

# 贪心策略的特点与在信息学竞赛中的应用

本文为剪报式资料初稿，仅供个人学习参考，不宜外传。

## 一、 引 论

信息，人类社会发展的标志。人类对信息的记载，可以追溯到原始社会。在漫长的人类社会发展过程中，伴随着科学技术的发展，人类对客观世界的认识不断加深，现实世界的信息量急剧增大。为了满足人们对大数据量信息处理的渴望，1946 年世界上第一台电子数字计算机 ENIAC 应运而生。在此后的半个世纪中，为解决各种实际问题，计算机算法学得到了飞速的发展。线形规划、动态规划等一系列运筹学模型纷纷运用到计算机算法学中，解决了诸如经济决策等一系列现实问题。在众多的计算机解题策略中，贪心策略可以算得上是最接近人们日常思维的一种解题策略，正基于此，贪心策略在各级各类信息学竞赛、尤其在对 NPC 类问题的求解中发挥着越来越重要的作用。

## 二、 贪心策略的定义

**【定义 1】** 贪心策略是指从问题的初始状态出发，通过若干次的贪心选择而得出最优值(或较优解)的一种解题方法。

其实，从“贪心策略”一词我们便可以看出，贪心策略总是做出在当前看来是最优的选择，也就是说贪心策略并不是从整体上加以考虑，它所做出的选择只是在某种意义上的局部最优解，而许多问题自身的特性决定了该题运用贪心策略可以得到最优解或较优解。

## 三、贪心算法的特点

通过上文的介绍，可能有人会问：贪心算法有什么样的特点呢？我认为，适用于贪心算法解决的问题应具有以下 2 个特点：

### **1、贪心选择性质：**

所谓贪心选择性质是指应用同一规则  $f$ ，将原问题变为一个相似的、但规模更小的子问题、而后的每一步都是当前看似最佳的选择。这种选择依赖于已做出的选择，但不依赖于未做出的选择。从全局来看，运用贪心策略解决的问题在程序的运行过程中无回溯过程。关于贪心选择性质，读者可在后文给出的贪心策略状态空间图中得到深刻地体会。

### **2、局部最优解：**

我们通过特点 2 向大家介绍了贪心策略的数学描述。由于运用贪心策略解题在每一次都取得了最优解，但能够保证局部最优解得不一定是贪心算法。如大家所熟悉得动态规划算法就可以满足局部最优解，在广度优先搜索（BFS）中的解题过程亦可以满足局部最优解。

在遇到具体问题时，许多选手往往分不清哪些题该用贪心策略求解，哪些题该用动态规划法求解。在此，我们对两种解题策略进行比较。

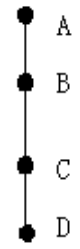
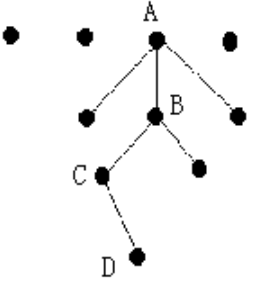
	贪心策略	动态规划
相同点	1 时间效率较高、适用于对问题的求解有严格时间限制的题目。 2 均能保证局部最优解。	
不同点	所占空间较小	所占空间较大
	某次具体选择必是最优选择	某次具体选择不一定是最优选择
	解题大体框架： 线形  A—B—C—D为最优选择，每一中间结点B、C也为最优选择	解题大体框架： 图或树  A—B—C—D为最优解，B、C不一定是最优选择

图 1

【引例】在一个  $N \times M$  的方格阵中，每一格子赋予一个数（即为权）。规定每次移动时只能向上或向右。现试找出一条路径，使其从左下角至右上角所经过的权之和最大。

我们以  $2 \times 3$  的矩阵为例。

若按贪心策略求解，所得路径为：1 3 4 6；

3	4	6
1	2	10

图 2

若按动态规划法求解，所得路径为：1 2 10 6。

由于贪心策略自身的特点，使得数字 10 所在的格子成为一个“坏格子”，即运用

贪心策略找不到它，而运用动态规划法求解的第一步（1 2）并不是最优选择，但却保证了全局最优解；运用贪心策略求解的第一步（1 3）保证了

$a_{11}$	$a_{12}$	$a_{13}$	$a_{14}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{24}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{34}$

图 3

局部最优解，却无法保证全局最优解。我们若用图 3 所示的  $N \times M$  的矩阵表示一组数,设运用与引例同样的移动规则后得到了由若干个元素组成的数列 A。可得如下结论：

若  $a_{n-1,1} > a_{n,2}$  ,则  $a_{n,2}$ 、 $a_{n,3}$ 、... $a_{n,m}$  一定不在数列 A 中。对于一元素  $a_{n,p}$  ( $2 \leq p \leq m$ ) , 设  $a_{n,p} =$  ,若保证全局最优解，则  $a_{n,p}$  必在数列 A 中，但运用贪心策略求解时  $a_{n,p}$  不在数列 A 中。由此可见，贪心策略并不到达问题状态的全部空间。若用空间图来表示贪心算法和动态规划算法（如下图），我们可以清楚地看到，贪心算法

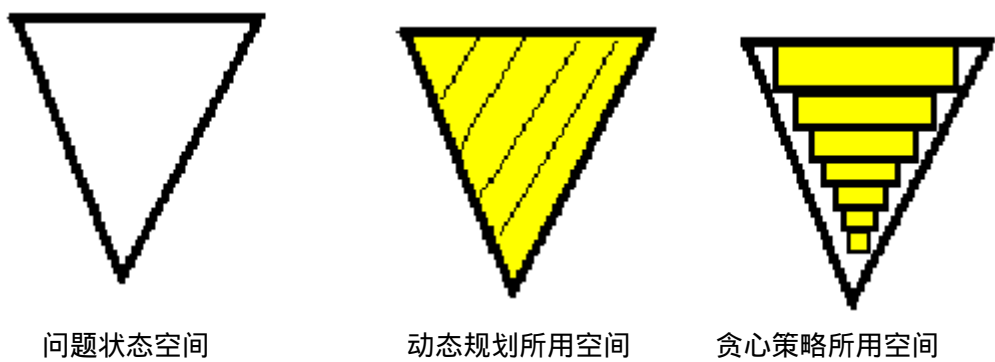


图 4

是一种对输入数据进行不断收缩的过程。它并不到达问题的全部状态空间。这是由本文所述的贪心策略的线形解题框架所决定的。

四、 贪心策略的理论基础

——矩阵胚

正如前文所说的那样，贪心策略是最接近人类认知思维的一种解题策略。但是，越是显而易见的方法往往越难以证明。下面我们就来介绍贪心策略的理论——矩阵胚。

“矩阵胚”理论是一种能够确定贪心策略何时能够产生最优解的理论，虽然这套理论还很不完善，但在求解最优化问题时发挥着越来越重要的作用。

**【定义 3】** 矩阵胚是一个序对  $M=[S, I]$ ，其中  $S$  是一个有序非空集合， $I$  是  $S$  的一个非空子集，成为  $S$  的一个独立子集。

如果  $M$  是一个  $N \times M$  的矩阵的话，即

$$M = \begin{pmatrix} a_{11}, a_{12}, \dots, a_{1m} \\ \dots \dots \dots \\ a_{i1}, a_{i2}, \dots, a_{im} \\ \dots \dots \dots \\ a_{n1}, a_{n2}, \dots, a_{nm} \end{pmatrix}$$

$S$  是  $M$  的各个行， $S = (a_1, a_2, \dots, a_n)$ ， $I$  是线形无关的若干行  $a_i, a_j, a_p, \dots$

若  $M$  是无向图  $G$  的矩阵胚的话，则  $S$  为图的边集， $I$  是所有构成森林的一组边的子集。

如果对  $S$  的每一个元素  $X (X \in S)$  赋予一个正的权值  $W(X)$ ，则称矩阵胚  $M = (S, I)$  为一个加权矩阵胚。

适宜于用贪心策略来求解的许多问题都可以归结为在加权矩阵胚中找一个具有最大权值的独立子集的问题，即给定一个加权矩阵胚， $M = (S, I)$ ，若能找出一个独立且具有最大可能权值的子集  $A$ ，且  $A$  不被  $M$  中比它更大的独立子集所包含，那么  $A$  为最优子集，也是一个最大的独立子集。

我们认为，针对绝大多数的信息学问题，只要它具备了“矩阵胚”的结构，便可用贪心策略求解。矩阵胚理论对于我们判断贪心策略是否适用于某一复杂问题是十分有效的。

## 五、 几种典型的贪心算法

贪心策略在图论中有着极其重要的应用。诸如 Kruskal、Prim、Dijkstra 等体现“贪心”思想的图形算法更是广泛地应用于树与图的处理。下面就分别来介绍 Kruskal 算法、Prim 算法和 Dijkstra 算法。

### 、库鲁斯卡尔 (Kruskal) 算法

【定义 4】 设图  $G=(V, E)$  是一简单连通图,  $|V|=n, |E|=m$ , 每条边  $e_i$  都给以权  $W_i$ ,  $W_i$  假定是边  $e_i$  的长度 (其他的也可以),  $i=1, 2, 3, \dots, m$ 。求图  $G$  的总长度最短的树, 这就是最短树问题。

kruskal 算法的基本思想是: 首先将赋权图  $G$  的边按权的升序排列, 不失一般性为:  $e_1, e_2, \dots, e_m$ 。其中  $W_i \leq W_{i+1}$ , 然后在不构成回路的条件下择优取进权最小的边。

其流程如下:

- (1) 对属于  $E$  的边进行排序得  $e_1, e_2, \dots, e_m$ 。
- (2) 初始化操作  $w=0, T=\emptyset, k=0, t=0$ ;
- (3) 若  $t=n-1$ , 则转 (6), 否则转 (4)
- (4) 若  $T \cup \{e_k\}$  构成一回路, 则作  
    【 $k=k+1$ , 转 (4)】
- (5)  $T=T \cup \{e_k\}, w=w+W_k, t=t+1, k=k+1$ , 转 (3)
- (6) 输出  $T, w$ , 停止。

下面我们对这个算法的合理性进行证明。

设在最短树中, 有边  $v_i, v_j$ , 连接两顶点  $v_i, v_j$ , 边  $v_i, v_j$  的权为  $w_p$ , 若  $v_i, v_j$  加入到树中不能保证树的总长度最短, 那么一定有另一条边  $v_i, v_j$  或另两条边  $v_i, v_k$ 、 $v_k, v_j$ , 且  $W(v_i, v_j) > W_p$  或  $W(v_i, v_k) + W(v_k, v_j) < W_p$ , 因为  $v_i, v_q$ 、 $v_q, v_p$  不在最短树中, 可知当  $v_i, v_q$ 、 $v_q, v_p$  加入到树中时已构成回路, 此时程序终止。因为  $v_i, v_k \in T, v_k, v_j \in T$  且  $W(v_i, v_k) + W(v_k, v_j) < W_p$ , 与程序流程矛盾。

### 、普林 (Prim) 算法:

Kruskal 算法采取在不构成回路的条件下, 优先选择长度最短的边作为最短树的

边，而 Prim 则是采取了另一种贪心策略。

已知图  $G=(V, E)$ ,  $V=\{v_1, v_2, v_3, \dots, v_n\}$ ,  $D=(d_{ij})_{n \times n}$  是图  $G$  的矩阵，若  $v_i, v_j \in E$ ，则令  $d_{ij} =$  ，并假定  $d_{ij} =$

Prim 算法的基本思想是：从某一顶点（设为  $v_1$ ）开始，令  $S = \{v_1\}$ ，求  $V \setminus S$  中点与  $S$  中点  $v_1$  距离最短的点，即从矩阵  $D$  的第一行元素中找到最小的元素，设为  $d_{1j}$ ，则令  $S = S \cup \{v_j\}$ ，继续求  $V \setminus S$  中点与  $S$  的距离最短的点，设为  $v_k$ ，则令  $S = S \cup \{v_k\}$ ，继续以上的步骤，直到  $n$  个顶点用  $n-1$  条边连接起来为止。

流程如下：

(1) 初始化操作： $T =$  ，  $q(1) = -1$ ， $i$  从 2 到  $n$  作

    【 $p(i) = 1, q(i) = d_{i1}$ 】， $k = 1$

(2) 若  $k = n$ ，则作【输出  $T$ ，结束】

    否则作【 $\min$  ，  $j$  从 2 到  $n$  作

        【若  $0 < q(i) < \min$  则作

            【 $\min = q(i), h = j$ 】

        】

    】

(3)  $T = T \cup \{h, p(h)\}, q(h) = -1$

(4)  $j$  从 2 到  $n$  作

    【若  $d_{hj} < q(j)$  则作【 $q(j) = d_{hj}, p(j) = h$ 】】

(5)  $k = k + 1$ ，转(2)

算法中数组  $p(i)$  是用以记录和  $v_i$  点最接近的属于  $S$  的点， $q(i)$  则是记录了  $v_i$  点和  $S$  中点的最短距离， $q(i) = -1$  用以表示  $v_i$  点已进入集合  $S$ 。算法中第四步： $v_n$  点进入  $S$  后，对不属于  $S$  中的点  $v_j$  的  $p(j)$  和  $q(j)$  进行适当调整，使之分别记录了所有属于  $S$  且和  $S$  距离最短的点和最短的距离，点  $v_1, v_2, \dots, v_n$  分别用 1, 2, ...,  $n$  表示。

### 、戴克斯德拉 (Dijkstra) 算法：

已知图  $G=(V, E)$ ,  $V=\{v_1, v_2, \dots, v_n\}$ ，起始顶点  $v_0$ ，求  $v_0$  点到其他各点的最短路径。

算法如下：

令  $S$  表示已求出最短路径的顶点集合。对于不在  $S$  中的顶点  $w$ ，令  $d(w)$  表

示从  $v_0$  开始且通过  $S$  中的顶点到达  $w$  的最短路径的长度。 $S$  的初态为空，若  $v_0$  通过  $S$  中的顶点到  $w$  有路径，则  $d(w)$  为  $v_0$  到  $w$  的最短的一条路径的长度；若  $v_0$  没有经  $S$  中的顶点到  $w$  的路径，则  $d(w) = \infty$ ，初始时设  $d(v_n) = 0$ ，除  $v_0$  外的所有顶点  $v_i, v_j$  ( $v_i \in V$  且  $v_i \neq v_j$ )， $d(v_i) = \infty$ 。

## 六、 贪心策略的应用

在现实世界中，我们可以将问题分为两大类。其中一类被称为  $P$  类问题，它存在有效算法，可求得最优解；另一类问题被称为  $NPC$  类问题，这类问题到目前为止人们尚未找到求得最优解的有效算法，这就需要每一位程序设计人员根据自己对题目的理解设计出求较优解的方法。下面我们着重分析贪心策略在求解  $P$  类问题中的应用。

### § 6.1 贪心策略在 $P$ 类问题求解中的应用

#### § 6.1.1 贪心策略在求 $P$ 类最优解问题中的应用

在现实生活中， $P$  类问题是十分有限的，而  $NPC$  类问题则是普遍的、广泛的。在国际信息学奥林匹克竞赛的发展过程中，由于受到评测手段的限制，在 1989 年至 1996 年的 8 年赛事中，始终是以  $P$  类问题为主的，且只允许求最优解。在这些问题上，有的题目可以用贪心策略来直接求解，有的题目运用贪心策略后可以使问题得到极大的简化，使得程序对大信息量的处理提供了可能。

#### [例 1] 删数问题

**试题描述** 键盘输入一个高精度的正整数  $N$ ，去掉其中任意  $S$  个数字后剩下的数字按左右次序组成一个新的正整数。对给定的  $N$  和  $S$ ，寻找一种删数规则使得剩下得数字组成的新数最小。

**试题背景** 此题出自 NOI94

**试题分析** 这是一道运用贪心策略求解的典型问题。此题所需处理的数据从表面上看是一个整数。其实，大家通过对此题得深入分析便知：本题所给出的高精度正整数在具体做题时将它看作由若干个数字所组成的一串数，这是求解本题的一个重要



突破。这样便建立起了贪心策略的数学描述。

## [例 2] 数列极差问题

**试题描述** 在黑板上写了  $N$  个正整数作成的一个数列，进行如下操作：每一次擦去其中的两个数  $a$  和  $b$ ，然后在数列中加入一个数  $a \times b + 1$ ，如此下去直至黑板上剩下一个数，在所有按这种操作方式最后得到的数中，最大的  $\max$ ，最小的为  $\min$ ，则该数列的极差定义为  $M = \max - \min$ 。

编程任务：对于给定的数列，编程计算出极差  $M$ 。

**试题背景** 这是 1997 年福建队选拔赛的一道题目。

**试题分析** 当看到此题时，我们会发现求  $\max$  与求  $\min$  是两个相似的过程。若我们把求解  $\max$  与  $\min$  的过程分开，着重探讨求  $\max$  的问题。

下面我们以求  $\max$  为例来讨论此题用贪心策略求解的合理性。

讨论：假设经  $(N - 3)$  次变换后得到 3 个数： $a, b, \max'$

$(\max', a, b)$ ，其中  $\max'$  是  $(N - 2)$  个数经  $(N - 3)$  次  $f$  变换后所得的最大值，此时有两种求值方式，设其所求值分别为  $Z_1, Z_2$ ，则有： $Z_1 = (a \times b + 1) \times \max' + 1$ ， $Z_2 = (a \times \max' + 1) \times b + 1$  所以  $Z_1 - Z_2 = \max' - b$  若经  $(N - 2)$  次变换后所得的 3 个数为： $m, a, b$  ( $m, a, b$ ) 且  $m$  不为  $(N - 2)$  次变换后的最大值，即  $m < \max'$  则此时所求得的最大值为： $Z_3 = (a \times b + 1) \times m + 1$  此时  $Z_1 - Z_3 = (1 + a \times b)(\max' - m) > 0$  所以此时不为最优解。

所以若使第  $k$  ( $1 \leq k \leq N - 1$ ) 次变换后所得值最大，必使  $(k - 1)$  次变换后所得值最大(符合贪心策略的特点 2)，在进行第  $k$  次变换时，只需取在进行  $(k - 1)$  次变换后所得数列中的两最小数  $p, q$  施加  $f$  操作： $p \leftarrow p \times q + 1, q \leftarrow p$  即可(符合贪心策略特点 1)，因此此题可用贪心策略求解。讨论完毕。

在求  $\min$  时，我们只需在每次变换的数列中找到两个最大数  $p, q$  施加作用  $f$ ： $p \leftarrow p \times q + 1, q \leftarrow p$  即可。原理同上。

这是一道两次运用贪心策略解决的一道问题，它要求选手有较高的数学推理能力。

## [例 3] 最优乘车问题

**试题描述** H 城是一个旅游胜地，每年都有成千上万的人前来观光。为方便游客，

巴士公司在各个旅游景点及宾馆、饭店等地都设置了巴士站，并开通了一些单向巴士线路。每条单向巴士线路从某个巴士站出发，依次途径若干个巴士站，最终到达终点巴士站。

阿昌最近到H城旅游，住在CUP饭店。他很想去看S公园游玩。听人说，从CUP饭店到S公园可能有也可能没有直通巴士。如果没有，就要换乘不同线路的单向巴士，还有可能无法乘巴士到达。

现在用整数1, 2, ..., n给H城的所有巴士站编号，约定CUP饭店的巴士站编号为1，S公园巴士站的编号为N。

写一个程序，帮助阿昌寻找一个最优乘车方案，使他在从CUP饭店到S公园的过程中换车的次数最少。

**试题背景** 出自NOI 97

**试题分析** 此题看上去很像一道搜索问题。在搜索问题中，我们所求的使经过车站数最少的方案，而本题所求解的使换车次数最少的方案。这两种情况的解是否完全相同呢？我们来看一个实例：

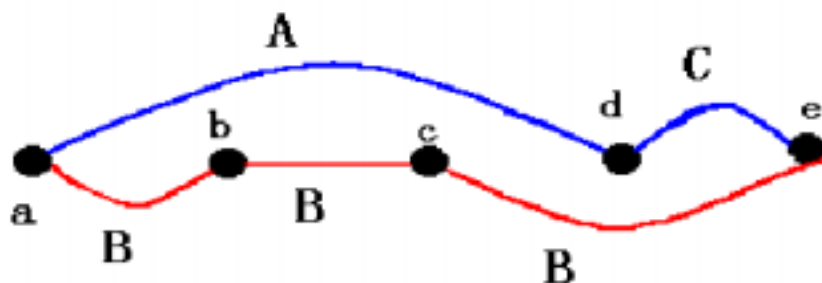


图 5

如图5所示：共有5个车站（分别为a、b、c、d、e），共有3条巴士线（线路A：a d；线路B：a b c e；线路C：d e）。此时要使换车次数最少，应乘坐线路B的巴士，路线为：a b c e，换车次数为0；要使途经车站数最少，乘坐线路应为a d e，换车次数为1。所以说使换车次数最少的路线和使途经车站数最少的方案不一定相同。这使不能用搜索法求解此问题的原因之一。

原因之二，来自对数学模型的分析。我们根据题中所给数据来建立一个图后会发现该图中存在大量的环，因而不适合用搜索法求解。

题目分析到这里，我们可以发现此题与 NOI93 的求最长路径问题有相似之处。其实，此题完全可以套用上文所提到的 Dijkstra 算法来求解。

以上三道题只是使用了单一的贪心策略来求解的。而从近几年的信息学奥林匹克竞赛的命题方向上看，题目更加灵活，同时测试数据较大，规定的出解时间较短。在一些问题中，我们采用贪心策略对问题化简，从而使程序具有更高的效率。

#### [ 例 4 ] 最佳浏览路线问题

**试题描述** 某旅游区的街道成网格状（见图），其中东西向的街道都是旅游街，南北向的街道都是林荫道。由于游客众多，旅游街被规定为单行道。游客在旅游街上只能从西向东走，在林荫道上既可以由南向北走，也可以从北向南走。

阿隆想到这个旅游区游玩。他的好友阿福给了他一些建议，用分值表示所有旅游街相邻两个路口之间的道路值得浏览得程度，分值从-100到100的整数，所有林荫道不打分。所有分值不可能全是负值。

例如下图是被打过分的某旅游区的街道图：

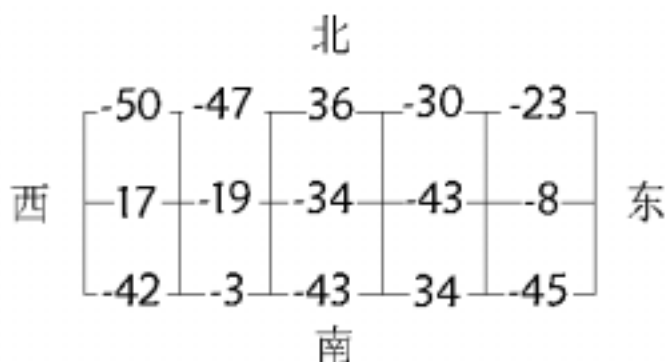


图 6

阿隆可以从任一路口开始浏览，在任一路口结束浏览。请你写一个程序，帮助阿隆寻找一条最佳的浏览路线，使得这条路线的所有分值总和最大。

**试题背景** 这道题同样出自 NOI 97，'97 国际大学生程序设计竞赛的第二题（吉尔的又一个骑车问题）与本题是属于本质相同的题目。

**试题分析** 由于林荫道不打分，也就是说，无论游客在林荫道中怎么走，都不会影响得分。因题可知，若游客需经过某一列的旅游街，则他一定要经过这一列的M条旅游街中分值最大的一条，才会使他所经路线的总分值最大。这是一种贪心策略。

贪心策略的目的是降维，使题目所给出的一个矩阵变为一个数列。下一步便是如何对这个数列进行处理。在这一步，很多人用动态规划法求解，这种算法的时间复杂度为  $O(n^2)$ ，当林荫道较多时，效率明显下降。其实在这一步我们同样可以采用贪心法求解。这时的时间复杂度为  $O(n)$ 。

### § 6.1.2 贪心策略在求 P 类较优解问题中的应用

正如其他学科奥林匹克竞赛一样，国际信息学奥赛的发展同样经历了一个逐步成熟的发展过程。回顾十余年赛事的发展，我们不妨将国际信息学奥赛的发展分为两个阶段：第一阶段是 1989—1996 年，这一时期奥赛题目的特点是：试题全部为 P 类问题，且只允许求最优解，题目的设计强调对选手基本算法的掌握。第二阶段为 1997 年至今。在南非举行的 IOI97 中，命题方向一举突破传统模式，NPC 类问题在竞赛中大量出现，每道题目到具有一定的实际背景，引进了崭新的程序评测机制。在求解 P 类问题时允许得出较优解并得到相应的分数。这些变化无疑更好地考察了选手的综合素质。在对 P 类较优解问题的求解过程中，贪心策略无疑扮演着重要角色。IOI97 中的障碍物探测器问题便是运用贪心策略来求得较优解的 P 类问题。

#### [例 5] 障碍物探测器问题

**试题描述** 有一个登陆舱 (POD)，里面装有许多障碍物探测车 (MEV)，将在火星表面着陆，着陆后，探测车离开登陆舱向相距不远的先期到达的传送器 (Transmitter) 移动。MEV 一边移动，采集岩石 (ROCK) 标本，岩石由第一个访问到它的 MEV 所采集，每块岩石只能被采集一次，但是这以后，其他 MEV 可以从该处通过。探测车 MEV 不能通过有障碍的地面。

本题限定探测车 MEV 只能沿着格子向南或向东从登陆处向传送器 transmitter 移动，允许多个探测车 MEV 在同一时间占据同一位置。

警告：如果某个探测车 MEV 在到达传送器以前不能在继续合法前进时，则车中的石块必定不可挽回地全部丢失。

任务：计算机探测车的每一步移动，使其送到传送器的岩石标本的数量尽可能多。这两项都做到会使你的得分最高。

输入：火星表面上登陆舱 POD 和传送器之间的位置用网格 P 和 Q 表示，登陆舱 POD 的位置总是在 (1, 1) 点，传送器的位置总是在 (P, Q) 点。

火星上的不同表面用三中不同的数字符号来表示：

0 代表平坦无障碍

1 代表障碍

2 代表石块

输入文件的第一行为探测车的个数，第二行为 P 的值，第三行为 Q 的值。接下来的 Q 行为一个  $Q \times P$  的矩阵。

输出：表示 MEV 移向 transmitter 的行动序列。每行包含探测车号和一个数，0 或 1，这里 0 表示向南移动，1 表示向东移动。

得分：分数的计算将根据收集的岩石样本（取到传送器上）的数目，MEV 到达传送器和不到达传送器的数目有关

非法移动将导致求解无效，并记作零分，当 MEV 的障碍物上移动或移出网格，即视为非法。

得分=（收集的样品并取到传送器上的数目+MEV 到达传送器上的数目- MEV 没有到达传送器上的数目）与应得的最大的数目之比（%）  
最高分为 100%，最低分为 0%

**试题背景** IOI ' 97 中的第一试第一题。国际信息学奥赛中出现的第一道 NPC 类问题。1997 年美国的探测器再次到达火星。火星及太空搜索引起了人们的广泛关注，此题便是以此为素材而创作的。

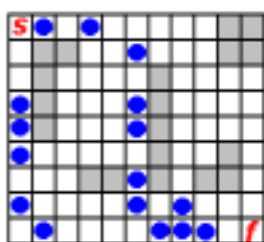


图 7

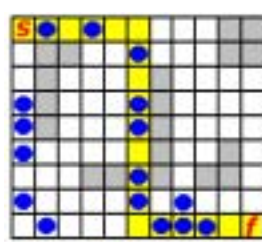


图 8

**试题分析** 关于迷宫问题相信每一个参加信息学奥赛的选手都不会陌生。对于不同的迷宫，我们可用搜索策略或动态规划进行求解。在本题中，无论运用哪种解题策略均不能得到问题的最优解，我们的任务是合理选择一种解题策略，使我们运用该策略得到的较优解尽可能地接近最优解。我们先来看一个例子（如图 7 所示）。对于一个探测车而言，我们运用动态规划的方法使探测车经过岩石最多的一条路线便可

得到问题的最优解（如图 8 所示），这时共可收集到岩石 10 个。

当有 2 个探测车时，我们让第 2 辆探测车在图 8 的基础上从地图的起点 S 行进至终点 f（如图 9 所示），这时我们共收集到岩石 15 个。而实际上两辆探测车可收集到地图中的全部岩石（共 16 个），

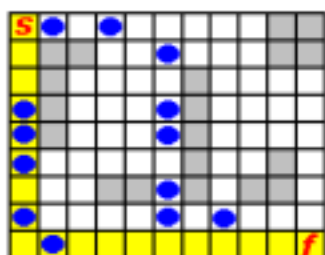


图 9

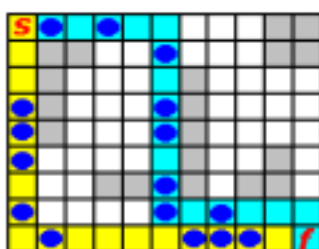


图 10

当探测车数量为 3 时，我们可以收集到全部的 16 个岩石。

我们可让从起点出发的每一辆探测车都收集到尽可能多的岩石，这实际上是一种贪心策略。对于本题而言，贪心策略并不能保证所得结果全部为最优解，但由于每一辆探测车都收集尽可能多的岩石，而对于由计算机随机产生的测试数据而言，岩石是比较均匀地分布在地图中的，于是我们认为：

## 探测车收集岩石数 探测车所游历的地图空间

让每一辆探测车收集尽可能多的岩石，也就是让探测车经过尽可能大的地图空间。所以在探测车数量逐渐增多时，所有探测车所经过的地图空间越多，收集到的岩石也就越多，此时也就越接近最优解。

此题是否存在最优解呢？其实，我们可以用网络流的算法来解决此题。但实践证明，用网络流算法去求解本题所占空间较大，编程复杂度较高且程序调试起来较为困难，因此在实际比赛中，在限定的时间内用贪心策略完成对题目的求解不失为上策。

## § 6.2 关于运用贪心策略求解 NPC 类问题的讨论

正如前面所讲的那样，在南非举行的第九届国际奥林匹克信息学竞赛中首次引入了 NPC 类问题，在杭州举行的 NOI98 中引入了 NOI 发展史上的第一道 NPC 类问题——并行计算。可以说，NPC 类问题正在日益引起人们的兴趣。它要求选手根据题意自己建立适当的模型，使程序的解尽量逼近最优解。目前，信息学竞赛所涉及到的少量 NPC 类问题主要是运用贪心策略或随机化算法去求较优解的。但是对于同一道 NPC 类问题来说，运用不同的贪心选择所求得的最优解是不同的，不同的贪心选择针对不同的测试数据所得解与最优解的逼近程度也是不同的。所以有关 NPC 类问题的众多特性以及哪些问题运用贪心策略求得的较优解逼近于最优解仍是需要我们花费大量精力去研究的。信息学奥林匹克的精华——激励创新也许在求解 NPC 类问题时会得到最大程度的体现。

## 七、 结束语

通过对贪心策略的分析，大家可以看出：贪心策略作为一种高效算法，广泛地应用与信息学奥林匹克竞赛中。即使表面上看起来与贪心策略关系甚微的题目，运用贪心算法也可使程序的运行效率大大提高。因此，深刻理解贪心策略的数学模型、特点、理论基础、尤其是运用其基本思想解决具体问题是十分重要的。希望本文能给参赛选手以一定的启发。

### 【附 录】

#### 【部分试题及源程序】

##### 1、 吉尔的又一个乘车问题：

Input file:jill.in

Jill likes to ride her bicycle, but since the pretty city of Greenhills where she lives has grown, Jill often uses the excellent public bus system for part of her journey. She has a folding bicycle which she carries with her when she uses the bus for the first part of her

trip. She follows the bus route until she reaches her destination or she comes to a part of the city she does not like. In the latter event she will board the bus to finish her trip.

Through years of experience, Jill has rated each road on an integer scale of “niceness”. Positive niceness values indicate roads Jill likes; negative values are used for roads she does not like. Jill plans where to leave the bus and start bicycling, as well as where to stop bicycling and re-join the bus, so that the sum of niceness values of the roads she bicycles on is maximized. This means that she will sometimes cycle along a road she does not like, provided that it joins up two other parts of her journey involving roads she likes enough to compensate. It may be that no part of the route is suitable for cycling so that Jill takes the bus for its entire route. Conversely, it may be that the whole route is so nice Jill will not use the bus at all.

Since there are many different bus routes, each with several stops at which Jill could leave or enter the bus, she feels that a computer program could help her identify the best part to cycle for each bus route.

#### INPUT

The input file contains information on several bus routes. The first line of the file is a single integer  $b$  representing the number of route descriptions in the file. The identifier for each route ( $r$ ) is the sequence number within the data files,  $1 \leq r \leq b$ . Each route description begins with the number of stops on the route: an integer  $s$ ,  $2 \leq s \leq 20,000$  on a line by itself. The number of stops is followed by  $s-1$  lines, each line  $i$  ( $1 \leq i \leq s$ ) is an integer  $n_i$  representing Jill's assessment of the niceness of the road between the two stops  $i$  and  $i+1$ .

#### OUTPUT

For each route  $r$  in the input file, your program should identify the beginning bus stop  $i$  and the ending bus stop  $j$  that identify the segment of the route which yields the maximal sum of niceness  $m = n_i + n_{i+1} + \dots + n_{j-1}$ . If more than one segment is maximally nice, choose the one with longest cycle ride (largest  $j-i$ ). To break ties in longest maximal segments, choose the segment that begins with the earliest stop (lowest  $i$ ). For each route  $r$  in the input file, print a line in the form:



The nicest part of route  $r$  is between stops  $i$  AND  $j$ .

However, if the maximal sum is not positive, your program should print:

Route  $r$  has no nice parts.

INPUT SAMPLE

```
3
3
- 1
6
10
4
5
4
3
4
4
4
4
- 5
4
2
3
4
```

OUTPUT SAMPLE

The nicest part of route 1 is between stops 2 and 3

The nicest part of route 2 if between stops 3 and 9

Route 3 has no nice parts

## 2、求最长路径问题 (NOI 93): (请查原题参考)

**解题简析** 对一个不存在回路的有向图, 编程求出途经结点数最多的一条路径。有向图存放在一个文本文件中, 第 0 行为一个数字, 为该图的结点总数  $N$ , 其下还有  $N$  行, 每行有  $N$  个非 0 即 1 的数字。若第  $i$  行第  $j$  列的数字为 1, 则表示结点  $i$  到结

点  $j$  存在由  $i$  指向  $j$  的边，否则该数为 0。

### 3、删数问题的源程序：

输入数据：一个高精度正整数  $N$ ，所删除的数字个数  $S$ 。

输出数据：去掉的数字的位置和组成的新的正整数。

```
Program Delete_digit;
Var n:string; {n 是由键盘输入的高精度正整数}
    s,a,b,c:byte; {s 是所要删除的数字的个数}
    data:array[1..200] of 0..9; {记录删除的数字所在位置}
begin
    readln(n);
    readln(s);
    for a:=1 to s do
        for b:=1 to length(n) do if n[b]>n[b+1] then {贪心选择}
            begin
                delete(n,b,1);
                data[a]:=b+a-1; {记录所删除的数字的位置}
                break;
            end;
    while n[1]='0' do delete(n,1,1); {将字符串首的若干个“0”去掉}
    writeln(n);
    for a:=1 to s do writeln(data[a], ' ');
end.
```

### 4、最优乘车问题

输入数据：输入文件 INPUT.TXT。文件的第行是一个数字  $M$  ( $1 \leq M \leq 100$ ) 表示开通了  $M$  条单向巴士线路，第 2 行是一个数字  $N$  ( $1 \leq N \leq 500$ )，表示共有  $N$  个车站。从第 3 行到第  $M+2$  行依次给出了第一条到第  $M$  条巴士线路的信息。其中第  $i+2$  行给出的是第  $i$  条巴士线路的信息，从左至右依次按行行驶顺序给出了该线路上的所有站点，相邻两个站号之间用一个空格隔开。

输出数据：输出文件是 OUTPUT.TXT。文件只有一行，为最少换车次数（在  $0, 1, \dots, M-1$  中取值），0 表示不需换车即可达到。如果无法乘车达到  $S$  公园，则输出“NO”。

```
Program Travel;
var m:1..100; {m 为开通的单向巴士线路数}
    n:1..500; {n 为车站总数}
    result:array[1..501] of -1..100; {到某车站的最少换车数}
    num:array[1..500,1..50] of 1..500; {从某车站可达的所有车站序列}
    sum:array[1..500] of 0..50; {从某车站可达的车站总数}
    check:array[1..500] of Boolean; {某车站是否已扩展完}
Procedure Init;
var f1:text;
```

```

    a, b, c, d: byte;
    data: array[1..100] of 0..100;
begin
    assign(f1, 'input.txt');
    reset(f1);
    readln(f1, m);
    readln(f1, n);
    result[501]:=100;
    for a:=1 to m do
    begin
        for b:=1 to 100 do data[b]:=0;
        b:=0;
        repeat
            inc(b);
            read(f1, data[b]);
        until eoln(f1);
        for c:=1 to b-1 do
            for d:=c+1 to b do
            begin
                inc(sum[data[c]]);
                num[data[c], sum[data[c]]]:=data[d];
            end;
        end;
    end;
end;
Procedure Done;
var min, a, b, c, total: integer;
begin
    fillchar(result, sizeof(result), -1);
    result[1]:=0;
    for c:=1 to sum[1] do result[num[1, c]]:=0;
    b:=data[1, 1];
    repeat
        for c:=1 to sum[b] do
            if (result[num[b, c]]=-1) then result[num[b, c]]:=result[b]+1;
        min:=501;
        for c:=1 to n do if (result[c]<>-1) and (result[c]<result[min])
            then min:=c;
        b:=min;
    until result[n]<>-1;
    writeln(result[n]); {到达 S 公园的最少换车次数}
end;
begin
    Init;
end.

```

## 5、最佳游览路线问题

输入数据：输入文件是 INPUT.TXT。文件的第一行是两个整数 M 和 N，之间用一个空格符隔开，M 表示有多少条旅游街（ $1 \leq M \leq 100$ ），N 表示有多少条林荫道（ $1 \leq N \leq 20000$ ）。接下来的 M 行依次给出了由北向南每条旅游街的分值信息。每行有 N-1 个整数，依次表示了自西向东旅游街每一小段的分值。同一行相邻两个数之间用一个空格隔开。

输出文件：输出文件是 OUTPUT.TXT。文件只有一行，是一个整数，表示你的程序找到的最佳浏览路线的总分值。

Program Tour;

var m,n: integer; {M 为旅游街数，N 为林荫道数}

data: array[1..20000] of -100..100; {data 是由相邻两条林荫道所分}

procedure Init; {隔的旅游街的最大分值}

var a,b,c: integer;

f1: text;

begin

assign(f1, 'input.txt');

reset(f1);

read(f1, m, n);

for a:=1 to n-1 do read(f1, data[a]); {读取每一段旅游街的分值}

for a:=2 to m do

for b:=1 to n-1 do

begin

read(f1, c);

if c>data[b] then data[b]:=c; {读取每一段旅游街的分值，并选择}

end; {到目前位置所在列的最大分值记入数组 data}

close(f1);

end;

procedure Done;

var a, sum, result, c: integer;

f2: text;

begin

result:=0;

sum:=0;

a:=0;

while (a<n) do

begin

inc(a); {从数组的第一个数据开始累加，将累加所}

sum:=sum+data[a]; {得到的最大分值记入 result}

if sum>result then result:=sum;

if sum<0 then sum:=0; {若当前累加值为负数，则从当前状态起从新}

```
end;                                     {累加}
assign(f2, 'output.txt');
rewrite(f2);
writeln(f2, result);
close(f2);
end;
begin
  Init;
  Done;
end.
```