P1: OpenStreetMap Beijing with SQL

地图范围

本次项目所选地图是**北京市地图**。

文件名为：sample.osm

选择北京市地图的原因：

北京是祖国首都，多次的旅行经历让我非常有兴趣。

地图遇到的问题

1. 街道名称不统一

查看街道地址时，有的用的中文，有的用的拼音，有的用的英文。

2. 审查标签类型时，出现异常标签

使用正则表达式查看标签类型时，发现有异常标签。

3. 审查时发现部分邮政编码有问题

审查邮政编码时，发现部分邮政编码不是6位数字，而中华人民共和国邮政编码应该为6位

问题解决如下：

1. 街道名称不统一

解读 sample.osm，发现街道地址在<way>标签下的 <tag> 标签中，如下：

```
<tag k="addr:street" v="农展馆南里" />
<tag k="addr:street" v="四王府村" />
<tag k="addr:street" v="科荟路" />
```

在此，用 street_name.py 读取街道地址如下：

```
import xml.etree.cElementTree as ET
filename = 'sample.osm'

def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")

for event, elem in ET.iterparse(filename,events=("start",)):
    if elem.tag == "node" or elem.tag == "way":
        for tag in elem.iter("tag"):
            if is_street_name(tag):
                print tag.attrib['v']
```

运行结果如下：

```
学院路
牛街
金台路
Gongti Beilu (behind Pacific Century Place), Chaoyang District
中关村南二条
西什库大街
光华路
酒仙桥路
金宝街
Nanluoguxiang
广渠门内大街
平安大街
农展馆南里
四王府村
科荟路
鲁谷东街
石佛营路
Maizidian Street
手帕口南街
日坛路
平乐园
中滩村大街 8号院
```

从上面的运行结果可以看出，街道地址描述

既有中文，如日坛路，

也有拼音，如Nanluoguxiang，Gongti Beilu (behind Pacific Century Place)，Chaoyang District，

也有英文，如Maizidian Street，

对于街道地址，最好能够统一用中文附带英文来表示，类似于google地图，能规范，方便更多人使用。

2. 审查标签类型时，出现异常标签

本处主要是为了审查标签类型，并尝试处理有问题的标签。

```
<tag k="addr:street" v="农展馆南里" />
<tag k="addr:street" v="四王府村" />
<tag k="addr:street" v="科荟路" />
```

如上面的代码所示，本例主要是审查 `<tag>` 标签下的 `k` 属性类型。

在此，用正则表达式 `r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]'` 判断 `tag['k']` 中是否有特殊字符 `space . & + , # $ % ;`等

使用 `tags.py` 编码如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import xml.etree.cElementTree as ET
import re

lower = re.compile(r'^([a-z]|_)*$')    #全部小写
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')    #有冒号"："
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')     #有特殊字符属于问题标钱类型

def key_type(element, keys):
    if element.tag == "tag":
        k_attrib = element.attrib["k"]
            if lower.match(k_attrib) is not None:
                keys["lower"] += 1
            elif lower_colon.match(k_attrib) is not None:
                keys["lower_colon"] += 1
            elif problemchars.search(k_attrib) is not None:
                keys["problemchars"] += 1
                print k_attrib    #打印出错的标签
            else:
                keys["other"] += 1
```

```
    return keys

def process_map(filename):
    keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}
    for _, element in ET.iterparse(filename):
        keys = key_type(element, keys)
    return keys

if __name__ == "__main__":
    key = process_map("sample.osm")
    print key
```

运行结果如下所示：

```
emergency shelter
{'problemchars': 1, 'lower': 33150, 'other': 114, 'lower_colon': 2390}
```

从上述运行结果可以看出，有问题的标签类型有一个，打印出来是 `emergency shelter`**，这里的特殊字符是空格，可以将空格去掉，使用"_"短横线将单词链接起来。**

使用函数如下：

```
temp_attrib = k_attrib.split(" ") #将k_attrib按空格分开
k_attrib_new = "_".join(temp_attrib) #将分开后的list用“_”链接成一个新字符串
print k_attrib_new
```

运行结果

```
emergency_shelter
```

将修正后的标签存储到数据中，这部分将在 data.py 文件中进行修改，方便将修改后的文件直接存入csv文件中。**由于特殊字符种类较多，本项目中，只出现了空格的情况，故处理中，只对空格进行修改，其他特殊字符暂不处理。**

修改代码如下：

```
# 判断如果标签中有特殊字符，则先将空格用短横线代替，
# 再次进行判断，如果用代替后没有特殊字符了，就将代替后的数据存入csv文件中，如果还有特殊字符，则不用再处理。
elif PROBLEMCHARS.search(key) is not None:
    temp = key.replace(' ', '_')
    print 'This:  ' + temp   #打印出代替后的类型
    if PROBLEMCHARS.search(temp) is None:   #如果用代替后没有特殊字符了，就将代替后的数据存入csv文件中
        temp_tag['type'] = 'regular'
        temp_tag['key'] = temp
        temp_tag['id'] = int(element.attrib['id'])
        temp_tag['value'] = child.attrib['v']
        tags.append(temp_tag)
```

查看打印结果可得到：

```
This:  emergency_shelter
```

查看生成的csv文件，可以在 ways_tags.csv文件中找到 `key`值为`emergency_shelter` 的值，说明修正完成。

3. 审查邮政编码是否准确

使用 `post_code.py` 来打印本文中的邮政编码，代码如下：

```
import xml.etree.cElementTree as ET
filename = 'beijing_china.osm'

def is_post_code(elem):
    return (elem.attrib['k'] == 'addr:postcode')
```

```
for event, elem in ET.iterparse(filename, events=("start",)):
    if elem.tag == "node" or elem.tag == "way":
        for tag in elem.iter("tag"):
            if is_post_code(tag):
                post_code = tag.attrib['v']
                if len(str(post_code)) != 6:
                    print post_code   #打印出出现异常的邮编
```

运行结果如下：

```
3208
10060
010-62332281
10080
10040
10043
```

从以上结果中可以看出，部分邮编出现了4位，5位，或者电话号码，均为异常邮编，在写入数据库的时候，应该放弃这些异常值。

在 data.py 文件中对邮编进行修改如下，只保留6位数的邮编，其他的邮编放弃。

```
if LOWER_COLON.match(key) is not None:
    temp = key.split(':')
    temp_tag['type'] = temp[0]
    temp_tag['key'] = ':'.join(temp[1:])
    temp_tag['id'] = int(element.attrib['id'])
    temp_tag['value'] = child.attrib['v']
    if temp_tag['key'] == 'postcode':   #当key值为postcode时，判断postcode的位数，满足6位，就加入到tags里面
        if len(temp_tag['value']) == 6:
            tags.append(temp_tag)
    else:   #当key值不为postcode时，直接加入tags里面
```

```
        tags.append(temp_tag)
```

通过上述代码，可以有效的保留邮编为6位的值，去掉不正确的邮编

将数据写入csv文件

按照“案例研究：OpenStreetMap数据[SQL]”中准备数据集的方法，将北京市地图相关数据读入csv文件中，请查看data.py中的代码

将北京市地图写入数据库

将上面处理过的csv文件写入 `beijing.db` 数据库中

创建数据库

创建 `beijing.db` 数据库

将csv文件导入数据表

用python代码将 `nodes.csv`, `nodes_tags.csv`, `ways.csv`, `way_nodes.csv`, `ways_tags.csv`这5个csv文件分别写进数据库 `beijing.db`中，对应表格分别为 `nodes`, `nodes_tags`, `ways`, `ways_nodes`, `ways_tags`。

python代码见 `import_nodes_csv.py` 文件，以下以 `nodes.csv` 导入 `nodes` 数据表为例，代码如下：

```
# coding=utf-8
import csv, sqlite3

con = sqlite3.connect("beijing.db")
con.text_factory = str

cur = con.cursor()

cur.execute('drop table if exists nodes')
```

```
nodes = '''create table nodes(
id Integer,
lat float,
lon float,
user Text,
uid Integer,
version Text,
changeset Integer,
timestamp Text);
'''

cur.execute(nodes)

with open('nodes.csv','rb') as fin:
    dr = csv.DictReader(fin)
    for row in dr:
        id_value = int(row['id'])
        lat_value = float(row['lat'])
        lon_value = float(row['lon'])
        user_value = str(row['user'])
        uid_value = int(row['uid'])
        version_value = str(row['version'])
        changeset_value = int(row['changeset'])
        timestamp_value = str(row['timestamp'])
        cur.execute('INSERT INTO nodes VALUES (?,?,?,?,?,?,?,?)',

(id_value,lat_value,lon_value,user_value,uid_value,version_value,changeset_valu
e,timestamp_value))

con.commit()
con.close()
```

其他csv文件的导入见相应的python文件，代码见`import_nodes——tags_csv.py`，`import_ways_csv.py`，`import_ways_nodes_csv.py`，`import_ways_tags_csv.py`

数据库基本信息：

查询数据库中的表格

```
sqlite> .tables
nodes       nodes_tags  ways        ways_nodes  ways_tags
```

从上面运行结果可以看出，该数据库中有5个表格。

查询唯一用户的数量

唯一用户是用字段`uid`来表示，其中`nodes`, `ways`表格中均有记录用户信息，这里需要查询的是唯一用户的数量，需要将 nodes 表格中的 uid 字段和 ways 表格中的 uid 取出，再去重，取总数。python代码如下：

```
# coding=utf-8
import csv, sqlite3
con = sqlite3.connect("beijing.db")
cur = con.cursor()

#提取 nodes 数据表中的独立用户数
query1 = "select uid from nodes group by uid order by uid"
cur.execute(query1)
nodes_uid = cur.fetchall()
print 'Unique uid in table nodes is:'
print len(nodes_uid)

#提取 ways 数据表中的独立用户数
query2 = "select uid from ways group by uid order by uid"
cur.execute(query2)
ways_uid = cur.fetchall()
print 'Unique uid in table ways is:'
print len(ways_uid)

#计算整个数据库中的独立用户数，需要将nodes数据表和ways数据表中的独立用户数相加并去重
for i in range(len(ways_uid)):
    if ways_uid[i] not in nodes_uid:
        nodes_uid.append(ways_uid[i])
```

```
unique_uid = len(nodes_uid)
print 'total unique users is:'
print unique_uid
```

打印结果如下：

```
Unique uid in table nodes is:
947
Unique uid in table ways is:
513
total unique users is:
1018
```

唯一用户的数量是1018.

节点 node 的数量

```
sqlite> select count(*) from nodes;
82648
```

节点数量为 82648

途径 way 的数量

```
sqlite> select count(*) from ways;
12328
```

途径的数量为12328

所选节点类型的数量

查看节点类型为“shop”的数量，此处主要是在 `nodes_tags` 中进行查询

```
sqlite> select key, count(*)
   ...> from nodes_tags
   ...> where key = 'shop'
   ...> group by key;
shop|149
```

节点类型为“shop”的数量有149个

关于数据集的其他想法

对于非英语国家，在地址描述上会显得比较杂乱，可以设置约束，某些字段要求用英文来描述，在增加本国语言描述的字段，方便使用。

有统一的约束条件之后，可以让不同用户编辑的时候，有统一的标准，避免每个用户都按照自己的标准来编辑，导致最终清理时，出现不统一的情况。