Experiment No.4
Implementation of Bidirectional search for problem solving.
Date of Performance:
Date of Submission:

Aim: Implementation of Bidirectional search for problem solving.

Objective: To study the Bidirectional searching techniques and its implementation for problem solving.

Theory:

Bidirectional search is a graph search algorithm which find smallest path from source to goal vertex. It runs two simultaneous search –

1. Forward search from source/initial vertex toward goal vertex
2. Backward search from goal/target vertex toward source vertex

Bidirectional search replaces single search graph(which is likely to grow exponentially) with two smaller sub

graphs – one starting from initial vertex and other starting from goal vertex. The search terminates when two

graphs intersect.

Algorithm:

Steps for Bidirectional Search Algorithm

1. **Initialization**:
   - Create two frontiers:
     - One for the forward search starting from the initial node (start).
     - One for the backward search starting from the goal node (goal).
   - Create two sets to keep track of visited nodes for each search direction:
     - visited_start for the forward search.
     - visited_goal for the backward search.
   - Initialize the frontiers by adding the start node to frontier_start and the goal node to frontier_goal.
   - Initialize the visited_start and visited_goal sets with their respective starting nodes.
2. **Search Expansion**:
   - Repeat the following steps until a meeting point is found or one of the frontiers is empty:

- **Expand Forward Search**:

  - Remove the current node from frontier_start.

  - Expand all neighboring nodes of the current node.

- For each neighbor:
  - If the neighbor is not in visited_start:
    - Add the neighbor to frontier_start.
    - Mark the neighbor as visited in visited_start.
    - Check if the neighbor is already in visited_goal:
      - If yes, a meeting point is found. Proceed to reconstruct the

        path.

- **Expand Backward Search**:
  - Remove the current node from frontier_goal.
  - Expand all neighboring nodes of the current node.
  - For each neighbor:
    - If the neighbor is not in visited_goal:
      - Add the neighbor to frontier_goal.
      - Mark the neighbor as visited in visited_goal.
      - Check if the neighbor is already in visited_start:
        - If yes, a meeting point is found. Proceed to reconstruct the

          path.

3. **Meeting Point**:
   - The search stops when a node from frontier_start is found in visited_goal or a node from

     frontier_goal is found in visited_start. This node is the meeting point.

4. **Path Reconstruction**:

    o Reconstruct the path from the start node to the goal node by combining the paths from both searches at the meeting point.

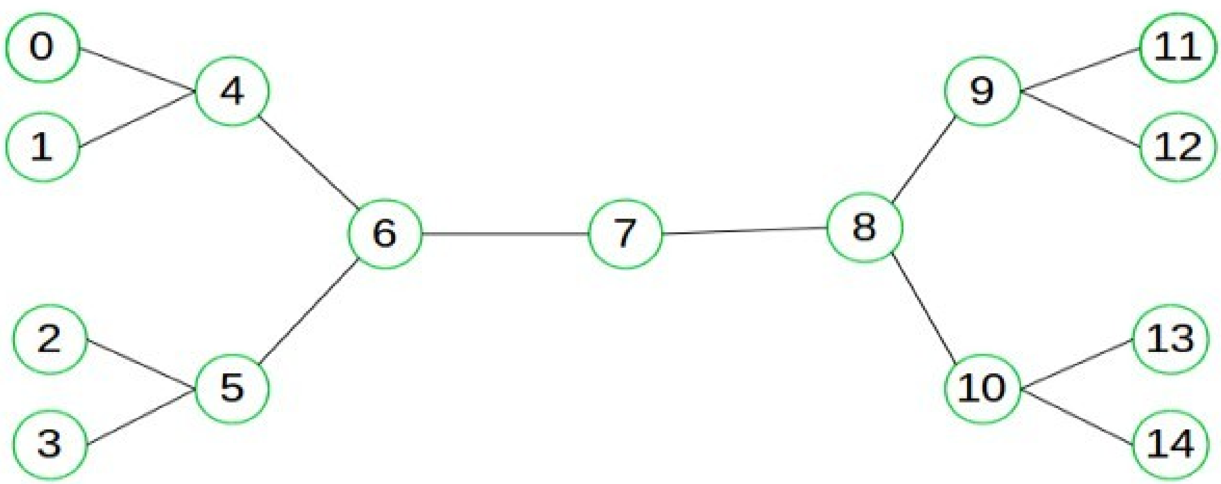
- o Start from the meeting point:
  - ▪ Trace back to the start node using the information in visited_start.
  - ▪ Trace back to the goal node using the information in visited_goal.
- o Concatenate the two paths to form the complete path from start to goal.

5. **Termination**:

   o If one of the frontiers is empty and no meeting point is found, it means there is no path from the

   start node to the goal node.

Example:



Suppose we want to find if there exists a path from vertex 0 to vertex 14. Here we can execute two searches, one from vertex 0 and other from vertex 14. When both forward and backward search meet at vertex 7, we know that

we have found a path from node 0 to 14 and search can be terminated now. We can clearly see that we have

successfully avoided unnecessary exploration.

When to use bidirectional approach?

We can consider bidirectional approach when-

1. Both initial and goal states are unique and completely defined.
2. The branching factor is exactly the same in both directions.

Performance measures

Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS is $O(b^d)$.

Space Complexity: Space complexity of bidirectional search is $O(b^d)$.

Optimal: Bidirectional search is Optimal.

Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

Disadvantages:

- Implementation of the bidirectional search tree is difcult.
- **In bidirectional search, one should know the goal state in advance.**

CSL502: Artificial Intelligence Lab