Assignment 3

Adrian Westh, Emil Lynegaard and Simon Munck

November 22, 2017

Link to prototype: `https://ecly-178408.appspot.com`
To query an address: `/images?address=`
To query a coordinate: `/images?lat=&lng=`
To query an area: `/images/area/north_lat=&south_lat=&east_lng=&west_lng=`
To query the number of images within a country: `/poly/region/country`

Examples:
`https://ecly-178408.appspot.com/images?address=RuedLanggaardsVej,7,2300,K\OT1\obenhavnS`
`https://ecly-178408.appspot.com/images?lat=37.4224764&lng=-122.0842499`
`https://ecly-178408.appspot.com/images/area?north_lat=-2.89&south_lat=-6.55&east_lng=29.63&west_lng=25.93`
`https://ecly-178408.appspot.com/poly/europe/denmark`
`https://ecly-178408.appspot.com/poly/asia/nepal`

1 Parsing PSLG Data

The first step is to fetch the .poly file for the given country by matching the URL on `http://download.geofabrik.de`. This file is parsed and converted to `Polygon`s utilizing the given golang library and the coordinates contained in the file. Each polygon is split into multiple `Cell`s using a `RegionCoverer` based on the polygon, aggregating all the `Cell`s to a single list used as boundaries for BigQuery. The golang library is the offical S2 library[1].

2 Counting Images For Country

This service relies heavily on the *Scaling Spatially* implementation from Assignment 2, which enables the user to request all images within a bounding box (represented by two coordinates).
When the PSLG data is successfully parsed, the latitude and longitude of the individual `Cell`s is extracted. A cell contains a bounding box encapsulating its

[1] https://github.com/golang/geo

area, which can be queried for its contained images, using the above mentioned implementation. The implementation from Assignment 2 utilizes `goroutines` and, as shown below, the separate queries do as well. This makes the implementation scale in the size of the country, and in practice scale in the size of the returned image count, but due to memory restrictions on the AppEngine and/or massive wait time on BigQuery for large cells, this fails or executes in unreasonable time. E.g. one query in Bangladesh uses 460MB.

```
func getImageCountFromCells(ctx context.Context, cells []s2.Cell) int {
    c := make(chan int)
    count := 0

    // concurrently retrieve image count for each cell
    for _, cell := range cells {
        go func(cell s2.Cell) {
            bounds := cell.RectBound()
            lo := bounds.Lo()
            hi := bounds.Hi()
            // this call to retrieve urls is also concurrent internally
            urls := getUrlsBetweenCoords(ctx, hi.Lat.Degrees(), lo.Lat.Degrees(),
                                                hi.Lng.Degrees(), lo.Lng.Degrees())
            c<-len(urls)
        } (cell)
    }

    for range cells {
        count += <-c
    }

    return count
}
```

3 Struggles

The provided S2 library turned out to give us quite a few problems. The function `s2.LoopFromPoints` does not guarantee a reasonable fit. It returns a loop with a bounding box of e.g. high lat: 90, high lng: 180, low lat: -90, low lng: -180 for Bangladesh. This is the case for several countries; as such the library does not seem suited for this type of problems. At the very least, one would have to somehow, using the library or with a custom fit, produce a better fit of some countries based on the given points.