小象和老鼠

(lemouse.pas/c/cpp)

解法一：

对于 10%的数据，我们可以采用搜索。

DFS 生成从（1,1）到（N,M）的所有路径，更新答案。

复杂度：C(N-1,N+M-2)。

解法二：

对于 100%的数据，我们可以采用动态规划。

定义：

a[i][j]表示格子(i,j)中老鼠的数量。

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。

我们可以得到转移方程：

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]

最后答案为 min(f[n][m][0],f[n][m][1])。

复杂度：O(N*M)。

题目来源：

改编自 Codechef　June Challenge 2013 <Little Elephant and Mouses>

网络服务

(serves.pas/c/cpp)

解法一：

对于 10%的数据，我们可以采用 floyd 求出每对 d(i,j)，然后枚举城市 B 是否愿意与城市 A 建立合作关系，并枚举所有满足条件的 C 进行判断。

复杂度：O(N^3)。

解法二：

对于 40%的数据，我们可以通过枚举+堆优化 dijkstra 的做法。

首先枚举每个城市 A，然后计算出 A 到达所有点的最短路。然后将所有点按照 dist 为第一关键字升序，Ri 为第二关键字降序排序。扫描一遍即可得到愿意与 A 建立友好关系的点。

复杂度：O(N(M+N)logN)。

解法三：

对于 100%的数据，我们将问题转化之后，仍然可以通过枚举+堆优化 dijkstra 的做法解决。

“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)。”我们将这一条件转化“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)。”

定义 lim[i]表示 R 值比点 i 大的所有点中到点 i 的最小距离。

注意到问题是建立在无向图上的，所以有 d(i,j)=d(j,i);

问题经过转化后，我们按照 Ri 降序枚举每个点作为源点。在 dijkstra 算法过程中，假设当前枚举的点为 B，设 dist[A]为点 B 到 A 的最短路，我们发现如果点 A 需要被更新的话，当且仅当 lim[A]>dist[A]，**注意到 lim[A]的定义是所有 R 值比 A 大的点中到点 A 的最小距离**，于是我们可以得到，**lim[A]>dist[A]等价于“对于所有满足 R(C)>R(B)的城市 C，都有 d(A,C)>d(A,B)。”**所以一个点 A 入堆的条件是当且仅当它愿意与源点 B 建立合作关系，也就是对于答案的贡献为 1。

所以我们按照 R 值降序枚举源点 B，修改 dijkstra 松弛的条件，将 dist 与 lim 比较即可。

复杂度：O(ans*logN)。

题目来源：

SWERC 2002 Serves(POJ 1206)

秘密武器

(weapon.pas/c/cpp)

解法一：

对于 10%的数据，我们可以采用枚举的方法。枚举第一段起点 i 与第二段起点 j 然后 O(N)判断是否可行。

复杂度：O(N^3)。

解法二：

对于 30%的数据，我们可以采用枚举的方法。枚举第一段起点 i 与第二段起点 j 然后二分+hash O(logN)的判断是否可行。

复杂度：O(N^2logN)。

解法三：

对于 60%的数据，我们可以采用枚举的方法。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，我们发现答案加一，当且仅当 L>=len 的时候，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])

枚举 len 之后我们可以 O(N)的递推得到答案。

复杂度：O(N^2)。

解法四：

对于 100%的数据，我们可以考虑优化 60%的数据做法。

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：如果 L<len,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。

复杂度分析，对于每一个长度 len 需要递推$\frac{N}{len}$次，求公共前（后）缀的复杂度 O（logN）。

$$\log N * \sum_{i=1}^{N}\frac{N}{i} = N*\ln N*\log N$$

复杂度：O(N*lnN*logN)。

题目来源：

改编自 BZOJ 2119。