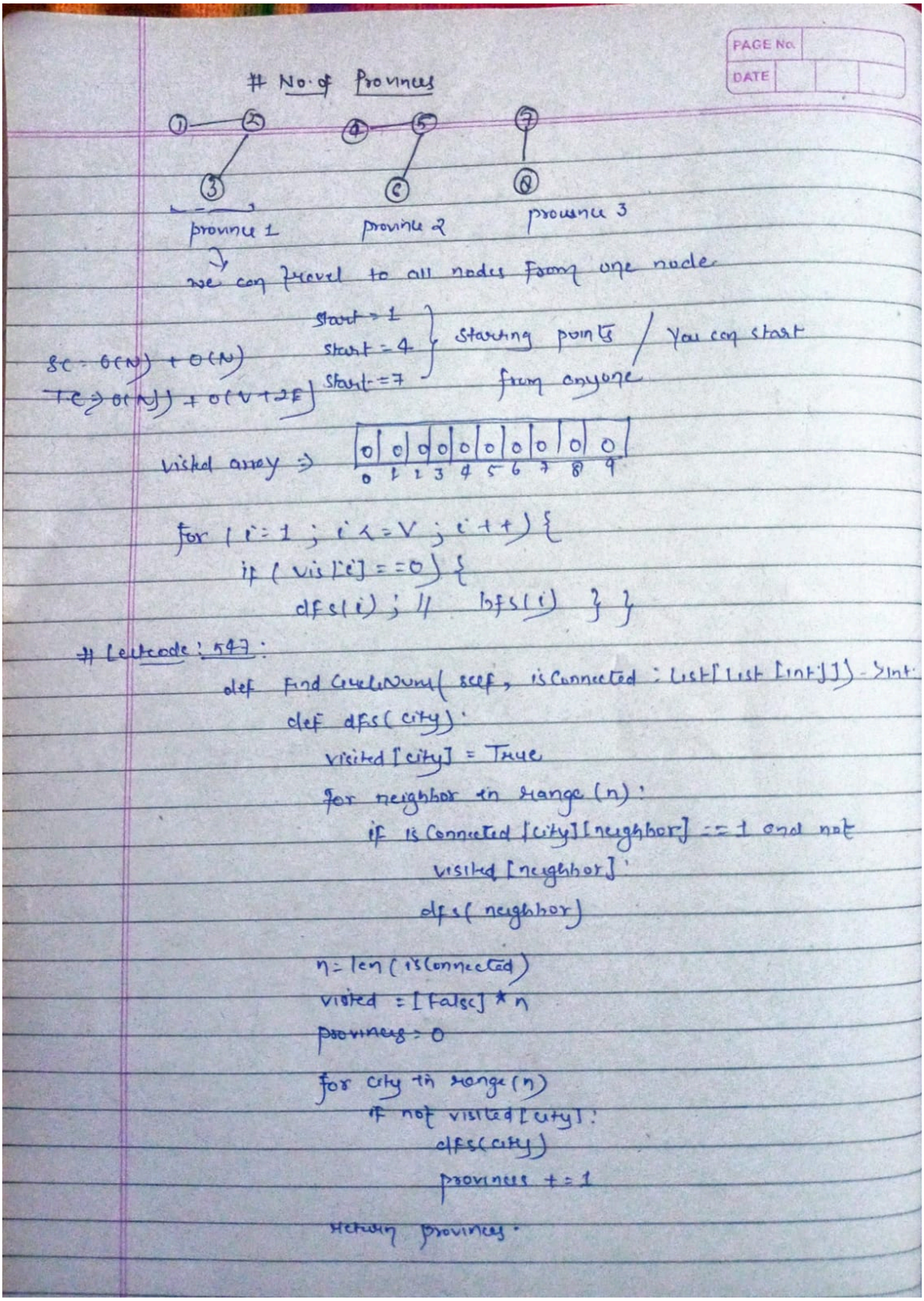
Graph_Day_2



PAGE No.

DATE

No. of Provinces

1
2
3
4
5
6
7
8
provnice 1
provinie 2
provsnce 3

we can travel to all nodes from one node

SC - O(N) + O(N)

TC ⇒ O(N)) + O(V + 2E)

start = 1
start = 4 } starting points / You can start
start = 7   from anyone

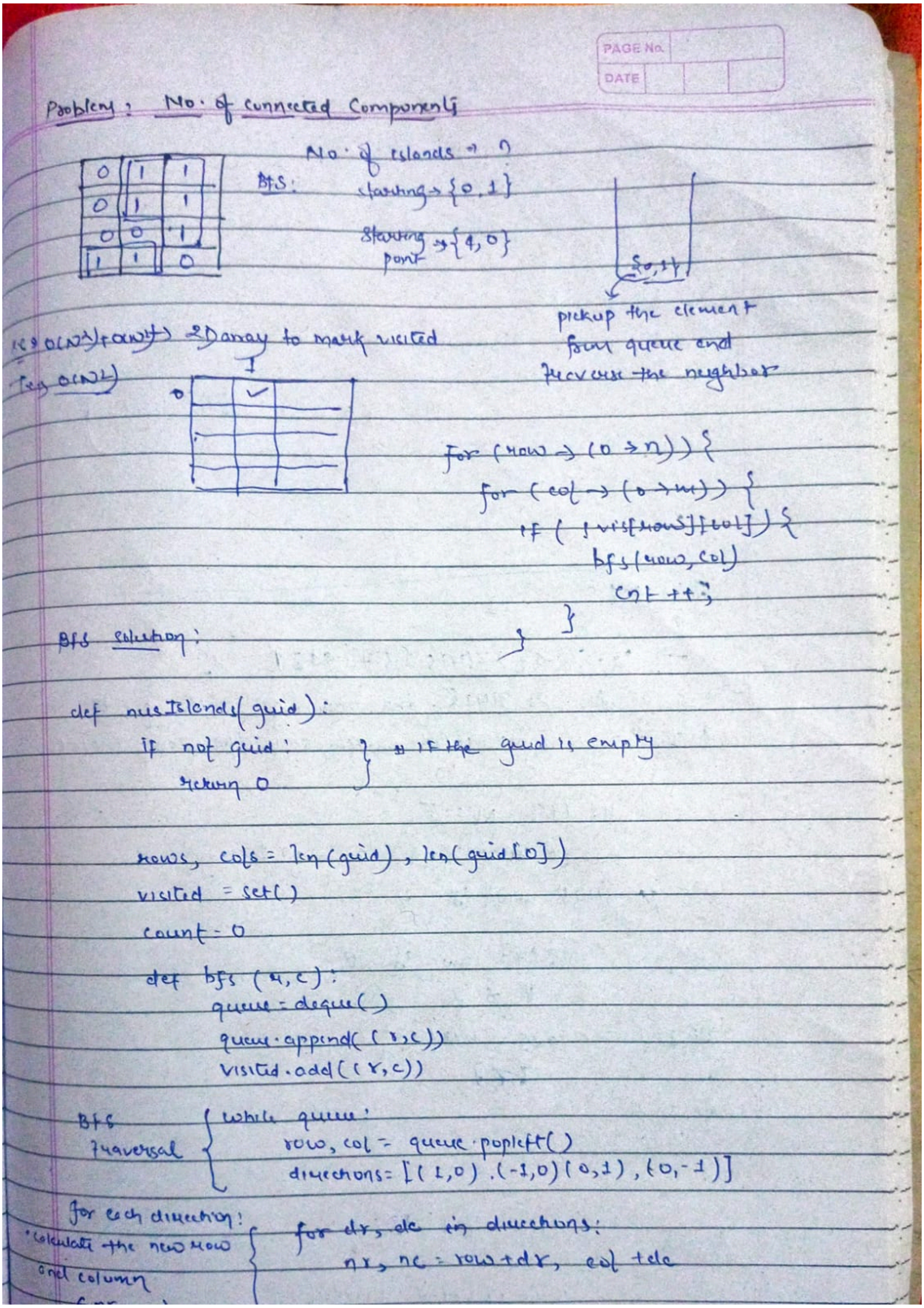
visited array ⇒

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

```
for ( i = 1 ; i <= V ; i++) {
    if ( vis[i] == 0) {
        dfs(i); // bfs(i) } }
```

Leetcode : 547 :

```
def FindCircleNum( self, isConnected : List[List [int]]) -> int:
    def dfs( city):
        visited [city] = True
        for neighbor in range (n):
            if isConnected [city][neighbor] == 1 and not
                visited [neighbor]:
                dfs( neighbor)

    n = len ( isConnected)
    visited = [False] * n
    provinces = 0

    for city in range (n)
        if not visited [city]:
            dfs(city)
            provinces += 1

    return provinces.
```



PAGE No.

DATE

Problem : No. of connected Components

0	1	1
0	1	1
0	0	1
1	1	0

No. of islands → ?

BFS: starting → {0, 1}

Starting point → {4, 0}

{0, 1}

pickup the element from queue and traverse the neighbor

TC : O(N²) + O(N⁴) 2D array to mark visited

SC O(N²)

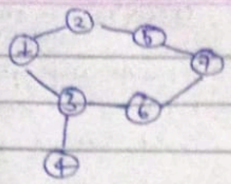
		1	
0		✓	

```
for (row → (0 → n)) {
    for (col → (0 → m)) {
        if ( !vis[rows][col]) {
            bfs(row, col)
            cnt ++;
        }
    }
```

Bfs Solution:

```
def numIslands(grid):
    if not grid:        } # If the grid is empty
        return 0

    rows, cols = len(grid), len(grid[0])
    visited = set()
    count = 0

    def bfs (r, c):
        queue = deque()
        queue.append( (r,c))
        visited.add((r,c))

BFS          { while queue:
Traversal    {        row, col = queue.popleft()
             {        directions= [(1,0) .(-1,0) (0,1) , (0,-1)]

For each direction:
• Calculate the new row     { for dr, dc in directions:
  and column                {     nr, nc = row + dr, col + dc
```

Detect cycle in undirected graph



adj. list

1 - {2, 3}
2 - {1, 5}
3 - {1, 4, 6}
4 - {3}
5 - {2, 7}
6 - {3, 7}
7 - {5, 6}

TC → O(N+2E)
SC → O(N).

Multiple Components

```
for (i=1 ; i<=N; i++)
{   if (!vis[i]) {
        if (detectCycle (i) == true
        } return true;
    }
return False
```

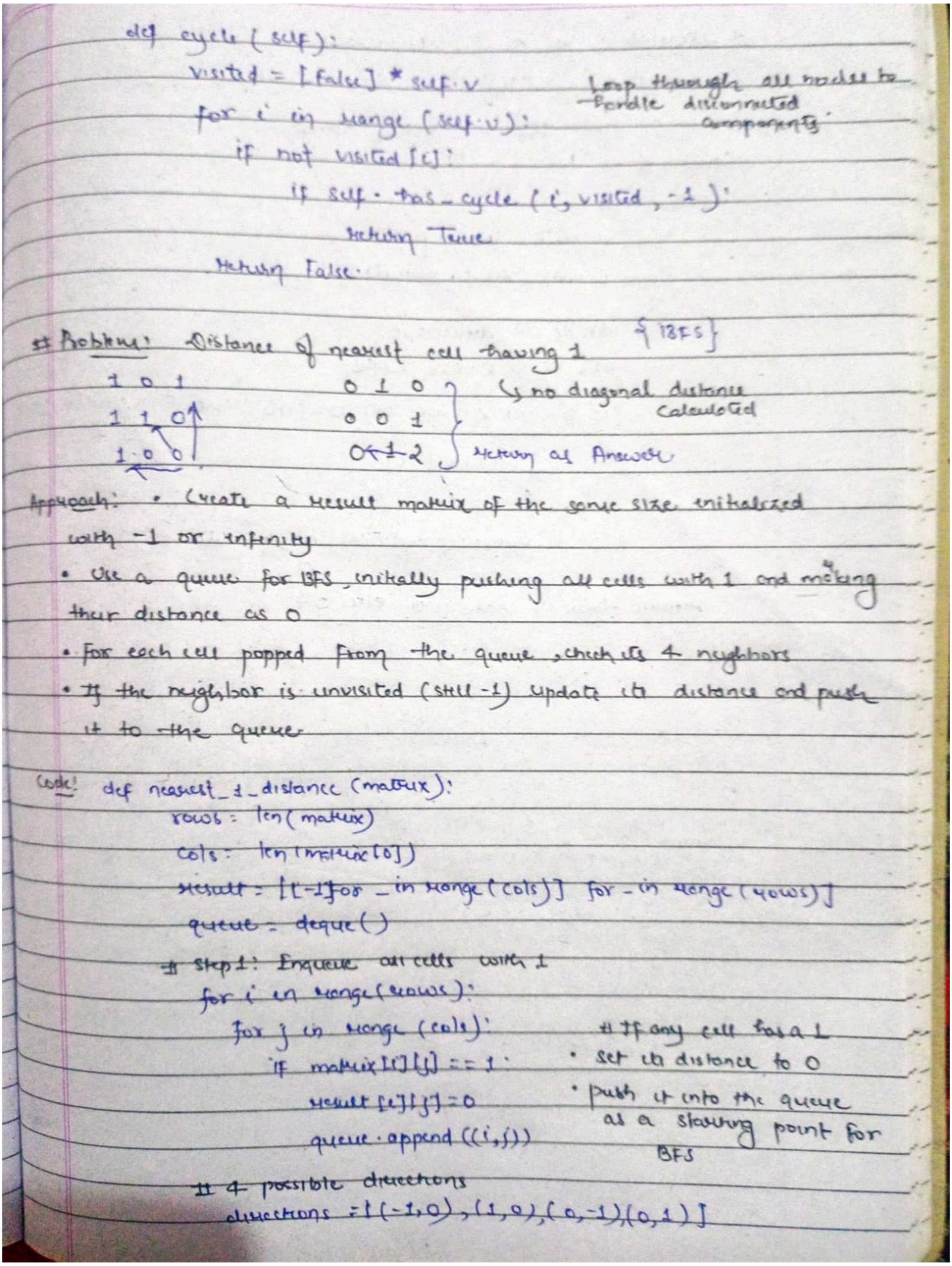
Code:

```
def has_cycle (self):
    visited = [false] * self.v

    for start in range (self.v):
        if not visited [start]:
            queue = deque( )
            queue.append ((start, -1)) # current node, parent node
            visited [start] = True,
```

- Dequeue the front and get its parent

```
            while queue:
                current, parent = queue.popleft( )

                for neighbor in self.graph [current]:
                    if not visited [neighbor]:
                        visited [neighbor] = True

                        queue.append((neighbor, current)

                    elif neighbor != parent
                        return True

    return False
```

- If not visited
  → Mark it visited
  → Add it to the queue with current node as its parent
- If the neighbor is visited and not the parent then there's cycle.



```
def cycle(self):
    visited = [False] * self.v                 Loop through all nodes to
    for i in range(self.v):                    handle disconnected
        if not visited[i]:                          components
            if self.has_cycle(i, visited, -1):
                return True
    return False.
```

Problem: Distance of nearest cell having 1 {BFS}

```
1 0 1          0 1 0 }  ↳ no diagonal distance
1 1 0          0 0 1 }            Calculated
1 0 0          0 1 2 }  return as Answer
```

Approach: • Create a result matrix of the same size initialized with -1 or infinity

- Use a queue for BFS, initially pushing all cells with 1 and making their distance as 0
- For each cell popped from the queue, check its 4 neighbors
- If the neighbor is unvisited (still -1) update its distance and push it to the queue

Code:
```
def nearest_1_distance(matrix):
    rows = len(matrix)
    cols = len(matrix[0])
    result = [[-1 for _ in range(cols)] for _ in range(rows)]
    queue = deque()
    # Step 1: Enqueue all cells with 1
    for i in range(rows):
        for j in range(cols):                 # If any cell has a 1
            if matrix[i][j] == 1:             • set its distance to 0
                result[i][j] = 0              • push it into the queue
                queue.append((i,j))             as a starting point for
                                                  BFS
    # 4 possible directions
    directions = [(-1,0), (1,0), (0,-1)(0,1)]
```