COVID-19 疫情环境下低风险旅行查询模拟系统

各模块设计说明

姓　　名：刘立敏

学　　号：2018211398

学　　院：计算机学院

专　　业：计算机类

班　　级：2018211310

指导老师：张海旸

各模块设计说明

一、旅行线路设计和输出模块

1、单源最短路径算法（Dijkstra）

①算法思想：采用贪心策略。Dijkstra 算法从源结点开始，每次遍历到源结点距离最近且未访问过的顶点的邻接结点，直至扩展到终点。

> 有向图 G={V,E}，V 为顶点集，E 为边集，V0 为源，S 为已确定的顶点集合，T=V-S 为尚未确定的顶点集合。
>
> 1. 初始时令 S={V0},T=V-S={其余顶点},
>
> 若存在弧< Vi,Vj >，d(Vi,Vj)为< Vi,Vj >弧上的权值
>
> 若不存在弧< Vi,Vj >，d(Vi,Vj)为∞
>
> 2. 从 T 中选取一个与 S 中顶点有关联边且权值最小的顶点 W，加入到 S 中
>
> 3. 对其余 T 中顶点的距离值进行修改：若加进 W 作中间顶点，从 V0 到 Vi 的距离值缩短，则修改此距离值
>
> 重复上述步骤 2、3，直到 S 中包含所有顶点。

②算法内容：

在本系统中，所有城市与路线的集合构成一张有多重边的有向图，城市相当于图的顶点，路线相当于图的边。边的权值是感染的风险值。本系统采用邻接表方式存储图。

对于基础版本，只计算停留城市的风险。因此边（也就是班次路线）的权值实际上是在城市中停留的风险值。我们不妨把一个班次的风险值定义为在这条路线上的终点城市停留的风险值（终点是旅客目的地的情况除外）。每到达一个（中转的）城市，就要加上在这里停留的风险。

对于扩展版本，还需考虑乘坐交通工具的风险。因此，边权被定义为在这趟交通工具上的风险值加上终点城市停留的风险值（目的地除外）。

至此，我们就将实际情况转化为了标准的 Dijkstra 算法适用模型，从而利用

Dijkstra 算法求出了风险最小的路线。

③时间复杂度

采用邻接表存储方式。时间复杂度主要由两层 while 循环决定。其中外层循环用于遍历顶点，因此循环次数不会超过 V 次（V 为顶点数，即城市数量），内层循环用于遍历与一个顶点关联的所有边。由于两个城市之间可能有多条线路（即有多重边），也可能没有线路（规定了不能所有城市均直达），平均来看，内层循环次数也接近于 V。

因此该算法的时间复杂度为 O（V^2）。（V 为顶点数，即城市数）

④空间复杂度

该算法消耗的空间主要用于存储每个城市的风险值、到达时间、上个城市序号、到达的班次号等数组，因此空间复杂度为 O（V）。（V 为顶点数，即城市数）

⑤算法特点

Dijkstra 算法虽然采用贪心思想，每次都追求局部最优，但是最终却能达到全局最优。

Dijkstra 算法只适用于不含负权边的图。

如果图的边数远小于顶点数的平方，则可以采用堆这种数据结构进行优化，降低时间复杂度。

⑥算法与模块联系

扩展版本的非限时最小风险策略（Extended_non_time_limited）直接运用了该算法。

2、深度优先搜索（DFS）

①算法思想：

> 深度优先遍历图的方法是，从图中某顶点 v 出发：
>
> （1）访问顶点 v；
>
> （2）依次从 v 的未被访问的邻接点出发，对图进行深度优先遍历；直至图中和 v 有路径相通的顶点都被访问；
>
> （3）若此时图中尚有顶点未被访问，则从一个未被访问的顶点出发，重新进行深度优先遍历，直到图中所有顶点均被访问过为止。

②算法内容：

为了获取从出发城市到达目的城市的最小风险路径，我们可以使用最朴素的搜索算法直接得到所有能到达目的地的路径，然后在这些路径中进行比较，剔除超出时间限制的（如果有的话），选择其中风险最小的。

但是，由于我们的系统是尽量贴合实际情况的。在不加以限制的情况下，DFS 可能得到一条中转次数非常多的很长的路线，但现实情况下，这样的线路几乎不可能是最优路线，更不具有实际意义。为了避免这些无谓的时间和空间开销，我们采用了剪枝策略和回溯思想对 DFS 进行优化。

我们采用了最大中转次数（MAX_TRANSITION_TIMES）作为剪枝条件，将其

设置为值为 4 的常量。这意味着，在搜索过程中，任何中转次数超过 MAX_TRANSITION_TIMES 的分枝都将直接被舍弃。

经过测试，我们发现根据测试样例的时刻表数据获得的任意两个城市之间的路线中，中转次数都没有超过 3 的。我们稍稍放宽一些条件，把值设为 4，以应对更大规模的城市数量和时刻表。如果希望程序拥有更高的性能，也可以直接修改为 3。

在用 DFS 求出了所有可能满足条件的解决方案后，我们将对这些路线一一进行计算和比较，找出最优的那条路线。

③时间复杂度：

由于我们采用了剪枝策略进行优化，时间复杂度虽然下降了，但也不便计算了。因此这里只分析标准的邻接表表示的 DFS 时间复杂度。

设 E 为边数（路线数），V 为顶点数（城市数）。在 DFS 中，访问所有顶点所需时间为 O（V）；为所有顶点找到它们的所有邻接顶点的时间为遍历整个边表的时间，即 O（E），因此，总的时间复杂度为 O（V+E）。

④空间复杂度：

DFS 需要递归工作栈。最坏的情况是图退化成线性结构，递归栈沿着线性结构不断加深，最终形成 V 层栈。每层栈的复杂度为 O（1）。因此，总的空间复杂度为 O（V）。

⑤算法特点：

深度优先搜索有递归和非递归两种设计方法。递归方式的程序结构相对简单易懂，但是当数据量较大时，递归以堆栈方式大量消耗空间资源，且可能受到系统堆栈容量的限制，产生溢出，此时采用非递归实现更好。

由于该系统的数据规模不算很大，因此使用递归方法完全没有问题。

⑥算法与模块联系：

扩展版本的限时最小风险策略（Extended_time_limited），基础版本的非限时最小风险策略(Basic_non_time_limited)以及基础版本的限时最小风险策略(Basic_time_limited)都采用了先 DFS 后计算并选出路径的算法。

3、算法优劣对比分析及小结

◆ **对比分析**

①Dijkstra 算法的时间复杂度为 $O(V^2)$，DFS 时间复杂度为 O(V+E)，两者空间复杂度均为 O(V)，从性能上看，DFS 算法性能更佳。

②我采用了最大中转次数对 DFS 进行剪枝，使得其性能更进一步，但是相应的代价是，如果时刻表中出现了一些非常极端的数据，那么某条特别长的路径可能恰恰是最优路径，却被我们剪枝掉了。不过，考虑到我的系统是模拟真实情况的，且时刻表都是自己设计的，所以这个影响完全可以忽略不计。

③面对某些极端的时刻表，Dijkstra 算法同样有可能不能得出最优路径。考虑这样一种情况，旅客现在处于中转城市，有一早一晚两趟班次可以去往目的城市。按照 Dijkstra 每次寻求局部最优解的思想，应该选择早出发的那趟班次，以避免在城市过多停留造成风险增加。然而，如果该时刻表的数据非常极端，早出发的班次需要 48 小时才能到达，晚出发的只需要 1 小时即可到达，那么显然又该选择晚出发的城市。这时 Dijkstra 算法得出的解就不是最优解。

同样，这样的极端情况并没有现实意义。而且我经过大量数据的测试（所有城市两两之间查询路线）发现，在正常的时刻表下，用 Dijkstra 作为主算法和用 DFS 作为主算法得到的结果完全一样。因此可以认为，两种算法得到的结果都是最优解。

◆ **小结**

DFS 算法略优于 Dijkstra 算法，我们更多地采用前者。

旅行线路设计和输出模块共分为四个子模块

- 基础版本的有时间限制最小风险策略
- 基础版本的无时间限制最小风险策略
- 扩展版本的有时间限制最小风险策略
- 扩展版本的无时间限制最小风险策略

前三者基于 DFS 算法，后者基于 Dijkstra 算法。

旅行线路设计和输出模块是其他所有模块的基础，其他各模块的功能都是根据本模块得出的路线而实现的，因此这也是整个程序的核心部分。

二、图形界面展示模块

该模块融合了参考结构的主模块部分，调用了系统的其它各个模块，用于控制整个系统的运行，是用户与系统交互的核心模块。该模块分为 7 个子模块。

1、旅客添加与选择子模块

算法思想：用结构表示一名旅客的全部信息，用一个列表（容器）存储各位旅客的信息。

实现功能：支持多旅客查询和模拟。

2、旅客需求获取子模块

算法思想：通过用户在图形界面上的鼠标、键盘操作，触发相应的事件，从而获取用户的旅行计划信息。

实现功能：获取旅客对出发地、目的地、策略等需求的选择，并将其存储到系统的数据文件中。

3、时间线程子模块

算法思想：用一个计时器来显示时间，用其它按钮来控制计时器的计时与暂停。

实现功能：用于控制系统时间的推进。

4、路线查询与显示子模块

算法思想：对第一个模块（路线设计与输出）的应用和图形显示。

实现功能：根据旅客的需求和相关的算法，找出最合适的路线输出给旅客。在有多条最优线路的情况下，会同时显示以供旅客选择。

5、旅行模拟与实时状态显示子模块

算法思想：主要体现在旅客实时位置的展现上。该模块先根据路线设计模块输出的路线来确定旅客在不同时刻的状态（包含出发、到达、停留等），然后再计算并建立出各个时刻与坐标的映射。随着时间的推进，用户的坐标不断改变，并体现在地图中。

实现功能：该模块用于模拟旅客的旅行，并将旅客的实时状态信息在地图中展现出来。同时，该模块也有文字形式展示的旅客实时状态信息。

6、旅客信息显示子模块

算法思想：根据路线设计模块输出的路线来确定旅客在不同时刻的状态（包含出发、到达、停留等），建立映射关系，从而输出用户所查询时间发生的事件。

实现功能：反映旅客在旅行过程中不同时刻的信息和状态。

7、旅行模拟控制子模块

实现功能：控制旅行模拟和开始、暂停与终止，以及其它各模块的控制。'

三、状态动态查询显示模块

设计思想：先建立时间与旅行事件的映射，在查询时根据时间获取旅行动态事件。

实现功能：该模块用于获得旅客选择的路线后对旅客各时刻所处的状态进行查询。

查询的结果将会通过“图形界面展示模块”的“旅客信息显示子模块”展现出来。

四、日志文件处理模块

该模块用于将各种查询和模拟结果输出到文件中，以日志形式保存下来。

在图形界面展示模块里，只要是需要输出日志的地方，就会调用该模块。