部分贪心在信息学竞赛中的应用

北京市清华附中 高逸涵

引言

- 引入
  - 众所周知，贪心算法是一个在信息学竞赛中应用广泛的高效算法。
  - 但是有的时候，由于小规模针对性数据的存在，使得贪心算法
  - 如何解决这
- 部分贪心，顾名思义，就是在问题的局部采用贪心算法，而在其他部分采用其他算法。

部分贪心

引言

- 为什么要“部分”贪心?
  - 当问题的特殊情况普遍较小的时候，对于边界数据采用其他算法处理可以有效的回避特殊情况的讨论。
  - 部分的普通算法对于总体时间复杂度影响并不大。
  - 部分的贪心可以极大的提高算法的时间效率。

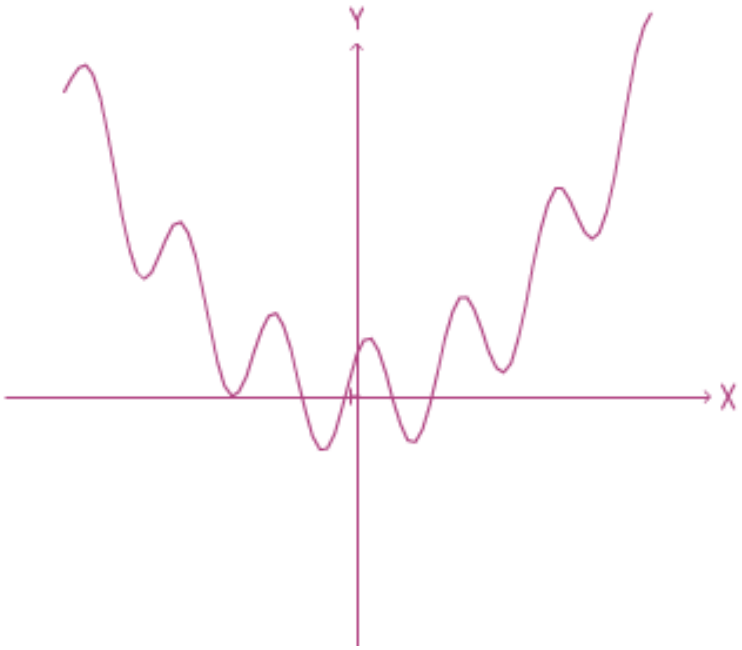
引言

- 举个例子：我们要最优化目标函数 $f(x)$

$$f(x) = g(x) + h(x)$$

$$g(x) = x^2$$

$$h(x) = 2\sin(5x + \frac{5\pi}{4})$$



为了求得目标函数的最小值，我们可以首先贪心的求出趋势函数的最小值，然后在其左右寻找目标函数的最小值。

例题

- [例题 1] 骆驼
- [例题 2]Cow Relays

[例题 1] 骆驼

- 旨 x 个小包 y 个大包穿越沙漠
- 必背的物体只能是下列四种组合之
  3 小包；不超过 2 个大包； 1 个
  2 小包， 1 个人和 1 个大包。

- 问要多 骆驼?
- 数

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

[例题 1] 骆驼

- 首先，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。
- 很容易得到如下公式： (p,x,y 分别为人，小包，大包数)

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

- 但由于数据规模巨大，直接枚举显然行不通，需要另想办法。

[例题 1] 骆驼

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

- 由于取整运算符的存在，导致直接数学计算变得比较困难。
- 但是从整体趋势上来看， i 的增加显然有利于 ans 的减小。
- 那么按照贪心思想，我们需要尽量让人带着小包。

[例题 1] 骆驼

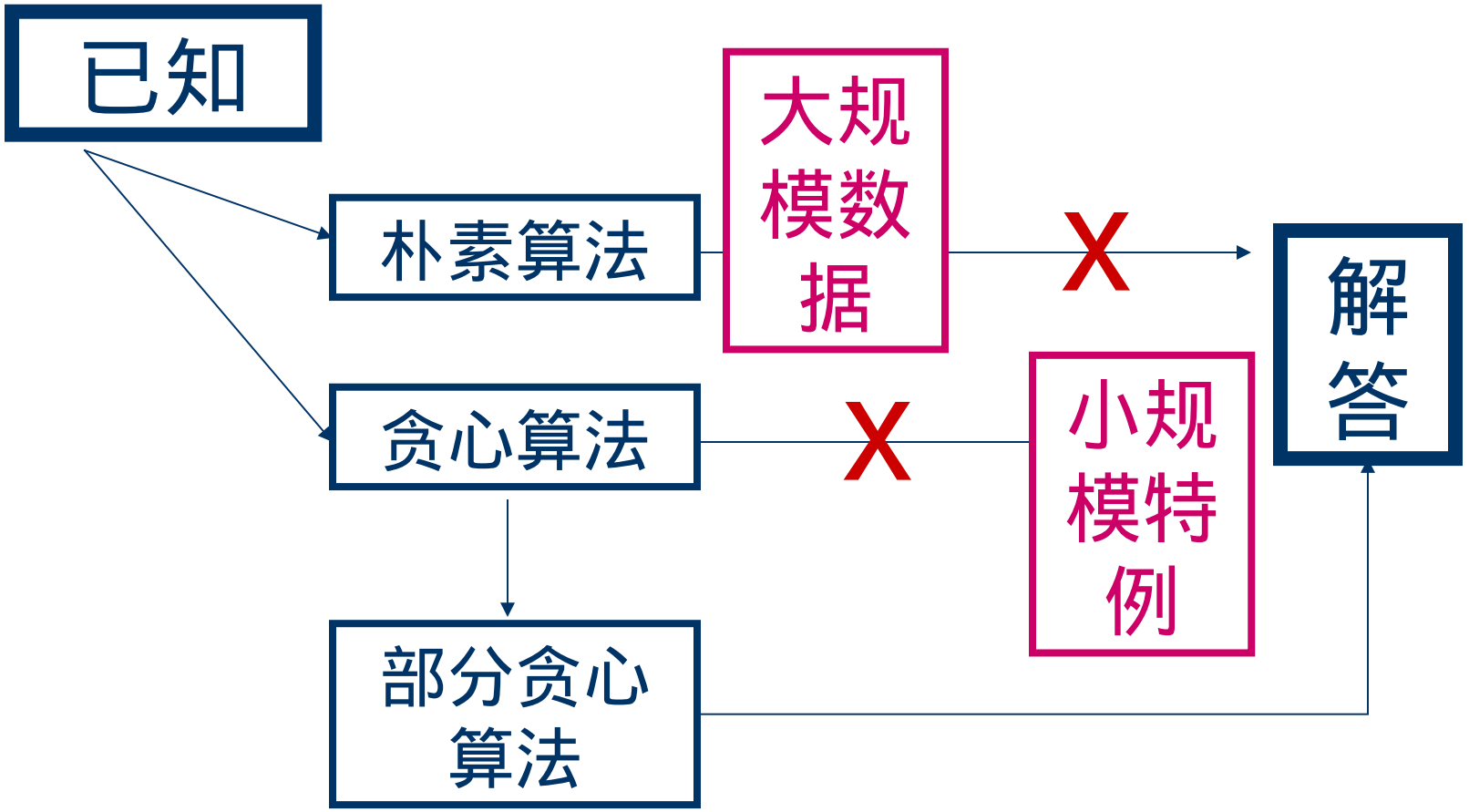
- 我们很容易得到一个贪心算法：如果当前小包的个数 >=2 并且还有大包的，那么令这个人带着小包，否则令这个人带大包。
- 很不幸，这个算法是错误的 (x=3,y=3,p=1) 。
- 如何改进?

正确性？

[例题 1] 骆驼

- 当人数和小包的数量都充分大的时候，令人带小包显然是没错的。
- 经过验证，人数和小包数 >=20 的时候，一定存在一个最优解使得存在一个骆驼带着人和小包。
- 算法的正确性采用调整法很容易证明。
- 而当人数和小包数有一个小于 20 时，可以采用枚举法解决问题。

[例题 1] 骆驼



[例题 2]Cow Relays

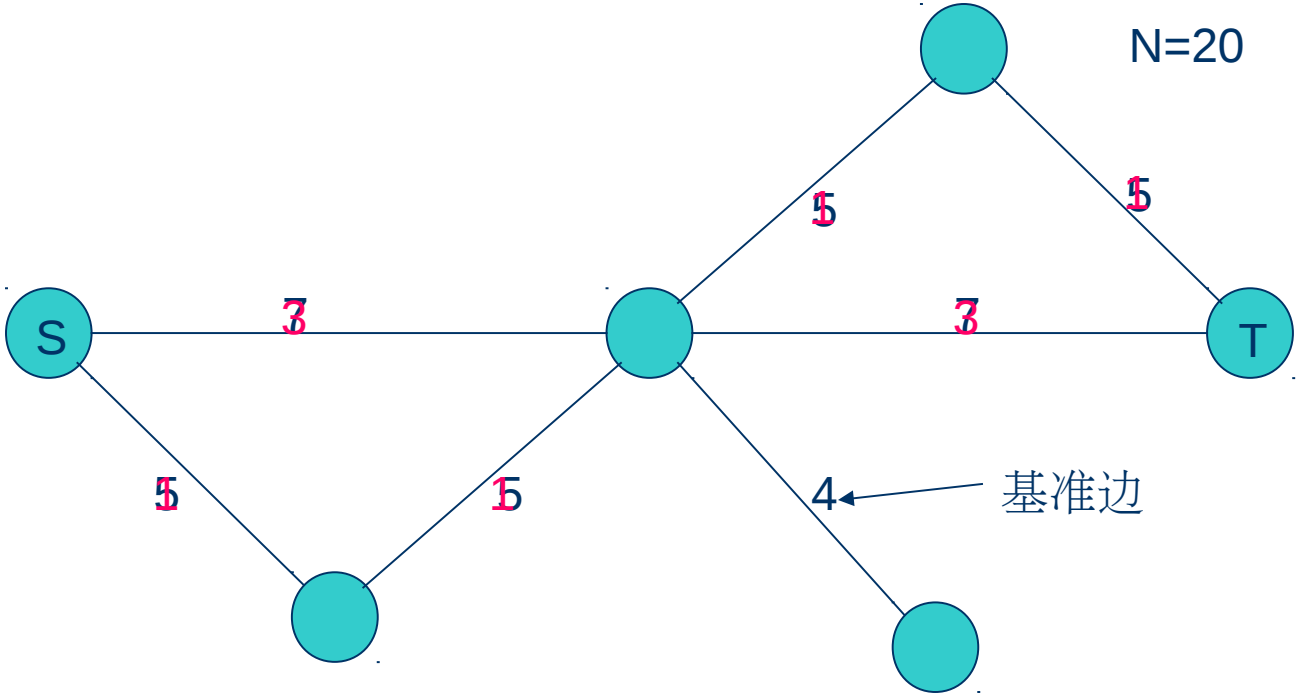
- 在一个无向图中有 T 条边，每个点至少和两条边相连，每条边有一个长度，询问从给定起点到给定终点的包含 N 条边的路最短是多长。
- 数据规模： 1<=*N*<=1000000000, 1<=T<=100

[例题 2]Cow Relays

- 首先看到这一题目，我们的直观感受是，最优解一定是这样的一条路经:
  - 首先从起点运动到某一个点上。
  - 然后在这个点所连接的最小边上往复运动。
  - 最后从这个点直接运动到终点。
- 针对这一思想，我们很容易设计出一个贪心算法——枚举一条边做往复运动，然后从起点和终点分别向这条边走增量最短的路径到达。

[例题 2]Cow Relays

- 所谓增量最短路径，就是将所有边减去基准边之后得到的新图内的最短路径。

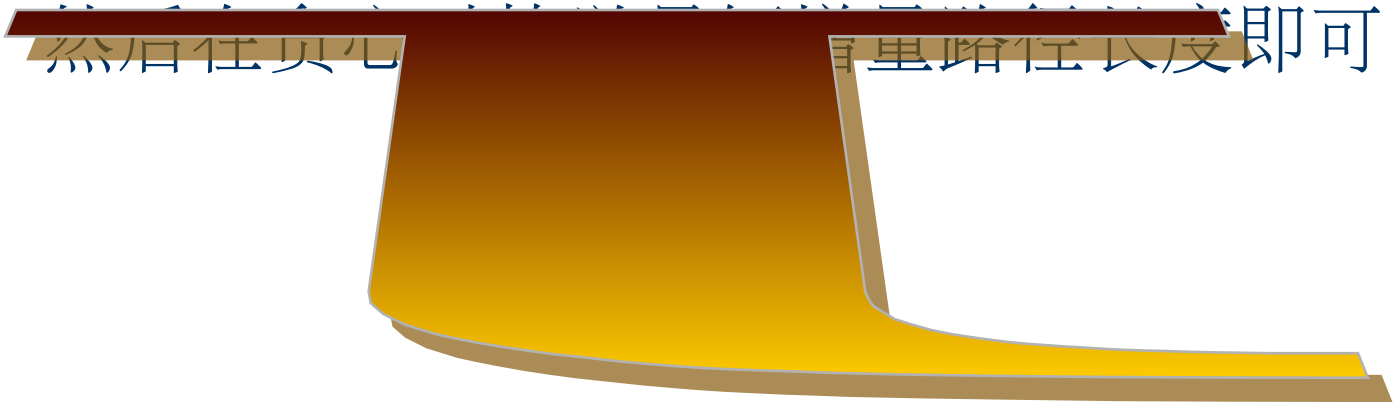


[例题 2]Cow Relays

- 这样的贪心算法的复杂度为 $O(T^3)$，但是运用部分贪心算法避免重复计算，可以将复杂度进一步降为 $O(T^2)$
- 算法瓶颈在于对于每条边，我们都要求一次最短路，我们希望在一次中解决所有最短路问题。

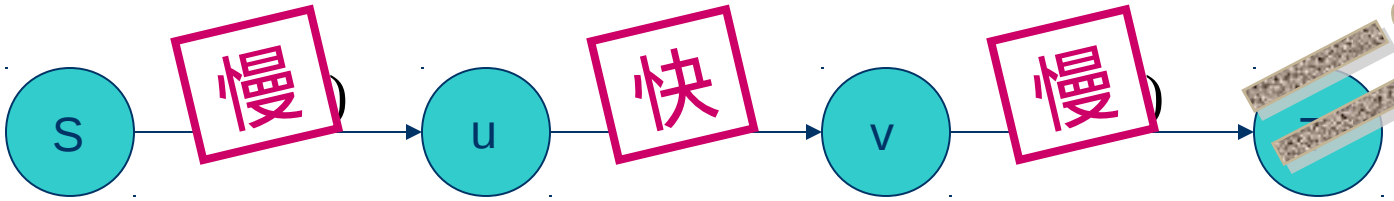
[例题 2]Cow Relays

- 回顾我们求最短增量路径的过程，显然，我们所求的最短增量路径一定是在边数确定时的最短路径。
- 因此，我们只需要用动态规划预处理出源点到每一个点所走边数一定时所得的最短路径的长度，然后在贝[illegible]量路径长度即可。

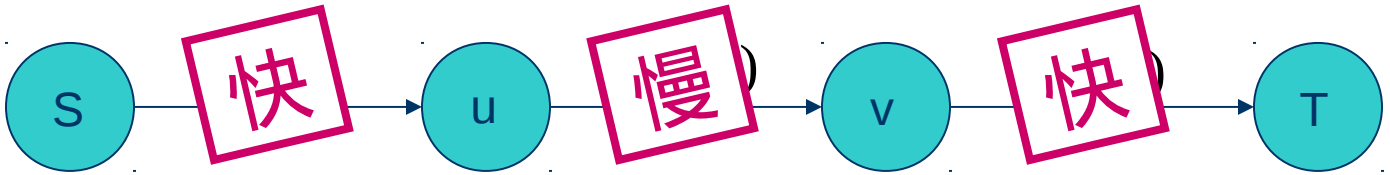


[例题 2]Cow Relays

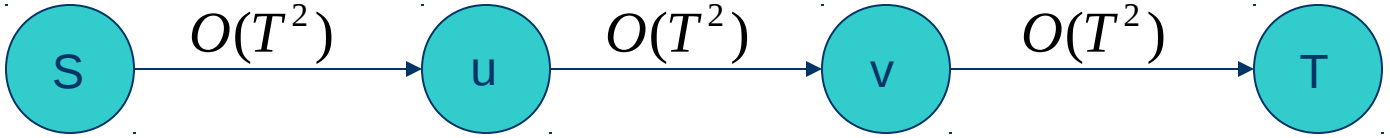
- 贪心算法：



- 朴素动态规划：



- 部分贪心：



结果

- [例题 1] 骆驼

	时间复杂度	效果
贪心算法	O(1)	Wrong Answer
朴素算法	O(N)	Time Limit Exceed
部分贪心算法	O(1)	Accepted

结果

- [例题 2]Cow Relays

	时间复杂度	效果
倍增算法	$O(T^3 \log N)$	较慢
贪心算法	$O(T^3)$	较慢
朴素算法	$O(NT)$	很慢
部分贪心算法	$O(T^2)$	很快

总结

- 朴素算法 —— 思路简单，算法低效，不易出错
- 贪心算法 —— 思路复杂，算法高效，易出错

- 部分贪心 —— 思路简单，算法高效，不易出错


Thank You !