AI Searching Techniques

Uninformed Search Techniques

Dr.Sherif Saad

Learning Objectives

- Introduce the students to different search techniques and algorithms
- Present a general approach to model and represent problems as search problems
- Demonstrate how to implement AI search techniques and algorithms

Last-Time

- Depth First Search (Uninformed Search Techniques)
- Solving N-Queens with DFS
- Constraint Satisfaction Problem

Outlines

- Breadth First Search Algorithm
- Lowest-Cost-First Search
- Course | Research Topic Project.

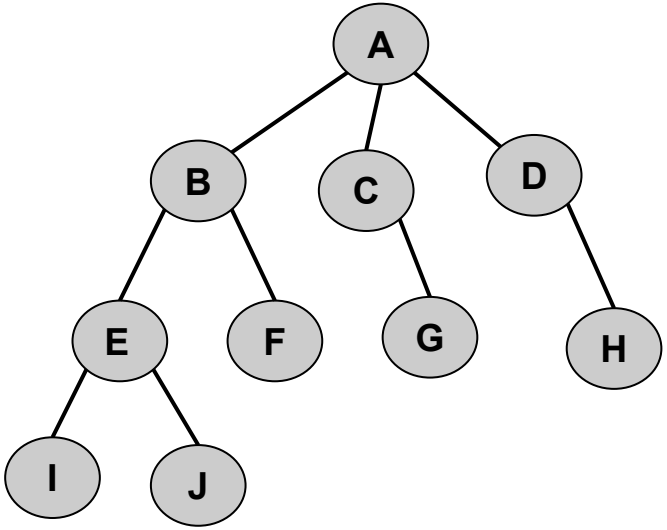
Uninformed Search: Breadth First Search

What is Breadth First Search?

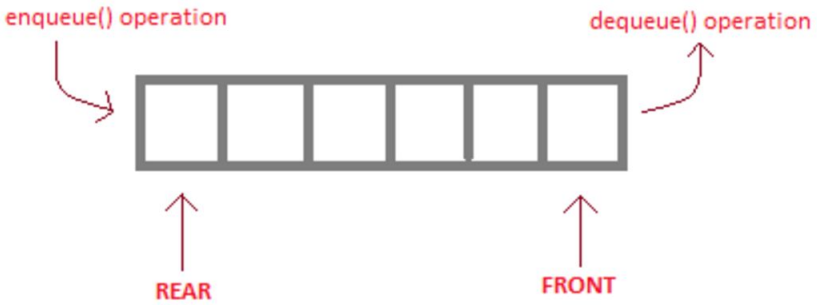
- A a general algorithm for graph traversal (searching tree or graph data structure)
- Works on directed and undirected graphs
- Implemented using a data structure called queue.
- Time Complexity:
  - O(|V|+|E|) traversed without repetition
  - O(b^d) in implicit graph (where b is the branching factor and d is the depth)
- Space Complexity: O (b^d)

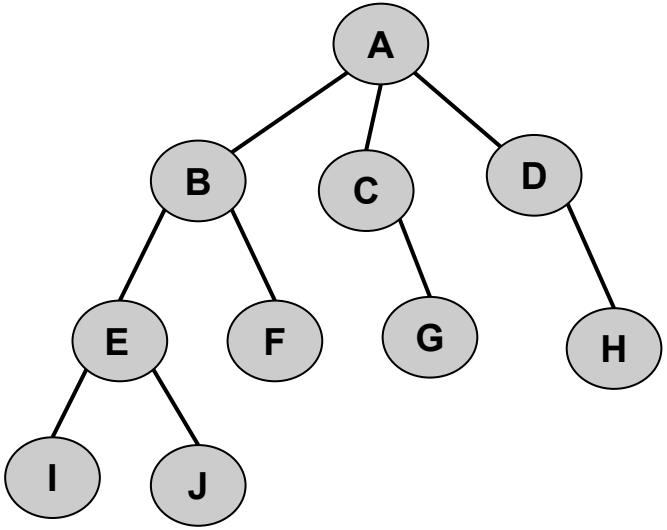
Breadth First Search: How it works?

- Traverse the graph one level (layer) at a time
- We need to keep track of visited (explored) nodes, so we do not visited a node infinite times. (graph could have a cycle)
- There is no true backtracking in BFS. What is true backtracking?



Breadth First Search: In Action





Visited Node

output:

Breadth First Search: Pseudo Code

```
def BFS(G,v):
    # Let Q be an empty queue
    Q = Que()

    for u in V of G:
        visited[u]= False

    Q.enque(v)

    while Q.isEmpty() == False:
        v = Q.deque()
        if v not in visited:
            visited[v] = True

            for w in neighbours of v and visited[w] is False:
                Q.enque(w)
```

Breadth First Search: Intelligence Properties

- Complete: Yes it always reaches the goal if b (branching factor) in finite
- Optimality: Yes (with conditions)
- Time Complexity: O(b^d+1)
- Space Complexity: O(b^d+1)

BFS vs DFS

Depth First Search

Complete: NO

Optimal: NO

Time Complexity: O(b^d)

Space Complexity: O(|V|)

Breadth First Search

Complete: Yes

Optimal: Yes

Time Complexity: O(b^d+1)

Space Complexity: O(b^d+1)

Branching Factor

- A branching factor ***b*** is the number of successors (children) that can be generated from a given node.
- If the value of **b** is not uniform, then we can calculate the average branching factor.
- Examples:
  - The average branching factor in a chess game for a piece in a legal position is 35
  - The average branching factor in Go (game) is 250
  - What is the branching factor of an N-Queen problem with constraints propagation and without constraints propagation ?

Branching Factor: Problem

Given the problem on the class board what is the branching factor? Is it a uniform value or not? You may use drawing to example your answer

Uninformed Search: Lowest-Cost-First Search

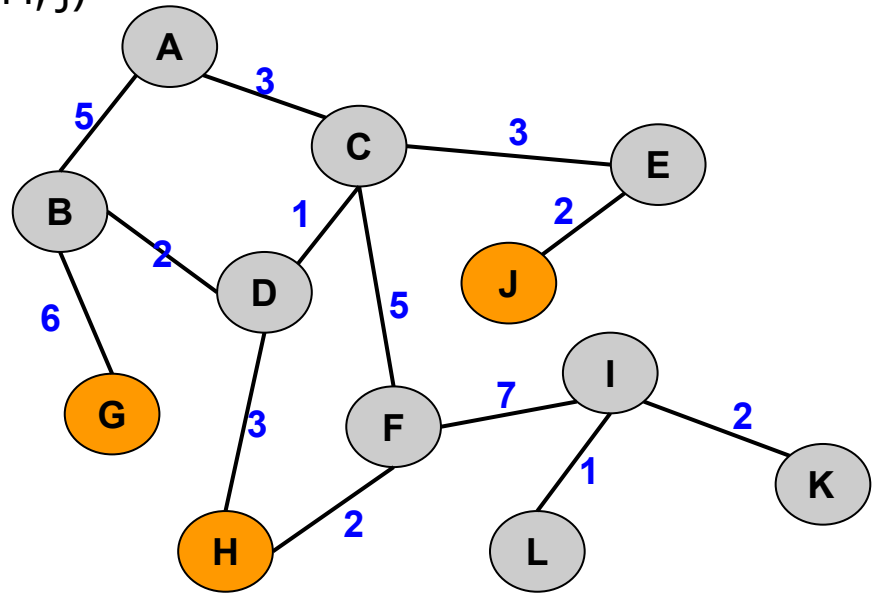
- Lowest-Cost-First Search or Uniform Cost Search
- We can use it to find the shortest path (path with minimum cost) in a weighted **direct** or **indirect graph**,
- It is a pathfinding algorithm. We can use it to solve any search graph with optimal cost
- It is a **Dijkstra's** algorithm but with a goal state.
- It is implemented using a priority queue.
- It guarantees optimality for any step cost

Lowest-Cost-First Search: How it Works?

- Visit the unvisited node with the smallest known cost from the current node.
- For the new visited node (new current), check its unvisited neighbors.
- For the current node calculate the cost of visiting its neighbors from the starting node (initial state)
- update the previously calculated cost of a given node if the newly calculated cost for this node is less than the previous cost
- If you reach the goal node and all the other paths have a cost greater than the path to the goal node terminate and return the path to the goal as the solution.

Uninformed Search: In Action

Find the shortest path to the goal node? In this graph there are three possible nodes each with a goal state (G, H, J)



Uninformed Search: Intelligence Properties

Complete: guaranteed provided the cost of every step exceeds some small positive constant. Otherwise it could enter infinite loop (zero-cost actions)

Optimality: Yes (with conditions)

Time Complexity: number of nodes with path cost <= cost of optimal solution

Space Complexity: number of nodes with path cost <= cost of optimal solution

Uninformed Search: Pseudo Code

```
def UCS(G,v, g):
    # Let Q be an empty priority queue
    Q = Que()
    # Let dist an empty list
    dist = List()
    # Let prev an empty list
    prev = List()
    dist[v] = 0
    prev[v] = v
    for each u in V of G:
        if u not equal v:
            dist[u] = Infinity
            prev[u] = None

        Q.enque(v, dist[v])
    while Q.isEmpty is False:
        u = Q.deque()
        for each neighbor w of u:
            ncost = dist[u] + cost(u,w)
            if ncost < dist[w]:
               dist[w] = ncost
               prev[w] = u

    return dist, prev
```

Need a minor fix to terminate when the goal is found. Can you fix it?

Project | Research Topic

Extra Resources

- **NetworkX** is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.
- http://networkx.github.io/index.html
- Reading ***The Branching Factor of Regular Search Spaces***

Questions