# 贪心算法概论

**贪心算法**一般来说是解决“**最优问题**”，具有编程简单、运行效率高、空间复杂度低等特点。是信息学竞赛中的一个有为武器，受到广大同学们的青睐。本讲就贪心算法的特点作些概念上的总结。

## 一、贪心算法与简单枚举和动态规划的运行方式比较

贪心算法一般是求“最优解”这类问题的。最优解问题可描述为：有 n 个输入，它的解是由这 n 个输入的某个子集组成，并且这个子集必须满足事先给定的条件。这个条件称为**约束条件**。而把满足约束条件的子集称为该问题的**可行解**。这些可行解可能有多个。为了衡量可行解的优劣，事先给了一个关于可行解的函数，称为**目标函数**。目标函数最大（或最小）的可行解，称为**最优解**。

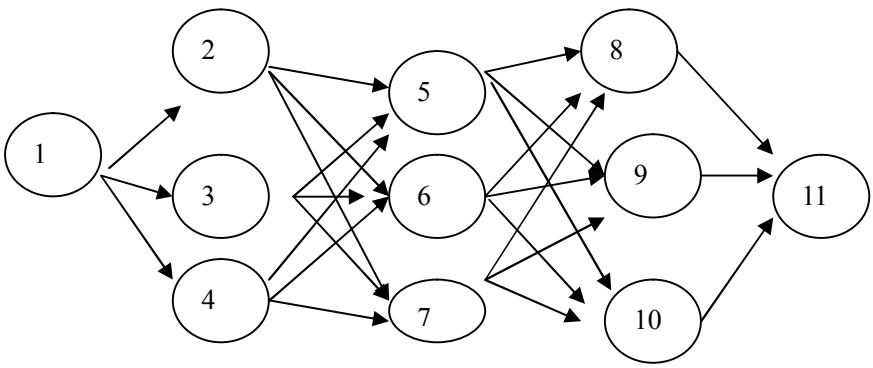
a)求“最优解”最原始的方法为搜索枚举方案法（一般为回溯法）。

除了极简单的问题，一般用深度优先搜索或宽度优先搜索。通常优化方法为利用约束条件进行可行性判断剪枝；或利用目标函数下界（或上界），根据当前最优解进行分枝定界。

b)其次现今竞赛中用的比较普遍的动态规划（需要满足阶段无后效性原则）。

动态规划主要是利用最最优子问题的确定性，从后向前（即从小规模向大规模）得到当前最优策略，从而避免了重复的搜索。

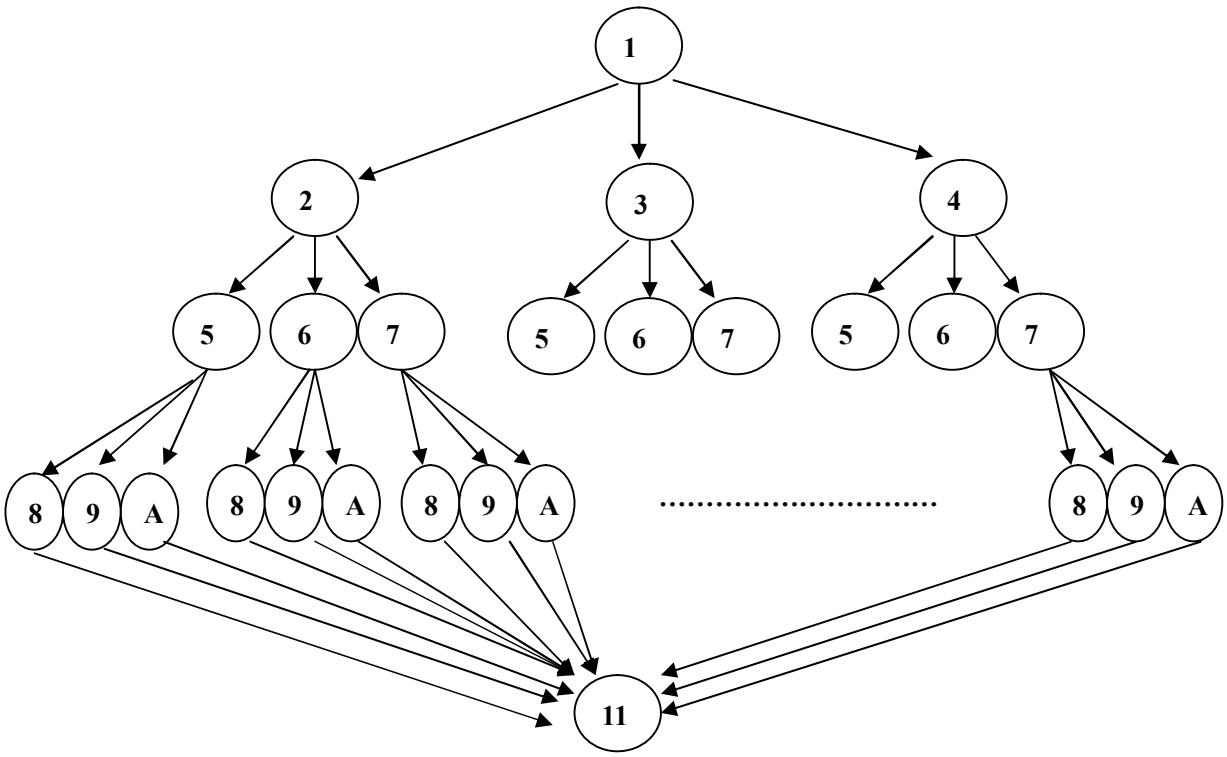
举例说明：求多段图的最短路径。




**图（1）**

在图（1）中，我们省略了各线段的长度。

如果用回溯法，搜索树大致如下：




**图（2）**

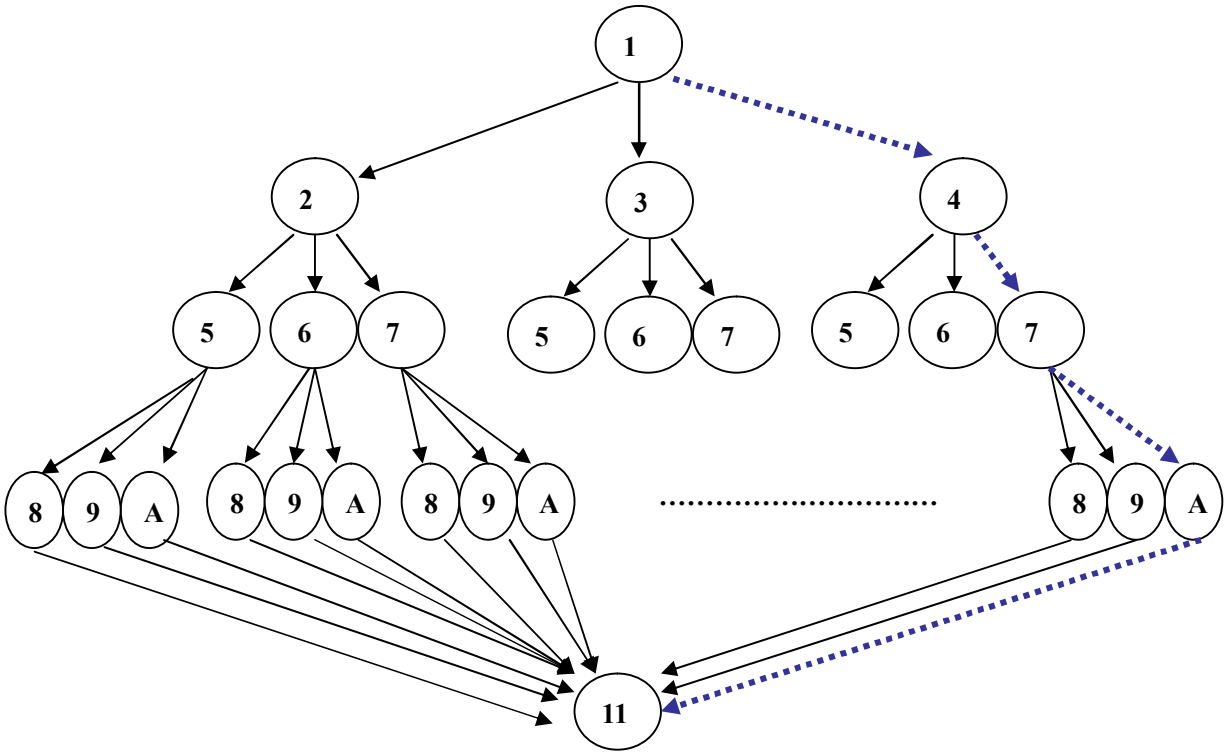
显然，上面的搜索有大量重复性工作。比如节点 8、9、10 到 11 的最短路分别被调用了 9 次，从节点 5、6、7 到节点 11 也分别搜索了 3 次。

如果先算出节点 8、9、10 到 11 的最短路，由于它与前面的点无关，因此最优值确定下来，再用它们求定节点 5、6、7 到节点 11 的最短路径。同理，再用节点 5、6、7 的最优值，来求节点 2、3、4 优值。最后从节点 2、3、4 推出 1 到 11 的最优值。显然复杂度大为降低。

当然，如果本题把简单搜索改为**搜索＋记忆化**的方法，则就是得能动态规划的原理，本质上就是动态规划，只是实现的方法不同与传统的表格操作法。搜索＋记忆化算法有其特有的特点，以后再讨论。

c)贪心算法则不同，它不是建立在枚举方案的基础上的。它从前向后，根据当前情况，“贪心地”决定出下一步，从而一步一步直接走下去，最终得到解。

假如上面的例子中，我们定下这样的贪心策略：节点号 k%3= =1。则有图 3：




图（3）

显然，它只访问了节点 1、4、7、10、11，工作量最少，效率最高。当然，对本题来说，它不能得到最优解－－－最短路径。

从图 3 中可以看出，贪心算法是一种比动态规划更高效的算法。只是要保证得到最优解是贪心算法的关键。

贪心算法的一般框架为：

```
Procedure   Greedy（A，n）    //A 问题有 n 个输入
Begin
   Init(ans)                    //初始化
   for i:=1 to n do
      now=select(A)             //按设计的标准选取一个“节点”
      delete(A,now)             //把 A 中的 now 删掉
      if   can(ans,now)   then unio(ans,now)        //如果可行性通过，放入解中。
   end for
end;
```

# 二、贪心算法的关键－－－－正确性证明

动态规划要满足最优子结构原则，贪心算法呢？就目前的资料看，有个证明模型是“矩阵胚理论”，但要证明一个问题是矩阵胚，十分困难。并且也不常见的可用贪心算法问题都能用矩阵胚来证明。因此，可以认为贪心算法的正确性证明是个难点。因此有些选手在没能证明时，也大胆设想结论应该是正确的。有时这难免会放错。

例如 95 年 NOI 中的“石子合并”一题：

## 例一、石子合并

试题：

在一个圆形操场的四周摆放 N 堆石子(N≤100)，现要将石子有次序地合并成一堆。规定每次只能选相邻的两堆合并成新的一堆，并将新的一堆的石子数，记为该次合并的得分。

编一程序，由文件读入堆数 N 及每堆石子数(≤20)选择一种合并石子的方案，使得做 N-1 次合并，得分的总和最小；选择一种合并石子的方案，使得做 N-1 次合并，得分的总和最大。

例如，图 4 所示的 4 堆石，每堆石子数（从最上面的一堆数起，顺时针数）依次为 4594。则 3 次合并得分总和最小的方案为图 5，得分总和最大的方案为图 6。

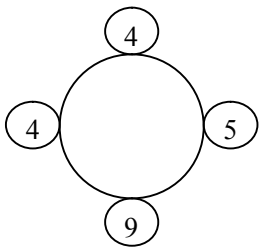
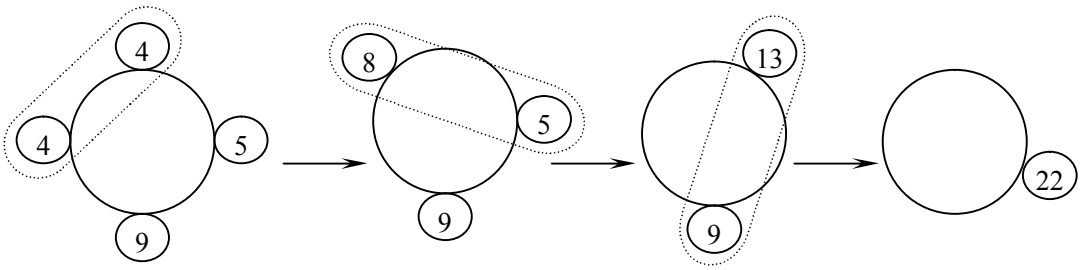
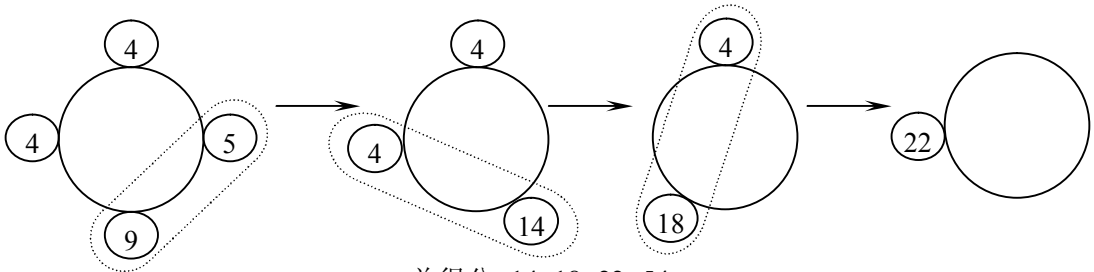



图 4




总得分=8+13+22=43

图 5




总得分=14+18+22=54

图 6

**输入数据**

文件名由键盘输入，该文件内容为：

第 1 行为石子堆数 N；

第 2 行为每堆石子数，每两个数之间用一个空格符分隔。

**输出数据**

输出文件名为 output.txt；

从第 1 至第 N 行为得分最小的合并方案。第 N+1 行是空行。从第 N+2 行到第 2N+1 行是得分最大合并方案。

每种合并方案用 N 行表示，其中第 i 行(1≤i≤N)表示第 i 次合并前各堆石子数(依顺时针次序输出，哪一堆先输出均可)。要求将待合并的两堆石子数以相应的负数表示，以便标识。

**输入输出示例：**

| **INPUT.TXT** | **OUTPUT.TXT** |
|---|---|
| 4 | -4　5　9　-4 |
| 4 5 9 4 | -8　-5　9 |
| | -13　-9 |
| | 22 |
| | 4　-5　-9　4 |
| | 4　-14　-4 |
| | -4 -18 |
| | 22 |

当时这题是第一次出现，几乎有半数以上人都选择了贪心算法－－－每次选相邻和最小的两堆石子合并。

其实由于只能是相邻的两堆石子合并，并不能套用经典的哈夫曼树算法。本题给的样例数据实际上是一个“陷阱”，造成了用贪心法即可解决的假象。

当一个贪心算法不能确定其正确性时，在使用之前，应该努力去证明它的不正确性。而要证明不正确性，一种最简单的形式就是举一个反例。本例的反例见图 7。

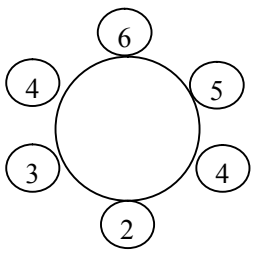



图 7

| 贪心算法的最小值为： | 另一种方法的最小值为： |
|---|---|
| (2＋3)=5 | (2＋4)=6 |
| (4+5)=9 | (3+4)=7 |
| (4+5)=9 | (5+6)=11 |
| (9+6)=15 | (7+6)=13 |
| (15+9)=24 | (11＋13)=24 |
| 5＋9＋9＋15＋24＝62 | 6＋7＋11＋13+24＝61 |

## 常见的证明方法：

贪心算法的正确性证明虽然不容易，但一些常见的方法还是值得总结的。

a) 构造法

根据描述的算法，用贪心的策略，依次构造出一个解，可证明一定是合法的解。即用贪心法找可行解。

b) 反证法

用贪心的策略，依次构造出一个解 S1。假设最优解 S2 不同与 S1，可以证明是矛盾的。从而 S1 就是最优解。

c) 调整法

用贪心的策略，依次构造出一个解 S1。假设最优解 S2 不同与 S1，找出不同之处，在不破坏最优性的前提下，逐步调整 S2，最终使其变为 S1。从而 S1 也是最优解。

下面分别举例说明。

### 例 2 士兵排队问题（1996 年中国队选拔赛）----构造法证明

试题：

有 N 个士兵(1≤N≤26)，编号依次为 A，B，C，...。队列训练时，指挥官要把一些士兵从高到矮依次排成一行，但现在指挥官不能直接获得每个人的身高信息，只能获得“P1 比 P2 高”这样的比较结果(P1，P2∈A，B，C，...，Z，记为 P1>P2)，如“A>B”表示 A 比 B 高。

请编一程序，根据所得到的比较结果求出一种符合条件的排队方案。

*注*：比较结果中没有涉及到的士兵不参加排队

**输入数据**

比较结果从文本文件中读入(文件名由键盘输入)，每个比较结果在文本文件中占一行。

**输出要求**

若输入数据无解，打印“No Answer!”信息，否则从高到矮依次输出每一个士兵的编号，中间无分隔符，并把结果写入文本文件中，文件名由键盘输入。

**输入输出示例**

| INPUT.TXT | OUTPUT.TXT |
|---|---|
| A>B | AFBD |
| B>D | |
| F>D | |

显然用每次选“偏序中最高士兵”的贪心算法就可构造出一个可行解。这个问题转就是“拓扑排序”问题。可这样证明：如果有解，就一定无环，因此每步贪心都一定能进行下去。

经典的“找欧拉回路”问题也类似。

## 例 3 排队问题----反证法证明

**试题：**

在一个超市，有 N 个人排队到一个柜台付款，已知每个人需要处理的时间为 Ti（0<i≤N），请你找一种排列次序，使每个人排队的时间总和最小。

**输入数据**

文本文件中第一行为一个正整数 N≤10000，第二行有 N 个不超过 1000 的正整数 Ti。

**输出要求**

只一个正整数：大家排队时间的最小总和。

**输入输出示例**

| INPUT.TXT | OUTPUT.TXT |
|---|---|
| 4<br>5　10　8　7 | 67 |

本题的贪心算法为：N 个数据从小到大排序，就是这 N 个人的最佳排序方案。求部分和的和即可得到答案。

反证法证明如下：

假设有最优解序列 S1,S2,S3….,Sn，如果 S1 不是最小的 Tmin，不妨设 Sk=Tmin，将 S1 与 Sk 对调。显然对 Sk 之后的人没影响，对 Sk 之前的人等待时间都减少了(S1-Sk)>0。从而新的序列比原最优解序列好，矛盾！

固 S1 为最小时间。

同理可证 S2 是第二小的数，……。

证毕。

下面的调整法也有类似的思想，只是侧重点在“这样调整不会差”。

## 例 4 切巧克力问题----调整法证明

**试题**

我们提供给你一块 *m***n* 的巧克力（如图 8 所示），你需要将这块巧克力切割成 1*1 的**单位巧克力**。

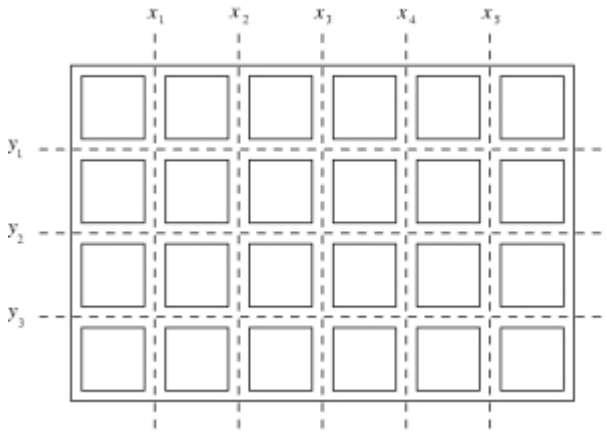



**图 8　巧克力**

具体来说，对于一块巧克力，你有两种切割的方法。一种是沿着某条竖线将巧克力切成两块，另外一种自然是沿着某条横线将巧克力切成两块。切割需要一定的花费，费用的计算与所切割的巧克力的大小没有关系，而只于沿着哪条线切割有关。我们将沿着横线切割的费用定义为 $y_1$、$y_2$、……、$y_{m-1}$；同样的，将沿着竖线切割的费用定义为 $x_1$、$x_2$、……、$x_{n-1}$。现在，需要你计算的就是，怎样切才能**使总费用最少**？

在图 8 中，如果我们先沿着横线将整块巧克力切成四块，然后对于每一小块，分别沿着竖线切割，那么总共的费用就是 $y_1 + y_2 + y_3 + (x_1 + x_2 + x_3 + x_4) \times 4$。

## 【输入格式】

输入文件 chocolate.in 的第一行包含两个整数 $n, m (1 \le n, m \le 1000)$，表示巧克力的大小。接下来的 $n$-1 行每行包含一个整数，分别表示沿竖线切割的费用（上图中 $x_1$，$x_2$，……），接下来的 $m$-1 行也是每行包含一个整数，表示沿横线切割的费用（上图中的 $y_1$，$y_2$，……）

## 【输出格式】

输出文件 chocolate.out 应该包含一个整数，表示你得到的最小切割费用。

| **样例输入** | **样例输出** |
|---|---|
| 6 4<br>2<br>1<br>3<br>1<br>4<br>4<br>1<br>2 | 42 |

分析：

不妨设，Xi 已经从大到小排序；Yi 也同样。

如果把问题简化，只有横的切 Xi 或只有竖的切 Yi，则显然和排队问题是一样的，可以用贪心算法。横、竖可以交错切呢？

对任意一种方案，只看其中横切的 Xi 序列，如果不是从大到小排列，找到第一个违反有序原则的，设为 Xp，找出应该在这个位置的－－即后面不小于 Xp 的 Xq，把 Xp 与 Xq 对调，显然新方案会更好。如此，有

**结论一：**

任意最优方案在不改变竖切次序前提下，可调整成横切是有序的最优解。

同理，有

**结论二：**

任意最优方案在不改变横切次序前提下，可调整成竖切是有序的最优解。

综合结论一和结论二，有

**结论三：**

任意一种最优方案，可调整成横切、竖切都有序的最优解。

如此，可简单得到动态规划算法，时空复杂度约为 $O(10^6)$。

上面虽然用了贪心算法的反证法解决了问题，但没有将贪心算法思想进行到底。我们可以进一步用调整法和反证法的思想，将最优解调整到不分横切、竖切，都是递减的。

简单证明如下：

设有最优方案序列 XY[1..n+m]，序列中任意两相邻项 XY[i]和 XY[i+1]，记为 a 和 b，若 a≤b，分三种情况考虑：

(1) 若 a、b 同是横切，显然交换 a、b 不影响最优解；

(2) 若 a、b 同是竖切，显然交换 a、b 也不影响最优解；

(3) a 和 b 不同，最交换 a、b 后，增加费用 a，减少费用为 b，共增加费用为 (a-b) ≤0，不会破坏最优性。

反复检查，最终调整为全部有序的最优解。

这样最终得到一个简单的算法：只要排序就可以了。

进一步，如果切过的巧克力可分别拿开，各自选方案切割。也可用调整法证明上面的方案也是最优解。

流水线调度问题

由于篇幅关系，后面的两节就简单列个提纲。

## 三、一些经典贪心算法问题

磁带最优存储

背包问题

有限期的作业排序

哈夫曼树

最小生成树---Prim 算法

最小生成树---Kruskla 算法

教室问题

## 四、贪心算法应用的多样性

（1）构造出次序
在众多的选择中，按预先设计的某种次序来确定当前要找的下一步选择。

单源最短路径：Dijkstra 算法
最小生成树

（2）局部（阶段）正确
有时整个问题不能用贪心法，但我们确定了部分因素后，后面的方案就可以用贪心算法了。即通常简称：枚举＋贪心

从汽车过沙漠到登山问题

（3）快速求得一“较好”可行解
前面提到的搜索枚举方案法中，要很多题都可通过贪心算法得到一个“较优可行解”。有时甚至用“启发＋随机化”多次贪心，能得到一个很好的上界（或下界）。

（4）快速分枝定界

接上面话题，在搜索过程中，有时也可以用贪心算法预测已有方案以后可能达到的最小值（或最大值），再用“当前最好解”进行分枝定界。

从原始背包问题到 0/1 背包

（5）对称性游戏

根据问题的特点，直接确定一步，保证次把必败状态留给对方。

圆桌放硬币游戏

取数游戏