CS633/362 Artificial Intelligence

Winter 2020-21

Week 1 : 11 Jan - 15 Jan 2021

Lab Assignment – 1

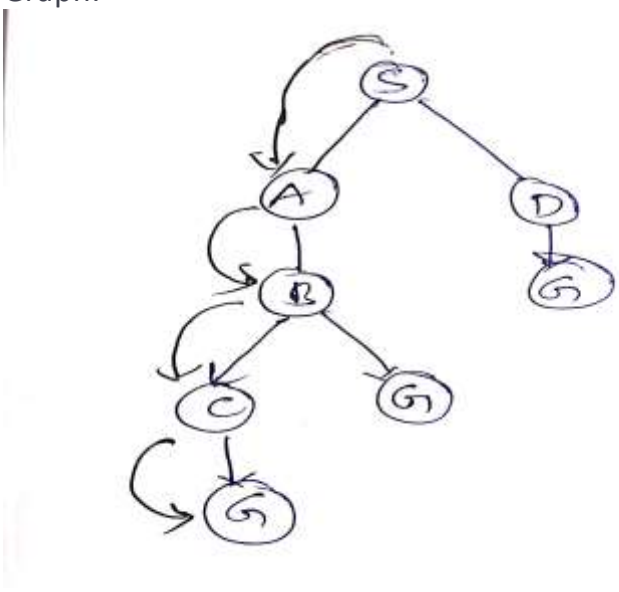
A. Write a pseudocode for a graph search agent. Represent the agent in the form of a flow chart. Clearly mention all the implementation details with reasons.

The following graph search algorithms are

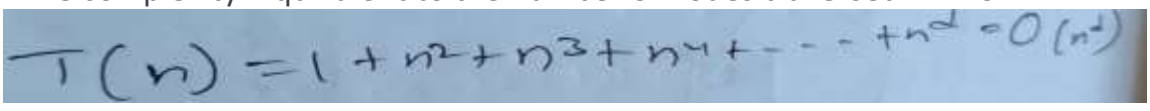
1. Depth First Search

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

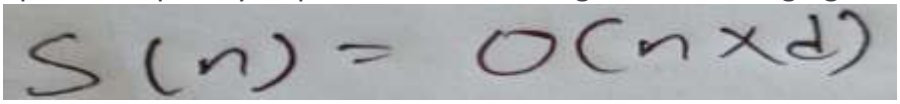
Graph:



Time complexity: Equivalent to the number of nodes traversed in DFS.



Space complexity: Equivalent to how large can the fringe get.



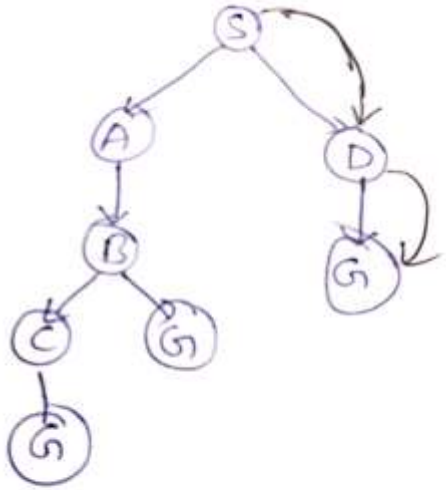
Completeness: DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.

Optimality: DFS is not optimal, meaning the number of steps in reaching the solution, or the cost spent in reaching it is high.

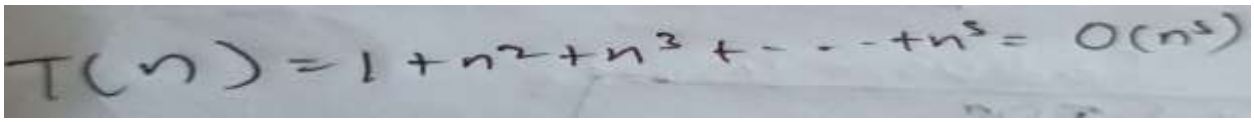
2. Breath First Search

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

Graph:



Time complexity: Equivalent to the number of nodes traversed in BFS until the shallowest solution.



Space complexity: Equivalent to how large can the fringe get.

Completeness: BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.

Optimality: BFS is optimal as long as the costs of all edges are equal.

3. Uniform Cost Search

UCS is different from BFS and DFS because here the costs come into play. In other words, traversing via different edges might not have the same cost. The goal is to find a path where the cumulative sum of costs is least.

Time complexity: 

Space complexity: 

C. Describe what is Iterative Deepening Search.

There are two common ways to break the graph, BFS and DFS. If you look at the tree (or Graph) height and width, both BFS and DFS do not work well for the following reasons.

DFSs begin to cut through nodes passing through one area near the roots, and then follow closely. The problem with this approach is that, if there is an area close to the roots, but not in the first few subtrees tested by DFS, DFS reaches that node too late. Also, DFS may not find the shortest route to the node (depending on the number of edges).

BFS goes a step further, but needs more space. The space required by DFS is O (d) where d is the depth of the tree, but the space required by BFS is O (n) when in the number of notes in the tree (Why? Note that the final tree level can be approximately n / 2 nodes and level the last second of the n / 4 node and in BFS we need to have all the levels one by one in line).

IDDFS combines the depth of initial search functionality with the scope of rapid search (root near nodes).

How does IDDFS work?

IDDFS calls DFS a different depth starting from the original value. In all calls, DFS is restricted from exceeding the given depth. So basically we do DFS in the form of BFS.

Algorithm:

// Returns the truth when the target is accessible from

// src within max_depth

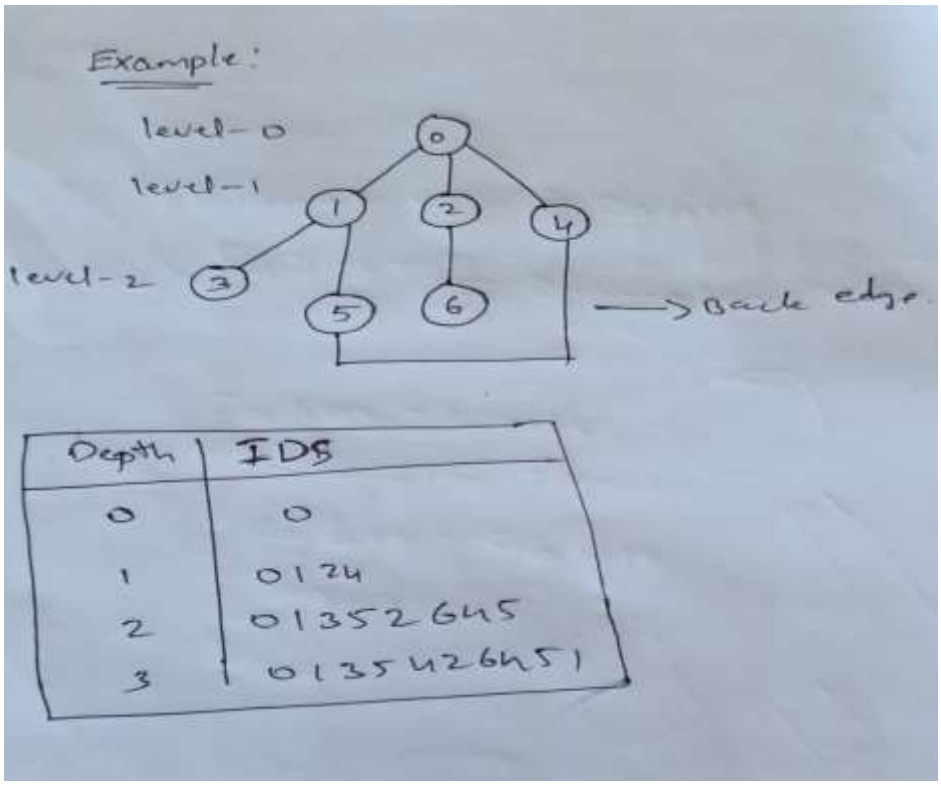
bool IDDFS (src, target, max_depth)

```
   in the range from 0 to max_depth
     if DLS (src, target, limit) == is true
        return true
  return false


bool DLS (src, target, limit)
   uma (src == target)
      true return;


   // When reached the maximum depth,
   // stop repeating.
   if (limit <= 0)
      return false;
 approaching near src
      if DLS (i, target, limit? 1)
         return true
       return false
```

There can be two cases-

a) Where the graph is not circular: This case is simple. We can DFS multiple times with different height limits.
b) the graph has cycles:This is interesting because no flag was visited on IDDFS.

Where



Time Distress: Suppose we have a tree with branching elements 'b' (number of children per node), and its depth 'd', that is, there are bd nodes.

In repeated depth studies, the lower nodes are expanded once, the lower level nodes are expanded twice, and so on, up to the root of the search tree, which is multiplied by d + 1.

Therefore the total expansion value in iterative search is

$(d)b + (d-1)b^2 + ... + 3b^{d-2} + 2b^{d-1} + b^d$

That is,

$$\text{Summation}[(d + 1 - i)\, b^i], \text{ from } i = 0 \text{ to } i = d$$

Which is same as $O(b^d)$

Space complexity-O(bd).

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.

In BFS and DFS, when we are in a node, we can view any proximity as follows

node. So BFS and DFS are both exploring methods without looking at any cost activity. The

the idea of Best First Search is to use the search function to determine which one is closest

promising and check. Best First Search falls under the category of Heuristic Search or

Informative search.

We use the front row to save the cost of nodes. So the implementation is a variation of BFS, us

i just need to change the line to PriorityQueue.

Algorithm:

Best-First-Search(Grah g, Node start)

1) Create an empty PriorityQueue PriorityQueue pq;

2) Insert "start" in pq. pq.insert(start)

3) Until PriorityQueue is empty

u = PriorityQueue.DeleteMin

If u is the goal

Exit

Else

Foreach neighbor v of u

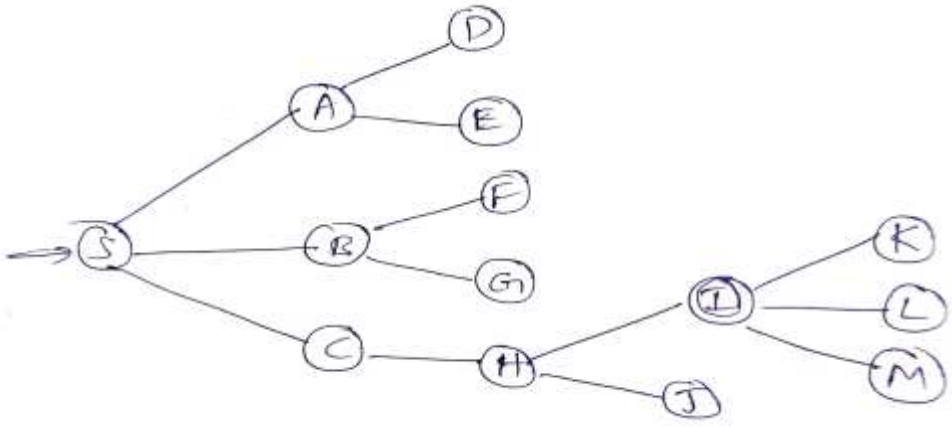
If v "Unvisited" Mark v "Visited"

pq.insert(v)

Mark v "Examined"

End procedure

Let us consider below example:



We start from source "S" and search for goal "I" using given costs and Best First search.

pq initially contains S

We remove s from and process unvisited

neighbors of S to pq.

pq now contains {A, C, B} (C is put before B because C has lesser cost)

We remove A from pq and process unvisited

neighbors of A to pq.

pq now contains {C, B, E, D}

We remove C from pq and process unvisited

neighbors of C to pq.

pq now contains {B, H, E, D}

We remove B from pq and process unvisited

neighbors of B to pq.

pq now contains {H, E, D, F, G}

We remove H from pq.

Since our goal

"I" is a neighbor of H, we return.

Analysis

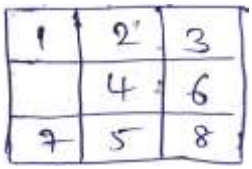
♣ The worst case time complexity for Best First Search is O(n * Log n) where n is number of nodes. In worst case, we may have to visit all nodes before we reach goal. Note that priority queue is implemented using Min(or Max) Heap, and insert and remove operations take O(log n) time.

♣ Performance of the algorithm depends on how well the cost or evaluation function is designed.

E. Generate Puzzle-8 instances with the goal state at depth "d".

Here We are solving a problem **of 8 puzzle** that is **a 3x3 matrix.**

Initial state



Goal state

Solution:

The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state.

Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile.

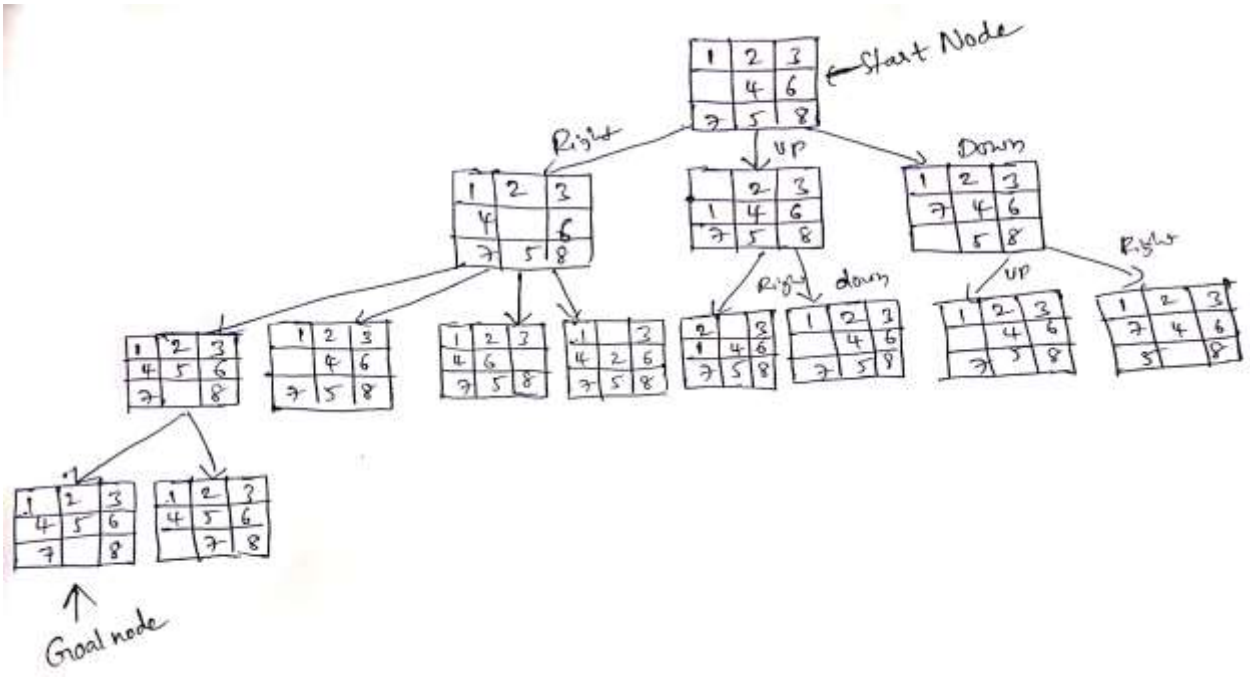
The empty space can only move in four directions (Movement of empty space)

1. Up
2. Down
3. Right or
4. Left

The empty space **cannot move diagonally** and can take only one step at a time.

Let's solve the problem without **Heuristic Search** that is **Uninformed Search or Blind Search (Breadth First Search and Depth First Search)**

Breath First Search to solve Eight puzzle problem



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

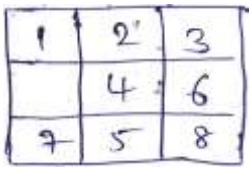
Time complexity: In worst case time complexity in BFS is O(b^d) know as order of b raise to power d. In this particular case it is (3^20).

b-branch factor

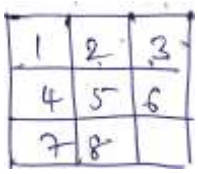
d-depth factor

To solve the problem with Heuristic search or informed search we have to calculate Heuristic values of each node to calculate cost function. (f=g+h)

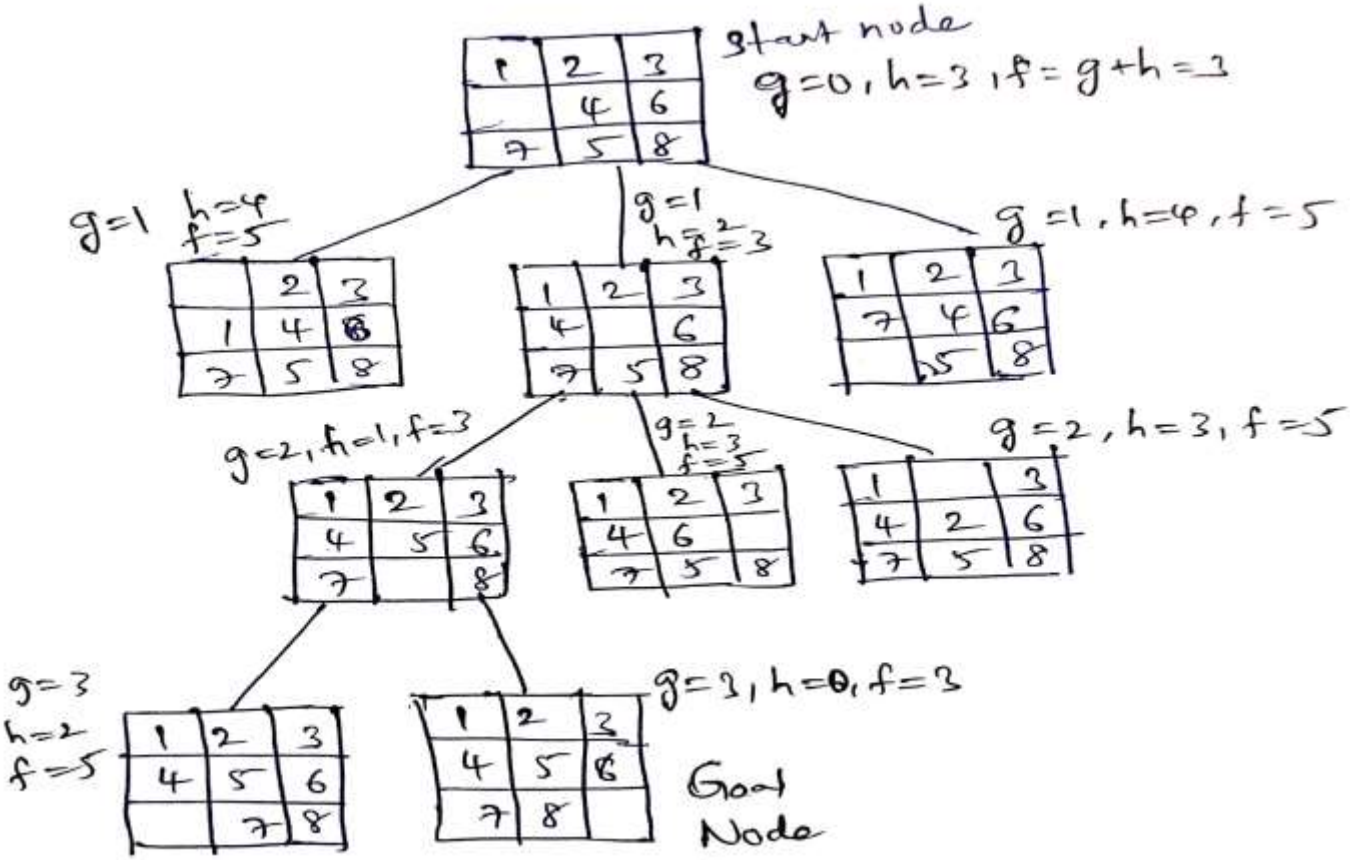
Initial state



Goal state



See the initial state and goal state carefully all values except (4,5 and 8) are at their respective places. **is** so, the heuristic value for first node 3**.**(Three values are misplaced to reach the goal). And **l**et's take actual cost (g) according to depth**.**


start node
g=0, h=3, f=g+h=3
g=1 h=4 f=5
g=1 h=2 f=3
g=1, h=4, f=5
g=2, h=1, f=3
g=2 h=3 f=5
g=2, h=3, f=5
g=3 h=2 f=5
g=3, h=0, f=3
Goal Node