本章主要考察优化算法,图论建模和模型转化(出现博弈问题)

Riding the Fences 骑马修栅栏

译 by Jeru

描述

Farmer John 每年有很多栅栏要修理。他总是骑着马穿过每一个栅栏并修复它破损的地方。

John 是一个与其他农民一样懒的人。他讨厌骑马，因此从来不两次经过一个栅栏。你必须编一个程序，读入栅栏网络的描述，并计算出一条修栅栏的路径，使每个栅栏都恰好被经过一次。John 能从任何一个顶点(即两个栅栏的交点)开始骑马，在任意一个顶点结束。

每一个栅栏连接两个顶点，顶点用 1 到 500 标号(虽然有的农场并没有 500 个顶点)。一个顶点上可连接任意多(>=1)个栅栏。所有栅栏都是连通的(也就是你可以从任意一个栅栏到达另外的所有栅栏)。

你的程序必须输出骑马的路径(用路上依次经过的顶点号码表示)。我们如果把输出的路径看成是一个 500 进制的数，那么当存在多组解的情况下，输出 500 进制表示法中最小的一个 (也就是输出第一个数较小的，如果还有多组解，输出第二个数较小的，等等)。

输入数据保证至少有一个解。

格式

PROGRAM NAME: fence

INPUT FORMAT：

(fence.in)

第 1 行: 一个整数 F(1 <= F <= 1024)，表示栅栏的数目

第 2 到 F+1 行: 每行两个整数 i, j(1 <= i,j <= 500)表示这条栅栏连接 i 与 j 号顶点。

OUTPUT FORMAT：

(fence.out)

输出应当有 F+1 行，每行一个整数，依次表示路径经过的顶点号。注意数据可能有多组解，但是只有上面题目要求的那一组解是认为正确的。

SAMPLE INPUT

```
9
1 2
2 3
3 4
4 2
4 5
2 5
5 6
5 7
4 6
```

SAMPLE OUTPUT

```
1
2
3
4
2
5
4
6
```

```
5
7
```

分析

这道题是要求我们求出一条欧拉路，所以我们要首先判断图中是否有欧拉路。对于一个无向图，如果它每个点的度都是偶数，那么它存在一条欧拉回路；如果有且仅有 2 个点的度为奇数，那么它存在一条欧拉路；如果超过 2 个点的度为奇数，那么它就不存在欧拉路了。

由于题目中说数据保证至少有 1 个解，所以一定存在欧拉路了。但是我们还要选一个点作为起点。如果没有点的度为奇数，那么任何一个点都能做起点。如果有 2 个奇点，那么就只能也这两个点之一为起点，另一个为终点。但是我们要注意，题目要求我们输出的是进行进制转换之后最小的(也就是输出第一个数较小的，如果还有多组解，输出第二个数较小的，等等），所以我们要以最小的点做起点。

找出欧拉路的方法就是采用深搜的方式，对于当前的点，把所有点从小到大的搜索，找到和它相连的，找到一个之后删除它们之间的连线，并去搜索新的那个点，如果没有找到点和它相连，那么就把这个点加入输出队列。

不过我们这么操作之后，顺序是反着的，输出时反着输出即可。

Shopping Offers 商店购物

IOI'95

译 by Felicia Crazy

描述

在商店中，每一种商品都有一个价格（用整数表示）。例如,一朵花的价格是 2 zorkmids （z），而一个花瓶的价格是 5z 。为了吸引更多的顾客，商店举行了促销活动。

促销活动把一个或多个商品组合起来降价销售，例如：

三朵花的价格是 5z 而不是 6z， 两个花瓶和一朵花的价格是 10z 而不是 12z。 编写一个程序，计算顾客购买一定商品的花费，尽量利用优惠使花费最少。尽管有时候添加其他商品可以获得更少的花费，但是你不能这么做。

对于上面的商品信息，购买三朵花和两个花瓶的最少花费是：以优惠价购买两个花瓶和一朵花（10z），以原价购买两朵花（4z）。

格式

PROGRAM NAME: shopping

INPUT FORMAT：

（file shopping.in)

输入文件包括一些商店提供的优惠信息，接着是购物清单。

第一行 优惠商品的种类数（0 <= s <= 99）。

第二行..第 s+1 行 每一行都用几个整数来表示一种优惠方式。第一个整数 n （1 <= n <= 5），表示这种优惠方式由 n 种商品组成。后面 n 对整数 c 和 k 表示 k （1 <= k <= 5）个编号为 c （1 <= c <= 999）的商品共同构成这种优惠，最后的整数 p 表示这种优惠的优惠价（1 <= p <= 9999）。优惠价总是比原价低。

第 s+2 行 这一行有一个整数 b （0 <= b <= 5），表示需要购买 b 种不同的商品。

第 s+3 行..第 s+b+2 行 这 b 行中的每一行包括三个整数：c ，k ，和 p 。C 表示唯一的商品编号（1 <= c <= 999），k 表示需要购买的 c 商品的数

量（1 <= k <= 5）。p 表示 c 商品的原价（1 <= p <= 999）。最多购买 5*5=25 个商品。

OUTPUT FORMAT：

(file shopping.out)

只有一行，输出一个整数：购买这些物品的最低价格。

SAMPLE INPUT

```
2
1 7 3 5
2 7 1 8 2 10
2
7 3 2
8 2 5
```

SAMPLE OUTPUT

```
14
```

分析

0 <= b <= 5,1 <= k <= 5,可用 5*5*5*5*5 的 DP 每种买 0~5 个，可以用 6 进制表示，然后 5 维 DP~OK!

状态设置：F[a1][a2][a3][a4][a5]为买 a1 件物品 1,a2 件物品 2,a3 件物品 3,a4 件物品 4,a5 件物品 5 时，所需的最少价格

边界条件：F[0][0][0][0][0]=0;

状态转移方程：

F[a1][a2][a3][a4][a5]=min{F[a1-P[i][1]][a2-P[i][2]][a3-P[i][3]][a4-P[i][4]][a5-P[i][5]]+P[i][0]}

其中 i=1..s+b; 且 ak-p[i][k]>=0

官方

官方解法为最短路把每种状态[a1][a2][a3][a4][a5]（a1 件物品 1,a2 件物品 2,a3 件物品 3,a4 件物品 4,a5 件物品 5）看成一个点，则至多 7776 个点，而每个优惠就是一条边，则至多 105 条边。接下来就是求[0,0,0,0,0]到目标状态的最短路，用 Dijkstra(Heap 优化)即可。

感觉 USACO 官方很牛。。。

Camelot 亚瑟王的宫殿

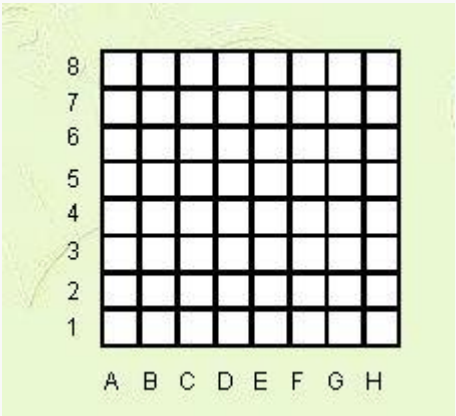
IOI 98

译 by leontea

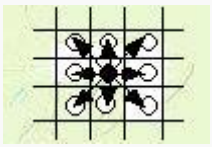
描述

很久以前，亚瑟王和他的骑士习惯每年元旦去庆祝他们的友谊。为了纪念上述事件，我们把这些是看作是一个有一人玩的棋盘游戏。有一个国王和若干个骑士被放置在一个由许多方格组成的棋盘上，没有两个骑士在同一个方格内。

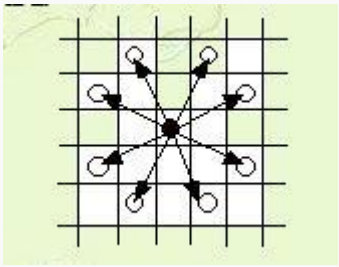
这个例子是标准的 8*8 棋盘



国王可以移动到任何一个相邻的方格，从下图中黑子位置到下图中白子位置前提是他不掉出棋盘之外。



一个骑士可以从下图中黑子位置移动到下图中白子位置(走“日”字形） 但前提是他不掉出棋盘之外。



在游戏中，玩家可在每个方格上放不止一个棋子，假定方格足够大，任何棋子都不会阻碍到其他棋子正常行动。

玩家的任务就是把所有的棋子移动到同一个方格里——用最小的步数。为了完成这个任务，他必须按照上面所说的规则去移动棋子。另外，玩家可以选择一个骑士跟国王从他们两个相遇的那个点开始一起行动，这时他们按照骑士的行动规则行动，其他的单独骑士则自己一直走到集中点。骑士和国王一起走的时候，只算一个人走的步数。

写一个程序去计算他们集中在一起的最小步数，而且玩家必须自己找出这个集中点。当然，这些棋子可以在棋盘的任何地方集合。

格式

PROGRAM NAME: camelot

INPUT FORMAT

(file camelot.in)

第一行： 两个用空格隔开的整数：R,C 分别为棋盘行和列的长。不超过 26 列，30 行。

第二行..结尾： 输入文件包含了一些有空格隔开的字母/数字对，一行有一个或以上。第一对为国王的位置，接下来是骑士的位置。可能没有骑士，也可能整个棋盘都是骑士。行从 1 开始，列从大写字母 A 开始。

OUTPUT FORMAT

(file camelot.out)

单独一行表示棋子集中在一个方格的最小步数。

SAMPLE INPUT

```
8 8
D 4
A 3 A 8
H 1 H 8
```

国王位置在 D4。一共有四个骑士，位置分别是 A3,A8,H1 和 H8。

SAMPLE OUTPUT

```
10
```

SAMPLE OUTPUT ELABORATION

```
他们集中在 B5。
骑士 1: A3 - B5 (1 步)
骑士 2: A8 - C7 - B5 (2 步)
骑士 3: H1 - G3 - F5 - D4 (此时国王开始与这个骑士一起走) - B5 (4 步)
骑士 4: H8 - F7 - D6 - B5 (3 步)
1 + 2 + 4 + 3 = 10 步
```

分析

本题目的主要矛盾是:目标位置不定,国王是否要与骑士同行,并且汇合点不定.与国王汇合的骑士也不定= =|||.而且行进过程中没有特定的规律可循.这就意味着必须要枚举了.因为这是最短路径问题.为了降低时间复杂度需要先将骑士的可到达的最短路与国王的最短路构造出来.骑士是 4 维的记做:chess[x][y][i][j],表示坐标在(x,y)的骑士到达(i,j)的最短路径.当无法到达时 chess[x][y][i][j]=-1;当(x,y)=(i,j)时 chess[x][y][i][j]=0;国王是 2 维的与骑士表述类似.

首先使用 BFS 计算出骑士的最短路程.然后根据公式 MAX(abs(KINGx-i),abs(KINGy-j))计算国王的最短距离.含义是国王到那个点的最短距离是横坐标距离差与纵坐标距离差的最大值.至于为什么是这样.画图看看即可.

最后就是最重要的计算了.因为要枚举最终位置,汇合点和骑士,直接枚举时间复杂度为(26*30)*(26*30)*(26*30)>4x10^8 严重超时,(况且途中还要计算步数总合).之所以这样是因为我们做了很多重复操作.(我开始是这样做的,后来自己测了一个 20*20 的就超了= =).我们只要提前构造好要反复利用的就可以将时间减少几个数量级.(^o^)

先用 Dis[x][y]表示所有骑士(不包括国王)到(x,y)的总步数.把这个要重复利用的提前计算出来.然后枚举会合点.我们只要枚举会合点就行了.不用再枚举到达目标点的距离.因此时间复杂度(26*30)*(26*30+26*30)是百万级的完全没问题.也就是需要一个结构型的数据结构 kdis[len].(len 是总的会合点的个数)他有以下属性:val,x,y,hx,hy.val 是国王与坐标在(x,y)的骑士汇合于(hx,hy)的步数.而且会和于(hx,hy)一定要贪心的找步数最少的那个骑士(x,y).不然就失去了提前构造的意义.到此为止构造完毕.

最终的最短距离用如下公式计算,需要枚举终点:假设终点为(x,y).**MIN(dis[x][y]+kdis[len'].val+当前骑士从当前会合点到(x,y)的距离-当前骑士到(x,y)本身的距离)**;这是根据方程和已构造的值得意义得道的.

Home on the Range 家的范围

译 by tim green

描述

农民约翰在一片边长是 N (2 <= N <= 250)英里的正方形牧场上放牧他的奶牛。

(因为一些原因，他的奶牛只在正方形的牧场上吃草。)遗憾的是,他的奶牛已

经毁坏一些土地。(一些 1 平方英里的正方形)

农民约翰需要统计那些可以放牧奶牛的正方形牧场(至少是 2x2 的,在这些较大的正方形中没有一个点是被破坏的，也就是说，所有的点都是“1”)。

你的工作要在被供应的数据组里面统计所有不同的正方形放牧区域(>=2x2)的个数。当然，放牧区域可能是重叠。

格式

PROGRAM NAME: range

INPUT FORMAT:

(file range.in)

第 1 行:N,牧区的边长。

第 2 到 n+1 行:N 个没有空格分开的字符。

0 表示 "那一个区段被毁坏了";1 表示 " 准备好被吃"。

OUTPUT FORMAT:

(file range.out)

输出那些存在的正方形的边长和个数，一种一行。

SAMPLE INPUT

```
6
101111
001111
111111
001111
101101
111001
```

SAMPLE OUTPUT

```
2 10
```

```
3 4
4 1
```

分析

1.这道题可以动态规划。二维的动态规划。

状态定义：a[i][j]为以(i,j)为左上角顶点的正方形的最大边长。

边界条件：a[i][j]为初始读入的矩阵。

状态转移方程： a[i][j]=min{ a[i+1][j] , a[i][j+1] , a[i+1][j+1] } + 1;

解析： a[i+1][j] , a[i][j+1] , a[i+1][j+1]分别为(i,j)向下、向右、向右

下一格的状况。 在(n-1,n-1)当且仅当三者都为 1 的时候，正方形才能扩充。从最右

下向上，依次扩充即可。

2.可以枚举.但需要优化,用 sum[i][j]表示从(1,1)到达(i,j)的 1 的个数,那么当前区域可以形成正方形的充要条件就是当前的区间的 1 的个数等于边长 lenˆ2.而 sum 已经构造出来了.所以只需要简单的枚举计算就行了.即满足
sum[i+len-1][j+len-1]+sum[i-1][j-1]-sum[i+len-1][j-1]-sum[i-1][j+len-1]=lenˆ2
就行.

A Game 游戏

IOI'96 - Day 1

译 by 逍遥

描述

有如下一个双人游戏:N(2 <= N <= 100)个正整数的序列放在一个游戏平台上，两人轮流从序列的两端取数，取数后该数字被去掉并累加到本玩家的得分中，当数取尽时，游戏结束。以最终得分多者为胜。

编一个执行最优策略的程序，最优策略就是使自己能得到在当前情况下最大的可能的总分的策略。你的程序要始终为两位玩家执行最优策略。

格式

PROGRAM NAME: game1

INPUT FORMAT:

(file game1.in)

第一行: 正整数 N, 表示序列中正整数的个数。

第二行至末尾: 用空格分隔的 N 个正整数（大小为 1-200）。

OUTPUT FORMAT:

(file game1.out)

只有一行，用空格分隔的两个整数: 依次为玩家一和玩家二最终的得分。

SAMPLE INPUT

```
6
4 7 2 9
5 2
```

SAMPLE OUTPUT

```
18 11
```

分析 1

博弈问题，可以用动态规划解决

设 sum[i][j]表示从 i 到 j 的所有数字和，dp[i][j]表示从 i 到 j 这部分的先取者能获得的最大数字和

dp[i][j]=sum[i][j]-min(dp[i][j-1],dp[i+1][j]);

以 i~j 的区域的长度作为阶段即可。

分析 2

博弈问题，可以使用动态规划求解。
状态定义：用 F[i,j]表示第一个玩家先取时，在第 i 到第 j 的子序列中能拿到的最高分；用 S[i][j]表示第 i 到第 j 的子序列中所有数字的和;用 num[i]表示第 1 到第 n 的序列中第 i 个数。

边界条件：F[i][i]=num[i]

状态转移方程：
F[i][j]=max{num[i]+S[i+1][j]-F[i+1][j],num[j]+S[i][j-1]-F[i][j-1]}

结果
p1=F[1][n];
p2=S[1][n]-F[1][n];

解析：
num[i]+S[i+1][j]-F[i+1][j]表示的是,p1 拿第 i 到第 j 最左边的数,然后轮到 p2 在第 i+1 到第 j 的序列中先取，会剩下 S[i+1][j]-F[i+1][j]，这些归 p1。

分析 3

当你在 i 到 j 中取了 i 之后,那么你就只能在 i + 1 到 j 中作为后手取值,故你所能取得的值为 i + 1 到 j 的和减去作为先手所取得的最大值;

初始化

```
 f[i][i] = sum[i][i] = a[i];
```

状态转移方程:

```
 f[i][j] = max(a[i] + sum[i + 1][j] - f[i + 1][j], a[j] + sum[i][j - 1] - f[i][j - 1]);
```

f[i][j] 表示在从第 i 个数到第 j 个数里先手能取得的最大值;
a[i] 表示第 i 个数的值;
sum[i][j] 表示在从第 i 个数到第 j 个数的和;