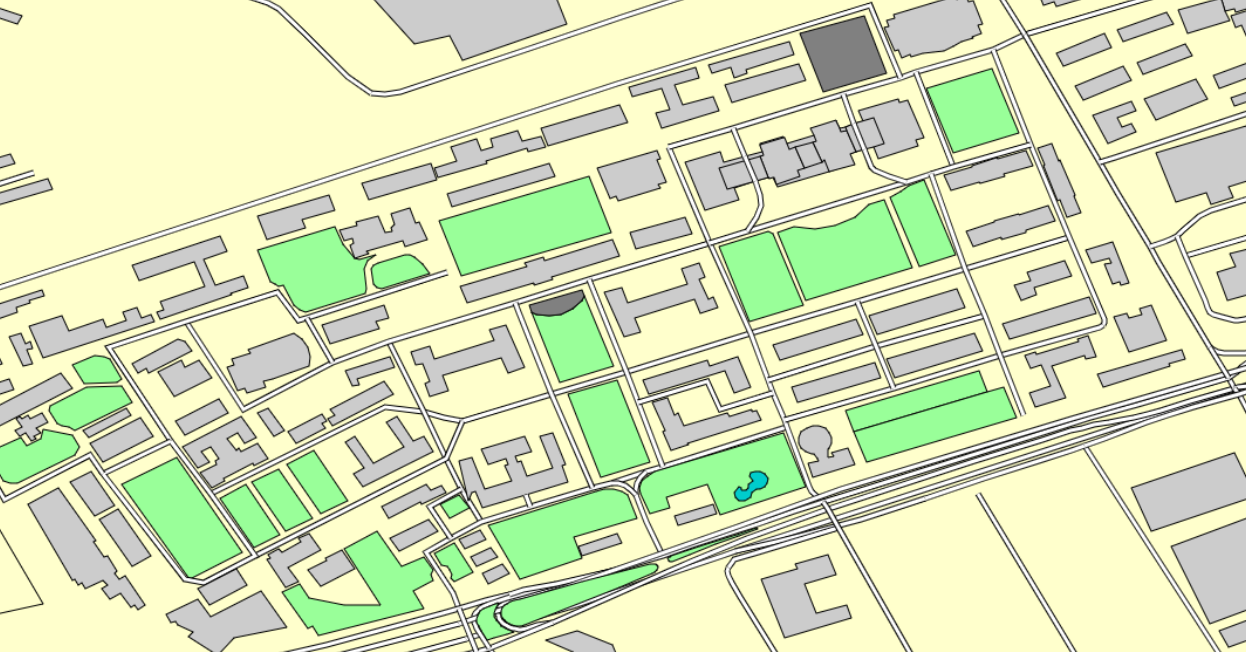
数据结构PJ报告

一.基本功能

1.地图绘图：

地图绘图方面使用html的canvas进行绘图，获取osm的数据，将同在一条way上的node全部连起来，传入到前端，前端沿着这些线描边，根据way的tag进行判断是否要填充颜色，填充什么颜色，以及是否是道路，有道路的话是另一种画法，白色填充加黑色描边。选取的主要是复旦周围区域。效果展示如下：

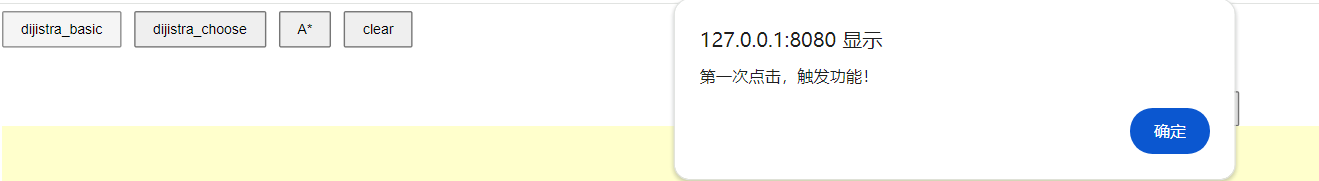




2.dijistra最短路寻址：

这个功能要实现先要预处理，将way上的点都连起来，为了减少计算量，提前将相邻两点间距离算好存起来。因为我不太会处理json的数据，所以用c++重新定义了一系列变量来存储way和nodes等元素，然后就是照着dijistra的思路写好程序即可。而且起点不是很好找，我基础功能的dijistra是找到离起点终点最近的点作为dijistra算法的起点，然后在前端把鼠标点击的点和这两个点分别相连，距离路网很近的点结果才较为准确。示意如下：

点击左上角dijistra\_basic按钮触发功能



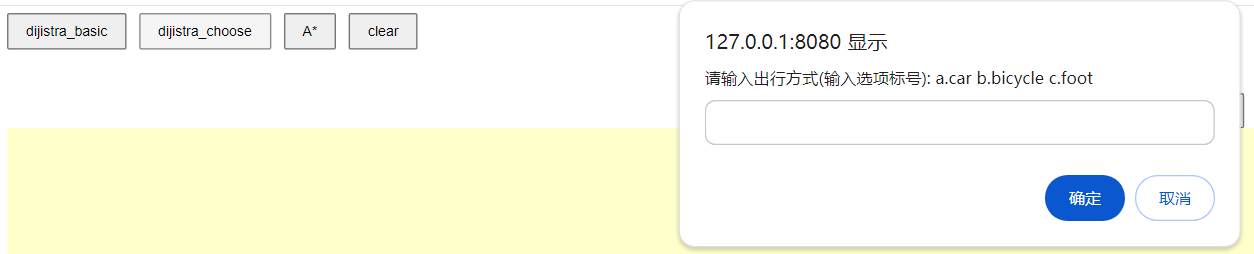
鼠标在路网附近点两下即可画出最短路，点第三下重置：



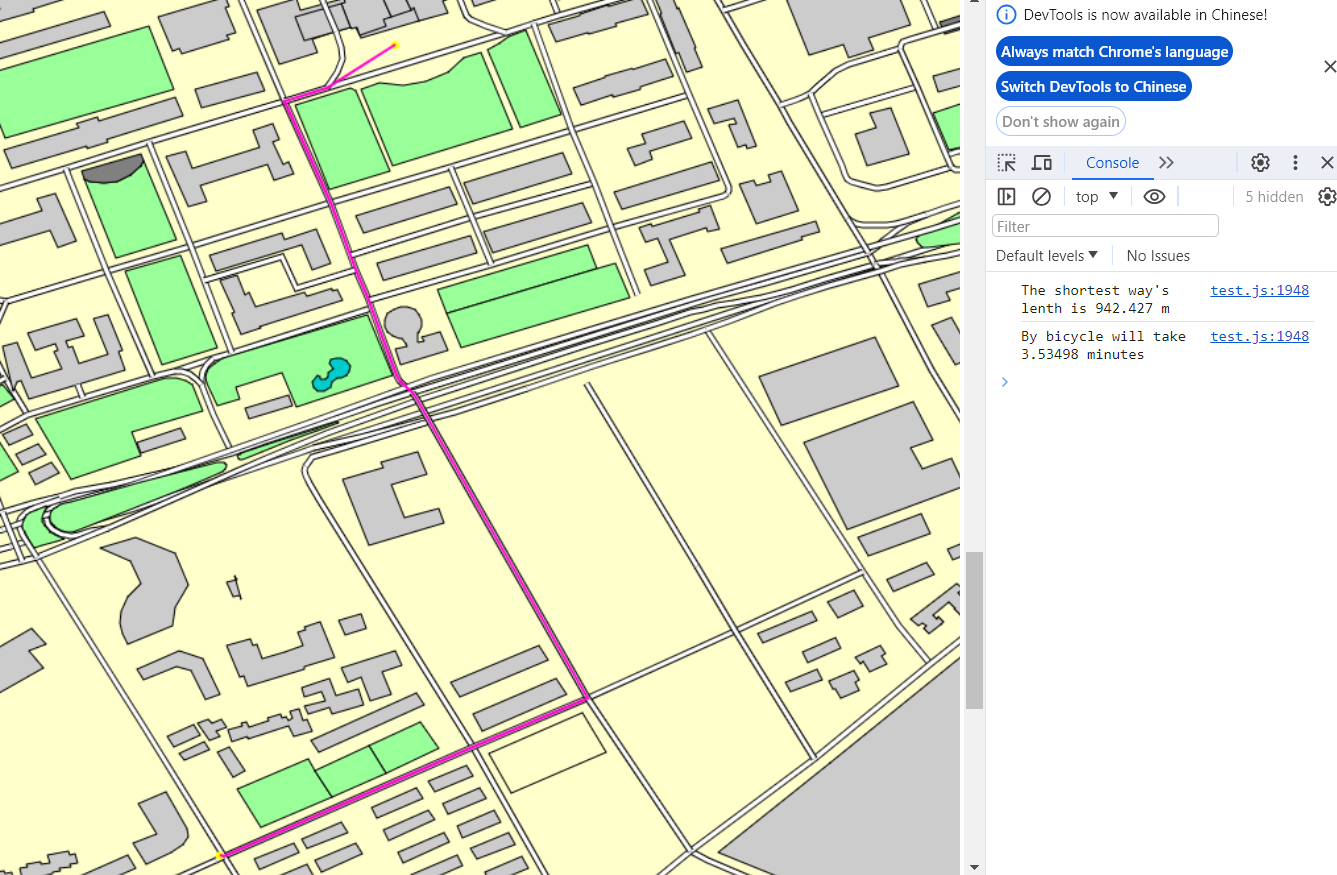
二．扩展功能

1. 带有出行方式选择的dijistra最短路寻址：

因为基础功能的最短路只找最近的点，其实不太准确，所以这个dijistra\_choose选择将距离起点和终点最近的七个节点加入边，直接在预处理时维护一个大小为7的大根堆，元素大于堆顶即更新堆。这样处理好后起点和终点分别有七个点和它们相连。然后再开始最短路寻址。点击dijistra\_choose按钮触发功能，会有abc三个选项选择出行方式。



取消的话默认是car出行。然后与基础功能一样，点击两点会显示两点间最短路，控制台会输出距离以及预估时间。



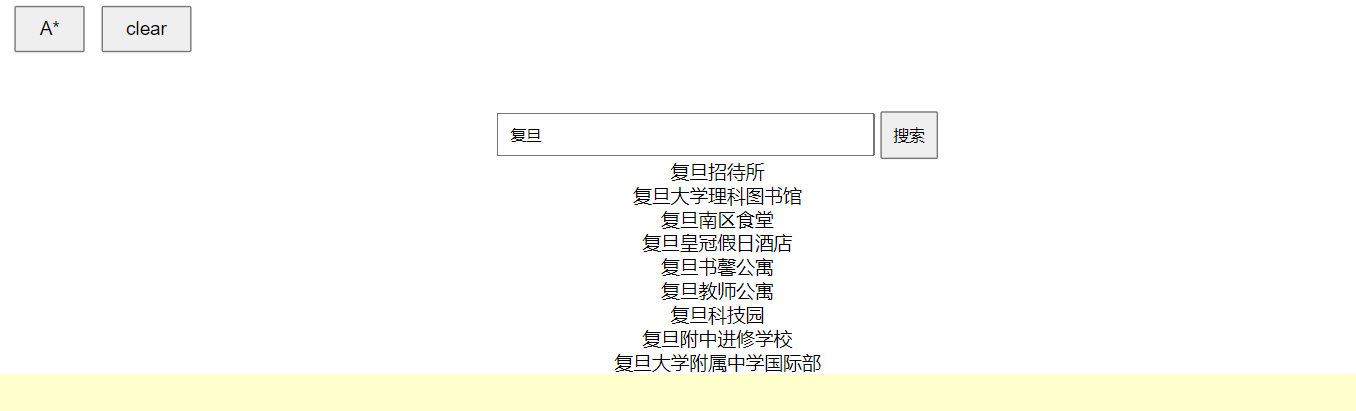
比如我从南区门口骑车到光华楼如上所示。

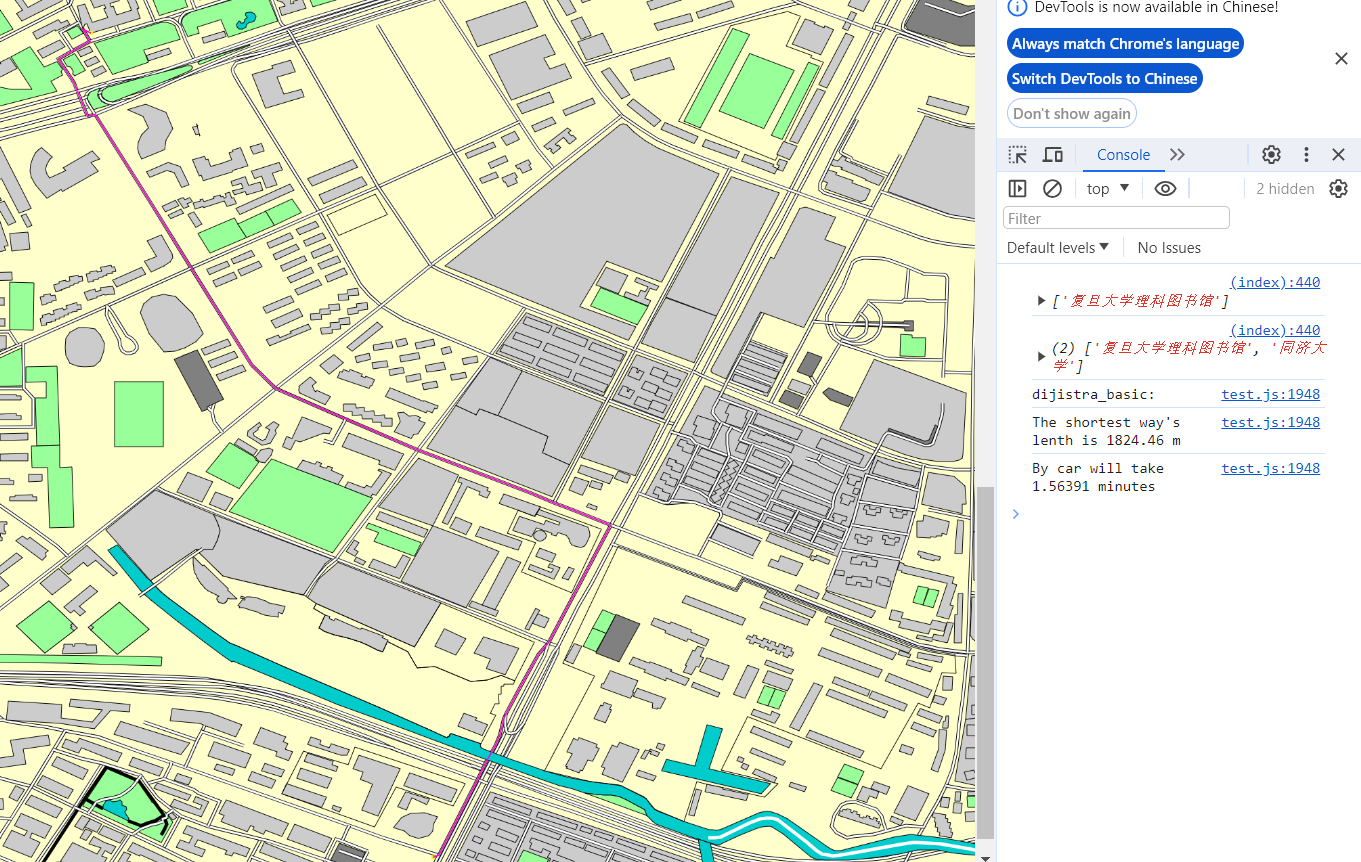
2.A\*寻址算法，因为A\*需要有一个权值函数，每次贪心地选择还未选择的点钟权值最小的，我在地图中将该函数设置成该点到起点的距离与该点到终点的距离之和（两点间距离）。每次贪心选择权值最小的未在路径中的点。最后到达终点即停止。起始点也与和它最近的七个点相连。同样的，A\*算法我也设置有可选出行方式的功能。示意如下：



需要注意的是，切换功能时要先第二次点击已经开启的功能按钮以关闭原来的功能，再开启新功能，否则会报错。

3.接着是写的一个搜索，借助html自带的搜索框功能，写了一个带提示语的搜索框模块，输入关键字后会得到相应的提示，如同常用的搜索功能，点击提示的文字即可填入文本框。这里主要是要在后端将way的name进行预处理并全部传给前端，搜索框才能以这些names为依据来进行搜索提示。点击搜索后，便会传参给后端，后端有一个红黑树（map）用来搜索确认起点，再次重新输入第二个地点作为终点，点击搜索即可显示出最短路，这里的最短路是用dijistra实现的。如果输入非法，控制台会报错。

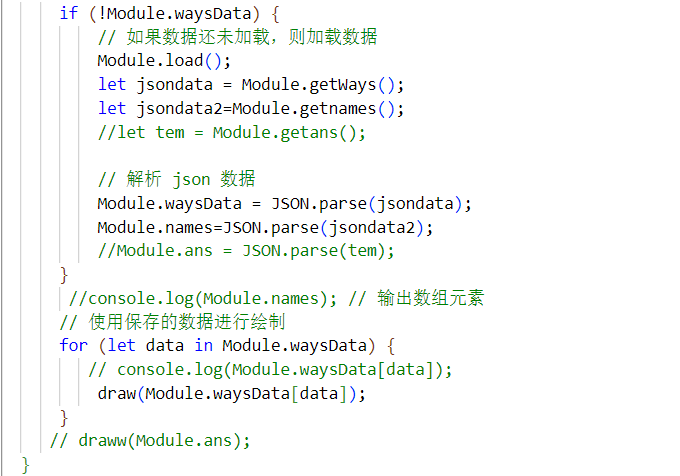




如上图，选择后控制台会输出两个地点并且地图上有最短路显示。值得提醒的是，下一次寻找最短路前，必须点击上方的clear按钮清除路径，否则原有路径会一直存在。

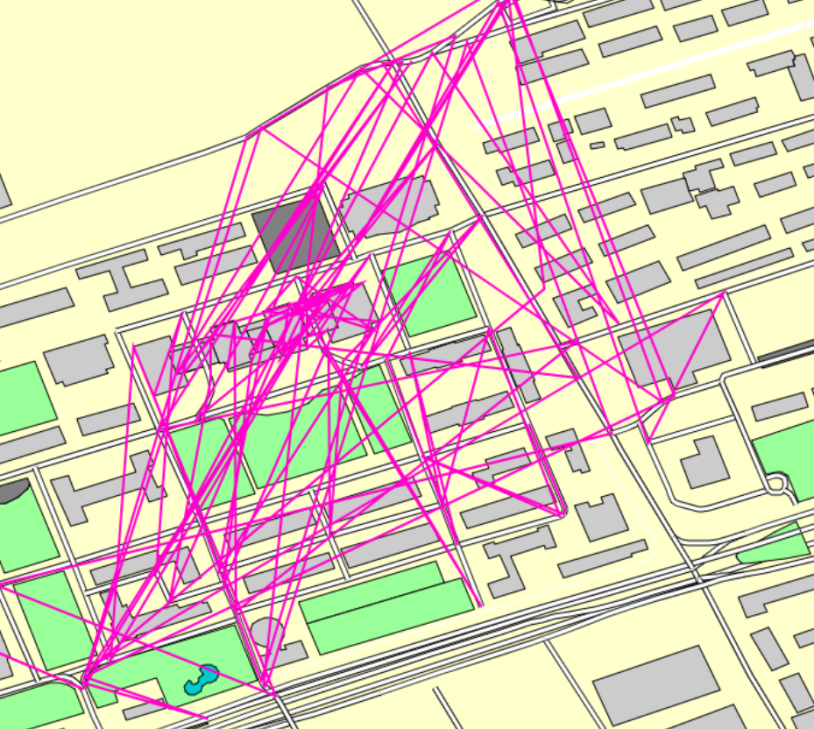
另外，点击搜索后，前端会调用一个函数scrollToCenter来移动视窗，使得搜索到的地点处于视窗的中心，此处是平滑地移动，方便查看自己搜索的地点在何处，如果输入没有查找到，则不会移动视窗。具体实现是获取视窗的长和高，再结合找到的点的坐标进行移动即可。

4.然后就是一些小细节了，每次消除最短路都意味着要重画一遍地图，但其实数据已经读好了，没必要再次load，所以重画时进行判断，waydata和names的数据非空，就不必再次load，直接调用函数draw即可。这样地图擦除路径快了很多。



三．实验过程

1.之前没有接触过html编程，也没有接触过javascript，所以刚上手时是非常困难的，绘图有助教大大给的示例代码，上网学一学上手很快，根据各种way描线以及根据tag进行填色就好。一开始写最短路没有用c++的数据结构预处理，写得一塌糊涂，这是我当时的截图（当事人脸都气紫了）



2.后面觉得还是得用自己熟悉的c++的数据结构，于是进行了预处理，再写最短路快了许多，虽然debug仍然非常痛苦。

3.然后是功能选择，根据用户的输入给相应的函数设置条件分支即可，控制台会给出相应的反馈。A\*算法的话，因为有之前的预处理在，只要自己定义好权值函数就写得很快。

4.接着是搜索功能，借助html的搜索框模块，可以很快地写出来，但是将names传入到前端数组的时候犯了低级错误，把ways和way两种数据类型搞反了，debug了半天才发现。另外我还增添了移动视角的功能，使用户能更方便地查看自己搜索的位置在哪。

四．运行方式

进入文件夹后打开命令窗口，运行以下命令：

emcc tinyxml/tinyxml.cpp tinyxml/tinystr.cpp tinyxml/tinyxmlerror.cpp tinyxml/tinyxmlparser.cpp jsoncpp/json\_reader.cpp jsoncpp/json\_value.cpp jsoncpp/json\_writer.cpp test.cpp -o frontend/test.js -lembind --preload-file map1 -sINITIAL\_MEMORY=536870912 -s NO\_DISABLE\_EXCEPTION\_CATCHING=1  
cd frontend  
python -m http.server

接着打开浏览器如Google，输入127.0.0.1:8080即可。

五．实验总结和心得

其实一开始听到pj要做地图我是相当迷茫的，以前从来没有接触过前后端交互，在网上找过很长一段时间的代码，但是没找到我可以用的，后来展示课上看到大家都用的qt，我又在中途有所动摇，有过想要转qt的想法，但在问过助教大大后放弃了，由于长期的迷茫，我起步很晚，临近ddl两周才开始写最短路，在各种期末考试的压榨下挤出时间来写pj，无数次debug到破防，无数次想放弃，但我坚持下来了，此时回望，才发现自己做到了很多之前从未想过的事，感谢老师和助教的辛苦付出，无论是pj还是课程，这学期我学到了很多很多知识，也越来越有毅力和抗压力，真的获益匪浅，再次感谢老师和助教！