# Nearest State/County Finder:

## Leveraging Geospatial Data for Efficient Query Processing

Aowei Zhao Haolin Ye Jiayu Wang Sen Wang

Department of Electrical and Computer Engineering

College of Engineering

11.21.2023

BOSTON UNIVERSITY

# Introduction

**Geospatial Data:**

Geographic location

Characteristics of natural or constructed features

Boundaries on Earth

**Dataset Source:**

Utilizing the US Board on Geographic Names dataset, featuring extensive geographical reference points across the United States

**Project Objective:**

Develop a system that quickly and accurately finds the nearest state or county in the US based on given **geographic coordinates**

**Project Challenge:**

Processing a **vast dataset** to respond to queries in **real-time**

# Key Tasks

Task 1 - Data Structure Implementation (KD-Tree)

- Space-partitioning data structure
- Organizing points in a k-dimensional space

Task 2 - Efficient Query Processing

- User queries for coordinates
- leveraging the KD-Tree
- Determine the corresponding state and county

# Data Lording

## Implementing KD-Tree for Data Loading

**KD-Tree:**

- **Latitude**
- **Longitude**

Code Snippet:

```
class KDTreeNode:
    def __init__(self, point, left=None, right=None):
        self.point = point  # Point containing [latitude, longitude]
        self.left = left
        self.right = right
```

# Query Processing

**Handling User Queries:**

- Accepting Queries
- Nearest Neighbor Search

Code Snippet:

```
// Utility function to calculate the distance between two points

double distance(const Point& p1, const Point& p2) {
    double x = (p2.lon - p1.lon) * std::cos((p1.lat + p2.lat) / 2);
    double y = p2.lat - p1.lat;
    return std::sqrt(x * x + y * y) * 6371;  // Earth's radius in km
```

# Distance Calculation

## Distance Calculation Methodology

Equirectangular Approximation Formula:

- $x = (\lambda_2 - \lambda_1) \cdot \cos\left(\frac{\phi_1 + \phi_2}{2}\right)$
- $y = \phi_2 - \phi_1$
- $\text{Distance} = \sqrt{x^2 + y^2} \cdot R$

Implementation:

```
def equirectangular_distance(lat1, lon1, lat2, lon2):
    R = 6371  # Earth's radius in km

    x = (math.radians(lon2) - math.radians(lon1)) * math.cos(0.5 * (math.radians(lat1) +
            math.radians(lat2)))

    y = math.radians(lat2) - math.radians(lat1)

    return math.sqrt(x*x + y*y) * R
```

# THANK YOU

# Q&A

BOSTON
UNIVERSITY