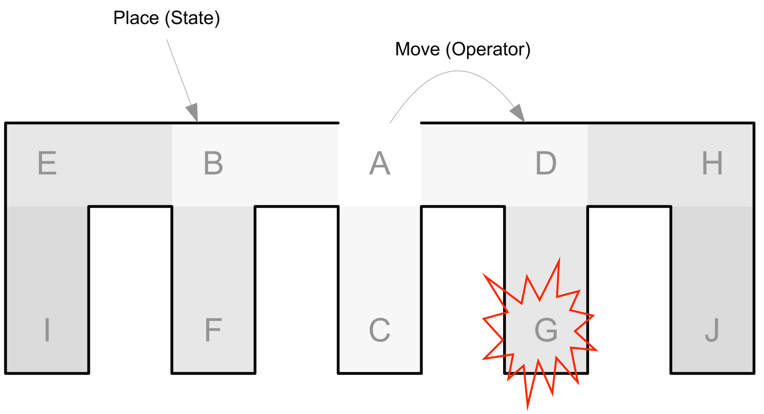
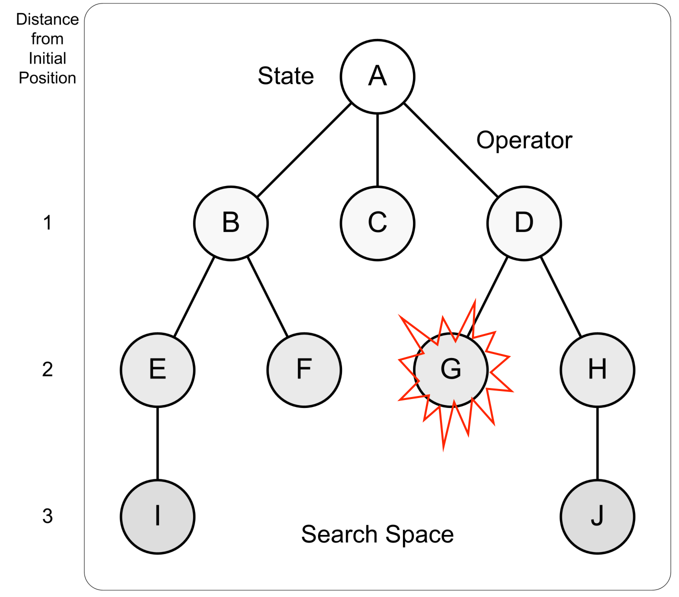
Searching

- Searching is a fundamental problem solving paradigm in AI
- Recall that we view AI as the computational part of satisfying a goal; this can in most cases be viewed as a search
- There are two broad classes of search algorithms
  - *Uninformed* - the search does *not* take domain information into account
  - *Informed* - the search does take domain information into account

(Physical) State Space Search

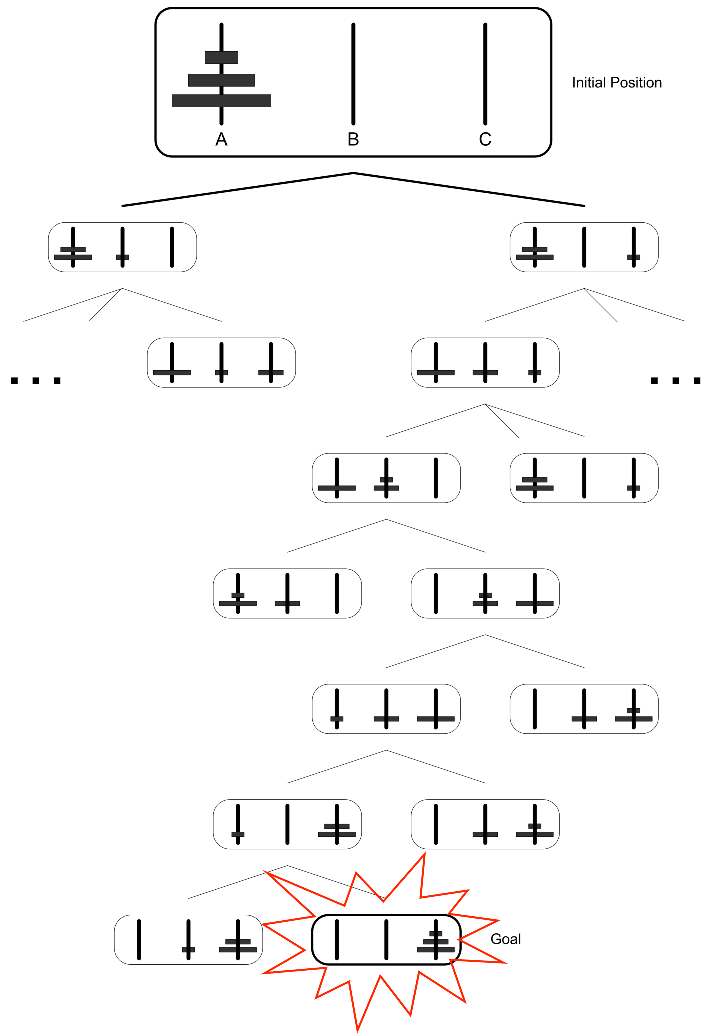
- Search operates on a search space with states where each state represents a possible position in physical space
- Trees (graphs) are an obvious representation with states as nodes and state transitions as links
- Can be applied to other AI problems in a variety of ways
  - Consider responding to an attack; here each state represents a different defense mode
- Hallmark: have to search the whole tree for a solution.



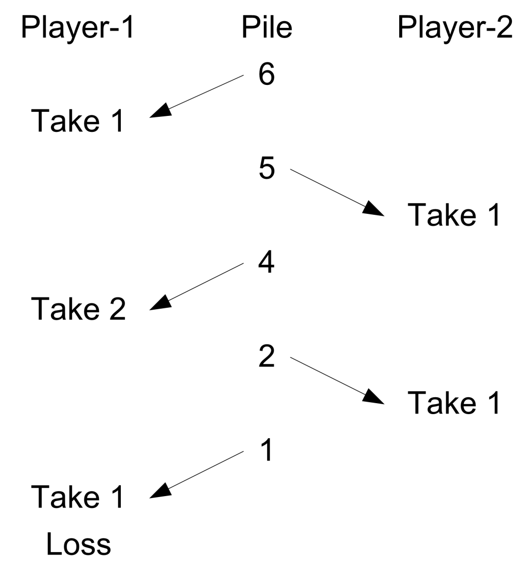


Search Applied to Puzzles

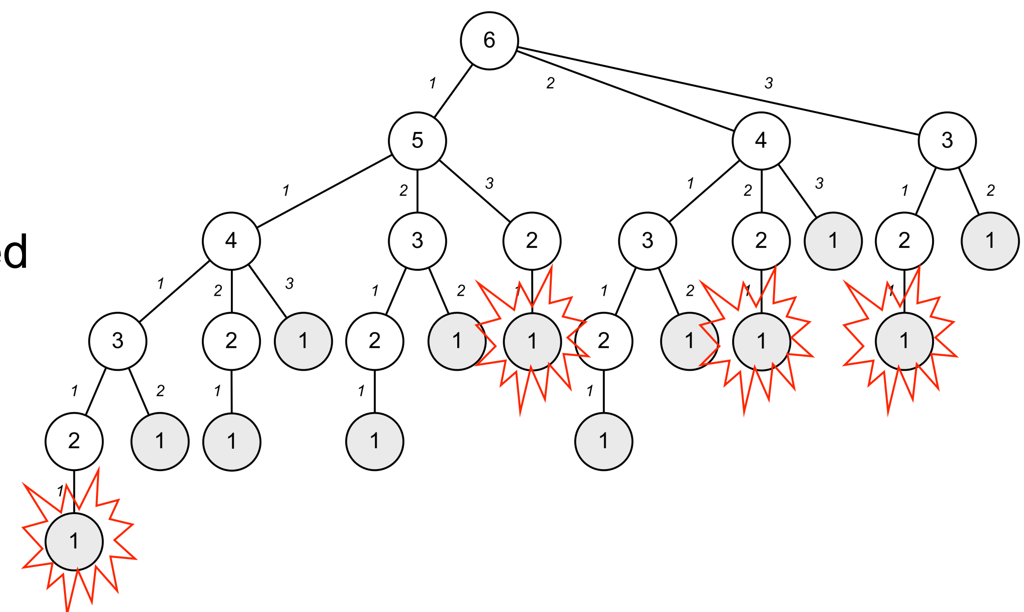
- Puzzles can be easily represented by state spaces
- One interesting example is the "Tower of Hanoi."
- Each move (operation) results in a new configuration (state)
- Brute-force can be used to find a specific state given an initial state (find a solution).
- **Problem**: State spaces can be enormous and brute force search can be slow to find a solution.



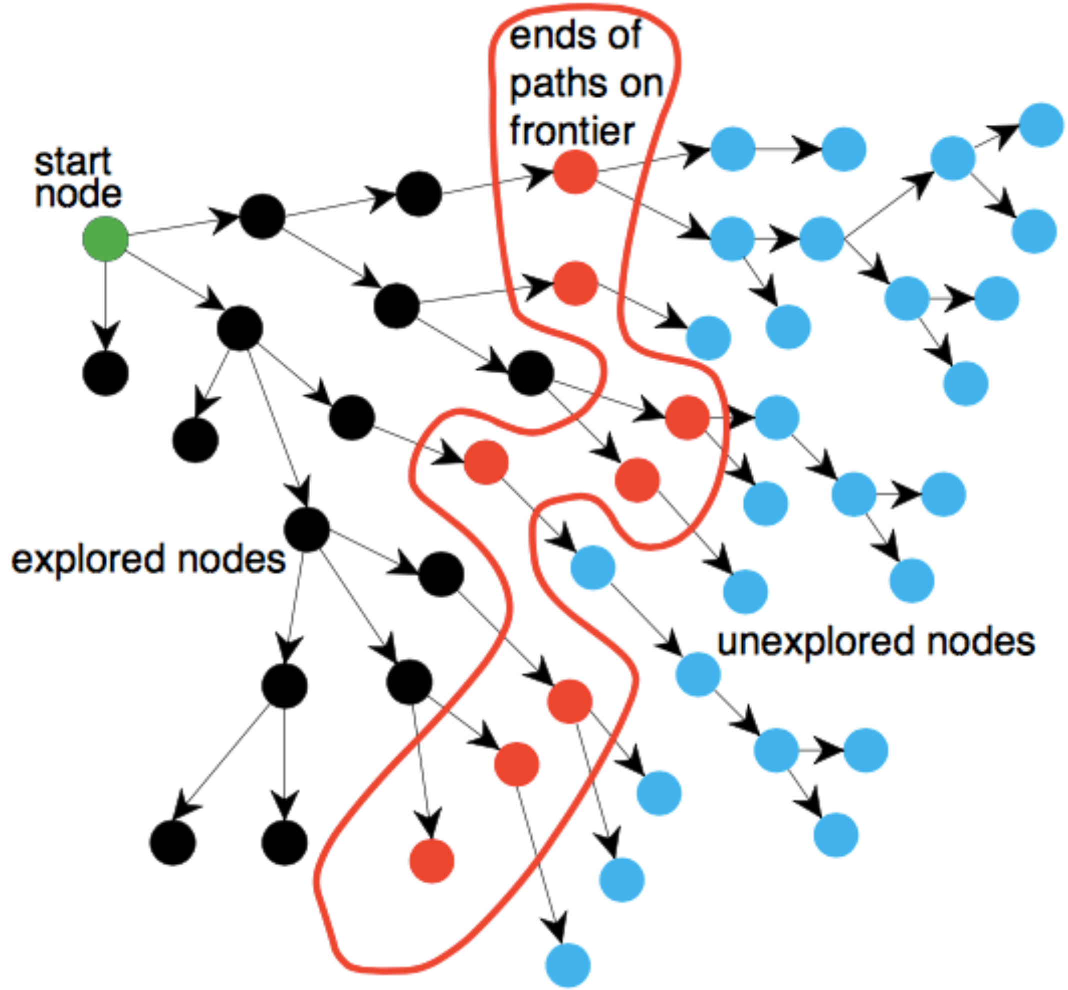
Adversarial Search



- Adversarial search allows a computer to find an effective strategy for playing against a human.
- The Game of Nim shown
- Computer/Human moves restrict the search space.
  - For example, if at the start the human chooses 1 item, the left subtree is used.
- Search is used to identify the next move to make to ensure a win
- If player-1 moves first then all the win states for player-2 are identified in the tree (positions that force player-1 to take the last stone)



Generic Search



Generic Search

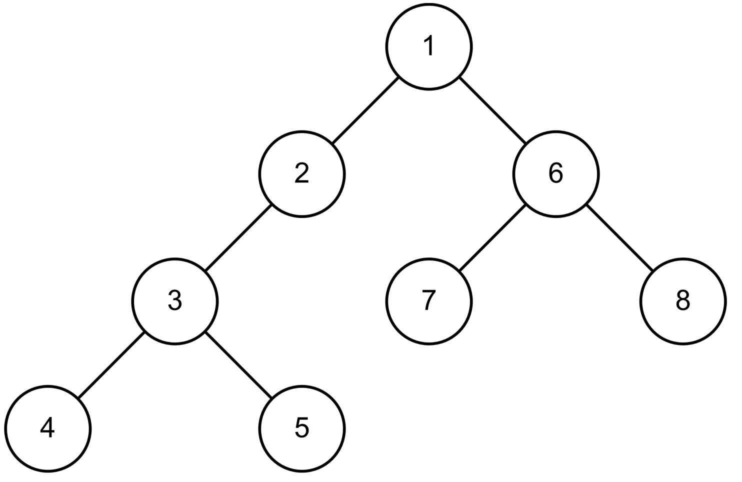
Input: a graph,
a set of start nodes,
Boolean procedure $goal(n)$ that tests if n is a goal node.
$frontier := \{\langle s \rangle : s \text{ is a start node}\}$;
while *frontier* is not empty:
  select and **remove** path $\langle n_0, \ldots, n_k \rangle$ from *frontier*;
  if $goal(n_k)$
    return $\langle n_0, \ldots, n_k \rangle$;
  for every neighbor n of n_k
    add $\langle n_0, \ldots, n_k, n \rangle$ to *frontier*;
end while

Uninformed Search Algorithms

- Depth-First-Search (DFS)
- Breadth-First-Search (BFS)
- Uniform Cost Search (UCS)

Depth-First-Search (DFS)

- Search each branch to its greatest depth, backtrack, explore previously unexplored branches.
- Simple, but favors depth over breadth.
- Note: not usable in trees with possibly infinite branches
  - E.g. trees that represent some sort of iteration; classic example is Prolog proof trees.

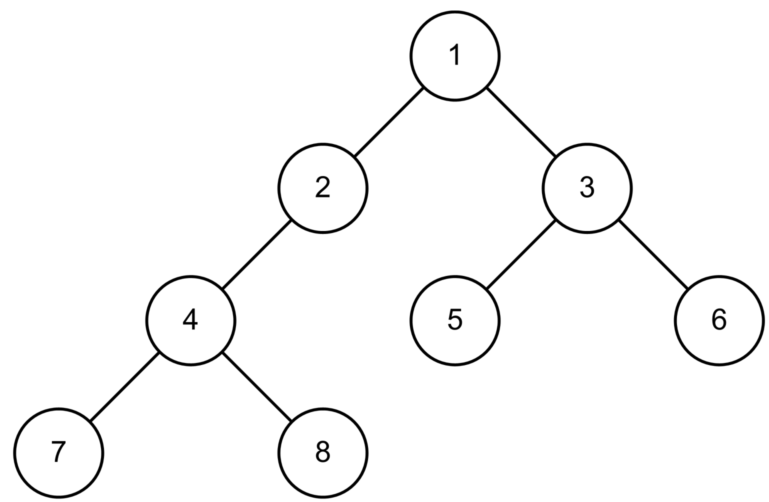


DFS Algorithm

```
procedure DFS(G,v,goal):
    % G -- a graph
    % v -- start node
    % goal -- goal function
    let Frontier be a stack
    Frontier.push(v)
    while Frontier is not empty
        n ← Frontier.pop()
        if goal(n)
            return n
        for all neighbors w of v in reverse order do
            Frontier.push(w)
```

Breadth-First-Search (BFS)

- Search nodes shallowest first.
- Favors breadth over depth.
- Note: can be used with infinite trees!

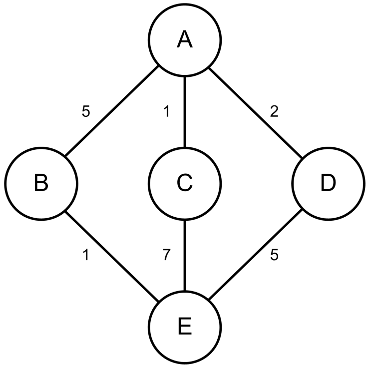


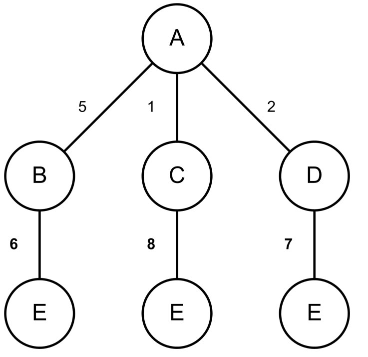
BFS Algorithm

```
procedure BFS(G,v,goal):
    % G -- a graph
    % v -- start node
    % goal -- goal function
    let Frontier be a queue
    Frontier.add(v)
    while Frontier is not empty
        n ← Frontier.pop()
        if goal(n)
            return n
        for all neighbors w of v in reverse order do
            Frontier.add(w)
```

Uniform-Cost Search (UCS)

- Find the least-cost path through a graph.
- Not all edges the same cost.
- Goal to find the path from start to finish with *least* cost (A->E).
- Note: this is important in navigation, least cost path to move from one location to another.
- Can be efficiently be implemented with a priority queue.





Step	Investigating Node	Priority Queue		
1		A(0)		
2	A	C(1)	D(2)	B(5)
		A(0)	A(0)	A(0)
3	C	D(2)	B(5)	**E(8)**
		A(0)	A(0)	C(1)
				A(0)
4	D	B(5)	**E(7)**	**E(8)**
		A(0)	D(2)	C(1)
			A(0)	A(0)
5	B	**E(6)**	**E(7)**	**E(8)**
		B(5)	D(2)	C(1)
		A(0)	A(0)	A(0)

UCS Algorithm

```
procedure UCS(Graph, root, goal)
  n := root
  cost := 0
  Frontier := priority queue containing n only
  while Frontier is not empty
    n := Frontier.pop()
    if goal(n)
      return n
    for all neighbors w of n
      if w is not in Frontier
        Frontier.add(w)
      if w is in Frontier with higher cost
        replace existing node with w
```

Uninformed Search Algorithms

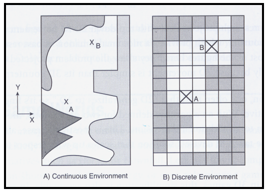
- Depth-First-Search (DFS)
- Breadth-First-Search (BFS)
- Uniform Cost Search (UCS)
- Depth-Limited-Search (DLS)
- Iterative-Deepening Search (IDS)
- Bidirectional Search (BIDI)

Searching an QII

- Searching is typically the first half of achieving a goal:
  - Once a solution is identified we need to *schedule/ plan* the actions required to achieve this goal.

Searching

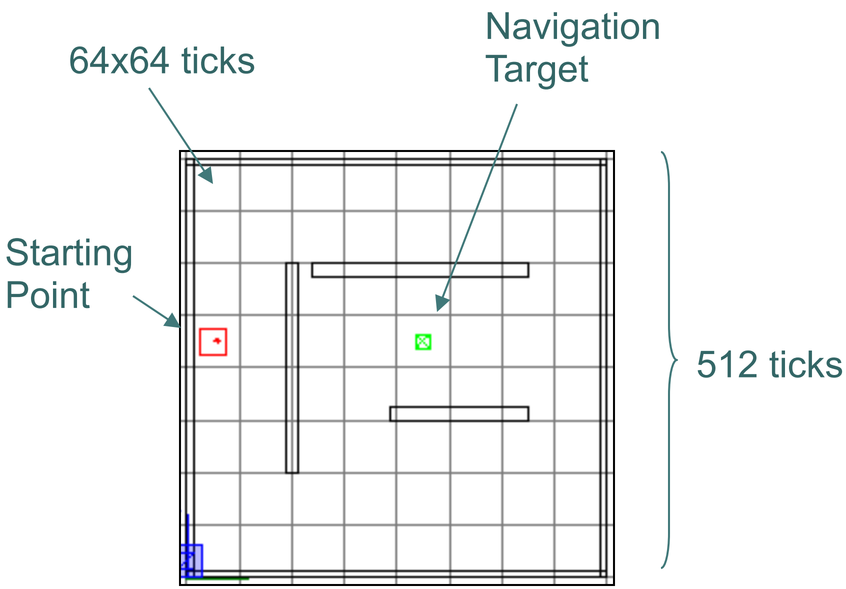
- Basic assumptions
  - we are given a global navigation target
  - the environment is given in a discrete representation (so far we have only considered continuous representations)

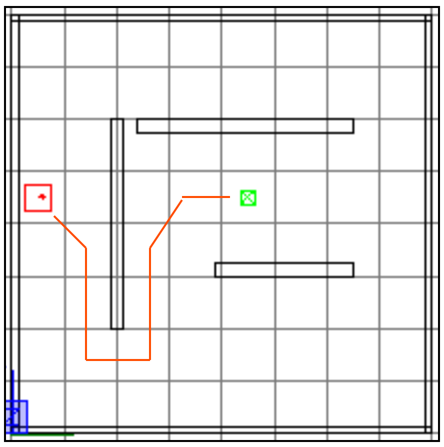


- Goal
  - given our current location and given the location of our navigation target
  - search for a path to reach this target
  - plan the actions necessary to travel from our current location to the desired target
  - respect obstacles!

Searching

The floor plan of the 'Obstacle Room'





Search for all possible paths from the starting point to the target.

Many paths possible -
Seven choices at each node -
Depth limited search - 7^{16}