# Algorithms in AI

Breadth First Search and A*
Path Planning with python 101

Bipin Vijayasenan

# Introduction

- Uninformed Search
  - Breadth First Search
  - Depth First Search
- Informed search (Using heuristics)
  - Greedy
  - A*

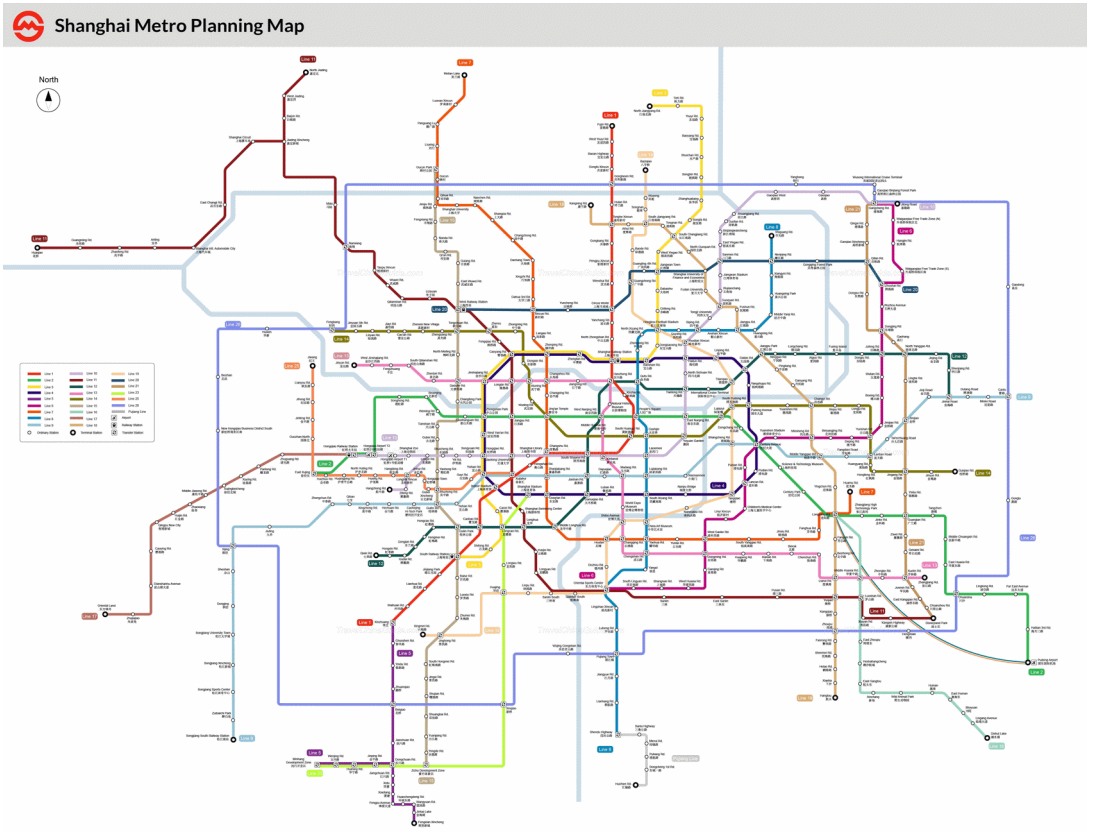
# Workshop

- Python knowledge expected from you
  - basic operations
  - list, dict, tuples
  - functions
- We will code most of the algorithms and supporting functions
- If you are stuck, please let me know. A solutions.py is provided.
- Rough schedule
  - BFS/DFS - 1hr
  - Greedy/A* - 1hr
  - Debugging and experimentation - 1hr

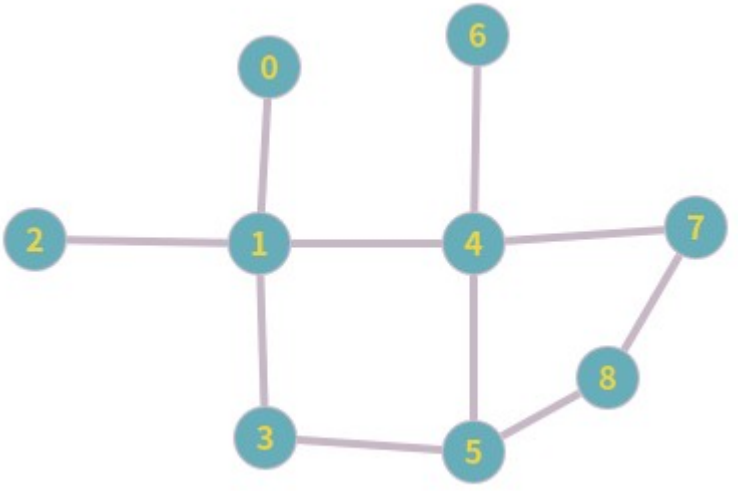
# Applications for coding

- Metro map (a simple subset of shanghai metro)
- Path finding in a grid world
- Solving 8Puzzle

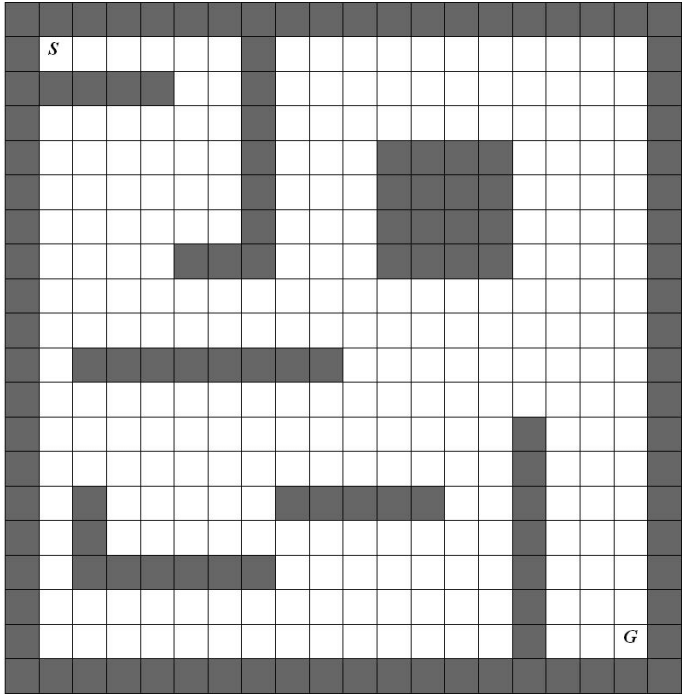
# Metro graph – Problem Introduction




https://www.travelchinaguide.com/images/map/shanghai/shanghai-subway.gif



# Grid World




https://www.researchgate.net/figure/The-example-is-a-gridworld-environment-with-cell-to-cell-actions-up-down-left-and_fig6_23249669

# 8Puzzle

- A tile adjacent to the blank space can slide into the space. The object is to reach a specified goal state.

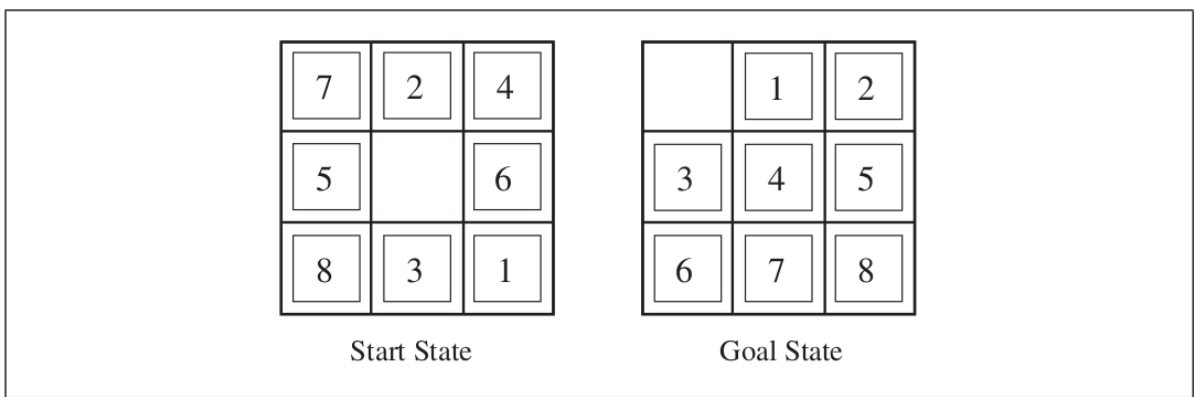



**Figure 3.4** A typical instance of the 8-puzzle.

# Example Real World Applications

- Route finding
- Games
- Robotics
- Web Crawling
- Social Networking
- Garbage collection in programming languages
- VLSI Layout

# Elements

- States
- Actions
- Successors (Next states)
- Start State, Goal State and Goal Test
- Path (Shortest path, Any path )
- Edges and edge weights (For this workshop we assume edge weight as 1 always)

# Breadth First Search

- Explain algorithm
- Code using the metro graph (with/without path)
- Trace the algorithm
- Introduce the Depth First Search change

# Pseudocode - Search

**function** TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  **loop do**
    **if** the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    **if** the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

**function** GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  ***initialize the explored set to be empty***
  **loop do**
    **if** the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    **if** the node contains a goal state **then return** the corresponding solution
    ***add the node to the explored set***
    expand the chosen node, adding the resulting nodes to the frontier
      ***only if not in the frontier or explored set***

# Grid World

- Explain Grid World
- Move the BFS code to GridWorld and check
- Show visited list in prints
- Bigger worlds?

# Limitations of Uninformed Search

- Idea of heuristics
- Idea of priority queues
- Code a simple/stupid priority queue
- Greedy and A* algorithms

# Uniform Cost Search

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

**Figure 3.14** Uniform-cost search on a graph. The algorithm is identical to the general graph search algorithm in Figure 3.7, except for the use of a priority queue and the addition of an extra check in case a shorter path to a frontier state is discovered. The data structure for *frontier* needs to support efficient membership testing, so it should combine the capabilities of a priority queue and a hash table.

# 8Puzzle

- Move the code to 8Puzzle environment
- Check path length by bfs and astar (length should be identical – paths can differ)
- Web UI?

# Reference


- Artificial Intelligence A Modern Approach (Russel & Norvig)
- CS188 Berkeley – Lectures 2 & 3 (http://ai.berkeley.edu/home.html)
- Algorithms (Sedgewick & Wayne) (https://algs4.cs.princeton.edu/home/)
- Introduction to Algorithms MIT OCW ( https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/ )