

# Project 4

## 校园导游

计科一班 17341009 曾天宇

计科一班 17341015 陈鸿峥

计科二班 17341059 黄杨峻

## 1 题目要求

1. 从中山大学东校区的平面图中选取有代表性景点(20-30个), 抽象成一个无向带权图。以图中顶点表示校内各景点, 存放景点名称、代号、简介等信息; 以边表示路径, 存放路径长度等信息。
2. 为来访客人提供图中任意景点相关信息的查询。
3. 为来访客人提供图中任意景点的问路查询, 即查询任意两个景点之间一条最短的简单路径。
4. 区分汽车线路与步行线路。



## 2 数据结构与算法

由于这次的题目较为简单, 若单纯做一个算法实现那没什么意思, 故我们想挑战一下自己: 将其做成项目, 并实现全平台导览, 包括PC端(网页版)、微信小程序(Android)及iOS设备。详情见下面的描述。

## 2.1 PC端(网页版)实现

### 2.1.1 使用语言

html、javascript、css

### 2.1.2 设计思想

设计一个PC端(网页)实现校园导览。

调用百度地图的API求得两点之间的路径长度，存入矩阵，用Dijkstra算法求得两点的最短路径，在地图上将路径、路径描述、相关地点简介、途径点进行显示。网页作品已上传至服务器<http://193.112.60.186:81/CampusMap.html>，保留期限为本学期。

### 2.1.3 数据结构

采用邻接矩阵存储各顶点间的距离。

定义了地图的container，定义了文本框ui-box，按钮button等类的结构。

### 2.1.4 算法过程

1. 在网页上抓取用户输入的起点和终点，调用查询search()函数
2. 在预定义的数组中得到起点和终点的经纬度坐标，调用百度API中步行或驾车的search()函数展示路径

```
320 function search() {
321     map.clearOverlays();
322     description = "";
323     var methodStr = document.getElementById("walk").checked ? "1" : "2";
324     var method = methodStr == "1" ? "步行" : "驾车";
325     var srcIndex = document.getElementById("start").value;
326     var dstIndex = document.getElementById("end").value;
327     var src = new BMap.Point(locations[srcIndex][0], locations[srcIndex][1]);
328     var dst = new BMap.Point(locations[dstIndex][0], locations[dstIndex][1]);
329     if (methodStr == "1")
330         walking.search(src, dst);
331     else
332         driving.search(src, dst);
333 }
```

3. 另一方面，调用getAdjPoints函数，用Dijkstra算法（见下图）求得要连接这两点最短路径的中间点（由于已预定义了两点之间的相对距离，存在矩阵内，故可以求SSSP，而无需在运行时再重新估算距离），在地图上显示出来，并在侧边文本框输出景点简介

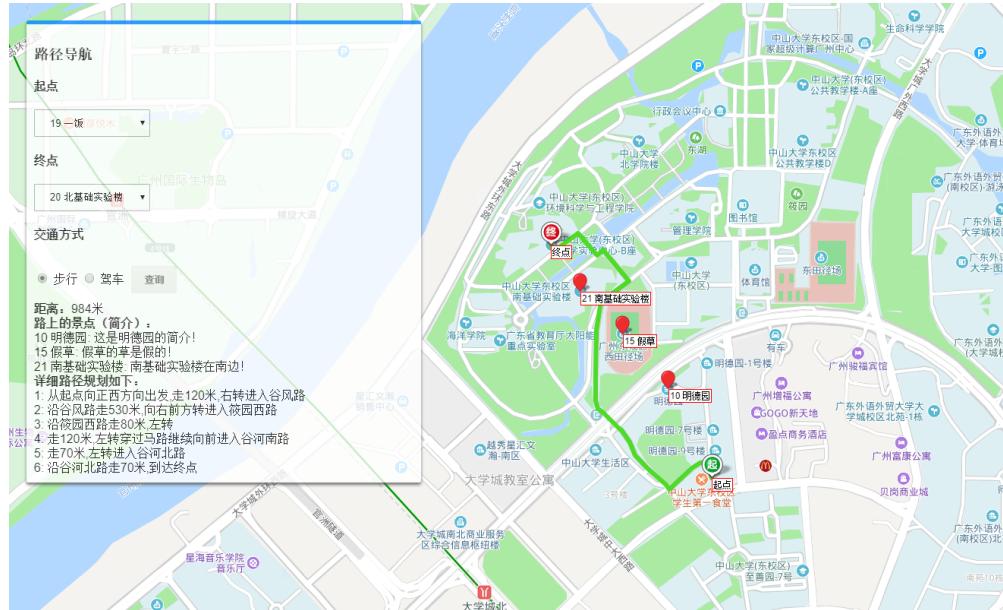
```

163  function dijkstra(){
164      var n = locations.length;
165      var d = new Array(n);
166      var v = new Array(n);
167      var INF = 0x3F3F3F3F;
168      // initialization
169      for (var i = 0; i < n; ++i){
170          d[i] = INF;
171          v[i] = false;
172      }
173
174      d[start] = 0;
175      for (var i = 0; i < n; ++i){
176          var temp;
177          var minx = INF;
178          for (var j = 0; j < n; ++j)
179          {
180              if (!v[j] && d[j] < minx)
181              {
182                  temp = j;
183                  minx = d[j];
184              }
185          }
186          v[temp] = true;
187          // refresh distance array
188          for (var k = 0; k < n; ++k)
189          {
190              if (d[temp] + edge[temp][k] < d[k]){
191                  d[k] = d[temp] + edge[temp][k];
192                  path[k] = temp; // record path
193              }
194          }
195      }
196      return dis;
197  }

```

4. 搜索结束后在地图上标注路径、起点、终点、途径点，并在侧边文本框输出全程距离和具体路径实施规划

## 2.1.5 测试截图



由于驾车路线涉及到的问题比较多，还在开发，故这里没有给出相应截图。

## 2.2 微信小程序实现

### 2.2.1 使用语言

wxml、wxss、js、json

### 2.2.2 设计思想

设计一个微信小程序实现题目要求，使得手机移动端（iOS和Android）可用。调用腾讯地图的API，实现从点到点的路径规划，并显示途径的景点。

### 2.2.3 数据结构

定义了Placelist类，存储所有景点的信息（包括经纬度、地点名称等）

```
Placelist({
    id: ,
    latitude: ,
    longitude: ,
    placeName: '---',
})
```

定义了Markers类，用于显示地图上的景点气泡，图标根据景点类型而决定（起点、终点、经过点、其他点）

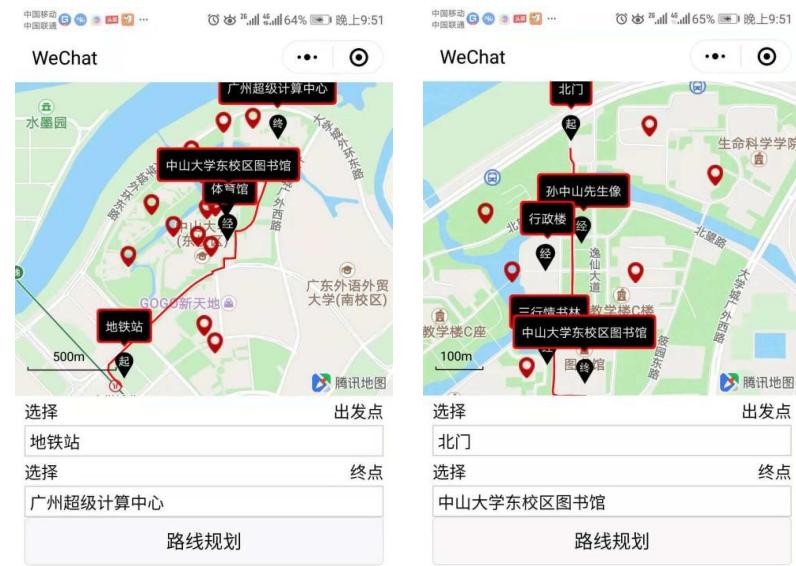
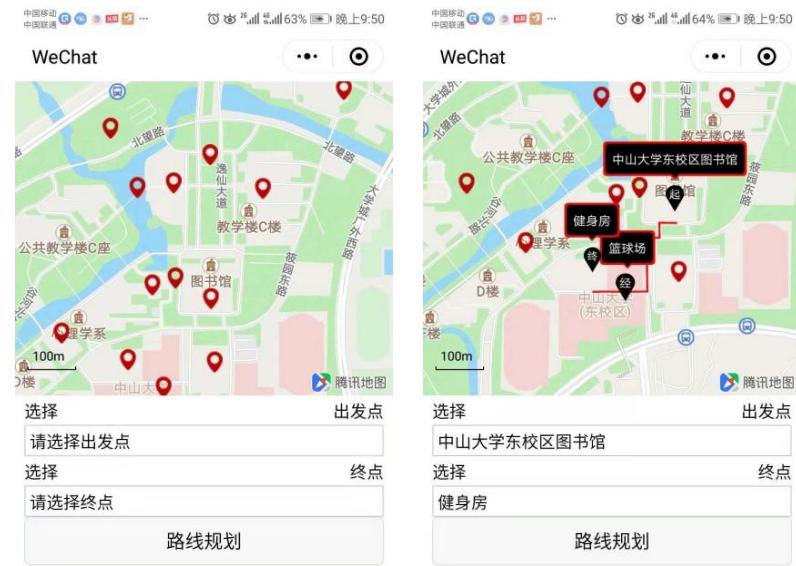
```
Markers({
    Id: ,
    Latitude: ,
    Longitude: ,
    iconPath: '---',
    Height: ,
    Width: ,
    Callout{
        Content: '---',
        Display: '---'
    }
})
```

Polyline: 用于存储请求API返回的路径Polyline[(longitude,latitude)]

### 2.2.4 算法过程

1. 寻路：首先定义景点之间的距离为一个邻接矩阵（相当于一个导游图），然后通过在线Dijkstra算法求最短路径（抽象），最后通过API逐段请求路径中每两个景点的真实路径，将其push到polyline这个list里面，拼接成完整的路径
2. 中间点：由于景点不一定在路上，所以采用标准化欧氏距离 $< 0.0000003$ 来衡量中间点。

## 2.2.5 测试截图



## 2.2.6 不足之处

1. 响应速度不够，没有设置加载中的圈圈以减少用户焦虑

2. Markers的气泡太丑，而且透明度不足
3. 无法支持实时导航（因为没权限）

## 2.3 iOS平台实现

### 2.3.1 使用语言

Swift、Objective-C

### 2.3.2 设计思想

设计一个iOS APP实现题目要求，能够支持有序浏览和无序浏览，支持线路指示输出，支持多种交通方式，支持多地图，支持多数据集。

### 2.3.3 数据结构

定义了Swift类：Scenery Point储存景点信息，数据从csv文件Places.csv中读取：

```
class SceneryPoint: NSObject, MKAnnotation {
    let tit: String
    let title: String?
    let locationName: String
    let discipline: String
    var coordinate: CLLocationCoordinate2D
```

定义了Matrix类处理邻接矩阵：

```
typealias sendMatrixClosure = (_ scList : [Int], _ length:Int) -> Void
class Matrix {
    var mat = [[Int]]()
    var size: (Int, Int)!
    var scList: [SceneryPoint]!
    var count = Int(0)
    var matrixClosure: sendMatrixClosure!
```

定义了DataListTableViewCell作为数据显示单元：

```
class DataListTableViewCell: UITableViewCell {
    @IBOutlet weak var longlabel: UILabel!
    @IBOutlet weak var positag: UILabel!
    @IBOutlet weak var title: UILabel!
    var scene: SceneryPoint!
```

定义了 MapClassify 作为数据实时请求处理单元：

```

class MapClassify{
    var sclist : [SceneryPoint]
    var locationManager : CLLocationManager
    var transType = Int(0)
    let que = DispatchQueue.init(label: "com.zengtianyu.campusMap.dirRequest")
}

```

### 2.3.4 算法过程

- 有序浏览有序浏览采用分段请求路径的方式即可实现。通过MKMapView请求得到点对点之间的最小路径，并且显示在MapView上。
- 无序浏览由于是用户先确定浏览点，故无序浏览采用TSP问题的贪心算法。请求点到点之间的有向最小路径，处理邻接矩阵获取最小回路，并将路线显示在MapView上面。

```

func travel() -> Void{
    Print()
    var j = Int()
    var i = 1
    var sum = Int(0)
    var S = [Int]()
    S.append(0)
    repeat{
        var k = 1, Dtemp = 10000;
        repeat{
            var l = 0;
            var flag = Bool(false);
            repeat{
                if S[l] == k {
                    flag = true;
                    break
                }else{
                    l += 1
                }
            }while (l < i)
            if !flag && (mat[k][S[i-1]] < Dtemp) {
                j = k;
                Dtemp = mat[k][S[i-1]]
            }
            k += 1;
        }while (k < size.0)
        S.append(j)
        i += 1
        sum += Dtemp
    }while (i<size.0)
    sum += mat[0][j]
    print(sum)
    print(S)
    self.matrixClosure(S, sum)
}

```

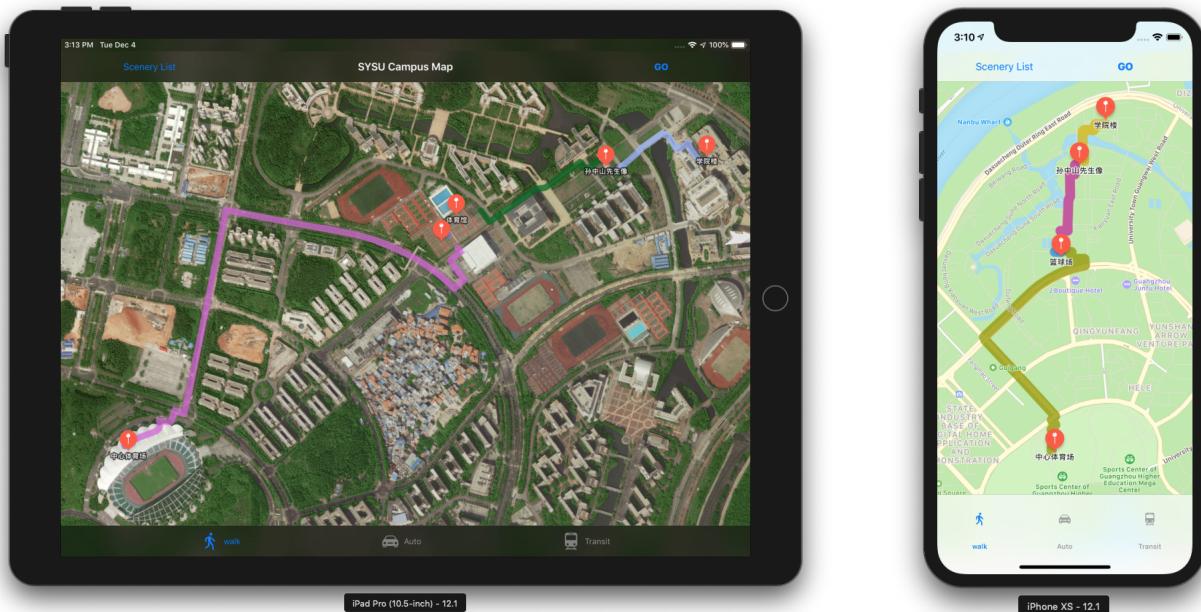
- 多线程并发请求处理 MapView中的MKDirection.Request 是一个用于请求路径的类，我们由于需要快速获取路径，所以采用了多线程请求的方式，但是短时多次请求会导致服务器认为这是一个Spam程序，因此做了一定的随机访问处理进行消除。
- 串行并行数据冒险消除由于数据是并行请求的，因此回到串行处理是要使用对应的优化算法和控制方式。iOS提供了DispatchQueue以消除部分数据冒险，我在此基础上进行了改进，添加了串行验证的方式，进一步防止数据冒险。
- 显示优化算法判断用户显示环境，切换对应的地图

## 6. 函数闭包返回

### 2.3.5 实现功能

1. 2D地图实时显示
2. 3D地图实时显示
3. 用户地理位置显示
4. 用户POI请求
5. 有序浏览模式
6. 无序浏览模式
7. 用户兴趣点显示
8. 步行浏览模式
9. 驾车浏览模式
10. 简单地图显示
11. 卫星地图显示
12. 用户地理位置检索
13. 路线导航与指示

### 2.3.6 测试截图



### 2.3.7 可扩展性

现在这个软件不仅能够在iOS平台使用，只需要修改40行代码就能够在Mac OS X平台上面运行，真正做到了全平台开发。

### 3 分工、贡献与自我评分

	分工	贡献度	自我评分
曾天宇	完成iOS平台开发，报告撰写	0.33	10/10
陈鸿峰	完成网页端平台开发，报告撰写	0.33	10/10
黄杨峻	完成微信小程序平台开发，报告撰写	0.33	10/10

### 4 项目总结

本次项目是数据结构课的最后一次项目，虽然最短路径算法实现起来比较简单，但我想做一些更具有挑战性的事情，所以就想到把项目做大，做成一个全平台可用的应用。总体来说，这次项目汲取了之前几次项目的经验，大家各自用不同语言不同平台开发，最后再整合再一起，有一个完整的效果，还是比较好的。希望我们这种迎难而上的精神以后可以继续保持。