T1:
算法：动态规划。
f[i][j][0]表示当前小象位于格子(i,j)且上一个位置是(i-1,j)所看见的老鼠的最少数量。
f[i][j][1]表示当前小象位于格子(i,j)且上一个位置是(i,j-1)所看见的老鼠的最少数量。
f[i][j][0]=min(f[i-1][j][0]+a[i][j-1],f[i-1][j][1])+a[i+1][j]+a[i][j+1]
f[i][j][1]=min(f[i][j-1][0],f[i][j-1][1]+a[i-1][j])+a[i+1][j]+a[i][j+1]
answer=min(f[n][m][0],f[n][m][1])。
复杂度为 O(NM)。

T2:
算法：最短路。
将“如果城市 B 愿意与城市 A 建立合作关系，当且仅当对于所有满足 d(A,C)<=d(A,B)的城市 C，都有 R(C)<=R(B)。”这一条件转化为“如果城市 B 愿意与城市 A 建立合作关系，当且仅当对于所有满足 R(C)>R(B)的城市 C，都有 d(A,C)>d(A,B)。”
我们倒序枚举 R 的值 r，然后枚举 R(X)=r 的点 X，以每个点为起点对原图做最短路。设 lim[i]表示所有点 K(R(K)>R(X))中到点 i 的最小距离，设 dist[i]表示 X 到 i 的最短路。对于点 i，在最短路过程中，如果 dist[i]>=lim[i]，那么表示比 X 的 R 值大的某个点到点 i 的最短距离要比 X 到 i 的距离短，所以 i 不会与 X 建立合作关系，而且不会用 dist[i]去更新其它点的最短路了。因此，一个点用来更新其它点的条件是 dist[i]<lim[i]，此时答案+1，因为答案<=30N，所以总的更新次数不会超过 30N 次。所以最后复杂度为 O(kNlogN)，k 为常数。

T3:
算法：字符串 hash
枚举两段的长度 len 和第一段的起点 i，我们定义 L 为第一段与第二段的最长公共后缀，当 L>=len 的时候答案+1，而起点为 i+1 时 L 的大小仅仅取决于起点为 i 时 L 大小和 a[i+len]与 a[i+2*len+F]的相等关系：

L[i+1] = L[i] + 1 (a[i+len]=a[i+2*len+F])

L[i+1] = 0 (a[i+len]!=a[i+2*len+F])

这样朴素的枚举 len 后扫描整个序列是 N^2 的，我们考虑优化这个算法。
首先枚举两段的长度 len，然后我们在递推的时候可以发现，在长度为 len 时，我们没有必要一格一格的递推，而可以每次向右递推 len 格。我们不妨设第一段的末尾位置为 i，第二段的末尾位置为 j，设 frontL 表示 a[i+1]…a[i+len]与 a[j+1]…a[j+len]的最长公共前缀，设 backL 表示 a[i+1]…a[i+len]与 a[j+1]…a[j+len]的最长公共后缀，令 L 表示当前的最长公共后缀。
下面分两种情况考虑对于答案的贡献：

情况一：如果 L>=len，ans+=frontL。

情况二：反之，ans+=max(0,L+frontl-i+1)。

下面分两种情况考虑递推后的最长公共后缀 nL:

情况一：如果 a[i+1]…a[i+len]与 a[j+1]…a[j+len]整段相同，nL=L+len。

情况二：反之，nL=backl。

这样对于每个长度 len，需要递推 N/len 次，每次采用 hash+二分的方法 O(logN)的计算最长公共前/后缀，总的复杂度为 O(NlnNlogN)。