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

12.7.2023

BOSTON UNIVERSITY

Overview

Geospatial Data:

Geographic location

Characteristics of natural or constructed features

Boundaries on Earth

Workflows:

Given the locations of cities and counties in the US as reference points

User could enter a latitude and longitude to find the nearest counties

Return the nearest K(1<=K<=10) counties and their 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**

BOSTON UNIVERSITY

Approach

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

BOSTON UNIVERSITY

Data Loading

Represents geographical location

- State, county
- Geographical coordinates

```
struct Province_data{
    string state;
    string county;
    double latitude;
    double longitude;
};
```

A basic binary tree node structure

- Stores *Province_data*
- Pointers to its left and right child nodes.

```
struct Node{
    Province_data point;
    Node* left;
    Node* right;
};
```

Distance Calculation Methodology

Equirectangular Approximation Formula:

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

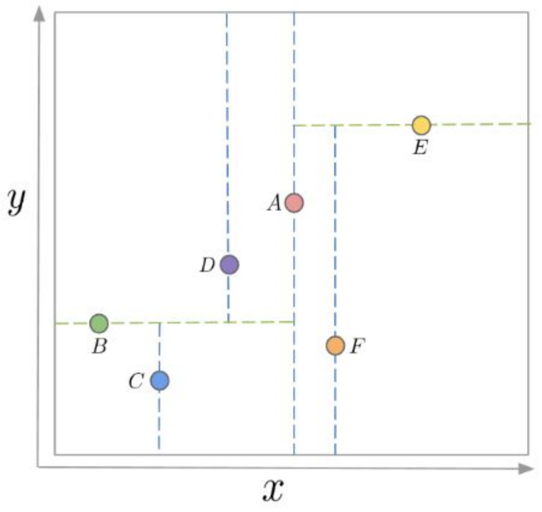
Implementation:

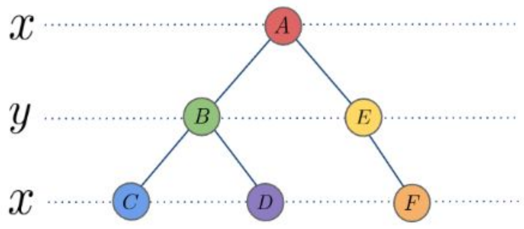
```
double distance_calculator(const Province_data &d1, Province_data &d2){
    const double R = 6371;

    double x = (d2.longitude - d1.longitude) * cos((d1.latitude + d2.latitude) / 2);
    double y = d2.latitude - d1.latitude;
    double d = sqrt(x*x + y*y) * R;
    return d;
};
```

BOSTON UNIVERSITY

Build KD-Tree

- **Data structure for organizing points in a K-dimensional space**
- **Binary search tree where data in each node is a K-Dimensional point in space**





Step 1: Choose the middle node on the x-axis and draw a vertical line.

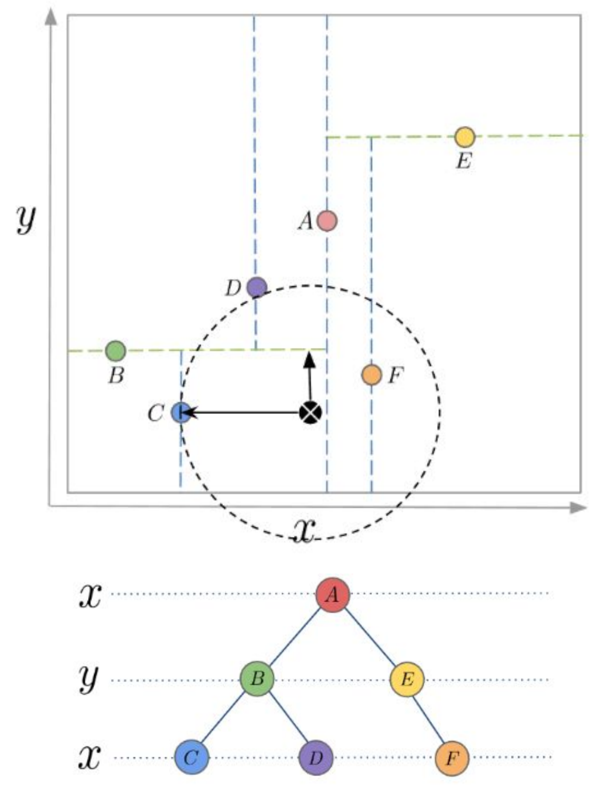
Step 2: Choose the middle node on the y-axis and draw a horizontal line.

Step 3: Repeat steps above until all the nodes have drawn lines.

```
int mid = (start + end) / 2;
sort(points.begin() + start, points.begin() + end + 1,
        [level](const Province_data& a, const Province_data& b) {
            return comparePoints(a, b, level);
        });

Node* root = newNode(points[mid]);
root->left = buildKdTree(points, start, mid - 1, level + 1);
root->right = buildKdTree(points, mid + 1, end, level + 1);
return root;
```

BOSTON UNIVERSITY

KD-Tree Search



$$d(\otimes, C) > |B - C|$$

Step 1: DownSearch: Compare from the root to bottom: X-Y-X-Y-X…

Step 2: Calculate the distance. Update or Discard.

Step 3: Upsearch: Go to the upper level. Calculate the distance.

Step 4: Decide whether to search other subtrees

```
priority_queue<pair<double, Province_data>, vector<pair<double, Province_data>>, CompareDistance> minHeap;

function<void(Node*)> search = [&](Node* current){
    if (current == nullptr)
        return;

    double distance = distance_calculator(current->point, query);
    minHeap.push(make_pair(distance, current->point));
    if (minHeap.size() > k)
        minHeap.pop();
    bool isLeft = (query.latitude < current->point.latitude);
    if ((isLeft && current->left) || (!isLeft && current->right)) {
        if (isLeft)
            search(current->left);
        else
            search(current->right);
    }
    double currentAxisDistance = (isLeft) ? abs(query.latitude - current->point.latitude) : abs(query.longitude - current->point.longitude);
    if (minHeap.size() < k || currentAxisDistance < minHeap.top().first) {
        if (isLeft)
            search(current->right);
        else
            search(current->left);
    }
};

search(root);
while (!minHeap.empty()) {
    k_nearest_neighbors.push_back(minHeap.top().second);
    minHeap.pop();
}
reverse(k_nearest_neighbors.begin(), k_nearest_neighbors.end());
return k_nearest_neighbors;
```

Linear Search for K-NN

Unlike the KD-Tree method, this linear approach compares the query point with every point in the dataset to find the k nearest neighbors. It's simpler but less efficient for large datasets.

```
vector<Province_data> liner_nearest_neighbor(const vector<Province_data> &Data, Province_data target, int k){
    auto cmp = [target](const Province_data& a, const Province_data& b) {
        auto distanceToA = distance_calculator(target, a);
        auto distanceToB = distance_calculator(target, b);
        return distanceToA < distanceToB;
    };
    priority_queue<Province_data, vector<Province_data>, decltype(cmp)> min_heap(cmp);
    for (auto data : Data){
        if (min_heap.size() < k){
            min_heap.push(data);
        }
        else{
            double dist = distance_calculator(target, data);
            if (dist < distance_calculator(target, min_heap.top())){
                min_heap.pop();
                min_heap.push(data);
            }
        }
    }
    vector<Province_data> k_nearest_neighbors;
    while(!min_heap.empty()){
        k_nearest_neighbors.push_back(min_heap.top());
        min_heap.pop();
    }
    reverse(k_nearest_neighbors.begin(), k_nearest_neighbors.end());
    return k_nearest_neighbors;
};
```

BOSTON UNIVERSITY

Result

```
Please enter the latitude and longitude: 123.456 99.99
Elapsed time: 0.000285608 seconds
(AK, Copper River)  (AK, Ketchikan Gateway)  (AK, Kusilvak)  (AK, Prince of Wales-Hyder)  (AK, Aleutians East)
(AK, Matanuska-Susitna)  (IA, Floyd)  (WY, Niobrara)  (IA, Chickasaw)  (WI, Dane)
Elapsed time: 0.000465209 seconds
(AK, Copper River)  (AK, Ketchikan Gateway)  (AK, Kusilvak)  (AK, Prince of Wales-Hyder)  (AK, Aleutians East)
(AK, Matanuska-Susitna)  (IA, Floyd)  (WY, Niobrara)  (IA, Chickasaw)  (WI, Dane)
```

```
Please enter the latitude and longitude: 1.4567 9.1453
Elapsed time: 0.000466117 seconds
(FL, Charlotte)  (TX, Kenedy)  (FL, Glades)  (FL, Palm Beach)  (FL, Lee)
(TX, Zapata)  (FL, Hendry)  (TX, Brooks)  (FL, Martin)  (TX, Jim Hogg)
Elapsed time: 0.000615125 seconds
(FL, Charlotte)  (TX, Kenedy)  (FL, Glades)  (FL, Palm Beach)  (FL, Lee)
(TX, Zapata)  (FL, Hendry)  (TX, Brooks)  (FL, Martin)  (TX, Jim Hogg)
```

Conclusion

KD-Tree vs. Linear Scanning

KD-Tree is generally more efficient for large datasets, while Linear Scanning is simpler but less efficient.

Data Structures and Algorithms

The KD-Tree construction uses recursive median finding and space partitioning

The search algorithm uses a priority queue to efficiently find the k-nearest neighbors

Efficiency and Accuracy

While KD-Trees offer greater efficiency for larger datasets, they come with the complexity of implementation.

BOSTON UNIVERSITY

THANK YOU

Q&A