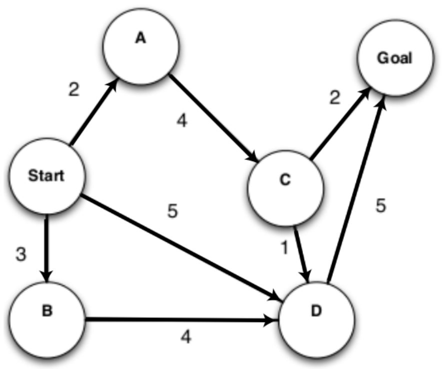
Uninformed Search

IFN680: Artificial Intelligence and Machine Learning
Week 8

1 Aims

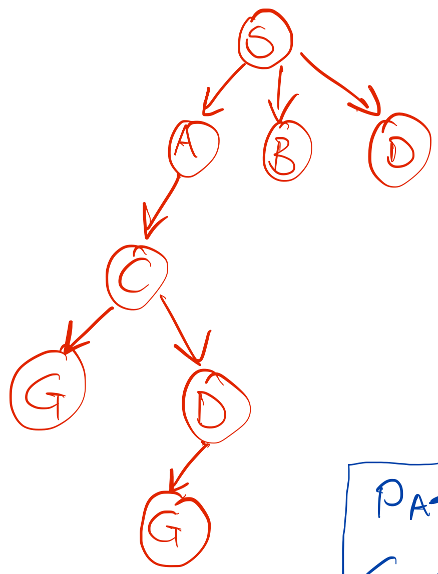
In this practical, you will implement and test some of the search algorithms introduced during the lectures, specifically uninformed search methods. You will do this both manually by hand and in Python.

2 Search Algorithms in Action



For each of the following graph search strategies, work out the order in which states are expanded, as well as the path returned by graph search. In all cases, assume ties resolve in such a way that states with earlier alphabetical order are expanded first. Remember that in graph search, a state is expanded only once.

a) Depth-first search.

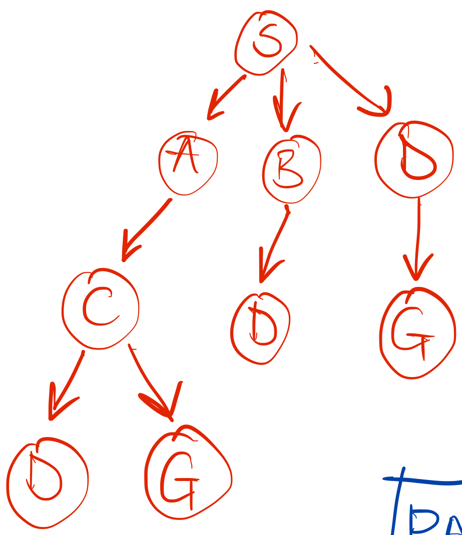


Fringe

~~S~~
~~S→A~~
S→B
S→D
~~S→A→C~~
S→A→C→G
~~S→A→C→D~~
S→A→C→D→G ✓✓

PATH FOUND:
S→A→C→D→G

b) Breadth-first search.



Fringe

~~S~~

~~S → A~~

~~S → B~~

~~S → D~~

~~S → A → G~~

S → B → D

S → D → G ✓✓

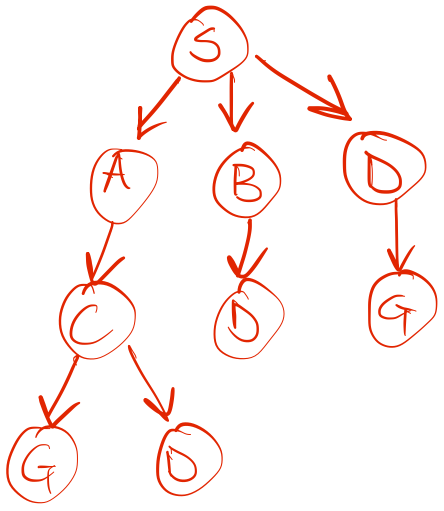
S → A → C → D

S → A → C → G

PATH FOUND

S → D → G

c) Uniform cost search.



Fringe	Cost
~~S~~	
~~S → A~~	~~2~~
~~S → B~~	~~3~~
~~S → D~~	~~5~~
~~S → A → C~~	~~6~~
S → B → D	7
S → D → G	10
S → A → C → G	8 ✓✓
S → A → C → D	7

PATH FOUND:

S → A → C → G

3 Sliding Puzzle Game

The **Sliding Puzzle** is a classic puzzle game that challenges the player's problem-solving and logical thinking skills. The game consists of a grid of numbered tiles with one tile missing. The numbers are scrambled, and the goal of the game is to arrange the tiles back into numerical order by sliding the blank tile around.

The rules of the game are:

- **Game Setup:** The game starts with all the numbered tiles in random order and one space is left blank. The size of the grid can vary, but common sizes are 3x3, 4x4, or 5x5.
- **Moving Tiles:** A tile can be moved only if it is adjacent to the blank space. A tile can be slid horizontally or vertically, but not diagonally. To simplify the action space in your code, you will instead imagine moving the blank tile around to an allowed location using the actions (UP, DOWN, LEFT and RIGHT).
- **Solving the Puzzle:** The puzzle is considered solved when all tiles are in their correct numerical position, and the blank space is in the first cell of the grid. Note: The position of this blank space can vary based on the user's choice when defining the goal state.

The aim of this challenge is to design and implement an AI agent that can solve the Sliding Puzzle game.

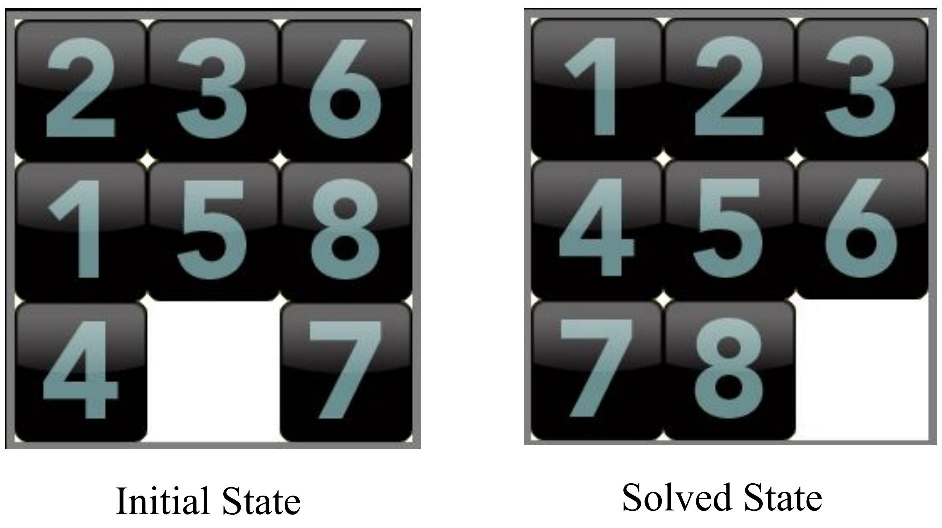


Figure 1: Sliding Puzzle environment.

3.1 Task

Task 1

- Load all the necessary files into Jupyter by clicking here.
- You can get a feel of the sliding puzzle game by playing the game here.
- Open the file `search.py`, and `sliding_puzzle.ipynb` files and browse through the code. Ask yourself how the classes are related. Map them to the slides of last week that contain pseudo-code. In particular, check the attributes of the `Node` and `Problem` classes.
- The `search.py` file provides you with a range of search algorithms which you can use for a wide range of problems. The key step to utilising this set of functions is to correctly define your problem. This is done by inheriting the `Problem` class and defining each of the required functions - `actions`, `results` etc.

- For this practical you must complete the functions in `sliding_puzzle.ipynb` marked with `INSERT YOUR CODE HERE` to implement the required solver.

Task 2

- Try to use a depth-first tree search. What happens? Why?
- Compare the running time of depth-first search and breadth-first search.

Should have observed that BFS tree search always terminates. However, DFS tree search might run forever if it gets stuck in a cycle.