

Introduction

Please chose the service that you want: SUBMIT

Computing type:

- ☒ Closest available facility
- ☐ All available facilities within ____ km

Facility type:

汽车服务/ Vehicle Service 所有 所有

MAP and result:

Select location on the map

Your location:
LatLong:31.061,121.516

Facility list:

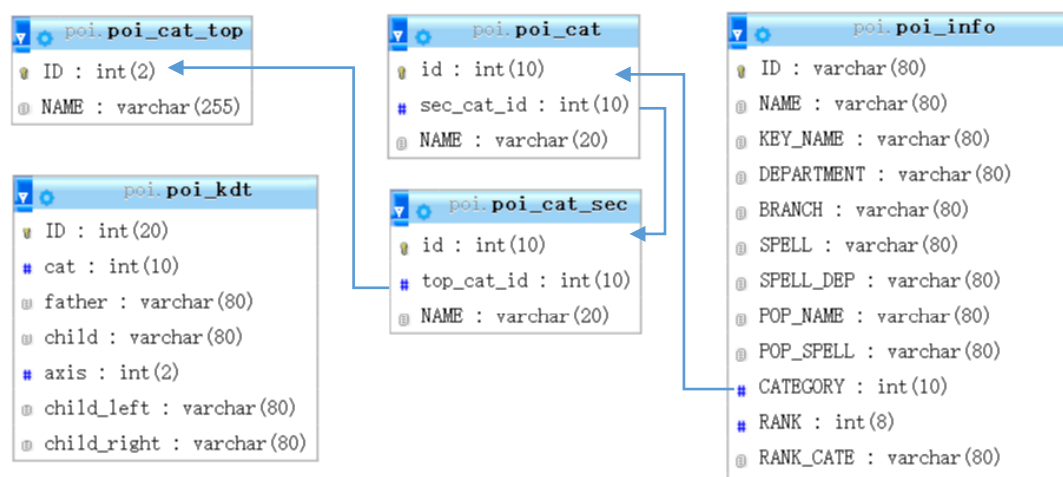
- 1:闵行区北桥制盒厂
上海市 上海市 闵行区
- 2:上海灯塔公司
上海市 上海市 闵行区 沪闵路
- 3:上海关星实业公司
上海市 上海市 闵行区

My final app is a web application based on PHP + JavaScript (jQuery included) + MySQL, which is accessible on the internet with the link : <http://cais1991.19.5v.cc/> . the link is functioning before 2015, Jan, 28th. Users are able to choose multiple levels of category, the nearest or all available facilities within range, and their location as well. Searching algorithm are based on KD-Tree. By using ajax, all the calculation results can be demonstrated on the FrontPage dynamically.

And in the following sections, details about the implementation will be included.

Data factory


In this part, all the data stored for the application will be introduced in detail. The data construction is shown below:



Data in the txt file and excel file was processed and stored in the database. Where the table poi_info is from the txt file and poi_cat, poi_cat_sec, poi_cat_top correspond to the different level of categories. All these are for supporting the user interface.


While poi_kdt is the kd-tree result stored in form of (father, child) arch, which is generated via R, the details will be covered later. The fields child_left and child_right are the child node of "child", which are here for the sake of the kd-tree structure.


File structure


 data


 img


 js


 ajax.php

 config.php

 functions.php

 index.php

 style.css

 tree-class.php

The webpage is edited with the help of Dreamweaver. The file structure is on the left and the files are attached.

Index.php is the main page and the only showing page. Part of the unfavorable actions are forbidden through unable controls.

The ajax.php is the main file for doing urban computing, config.php is for storing the database parameter and functions.php for accessory functions namely dealing with json encode, decode in this application and calculate sphere distance between two location using latitude and longitude.

In the tree-class.php file, the class of tree and tree node are included and related search function.

In the js folder, there are scripts for map demonstration and other scripts used in the user interface. Including the dynamic generation of category combo.

KD tree

After balancing all the features, k-d tree is used for the urban computing in this case.

Tree data structure in php file

In implementation research algorithm, the data structure array is used for the tree structure, where the node are set as the key of element, and the axis, the father node and child node are included in entity. When axis equals -1, the node is a leaf node and both child_left and child_right are "0".

```
SELECT b.child as node, b.father, b.axis, b.child_left, b.child_right, a.x_coord, a.y_coord
FROM `poi_info` AS a
JOIN poi_kdt AS b ON a.ID = b.child
```

Figure 1 Tree data construction from MySQL database

Tree generation

The generation of k-d tree is implemented in R, which is attached in the data file. The method is the classical one. First, all nodes are split into 183 groups according to their category, and foreach group, a kd-tree is generated. Variance for coord_x and coord_y are generated for determining which axis to start from, then the median is found. In the case, for even sample, the location which is closest to the median in the axis will be set as the node for splicing, for multiple median, the first

one will be chosen.

All the result will be stored in the database in the form of (id, category, father, child) where id is auto increase and category stands for the category of the location. Child and father are the ID of locations, which are strings. And for the root node, ID is set to be "0".

Afterwards, child_right and child_left are updated from the result generated in the last step, the sql code are shown below. Children node are assigned to left or right according to the comparison with the father node's position on the corresponding axis.

```
CREATE TABLE tmpTx(
node VARCHAR( 80 ) ,
child VARCHAR( 80 ) ,
po INT( 2 )
) SELECT a.child AS node, b.child AS child, IF( d.x_coord > c.x_coord, 1 , -1 ) AS po
FROM `poi_kdt` AS a
JOIN poi_kdt AS b ON a.child = b.father
JOIN poi_info AS c ON a.child = c.ID
JOIN poi_info AS d ON b.child = d.ID
WHERE a.axis =1;
UPDATE `poi_kdt` JOIN tmpTx ON poi_kdt.child = tmpTx.node SET `child_left` = tmpTx.child
WHERE tmpTx.po <0;
UPDATE `poi_kdt` JOIN tmpTx ON poi_kdt.child = tmpTx.node SET `child_right` = tmpTx.child
WHERE tmpTx.po >0;
UPDATE `poi_kdt` JOIN tmpTy ON poi_kdt.child = tmpTy.node SET `child_left` = tmpTy.child
WHERE tmpTy.po <0;
UPDATE `poi_kdt` JOIN tmpTy ON poi_kdt.child = tmpTy.node SET `child_right` = tmpTy.child
WHERE tmpTy.po >0;
DROP TABLE tmpTx; DROP TABLE tmpTy;
```

Figure 2 related Sql code

Nearest searching

The steps can be described in the following pseudocode.

```
Step 1: assign the target point  $x^*(x_0, y_0)$  to a leaf node according to the split rule for
node and branches and record all the nodes visited, update the leaf node as the closest node
and the closest distance d.
Step 2: searching back, for any given visited node if the split axis have intersections with
the circle with node  $x^*$  as center and d as radius, then go process the step 1 with the sub
tree start from other child and repeat; else delete this visited node from record and continue
searching backward.
Step 3: if the record set is empty, the current closest is the closest point.
```

Range searching

The range searching is an easy branch cutting procedure, the detailed steps can be described in the following pseudocode.

Step 1: Start from root node. Denote the circle with target node x^* as center and limited distance d^* for radius as circle C .

Step 2: For any given node, first calculating whether the splitting axis have intersections with the circle C (considering that if there is no intersection). If no, cut the further branch, and continue the procedure with the closer branch; else calculate whether the node is within the circle, if yes, add it to the point set, and continue the procedure on both branches.

All implementation code files are attached.