TDDD63 Project: Map-based Applications

Fredrik Heintz Tommy Färnqvist

May 20, 2013

1 Introduction

Maps represent the world we live in. They are central in many computer applications. The goal of this project is to create a map-based application including some of the underlying data structures and algorithms.

As we all know, the numbering of the houses in Ryd can be quite confusing. It has been rumored that there is a research report stating that there is no simple set of rules that describe the numbering. To help you find your way to your friends you need a map application that for example shows the shortest path between your house and your friends houses. Since you want to become an expert programmer you decide to write your own application to learn more about the data structures, algorithms, and problems involved in map applications.

In the first part of the project a basic map application is developed which can then be extended in a number of different directions in the second part.

2 Background

Maps have always played a central role in human society. According to Wikipedia, a map is "a visual representation of an area – a symbolic depiction highlighting relationships between elements of that space such as objects, regions, and themes."

The earliest known maps are of the heavens, not the earth. Dots dating to 16,500 BCE found on the walls of the Lascaux caves map out part of the night sky, including the three bright stars Vega, Deneb, and Altair, as well as the Pleiades star cluster.

Computerized maps have gained popularity especially with the introduction of smart phones with GPS receivers. Equipped with a GPS receiver you know the coordinates of your current location, i.e. you know where you are (roughly, depending on the accuracy of the receiver). To make those coordinates much more useful, you also need to know where those coordinates are in the world. In other words, you need a map. With a map displaying your location, according to the GPS, you can do many things such as finding your way to your friend somewhere in Ryd and find the closest pizza place.

There are many popular map-based applications:

- Navigation aids for smart phones and cars.
- Data visualization aids to explore and interact with data such as corruption, crimes and natural resources.

- Use crowd sourcing to monitor elections and support disaster relief, such as Crowdmap by Ushahidi.
- Recreational applications such as Geocaching, FourSquares, and MapMyRun.
- For more examples, see `http://googlemapsmania.blogspot.se`.

Google Maps is probably the most popular online map as well as map API. With their many users Apple might manage to catch up. An interesting fact about Apple's maps is that it is partially developed by the Linköping company C3 Technologies which was acquired by Apple. C3 Technologies is a spin-off company from SAAB that generates accurate 3D maps.

3 Project Assignment: Introductory Phase

The goal of the introductory phase is to create a web application using the Django framework:

- `https://www.djangoproject.com/`,

display a map using the Google Maps API:

- `https://developers.google.com/maps/documentation/javascript/`,

and to explore map data from the Open Street Map project:

- `http://www.openstreetmap.org/`.

This part of the project is structured around the following four milestones that each consists of a series of exercises. Each exercise has a number of tasks that should be completed.

1. Create a web application with Django (which requires Python 2.7)
2. Display a map using the Google Maps API
3. Explore Open Street Map data
4. Display Open Street Map data on a map

When you have reached the last milestone you will have a web application that displays a set of locations from Open Street Map on a map using the Google Maps API.

3.1 Creating a Web Application with Django

The goal of this milestone is to create a basic web application using the Django framework. Django is a Python framework for creating websites. It is very easy to get started with and provides all the functionality needed for this project.

Exercise 3.1.1 (Create a Django Project)

To get started you need to create a Django project. This is done with the help of the `django-admin.py` program. A Django project is a Python package, i.e. a directory of code, that contains all the settings for an instance of Django. This includes database configurations, Django-specific options and application-specific settings.

Task 3.1.1.1 (Load the `tddd63maps` module)

If you are using Solaris, then you need to load a module with Django and other software that you need for this project. The name of the module is `tddd63-maps`. To load it into your current terminal use the command
`module add home/TDDD63/modulefiles/tddd63-maps`.
To automatically load the module every time you create a new terminal execute the command `module initadd /home/TDDD63/modulefiles/tddd63-maps` once.

If you are using your own Linux system then you need to install Django 1.4.1 by yourself (`https://docs.djangoproject.com/en/dev/topics/install/`).

Task 3.1.1.2 (Create the lmap project)

The following command creates a Django project called lmap: `django-admin.py startproject lmap`

Task 3.1.1.3 (Explore project files)

Explore the file structure created by Django and read about what the files are here `http://docs.djangoproject.com/en/1.4/intro/tutorial01/#creating-a-project`.

Task 3.1.1.4 (Start a web server)

To test your application you need to start a web server. Django comes with a development server which you can use. Change into the outer lmap directory, if you haven't already, and run the command `python manage.py runserver 0.0.0.0:80XY`, where X is the group number (1–5) and Y is a number that each group assigns to each student in their group (0–9). The command starts the development web server and informs you about potential problems as well as the URL of the server. To get the URL, find the line "Development server is running at *URL*". Check that your web server is working by opening the URL found in your browser.

Exercise 3.1.2 (Set up an SQLite database)

To store information your application needs a database. SQLite is a simple database which is stored in a single file in the file system.

Task 3.1.2.1 (Update database configuration)

To configure Django to use an SQLite database open the file `lmap/settings.py`, find the variable `DATABASES`, and make `'django.db.backends.sqlite3'` be the value for `'ENGINE'`, and change the value for `'NAME'` to `'/home/`*USER*`/lmap/lmap.sqlite3'` where *USER* is your user name.

An alternative to hard coding the user name is to import the module `os` (`import os`) and set `'NAME'` to `os.environ['HOME']+'/lmap/lmap.sqlite3'`.

Task 3.1.2.2 (Read about the database configuration)

Read more about the database options at `http://docs.djangoproject.com/en/1.4/intro/tutorial01/#database-setup`.

Task 3.1.2.3 (Create database and initial tables)

Your Django project is by default configured to use several database tables. These must be created in the database before we can use them. To do this, execute the command `python manage.py syncdb`.

When asked about creating a superuser, say "yes". Create a superuser "admin" with your email address and the password "admin".

Exercise 3.1.3 (Create a Django app)

A Django app is a Python package, somewhere on your Python path, that follows a certain convention. Django comes with a utility that automatically generates the basic directory structure of an app, so you can focus on writing code rather than creating directories. An app is a reusable Django component that can be used in multiple projects.

Task 3.1.3.1 (Projects vs. Apps)

Read about the difference between a project and an app at `https://docs.djangoproject.com/en/1.4/intro/tutorial01/#creating-models`.

Task 3.1.3.2 (Create the mapvis app)

Change directory to the lmap directory so that you are in the directory with `manage.py`. To create the mapvis app, execute the command `python manage.py startapp mapvis`.

Task 3.1.3.3 (Explore app files)

Explore the file structure created by Django and read about what the files are here `https://docs.djangoproject.com/en/1.4/intro/tutorial01/#creating-models`.

Exercise 3.1.4 (Make your app generate a web page)

The next step towards a web application is to generate your first web page. In Django, and in other frameworks based on the Model View Controller (`http://en.wikipedia.org/wiki/Model-view-controller`) design pattern, this is done through a *view*. To make a page available to a user it has to be associated with a URL. From a Django programming perspective, the content of a page is produced by a view function and a URL is specified in a URLconf.

You can read about views and associating URLs to views here `http://www.djangobook.com/en/2.0/chapter03.html`.

Task 3.1.4.1 (Create a view displaying a message)

In the mapvis app directory, there is a file `views.py`. This is were you should place your view functions. Enter the code below to create a view function that generates a page with the classical message "Hello world".

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello world")
```

If you haven't already, read about what this code does here `http://www.djangobook.com/en/2.0/chapter03.html`.

Task 3.1.4.2 (Connect a URL to your view)

Let us continue by associating a URL with the view function so that you can view the result in your web browser. To do this you have to edit the file `lmap/urls.py`. When you created the lmap project a default files was created. The important part of that file is the variable `urlpatterns`. If you inspect the file, you will see several examples of how to associate URLs with views.

To associate the view you created in the previous task with the relative URL "hello" you need to do two things:

1. Import the view function by adding `from mapvis.views import hello` after the existing import statement.

2. Create a URL pattern by adding the line `('∧hello/$', hello),` after `urlpatterns = patterns('',`.

The file will now look something like this:

```
from django.conf.urls import patterns, include, url
from mapvis.views import hello

# Uncomment the next two lines to enable the admin:
# from django.contrib import admin
# admin.autodiscover()

urlpatterns = patterns('',
    ('^hello/$', hello),
    # Examples:
    # url(r'^$', 'lmap.views.home', name='home'),
    # url(r'^lmap/', include('lmap.foo.urls')),

    # Uncomment the admin/doc line below to enable admin documentation
                                                        :
    # url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    # url(r'^admin/', include(admin.site.urls)),
)
```

Now you can start the development web server and open the page *URL*/hello in your web browser. Congratulations, you have now generated a web page using Python!

If you haven't already, read about what this code does here `http://www.djangobook.com/en/2.0/chapter03.html`.

Task 3.1.4.3 (Create a view that generates a page showing the current time)

It is important to emphasize that the view function you just created is a plain Python function which generates a string, which is interpreted as a HTML page by your browser.

The hello view you created returns a static string. Let us make it a bit more interesting by including the current date and time. This means that every time your reload the page a new string is generated (unless you reload it faster than the resolution of the clock, but then you have to be pretty darn fast).

To achieve this you need to import `datetime`, get the current date and time with the function `datetime.datetime.now()` and then generate an HTML page in the form of a string. Why not try doing this on your own?

When you are done your code should look something like this:

```
from django.http import HttpResponse
import datetime

def hello(request):
    now = datetime.datetime.now()
    html = "<html><body>Hello world! It is now %s.</body></html>" %
                                            now
    return HttpResponse(html)
```

Task 3.1.4.4 (Play around with your view)

If you know some HTML this would be a good time to play around with the view function and elaborate the page generated. For example, make the current time bold. You can also use your Python skills to for example display only the current time or format the date a bit nicer.

A good HTML tutorial is available here `http://www.w3schools.com/html`.

Exercise 3.1.5 (Generate a web page using a template)

In the previous exercise you created a web page by hard coding the content. A much better approach is to use the Django template facility. It allows you to combine hard coded and generated HTML/CSS/JavaScript in more user friendly manner than creating a very long string by concatenating smaller strings.

You can read more about Django's templates here `http://www.djangobook.com/en/2.0/chapter04.html`.

Task 3.1.5.1 (Create a directory for your templates)

You need somewhere to store your templates. The convention is to create a directory called `templates` in the project directory and in it a sub directory for each application with the same name as the application.

Create a directory `templates` in the project directory and a directory `mapvis` in that directory.

Task 3.1.5.2 (Update the project configuration to find your templates)

Now that you have a directory for templates you need to configure your project to find those templates. This is done by editing the file `lmap/settings.py`.

Find the variable `TEMPLATE_DIRS` in the file `lmap/settings.py` and add the absolute path to newly created `templates` directory. Remember that you can use the environment variable `HOME` to avoid hard coding the path for one particular user (which simplifies using the same code by several people). If you have followed the instructions, you should add `os.environ['HOME']+'/lmap/templates',` to the list `TEMPLATE_DIRS`.

Task 3.1.5.3 (Create a view rendering a page from the template)

A template is basically a text file where some parts are interpreted as variables and macros which are expanded to generate several different instances of the text. To generate an instance of a template it is instantiated with a *context* which is basically a Python dictionary.

Create a view called `mapapp` in the file `views.py` with the following code. The code creates a context associating the key `NOW` with the current date and time, instantiates the template `map/mapapp.html` and returns it as a response to an HTTP request.

```
from django.shortcuts import render_to_response
from django.template import Template, Context

def mapapp(request):
    now = datetime.datetime.now()
    c = Context({'NOW': now})
    return render_to_response('mapvis/mapapp.html', c)
```

You also need to associate a URL with the view, this is done by adding `('^mapapp/$', mapapp),` to `urlpatterns` in the file `lmap/urls.py`.

Task 3.1.5.4 (Create a page template)

The template itself is a text file `mapapp.html` in the directory `templates/mapvis`.

To create a page with the same content as the hello view use the following template:

```
<html>
  <body>
    Hello world!  It is now {{ NOW }}.
  </body>
</html>
```

Task 3.1.5.5 (What happens if you open the site root?)

Open the site root. What happens? Why?

Task 3.1.5.6 (Associate the mapapp view with the site root)

Now you should know enough to be able to create a URL pattern that associates the `mapapp` view with the site root. If you need advice, read `http://www.djangobook.com/en/2.0/chapter03.html`.

Task 3.1.5.7 (Play around with views and templates)

To learn more about vies and templates we suggest that you play around with them for a while. Try to create something impressive and show it to the other groups.

3.2 Displaying a Map with the Google Maps API

The goal of this milestone is to extend the web application created to display a map using the Google Maps API. Since this API is written for JavaScript this milestone involves some very simple JavaScript programming. Most of the code needed will be provided for you. As this project shows, most applications require several different languages since each language has its own purpose and application domain.

Exercise 3.2.1 (Learn more about Google Maps)

Take a few minutes to read about Google Maps at `http://developers.google.com/maps/`.

Exercise 3.2.2 (Learn more about the Google Maps JavaScript API)

Take a few minutes to read about the Google Maps JavaScript API version 3 at `http://developers.google.com/maps/documentation/javascript/`.

Exercise 3.2.3 (Get your own Google Maps API Key)

To access the Google Maps API you need a key. Obtain your own key by following these instructions `http://developers.google.com/maps/documentation/javascript/tutorial#api_key`.

Exercise 3.2.4 (Display a map)

Now that you know what the Google Maps API is and have your own API key, you are ready to display your first map.

Task 3.2.4.1 (Import Google Maps JavaScript code)

To display a map, replace the content of the file `templates/mapvis/mapapp.html` with the following HTML, CSS and JavaScript code.

```
<!DOCTYPE html>
<html>
  <head>
    <metaname="viewport"content="initial-scale=1.0, user-scalable=no"/>
    <styletype="text/css">
      html { height:  100% }
      body { height:  100%; margin:  0; padding:  0 }
      #map_canvas { height:  80%; width:  80% }
    </style>
    <scripttype="text/javascript"
      src="http://maps.googleapis.com/maps/api/js?key={{ GMAPS_API_KEY }}&sensor=false">
    </script>
    <script type="text/javascript">
      var map;

      function initialize() {
        var mapOptions = {
          center:  new google.maps.LatLng(-34.397, 150.644),
          zoom:  8,
          mapTypeId:  google.maps.MapTypeId.ROADMAP
        };
        map = new google.maps.Map(document.getElementById("map_canvas"), mapOptions);
      }

      google.maps.event.addDomListener(window, 'load', initialize);
    </script>
  </head>
  <body>
    <divid="map_canvas"></div>
  </body>
</html>
```

To understand what the code above does, take a look at the Google Maps JavaScript API tutorial `https://developers.google.com/maps/documentation/javascript/tutorial`.

Task 3.2.4.2 (Create a view that generates the map page)

To display the map, you also need to update the `mapapp` view function by including your Google Maps API key in the Context so that it is included in the generated web page. The complete code for the view function is:

```
def mapapp(request):
    c = Context({'GMAPS_API_KEY': 'YOUR_GMAPS_API_KEY'})
    return render_to_response('mapvis/mapapp.html', c)
```

Task 3.2.4.3 (Change the map center)

Currently the map is centered on Sydney, Australia. This can be changed. The latitude and longitude of Linköping University is 58.3985303, 15.5745319. To change the center of the map, update the value of the mapOptions property center to `center: new google.maps.LatLng(58.3985303, 15.5745319),`.

Task 3.2.4.4 (Change the zoom level)

When you reload the page you see that the map is centered on Linköping. To zoom in on the university change the zoom level in the template code from 8 to 16.

Task 3.2.4.5 (Change the map type)

There are several types of maps that can be displayed. The following map types are available in the Google Maps API:

- `MapTypeId.ROADMAP` displays the default road map view,
- `MapTypeId.SATELLITE` displays Google Earth satellite images,
- `MapTypeId.HYBRID` displays a mixture of normal and satellite views, and
- `MapTypeId.TERRAIN` displays a physical map based on terrain information.

Try the different types out! In the end we suggest you use either `ROADMAP` or `HYBRID`.

Exercise 3.2.5 (Place a marker with a click on the map)

You now have a web application displaying a map. To make it more interesting you will now allow a user to add markers to the map by clicking on it and displaying the latitude and longitude of that location when the mouse is over the marker.

Task 3.2.5.1 (Create a JavaScript function for displaying a marker)

To display a marker you need to create a new object of the type `google.maps.Marker` with a reference to the map, the latitude and longitude of the marker, and the title of the marker. The title is a string that is displayed when the mouse is placed on the marker.

Copy the following helper function into your template after the line `google.maps.event.addDomListener(window, 'load', initialize);`:

```
function createMarker(map, latlng, title) {
  var marker = new google.maps.Marker({
          map:  map,
          position:  latlng,
          title:  title
  });
}
```

Task 3.2.5.2 (Create a marker on the click of the mouse)

To create a marker when the user clicks the mouse on the map you need to associate a function with the event handler for the event `click`. When a user clicks on the map, a `click` event is generated which contains the latitude and longitude of the position of the click in the event attribute `latLng`.

The event handler is the createMarker function you created above.

To create the event handler, copy the following code to the end of the initialize() function you created before:

```
google.maps.event.addListener(map, 'click', function(event) {
  createMarker(map, event.latLng, "Click:  "+event.latLng);
});
```

Test the functionality by reloading the web page and finding the coordinates of for example Zenit, your current location, and your house.

You now have a basic map application. Pretty nice, right?

3.3 Exploring Data from Open Street Map

Open Street Map (`http://www.openstreetmap.org/`) (OSM) is an open source alternative to Google Maps which is created by volunteers, where many people, including members of Lysator, have contributed. Open Street Map contains data about for example nodes, roads and buildings. A node is a point which is of interest, usually as part of a road or a building.

The goal for this milestone is to download and explore the data in the Open Street Map project for a suitable area of Linköping.

Exercise 3.3.1 (The Linköping part of the Open Street Map project)

Familiarize yourself with the Linköping part of the Open Street Map project

Task 3.3.1.1 (Linköping working group)

The url `http://wiki.openstreetmap.org/wiki/WikiProject_Sweden/Linköping` is the homepage of the Linköping part of the OSM project. Follow the link and investigate how this group structures their work.

Task 3.3.1.2 (A map of Linköping)

The following url `http://www.openstreetmap.org/index.html?lat=58.40&lon=15.61&zoom=12` leads to a visual representation of the OSM data for the Linköping area, i.e. a map! Spend enough time exploring the map to be able to form an informed decision regarding whether the OSM data seems to be of high quality or not. In particular, it might be good to compare this map to the corresponding map area in Google Maps (hint: change the center of the map and the zoom level in your web application to the same values as above and reload your application).

Exercise 3.3.2 (Gathering map data)

Download a suitable portion of the OSM data for Linköping.

Task 3.3.2.1 (Fetch OSM data from the web)

Use your web browser to go to `http://www.openstreetmap.org/index.html?lat=58.40&lon=15.61&zoom=12` and click on “Export” in the menu on top of the page.

Task 3.3.2.2 (Choose an area)

Choose minlat="58.3773000", minlon="15.5207000", maxlat="58.4236000", maxlon="15.6248000", and “OpenStreetMap XML-data”.

Task 3.3.2.3 (Export the data)

Press “Export”. You will now receive a file containing roughly 11 MB of map data around which you will build your map application. Be sure to always keep an untouched version of this file backed-up somewhere and always work on a copy of this file.

Exercise 3.3.3 (A first look at the OSM data)

In this exercise you will take a first look at the OSM data.

Task 3.3.3.1 (Actually look at the data)

Open the OSM data file you have just downloaded in a text-editor of your choice.

Task 3.3.3.2 (XML)

The data is in XML format. XML is a way to present structured data so that is is readable both by humans and machines. Read the Wikipedia page about XML (`http://en.wikipedia.org/wiki/XML`) to learn the basics about how this format works.

Task 3.3.3.3 (OSM XML)

OSM has its own particular set of XML tags and rules for how to use them. The web page `http://wiki.openstreetmap.org/wiki/OSM_XML` contains specifications of the basic tags as well as links to their intended usage. Study this page. In particular, it is important that you learn how the data primitives `nodes`, `ways`, and `relations` are used to build the geographical information contained in a map.

Task 3.3.3.4 (Which information does a `node` contain?)

Find out what information a `node` contains.

Task 3.3.3.5 (Which information does a `way` contain?)

Find out what information a `way` contains.

Task 3.3.3.6 (Which information does a `relation` contain?)

Find out what information a `relation` contains.

Task 3.3.3.7 (How do you represent a road using OSM XML data?)

Find out how to represent a road using OSM XML data.

Task 3.3.3.8 (How do you represent a building using OSM XML data?)

Find out how to represent a building using OSM XML data.

Exercise 3.3.4 (Using web tools to investigate OSM data)

In this exercise you will use the web tool osmhack to investigate OSM data.

The web page `http://toolserver.org/~kolossos/osm/osmhack.php?lat=58.40&lon=15.61&name=Linköping` automatically generates a list of the OSM map features used in the Linköping map. Clicking a link in the “Values” column generates a map with all map elements carrying the corresponding map feature highlighted in red.

Task 3.3.4.1 (Motorways)

Click "Motorway". Which major road is highlighted in red?

Task 3.3.4.2 (Roads and streets)

Investigate your map data by checking which map features of roads and streets might be interesting for your application to utilize.

Task 3.3.4.3 (Campus Valla)

In particular, which features do the roads, cycleways and foot paths on Campus Valla put to use?

Exercise 3.3.5 (Using Python tools to parse and investigate OSM data)

We now want you to start processing the OSM data by executing programs you have written yourself. Parsing the rather extensive OML XML format is not an easy task. You will therefore use a ready made parser for at least the introductory phase.

The OSM parser is part of the `imposm.parser` Python library (`http://dev.omniscale.net/imposm.parser/`). This library provides functionality to efficiently parse OpenStreetMap XML data and exposes a very simple API to access the data parsed. The library is installed on the Solaris system; if you use your own computer, you need to follow the instructions on the library web page to install it yourself.

Task 3.3.5.1 (Python code)

The following Python snippet uses the `imposm.parser` libray to count the number of nodes in the file `map.osm`:

```python
from imposm.parser import OSMParser

# simple class that handles the parsed OSM data.
class NodeCounter(object):
    counter = 0

    # callback method for coords
    def count(self, coords):
        self.counter += len(coords)

# instantiate counter and parser and start parsing
nodes = NodeCounter()
p = OSMParser(coords_callback=nodes.count)
p.parse('map.osm')

# done
print nodes.counter
```

Hint: You might have to change the directory and file name of the OSM map file to make the code work.

Task 3.3.5.2 (How many nodes does your Linköping map data have?)

Find out how many nodes your Linköping map data contains by writing a program that counts them.

Task 3.3.5.3 (Learn more about the parser)

Read the `imposm.parser` documenation at `http://dev.omniscale.net/imposm.parser/` and write Python programs to answer the following questions.

1. How many untagged nodes are there in your OSM data?
2. How many `ways` are there in your OSM data?
3. How many `relations` are there in your OSM data?
4. How many `ways` that are tagged as highways are there in your OSM data?
5. How many buildings are there in your OSM data?

Task 3.3.5.4 (Your own investigations)

Try to come up with at least 3 questions of your own about the map data and answer them by using the `imposm.parser` library.

3.4 Displaying Open Street Map Data

Now you will put together all the pieces you have learned so far by extending the web application to display locations from Open Street Map on the map.

The goal of this milestone is to create a simple data structure for storing locations (nodes), populate the data structure with data from the Open Street Map project, and display all the nodes on the map.

Exercise 3.4.1 (Create a data structure to store locations)

At the moment, all we want to be able to do is to keep all nodes with their longitude and latitude in memory. To do this, we represent nodes with a simple Python class holding the node id and its associated longitude and latitude:

```
# A simple node class
class Node:
    def __init__(self, id, lng, lat):
        self.id = id
        self.lng = lng
        self.lat = lat
```

The data structure to store all the nodes will now simply be a dictionary using the node id as key. Using the `osm.parser` to parse our map data we get the following Python code:

```
from imposm.parser import OSMParser

# This class reads an OSM file and stores its nodes in memory
class StoreNodes(object):

    def __init__(self, osmfile):
        # parse the input file and save its contents in memory

        # node ids and coord pairs as returned from imposm
        self.nodes = dict()

        # actual min and max latitude and longitude of coordinates
        self.bounds = dict()
        self.bounds["min_lat"] = 9999
        self.bounds["max_lat"] = -9999
        self.bounds["min_lng"] = 9999
        self.bounds["max_lng"] = -9999

        p = OSMParser(coords_callback = self.coords_callback)
        p.parse(osmfile)

    def coords_callback(self, coords):
        for osmid, lng, lat in coords:
            node = Node(osmid, lng, lat)
            self.nodes[osmid] = node
            self.bounds["min_lat"] = min(self.bounds["min_lat"], lat)
            self.bounds["min_lng"] = min(self.bounds["min_lng"], lng)
            self.bounds["max_lat"] = max(self.bounds["max_lat"], lat)
            self.bounds["max_lng"] = max(self.bounds["max_lng"], lng)
```

Task 3.4.1.1 (Use the `StoreNodes` class)

Write Python code that uses the `StoreNodes` class to read your OSM data and then print out all nodes that are stored in memory.

Exercise 3.4.2 (Only use nodes in a particular rectangular area)

In the map application we will need to draw all nodes in the rectangular area currently visible in the map window. This operation may have to be done multiple times, which is why we need a class holding only the nodes inside this particular rectangle. The following Python snippet creates an object holding only the nodes fitting that description:

```
# A class for storing nodes within a rectangular area
class ClipNodes:
    def __init__(self, nodes, minlat, maxlat, minlng, maxlng):
        # node ids and coord pairs
        self.nodes = dict()

        # min and max latitude and longitude
        self.minlat = minlat
        self.maxlat = maxlat
        self.minlng = minlng
        self.maxlng = maxlng

        # actual min and max latitude and longitude of coordinates
        self.bounds = dict()
        self.bounds["min_lat"] = 9999
        self.bounds["max_lat"] = -9999
        self.bounds["min_lng"] = 9999
        self.bounds["max_lng"] = -9999

        for node in nodes.values():
            if self.minlng < node.lng and node.lng < self.maxlng and
                                        self.minlat < node.lat
                                        and node.lat < self.
                                        maxlat:
                new_node = Node(node.id, node.lng, node.lat)
                self.nodes[node.id] = node
                self.bounds["min_lat"] = min(self.bounds["min_lat"],
                                            node.lat)
                self.bounds["min_lng"] = min(self.bounds["min_lng"],
                                            node.lng)
                self.bounds["max_lat"] = max(self.bounds["max_lat"],
                                            node.lat)
                self.bounds["max_lng"] = max(self.bounds["max_lng"],
                                            node.lng)
```

Task 3.4.2.1 (Use the `ClipNodes` class)

Write Python code to get hold of, and print out, only the nodes within the rectangular area given by

```
minlat = 58.3984, maxlat = 58.3990, minlng = 15.5733, maxlng = 15.576
```

Exercise 3.4.3 (Write unit tests for the `ClipNodes` class)

When you develop software it is essential to *test* that it does what it is supposed to do. Especially when you develop data structures and algorithms creating good test cases is both easy and could save a lot of time. To test a single unit of a program, such as a function or a class, is called *unit testing*. In this exercise you will implement unit tests for the `ClipNodes` class.

Task 3.4.3.1 (Learn more about unit testing)

Read the Wikipedia article on unit testing `http://en.wikipedia.org/wiki/Unit_testing`.

Task 3.4.3.2 (Create a simple unit test Using Python's `unittest` package)

One of many advantages of Python is that it has a built in unit testing framework which is called `unittest`. Read more about it here `http://docs.python.org/library/unittest.html`.

Here is an initial unit test for the `ClipNodes` class.

```
import random
import unittest
from nodes import *

def node_inside(node, min_lat, max_lat, min_lng, max_lng):
    return (min_lat <= node.lat and node.lat <= max_lat
            and min_lng <= node.lng and node.lng <= max_lng)

class TestClipNodes(unittest.TestCase):

    def test_empty_nodes(self):
        nodes = dict()
        selected_nodes = ClipNodes(nodes, -10, 10, -10, 10)
        self.assertEqual(nodes, selected_nodes.nodes)

    def test_empty_clip_nodes(self):
        nodes = dict()
        nodes[1] = Node(1, -10, -10)
        nodes[2] = Node(2, -5, -5)
        selected_nodes = ClipNodes(nodes, 0, 10, 0, 10)
        self.assertFalse(selected_nodes.nodes)

    #def test_some_nodes_inside(self):

    #def test_all_nodes_inside(self):

    def test_random(self):
        # Create 100 random nodes
        nodes = dict()
        for i in range(100):
            node = Node(i, random.randint(-180, 180),
                           random.randint(-90, 90))
            nodes[i] = node

        # Randomly choose the min and max latitude and longitude
        min_lat = random.randint(-90, 90)
        min_lng = random.randint(-180, 180)
        max_lat = min_lat + random.randint(0, 90-min_lat)
        max_lng = min_lng + random.randint(0, 180-min_lng)

        # Extract all nodes within the randomly generated rectangle
        selected_nodes = ClipNodes(nodes, min_lat, max_lat,
                                   min_lng, max_lng)

        # Make sure that all nodes that should be inside are inside
        for n in nodes.values():
            if n in selected_nodes.nodes.values():
                self.assertTrue(node_inside(n, min_lat, max_lat,
                                            min_lng, max_lng))
            else:
                self.assertFalse(node_inside(n, min_lat, max_lat,
                                             min_lng, max_lng))


if __name__ == '__main__':
    unittest.main()
```

The code assumes that `ClipNodes` is in the file `nodes.py`, change this if necessary.

Run the unit test by executing the Python script (`python nodes_test.py`). Make sure that your existing codes passes the three test cases.

Task 3.4.3.3 (Implement the `test_all_nodes_inside` test case)

In the code above the test case `test_all_nodes_inside` is not implemented. To implement it, add the following code:

```
def test_all_nodes_inside(self):
    nodes = dict()
    nodes[1] = Node(1, -10, -10)
    nodes[2] = Node(2, -5, -5)
    selected_nodes = ClipNodes(nodes, -10, 10, -10, 10)
    self.assertTrue(nodes == selected_nodes.nodes)
```

Run the updated unit test. Why does it fail? What is wrong, the implementation of `ClipNodes` or the test case? Fix the problem and run the unit tests to make sure your code works.

Task 3.4.3.4 (Implement the `test_some_nodes_inside` test case)

Using what you have learned so far, implement the test case `test_some_nodes_inside` and make sure all unit tests are passed by your code.

Task 3.4.3.5 (Add unit tests for the `Node` class)

Now that you know how to create unit tests create one for the `Node` class. There should be for example be tests to make sure that the latitudes and longitudes are always within their bounds. Create both hand made test cases as well as randomly generated test cases. You can read more about latitudes and longitudes here `http://en.wikipedia.org/wiki/Geographic_coordinate_system`.

If you want another concrete example of a unit test take a look here `http://www.diveintopython.net/unit_testing/`.

Exercise 3.4.4 (Extend the map application to display all locations)

To display nodes from OSM as markers on the map you need to select an area, extract those nodes you would like to display, and call createMarker for each of those. This is done in two steps, extend the view and then extend the template.

Task 3.4.4.1 (Extend the view to include a set of markers)

The following code parses the file `linkoping_map.osm` in the same directory as the view code, extracts all nodes in a rectangle corresponding to the area between the E-house and the physics building, and adds the list of nodes to the context used to render the map.

```
def mapapp(request):
    data = StoreNodes(os.environ['HOME']+'/lmap/mapvis/linkoping_map.
                                         osm')
    nodes = ClipNodes(data.nodes, 58.3984, 58.3990, 15.5733, 15.5760)
    c = Context({'GMAPS_API_KEY': 'YOUR_GMAPS_API_KEY',
                 'COORDS': nodes.nodes.values()},)
```

Task 3.4.4.2 (Extend the template to display a set of markers)

In the previous task you created a list of node objects and associated them with the identifier `COORDS` in the context. To display each of these nodes as a marker you can use the excellent Django template functionality. Basically, you can use simplified Python code to loop over a list and extract the attributes of each object. The following Django template code generates a JavaScript function to display all the nodes associated with `COORDS` as markers. Include the code last in the JavaScript section of the `mapapp.html` template.

```
function displayMarkers(map) {
  {% for coord in COORDS %}
    createMarker(map, new google.maps.LatLng({{coord.lat}},
                                             {{coord.lng}}),
                                             ""+{{coord.id}});
  {% endfor %}
}
```

To display the markers you also need to call the function (`displayMarkers(map);`) in the initialize() function, preferably at the end.

4 Project Assignment: Main Phase

Congratulations, you have now learned how to create a web application displaying a map, how to extract data from the Open Street map project, store it in a data structure and display it in your web application. Equipped with this, you will now create more advanced map-based web applications.

The first step is to extend your web application to assist a user to find her way between two locations. This is the basis for direction and route finding applications such as those provided by Google and Apple.

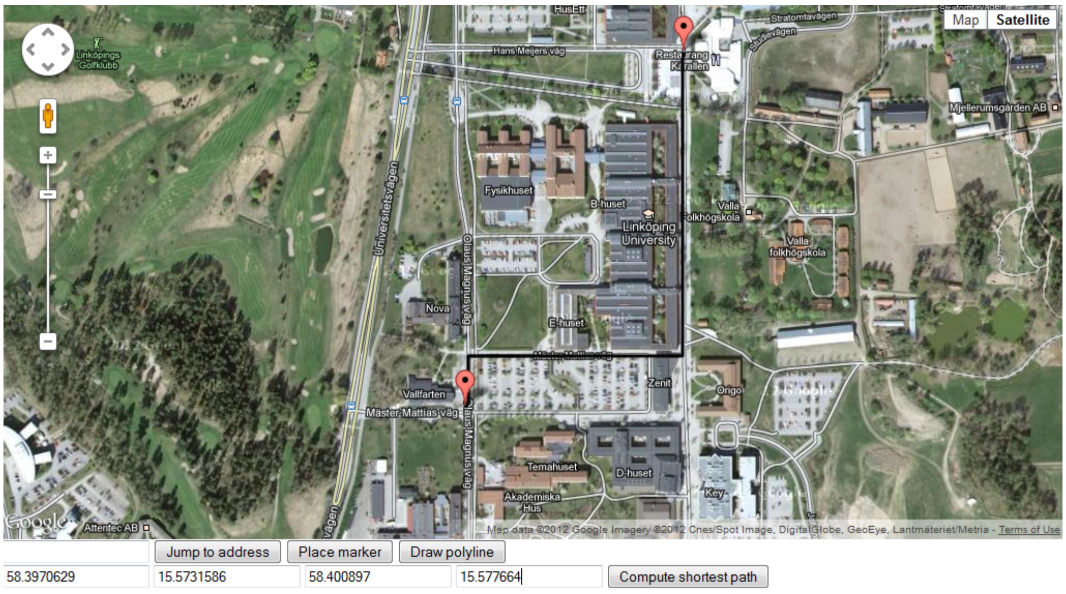
4.1 Assisting a User to Get From A to B

The first goal of this part of the project is to develop a web application that assists a user to get from a location A to a location B. To achieve this, the application should allow a user to select two locations on a map, compute the shortest path between these locations and display the path on the map. To compute the shortest path you also need to create a data structure storing information about which locations are connected and the distances between them as well as populate it with real map data. At the end of this part of the project you will have an application similar to the one in Figure 1.

Milestones:

1. Extend the map application to store and display roads.

   (a) Create a data structure to store roads and appropriate unit tests. Make sure that your code passes all the unit tests. A suggestion is to create a data type called `Road` that stores a single road. The simplest version is to store the id of the road and a list of node ids, just like the OSM data. The class could look like this:



58.3970629 15.5731586 58.400897 15.577664

Figure 1: The web interface of the application assisting users to get from A to B.

```
# A simple road class
class Road:
    def __init__(self, id, nodes):
        self.id = id       # osm id from file
        self.nodes = nodes # list of node ids
```

(b) Populate the data structure with roads from Open Street Map. The following snippet contains a suggested way to structure the code. You will have to fill in the missing pieces of code.

```
# This class reads an OSM file and stores its nodes in memory
class StoreRoads:
    def __init__(self, osmfile):

        # parse the input file and save its contents in
                                            memory

        # road ids and node refs as returned from imposm
        self.roads = dict()

        p = OSMParser(ways_callback = self.ways_callback)
        p.parse(osmfile)

    def ways_callback(self, ways):
            # fill in the missing code
```

(c) Extend the map application to display all roads. To display all roads, we suggest that you create a new data type, or update the `Road` data type, to represent a road as a sequence of coordinates where each coordinate has at least a latitude and a longitude value. You can then store all roads as a sequence of such road objects. By having this structure it becomes straight forward to create a Django template for iterating through all roads and for each road create a polyline by iterating through all coordinates and adding each coordinate to the polyline. To help you, we provide the following example JavaScript code:

```
function createPolyline(map, path) {
  var polyline = new google.maps.Polyline({
      path:  path,
      map:  map
  });
}

function displayRoads(map) {
  var points = new google.maps.MVCArray();
  points.push(newgoogle.maps.LatLng(58.3970323, 15.5738155));
  points.push(newgoogle.maps.LatLng(58.3976588, 15.5738047));
  createPolyline(map, points);
}
```

2. Implement a shortest path algorithm together with appropriate unit tests. Now might be a good time to try a test-driven development methodology (see `http://en.wikipedia.org/wiki/Test-driven_development`), where you start by defining the test cases and then develop the code until it passes all the test cases. It is of course possible to start with a few initial test cases which are then extended together with the code they test.

   (a) Create a data structure to store connections between locations and the lengths of these connections. The usual way to model interconnected locations that are a certain distance apart is by using a (combinatorial) graph. Read up on:

   - the mathematical concept of a graph (`http://en.wikipedia.org/wiki/Graph_(mathematics)`)
   - typical data structures used to represent graphs (`http://en.wikipedia.org/wiki/Graph_(data_structure)`)

   You now have to make a preliminary decision concerning how you want to store the distance graph. We suggest that you make your choice between using

   - adjacency lists, or (`http://en.wikipedia.org/wiki/Adjacency_list`)
   - an adjacency matrix (`http://en.wikipedia.org/wiki/Adjacency_matrix`).

   (b) Implement your chosen data structure in Python and make sure you have appropriate test cases.

   (c) Populate the data structure with data from Open Street Map. To do this you will have to decide how to calculate the geographical distance between two nodes that are part of a road. Since Earth is not flat, the distance between two points on the map cannot simply be calculated as the two dimensional Euclidean distance. One way is to use the haversine formula, as in the following Python code snippet:

```
from math import sqrt, radians, sin, cos, asin

def length_haversine(p1, p2):
        # calculate distance using the haversine formula,
        # which incorporates earth curvature
        # see http://en.wikipedia.org/wiki/Haversine_formula
        lat1 = p1.lat
        lng1 = p1.lng
        lat2 = p2.lat
        lng2 = p2.lng
        lat1, lng1, lat2, lng2 = map(radians, [lat1, lng1,
                                  lat2, lng2])
        dlng = lng2 - lng1
        dlat = lat2 - lat1
        a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlng
                                  /2)**2
        c = 2 * asin(sqrt(a))
        return 6372797.560856 * c # return distance in m
```

(d) Implement a provable optimal algorithm to compute the shortest path between two locations using the graph data structure together with appropriate test cases (you do not have to prove that it is optimal). You should write your own code implementing and testing an existing algorithm, There are many algorithms for finding the shortest path between two nodes in an edge weighted graph. For this project we suggest you start out by implementing Dijkstra's algorithm (`http://en.wikipedia.org/wiki/Dijkstra's_algorithm`).

3. Extend the map application to find and display the shortest **distance** between two locations.

   (a) Extend the application to allow entering the coordinates of the start and destination by clicking on the map. To achieve this you have to do at least the following:

      i. Create an HTML form with at least four fields, start latitude, start longitude, destination latitude and destination longitude. Read about HTML forms here `http://www.w3schools.com/html/html_forms.asp`.

      ii. Create an on click event handler that fills in the fields in the form when a user clicks on the map. Since the user needs to select two coordinates you have to find a way to do that (one approach is to have a very simple state machine with two states, one for entering the start coordinate and one for entering the destination coordinate). In the introductory part you created an on click event handler so the new thing is to populate a form with that data. One page discussing this is `http://www.javascript-coder.com/javascript-form/getelementbyid-form.phtml`.

      iii. Process the content of the form in the view. This is quite straight forward as Django provides `request.POST` which is a dictionary containing all posted variables and their values.

iv. To prevent cross site request forgery (CSRF) Django requires you to take some measures that are described here `http://docs.djangoproject.com/en/dev/ref/contrib/csrf/`. Make sure that you follow these instructions.

(b) Since a user might click anywhere on the map, you have to implement an algorithm that finds the node in the map database closest to a given coordinate. An initial approximation can be to find the node which has the closest distance to the selected coordinate. Luckily, you have already implemented function that computes the distance between two coordinates represented as a pair of latitude and longitude. A second approximation could be to find the N closest nodes and make a somewhat smart selection among these. A third approximation is to find the closest point on the edges between the nodes (and implicitly add a node there). This is very useful if there are long road segments and the selected location is right in the middle of that segment.

As always, you should also create appropriate test cases.

(c) Extend the map application to display the start and destination with for example markers.

(d) Extend the application to show the computed distance somewhere.

4. Extend the map application to find and display the shortest **path** between two locations.

   (a) Extract the shortest path from the output of your shortest path algorithm. Remember to extend the unit tests by adding new test cases.

   (b) Extend the application to display the computed shortest path on the map using, for example, the code developed as part of the previous milestones.

5. Most applications that provide directions give the user some alternatives to choose from. Extend the map application to find alternative routes between two locations.

   (a) The simplest approach is probably to extend the shortest path algorithm to compute the X shortest paths and display these. Does this provide good alternatives?

   (b) A more interesting approach could be to compute the X shortest paths and then find the Y paths among these that share the fewest number of nodes.

   (c) Can you find a better approach to compute alternative paths?

4.2 Extending Your Application Further

After you have completed all these tasks, you have accomplished a lot. You have created and tested a graph data structure, populated it with data from the Open Street Map project, implemented and tested a shortest path algorithm, created a web application where a user can select multiple locations (and map these locations into nodes in the graph), displayed single and multiple paths in the web application, and more.

What you have done so far provides you with both a set of components for more advanced map-based applications as well as an understanding of some of the basic issues involved in map-based applications. Now you will extend your application further or develop new applications starting from the code you already have. This is a very open ended project with many possibilities to create exciting map-based applications. Below are some example suggestions that you can chose from and extend as you like. You are also welcome to suggest your own improvements.

- Smaller improvements to the shortest path application:
  - Add speed to the edges and let the user ask for the fastest way rather than the shortest. Can you provide better choices by minimizing the number of crossings that the user needs to pass?
  - Improve the efficiency of the application by for example adding a bounding box to the roads to filter out as many of them as possible before drawing them on the map. If you do this, then it is a good idea to measure how long it takes to create the page so that you see that you are making progress.
  - Add a search box so that a user can enter an address which is looked up using the Google Geocoding API (`https://developers.google.com/maps/documentation/geocoding/`).
  - Compare the shortest paths you find with the ones provided by the Google Directions API (`https://developers.google.com/maps/documentation/directions/`).
  - Add support for buildings. This would involve at least parsing the buildings from the OSM data, storing them in your application and displaying them on the map. You could also add further information so that when the mouse is hovering over a building you display some interesting piece of information about it.
- Currently all the data is stored in memory and if you reload the page then all the data is loaded again. Extend the application to store all the data in a database instead. This makes the data persistent and potentially improves the efficiency of the application. This is also a good exercise since all modern web applications are database driven. To get started we suggest that you read about Models in the Django book (`http://www.djangobook.com/en/2.0/chapter05.html`). Try to move as much functionality into the database as possible, for example by replacing ClipNodes with a database query.
- Create an application which lets users record their current or planned future location. This can be used to see where your friends were last seen and potentially where to look for them. Maybe you can connect the central schedule database to your application to show where you should be depending on what courses you take?

- Extend the application by collecting information from the user. Initially it could be to only ask for the current position of the user. Maybe you are even able to get that information from the GPS on the users device?

- Extend the application to allow a user to select a set of locations and find the shortest path between each pair of locations.

    1. Extend the map application to allow a user to select a set of coordinates.
    2. Implement an all-pairs shortest path algorithm using the previously developed graph data structure including the API and the unit tests.
    3. Extend the map application to display all the paths.

- You have been given the task of building a beer pipeline by connecting plastic tubes to provide all your friends with beer directly from HG. Since you are thirsty you want to minimize the total amount of plastic tubes that you have to connect before finishing the project. Extend the map application to allow a user to select a set of end points and then find the edges along which the pipeline should be built so that the total length of the pipeline is minimized and display the result on the map. You can make this problem easier or harder depending on how much freedom you consider to be allowed when constructing the pipeline. The simpler version is to only build the pipeline along existing roads. To make the problem harder, and the total length of the pipeline shorter, you can allow new nodes to be added. If you allow new nodes then you can start by ignoring buildings and then add these later. The more accurate you want the result, the more complicated it becomes.

- Your friend, an enthusiastic geocacher, has heard that you are building awesome map-based applications. Since he is a bit lazy he has asked you to extend your application to allow him to select a set of geocaches and then automatically find the shortest route from his apartment visiting all the caches and then back to the apartment.

In any case you can either focus on developing the necessary algorithms, the web application or some combination of them. Remember to also implement the appropriate testing to increase the quality of your code and hopefully decrease the development time. If you decide to work more on the web application rather than the underlying algorithms you may use existing code libraries such as Google's APIs or graph libraries (for example `http://networkx.lanl.gov`, `http://igraph.sourceforge.net`).