部分贪心思想在信息学竞赛中的应用

清华附中 高逸涵

(gaoyihan@gmail.com)

【摘要】在某些数据规模非常大的问题当中，我们常常希望使用贪心法解决问题，但是纯粹的贪心在某些情形下会有反例存在。在这些情况下，我们可以采取一种折中的方案——部分贪心。降问题规模降低到较小的范围内以后，再采用其他方法解决。

【关键字】

部分贪心

【正文】

引言

贪心在信息学竞赛中，是一种非常重要的思想。

一般来说，如果贪心算法可以证明其正确性，那么时间复杂度将远远优于其它算法。

但是某些时候，一些看上去十分正确而且效果明显的贪心，会存在为数不多的一些反例，这时便是部分贪心排上用场的时机，再能保证贪心正确性的前提下，尽量减小待处理问题规模，然后对剩下的小规模问题采用其他方法解决。

正文

在我们详细介绍部分贪心算法之前，首先要知道一些比较基础的东西：

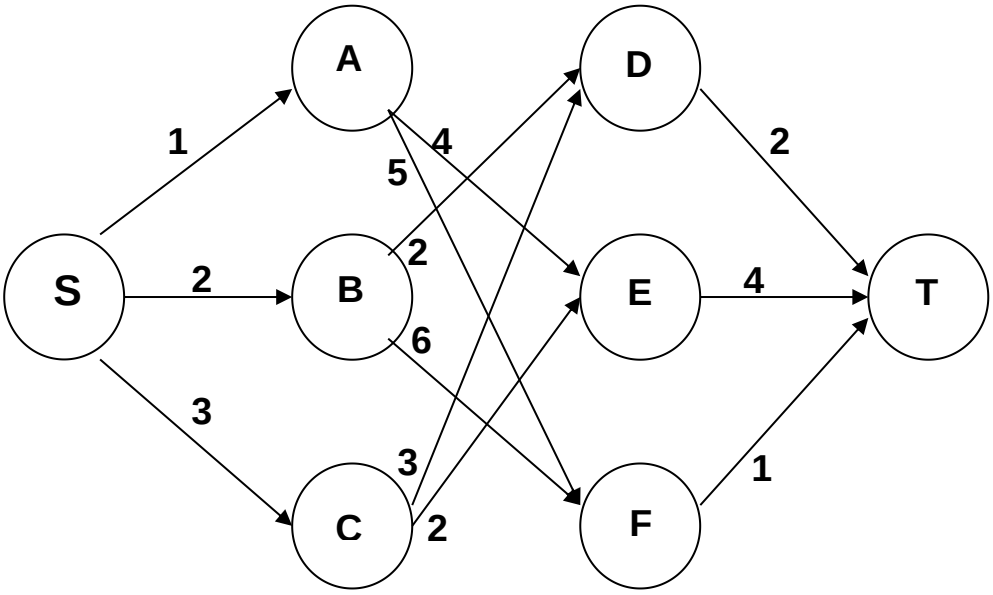
1. 什么是贪心法
2. 贪心法有什么优势和劣势

3. 什么是部分贪心

4. 部分贪心算法有什么优势

什么是贪心算法

贪心算法，顾名思义，就是贪婪地对问题进行决策，在每一个选择面前，寻找当前看起来是最优的一项决策来继续，这样下去，直到达到最终状态，举一个直观的例子。在如下图中寻找从 S 到 T 的最短路，那么贪心法的决策过程会是这样：



1. 首先从 S 找一条最短的路到下一层节点，即 S->A

2. 然后从 A 找一条最短的路到下一层节点，即 A->E

3. 最后从 E 到 T 是唯一的决策。

这样，我们得到的路径为 S->A->E->T，路径长度为 1+4+4=9。

当然，这样做显然是不正确的，事实上，有更短的路径 S->B->D->T，路径长度为 2+2+2=6

贪心法的优势和劣势

一个正确的贪心法有着许许多多的优点，比如说思路直接，程序效率高，代码量小，空间消耗小等等。然而不幸的是，对于大多数问题，我们难以找到一个简单可行的贪心思路，

即使我们找到一个看上去很正确的贪心思路，我们也需要严格的正确性证明来支持这一思路。这往往给我们直接使用贪心算法带来了巨大的困难。

什么是部分贪心

很多时候，当我们试图使用贪心算法解决问题，却发现存在种种特殊情况，使得我们的贪心算法会产生一些小的错误，这些特殊情况的普遍特点是，数据规模很小，而我们的贪心思路往往是针对大局设计的，而这些细节上的特殊情况往往会使我们感到无从下手。

在这种情况下，我们可以考虑一种特殊贪心方式，在接近初始状态或者目标状态的决策中采用搜索或者动态规划这类可以保证正确性的算法来处理，而对于中间的状态则采用贪心思想解决。这样平衡了算法的效率和正确性，得到了一个相对理想的结果，这种算法便是部分贪心算法。

举一个例子：求函数 $f(x)=x^2+\sin(x+\varphi)$ 的最小值，从总体上来看，这个函数的取值主要和 x^2 的取值有关，而 $\sin(x+\varphi)$ 则起小规模干扰作用，但是如果单纯的取 x^2 取最小值的位置却不能得到整个函数的最小值。事实上，取最小值的位置一定在 x=0 的附近位置。这样，我们贪心地得到了一个最小值 x=0，然后在其附近采用枚举法取得真正的最小值。这便是部分贪心算法的一个形象化的例子。

部分贪心算法的优势

部分贪心在不影响算法总体复杂度的前提下，将边界上的特殊情况交给一些可以容易保证正确性的算法解决。可以视为对贪心算法的一个改进和推广。

如果说，贪心法的优势是效率高，缺点是应用范围狭小，普通算法应用范围广但是效率低下，那么部分贪心则是综合了两者，使得在正确性和算法效率之间得到了一个平衡。

一般来说，信息学竞赛需要在规定的时间限制下解决规定的题目，需要算法有很好的正确性和不错的效率。这样，部分贪心算法就是唯一能够在时间效率和正确性的双重限制下较

好的解决问题的优秀算法。

应用实例及结果

这里我们通过三个例题具体分析来讨论部分贪心算法的具体应用形式。

例一、骆驼

题目大意：

有 *p* 个人带着 *x* 个小包和 *y* 个大包准备穿过沙漠。所有人都不愿意徒步旅行，因此需要一些骆驼把他们自己和所有大包小包驮在背上。每匹骆驼可以背的物体只能是下列四种组合之一（因此不能同时背小包和大包）：不超过 3 个小包；不超过 2 个大包；1 个人与不超过 2 个小包；1 个人和 1 个大包。最少需要多少骆驼？

问题规模：

1<=*p*<=1000000000, 0<=*x*,*y*<=1000000000

初步分析：

当所有人所带的包的种类确定以后，剩下需要的骆驼数目可以直接算出来，所以我们需要求的只是有多少个人带大包，多少个人带小包。

所以可以很容易得到公式：

$$ans = \min\{\left\lfloor \frac{\max(x-2i,0)}{3} \right\rfloor + \left\lfloor \frac{\max(y-p+i,0)}{2} \right\rfloor + p, 0 \le i \le p\}$$

由于数据规模较大，所以枚举法显然会超时。而直接利用数学公式计算，则由于取整运算符的存在而显得略微有些麻烦，我们考虑用部分贪心法解决该问题。

采用部分贪心法：

我们换一个角度思考问题：

1. 当一个人带着小包的时候，一个人带 2 个小包，而一个不带人的骆驼可以带 3 个小包，这样相当于把人当成小包来算。
2. 当一个人带着大包的时候，一个人带 1 个大包，而一个不带人的骆驼可以带 2 个大包，这样相当于把人当成大包来算。

显然，一般来说，把人当成小包来算更合适一些。这一贪心思想是我们采用部分贪心法的前提。我们直觉得可以感觉到，如果小包和人的数目都比较大，那么让人带着小包应该是正确的选择。

事实上，如果剩下的小包的个数大于 6，并且人还剩 2 个以上，那么上述贪心肯定是正确的，这时如果不让人带着小包，即让人全部带着大包，那么我们可以进行如下调整，使解至少不会变差：找一个带着 3 个小包的骆驼，以及两个带着人和一个大包的骆驼，改变骆驼的携带物为两个带着小包和人的骆驼加上一个带着只带大包的骆驼，这样我们不仅完成了原先的任务，还额外多带了一个小包。

由此可以得到算法：

1. 如果小包的个数>20，且人的个数>5，那么让一个骆驼带着人和小包。[1]
2. 否则枚举人带小包的骆驼数，这个数目不会大于小包个数的一半，也不大于人的个数，所以这个数目小于等于 10。这个时候只需枚举即可。

这样得到的代码非常简短：

```
int solve( int p, int x, int y )
{
    if ( x < 0 ) x = 0;
    if ( y < 0 ) y = 0;
```

[1] 为了避免边界上可能存在的错误或者讨论，我们常常加大数据，这是信息学竞赛中的常用手法。

```
    if ( p == 0 ) return ( x + 2 ) / 3 + ( y + 1 ) / 2;
    if ( p <= 10 || x <= 40 )
    {
        int tmax = 0x7fffffff, tr = min( ( x + 1 ) / 2, p );
        for (int i = 0; i <= tr; i++ )
        {
            int t = solve( 0, x - 2 * i, y - ( p - i ) );
            if ( t < tmax ) tmax = t;
        }
        return tmax + p;
    }
    else
    {
        int t = min( p - 5, ( x - 20 ) / 2 );
        return solve( p - t, x - t * 2, y ) + t;
    }
}
```

注：

1. 对于这类数学讨论问题，部分贪心是很常见的一种处理方法。在 ICPC 和 topcoder 这类比较注重做题速度以及程序正确性的比赛当中经常可以看到有人使用该方法。
2. 采用该方法可以将这个题目很容易的扩展到多个物品的情形，只需将题目最后的枚举改为动态规划即可。

算法回顾：

本题是一个非常经典的应用部分贪心算法的例子。具有以下几个特点：题目限制小，数据范围大，贪心意图明显，在数据规模较小的时候会出现贪心的反例。相信这个题目能给那些对为何要采用部分贪心有疑惑的读者一些感性的认识。

例二、Huge Knapsack(SPOJ 699 HKNAP)

题目大意：

你来到一个地方，这个地方有 N 种纯金雕像，每种雕像都有无数个，每种雕像有特定的体积 V[i]和重量 W[i]，你有 S 个包，每个容量为 Y，每次可以花费 C 的代价将两个包合并成 1 个包，即合并 3 个包为 1 个包需要 2*C 的代价，问最后可以得到的金子的数目减去合并包的代价最大是多少？

数据规模：

1<= S <= 1000 000 000

1<= Y <= 1000 000 000

1<= N <= 1000

1<= W[i] <= 100;

1<= V[i] <= 18;

初步想法：

首先对于所有的体积 V[i]，保留最大的重量。这样 N 的范围可以缩小至 18。

这是一个很经典的背包问题，但由于背包容量很大，所以不可能直接使用动态规划算法。对于大多数情况，当 Y 大到一定程度的时候，我们所选取的雕像一定是取其中某个填充背包的主要部分，剩下的部分用其他雕像填满。这一直观的印象启示我们采用部分贪心法。

算法分析：

首先我们计算出每种雕像的性价比：$\frac{W[i]}{V[i]}$，然后选取其中最大的作为主要填充物。

然后我们需要求出一个常数 Limit，当背包空间大于 Limit 时，我们一定可以继续填充主要填充物。一个显而易见的想法是，其他每个物品的个数一定<V[i]（否则可以将 V[i]个该物品用主要填充物替代），所以 Limit 可以为 N*V[i]。这里的 Limit<324。再用动态规划算法，时间上完全可以接受。

这样做以后，我们可以用 O(1)的时间查询背包容量为 k 的最大重量，这个最大重量的表达式为：

$$weight = tW_i + knapsack[k - tV_i](t = \left\lfloor \frac{k - Limit}{V_i} \right\rfloor)$$

问题分析至此，已经解决了一半。由于 S 的范围很大，我们只有继续采用部分贪心解决问题。模 Limit 的剩余系一共只有 Limit 个，所以我们可以很容易得到一个有用的事实：我们不必将 Limit 以上个包合并在一起。这是因为如果将 A 个包合并以后得到的 $k - tV_i$ 和 B 的大小相同（A>B），那么考虑 A 中的 A-B 个包，其一定被主要填充物充满，则 A-B 个包和剩下的 B 个包可以不必合并。用数论知识很容易得到，Limit 以上个包的合并没有必要。

于是我们希望求解的问题可以视为：有一个 S 容量的背包，以及至多 Limit 个物品，则总重量最大为多少。可以用和前面相同的算法解决。

问题至此已完美解决。

算法回顾：

在上述问题的解决过程中，我们进行了两次局部贪心，分别将 Y 和 S 降低到了可以承受的复杂度范围内，从而解决了问题。将某一项输入的规模降低，这是部分贪心法最主要的用途。

另外可以看到的是，纯粹的贪心虽然有一定道理，但在小规模数据则不一定正确，动态

规划算法虽然可以保证正确性，但是时间复杂度太高。而部分贪心法成功的融汇了两个算法的优势，成功的解决了问题。事实上，我们可以将部分贪心法视为贪心法与其他算法和并使用的一种技巧。

例三、Cow Relays(USACO Gold Nov.)

题目大意：

在一个无向图中有 T 条边，每个点至少和两条边相连，每条边有一个长度，询问从给定起点到给定终点的包含 N 条边的路最短是多长。

问题规模：

1<=*N*<=1000000000, 1<=T<=100

算法分析：

首先可以确定的是，这个图里面最多有 T 个点。

图很小，N 很大，这一反常现象使我们很容易联想到部分贪心。

首先，我们直观的感觉到，要想让包含 N 条边的路最短，必定要在一个较短的边上来回走几乎 N 次，我们首先要证明的一件事是，我们只可能在某一条边上来回走很多次，而非两条边或者两条边以上。

这个事情事实上很好证明，考虑所求路径以及其上最短的一条边。如果在该路径上其他边有来回走多次的现象，那么删去这一对来回走的有向边，改为在最短的一条边上来回多走一次。则新的图依然是从起点到终点的一条路径。以上证明还排除了在某一条边上来回走一次以外又从这条边走了第三次的情况。

其次，我们要证明某一条边上不会在同一方向连续走 3 次。假设这种情况发生了，那么我们一定可以得到从出发点开始的两个环。如果其中任何一个的长度是偶数，那么去掉这

个环，改为在最短边上来回走，所得路径长度不变大，否则两个环的长度都是奇数，则同时去除这两个环，改为在最短边上来回走，所得路径长度不变大。

综上，对于除最短边以外的其他边，我们至多在上面经过 2 次，所以我们在非最短边外的边上至多经过了 2*T 次。这样采用动态规划的方法可以在 $O(T^2)$ 的复杂度内算出起点和终点到每个点经过 k(0<=k<=2*T)条边的最短长度。然后通过枚举每条边取最小值即可。

官方解答采用倍增的思想，在 $O(T^3 \log N)$ 的复杂度内解决了本题，但事实上，如果采用部分贪心思想，可以得到复杂度为 $O(T^2)$ 的算法，远比标准算法要好。

【结果】

以上 3 个例题，各自从不同的角度展现了部分贪心算法的具体应用形式。

第一题，我们利用部分贪心算法，巧妙的缓和了高效的贪心算法和繁多的特殊情况之间矛盾，完美的解决了问题。

第二题，利用部分贪心算法，我们压缩了题目中两个相对较大的数据范围，使得算法的整体复杂度处在一个可接受的范围内。

第三题，则是一个比较综合的题目，利用部分贪心算法，我们不但避免了小规模数据的特殊情况讨论，还压缩了一个题目中非常巨大的数据规模 N，可以说是一个利用部分贪心算法非常巧妙的例子。

【结论】

部分贪心法从本质上来看可以认为是贪心法与其他算法的结合，其具有贪心法思维直观，思路清晰，编程简单，程序效率高等优点，同时能解决一些贪心法所不能直接解决的问题。以上种种优点，使得部分贪心法可以在算法之林中占有一席之地。

部分贪心法作为一种非正统算法，往往为人所难以接受，因为它看起来并不十分优美，并不是一种“纯粹”的算法。但无可争议的是，他在某些问题上有着别的正统算法所不能比

拟的巨大优势。

当我们遇到正统算法难以解决或解决起来很麻烦的情况时，不妨考虑一下部分贪心法，或许它可以给你意外的惊喜。

【参考文献】

刘汝佳，黄亮《算法艺术与信息学竞赛》 清华大学出版社

【附件】

例三《Cow Relays》一题的源程序 

Relays.cpp