

TJUNavigation

- 姓名：陈姝宇
- 日期：2019年6月9日

TJUNavigation

- 1.需求说明
 2. 关键技术
 - 2.1 数据结构与算法
 - 2.1.1无向图
 - 2.1.2 最短路径算法
 - 2.2 代码技术
 - 2.2.1 Android
 - 2.2.2 kotlin1.3
 - 2.2.3 高德开发平台
 3. 程序运行说明
 - 3.1 自己编译运行
 - 3.1.1 获得初始化数据
 - 3.1.2 运行 app
 - 3.2 安装打开apk
 4. app运行使用说明
- 参考文献

1.需求说明

编写一个天大新校区的导航软件，要求给出校内所有建筑和道路并具有道路是否可通行的标志，软件可以给出两点间机动车和自行车的最短路线。

教学模式：理解并分析题目需求，查找资料，设计数据结构和算法，选择最优算法和编程工具。

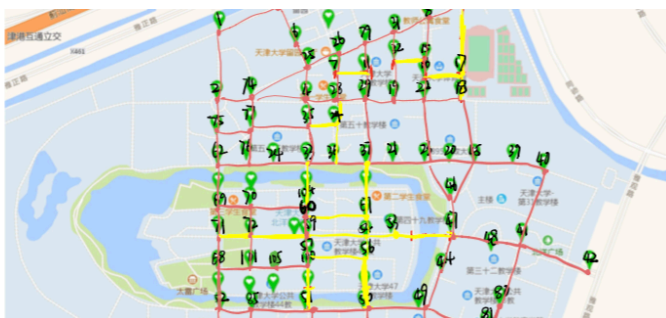
2. 关键技术

2.1 数据结构与算法

2.1.1无向图

结点 -- Node

将地图中的所有建筑物和路口看作无向图中的结点，如果这些地方能够相互直接达到，而不穿过别的建筑物或者路口则有通路，可以看项目 `res/raw/marker.pdf` 里面标注的点、边信息，黄色线为只有步行能够通过，部分截图如下



```
/**
 * 每个点的名称、对应的编号、坐标
 */
data class Node(
    val number: Int,
    val location: String,
    val name: String
)
```

路径 -- Edge

封装类 Edge，内部具有通过网络请求获得两个连通结点的距离和时长的方法，并将边的信息写入文件当中，便于后续根据边的信息构造无向图的邻接矩阵

```
class Edge(private val n1: Int, private val n2: Int, private val p: String)
{
    .....
    private fun getDistance(p: String) {
        if (node1 != null && node2 != null) {
```

```

        GlobalScope.launch(Dispatchers.Unconfined) {
            RetrofitFactory.api.getDistance(ConstValue.KEY,
            node1.location, node2.location).awaitAndHandle {
                File("app/src/main/res/raw/edges.csv").writeText(
                    "$p,${node1.number},${node2.number},ERROR",
                    Charsets.UTF_8
                )
            }?.let {
                distance = it.route.paths[0].distance.toDouble()
                time = it.route.paths[0].duration.toDouble()
                File("app/src/main/res/raw/edges.csv").appendText(
                    "$p,${node1.number},${node2.number},${distance},${time}\n",
                    Charsets.UTF_8
                )
            }
        }
    }
}
}
}

```

2.1.2 最短路径算法

SPFA算法

Bellman-Ford using queue optimization

最短路径算法比较：Dijkstra：适用于权值为非负的图的单源最短路径，用斐波那契堆的复杂度 $O(E + V \lg V)$ BellmanFord：适用于权值有负值的图的单源最短路径，并且能够检测负圈，复杂度 $O(VE)$ SPFA：适用于权值有负值，且没有负圈的图的单源最短路径，论文中的复杂度 $O(kE)$ ， k 为每个节点进入 Queue 的次数，且 k 一般 ≤ 2 ，但此处的复杂度证明是有问题的，其实 SPFA 的最坏情况应该是 $O(VE)$ 。Floyd：每对节点之间的最短路径。

算法思想：我们用数组 d 记录每个结点的最短路径估计值，用邻接表来存储图 G 。我们采取的方法是动态逼近法：设立一个先进先出的队列用来保存待优化的结点，优化时每次取出队首结点 u ，并且用 u 点当前的最短路径估计值对离开 u 点所指向的结点 v 进行松弛操作，如果 v 点的最短路径估计值有所调整，且 v 点不在当前的队列中，就将 v 点放入队尾。这样不断从队列中取出结点来进行松弛操作，直至队列为空为止

个人实现算法时做的调整：① 为了便于回溯，构造了一个数据结构 Info 存有最短路径值和使其松弛的点的编号 ② 为了利用存储空间，一开始不给数组赋初值，将 null 处的结点的最短路径看做无穷大。数组的下标号对应结点编号。

数据结构

```

/**
 * 存放上一个结点
 * 和权重
 */

```

```

data class Info(
    val node: Int,
    val w: Double,
    val t: Double
)

/**
 * map 为一个邻接矩阵的存储, key存的是结点, value存的是和这个结点相邻的所有结点以及路径
 * 所购成的小 map
 */
map: HashMap<Node, HashMap<Node, String>>

```

算法实现的源代码如下:

```

/**
 * 邻接矩阵
 * n1: 起始点
 * n2: 结束点
 */
fun getPaths(map: HashMap<Node, HashMap<Node, String>>, n1: Int, n2:
Int): List<Info> {

    // 权重记录表
    val table = arrayOfNulls<Info>(108)

    // 队列
    val queue = LinkedBlockingQueue<Node>()

    // 初始化队列和列表, 加入源结点
    queue.add(NodeUtils.getNodeByNumber(n1))
    table[n1] = Info(n1, 0.0, 0.0)

    // 找出其他结点到 n1 的最短路径
    // 如果队列不为空, 则循环
    while (queue.isNotEmpty()) {

        // 先取出队头元素并删除
        val head = queue.poll()

        // 如果队头元素不为空则继续执行
        head?.let { head ->

            // 队头元素的权重
            val headW = table[head.number]!!.w
            val headT = table[head.number]!!.t

            // 先取出这个队头元素的邻接 map
            val headMap = map[head]

```

```

        // 邻接 map 不为空则继续执行
        headMap?.forEach { temp ->
            // 和 head 结点之间的权重
            val dd = temp.value.split(",")
            val dw = dd[0].toDouble()
            val dt = dd[1].toDouble()

            // 如果表中的该元素是null, 说明是 ∞
            if (table[temp.key.number] == null) {
                // 更新表
                table[temp.key.number] = Info(head.number, headW +
dw, dt)

                // 将被松弛的元素加入队列中

                queue.add(NodeUtils.getNodeByNumber(temp.key.number))
            } else {
                table[temp.key.number]?.let { info ->
                    // 本来的权重
                    val weight0 = info.w
                    if (weight0 > dw + headW) {
                        table[temp.key.number] = Info(head.number,
dw + headW, headT)

                        queue.add(NodeUtils.getNodeByNumber(temp.key.number))
                    }
                }
            }
        }

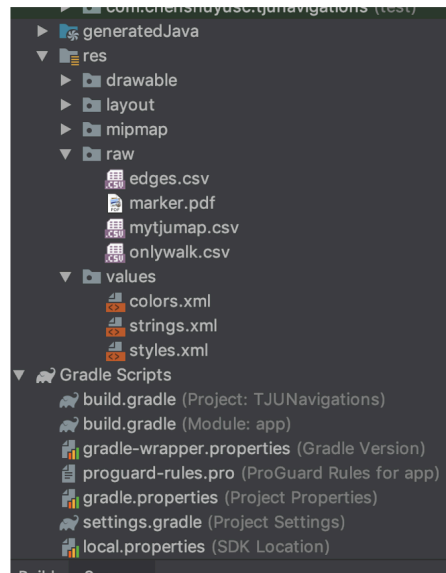
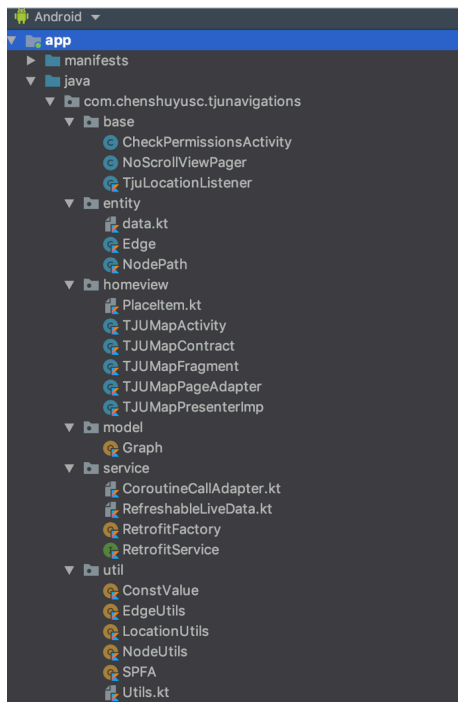
        val infos = mutableListOf<Info>()
        table.getInfos(n1, n2, infos)
        infos.reverse()
        return infos
    }

    // 回溯
    private fun Array<Info?>.getInfos(n1: Int, n2: Int, infos:
MutableList<Info>) {
        .....
    }
}

```

2.2 代码技术

整个项目架构图如下：



2.2.1 Android

MVP设计模式 MVP 把 Activity 中的 UI 逻辑抽象成 View 接口，把业务逻辑抽象成 Presenter 接口，Model 类为数据 (JavaBean 实体类，用于保存实例数据)。在 MVP 模式中 Activity 的功能就是响应生命周期和显示界面，具体其他的工作都丢到了 Presenter 层中进行完成，Presenter 其实是 Model 层和 View 层的桥梁。Model 层和 View 层解藕，提高了代码的复用性和扩展性，并且更有利于管理整个项目。

```
/**
 * 使用 contract 来统一管理 view 层和 presenter 层的接口
 */
class TJUMapContract {
    interface TJUMapView {
        fun onSuccess(ns: List<Info>, n1: Int, n2: Int)
        fun onNull()
    }

    interface TJUMapPresenter {
        fun getNavigation(n1: String, n2: String, kind: String)
    }
}
```

Retrofit 网络请求封装成 RetrofitFactory、RetrofitService

目前这个项目只使用了一个高德地图提供的api，用来获得两点之间的路径

```
object RetrofitFactory {

    private val client = OkHttpClient.Builder()
        .retryOnConnectionFailure(false)
        .connectTimeout(20, TimeUnit.SECONDS)
```

```

        .readTimeout(20, TimeUnit.SECONDS)
        .writeTimeout(20, TimeUnit.SECONDS)
        .build()

    private val retrofit: Retrofit = Retrofit.Builder()
        .client(client)
        .addConverterFactory(GsonConverterFactory.create())
        .baseUrl("https://restapi.amap.com/")
        .addCallAdapterFactory(CoroutineCallAdapterFactory())
        .build()

    val api: RetrofitService = retrofit.create(RetrofitService::class.java)
}

interface RetrofitService {
    @GET("v3/direction/walking")
    fun getDistance(@Query("key") key: String, @Query("origin") origin:
String, @Query("destination") destination: String): Deferred<Distance>
}

```

Coroutines 利用协程来打造比线程更加轻量级的线程，来进行读取文件和网络请求操作等耗时操作，同时扩展协程的 `await` 方法将异步的操作同步化

```

GlobalScope.launch {
    isReady = GlobalScope.async {
        val inputStream0 =
this@TJUMapActivity.resources.openRawResource(R.raw.mytjumap)
        val bufferedReader0 =
inputStream0.bufferedReader(Charsets.UTF_8)
        val lines0 = bufferedReader0.readLine()
        .....
        // 读取文件等耗时操作
    }.await()
}

```

自定义 view 通过继承和扩展来构造切合自己需求的 view 控件

```

/**
 * 不能左右滑动的 ViewPager，防止和地图滑动冲突
 */
public class NoScrollViewPager extends ViewPager {

    .....

    @Override
    public boolean onTouchEvent(MotionEvent arg0) {

```

```

        // TODO Auto-generated method stub
        if (isCanScroll) {
            return super.onTouchEvent(arg0);
        } else {
            return false;
        }
    }

    @Override
    public boolean onInterceptTouchEvent(MotionEvent arg0) {
        // TODO Auto-generated method stub
        if (isCanScroll) {
            return super.onInterceptTouchEvent(arg0);
        } else {
            return false;
        }
    }
}

```

2.2.2 kotlin1.3

扩展函数/高阶函数 让代码质量更高，更优雅

```

fun MutableList<Item>.addPlace(place: String, position: String, click:
(String) -> Unit) =
    add(PlaceItem(place, position, click))
recyclerView.withItems {
    .....
        lines.forEach {
            .....
                addPlace(strs[3], "${strs[1]},${strs[2]}") { name -
>
                    // 点击之后将名字填充到相应的 ET
                    when (clickP) {
                        ConstValue.CLICK_BEGIN -> {
                            begin.setText(name)
                        }
                        ConstValue.CLICK_END -> {
                            end.setText(name)
                        }
                    }
                }
            }
        }
    }
}

```


`data class` 构造类似于 C++ 语言的结构体，也就是算法的数据结构，便于进行数据的存储

```
/**
 * 邻接矩阵
 */
data class AdjacencyNode(
    val node: Node,
    val map: HashMap<Node, Double> // 存有这个结点所有相邻结点的编号和对应的路径长度
)
```


`Object` 单例类，提供整个项目的常量池，防止出现“magic number”，并提供一些很多类都要用的公用方法

```
/**
 * 存放常量
 */
object ConstValue {
    val KEY = "88a8031eb0b7b37a0cb81d287e5c29b9"
    .....
}

object NodeUtils {
    .....
    /**
     * 根据编号获得对应的地名
     * 利用 kotlin 语言提供的 map 的 foreach 扩展函数来遍历map
     */
    fun getNodeByNumber(number: Int): Node? {
        nodeSet.forEach { node ->
            if (node.number == number)
                return node
        }
        return null
    }
    .....
}
```

2.2.3 高德开发平台

使用高德地图的 Android 地图 SDK，获得地图并进行自定义标点、绘线等，下图为高德地图控制台我的应用信息

 TJUNavigation
2019-06-03创建

[🔗](#) [🗑️](#) [+](#) [▼](#)

Key名称	Key	绑定服务	操作 ①
DemoLocationKey	e83415528db225c2f89f0f90bfabdb3e	Android平台	设置 删除
TJUKey	d391c49ef2d329c32ecb7cd12eedfd1d	Android平台	设置 删除

阅读高德地图的 api 文档后对高德地图提供的方法再次进行封装，下面举一个例子：

```
// 封装结点的扩展方法 ---- 根据结点特征在地图上标点
private fun Node.drawMarker(p: String) {
    val ll = this.location.split(",")
    val latlng = LatLng(ll[1].toDouble(), ll[0].toDouble())
    // 绘制终点
    val markerOption = MarkerOptions()
    markerOption.position(latlng)
    markerOption.title(this.name).snippet(this.location)
    markerOption.draggable(true) // 设置Marker可拖动
    when (p) {
        CLICK_END -> markerOption.icon(
            BitmapDescriptorFactory.fromBitmap(
                BitmapFactory
                    .decodeResource(resources,
                        R.drawable.ic_end).changeSize(106f)
            )
        )
        .....
    }
}
// 将Marker设置为贴地显示，可以双指下拉地图查看效果
markerOption.isFlat = true // 设置marker平贴地图效果
aMap.addMarker(markerOption)
}
```

使用高德地图的数据管理平台，在线标点，便于提前测试各种情况



3. 程序运行说明

3.1 自己编译运行

3.1.1 获得初始化数据

将 util 包下的的所有 Object 类的 init 方法取消注释，运行 EdgeUtils 下的 main 方法

(注：如果res/raw下的 edges.csv 和 onlywalk.csv 已有数据，则这一步可以省略)

```
17  */
18  object EdgeUtils {
19
20      val edges : ArrayList<EdgeInfo> = arrayListOf<EdgeInfo>()
21
22      fun getEdges(lines: List<String>){...}
23
24
25
26
27
28
29
30
31
32
33
34
35
36      init {
37          val lines : List<String> = File( pathname: "app/src/main/res/raw/edges.csv").readLines(Charsets.UTF_8)
38          lines.forEach { line ->
39              val strs : List<String> = line.split( ...delimiters: ",")
40              val node1 : Node? = NodeUtils.getNodeByNumber(strs[1].toInt())
41              val node2 : Node? = NodeUtils.getNodeByNumber(strs[2].toInt())
42              if (node1 != null && node2 != null) {
43                  edges.add(
44                      EdgeInfo(strs[0], node1, node2, strs[3], strs[4])
45                  )
46              } else {
47                  Log.d( tag: "ERROR", msg: "node no exist")
48              }
49          }
50      }
51  }
```

```
object NodeUtils {
    private val nodeSet : MutableSet<Node> = mutableSetOf<Node>()
    private val nodePaths : MutableList<NodePath> = mutableListOf<NodePath>()

    /**
     * 获得所有结点的信息
     */
    fun getNodes(lines: List<String>) {...}

    init {
        val lines : List<String> = File( pathname: "app/src/main/res/raw/mytjumap.csv").readLines(Charsets.UTF_8)
        repeat(lines.size) { i ->
            if (i > 0) {
                val strs : List<String> = lines[i].split( ...delimiters: ",")
                nodeSet.add(Node(strs[0].toInt(), location: strs[1] + "," + strs[2], strs[3]))
            }
        }
    }
}
```

```
53  /**
54  * 获得所有连通的边
55  * 最后一个参数是指车辆可以通过，还是只有步行能够通过
56  * 由于每获得一次距离需要发一次网络请求，所以这个函数单独运行，生成csv文件保存网络请求的信息
57  * 可以看 res/raw/marker.pdf 里面标注的边信息，黄色线为只有步行能够通过
58  * 每个点的信息在 mytjumap.csv 中
59  */
60  @JvmStatic
61  fun main(args: Array<String>) {
62      Edge( n1: 1, n2: 12, CAR)
63      Edge( n1: 1, n2: 2, CAR)
64      Edge( n1: 2, n2: 74, CAR)
65      Edge( n1: 12, n2: 6, CAR)
66      Edge( n1: 6, n2: 25, CAR)
67      Edge( n1: 26, n2: 7, CAR)
68      Edge( n1: 25, n2: 26, CAR)
```

看到控制台输出数据，查看res/raw下的 edges.csv 和 onlywalk.csv ，均出现初始数据

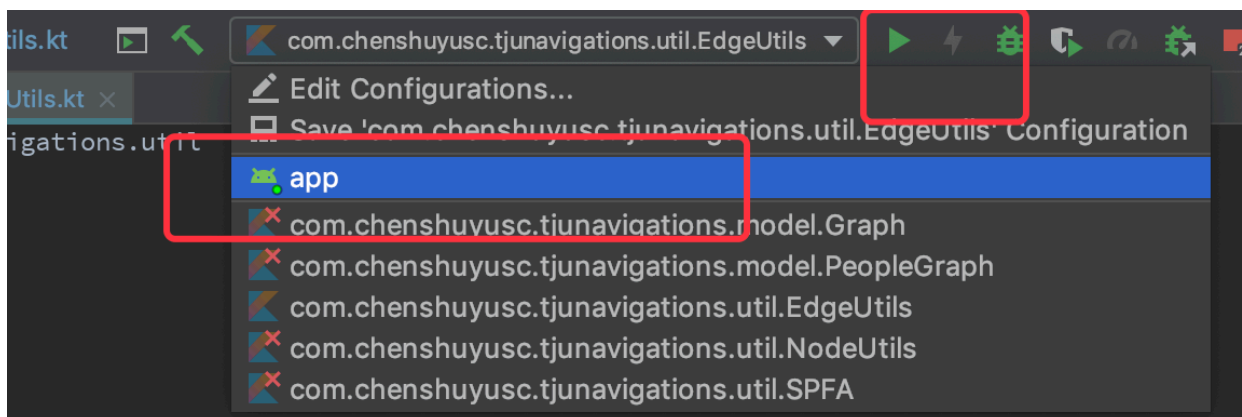
```
app x com.chenshuyusc.tjunavigations.util.EdgeUtils x
"/Applications/Android Studio.app/Contents/jre/jdk/Contents/Home/bin/java" ...
知园,117.316104,39.001887,第五十四教学楼,117.317154,39.001886
游泳馆1895路口,117.319316,39.001140,游泳馆,117.319301,39.001757
体育馆,117.319341,39.002192,体育馆路口,117.318243,39.002279
游泳馆,117.319301,39.001757,足球场,117.320591,39.001830
诚园八斋,117.315091,38.999438,正园,117.315097,38.998203
足球场,117.320591,39.001830,羽毛球场,117.320598,39.001171
足球场,117.320591,39.001830,北洋门诊部,117.320715,39.003853
第四十九教学楼,117.318130,38.997452,圆圈楼路口,117.320247,38.997504
诚七五十教路口,117.317216,38.999371,第二学生食堂,117.317210,38.997912
修园,117.315169,38.996487,第四十三教学楼,117.315067,38.995334
第四十七教学楼,117.317221,38.996712,齐园,117.317130,38.995357
第一学生食堂知园路口,117.316062,39.001098,诚园六斋,117.316081,39.000449
第一食堂诚六路口,117.315118,39.000480,诚园六斋,117.316081,39.000449
诚园六斋,117.316081,39.000449,诚园七斋,117.316076,38.999407
图书馆路口,117.311948,38.997401,图书馆西,117.313104,38.997341
图书馆西,117.313104,38.997341,图书馆东,117.315075,38.997295
第四十五教学楼,117.315164,38.996781,第四十七教学楼,117.317221,38.996712
第四十七教学楼,117.315067,38.995334,第二学生食堂,117.317210,38.997912
```

```
onlywalk.csv x
1 知园,117.316104,39.001887,第五十四教学楼,117.317154,39.001886
2 游泳馆1895路口,117.319316,39.001140,游泳馆,117.319301,39.001757
3 游泳馆,117.319301,39.001757,足球场,117.320591,39.001830
4 体育馆,117.319341,39.002192,体育馆路口,117.318243,39.002279
5 足球场,117.320591,39.001830,羽毛球场,117.320598,39.001171
6 足球场,117.320591,39.001830,北洋门诊部,117.320715,39.003853
7 诚七五十教路口,117.317216,38.999371,第二学生食堂,117.317210,38.997912
8 诚园八斋,117.315091,38.999438,正园,117.315097,38.998203
9 第四十九教学楼,117.318130,38.997452,圆圈楼路口,117.320247,38.997504
10 修园,117.315169,38.996487,第四十三教学楼,117.315067,38.995334
11 第四十七教学楼,117.317221,38.996712,齐园,117.317130,38.995357
12 第一食堂诚六路口,117.315118,39.000480,诚园六斋,117.316081,39.000449
```

```
edges.csv x
1 CAR,12,6,128.0,102.0
2 CAR,1,12,204.0,163.0
3 CAR,1,2,194.0,155.0
4 CAR,6,25,76.0,61.0
5 CAR,2,74,120.0,96.0
6 CAR,25,26,97.0,78.0
7 CAR,26,7,53.0,42.0
8 CAR,31,8,110.0,88.0
9 CAR,26,79,85.0,68.0
10 CAR,79,31,101.0,81.0
```

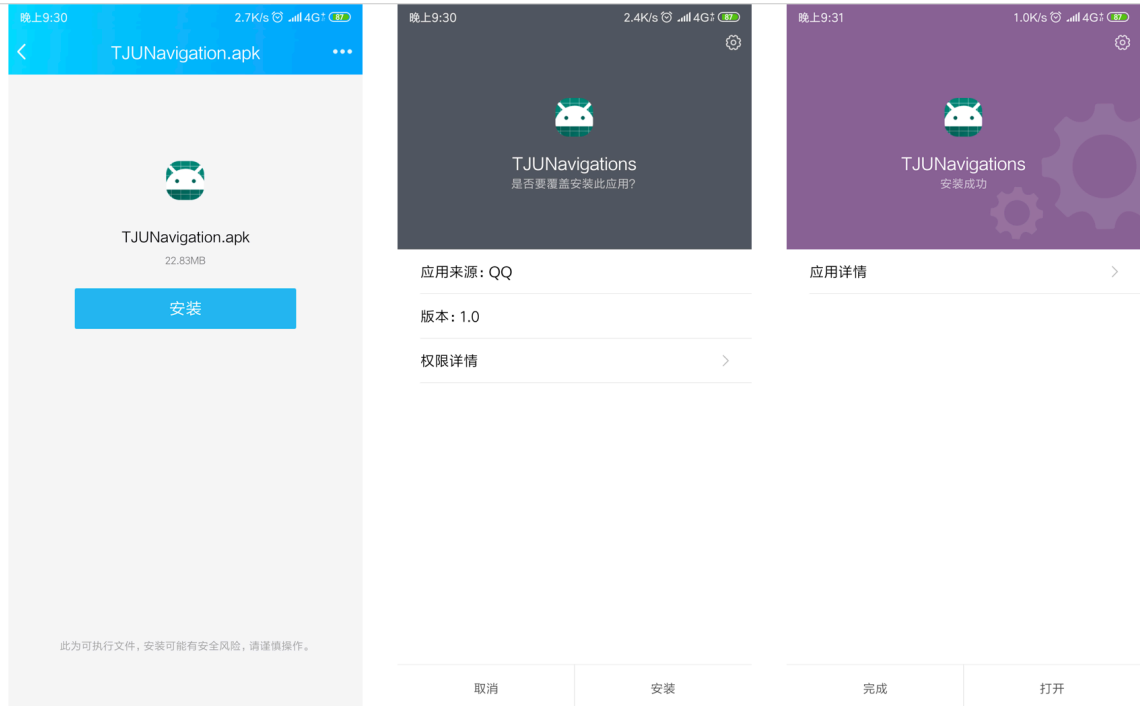
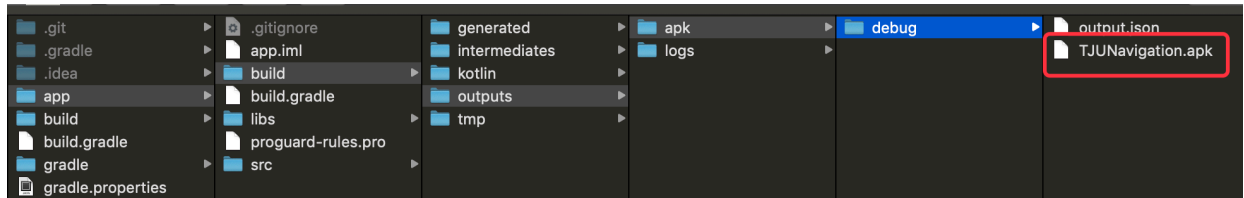
3.1.2 运行 app

将 util 包下的所有 Object 类的 init 方法注释掉，将运行内容改为 app



3.2 安装打开apk

按照如下文件路径找到 apk 包，发送到手机上安装打开即可



4. app运行使用说明

参见视频（在 /video 目录下）

参考文献

[Dijkstra Bellman Ford SPFA Floyd算法复杂度比较](#)

[SPFA 算法详解\(最短路径\)](#)