CPP104: Graph Data Structure

Introduction

The Graph Data Structure is a fundamental and adaptable structure used in computer science and other fields to represent a network of nodes connected by edges. Each node, or vertex, represents an entity, while each edge represents a relationship or connection between two entities. This structure is especially useful for representing complex networks and relationships in various applications. Graphs can be classified into several types based on their properties. For example, a graph can be either directed or undirected. A directed graph has edges that go from one node to another, indicating a one-way relationship.

Graphs are widely used in various real-world applications. For instance, social networks use graphs to represent connections between users, with nodes representing individuals and edges representing friendships or follow relationships. In transportation networks, graphs can represent cities as nodes and roads or flight paths as edges, helping to solve route optimization problems. In biology, graphs can model molecular structures or the interaction networks of proteins and genes.

In conclusion, the Graph Data Structure is an important and powerful tool for representing and analyzing relationships between entities. Graphs, with their diverse types and a wealth of algorithms for manipulating and studying them, enable solutions to a wide range of complex problems in computer science, engineering, biology, social sciences, and many other domains. Anyone interested in fields that require the modeling and analysis of complex networks must first understand graphs and their associated algorithms.

Explanation

In this activity, the group was tasked with creating a graph data structure and applying it creatively by using each letter of each member's full name as a vertex. This was accomplished using the graph's add_vertex function, which treated each letter as an individual vertex. Following the addition of vertices, the add_edge function was used to connect each new vertex, resulting in edges between them. The goal was to show connectivity between all letters using both Breadth-First Search (BFS) and Depth-First Search (DFS).

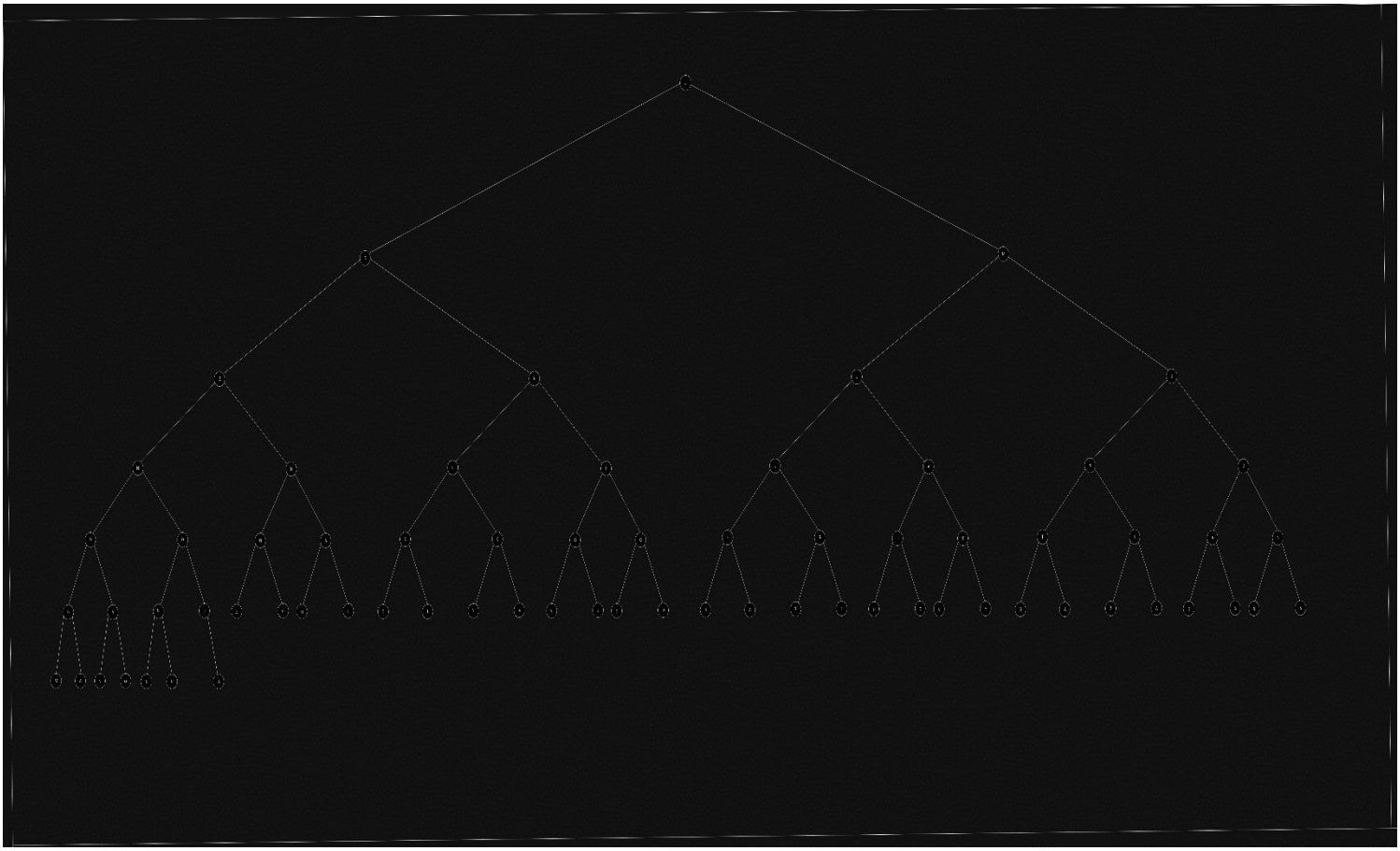
The process started with the creation of the graph. Each letter from the group members' full names was treated as a separate node in the graph. The function add_vertex was used to ensure that each letter was properly represented as a vertex in the graph. After all of the vertices had been added, the next step was to connect them. This was accomplished with the add_edge function. Edges were added in accordance with set edges. For example, the group could have decided to connect letters that appear consecutively in the names, or they could have connected all identical letters from different names. This step was critical in establishing a network of

vertices that could be navigated using graph algorithms. After constructing the graph, the next goal was to demonstrate that all of the letters were connected. To accomplish this, the group used two fundamental graph traversal algorithms: breadth-first search (BFS) and depth-first search (DFS).

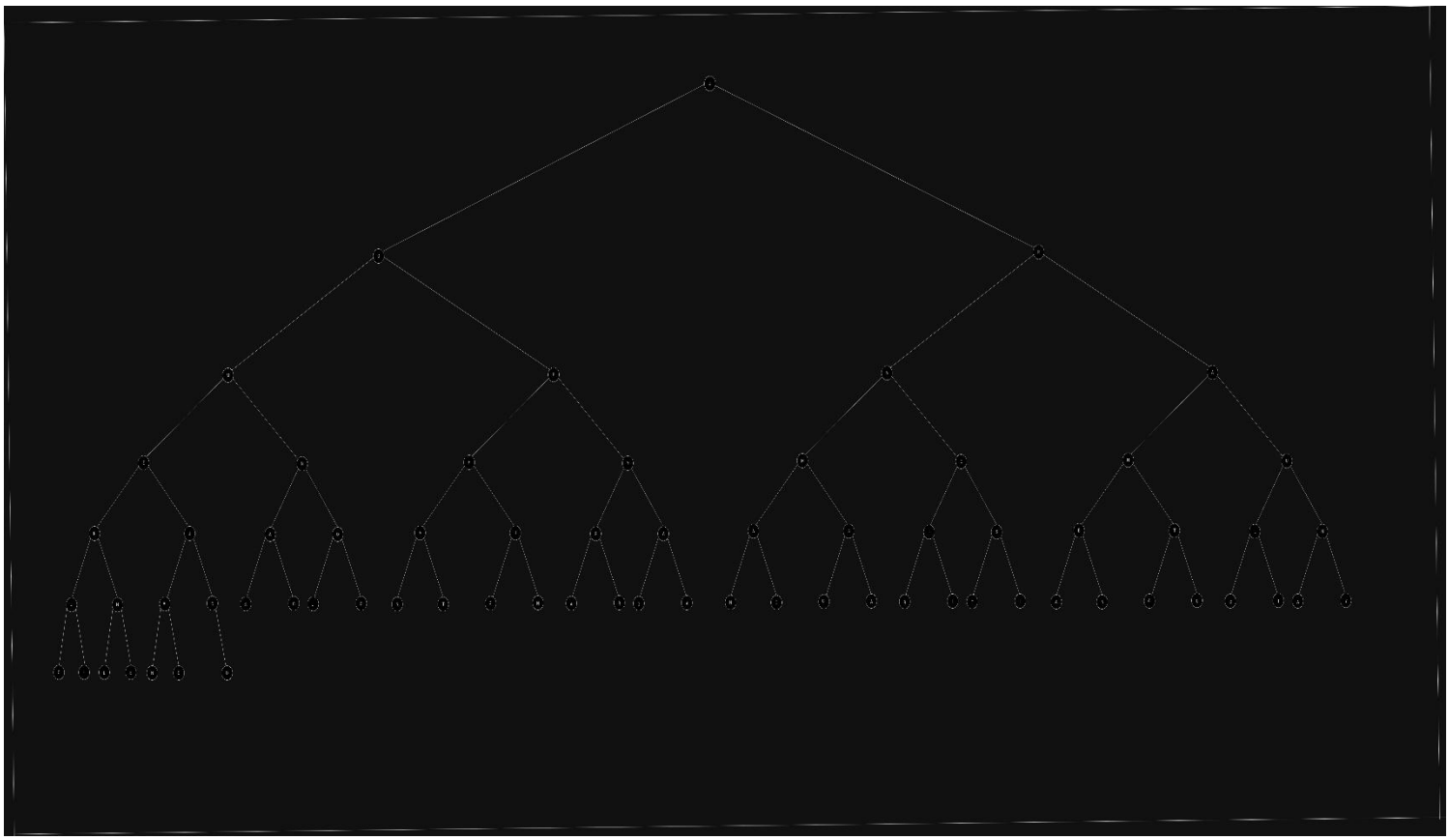
In conclusion, this activity highlighted the versatility and power of graph data structures in representing and analyzing relationships. By using the letters of the full names of group members as vertices and connecting them with edges, the group demonstrated the practical application of graph algorithms such as BFS and DFS. This not only reinforced their understanding of graph theory but also showcased the practical implications of graphs in solving real-world problems.

Graph Illustration

Breadth-first Search



Depth-first Search



Source Code

```
class Graph:

    def __init__(self):

        self.graph = {}


    def add_vertex(self, vertex):

        if vertex not in self.graph:

            self.graph[vertex] = []
```

```
def add_edge(self, vertex1, vertex2):
    if vertex1 in self.graph and vertex2 in self.graph:
        self.graph[vertex1].append(vertex2)
        self.graph[vertex2].append(vertex1)  # For undirected graph

def remove_edge(self, vertex1, vertex2):
    if vertex1 in self.graph and vertex2 in self.graph:
        self.graph[vertex1].remove(vertex2)
        self.graph[vertex2].remove(vertex1)

def remove_vertex(self, vertex):
    if vertex in self.graph:
        for adj in self.graph[vertex]:
            self.graph[adj].remove(vertex)
        del self.graph[vertex]

def bfs(self, start_vertex):
    visited = set()
    queue = [start_vertex]
    bfs_order = []

    while queue:
```

```
        vertex = queue.pop(0)
        if vertex not in visited:
            bfs_order.append(vertex)
            visited.add(vertex)
            queue.extend([v for v in self.graph[vertex] if v not in visited])

    return bfs_order

def dfs(self, start_vertex):
    visited = set()
    stack = [start_vertex]
    dfs_order = []

    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            dfs_order.append(vertex)
            visited.add(vertex)
            stack.extend([v for v in self.graph[vertex] if v not in visited])

    return dfs_order
```

```
def find_path(self, start_vertex, end_vertex, path=[]):
    path = path + [start_vertex]
    if start_vertex == end_vertex:
        return path
    if start_vertex not in self.graph:
        return None
    for vertex in self.graph[start_vertex]:
        if vertex not in path:
            new_path = self.find_path(vertex, end_vertex, path)
            if new_path:
                return new_path
    return None


def find_all_paths(self, start_vertex, end_vertex, path=[]):
    path = path + [start_vertex]
    if start_vertex == end_vertex:
        return [path]
    if start_vertex not in self.graph:
        return []
    paths = []
    for vertex in self.graph[start_vertex]:
        if vertex not in path:
```

```
            new_paths = self.find_all_paths(vertex, end_vertex, path)
            for p in new_paths:
                paths.append(p)
    return paths


def is_connected(self):
    start_vertex = list(self.graph.keys())[0]
    visited = self.dfs(start_vertex)
    return len(visited) == len(self.graph)


def has_cycle(self):
    visited = set()
    for vertex in self.graph:
        if vertex not in visited:
            if self._has_cycle(vertex, visited, -1):
                return True
    return False


def _has_cycle(self, vertex, visited, parent):
    visited.add(vertex)
    for adj in self.graph[vertex]:
        if adj not in visited:
```

```
            if self._has_cycle(adj, visited, vertex):
                return True
        elif parent != adj:
            return True
    return False


### Output
graph = Graph()


jomer = "JOMERJOHNLEJANODONASCO"
joph = "JOPHANTHONYGARAGARAMANOSCA"
inay = "CHRISTIANBANTACULOINAY"
nameList = []
nameList.extend(jomer)
nameList.extend(joph)
nameList.extend(inay)
counts = {}


def addVertexLoop(names,counter):
    for letter in names:
        count = counter.get(letter,0)
        if count == 0:
```

```
            graph.add_vertex(letter)
        else:
            graph.add_vertex(letter + str(count))
        counter[letter] = count + 1


def addBFSEdge():
    graph.graph.clear()
    addVertexLoop(nameList,counts)
    vertexList = list(graph.graph.keys())
    root = vertexList[0]
    stack = [root]
    count = 1

    while stack and count < len(vertexList):
        current = stack.pop(0)
        children = 2
        for _ in range(children):
            if count < len(vertexList):
                child = vertexList[count]
                graph.add_edge(current, child)
                stack.append(child)
                count += 1
```

```
    output = numberRemove(graph.bfs(vertexList[0]))
    return output


def addDFSEdge():
    graph.graph.clear()
    counts.clear()
    addVertexLoop(nameList,counts)
    vertexList = list(graph.graph.keys())
    #JOMER
    graph.add_edge("J","R2")
    graph.add_edge("J","O")
    graph.add_edge("O","P")
    graph.add_edge("O","M")
    graph.add_edge("M","N2")
    graph.add_edge("M","E")
    graph.add_edge("E","J2")
    graph.add_edge("E","R")
    graph.add_edge("R","N")
    graph.add_edge("R","J1")
    graph.add_edge("J1","H")
    graph.add_edge("J1","O1")
```

```
graph.add_edge("N","E1")

graph.add_edge("N","L")

graph.add_edge("J2","D")

graph.add_edge("J2","A")

graph.add_edge("A","O2")

graph.add_edge("A","N1")

graph.add_edge("D","O3")

graph.add_edge("N2","O4")

graph.add_edge("N2","A1")

graph.add_edge("A1","C")

graph.add_edge("A1","S")


#JOPH

graph.add_edge("O4","O5")

graph.add_edge("O4","J3")

graph.add_edge("P","Y")

graph.add_edge("P","H1")

graph.add_edge("H1","H2")

graph.add_edge("H1","A2")

graph.add_edge("A2","T")

graph.add_edge("A2","N3")

graph.add_edge("H2","N4")
```

```
graph.add_edge("H2","O6")

graph.add_edge("Y","A4")

graph.add_edge("Y","G")

graph.add_edge("G","R1")

graph.add_edge("G","A3")

graph.add_edge("A4","A5")

graph.add_edge("A4","G1")

graph.add_edge("R2","A9")

graph.add_edge("R2","A6")

graph.add_edge("A6","C2")

graph.add_edge("A6","M1")

graph.add_edge("M1","S1")

graph.add_edge("M1","A7")

graph.add_edge("A7","O7")

graph.add_edge("A7","N5")

graph.add_edge("S1","A8")

graph.add_edge("S1","C1")


#INAY

graph.add_edge("C2","S2")

graph.add_edge("C2","H3")

graph.add_edge("H3","I")
```

```
    graph.add_edge("H3","R3")
    graph.add_edge("S2","I1")
    graph.add_edge("S2","T1")
    graph.add_edge("A9","U")
    graph.add_edge("A9","N6")
    graph.add_edge("N6","T2")
    graph.add_edge("N6","B")
    graph.add_edge("B","N7")
    graph.add_edge("B","A10")
    graph.add_edge("T2","C3")
    graph.add_edge("T2","A11")
    graph.add_edge("U","N8")
    graph.add_edge("U","L1")
    graph.add_edge("L1","I2")
    graph.add_edge("L1","O8")
    graph.add_edge("N8","Y1")
    graph.add_edge("N8","A12")
    output = numberRemove(graph.dfs(vertexList[0]))
    return output


def numberRemove(array):
    output = []
```

```
    for element in array:
        output.append(element[0])
    return output


print("===BREADTH FIRST SEARCH===")
print(addBFSEdge())
print()
print("===DEPTH FIRST SEARCH===")
print(addDFSEdge())
```

References

ChatGPT. https://chatgpt.com/

Draw.io https://app.diagrams.net/

GeeksforGeeks. (2024, April 3). *Graph Data Structure And Algorithms*. GeeksforGeeks. https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Quillbot. Paraphrasing-tool. https://quillbot.com/paraphrasing-tool

Members:

Name	Contribution
Donasco, Jomer John L.	Back-End Development
Inay, Christian B.	Documentation
Mañosca, Joph Anthony G.	Graph Diagram

JOMER JOHN L. DONASCO

Blk 2 Lot 4 San Isidro Heights Phase 2 Banlic, City of Cabuyao, Laguna
pjayjaydonasco@gmail.com
09152223913

PROFILE

I am a dedicated computer engineering student that is eager to learn more about the hardware and software aspects of technology. I am proficient in programming and other computer related skills

TECHNICAL SKILLS

- Programming skills
- Computer Literate
- Office Productivity Utilization Skills

EDUCATIONAL BACKGROUND

PAMANTASAN NG CABUYAO
Bachelor in Computer Engineering

LAGUNA BELAIR SCIENCE SCHOOL
Senior High School

LAGUNA BELAIR SCIENCE SCHOOL
Junior High School

PERSONAL DATA

GENDER:	Male
HEIGHT:	164cm
WEIGHT:	50kg
CITIZENSHIP:	Filipino
CIVIL STATUS:	Single
RELIGION:	Roman Catholic
LANGUAGE:	Filipino
BIRTH DATE:	June 7, 2000
PARENTS:	Mercedita L. Donasco Joselito V. Donasco

I hereby certify that the above information is true and correct based on my knowledge.

DONASCO, JOMER JOHN L.

CAREER OBJECTIVES

I am an aspiring successful computer engineering student. I am committed to develop my skills in terms of the hardware and software aspects of computers. I am eager to learn more about the theories and concepts in computer engineering and apply them in the professional field

EXPERIENCES, SEMINARS, AND AFFILIATIONS

WAREHOUSE SORTER
Ninja Van Express Tech. Philippines
Lot B, Pulo-Diezmo Road Cabuyao Laguna, Philippines

MEMBER
AWS Cloud Club - University of Cabuyao
https://www.facebook.com/awscepnc

CHARACTER REFERENCES

ENGR. JIM MARCO M. PABELLON
Pamantasan ng Cabuyao
Cabuyao, Laguna
pabellonjimmarco@gmail.com

FRANCIS DELOS SANTOS
Ninja Van Express Tech. Philippines
Cabuyao, Laguna
09561223174

Christian B. Inay

Block 9 Lot 6 Centennial Townhomes 2, Brgy. Pulo, Cabuyao, Laguna
0907 – 529 – 6948
Christianinay98@gmail.com

OBJECTIVE

To strengthen a solid foundation in computer engineering principles, obtain practical experience through internships and projects, improve problem-solving abilities, and stay current with technology changes. Aim for academic excellence while actively participating in team collaborations and networking. Develop a strong foundation in computer engineering principles, gain practical experience through internships and projects, improve problem-solving skills, and stay up to date on the latest technological advancements. Aim for academic excellence by actively participating in team collaborations.

TECHNICAL SKILLS

Microsoft Office
Basic Computer Programming
Java
Python

PERSONAL SKILLS

Time Management
Adaptability
Teamwork
Communication Skill
Problem-Solving Skill

EDUCATION

COLLEGE
University of Cabuyao
Katapatan Homes, Brgy. Banay-banay, Cabuyao, Laguna

SENIOR HIGH SCHOOL
University of Cabuyao
Katapatan Homes, Brgy. Banay-banay, Cabuyao, Laguna

SECONDARY
Balibago Integrated High School
Balibago, Sta. Rosa, Laguna

PRIMARY
Fourth Estate Elementary School
Fourth Estate Subdivision, Sucat, Parañaque City

INFORMATION

Civil Status: Single
Religion: Roman Catholic
Birthday: December 14, 1998
Mother's Name: Arlene Inay
Father's Name: Alex Inay Sr.
Nationality: Filipino
Height: 170 cm
Weight: 75 Kg

ACHIEVEMENTS

Senior High School
With Honor - Grade 11
With Honor - Grade 12
Top 2 - Grade 12

Work Experience/ Affiliation

N/A

Character Reference

Ms. Maricar Cruz
Senior High School Adviser

CHRISTIAN B. INAY
APPLICANT'S NAME

JOPH ANTHONY G. MAÑOSCA

BLK 17 L 23 PH 1 SJV 7, MARINIG, CABUYAO, LAGUNA
Akosijopjop123@gmail.com
09855482659

PROFILE

Dedicated and cautious student applying to the University of Cabuyao's Bachelor of Computer Engineering program. aiming to build a solid foundation in the topic by obtaining practical experience via school projects, internships, and volunteer activity. keen to utilize technical skills in a fast-paced, collaborative environment and add to team initiatives.

TECHNICAL SKILLS

- Management Skills
- Creativity
- Negotiation
- Critical Thinking

EDUCATIONAL BACKGROUND

Pamantasan ng Cabuyao
Bachelor of Science in Computer Engineering

Westbridge Institute of Technology, Inc
2021 – 2023

Cabuyao Integrated National School
2016 - 2020

PERSONAL DATA

GENDER:	MALE
HEIGHT:	175.26cm
WEIGHT:	77kg
CITIZENSHIP:	FILIPINO
CIVIL STATUS:	SINGLE
RELIGION:	CATHOLIC
LANGUAGE:	FILIPINO
BIRTHDATE:	JULY12,2005
PARENTS:	ANALISAGARAGARA JOPERTC.MAÑOSCA

I hereby certify that the above information is true and correct based on my knowledge.

MAÑOSCA , JOPH ANTHONY

CAREER OBJECTIVES

Career objectives a dedicated and meticulous student seeking an entry-level position in Computer Engineering where I can use my skills and advance my career. I can't wait to grow in a progressive company and add value to a vibrant team..

EXPERIENCES, SEMINARS, AND AFFILIATIONS

CHARACTER REFERENCES

Mark Aldrin Ramos
Mabuhay phase 1 Blk 20 L 3
macmacramos08@gmail.com