Experiment no: 1 **Date:**

BFS & DFS

Aim: To implement Breadth First Search and Depth First Search algorithm

Theory :

Breadth-First Search (BFS)

Breadth-First Search (BFS) is a graph traversal algorithm that systematically explores a graph level by level. It starts from the source node and visits all neighbors of the current node before moving on to the next level. BFS is widely used in various applications, including network routing, shortest path finding, and state space exploration in Artificial Intelligence.

Properties:

1. Solution Quality: BFS guarantees the shortest path in an unweighted graph, making it suitable for finding optimal solutions.

2. Completeness: BFS is complete, meaning it will find a solution if one exists.

3. Time Complexity: In the worst case, BFS has a time complexity of O(V + E), where V is the number of vertices and E is the number of edges.

4. Space Complexity: BFS has a space complexity of O(V), as it needs to store all vertices in the queue during traversal.

Applications:

1. Shortest Path Finding: BFS is used to find the shortest path between two nodes in an unweighted graph.

2. Network Routing: BFS helps in determining the optimal route in computer networks.

3. Maze Solving: It can be applied to solve mazes by exploring possible paths in a systematic manner.

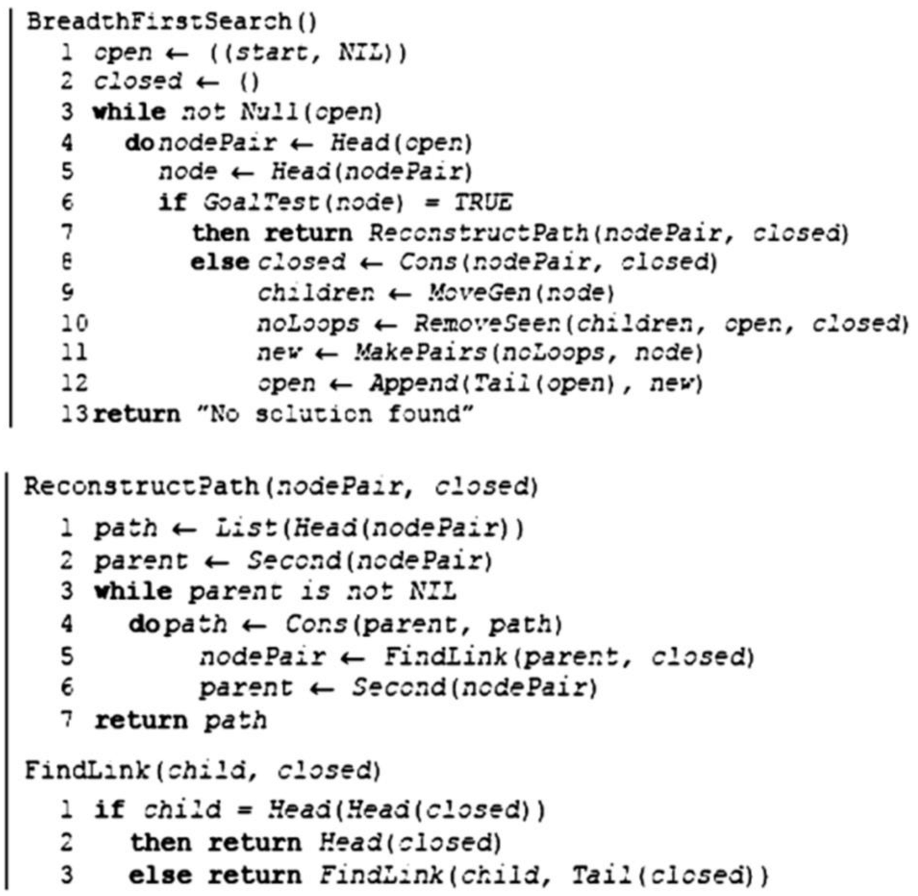
Advantages:

1. Guarantees optimal solutions in unweighted graphs.
2. Complete - will find a solution if one exists.
3. Suitable for scenarios where the shortest path is crucial.

Limitations:

1. Higher memory requirements compared to DFS.
2. May not be efficient for large graphs with varying edge weights.

Algorithm:



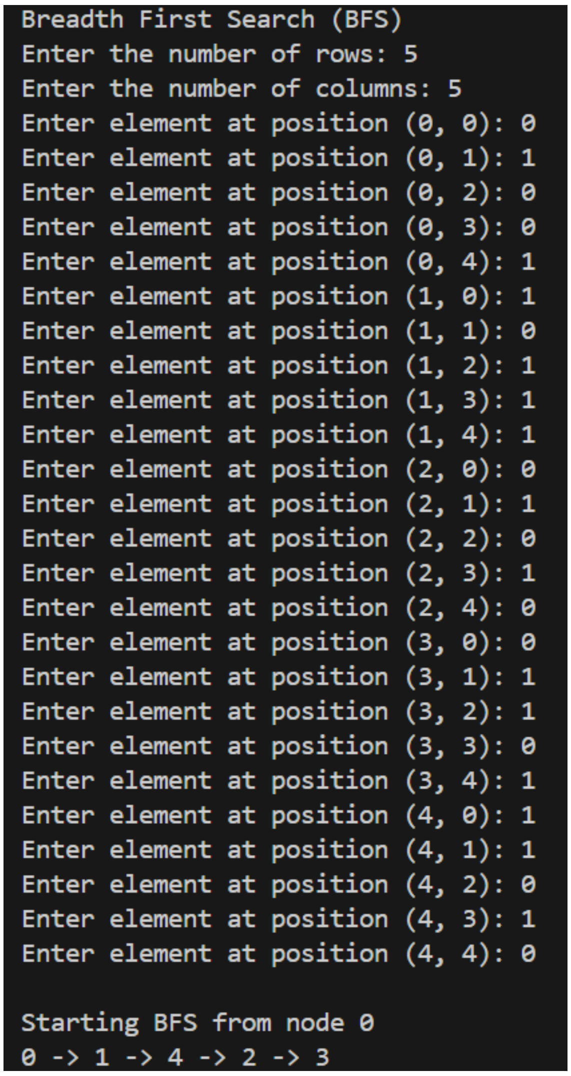
```
BreadthFirstSearch()
  open ← ((start, NIL))
  closed ← ()
  while not Null(open)
    do nodePair ← Head(open)
       node ← Head(nodePair)
       if GoalTest(node) = TRUE
         then return ReconstructPath(nodePair, closed)
         else closed ← Cons(nodePair, closed)
              children ← MoveGen(node)
              noLoops ← RemoveSeen(children, open, closed)
              new ← MakePairs(noLoops, node)
              open ← Append(Tail(open), new)
  return "No solution found"

ReconstructPath(nodePair, closed)
  path ← List(Head(nodePair))
  parent ← Second(nodePair)
  while parent is not NIL
    do path ← Cons(parent, path)
         nodePair ← FindLink(parent, closed)
         parent ← Second(nodePair)
  return path

FindLink(child, closed)
  if child = Head(Head(closed))
    then return Head(closed)
    else return FindLink(child, Tail(closed))
```

Code:

```
from collections import deque
print('Breadth First Search (BFS)')
rows = int(input("Enter the number of rows: "))
cols = int(input("Enter the number of columns: "))
matrix = []
for i in range(rows):
   row = []
   for j in range(cols):
      element = int(input(f"Enter element at position ({i}, {j}): "))
      row.append(element)
   matrix.append(row)
def bfs(matrix, start):
   num_vertices = len(matrix)
   visited = [False] * num_vertices
```

```
    queue = deque([start])
    visited[start] = True
    while queue:
        current_node = queue.popleft()
        if current_node==0:
            print(f"{current_node}",end="")
        else:
            print(f" -> {current_node}",end="")
        for neighbor in range(num_vertices):
            if matrix[current_node][neighbor] == 1 and not visited[neighbor]:
                queue.append(neighbor)
                visited[neighbor] = True
print("\nStarting BFS from node 0")
bfs(matrix, 0)
```

Output:



```
Breadth First Search (BFS)
Enter the number of rows: 5
Enter the number of columns: 5
Enter element at position (0, 0): 0
Enter element at position (0, 1): 1
Enter element at position (0, 2): 0
Enter element at position (0, 3): 0
Enter element at position (0, 4): 1
Enter element at position (1, 0): 1
Enter element at position (1, 1): 0
Enter element at position (1, 2): 1
Enter element at position (1, 3): 1
Enter element at position (1, 4): 1
Enter element at position (2, 0): 0
Enter element at position (2, 1): 1
Enter element at position (2, 2): 0
Enter element at position (2, 3): 1
Enter element at position (2, 4): 0
Enter element at position (3, 0): 0
Enter element at position (3, 1): 1
Enter element at position (3, 2): 1
Enter element at position (3, 3): 0
Enter element at position (3, 4): 1
Enter element at position (4, 0): 1
Enter element at position (4, 1): 1
Enter element at position (4, 2): 0
Enter element at position (4, 3): 1
Enter element at position (4, 4): 0

Starting BFS from node 0
0 -> 1 -> 4 -> 2 -> 3
```

Depth-First Search (BFS)

Depth-First Search (DFS) is another fundamental graph traversal algorithm that explores a graph by going as deep as possible along each branch before backtracking. DFS can be implemented using a stack or recursion. It is widely used in various applications, including maze solving, topological sorting, and certain scenarios in Artificial Intelligence.

Properties:

1. Solution Quality: DFS does not guarantee the shortest path in a general graph and is usedwhen finding any solution is acceptable.

2. Completeness: DFS may not find a solution if one exists, especially if not properlyimplemented to handle cycles.

3. Time Complexity: In the worst case, DFS has a time complexity of O(V + E), similar to BFS.

4. Space Complexity: DFS has a space complexity of O(V) for the recursive version but can bemore memory-efficient using iterative methods.

Applications:

1. Maze Solving: DFS is used to explore possible paths in a maze.

2. Topological Sorting: It is applied to order nodes in a directed acyclic graph.

3. Backtracking Problems: DFS is suitable for problems that involve making choices andbacktracking.

Advantages:

1. Memory-efficient, especially in certain iterative implementations.

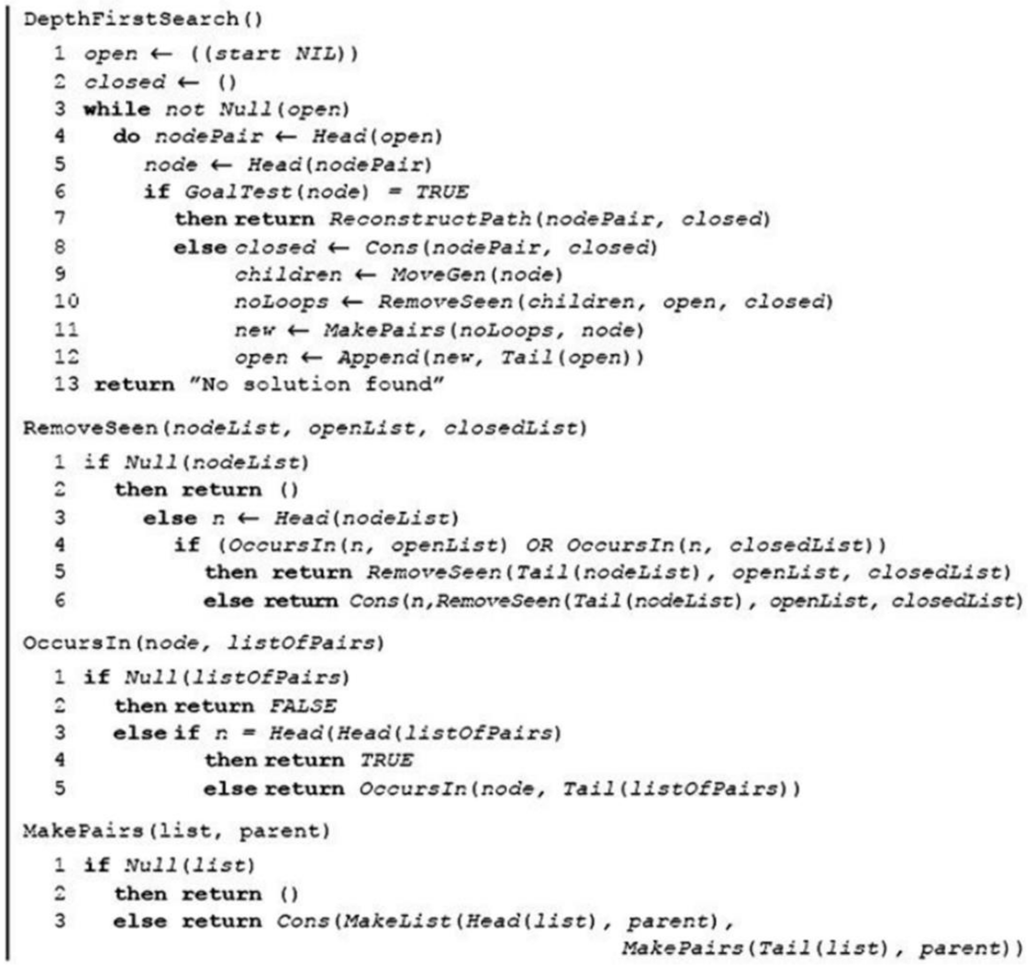
2. Well-suited for problems involving backtracking and deep exploration.

Limitations:

1. Does not guarantee optimal solutions.

2. May get stuck in infinite loops if not properly implemented to handle cycles.

Algorithm:

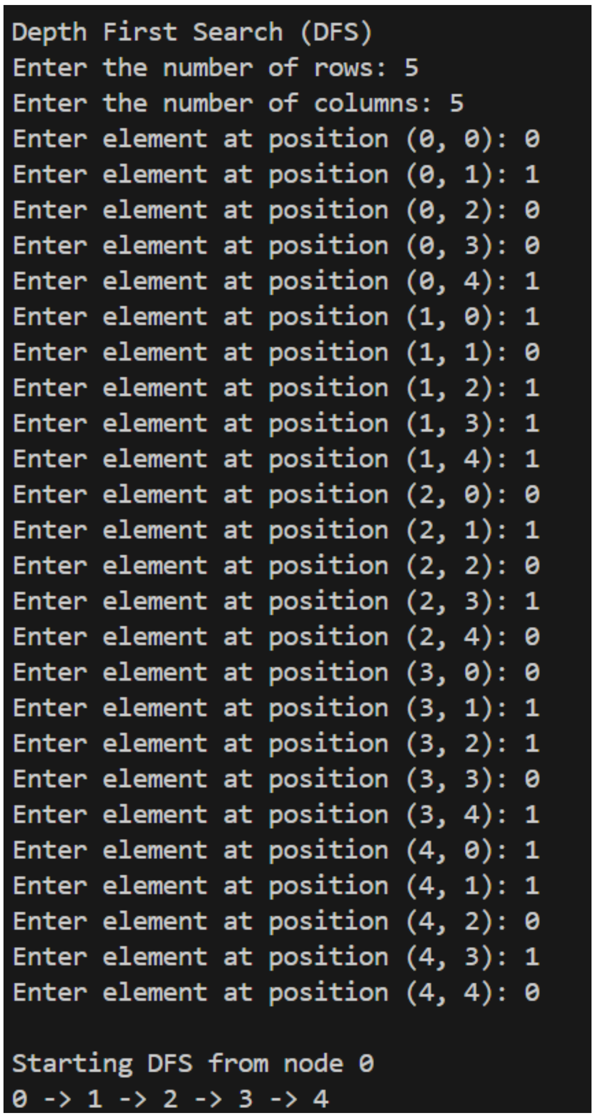


```
DepthFirstSearch()
  open ← ((start NIL))
  closed ← ()
  while not Null(open)
    do nodePair ← Head(open)
      node ← Head(nodePair)
      if GoalTest(node) = TRUE
        then return ReconstructPath(nodePair, closed)
        else closed ← Cons(nodePair, closed)
            children ← MoveGen(node)
            noLoops ← RemoveSeen(children, open, closed)
            new ← MakePairs(noLoops, node)
            open ← Append(new, Tail(open))
  return "No solution found"

RemoveSeen(nodeList, openList, closedList)
  if Null(nodeList)
    then return ()
      else n ← Head(nodeList)
        if (OccursIn(n, openList) OR OccursIn(n, closedList))
          then return RemoveSeen(Tail(nodeList), openList, closedList)
          else return Cons(n,RemoveSeen(Tail(nodeList), openList, closedList)

OccursIn(node, listOfPairs)
  if Null(listOfPairs)
    then return FALSE
    else if n = Head(Head(listOfPairs)
          then return TRUE
          else return OccursIn(node, Tail(listOfPairs))

MakePairs(list, parent)
  if Null(list)
    then return ()
    else return Cons(MakeList(Head(list), parent),
                                      MakePairs(Tail(list), parent))
```

Code

```
from collections import deque
print('Depth First Search (DFS)')
rows = int(input("Enter the number of rows: "))
cols = int(input("Enter the number of columns: "))
matrix = []
for i in range(rows):
  row = []
  for j in range(cols):
    element = int(input(f"Enter element at position ({i}, {j}): "))
    row.append(element)
  matrix.append(row)
def dfs(matrix, start):
  num_vertices = len(matrix)
  visited = [False] * num_vertices
  stack = [start]
  visited[start] = True
  while stack:
    current_node = stack.pop()
    if current_node==0:
            print(f"{current_node}",end="")
        else:
            print(f" -> {current_node}",end="")
        for neighbor in range(num_vertices - 1, -1, -1):
            if matrix[current_node][neighbor] == 1 and not visited[neighbor]:
                stack.append(neighbor)
                visited[neighbor] = True
print("\nStarting DFS from node 0")
dfs(matrix, 0)
```

Output:



```
Depth First Search (DFS)
Enter the number of rows: 5
Enter the number of columns: 5
Enter element at position (0, 0): 0
Enter element at position (0, 1): 1
Enter element at position (0, 2): 0
Enter element at position (0, 3): 0
Enter element at position (0, 4): 1
Enter element at position (1, 0): 1
Enter element at position (1, 1): 0
Enter element at position (1, 2): 1
Enter element at position (1, 3): 1
Enter element at position (1, 4): 1
Enter element at position (2, 0): 0
Enter element at position (2, 1): 1
Enter element at position (2, 2): 0
Enter element at position (2, 3): 1
Enter element at position (2, 4): 0
Enter element at position (3, 0): 0
Enter element at position (3, 1): 1
Enter element at position (3, 2): 1
Enter element at position (3, 3): 0
Enter element at position (3, 4): 1
Enter element at position (4, 0): 1
Enter element at position (4, 1): 1
Enter element at position (4, 2): 0
Enter element at position (4, 3): 1
Enter element at position (4, 4): 0

Starting DFS from node 0
0 -> 1 -> 2 -> 3 -> 4
```

Conclusion: The Algorithms for Breadth First Search and Depth First Search was implemented and executed successfully.