PART A
Experiment No.02

A.1 Aim: To Implement uninformed search methods using C,python or Java.

A.2 Prerequisite: Data Structure, Searching Techniques

A.3 Outcome:

After successful completion of this experiment students will be able to

- Solve the problems like water jug, tic-tac-toe using uninformed search methods like DFS/BFS.

Tools Required: C /Java

A.4 Theory:

A problem determines the graph and the goal but not which path to select from the frontier. This is the job of a search strategy. A search strategy specifies which paths are selected from the frontier. Different strategies are obtained by modifying how the selection of paths in the frontier is implemented.

The uninformed search strategies that do not take into account the location of the goal. Intuitively, these algorithms ignore where they are going until they find a goal and report success.

- Depth-First Search
- Breadth-First Search
- Lowest-Cost-First Search

- Depth-First Search: The first strategy is depth-first search. In depth-first search, the frontier acts like a last-in first-out stack. The elements are added to the stack one at a time. The one selected and taken off the frontier at any time is the last element that was added.

Algorithm:

DEPTH(AdMax): Depth first search in state space

1. Init lists OPEN $\{S_i\}$, CLOSED {}
2. if OPEN = {}
   then return FAIL
3. Remove first node S from OPEN and insert it into CLOSED

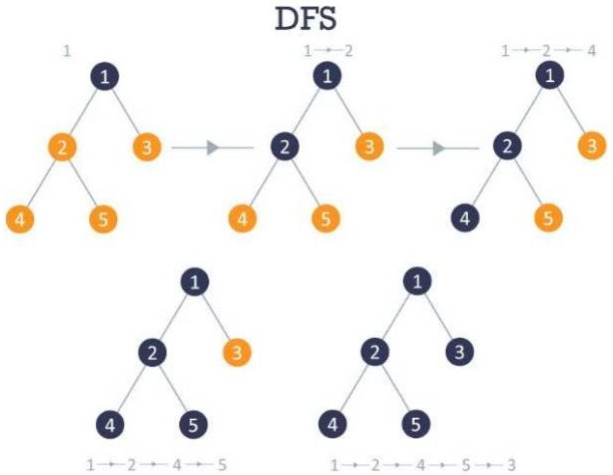
3'. if Ad(S) = AdMaxthen repeat from 2

4. Expand node S
   - 4.1. Generate all successors S_j of node S
   - 4.2. for each succesorS_j of S do
     - 4.2.1. Set link S_j S
     - 4.2.2. if S_j is final state
       then
       - i. Solution is (S_j,.., S_i)
       - ii. return SUCCESS
     - 4.2.3. Insert S_j in OPEN, at the beginning

5. repeat from 2

end.

Example:



- Breadth-First Search : In breadth-first search the frontier is implemented as a FIFO (first-in, first-out) queue. Thus, the path that is selected from the frontier is the one that was added earliest. This approach implies that the paths from the start node are

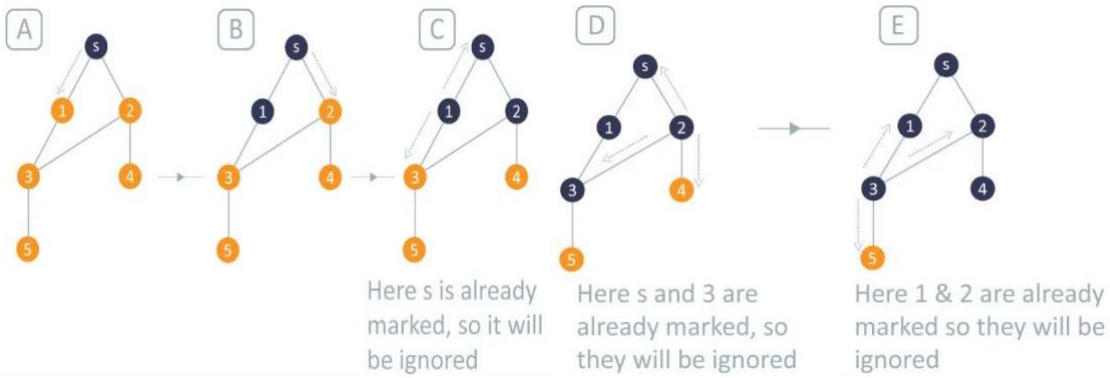
generated in order of the number of arcs in the path. One of the paths with the fewest arcs is selected at each stage.

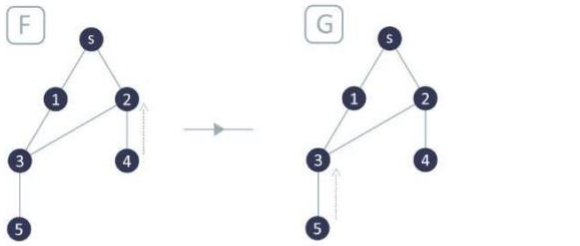
Algorithm BREADTH: Breadth first search in state space

1. Init lists OPEN $\{S_i\}$, CLOSED {}
2. if OPEN = {}
   then return FAIL
3. Remove first node S from OPEN and insert it in CLOSED
4. Expand node S
   - 4.1. Generate all direct successors S_j of node S
   - 4.2. for each successor S_j of S do
     - 4.2.1. Make link S_j S
     - 4.2.2. if S_j is final state
       then
       i. Solution is (S_j, S, .., S_i)
       ii. return SUCCESS
     - 4.2.3. Insert S_j in OPEN, at the end
5. repeat from 2

end.

Example:





Here 2 is already marked, so it will be ignored

Here 3 is already marked, so it will be ignored

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

Roll. No. B30	Name: Bhatt Pranjal
Class : TE B COMPS	Batch: B2
Date of Experiment: 21-01-2025	Date of Submission: 10-02-2025
Grade:	

B.1 Software Code written by student: BFS =>

```
from collections import deque

# BFS function def
bfs(graph, start, goal):
   visited = set()  # Set to keep track of visited nodes
queue = deque([start])  # Queue for BFS     path = []  # To
store the traversal path

   while queue:
      node = queue.popleft()
if node not in visited:
visited.add(node)
          path.append(node)

          # Check if goal is found
if node == goal:               print("Goal
found:", path)              return path

          # Enqueue all the neighbors of the current node
```

```
        for neighbor in graph[node]:
if neighbor not in visited:
queue.append(neighbor)

   print("Goal not found")
return None

# Example usage graph =
{
   '1': ['2', '3'],
   '2': ['1', '4', '5'],
   '3': ['1', '6'],
   '4': ['2'],
   '5': ['2', '6'],
   '6': ['3', '5'] }

start = '1' goal = '6'
print("Breadth-First Search:")
bfs(graph, start, goal)
```

DFS

```
# DFS function def dfs(graph, start, goal, visited=None,
path=None):    if visited is None:
      visited = set()  # Set to keep track of visited nodes    if
path is None:        path = []  # List to store the path

   visited.add(start)    path.append(start)

   # Check if goal is found    if
start == goal:        print("Goal
found:", path)        return path

   # Recurse for all the neighbors
for neighbor in graph[start]:        if
neighbor not in visited:
        result = dfs(graph, neighbor, goal, visited, path)
if result:  # If result is not None, goal is found
return result
   path.pop()   # Backtrack if goal is not found
return None graph = {
   '1': ['2', '3'],
   '2': ['1', '4', '5'],
   '3': ['1', '6'],
   '4': ['2'],
```

```
    '5': ['2', '6'],
    '6': ['3', '5']
}

start = '1' goal = '6'
print("Depth-First Search:")
dfs(graph, start, goal)
```

B.2 Input and Output:

BFS:

```
Breadth-First Search:
Goal found: ['1', '2', '3', '4', '5', '6']
['1', '2', '3', '4', '5', '6']
```

DFS :

```
Depth-First Search:
Goal found: ['1', '2', '5', '6']
['1', '2', '5', '6']
```

B.3 Observations and learning:

Uninformed search methods, such as Breadth-First Search (BFS) and DepthFirst Search (DFS), are fundamental techniques used in AI for problem-solving, where the search algorithm explores the solution space without any prior knowledge of the goal

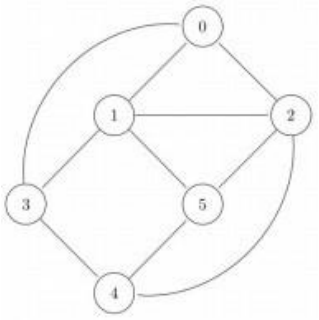
B.4 Conclusion:

The implementation of uninformed search methods highlights the efficiency and flexibility offered by Python, especially in terms of code simplicity and readability, while C provides more control over memory management and performance optimization

B.5 Question of Curiosity

(To be answered by student based on the practical performed and learning/observations)

Q1) Which sequence corresponds to that of depth first search for the graph given below. The search starts at vertex 0 and lexicographic ordering is assumed for the edges emanating from each vertex.



A. 0 1 2 4 3 5
B. 0 1 2 5 4 3
C. 0 1 2 3 4 5
D. 0 1 3 4 2 5

Q2) Given a rooted tree, one desires to find the shortest path from the root to a given node v. Which algorithm would one use to find this shortest path?
A.DFS
B. BFS
C. Either BFS or DFS

Q3) Consider a graph G. Let T be a BFS tree with root r. Let d(u,v) denote the length of the shortest path between the nodes u and v. If v is visited before u in the breadth first search traversal, which of the following statements is true ?
A. d(r,v) > d(r,u)
B. d(r,v) = d(r,u)
C. d(r,v) < d(r,u)
D. insufficient information to comment on d(r,v) and d(r,u)