Assignment 03

You will continue to work with the Sentinel2 dataset in Assignment 03. You should build on your implementation from Assignment 01 and 02. In Assignment 02, you implemented a webservice that allows to query rectangle areas based on two latitude and two longitude bands. However, this does not allow users to query for meaningful areas like the one of a country easily. You are going to change that now.

There are two pieces that you will need:

- Geofabrik contains polygons of countries, specified as a Planar straight-line graph (PSLG). E.g., here is Denmark.
- S2 is a geometry library that allows the manipulation of geometric shapes, specifically spherical ones like the Earth.

1. Write a web service that allows you to fetch and parse PSLG data of a country the user inputs from Geofabrik.
2. Use S2's RegionCover type to create an approximation of the resulting polygon (consisting of rectangles). RegionCoverer allows arbitrary regions to be approximated as unions of cells (CellUnion). This is useful for approximating operations on a country. In this assignment, your task is to count the number of images associated to a country (at a given point in time).

To help you get started, at the end of this assignment is some sample code that creates a region cover for a simplified Denmark polygon.

Submit a report describing your design and implementation of both the service as well as an evaluation of how your service scales with the size of a country. You should use snippets from your code to illustrate your design decisions.

Sample Code:

```
package main

import (
	"fmt"
	"math"

	"github.com/golang/geo/s2"
)

func main() {

	// "Denmark Rectangle"
	p1 := s2.PointFromLatLng(s2.LatLngFromDegrees(54.918, 8.552))
	p2 := s2.PointFromLatLng(s2.LatLngFromDegrees(55.048, 8.471))
	p3 := s2.PointFromLatLng(s2.LatLngFromDegrees(55.481, 12.736))
	p4 := s2.PointFromLatLng(s2.LatLngFromDegrees(54.837, 9.392))
	p5 := s2.PointFromLatLng(s2.LatLngFromDegrees(54.918, 8.552))

	// synthetic example
	//p1 := s2.PointFromLatLng(s2.LatLngFromDegrees(1, 1))
	//p2 := s2.PointFromLatLng(s2.LatLngFromDegrees(2, 1))
	//p3 := s2.PointFromLatLng(s2.LatLngFromDegrees(2, 2))
	//p4 := s2.PointFromLatLng(s2.LatLngFromDegrees(1, 2))
	//p5 := s2.PointFromLatLng(s2.LatLngFromDegrees(1, 1))

	points := []s2.Point{p5, p4, p3, p2, p1}

	l1 := s2.LoopFromPoints(points)
	rect := l1.RectBound()
	loops := []*s2.Loop{l1}
	poly := s2.PolygonFromLoops(loops)

	fmt.Printf("No. of edges %v\n", poly.NumEdges())

	// one big rectangle bounding box, just to test
	rect = poly.RectBound()
	fmt.Printf("Rect. Lat. Lo: %v \n", rect.Lat.Lo*180.0/math.Pi)
	fmt.Printf("Rect. Lat. Hi: %v \n", rect.Lat.Hi*180.0/math.Pi)
	fmt.Printf("Rect. Lng. Lo: %v \n", rect.Lng.Lo*180.0/math.Pi)
	fmt.Printf("Rect. Lng. Hi: %v \n", rect.Lat.Hi*180.0/math.Pi)
	fmt.Printf("\nOne Big Rect. Area %v\n\n", rect.Area())

	rc := &s2.RegionCoverer{MaxLevel: 30, MaxCells: 100}
	cover := rc.Covering(poly)
	var c s2.Cell
	var totalArea float64
	totalArea = 0
	for i := 0; i < len(cover); i++ {
		fmt.Printf("Cell %v : ", i)
		c = s2.CellFromCellID(cover[i])
		fmt.Printf("Low: %v - ", c.RectBound().Lo())
		fmt.Printf("High: %v \n", c.RectBound().Hi())
		totalArea = totalArea + c.RectBound().Area()
	}
	fmt.Printf("Total Area with multiple rectangles: %v", totalArea)
}
```