State Space Search

Intensive Programming in Linux
CS288-102 Spring 2019

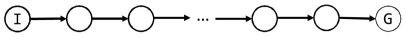
Formulating a Search Problem

Given:

- an initial state
- a set of operators
- a set of goal states
- an optional cost function associated with applying a given operator

Objective:

Find a finite sequence of operators that leads from the initial state to a goal state.



Eight Puzzle

Initial State:

Some configuration of the tiles

1	4	3
7		6
5	8	2

Actions:

Move the blank square ***up***, ***left***, ***down***, ***right***.

Goal State:

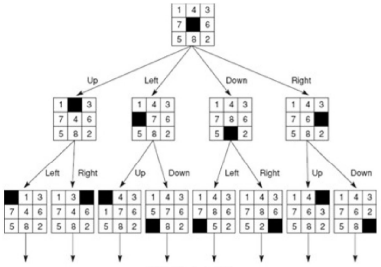
Some configuration of the tiles

1	2	3
4	5	6
7	8	

What is the State Space?

The **state space** is the implicit graph defined by the initial state and the operators. The nodes are the states, and the arcs between nodes are the operators. A solution in the state space is a path from the initial state to a goal state.

State Space for 8-Puzzle



How Big Is a State Space?

Most real problems have overwhelmingly large number of states. It is possible for a state space to have an infinite number of states.

For toy problems, the state space is relatively small. For example, 8-Puzzle has $\frac{9!}{2} = 181{,}440$ states.

Basic Search Algorithm

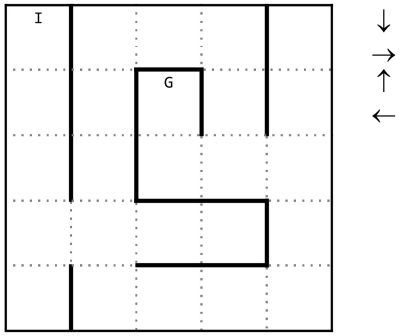
```
function search(initialState, goalStates)
    fringe = [new node from initialState]
    explored = []
    while (fringe not empty)
        node = fringe.remove()
        if node.state not in explored
            if node.state in goalStates report SUCCESS
            fringe += node.successors()
            explored += node.state
    report FAILURE
```

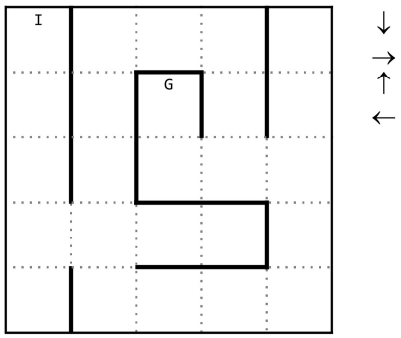

Uninformed Search Algorithms

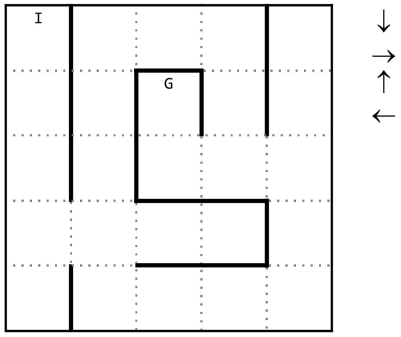
- Uses the Basic Search Algorithm
- If ***fringe*** is a Queue data structure, we have Breadth-First Search (BFS)
- If ***fringe*** is a Stack data structure, we have Depth-First Search (DFS)
- If ***fringe*** is a Priority Queue data structure, sorted according to cost, we have a Uniform-Cost Search

Informed Search Algorithms

- Uses the Basic Search Algorithm
- If ***fringe*** is a Priority Queue data structure, sorted according to most promising first, we have Best-First Search.
- A*: Choose from ***fringe*** node *N* with the smallest value of ***g(N) + h(N)***, where:
  - ***g(N)*** is the distance node *N* is from the initial node
  - ***h(N)*** is an estimation of the distance between node *N* and a goal node


I
G


I
G


I
G