```
# -*- coding: utf-8 -*-
import xml.etree.ElementTree as ET
import re
import json

filename = "D:\Udacity\wien.osm"
tree = ET.parse(filename)
root = tree.getroot()

'''
When checking the small sample osm file, I found that in the "opening_hours"
tag, there are lots of different expressions, in the form of day and time or
time-only, some with PH(public holiday) but some not, etc. I want to find out
how many of them are available on Sundays,
so I modify the "opening_hours" data into "sun_opening" tag to show whether they
are open on Sunday or not.
'''
def sun_opening(element):
    if (element.tag == "node" or element.tag == "way") and element.find("tag")
!= None:
        tags = element.findall("tag")
        pat = re.compile('^\d{1,2}:\d{2} ?\- ?\d{1,2}:\d{2}$')
        for tag in tags:
            if "opening_hours" in tag.get("k"):
                if "Su" in tag.get("v") and "Su off" not in tag.get("v") and
"Su,PH off" not in tag.get("v") or pat.match(tag.get("v")):
                    tag.set("k", "sun_opening",)
                    tag.set("v", "yes")
                else:
                    tag.set("k", "sun_opening",)
                    tag.set("v", "no")
    return element

'''
The MongoDB data model includes "contact" field, which is composed of "phone",
"website" and "email", most of these keywords are contained in the "k" tag
behind the "contact" keyword with a colon, so the function here is to get the
three keywords without the form of "contact:", and put them in the "contact"
list.
'''
def fetch_contact(element):
    if (element.tag == "node" or element.tag == "way") and element.find("tag")
!= None:
        tags = element.findall("tag")
        v = []
        for tag in tags:
            k = tag.get("k")
            v.append(k)
        if "phone" in v or "website" in v or "email" in v or \
        "contact:phone" in v or "contact:website" in v or "contact:email" in v:
            contact = []
            for child in element:
                if child.tag == "tag":
                    if "contact" in child.get("k"):
                        k = child.get("k").split(':')[1]
                        if k == "phone":
                            contact.append(k)
                        elif k == "website":
                            contact.append(k)
                        elif k == "email":
                            contact.append(k)
                    elif "contact" not in child.get("k"):
                        k = child.get("k")
                        if k == "phone":
                            contact.append(k)
                        elif k == "website":
                            contact.append(k)
                        elif k == "email":
                            contact.append(k)
            return contact

def fetch_contact_val(element):
    if (element.tag == "node" or element.tag == "way") and element.find("tag")
!= None:
        tags = element.findall("tag")
        v = []
        for tag in tags:
            k = tag.get("k")
            v.append(k)
        if "phone" in v or "website" in v or "email" in v or \
        "contact:phone" in v or "contact:website" in v or "contact:email" in v:
            contact_val = []
            d = {}
            for child in element:
                if child.tag == "tag":
                    if "contact" in child.get("k"):
                        k = child.get("k").split(':')[1]
                        d[k] = child.get("v")
                        if k == "phone":
                            contact_val.append(d[k])
                        elif k == "website":
                            contact_val.append(d[k])
                        elif k == "email":
                            contact_val.append(d[k])
                    elif "contact" not in child.get("k"):
                        k = child.get("k")
                        d[k] = child.get("v")
                        if k == "phone":
                            contact_val.append(d[k])
                        elif k == "website":
                            contact_val.append(d[k])
                        elif k == "email":
                            contact_val.append(d[k])
            return contact_val

'''
The "service" field is composed of "wheelchair", "sun_opening" and "smoking",
which are obtained from osm file tag, "wheelchair" and "sun_opening" contain
```

```
boolean values, however, "smoking" contains several values, which will be
aggregated in MongoDB later to find out smoking availability of "amenity"(in
this case is "restaurant").
'''
def fetch_service(element):
    element = sun_opening(element)
    if (element.tag == "node" or element.tag == "way") and element.find("tag")
!= None:
        tags = element.findall("tag")
        v = []
        for tag in tags:
            k = tag.get("k")
            v.append(k)
        if "wheelchair" in v or "sun_opening" in v or "smoking" in v:
            service = []
            for child in element:
                if child.tag == "tag":
                    k = child.get("k")
                    if k == "wheelchair":
                        service.append(k)
                    elif k == "sun_opening":
                        service.append(k)
                    elif k == "smoking":
                        service.append(k)
            return service

def fetch_service_val(element):
    element = sun_opening(element)
    if (element.tag == "node" or element.tag == "way") and element.find("tag")
!= None:
        tags = element.findall("tag")
        v = []
        for tag in tags:
            k = tag.get("k")
            v.append(k)
        if "wheelchair" in v or "sun_opening" in v or "smoking" in v:
            service_val = []
            d = {}
            for child in element:
                if child.tag == "tag":
                    k = child.get("k")
                    d[k] = child.get("v")
                    if k == "wheelchair":
                        service_val.append(d[k])
                    elif k == "sun_opening":
                        service_val.append(d[k])
                    elif k == "smoking":
                        service_val.append(d[k])
            return service_val

'''
The "address" field is composed of "housenumber", "postcode" and "street",
however, in the osm file, the expressions of contact are not uniform, and there
are some other different names of one street, which are supplied by another
```

```
colon behind, which I don't need at all, so the below functions just get the
above three contacts from the tags.
'''
def fetch_address(element):
    if (element.tag == "node" or element.tag == "way") and element.find("tag")
!= None:
        tags = element.findall("tag")
        v = []
        for tag in tags:
            k = tag.get("k")
            v.append(k)
        if "housenumber" in v or "postcode" in v or "street" in v or \
        "addr:housenumber" in v or "addr:postcode" in v or "addr:street" in v:
            address = []
            for child in element:
                if child.tag == "tag":
                    if "addr" in child.get("k"):
                        k = child.get("k").split(":")[1]
                        if k == "housenumber":
                            address.append(k)
                        elif k == "postcode":
                            address.append(k)
                        elif k == "street":
                            address.append(k)
                    elif "addr" not in child.get("k"):
                        k = child.get("k")
                        if k == "housenumber":
                            address.append(k)
                        elif k == "postcode":
                            address.append(k)
                        elif k == "street":
                            address.append(k)
            return address

def fetch_address_val(element):
    if (element.tag == "node" or element.tag == "way") and element.find("tag")
!= None:
        tags = element.findall("tag")
        v = []
        for tag in tags:
            k = tag.get("k")
            v.append(k)
        if "housenumber" in v or "postcode" in v or "street" in v or \
        "addr:housenumber" in v or "addr:postcode" in v or "addr:street" in v:
            address_val = []
            d ={}
            for child in element:
                if child.tag == "tag":
                    if "addr" in child.get("k"):
                        k = child.get("k").split(":")[1]
                        d[k] = child.get("v")
                        if k == "housenumber":
                            address_val.append(d[k])
                        elif k == "postcode":
                            address_val.append(d[k])
                        elif k == "street":
                            address_val.append(d[k])
                    elif "addr" not in child.get("k"):
                        k = child.get("k")
                        d[k] = child.get("v")
                        if k == "housenumber":
                            address_val.append(d[[k]])
                        elif k == "postcode":
                            address_val.append(d[k])
                        elif k == "street":
                            address_val.append(d[k])
            return address_val

'''
The below functions are about "lit" and "maxspeed" tag keywords from the parent
"way" tag, just to show the limited speed of this way.
'''
def fetch_lit_speed(element):
    if element.tag == "way" and element.find("tag") != None:
        tags = element.findall("tag")
        v = []
        for tag in tags:
            k = tag.get("k")
            v.append(k)
        if "lit" in v or "maxspeed" in v:
            lit_speed = []
            for child in element:
                if child.tag == "tag":
                    k = child.get("k")
                    if k == "lit":
                        lit_speed.append(k)
                    elif k == "maxspeed":
                        lit_speed.append(k)
            return lit_speed

def fetch_lit_speed_val(element):
    if element.tag == "way" and element.find("tag") != None:
        tags = element.findall("tag")
        v = []
        for tag in tags:
            k = tag.get("k")
            v.append(k)
        if "lit" in v or "maxspeed" in v:
            lit_speed_val = []
            d = {}
            for child in element:
                if child.tag == "tag":
                    k = child.get("k")
                    d[k] = child.get("v")
                    if k == "lit":
                        lit_speed_val.append(d[k])
                    elif k == "maxspeed":
                        lit_speed_val.append(d[k])
            return lit_speed_val



# "created" field in data model
created = ["version", "changeset", "timestamp", "user", "uid"]

def jsn_doc(element):
    if element.tag == "node" or element.tag == "way":
        doc = {}
        #another two fields "id" and "visible"
        doc["id"] = element.get("id")
        doc["visible"] = element.get("visible")
        #add "created" field values into the json document
        created_val = [element.get("version"), element.get("changeset"),
element.get("timestamp"), element.get("user"), element.get("uid")]
        created_dic = dict(zip(created, created_val))
        doc["created"] = created_dic
        #add "address" field values into the json document
        address = fetch_address(element)
        address_val = fetch_address_val(element)
        if address != None and address_val != None:
            address_dic = dict(zip(address, address_val))
            doc["address"] = address_dic
        #add "contact" field values into the json document
        contact = fetch_contact(element)
        contact_val = fetch_contact_val(element)
        if contact != None and contact_val != None:
            contact_dic = dict(zip(contact, contact_val))
            doc["contact"] = contact_dic
        #add "service" field values into the json document
        service = fetch_service(element)
        service_val = fetch_service_val(element)
        if service != None and service_val != None:
            service_dic = dict(zip(service, service_val))
            doc["service"] = service_dic
        #add "name", "amenity", "cuisine", "shop" and "sport" fields and
respective values into the json document
        #if they don't exist in the osm file tag, they will be "Null" in json
data, which could be easily filtered by Mongo query
        d = {}
        if element.find("tag") != None:
            for child in element:
                if child.tag == "tag":
                    k = child.get("k")
                    d[k] = child.get("v")
                    if k == "name":
                        doc["name"] = d[k]
                    if k == "amenity":
                        doc["amenity"] = d[k]
                    if k == "cuisine":
                        doc["cuisine"] = d[k]
                    if k == "shop":
                        doc["shop"] = d[k]
                    if k == "sport":
                        doc["sport"] = d[k]
        #add "type" and "pos" fields and respective values into json document if
the tag is "node"
        if element.tag == "node":
            doc["type"] = "node"
            doc["pos"] = [element.get("lat"), element.get("lon")]
        #add "type" field value into the json document if the tag is "way"
        elif element.tag == "way":
            doc["type"] = "way"
            #add "lit_speed" field values into json document
            lit_speed = fetch_lit_speed(element)
            lit_speed_val = fetch_lit_speed_val(element)
            if lit_speed != None and lit_speed_val != None:
                lit_speed_dic = dict(zip(lit_speed, lit_speed_val))
                doc["lit_speed"] = lit_speed_dic
            #add "ref" field values into json document
            ref = []
            for child in element:
                if child.tag == "nd":
                    ref.append(child.get("ref"))
            doc["ref"] = ref

        return doc

'''
Write the file as a json file, getting rid of those documents without any
values, and then import to MongoDB as a collection named "wien"
'''

fname = open("D:\Udacity\wien.json","w")
for _, element in ET.iterparse(filename):
    doc = jsn_doc(element)
    if doc != None:
        fname.write(json.dumps(doc))


#load file from MongoDB
from pymongo import MongoClient
client = MongoClient()

db = client.UdaPro
collection = db.wien

#---Data Overview
'''file size
wien.osm -------- 757MB
wien.json ------- 699MB
'''

#number of documents
print db.wien.count()
##3309431
```

```
#number of nodes
print db.wien.find({"type": "node"}).count()
##3261548

#number of ways
print db.wien.find({"type": "way"}).count()
##47883

#number of unique users
uni_users = db.wien.aggregate([
    {"$group": {"_id": "$created.user"}},
    {"$group": {"_id": "uni_users", "count": {"$sum": 1}}}
])
for doc in uni_users:
    print doc
##{u'count': 2735, u'_id': u'uni_users'}
#simple way to calculate the number:
print len(db.wien.distinct("created.user"))
##2735

#top 3 contributor user
users = db.wien.aggregate([
    {"$group": {"_id": "$created.user", "count": {"$sum": 1}}},
    {"$sort": {"count": -1}},
    {"$limit": 3}
])
for doc in users:
    print doc
##{u'count': 545146, u'_id': u'Linie29'}
##{u'count': 222737, u'_id': u'ecdos'}
##{u'count': 220243, u'_id': u'Dima M'}

#number of users who posted only once
users = db.wien.aggregate([
    {"$group": {"_id": "$created.user", "count": {"$sum": 1}}},
    {"$group": {"_id": "$count", "count": {"$sum": 1}}},
    {"$sort": {"_id": 1}},
    {"$limit": 1}
])
for doc in users:
    print doc
##{u'count': 686, u'_id': 1}


#---Additional Data Exploring
# top 10 cuisine
top_cui = db.wien.aggregate([
    {"$match": {"cuisine": {"$exists": True}}},
    {"$group": {"_id": "$cuisine", "count": {"$sum": 1}}},
    {"$sort": {"count": -1}},
    {"$limit": 10}
])
for doc in top_cui:
    print doc
```

```
##{u'count': 434, u'_id': u'regional'}
##{u'count': 211, u'_id': u'pizza'}
##{u'count': 181, u'_id': u'italian'}
##{u'count': 133, u'_id': u'asian'}
##{u'count': 126, u'_id': u'chinese'}
##{u'count': 86, u'_id': u'heuriger'}
##{u'count': 63, u'_id': u'burger'}
##{u'count': 58, u'_id': u'ice_cream'}
##{u'count': 57, u'_id': u'cake'}
##{u'count': 55, u'_id': u'kebab'}


#street with the most cafes
cafe = db.wien.aggregate([
    {"$match": {"amenity": "cafe", "address.street": {"$exists": True}}},
    {"$group": {"_id": "$address.street", "count": {"$sum": 1}}},
    {"$sort": {"count": -1}},
    {"$limit": 1}
])
for doc in cafe:
    print "street: %s, count: %d" % (doc["_id"].decode("utf-8"), doc["count"])
##street: Hauptstraße, count: 11


#number of restaurants or cafes open on Sundays and with wheelchair available
service_avai = db.wien.aggregate([
    {"$match": {"$or": [{"amenity": "restaurant"}, {"amenity": "cafe"}], "$and":
[{"service.sun_opening": "yes"}, {"service.wheelchair": "yes"}]}},
    {"$group": {"_id": "service_avai", "count": {"$sum": 1}}}
])
for doc in service_avai:
    print doc
##{u'count': 35, u'_id': u'service_avai'}


#district with the most pubs
pub = db.wien.aggregate([
    {"$match": {"amenity": "pub", "address.postcode": {"$exists": True}}},
    {"$group": {"_id": "$address.postcode", "count": {"$sum": 1}}},
    {"$sort": {"count": -1}},
    {"$limit": 1}
])
for doc in pub:
    print doc
##{u'count': 23, u'_id': u'1010'}


#top 3 popular sports
pop_spt = db.wien.aggregate([
    {"$match": {"sport": {"$exists": True}}},
    {"$group": {"_id": "$sport", "count": {"$sum": 1}}},
    {"$sort": {"count": -1}},
    {"$limit": 3}
])
```

```
for doc in pop_spt:
    print doc
##{u'count': 247, u'_id': u'soccer'}
##{u'count': 239, u'_id': u'tennis'}
##{u'count': 72, u'_id': u'swimming'}


#top 3 sport-type-clubs that have both website and email contacts
wemail = db.wien.aggregate([
    {"$match": {"$or": [{"contact.email": {"$exists": True}},
{"contact.website":{"$exists": True}}], "sport": {"$exists": True}}},
    {"$group": {"_id": "$sport", "count": {"$sum": 1}}},
    {"$sort": {"count": -1}},
    {"$limit": 3}
])
for doc in wemail:
    print doc
##{u'count': 52, u'_id': u'table_soccer'}
##{u'count': 10, u'_id': u'golf'}
##{u'count': 7, u'_id': u'equestrian'}


#---Additional Ideas
#Obviously, there are quite a lot of restaurant "amenity"s in the dataset, here
is the number of restaurant amenity in the dataset:
#print db.wien.find({"amenity": "restaurant"}).count()
#2416
#I wonder how many of them are supplied with wheelchair and different smoking
setups, here is the number of restaurant with wheelchair and
#smoking setups
ws = db.wien.aggregate([
    {"$match": {"$and": [{"service.wheelchair": {"$exists": True}},
{"service.smoking": {"$exists": True}}]}},
    {"$project": {
            "wheelchair": {"$eq": ["$service.wheelchair", "yes"]},
            "smoking": {"$or": [{"$eq": ["$service.smoking", "separated"]},
{"$eq": ["$service.smoking", "isolated"]}, {"$eq": ["$service.smoking",
"no"]}]},
            "amenity": {"$eq": ["$amenity", "restaurant"]}
        }},
    {"$group": {"_id": {"smoking": "$smoking", "wheelchair": "$wheelchair",
"amenity": "$amenity"}, "count": {"$sum": 1}}}
])
for doc in ws:
    if doc["_id"]["amenity"] == True:
        print doc
##{u'count': 9, u'_id': {u'smoking': False, u'wheelchair': False, u'amenity':
True}}
##{u'count': 58, u'_id': {u'smoking': True, u'wheelchair': False, u'amenity':
True}}
##{u'count': 20, u'_id': {u'smoking': True, u'wheelchair': True, u'amenity':
True}}
##{u'count': 4, u'_id': {u'smoking': False, u'wheelchair': True, u'amenity':
True}}
```

```
'''
There are 9 restaurants out of 2416, say 0.3% don't have either wheelchair or
smoking setup, which sounds nice;
there are 58 restaurants out of 2416, say 2.4% do have smoking setups but no
wheelchair supply;
there are 20 restaurants out of 2416, say 0.8% have both setups, which is not
very a high number;
finally, there are 4 restaurants out of 2416 don't have smoking setup but do
have wheelchair supply, which is incredibly low.
The numbers here are challenging, there could be many reasons why they are so
low compared to the total number of restaurants. Probably, most of the
restaurants don't realize that it is necessary to make smoking setups for people
who don't smoke or who don't like that smoky smell; more than smoking-resistant
awareness, restaurants may pay less attention to the importance of wheelchair
accommodation.
But as likely as the situation above, it is possible that the dataset is not
complete enough, contributors made their huge efforts to contribute the dataset,
it's nothing but normal that some details could be missed, if this is the case,
all we need is just a little more carefulness to make the dataset more perfect.
'''

#---Conclusion
'''
The original dataset is quite large, and I didn't expect it so large, I cut a
little bit the end of the file, that's why the json file is smaller than the osm
file, but I think it would not affect the data query.
This dataset contains lots of information, and I just make part use of it, I
believe there are much more I didn't cover, but for now, the result I've got is
informative at some point.
'''
```