# BFS algorithm in Python

import collections

# BFS algorithm

def bfs(graph, root):

visited, queue = set(), collections.deque([root])

visited.add(root)

while queue:

# Dequeue a vertex from queue

vertex = queue.popleft()

print(str(vertex) + " ", end="")

# If not visited, mark it as visited, and # enqueue it

for neighbour in graph[vertex]:

if neighbour not in visited:

visited.add(neighbour)

queue.append(neighbour)

if \_\_name\_\_ == '\_\_main\_\_':

graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}

print("Following is Breadth First Traversal: ")

bfs(graph, 0)

output

Following is Breadth First Traversal:

0 1 2 3

DFS

graph = {

    '0':['1', '2'], '1':['2'],'2':['3'],'3':['1', '2']

}

visited =set()

def dfs(visited,graph,root):

    if root not in visited:

       print(root)

       visited.add(root)

       for neighbour in graph[root]:

          dfs(visited,graph,neighbour)

dfs(visited,graph,'0')

OUTPUT :

0

1

3

4

2

**A\* Program**

Graph\_nodes = {

    'A': [('B', 6), ('F', 3)],

    'B': [('C', 3), ('D', 2)],

    'C': [('D', 1), ('E', 5)],

    'D': [('C', 1), ('E', 8)],

    'E': [('I', 5), ('J', 5)],

    'F': [('G', 1),('H', 7)] ,

    'G': [('I', 3)],

    'H': [('I', 2)],

    'I': [('E', 5), ('J', 3)],

}

def get\_neighbors(v):

    if v in Graph\_nodes:

        return Graph\_nodes[v]

    else:

        return None

def h(n):

        H\_dist = {

            'A': 10,

            'B': 8,

            'C': 5,

            'D': 7,

            'E': 3,

            'F': 6,

            'G': 5,

            'H': 3,

            'I': 1,

            'J': 0

        }

        return H\_dist[n]

def aStarAlgo(start\_node, stop\_node):

        open\_set = set(start\_node)

        closed\_set = set()

        g = {}

        parents = {}

        g[start\_node] = 0

        parents[start\_node] = start\_node

        while len(open\_set) > 0:

            n = None

            for v in open\_set:

                if n == None or g[v] + h(v) < g[n] + h(n):

                    n = v

            if n == stop\_node or Graph\_nodes[n] == None:

                pass

            else:

                for (m, weight) in get\_neighbors(n):

                    if m not in open\_set and m not in closed\_set:

                        open\_set.add(m)

                        parents[m] = n

                        g[m] = g[n] + weight

                    else:

                        if g[m] > g[n] + weight:

                            g[m] = g[n] + weight

                            parents[m] = n

                            if m in closed\_set:

                                closed\_set.remove(m)

                                open\_set.add(m)

            if n == None:

                print('Path does not exist!')

                return None

            if n == stop\_node:

                path = []

                while parents[n] != n:

                    path.append(n)

                    n = parents[n]

                path.append(start\_node)

                path.reverse()

                print('Path found: {}'.format(path))

                return path

            open\_set.remove(n)

            closed\_set.add(n)

        print('Path does not exist!')

        return None

aStarAlgo('A', 'J')

**Output**

Path found: ['A', 'F', 'G', 'I', 'J']

['A', 'F', 'G', 'I', 'J']