 0)
                        {
                                output sites[col]
                        }
                }
                else
                        output "Invalid site index"
        }

}
```

```
Algorithm closest(Site site)
{
        index = isSite(site)

        if(index != -1)
        {
                for(col = 0; col < count; col++)
                {
                        //If the weight is less than the smallest AND isn't a value of 0
                        if(graph[index][col] < smallest AND graph[index][col] != 0)
                        {
                                smallest = graph[index][col]
                        }
                }
        }

        output smallest
}
```

```
Function Search(site)
{
        for(i = 0; i < count; i++)
        {
                if(sites[i] == site)
                {
                        output sites[i]
                }
        }
}
```

```
Function Insert(site1, site2, weight)
{
        //Get the indeces of both input sites
        i = isSite(site1Name)
        j = isSite(site2Name}

        if (site 1 == -1 OR site2 == -1)
        {
                output "Not found"
        }
        else
        {
                //Create a new edge with both sets of co-ords
                graph[i][j] = weight;
                graph[j][i] = weight;
        }
}
```

Method Description

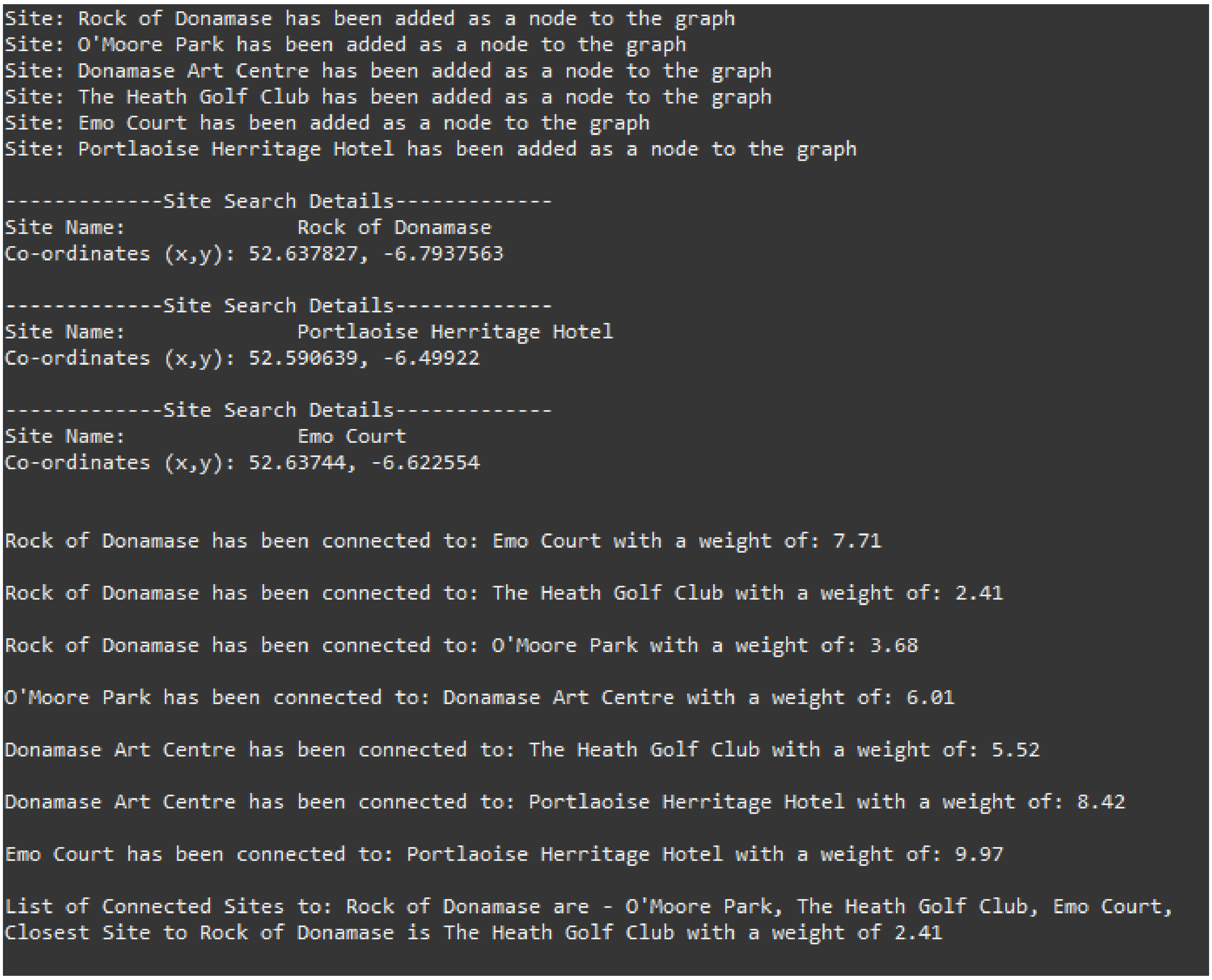
Graph Class

```
//Initialises all nodes and edges to zero on the graph

public void initialiseGraph()


//Add a site to the array with a name, x and y value (co-ordinates)

public void addSite(String name, double x, double y)


//Search and output the details of a given site

public void search(String site)


//Check if a site name input by user is valid

public int isSite(String name)


//Insert a weight between two nodes

public void insert(String site1, String site2, double weight)


//Outputs list of sites that are connected to a given input site

public void allCons(String site)


//Find the smallest weight (distance) between connected sites (nodes)

public void closest(String site)
```

Site Class

```
//Set name of a site
public void setName(String name)

//Returns the name of a site
public String getName()

//Methods to get coordinates
public double getX()
public double getY()

//Sets coords[0] to x, coords[1] to y
public void setCoOrds(double x, double y)

//Sets index via a count variable that increments automatically
public void setIndex()
public int getIndex()

//ToString override method to output all details of a given site
public String toString()
```

Sample Output



```
Site: Rock of Donamase has been added as a node to the graph
Site: O'Moore Park has been added as a node to the graph
Site: Donamase Art Centre has been added as a node to the graph
Site: The Heath Golf Club has been added as a node to the graph
Site: Emo Court has been added as a node to the graph
Site: Portlaoise Herritage Hotel has been added as a node to the graph

-------------Site Search Details-------------
Site Name:               Rock of Donamase
Co-ordinates (x,y): 52.637827, -6.7937563

-------------Site Search Details-------------
Site Name:               Portlaoise Herritage Hotel
Co-ordinates (x,y): 52.590639, -6.49922

-------------Site Search Details-------------
Site Name:               Emo Court
Co-ordinates (x,y): 52.63744, -6.622554


Rock of Donamase has been connected to: Emo Court with a weight of: 7.71

Rock of Donamase has been connected to: The Heath Golf Club with a weight of: 2.41

Rock of Donamase has been connected to: O'Moore Park with a weight of: 3.68

O'Moore Park has been connected to: Donamase Art Centre with a weight of: 6.01

Donamase Art Centre has been connected to: The Heath Golf Club with a weight of: 5.52

Donamase Art Centre has been connected to: Portlaoise Herritage Hotel with a weight of: 8.42

Emo Court has been connected to: Portlaoise Herritage Hotel with a weight of: 9.97

List of Connected Sites to: Rock of Donamase are - O'Moore Park, The Heath Golf Club, Emo Court,
Closest Site to Rock of Donamase is The Heath Golf Club with a weight of 2.41
```

Minimum Spanning Tree – Kruskal's Algorithm

When considering Kruskal's algorithm first, the list of edge weights should be ordered in increasing order.

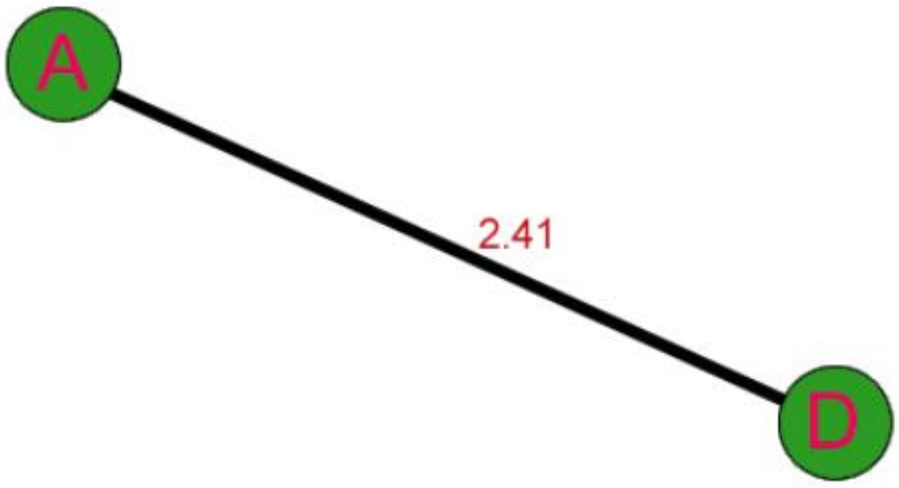
Step 1 - Sort

Site 1	Site 2	Weight
Rock of Donamase	The Heath Golf Club	2.41
Rock of Donamase	O'Moore Park	3.68
The Heath Golf Club	Donamase Art Centre	5.52
O'Moore Park	Donamase Art Centre	6.01
Rock of Donamase	Emo Court	7.71
Donamase Art Centre	Portlaoise Heritage Hotel	8.42
Emo Court	Portlaoise Heritage Hotel	9.97

Edges are considered for inclusion should a **cycle** not be formed. A cycle is where a set of nodes are connected to form a loop.

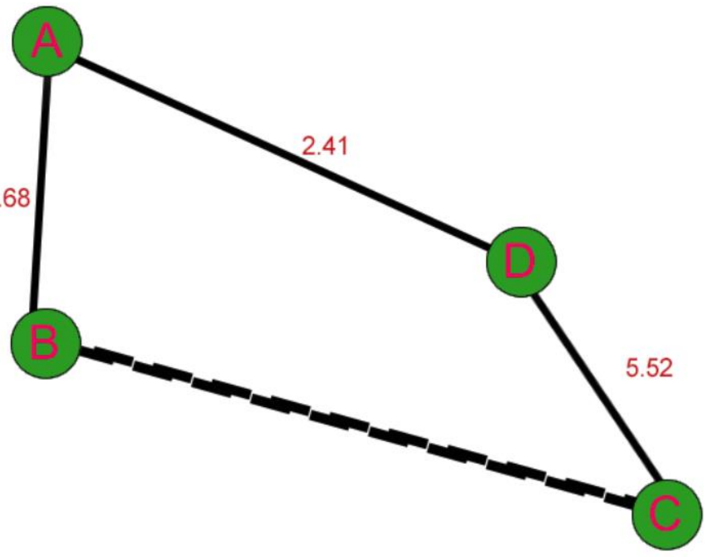
Step 2 – Consider Edges

i) **2.41** – Rock of Donamase – The Heath Golf Club



The first edge considered is the one pictured above.

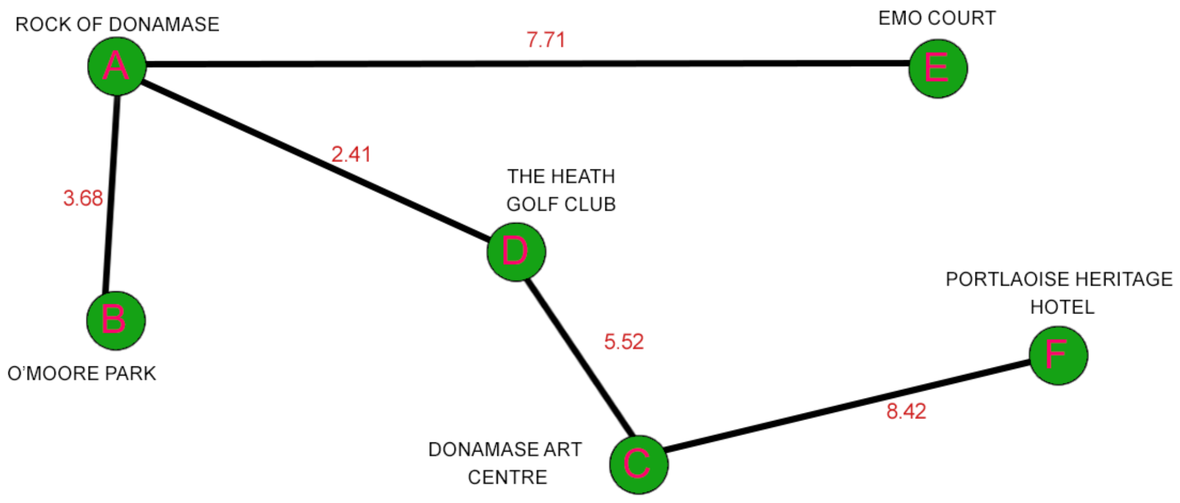
Count	Site 1	Site 2	Weight
i)	Rock of Donamase	The Heath Golf Club	2.41
ii)	Rock of Donamase	O’Moore Park	3.68
iii)	The Heath Golf Club	Donamase Art Centre	5.52
iv)	O’Moore Park	Donamase Art Centre	6.01



Above showcases how a cycle is formed, meaning that we can no longer consider the nodes O’Moore park (B) to Donamase Art Centre (C)

Final Output after Kruskal's Algorithm

Count	Site 1	Site 2	Weight
i)	Rock of Donamase	The Heath Golf Club	2.41
ii)	Rock of Donamase	O'Moore Park	3.68
iii)	The Heath Golf Club	Donamase Art Centre	5.52
iv)	Rock of Donamase	Emo Court	7.71
v)	Donamase Art Centre	Portlaoise Heritage Hotel	8.42



The final Minimum Spanning Tree will have **n** nodes and **n-1** edges.

Conclusion

In conclusion we explored the application of hash tables and graphs – testing our knowledge on the fundamentals of building a data structure of our own.

The hash table application clearly demonstrates how to take an input, perform a **hash function**, insert the unit of information to an array using the index from said hash function and, should a **collision** occur we use **linear probing**.

The graph application involved implementing an **undirected, weighted graph with the use of an adjacency matrix** to store the graph. To implement the adjacency matrix, we used a **2D matrix.** This allowed us to model a real-life example of the use of a Graph data-structure. In our case, using the example of tourist attractions.

Various functions were performed on our graph in code to help solve problems such as finding the closest site to a given node or output a list of all connected sites. A model of Kruskal's algorithm displays how a minimum spanning tree can be used in the real world also.

Overall, this project showcases our knowledge and understanding of the algorithms used to build data structures as well as our teamwork and collaborations skills.

Bibliography

Byrne, Á. (2003*), Chapter 6 – Hash Tables,* [PowerPoint Slides], Module: Data Structures and Algorithms, available: https://blackboard.itcarlow.ie/bbcswebdav/pid-957600-dt-content-rid-5807106_1/xid-5807106_1 [accessed – November 2024]

Byrne, Á. (2021*), Chapter 5 - Graphs and Mazes,* [PowerPoint Slides], Module: Data Structures and Algorithms, available:https://blackboard.itcarlow.ie/bbcswebdav/pid-951798-dt-content-rid-5731596_1/xid-5731596_1 [accessed – November 2024]

Acknowledge, Describe, Evidence Document

Include a completed version of this document as an appendix to any submitted work.

Acknowledge

[✓] I did NOT use any AI Technology or online resource.

I acknowledge the use of <insert AI system(s) and link> or <weblink> for the following purposes:

☐ to generate materials for background research and self-study in the drafting of this assessment.

☐ to generate materials that were included within my final assessment in modified form.

Describe

Please provide a short summary of how you used generative AI/website in your assignment. You may wish to include the following information:

What prompts did you use?

What outputs did you generate?

How did you use/adapt/develop the outputs?

Summary:

Evidence

Please provide evidence of the outputs that you generated by copying and pasting below or by providing a screenshot.

Generative AI system/Website: Prompt: Output:

Declaration

[✓] I confirm that no content created by generative AI technologies or website research has been presented as my own work.