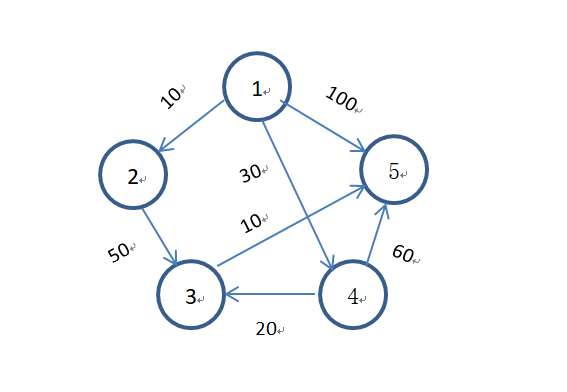
1.问题描述：

以源点1开始，求其到剩下点的最短路径。

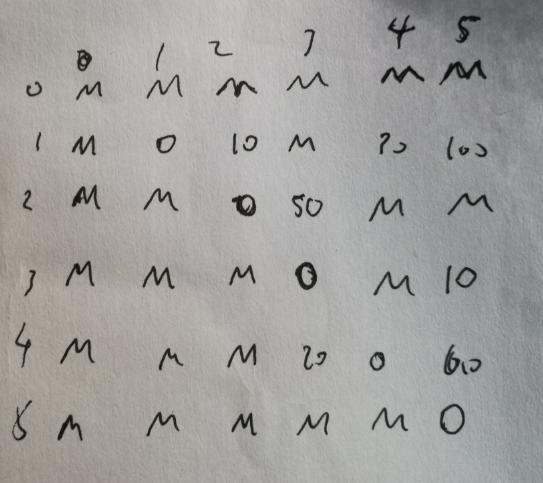
如图：



2.解决方法：

首先用二维数组存放有向带权图，其中max代表没有路径，有值的话代表有路径和其权值。

如图，有向带权图矩阵的矩阵表示：



注：第一行不放数据是为了符合现实，从1开始

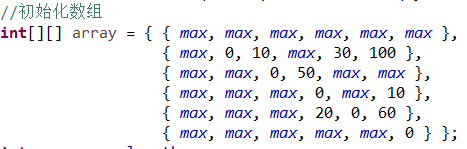
运用贪心法，每一次都求局部最优，最后在局部最优中找到最终结果。以源点开始每次找两点之间的最短路径，通过求和和比较，求出源点到各个点的最短路径。

dist[i]表示当前从源到顶点i的最短特殊路径长度,prev[i]=j：最短路径中顶点i的前一个顶点是j。s[array.length]用来存放找到的最短距离的点放入其中。

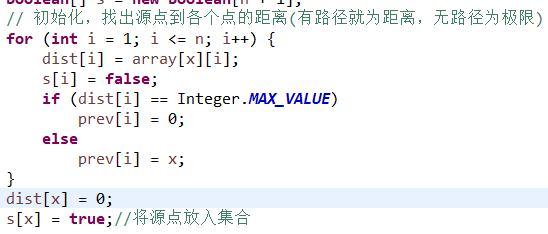
核心：就是通过局部的最短距离来和局部的最短距离比较，找出最终的最短路径。举个大致的例子，例如：1到2，3，4，5中3的距离最短，其他各点都比其大，如果此时找路径1-4，就可以以直接1-4和1-3-4比谁的路径更短,如果后面的较短，就把3放到prev的第4个位置，带表最短路径中4的前一个结点是3……

### 4.2.2 具体实现

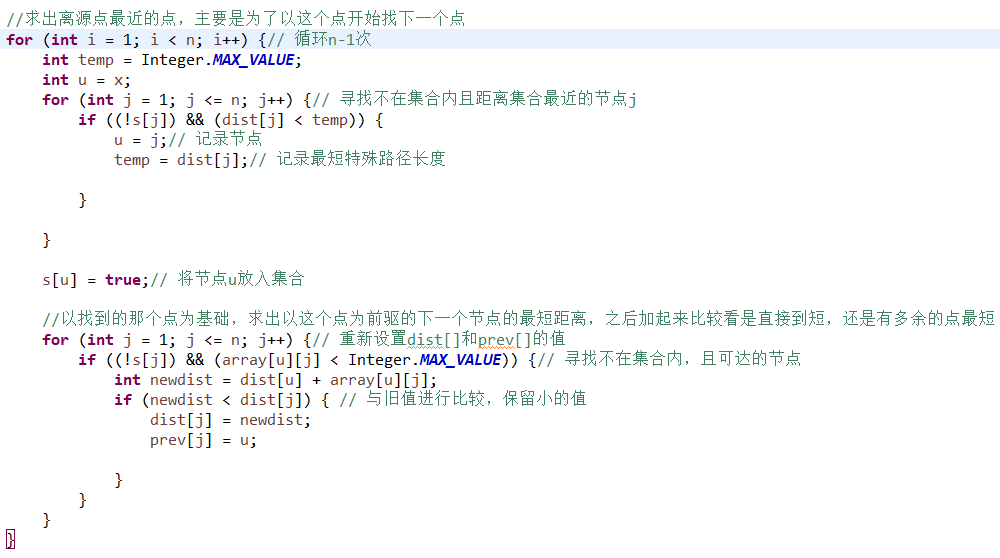
1.初始化数组：



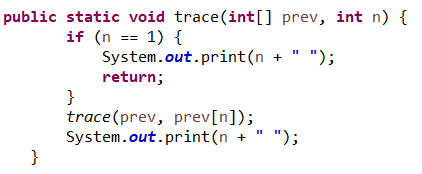
2.初始化源点1，到各个点的距离(有路径就为距离，无路径就为极限max)，同时初始化prev



3. 接下来求出离源点最近的点，在以这个点为起点寻找下一个点，比较原距离及它们的之和的距离，选出最短，并更新prev。

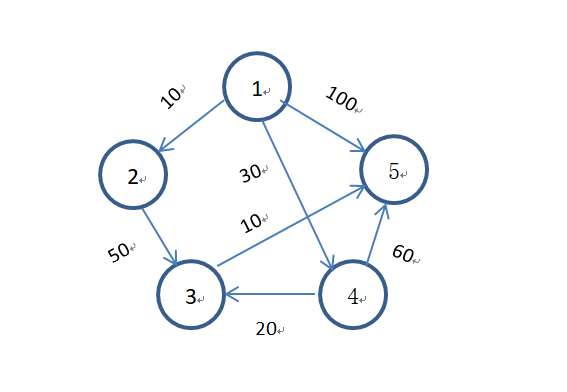


4.在集合prev的基础上递归求出最短路径

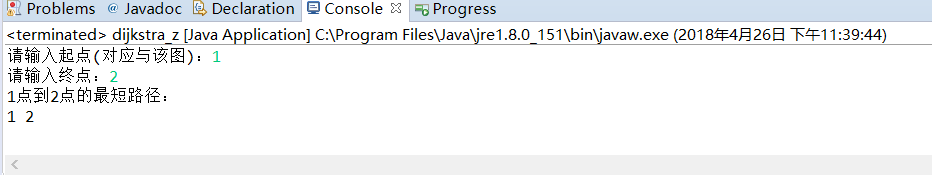


### 4.2.3 结果验证与显示

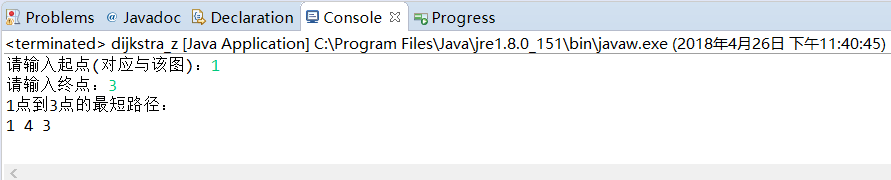
根据图来验证：



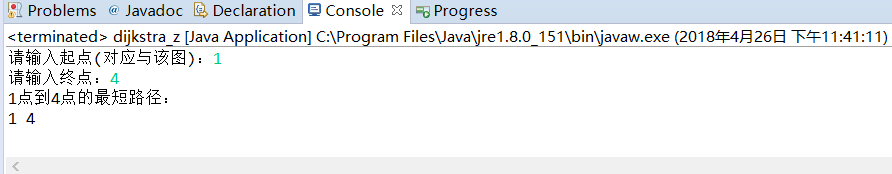
1.输入起点1，输入终点2



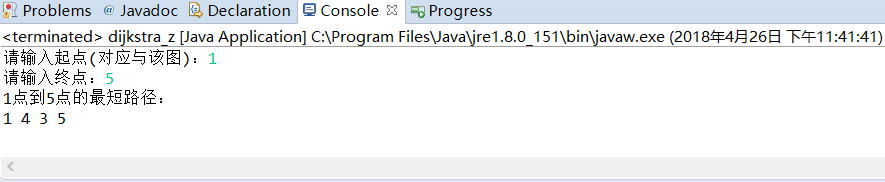
2.输入起点1，输入终点3



3.输入起点1，输入终点4



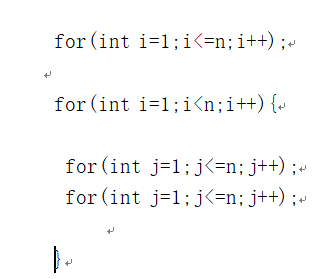
4.输入起点1，输入终点5



由图对比可知，结果正确。

### 4.2.4 算法时间复杂度分析

由于在算法结构中，有这样的一个结构：



所以，可以得到O(N)=N+N(N-1)+N(N-1)=2N^2-N

所以该算法的时间复杂度近似于：O(N^2)