Spatial Data Indexing using Grid-Based Method and KD-Tree Method

Kaichun Mo
Task 2: Spatial Data Indexing

Task 2

- Find the nearest ATM round this building?
- How many Chinese restaurants within 500 meters of the 拖鞋门?
Task 2: Methods

- Brute-Scan.
- Grid-Based.
- KD-Tree.

- Quad-Tree.
  - Query is hard.
- R-Tree.
  - Construction is hard.

**Common Idea:** To be classified by CATEGORY!
Grid-Based Method: Construction

```
vector<list<POI*>> vsects;
```
Grid-Based Method: Rectangle Range Query

- UpperLeft Point & BottomRight Point.
Grid-Based Method: Circle Range Query

- Pruning some impossible grids out. (Little Optimization)
Grid-Based Method: KNN Range Query

- Figure out the top-K Nearest Neighbors.
- Use a maximal heap to maintain. $a[k]$ is the current worst.
- Round by round exploring until

$$\min_{\text{new grid } x} (\text{dist}(x, \text{center})) > a[k] \quad (1)$$
KD-Tree Method: Construction

```
struct kdNode
{
    POI* separator;
    bool flag, isLeaf;
    kdNode *left, *right;
};
```
KD-Tree Method: Rectangle Range Query

- Recursively invoke left part and right part of the original range query, if it intersects with the separator.
KD-Tree Method: KNN Range Query

- Maintain a maximal heap.
KD-Tree Method: KNN Range Query

- Maintain a maximal heap.
- Each time, to explore nearest part if necessary.

```c
KNN(kdNode *p)
{
    if (p->isLeaf) linearScan(p);
    else
    {
        d1 = dist(p->left, queryPoint);
        d2 = dist(p->right, queryPoint);

        part1 = (d1 < d2?) p->left: p->right;
        part2 = (d1 < d2?) p->left: p->right;

        if (dist(part1, queryPoint) < a[k]) KNN(part1);
        if (dist(part2, queryPoint) < a[k]) KNN(part1);
    }
}
```
KD-Tree Method: KNN Range Query

- Maintain a maximal heap.
- Each time, to explore nearest part if necessary.

```c
KNN(kdNode *p)
{
    if (p->isLeaf) linearScan(p);
    else
    {
        d1 = dist(p->left, queryPoint);
        d2 = dist(p->right, queryPoint);

        part1 = (d1 < d2) ? p->left : p->right;
        part2 = (d1 < d2) ? p->right : p->left;

        if (dist(part1, queryPoint) < a[k]) KNN(part1);
        if (dist(part2, queryPoint) < a[k]) KNN(part1);
    }
}
```

- Pruning by little optimization.
Performance Analysis

- Grid-Based Method:
  - Direct Access, Time Efficiency!
  - Large Memory Needed!
  - Dealing with Unbalanced POI Distribution!

- KD-Tree Method:
  - Distribution Free! Heuristic Method!
  - Tolerant Memory Consumption!
  - Undirected Access!
  - $O(\ln n)$ time per visiting!
Performance Analysis

- Grid-Based Method:
  - Direct Access, Time Efficiency!

- KD-Tree Method:
  - Distribution Free!
  - Heuristic Method!
  - Tolerant Memory Consumption!
  - Undirected Access!
  - \( O(\ln n) \) time per visiting!
Performance Analysis

- **Grid-Based Method:**
  - Direct Access, Time Efficiency!
  - Large Memory Needed!

- **KD-Tree Method:**
  - Distribution Free!
  - Heuristic Method!
  - Tolerant Memory Consumption!
  - Undirected Access!
  - $O(\log n)$ time per visiting!
Performance Analysis

- Grid-Based Method:
  - Direct Access, Time Efficiency!
  - Large Memory Needed!
  - Dealing with Unbalanced POI Distribution!
Performance Analysis

- **Grid-Based Method:**
  - Direct Access, Time Efficiency!
  - Large Memory Needed!
  - Dealing with Unbalanced POI Distribution!
Performance Analysis

- **Grid-Based Method:**
  - Direct Access, Time Efficiency!
  - Large Memory Needed!
  - Dealing with Unbalanced POI Distribution!

- **KD-Tree Method:**
Performance Analysis

- **Grid-Based Method:**
  - Direct Access, Time Efficiency!
  - Large Memory Needed!
  - Dealing with Unbalanced POI Distribution!

- **KD-Tree Method:**
  - Distribution Free! Heuristic Method!
Performance Analysis

Grid-Based Method:
- Direct Access, Time Efficiency!
- Large Memory Needed!
- Dealing with Unbalanced POI Distribution!

KD-Tree Method:
- Distribution Free! Heuristic Method!
- Tolerant Memory Consumption!
Performance Analysis

- **Grid-Based Method:**
  - Direct Access, Time Efficiency!
  - Large Memory Needed!
  - Dealing with Unbalanced POI Distribution!

- **KD-Tree Method:**
  - Distribution Free! Heuristic Method!
  - Tolerant Memory Consumption!
  - Undirected Access! $O(\ln n)$ time per visiting!
Numerical Experiment

- Regard all categories as one! 2020 POIs in all.
- Each leaf contains at most 3 POIs.

<table>
<thead>
<tr>
<th>Method</th>
<th>Rec Query</th>
<th>Cir Query</th>
<th>10-NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute Scan</td>
<td>2020</td>
<td>2020</td>
<td>2020</td>
</tr>
<tr>
<td>5 × 5 G-B</td>
<td>619</td>
<td>237</td>
<td>244</td>
</tr>
<tr>
<td>10 × 10 G-B</td>
<td>128</td>
<td>202</td>
<td>227</td>
</tr>
<tr>
<td>50 × 50 G-B</td>
<td>255</td>
<td>110</td>
<td>156</td>
</tr>
<tr>
<td>KD-Tree</td>
<td>101</td>
<td>117</td>
<td>68</td>
</tr>
</tbody>
</table>

- Use my special way to gauge the time efficiency!
Thank you for listening!