**EXPERIMENT NO: 01 DATE: 06/02/24**

**AIM:** Write a program to implement

1. BFS Algorithm
2. DFS Algorithm

**Theory:**

1. Breadth First Search (BFS)

Breadth first search is a general technique of traversing a graph. Breadth first search may use more memory but will always find the shortest path first. In this type of search the state space is represented inform of a tree. The solution is obtained by traversing through the tree. The nodes of the tree represent the start value or starting state, various

intermediate states and the final state. In this search a queue data structure is used and it is level by level traversal. Breadth first search expands nodes in order of their distance from the root. It is a path finding algorithm that is capable of always finding the solution if one exists. The solution which is found is always the optional solution. This task is completed in a very memory intensive manner. Each node in the search tree is expanded in a breadth wise at each level.

Advantages:

1. In this procedure at any way it will find the goal.
2. It does not follow a single unfruitful path for a long time.
3. It finds the minimal solution in case of multiple paths.

Disadvantages:

1. BFS consumes large memory space.
2. It can be slow since it expands all the nodes at each level before moving on to the next level
3. It has long pathways, when all paths to a destination are on approximately the same search depth.
4. Depth First Search (DFS)

DFS visits all the vertices in the graph. This type of algorithm always chooses to go deeper into the graph. After DFS visited all the reachable vertices from a particular sources vertex it chooses one of the remaining undiscovered vertices and continues the search. DFS reminds the space limitation of breath first search by always generating next a child of the deepest unexpanded node. The data structure stack or last in first out (LIFO) is used for D

Advantages:

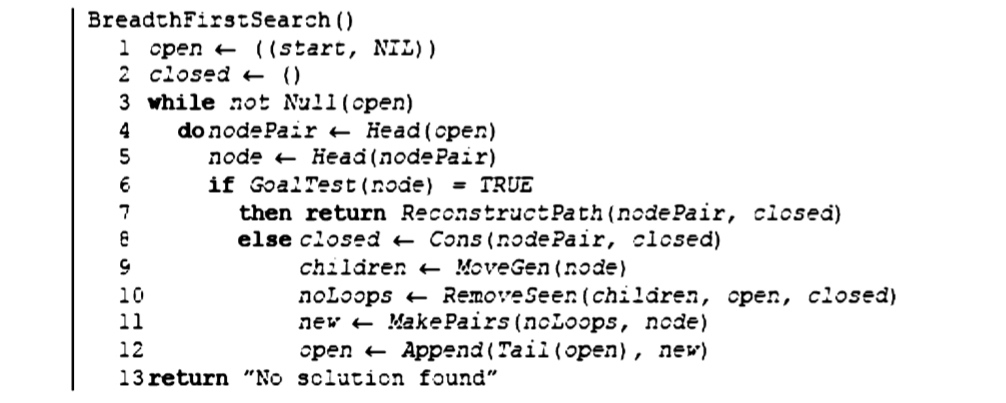
1. DFS consumes very less memory space.
2. It may find a solution without examining much of search because we may get the desired solution in the very first go.

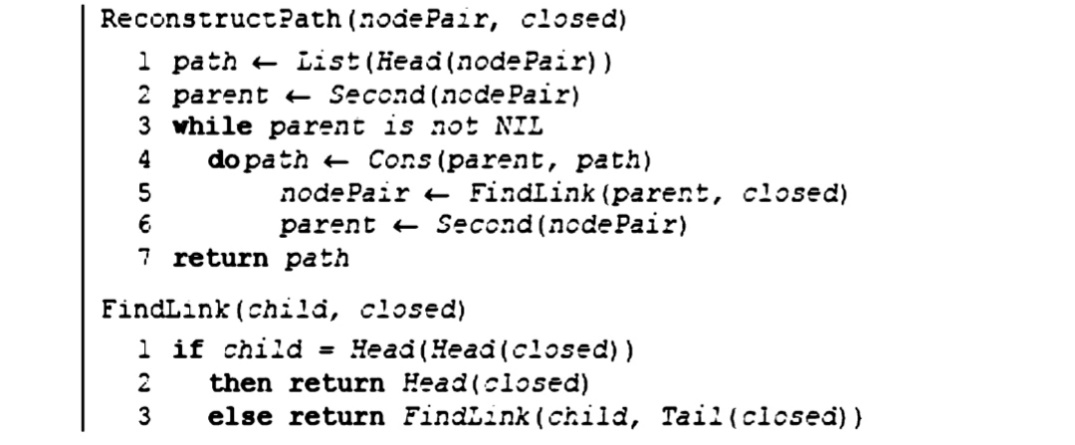
Disadvantages:

1. It is possible that may states keep reoccurring.
2. There is no guarantee of finding the goal node.
3. Sometimes the states may also enter into infinite loops.

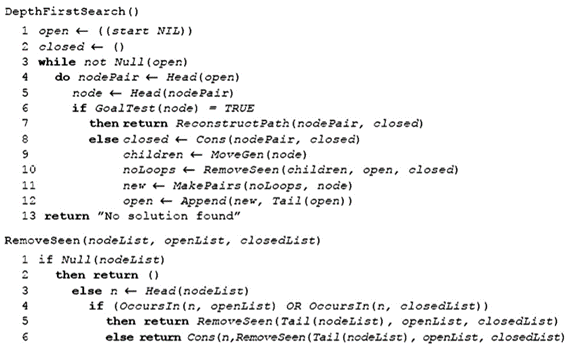
Algorithm:

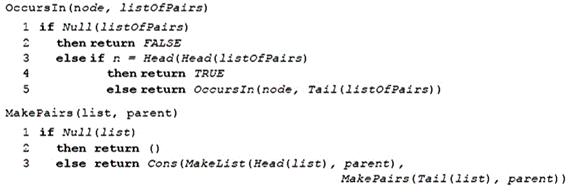
1. **BFS**





1. **DFS**





**Code:**

1. **BFS**

graph = {

'A' : ['B','C','D'],

'B' : ['A','E','F','C'],

'C' : ['A','E','F','G'],

'D' : ['A','G'],

'E' : ['B','H'],

'F' : ['B','C','G','F','I'],

'G' : ['C','G','D','I'],

'H' : ['E','F','I','K','J'],

'I' : ['F','G','F','K'],

'J' : ['J'],

'K' : ['H','I']

}

visited = []

queue = []

def bfs(visited, graph, node):

visited.append(node)

queue.append(node)

while len(queue)>0:

m = queue.pop(0)

print(m, end = " ")

for neighbour in graph[m]:

if neighbour not in visited:

visited.append(neighbour)

queue.append(neighbour)

print("Following is the Breadth-First Search")

bfs(visited, graph, 'A') # function calling

**Output:**

1. **DFS**

# Using a Python dictionary to act as an adjacency list

graph = {

'A' : ['B','C','D'],

'B' : ['A','E','F','C'],

'C' : ['A','E','F','G'],

'D' : ['A','G'],

'E' : ['B','H'],

'F' : ['B','C','G','F','I'],

'G' : ['C','G','D','I'],

'H' : ['E','F','I','K','J'],

'I' : ['F','G','F','K'],

'J' : ['J'],

'K' : ['H','I']

}

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node): #function for dfs

if node not in visited:

print (node)

visited.add(node)

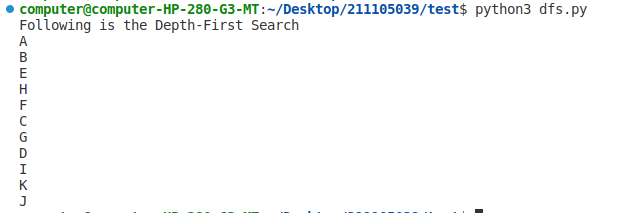
for neighbour in graph[node]:

dfs(visited, graph, neighbour)

print("Following is the Depth-First Search")

dfs(visited, graph, 'A')

**Output:**



**Conclusion:**

Breadth first search and Depth First Search algorithm were studied , and their codes were implemented.