ACKNOWLEDGEMENT

First of all, I would like to send my sincere thanks to Dr. Duong Le Minh for his great guidance, guidance and motivation in the course of this graduation thesis.

I would like to express my gratitude to all my staffs in GEM JSC for always helping me, contributing my ideas to help me complete the topic in the best way.

I would like to thank the teachers in Faculty of Information Technology - University of Engineering and Technology who devoted to teaching and imparting to me valuable knowledge in the last four years, help me have a background to implement this thesis.

In the last word, I would like to thank my family, relatives and friends who always care for me, always with me in the most difficult times, in study and life.

I sincerely thank!

Student

Nguyen Van Quan

AUTHORSHIP

"I hereby declare that the work contained in this thesis is of my own and has not been previously submitted for a degree or diploma at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no materials previously published or written by another person except where due reference or acknowledgement is made."

Signature:………………………………………………

ABSTRACT

Over the past few years, mobile devices, especially smartphones have been growing rapidly in popularity. Smartphones are becoming an indispensable part of our daily lives, and there has been an increasing trend in the development of mobile applications. At the present time, new generations of smartphones are more powerful with sensitive sensor and High-speed wireless transmission, allows updating user's current location with very small errors.

This paper presents a microscopic traffic simulation-based method for urban traffic state estimation using Assisted Global Positioning System (A-GPS) mobile phones. In this approach, real-time location data are collected by A-GPS mobile phones to track vehicles traveling on urban roads. In addition, tracking data obtained from individual mobile probes are aggregated to provide estimations of average road link speeds along rolling time periods. Moreover, the estimated average speeds are classified to different traffic condition levels, which are prepared for displaying a real-time traffic map on mobile phones. Simulation results demonstrate the effectiveness of the proposed method, which are fundamental for the subsequent development of a system demonstrator.

The Android based application allows users to have an overview of the city's traffic conditions and also allows users to navigate away from the ignition points.

Keywords: Traffic State Estimation; A-GPS Mobile Phones; waypoint avoiding traffic jam

CONTENT

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Traffic information is essential to support the development of many intelligent traffic systems (ITS) applications: crash detection, vehicle navigation, traffic signal control, traffic tracking and so on. For example, from 2007, Google began integrating real time traffic information with mapping services. Traffic data is aggregated from a variety of sources, for example, road sensors, cars, taxi fleets, and newer mobile users [1].

State-of-the-practice traffic data collection in most parts of the world is based on a network of road-side sensors, for example, inductive loop detectors (ILDs), to collect information on traffic at fixed points on the road network [2-4]. Although fixed sensors are a proven technology, they are not deployed at scale largely due to its high cost. Moreover, with fixed sensors, it is only possible to measure the speed in place, which is a capital shortfall in the overall flection speed on the entire road link. In addition, the model type is the specific alignment and detector location, requiring careful calibration [5]. An alternative to the luxury road-side infrastructure is to use a dedicated vehicle such as a floating flow detector [6-8]. Specialized vehicle probes (PVs) are usually equipped with a GPS receiver and a dedicated communications link. A large number of vehicles should be equipped to have enough probes. Enough probes limit the ability to generate information for large areas and the accuracy of results [9]. With the GPS-equipped trend is expected to increase in the future, the capacity and cost of dedicated communication links between in-vehicle equipment and traffic management centers will still limit the size Model of PV [5]. Furthermore, since PV is selected from a particular type of vehicle, for example, taxis or buses, traffic information may be biased and may not represent the entire population [10].

Enough probes limit the ability to generate information for large areas and the accuracy of results [9]. With the GPS-equipped trend is expected to increase in the future, the capacity and cost of dedicated communication links between in-vehicle equipment and traffic management centers will still limit the size Model of PV [5]. Furthermore, since PV is selected from a particular type of vehicle, for example, taxis or buses, traffic information may be biased and may not represent the entire population [10].

Along with the development of mobile information technology, more and more mobile phones are used to collect data. This method avoids installation and maintenance costs, whether in the car or in the vertical. In addition, using mobile phones as probes over - come coverage limitation in road-side sensors and insufficient probes in dedicated PVs. By the end of 2015, there have been 128 million registered cell phones in Vietnam (11), so everyone has more than one. Ideally, any cell phone, even if not in use, could be the probe. Therefore, there may be a large sample size of the system that can be used to probe.

Over the past ten years, the field experiments have shown that the calculation of the possible use of mobile phones as traffic probes [12]. However, most of the experiments have attempted to estimate the state of traffic on the highway and some of the deployment attempts to track urban traffic. It has been suggested that future research efforts should focus on the collection of data traffic for the engine, where there is no current data, rather than gathering data on road traffic, in fixed sensor deployments. Nevertheless, traffic estimation on arterials is more challenging than on freeways due to the following facts [13]:

- ✓ Expressway incident traffic artery is much lower;
- ✓ Artery have more variability in speed.
- ✓ Control pulse signal distribution through the junction.

Among the actual deployment in the past, the majority of them have used the tracking method using network information network signaling, such as transferring measurements or time/angle (the difference) of the times to come. Only very few of them are mobile phones (using GPS supported devices), for example was a pioneering experiment in the field held by Globis Data in 2004 [18] and Mobile field test Century made in 2008 [19]. Results of reviews from the field assay showed that the exploration system based on the network cannot provide accurate traffic data enough for the artery. Because the arteries tend to create these complex, sophisticated probes A-GPS mobile phones more accurate will be the better solution for the urban roads; However, this has not been verified (both [18] and [19] only provide estimates of traffic on the highway). Two issues were identified as the main obstacle to the success of A-GPS mobile phones as traffic probes [20]:

- ✓ Additional communication costs.
- ✓ The slow phone's GPS support.

The two issues are no longer a problem in the current context:

- ✓ The modern mobile networks have extensive communication bandwidth.
- ✓ A-GPS mobile phones are increasingly available on the global market.

When evaluating the results from field tests, there is hardly any available ground traffic data to compare with, especially for arterial roads. Additionally, the field test data is not suitable for statistical analysis due to variations in different tests and limited number of observations. In simulation-based studies, on the other hand, individual vehicle tracks and aggregated traffic states can be extracted as “ground truth” [15]. In addition, traffic simulations can generate traffic data under a variety of traffic conditions, featured by different volumes and road net- works [21]. Although these simulation studies do not replicate the actual conditions precisely, they may still provide valuable indication of the potential performance of a probe-based traffic information system.

When evaluating the results from the test field, there is little data on the ground traffic to compare with, especially for road traffic. In addition, the field test data are inconsistent with the reasons for the analysis of statistical variables that can vary in the number of checks and observations. On the basis of simulation studies, on the other hand, the synthesis of individual lanes and flow states can be extracted as "ground truth" [15]. Additionally, the simulated traffic can produce more data traffic in different traffic conditions because of different mass and road network [21]. Although the study is not accurate to simulate realistic conditions, they can still provide valuable markers for the efficiency of the potential traffic information system based on probe.

In our traffic information system, A-GPS mobile phone is used to locate the vehicle. They are also used as on-board processing facility. In addition, the mobile is turned as probes to collect traffic data, based on which the circulation status of municipalities can be estimated and submitted as feedback for the subscription service. In this thesis, we will show how to collect location data using A-GPS mobile phones. Urban traffic is generated by micro-simulation, while the small scene measurements are used to simulate the measurements A-GPS. This thesis is organized as follows: chapter 2 provides an overview of the problem. In chapter 3, we will show the related knowledge and the implementation of the problem. And finally, chapter 4 concludes this thesis.

Chapter 2

TRAFFIC JAM DETECTION PROBLEM ON MOBILE DEVICE

2.1. The problem

The traffic jam detection problem allows users as well as the authorities to have an overview of the traffic situation in Hanoi. Specifically, users can see areas of traffic as well as areas that are jamming, to be able to choose the right route. Functionalists can detect areas with high levels of congestion that are frequently subject to congestion, so that they can be reasonably regulated.

The problem can be expressed as follows:

- ✓ Input: Traffic data, including location, speeds that correspond to the periods of time of traffic participants, are collected through mobile devices.
- ✓ Output: overview of traffic situation in Hanoi.

2.2. Traffic jam detection system

This system is developed to emulate the A-GPS mobile phone-based urban traffic estimation. The framework consists of three parts:

- ✓ Traffic data collection.
- ✓ Location data processing and speed aggregation.

- ✓ Result presentation.

The second step is handle in server, thus, in this thesis, I am not going to show detail about data processing.

2.3. Analysis and design

2.3.1. Requirement analysis

The application to detect traffic conditions on mobile devices has the main function of providing a visual overview of the traffic situation in Hanoi by processing traffic information collected from mobile devices. When a user joins the traffic and starts the application, the application retrieves traffic status data from the server, displays it on the user interface, and records traffic data, stores it on the device, and Send to the server after a certain time period.

Users can also perform other operations with the map, such as finding a location, finding a way between two locations, and, in particular, finding a way to avoid high traffic areas.

2.3.1.1 Functional requirements

The application has 3 main functions:

- Overview traffic condition
- Finding shortest path between 2 locations
- Finding path between 2 locations avoiding high traffic area

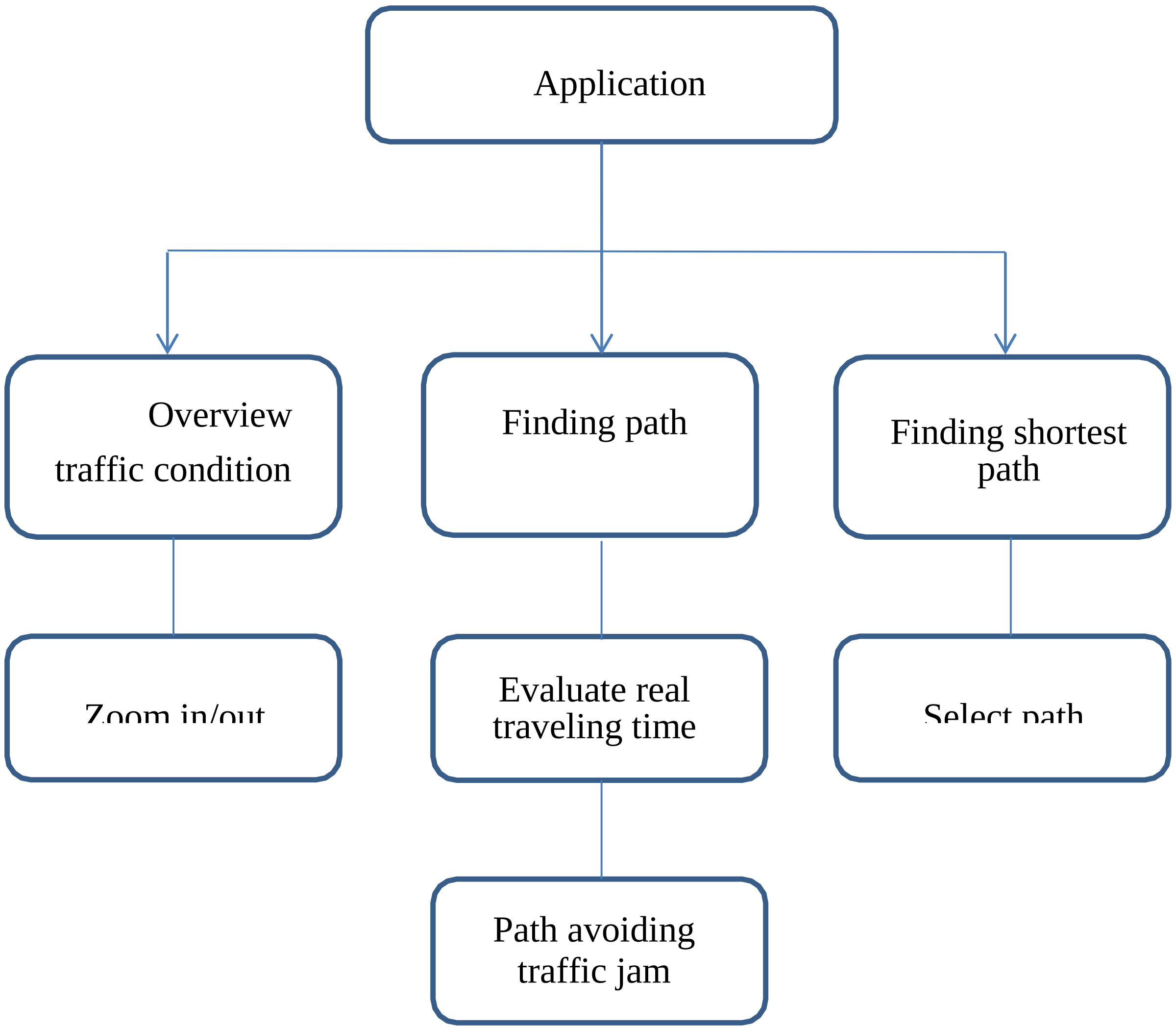

Application
Overview traffic condition
Finding path
Finding shortest path
Zoom in/out
Evaluate real traveling time
Select path
Path avoiding traffic jam

Figure 2.1: Main functions of application

Table 2.1 Main function description

	Function	**Description**
1	Overview traffic condition	Overview traffic condition: The map is divided into small squares with colors representing traffic density
		Zoom in/out: the map is zoom-able
2	Finding shortest path	Finding shortest path: enter 2 separate locations, the map will show the shortest path to get the destination
		Select another path: the map also provide other options for getting destination.
3	Finding path avoiding traffic jam	Finding path avoiding traffic jam: evaluate paths not by distance, but by the smallest number of high traffic area that the path go through.
		Choose another option: the map also provide other options for getting destination.

2.3.1.2 Non-functional requirement

Application should ensure non-functional requirements as follows:

- Ease of Use: User-friendly interface, simple operation, reasonable screen layout.
- Reliability: The system operates according to the required functions, displaying the correct information.
- Resource save: consume less power, storage and bandwidth.

2.3.2 Function analysis

2.3.2.1 Overview traffic condition

Table 2.2 Overview traffic condition function analysis

Use Case	Overview traffic condition
Actor	User
Brief Description	Overview about traffic condition in Ha Noi.
Main Flow	1. User open left menu then click on Traffic State. 2. System will show the traffic state in Ha Noi
Alternative Flows	1. No internet connection → show error message. 2. User can zoom in and out the map.
Pre-condition	None
Post-conditions	None
Extension point	None.
Special Requirements	1. Friendly user interface. 2. Delay time less than 5 seconds.

2.3.2.2 Show your location

Table 2.3 Show your location function analysis

Use Case	Show your location
Actor	User
Brief Description	Show your location with marker at the center.
Main Flow	1. User open left menu then click on Your location. 2. System will show the map with marker to mark user location.
Alternative Flows	1. No internet connection → show error message. 2. User can move the map to see other map area.

Pre-condition	1.Internet connection 2. User accept permission to access GPS.
Post-conditions	None
Extension point	None.
Special Requirements	1. Friendly user interface. 2. Delay time less than 5 seconds.

2.3.2.3 Finding shortest path

Table 2.4 Finding shortest path function analysis

Use Case	Finding shortest path
Actor	User
Brief Description	Show the shortest path between 2 separate locations.
Main Flow	1. User open left menu then click on Normal Search. 2. On map fragment, click on Choose Location 3. Enter the location 4. On map fragment, click on Choose Destination 5. Enter the destination 6. Click on enter button
Alternative Flows	1. No internet connection → show error message. 2. The map show the shortest path. 3. User also can choose another path.
Pre-condition	Internet connection
Post-conditions	None
Extension point	None.
Special	1. Friendly user interface.

Requirements	2. Delay time less than 5 seconds. 3. Place auto-complete when choose location.

2.3.2.4 Finding path avoiding traffic

Table 2.5 Finding path avoiding traffic function analysis

Use Case	Finding path avoiding traffic
Actor	User
Brief Description	Show the path between 2 separate locations avoiding high-traffic areas
Main Flow	1. User open left menu then click on Advance Search. 2. On map fragment, click on Choose Location 3. Enter the location 4. On map fragment, click on Choose Destination 5. Enter the destination 6. Click on enter button
Alternative Flows	1. No internet connection → show error message. 2. The map show the path with less traffic areas. 3. User also can choose another path.
Pre-condition	1. Internet connection. 2. System has got traffic condition from server.
Post-conditions	None
Extension point	None.
Special Requirements	1. Friendly user interface. 2. Delay time less than 5 seconds. 3. Place auto-complete when choose location.

2.3.2.5 Collecting traffic data

Table 2.6 Collecting traffic data function analysis

Use Case	Collecting traffic data
Actor	None
Brief Description	Collect traffic data when users in the traffic, including location, speed, time,… written in log file, saved in local storage and sent to server in every 15 minutes.
Main Flow	1. User start the application. 2. User joint the traffic (location changed) 3. The system auto collect traffic data.
Alternative Flows	1. Full of local storage → show error message
Pre-condition	1. Internet connection. 2. Permission for write local storage.
Post-conditions	None.
Extension point	None.
Special Requirements	None.

2.2.3 Design

2.2.3.1 Use case diagram

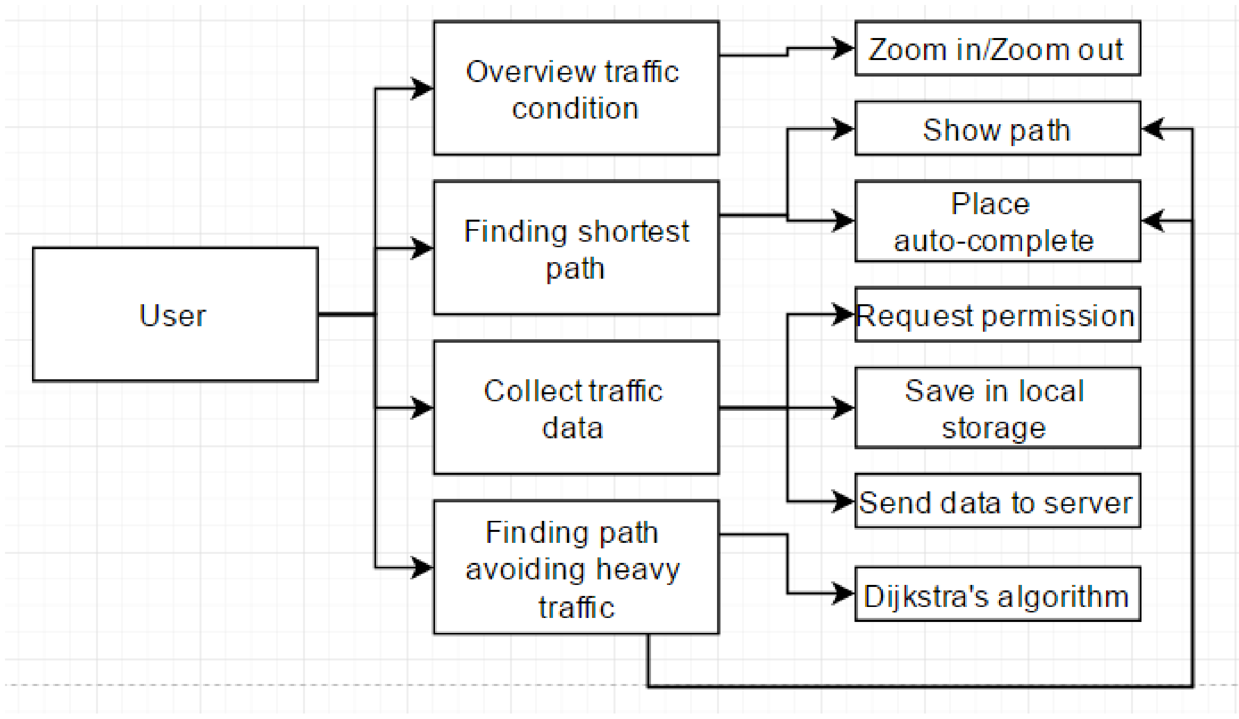


Figure 2.2 Use case diagram

2.2.3.2 Overview traffic condition function

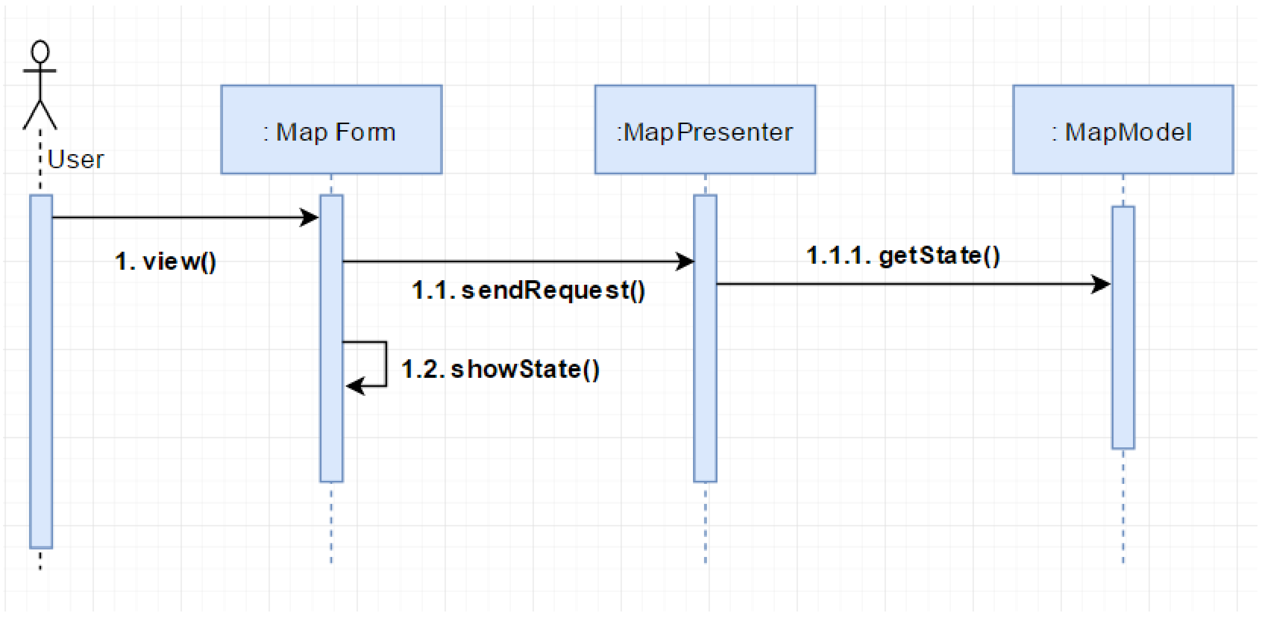


Figure 2.3 Overview traffic condition function

2.2.3.3 Finding shortest path

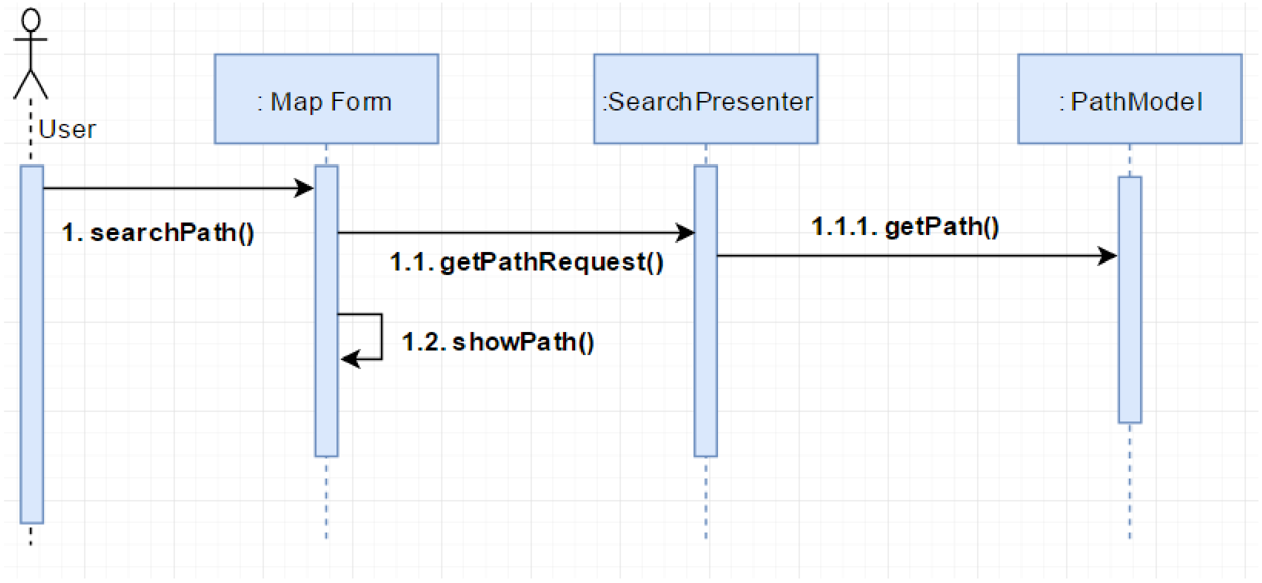


Figure 2.4 Finding shortest path

2.2.3.4 Finding path avoiding traffic

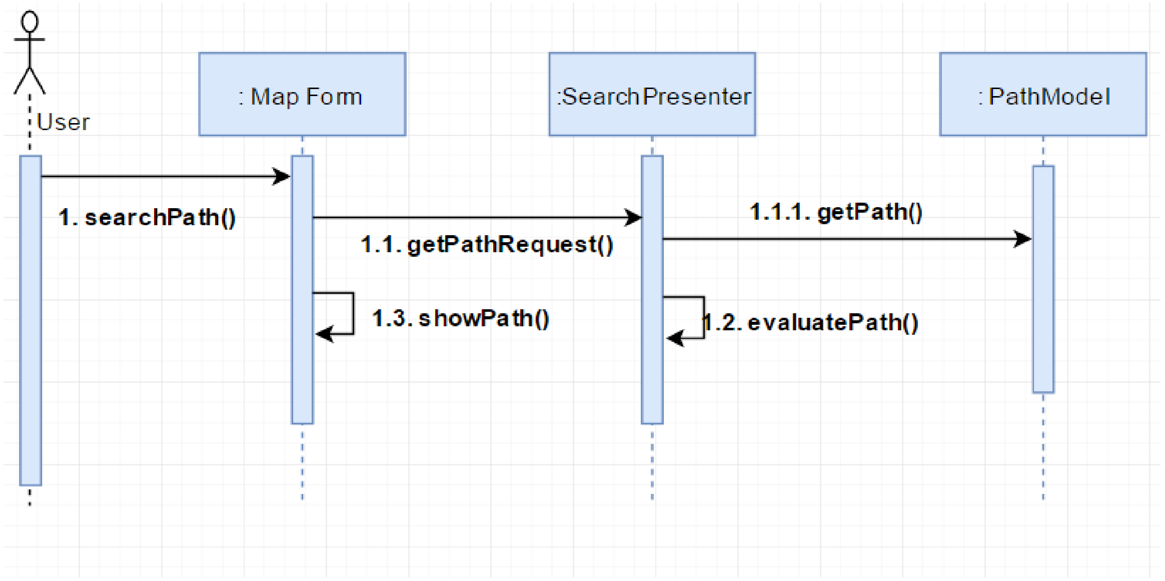


Figure 2.5 Finding path avoiding traffic

Chapter 3

TRAFFIC DETECTION SYSTEM IMPLEMENTATION

3.1 Related knowledge

3.1.1. Google map API

3.1.1.1. What is google map

Google Maps is a Web-based service that provides detailed information about geographical regions and sites around the world. In addition to conventional road maps, Google Maps offers aerial and satellite views of many places. In some cities, Google Maps offers street views comprising photographs taken from vehicles [23].

Google Maps offers several services as part of the larger Web application. A route planner offers directions for drivers, bikers, walkers, and users of public transportation who want to take a trip from one specific location to another.

The Google Maps application program interface (API) makes it possible for Web site administrators to embed Google Maps into a proprietary site such as a real estate guide or community service page.

Google Maps for Mobile offers a location service for motorists that utilizes the Global Positioning System (GPS) location of the mobile device (if available) along with data from wireless and cellular networks.

Google Street View enables users to view and navigate through horizontal and vertical panoramic street level images of various cities around the world.

Supplemental services offer images of the moon, Mars, and the heavens for hobby astronomers.

Google Maps provides a highly responsive, intuitive mapping interface with embedded, detailed street and aerial imagery data. In addition, map controls can be embedded in the product to give users full control over map navigation and the display of street and imagery data. Users can also perform map panning through the use of the "arrow" keys on a keyboard, as well as by dragging the map via the mouse. All of these capabilities combine to provide a compelling product, but the primary driver behind its rapid acceptance as an Internet mapping viewer is its ability to customize the map to fit application-specific needs. For instance, a real estate agency might develop a Web-based application that allows end user searching for residential properties to display the results on a Google Maps application [24].

3.1.1.2 Google map API and google API key

At this time, the Google Maps API is a free beta service. Before you can get started developing Google Maps applications you will need to sign up for an API key. When you sign up for an API key, you must specify a Web site URL that will be used in your development. One problem frequently associated with Google Maps is that you must acquire a unique key for each directory that will serve Google Maps. Google Maps will generate a unique key-code for the directory that you specify. You must use this key-code in each script that accesses the Google Maps API.

Google also provides documentation for its product, including full documentation of the classes, methods and events available for the Google Maps objects as well as code examples to get you started. In addition, Google provides a blog and discussion group for additional information on using the API

3.1.2 Assisted Global Positioning System (A-GPS)

GPS was originally built solely for military purposes, where it was used to guide aircraft, soldiers, and even bombs. In most instances, the receivers were positioned in open areas having line-of-sight access to satellites. But since GPS was opened for commercial use, new applications introduced greater demands on the system [25].

These new applications required GPS signals to reach places blocked by some sort of overhead cover such as trees or roofs. Thus, the concept of A-GPS or assisted GPS was created. In addition to providing better coverage, A-GPS also improves the start-up time, which is the time required by the satellites and the receivers to establish a reliable connection. This originally took about one minute. For even better coverage, some cellphones make use of a combination of A-GPS and other location-based technologies such as a Wi-Fi positioning system and cell site triangulation.

Some of the first A-GPS-enabled cell phones, such as those from Verizon Wireless and Sprint-Nextel, had special chips allowing them to lock into the system, even with only two satellites in the line-of-sight. Additional information was provided by the carriers' wireless network. These chips are much less expensive than regular GPS chips and consume less power.

3.1.3 Android platform

3.1.3.1. A brief history of Android

The Android platform is the product of the Open Handset Alliance, a group of organizations collaborating to build a better mobile phone. The group, led by Google, includes mobile operators, device handset manufacturers, component manufacturers, software solution and platform providers, and marketing companies. From a software development standpoint, Android sits smack in the middle of the open source world.

The first Android-capable handset on the market was the G1 device manufactured by HTC and provisioned on T-Mobile. The device became available after almost a year of speculation, where the only software development tools available were some incrementally improving SDK releases. As the G1 release date neared, the Android team released SDK V1.0 and applications began surfacing for the new platform. [26]

To spur innovation, Google sponsored two rounds of "Android Developer Challenges," where millions of dollars were given to top contest submissions. A few months after the G1, the Android Market was released, allowing users to browse and download applications directly to their phones. Over about 18 months, a new mobile platform entered the public arena.

3.1.3.2 The Android platform

With Android's breadth of capabilities, it would be easy to confuse it with a desktop operating system. Android is a layered environment built upon a foundation of the Linux kernel, and it includes rich functions. The UI subsystem includes:

- ✓ Windows
- ✓ Views
- ✓ Widgets for displaying common elements such as edit boxes, lists, and drop-down lists

Android includes an embeddable browser built upon WebKit, the same open source browser engine powering the iPhone's Mobile Safari browser.

Android boasts a healthy array of connectivity options, including Wi-Fi, Bluetooth, and wireless data over a cellular connection (for example, GPRS, EDGE, and 3G). A popular technique in Android applications is to link to Google Maps to display an address directly within an application. Support for location-based services (such as GPS) and accelerometers is also available in the Android software stack, though not all Android devices are equipped with the required hardware. There is also camera support.

Historically, two areas where mobile applications have struggled to keep pace with their desktop counterparts are graphics/media, and data storage methods. Android addresses the graphics challenge with built-in support for 2-D and 3-D graphics, including the OpenGL library. The data-storage burden is eased because the Android platform includes the popular open source SQLite database. Figure 2.1 shows a simplified view of the Android software layers.

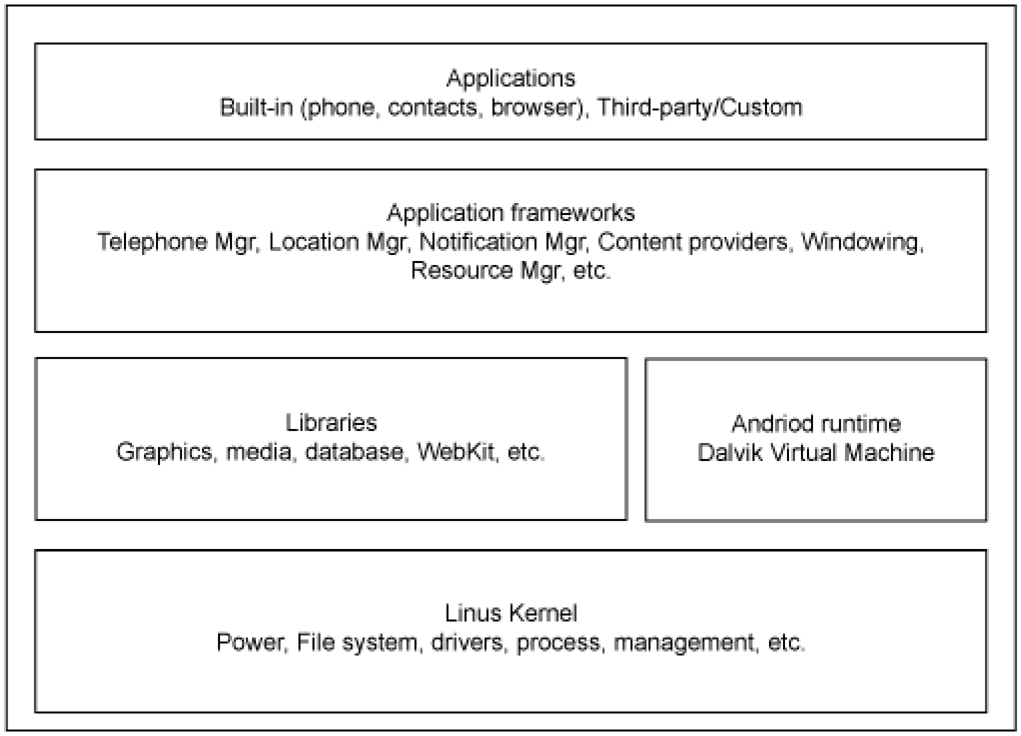

Applications
Built-in (phone, contacts, browser), Third-party/Custom
Application frameworks
Telephone Mgr, Location Mgr, Notification Mgr, Content providers, Windowing, Resource Mgr, etc.
Libraries
Graphics, media, database, WebKit, etc.
Andriod runtime
Dalvik Virtual Machine
Linus Kernel
Power, File system, drivers, process, management, etc.

Figure 3.1. Android software layers

3.1.3.3 Why android?

- Android is popular: Android and iOS accounted for 99.6 percent of all smartphone sales in the fourth quarter of 2016. Of the 432 million smartphones sold in the last quarter, 352 million ran Android (81.7 percent) and 77 million ran iOS (17.9 percent) [27] , then, Android dominated the smartphone market with a share of 86.8% [28]
- Android is free: Android, since the day it was launched, has been available free of cost and Google made it clear that it will be free in future as well. The OS caught the attention of manufacturers across the world and many initially adopted it for low cost smartphones. Seeing the huge response of the buyers to the Android-based phones, many equipment manufacturers joined the league and more are joining every day. Therefore the free nature of this operating system help in reducing cost thereby helping buyers too.
- Android is open source: Unlike other operating systems that are protected by

lots of copyrights, Google chose to keep Android open for all. By doing this, the company got many programmers from around the globe to develop applications, while keeping its liabilities to a minimum. With many brains working on the system, newer and newer ideas were incorporated, which in turn helped in making Android a preferred choice.

- Android is open for customization: Unlike Windows or IOS, device manufacturers are free to modify Android as per their needs. Users enjoy much needed flexibility and ease of use because manufacturers are now able to modify anything and everything they need to make the experience a pleasant one.
- Android has large number of applications: Android Market created an opportunity for millions of application developers around the globe to show their skills and come up with newer applications for Android phones. Its users therefore have a wide variety of applications to choose from and can customize their phones for a personal experience. Android opened numerous possibilities for both device manufacturers and application developers, while helping to reduce the cost of smartphones, thus making them accessible to the common man.

3.1.4 Model-View-Presenter model

Architecture Pattern is a fundamental part of computer science. It’s the only way to maintain a project clean, expansible and testable. The patterns are recognized solutions that have been developed over the years and are considered industry standards. They’re constantly evolving, and in the Android SDK, the reliable Model View Controller (MVC) pattern is steadily being dropped for the Model View Presenter (MVP).

3.1.4.1. Differences between MVC and MVP

Model–view–presenter (MVP) is a derivation of the model–view–controller (MVC) architectural pattern, and is used mostly for building user interfaces.

In MVP the presenter assumes the functionality of the “middle-man”. In MVP, all presentation logic is pushed to the presenter.

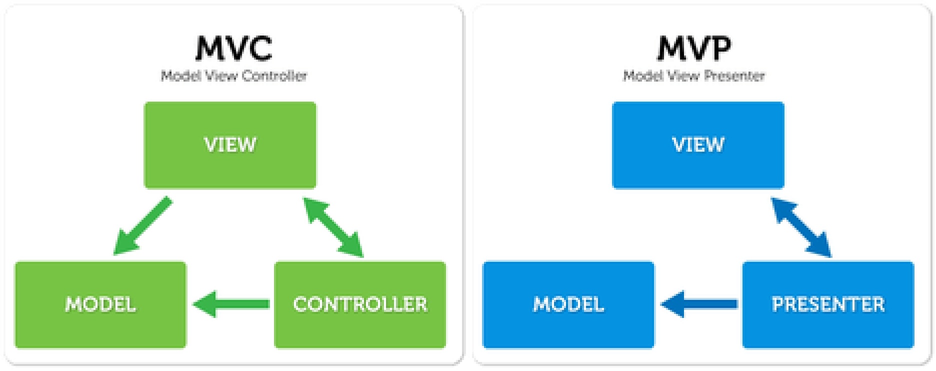


Fig 3.2 MVC vs MVP

Model View Presenter

- ✓ View more separated from Model. The Presenter is the mediator between Model and View.
- ✓ Easier to create unit tests
- ✓ Generally there is a one to one mapping between View and Presenter, with the possibility to use multiple Presenters for complex Views

Model View Controller

- ✓ Controllers are behavior based and can share multiple views.
- ✓ View can communicate directly with Model

3.1.4.2. Model View Presenter (MVP) on Android

Android's separation of concerns isn't well defined. The Activities for example, have a too close relationship with data mechanisms. Although, for an application to become expandable, maintainable and testable, it's extremely important to create a deep separation of concerns and this is probably the biggest advantage that we'll get with the MVP adoption.

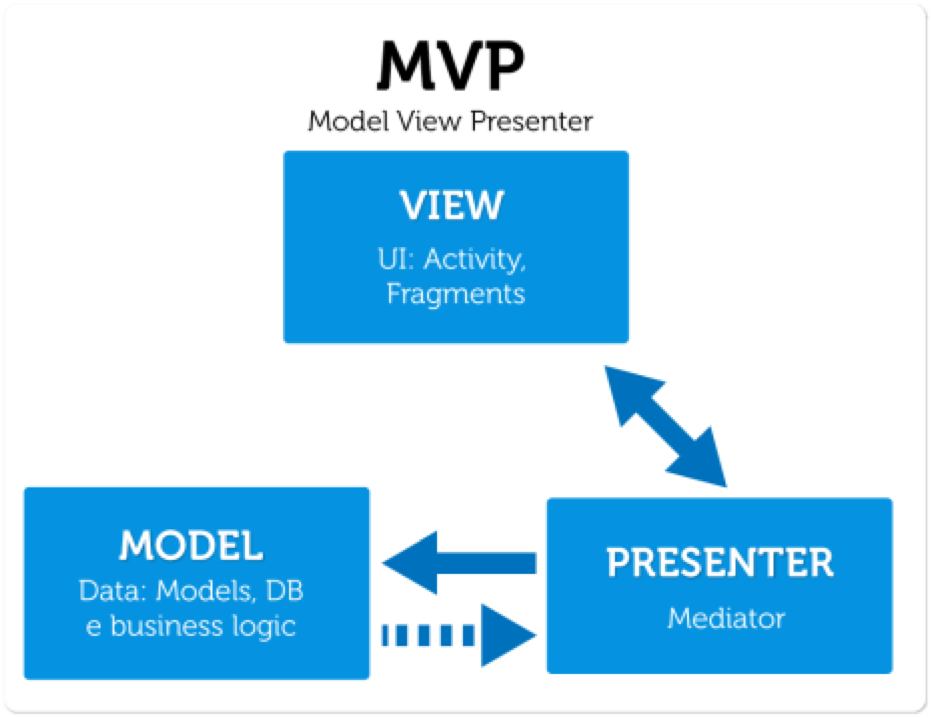


Figure 3.3 MVP model

Presenter:

The Presenter is responsible to act as the middle man between View and Model. It retrieves data from the Model and returns it formatted to the View. But unlike the typical MVC, it also decides what happens when you interact with the View.

View:

The View, usually implemented by an Activity, will contain a reference to the presenter. The only thing that the view will do is to call a method from the Presenter every time there is an interface action.

Model:

In an application with a good layered architecture, this model would only be the gateway to the domain layer or business logic. See it as the provider of the data we want to display in the view.

3.1.4.3. MVP action diagram

Let’s imagine an extremely simple application, that allows the user to take note. Basically, the user inserts notes while the system saves and exhibit the data. If we trace the insert note action, considering that the app was developed using MVP pattern, we’ll get the following diagram:

- ✓ User clicks on “insert note”. View sends note to Presenter

  getPresenter.newNote(textNote)
- ✓ **Presenter** creates a new Note, using the given String sent and invokes on **Model** the method responsible to insert the data on DB getModel.insertNote(note, this)
- ✓ **Model** inserts the Note on DB and informs **Presenter** about the success/error using the callback sent callback.onSuccess()
- ✓ **Presenter** processes the result and require **View** to exhibit a Toast success message getView.showToast(msg)

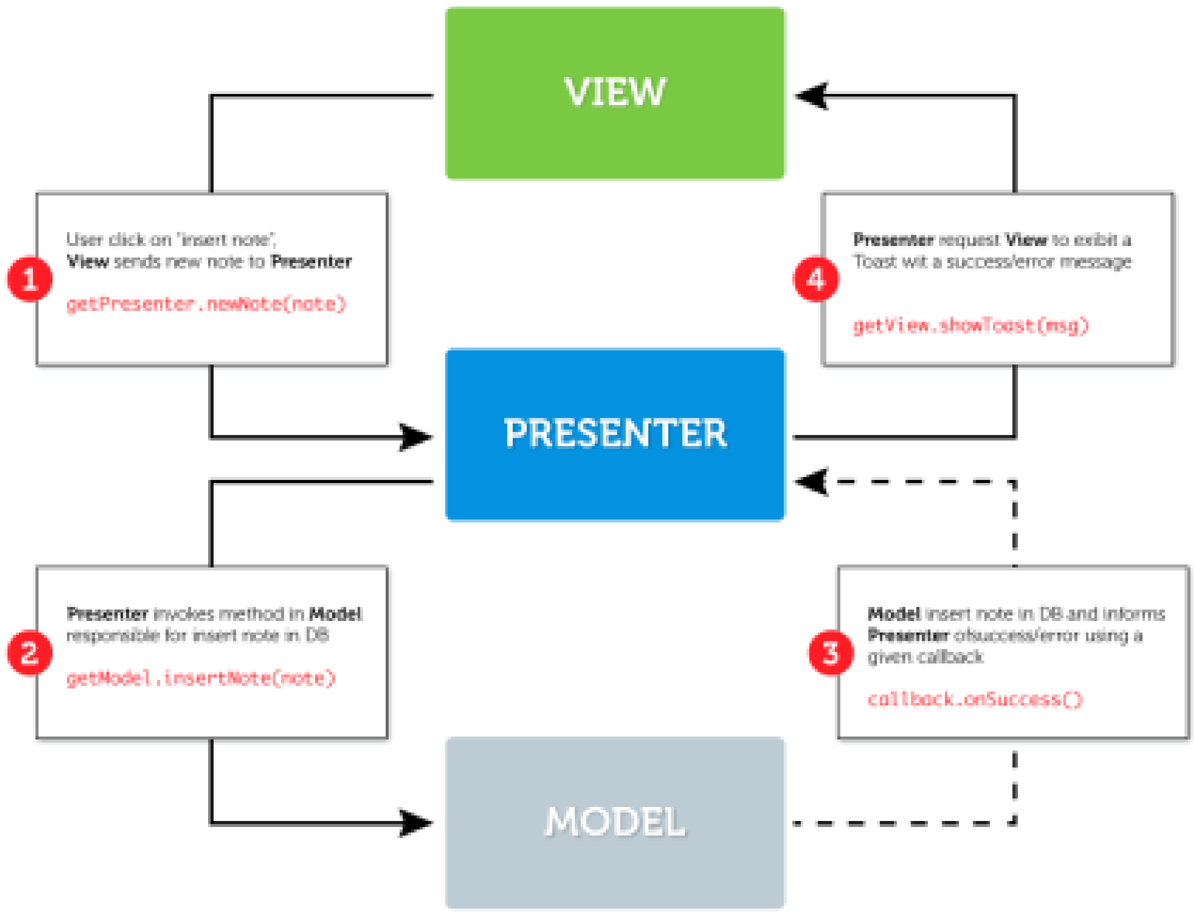

VIEW
User click on 'insert note', View sends new note to Presenter
getPresenter.newNote(note)
Presenter request View to exibit a Toast wit a success/error message
getView.showToast(msg)
PRESENTER
Presenter invokes method in Model responsible for insert note in DB
getModel.insertNote(note)
Model insert note in DB and informs Presenter ofsuccess/error using a given callback
callback.onSuccess()
MODEL
1
2
3
4

Figure 3.4 Model View Presenter (MVP) action diagram

3.1.4.4 MVP class diagram

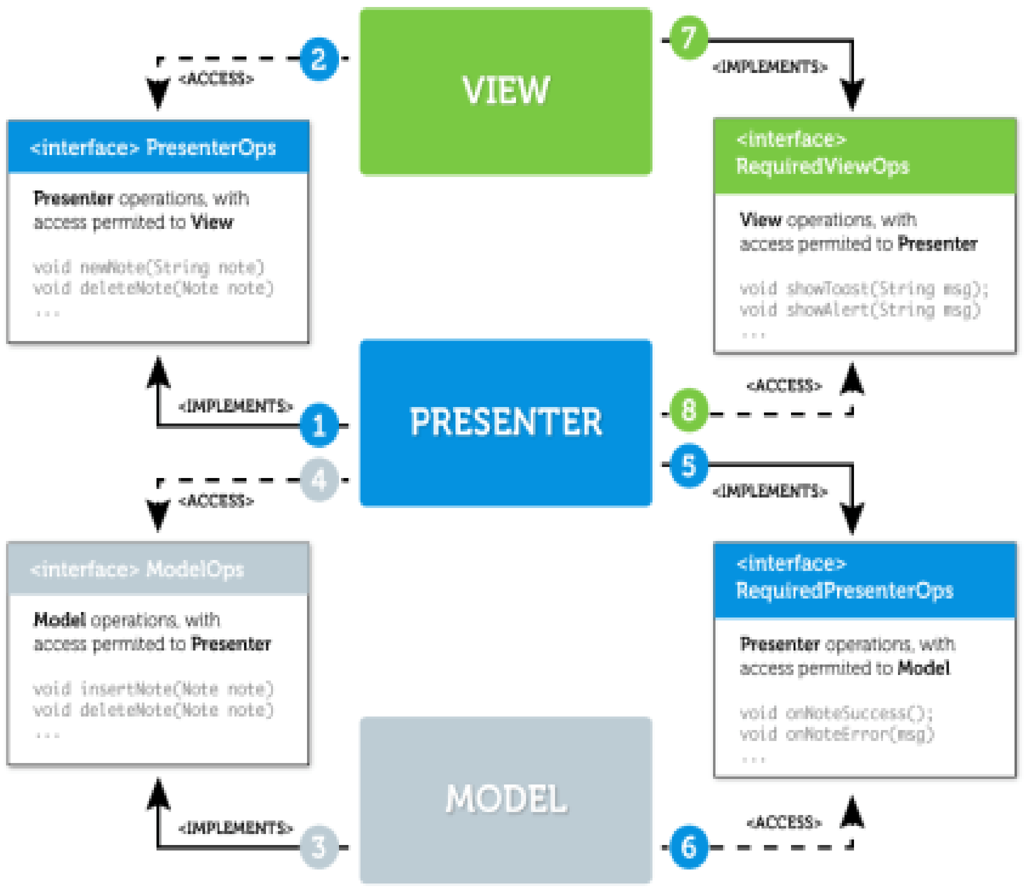

VIEW
PRESENTER
MODEL
<interface> PresenterOps
Presenter operations, with access permited to View
void newNote(String note)
void deleteNote(Note note)
...
<interface> RequiredViewOps
View operations, with access permited to Presenter
void showToast(String msg);
void showAlert(String msg)
...
<interface> ModelOps
Model operations, with access permited to Presenter
void insertNote(Note note)
void deleteNote(Note note)
...
<interface> RequiredPresenterOps
Presenter operations, with access permited to Model
void onNoteSuccess();
void onNoteError(msg)
...
<ACCESS>
<IMPLEMENTS>
1
2
3
4
5
6
7
8

Figure 3.5 Model View Presenter (MVP) class diagram

- ✓ Presenter implements interface PresenterOps.
- ✓ View receives reference from PresenterOps to access Presenter.
- ✓ Model implements interface ModelOps.
- ✓ Presenter receives reference from ModelOps to access Model.
- ✓ Presenter implements RequiredPresenterOps.
- ✓ Model receives reference from RequiredPresenterOps to access Presenter.
- ✓ View implements RequiredViewOps.

- ✓ Presenter receives reference from RequiredViewOps to access View.

3.1.5. Shortest path problem

In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

The problem of finding the shortest path between two intersections on a road map (the graph's vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of its road segment) may be modeled by a special case of the shortest path problem in graphs.

3.1.5.1. Definition

The shortest path problem can be defined for graphs whether undirected, directed, or mixed. It is defined here for undirected graphs; for directed graphs the definition of path requires that consecutive vertices be connected by an appropriate directed edge. [29]

The shortest path problem can be defined for graphs whether undirected, directed, or mixed. It is defined here for undirected graphs; for directed graphs the definition of path requires that consecutive vertices be connected by an appropriate directed edge.

Two vertices are adjacent when they are both incident to a common edge. A path in an undirected graph is a sequence of vertices $\boldsymbol{P} = (\boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_n}) \in \boldsymbol{V} \times \boldsymbol{V} \dots \times \boldsymbol{V}$ such that $\boldsymbol{v_i}$ is adjacent to $\boldsymbol{v_{i+1}}$ for $\boldsymbol{1} \leq \boldsymbol{i} < n$. Such a path $\boldsymbol{P}$ is called a path of length $\boldsymbol{n} - \boldsymbol{1}$ from $\boldsymbol{v_1}$ to $\boldsymbol{v_n}$. (The $\boldsymbol{v_i}$ are variables; their numbering here relates to their position in the sequence and needs not to relate to any canonical labeling of the vertices.)

Let $\boldsymbol{e_{i,j}}$ be the edge incident to both $\boldsymbol{v_i}$ and $\boldsymbol{v_j}$. Given a real-valued weight function $\boldsymbol{f}: \boldsymbol{E} \to \boldsymbol{R}$, and an undirected (simple) graph $\boldsymbol{G}$, the shortest path from $\boldsymbol{v}$ to $\boldsymbol{v'}$ is the path $\boldsymbol{P} = (\boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_n})$ (where $\boldsymbol{v_1} = \boldsymbol{v}$ and $\boldsymbol{v_n} = \boldsymbol{v'}$) that over all possible $\boldsymbol{n}$ minimizes the sum $\sum_{\boldsymbol{i=1}}^{\boldsymbol{n-1}} \boldsymbol{f}(\boldsymbol{e_{i,i+1}})$ When each edge in the graph has unit weight or $\boldsymbol{f}: \boldsymbol{E} \to \{\boldsymbol{1}\}$, this is equivalent to finding the path with fewest edges.

The problem is also sometimes called the single-pair shortest path problem, to distinguish it from the following variations:

- ✓ The single-source shortest path problem, in which we have to find shortest paths from a source vertex *v* to all other vertices in the graph.
- ✓ The single-destination shortest path problem, in which we have to find shortest paths from all vertices in the directed graph to a single destination vertex *v*. This can be reduced to the single-source shortest path problem by reversing the arcs in the directed graph.

The all-pairs shortest path problem, in which we have to find shortest paths between every pair of vertices *v*, *v'* in the graph.

These generalizations have significantly more efficient algorithms than the simplistic approach of running a single-pair shortest path algorithm on all relevant pairs of vertices.

In the next part, I will introduce 3 most common shortest path algorithms, including Dijkstra's algorithm, Ford-Bellman algorithm and Floyd algorithm.

3.1.5.2 Dijkstra's algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later [30]

The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path algorithm is widely used in network routing protocols, most notably IS-IS and Open Shortest Path First (OSPF). It is also employed as a subroutine in other algorithms such as Johnson's.

3.1.5.2.1 Algorithm

Let the node at which we are starting be called the initial node. Let the distance of node *Y* be the distance from the initial node to *Y*. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.

Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.

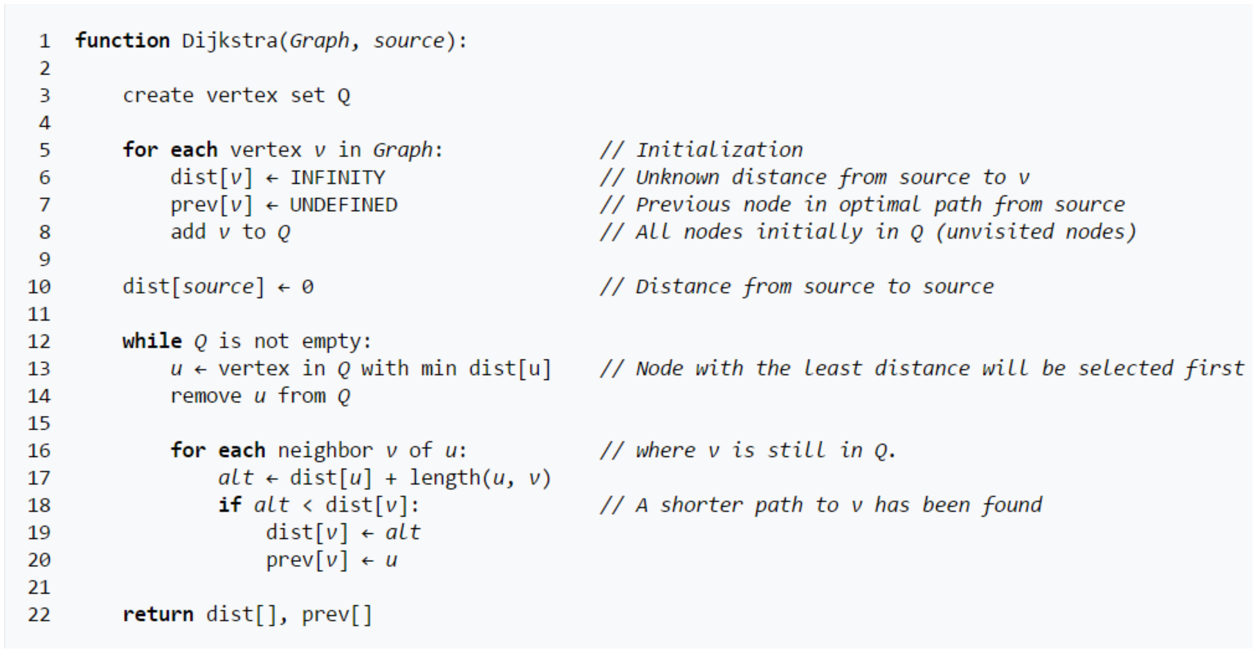
For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbor *B* has length 2, then the distance to *B* (through *A*) will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.

When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.

If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.

Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

3.1.5.2.2 Pseudocode

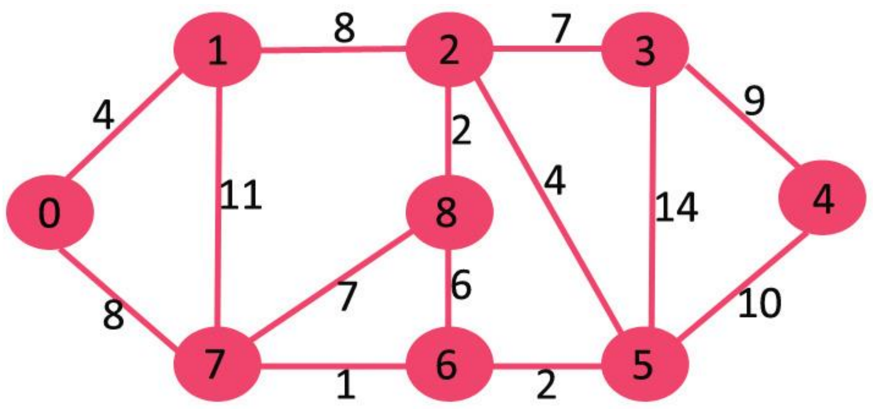


```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:            // Initialization
        dist[v] ← INFINITY                 // Unknown distance from source to v
        prev[v] ← UNDEFINED                // Previous node in optimal path from source
        add v to Q                         // All nodes initially in Q (unvisited nodes)

    dist[source] ← 0                       // Distance from source to source

    while Q is not empty:
        u ← vertex in Q with min dist[u]   // Node with the least distance will be selected first
        remove u from Q

        for each neighbor v of u:          // where v is still in Q.
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:              // A shorter path to v has been found
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

Figure 3.6 Dijkstra's algorithm

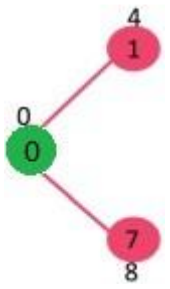
3.1.5.2.3 Example

Let's make it easy to understand with the following example:

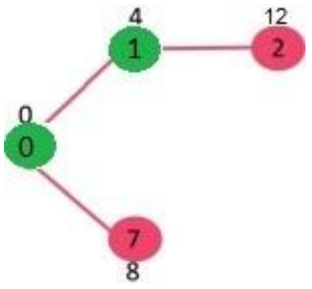


Step 1: The set sptSetis initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum distance value. The vertex 0 is picked, include it in sptSet. So sptSet becomes {0}. After including 0 to sptSet, update distance values of its adjacent vertices. Adjacent vertices of 0

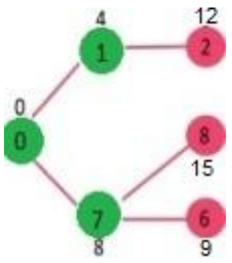
are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their distance values, only the vertices with finite distance values are shown. The vertices included in SPT are shown in green color.



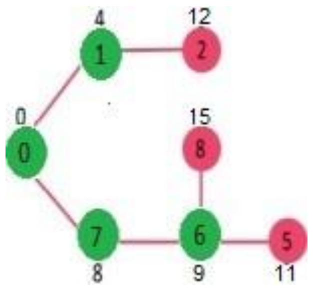
Step 2: Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). The vertex 1 is picked and added to sptSet. So sptSet now becomes {0, 1}. Update the distance values of adjacent vertices of 1. The distance value of vertex 2 becomes 12.



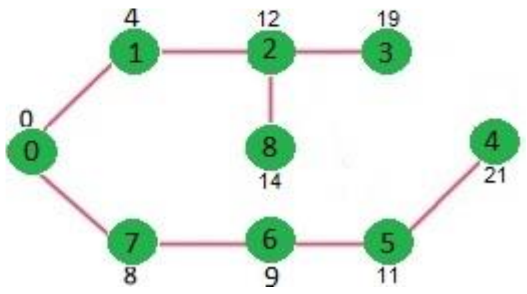
Step 3: Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 7 is picked. So sptSet now becomes {0, 1, 7}. Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (15 and 9 respectively).



Step 4: Pick the vertex with minimum distance value and not already included in SPT (not in sptSET). Vertex 6 is picked. So sptSet now becomes {0, 1, 7, 6}. Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.



Step 5: We repeat the above steps until sptSet doesn't include all vertices of given graph. Finally, we get the following Shortest Path Tree (SPT).



3.1.5.3 Ford-Bellman algorithm

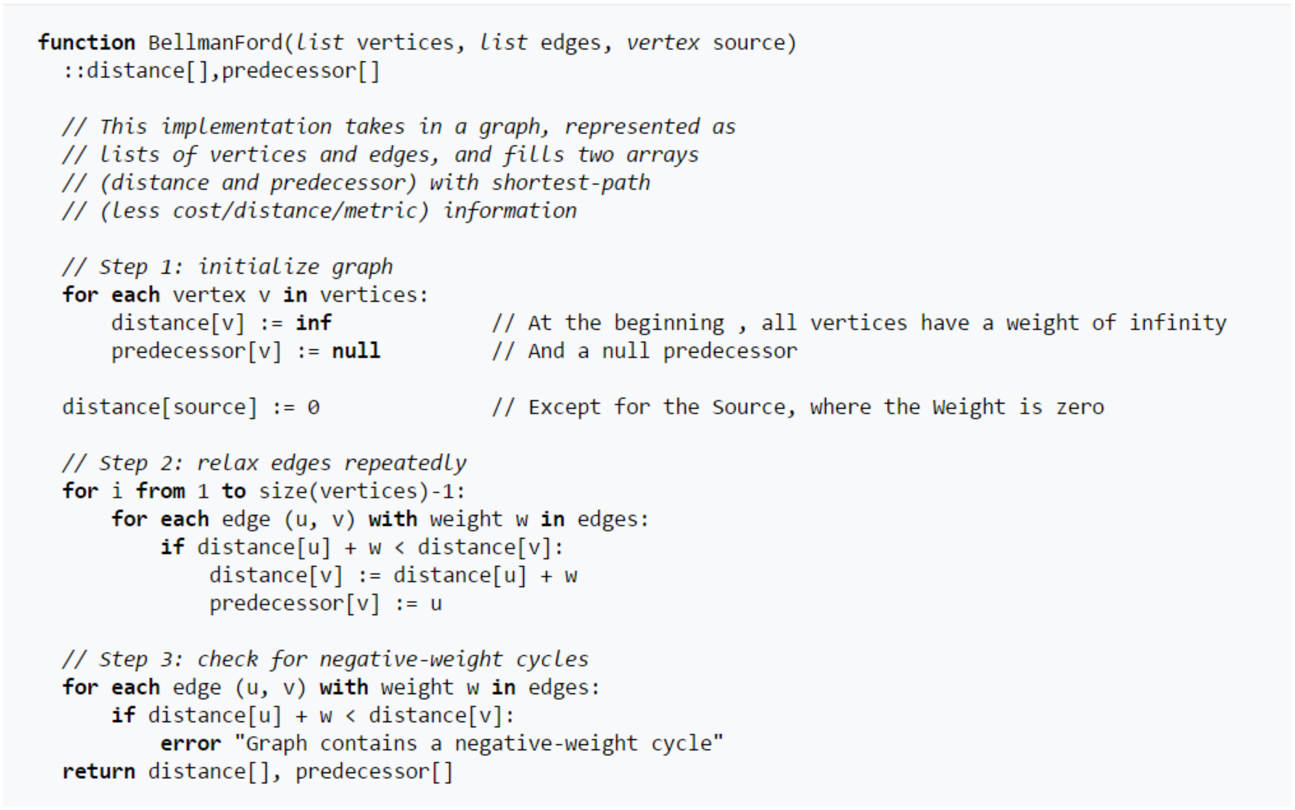
The Ford-Bellman algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. The algorithm was first proposed by Alfonso Shimbel in 1955, but is instead named after Richard Bellman and Lester Ford, Jr., who published it in 1958 and 1956, respectively. Edward F. Moore also published the same algorithm in 1957, and for this reason it is also sometimes called the Bellman–Ford–Moore algorithm [31].

3.1.5.3.1 Algorithm

Like Dijkstra's Algorithm, Bellman–Ford is based on the principle of relaxation, in which an approximation to the correct distance is gradually replaced by more accurate values until eventually reaching the optimum solution. In both algorithms, the approximate distance to each vertex is always an overestimate of the true distance, and is replaced by the minimum of its old value with the length of a newly found path. However, Dijkstra's algorithm uses a priority queue to greedily select the closest vertex that has not yet been processed, and performs this relaxation process on all of its outgoing edges; by contrast, the Bellman–Ford algorithm simply relaxes all the edges, and does this |V|-1 times, where |V| is the number of vertices in the graph. In each of these repetitions, the number of vertices with correctly calculated distances grows, from which it follows that eventually all vertices will have their correct distances. This method allows the Bellman–Ford algorithm to be applied to a wider class of inputs than Dijkstra.

Bellman–Ford runs in O(|V| . |E|) time, where |V| and |E| are the number of vertices and edges respectively.

3.1.5.3.2 Pseudocode

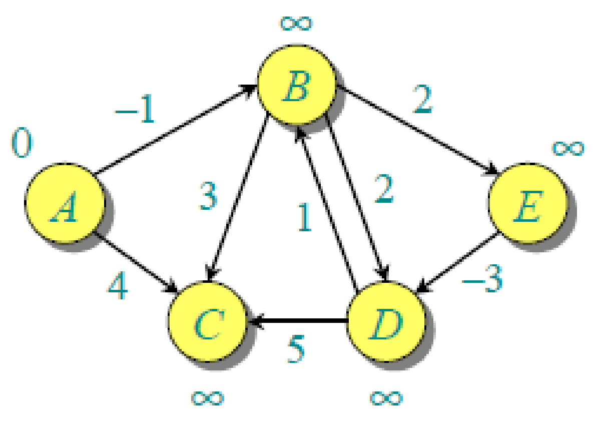


```
function BellmanFord(list vertices, list edges, vertex source)
  ::distance[],predecessor[]

  // This implementation takes in a graph, represented as
  // lists of vertices and edges, and fills two arrays
  // (distance and predecessor) with shortest-path
  // (less cost/distance/metric) information

  // Step 1: initialize graph
  for each vertex v in vertices:
      distance[v] := inf             // At the beginning , all vertices have a weight of infinity
      predecessor[v] := null         // And a null predecessor

  distance[source] := 0              // Except for the Source, where the Weight is zero

  // Step 2: relax edges repeatedly
  for i from 1 to size(vertices)-1:
      for each edge (u, v) with weight w in edges:
          if distance[u] + w < distance[v]:
              distance[v] := distance[u] + w
              predecessor[v] := u

  // Step 3: check for negative-weight cycles
  for each edge (u, v) with weight w in edges:
      if distance[u] + w < distance[v]:
          error "Graph contains a negative-weight cycle"
  return distance[], predecessor[]
```

Figure 3.7 Bellman-Ford algorithm

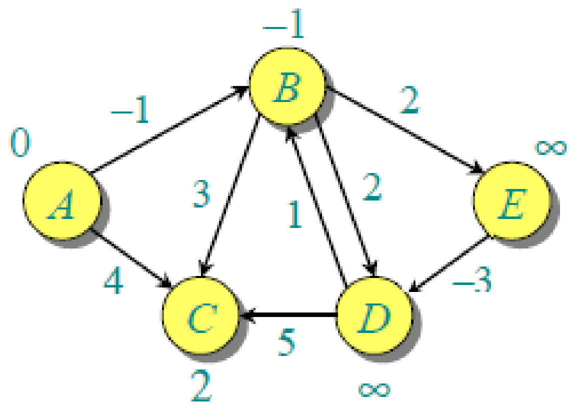
3.1.5.3.3 Example

Let the given source vertex be 0. Initialize all distances as infinite, except the distance to source itself. Total number of vertices in the graph is 5, so all edges must be processed 4 times.



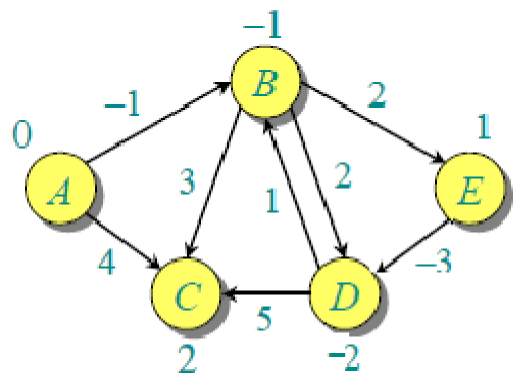
A	B	C	D	E
0	∞	∞	∞	∞

Let all edges are processed in following order: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D). We get following distances when all edges are processed first time. The first row in shows initial distances. The second row shows distances when edges (B,E), (D,B), (B,D) and (A,B) are processed. The third row shows distances when (A,C) is processed. The fourth row shows when (D,C), (B,C) and (E,D) are processed.



A	B	C	D	E
0	∞	∞	∞	∞
0	−1	∞	∞	∞
0	−1	4	∞	∞
0	−1	2	∞	∞

The first iteration guarantees to give all shortest paths which are at most 1 edge long. We get following distances when all edges are processed second time (The last row shows final values).



A	B	C	D	E
0	∞	∞	∞	∞
0	−1	∞	∞	∞
0	−1	4	∞	∞
0	−1	2	∞	∞
0	−1	2	∞	1
0	−1	2	1	1
0	−1	2	−2	1

3.1.5.4 Floyd algorithm

The Floyd–Warshall algorithm is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles). A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices. Although it does not return details of the paths themselves, it is possible to reconstruct the paths with simple modifications to the algorithm. Versions of the algorithm can also be used for finding the transitive closure of a relation R, or (in connection with the Schulze voting system) widest paths between all pairs of vertices in a weighted graph [32].

3.1.5.4.1 Algorithm

The Floyd–Warshall algorithm compares all possible paths through the graph between each pair of vertices. It is able to do this with $\boldsymbol{\Theta(|V|^3)}$comparisons in a graph. This is remarkable considering that there may be up to $\boldsymbol{\Omega(|V|^2)}$edges in the graph, and every combination of edges is tested. It does so by incrementally improving an estimate on the shortest path between two vertices, until the estimate is optimal.

Consider a graph$\boldsymbol{G}$ with vertices $\boldsymbol{V}$ numbered 1 through $\boldsymbol{N}$. Further consider a function shortestPath$\boldsymbol{(i,j,k)}$ that that returns the shortest possible path from $\boldsymbol{i}$ to$\boldsymbol{j}$using vertices only from the set $\boldsymbol{\{1,2,\ldots,k\}}$ as intermediate points along the way. Now, given this function, our goal is to find the shortest path from each $\boldsymbol{i}$ to each $\boldsymbol{j}$ using only vertices in$\boldsymbol{\{1,2,\ldots,k+1\}}$.

For each of these pairs of vertices, the true shortest path could be either:

- ✓ A path that only uses vertices in the set**{1,...,*k*}** or
- ✓ A path that goes from ***i*** to ***k*+1** and then to ***k*+1** to ***j* .**

We know that the best path from***i*** to ***j***that only uses vertices 1 through k is defined by shortestPath**(*i,j,k*)** , it is clear that if there were a better path from ***i*** to ***k*+1** to ***j*** , then the length of this path would be the concatenation of the shortest path from ***i*** to ***k*+1**(using vertices in **{1,…,*k*}**) and the shortest path from **{*k*+1}** to ***j*** (also using vertices in **{1,…,*k*}**).

If ***w*(*i,j*)** is the weight of the edge between vertices i and j , we can define**shortestPath(*i,j,k*+1)** in terms of the following recursive formula: the base case is

shortestPath**(*i,j*,0) = *w*(*i,j*)**

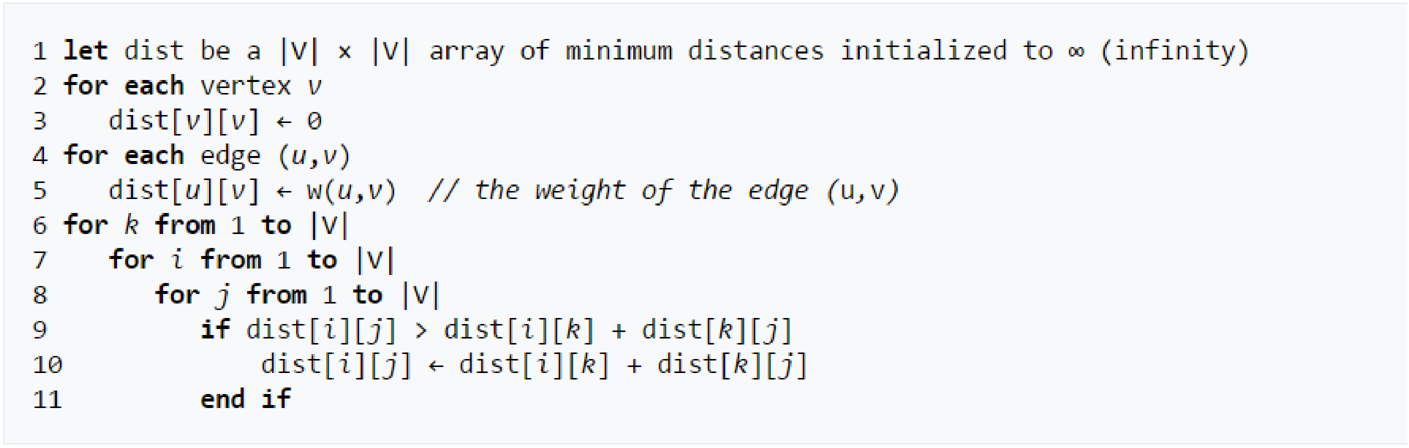
and the recursive case is

shortestPath**(*i,j,k*+1) =**

min(shortestPath**(*i,j,k*),**

shortestPath**(*i, k*+1,*k*)** + shortestPath**(*k*+*1,j,k*)*)***

3.1.5.4.2 Pseudocode



```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each vertex v
   dist[v][v] ← 0
for each edge (u,v)
   dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
for k from 1 to |V|
   for i from 1 to |V|
      for j from 1 to |V|
         if dist[i][j] > dist[i][k] + dist[k][j]
             dist[i][j] ← dist[i][k] + dist[k][j]
         end if
```

Figure 3.8 Floyd algorithm

3.2 Implementation

3.2.1 The environment

3.2.1.2 The mobile device

The mobile device used for the experimental part of this thesis is Google Nexus 5 - Android operating system 6.0.1, the relevant configuration is as follows:

Table 3.1: Hardware configuration

	Hardware	Index
1	Chipset	Qualcom snapdragon 800
2	ROM	16GB
3	RAM	2GB
4	Camera	8MP

3.2.2 Tools and libraries

Tools and libraries that used including:

Table 3.2: Tools and libraries

	Tools/libraries	**Source**
1	Android Studio	http://developer.android.com/intl/vi/sdk/index.html
2	Android Support Appcompat v7	https://github.com/android/platform_frameworks_support
3	Android Support v4	https://github.com/android/platform_frameworks_support

4	Google play service	http://google-play-services.en.uptodown.com/android
5	Circle Image View	https://github.com/hdodenhof/CircleImageView

3.2.3 Traffic data collection function

The most essential problem in the traffic system using GPS equipped probe is the sampling interval data/reports. Typical GPS receivers receive updated location in each 1-3 seconds. How often to collect data from a large number of probes may cause network congestion. To avoid this problem, a method often applied over time, in which the probes to report their data in a predefined time. In addition, [33] declares that the use of a longer sampling interval allows to gather information on the longer distances, thus reducing the chance of occupying a speed not represented. However the sampling interval can not be too long, because it affects the timeliness of data and coverage of the system for short. About how to obtain samples from 10 to 20 seconds can be used in practice, as was pointed out in the previous study [34].

3.2.4 Show traffic condition function

The application uses google map as base to display traffic status in Hanoi. First, identify 4 points covering the whole area of Hanoi. In this article, I will use 4 points as 4 corners of the map:

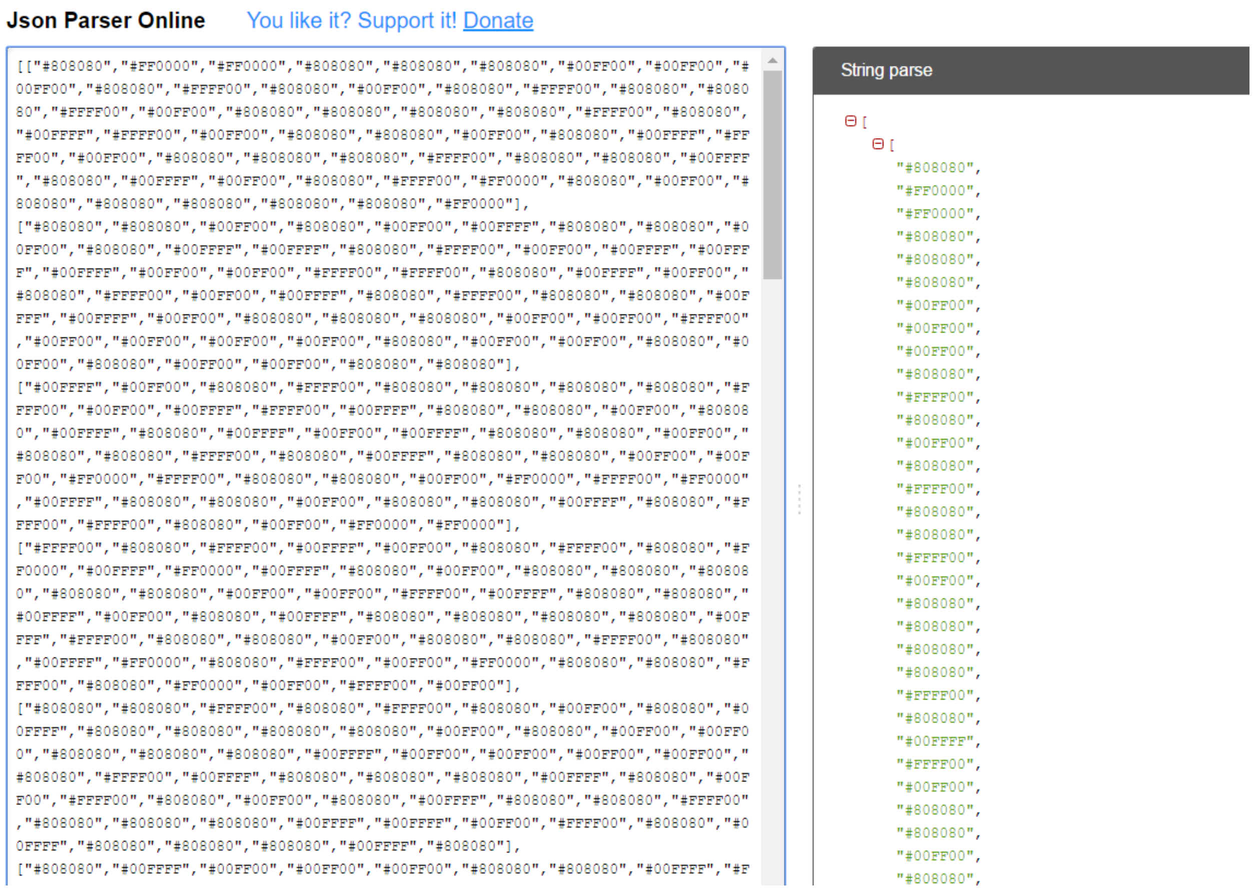
- 21.157200, 105.456390
- 21.157200, 105.951180
- 20.951180, 105.456390
- 20.951180, 105.951180

Then create an overlay on the map, which is divided into small squares, each of those will be colored in a different color, representing the density of traffic. Here, I divided the map into 56 * 23 small cells, so that each cell would have an area of 1 km * 1 km.

Traffic density is expressed in colors. We define five color codes for four levels of traffic density:

- Red (#FF000) – with speed less than 10km/h
- Yellow (#FFFF00) – with speed from 10km/h to 20km/h
- Green (#00FF00) – with speed from 20km/h to 30km/h
- Blue(#0000FF) – with speed more than 30km/h
- Grey (#808080) – no data collected

The data returned from the server is an array of 56 * 23 elements, each of which is the color code of a small map cell. Figure 3.9 shows a sample JSON returned from the server.



```
[["#808080","#FF0000","#FF0000","#808080","#808080","#808080","#00FF00","#00FF00","#
00FF00","#808080","#FFFF00","#808080","#00FF00","#808080","#FFFF00","#808080","#8080
80","#FFFF00","#00FF00","#808080","#808080","#808080","#808080","#FFFF00","#808080",
"#00FFFF","#FFFF00","#00FF00","#808080","#808080","#00FF00","#808080","#00FFFF","#FF
FF00","#00FF00","#808080","#808080","#808080","#FFFF00","#808080","#808080","#00FFFF
","#808080","#00FFFF","#00FF00","#808080","#FFFF00","#FF0000","#808080","#00FF00","#
808080","#808080","#808080","#808080","#808080","#FF0000"],
["#808080","#808080","#00FF00","#808080","#00FF00","#00FFFF","#808080","#808080","#0
0FF00","#808080","#00FFFF","#00FFFF","#808080","#FFFF00","#00FF00","#00FFFF","#00FFF
F","#00FFFF","#00FF00","#00FF00","#FFFF00","#FFFF00","#808080","#00FFFF","#00FF00","
#808080","#FFFF00","#00FF00","#00FFFF","#808080","#FFFF00","#808080","#808080","#00F
FFF","#00FFFF","#00FF00","#808080","#808080","#808080","#00FF00","#00FF00","#FFFF00"
,"#00FF00","#00FF00","#00FF00","#00FF00","#808080","#00FF00","#00FF00","#808080","#0
0FF00","#808080","#00FF00","#00FF00","#808080","#808080"],
["#00FFFF","#00FF00","#808080","#FFFF00","#808080","#808080","#808080","#808080","#F
FFF00","#00FF00","#00FFFF","#FFFF00","#00FFFF","#808080","#808080","#00FF00","#80808
0","#00FFFF","#808080","#00FFFF","#00FF00","#00FFFF","#808080","#808080","#00FF00","
#808080","#808080","#FFFF00","#808080","#00FFFF","#808080","#808080","#00FF00","#00F
F00","#FF0000","#FFFF00","#808080","#808080","#00FF00","#FF0000","#FFFF00","#FF0000"
,"#00FFFF","#808080","#808080","#00FF00","#808080","#808080","#00FFFF","#808080","#F
FFF00","#FFFF00","#808080","#00FF00","#FF0000","#FF0000"],
["#FFFF00","#808080","#FFFF00","#00FFFF","#00FF00","#808080","#FFFF00","#808080","#F
F0000","#00FFFF","#FF0000","#00FFFF","#808080","#00FF00","#808080","#808080","#80808
0","#808080","#808080","#00FF00","#00FF00","#FFFF00","#00FFFF","#808080","#808080","
#00FFFF","#00FF00","#808080","#00FFFF","#808080","#808080","#808080","#808080","#00F
FFF","#FFFF00","#808080","#808080","#00FF00","#808080","#808080","#FFFF00","#808080"
,"#00FFFF","#FF0000","#808080","#FFFF00","#00FF00","#FF0000","#808080","#808080","#F
FFF00","#808080","#FF0000","#00FF00","#FFFF00","#00FF00"],
["#808080","#808080","#FFFF00","#808080","#FFFF00","#808080","#00FF00","#808080","#0
0FFFF","#808080","#808080","#808080","#808080","#00FF00","#808080","#00FF00","#00FF0
0","#808080","#808080","#808080","#00FFFF","#00FF00","#00FF00","#00FF00","#00FF00","
#808080","#FFFF00","#00FFFF","#808080","#808080","#808080","#00FFFF","#808080","#00F
F00","#FFFF00","#808080","#00FF00","#808080","#00FFFF","#808080","#808080","#FFFF00"
,"#808080","#808080","#808080","#00FFFF","#00FFFF","#00FF00","#FFFF00","#808080","#0
0FFFF","#808080","#808080","#808080","#00FFFF","#808080"],
["#808080","#00FFFF","#00FF00","#00FF00","#00FF00","#808080","#808080","#00FFFF","#F
```

```
[
  [
    "#808080",
    "#FF0000",
    "#FF0000",
    "#808080",
    "#808080",
    "#808080",
    "#00FF00",
    "#00FF00",
    "#00FF00",
    "#808080",
    "#FFFF00",
    "#808080",
    "#00FF00",
    "#808080",
    "#FFFF00",
    "#808080",
    "#808080",
    "#FFFF00",
    "#00FF00",
    "#808080",
    "#808080",
    "#808080",
    "#808080",
    "#FFFF00",
    "#808080",
    "#00FFFF",
    "#FFFF00",
    "#00FF00",
    "#808080",
    "#808080",
    "#00FF00",
    "#808080",
```

Figure 3.9 Sample JSON file

Fig. 3.10 shows the density of traffic in Hanoi by coloring the areas in different colors corresponding with various level of traffic jam.

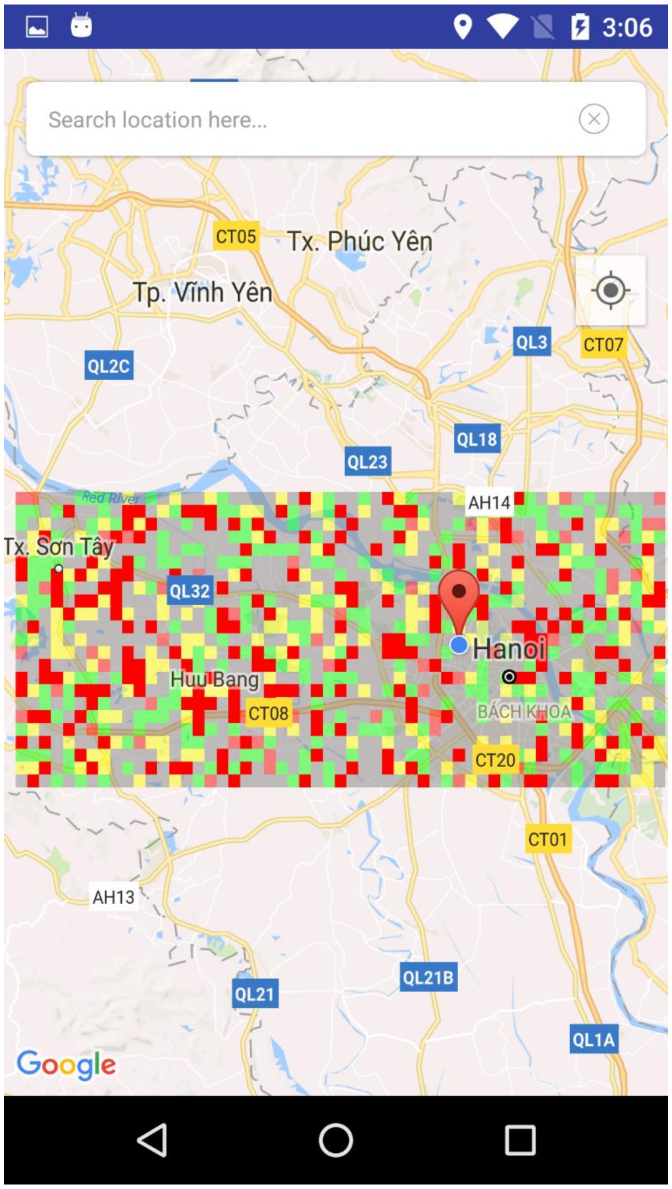


Figure 3.10 show traffic condition user interface

3.2.5 Show route

When users want to find a route between two places, we use google map API to get a list of routes, each route is a list of points. What we need to do is to link those points into complete routes. Google map allows drawing polyline, and we will use polyline to represent the route.

The following code to draw polyline when get routes from google map API.

```
@Override
 public void onRoutingSuccess(List<Route> route, int shortestRouteIndex) {
  mprogress.dismiss();
  CameraUpdate center = CameraUpdateFactory.newLatLng(mOrigin);
  CameraUpdate zoom = CameraUpdateFactory.zoomTo(16);

  mMap.moveCamera(center);
  polylines = new ArrayList<>();
  //add route(s) to the map.
  for (int i = 0; i < route.size(); i++) {

   //In case of more than 5 alternative routes
   int colorIndex = i % COLORS.length;

   PolylineOptions polyOptions = new PolylineOptions();
   polyOptions.color(getResources().getColor(COLORS[colorIndex]));
   polyOptions.width(10 + i * 3);
   polyOptions.addAll(route.get(i).getPoints());
   Polyline polyline = mMap.addPolyline(polyOptions);
   polylines.add(polyline);
  }

  // Start marker
  MarkerOptions options = new MarkerOptions();
  options.position(mOrigin);
  options.icon(BitmapDescriptorFactory.fromResource(R.drawable.start_blue));
  mMap.addMarker(options);
```

```
        // End marker
        options = new MarkerOptions();
        options.position(mDestination);
        options.icon(BitmapDescriptorFactory.fromResource(R.drawable.end_green));
        mMap.addMarker(options);
      }
```

Fig. 3.11 shows the map interface when navigating using the google map API. The route is the color line on the map.

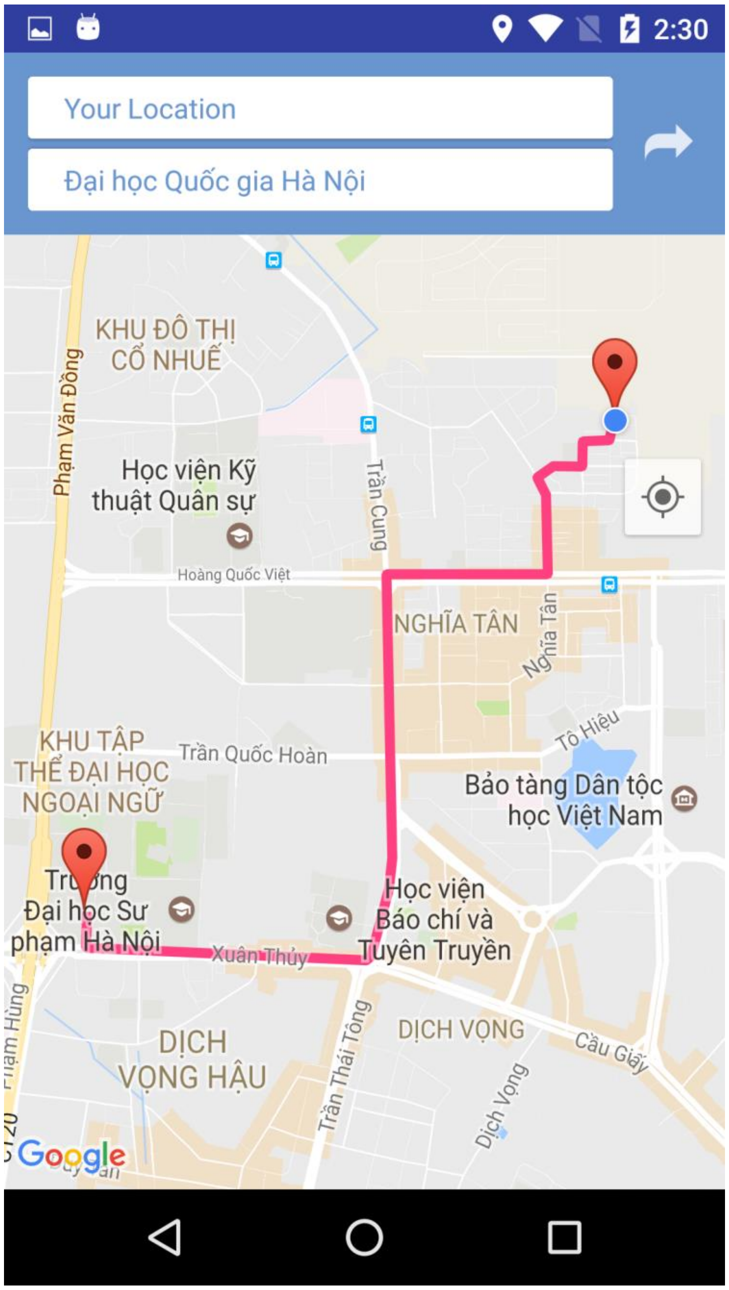


Figure 3.11 show route user interface

3.2.6 Show route avoiding traffic.

As mentioned in 3.2.5, when users want to find a route between two places, we use google map API to get a list of routes, each route is a list of points, then link these points into complete routes. We have to do one more thing, is evaluation these routes by estimating traveling time based on traffic state shown on 3.3.2.1. To do that, we first have to take the weights of the points corresponding to the cells on the map, from which we get the transport speed at the points. Calculating the distance between two points in succession, we will calculate the transition time between the two points, from which to calculate time on the whole route. Here we will describe these steps in detail.

As shown in 3.3.2.1, we use 4 points to cover the whole area of Hanoi:

- 21.157200, 105.456390
- 21.157200, 105.951180
- 20.951180, 105.456390
- 20.951180, 105.951180

The overlay of the map is divided into 56*23 small squares, from this we can calculate the latitude difference between two consecutive points as follows:

$\alpha = (21.157200 – 20.951180) / 23 = 0.0089573$

$\beta = (105.951180 – 105.456390) / 56 = 0.0088355$

From that, we can calculate the coordinates of 4 peaks of each cells. For example, the cell in the top on the left of the map will have 4 peak:

- 21.157200, 105.456390
- 21.157200, 105.4475545
- 21.1482427, 105. 4475545
- 21.1482427, 105.456390

Traffic density is expressed in colors. We define five color codes for four levels of traffic density:

Table 3.3. Color corresponds to speed

Color	Color code	Speed	Average speed
Red	#FF000	<10km/h	5km/h
Yellow	#FFFF00	10-20km/h	15km/h
Green	#00FF00	20-30km/h	25km/h
Blue	#0000FF	>30km/h	40km/h
Grey	#808080	Default:30km/h	Default:30km/h

Then cell point is mark a weight is by its average speed.

Google map API returns a list of routes, each route is a list of points. Each point contain its coordinates and information of the next point. From coordinate of the point, we will know its corresponding cell on the map, then this point is marked the weight of the cell. From the coordinates of the points, we calculate the distance between two consecutive points as follows:

```
float distance = locationA.distanceTo(locationB);
```

And then, the travelling time between these points is:

```
float time = distance / weightA
```

So, the total travelling time of the whole route is:

float totalTime = $\sum$ time

Fig 3.12 shows the map interface when navigating using the google map API. The route is the color line on the colored map showing the traffic density.

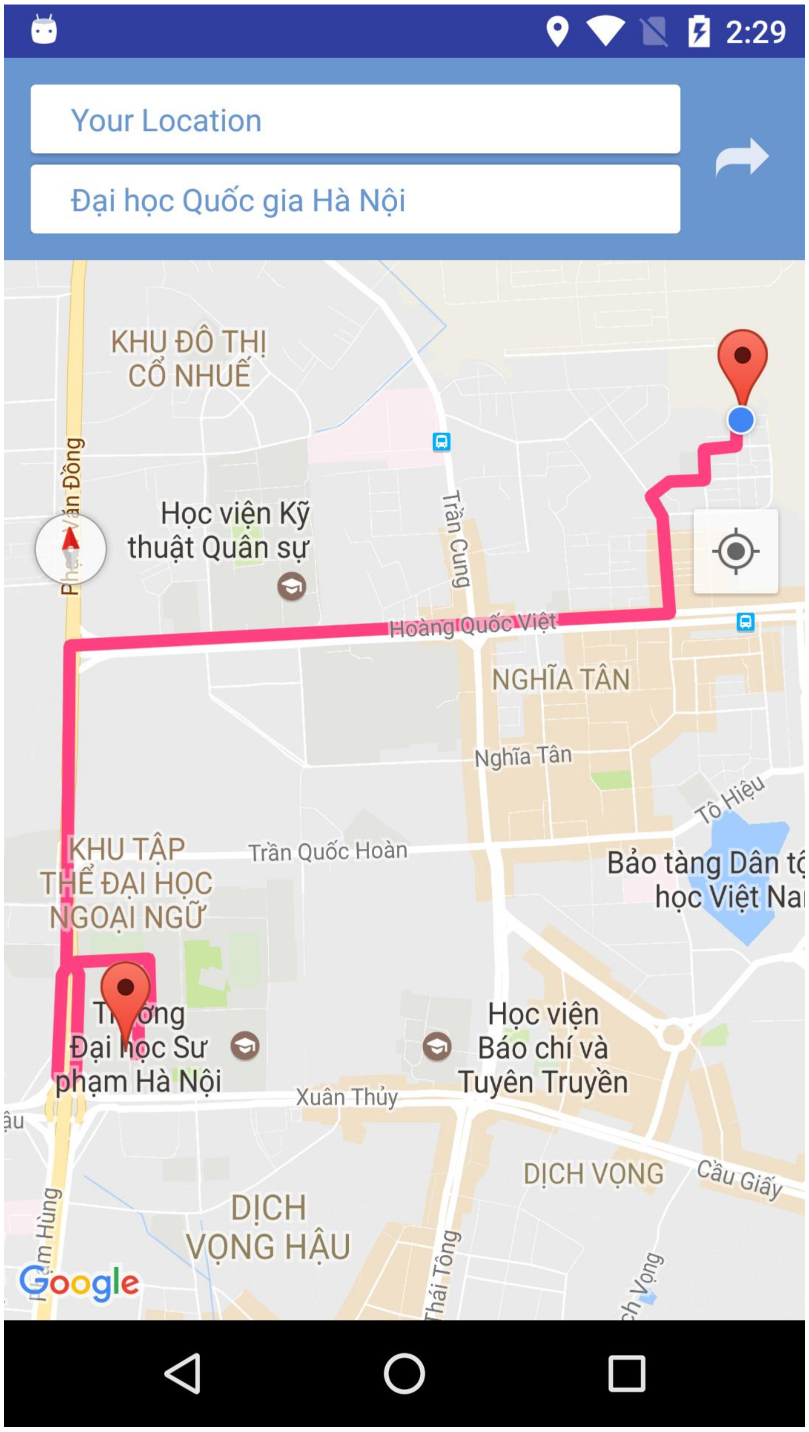


Figure 3.11 Show route avoiding traffic user interface

3.2.7 Show your current location

Fig. 3.12 shows the interface when the user clicks on "Your Current Location" on the left menu bar, including the marker marking the user position and zooming the map to a certain scale

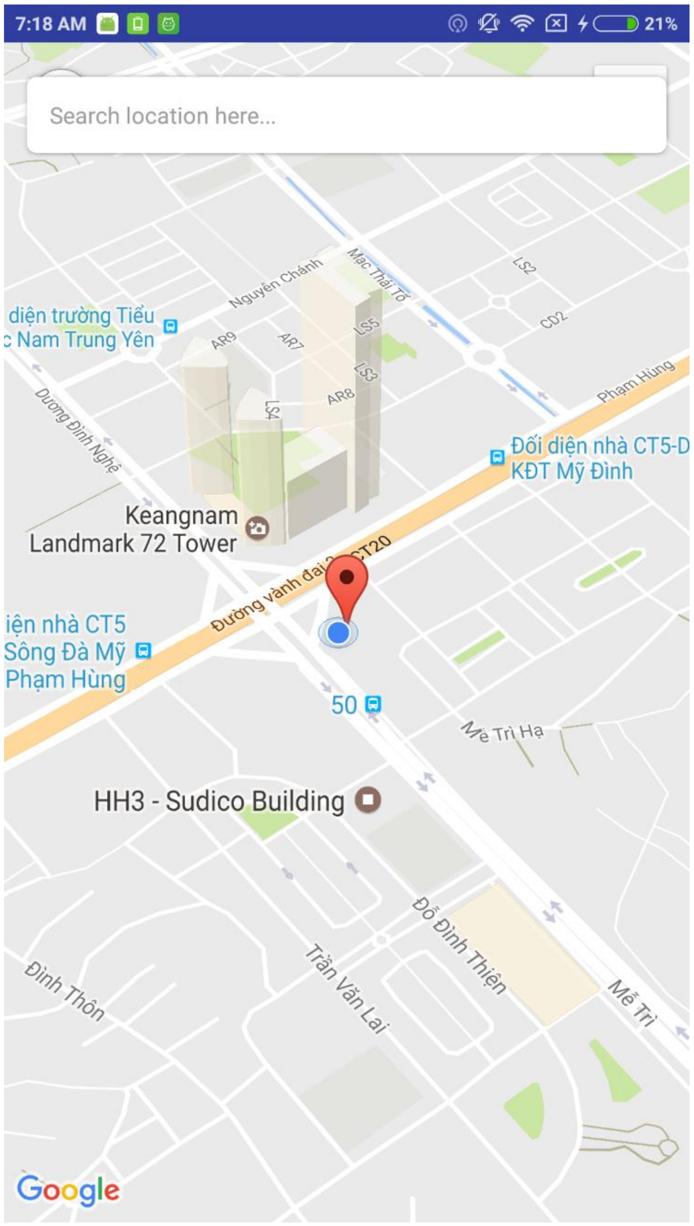


Figure 3.12 show current location user interface

3.2.8 Left menu

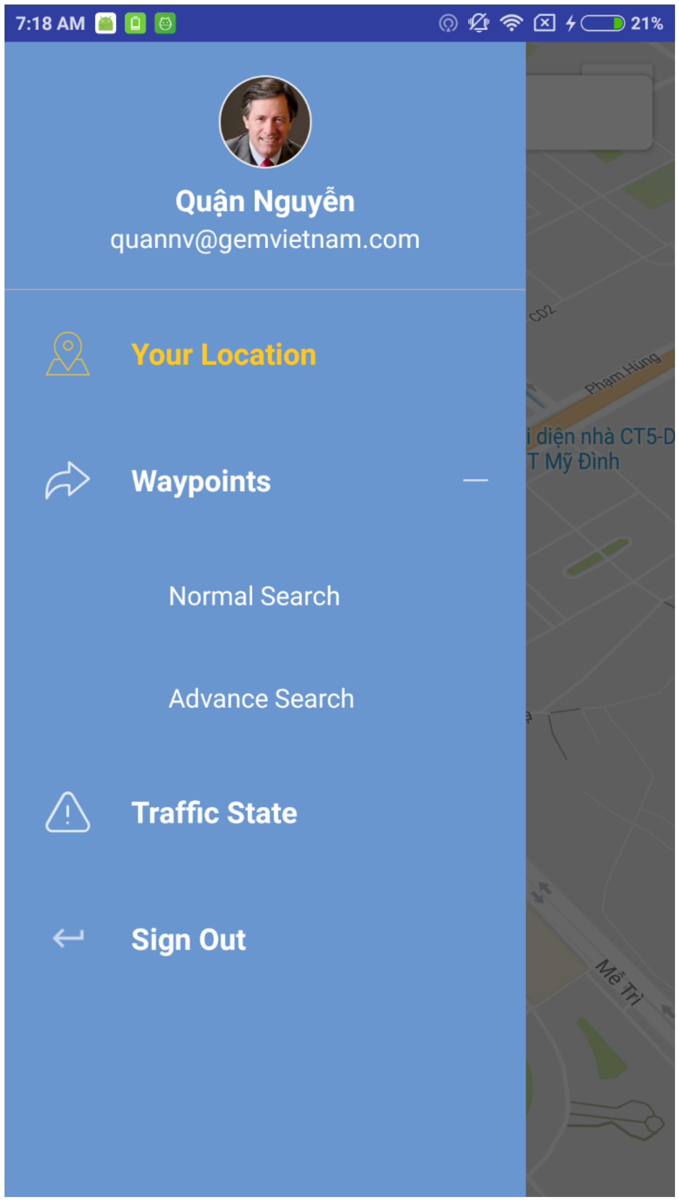


Figure 3.13 left menu user interface

The left menu includes a list of application functions including:

- Show your current location.
- Show shortest path between 2 locations.
- Show path avoiding high-traffic areas.
- Show traffic condition.

Chapter 4

CONCLUSION

In this thesis, a method of urban traffic state estimation has been presented. The proposed method, taking advantage of the recently booming A-GPS mobiles phones, potentially solves the problems (e.g., cost and urban coverage) in the current state-of-the-practice traffic systems. Based on the microscopic traffic simulation and field tests, “realistic” A-GPS mobile probe data is emulated and “ground truth” traffic data is generated. The resultant position/speed estimates are allocated to nearest road links through map-matching. By aggregating the speed estimates on each road link, traffic states (i.e., average link speeds) are determined every 15 minutes. The achieved simulation results suggest that reliable average link speed estimations can be generated, which are used for indicating the real-time urban road traffic condition. Future work targets a smart traffic information system demonstrator that employs the proposed urban traffic state estimation method.

However, due to time and knowledge constraints, we were experimenting on a small-scale database, and the data collection was done on ideal terms. Under actual conditions, the data collection is limited, such as not knowing when to start collecting data, even not knowing the type of users’ transportation, to have a reasonable estimation.

In the coming time, the system needs to be improved to increase accuracy as well as speed of execution. Finding path algorithms as well as parameters need to be studied and applied to increase system stability. Also, consider some of the more powerful algorithms applied to the route evaluation step.

REFERENCE

[1] Google Maps Mobile Users Send Traffic Data.

http://googlesystem.blogspot.com/2009/08/google- maps-mobile-users-send-traffic.html

[2] Y. Wang, M. Papageorgiou and A. Messmer, “Real-Time Freeway Traffic State Estimation Based on Extended Kal- man Filter: Adaptive Capabilities and Real Data Testing,” Transportation Research Part A, Vol. 42, No. 10, 2008,pp. 1340-1358. doi:10.1016/j.tra.2008.06.001

[3] J. Guo, J. Xia and B. Smith, “Kalman Filter Approach to Speed Estimation Using Single Loop Detector Measurements under Congested Conditions,” Journal of Transportation Engineering, Vol. 135, No. 12, 2009, pp. 927-

934. doi:10.1061/(ASCE)TE.1943-5436.0000071

[4] X. J. Ban, Y. Li, A. Skabardonis and J. D. Margulici, “Performance Evaluation of Travel-Time Estimation Methods for Real-Time Traffic Applications,” Journal of Intelligent Transportation Systems, Vol. 14, No. 2, 2010, pp.54-67. doi:10.1080/15472451003719699

[5] R. L. Cheu, C. Xie and D. H. Lee, “Probe Vehicle Population and Sample Size for Arterial Speed Estimation,” Computer-Aided Civil and Infrastructure Engineering, Vol. 17, No. 1, 2002, pp. 53-60.

doi:10.1111/1467-8667.00252

[6] C. Nanthawichit, T. Nakatsuji and H. Suzuki, “Applica- tion of Probe-Vehicle Data for Real-Time Traffic-State Estimation and Short-Term Travel-Time Prediction on a Freeway,” Transportation Research Record, 2003, pp. 49-59. doi:10.3141/1855-06

[7] M. Ferman, D. Blumenfeld and X. Dai, “An Analytical

Evaluation of a Real-Time Traffic Information System Using Probe Vehicles,” Journal of Intelligent Transportation Systems, Vol. 9, No. 1, 2005, pp. 23-34. doi:10.1080/15472450590912547

[8] Y. Chen, L. Gao, Z. Li and Y. Liu, “A New Method for Urban Traffic State Estimation Based on Vehicle Tracking Algorithm,” Proceedings of IEEE Intelligent Transportation Systems Conference, Seattle, 30 September-3 October 2007, pp. 1097-1101. doi:10.1109/ITSC.2007.4357646

[9] R. Cayford and T. Johnson, “Operational Parameters Affecting the Use of Anonymous Cell Phone Tracking for Generating Traffic Information,” Transportation Research Board Annual Meeting, Washington DC, 2003.

[10] B. Hellinga and P. Izadpanah, “An Opportunity Assessment of Wireless Monitoring of Wide-Area Road Traffic Conditions,” Technical Report, 2007, pp. 1-78.

[11] ITU, “Key Global Telecom Indicators for the World Tele- communication Service Sector.” http://www.itu.int/ITU-D/ict/statistics/at_glance/KeyTele com.html

[12] MobiThinking, “Global Mobile Statistics 2011.” http://mobithinking.com/stats-corner/global-mobile-statist ics-2011-all-quality-mobile-marketing-research-mobile-web-stats-su

[13] Y. Yim, “The State of Cellular Probe,” California PATH Research Report, 2003.

[14] M. Fontaine, B. Smith, A. Hendricks and W. Scherer, “Wireless Location Technology-Based Traffic Monitoring: Preliminary Recommendations to Transportation Agencies Based on Synthesis of Experience and Simulation Results,” Transportation Research Record, Vol. 1993, 2007, pp. 51-58. doi:10.3141/1993-08

[15] H. Bar-Gera, “Evaluation of a Cellular Phone-Based Sys- tem for Measurements of Traffic Speeds and Travel Times: A Case Study from Israel,” Transportation Research Re-search Part C, Vol. 15, 2007, pp. 380-391.

[16] J. C. Herrera, et al., “Evaluation of Traffic Data Obtained via GPS-Enabled Mobile Phones: The Mobile Century Field Experiment,” Transportation Research Part C, Vol. 18, No. 4, 2010, pp. 568-583.

[17] B. R. Hellinga and L. Fu, “Reducing Bias in Probe-Based Arterial Link Travel Time Estimates,” Transportation Re- search Part C, Vol. 10, No. 4, 2002, pp. 257-273.

[18] Transport Canada, “Development and Demonstration of a System for Using Cell Phones as Traffic Probes.”

http://www.tc.gc.ca/eng/innovation/tdcsummary-14300- 14359e-1430.htm

[19] Mobile Millennium, “The Mobile Century Field Test.” http://traffic.berkeley.edu/project/mobilecentury

[20] G. Rose, “Mobile Phones as Traffic Probes: Practices, Prospects and Issues,” Transport Reviews, Vol. 26, No. 3, 2006, pp. 275-291. doi:10.1080/01441640500361108

[21] X. Dai, M. Ferman and R. Roesser, “A Simulation Evaluation of a Real-Time Traffic Information System Using Probe Vehicles,” Proceedings of IEEE Intelligent Transportation Systems Conference, 12-15 October 2003, pp. 475-480.

[22] V. Manolopoulos, S. Tao, S. Rodriguez, M. Ismail and A. Rusu, “MobiTraS: A Mobile Application for a Smart Traffic System,” Proceedings of IEEE International NEWCAS conference, Montrea, 20-23 June 2010, pp. 365-368

[23] Google Map

http://whatis.techtarget.com/definition/Google-Maps

[24] Introduction to Developing with Google Maps

http://www.directionsmag.com/entry/introduction-to-developing-with-google-maps/123188

[25] Assisted - GPS

https://www.techopedia.com/definition/24218/assisted-gps-a-gps

[26] Introduction to Android development

https://www.ibm.com/developerworks/library/os-android-devel

[27] 99.6 percent of new smartphones run Android or iOS

http://www.theverge.com/2017/2/16/14634656/android-ios-market-share-blackberry-2016

[28] Smartphone OS Market Share, 2016 Q3

http://www.idc.com/promo/smartphone-market-share/os

[29] Shortest path problem

https://en.wikipedia.org/wiki/Shortest_path_problem

[30] Dijkstra's algorithm

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[31] Bellman–Ford algorithm

https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm

[32] Floyd–Warshall algorithm

https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

[33] M. Fontaine and B. L. Smith, “Probe-Based Traffic Monitoring Systems with Wireless Location Tec Investigation of the Relationship between System Design and Effectiveness,” Transportation Research Record, 2005,pp. 3-11. doi:10.1049/iet-its.2009.0053

[34] World Geodetic System, 1984.

http://en.wikipedia.org/wiki/World_Geodetic_System