# Group Spirits
# First Lab Report CS362*

Submitted to Prof Pratik Shah

1st Devang Patel
*202051062*
202051062@iiitvadodara.ac.in

2nd Anik Bepari
*202051024*
202051024@iiitvadodara.ac.in

3rd Manoj kumar gautam
*202051114*
202051114@iiitvadodara.ac.in

4th Aditya tomar
*202051012*
202051012@iiitvadodara.ac.in

## I. Week 1 : Lab Assignment - 1

**Learning Objective:** To design a graph search agent and understand the use of a hash table, queue in state space search.

**Reference:** [1] Artificial Intelligence: a Modern Approach, Russell and Norvig (Fourth edition) Chapter 2 and 3

A first course in Artificial Intelligence, Deepak Khemani Chapter 2

**Problem Statement:** A. Write a pseudocode for a graph search agent. Represent the agent in the form of a flow chart. Clearly mention all the implementation details with reasons.
B. Write a collection of functions imitating the environment for Puzzle-8.
C. Describe what is Iterative Deepening Search.
D. Considering the cost associated with every move to be the same (uniform cost), write a function which can backtrack and produce the path taken to reach the goal state from the source/ initial state.
E. Generate Puzzle-8 instances with the goal state at depth "d".
F. Prepare a table indicating the memory and time requirements to solve Puzzle-8 instances (depth "d") using your graph search agent.

### A. A Part

A. Write a pseudocode for a graph search agent. Represent the agent in the form of a flow chart. Clearly mention all the implementation details with reasons.

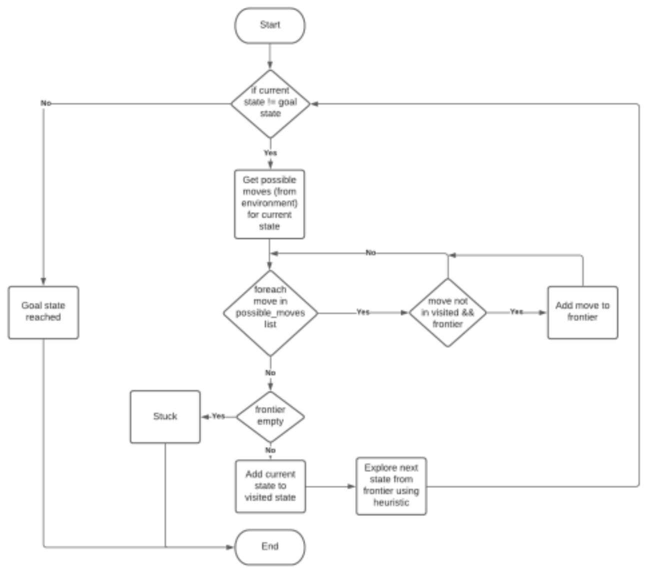

Start
if current state != goal state
No
Yes
Get possible moves (from environment) for current state
No
foreach move in possible_moves list
Yes
move not in visited && frontier
Yes
Add move to frontier
Goal state reached
No
Stuck
Yes
frontier empty
No
Add current state to visited state
Explore next state from frontier using heuristic
End

Fig. 2: Flowchart for Agent

**pseudocode:**

```
priority queue pq
if pq.isEmpty() return false
puz ⟵ pq.extract()
if search(pq) return true
for each suc in successors do
if suc not in visited then
pq.insert(suc)
visited.insert(suc)
if search(pq.visited) return true
else return false
```

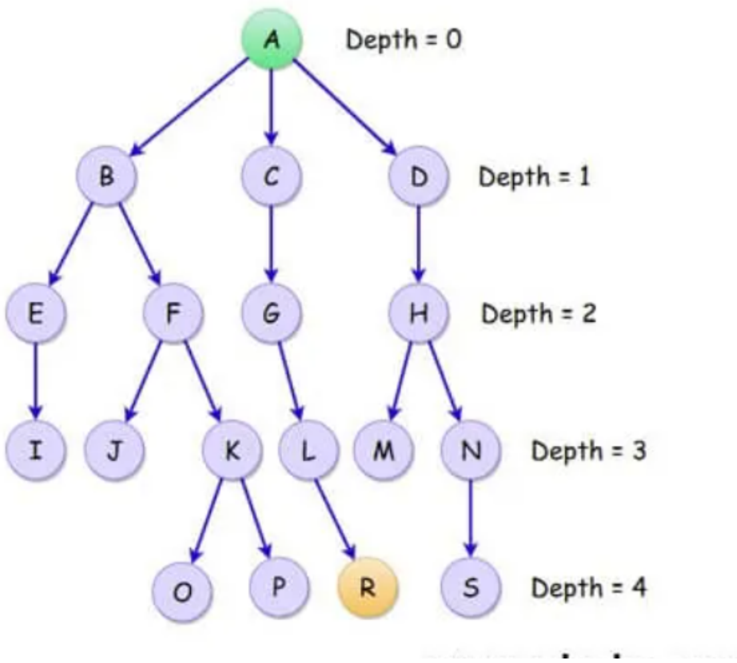
### B. B part:

B. Write a collection of functions imitating the environment for Puzzle-8.

- **Heuristic 0** This is a null heuristic, returns 0 for any state. Essentially uniform cost search.
- **Heuristic 1** This heuristic counts the number of misplaced tiles
- **Heuristic 2** This is the manhattan distance, it returns the sum of the horizontal and vertical distances of the all marble in current state from center.
- **Node** The Node class greates the graph node. It has the following values
  It makes use of the following built in functions:
  1) hash : This provides the hash value for every node, which is required for the hashset
  2) eq : To check if 2 nodes are equal (Operator overload)
  3) ne : To check if 2 nodes are not equal (Operator overload)
  4) str : To get string representation of state in node
- **PriorityQueue** The Priority Queue is used to store the nodes along with the cost, and pop the node having the least cost for BFS It makes use of the following functions:
  1) push : Add node to queue
  2) pop : Pop node having least cost
  3) is-empty : To check if queue is empty
  4) len : To get length of queue
  5) str : To get string representation of queue
- **Environment** The environment is what the agent plays in. It has the following entities:
  1) get-start-state : returns the start state
  2) reached-goal : returns goal-state
  3) get-next-states : Given current state, it returns all possible next states
  4) generate-start-state : Given goal state and depth d, performs d moves to generate a start state
- **Agent** The agent is the player who plays the game against the environment to win. It has the following entities:
  1) run(): Is the function that explores the environment and finds the goal node. Uses the built in heuristic function to get the path costs
  2) print-nodes(): To print the path from the start node to goal node

### C. C part

Describe what is Iterative Deepening Search.
**IDDFS** combines depth-first search's space-efficiency and breadth-first search's fast search (for nodes closer to root).
IDDFS calls DFS for different depths starting from an initial value. In every call, DFS is restricted from going beyond given depth. So basically we do DFS in a BFS fashion.



**Time complexity :**O($b^d$)
**Space complexity :**O(bd)

### D. D Part

Considering the cost associated with every move to be the same (uniform cost), write a function which can backtrack and produce the path taken to reach the goal state from the source/ initial state

**pseudo code:** def backtrack-path(goal-state, explored-states):

```
current-state = goal-state
path = [current-state]
while current-state in explored-states:
    current-state = explored-states[current-state]
    path.append(current-state)
path.reverse()
return path
```

Backtrack and produce the path taken to reach the goal state from the source/initial state.
**Args:**
- goal-state (tuple): the goal state represented as a tuple of integers
- explored-states (dictionary): a dictionary of explored states, where the keys are states represented as tuples and the values are the parent states that led to the explored state

**Returns:**
- path (list): a list of states representing the path from the source/initial state to the goal state

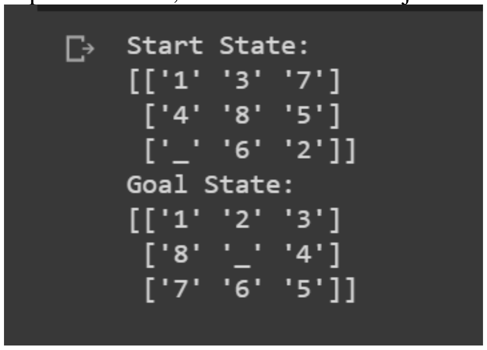
### *E. E part*

Generate Puzzle-8 instances with the goal state at depth “d”. We can generate random instances of 8-puzzle at a given depth from goal state at all the depths,given bolow are some examples Depth 1: Starting state: $\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & \end{matrix}$

Depth 2: Starting state: $\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & \\ 7 & 8 & 6 \end{matrix}$

Depth 3: Starting state: $\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & \end{matrix}$

So, it represents state at a given depth Note that the number of possible starting states for a given depth is vast, and these are just a few examples.

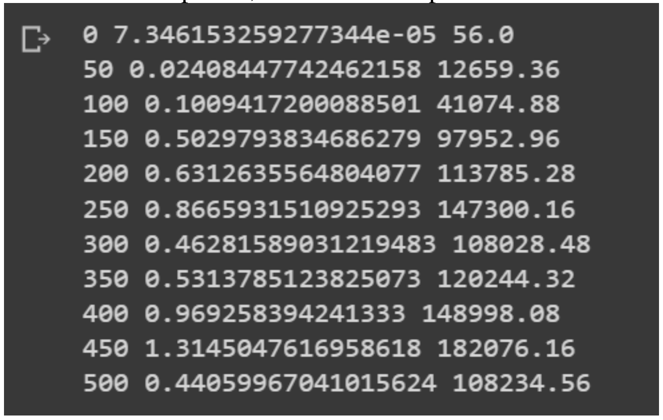


```
Start State:
[['1' '3' '7']
 ['4' '8' '5']
 ['_' '6' '2']]
Goal State:
[['1' '2' '3']
 ['8' '_' '4']
 ['7' '6' '5']]
```

### *F. F Part*

Prepare a table indicating the memory and time requirements to solve Puzzle-8 instances (depth “d”) using your graph search agent.

We have prepared a table in code given at a certain depth ,and the output is as follow



```
0 7.346153259277344e-05 56.0
50 0.02408447742462158 12659.36
100 0.1009417200088501 41074.88
150 0.5029793834686279 97952.96
200 0.6312635564804077 113785.28
250 0.8665931510925293 147300.16
300 0.46281589031219483 108028.48
350 0.5313785123825073 120244.32
400 0.969258394241333 148998.08
450 1.3145047616958618 182076.16
500 0.44059967041015624 108234.56
```

so we can see representation of depth ,time and memory as shown above. for more information about code checkout our github repository.