Week 1 :

Lab Assignment - 1

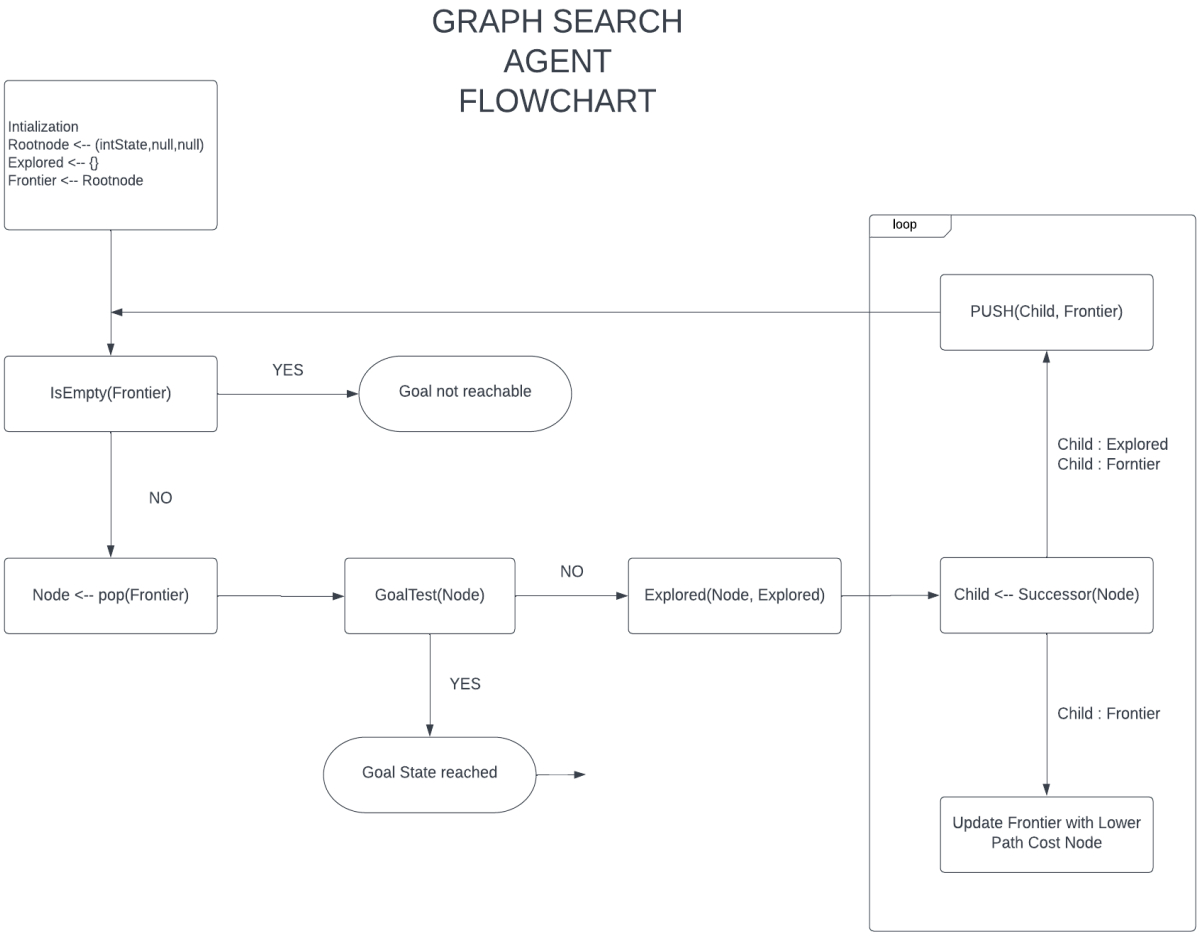
Here there are different space search agents which are informed.
Informed Space agent are further classified into different methods based on the weights:

$f(n) = g(n) + w*h(n)$

If w=0 BEST First Search (Djkstra)
If w=1 A* search algorithm
If w = infinite then it is hill climbing Algorithm
Below is the general flowchart of graph Search Agent.

We are going to demonstrate

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.



https://lucid.app/lucidchart/b2c82c20-6908-4325-98b2-6ded8165b09a/edit?viewport_loc=-760%2C-81%2C3485%2C1462%2C0_0&invitationId=inv_4e14ca5c-0b32-4015-8731-444fe3d3344f

Pseudocode for a graph search agent:

Current ← {start}
Explored ← {}
While Current is not empty
Do current .pop
if GoalTest(H) = TRUE
then return reverse(N) // returning the path
else
Explored ← Explored U {h}
Current. ←get(successor)

for each S in Successors
do Add Cons(S,N) to Current
return Path not found

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

  The environment is responsible for providing the states to the agents.It consists of start state goal state child states(successor states)
  Some different states with their execution:

1. get_start_state : returns the start state:

   code base:

   ```
   def get_start_state(self):

       return self.start_state
   ```

2. reached_goal : returns goal_state

```
def reached_goal(self, state):


    for i in range(3):
        for j in range(3):
            if state[i,j] != self.goal_state[i,j]:
                return False


    return True
```

3. get_next_states : Given current state, it returns all possible next states

This is the major function that is executed by the environment it provides subsequent next states:

```
def get_next_states(self, state):


    space = (0,0)
    for i in range(3):
        for j in range(3):
            if state[i,j] == '_':
                space = (i,j)
                break


    new_states = []


    if space[0] > 0:# Move Up
        new_state = np.copy(state)


        val = new_state[space[0], space[1]]
```

```
        new_state[space[0], space[1]]  = new_state[space[0]-1, space[1]]
        new_state[space[0]-1, space[1]] = val

        new_states.append(new_state)

    if space[0] < 2: #Move down
        new_state = np.copy(state)

        val = new_state[space[0], space[1]]
        new_state[space[0], space[1]]  = new_state[space[0]+1, space[1]]
        new_state[space[0]+1, space[1]] = val

        new_states.append(new_state)

    if space[1]<2: #Move right
        new_state = np.copy(state)

        val = new_state[space[0], space[1]]
        new_state[space[0], space[1]] = new_state[space[0], space[1]+1]
        new_state[space[0], space[1]+1] = val

        new_states.append(new_state)

    if space[1] > 0: #Move Left
        new_state = np.copy(state)

        val = new_state[space[0], space[1]]
```

```
        new_state[space[0], space[1]] = new_state[space[0], space[1]-1]

        new_state[space[0], space[1]-1] = val


        new_states.append(new_state)


    return new_states
```

4. generate_start_state : Given goal state and depth d, performs d moves to generate a start state

It provides random start state:

```
def generate_start_state(self):

    past_state = self.goal_state
    i=0
    while i!= self.depth:
        new_states = self.get_next_states(past_state)
        choice = np.random.randint(low=0, high=len(new_states))

        if np.array_equal(new_states[choice], past_state):
            continue

        past_state = new_states[choice]
        i+=1
```

- Describe what is Iterative Deepening Search.

IDS 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.

In IDS, we visit top level nodes multiple times. The last (or max depth) level is visited once, second last level is visited twice, and so on. It may seem expensive, but it turns out to be not so costly, since in a tree most of the nodes are in the bottom level. So it does not matter much if the upper levels are visited multiple times.

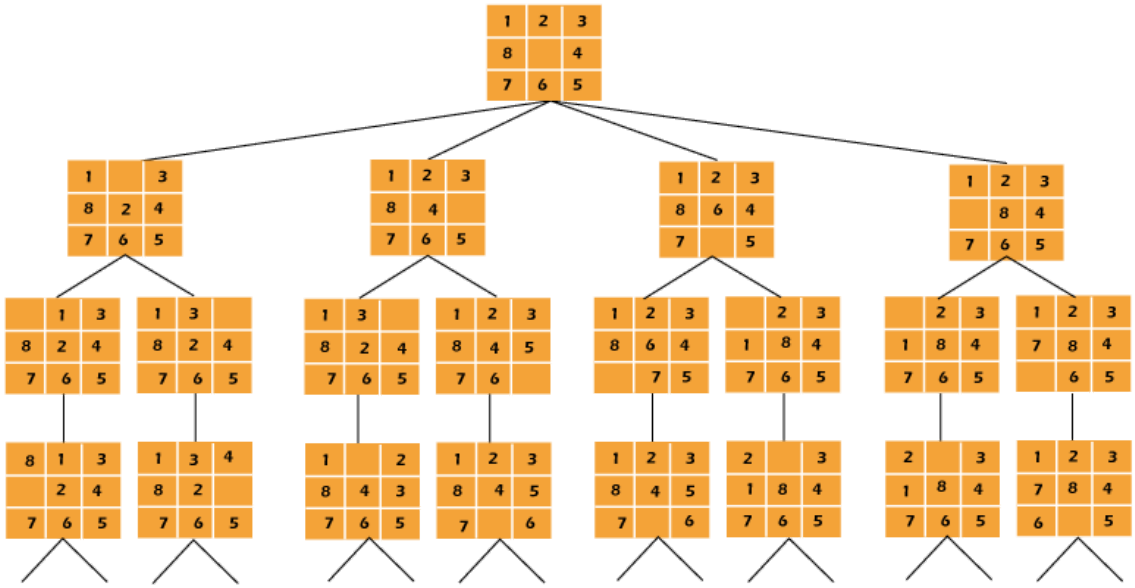
Time Complexity: Suppose we have a tree having branching factor 'b' (number of children of each node), and its depth 'd', i.e, there are **b^d** nodes. In an iterative deepening search, the nodes on the bottom level are expanded once, those on the next to bottom level are expanded twice, and so on, up to the root of the search tree, which is expanded d+1 times. So the total number of expansions in an iterative deepening search is-

(d)b + (d-1)b^2 + (d-2)b^3 + ... + 3b^d-2 + 2b^d-1 + b^d

which is the same as O(b^d).

After evaluating the above expression, we find that asymptotically IDS takes the same time as that of DFS and BFS, but it is indeed slower than both of them and takes less space compared to BFS i.e, O(bd).

- 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.



- Generate Puzzle-8 instances with the goal state at depth “d”.

Instances generated

1 8 2
0 4 3
7 6 5

1 8 2
4 0 3
7 6 5

1 0 2
4 8 3
7 6 5

1 2 0
4 8 3

7 6 5

1 2 3
4 8 0
7 6 5

1 2 3
4 8 5
7 6 0

1 2 3
4 8 5
7 0 6

1 2 3
4 0 5
7 8 6

1 2 3
4 5 0
7 8 6

1 2 3
4 5 6
7 8 0

ACKNOWLEDGMENT ::

1)environment functions :

https://www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/
Textbook : Artificial Intelligence - A modern Approach Chapter 3

2)heuristic function:
http://science.slc.edu/~jmarshall/courses/2005/fall/cs151/lectures/heuristic-search/#:~:text=A%20good%20heuristic%20for%20the%208%2Dpuzzle%20is%20the%20number,direct%20adjacent%20tile%20reversals%20present.

3)pseudocode- ques 1:

Taken from the reports of seniors.