# 互联网出行

> 计64 翁家翌 2016011446

## 环境配置与运行

0. 首先假设是ubuntu/debian机子，已经安装常用的软件如cmake、python3等，能正常上网；数据文件存放在 `data/` 下面，有 `car.txt`、`road.cnode`、`road.nedge` 三个文件
1. 安装metis：`sudo apt install libmetis-dev`
2. 安装web.py：`sudo pip3 install "web.py==0.40.dev1"`
3. 编译：`mkdir build && cd build && cmake .. && make -j`
4. `cd ../src; python3 main.py`，此时访问 `http://localhost:8080` 即可看到效果。如果想换别的端口，可以使用命令 `python3 main.py [port]` 即可切换成 `http://localhost:port`。

第一次运行 `main.py` 时会生成 `GP_Tree.data` 在当前文件夹下，需要比较久的时间（一分钟左右），文件大小为319M，在此之后再次运行就很快了。

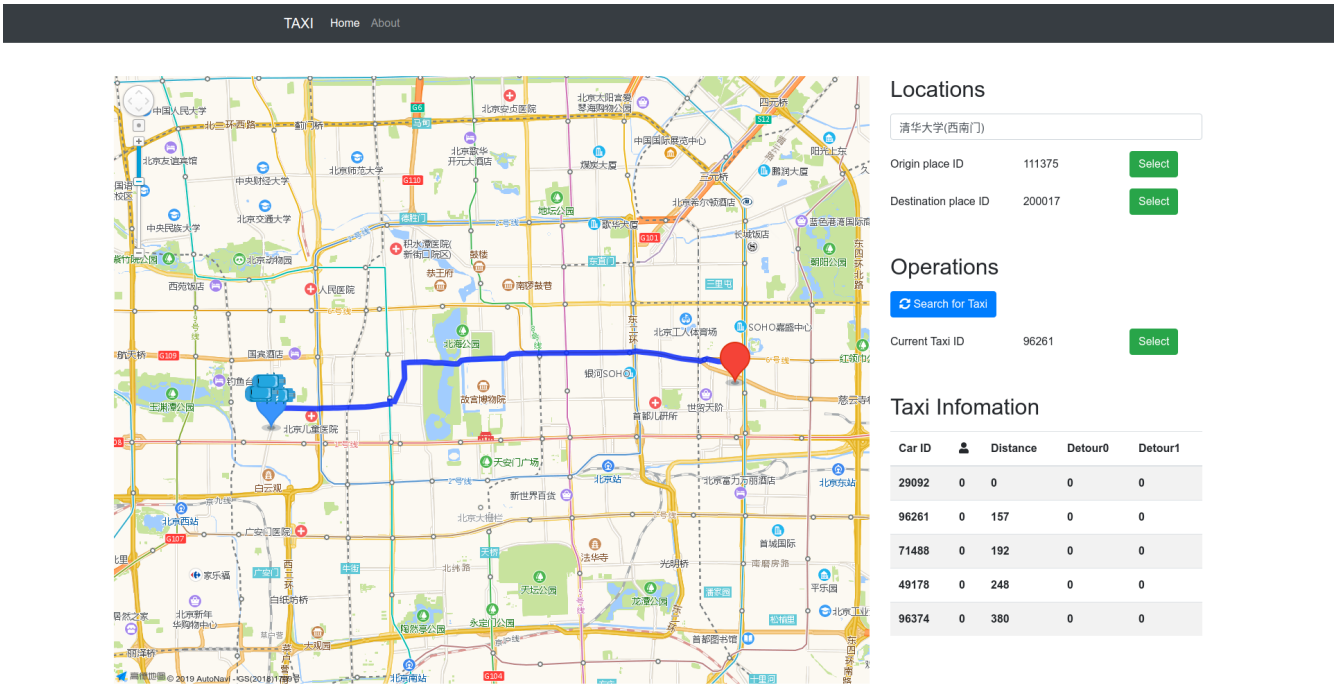
## 测试效果

编译会生成 `build/test` 和 `build/libtaxi.so` 两个文件，其中test是测试所用，运行结果大概长下面这样：

```
➜  build git:(master) ✗ ./test
begin read
338024 440525
read over
root's part:58
begin_build_border_in_father_son
begin_build_dist
begin_build_dist2
begin save
save_over
build gptree: 41.167996s
-1 -1
search 10000 times: avg 0.000068s
```

说明正确性和性能均无问题。

前端的页面效果如下：




可支持地图搜索目标、地图选点、实时找车和生成路径等功能。运行时间均不超过0.1s。

一个有意思的现象：我怎么选起点终点都几乎是选到空车。我尝试着把空车排除掉，只能选1-3人车，结果发现大部分情况只能选1人，2-3人几乎不能满足拼车条件。

# 实现方法

## 后端

算法使用 GPTree，大致查找流程如下：

1. 初始化的时候先把所有满员的车辆删去，会删掉约1w辆
2. 每次查询时，使用KNN方法查询距离当前出发点最近的100辆车
3. 依次判断每一辆车是否满足条件：
    1. 如果是空车，距离不超过10KM，则认为符合条件；如果超过则直接排除
    2. 如果车上之前有n个人，则分别求出d1, d2, d3, d4的值，再计算是否符合条件；d1和d3的值需要枚举全排列进行计算，考虑到问题规模不是很大，采用打表方式替代深搜枚举全排列
4. 将满足条件的车辆按照一定比例的权重进行排序
5. 将结果（出租车及其对应路径）返回给前端，如果没有则返回空

## 前端

使用 bootstrap4 + jquery + 高德地图API。由于高德地图接受的坐标是GCJ，无需转换，可以直接使用。

## 拼接

使用了python3把前后端接起来，主要使用了ctypes和web.py两个库，其中ctypes负责导入 `libtaxi.so`，计算数据；`web.py` 负责前端框架。

# 坑

在使用GPTree的时候，经常会莫名其妙的crash，发现是里面的时钟中断调用过多，把clock相关语句注释掉即可。