# *Introduction to Artificial Intelligence (G51IAI)*

Dr Rong Qu

## *Blind Searches - Introduction*

# Aim of This Section – (2 hours)

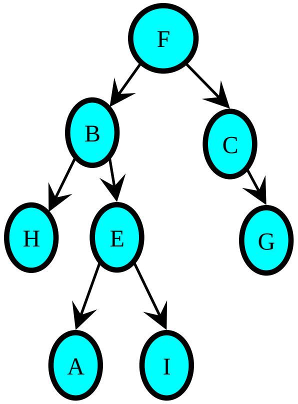
- Introduce the blind searches on search tree
- Specifically, the "general search pseudo-code" in AIMA and in the course notes
  - Understanding these notions is crucial for this section on search
  - The pseudo-code might be difficult to follow without first having a high-level understanding of what the algorithm is trying to do

# AI Techniques

- Use the relevant knowledge that people have to solve problem

- Problem solving techniques
  - Uninformed algorithms (blind search)
  - Informed algorithms (heuristic search)

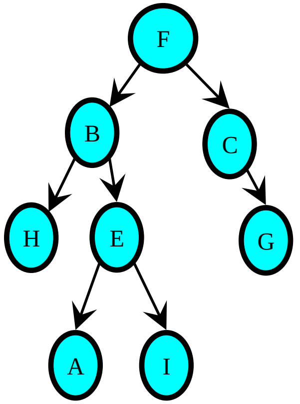
- Techniques in this module
  - Mainly based on tree search



# Problem Definition - 1

- Initial **State**
  - The initial state of the problem, defined in some suitable manner

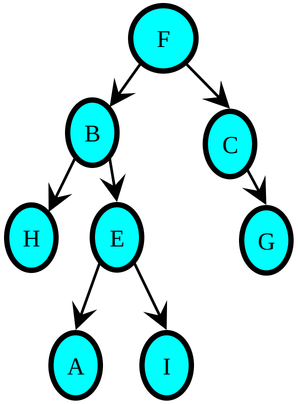
- **Operator**
  - A set of actions that moves the problem from one state to another



# Problem Definition - 1

- **Neighbourhood** (by Successor Function)
  - The set of all possible states reachable from a given state

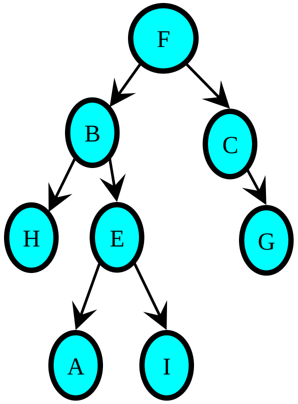
- **State Space**
  - The set of all states reachable from the initial state



# Problem Definition - 2

- **Goal Test**
  - A test applied to a state which returns if we have reached a state that solves the problem
- **Path Cost**
  - How much it costs to take a particular path

Examples: TSP, nQueen



# Problem Definition - Example

| | | |
|---|---|---|
| 5 | 4 | |
| 6 | 1 | 8 |
| 7 | 3 | 2 |

Initial State

| | | |
|---|---|---|
| 1 | 4 | 7 |
| 2 | 5 | 8 |
| 3 | 6 | |

Goal State

# Problem Definition - Example

- States
  - A description of each of the eight tiles in each location that it can occupy.
  - It is also useful to include the blank
- Operators
  - The blank moves left, right, up or down

# Problem Definition - Example

- Goal Test
  - The current state matches a certain state (e.g. one of the goal states shown on previous slide)
- Path Cost
  - Each move of the blank costs 1

# Exercise
# – state space of 8-queen problem

- Initial state
- Operator
- States
- Goal state

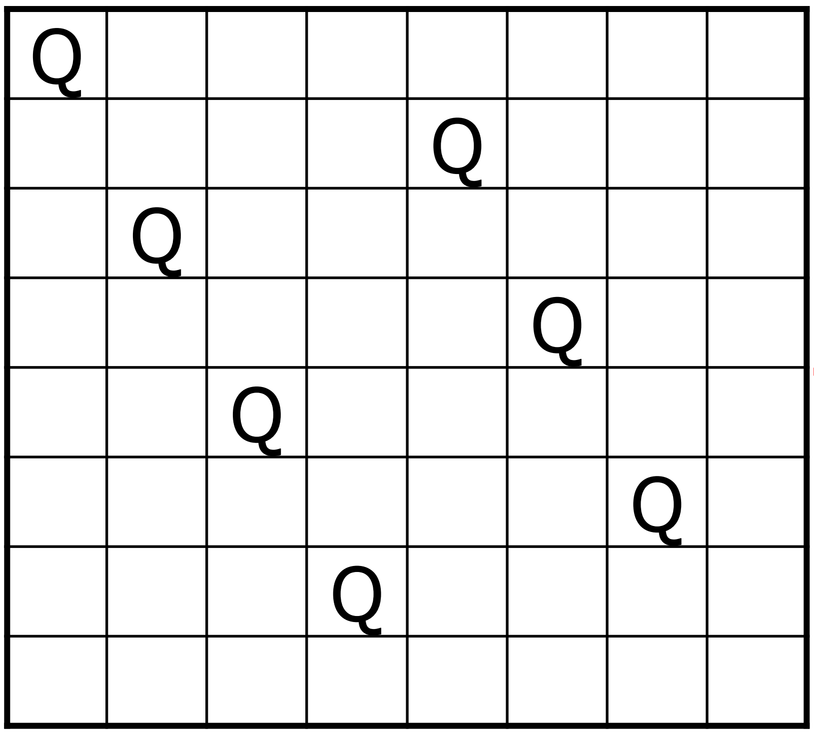
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Q | | | | | | | |
| | Q | | | Q | | | |
| Q | | | | | | | |
| | | | | | Q | | |
| | | Q | | | | | |
| | | | | | Q | | |
| | | | Q | | | | |
| | | | | | | | |

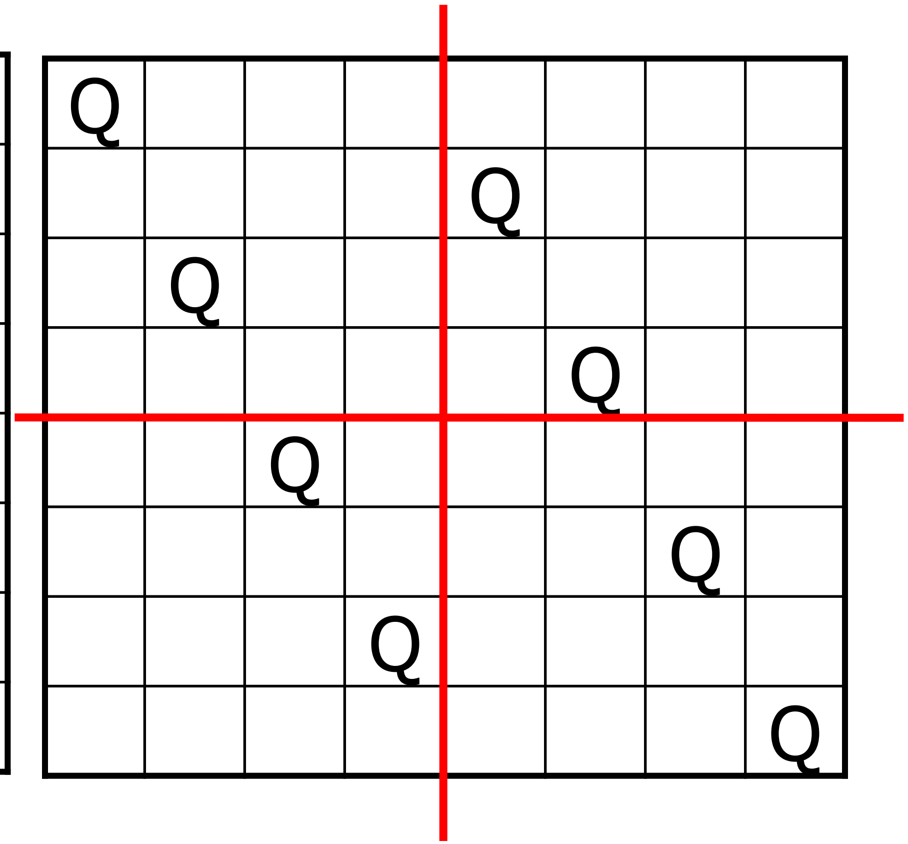
# Exercise
# – state space of 8-queen problem

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Q | | | | | | | |
| | Q | | | Q | | | |
| Q | | | | | | | |
| | | | | | Q | | |
| | | Q | | | | | |
| | | | | | Q | | |
| | | | Q | | | | |
| | | | | | | | |

# Exercise
# – state space of 8-queen problem

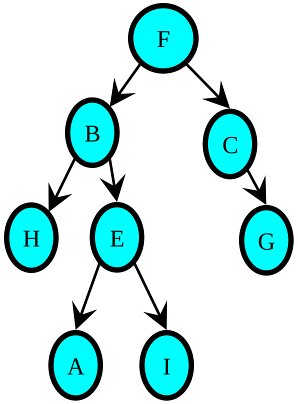


# Search Trees

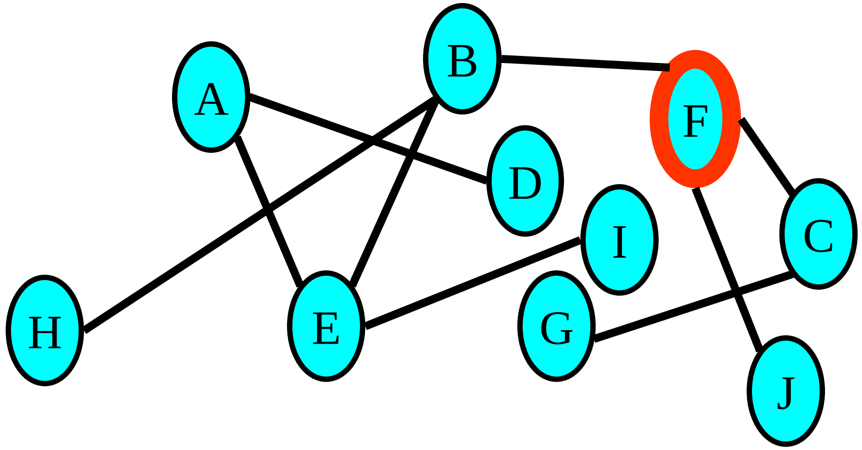
- A tree is a graph that:
  - is connected but becomes disconnected on removing any edge (branch)
  - has precisely one path between any two nodes

- Unique path
  - makes them much easier to search
  - so we will start with search on trees



# Search Trees

- Does the following tree contain a node “I”?




  - Yes. How did you know that?
  - so why the big deal about search?

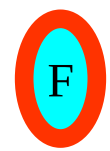
# Search Trees

- Because the graph is not given in a nice picture "on a piece of paper"

- Instead the graph/tree is usually
  - Explicitly known, but "hidden". You need to discover it "on the fly" i.e. as you do the search
  - Implicitly known only. You are given a set of rules with which to create the graph "on the fly"

# Search Trees

- Does the tree under the following root contain a node "G"?




- All you get to see at first is the root
  - and a guarantee that it is a tree
- The rest is up to you to discover ***during*** the process of search

# Evaluating a Search

- Does our search method actually find a solution?
- Is it a good solution?
  - Path Cost
  - Search Cost (Time and Memory)
- Does it find the optimal solution?
  - But what is ***optimal***?

# Evaluating a Search

We'll evaluate all the later search techniques w.r.t the below 4 criteria

1. Completeness
   - Is the strategy guaranteed to find a solution if one exist?

2. Time Complexity
   - How long does it take to find a solution?

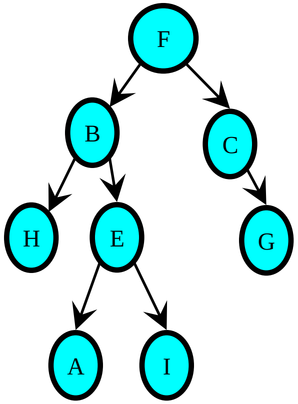
# Evaluating a Search

We'll evaluate all the later search techniques w.r.t the below 4 criteria

3. Space Complexity
   - How much memory does it take to perform the search?

4. Optimality
   - Does the strategy find the optimal solution where there are several solutions?

# Blind Searches



# Blind Searches - Characteristics

- Simply searches the State Space
- Can only distinguish between a goal state and a non-goal state
- Sometimes called an uninformed search as it has no knowledge about its domain

# Blind Searches - Characteristics

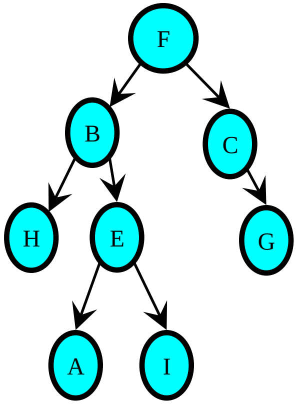
- Blind Searches have no preference as to which state (node) that is expanded next
- The different types of blind searches are characterised by the order in which they expand the nodes
- This can have a dramatic effect on how well the search performs when measured against the **four criteria** we defined earlier

# Blind Searches - implementation

Fundamental actions (operators):

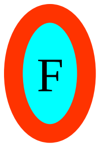
1. "Expand"
   Ask a node for its children

2. "Test"
   Test a node to see whether it is a goal



# Blind Searches - implementation

- Does the tree under the following root contain a node “G”?

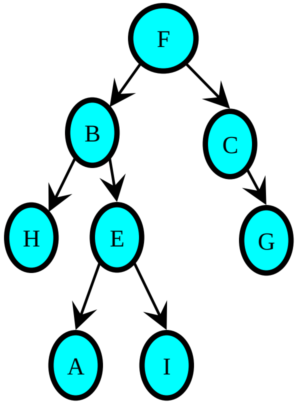



  - Allowed:
    - Expand
    - Test

# Blind Searches - implementation

We'll have 3 types of nodes during the search

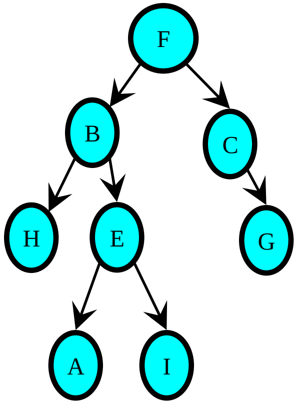
- Fringe nodes
  - have been discovered
  - have not yet been "processed":
    1. have not yet discovered their children
    2. (have not yet tested if they are a goal)
  - Also called
    - open nodes



# Blind Searches - implementation

We'll have 3 types of nodes during the search

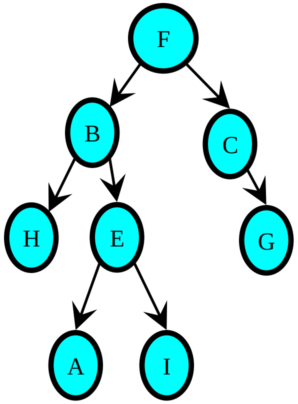
- Visited nodes
  - have been discovered
  - have been processed:
    1. have discovered all their children
    2. (have tested whether are a goal)
  - Also called
    - closed nodes



# Blind Searches - implementation

We'll have 3 types of nodes during the search

- Undiscovered nodes
  - The set of nodes that have not yet been discovered as being reachable from the root



# Blind Searches - implementation

- Fundamental Search Ideas
  - Maintain the list of fringe nodes
    - **`Queue`**
  - A method to expand the node
    - to discover its children
  - A method to pick a fringe node
    - to be expanded
  - Move node
    - To fringe: once it's been discovered **`insert`**
    - Out of fringe and into visited: after they have been processed **`remove`**

# Blind Searches - implementation

- Need a data structure to store the fringe
- AIMA uses a generic notion of
  - **`Queue`**
  - A list of nodes - general memory
- Need methods to
  - add nodes : **`INSERT`**
  - remove nodes : **`REMOVE-FIRST`**

# Blind Searches – ordering of nodes

- Does the ordering of nodes matter?
  - does the completeness depend on the way in which we implement **`INSERT`**?
    - Each node is expanded only once, and then removed from the fringe
    - Independently of the ordering, all nodes will be expanded, and expanded only once
    - We assumed (implicitly) that the tree is finite

# Blind Searches – ordering of nodes

- If search is complete, why ordering of nodes?
  - different node orderings affect the shape of the fringe
  - different shapes of the fringe can lead to very different memory usages

# Blind Searches – ordering of nodes

- If search is complete, why ordering of nodes?
  - The difference between searches lies in the order in which nodes are selected for expansion
  - The search always visits the first node in the fringe queue
  - The only way to control the ordering is to control the **`INSERT`**

# Blind Searches

- Breadth first
- Uniform cost
- Depth first
- Depth limited
- Iterative deepening