In [ ]:

```
from Graph import Graph, Vertex
from Queue import Queue
```

BREADTH-FIRST SEARCH

In [ ]:

```
def breadth_first_search(graph, start_vertex, distances = {}):
    discovered_set = set()
    frontier_queue = Queue()
    visited_list = []

    distances[start_vertex] = 0 # start vertex has a distance of 0 from i

    frontier_queue.enqueue(start_vertex)
    discovered_set.add(start_vertex)

    while (frontier_queue.list.head != None):
        current_vertex = frontier_queue.dequeue()
        visited_list.append(current_vertex)
        for adjacent_vertex in graph.adjancency_list[current_vertex]:
            if adjacent_vertex not in discovered_set:
                frontier_queue.enqueue(adjacent_vertex)
                discovered_set.add(adjacent_vertex)

                distances[adjacent_vertex] = distances[current_vertex] +
    return visited_list
```

In [ ]:

```
def depth_first_search(graph, start_vertex, visited_func):
    vertex_stack = [start_vertex]
    visited_set = set()

    while len(vertex_stack) > 0:
        current_vertex = vertex_stack.pop()
        if current_vertex not in visited_set:
            visited_func(current_vertex)
            visited_set.add(current_vertex)
            for adjacent_vertex in graph.adjacency_list[current_vertex]:
                vertex_stack.append(adjacent_vertex)
                if adjacent_vertex not in visited_set:
                    visited_func(adjacent_vertex)
                    visited_set.add(adjacent_vertex)
    return visited_set
```

In [ ]:

```
g = Graph()
vertex_a  =  Vertex('Joe')
vertex_b  =  Vertex('Eva')
vertex_c  =  Vertex('Taj')
vertex_d  =  Vertex('Chen')
vertex_e  =  Vertex('Lily')
vertex_f  =  Vertex('Jun')
vertex_g  =  Vertex('Ken')

vertices = [vertex_a, vertex_b, vertex_c, vertex_d, vertex_e, vertex_f, v

for vertex in vertices:
    g.add_vertex(vertex)
```

In [ ]:

```
g.add_undirected_edge(vertex_a, vertex_c)
g.add_undirected_edge(vertex_b, vertex_e)
g.add_undirected_edge(vertex_c, vertex_d)
g.add_undirected_edge(vertex_c, vertex_e)
g.add_undirected_edge(vertex_d, vertex_f)
g.add_undirected_edge(vertex_e, vertex_f)
g.add_undirected_edge(vertex_f, vertex_g)


start_name = input("Enter starting person's name")
print()
```

In [ ]:

```
start_vertex = None

for vertex in vertices:
    if vertex.label == start_name:
        start_vertex = vertex

if start_vertex is None:
    print(f"Start vertex not found {start_name}")
else:
    vertex_distances = {}
    visited_list = breadth_first_search(g, start_vertex, vertex_distances

    print("Breadth-first search transerval")
    print(f"start vertex {start_vertex.label}")
    for vertex in visited_list:
        print(f"{vertex.label} : {vertex_distances[vertex]}")
```

```
Start vertex not found raf
```

In [ ]:

```
# server:

g = Graph()
vertex_a  =  Vertex('A')
vertex_b  =  Vertex('B')
vertex_c  =  Vertex('C')
vertex_d  =  Vertex('D')
vertex_e  =  Vertex('E')
vertex_f  =  Vertex('F')
vertex_g  =  Vertex('G')
vertex_h  =  Vertex('H')
vertex_i  =  Vertex('I')
vertex_j  =  Vertex('J')


vertices = [vertex_a, vertex_b, vertex_c, vertex_d, vertex_e, vertex_f, v

for vertex in vertices:
    g.add_vertex(vertex)
```

In [ ]:

```
g.add_undirected_edge(vertex_a, vertex_b)
g.add_undirected_edge(vertex_b, vertex_c)
g.add_undirected_edge(vertex_b, vertex_f)
g.add_undirected_edge(vertex_c, vertex_d)
g.add_undirected_edge(vertex_c, vertex_g)
g.add_undirected_edge(vertex_d, vertex_g)
g.add_undirected_edge(vertex_d, vertex_h)
g.add_undirected_edge(vertex_e, vertex_b)
```

```
g.add_undirected_edge(vertex_e, vertex_f)
g.add_undirected_edge(vertex_e, vertex_i)
g.add_undirected_edge(vertex_f, vertex_c)
g.add_undirected_edge(vertex_f, vertex_i)
g.add_undirected_edge(vertex_g, vertex_h)
g.add_undirected_edge(vertex_g, vertex_j)



start_name = input("Enter server's name")
print()
```

In [ ]:

```
star_vertex = None

for vertex in vertices:
    if vertex.label == start_name:
        start_vertex = vertex

if start_vertex is None:
    print(f"Start vertex not found {start_name}")
else:
    vertex_distances = {}
    visited_list = breadth_first_search(g, start_vertex, vertex_distances

    print("Breadth-first search transerval")
    print(f"start vertex {start_vertex.label}")
    for vertex in visited_list:
        print(f"{vertex.label} : {vertex_distances[vertex]}")
```

```
Start vertex not found malo
```

DEPTH-FIRST SEARCH

In [ ]:

```
from Graph import Vertex, Graph
```

In [ ]:

```
def depth_first_search(graph, start_vertex, visitor, visited=None):
    if visited is None:
        visited = set()
    visitor(start_vertex)
    visited.add(start_vertex)
    for neighbor in graph.adjacency_list[start_vertex]:
        if neighbor not in visited:
            depth_first_search(graph, neighbor, visitor, visited)
```

In [ ]:

```
import time

class Graph:
    def __init__(self):
        self.vertices = {}
        self.adjacency_list = {}  # Add adjacency_list attribute

    def add_vertex(self, vertex):
        self.vertices[vertex.label] = vertex
        self.adjacency_list[vertex] = []  # Initialize adjacency list for

    def get_vertex(self, label):
        return self.vertices[label]
    def add_undirected_edge(self, vertex1, vertex2):
        vertex1.add_neighbor(vertex2)
        vertex2.add_neighbor(vertex1)
        self.adjacency_list[vertex1].append(vertex2)  # Update adjacency
        self.adjacency_list[vertex2].append(vertex1)  # Update adjacency

class Vertex:
    def __init__(self, label):
        self.label = label
        self.neighbors = []

    def add_neighbor(self, vertex):
        self.neighbors.append(vertex)

    def get_neighbors(self):
        return self.neighbors

def depth_first_search(graph, start_vertex, visitor, visited=None):
    if visited is None:
        visited = set()
    visitor(start_vertex)
    visited.add(start_vertex)
    for neighbor in graph.adjacency_list[start_vertex]:
        if neighbor not in visited:
            depth_first_search(graph, neighbor, visitor, visited)

vertex_names = [chr(i) for i in range(ord('A'), ord('Z')+1)]  # Increase

graph1 = Graph()
graph2 = Graph()
graph3 = Graph()
graphs = [graph1, graph2, graph3]

for vertex_name in vertex_names:
    for graph in graphs:
        graph.add_vertex(Vertex(vertex_name))

# Add more edges to each graph
for i in range(len(vertex_names) - 1):
    graph1.add_undirected_edge(graph1.get_vertex(vertex_names[i]), graph1
    graph2.add_undirected_edge(graph2.get_vertex(vertex_names[i]), graph2
    graph3.add_undirected_edge(graph3.get_vertex(vertex_names[i]), graph3

visitor = lambda x: print(x.label, end = ' ')

start_vertex_label = "A"

for i in range(0, len(graphs)):
    start_time = time.time()
    print(f"Graph {i+1} : ", end="")
    depth_first_search(graphs[i], graphs[i].get_vertex(start_vertex_label
    print("\n")
    end_time = time.time()
    print(f"Execution time for Graph {i+1}: {end_time - start_time} secon
Graph 1 : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Execution time for Graph 1: 0.0008463859558105469 seconds
Graph 2 : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Execution time for Graph 2: 0.00027704238891601560 seconds
Graph 3 : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Execution time for Graph 3: 0.0002911090850830078 seconds
```

```
In [ ]:
```