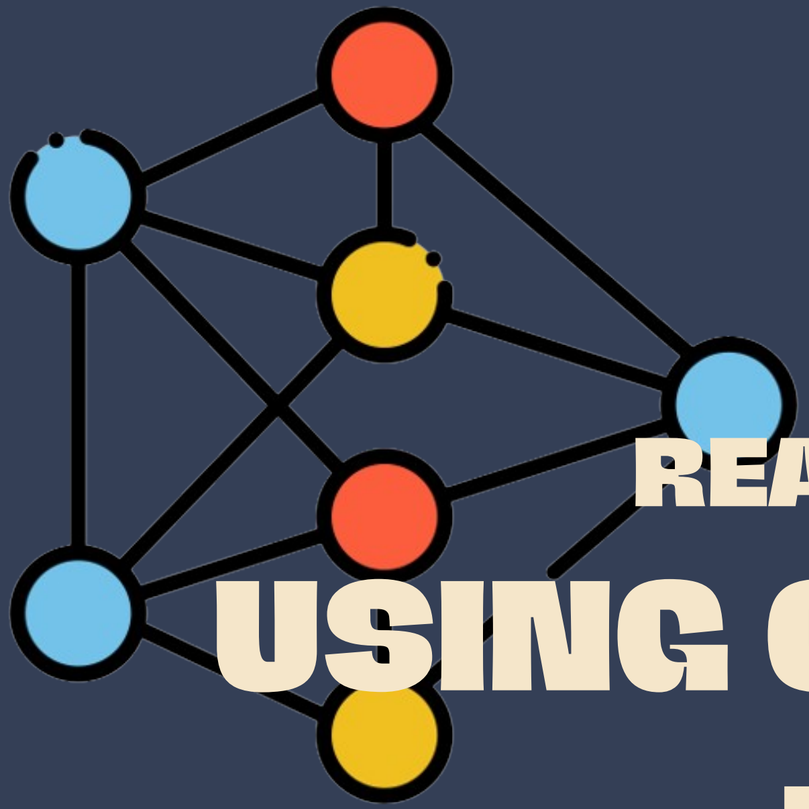
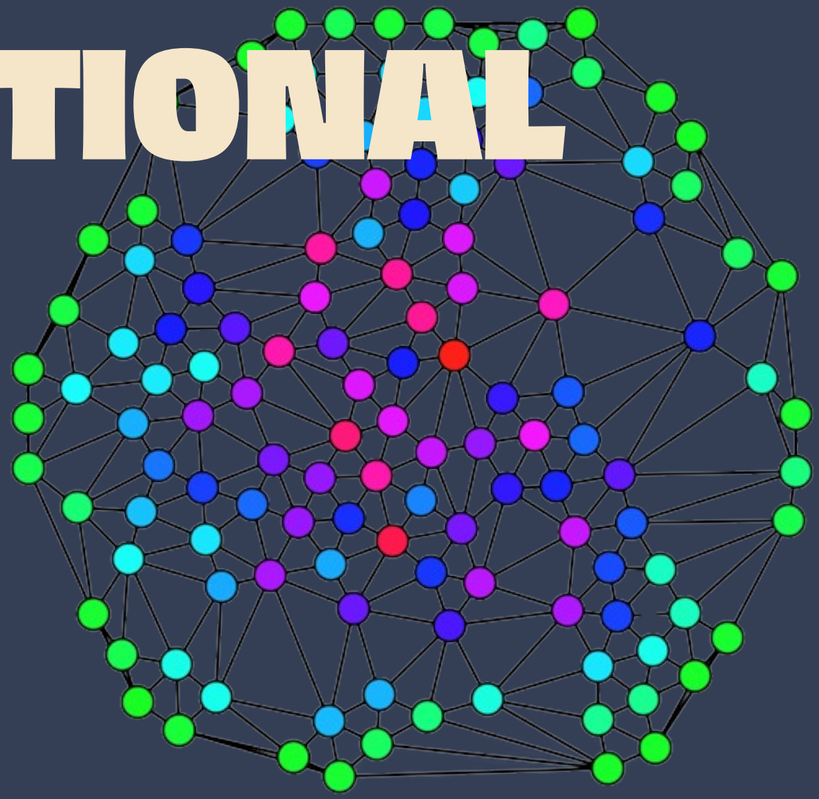


# REAL WORLD PRBLEM USING COMPUTATIONAL THINKING



BY TEAM 7

# OVERVIEW

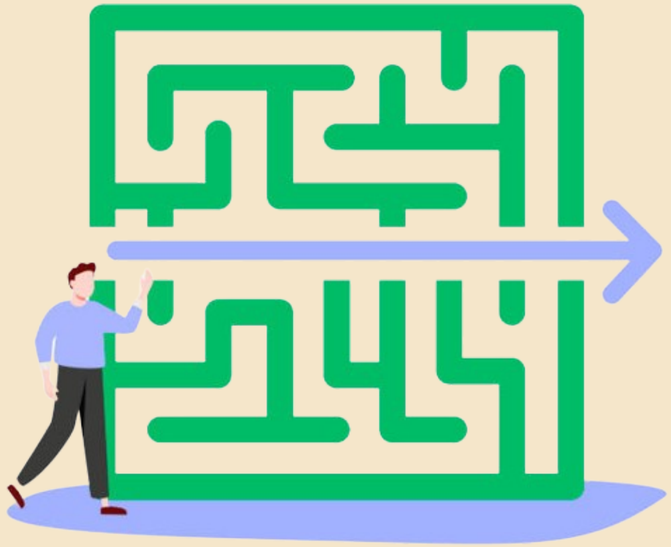
INTRODUCTION

PROBLEM

IMPLEMENTATION

# INTRODUCTION

When it comes to remembering and navigating the TIP campus, freshman students find it overwhelming to go from one building to another for their next class. Also, they find it difficult to find out what is the shortest route to go from their source to their destination. This leads to problems such as emotional distress and can affect the overall performance and experience of the student such as missing a part of the lecture and attendance.



# PROBLEM

Freshmen or Visitors on the TIP QC campus encounter difficulties finding their designated building, which wastes a lot of their time.

# OBJECTIVES

TEAM 7

## OBJECTIVE 01

To create a program that can navigate students to their designated building for their next class

## OBJECTIVE 02

To create an algorithm that can decrease the distance that students must travel in order to reach their assigned building by finding the shortest path from source to destination, particularly for new students or guests who are unfamiliar with a school's layout.

BY TEAM 7

# 1. PROBLEM IDENTIFICATION

How to get the visitors and freshmen familiarized with the campus?

## DECOMPOSITION

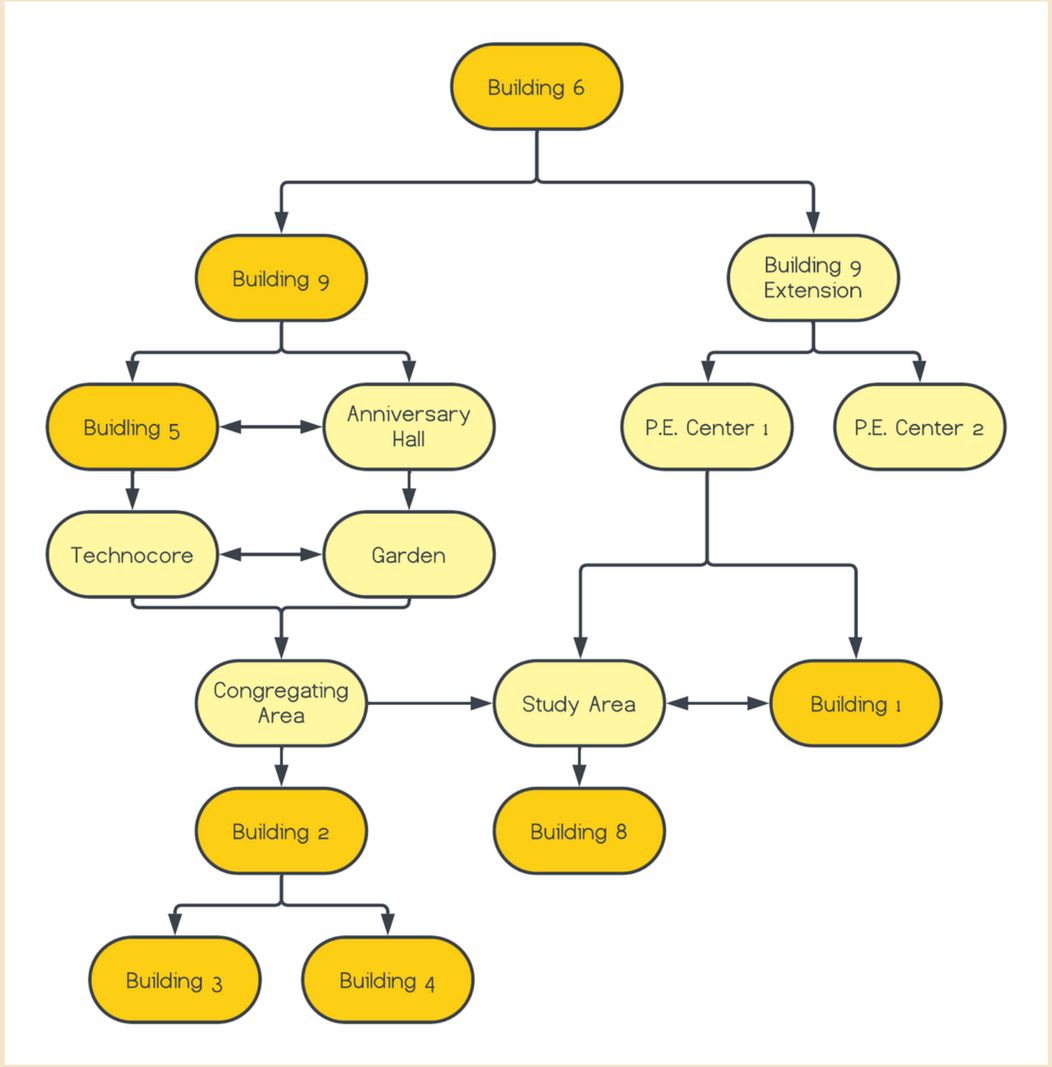
- Provide them with a graph map of the campus
- Providing an optimal way to their destinations

## PATTERN RECOG.

- Newcomers struggles to find buildings in the campus

## ABSTRACTION

Relevant: Buildings
Irrelevant: Freshmen or Visitors


Building 6
Building 9
Building 9 Extension
Buidling 5
Anniversary Hall
P.E. Center 1
P.E. Center 2
Technocore
Garden
Congregating Area
Study Area
Building 1
Building 2
Building 8
Building 3
Building 4

# 2. PROBLEM IDENTIFICATION

How they can move from one building to another with minimized time travel.

## DECOMPOSITION

- Identify the locations of the buildings
- Implement Python graph theory and BFS algorithm

## PATTERN RECOG.

- When they want to be on time to their schedule.

## ABSTRACTION

Relevant: Connection of the graph.
Irrelevant: Floors of the buildings

```
tip_graph = {'Building 6': ['Building 9', 'Building 9 Extension'],
             'Building 9': ['Building 6', 'Building 5', 'Anniversary Hall'],
             'Building 9 Extension': ['Building 6', 'P.E. Center 1', 'P.E. Center 2'],
             'Building 5': ['Building 9', 'Anniversary Hall', 'Technocore'],
             'Anniversary Hall': ['Building 9', 'Building 5', 'Garden'],
             'P.E. Center 1': ['Building 9 Extension', 'Study Area', 'Building 1'],
             'P.E. Center 2': ['Building 9 Extension'],
             'Technocore': ['Building 5', 'Garden', 'Congregating Area'],
             'Garden': ['Anniversary Hall', 'Technocore', 'Congregating Area'],
             'Study Area': ['P.E. Center 1','Congregating Area', 'Building 1', 'Building 8'],
             'Building 1': ['P.E. Center 1', 'Study Area'],
             'Congregating Area': ['Technocore', 'Garden', 'Study Area'],
             'Building 2': ['Congregating Area', 'Building 4', 'Building 3'],
             'Building 8': ['Study Area'],
             'Building 4': ['Building 2'],
             'Building 3': ['Building 2']}
```

```
class Node(object):
    def __init__(self, name):
        self.name = name

    def getName(self):
        return self.name

class Graph(object):
    def __init__(self):
        self.edges = {}

    def addNode(self, node):
        if node in self.edges:
            raise ValueError("Duplicate Node!")
        self.edges[node] = []

    def addEdge(self, a, b):
        if not (a in self.edges and b in self.edges):
            raise ValueError("Node does not exist")
        self.edges[a].append(b)
        self.edges[b].append(a)
```

```python
# BFS Algorithm
def BFS(graph, start, end):
    init_path = [(start, [start])]
    visited = set()
    while len(init_path) != 0:
        node, path = init_path.pop(0)
        visited.add(node)
        if node == end:
            return path
        for next_node in graph.edges[node]:
            if next_node not in visited:
                init_path.append((next_node, path + [next_node]))
    return None
```

```python
# Creating nodes and graph
nodes = [Node(name) for name in tip_graph.keys()]

graph = Graph()

for node in nodes:
    graph.addNode(node)

for node, neighbors in tip_graph.items():
    for neighbor_name in neighbors:
        graph.addEdge(nodes[list(tip_graph.keys()).index(node)],
                      nodes[list(tip_graph.keys()).index(neighbor_name)])
```

```python
# Test BFS
startNode = nodes[list(tip_graph.keys()).index('Building 6')]
endNode = nodes[list(tip_graph.keys()).index('Building 8')]

bfs = BFS(graph, startNode, endNode)
```

```python
# Output
print(f"The shortest path from {bfs[0].getName()} to {bfs[-1].getName()} is {' -> '.join([node.getName() for node in bfs])}")
```

```
The shortest path from Building 6 to Building 8 is Building 6 -> Building 9 Extension -> P.E. Center 1 -> Study Area -> Building 8
```

# THANK YOU!