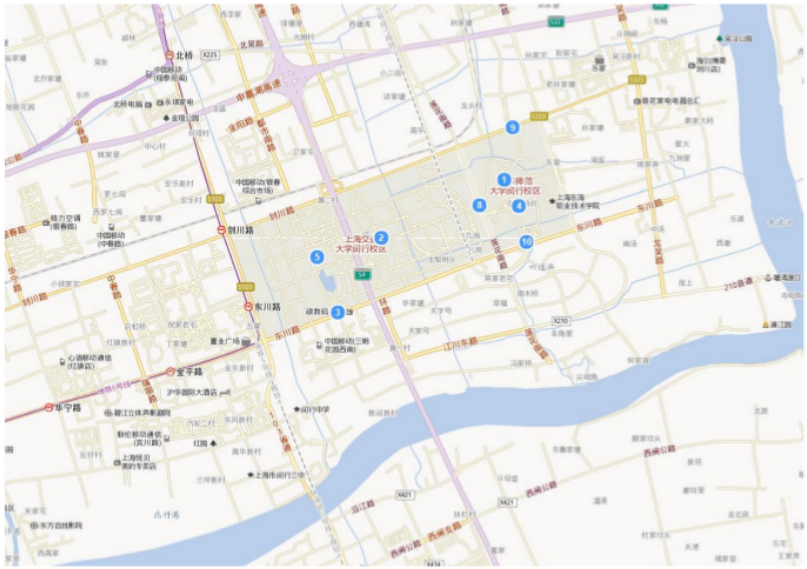
Spatial Data Indexing using Grid-Based Method and KD-Tree Method

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?



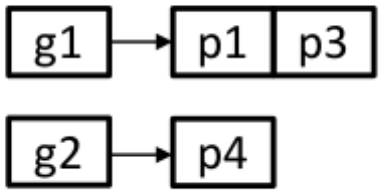
Task 2: Methods

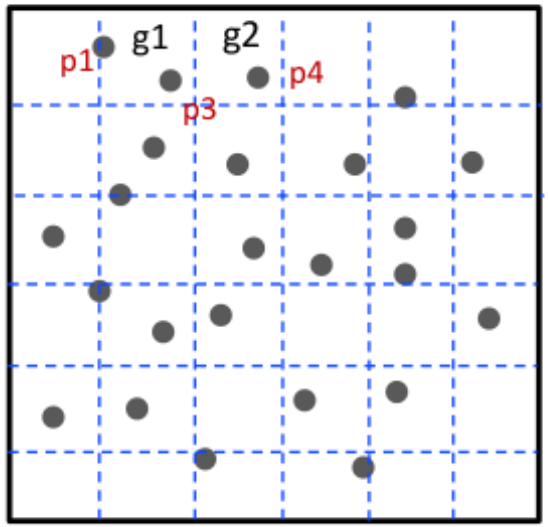
- Brute-Scan.
- Grid-Based.
- KD-Tree.

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

- **Common Idea:** To be classified by CATEGORY!

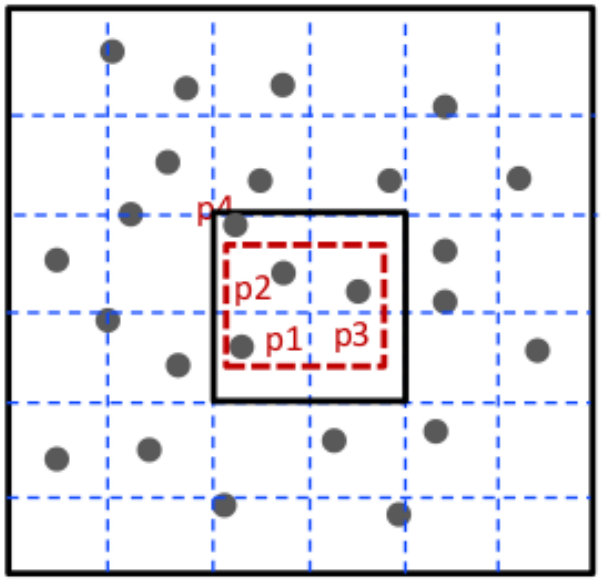
Grid-Based Method: Construction





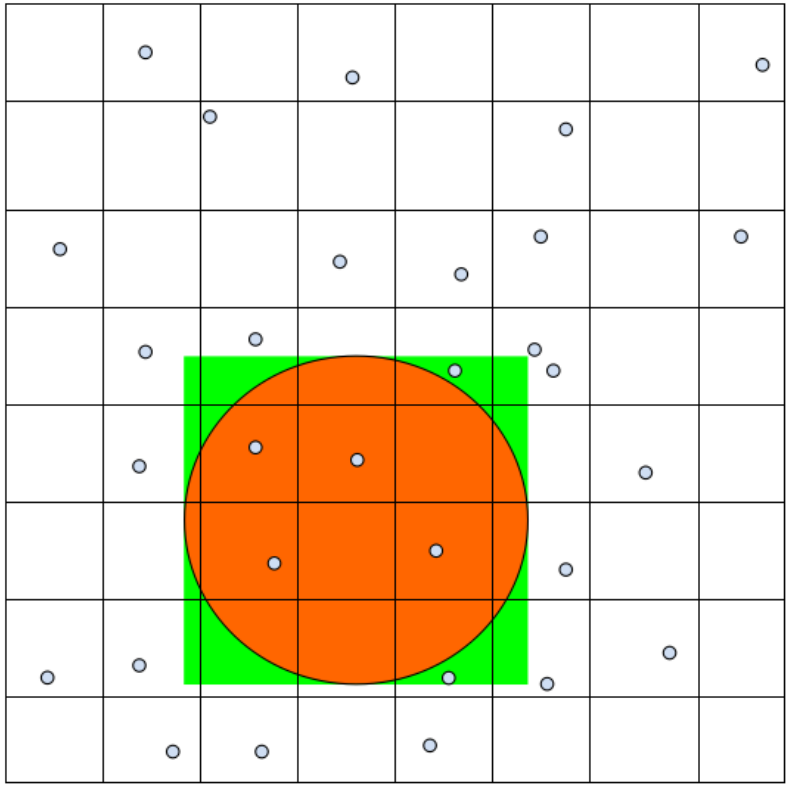
- `vector<list<POI*>*> vects;`

Grid-Based Method: Rectangle Range Query



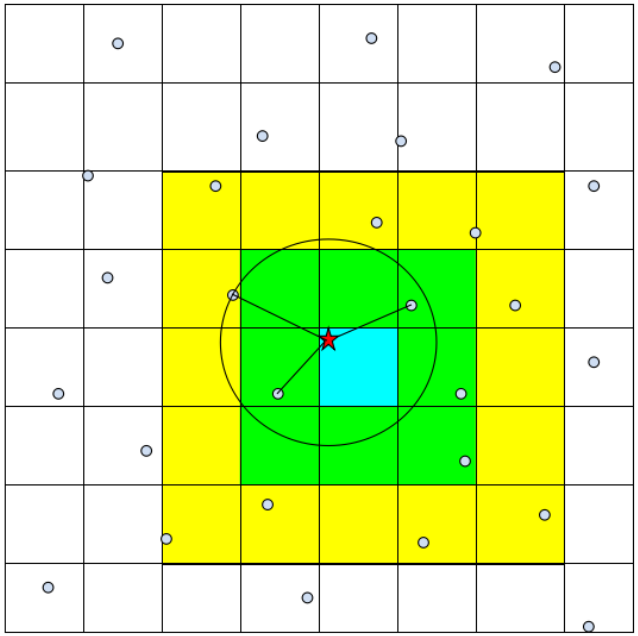
- UpperLeft Point & BottomRight Point.

Grid-Based Method: Circle Range Query



- Pruning some impossible grids out.(Little Optimization)

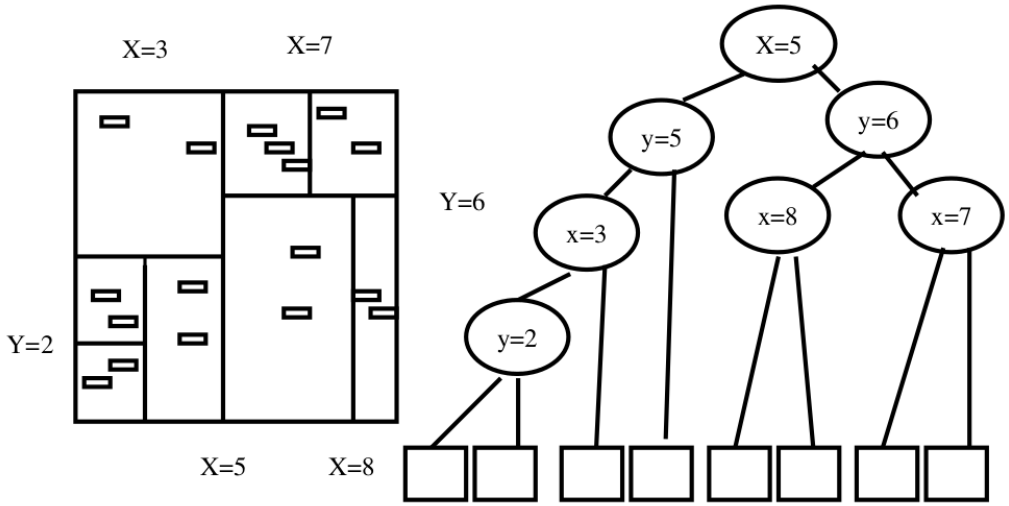
Grid-Based Method: KNN Range Query



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

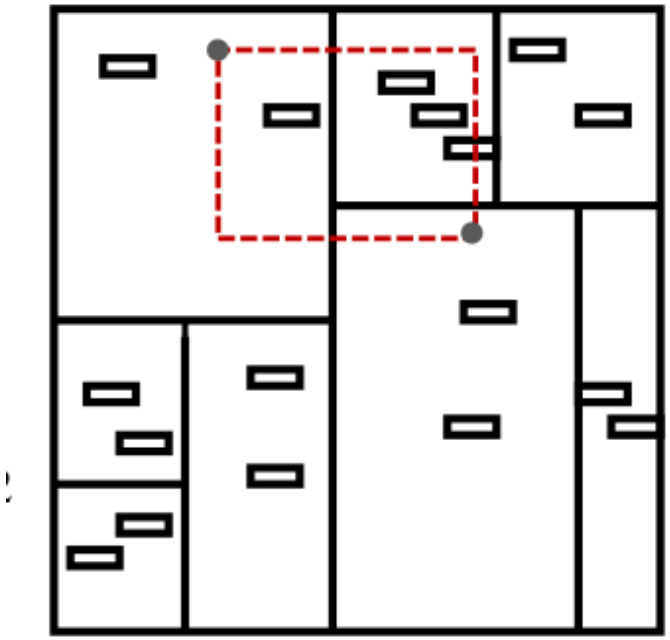
$$\min_{\text{new grid } x} (dist(x, center)) > a[k] \tag{1}$$

KD-Tree Method: Construction




```
1  struct kdNode
2  {
3          POI* separator;
4          bool flag, isLeaf;
5          kdNode *left, *right;
6  };
```

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.

```
1  KNN(kdNode *p)
2  {
3          if(p->isLeaf) linearScan(p);
4          else
5          {
6                  d1 = dist(p->left, queryPoint);
7                  d2 = dist(p->right, queryPoint);
8
9                  part1 = (d1 < d2?) p->left: p->right;
10                 part2 = (d1 < d2?) p->left: p->right;
11
12                 if(dist(part1, queryPoint) < a[k]) KNN(part1);
13                 if(dist(part2, queryPoint) < a[k]) KNN(part1);
14         }
15 }
```

KD-Tree Method: KNN Range Query

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

```
1  KNN(kdNode *p)
2  {
3          if(p->isLeaf) linearScan(p);
4          else
5          {
6                  d1 = dist(p->left, queryPoint);
7                  d2 = dist(p->right, queryPoint);
8
9                  part1 = (d1 < d2?) p->left: p->right;
10                 part2 = (d1 < d2?) p->left: p->right;
11
12                 if(dist(part1, queryPoint) < a[k]) KNN(part1);
13                 if(dist(part2, queryPoint) < a[k]) KNN(part1);
14         }
15 }
```

- Pruning by little optimization.

Performance Analysis

- Grid-Based Method:

Performance Analysis

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

Performance Analysis

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

Performance Analysis

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

Performance Analysis

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

Performance Analysis

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

Performance Analysis

- Grid-Based Method:
  - Direct Access, Time Efficiency!
  - Large Memory Needed!
  - Dealing with Unbalanced POI Distribuion!
- KD-Tree Method:
  - Distribution Free! Heuristic Method!

Performance Analysis

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

Method	Rec Query	Cir Query	10-NN
Brute Scan	2020	2020	2020
5×5 G-B	619	237	244
10×10 G-B	128	202	227
50×50 G-B	255	110	156
KD-Tree	101	117	68

- Use my special way to gauge the time efficiency!

Thank you for listening!