Assignment no. 01

Problem statement:

Implement Depth first search algorithm and Breadth first search algorithm use an undirected graph and develop a recursive algorithm for searching all the vertices of the graph or tree data structure

Objective:

- To learn DFS algorithm
- To learn BFS algorithm

Software Requirements:

- Python3

Theory:

Depth first search (DFS):

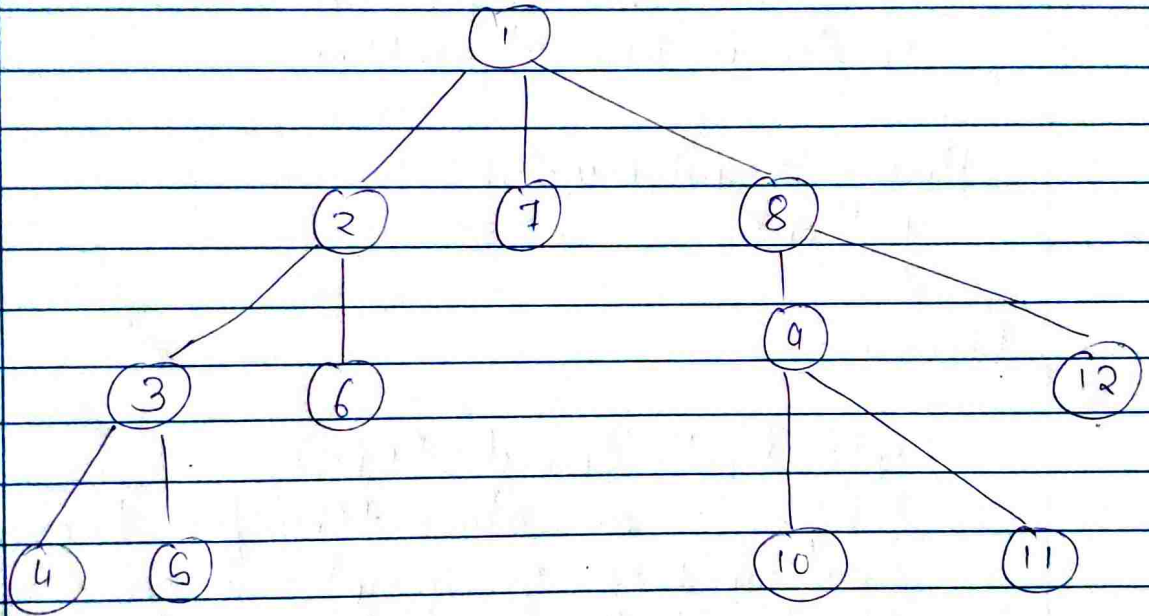
DFS is an algorithm for traversing or searching tree or graph data structures One starts at the root (selecting some arbitrary node as the root for a graph) and explore as far as possible along each branch before backtracking.

Depth first Search in Trees:

A tree Search in Trees in which any two vertices are connected by exactly one path. In other words, any acyclic connected graph is a tree. for a tree, we connected graph is a have the following traversal method:

- Preorder: visit each node before its children.
- Postorder: visit each node after its children.
- Inorder: visit left subtree, node, right subtree.

The following graph shows the order in which the nodes are discovered in DFS.



So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node.

Then backtrack and check for other unmarked nodes and traverse

adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally print the nodes in the path.

Below are steps to solve the problem:

- Create a recursive function that takes the index of the node and a visited array.
- Mark the current node are visited and print the node.
- Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

Advantages of Depth first search:

Memory requirement is only linear with respect to the search graph. This is in contrast with breadth-first search which requires more space. The reason is that the algorithm only needs to store a stack of nodes on the path from the root to the current node.

The time complexity of a depth first search to depth d is $O(b^d)$ since it generates the same set of nodes as breadth-first search, but simply in a different order. Thus practically depth-first search is time limited rather than space-limited.

If depth-first search finds solution without exploring much in a path then the time and space it takes will be very less.

DFS requires less memory since only the nodes on the current path are stored. By chance DFS may find a solution without examining much of the search space at all.

Breadth first Search (BFS)

BFS is an algorithm is used to search a tree or graph data structure for a node that meets a set of criteria. It starts at the tree's root or graph and searches / visits all nodes at the current depth level before moving on to the nodes at the next depth level. Breadth-first search can be used to solve many problems in a graph theory.

Breadth-first Traversal (or search) for a graph is similar to the Breadth-first Traversal of a tree.

Unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we divide the vertices into two categories visited and not visited.

A Boolean visited array is used to mark the visited vertices. It is assumed that all vertices are reachable from the starting vertex. BFS uses a queue data structure for traversal.

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next level neighbour nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

First move horizontally and visit all the nodes of the current layer.

Move to the next layer.

A graph traversal is a commonly used methodology for locating the vertex position in the graph. It is an advanced search algorithm that can analyze the graph with speed and precision along with marking the sequence of the visited vertices. This process enables you to quickly visit each node in a graph without being locked in an infinite loop.

In the various levels of the data, we can mark any node as the starting or initial node to begin traversing. The BFS will visit the node and mark it as visited and places it in the queue.

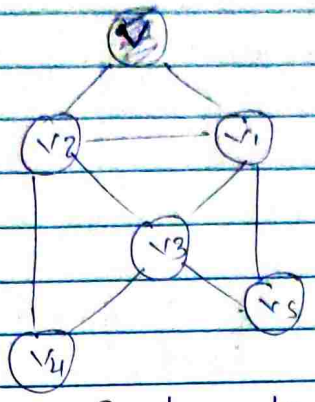
Now the BFS will visit the nearest and unvisited nodes and mark them. These values are also added to the queue. The queue works on the FIFO model.
In a similar manner, the remaining nearest and unvisited nodes on the graph are analyzed marked and added to the queue. These items are deleted from the queue as receive and printed as the result.

Graph traversal requires the algorithm to visit, check, and/or update every single unvisited node in a tree-like structure. Graph traversals are categorized by the order in which they visit the nodes on the graph.

BFS algorithm starts the operation from the first or starting node in a graph and traverses it thoroughly. Once it successfully traverses the initial node, then the next non-traversed vertex in the graph is visited and marked.

All nodes adjacent to the current vertex are visited and traversed in the first iteration.
A simple

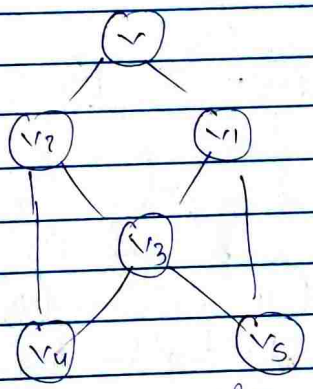
Step 1)



Root node = v

Each vertex or node in the graph is known. for instance, you can mark the node as V.

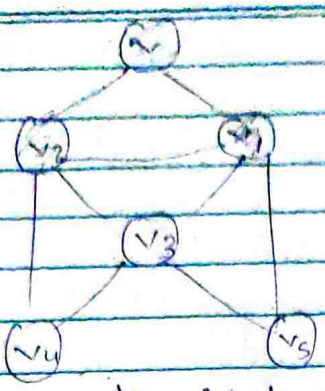
Step2)



In case v is not visited add it to the BFS queue.

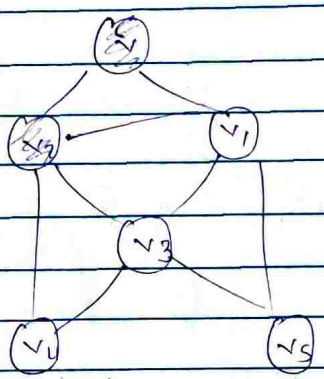
Queue : | v |

Step3) Start the BFS search, and after completion, Mark vertex v as visited



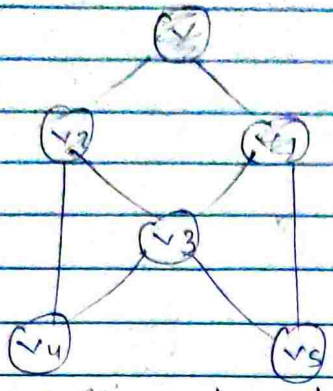
Delete vertex V from the queue.

Step4 : The BFS queue is still not empty, hence remove the vertex V of the graph from the queue.



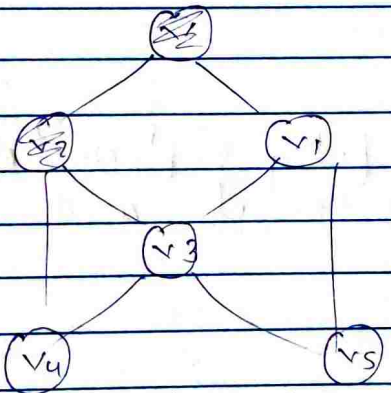
1. Start the BFS search
2. After completion, mark V as completed.

Step5 : Retrieve all the remaining vertices on the graph that are adjacent to the vertex V.



visit and retrieve all the adjacent and un-visited nodes from the node v.

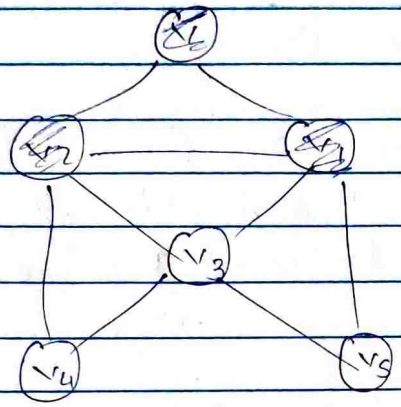
Step 6) for each adjacent vertex let's say v1, in case it is not visited yet then add v1 to the BFS queue.



for adjacent and unvisited vertex say v1, add it to BFS queue.

Queue : ~~v~~ v1

Step 1) BFS will visit v1 and mark it as visited and delete it from the queue.



BFS will visit v1, mark it as visited and delete it from the queue.

Queue = ~~v1~~ | ~~v1~~

Conclusion :

Successfully implemented Depth first search (DFS) search and Breadth first Search (BFS).

JSPM
Page No.

Assignment no. 02

Problem statement :
Implement A star (A*) Algorithm for any game search problem.

Objective
- To learn A start (A*) Algorithm.

Software Requirements :
- Python 3

Theory :

A* Algorithm

→ A* Algorithm is one of the best and popular techniques and used for path finding and graph traversals.

→ A lot of games and web based maps use this algorithm for finding the shortest path efficiently.

→ It is essentially a best first search algorithm.

Algorithm

The implementation of A* Algorithm involves maintaining two lists OPEN and CLOSED.

→ OPEN contains those nodes that have been evaluated by the heuristic function but have not been expanded into successors yet.
→ CLOSED contains those nodes that have already been visited.

The algorithm is as follows -

Step - 01 :
Define a list OPEN
Initially, open contains solely of a single node, the start node.

Step-02 :
If the list is empty, return failure and exit.

Step-03 :
Remove node n with the smallest value of f(n) from OPEN and move it to list CLOSED.
If node n is goal state, return success and exit

Step 4:
Expand node n.

Step 5:
If any successor to n is the goal node, return success and the solution by tracing the path from goal node to S.
Otherwise, go to step-06.

Step 6:
For each successor node,
→ Apply the evaluation function f to the node.
→ If the node has not been in either list, add it to OPEN.

Step 7:
Go back to step-02.

Practice problem based on A* Algorithm

Given an initial state of a 8-puzzle problem and final state to be reached.

2	8	3
1	6	4
7		5

initial state

1	2	3
8		4
7	6	5

final state

Find the most cost-effective path to reach the final state from initial state using A* Algorithm.

Consider $g(n)$ = Depth of node &
$h(n)$ = No of misplaced tiles.

Solution -

A* Algorithm maintains a tree of paths originating at the initial state.

→ It extends those paths one edge at a time.

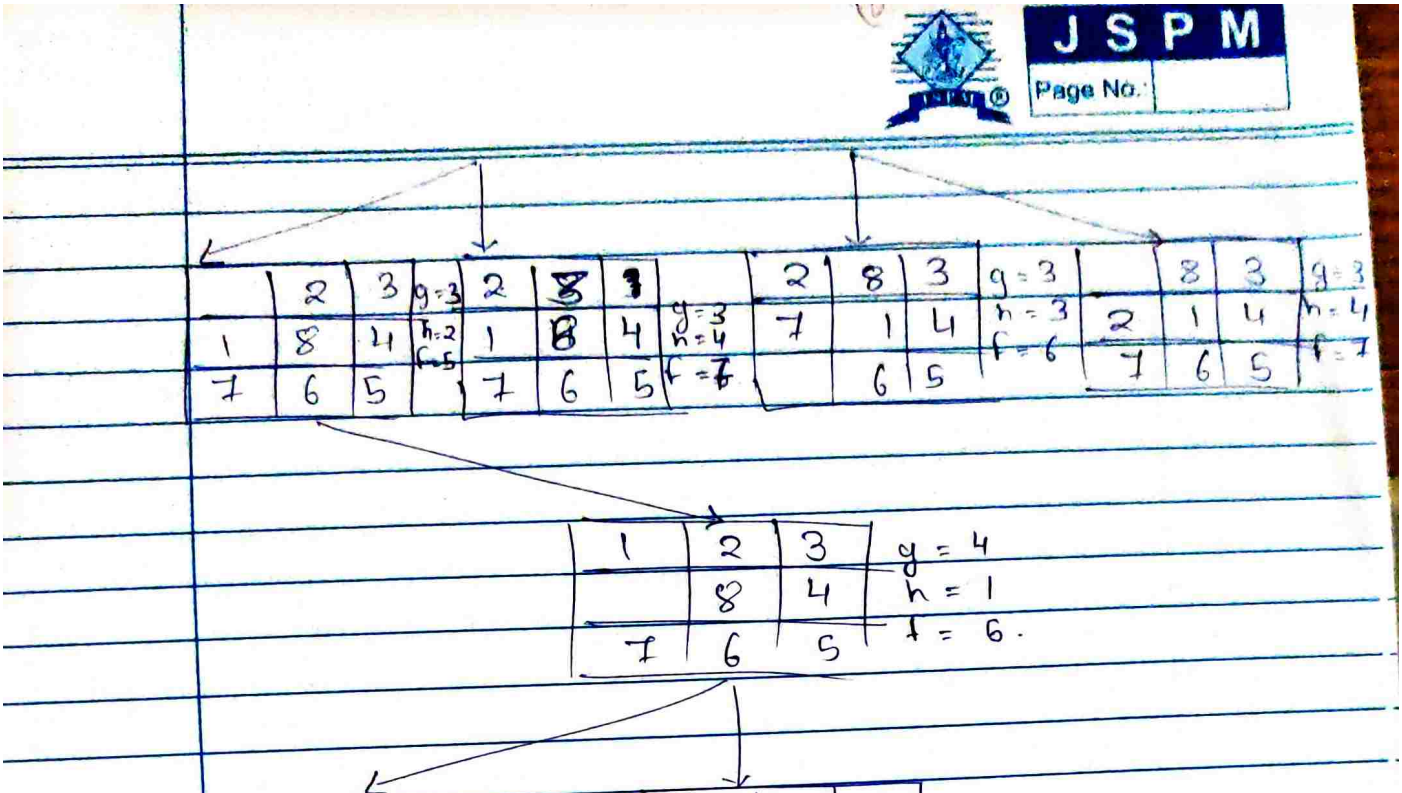
→ It continues until final state is reached

Initial state.

2	8	3	g = 0
1	6	4	h = 4
7		5	f = 0 + 4 = 4

2	8	3	g=1	2	8	3	g=1	2	8	3	g = 1
1	6	4	h=5	1		4	h=3	1	6	4	h = 5
	7	5	f=6	7	6	5	f=4	7	5		f = 6

2		3	g=2	2	8	3	g=2				g = 0
1	8	4	h=3		1	4	h=3				h = 4
7	6	5	f=5	7	6	5	f=5				f = 6



Final state.

Conclusion:

Successfully implement A star (A*) algorithm for 8-puzzle problem.