华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析　　**年级：**22 级　　**上机实践成绩：**

指导教师：金澈清　　**姓名：**郭夏辉

上机实践名称：路径规划　　**学号：**10211900416　　**上机实践日期：**2023 年 6 月 1 日

上机实践编号：No. 13　　**组号：**1-416

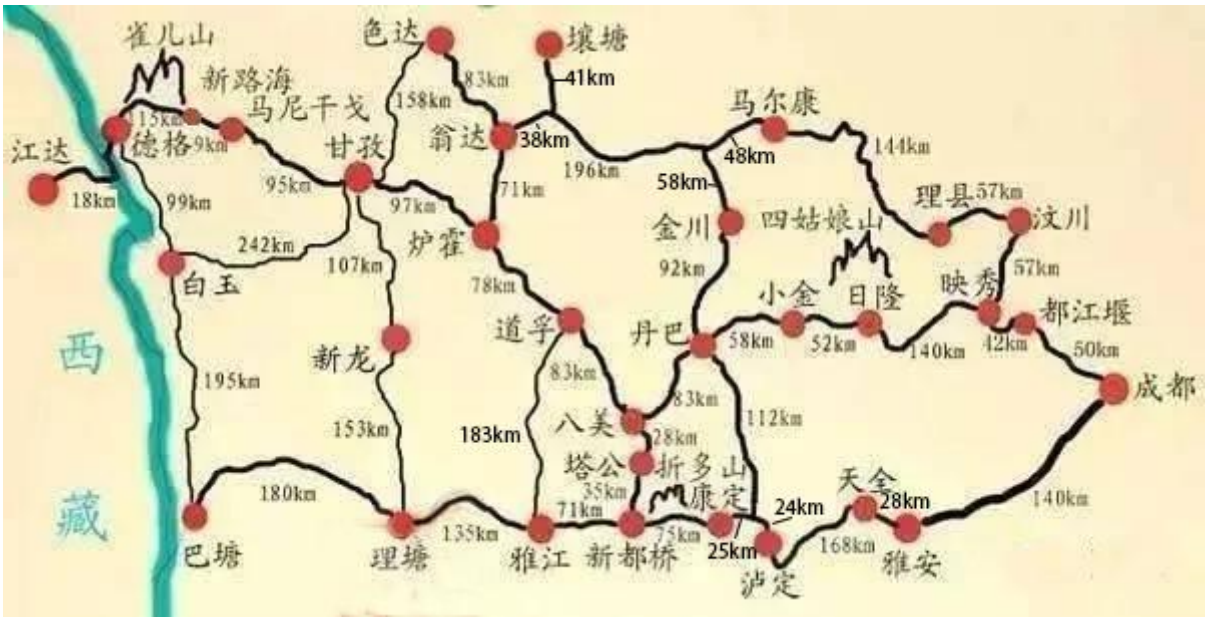
一、目的

1．熟悉算法设计的基本思想

2．掌握最短路径相关算法的思路

二、内容与设计思想

川西风光几枚，以下图片是川西路线图。张三是旅游爱好者，他从成都出发自驾到西藏江达。



1）从成都到江达的最短自驾路线是什么？可以用 Dijkstra 算法来求解。

2）张三把理塘列为必游之地。怎么规划路线，使得总行程最短？

3）张三觉得理塘风景很美，道孚也不错，两个地方如果能够去一个地方的话就心满意足了。应该怎么安排行程使得总行程最短？

4）张三在规划线路的时候，发现不同路况行驶速度不一样。地图中最粗的路径（成都-雅安）表示平均时速可以达到 80 公里每小时，而中等的路径表示平均时速每小时 60 公里每小时，最细的路径表示平均速度仅仅只有 40 公里每小时。那么(2)(3)中用时最短的路径分别是哪一条？

5）（**思考题**）考虑到 Dijkstra 算法仅仅从一段开始寻找路径，效率不高。李教授想到一个高招，就是同时从出发地和目的地进行搜索，扩展搜索节点，然后两个方向扩展的路径会在中途相遇，则拼接起来的路径就是最短路径。如何实现李教授这个想法？

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

1. 编写相关实验代码
2. 写出算法的思路。

首先我们还是来回顾一下上课时学过的求单源最短路径的 Dijkstra 和 Bellman-Ford 算法,这是本次实验的基础:

Bellman-Ford 算法

Bellman-Ford 算法解决的是一般情况下的单源最短路径问题，在这里，边的权重可以为负值。给定带权重的有向图 $G=(V, E)$和权重函数 $w: E\rightarrow \mathbf{R}$，Bellman-Ford 算法返回一个布尔值，以表明是否存在一个从源结点可以到达的权重为负值的环路。如果存在这样一个环路，算法将告诉我们不存在解决方案。如果没有这种环路存在，算法将给出最短路径和它们的权重。

Bellman-Ford 算法通过对边进行松弛操作来渐近地降低从源结点 s 到每个结点 v 的最短路径的估计值 $v.d$，直到该估计值与实际的最短路径权重 $\delta(s, v)$相同时为止。该算法返回 TRUE 值当且仅当输入图不包含可以从源结点到达的权重为负值的环路。

```
BELLMAN-FORD(G,w,s)
1  INITIALIZE-SINGLE-SOURCE(G,s)
2  for i=1to|G.V|-1
3       for each edge(u,v)∈G.E
4            RELAX(u,v,w)
5  for each edge(u,v)∈G.E
6       if v.d>u.d+w(u.v)
7            return FALSE
8  return TRUE
```

就我的个人理解来看，Bellman-Ford 算法可以解决具有负边权的单元最短路径问题，它针对的是每条边，每次迭代都是在把每条边都松弛操作了一遍。算法第 k 轮进行的操作本质上是在求解从源点开始，最多经过 k 条边到达其余各点的最短路径长度，因此只需要进行 n-1 轮便可以完成求解了。（n 为顶点个数，当然这里的图不能包含回路）。

由于算法第 1 行的初始化操作所需时间为 $\Theta(V)$，第 2～4 行循环的运行时间为 $\Theta(E)$，且一共要进行$|V|-1$ 次循环，第 5～7 行的 for 循环所需时间为 $O(E)$，Bellman-Ford 算法的总运行时间为 $O(VE)$。

Bellman-Ford 算法有很多优化的方法，其中一种是采用队列来优化。我们每次仅真正意义上地对最短路程发生变化了的点的相邻边进行松弛操作，但是如何知道当前哪些点的最短路程发生了变化呢？这里可以用一个队列来维护这些点。每次选取队首顶点 u，对顶点 u 的所有出边进行松弛操作。假如说有一条边是从 u 到 v,如果这条边使得源点到 v 的最短路程变短了，即松弛操作成功进行(dis[u]+e[u][v]<dis[v]),并且 v 不在这个维护的队列中，我们就可以将 v 入队。接下来不断地从队列中取出新的队首顶点再进行类似的操作指导队列为空停止。这个思路特别像 BFS,我们在纸上推导时，其实也可以发现 Bellman-Ford 进行的过程与 BFS 类似。

Dijkstra 算法

该算法不能用于解决负边权的问题，采用的其实是一种贪心的策略——每次新拓展一个路程最短的点，更新与其相邻点的路程，当所有边权都为正时，因为不可能存在一个路程更短的且没拓展过的点，所以这个点的路程(即最初时那个估计值)永远不会再发生改变，算法的正确性得到了有效保证。

结合这个算法的具体运行流程(如下页所示)，我们可以发现利用优先队列，可以极大地对时间复杂度进行优化，从 O(n^2)降低至 O(nlgn),当然这是老生常谈的事情了，我额外地发现对于边数较少的稀疏图来说，采用邻接表可以近一步地将时间复杂度下降。

而且结合上一次实验的内容来看，prim 算法与 Dijkstra 算法有挺大的相似之处，只不过前者每次都在更新已知集合与未知集合中临界点处的距离，而后者是在更新从起始点出发到临界点处的距离。

Dijkstra 算法解决的是带权重的有向图上单源最短路径问题，该算法要求所有边的权重都为非负值。因此，在本节的讨论中，我们假定对于所有的边$(u, v) \in E$，都有$w(u, v) \geqslant 0$。我们稍后将看到，如果所采用的实现方式合适，Dijkstra 算法的运行时间要低于 Bellman-Ford 算法的运行时间。

Dijkstra 算法在运行过程中维持的关键信息是一组结点集合 S。从源结点 s 到该集合中每个结点之间的最短路径已经被找到。算法重复从结点集 $V-S$ 中选择最短路径估计最小的结点 u，将 u 加入到集合 S，然后对所有从 u 发出的边进行松弛。在下面给出的实现方式中，我们使用一个最小优先队列 Q 来保存结点集合，每个结点的关键值为其 d 值。

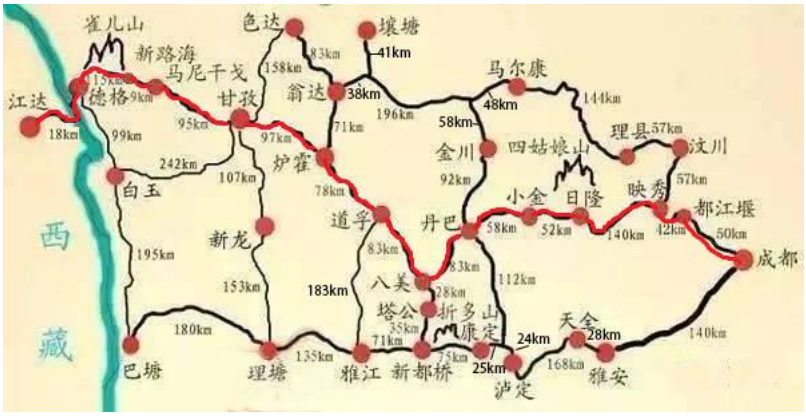
```
DIJKSTRA.(G,w,s)
1  INITIALIZE-SINGLE-SOURCE(G,s)
2  S=∅
3  Q=G.V
4  while Q≠∅
5      u=EXTRACT-MIN(Q)
6      S=S∪{u}
7      for each vertex v∈G.Adj[u]
8          RELAX(u,v,w)
```

在温习了一下两种算法之后，让我们正式地开始实验吧！

1 从成都到江达的最短自驾路线是什么？可以用 Dijkstra 算法来求解。

这个问题很简单，我利用的是基本的 Dijkstra 算法，每次从未知的点的集合中选择距离源点最短的加入到已知的点的集合，并更新它与其相邻所有点的从源点出发的最短距离，之后执行类似的操作即可，直到未知的点的集合为空。

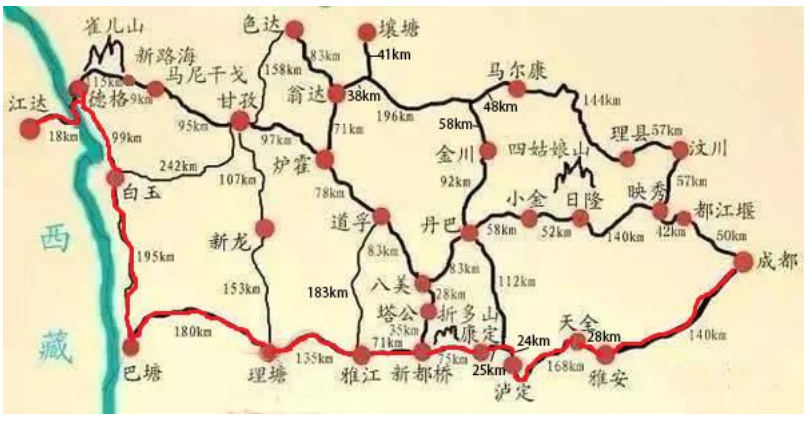
为了回溯方便，我在进行加入点后的更新操作时顺便更新了对应的前驱结点编号，并利用了一个栈来存储路径。为了描绘各个城市更加清晰，我再次使用了 map 容器，而不是构造一个结构体存储编号和 string 城市名。运行结果如下所示：



Path:成都>都江堰>映秀>日隆>小金>丹巴>八美>道孚>炉霍>甘孜>马尼干戈>新路海>德格>江达

Total: 920km

2 张三把理塘列为必游之地。怎么规划路线，使得总行程最短？



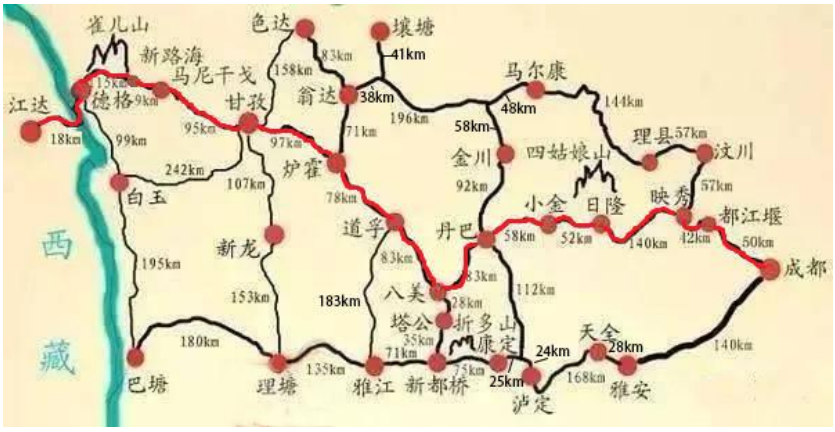
这个题目我的思路是这样的，就是先利用 Dijkstra 算法求出成都出发到理塘的最短路线，然后再求出从理塘出发到江达的最短路线，把两条路线拼在一起便可以求得所需要的路径。在问题的求解过程中，不要忘记每次开始迭代之前的初始化操作。为了回溯方便，我这次求解的顺序是先求出从理塘出发到江达的最短路线，再求出成都出发到理塘的最短路线，这并没有颠覆我们最初的思路，效果上是一样的。

运行结果如下所示:

Path:成都>雅安>天全>泸定>泸定和康定的中间点>康定>新都桥>雅江>理塘>巴塘>白玉>德格>江达

Total: 1158km

3 张三觉得理塘风景很美，道孚也不错，两个地方如果能够去一个地方的话就心满意足了。应该怎么安排行程使得总行程最短？



这个题目其实解决方案和第二题很类似，先找出从成都出发经过理塘到达江达的最短距离，再找出从成都出发经过道孚到达江达的最短距离，两相比较取其最短即可。

经过程序实测，应该选择途径道孚的道路，运行结果如下:

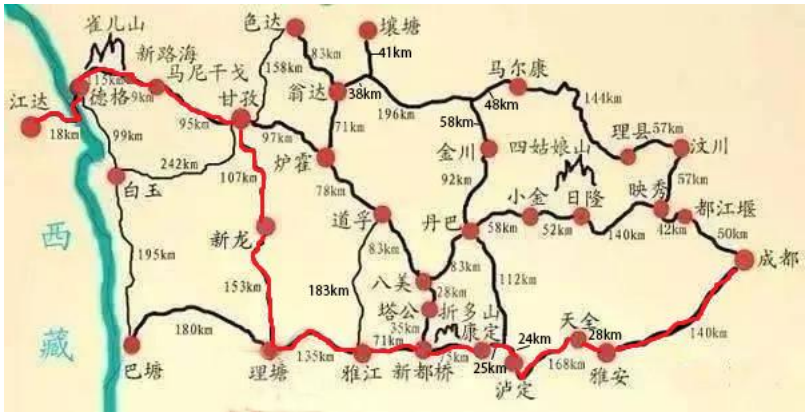
途径 道孚

Path:成都>都江堰>映秀>日隆>小金>丹巴>八美>道孚>炉霍>甘孜>马尼干戈>新路海>德格>江达

Total: 920km

4 张三在规划线路的时候，发现不同路况行驶速度不一样。地图中最粗的路径（成都-雅安）表示平均时速可以达到 80 公里每小时，而中等的路径表示平均时速每小时 60 公里每小时，最细的路径表示平均速度仅仅只有 40 公里每小时。那么(2)(3)中用时最短的路径分别是哪一条？

这个问题其实也很常规，就是在建图的时候，将路程/时速作为每条边的权值，剩下的便交给 Dijkstra 算法即可。有个问题是这里涉及到了浮点数的运算，要注意合适地选择一个“很大的数”去比较，否则就会出现很奇怪的结果。



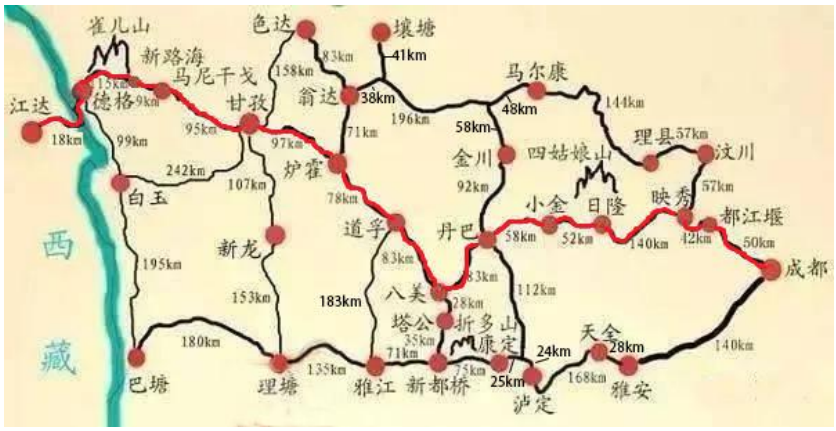
经过运行，(2) 中用时最短的路径结果为:

Path:成都>雅安>天全>泸定>泸定和康定的中间点>康定>新都桥>雅江>理塘>新龙>甘孜>马尼干戈>新路海>德格>江达

Total Time: 20.9667h

Total Dist: 1163km

(3) 中用时最短的路径结果为:



途径 道孚

Path:成都>都江堰>映秀>日隆>小金>丹巴>八美>道孚>炉霍>甘孜>马尼干戈>新路海>德格>江达

Total Time: 15.3333h

Total Dist: 920km

五、总结

对上机实践结果进行分析，问题回答，上机的心得体会及改进意见。

5（思考题）考虑到 Dijkstra 算法仅仅从一段开始寻找路径，效率不高。李教授想到一个高招，就是同时从出发地和目的地进行搜索，扩展搜索节点，然后两个方向扩展的路径会在中途相遇，则拼接起来的路径就是最短路径。如何实现李教授这个想法？

其实这个利用的还是第一题中的 dijkstra 算法，分别从两边开始遍历图，当其中一个遍历过程遍历到另一个路径上时，记录相遇节点并停止遍历，再分别回溯即可拼接成完整的最终路径。

这个思路是简单的，但是实现起来却并不是很容易，我在本次实验中虽然采取的还是常规的方法，但是结合自己在操作系统中的所学知识，还是想谈一些额外的实现方法。为了保证两个 dijkstra 算法的执行流运行的公平性，也为了让它们共享相同的数据——即遍历树，我们可以使用两个线程来并发地执行各自的任务——一个是求解从出发地开始、目的地结束的最短路；另一个求解从目的地开始、出发地结束的最短路，直到这两个线程各自的已知点的集合有所交集，这样就可以停下来回溯、拼接形成完整路径了。利用并发，可以更大程度地利用 CPU 资源，进而也可以提高算法的使用效率。

运行结果:

Path:成都>都江堰>映秀>日隆>小金>丹巴>八美>道孚>炉霍>甘孜>马尼干戈>新路海>德格>江达

Total: 920km

结果其实与第一题完全相同，证明我的实现过程是没有问题的。那么采用新的思路之后，到底在时间上快了多少呢？由于数据集较小，两者的运行时间都很小，所以并不太方便比较。

通过本次实验，我在形象的地图中，对单源最短路径算法有了更深刻的理解。最短路在我们日常生活中有着各种而样的使用场景，希望自己未来能结合实际情况，最大程度地利用好最短路，尤其是为其他算法的运用扫除障碍。