非完美算法的应用

——河北唐山一中 任一恒

在平时的练习和考试中，我们都是尽量设计出完全正确的算法来解决问题。可是，实际中很多问题都是不能完美解决的，还有很多问题完美解决所需要的时间&空间是根本无法接受的，所以，非完美的算法在实际中有着很广的应用。随着竞赛的题目越来越接近时际，以按优劣计分的题目为代表的考察非完美算法的题目越来越多，本文将讨论一些常用的非完美算法，希望给读者一些启发。

1、 贪心算法

贪心算法的基本思路是从问题的某一个初始解出发逐步逼近给定的目标，以尽可能快的地求得更好的解。当达到某算法中的某一步不能再继续前进时，算法停止。这样我们就得到了一个解，但是我们无法保证解是最优的。下面我们来看看贪心算法的表现。

例题 1 NOI2007 追捕盗贼

某国家要追捕一个大盗，该国家的城市网络是一棵树，现在要你通过在某城市空降警察，让警察从某城市移动到有道路连接的城市，收回某警察来达到捕捉到盗贼的目的。用到的警察越少越好。

这道题的标准算法用到了很多高等知识，而且实现也是相当复杂的，在限定的时间内完美的解决这道题可以说是不能完成的任务，那么我们贪心算法在这道题上的表现如何呢？

我们不妨将原树想象为一棵有根的树，先在根结点空降一个警察，然后再次在根结点空降一个警察，让这个警察走向某棵子树，对这棵子树重复上面的过程，这样一棵子树一棵子树的排除，直到整棵树被排除。这里可以采取一个十分有效的优化就是在只剩一棵子树的时候，不用再安排新的警察，直接让一直守在根结点的那个警察走过去即可。所以不妨安排需要警察最多的那颗子树最后走，这样可以使结果得到很大优化。由于结点数不超过 1000，所以我们可以枚举每个结点为根结点，找出其中需要警察最少的那个。

这个算法虽然存在着反例，但是由于那个十分有用的优化，可以使结果十

分接近标准结果。通过数据试验的结果，有 90%的结果和标准算法产生的结果一致，10%不一致的相差也是十分的小。可以说贪心算法在这道题目上发挥的很好。

贪心算法的优点在于实现容易，时间效率高，而其缺点则是在很多时候和标准算法的差距过大，精确度不够。所以我们要尽量优化贪心算法。优化贪心算法主要从两个方面来着手：1、多和完美算法结合，可以通过在部分使用完美算法而总体使用贪心的方法，通过贪心来结合若干部分的最优结果，尽量的接近真正最优的结果；2、贪心权值的选择，一个贪心算法的关键就是所选的策略，所以在选择权值时我们要尽可能的包含更多的能对全局产生影响的信息，使最终解能更加接近最优解。

二、随机算法

和上面的贪心算法相同，随机算法也是使用较多的一种算法。下面我们来看一道例题。

例题 2 SPOJ1481 Yet another computer network problem

给你一个 n 个点，m 条边的无向图，让你构建一棵生成树,但是每个点的度数不能超过一个给定的限制 b，求所能构建的最小生成树。（1<=n<=10^4,1<=m<=10^5,1<=b<=n）

如果没有度数的限制，那么这道题就是一个十分简单的最小生成树，有了这道题之后我们应该如何处理呢？我们不妨在考虑每个点连出的边选取不超过 b 的前提下用 kruskal 算法计算出一棵生成树。然后，能对结果产生优化的变化必然是用一条由度数已经达到 b 的点连出的一条边和某条边替换另一条由该度数达到b的点连出的边和另一条边。所以我们不妨随机一个度数达到b的点，随机选取它的一条被选中的边和未被选中的边交换，然后添加一条边，这时图中产生了一个环，在环中删除长度最长的边，如果结果比之前更优则保留结果。实践证明，此方法对于该题目能产生十分优秀的结果。

其实，随机算法很少单独应用，更多时候是和其它算法相结合，来使我们的非完美算法产生更加优秀的解。

三、抽样测试法

抽样，即从统计总体中，任意抽出一部分单位作为样本，并以其结果推算

总体的相应指标。在某些问题中，需要让我们检查一系列测试元 s，如果 s 中的某个测试元满足了某个条件，那么则说 s 满足了某个性质。在大度数情况下，我们需要将 s 中的测试元一个一个的进行验证，才能确定 s 是否满足该性质。但是如果 s 满足如下性质，要不 s 中不含满足条件的测试元，要不满足某条件的测试元很多，则可以直接选取几个具有代表性的测试元进行测试，通过这几个测试元来确定 s 是否满足该性质。

例题 3 SPOJ919 Prime Checker

存在一个数列，第一项为 1，以后各项根据公式 $a_i=(a_{i-1}+1234567890) \bmod 2^{31}$ 推出，问这个数列的各项是否为质数。如果是质数的就输出 0，是合数的就输出 1，在时间限制内输出的越多，得到的分数越多。

看到这个题目，我们首先想到的是最为朴素的做法，先求出 1-100000 的素数表，然后对于每个数 a_i，用 1-trunc(sqrt(a_i))间质数去除这个数，如果某个质数能够整除 a_i，则 a_i 为合数，如果均不能整除，则 a_i 为质数。这种方法实现起来无疑是十分简便的，但是由于复杂度较高，所以在时间限制内仅能完成 200000 多一点的数。

下面我们来看一个使用抽样测试法的质数判断方法。对于一个整数 n，设 n-1=d*2^s(d 是奇数)，对于给定的基底 a，如果存在 a^d=1(mod n)或者对于 0<=r<s，存在 a^(d*2^r)=-1(mod n)则称 n 是以 a 为基底的强伪质数。一个质数 n 以所有小于 n 的整数为底都是强伪质数，而一个合数 n，以小于 n 的数为底，n 最多有 1/4 的机会成为强伪质数。根据这条，我们不妨随机地抽取小于 n 的数为底，如果抽取了 k 个数，则正确的概率为 1-1/4^k，k 取到几十的时候一般不会出错了。如果我们不采用随机的方法而是直接抽样特殊底——最小的几个质数，情况怎样呢？

经过试验发现，如果以 2 为底，第一个不符合的数为 2047；如果以 2、3 为底，第一个不符合的数为 1373653；如果以 2、3、5 为底，第一个不符合的数为 25326001；如果以 2、3、5、7 为底，则 2^31 内的数均符合了。

所以，对于本题，我们直接以 2、3、5、7 为底进行测试即可。时间复杂度变为了 O(NlogN)，比之前的朴素算法有了很多的提高。

四、模拟退火算法

模拟退火算法来源于固体退火原理，将固体加温至充分高，再让其徐徐冷却，加温时，固体内部粒子随温升变为无序状，内能增大，而徐徐冷却时粒子渐趋有序，在每个温度都达到平衡态，最后在常温时达到基态，内能减为最小。根据 Metropolis 准则，粒子在温度 T 时趋于平衡的概率为 E-ΔE/(kT)，其中 E 为温度 T 时的内能，ΔE 为其改变量，k 为 Boltzmann 常数。用固体退火模拟组合优化问题，将内能 E 模拟为目标函数值 f，温度 T 演化成控制参数 t，即得到解组合优化问题的模拟退火算法：由初始解 i 和控制参数初值 t 开始，对当前解重复“产生新解→计算目标函数差→接受或舍弃”的迭代，并逐步衰减 t 值，算法终止时的当前解即为所得近似最优解。退火过程由冷却进度表(Cooling Schedule)控制，包括控制参数的初值 t 及其衰减因子 Δt、每个 t 值时的迭代次数 L 和停止条件 S。

模拟退火的基本思想:

(1) 初始化：初始温度 T(充分大)，初始解状态 S(是算法迭代的起点)， 每个 T 值的迭代次数 L

(2) 对 k=1，……，L 做第 3 至第 6 步：

(3) 产生新解 S′

(4) 计算增量 Δt′=C(S′)-C(S)，其中 C(S)为评价函数

(5) 若 Δt′<0 则接受 S′作为新的当前解，否则以概率 exp(-Δt′/T)接受 S′作为新的当前解.

(6) 如果满足终止条件则输出当前解作为最优解，结束程序。终止条件通常取为连续若干个新解都没有被接受时终止算法。

(7) T 逐渐减少，且 T->0，然后转第 2 步。

例题 4 TSP 问题

设有 n 个城市，用数码 1,…,n 代表。城市 i 和城市 j 之间的距离为 d(i，j) i，j=1,…,n. 找出遍访每个域市恰好一次的一条回路，且其路径总长度为最短。

求解 TSP 的模拟退火算法模型可描述如下：

解空间 解空间 S 是遍访每个城市恰好一次的所有回路，是{1，……，n}的所有循环排列的集合，S 中的成员记为(w1,w2 ,……，wn)，并记 wn+1= w1。初始解可选为(1，……，n)

目标函数 此时的目标函数即为访问所有城市的路径总长度或称为代价函数：我们要求此代价函数的最小值。

新解的产生 随机产生1和n之间的两相异数k和m，若k<m，则将 (w1, w2 ,…，wk , wk+1 ,…，wm ,…，wn) 变为： (w1, w2 ,…，wm , wm-1 ,…，wk+1 , wk ,…，wn). 如果是k>m，则将 (w1, w2 ,…，wm , wm+1 ,…，wk,…，wn)变为：(wm, wm-1 ,…，w1 , wm+1 ,…，wk-1 ,wn , wn-1 ,…，wk). 上述变换方法可简单说成是“逆转中间或者逆转两端”。

代价函数差 设将(w1, w2 ,……，wn)变换为(u1, u2 ,……，un), 则代价函数差为：[d(u1,u2)+d(u2,u3)+…+d(un-1 ,un)+d(un,u1)]-[d(w1,w2)+d(w2,w3)+…+d(wn-1 ,wn)+d(wn,w1)]。

将这些部分代入模拟退火算法的基本过程即可解决TSP问题了。

模拟退火算法的应用很广泛，在应用的时候主要注意以下3点：(1) 温度T的初始值设置问题。温度 T 的初始值设置是影响模拟退火算法全局搜索性能的重要因素之一，初始温度高，则搜索到全局最优解的可能性大，但因此要花费大量的计算时间；反之，则可节约计算时间，但全局搜索性能可能受到影响。实际应用过程中，初始温度一般需要依据实验结果进行若干次调整。(2) 退火速度问题。模拟退火算法的全局搜索性能也与退火速度密切相关。一般来说，同一温度下的“充分”搜索(退火)是相当必要的，但这需要计算时间。实际应用中，要针对具体问题的性质和特征设置合理的退火平衡条件。(3) 温度管理问题。温度管理问题也是模拟退火算法难以处理的问题之一。实际应用中，由于必须考虑计算复杂度的切实可行性等问题，常采用如下所示的降温方式：T(t+1)＝k×T(t) 式中k为正的略小于1.00的常数，t为降温的次数。

实践

例题5 百度之星2007 紧急修复

某市的k家公司瘫痪，要在T小时之后才能自动修复，每家公司每小时都在受到损失，第i家公司每小时受损为P(i)，现在派遣n只维修队进行抢修，力求在自动修复之前将损失降到最小。城市被划分为r*c的网格，现给出了每个公司的坐标，每个维修队的初始坐标，每个公司的受损程度。每小时每个维修队可以移动（最多s格），也可以维修它所处在的格子，现在希望你设计一种方案使

损失降低到最小。

对于这道题目，并不存在一个完美的算法，所以我们要想出一个尽可能优秀的算法。首先，我们要进行一个 bfs，计算两点间的距离，由于每个维修队肯定都是以一个公司为某段行程的终点，所以并不需要计算任意两点间的距离，只需要算出每个点到每个公司的距离即可，这一步的时间复杂度为 O(RCK)。

按照题目要求，我们在 T 时间之前修理完越多的公司越好。所以我们不妨安排一个要修理的公司的顺序，按照顺序依次全力修理完每个公司。对于当前选中的公司，如果某个修理队赶到那里能提前结束维修，就让他赶过去修理。如果不能，他就可以前往下一个公司了。公司顺序的选择关系到了我们算法的优秀程度，我们可以使用前面提到的模拟退火方法来实现公司顺序的选择，这样整个算法的结果就十分优秀了。

总结

本文介绍了几种常用的非完美算法，通过以上问题的分析，我们可以看出，非完美算法有着很广泛的应用，它有着时空消耗低，编程复杂度低，思维容易的优点，而且在处理很多题目时，有着不逊于甚至优于完美算法的表现。所以，合理的使用非完美算法能给我们满意的结果。由于学识有限，所以本文的探讨比较肤浅，非完美算法还需要大家在学习时更多的探讨和总结。

参考文献

《非最优化算法初探》杨培 2000 年论文。

《浅析非完美算法在信息学竞赛中的应用》胡伟栋 2005 年论文。

楼天成百度之星解题报告。

《SAA，模拟退火算法》。

【附录】例题原题

NOI2007 DAY2

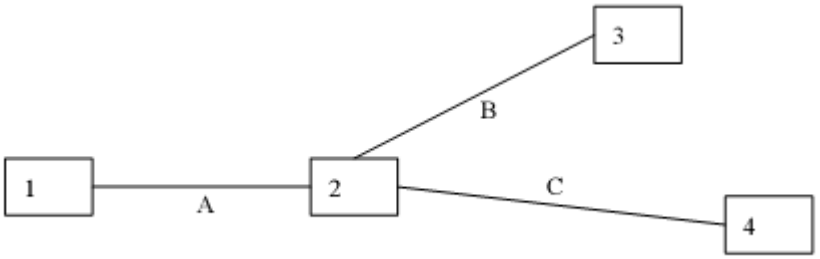
追捕盗贼

问题描述

魔法国度 Magic Land 里最近出现了一个大盗 Frank，他在 Magic Land 四处作案，专门窃取政府机关的机密文件（因而有人怀疑 Frank 是敌国派来的间谍）。为了捉住 Frank，Magic Land 的安全局重拳出击！

Magic Land 由 N 个城市组成，并且这 N 个城市又由恰好 N-1 条公路彼此连接起来，使得任意两个城市间都可以通过若干条公路互达。从数据结构的角度我们也可以说，这 N 个城市和 N-1 条公路形成了一棵树。

例如，下图就是 Magic Land 的一个可能格局（4 个城市用数字编号，3 条公路用字母编号）：



大盗 Frank 能够在公路上以任意速度移动。

比方说，对于上图给出的格局，在 0.00001 秒钟内（或者任意短的一段时间内），Frank 就可以从城市 1 经过城市 2 到达城市 4，中间经过了两条公路。

想要生擒 Frank 困难重重，所以安全局派出了经验丰富的警探，这些警探具有非凡的追捕才能：

1. 只要有警探和 Frank 同处一个城市，那么就能够立刻察觉到 Frank，并且将其逮捕。

2. 虽然 Frank 可以在公路上以任意快的速度移动，但是如果有警探和 Frank 在同一条公路上相遇，那么警探也可以立刻察觉到 Frank 并将其逮捕。

安全局完全不知道 Frank 躲在哪个城市，或者正在哪条公路上移动，所以需要制定一个周密的抓捕计划，计划由若干步骤组成。在每一步中，可以做如下几件事中的一个：

1. 在某个城市空降一位警探。警探可以直接从指挥部空降到 Magic Land 的任意一个城市里。此操作记为“L x”，表示在编号为 x 的城市里空降一位警探。耗时 1 秒。

2. 把留在某个城市里的一位警探直接召回指挥部。以备在以后的步骤中再度空降到某个城市里。此操作记为“B x”。表示把编号为 x 的城市里的一位警探召回指挥部。耗时 1 秒。

3. 让待在城市 x 的一位警探沿着公路移动到城市 y，此操作记为“M x y”。耗时 1 秒。当然，前提是城市 x 和城市 y 之间有公路。如果在警探移动的过程中，大盗 Frank 也在同一条公路上，那么警探就抓捕到了 Frank。

现在，由你来制定一套追捕计划，也就是给出若干个步骤，需要保证：无论大盗 Frank 一开始躲在哪儿，也无论 Frank 在整个过程中如何狡猾地移动（Frank 大盗可能会窃取到追捕行动的计划书，所以他一定会想尽办法逃避），他一定会被缉拿归案。

希望参与的警探越少越好，因为经验丰富的警探毕竟不多。

例如对于前面所给的那个图示格局，一个可行的计划如下：

1. L 2 在城市 2 空降一位警探。注意这一步完成之后，城市 2 里不会有 Frank，否则他将被捉住。

2. L 2 再在城市 2 空降一位警探。

3. M 2 1 让城市 2 的一位警探移动到城市 1。注意城市 2 里还留有另一位警探。这一步完成之后，城市 1 里不会有 Frank，公路 A 上也不会有 Frank。也就是说，假如 Frank 还没有被逮捕，那么他只能是在城市 3 或城市 4 里，或者公路 B 或公路 C 上。

4. B 1 召回城市 1 的一位警探。注意虽然召回了这位警探，但是由于我们始终留了一位警探在城市 2 把守，所以 Frank 仍然不可能跑到城市 1 或者是公路 A 上。

5. L 3 在城市 3 空降一位警探。注意这一步可以空降在此之前被召回的那位警探。这一步完成之后，城市 3 里不会有 Frank，否则他会被捉住。

6. M 3 2 让城市 3 里的一位警探移动到城市 2。这一步完成之后，如果 Frank 还没有被捉住，那他只能是在公路 C 上或者城市 4 里。注意这一步之后，城市 2 里有两位警探。

7. M 2 4 让城市 2 里的一位警探移动到城市 4。这一步完成之后，Frank 一定会被捉住，除非他根本 Magic Land。

这个计划总共需要 2 位警探的参与。可以证明：如果自始至终只有 1 名或者更少的警探参与，则 Frank 就会逍遥法外。

你的任务很简单：对于一个输入的 Magic Land 的格局，计算 S，也就是为了追捕 Frank 至少需要投入多少位警探，并且给出相应的追捕计划步骤。

输入文件

输入文件给出了 Magic Land 的格局。

第 N，代表有 N 个城市，城市的编号是 1~N。

接下来 N-1 行，每行有两个用空格分开的整数 x_i，y_i，代表城市 x_i，y_i 之间有公路相连。保证 $1 \le x_i, y_i \le N$

输出文件

向输出文件输出你所给出的追捕计划。

第一行请输出一个整数 S，代表追捕计划需要多少位警探。

第二行请输出一个整数 T，代表追捕计划总共有多少步。

接下来请输出 T 行，依次描述了追捕计划的每一步。每行必须是以下三种形式之一：

“L x”，其中 L 是大写字母，接着是一个空格，再接着是整数 x，代表在城市 x 空降一位警探。你必须保证 $1 \le x \le N$。

“B x”，其中 B 是大写字母，接着是一个空格，再接着是整数 x，代表召回城市 x 的一位警探。你必须保证 $1 \le x \le N$，且你的计划执行到这一步之前，城市 x 里面确实至少有一位警探。

“M x y”，其中 M 是大写字母，接着是一个空格，再接着是整数 x，再跟一个空格，最后一个是整数 y。代表让城市 x 的一位警探沿着公路移动到城市 y。你必须保证 1≤x, y≤N，且你的计划执行到这一步之前，城市 x 里面确实至少有一位警探，且城市 x, y 之前确实有公路。

必须保证输出的 S 确实等于追捕计划中所需要的警探数目。

样例输入

4

1 2

3 2

2 4

样例输出

2

7

L 2

L 2

M 2 1

B 1

L 3

M 3 2

M 2 4

评分标准

对于任何一个测试点：

如果输出的追捕计划不合法，或者整个追捕计划的步骤数 T 超过了 20000，或者追捕计划结束之后，不能保证捉住 Frank，则不能得分。

否则，用你输出的 S 和我们已知的标准答案 S*相比较：

1. 若 S<S*，则得到 120%的分。
2. 若 S=S*，则得到 100%的分。
3. 若 S*<S≤S*+2，则得到 60%的分。

4. 若 S*+2<S≤S*+4，则得到 40%的分。
5. 若 S*+4<S≤S*+8，则得到 20%的分。
6. 若 S>S*+8，则得到 10%的分。

数据规模和约定

输入保证描述了一棵连通的 N 结点树 1≤N≤1 000。

SPOJ Problem Set

919. Prime checker

Problem code: PRIC

For this task you will have to check as many numbers as possible to see if they are prime. As not to make the problem I/O oriented, consider the numbers you should check in the following order: first take 1 and then construct the numbers in the sequence after the recursion: $a_i=(a_{i-1}+1234567890) \bmod 2^{31}$. Be careful not to use more than 4096 bytes of code.

Output

For each number you should write to output the digit "1" if the number is prime or the digit "0" if it is not prime.

Score

The score of your program will be the index of the first number in the sequence after which you do not have a correct answer. Because of some limitation you should not write more than 33 333 333 characters to output. If you reach this limit, your score will be adjusted in accordance to your runtime.

Example

```
Output:
01000000000000000000000000000100001000000000011000000
```

should receive 50 points.

SPOJ Problem Set

1481. Yet another computer network problem

Problem code: PT07E

ACRush and Jelly are practising in the computer room for next TCO. Suddenly, they found the network is very slow. After a few diagnoses, they realized that there are many redundant wires. So they plan to repair the network, change it to an optimal tree topology. And they can't spend too much money to purchase wires. Then.. too easy? Are you thinking about minimum spanning tree?

But the real trouble is the connectors have their own limitation. They can only allow one computer connects with at most *B* computers.

There are totally 10 cases, arranged in increasing order of the size of *N* (number of computers). Weight of case i-th is w[i] = i. We define *infinity* = $4 * 10^9$. And in a tree, let's call number of computers that computer i connects with is *degree* of computer i.
For case i-th you must show us a satisfied tree with total cost C[i] and maximum degree M[i], considering all computers of that tree.
The formula to compute score is as below:
With case i-th:
If your M[i] <= B then Score[i] = w[i] * C[i]
If your M[i] > B then Score[i] = (w[i] + 10) * C[i] * M[i]

To make the challenge more interesting, with a simple brute force program, we generated 10 upper bound degrees U[i] (1 <= i <= 10) for each of 10 cases.
For any case i-th:
If your M[i] > U[i] then Score[i] = infinity
Finally, TotalScore = (Score[1] + Score[2] + ... + Score[10]) / 10
Try to minimize the *TotalScore*.

Input

First line contains 3 integers *N, M, B* -- number of computers, number of pairs of computers can be connected and the maximum number of computers that a computer can connect with. (1 <= *N* <= 10^4, 1 <= *M* <= 10^5, 1 <= *B* <= N)
Next *M* lines, line i-th contains a triple (u[i], v[i], c[i]) -- means if we want to connect computers u[i] and v[i] we should purchase a wire, cost c[i] (1 <= u[i], v[i] <= N, 1 <= c[i] <= 20000). The wires are bidirectional.

Output

The first line contains 2 numbers --- total cost of your tree and the maximum degree in all computers of that tree. Next, print *N*-1 lines, corresponding to *N*-1 edges of the tree, each edge on one line, forms *u v*.

Example

Input:
3 3 2
1 2 1
2 3 1
1 3 5

Output:
2 2
1 2
2 3

百度之星 2007

紧急修复

背景

2050 年的一天，某市 k 家公司的计算机系统突然同时瘫痪。市政府很快找到了问题的所在，并开始研发一套自动修复整个城市的系统。自动修复系统启用后整个城市的计算机系统将恢复正常，但由于研发时间较长，在此之前各公司仍将蒙受不小的损失。为此，市政府决定派出 n 支维修能力相同的维修队到各公司进行手工修复，力图在自动修复系统启用前整个城市的总损失达到最小。

参数

城市被划分成 R*C 的网格，各行从上到下编号为 1~R，各列从左到右编号为 1~C。每个格子为空地、障碍物和建筑物之一（公司总是位于某个建筑物格子中）。维修队每次只能往上（行编号减 1）、下（行编号加 1）、左（列编号减 1）、右（列编号加 1）四个方向移动，第 i 个维修队每个小时可以移动 s(i)格。维修队在任何时候都不能位于障碍物中，但建筑物可以从它上下左右的相邻空地进入，也可以从这些格子出去。注意：不能直接从一个建筑物格移动到另一个建筑物格，而必须先移动到相邻的空地，在空地上移动，然后再进入另一个建筑物。每个格子的实际尺寸很大，因此可以有多个维修队同时在同一个格子中。

第 i 家公司的位置为(r(i), c(i))，瘫痪程度为 B(i)，每小时的经济损失为 P(i)元。瘫痪程度的单位是“队·小时”，即一支维修队每小时可以把一个公司的瘫痪程度降低 1，而如果 m 支维修队同时为一个公司修复，每小时可以把该公司的瘫痪程度降低 m。

注意，在完全修复之前，每个小时的经济损失不变。

修复计划

政府的修复计划应包含每个小时各维修队所执行的命令。这些命令包括：

1. REST
该命令不进行任何操作。

2. MOVE <seq>
按照 seq 进行移动。其中 seq 为一个长度不超过 s 的非空字符串（如果不需要移动，请使用 REST 命令）。如果 seq 的长度超过 s，则从第 s+1 个字符开始的剩余部分将被删除；如果

该命令不能完全成功的执行，则从第一个非法移动开始的所有后续移动将被忽略。

3. REPAIR
对当前公司进行维修。如果当前公司已经修复或者当前位置不是公司，该命令将被忽略。该命令后面加的所有参数将被忽略。

需要注意的是，每个小时只能执行一条指令，例如不能在 MOVE 之后立刻 REPAIR，必须等待下一个小时。

输入格式

输入的第一行为三个正整数 R, C, T 即网格的行数、列数和自动修复系统的启用时间。以下 R 行每行 C 个字符，表示该城市的地图。点表示空地，#表示障碍物，O 表示建筑物。

下一行包含一个正整数 k，表示公司的数目。以下 k 行每行四个整数 r, c, B, P，其中(r,c)表示该公司坐标(1<=r<=R, 1<=c<=C)，B 为该公司的初始瘫痪程度，P 为每小时的经济损失。(r,c)处保证为一个建筑物格。B 和 P 保证大于 0。

下一行包含两个正整数 n, s，表示维修队的个数和每小时最多移动的格子数。以下 n 行每行包含两个整数 r, c，表示该维修队的初始位置。维修队按照输入文件中的顺序编号为 1~n。

输出格式

输出每 n 行为一组，描述一个小时各维修队的发出的命令。共 T 组，一共 nT 行。所有格式非法的指令将被忽略(等效于 REST)。尽管如此，如果程序输出的命令中没有 REPAIR，或者所有的 REPAIR 都没有成功执行，则输出将视为非法。

输入样例

```
4 7 5
...#OO#
#.....#
O...##O
#......
2
1 5 1 5
3 7 4 6
3
4 7 5
1 1 5
3 1 5
```

输出样例

```
MOVE U
MOVE RRRD
MOVE RDRURD
REPAIR
MOVE DRRU
```

```
MOVE DRRRU
REPAIR
REPAIR
REPAIR
REPAIR
MOVE DRUL
REPAIR
SLEEP
REPAIR
REST
```

模拟器

每小时的模拟过程如下：

*第一步：*对于每个尚未修复的公司 i，将 P(i)累加进总损失。

*第二步：*按照序列从小到大的顺序依次执行各维修队的指令。

为了更清晰的说明规则并帮助选手设计和测试程序，命题组开发了一个简单的模拟器。该模拟器根据输入数据和程序输出进行模拟，给出详细模拟过程，以及最后的结果。最终的测试将严格按照该脚本的输出进行评判。

模拟器用 python 编写，没有安装 python 的选手可以在 http://www.python.org 下载最新版本。

对刚才的样例输入输出，模拟器对各条非法指令给出了警告信息（例如非法的移动、无法识别的指令等）。

评分方法

本题包含 30 个测试点，每个测试点 10 分，共 300 分。

测试点 1~10 满足 R, C<=10, n<=5, k<=10, B<=15, T<=30

测试点 11~20 满足 R, C<=30, n<=10, k<=20, B<=50, T<=500

测试点 21~30 满足 R, C<=100, n,<=100, k<=500, B<=1000, T<=10000

每个测试点独立评分。对于每个测试点，非法输出的得分为 0，合法输出的得分大于 0。设合法输出的程序数为 M，比程序 i 的方案严格更优的程序数为 Y(i)，则该测试点程序 i 的分值为 10(1-Y(i)/M)。换句话说，输出该测试点最优解的程序将获得 10 分，而最差解惟一的情况，输出最差解（但合法）的选手将得到 10/M 分。注意：每个测试点的得分不必为整数。