Report-Project2

黄子骕 21307130013

算法选择

在此次 pj 中，我选择使用 Dijkstra 算法来实现导航系统中最短路径的查找，因为在地图为一个带权图，其中没有负权边，并且极大概率有环出现，综合以上几点，可选的最短路径算法有 Dijkstra 算法以及 Bellman-Ford 算法，考虑到没有负权出现，因此选择效率相对较高的 Dijkstra 算法。

具体实现

edge结构

- is_access：记录两个地点之间是否有边（由于考虑到在通行时可以从A到B，一定可以以相同的方式从B到A，因此只表示是否有边，不记录方向）
- bus_time：记录公交通行时间
- walk_time：记录步行时间

node结构

- name：保存地点名称
- number：保存地点编号（便于实现邻接矩阵）
- distance：用于在heap中保存距离
- 重载<：使得在使用make_heap时可以获得极小堆

navigator类

- 私有成员
  - map：保存地图的边信息
  - nodes：保存地图所有地点信息
- 公有成员函数：
  - ***init_map***：从 `map.txt` 中读取地图信息并且初始化地图
  - ***find_shortest_path_walk***：输入起点与终点，输出最短路径，记录运行时间并输出
    1. 调用 ***algorithm*** 库中的 ***clock()*** 开启计时

2. 在地图中找到起点与终点位置
3. 利用 ***algorithm*** 库中的 ***make_heap()*** 函数构造极小堆并初始化其距离（起点为0，其他为正无穷）
4. 按照 Dijkstra 算法， 利用循环不断从堆顶取距离最小元素并进行 relaxation 操作更新堆中部分元素的距离，最后维护堆
5. 调用 ***print_map*** 输出最短路径
6. 结束计时并输出

- ***find_shortest_path_bus*** ： 与 ***find_shortest_path_walk*** 相同，区别在于选择公交通行时间数据
- ***find_shortest_path_nolimits***：同上，区别在于在公交、步行中选择较快的数据
- ***decrease_dis***：实现对于 heap 中元素进行 relaxation 操作
- ***print_path***：根据 Dijkstra 算法得到的记录前驱的 pre 数组来还原并打印最短路径
- ***find_number***：输入地点的名字，返回其编号（用于确定其在邻接矩阵中的位置）
- ***find_node***：在 heap 中寻找需要修改的 node，返回其位置
- ***print_nodes***：输出地图中包含的所有地点
- ***print_map***：输出 map 中包含的边信息

使用方法

1. 运行程序后会有如下提示：

```
Please enter your command(1-quit , 2-navigator):
2
```

此时输入命令，1表示退出，2表示进入导航

2. 进入导航后会依次提示输出起点、终点、出行方式信息，样例如下：

```
Please enter your start point: TJU
Please enter your end point: SHU
Please decide your trip mode(1-on foot, 2-by bus, 3-no limits): 1
```

3. 信息输入完成后，将进行最短路径的计算并输出相应信息：样例如下：

```
Your best route: TJU ---> WJC ---> FDU ---> SHU
Total time on foot is: 120 min
Running time is: 1 ms
```

性能分析

- ***find_shortest_path*** 复杂度分析（设地点数为 N ，边数为 E）：
    1. 在图中寻找给定起点与终点需要 O(N)
    2. 初始化堆中需要 O(N)
    3. 对于不断查找的循环，最多循环 N 次（所有点都走过一遍）
        1. 取出堆顶元素，即直接索引 heap 中的第一个元素即可
        2. 遍历堆，同时查地图的相应两点间是否有边，若有则进行 relaxation 操作， 每次relaxation操作在本算法在实际上就是一个赋值操作，并没有进行维护，且该操作总的次数受 E 限制，总的时间复杂度为 O(E)
        3. 所有的 relaxation 操作结束后，将堆顶元素 pop 出去并重新建堆，此过程时间复杂度为 O(N)
        综上，这个循环的总时间复杂度为 O(E + N^2)，即 O(N^2)
    4. 最后还原路径的操作最多需要 N 次操作
    综上，寻找最短路径函数的复杂度为

$$O(N^2)$$

本算法相较于课上所描述的 Dijkstra 算法区别在于进行 relaxation 操作时，由于c++没有提供对 heap 进行 decrease_key 的操作，由于pj时间较紧张，因此使用了的令 relaxation 仅仅修改值，一个循环中所有 relaxation 结束后再 make_heap 重新建堆的策略，故复杂度高了一些。

对于改进方法，可以自己实现一个 decrease_key 操作在修改的同时进行维护，这样 relaxation 操作的复杂度就为 O(logN)，同时省去了了每次循环重新建堆的开销，函数总的复杂度降低为

$$O(ElogN)$$