Getting Map Data from OpenStreetMap

OSM Data Model & Querying

Unchitta Kan
ukanjana@gmu.edu | unchitta.com
Mar 23, 2022
George Mason University

Overview

- **Part I**
  - OSM data model and tags
- **Part II**
  - Intro to Overpass
  - Overpass QL basics and simple examples
  - Advanced topics
- **Part III**
  - Demo (let’s try it together)

Part I:
OpenStreetMap Data Model

OSM Data Model

How are data represented in OSM?

Node

Defined by lat, long, along with unique ID.

Way

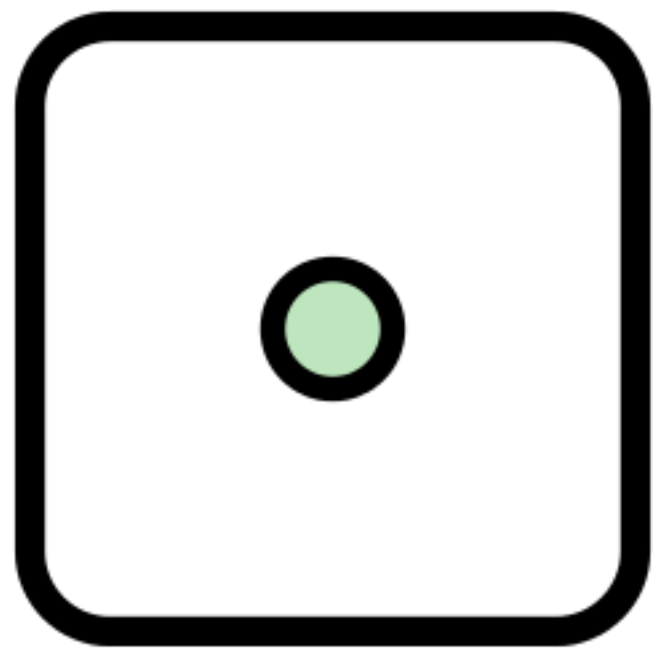
Linear feature made up of nodes, along with unique ID.
Can also be closed way or area.

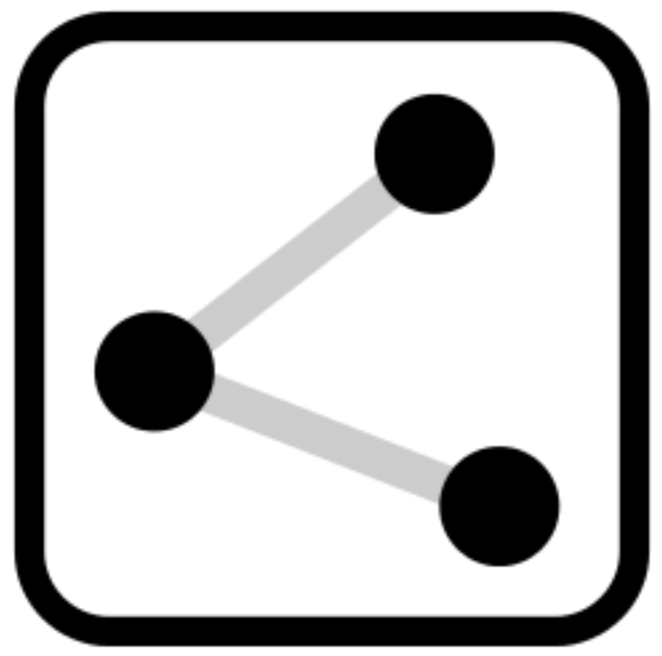
Relation

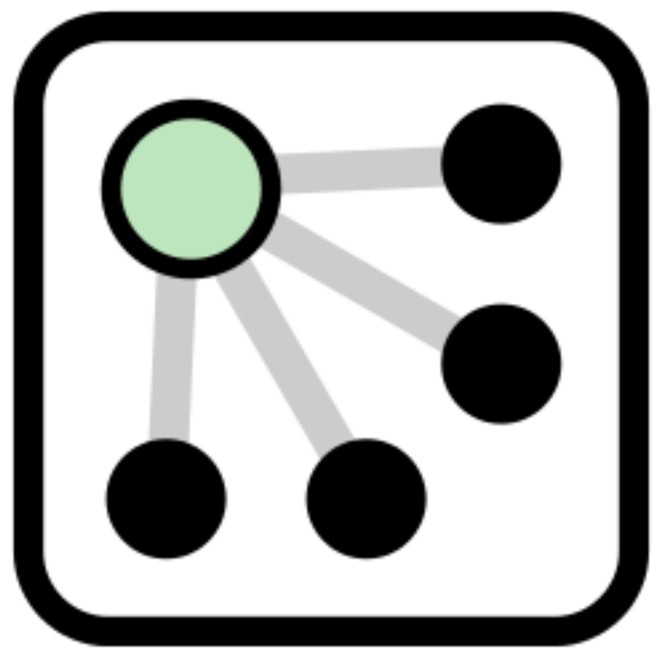
Collection of member objects along with relations between them (“role”), e.g., bus routes. Must have at least “type” tag.

OSM Data Model

How are data represented in OSM?







Tags provide semantics

Tags

(a.k.a the thing that gives meaning to OSM data objects)

- Tags are **key-value** pairs that describe attributes of OSM data objects
  - e.g., `amenity=cafe` or `highway=motorway`
- Free tagging system — an object can have any number of attributes
- Useful keys: **`name`**, `amenity`, `building`, `highway`, `office`, `shop`, `public transport`, `route`
- https://wiki.openstreetmap.org/wiki/Map_features
- OSM data objects also have other attributes such as user id of contributor, change log, timestamp, etc.

Exploring OSM objects

https://www.openstreetmap.org/relation/11158003

Part II:
Intro to Overpass API

Querying Data from OSM

via Overpass API

- Let’s say you want to find the locations of all parks in Washington D.C. that are close to a metro stop, or you want to get the coordinates of major highways in Spain… you can get all of these data via Overpass API
- OpenStreetMap API vs Overpass API
  - Good for querying ~10 million objects in a few minutes by location, tags, proximity, etc.
- Overpass Query Language
- Web-based front end: Overpass Turbo -> can export as GeoJSON
  - http://overpass-turbo.eu/

Overpass QL Basics

Tl;dr: chronological logic + everything is a set

- Overpass Turbo's "query wizard" — why bother learning Overpass QL?
- Overpass QL is a procedural, imperative language
    - Commands are executed in sequence, each one altering the shared state of the program
    - Each statement ends with a `;`
- In Overpass QL, sets underlie program state; results are stored in a default set named `_` (unless explicitly named otherwise) and inputs are also read from this set in the next statement execution (unless explicitly told otherwise)
    - Sets are referred to using `.` followed by the set name (e.g., `._`)

Overpass QL Basics

Tl;dr: chronological logic + everything is a set

- In Overpass QL, sets underlie program state; results are stored in a default set named `_` (unless explicitly named otherwise) and inputs are also read from this set in the next statement execution (unless explicitly told otherwise)
- This means one can also do set operations such as unions and differences

```
node[name="Foo"];
```

```
nwr[name="Foo"]->._;
```

```
(
  node[name="Foo"];
  way[name="Bar"];
  rel[name="Baz"];
)->.a;
```

Overpass QL Basics

Anatomy of a query

- In general, you want to think about:
  - The type of OSM objects you want to query (node/way/rel/nwr)
  - Filters
    - Spatial (e.g., by proximity, or inside an area or bounding box), specified by parentheses `(..)`
    - Attributes/tags, specified by square brackets `[..]`
  - Pipelining/set operations, if necessary
  - Output details (full data, id only, attrs only, geom only, etc)

Example Query 1

Finding parks in Washington D.C.

- Data type
- Filters (order doesn’t matter)
  - Spatial `(..)`
  - Tags `[..]`
- Pipelining/set operations
- Output details

```
area[name="District of Columbia"];
node["leisure"="park"](area);
out;
```

Example Query 2

Finding supermarkets in Washington D.C. within walking distance from a public transit stop

```
area["name"="District of Columbia"] -> .dc;

node(area.dc)["public_transport"="station"];

node["shop"="supermarket"](around:1000);

out;
```

Advanced Topics

- Outputs
- Recursing up/down
- Regular expressions
- If conditions
- Aggregations

```
area[name="District of Columbia"];

/*
Search for park relations with
names ending with "park" (case-insensitive)
*/
rel["leisure"="park"]["name"~"park$",i](area);

/*
Nodes are needed to return way/rel coordiantes.
The following recurse down statement takes the input
set _ and returns a result set containing all nodes and
ways that are members of the relations in the input set.
*/
(._;>;);

/*
Print only tag information and bbox center point
of each object
*/
out tags center;
```

Part III: Demo
(Let’s try it together)

http://overpass-turbo.eu/

Where to go for more details…

- OSM elements
  https://wiki.openstreetmap.org/wiki/Elements
- OSM tags
  https://wiki.openstreetmap.org/wiki/Map_features
- taginfo
  https://taginfo.openstreetmap.org
- Overpass API user’s manual
  https://dev.overpass-api.de/overpass-doc/en/index.html
- Overpass QL doc
  https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL
- Overpass QL repository of examples
  https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_API_by_Example
- Overpass Turbo doc
  https://wiki.openstreetmap.org/wiki/Overpass_turbo

Slides available at
unchitta.com/resources