Unit = 02

Search Algorithms

Random search, search with closed and open list, Depth first and Breadth first search, Heuristic search, Best first search, A* algorithm, Game search.

Random Search.

- Random search is a blind, uninformed search strategy where solutions are generated randomly until a goal state is found.
- It does not use any domain knowledge or heuristics.
- Often considered a baseline method in AI problem-solving

Working Principle

Generate possible states or actions at random.
Test each state against the goal condition.
If the goal is found → stop; otherwise, continue generating.
It is like a brute force search but without systematically exploring the search space.

Characteristics.

Uninformed :- No guidance toward the goal.

Rajshree
PAGE NO.:
DATE:

Non-deterministic :- Each run may follow a different path.
Inefficient :- Can take a very long time, especially for large state spaces.

Algorithm

Step 1: Input the problem definition (state space, goal test).
Step 2: Repeat the following until a solution is found or a limit is reached:
- Randomly generate a candidate state from the state space.
- Test if this candidate is a goal state.
  - If yes → return the solution.
  - If no → discard it and continue.

Step 3: If search limit (time/memory/iteration) is reached and no solution found, return failure.

Code:-

```
def random_search(fitness_fn, bounds, max_iters):
    import random
    b
    best_solution = None
    best_score = float('inf')

    for _ in range(max_iters):

        candidate = [random.uniform(low, high) for
        (low, high) in bounds]
```

```
score = fitness_fn(candidate)
if score < best_score:
    best_score = score
    best_solution = candidate

return best_solution, best_score
```

~~Search with closed and open list~~

Advantages

- Extremely simple to implement.
- Works even when no heuristic information is available
- Sometimes finds a solution quickly in highly random environments.

~~Depth first search~~

- Depth first search (DFS) is an uninformed search strategy that explores a state space by expanding the ~~datase~~ deepest unvisited node first.
- It goes deep along one branch until it reaches a goal state or a dead-end, then backtracks and explores other branches.
- DFS uses a stack data structure (LIFO) for implementation.

Algorithm

Step 1: Start with the initial state (root node).
Step 2: If the node is a goal → return success.
Step 3: If not, expand the node by generating successors.
Step 4: Push successors onto the stack.
Step 5: Pop the top node from the stack and repeat.
Step 6:- If stack becomes empty → return failure.

OR

Step 1: SET ~~status~~ STATUS = 1 (ready state) for each node in G
Step 2: Push the starting node A on the stack and set its
~~status~~ STATUS = 2 (waiting state).
Step 3: Repeat steps 4 and 5 until STACK is empty.
Step 4: Pop the Top node N. Process it and set its STATUS =
(Processed state)
Step 5: Push on the stack all the neighbors of N that
are in the ready state (whose STATUS = 1) and set
their STATUS = 2 (waiting state)

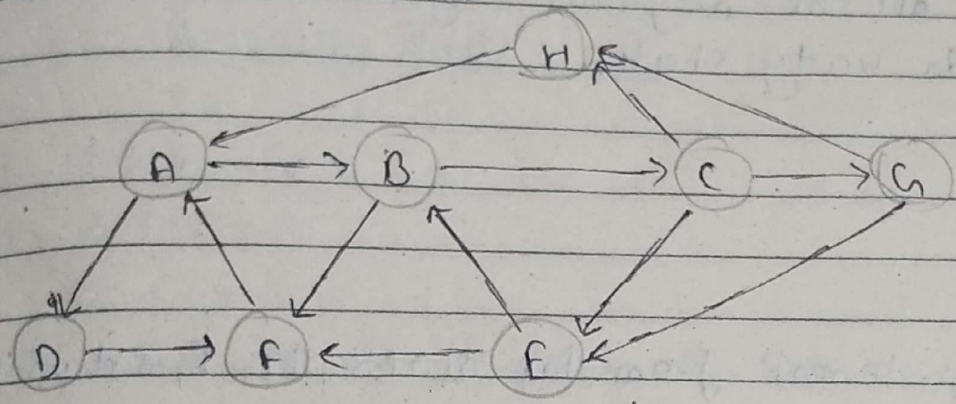
[END OF LOOP]
Step 6: EXIT.

Code:

```
from collections import deque
```

```
def dfs_queue(graph, start):
    visited = set()
    stack = deque([start])
    result = []

    while stack:
        node = stack.pop()
        if node not in visited:
            visited.add(node)
            result.append(node)
            stack.extend(reversed(graph[node]))

    return result

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

dfs_result = dfs_queue(graph, 'A')
print("DFS Traversal:")
print(dfs_result)
```

Example : There is a directed graph having 7 vertices.



Adjacency lists
A : B, D.
B : C, F
C : E, G, H
G : E, H.
E : B, F
F : A
D : F
H : A

Now, start examining the graph starting from Node H.

Step1 : First, push H onto the stack

1. STACK : H

Step2 : Pop the Top element from the stack, i.e. H, and print it. Now, PUSH all the neighbors of H onto the stack that are in ready state.

1 print : H

2 STACK : A

Step3 : Pop the Top element from the stack, i.e., A, and print it. Now, PUSH all the neighbors of A onto the stack that are in ready state.

1. Print : A

2. STACK : B, D.

Step 4 :- POP the top element from the stack, i.e. D, and print it. Now, PUSH all the neighbors of D onto the stack that are in ready state.

1. Print : D
2. STACK : B, F

Step 5 :- POP the top element from the stack, i.e. F, and print it. Now, PUSH all the neighbors of F onto the stack that are in ready state.

1. Print : F
2. STACK : B

Step 6 :- POP the top element from the stack i.e. B, and print it. Now, PUSH all the neighbors of B onto the stack that are in ready state.

1. Print : B
2. STACK : C

Step 7 :- POP the element from the stack i.e. C, and print it. Now, PUSH all the neighbors of C onto the stack that are in ready state.

1. Print : C
2. STACK : E, G

Rajshree
PAGE NO.:
DATE:

Step 8:- POP, the element from the stack. i.e. G, and PUSH all the neighbors of G onto the stack that are in ready state.

1. Print : G
2. STACK : E

Step 9:- POP, the element from the stack, i.e. E and Push all the neighbors of E onto the stack that are in ready state.

1. Print : E
2. STACK :

Now, all the graph nodes have been traversed, and the stack is empty.

Complexity of DFS.

The time complexity of the DFS algorithm is O(V+E), where V is the number of vertices and E is the no. of edges in the graph.

The space complexity of DFS algorithm is O(V)

Rajshree
PAGE NO.:
DATE:

Applications :-

- Solving Puzzle problems (like mazes).
- Pathfinding in game AI.
- Used in topological sorting and cycle detection in graphs.

Breadth first Search

- BFS is an uniformed search strategy.
- It explores the search tree level by level, expanding all nodes at a given depth before moving to the next.
- Implemented using a queue (FIFO) data structure.
- Guarantees finding the shortest path (minimum depth solution) if step cost = 1.

Algorithm

The steps involved in the BFS algorithm to explore a graph are given as follow :-

Step 1 :- SET STATUS = 1 (ready state) for each node in G.
Step 2 :- Enqueue the starting node A and set its STATUS = 2 (waiting state)
Step 3 :- Repeat steps 4 and 5 until QUEUE is empty.
Step 4 :- Dequeue a node N. Processes it and set its STATUS = 3 (processed state)
Step 5 :- Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set.

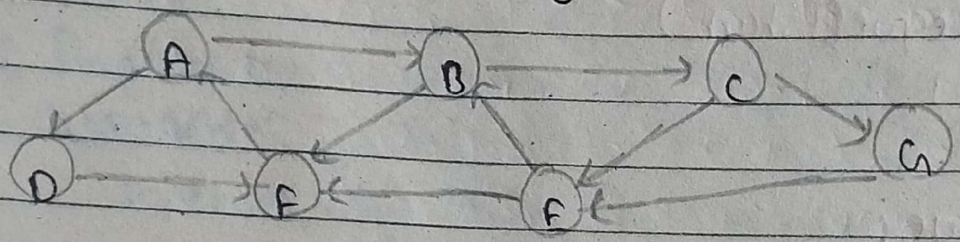
Their STATUS = 2
(waiting state)
(END OF LOOP)
Step 6:- EXIT

Code:-

```
from collection import deque

def Bfs_queue(graph, start):
    visited = set()
  queue stack = deque([start])
    result = []

    while queue:
      node = queue.popleft()
      if node not in visited:
          visited.add(node)
          result.append(node)
          queue.extend(graph[node])

    return result

graph = {
  'A': ['B', 'C'],
  'B': ['D', 'E'],
  'C': ['F'],
  'D': []
```

```
    'E' : ['F'],
    'F' : []
}

bfs_result = bfs_queue(graph, 'A')
print("\nBFS Traversal: ")
print(bfs_result).
```

Example: working of BFS algo by using an example.
In the example given below, there is a directed graph having 7 vertices.



Adjacency lists.
A: B, D.
B: C, F
C: E, G
G: E
E: B, F
F: A
D: F

Now, start examining the graph starting from node A.

Step1: First, add A to queue1 and NULL to queue2
1. QUEUE1 = {A}
2. QUEUE2 = {NULL}

Step2:- Now, delete node A from queue1 and add it into queue2. Insert all neighbours of node A to queue1.
1. QUEUE1 = {B, D}
2. QUEUE2 = {NULL A}.

Step 3: Now, delete node B from queue1 and add it into queue2. Insert all neighbors of node B to queue1.

1. QUEUE1 = {D, C, F}
2. QUEUE2 = {A, B}

Step 4:- Now delete node D from queue1 and add it into queue2. Insert all neighbors of node D to queue1. The only neighbor of Node D is F. Since it is already inserted, so it will not be inserted again.

1. QUEUE1 = {C, F}
2. QUEUE2 = {A, B, D}

Step 5:- Delete node C from queue1 and add it into queue2. Insert all neighbors of node C to queue1.

1. QUEUE1 = {F, E, G}
2. QUEUE2 = {A, B, D, G}

Step 6:- Delete node F from queue1 and add it into queue2. Insert all neighbors of node F to queue1 since all the neighbors of node F are already present, we will not insert them again.

1. QUEUE1 = {E, G}
2. QUEUE2 = {A, B, D, C, F}

Step 7 :- Delete node E from queue1. since all of its neighbors have already been added, so we will not insert them again. Now, all the nodes are visited and the target node E is encountered into queue2.

1. QUEUE 1 = {G}
2. QUEUE 2 = {A, B, D, C, F, E}

Complexity of BFS.

Time complexity of BFS depends upon the data structure used to represent the graph. The time complexity of BFS algorithm is O(V+E), since in the worst-case, BFS algorithm explores every node and edge. In a graph, the number of vertices is O(V), whereas the number of edges is O(E).

The space complexity of BFS can be expressed as O(V), where V is the number of vertices.

Hurisric Search.

A heuristic is a rule of thumb, educated guess, or strategy that helps in problem-solving when an exhaustive search is impractical.
In AI, Heuristic search uses domain-specific knowledge

to guide the search process more efficiently toward the goal state
Unlike uninformed searches (e.g. BFS, DFS), heuristic search reduces time and space complexity by prioritizing promising paths.

Key Concept

Heuristic function (h(n)):
Estimates the cost (distance, steps, etc), from the current node n to the goal.
Ex: In a pathfinding problem, straight-line-distance is often used as a heuristic.
- Admissible Heuristic: Never overestimates the cost to reach the goal.
- Informed Search: Search guided by heuristic information.

Common Heuristic Search Algorithms.

1. Best first Search
- Expands the node that appears most promising according to h(n).

2 A* Search.
- Uses evaluation function:-

$f(n) = g(n) + h(n)$.

3. Greedy Best first search.
   - Uses only $h(n)$ to select the next node (may not find an optimal path but faster).

4. Hill climbing.
   - Iteratively moves towards the neighbour with the best heuristic value.
   - Risk:- May get stuck on local maxima or plateaus.

Example:-
problem:- pathfinding on a map.
- start: A, goal: G
- Heuristic $h(n)$ = straight-line distance to G.
- Instead of exploring all possible paths, the search prefers nodes with lowest $h(n)$, leading to fast goal discovery.

Components of Heuristic Search.

State Space:- This implies that the totality of all possible states or settings, which is considered to be the solution for the given problem.

Initial state:- The instance in the search tree of the highest level with no null values, serving as the original state of the problem at hand.

Goal test :- The exploration phase ensures whether the present state is a terminal or containing state in which the problem is solved.

Successor function :- Use create a situation where individual states supplant the current state which represent the possible moves or solution in the problem space.

Heuristic function :- The function of a heuristic is to estimate the value or distance from a given state to the target state. It helps to focus the process on regions or states that has prospect of achieving the goal.