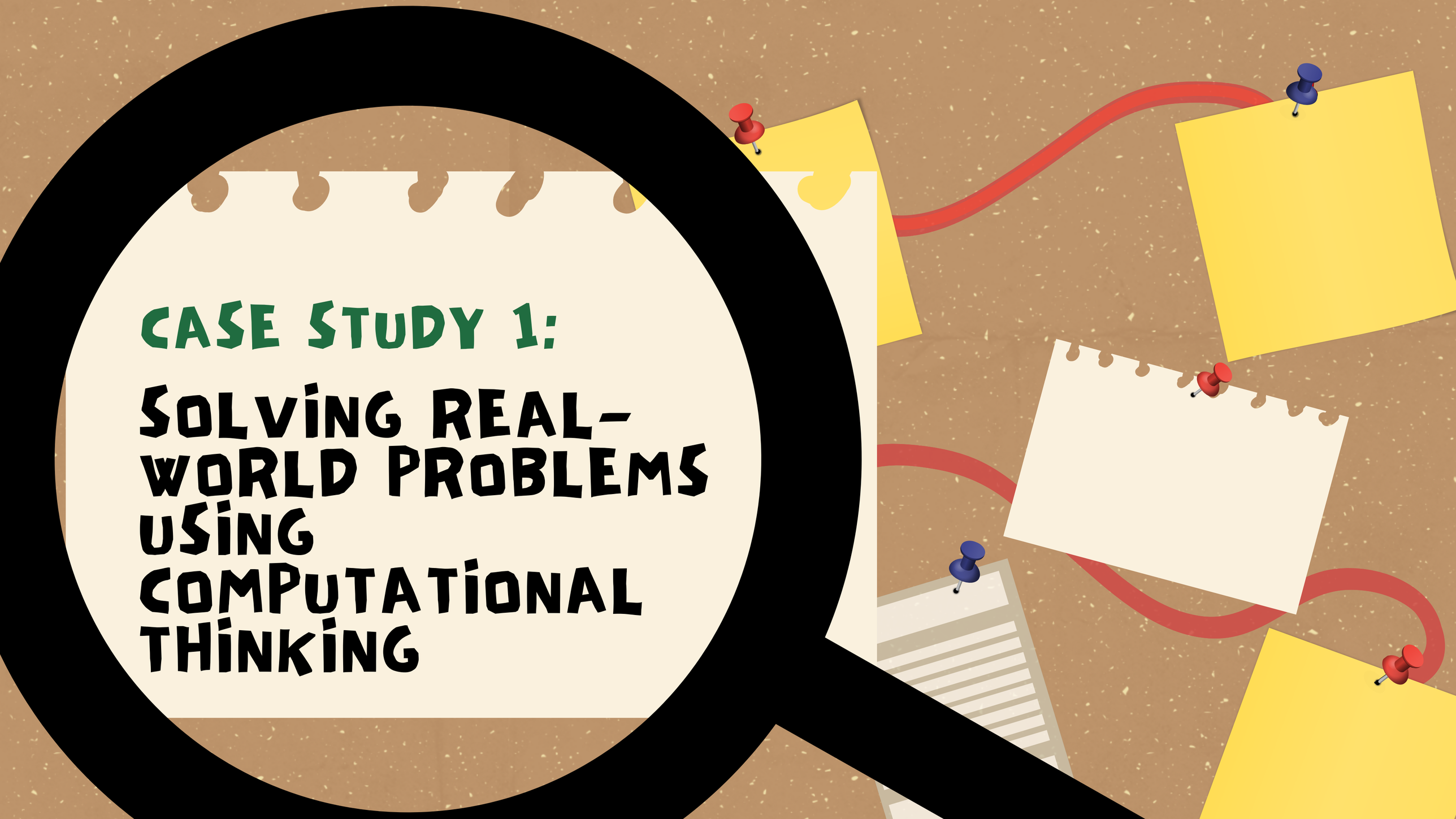
CASE STUDY 1:

SOLVING REAL-WORLD PROBLEMS USING COMPUTATIONAL THINKING



Problem: TIP is considerably a huge campus, there is an abundance of pathways to traverse from, thus searching for offices within the campus can be a real hassle.

iteration 1

Problem Identification

How can we locate the offices/ faculties of tip qc campus without getting lost.

Decomposition (How would you break down your problem into sub- problems?

- Searching building, the floor of each faculties and offices
- What is the best path that I will travel to a certain location without getting tired

Pattern Recognition ⟨Are there related solutions to draw on?⟩

1. Thousands place corresponds for the building number

2. The hundreds place corresponds to the floor of a certain building

3. The tens and one corresponds of the room number

Abstraction (How would you abstract this problem?)

- Relevant information: building numbers, floor numbers, room number

- Less relevant information that does not corresponds with the location of the offices

iteration 2

Problem Identification

I want to locate a certain office or faculty room inside the TIP QC Campus

Decomposition (How would you break down your problem into sub-problems?)

- How short the shortest path? from my current to my destination
- what are the pathways that I need to travel on to arrive at my destination?

Pattern Recognition (Are there related Solutions to draw on?)

- To find the shortest path/ or pathways to my location i will review the pathways in which I will travel on to review if the travel time lessens or lengthens

- The least amount of pathways taken

Abstraction (How would you abstract this problem?)

- **Relevant information** building numbers, floor numbers, room number

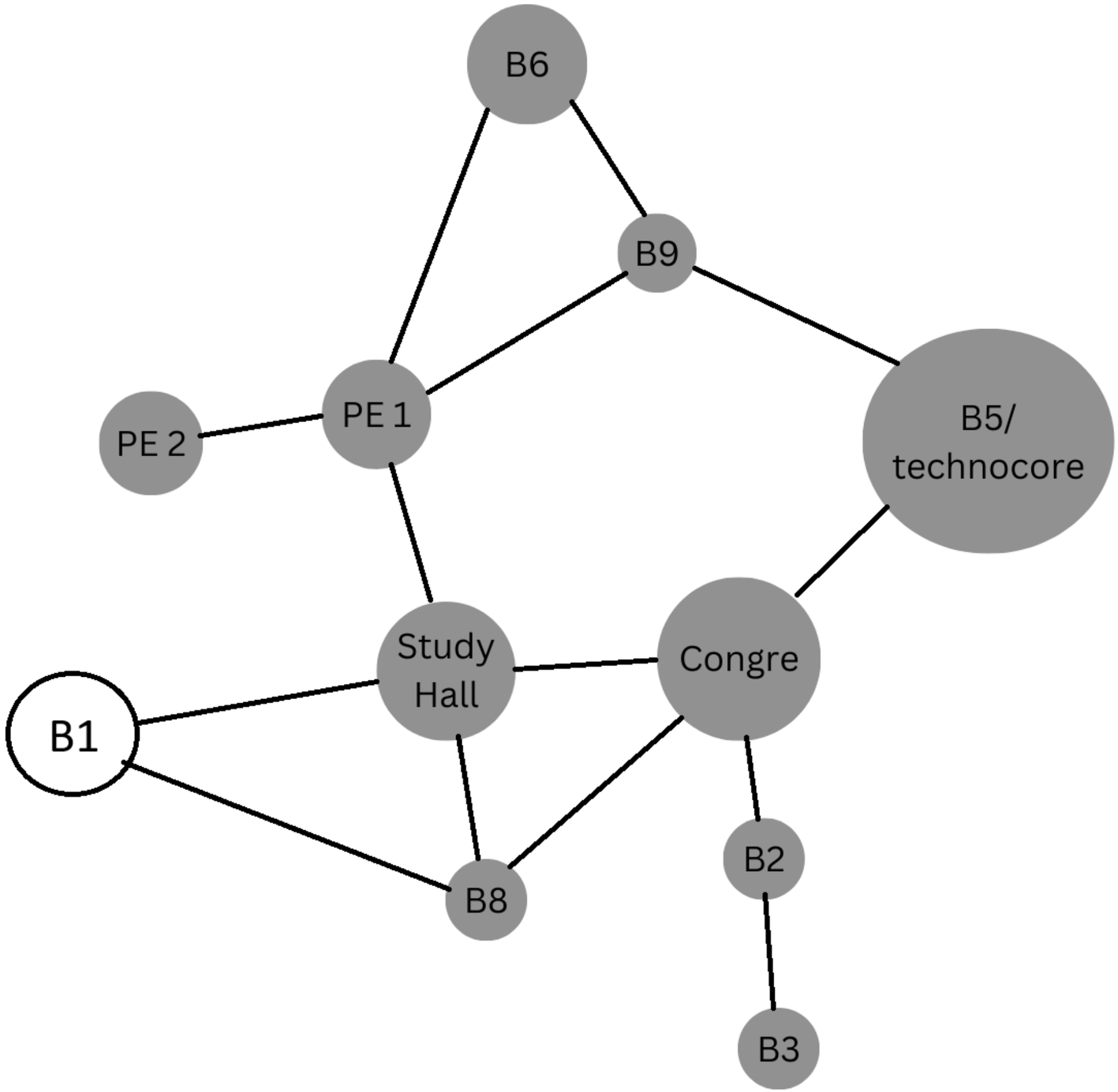
- **Less relevant information** a specific architectural designs or historical significance of campus buildings

The solution that we have come up with to solve this issue is a simple pathfinder inside the tip campus that utilizes graph theory. the pathfinder outputs what it deems to be the shortest path from the source to the destination with the use of BFS.

B - BUILDING

Congre - Congregating Area

PE 1/PE 2 - PE CENTER 1 or PE CENTER 2



BUILDINGS & LOCATIONS:

- BUILDING 1
- Career Center, Library
- Registrar, Tellering
- BUILDING 2
- ME faculty, CE faculty
- BUILDING 3
- ARCHI faculty
- BUILDING 5
- CpE faculty
- BUILDING 6

BUILDINGS & LOCATIONS:

- OSA, Guidance
- BUILDING 8
- ENG Library
- BUILDING 9
- SHS faculty, COA faculty
- PE CENTER 1 & 2
- CONGRE
- STUDY HALL

ALGORITHMS USED:

- BFS for getting the location
- Linear Search for getting the floor number
- Top-Down Dynamic Programming for storing the path calculated in the memory

Breakdown of Our Code

```python
def buildTIPGraph():
    g = Graph()
    # Create nodes for each building with its corresponding floors and offices
    buildings_info = {
        'PE 1': {'1st Floor': ['PE FACULTY']},
        'PE 2': {'1st Floor': ['PARKING']},
        'B6': {'1st Floor': [],'2nd Floor':[],'3rd Floor':[],
               '4th Floor':[],'5th Floor':[],'6th Floor':['OSA','GUIDANCE']},
        'B9': {'1st Floor':['SHS FACULTY','COA FACULTY']},
        'B5/TECHNOCORE': {1:[],2:[],3:[],4:['CpE FACULTY']},
        'CONGRE': {'1st Floor':['SAC']},
        'B2': {'1st Floor':['ME FACULTY'],2:['CE FACULTY']},
        'B3': {'3rd Floor':['ARCHI FACULTY']},
        'B8': {'2nd Floor':['ENG LIBRARY']},
        'STUDY HALL': {},  # Assuming no specific floors/offices are given
        'B1': {'1st Floor':['REGISTRAR','CAREER CENTER','TELLERING'],'2ND Floor':['MAIN LIBRARY']}
    }

    for name, BDict in buildings_info.items():
        g.addNode(Node(name, BDict))

    # This part needs to reflect the actual connections between buildings
    g.addEdge(Edge(g.getNode('PE 1'), g.getNode('PE 2')))
    g.addEdge(Edge(g.getNode('PE 1'), g.getNode('B9')))
    g.addEdge(Edge(g.getNode('PE 1'), g.getNode('B6')))
    g.addEdge(Edge(g.getNode('B6'), g.getNode('B9')))
    g.addEdge(Edge(g.getNode('B9'), g.getNode('B5/TECHNOCORE')))
    g.addEdge(Edge(g.getNode('B5/TECHNOCORE'), g.getNode('CONGRE')))
    g.addEdge(Edge(g.getNode('CONGRE'), g.getNode('B2')))
    g.addEdge(Edge(g.getNode('B2'), g.getNode('B3')))
    g.addEdge(Edge(g.getNode('CONGRE'), g.getNode('B8')))
    g.addEdge(Edge(g.getNode('CONGRE'), g.getNode('STUDY HALL')))
    g.addEdge(Edge(g.getNode('PE 1'), g.getNode('STUDY HALL')))
    g.addEdge(Edge(g.getNode('B1'), g.getNode('STUDY HALL')))
    g.addEdge(Edge(g.getNode('B1'), g.getNode('B8')))

    return g
```

```
def BFS(graph, start, end, toPrint = False):
    """Assumes graph is a Digraph; start and end are nodes
       Returns a shortest path from start to end in graph"""
    g = buildTIPGraph()
    initPath = [start]
    pathQueue = [initPath]
    while len(pathQueue) != 0:
        #Get and remove oldest element in pathQueue
        tmpPath = pathQueue.pop(0)
        if toPrint:
            print('Current BFS path:', printPath(tmpPath))
        lastNode = tmpPath[-1]
        buildDict = g.getDictionary(lastNode.getName())
        for i in buildDict:
            for j in buildDict[i]:
                if end == j:
                    finalPath = tmpPath.copy()
                    finalPath.append(i)
                    finalPath.append(j)
                    return finalPath
        for nextNode in graph.childrenOf(lastNode):
            if nextNode not in tmpPath:
                newPath = tmpPath + [nextNode]
                pathQueue.append(newPath)
    return None
```

```
def testSP(source,destination,memory):
    if source in memory and destination in memory[source]:
        print('the path is stored in the memory and will not go over to the algo')
        return memory[source][destination]
    else:
        g = buildTIPGraph()
        sp = shortestPath(g, g.getNode(source), destination)
        if sp != None:
            print('Shortest path from', source, 'to',
                  destination, 'is', printSPath(sp))
            if not source in memory:
                memory[source] = {}
                memory[source][destination] = sp
            else:
                memory[source][destination] = sp
        else:
            print('There is no path from', source, 'to', destination)
```


Any
Questions?