NOIP2010 题解

T1 机器翻译

题目大意：按顺序给定一篇带 n 个单词的文章，指定内存容量为 m.内存初始为空，程序可以从内存或者词典调用一个单词的释义。如果当前内存里没有该单词的释义，那么需要到词典里去查找，并将找到的单词释义放到内存里。如果内存已满，程序将会腾出最先放在内存里的单词，以给出空间给新放入的单词。问程序要调用多少次词典。

题解：【模拟/队列】

题目的意思就是模拟一个 FIFO 队列，然后问你总共插入了多少次队列。

由于题面描述中，单词就是一个不大于 1000 的正整数，所以连 Hash 都不要，直接用个 bool 数组看看这个单词在不在队列里面。如果在的话就继续，不在的话就入队并计数+1，如果队列已满就从队尾取出单词。

T2 关押罪犯

题目大意：给定 n 个罪犯和 m 对关系。关系(aj,bj,cj)代表罪犯 aj,bj 之间有一个怨气值 ci，你现在要把 n 个罪犯分到两个监狱。如果 aj,bj 在同一个监狱，那么就会发生冲突。监狱长表现值是所有发生冲突的关系的怨气值之和，你需要使这个表现值最小。求这个最小的表现值。

题解：【贪心/并查集】

我们观察一下题目。首先如果有三个罪犯 A,B,C，我们能确定 A,B 不在一个监狱，而 B,C 不在一个监狱，那么 A,C 肯定在一个监狱。如果 A,B 在一个监狱，B,C 也在一个监狱，那么 A,B,C 肯定也在一个监狱。

发现没有？是不是很像“我敌人的敌人是我的朋友，我朋友的朋友也是我的朋友”？有没有想起《团伙》这道题？所以我们就可以借《团伙》这道题目的方法来判断两个人是否存在冲突。就是建一个 ai+n 的虚点，代表 ai 的敌人（不在同一座监狱）。就可以使用并查集来判断 ai 和 bi 到底是不是朋友（在同一座监狱）了。如果是朋友，就会爆发冲突。

然后按照贪心思想，我们每次优先选择怨气值最大的一对关系，分配到两个不同的监狱中，来使怨气值总和最小。如果发现这对关系已经发生冲突，则答案加上这个怨气值。

这个贪心显然是正确的，有点像 Kruskal 的贪心思想。

这样的话，这道题目就顺利解决了。

T3 乌龟棋

题目大意：给定一个长度为 n 的序列 a1...an，序列的每个元素都有一个权值。给定 m 张卡片，卡片上只有一个且只可能有 1,2,3,4 四种数字。乌龟在起点 a1,每次可以选择一张卡片，向前进的步数就是卡片上的数字。乌龟最终到达终点 an 后，它的分数就是它到达过的序列上的元素的权值之和（显然乌龟出发点在 a1,因此它的分数一定包含 a1）。你需要求出乌龟的分数最大值，数据保证到达终点后m张牌全部使用。

数据范围：每种卡片不超过 40 张，序列不超过 350 个。

题解：【多维 dp】

首先观察题目。Emmm 卡片上只可能有四种数字？非常可疑的条件哦。

然后联想到 dp，秒懂。

影响 dp 的因素：现在所处的位置，四种卡片的剩余数量。所以这个 dp 大概就是五维的样子。

Dp 方程：dp[at][i][j][k][o]=max(dp[at+1][i+1][j][k][o]+
Dp[at+2][i][j+1][k][o]+
Dp[at+3][i][j][k+1][o]+
Dp[at+4][i][j][k][o+1])+a[at]

状态：当前处于 a[at]，已经使用了 1 号卡片 i 张，2 号卡片 j 张，3 号卡片 k 张，4 号卡片 o 张。

但五维的话就会出现这样的现象：350*40^4=8.96 亿，会爆空间。

考虑降维，我一开始傻傻的想到降个第五维，然而错了好一会儿才看见第一维可以换掉：At=1+i+j*2+k*3+o*4.因为 i,j,k,o 确定后，at 也已经确定了，因此现在所处的位置（at）不成为影响 dp 的因素，所以该维可以舍去。

之后就可以求出答案了。

答案：dp[0][0][0][0].

T4 引水入城

题目大意：

给定一个 n*m 的矩阵。第一行是靠河的一行，最后一行是靠沙漠的一行，也就是干旱区。每个格子都是一个城市。靠河的城市可以建立抽水站，水可以沿着城市海拔高度严格递减的顺序传递，直到传到干旱区的城市。

现在问是否可以有一种建抽水站的方案使得干旱区的城市全部有水，如果可以，至少要建立几个抽水站？如果不可以，有几个城市不可能有水？

题解：【搜索/贪心】

这题我想了一晚上，最后发现我离正解就差两步，所以 emmm。

本题也是 2010 提高组最难的题。

首先这题估计 dp 没法做，所以先考虑搜索。我们从干旱区往河边搜，然而这个方案通过举反例可知并不容易实现。

那么我们从河边往干旱区搜。就在这时我发现可以从河边的一个城市往干旱区搜，会得到一个这个城市对干旱区水的能覆盖到的区间。通过搜索河边的每个城市，可以得到一些区间，于是该题转换为贪心中的区间覆盖问题。

然而问题来了：区间可能并不连续。时间复杂度最坏可能达到 500^3，比较危险。

区间可能并不连续这个问题困扰了我很久。下面我们来讨论这两个问题。

影响的区间一定连续。原因是区间如果不连续，便是中间有不可能到达的城市阻断了这个区间。这样的话区间阻断转换成了无解一问，我们就可以接着处理有解一问的答案。

对于时间复杂度的优化，**如果第一行你准备出发的起点左右有比它海拔高的城市，那么这个城市不应该作为起点搜索**，因为它的区间肯定被左右两个城市的区间所包含。

保证了有解一问覆盖的区间肯定连续，题目便转换成了区间覆盖问题。这里简单介绍一下区间覆盖问题：

有 m 段区间[li,ri]，选择最少的段数覆盖区间[s,t]。

贪心策略：

1.将 s 之前的部分全部切除，因为无意义

2.将区间按 li 从小到大排序

3.如果 l1>s 则无解

4.选择 li==s 且 ri 最大的一个区间

5.S 更改为 ri。

6.返回第 1 步，直到 s>=t。

易知，这个贪心策略是肯定正确的。

然后这道题就这样解决了。至于搜索，建议使用 BFS 吧。

贪心的实现可能还要稍微考虑一下，而实际上只要用最暴力的方法实现就好了，这样最不容易出错……