Contest 5 Solution

By ACE

Overview

- 2001 Haha-Party [字符串匹配] *
- 2002 Saving Lelouch [贪心] ***
- 2003 Hotel [线段树] ***
- 2004 Traffic Light [枚举] *
- 2005 Unexclusion [矩阵快速幂] *****
- 2006 Dot Dot Dot [BFS] **

2001 Haha-Party

- 需要使用一个低于平方级别的字符串匹配算法，可以使用 KMP 。
- 用 string 类的 rfind() 函数可以线性地找到目标串在文本串中最后一次出现的起始位置。
- 由于目标串是给定的，而且长度小，可以枚举。

2002 Saving Lelouch

- 有N个敌人，自己与敌人都用攻击力与血量来描述。敌人一个一个和你打。
- 每一个回合，双方先给对面造成自己攻击力的血量伤害。然后，如果有人血量小于等于0，机体坠毁，战斗结束。否则进入下一回合。
- 无论对方血量多少，自己血量小于等于0就算输。
- 可以使用GEASS暂停时间，使得对面一回合无法攻击。每次消耗1点体力。体力有上限，不能小于0.每击败一个敌人会回复若干体力。

贪心

- GEASS应该尽量用在攻击力高的敌人身上。
- 先不使用GEASS一路往后，直到需要决策的时候，再贪心使用。
- 关键在于何时进行决策：
- 1、不使用就会死。
- 2、不使用就浪费（打完敌人后回复的体力值会溢出）

- 两种情况，都是去选取可以使用的情况中，抵挡攻击力最高的。用优先队列维护。

剩余体力值

- 关键在于，哪些情况是当前可用的。
- 情况的可用性，取决与：
- 1、理论可使用次数：由与敌人攻击的回合数决定（减去以用次数），容易解决。
- 2、受体力值制约的上限，无法即时更新。

- 使用线段树存储当前体力情况。

线段树

- 当遇到新的敌人i时，先将当前体力值存入线段树的节点i处。
- 然后判断是否需要考虑使用GEASS。（判断要先后进行，即先判断会不会死，再处理会不会溢出）
- 如果要使用，当从堆中得到攻击力最大的敌人j时（即希望在打j时使用GEASS），从线段树中得到区间[j,i]的最小值，来更新j的GEASS可以使用次数。
- 若打算在j处使用k次GEASS，那么对区间[j,i]删去k。

正确性

- 这是对于体力值情况的模拟。
- 在这里没有考虑之前的体力回复，因为所有体力回复都在状态2（不使用就浪费）中得到了处理，使得所有原本的过剩回复都会被合理使用或者由于实在无法使用（没有可以使用GEASS的状态了）而闲置。
- 前者不再有多余的回复值，后者不会在之前出现可用情况。

复杂度

- 每个敌人最多一次、一次出队，因此查询的总数为O(N+M)（N个敌人出队的次数，M表示决策次数，因为M<=N)，所以查询次数为O(N)的。
- 每次查询需要log(N)的时间得到最大值，再用log（N）的时间查询、维护线段树，复杂度为O(N·log(N))

2003 Hotel

- 你是一个酒店职员，有很多领导要到你们酒店入住。
- (1)每一个领导会告诉你他想住在哪一些候选楼层，但一个领导只需要一层楼的一个房间即可

  (2)你必须为每一个领导都安排他想要的某一层的一个房间，但多个领导可住在一层（假设每一层有无数个房间）。

  (3）酒店会将所有有领导居住的楼层之间的楼层清空，因此费用为 占用楼层数 = 最高楼层号-最低楼层号+1

- (4) 旅店中可被选择的楼层一直在变化。每次计算当前最低费用时老板会给出当前所能居住的最低楼层
- (5)每次计算当前最低费用时，费用是指在满足酒店的要求的前提下把开始到当前的所有批次的客人
- (6)若不能完成要求，输出-1

- F[i]为以i为最低楼层，当前安排所有客人的最低费用。（i必选）令G[i]=F[i]+i，为以i为最低楼层以最低费用安排所有批次客人所用到的最高楼层。
- 来看一个例子
- 初始状态

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
G	1	2	3	4	5	6	7	8	9	10	11	12	13	14
F	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- 第一批客人想住3 8 14三层楼之一

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
G	3	3	3	8	8	8	8	8	14	14	14	14	14	14
F	2	1	0	4	3	2	1	0	5	4	3	2	1	0

- 第二批客人想住2 6 9 13四层楼之一

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
G	3	3	6	8	8	8	9	9	14	14	14	14	14	无解
F	2	1	3	4	3	2	2	1	5	4	3	2	1	无解

- 不难看出，对于每批客人，将其候选楼层升序排序，设序列为A_i。Ai将所有楼层分为几份:$1\sim A_1, A_1+1\sim A_2, \ldots, A_{n-1}+1\sim A_n$。对于以1~A1为最低层数的方案，必然选择将这批客人安排在A_1，以$A_i+1\sim A_{i+1}$为最低层数的方案，必然选择将这批客人安排在A_{i+1}。

- 所以，每一个候选楼层A_i对G的影响是连续的一个区间 $[L, R] \subset [A_{i-1} + 1, A_i]$
- 其中G[L]~G[R]<A_i
- 由G的更新方式不难看出G是单调的，所以该区间的右边界可以二分求出，左边界即为A_{i-1}+1，该区间可为空。

- 这样我们就可以先对数据离散后，再用一个线段树来维护区间的G值，以及该区间中最小的F[i]。每次查询的时候，查询区间[Fi,max]内最小的F[i]。(max为最高楼层号)

- 时间复杂度O((sigma{Ti}+Q)logN)
- N为不同的楼层总数。
- sigma{Ti}为所有C操作中提到的楼层总数。
- Q 为总询问数。

2004 Traffic Light

- 用7段LED数码管表示数字，其中可能有一些LED灯损坏，无法发光。现在给出某N个时刻的显示情况，求出一组合法解，包括LED灯损坏情况，以及N个时刻所显示的数字。

- 结果用SPJ检验。

- 签名题，搜索损坏情况，然后枚举每个时刻的显示情况可能是什么数，判断是否合法。

- 每个显示情况的步骤：
- 1、所有坏掉的灯是否都是暗的。
- 2、枚举可能显示的数字，当前显示中亮的灯应该在显示该数字时都是应该亮的。
- 3、查看其余显示该数字时应该是亮的，而现在没亮的灯，看是否都是坏掉的灯。

表示状态

- 用二进制表示各种状态：灯的亮暗（1亮0暗）、好坏（1坏0暗）。
- 如果当前显示的状态为i，枚举的数字应该显示的状态为j，那么i & j==i就可以判断是否当前显示为亮的灯都是该亮的。
- 在i & j==i的基础上，i ^ j就是应该亮但是没亮的灯的状态。
- 如果灯好坏的状态为j，((i ^ j) & j)==j就可以判断是否那些灯都是坏的。

2005 Unexclusion

- 题目大意：

- 观察题目，(p,q) = 1。

- 比如p=5，q=6，我们可以列出这样的一张表格。这个表格第一行第一列为1，然后每向右走一格就+p(mod)，向下走一格就+q（mod)于是我们可以得到下面这样一张表格。

1	6	11	16	21	26
7	12	17	22	27	2
13	18	23	28	3	8
19	24	29	4	9	14
25	30	5	10	15	20

- 我们可以发现每个元素上下左右邻接的格子恰好是与这个元素排斥的位置，
也就是距离为p（左右）或者q（上下）的位置。

- 因为p和q互质，所以一定可以把所有元素不重复地填进p*q的矩阵中。（证明）

- 到这里，题目就转换成给一个p*q的矩阵，从中选出若干个元素，使得元素互不相邻。矩阵是上下相连，左右相接的。

- 于是我们可以状态压缩求解。DP[i][j]表示第i行的状态为j

$$DP[i][j] = \sum DP[i-1][k]$$

- k需要和j满足上下没有邻接的条件

- 因为q很大，p很小（2<=p<=10)，所以我们可以利用矩阵快速幂来加速求解。

- 当p = 10时，合法的状态只有123种，totstate = 123。最后总的时间复杂度就是O(totstate^3 * log(q))。

2006 Dot Dot Dot

- 一个格子图，有一个起点一个终点，要用最小步数从起点走到终点
- 图中有黄色或绿色的墙，被销毁之前墙无法通过
- 有黄色或绿色的开关（不超过5个），触发开关后会销毁同一连通块中所有同色的墙
- 每个开关只能用一次

分析

- 由于每个开关只能用一次，而触发后只能销毁自己所在的连通块中的墙，所以触发顺序不同，结果不同
- 在做最短路时，除了记录x和y坐标，还要记录已经触发了那些开关
- 注意：要记录顺序，mask是不行的！

解法

- 为了转移状态时不用每次触发开关都做floodfill，先进行预处理
- 枚举开关的触碰顺序（可能有某些顺序不合法），对于每一个顺序看能消掉哪些墙
- 记录下在这个触碰顺序下，哪些格子可达
- 做最短路的时候看能否转移：当前的触碰状态是否在目标格子的可达状态表中