**Task 1:**

**Drawing a Graph and complete BFS:**

from collections import deque

class graph:

    def \_\_init\_\_(self):

        self.l={}

    def addedge(self,p,c):

        if(p not in self.l):

            self.l[p]=[]

        self.l[p].append(c)

    def \_\_str\_\_(self):

        return str(self.l)

    def bfs(self,start):

        visited=[]

        queue=deque()

        visited.append(start)

        queue.append(start)

        while queue:

            node=queue.popleft()

            print(node)

            for neighbour in self.l[node]:

                if neighbour not in visited:

                    visited.append(neighbour)

                    queue.append(neighbour)

g=graph()

g.addedge(0,1)

g.addedge(0,2)

g.addedge(1,2)

g.addedge(2,0)

g.addedge(2,3)

g.addedge(3,3)

print(g)

print("The BFS Traversal of the Graph is:")

g.bfs(1)

**Task 2:**

**Implementing the BFS to a Specified Node:**

from collections import deque

class graph:

    def \_\_init\_\_(self):

        self.l={}

    def addedge(self,p,c):

        if(p not in self.l):

            self.l[p]=[]

        self.l[p].append(c)

    def \_\_str\_\_(self):

        return str(self.l)

    def bfs(self,start,goal):

        visited=[]

        queue=deque()

        visited.append(start)

        queue.append(start)

        while queue:

            node=queue.popleft()

            print(node)

            if(node==goal):

                print("GOAL NODE FOUND...!!!")

                return 1

            for neighbour in self.l[node]:

                if neighbour not in visited:

                    visited.append(neighbour)

                    queue.append(neighbour)

g=graph()

g.addedge('A','B')

g.addedge('A','F')

g.addedge('A','D')

g.addedge('A','E')

g.addedge('B','K')

g.addedge('B','J')

g.addedge('B','A')

g.addedge('F','A')

g.addedge('D','G')

g.addedge('D','A')

g.addedge('E','C')

g.addedge('E','H')

g.addedge('E','I')

g.addedge('E','A')

g.addedge('K','N')

g.addedge('K','M')

g.addedge('K','B')

g.addedge('J','B')

g.addedge('G','D')

g.addedge('C','I')

g.addedge('H','I')

g.addedge('I','I')

g.addedge('I','E')

g.addedge('I','L')

g.addedge('N','K')

g.addedge('M','K')

g.addedge('L','I')

print(g)

print("The BFS Traversal of the Graph is:")

g.bfs('A','G')

**Task 3:**

**Implementing the Priority Queue:**

class pqueue:

    def \_\_init\_\_(self):

        self.l=[]

    def enque(self,p,ele):

        self.l.append((p,ele))

        self.l.sort()

    def size(self):

        return len(self.l)

    def is\_empty(self):

        if(len(self.l)==0):

            return True

        return False

    def pop(self):

        if(self.is\_empty()):

            return "QUEUE IS EMPTY..."

        return self.l.pop(0)[1]

    def \_\_str\_\_(self):

        return str([ele for p ,ele in (self.l)])

q=pqueue()

q.enque(5,10)

q.enque(3,20)

q.enque(1,30)

q.enque(4,50)

print(q)

print(q.pop())

print(q)

q.pop()

q.pop()

print(q.size())

print(q)

q.enque(2,100)

print(q)

q.pop()

print(q.pop())

print(q.pop())

print(q)