Основы реализации географических онлайн-карт

В. В. Осокин, Н. Е. Горожанин, В. Р. Вавилов, Д. У. Камилов

В данной статье описывается процесс создания веб-приложения, реализующего функционал географической карты. Материал делится на две части. В первой части решаются проблемы отрисовки карты на экране, масштабирования карты, перемещения по карте, отображения объектов на карте. Во второй рассматривается задача кластеризации заранее определенного множества объектов, расположенных на карте, методом k-средних и методом расстояний.

Ключевые слова: онлайн-карта, одностраничные приложения, тайлы, php, jQuery, ajax, SQL, метод k-средних, метод расстояний, кластеризация.

Введение

В последнее десятилетие в силу популярности и доступности интернета актуальным направлением стало создание онлайн карт, доступных любому пользователю через браузер. В данной работе реализуется онлайн-карта Москвы с расположенными на ней обьектами (магазины, кафе, рестораны). Пользователь имеет возможность передвигаться по карте, переходить между тремя уровнями зума, переходить к таблице со списком обьектов и с ее помощью находить любой обьект на карте.

С развитием инфраструктуры городов увеличивается и детализация карт. Сейчас в одном городе могут быть тысячи торговых точек, сотни остановок, десятки площадей и многое другое. Предположим, мы захотим выделить такие обьекты на карте, допустим, обозначив их точками. Тогда при уменьшении масштаба карты в некоторой области экрана необходимо отрисовать огромное количество

этих точек. Это порождает две проблемы. Во-первых, точки могут находиться настолько близко, что начнут перекрывать друг друга. Во-вторых, отображение большого количества обьектов нагружает страницу. Для того, чтобы избежать этого, в данной работе реализуется кластеризация объектов на карте методом k-средних и методом расстояний.

Цель данной работы — дать описание основных приемов, которые можно использовать при создании онлайн-карт. Данная работа входит в цикл работ, рассказывающих основы построения современных онлайн-сервисов, таких как система классификации изображений и музыкальных файлов [1] и поисковая система [2]. Среди наиболее популярных картографических онлайн-систем можно выделить сервисы Google Maps [3] и Яндекс карты [4]. Стоит заметить, что кроме данных сервисов существуют также картографические JavaScript библиотеки, такие как Open Layers [5] и Leaflet JS [6], на основе которых можно реализовывать интерактивные карты. Несмотря на существование вышеописанных сервисов и библиотек, в рамках корпоративных систем может быть полезным написание картографических онлайн модулей с нуля. Например, когда требуется узконаправленный функционал или нет возможности использовать внешние сервисы, а существующие библиотеки слишком избыточны для решения конкретной задачи.

Постановка задачи и полученные результаты

В первой части работы решается задача реализации онлайн-карт с возможностью перемещения по карте сдвигом курсора мыши, масштабирования карты в месте, где находится курсор, размещения объектов на карте путем редактирования списка объектов, а также поиска объектов и перехода из списка к карте с центрированием соответствующего объекта на карте. Для решения поставленной задачи требуется решить ряд подзадач.

При реализации картографических систем карту не рассматривают как единое целое, ее разбивают на изображения небольшого размера, называемые тайлами. При сдвиге карты с сервера загружаются только те тайлы, которые попадают в видимую область. Связано это с тем, что пользователь системы, вообще говоря, не готов хранить на

своем ПК или мобильном устройстве всю карту, ему нужны только те области карты, которые он запрашивает при работе с картой. Соответсвенно, одной из основных задач является корректный расчет того, какие именно тайлы и в каком месте экрана необходимо отрисовывать в каждый момент времени. Описание решения данной задачи дано в разделе **«Отрисовка тайлов»**.

Также предстоит решить, как именно перемещать карту. Ответ на этот вопрос дается в разделе **«Событие движения дива map»**.

Другой важной задачей является корректная реализация переходов между уровнями зума. При увеличении или уменьшении уровня зума точка, на которую нажал пользователь, должна оказаться под курсором мыши после изменения размера. Описание реализации дается в разделе **«Измененние размера карты»**.

После решения вышеупомянутых задач необходимо решить проблему расстановки точек на карте. Каждая точка определяет местонахождение того или иного объека на карте. В базе данных хранятся географические координаты объектов в виде долготы и широты, и задача состоит в том, чтобы по ним верно расставить объекты. Об этом рассказывается в разделе **«Отрисовка обьектов на карте»**.

Также к подзадачам, связанным с работой с картой, стоит отнести центрирование на карте выбранной пользователем в таблице объектов точки. Эта проблема решается в разделе **«Центрирование точки»**.

Работа со списком объектов подробно описана в разделе **«Список объектов»**. В подразделе **«Отображение таблицы»** описывается вывод таблицы и поисковой формы на экран. В остальных подразделах описана серверная часть. В разделе **«Заполнение таблицы»** показано использование ajax запроса для заполнение строк данными из базы. В разделе **«Изменение данных в таблице»** описан механизм редактирования данных в таблице и сохранения их в базу. Последняя подзадача, а именно поиск данных в таблице базы данных описана в разделе **«Поисковая форма»**.

На рисунке 1 приведен окончательный результат работы — карта Москвы с обозначенными на ней объектами.

Во второй части ставится задача кластеризации выведенных на карту обьектов. Здесь рассматривается алгоритм «k-средних» и алгоритм кластеризации методом расстояний. Результатом работы каждого из алгоритмов является таблица в базе данных, в которой каж-

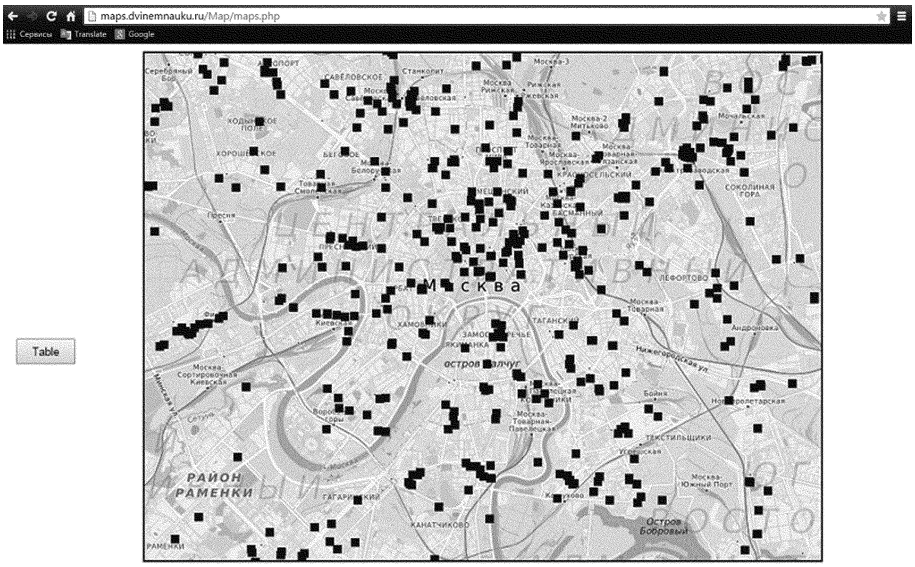

Рис. 1. Карта Москвы.

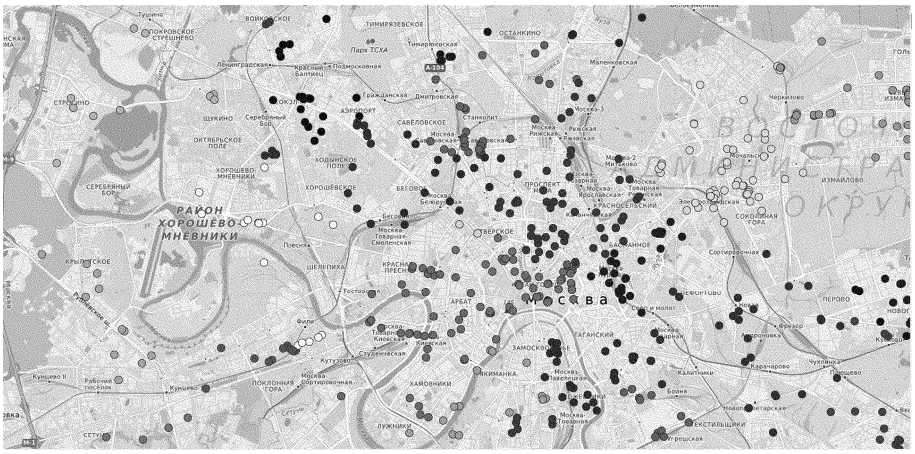

Рис. 2. Кластеризация методом расстояний. Расстояние равно 3000 метров.

дой точке сопоставлен соответствующий номер кластера. Алгоритм кластеризации методом расстояний описан в главе **«Алгоритм кластеризации методом расстояний»**. Результат представлен на ри-

сунке 2.Алгоритм кластеризации «k-средних» описан в главе **«Алгоритм k-средних»**. Результат представлен на рисунке 3.

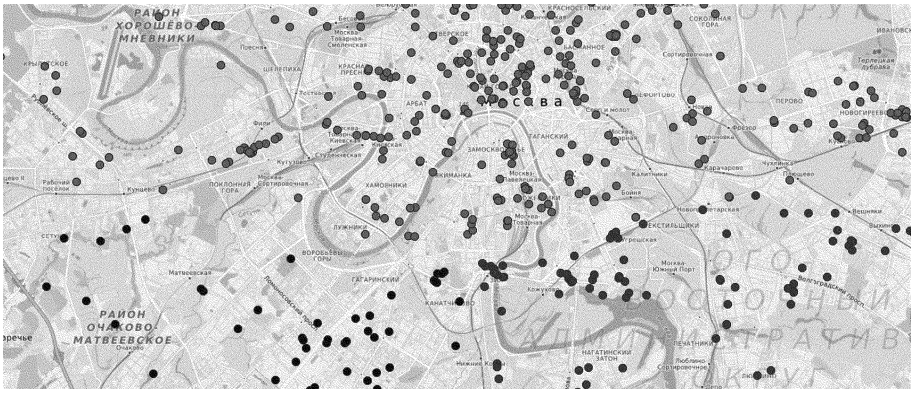

Рис. 3. Алгоритм k-средних. Параметр $k = 3$.

База данных

Как и любое полноценное современное веб-приложение, онлайн-карты используют для работы данные, а именно список обьектов, расположеных на ней. Список таких объектов хранится в базе данных с именем «maps». Для работы требуется две таблицы. В первой, с названием «moscow», хранится уникальное id каждого обьекта (столбец id), его название (столбец name), адрес (address), долгота (longitude) и широта (latitude), во второй, с названием «dots», хранятся экранные координаты (в пикселях) каждого обьекта. Для этого используются столбцы X13, Y13, X15, Y15, X17, Y17, по два для каждого уровня зума. Эта таблица создана для удобства отображения точек на карте и их кластеризации. Данные из «dots» позволяют не переводить географические координаты в экранные каждый раз когда требуются эти координаты. На рисунке 4 приводится скриншот данной таблицы.

Основной функционал карт

Кратко опишем содержимое php-файла, в котором реализуется основной функционал карт. Сперва идет php-скрипт, где происходит

id	name	address	X13	Y13	X15	Y15	X17	Y17
2	Арбор Мунди	Москва, Лермонтовский проспект, 2К1	1432	1141	5730	4565	22923	18263
3	Dreamwear	Москва, улица Сущёвский Вал, 5С1	694	671	2777	2687	11109	10748
4	Apple	Москва, Русаковская улица, 1	918	736	3675	2945	14702	11783
5	Волшебный мир компьютеров (Ф-Центр)	Москва, Сухонская улица, 7А	867	249	3470	996	13882	3987
6	Гвоздь-2	Москва, проспект Андропова, 36	897	1286	3591	5147	14367	20588

Рис. 4. Таблица dots.

подключение к базе данных. Далее следует HTML заголовок, стандартный для большинства веб-страниц, в нем подключаются css документ и библиотеки для работы с jQuery. Перед началом основного скрипта, описываемого далее, на страницу помещаются два основных дива и одна кнопка.

```
<div class = "mainbox">
    <div class = "map"></div>
</div>
<button class = "table_button">Table</button>
```

Внешний див «mainbox» представляет собой неподвижный прямоугольник с размерами 800 на 600, в котором помещается вся карта. Внутренний див «map» — это контейнер для тайлов, он не имеет размеров, и именно к нему привязывается событие «drag&drop», с помощью которого происходит движение карты внутри «mainbox». Кнопка «table_button» предназначена для перехода от карты к таблице со списком объектов.

Далее следует сам скрипт. Функции будут описаны в порядке их следования в файле.

Отрисовка тайлов

Функция Draw [7] отрисовывает тайлики внутри дива «mainbox». Функция принимает два параметра — левый и верхний сдвиг дива «map». Сначала считаем, с какого тайла начать отрисовку карты, другими словами, какой тайл окажется в верхнем левом углу дива

«mainbox». Для этого берем отдельно левый и верхний сдвиги относительно «mainbox» и делим их на размер тайлов (curTileSize). От частного берем целую часть.

$$tileY = \left[\frac{-topOffset + offsetTopWindow}{curTileSize}\right],$$
$$tileX = \left[\frac{-leftOffset + offsetLeftWindow}{curTileSize}\right].$$

Затем в двойном (вложенном) цикле с помощью метода append добавляем к диву «map» новый див, свойству «background-image» которого присваиваем изображение нужного тайлика.

```
for (i = tileY - 1; i < countY + tileY + 1; i++)
{
    for (j = tileX - 1; j < countX + tileX + 1; j++)
    {
        n = i+1;
        m = j+1;
        imgSrc = imgFile+'/'+i+'-'+j+'.png';
        $map.append('<div id = "'+n+'_'+m+'" class = "tile"> </div>');
        curTile = $('#'+n+'_'+m+'');
        curTile.css({
            'background-image': 'url('+imgSrc+')',
            'position': 'absolute',
            'top': (curTileSize) * i + 'px',
            'left': (curTileSize) * j  + 'px',
            'width': curTileSize  + 'px',
            'height': curTileSize  + 'px',
            'background-size' : curTileSize + 'px'
        });
    }
}
```

Отрисовка обьектов на карте

Функция DrawPoints [8] с помощью AJAX запроса получает обьекты, расположенные на карте в области, которая в данный момент отображается на экране, и рисует их в виде точек. Запрос отправляется в файл «getPoints.php», который возвращает результат в json-формате. Затем в цикле точки (дивы небольшого размера) методом append добавляются к диву «map». Если при этом пользователь захотел выделить какой-либо обьект, то есть перешел на карту из таблицы нажатием на какую-либо строку, то именно в этой функции

нужный обьект выделится цветом, отличным от остальных точек. В «getPoints.php» передаются координаты левого верхнего и правого нижнего углов экрана и текущий размер зума. С помощью этих данных выполняется SQL запрос к таблице «dots», где хранятся экранные координаты (в пикселях) каждого обьекта для каждого размера зума.

Центрирование точки

Функция CountPosition [9] решает задачу перевода географических координат (долготы и широты) в экранные (пиксели). Она вызывается, если пользователь перешел на карту из таблицы, желая выделить на карте какой-либо обьект. При этом с помощью GET запроса в файл передаются долгота и широта нужного обьекта, которые являются параметрами функции. Перевод координат производится по формулам, которые были получены из следущих соотношений. С помощью Google Maps были определены точные координаты левого верхнего и нижнего правого углов карты. Далее зная размеры карты для каждого из зумов, можно посчитать, сколько пикселей приходится на один градус широты и долготы. Также здесь меняется позиция дива «map» таким образом, чтобы выделенный обьект оказался в центре видимой области карты.

Событие движения дива map

Движение дива «map» — самое часто возникающее событие. Оно происходит при перетаскивании карты внутри дива «mainbox». При этом генерируются три новых события «start», «drag», «stop» — начало движения, сам момент движения и прекращение движения, соответственно. Мы используем только событие «stop», которое возникает в момент прекращения движения «map». Обработчик события в качестве параметра получает объект «ui», который имеет поле «ui.offset» — позиция перемещаемого элемента относительно начала документа. Внутри обработчика вызываются методы «Draw» и «DrawPoints».

```
$map.draggable(
{
    stop: function(event, ui)
```

```
    {
        Draw(ui.offset.left, ui.offset.top);
        DrawPoints((-ui.offset.top + offsetTopWindow),(-ui.offset.left +
        offsetLeftWindow), zoom);
    }
});
```

Измененние размера карты

Функция Zoom [10] реализует изменение размеров карты, то есть увеличивает или уменьшает зум. Она принимает три аргумента — координаты мышки (mouseX, mouseY) относительно «mainbox» (то есть в левом верхнем углу дива координиаты равны (0,0)) и тип зума (zoomtype), который принимает значения 1 или -1 — увеличение и уменьшение зума соответственно. Процесс увеличения зума осуществляется следующим образом: размеры тайлов постепенно увеличиваются в 1.25, 1.5, 1,75 раз, а потом заменяются на тайлы из директории для следущего зума. Плавность увеличения обеспечивается функцией setTimeout, которая делает паузу перед выполнением кода находящегося в нем, длина паузы устанавливается произвольно:

```
setTimeout(function()
{
...
},30);
```

Здесь код, который находится внутри фигурных скобок вместо многоточия, выполнится с задержкой в 30 милисекунд. Например, переход с зума уровня 13 на зум уровня 15 происходит следущим образом: сначала размер тайлов, который равен 256x256, умножается на 1.25, затем через 30 милисекунд — на 1.5, еще через 60 милисекунд умножается на 1.75 и наконец, через 90 милисекунд, размер возвращается в исходный(256x256), и тайлы заменяются на необходимые для зума уровня 15. Очевидно, что если просто заменить тайлы таким образом, то область, которая отображалась на экране сдвинется влево вниз, и точка которую пользователь хотел увеличить, убежит из его поля зрения, поэтому параллельно с изменением размера тайлов нужно менять положение карты. В представленном ниже коде описано увеличение тайлов на 1.25 и сдвиг карты таким образом, чтобы точка, на которую нажал пользователь, оказалась в том же месте, где и была изначально.

```
left_position = $map.position().left;
top_position = $map.position().top;
setTimeout(function()
{
    l = mouseX - 1.25*(mouseX - left_position);
    t = mouseY - 1.25*(mouseY - top_position);
    $map.css
    ({
        'top': t + 'px',
        'left': l  + 'px',
    });
    curTileSize = startTileSize*1.25;
    Draw(l, t);
},0);
```

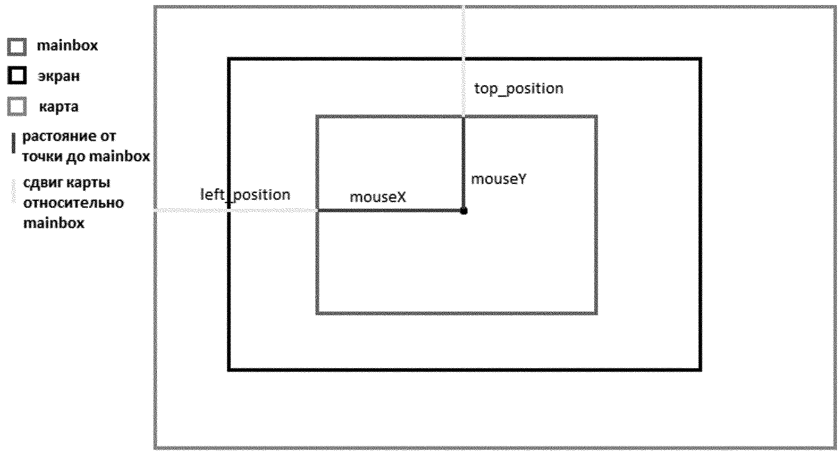

Рис. 5. Расчет новой позиции карты после изменения зума.

Разберем код построчно. Сначала получаем позицию карты относительно родительского элемента — дива «mainbox». Left_position — сдвиг карты влево, top_position — сдвиг карты вверх, средства jQuery позволяют получить эти данные с помощью метода position() элемента. Далее идет расчет новой позиции карты на которую она должна встать после увеличения, рассмотрим его подробнее. Очевидно, если увеличить все тайлы на произвольное число (в нашем случае на 1.25), то расстояние от точки нажатия до левого верхнего угла карты увеличится на это же число, значит, чтобы вернуть точку в нужную

позицию необходимо просто отнять от текущей позиции разность увеличенного и исходного размеров. Сделаем это для левого сдвига, для верхнего все аналогично. Чтобы найти расстояние от точки нажатия до левой границы карты необходимо к позиции курсора слева $mouseX$ прибавить левый сдвиг карты. Так как сдвиг — величина отрицательная, то расстояние равно $mouseX - left\_position$, значит после увеличения оно будет равно $1.25 * (mouseX - left\_position)$. Нам необходимо сделать следущее: $left\_position - (1.25 * (mouseX - left\_position) - (mouseX - left\_position))$. Раскрыв скобки и произведя простые алгебраические действия, получим выражение: $mouseX - 1.25 * (mouseX - left\_position)$.

Далее, через css свойства обьекта устанавливаем его в нужную позицию, увеличиваем размер тайликов и вызываем метод Draw. Таким же образом вызываем функцию setTimeout с задержками 30, 60 и 90. В последнем вызове заменяем директорию с тайлами на ту, где хранятся тайлы для следущего уровня зума. Например, для перехода от зума 13 к зуму 15, последний вызов setTimeout выглядит так:

```
setTimeout(function()
{
    l = mouseX - 4*(mouseX - left_position);
    t = mouseY - 4*(mouseY - top_position);
    $map.css
    ({
        'top': t + 'px',
        'left': l  + 'px',
    });
    curTileSize = startTileSize ;
    imgFile = "../maptiles/z15";
    Draw(l, t);
    DrawPoints(l,t, zoom);
},90);
```

Как видно, размер тайлов возвращается в оригинальный (256x256), но расстояние до точки, на которую нажал пользователь, увеличилось в 4 раза. Это происходит потому, что один тайлик для зума уровня 13 содержит область, которую содержат 16 тайликов для зума уровня 15. Такое же соотношение выполняется при переходе от зума уровня 15 к уровню 17.

Событие двойного клика левой кнопкой мыши

При двойном нажатии на карту левой кнопкой мыши должно произойти увеличение на один зум. В обработчике события запоминаются координаты курсора в момент нажатия и передаются как параметры в функцию Zoom, которая вызывается здесь же:

```
$map.on('dblclick', function(e){
    mouseX = event.x - offsetLeftWindow;
    mouseY = event.y - offsetTopWindow;
    Zoom(mouseX, mouseY, 1);
});
```

Обработчик события получает обьект event, в котором содержатся поля event.x и event.y — координаты курсора мыши относительно экрана, но удобнее использовать координаты относительно mainbox. Для этого от event.x отнимаем расстояние между mainBox и левой границей экрана (offsetLeftWindow), аналогично поступаем с event.y. Вызываем функцию Zoom с параметром ZoomType, равным 1, так как по двойному нажатию левой кнопкой мыши мы увеличиваем зум.

Событие двойного клика правой кнопкой мыши

При двойном нажатии на карту правой кнопкой мыши должно произойти уменьшение на один зум. В JavaScript нет события, отвечающего за двойное нажатие правой кнопкой мыши, но есть событие одинарного нажатия, при котором вызывается контекстное меню. Для того чтобы искусственно создать событие двойного нажатия, поступим следущим образом. При первом нажатии будем запоминать время нажатия, а при втором вычислять разницу между текущим временем и запомненным. Если разница не превышает одной секунды, то считаем, что пользователь совершил двойной клик правой кнопкой мыши и пожелал уменьшить уровень зума.

```
$map.on('contextmenu', function(e){
    if(cur_sec == -1)
    {
        cur_time = new Date();
        cur_sec = cur_time.getSeconds();
    }
    else
    {
```

```
        times = new Date();
        tmp_sec = times.getSeconds() - cur_sec;
        if(tmp_sec<=1)
        {
            mouseX = event.x - offsetLeftWindow;
            mouseY = event.y - offsetTopWindow;
            Zoom(mouseX, mouseY, -1);
        }
        cur_sec = -1;
    }
});
```

Список объектов

В текущем разделе мы опишем функционал, реализованный для работы с объектами на карте.

Отображение таблицы

При описании файла maps.php мы упомянули о кнопке, расположенной по левую сторону от карты, см. рис. 1. При нажатии на нее пользователь переходит на страницу с таблицей, содержащей список всех объектов. Таблица не растянута на всю страницу, а, как и карта, расположена внутри дива размеров 800x600, будем называть его «Box». Также на странице расположена поисковая форма, для поиска обьектов по имени или адресу. В начале файла идет php-скрипт, в котором вычисляются два важных значения — общее количество строк в таблице и количество строк которые помещются в Box.

```
<?php
    header('Content-Type: text/html; charset='cp-1251');
    $connect = mysqli_connect('localhost', 'maps', 'maps12345')
    or die("MySQL connection error");
    mysqli_select_db($connect, 'maps') or die("Select database error");
    mysqli_set_charset($connect,'cp-1251');
    $q = mysqli_query($connect, "SELECT COUNT(*) FROM `moscow`");
    $numRows = mysqli_fetch_array($q)[0];
    $row_on_screen = (int)((600/20));
?>
```

Как видно, общее количество строк сохраняется в переменную $numRows, а количество строк, помещающихся в Box — в

$row_on_screen. Число 600 в расчете $row_on_screen — высота Box, а 20 — высота строки.

Далее переместимся к тегу body. В нем расположен код, создающий саму таблицу. Так как количество строк достаточно велико, было принято решение создавать изначально количество строк, равное 2*$row_on_screen. То есть при обновлении страницы таблица имеет удвоенное количество строк, помещающихся в Box (в нашем случае это 60 строк). При скролле таблицы вниз к ней добавляются строки, а данные в них загружаются с помощью ajax запроса, иначе при большом количестве строк нагрузка на страницу была бы очень велика. Далее приведен код тега body, в котором создаются все элементы формы.

```
<body>
<div class ='find'>
<div>Name</div>
<input type="text" size="25" class='name'><br><br>
<div>Address</div>
<input type="text" size="25" class='address'><br><br>
<button class='find_button'>Find</button><br><br>
<button class='cancel_find'>Cancel</button>
</div>
<div class='Box'>
  <table class='table' border="1">
    <?php for($i=1;$i<=2*$row_on_screen;$i++) : ?>
      <tr class='Tr' id='<?=$i?>'>
        <td column_name= 'id' class='Td' ></td>
        <td column_name= 'name' class='Td' ></td>
        <td column_name= 'address' class='Td' ></td>
        <td column_name= 'longitude' class='Td'></td>
        <td column_name= 'breadth' class='Td' ></td>
      </tr>
    <?php endfor ?>
  </table>
</div>
</body>
```

Заполнение таблицы

Функция заполнения таблицы [11] вызывается при скролле таблицы. Она заполняет ее строки данными из базы. Функция принимает два параметра, begin — номер строки, которая при текущем уровне

прокрутки находится вверху таблицы и cnt — количество строк, которое необходимо заполнить. Сначала формируется GET запрос к файлу rows.php, в котором выполняется соответствующий SQL запрос.

```
var request = "n=" + begin + "&m=" + cnt;
$.ajax({
    url : "rows.php",
    data : request,
    ...
})
```

Данные, полученые в результате запроса, возвращаются из файла rows.php в json формате.

```
$.ajax({
    url : "rows.php",
    data : request,
    dataType : "json",
    success: function(data)
    {
        var str = JSON.stringify(data);
        var tmp = JSON.parse(str);
        var i;
        for (i = 0, j = begin; i < cnt; i++, j++)
        {
            $tr = $('#'+j);
            $tr.children("td").eq(0).text(tmp[i].id);
            $tr.children("td").eq(1).text(tmp[i].name);
            $tr.children("td").eq(2).text(tmp[i].address);
            $tr.children("td").eq(3).text(tmp[i].longitude);
            $tr.children("td").eq(4).text(tmp[i].breadth);
        }
    }
});
```

Функция $.ajax(...) содержит секцию success, код внутри которой выполняется только в случае успешного выполнения запроса. Если rows.php успешно вернул данные, то в цикле осуществляется заполнение строк таблицы.

Событие скролла таблицы

Обработчик события работает следущим образом. При прокрутке считается количество прокрученых строк и именно это количество

строк добавляется в конец таблицы. Для них вызывается функция FillTable.

Изменение данных в таблице

При нажатии на любую ячейку таблицы кроме ячейки id, она переходит в режим редактирования данных. То есть у пользователя есть возможность менять название или адрес любого обьекта. Сохранение новых данных происходит в момент, когда с редактируемой ячейки снимается фокус, другими словами, когда пользователь нажимает в другое произвольное место таблицы. Сохранением данных в базу занимается функция UpdateTable, в качестве параметров она принимает id строки, название колонки, ячейка которой редактировалась — column_name и новые данные — nw_data. Эти данные с помощью ajax запроса отправляются в файл updateRows.php, в котором формируется update запрос к базе данных. Код функции небольшой, поэтому приведем его целиком:

```
function UpdateTable(id, column_name, nw_data)
{
    var request = "id=" + id + "&column_name=" + column_name +
        "&nw_data="+nw_data;
    $.ajax({
        url : "updateRows.php",
        data : request,
    });
}
```

Поисковая форма

Поисковая форма позволяет пользователю искать объекты в таблице по имени или адресу. Пользователь вводит данные в поля Name и/или Address и при нажатии на кнопку Find в таблице остаются только строки, удовлетворяющие запросу пользователя. Вышесказанное реализовано в обработчике события нажатия на кнопку Find посредством ajax запроса в файл find_rows.php.

```
$(".find_button").on('click', function()
{
    var name = $('.name').val();
    var address = $('.address').val();
```

```
    var request = "name=" + name + "&address=" + address;
    $.ajax({
        url : "find_rows.php",
        data : request,
        dataType : "json",
        \\...
    });
});
```

Полученные данные, также как и в функции FillTable, записываются в строки таблицы. Кнопка Cancel предназначена для возврата таблицы в прежнее состояние.

Кластеризация объектов на карте

Перейдем к решению задачи кластеризации объектов на карте. Нам потребуются дополнительные таблицы в базе данных. В таблице *listofpoints* хранятся 1030 различных торговых точек Москвы. В таблице имеются следующие столбцы: идентификатор точки (*id*), название точки (*Name*), адрес каждой точки (*Address*), широта (*firstkoord*) и долгота (*secondkoord*) . Таблица показана на рисунке 6.

id	Name	Adress	firstkoord	secondkoord	color
1	Доминатор2	Москва, Малая Семёновская улица, 28С13	55.7862	37.7165	1
2	Арбор Мунди2	Москва, Лермонтовский проспект, 2К1	55.7049	37.8354	1
3	Dreamwear2	Москва, улица Сущёвский Вал, 5С1	55.7847	37.5969	1
4	Apple2	Москва, Русаковская улица, 1	55.7922	37.6590	1
5	Волшебный мир 22комп	Москва, Сухонская улица, 7А	55.8564	37.6714	1
6	Гвоздь-2	Москва, проспект Андропова, 36	55.6458	37.6518	1
7	Атриум2	Москва, улица Земляной Вал, 33	55.7672	37.6571	1
8	Очаково2	Москва, Рябиновая улица, 44С3	55.6957	37.4372	1
9	Волшебный2 мир компь	Москва, улица Миклухо-Маклая, 55	55.6507	37.5545	1
10	Плеер.ру2	Москва, улица Мастеркова, 4	55.7390	37.6660	1
11	Доминатор	Москва, Малая Семёновская улица, 28С13	55.7852	37.7155	1
12	Арбор Мунди	Москва, Лермонтовский проспект, 2К1	55.7039	37.8454	1

Рис. 6. Таблица listofpoints.

В таблице *million* 4 столбца. Первый и второй столбцы содержат идентификаторы точек из таблицы *listofpoints*, третий столбец задает расстояние между соответствующими точками в метрах. Четвёртый

столбец *del* является вспомогательным столбцом. Для вычисления расстояния между точками в таблице *million* используем формулу

$$L = \arccos(\sin(\varphi_A) \cdot \sin(\varphi_B) + \cos(\varphi_A) \cdot \cos(\varphi_B) \cdot \cos(\lambda_A - \lambda_B)) \cdot R,$$

где φ_A и φ_B — широта, λ_A, λ_B — долгота данных пунктов, d — расстояние между пунктами, измеряемое в радианах длиной дуги большого круга земного шара. Расстояние между пунктами, измеряемое в километрах, определяется по формуле $L = d \cdot R$, где $R = 6371$ км — средний радиус земного шара. Остальные 2 таблицы временные, они будут описаны в алгоритмах кластеризации.

Алгоритм кластеризации методом расстояний

В данном разделе мы описываем алгоритм решения задачи кластеризации объектов методом расстояний. Считаем, что алгоритм на входе получает некоторое число S и в каждом результирующем кластере любая точка находится на расстоянии не больше чем S от центра масс точек этого кластера. Разобьём задачу на следующие подзадачи.

- Записываем в таблицу *timemillion* все строки таблицы *million*, значение расстояния которых меньше S.
- Находим точку, в S-окрестности которой содержится максимальное число других точек.
- Для полученной точки находим все точки её S-окрестности. Считаем эти точки рассмотренными и удаляем из таблицы *timemillion* строки, содержащие эти точки.
- Запускаем рассмотренные шаги в цикле для сопоставления кластеров всем точкам таблицы *listofpoints*.
- Сохраняем итоговую кластеризацию в базе данных.

Далее рассматриваем каждый из описанных шагов более подробно.

Шаг 1. Записываем в таблицу *timemillion* все строки таблицы *million*, значение расстояния которых меньше S

Напомним, что в таблице *million* лежат все попарные расстояния между точками. Напишем запрос, который вынет из таблицы *million* строки, соответствующие точкам, расстояние между которыми меньше S. Результат запроса представлен на рисунке 7.

```
SELECT id1, id2, distance FROM million WHERE distance < @S
```

id1	id2	distance
1	11	128
1	49	1721
1	62	1081
1	67	788
1	72	486

Рис. 7. Таблица timemillion.

id1	amount
446	129
63	129
563	129
397	127
994	126

Рис. 8. Число элементов для центроидов.

Шаг 2. Находим точку, в S-окрестности которой содержится максимальное число других точек

Результатом следующего запроса является таблица, каждая строка которой состоит из идентификатора точки и количества точек в её S-окрестности, в порядке убывания количества. Результат представлен на рисунке 8.

```
SELECT id1, COUNT(*) as amount FROM timemillion
GROUP BY id1 ORDER BY amount DESC
```

Первая строка результата предыдущего запроса соответствует точке, имеющей наибольшее количество точек в ее S-окрестности. Будем называть её центроидом. Записываем этот центроид в переменную *maximum*. Если *maximum* равен 0, то каждая точка образует собственный кластер.

Шаг 3. Для полученной точки находим все точки её S-окрестности, убираем найденные точки из рассмотрения

В следующих двух запросах удаляем из таблицы *timemillion* центроид и все относящиеся к нему точки для того, чтобы далее они не рассматривались. Так как повторять запрос удаления для каждой точки не целесообразно, сначала накапливаем *id* точек в строку *str*, а уже потом одним запросом удаляем.

```
DELETE FROM timemillion WHERE id1 = @maximum
DELETE FROM timemillion WHERE id1 IN(...)
```

Запускаем рассмотренные шаги в цикле для сопоставления кластеров всем точкам таблицы *listofpoints*

Применяем шаги с первого по третий. При помощи шага 1 записываем в таблицу *timemillion* все строки таблицы *million*, значение расстояния которых меньше S. При помощи шага 2 находим точку, в S-окрестности которой содержится максимальное число других точек. При помощи шага 3 для полученной точки находим все точки её S-окрестности. Считаем эти точки рассмотренными и удаляем из таблицы *timemillion* строки, содержащие эти точки. На выходе получаем массив *finish*, где лежат *id* точки и номер кластера. Далее очищаем таблицу *timemillion*.

Сохраняем итоговую кластеризацию в базе данных

Обновляем таблицу *listofpoits* и записываем значения массива *finish* в соответствующие строки. Предположим, что функция *color* сопоставляет заданному числу от 0 до 256^3 отдельный цвет. Предоставляем её реализацию читателю. Применим эту функцию для раскраски всех точек. Результат показан на рисунке 9.

Алгоритм k-средних

В данном разделе мы описываем алгоритм кластеризации точек на карте методом k-средних. Реализация данного метода основывается на поиске центроидов и распределении точек по этим центрои-

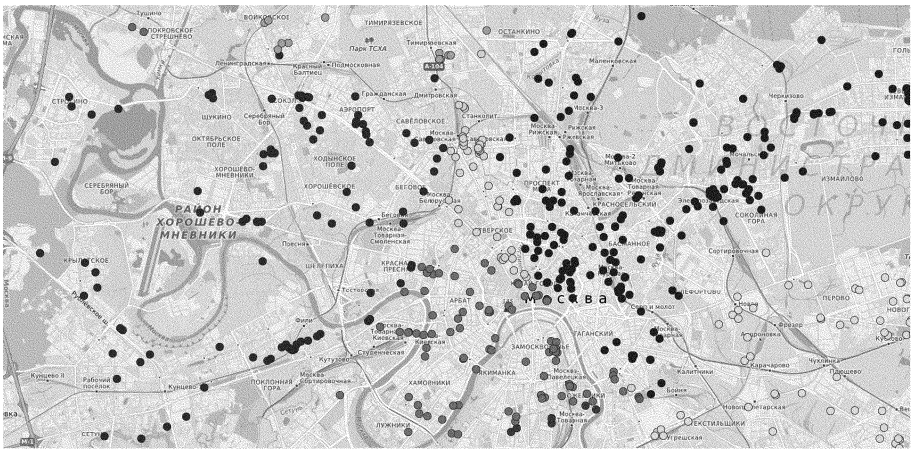

Рис. 9. Кластеризация методом расстояний. Расстояние равно 10000 метров.

дам. Считаем, что алгоритм на входе получает некоторое число k — количество результирующих кластеров будет равняться k. Разобьём задачу на следующие подзадачи.

- Выбираем k первых точек и запоминаем их.
- Записываем во временную таблицу строки. Каждая стока имеет следущий вид: центройд, точка, относящаяся к центройду, и расстояние между ними. Для каждого центройда перечисляются все относящиеся к нему точки.
- Находим минимальное расстояние от точек до центроидов.
- Для каждого кластера находим центр масс и ближайшую к нему точку по координатам. Переопределяем массив центроидов новыми данными.
- Проверяем на совпадения новых центроидов с любыми центроидами из предыдущих итераций.
- Сохраняем итоговую кластеризацию в базе данных.

Шаг 1. Выбираем k первых точек и запоминаем их

Изначально данный алгоритм позволяет выбрать любые k точек из исходной таблицы с точками, для определённости мы выбираем

первые из них. Выбранные точки мы назовём центроидами и удалим их из таблицы *million*. Пример удаления точек показан на рисунке 10. Удаленные точки не рассматриваются в следующих шагах алгоритма.

id1	id2	distance	del
1	2	11722	1
1	3	7488	0
1	4	3660	1
1	5	8308	0
1	6	16147	1

Рис. 10. Пример удаления точек.

id1	id2	distance
11	1	128
11	2	11677
11	3	7424
11	4	3621
11	5	8392

Рис. 11. Таблица *timemillion*.

```
UPDATE million SET ‘del‘ = 0 WHERE id1 IN (...)
```

Шаг 2. Заполняем таблицу *timemillion*

С помощью следующих двух запросов мы выбираем точки, относящиеся к нашим центроидам, и записываем их во временную таблицу *timemillion*. Пример полученного результата можно видеть на рисунке 11.

```
SELECT * FROM million WHERE id1 IN (...) AND del=1
INSERT INTO timemillion (id1, id2, distance) VALUES (...)
```

Шаг 3. Находим минимальное расстояние от точек до центроидов

Результатом следующего SQL запроса является временная таблица *newtimemillion*, в которой лежат центроиды с соответствующими точками и расстояние между ними, таблица отсортирована по центроидам и по расстоянию в порядке возрастания. Все точки, относящиеся к одному центроиду, назовём кластером. Пример результата запроса можно видеть на рисунке 12.

```
INSERT INTO newtimemillion (id1,id2,mindist)
SELECT tm3.id1, tm2.id2, tm2.mindist FROM (SELECT tm1.id2,
```

```
min(tm1.distance) as mindist FROM timemillion tm1 GROUP BY
id2) as tm2 JOIN timemillion tm3 ON tm3.distance =
tm2.mindist AND tm3.id2=tm2.id2
```

id1	id2	mindist
11	1	128
11	4	3621
11	5	8392
11	7	4169
11	10	6005

Рис. 12. Таблица newtimemillion.

SredneeX	SredneeY
55.77062618	37.71102833
55.63937532	37.71179319
55.75962312	37.55219427

Рис. 13. Центры масс кластеров.

Шаг 4. Для каждого кластера находим центр масс и ближайшую к нему точку по координатам. Переопределяем массив центроидов новыми данными

Данный запрос находит центр масс для каждого центроида путем сложения координат точек, относящихся к каждому из кластеров, и деления на их количество. Полученные координаты и будут являться центрами масс. Пример результата запроса представлен на рисунке 13.

```
SELECT AVG(firstkoord) as SredneeX,
AVG(secondkoord) as SredneeY FROM (SELECT * FROM
newtimemillion as tm1 JOIN listofpoints as tm2 ON
tm1.id2=tm2.id) as tm3 GROUP BY tm3.id1
```

Следующий SQL запрос позволяет определить точку, которая лежит ближе всего к центру масс. Пример результата запроса представлен на рисунке 14.

```
SELECT id, min(ABS(firstkoord+secondkoord-@summ))
FROM (SELECT * FROM newtimemillion as tm1
JOIN listofpoints as tm2 ON tm1.id2=tm2.id) as tm3
GROUP BY tm3.id1
```

id	min(ABS((firstkoord+secondkoord)-93.56114444))
1	0.00004444
2	0.00004444
3	0.00225556

Рис. 14. Поиск точек, ближайших к центрам масс своих кластеров.

Проверка на совпадение новых центроидов с любыми центроидами на предыдущих итерациях

Проверяем, были ли полученные центроиды зафиксированы на любой из предыдущих итерациях, если да, то завершаем алгоритм, если нет, добавляем полученные центроиды в массив со всеми остальными центроидами, определенными на прошлых шагах, и повторяем шаги 2, 3 и 4.

Сохраняем итоговую кластеризацию в базе данных

Обновляем таблицу *listofpoits* и записываем значения массива *finish* в соответствующие строки. Предположим, что функция *color* сопоставляет заданному числу от 0 до 256^3 отдельный цвет. Предоставляем её реализацию читателю. Применим эту функцию для раскраски всех точек. Результат показан на рисунке 15.

Заключение

В статье рассмотрена реализация web-приложения онлайн-карт. Подробно описано, как карта, разбитая на тайлы, отображается на экране в определенный момент времени, какие тайлы подгружаются при очередном перемещении видимой области, каким образом осуществляется корректное увеличение карты. Реализован перевод географических координат в экранные и обратно, описано, как таблица объектов заполняется данными при прокрутке, а также реализована связь между таблицей объектов и картой. Реализована кластеризация объектов на карте методом расстояний и методом k-средних.

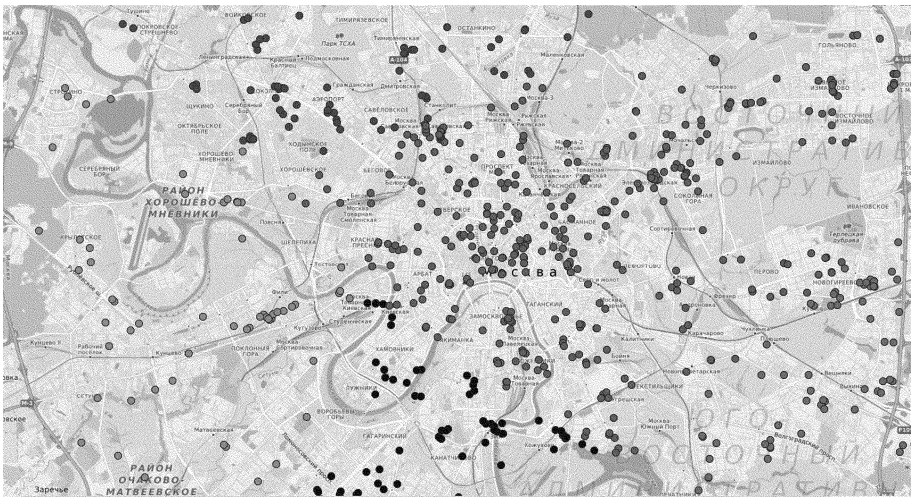

Рис. 15. Алгоритм k-средних. Параметр $k = 5$.

Список литературы

[1] Осокин В. В., Аипов Т. Д., Ниязова З. А. О классификации изображений и музыкальных файлов // Интеллектуальные системы. Теория и приложения. — 2015. Т. 19, вып. 1. — С. 49–70.

[2] Осокин В. В., Алимов Р. Ф., Хайдаров Р. Р. Основы реализации поисковой системы // Интеллектуальные системы. Теория и приложения. — 2015. Т. 19, вып. 1. — С. 71–98.

[3] Google maps. [https://maps.google.com].

[4] Яндекс карты. [https://maps.yandex.ru].

[5] Open Layers. [http://openlayers.org].

[6] Leaflet JS. [http://leafletjs.com/].

[7] Листинг: Отрисовка тайлов. Функция Draw. [http://maps.dvinemnauku.ru/Map/Listing/listing.php#Draw].

[8] Листинг: Отрисовка обьектов на карте. Функция DrawPoints. [http://maps.dvinemnauku.ru/Map/Listing/listing.php#DrawPoints].

[9] Листинг: Центрирование точки. Функция CountPosition. [http://maps.dvinemnauku.ru/Map/Listing/listing.php#CountPosition].

[10] Листинг: Изменение размера карты. Функция Zoom. [http://maps.dvinemnauku.ru/Map/Listing/listing.php#Zoom].

[11] Листинг: Заполнение таблицы. Функция FillTable. [http://maps.dvinemnauku.ru/Map/Listing/listing.php#FillTable].