Rutvik Chandrayan

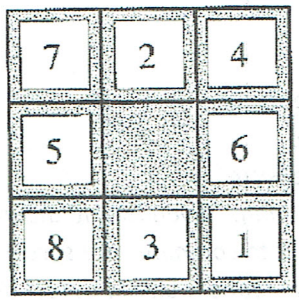
Unit 1 Searches Day 1 Note

What is an AI?

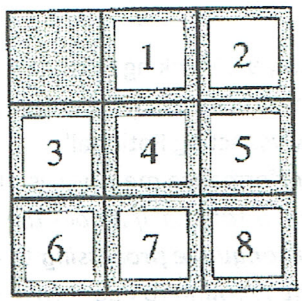
- Thinking Humanly vs. Thinking Rationally
  - Thought processes and reasoning vs. behavior
- Acting Humanly vs. Acting Rationally
  - Ideal performance measure vs. right measure
  - *A system is rational if it does the "right thing," given that it knows.*
  - ***Natural language processing*** *to enable it to communicate successfully in English*
  - ***Machine learning*** *to adapt to new circumstances and to detect and extrapolate patterns*
  - Rational agent approach
    - *An* ***agent*** *is just something that acts.*
    - *A* ***rational agent*** *is one that acts … to achieve the best outcome.*
    - *cf) Each* ***state*** *is represented as a conjunction of fluents that are ground, functionless atoms.*
- Task environments
  - Fully observable
    - *Fully observable environments are convenient because the agent need not maintain any internal state to keep track of the world.*
  - Partially observable
    - *An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data—for example, a vacuum agent with only a local dirt sensor cannot tell whether there is dirt in other squares.*
  - Deterministic vs. Stochastic
    - *If the next state of the environment is completely determined by the current state and the action executed by the agent,* it's deterministic.
  - Static vs. Dynamic
  - Discrete vs. Continuous
    - Time is handled? Possible to have a discrete set of percepts and actions?
  - Known vs. Unknown
    - *In a known environment, the outcomes for all actions are given.*
    - In an unknown environment, *the agent will have to learn how it works in order to make good decisions.*

Definition of a Problem:

- Initial State
- Actions(s) → $\{a_1, a_2, a_3, \dots\}$
- Result(s, a) → s'
- GoalTest(s) → True/False
- PathCost($s \xrightarrow{a} s \xrightarrow{a} s$) → n
  - StepCost(s, a, s') → n



Start State



Goal State

Example1) Define the problem of the 8-puzzle:

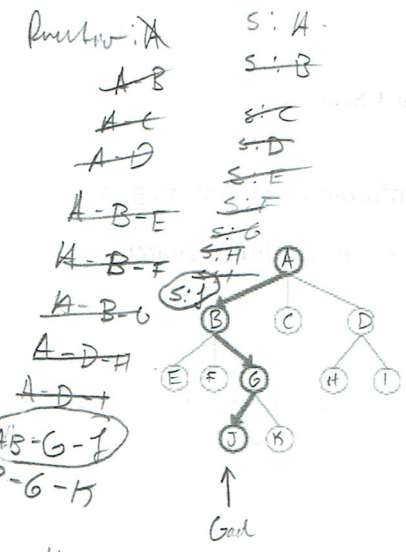
- States:
  A description of all the tile locations
- Initial state: Any state
  7 2 4 a string: "2245-6831" or
  5 3 6 a list
- Actions:
  Movement of the blank state. L, R, U, D, can be different depending on where the space is
- Transition model (a description of what each action does):
  Given an action + state, return the new location of all the tiles
- Goal test:
  Check if the string == "1-12345678" or list
- Path cost:
  Number of moves used to move the tiles to the ending state

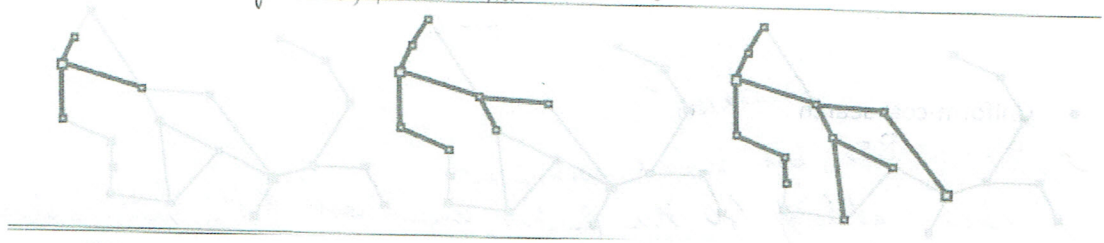
Three regions:

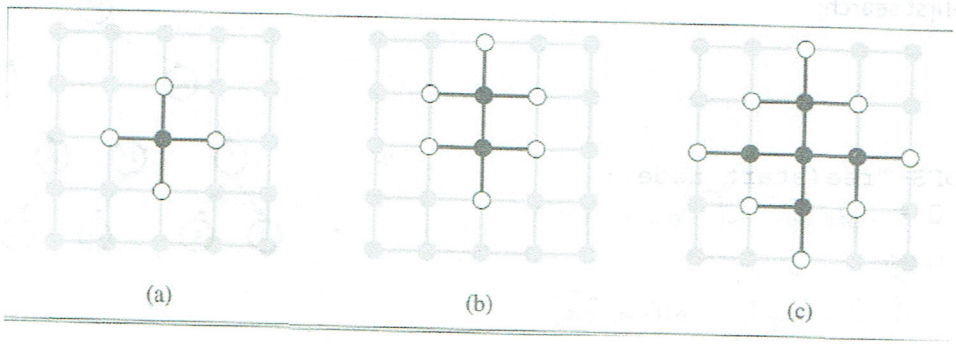
- Explored = where we've gone
- Frontier = where we are
- Unexplored = where we have not gone

Tree Search Vs. Graph Search

```
function Tree-Search(problem):
    frontier = (initial state)
    loop:
      if frontier is empty:
        return FAIL
      path = remove-choice(frontier)        <- pop()
      s = path.end
      if goal_test:
        return path
      for a in p.Actions:
        add [path+a, Result(s,a)] to frontier
```








```
function Graph-Search(problem):
    frontier = initial_state ; explored = { }
    loop:
      if (frontier is empty):
        return FAIL
      path = remove_choice(frontier
      s = path.end
      add s to explored
      if Goal_Test(s):
        return path
      for a in p.Actions(s):
        unless Result(s,a) is in frontier & explored
          add [path+a -> Result(s,a)] to frontier
```

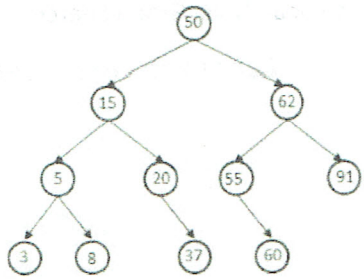
BFS:

Pop from front

DFS:

Pop from End

Uninformed search strategies



- Breadth-first search:

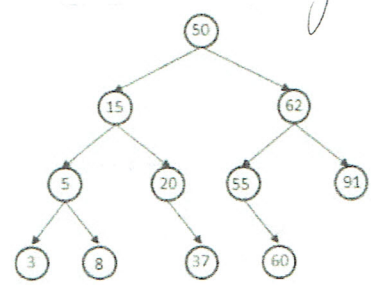
```
def BFS-Tree(start_node):
    Q = queue w/ starting node
    loop:
        if Q is empty: return -1
        path = S = Q.dequeue()
        if goal_test(S): return path
        for a in s.children:
            if a is not in Q
                Q.add(a)
```

- Uniform-cost search

Almost BFS, but lowest cost first

When adding to the frontier compare cost if the node is already there

- Depth-first search:




```
def DFS-Tree(start_node):
    Q = stack w/ start_node
    loop:
        if Q is empty: return Fail
        S = Q.pop()
        if goal_test(Q):
            return path
        for a in s.children:
            if a not in Frontier:
                Q.add(a)
```