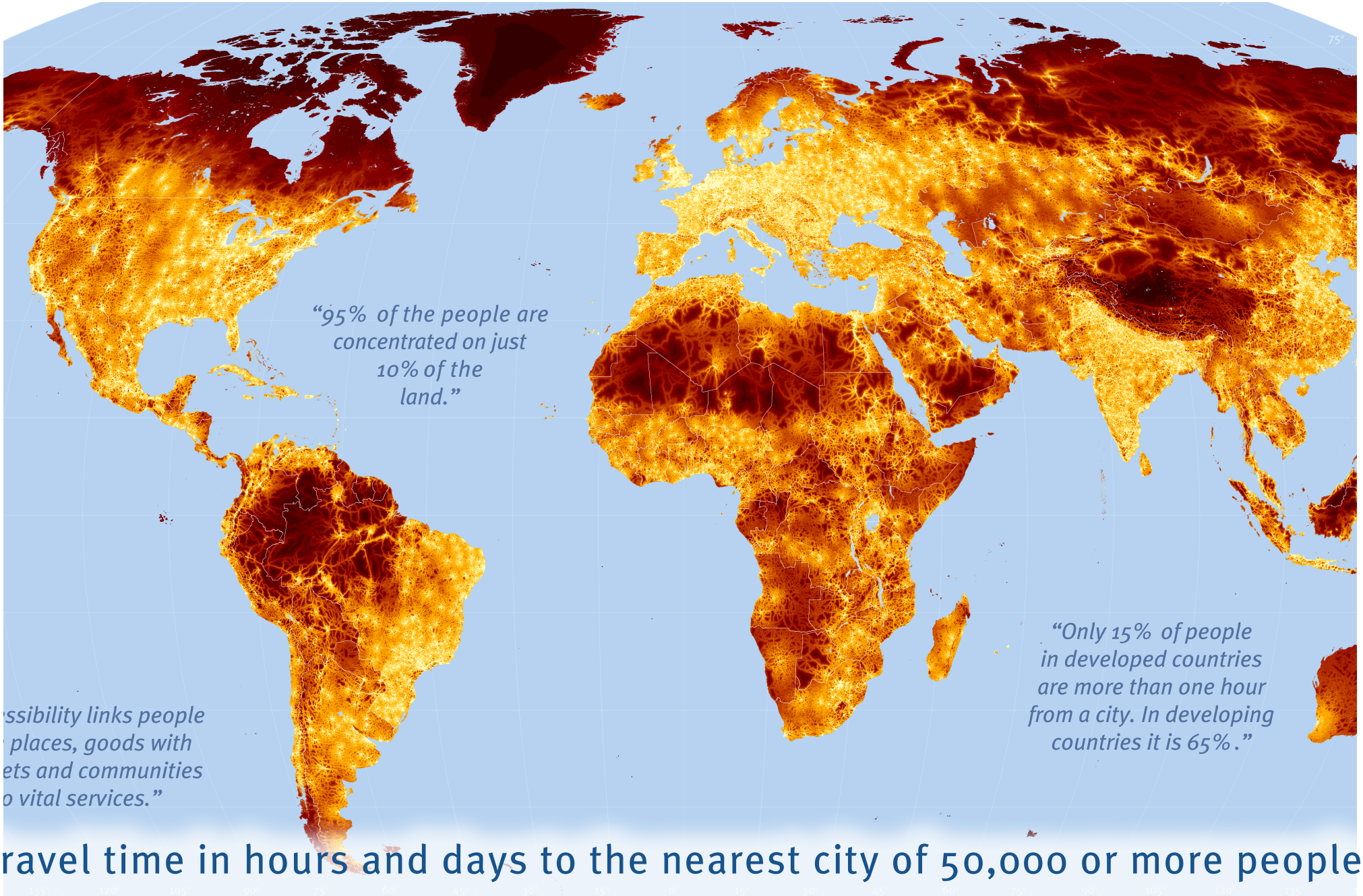


http://bioval.jrc.ec.europa.eu/products/gam/index.htm

Geography 360
November 16, 2016

Spatial Analysis (cont.)

1. *Questions and Announcements*
   - Extra Credit for tonight!
   - Extra Credit in December you should register for now…
2. *Spatial Analysis (continued)*
   - Thinking in terms of algorithms
   - A basic problem for algorithmic solution: point-in-polygon?
   - What algorithms make possible

Spatial Analysis

How does a spatial operation *happen*?

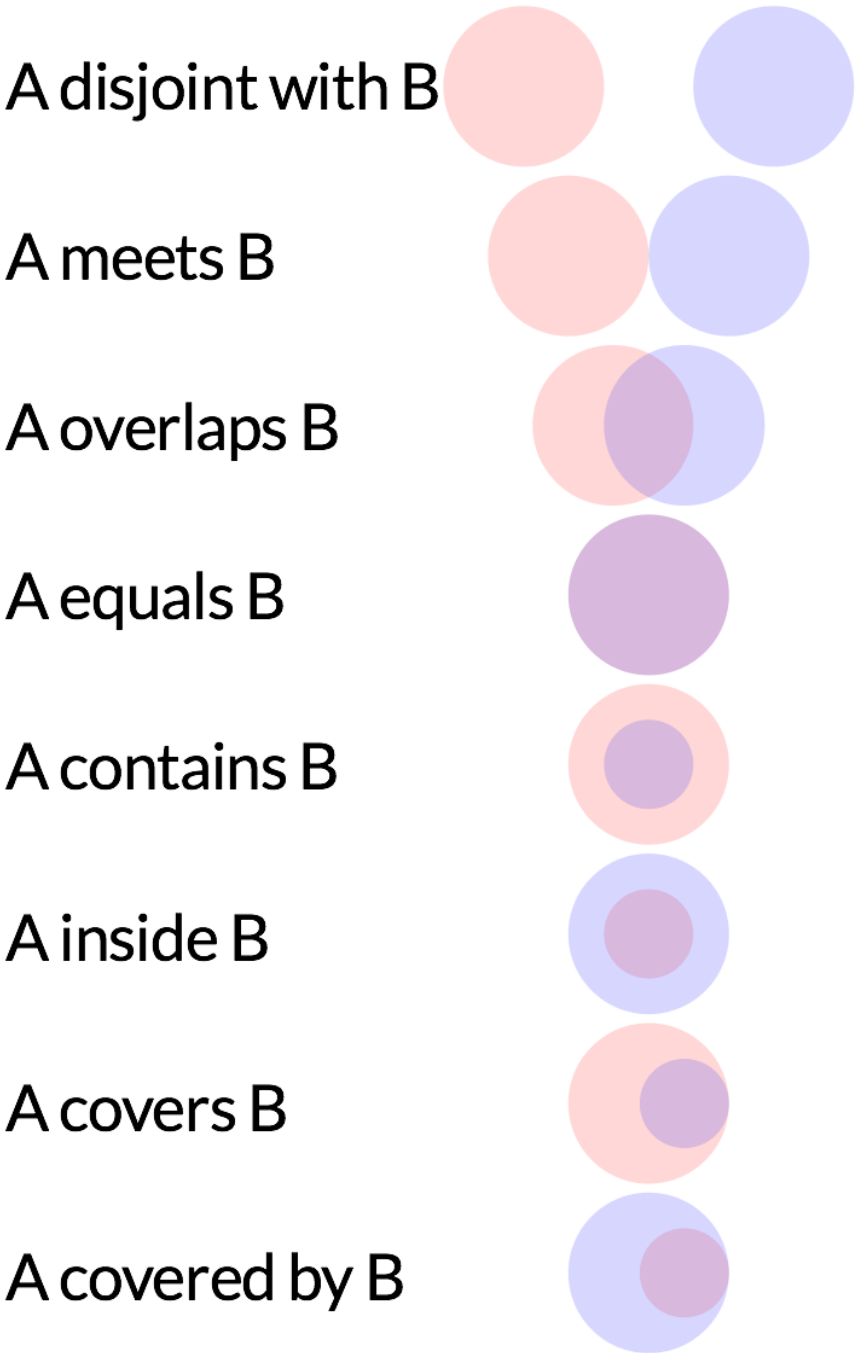
- Algorithms
  - What is an algorithm?
    - Literal, step-by-step procedures that allow a computer to solve a problem.
    - Someone writes them down in code.
      But they should be understood in human language, too.
  - For an algorithm, one can ask:
    - What does it seek to do?
    - What sort of data does it assume you will give it?
      - What data model used to represent data?
    - How does performance ‘scale up’?
      - Is the algorithm only feasible when the amount of data is small?

Spatial Analysis

How does a spatial operation *happen*? Algorithms.

- 'Big-O' notation is one way computer science uses to describe how fast or complex an algorithm is.
  - If an algorithm's solution is O(*n*) (which is read as follows: "has a big-Oh of *n*" or "is of order of *n*") for a problem involving *n* points, that means that the solution time tends to increase linearly with the number of points, *n*.
    - So, if the number of input points doubles, the solution time doubles.
  - O(n^2) means the solution time goes up with the square of the number of points, *n*.
    - So, if the number of input points doubles, the solution time quadruples.
  - O(*1*) means that the solution time is roughly constant and is independent of the number of points, *n*.
    - So, if the number of input points doubles, the solution time quadruples

Spatial Analysis: Considering both attribute and spatial relationships in analyzing data!

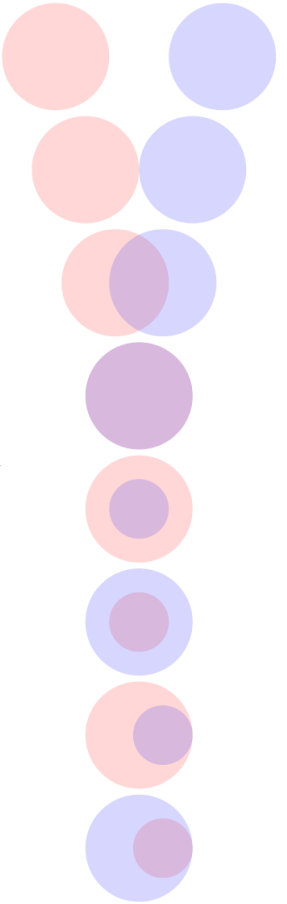


Egenhofer MJ & Franzosa RD 1991 Point-set topological spatial relations *International Journal of Geographical Information Systems* **5**:161-174.

http://southosullivan.com/geog80/lectures/02_Overview/#4/3

Finding algorithms to solve a problem: *How can you tell if a point is inside a polygon?*

- "Point in polygon?" is something your GIS calculates often.
  - Questions that this might help answer:
    Which state is this point in? Am I on land or on water?
    Or, figuring out many of these abstract relations →
  - Your GIS needs not only to know how to answer such a question, but to *know how to do it fast*.
  - Can you come up with procedures you can write down that a computer might be able to use to figure out if a point is in a polygon?



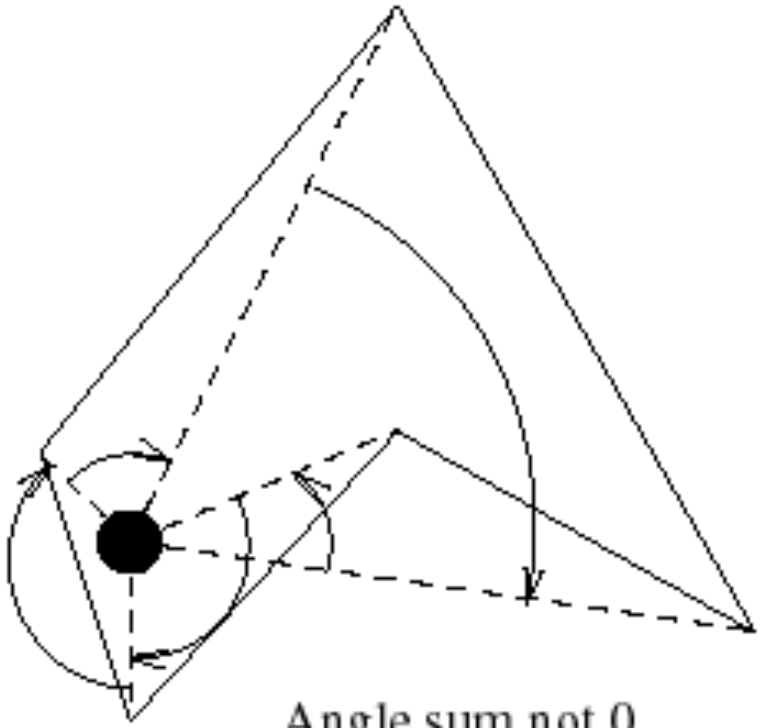
Finding algorithms to solve a problem:
How can you tell if a point is inside a polygon?

- Can you come up with procedures you can write down that a computer might be able to use to figure out if a point is in a polygon?

Finding algorithms to solve a problem:

How can you tell if a point is inside a polygon?

- An angle-summation algorithm: O(*average-number-vertices-per-poly*)



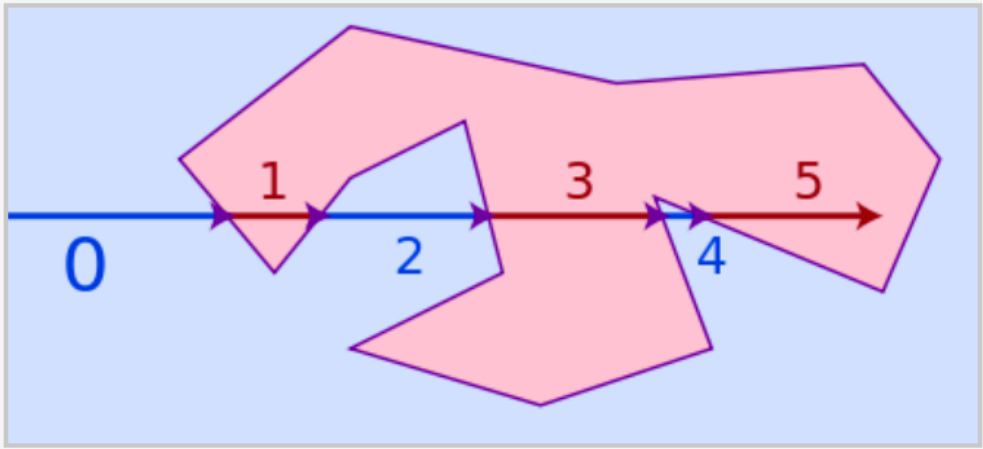
"The worst algorithm in the world for testing points is the angle summation method. It's simple to describe: sum the signed angles formed at the point by each edge's endpoints. If the sum is near zero, the point is outside; if not, it's inside (Figure 2). The winding number can be computed by finding the nearest multiple of 360 degrees. The problem with this scheme is that it involves a square root, arc-cosine, division, dot and cross product for each edge tested."

http://erich.realtimerendering.com/ptinpoly/

Finding algorithms to solve a problem:
How can you tell if a point is inside a polygon?

- A ‘Ray-casting’ or ‘Even-odd rule’ Algorithm

https://en.wikipedia.org/wiki/Point_in_polygon



The number of intersections for a ray passing from the exterior of the polygon to any point; if odd, it shows that the point lies inside the polygon. If it is even, the point lies outside the polygon; this test also works in three dimensions.

O(*average-number-vertices-per-poly*)

Finding algorithms to solve a problem: *How can you tell if a point is inside a polygon?*

What about when you have many polygons and many points?

Think about the case where you have a long list of (longitude, latitude) points and you want to know which county each belongs to.

- Do you really want to check each county polygon for whether each point is within it?
  - This “naïve solution” of endlessly repeating a point-in-polygon method (like the ray-casting algorithm) is *too expensive*: O(number-of-polygons *x* average-vertices-per-polygon *x* number-of-points-to-check)

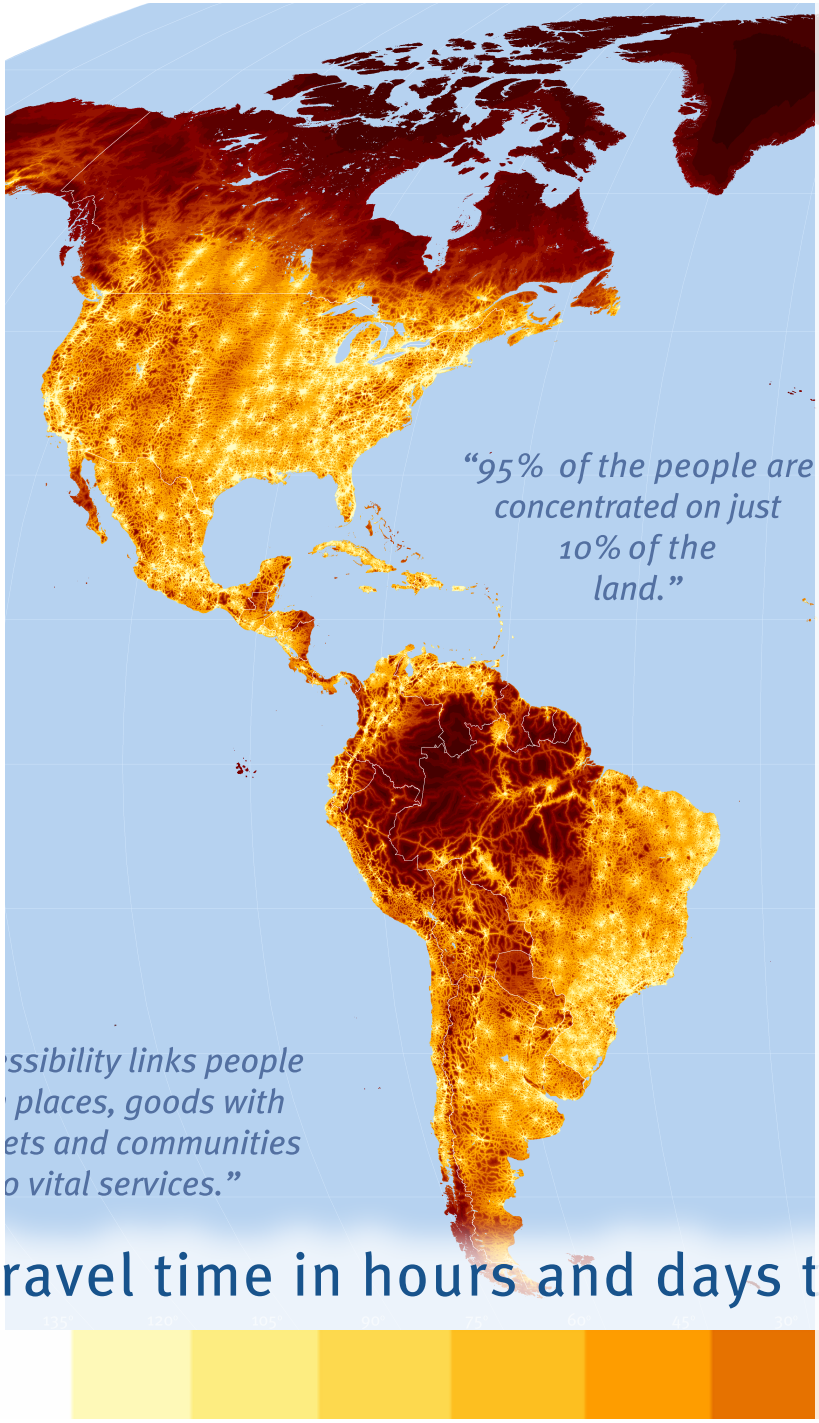
Finding algorithms to solve a problem: *How can you tell if a point is inside a polygon?*

What about when you have many polygons and many points?

- Clever algorithms are often interconnected with clever 'data structures': Ways of storing the data ahead of time that allow for easier computations.
- One simplistic possibility: Imagine making a raster where each cell's value is the ID of the polygon that contains that cell. Building that raster is 'costly', but then you can tell which polygon/county a new coordinate is in within O(*average-number-vertices-per-poly*) time!

This is the power of algorithms and data structures: they make things *feasible*, and thus, in a practical sense, *possible*.

- We wouldn't have GIS without spatial databases and their algorithms.
- We wouldn't have Google without an algorithm called 'Page Rank'



What algorithm can you use to figure out how far a point is from a big city?

This is not just a point-in-polygon style calculation. You need to calculate the shortest 'distance' measured in time. There are many paths from a point to all potential cities. And the 'shortest' path may not be a straight line.

You need three layers:

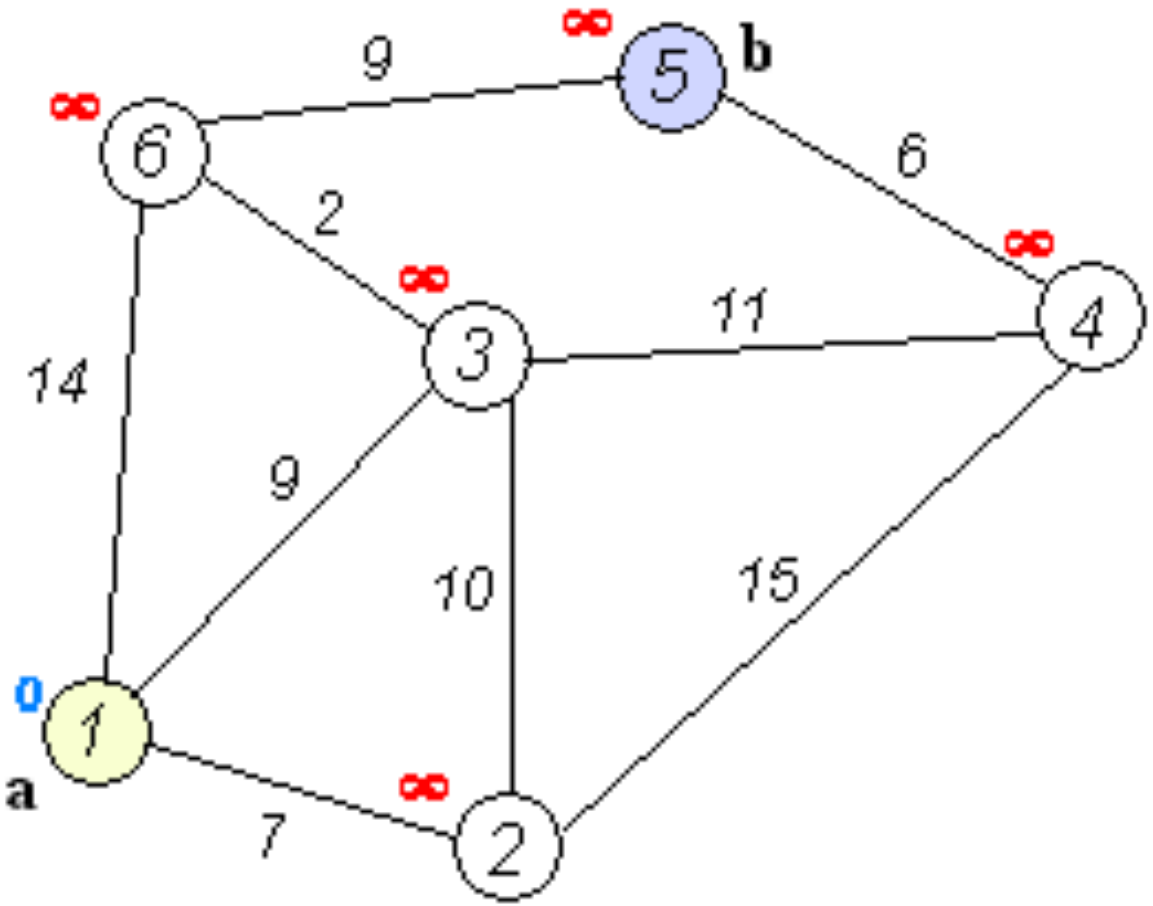
- Point layer
- City layer
- 'Cost' network showing how much time it takes to move from each place to each neighboring place. Speed varies by terrain and infrastructure.

You need a 'shortest path' algorithm. One classic one you learn about early in computation is 'Dijkstra's algorithm'. It's not perfect for this application, but it's close enough to be illustrative.

You then need to apply it repeatedly for each and every point on the earth so you can map the time-to-nearest-city for each, as in this map.

Go here to see the image below animated: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

"Dijkstra's algorithm to find the shortest path between *a* and *b*. It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with neighbors."



Our analogy:

Here, circles (nodes) are places/pixels.

Lines are ways to get from place to place.

Numbers next to lines are the times it takes to travel that line.

Node 1 is the pixel whose time to a nearby city we want to measure.

Node 5 is a 'nearby' large city.

We want to find how long the shortest path is.