



# [Data Structure]

[Final Report]

## Contents

---

Abstract Data Types.....	2
1. List:.....	3
Types of lists:.....	3
1. ArrayList:.....	3
2. LinkedList:.....	4
Advantages of Linked List.....	4
Disadvantages of Linked List:.....	4
Linked list operations:.....	5
The way it helped us with developing the code: .....	9
2. Stack:.....	11
Stack Operations:.....	11
3. Queue:.....	11
Queue Operations:.....	12
Graph.....	13
Flow Chart.....	14
Shortest path pseudocode and algorithm.....	15
Classes.....	17
• Inheritance: .....	18
Advantages of inheritance: .....	18
Polymorphism: .....	18
Types of polymorphism: .....	18
Advantages of polymorphism: .....	19
Encapsulation: .....	19
Advantages of Encapsulation:.....	19
Advantages of classes .....	19
Code Solution and variables:.....	20
GraphEdge class: .....	21
GraphNode class: .....	22
ShortestPath class: .....	22

Input validation and error handling 1:.....	25
Input validation and error handling 2:.....	27
GraphMain class .....	30
Solution Complexity: .....	32
Data Structures advantages .....	33
Presentation .....	34
References: .....	39

## Abstract Data Types

---

Abstract data is data that their behavior is usually known and configured by certain operations and values as well as that they are usually known as a mathematical model. In addition to that, they are well known for being censored from the user as the user can't notice the operations happening behind the scenes.

Moreover, the reason why programmers use abstract data types is that they help us with keeping the memory size into consideration and more efficient as usually it denies duplicates as well as that the speed as having ADT's uses the best way to deal with the data stored in it.

## 1. List:

---

Lists in general is a place where the programmer can hold certain data in a certain order. In addition to that normally it stores data in certain order as most of the times it follows the order the data had been inserted with which will make it easier to the programmer to access it as well as to insert more elements.

### Types of lists:

---

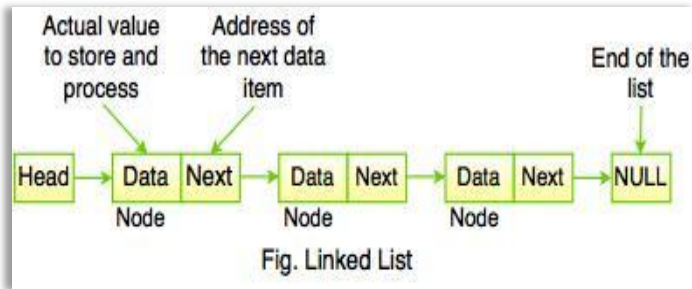
#### 1. ArrayList:

Array lists allows to increase the size of the list as whenever the program wants to add to the list it adds to the list memory. However, the size can be set, and no addition is possible. Moreover, it offers several operations that can help which we will discuss in the next paragraph.

Array List operations:

- Get () – Return the data chosen through position.
- Insert () – Inserts the data to a certain position.
- Remove () – Removes the first element seen from the list, however there should be data involved in the list.
- removeAt () – Removes data from a certain chosen position.
- Replace () – Replaces data chosen from any position in the list with another.
- Size () – Gives you the number of elements in the list.
- isEmpty () – Return true if the list is empty, otherwise return false.
- isFull () – Return true if the list is full, otherwise return false.

## 2. LinkedList:



Linked list is considered as data structure as the data in it or as we can say the elements isn't stored besides each other as each element is addressed differently as well as that each one has his own object. Moreover, every single element is usually called as a node where in this data structure and in this type of lists its way simpler to insert and remove elements. In addition to that, regarding what we said about that the elements are not beside each other the thing is that the elements in the linked list are connected to each other by pointers and addresses.

However, there are some disadvantages regarding linked lists that mostly the nodes can't be reached unless we go through every node that there is before the requested one.

In addition to that, and as we said that the elements are known as nodes they point at the neighbor node (next node) with a pointer, in the end they will make a something like a chain, therefore its best known to be used in graphs which is in our case we want to implement a graph.

### Advantages of Linked List

---

- Allocated memory.
- Easy to work with as well as flexibility of usage.
- Low access time.
- Easy to insert and to delete elements.

### Disadvantages of Linked List:

---

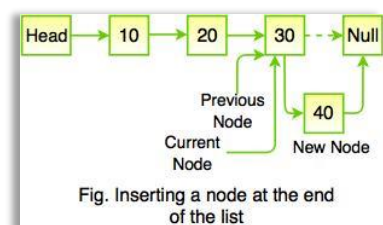
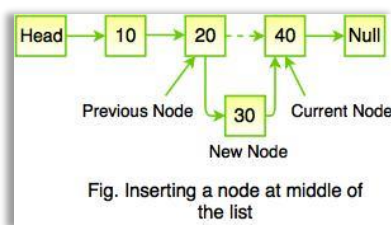
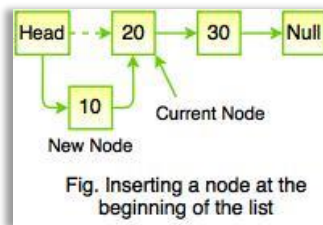
- Hard to reverse traversing.

- Must access every node in an ordered way to reach a certain node.
- Big memory size as the pointers takes some space.

### Linked list operations:

---

- Create: in this operation it creates a head node and the memory space should be one.
- Insert: inserts an element (node) to the linked list. Moreover, you can add insert the node wherever you want and that what makes the linked list special



- Delete: you can delete nodes from the beginning or middle as well as the end of a list
- Traverse: Checks all the nodes throughout the list as well as that its known that recursive functions to check the list from the opposite side.
- Search: to find a certain node and most of the time the sequential search is used with linked lists
- Concatenation: its known to be used in order to make two lists belongs to the same series
- Display: to provide the info about the node and to print it.

There are a lot of methods and operations in the linked list but its safe to say that they cover nearly everything that the list does as well as many more that's why most programmers consider linked list is much better to use than arrays. However to be more concise it depends on the scenario that's there and regarding our scenario I

have chosen to use linked list as well as regular lists because it's the right thing to use we have many methods that can help through designing the traffic solution as well as that its good to have the pointer part of the linked list where we will discuss more how it helped me in the solution plan.

Moreover, regarding the method as there are multiple methods, I'm going to discuss the methods that I have used.

Creating the Linked list:

```
public class GraphNode {  
  
    int p;  
    String key;  
    private boolean visited;  
    LinkedList<GraphEdge> verges;  
  
    GraphNode(int p, String key) {  
        this.p = p;  
        this.key = key;  
        visited = false;  
        verges = new LinkedList<>();  
    }  
}
```

In this code we initially create the list as well as that we assign the list in the variable verges

```
private void addSecondEdge(GraphNode a, GraphNode b, double distance) {  
  
    for (GraphEdge verge : a.verges) {  
        if (verge.startpoint == a && verge.endpoint == b) {  
            verge.distance = distance;  
            return;  
        }  
    }  
  
    a.verges.add(new GraphEdge(a, b, distance));  
}
```

here we used the linked list in our program to store the data of the nodes as the start point and the end point as well as the distance and regarding the for loop it checks weather the verge has already been added or not as well as if it's there then change the distance to the new one.

```
public boolean valid(GraphNode source, GraphNode endpoint) {
    LinkedList<GraphEdge> verges = source.verges;
    for (GraphEdge verge : verges) {
        if (verge.endpoint == endpoint) {
            return true;
        }
    }
    return false;
}
```

In a nutshell without the linked list there is no verges and if there are no verges their won't be distance and without the distance there is no shortest path.

In addition to that using linked list as we saw helped us to check multiple things way easier than if the data weren't ordered such as checking if there is verges as we see in the photo above the paragraph as well as checking and updating similar data we saw on the second image.

```
for (GraphEdge verge : currentPoint.verges) {
    if (verge.endpoint.Visited())
        continue;

    if (shortestPathGraph.get(currentPoint)
        + verge.distance
        < shortestPathGraph.get(verge.endpoint)) {
        shortestPathGraph.put(verge.endpoint,
            shortestPathGraph.get(currentPoint) + verge.distance);
        adjustTo.put(verge.endpoint, currentPoint);
    }
}
}
```

Moreover, having an ordered linked list helped us to use a loop in order to go by the unvisited nodes (the different routes) and by that we check which is the shortest path and we choose the shortest.



## HashSet:

---

The second data structure used is the hash set which can be considered as a data structure in some ways as in our case it uses Hash-table which in other word is called the Hash-map and the hash-map is considered as a data structure. In addition to that, the set interface isn't accurate when it comes to ordering elements as the compile time goes on. However, HashSet doesn't allow to have same elements as it denies duplicates. Moreover, performance wise it is un comparable as the time is constant regarding add, contains, size and remove, it uses hash code in order to manipulate objects as well as all that it doesn't allow null values which is really helpful in our case as we don't want a node without edges which will make it unreachable as well as that a node without a name which will cause us many errors.

```
public class ShortestPath {  
    private Set<GraphNode> points;  
    private boolean direction;  
  
    ShortestPath(boolean direction) {  
        this.direction = direction;  
        points = new HashSet<>();  
    }  
  
    public void addpoint(GraphNode... p) {  
        points.addAll(Arrays.asList(p));  
    }  
}
```

In this piece of code, we declare the HashSet function and we assign it to the variable nodes(points) as well as that we used the Arrays.asList() method which is where I used a data structure as arrays are considered as a data structure and the method that we used is to solve

the only problem of the HashSet as we said that the elements within the set won't be in the order they were inserted in, however the Arrays.asList method makes it an ordered list.

### Hash-map:

---

This Data structure can be used as the name says to link keys with values which is extremely important in our case as we want to keep the track of the nodes and the endpoint. Moreover, the data implemented in the HashMap is stored in an array and as we said that in this array each value has its special key which will make it more reachable. Therefore, the HashMap is considered a data structure. Moreover, HashMap doesn't have many null keys as it is allowed to have just one null key where it is allowed to have multiple null values as well as that it is not ordered and the space of a HashMap is sixteen.

### The way it helped us with developing the code:

---

```
public void DeclareShortestPath(GraphNode startnode, GraphNode endnode) {
    HashMap<GraphNode, GraphNode> adjustTo = new HashMap<>();
    adjustTo.put(startnode, null);

    HashMap<GraphNode, Double> shortestPathGraph = new HashMap<>();

    for (GraphNode point : points) {
        if (point == startnode)
            shortestPathGraph.put(startnode, 0.0);
        else shortestPathGraph.put(point, Double.POSITIVE_INFINITY);
    }

    for (GraphEdge verge : startnode.verges) {
        shortestPathGraph.put(verge.endpoint, verge.distance);
        adjustTo.put(verge.endpoint, startnode);
    }
}
```

Here we used the HashMap in order to follow the path that can give us the shortest distance to the endpoint and we can reach this result as we save how the program reached every point as there would be a pointer that would follow the parent point of each child point.

In addition to that, the `shortestPathMap = new HashMap` is used to contain the shortest path for each point as well as that the first for loop is used in order to set each distance of every point to the value of infinity except the starting node as the distance from the starting node to the starting point should be zero. Moreover, the second for loop is used to makes us visit every possible node from the start point.

Which means without the HashMap we can't find the best shortest path and without it we cant keep the track of the paths to each node which means the whole shortest path won't function well as everything we mentioned in the past two paragraphs won't function as well as several more things which I will mention next.

```
private GraphNode secondroute(HashMap<GraphNode, Double> shortestPathGraph) {  
  
    double shortestDistance = Double.POSITIVE_INFINITY;  
    GraphNode nearestpoint = null;  
    for (GraphNode point : points) {  
        if (point.Visited())  
            continue;  
  
        double currentDistance = shortestPathGraph.get(point);  
        if (currentDistance == Double.POSITIVE_INFINITY)  
            continue;  
  
        if (currentDistance < shortestDistance) {  
            shortestDistance = currentDistance;  
            nearestpoint = point;  
        }  
    }  
    return nearestpoint;  
}
```

Here we used the HashMap in order to check all different routes as well as to check every possible node as here we check the unvisited nodes and after checking them, we will compare them with the first path we found and choose the shortest.

## 2. Stack:

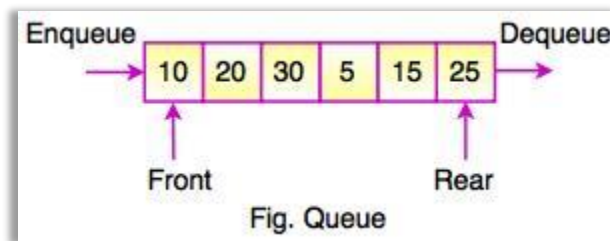
This abstract data type is a data structure that is known as a linear data structure where it usually follows a FIFO pattern where it means (first in first out) where the first element entered is the first one to be removed as well as the LIFO (last in first out) where the last element entered is the first element to be removed

Stack Operations:

- `isFull ()`, checks if the stack is full of elements
- `isEmpty ()`, checks if the stack has no elements
- `push (x)`, its used to insert an element to a stack
- `pop ()`, its used to delete an element from the head of the stack
- `peek ()`, its used to get the head element of the stack
- `size ()`, this provides the programmer with the number of elements that are in the stack.

## 3. Queue:

This ADT is a data structure where it has two points the first point is called *Rear* where it's the pointer where elements are inserted from and it is located at the end of the queue as well as that there is the *Front* and its where elements are deleted which is located at the beginning of the queue.



The way it works that at the start of initializing the queue both the Rear and the Front will be at the first index, while the Rear keeps on moving forward, the Front stays at the first index.

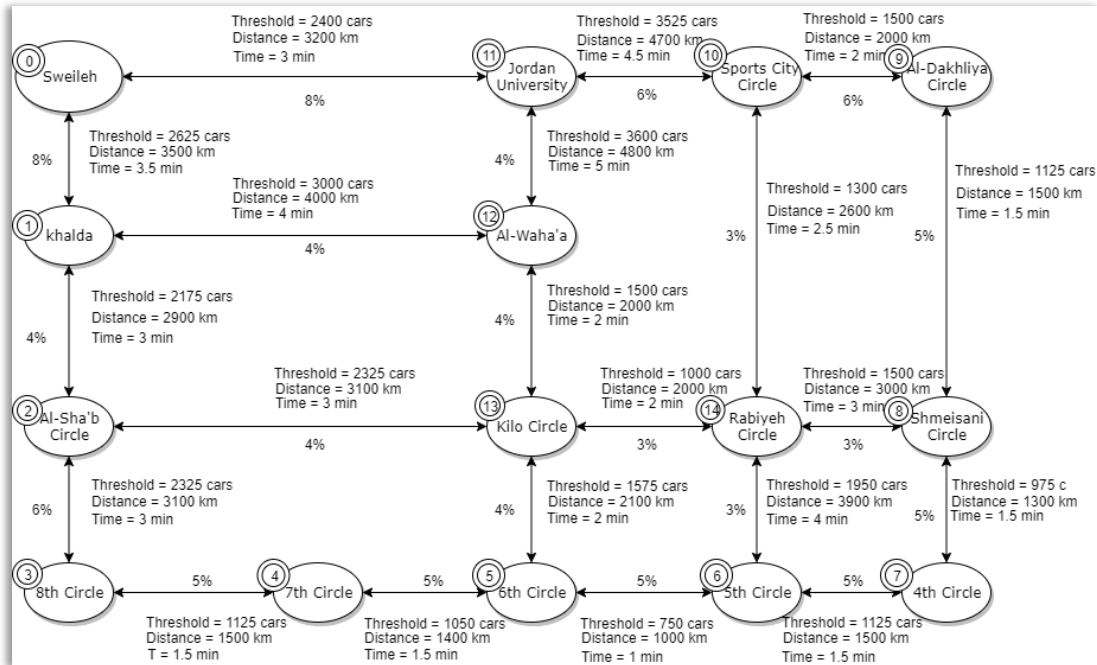
However, regarding in using FIFO method its mostly implemented by the Queue ADT as the queue inserts elements (enqueue) through the Rear which means that the first element inserted by the Rear will be pointed at by the Front (dequeue)

which deletes elements that's why and so goes on, which shows why the FIFO method is best implemented by the Queue.

#### Queue Operations:

- enqueue(): Inserts elements to the queue.
- dequeue() Removes elements from the queue
- init(): initialize the queue
- Front: gets the data that are stored in the front pointer.
- Rear: Gets the item stored in the rear pointer.
- isFull(), Checks if the queue is full of elements
- isEmpty(), Checks if the queue has no elements
- insert(x), This add an element x in the end of the queue
- delete(), delete the element in the end of the queue
- size(), Gives the programmer the number of elements in the queue.

## Graph

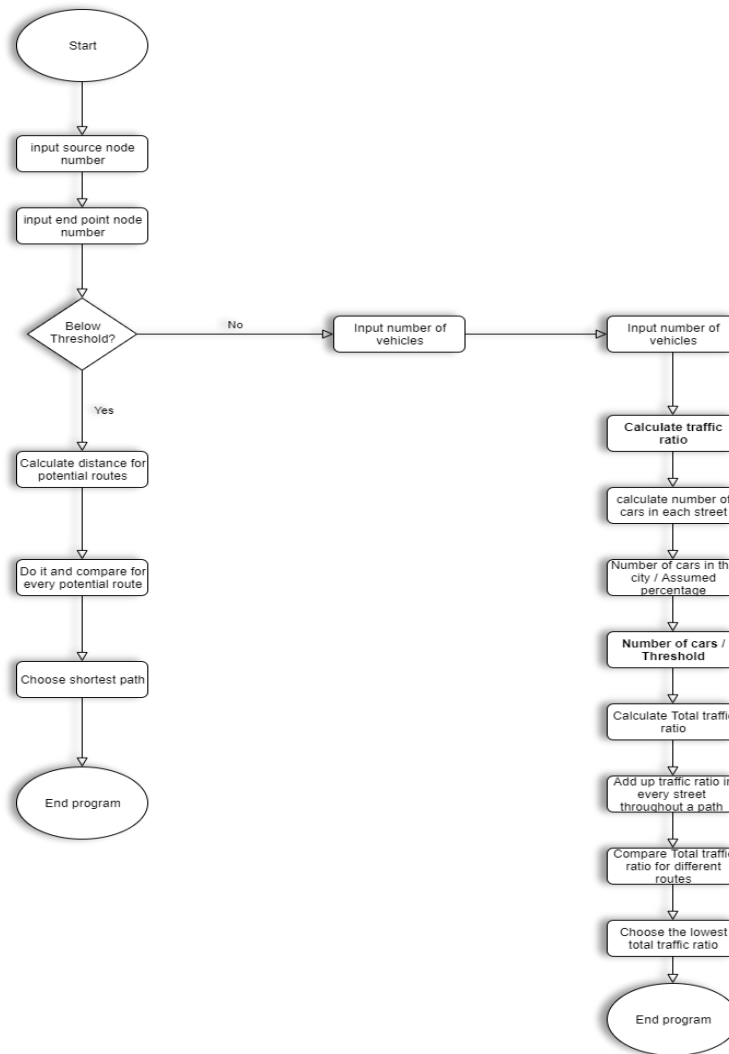


In this graph we see five types of information that we are going to use:

- **Nodes:** First of all, we have the nodes where it contains the names of the areas that we want to find the shortest path between and there are 15 nodes if you count node zero. Moreover, we gave each node a key which is in our case as shown are the numbers from zero to fourteen to make it easier to work with in the program.
- **Threshold:** It's a number that shows the maximum number of cars that a certain street can hold and if passed there will be traffic. However, it's a calculated number as it represents the following equation (distance/ 4" car size + safety") x (number lanes that the street has).
- **Distance:** I have calculated the distance throughout certain apps as well as expert's data
- **Percentage:** this one is an assumption of the number of cars that would be in each street if the number of cars in city is known based on experts' assumptions.

- Time: This indicates the time that a car would take in order to go by a street if under the threshold and there is no traffic. In addition to that its taken from certain apps as well as expert's judge.

## Flow Chart



In this flowchart and the reason that I have made a flowchart that it will help with both developing the pseudocode as well as the source code as the flowchart describes in a way how the program will work and what operation will the program go through in an ordered way. However, deep explanation will be shown on the pseudocode and algorithm part

In addition to that, I'm going to explain the solution which in which we want to find a way to solve the traffic in Amman and in order to find that had to put in my mind

three study cases which are that if the streets are all below threshold and if the street are all above threshold as well as if some are above and some other streets are below we will decide on another solution.

In our first case we had the case that all streets are under the threshold where I have decided to solve this problem by calculating the distance between the points and after that there will be several routes then we want to compare them with each other and choose the shortest distance between the points and then print it. Moreover, regarding the other two scenario I have found and made up an equation that solves both of these scenarios which is that I want to calculate something called traffic ratio which is calculated like this

Number of cars = number of vehicles/ percentage

Then: Number of cars/ Threshold

“As the number of vehicles in the city is an input from the user”

“Which will show us the number of cars in that street then we divide it into the threshold to check if it is above or not as if the last percentage is above one that means it is above threshold which eventually would mean to void the street if the final add up is high.”

Traffic ratio = Number of cars / Threshold

Which means we would have a percentage that shows how much the assumed cars had affected the certain street and the program would do that for each and every street throughout the path then he will add up all the traffic ratio for every street throughout the path and the one that have the smallest percentage is the chosen path.

## Shortest path pseudocode and algorithm

---

This is a simplified pseudocode which will briefly explain our solution and how our code will run.

If all Below threshold

This algorithm will explain the way the to find the shortest path. So, I will be discussing the way this code will work but not in a programming perspective and I will simplify it as much as possible as that what algorithms are for as well as to guide me to programming the whole thing.

- There is the Graph it contains group of vertices (the points) as well as a group of verges



- Input the source point
- Input end point
- Create variable time
- There is the source point and as it's the source point the distance from the source to the source which is the same point is set to zero.
- The distance from the source to any other point in the graph is set to infinite. Therefore, the program will run as if its impossible to reach any node from the source point.
- The code will put all the nodes as unvisited points except the source will be assigned as visited.
- If any point is unvisited then:
  - The program detects the node that has the shortest distance from the source point.
  - Set the detected node as visited.
  - For every edge between point and the unvisited point :  
$$\text{If } \text{shortestpath}(\text{source}, \text{point}) + \text{shortestpath}(\text{point}, \text{unvisited point}) < \text{shortest path}(\text{source}, \text{unvisited point})$$
  
Then change the shortest path between s and m to  
$$\text{shortestpath}(\text{source}, \text{point}) + \text{shortestpath}(\text{point}, \text{unvisited point})$$

If some are below threshold or If all above threshold:

- Input the source point
- Input end point point
- Input number of vehicles
- Create variable threshold for each street(verge)
- Create variable assumed percentage for each street(verge)
- Calculate traffic ratio which is a variable that stores a number that the less the better and the less traffic we have for every single street:  
$$\text{Number of cars} = \text{number of vehicles} / \text{percentage}$$
  
Then: 
$$\text{Number of cars} / \text{Threshold}$$
  
$$\text{Traffic ratio} = \text{Number of cars} / \text{Threshold}$$
- Calculate Total traffic ratio  
For every edge between the source point and the end point  
$$\text{Total traffic ratio route 1} = \text{Add traffic ratio of source point} + \text{the traffic ratio next point} + \dots + \text{traffic ratio for end point.}$$

- Different route between the source and the end point  
Total traffic ratio path 2 = Add traffic ratio of source point  
+ the traffic ratio next point + ..... +traffic ratio for end point.

If Total traffic ratio path2 < Total traffic ratio path1

Then print Total traffic ratio path2

Else:

Print total traffic ratio path1

- Print distance of path 2
- End program

## Classes

---

Classes: it defines the properties of the object or more that are associated with it. It can also be used in order to define a multiple object in a program.

Some programming languages that uses OOP: Python, Java, C++, C#, PHP, JavaScript, Ruby, Objective c, Object Pascal, Scala and Swift.

In addition to that, we have used classes in our project as we cant accomplish any of the object oriented features without having classes so we can say that without classes we can't attain the following features and I'm going to include a brief description of each feature.

- Inheritance: it is a strategy that allows us to eliminate redundant code. As the following feature enables the programmer to inherit the properties and the methods of existing classes.

### Advantages of inheritance:

---

- Makes things reusable as most of the methods and data will be inherited from a class to another
- Makes the code more readable
- Less cost regarding the maintenance and less time with the development
- Less code redundancy

### Polymorphism:

---

or in other words (many forms), which means that it processes the objects depending on the operation data type which represents the origin meaning of polymorphism which means that it is capable to take many forms.

### Types of polymorphism:

---

- Run time
  - Example Method overriding
- Compile time
  - Example Method overloading

## Advantages of polymorphism:

---

- Methods overloading makes it able to make methods apply the same functions by following a certain name as a programmer allows you to define methods with the exact same name, however it should be a different parameter.
- Overloading can work with constructors which will able the programmer to initialize objects of classes.
- Overriding makes it able that a sub class to use the definitions formed in a parent class.
- Overriding allows to function with inheritance so we could make the code reusable.

## Encapsulation:

---

it is right that you can have a program that is divided into a set of functions, so we have data that is stored in a group of variables and a function that operate on the data. This way is very simple and straight forward, although as your program grow you will end up with a lot of functions, if you make a change to one function then several others might change too. We can fix this in object-oriented programming by combining a group of related variables and functions into a unit we call this unit an object.

we refer to these variables as properties and the functions as methods.

## Advantages of Encapsulation:

---

Allows to hide important data and there is no main disadvantage as you can use if you want to secure your data.

However, getting back to classes all these features as we said is impossible to attain without classes as well as that there is no object oriented without classes. However, I'm going to list some of the direct advantages of classes as well

## Advantages of classes

---

- Using classes, it would be easier to understand the code and would make it easier to manage and to keep maintaining the code
- Saves us time, as the development process would be faster as classes provides us with parallel way of development
- Having classes would allow us to reuse the code multiple times which will save us both time and effort

## Code Solution and variables:

---

I have decided to work with java as I find java programming language as the simplest language to read so, it is kind of better to have the code readable for the stakeholders. Moreover, as is how present the code I'm going to explain some complicated parts of the code.

## GraphEdge class:

---

```
public class GraphEdge implements Comparable<GraphEdge> {  
  
    GraphNode startpoint;  
    GraphNode endpoint;  
    double distance;  
  
    GraphEdge(GraphNode s, GraphNode e, double d) {  
        startpoint = s;  
        endpoint = e;  
        distance = d;  
    }  
  
    public int compareTo(GraphEdge otherE) {  
  
        if (this.distance > otherE.distance) {  
            return 1;  
        }  
        else return -1;  
    }  
}
```

I have made the variables Startpoint, endpoint and distance as public variable as these variable will be accessed and operations like adding and removing will be done on them so they better be public as they help with building the graph too.

In this class I have created three variables which are the start point and the end point as well as the distance between each and other point as with the GraphEdge method we represent the graph in the code as this three variables declares the points and the connected points as well. In addition to that we compare the distance between the edges as if there are duplicates as the points will be same, but the distance would be difference if yes it will allow it if no it will deny the process.

## GraphNode class:

---

```
import java.util.LinkedList;

public class GraphNode {

    int p;
    String key;
    private boolean visited;
    LinkedList<GraphEdge> verges;

    GraphNode(int p, String key) {
        this.p = p;
        this.key = key;
        visited = false;
        verges = new LinkedList<>();
    }

    boolean Visited() {
        return visited;
    }

    void visit() {
        visited = true;
    }

    void devisit() {
        visited = false;
    }
}
```

The variables p and key are assigned as public as these two variables will be used in nearly every class. In addition to that, I want the main class to access as well as change the data that P and key holds as I want to create points and to assign key into them. However, the boolean variable visited is private because it is a sensitive variable as I want other classes to access info as it is important in order to decide the where the point will travel next. However, if the each and every class change it would be a mess and the code will fail to function.

As well as that the verges variable that holds the linked list is public as it is obvious that we want to add data into the list and to remove data as well so it better be public and every linked list throughout the code has the same explanation.

In this class we create the variables as well as the linked list data structure where the first two variable are used to create a point which is the p variable as well as that the key variable is to define and call the points whenever we want to add, remove and adjust. Moreover, this class is used to mark points as visited as well as to mark them as not visited and as we agreed in the algorithm that we want to mark the whole points except the source one to we could mark what paths have we visited. In addition to that, we have assigned the linked list into the variable verges. However, this whole class won't function without the following class.

## ShortestPath class:

---

```
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Set;

public class ShortestPath {
    private Set<GraphNode> points;
    private boolean direction;

    ShortestPath(boolean direction) {
        this.direction = direction;
        points = new HashSet<>();
    }

    public void addpoint(GraphNode... p) {
        points.addAll(Arrays.asList(p));
    }
}
```

The variables `points` and `direction` are private as the `points` variable is held in a set and this set shouldn't be manipulated from other classes but can be accessed in order to do some operations. In addition to that, the Boolean `direction` is a variable that is used as a decider to do certain operation or not therefore it shouldn't be public.

In this piece of code, first of all, we import the `java.util`'s as if we want to use arrays, `HashMap`, `HashSet`, and linked list furthermore in the code we have to import them. In addition to that, we declare the `HashSet` function in order to set the points that we inserted from the graph inside this set and I used `HashSet` in order to deny duplicate data as in our case it is not recommendable which we assign it to the variable `nodes(points)` as well as that we used the `Arrays.asList()` method to solve the only problem of the `HashSet` as we said that the elements within the set won't be in the order they were inserted in, however the `Arrays.asList` method makes it an ordered list.



```
public void addVerge(GraphNode startpoint, GraphNode endpoint, double distance) {  
    points.add(startpoint);  
    points.add(endpoint);  
  
    addSecondVerge(startpoint, endpoint, distance);  
  
    if (!direction && startpoint != endpoint) {  
        addSecondVerge(endpoint, startpoint, distance);  
    }  
}
```

In this second part, we create a method in order to add points from the graph as the add verge method will add the points if this point isn't added yet as the set method denies duplicates. In addition to that, we called the addSecondVerge method which we will discuss in the next part which basically helps us with avoiding duplicates.

```
private void addSecondVerge(GraphNode a, GraphNode b, double distance) {  
  
    for (GraphEdge verge : a.verges) {  
        if (verge.startpoint == a && verge.endpoint == b) {  
            verge.distance = distance;  
            return;  
        }  
    }  
  
    a.verges.add(new GraphEdge(a, b, distance));  
}
```

This method allows us to pass by all the verges we have which will give us the indicator whether there is duplicates or not verges wise as well as that the for loop checks the verges and if there is a duplicate with a difference distance it updates the new distance.

And if the input value isn't a duplicate the program adds it as a new verge.

## Input validation and error handling 1:

---

```
public void printVerges() {  
    for (GraphNode point : points) {  
        LinkedList<GraphEdge> verges = point.verges;  
  
        if (verges.isEmpty()) {  
            System.out.println("The Point " + point.key + " has no verges.");  
            continue;  
        }  
    }  
}
```

Regarding input validation and as we reached the piece of code that handles input validation and error handling as this piece of code sends an error by using the isEmpty() method that we were allowed to implement it because of the data structure linked list we have used as it checks if there is no verges implemented and if the list empty we send the user a message that the point entered has no verges, which means no operations can be applied on it. In addition to that, regarding the input validation I will discuss more about as soon as we reach the next part of the code that address this header.

```
public boolean valid(GraphNode source, GraphNode endpoint) {  
    LinkedList<GraphEdge> verges = source.verges;  
    for (GraphEdge verge : verges) {  
        if (verge.endpoint == endpoint) {  
            return true;  
        }  
    }  
    return false;  
}  
  
public void setNodesAsVisited() {  
    for (GraphNode point : points) {  
        point.devisit();  
    }  
}
```

The first method checks whether there is a verge that connect the points or not as well as that the second method is the process to set all the points as unvisited as the devist method apply this operation and this method is used in our code in order to be able to run the code for multiple time as each time would need to apply this method.

```
public void DeclareShortestPath(GraphNode startnode, GraphNode endnode) {
    HashMap<GraphNode, GraphNode> adjustTo = new HashMap<>();
    adjustTo.put(startnode, null);

    HashMap<GraphNode, Double> shortestPathGraph = new HashMap<>();

    for (GraphNode point : points) {
        if (point == startnode)
            shortestPathGraph.put(startnode, 0.0);
        else shortestPathGraph.put(point, Double.POSITIVE_INFINITY);
    }

    for (GraphEdge verge : startnode.verges) {
        shortestPathGraph.put(verge.endpoint, verge.distance);
        adjustTo.put(verge.endpoint, startnode);
    }

    startnode.visit();
}
```

This method could be our most important and the longest one I will divide it to parts to make it easier to understand. So, the first part is the part we declare our start and end points as well as that it is where we want to follow the path that provides us with the shortest distance and we will achieve that by following the way that we reached for each and every point as we keep a pointer to the point that we came from until we reach the end point and to do that we implemented the HasMap.

In addition to that we implemented the for loop in order to set each and every point distance to infinity except the start point as the distance from the source to the source should be zero. Moreover, the second for loop is to pass by all the points that is reachable from the start point and to keep track of the pointer we assign them to the adjust to method.

## Input validation and error handling 2:

---

```
while (true) {
    GraphNode currentPoint = secondroute(shortestPathGraph);
    if (currentPoint == null) {
        System.out.println("There isn't a path between " + startnode.key + " and " + endnode.key);
        return;
    }
    if (currentPoint == endnode) {
        System.out.println("The path with the smallest weight between "
            + startnode.key + " and " + endnode.key + " is:");

        GraphNode sub = endnode;

        String path = endnode.key;
```

This part of code is the second error handling situation as we see I created this loop which will keep on functioning as long as there is a point that is not marked as unvisited within our range. However, the first if statement is occurred whenever we didn't reach the end point and there are no points between them which means that the end point isn't reachable then we would send an error message which will notify the user that there are no path between the start point and the end point and with that we have completed the error handling and input validation report as there are no other possible ways to do an error and these two messages covers the errors that could occur throughout the journey of finding the shortest path.

Moreover, and going back to the code explanations the second if statement is to when the not visited point is the end point and if that is the case, we would want to print the found path.

```
if (currentPoint == endnode) {
    System.out.println("The path with the smallest weight between "
        + startnode.key + " and " + endnode.key + " is:");

    GraphNode sub = endnode;

    String path = endnode.key;
    while (true) {
        GraphNode head = adjustTo.get(sub);
        if (head == null) {
            break;
        }

        path = head.key + " " + path;
        sub = head;
    }
    System.out.println(path);
    System.out.println("The route distance: " + shortestPathGraph.get(endnode)+ "km" );
    return;
}
```

In this part of code we check if we reached the end point that we want to reach and if we reached it we will print a message that the shortest between the point is, and in order to find the path first we will assign end point into the variable sub and the key of the points into the variable path as well as that we enter the while loop and in this loop if the condition is true we will assign the adjustTo method that we have discussed previously which keeps track with the parent and child points which we will use to print the path as we will assign the adjustTo method

to the variable head and if the head variable is empty that means we should exit the if statement, however the while loop is to assign the path that we followed through the adjustTo and to assign it to the variable path where it holds the parent and the child which will print the path of the point from the start point to the end point as well as that we print the distance by calling it from the method shortestPathGraph which I will discuss later on.

```
currentPoint.visit();|  
  
for (GraphEdge verge : currentPoint.verges) {  
    if (verge.endpoint.Visited())  
        continue;  
  
    if (shortestPathGraph.get(currentPoint)  
        + verge.distance  
        < shortestPathGraph.get(verge.endpoint)) {  
        shortestPathGraph.put(verge.endpoint,  
                                shortestPathGraph.get(currentPoint) + verge.distance);  
        adjustTo.put(verge.endpoint, currentPoint);  
    }  
}  
}
```

In this piece of code I wanted to check the route with the points who we haven't visited yet and to do I created a for loop which will pass by the point where we know that the point we are in has verges to in order to compare if the shortest distance we will find is less than the distance we would found before.

```
private GraphNode secondroute(HashMap<GraphNode, Double> shortestPathGraph) {  
  
    double shortestDistance = Double.POSITIVE_INFINITY;  
    GraphNode nearestpoint = null;  
    for (GraphNode point : points) {  
        if (point.Visited())  
            continue;  
  
        double currentDistance = shortestPathGraph.get(point);  
        if (currentDistance == Double.POSITIVE_INFINITY)  
            continue;  
  
        if (currentDistance < shortestDistance) {  
            shortestDistance = currentDistance;  
            nearestpoint = point;  
        }  
    }  
    return nearestpoint;  
}
```

This method is declared so we could find the second route which is the route with the unvisited points and to compare the paths of the unvisited points as we see we will check if the point we are currently in is registered as visited which will make us continue to the next step which will make us compare whether the distance we had is better than the one from the unvisited path points as we see the third if statement that compares the new value with the old one and in the end returns the shortest.

## GraphMain class

---

```
public class GraphMain {  
    public static void main(String[] args) {  
        ShortestPath s = new ShortestPath(true);  
        GraphNode zero = new GraphNode(0, "0");  
        GraphNode one = new GraphNode(1, "1");  
        GraphNode two = new GraphNode(2, "2");  
        GraphNode three = new GraphNode(3, "3");  
        GraphNode four = new GraphNode(4, "4");  
        GraphNode five = new GraphNode(5, "5");  
        GraphNode six = new GraphNode(6, "6");  
        GraphNode seven = new GraphNode(7, "7");  
        GraphNode eight = new GraphNode(8, "8");  
        GraphNode nine = new GraphNode(9, "9");  
        GraphNode ten = new GraphNode(10, "10");  
        GraphNode eleven = new GraphNode(11, "11");  
        GraphNode twelve = new GraphNode(12, "12");  
        GraphNode thirteen = new GraphNode(13, "13");  
        GraphNode fourteen = new GraphNode(14, "14");  
    }  
}
```

This is the main class that you could say makes everything more clear to understand the code as you will know where the data enters and what the flow of the data would be. So first of all, the GraphNode method that is created in the GraphNode class is used to declare the point which means there is no points without this method and we wouldn't have been able to reach for any point and the key strings are there because we want to easily reach those points.

```
s.addVerge(zero, one, 3500);
s.addVerge(zero, eleven, 3200);
s.addVerge(one, two, 2900);
s.addVerge(one, twelve, 4000);
s.addVerge(two, three, 3100);
s.addVerge(two, thirteen, 3100);
s.addVerge(three, four, 1500);
s.addVerge(four, five, 1400);
s.addVerge(five, six, 1000);
s.addVerge(five, thirteen, 2100);
s.addVerge(six, seven, 1500);
s.addVerge(six, fourteen, 3900);
s.addVerge(seven, eight, 1300);
s.addVerge(eight, nine, 1500);
s.addVerge(eight, fourteen, 3000);
s.addVerge(nine, ten, 2000);
s.addVerge(ten, eleven, 4700);
s.addVerge(ten, fourteen, 2600);
s.addVerge(eleven, twelve, 4800);
s.addVerge(twelve, thirteen, 2000);
s.addVerge(thirteen, fourteen, 2000);
//another route
s.addVerge(eleven, ten, 4700);
s.addVerge(eleven, zero, 3200);
s.addVerge(ten, nine, 2000);
s.addVerge(nine, eight, 1500);
s.addVerge(eight, seven, 1300);
```

```
// another route
s.addVerge(twelve, eleven, 4800);
s.addVerge(thirteen, twelve, 2000);
s.addVerge(thirteen, five, 2100);
s.addVerge(twelve, one, 4000);
s.addVerge(thirteen, two, 3100);
s.addVerge(fourteen, thirteen, 2000);
s.addVerge(fourteen, six, 3900);
s.addVerge(fourteen, eight, 3000);
s.addVerge(fourteen, ten, 2600);

s.DeclareShortestPath(zero, seven);
}
```

This part is where we have the addVerge method in function as we can define the point and the point that is connected from it as well as the distance between those two points. However, regarding this part of code I would like to point out an error that I faced through the development as I found out that I should have connect the point from both ways as in the normal way and the backwards way in order to have the full on connections between every point.

In the end, the Declare shortest path is the line where we choose the start and endpoint that we want to find the shortest distance for.



## Solution Complexity:

---

When you think about it regarding the shortest path algorithm that I have implemented you'll think that as the program go through points the shortest path for each point will be saved which could be a valid point, although it is not true as we have to re run the code each time we want to find a path. However, regarding the worst-case scenario, this scenario and with using my type of solution suggest that to find the path for the worst-case the program should pass by every single node which will lead us to the conclusion that in order to run it the space complexity would be the amount of points - 1 which is the source point.

Moreover, having a linked list to build this algorithm lead us to the conclusion that it has the complexity of  $O(\text{number of verges})$  depending on what we have said regarding going pass by all the points to reach the required one.

However, it is known that in this case, and I mean with finding the shortest path that it is better to use the Heap sort rather than the linked list as the heap sort works well with the priority queue and regarding how it works it works really with finding the next unvisited point which lead us to the  $O(1)$  time, but that's not it having used the queue there are multiple other equations that can be added to this equation which to know that in order to add and remove affect the time with  $O(\log(\text{number of verges}))$  which stays better than the linked list time as well as if we want to certain number of iterations and a certain number of deletions and adding what would be our final result

The result would be  $O(\text{number of points} + \text{number of verges}) * \log(\text{number of verges})$

Which would still be better than our case using linked list as our result is

$O((\text{number of verges} + \text{number of points}) * \text{number of verges})$

However, the space complexity of linked list Space -  $O(n)$

And the priority queue Space -  $O(n)$ .

In addition to that,

The time operations of the linked list:

- Index  $O(n)$
- Insert  $O(1)$
- Search  $O(n)$

- Deletion  $O(1)$

Regarding the queue:

- Insert  $O(\log n)$
- Pop  $O(\log n)$
- Peek  $O(1)$

However, the number may differ with a good amount, but I saw that in my case its better to use the linked list as the number of points and verges is low, so it won't affect that much. However, it is way easier to do it than to do both heap sort and priority queue, although I have used it with the implementation of the other scenario which is whether the streets are all above threshold and if some streets are above and some are below and I chose this sorting algorithm as I saw that it suits our idea the most as other sorting algorithm may have taken a lot of time and doesn't suit my idea as using bubble sort would be ridiculous as if I wanted the time complexity to be higher than ever as well as that the amount of operations would be higher as well and to apply the algorithm we have may be extremely harder than using the priority queue heap sort as its is easier to find a point with it and more flexible as the bubble sort may take a huge amount of time to just push one index out.

## Data Structures advantages

---


- Efficiency: regarding the memory as having a data structure will optimize the use of the memory as without data structures everything would have an unlimited size which will increase the space complexity which will lead to have simple programs to run for a very long time. However, data structures provided us with linked list which will delete any unused space while the array would have just kept it.
- Reusability: Having data structures enable the developer to reuse the code written as whenever I had a data structure built I could just use it in another part of code which will save us the time to keep entering data as well as the space to keep entering data
- Abstraction: As we saw through the report that data structure covers a whole part of abstract data type as abstract data type is known as thoughts that follows a certain logic that can only be applied throughout data structures as in other words it gives them a real form. In addition to that, without data structure there wouldn't be high level computations as all operations would be extremely limited and there would be no place to creativity.

## Presentation


---




# Introduction



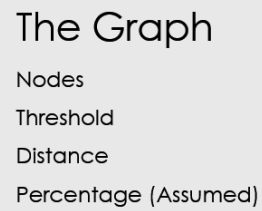
What is the problem?



What are the variables ?



What is the solution?





## Flowchart

Three scenarios with two solution

Below Threshold: Shortest path

Above and in between: Traffic ratio solution

## Why did we use classes



Advantages



Features

```
60
61
62     s.DeclareShortestPath(zero, seven);
63 }
64 }
65
66
67 |
```

<terminated> GraphMain [Java Application] C:\Program Files\Java\jdk-13.0.  
The path with the shortest distance between 0 and 7 is:  
0 11 10 9 8 7  
The route distance: 12700.0km

## Code outputs

- We decide the start point and the end point
- The program prints the path
- Prints the distance

## Complexity

- Time complexity
- Space complexity



## Advantages of data structure



REUSABLE



ABSTRACTION



MEMORY  
EFFICIENCY

## Questions?



Thank You



Done by Ali AbuAbboud



Date: 23/ 1/ 2020

## References:

---

In-text: (Anon, 2020)

Your Bibliography: Anon, (2020). [online] Available at: <https://www.enotes.com/homework-help/what-advantages-disadvantages-data-structure-479073> [Accessed 27 Jan. 2020].

In-text: (GeeksforGeeks, 2020)

Your Bibliography: GeeksforGeeks. (2020). *LinkedList in Java - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/linked-list-in-java/> [Accessed 27 Jan. 2020].

In-text: (GeeksforGeeks, 2020)

Your Bibliography: GeeksforGeeks. (2020). *ArrayList in Java - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/arraylist-in-java/> [Accessed 27 Jan. 2020].

In-text: (Wisdom Jobs, 2020)

Your Bibliography: Wisdom Jobs. (2020). *Abstract data type in Data Structures Tutorial 27 January 2020 - Learn Abstract data type in Data Structures Tutorial (7124) | Wisdom Jobs India*. [online] Available at: <https://www.wisdomjobs.com/e-university/data-structures-tutorial-290/abstract-data-type-7124.html> [Accessed 27 Jan. 2020].

In-text: (Cseworldonline.com, 2020)

Your Bibliography: Cseworldonline.com. (2020). *Introduction of Data Structures || CseWorld Online*. [online] Available at: <https://www.cseworldonline.com/data-structure/Introduction-Data-Structures.php> [Accessed 27 Jan. 2020].

In-text: (Tutorialspoint.com, 2020)

Your Bibliography: Tutorialspoint.com. (2020). *Data Structure and Algorithms Tutorial - Tutorialspoint*. [online] Available at: [https://www.tutorialspoint.com/data\\_structures\\_algorithms/index.htm](https://www.tutorialspoint.com/data_structures_algorithms/index.htm) [Accessed 27 Jan. 2020].

In-text: (GeeksforGeeks, 2020)



**Your Bibliography:** GeeksforGeeks. (2020). *FIFO vs LIFO approach in Programming - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/fifo-vs-lifo-approach-in-programming/> [Accessed 27 Jan. 2020].

**In-text:** (Tutorialspoint.com, 2020)

**Your Bibliography:** Tutorialspoint.com. (2020). *Abstract Data Type in Data Structures*. [online] Available at: <https://www.tutorialspoint.com/abstract-data-type-in-data-structures> [Accessed 27 Jan. 2020].

**In-text:** (Anon, 2020)

**Your Bibliography:** Anon, (2020). [online] Available at: [https://computersciencewiki.org/index.php/Abstract\\_data\\_structures](https://computersciencewiki.org/index.php/Abstract_data_structures) [Accessed 27 Jan. 2020].