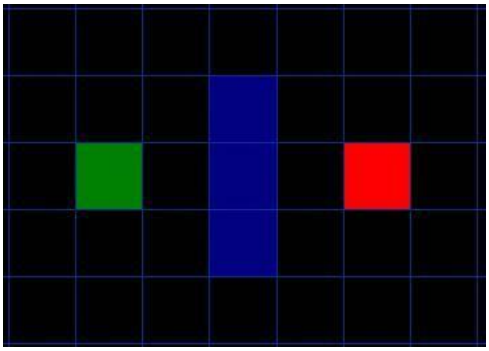
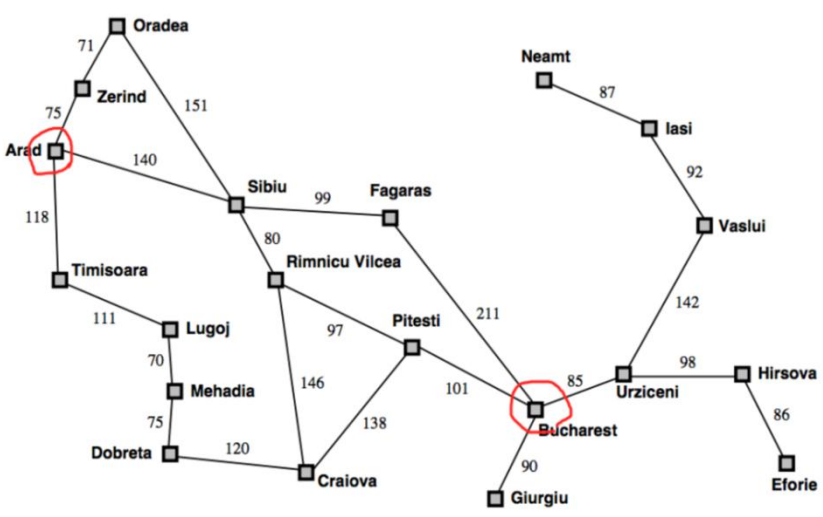
1. 这道题是让我们使用贝叶斯公式来对邮件进行分类，在这里我们需要对两个事件进行处理，即这封邮件是正常的邮件还是垃圾邮件，以及这封邮件中含不含中奖二字，我们要求的是在这封邮件出现"中奖"一词的条件下，这封邮件是垃圾邮件的概率
   在这里我们令 P(A)=正常邮件的概率，那么 P(非 A)=垃圾邮件的概率，P(B)=邮件中出现中奖的概率，那么 P(非 B)=邮件中没出现中奖的概率
   则可得下式
   P(非 A|B)=P(B|非 A)*P(非 A)/P(B)
   对于上面这个式子，如果单求左边可能不太容易，所以我们从右边的三项入手，如果我们把右边的三项全都求出来，那么左边的式子也就迎刃而解了
   在数据集中，我们可以知道一共有多少封邮件，以及哪些邮件是垃圾邮件，这样 P(非 A)就很容易求出来，数据集中还记录了每封邮件中每个词出现的次数，这样的话我们就可以通过对邮件的遍历来确定邮件中出现中奖一词的概率，也就是 P(B),最后要求的是 P(B|非 A)，这个量的意义是在这封邮件是垃圾邮件的条件下出现中奖这个词的概率
   在数值上为(即出现中奖这个词又是垃圾邮件的邮件个数)/(垃圾邮件的个数)
   在这里也可以通过对数据集的遍历求出上面两个量，从而求出 P(B|非 A)，当这三个量全都求出来之后，也就可以求出我们所要求的 P(非 A|B)的答案了
   数据集有两个文件，label 和 train，前者用来标记邮件是不是垃圾邮件，后者用来记录每封邮件出现了什么单词，以及单词出现的次数
   所以分别读取两个文件，第一个文件用来记录邮件的数量(num),垃圾邮件的数量(jab),以及邮件是否为垃圾邮件(为 1 则索引对应的是垃圾邮件)
   第二个文件用来记录每个词在多少封邮件里面出现过
   接下来再次对 train 文件进行遍历
   这次来求每个词出现在多少封垃圾邮件中(count),
   然后就可以求出 P(非 A),P(B)以及 P(B|非 A)的概率(对应下面的 pna,pb,pbna)
   pna=jab/num
   pb=a[i]/num
   pbna=count/jab
   然后就可以求出我们想要的 P(非 A|B)
   Pres1=(pna*pbna)/pb
   然后接下来还要求出带有中奖一词的垃圾邮件占总垃圾邮件的比例 pres2
   这个比较简单，因为我们已经知道带有中奖一词的垃圾邮件数量以及总垃圾邮件的数量，所以这里好办很多
   然后我们最后的结果就是 2/(1/pres1+1/pres2)
2. 这道题是让我们用 A*算法求图中两点之间的最短路径，其实求两点见最短路径的方法很多，图算法比如 Dijkstra,SPFA,Bellman-Ford,Floyd 等等，但是 A*算法相比上面几种，效率更高,因为 A*是一种启发式搜索,也就是说在 A*算法中,会对当前路径进行"估价"，从而实现剪枝的效果，下面看一下 A*算法的核心
   F(n)=H(n)+G(n)
   G(n):从初始位置到当前位置的距离。
   H(n):从当前位置到终点的估计距离。
   其实 A*算法的精髓就在 H(n)上，通过合理的设置 H(n)实现这种"启发式"的效果，在搜索的过程里面剪枝
   比方说在 A*算法里面最经典的一个例子



在这里 H(n)就设置为当前点到目的地的曼哈顿距离
而在我们的问题中



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

右边给出来的直线距离其实是 H(n)的一个好选择，但是这个问题要求程序具有通用性，就是说要能求出来任意两个城市之间的距离，而我们不知道除了 Bucharest 之外其他城市之间的距离，所以在这里我们没法采用这个直线距离作为 H(n)，所以我们在这里还要另想办法

其实我们在搜索的过程中，“当前位置”是在不断变化的，而目的地是不变的，这样我们就可以把问题转化成求我们在路途上经过的所有点到目的地的最短距离，更具体的说，是以目的地为源点的单源最短路问题，这个问题的话我们就有很多种解决的办法，因为在我们当前的问题中不涉及到负环的问题，所以用 Dijkstra 问题就可以解决我们的 H(n)

而 G(n)，我们可以把每个城市存储成一个结点(结构体)，里面存储着这个结点的 G(n)属性，每当到达一个结点的时候，结点的值就等于上一个结点的 G(n)+两个结点的距离

这里还有一个很巧妙的办法，是我参考了网上的资料之后了解到的，用到了 C++里面的优先队列，其实在 A*算法中算出了每个结点所对应的 F(n)之后，在搜索路径的过程中是在一圈路径中找到 F(n)最小的那个，也就是类似于广度优先搜索，但是广度优先搜索有它的局限性，就是不能将有权值的边加入运算，但在 A*里面，由于有了 F(n)，所以我们可以通过对队列中的结点进行排序(这一点可以通过优先队列实现，不过要让队列里面的函数升序排列的话需要自己定义比较函数)，这样可以确定 F(n)小的结点可以先出队，当我们的目标结点出队时，也就得出了最小距离，因为优先队列内的元素是按照从小到大的顺序排序，所以当有目标结点第一次出队的时候，得到的权值肯定是最

短路径