FIT2004 Assignment 4

Student: Siew Ming Shern
StudentID: 28098552

Ming Siew

Task 1

```
def buildGraph(self, filename_roads):
```

read content of filename into EdgeList[]

```
    maxV = getLargestVertex(EdgeList)

    self.list = [None] * (maxV+1)

    self.newGraph = []

    for index in range(len(self.list)):
        self.list[index] = Vertex(index)

    for edge in edgeList:
        self.add(edge)
```

Time complexity: O(V + E), occurs when V times is needed to create array with v size and all vertices will holds atmost E edges in total which will takes E time.

Space complexity: O(V + E), occurs when V space is needed to create array with v size and all vertices will holds atmost E edges in total which will takes E space.

where V is the total number of points/ nodes/ vertices in the road network and E is the total number of roads/edges in the road network.

```
def quickestPath(self,source, target):

    source = int(source)

    target = int(target)

    if new graph is empty or source out of range or target out of range
    : return [[], -1]

    for vertex in self.list:
        vertex.reset()

    self[source].time = 0

    self[source].originPath = target

    discover = MinHeap(len(self))

    discover.add(0, source)

    while len(discover) != 0:
        item = discover.extract()
        u = self[item[0]]
        u.visited = True
        for edges in u.edge:
            v = self[edges.vertex]
```

```
            if v.visited == False:
                if v.discovered == False:
                    discover.add(item[1] + edges.weight, edges.vertex)
                    v.update(item[1] + edges.weight, item[0])
                else:
                    if v.time > item[1] + edges.weight:
                        discover.update(edges.vertex, item[1] + edges.weight)
                        v.update(item[1] + edges.weight, item[0])

    if self[target].originPath == None:
            return [[], -1]

        location = self[target].originPath

        path = [target]

        while location != target:
            path.append(location)
            location = self[location].originPath

        fPointer = 0
        lPointer = len(path) - 1
        while fPointer <= lPointer:
            path[fPointer], path[lPointer] = path[lPointer], path[fPointer]
            fPointer += 1
            lPointer -= 1

        retVal = [path, self[target].time]
        return retVal
```

Time complexity: O(E Log V), occurs for each Edges, rise method of heap is needed to perform to ensure new item fits in the heap which result O(E)*O(Log V).

Space complexity: O(V), occurs when V space is needed to store V vertices in path and additional V space is needed to store atmost all vertex in graph into min heap which result O(v).

where V is the total number of points/ nodes/ vertices in the road network and E is the total number of roads/edges in the road network.

Task 2

```
def augmentGraph(self,filename_camera, filename_toll):

    read content of filename_camera into redLightList[]

    read content of filename_toll into tollList[]

    for redLight in redLightList:
        self.list[redLight].isRedLight = True

    for vertex in tollList:
        for edges in self.list[vertex[0]].edge:
            if edges.vertex == vertex[1]:
                edges.isToll = True
                break
```

Time complexity: O(V + E), occurs when V times is needed to create array with V size which are list of vertices considered unsafe,and list of edges with E size in graph considered unsafe.

Space complexity: O(V + E), occurs when V space is needed to create array with V size which are considered red light and E space is needed for all edges which are toll

where V is the total number of points/ nodes/ vertices in the road network and E is the total number of roads/edges in the road network.

```
def quickestSafePath(self, source, target):

    source = int(source)

    target = int(target)

    if new graph is empty or source out of range or target out of range
    : return [[], -1]

    for vertex in self.list:
        vertex.reset()

    self[source].time = 0

    if self[source].isRedLight == False:
        self[source].originPath = target

    discover = MinHeap(len(self))

    discover.add(0, source)

    while len(discover) != 0:
        item = discover.extract()
        u = self[item[0]]
        u.visited = True
        if u.isRedLight == False:
            for edges in u.edge:
```

```
                v = self[edges.vertex]
                if v.visited == False and v.isRedLight == False
                 and edges.isToll == False:
                    if v.discovered == False:
                        discover.add(item[1] + edges.weight, edges.vertex)
                        v.update(item[1] + edges.weight,item[0])
                    else:
                        if v.time > item[1] + edges.weight:
                            discover.update(edges.vertex, item[1] + edges.weight)
                            v.update(item[1] + edges.weight, item[0])

    if self[target].originPath == None:
        return [[], -1]

    location = self[target].originPath

    path = [target]


    while location != target:
        path.append(location)
        location = self[location].originPath

    fPointer = 0
    lPointer = len(path) - 1
    while fPointer <= lPointer:
        path[fPointer], path[lPointer] = path[lPointer], path[fPointer]
        fPointer += 1
        lPointer -= 1

    retVal = [path, self[target].time]
    return retVal
```

Time complexity: O(E Log V), occurs when there is no redlight and no toll which causes for each Edges which will be considered safe, rise method of heap is needed to perform
to ensure new item fits in the heap which result O(E)*O(Log V).

Space complexity: O(V), occurs when there is no redlight and no toll V space is needed to store V vertices in path and additional V space is needed to store atmost all vertex in graph into min heap which result O(v).

where V is the total number of points/ nodes/ vertices in the road network and E is the total number of roads/edges in the road network.

Task 3

```
def addService(self, filename_service):

    read content of filename_service into serviceList[]

    self.newGraph = []

    numOfRoad = len(self.list)

    for item in self.list:
       self.newGraph.append(item)

    for index in range(len(self.list)):
        items = self.list[index].copy(len(self))
        self.newGraph.append(items)

    for service in serviceList:
        self.newGraph[service].isService = True

    for index in range(len(self.list)):
        for edges in self.newGraph[index].edge:
            if self.newGraph[edges.v].isService:
               self.newGraph[index + numOfRoad].edge.append(edges)
```

Time complexity: O(V + E), occurs when list of service on vertices are as much as V size is needed to be created, a new graph is created based on original graph created in basicGraph.txt with number of vertices and number of edges is doubled and because the original graph will have two copy of its graph on new graph with one of a copy graph have vertices which are marked as an service node when it is a service. At this rate, the new graph,c(g , g')have a graph, g which have a service node in vertices marks with another graph, g'. Both are exactly the same with index differ only. Finally, an additional atmost E edges is needed to be created to link a vertices in g'and vertices in g which is a services and is a vertices mirror to a vertices g'. This leads to O(2V + 3E) which is also equivalent to O(V + E).

Space complexity: O(V + E), occurs when V space is needed to create array with 2V size and all vertices will holds atmost 3E edges in total which will takes O(V + E)

where V is the total number of points/ nodes/ vertices in the road network and E is the total number of roads/ edges in the road network.

```
def quickestDetourPath(self, source, target):
    source = int(source)

    target = int(target)

    if new graph is empty or source out of range or target out of range:
        return [[], -1]

    for vertex in self.newGraph:
        vertex.reset()

    if self.newGraph[source].isService:
        startPoint = source
    else:
        startPoint = source + len(self.list)

    self.newGraph[startPoint].time = 0

    self.newGraph[startPoint].originPath = target

    discover = MinHeap(len(self.newGraph))

    discover.add(0, startPoint)

    while len(discover) != 0:
        item = discover.extract()
        u = self.newGraph[item[0]]
        u.visited = True
        for edges in u.edge:
            v = self.newGraph[edges.v]
            if v.visited == False:
                if v.discovered == False:
                    discover.add(item[1] + edges.w, edges.v)
                    v.update(item[1] + edges.w,item[0])
                else:
                    if v.time > item[1] + edges.w:
                        discover.update(edges.v, item[1] + edges.w)
                        v.update(item[1] + edges.w,item[0])

    if self.newGraph[target].originPath == None:
        return [[], -1]

    location = self.newGraph[target].originPath

    path = [target]

    while location != target:
        path.append(location)
        location = self.newGraph[location].originPath

    fPointer = 0
    lPointer = len(path) - 1
    while fPointer <= lPointer:

        if path[fPointer] > len(self) -1: path[fPointer] -= len(self)

        if path[lPointer] > len(self) - 1: path[lPointer] -= len(self)
```

```
        path[fPointer], path[lPointer] = path[lPointer], path[fPointer]
        fPointer += 1
        lPointer -= 1
    retVal = (path, self.newGraph[target].time)
    return retVal
```

Time complexity: O(E Log V), occurs for each Edges, rise method of heap is needed to perform to ensure new item fits in the heap. However, when additional graph is created, number of vertex is doubled and number of edges is tripled atmost to a new graph which would lead to O(3E)*O(Log 2V) = O(3E)*O(Log V + log 2) which result O(E Log V).

Space complexity: O(V), occurs when V space is needed to store V vertices in path and additional 2V space is needed to store atmost all vertex in new graph into min heap which result O(v)

where V is the total number of points/ nodes/ vertices in the road network and E is the total number of roads/ edges in the road network.