



中国计算机学会
China Computer Federation



2022 全国青少年信息学奥林匹克冬令营

信息学竞赛中的直觉与证明

刘汝佳
曹钦翔

v18



中国计算机学会
China Computer Federation



上半部分 以信息学竞赛题目为例



中国计算机学会
China Computer Federation



nature

[Explore content](#) ▾

[About the journal](#) ▾

[Publish with us](#) ▾

[nature](#) > [articles](#) > [article](#)

Article | [Open Access](#) | [Published: 01 December 2021](#)

Advancing mathematics by guiding human intuition with AI

[Alex Davies](#) ✉, [Petar Veličković](#), [Lars Buesing](#), [Sam Blackwell](#), [Daniel Zheng](#), [Nenad Tomašev](#), [Richard Tanburn](#), [Peter Battaglia](#), [Charles Blundell](#), [András Juhász](#), [Marc Lackenby](#), [Geordie Williamson](#), [Demis Hassabis](#) & [Pushmeet Kohli](#) ✉

[Nature](#) **600**, 70–74 (2021) | [Cite this article](#)

2 Citations | **1598** Altmetric | [Metrics](#)



不是故意的

- 冬令营讲课的选题是 11 月定下来的
- 这篇文章是 12 月 1 日发表的，我是过了一段时间才知道的
- 但是我觉得挺有意思，所以加上来和大家分享
- 至少说明直觉对数学的发展挺有用的，不管是纯自发的还是有 AI 做**辅助**。注意是辅助，不要被误导了
- <https://www.nature.com/articles/s41586-021-04086-x>



委婉的吐个槽

- 现在网络发达，很多题目都能在网上找到题解，可惜很多题解在看完以后通常有以下疑问：
 - 这个是怎么想到的？
 - 这样子真的是对的？是不是数据弱？
- 分享精神值得肯定，如果题解质量能提高就更好了



本课目的

- 当然，很多人写题解主要是记录给自己看的，记录的重点是“这题怎么做”，而不是思考过程以及正确性证明。而且很多时候，这个解法也不是作者想到的，或者也不会证，自然就没法写了
- 本课堂的目的之一就是让大家重视、学会思考和证明，并能写出来分享，所以讲题的重点不是能 AC，而是学明白



如何思考？

- 虽然有很多方法论，但很多时候都是凭**直觉**
- 见过类似的题，或者仅仅是某个雷同的特殊条件，或者某大类题目的常见解法，这些都属于**套路**。见多了，就形成了**直觉**，还没深入思考就直接“想到了”
- 有些方法是**通用**的，可以一个一个试
- 还可以先动手实践，从中受到**启发**
- 做完题之后尽量**泛化**，看看方法的**适用范围**



泛化

- dijkstra 算法的适用条件?
- 从“模拟费用流”这类题目受到什么启发?
- 高斯消元适用于什么代数系统?
- 满足什么性质的东西可以存到线段树里维护?
- 算法证明的推导过程往往能看出算法的适用条件，关注这些适用条件，可以让你更敏锐，在 Mind Palace 里遨游的时候能帮助你快速排出错误选项，更快找到答案



中国计算机学会
China Computer Federation



第一部分 引例、直觉和证明



直觉和证明

- 有时候直觉会不准，甚至**误导**你
- 直觉可以引导证明
- 证明失败也没关系，如果找到**反例**，可能会进一步产生新的直觉
- 下面举一些普通人的直觉可能不准的例子



“欢迎尝试，保证解存在
可以使用任何工具包括计算机编程”

95%的人解不出这道题！

$$\frac{\text{Apple}}{\text{Banana} + \text{Pineapple}} + \frac{\text{Banana}}{\text{Pineapple} + \text{Apple}} + \frac{\text{Pineapple}}{\text{Apple} + \text{Banana}} = 4$$

你能找到 , , 和 
的正整数解吗？



最小解

- a=1544768021087461664419513150199198374856
64325669565431700026634898253202035277999
- b=3687513179412999982719781156522547482549
2979968971970996283137471637224634055579
- c=4373612677928697257861252602371390152816
537558161613618621437993378423467772036



有兴趣的话

- 椭圆曲线入门
- [https://www.quora.com/How-do-you-find-the-positive-integer-solutions-to-frac{x-y+z}{-frac{y-z+x}{-frac{z-x+y}{4}}}](https://www.quora.com/How-do-you-find-the-positive-integer-solutions-to-frac{x-y+z}{-frac{y-z+x}{-frac{z-x+y}{4}})



用 1×2 骨牌铺 $m \times n$ 棋盘

- 铺 $2 \times n$ 的方案数大家都知道是 Fibonacci 数
- 然而，对于一般情况， $m \times n$ 棋盘（假定 m 是偶数），有公式吗？有是有，但是长这样：

$$\prod_{k=1}^{\frac{1}{2}m} \prod_{l=1}^n 2 \sqrt{\cos^2 \frac{k\pi}{m+1} + \cos^2 \frac{l\pi}{n+1}}$$



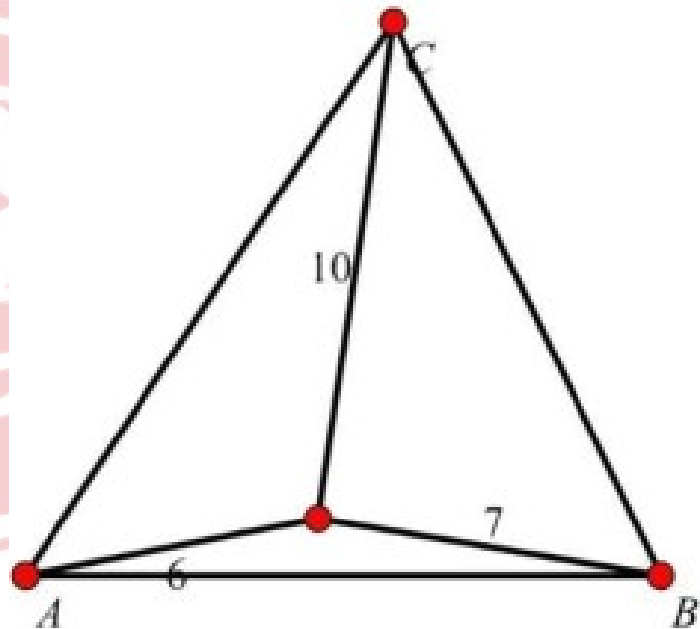
有兴趣的话

- 证明不是特别长（8页），不过有点绕，也不觉得优美
- <http://math.uchicago.edu/~may/REU2015/REUPapers/Borys.pdf>



最后来个网红数学题

- 角 ACB 为 60 度, $PA=6$, $PB=7$, $PC=10$, 求三角形 ABC 面积的最大值





有兴趣的话

- 答案是 $23*\sqrt{3}+5*\sqrt{39}$
- 单增老师《平面几何的知识与问题》第四章的第57题



引例 1 部分背包问题

- 有若干物体，每个物体有重量和价值
- 有一个背包，重量有限制
- 每个物体可以只带走一部分（比如液体），重量和价值都按比例计算
- 要求在最重量不超过限制的情况下总价值最大



直觉

- 先拿重量小的？可以多装点
 - 试着找个反例？（找反例是个好习惯）
- 先拿价值高的？毕竟目标是总价值尽量大
 - 再试着找个反例？
- 先拿平均价值（价值 / 重量）高的？



算法

- 事先排序好
- 先拿第一种，一次拿一丢丢
- 什么时候拿不了更多呢？
 - 第一种拿完了，或者
 - 背包装满了



证明思路

- 证明在任何时候换一个决策都不会有更好的结果（顶多同样好）
- 刚才的算法中，如果第一种还没拿完就拿一丢丢第二种，会怎样？
 - 我们先把这个“支线”策略执行完毕，得到一个完整的方案。讨论这个完整方案
 - 如果这个方案始终没有拿完第一种，把这一丢丢换成第一种，重量不变，价值变大，这个方案不是最优的！这个叫 exchange argument
 - 如果这个方案后来又继续拿第一种了，那么改成早点拿，同样好。这叫做把决策规范化 (canonicalize)



总结

- 找反例是个好习惯，往往可以比较精准的知道一个错误算法的缺陷究竟在哪里，然后加以改进
- 如果决策是一个排列（或者类似的东西），exchange argument 往往可以奏效。并且它的应用范围不止于此，比如我们熟知的 prim/kruskal 算法
- 规范化是个好东西，有时可以同时简化思路、证明和代码



引例 2. 田忌赛马 (ACM 上海 2004)

- 田忌赛马问题，赢一局赚 200，输一局赔 200，平局不赚不赔
- 第一行是田忌的 n 匹马的速度
- 第二行是齐王的 n 匹马的速度
- $n \leq 1000$



直觉

- 历史告诉我们：
 - 用最慢的马对齐王最快的马，划算
 - 用一匹马对齐王的一匹“只慢一丢丢”的马，也划算
- 这两个规则有冲突吗？



直觉好使不

- 题目没说完全符合历史
- 所以田忌的马有可能比齐王还要快 ...
- 非要拿最慢的马和齐王最快的比，有可能失去了“全胜”的机会！
- 不过不管怎么说，先排个序，田忌和齐王的马都按照速度降序排列，比较方便



最慢的马

- 如果你最慢的马比齐王最慢的马慢，那么索性和齐王最快的马比，不吃亏（这也是个 `exch...`）
- 如果你最慢的马快一点呢？根据前面的结论，选一个能战胜的最强马？当然也可以，不过没必要，因为当前考虑的是你最慢的马，现在能战胜的马，之后更是随便虐。所以直接选最慢的马比就行了，程序最简单



中国计算机学会
China Computer Federation



反过来考虑?

- 先考虑快的，会如何?
- 对称吗?
- 留给同学们思考



引例 3. 三值排序 (IOI96)

- 给一个只包含 1, 2, 3 的数组排序，每次可以交换任意位置的两个数，最少需要几次？
- 有 $n \leq 1000$ 个数
- 原题的第二问要输出方案，我们这里只做第一问（第一问做出来之后第二问也不难，但是和本课程关系不大所以略）



直觉？

- 虽然有很多个数，但是全是 1, 2, 3，感觉可以特殊处理
- 但是如何处理呢？好像没啥思路
- 可以试试具体解决一个问题，从实践中找灵感，再试着举一反三



中国计算机学会
China Computer Federation



实践

- 1 1 1
- 1 2 3
- 3 2 1
- 1 3 2
- 2 1 3
- 2 3 1
- 3 1 2





继续实践

- n 很小的情况常常有价值，比如一些特殊的模式 (pattern) 或者可以构建复杂解的积木 (building block)
- 但是稍微大一点的例子往往也是必须的，比如：

2 3 1 1 2 1 3 2 3

1 1 1 2 2 2 3 3 3



1的家 2的家 3的家
直觉

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 1 | 1 | 2 | 1 | 3 | 2 | 3 |
| 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 |

- 目标是让每个数“回家”
- 一开始已经到家的，就不用管了
- 1在2家 (1@2), 2在1家 (2@1), 直接调换一举两得



直觉

- 如果上述情况已经处理完，每个数至少得交换两次
- 还记得“实践”里发现的“最坏情况”吗？即使是 2 3 1 或者 3 1 2，也能两次搞定
- 然后就变成了一个**模拟题**（网上有些题解也是这么说的）
- 可是，如何严格证明？需要更深入的剖析题目



证明

- 很多题目的证明都是建立在高度抽象的基础上，并且一些高度抽象会得到更简洁的解法
- 所以在解任何题目的时候都可以尝试不断把题目抽象化，去掉无关信息，简化求解对象



剥茧抽丝

- 真正重要的是：有几个 1 在 2 的家？几个 1 在 3 的家？等等
- 记这些数量为 $at(1,2)$, $at(1,3)$ 等等，简称 $a_{1,2}$, $a_{1,3}...$
- 预处理出这些数之后，这个题目就变成了一个六元函数 $f(a_{1,2}, a_{1,3}, a_{2,1}, a_{2,3}, a_{3,1}, a_{3,2})$
- 于是，输入的 1000 个数一下子只剩 6 个了
- 还不满意？那就再看看这六个参数有没有**关联**



不变量 (invariant)

- 既然是重排列，那么 1 的个数是不变的
- 1 的个数是多少呢？有两种方法

- 直接数： $a_{1,1}+a_{1,2}+a_{1,3}$

- 1 的“家”有多大？ $a_{1,1}+a_{2,1}+a_{3,1}$

- 所以 $a_{1,1}+a_{1,2}+a_{1,3}=a_{1,1}+a_{2,1}+a_{3,1}$

$$a_{1,2}+a_{1,3}=a_{2,1}+a_{3,1}$$

- 类似的，有：

$$a_{2,1}+a_{2,3}=a_{1,2}+a_{3,2}$$

$$a_{3,1}+a_{3,2}=a_{1,3}+a_{2,3}$$



现在可以模拟了

- 首先是 1 和 2 交换，2 和 3 交换，1 和 3 交换
- 假定 $a_{1,2} > a_{2,1}$ ，第一次交换完之后还剩 x 个，根据之前的等式，第 2 次交换也剩 x 个，第 3 次交换也剩 x 个，符合 3 1 2 模式
- $a_{1,2} < a_{2,1}$ 是完全对称的
- 所以总交换次数就是

$$f(a_{1,2}, a_{1,3}, a_{2,1}, a_{2,3}, a_{3,1}, a_{3,2}) = \\ \min\{a_{1,2}, a_{2,1}\} + \min\{a_{2,3}, a_{3,2}\} + \min\{a_{1,3}, a_{3,1}\} + 2|a_{1,2} - a_{2,1}|$$



证明?

- 现在就有具体的证明目标了：记我们刚才求的最优交换序列为 S ，对于任意交换序列 S' ，它的交换次数 $|S'| \leq |S|$
- 用**数学归纳法**。假定所有长度不超过 $n-1$ 的序列都满足上述性质。对于一条长度为 n 的序列 S' ，设它的第一次交换为 $x_1@y_1$ 和 $x_2@y_2$



然后

- 可以分类讨论一下，是一举两得的交换，还是 312/231 这样的无奈交换，或者是开倒车的不靠谱交换？分别计算一下结果，有没有可能比我们找到的更优。
- 一般来说，比赛中做到这一步就已经够了，虽然没有具体写出来，但感觉上还是很有信心的。但是有人（比如我）可能并不满足于此：还是得写出来，不然怎么知道有没有瑕疵呢？然而，如果你真的写一下就会知道：这个证明是个体力劳动，**写了一次就不想写第二次了。**



证明略

- 凡是体力劳动，就有计算机出场的机会！
- 好在我们有计算机辅助证明！后面还会谈到这一点
- 详细证明见 IOI 官网
- <https://olympiads.win.tue.nl/ioi/ioi96/contest/ioi96s.pas>






总结

- 实践出真知，没思路的时候研究样例（或者自己举例）
- 多试试抽象题目，剥茧抽丝
- 不变量 (invariant) 是个重要的工具
- 别忘了数学归纳法
- 其他例子：15 数码问题、排序需要的相邻交换次数（这个是要变的，但也类似）



前面那篇 nature 文章

| z: Knot | X(z): Geometric invariants | | | | Y(z): Algebraic invariants | |
|--|----------------------------|--------------|------------------------|-----|----------------------------|--|
| | Volume | Chern–Simons | Meridional translation | ... | Signature | Jones polynomial |
|  | 2.0299 | 0 | i | ... | 0 | $t^{-2} - t^{-1} + 1 - t + t^2$ |
|  | 2.8281 | -0.1532 | $0.7381 + 0.8831i$ | ... | -2 | $t - t^2 + 2t^3 - t^4 + t^5 - t^6$ |
|  | 3.1640 | 0.1560 | $-0.7237 + 1.0160i$ | ... | 0 | $t^{-2} - t^{-1} + 2 - 2t + t^2 - t^3 + t^4$ |

We hypothesized that there was a previously undiscovered relationship between the geometric and algebraic invariants.



引例 4. 相等性控制 (NWERC 2018)

- 输入 Balloon 语言的两段程序，判断是否等价
- 四种语句：列表常数，concat, shuffle, sorted
- 其中 shuffle 是把输入列表进行均匀随机排列

Sample Input 3

```
concat(sorted([4,3,2,1]),shuffle([1]))  
concat(concat([1,2,3],shuffle([4])),sorted([1]))
```

Sample Output 3

```
equal
```



程序的等价性

- 本题关心的是程序的等价性，而不是对于某个具体的输入，两个程序的输出相同。
- 一般地，如果程序是函数式 (functional) 的，那么程序的等价性就相当于函数的等价性。证明函数等价的一个常见方法是证明两个函数对于任意相同输入产生相同结果。这依赖于一个叫 Functional Extensionality(简称 funext) 的公理



funext

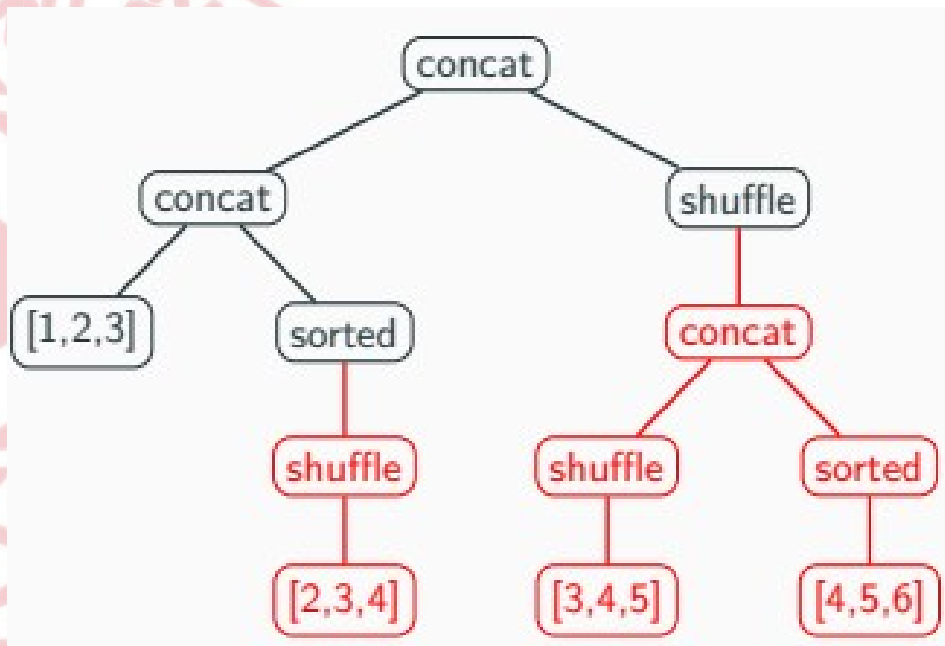
- 我们用 Coq 语言描述 funext 的简单形式:

```
Lemma func_ext : forall A B (f g : A -> B),  
(forall x, f x = g x) -> f = g.
```

- 可惜，由于随机性的存在，本题的程序并不是传统意义上的函数式程序，即使用上 funext 也无法（直接）证明它们的等价性

rewrite

- 另一个证明等价性的常用方法是改写 (rewrite)
- 比如多项式展开、微积分里的各种化简和变换规则都属于 rewrite
- 本题可以直接去掉 sorted 和 shuffle 下面的子树然后尽量计算 (包括元素全相同的 shuffle)





程序的代数

- 我在初学编程的时候，很早就接触到了这样的东西：
 $x = x + 1$ （ Pascal 的写法是 $x := x + 1$ ）
- 我被告知：这个是赋值，而不是数学上的等号，因为在数学上 x 不可能等于 $x + 1$
- 再后来，我学到了函数式编程，它重新鼓励用等式：
 $\text{square } x = x * x$
- 这个式子是说 $\text{square } x$ 就是 $x * x$ ，二者完全等价



引用透明性

- 如果你的程序是无副作用 (side-effect) 的，那么它真的是数学意义下的等式，具有引用透明性 (referential transparency)，可以在任何情况下把左端替换成右端（或者反过来）。比如刚才提到的 $\text{square } x = x * x$ ，那么任何碰到 $\text{square } x$ 的地方都可以换成 $x * x$ 。



副作用

- 副作用的一个典型例子是 I/O
- 比如，我们说 $x-x=0$ ，那么在 C 语言里是不是有 `getchar()-getchar()==0` 呢？
- 另一个典型的例子是全局变量。如果你的函数用到的全局变量，可能你就无法肯定的说 $f(3)=4$ ，因为也许 $f(3)$ 有时候等于 4，有时候不等于 4，看你啥时候调了



代数性质不显然

- 我们在中学代数里学过加法结合律： $(x+y)+z=x+(y+z)$ ，不难想象很多证明和常见的代数化简都依赖于它
- 但是浮点数却不满足加法结合律，比如也就不会有 Kahan 算法了（没学过的话赶紧学一下！）
- 还有当年 TopCoder 因为 NaN 错失冠军的故事



哦，还有一个题

- Find three numbers such that $x=y$ and $y=z$, but $x \neq z$ (IPSC 2008C Hard)

```
import java.util.*;
public class C2 {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in); sc.useLocale(Locale.US);
        ?????? x = sc.?????();
        ?????? y = sc.?????();
        ?????? z = sc.?????();
        System.out.println( (x==y) + " " + (y==z) + " " + (x==z) );
    }
}
```



程序的代数？

- 既然有些程序可以写成真正的等式，那么可以想象我们能用上很多初等代数的知识，证明有关程序的结论，进行程序 rewrite（可能会加速），甚至可以把你要的程序**算出来**！你想写的程序需要具有哪些性质？列一些**等式**，然后解方程！
- 当然，自动计算程序这种事情在实践中还是挺麻烦的 ... 但是不觉得很酷吗？

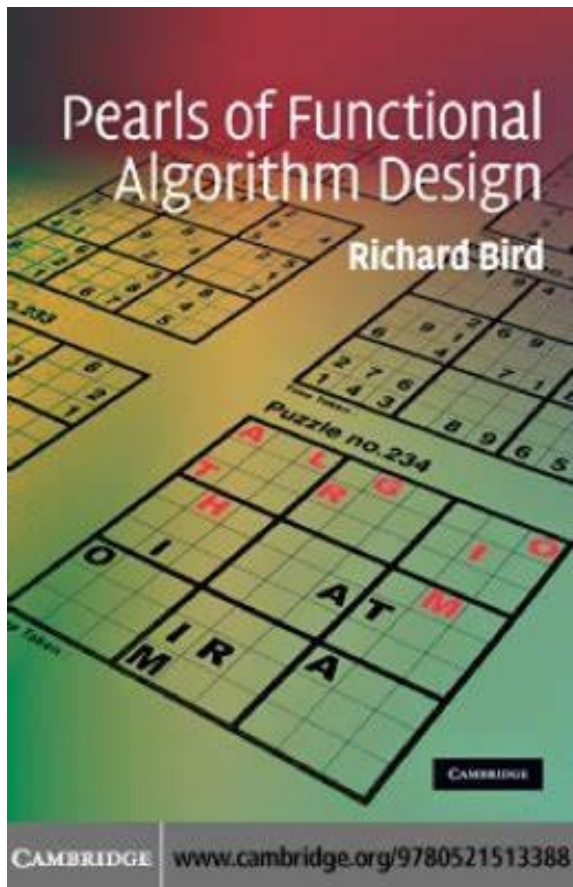


没时间了

- 虽然我还想再安利一下函数式编程，可惜没有时间了，以后写书里吧 :)
- 函数式编程也不是和信息学竞赛无关的，你们不是早就学会了持久化数据结构吗？



中国计算机学会
China Computer Federation





题解呢？

- 虽然我觉得可能看到前面的图就应该已经会做了，不过为了完整性还是说一下吧
- sorted 结点可以直接计算，而 shuffle 可以简化成 shuffle(一个具体的 list) 或者直接算出来（比如 shuffle([1,1,1])，而 concat 结点如果两个儿子都是 list 也得算出来。最后直接比较两棵树即可
- 虽然这题我是想讲 rewrite，但也可以不用



引例 5. Game Relics (NERC 2019)

- 有 n 种物品，直接买的价格是 c_i ，也可以花 x 块钱从所有 n 种物品中随机一个。如果随机到一个新物品，则直接得到；如果随机到一个已经有的物品，则返还 $x/2$ 块钱
- 输入 x, n 和所有 c_i ，求最优策略下需要花多少钱可以集齐所有物品



分析

- 准备完这道例题之后才发现 OI wiki 的讲解非常好，没必要重复一遍，所以讲稿里改成只讲思路，细节请移步 <https://next.oj-wiki.org/misc/rand-technique/>
- 核心就是几个比较直观的结论：
- 结论 1：每次一旦决定随机，就会一直随机到获取新物品为止。这个叫**确定性决策**（后面会碰到随机决策）



分析

- 结论 2：最优策略是先随机若干次，然后再买买买。首先用 exchange argument，比较“先买再随机”和“先随机再买”两种策略。注意，我们比较的是随机过程，得把两个过程的样本空间做一个一一对应，用一个容易理解的说法就是“选用同一个均匀随机数发生器”



中国计算机学会
China Computer Federation



第二部分、一些常见的套路性直觉



内容提要

- 主要讲思路，包括解题和证明
- 细节繁多的题目留到第三部分
- 主要内容包括：
 - Exchange argument
 - 等价转换
 - 由局部到整体
 - 几何中的常见猜想

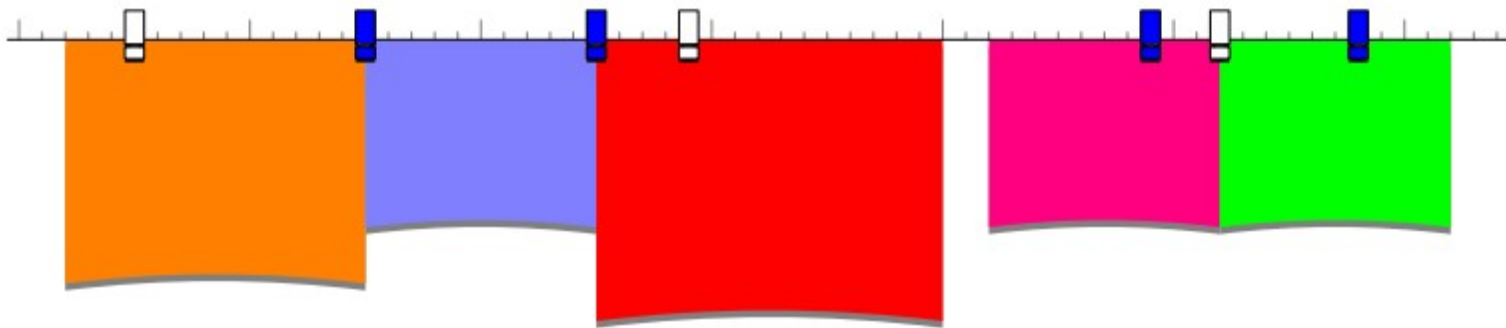


2.1 “传统算法题”

- 例 1. Canvas Line
- 例 2. Assigning Workstations
- 例 3. Swap Space
- 例 4. Quests

Canvas Line (NWERC 2019)

- 晾衣杆上有 n 张布，已经有 p 个夹子，要求再放尽量少的夹子，使得每张布上恰好有两个夹子。要输出方案。布的宽度至少为 10cm，夹子的宽度略小于 1cm。下图中，已有夹子用白色表示
- $n \leq 1000$, $0 \leq p \leq 2000$





分析

- 错误的贪心：从左到右考虑各个点，能加就加
- 正确的贪心：先考虑公共端点，能放就放，剩下的再从左到右贪心
- 证明？如果算法是错的，会是怎么个错法？个数不是最少？算出非法解？有解却找不到？



分配工作站 (NWERC 2015)

- 某研究所的每台工作站在使用完 $m(1 \leq m \leq 10^8)$ 秒钟后都会自动锁屏。现在有 $n(1 \leq n \leq 300,000)$ 个研究人员需要使用工作站，其中第 i 个人需要从第 a 秒开始使用，使用时间为 $s(1 \leq a, s \leq 10^8)$ 秒。
- 你的任务是给每个研究人员分配工作站，使得每个人都不会遇到自动锁屏。要求用到的工作站总数尽量小。



分析

- 设想我们是所里的研究人员。我们会用哪一台电脑呢？仔细一想，这些电脑唯一的区别就是多长时间后锁屏，所以直接选最先锁屏的用就行了！
- 启发：要选择的東西的所有特征可以用一个数概括，往往排个序之后就稳了



交换空间 (Swap Space, WF 2016)

- 你有 n ($1 \leq n \leq 10^6$) 块硬盘，第 i 块的容量为 a_i ，格式化之后为 b_i ($1 \leq a, b \leq 10^9$ ，单位：GB)。目前你的数据占满了所有硬盘，而你想格式化每块硬盘。为此，你只能买一些新的硬盘作为中转。要求新硬盘的总容量尽量小。在中转的时候，数据可以分成若干份。
- 比如你有四块硬盘 A, B, C, D，容量分别为 6, 1, 3, 3。格式化之后的容量分别为 6, 7, 5, 5。则只需要额外有 1GB 就能搞定。



比赛情况

- 大部分队伍能做出
- 但是大部分队伍都至少错过一次
- 402 提交, 108AC
- 算法简单, 但容易想错!
- 强烈建议同学们听完课后再花点时间复盘这个题



两个东西

- 和上题相比，难度一下子增加了。每块硬盘得用两个数 a_i 和 b_i 描述。排序还好使吗？按 a_i 还是 b_i ？直观的感觉，先格式化 a_i 比较小的硬盘有利于控制答案“不要蹭蹭蹭往上涨”，而先格式化 b_i 比较大的硬盘有利于“早点处于躺平模式”，不再为了空间而发愁，**感觉都挺有道理**



排除一些明显不靠谱的决策

- 我们觉得先格式化 a_i 小的比较好，有一个隐含的前提是 ... 格式化之后容量增加。否则我们干嘛费力不讨好？当然，除非所有硬盘格式化之后的容量都会变小。证明？还是 exchange argument
- 不过总的来说，我们应当**优先格式化那些容量会变大的硬盘**。在这个前提下，应该先格式化 a_i 比较小的，还是 b_i 比较大的？



剩下的

- 答案是 a_i 比较小的。继续用 exchange argument 试试
- 然后再处理容量会变小的硬盘。由于对称性，这次就应该按照 b_i 排序了。
- 启发：不要着急排序，可能要先分类。分类的一个方法是找一找关键性质及其成立的条件



Quests (WF 2020)

- 你在玩一个游戏，每获得 v 个经验值 (XP) 就会升一级。一共有 n 个任务 (quest)，每个 quest 由两个整数 (x, d) 表示。如果你的级数 $\geq d$ ，完成后获得经验 x ，如果级数小于 d ，将会获得 c 倍经验值
- 输入 n, v, c 和每个任务的 x 和 d ，求做完所有 quest 之后的经验值的最大值（可以按任意顺序完成，而且你的水平非常高，可以以任何等级完成任意 quest）
- $1 \leq n, v \leq 2000, 2 \leq c \leq 2000, 1 \leq x_i \leq 2000, 1 \leq d_i \leq 10^6$



分析

- 根据上一题的启发，要不要分类？显然每个 quest 都要做，而且没有奖励的 quest 不需要着急做，所以先分两个大类。问题转化为：有奖励的 quest 应该按照什么顺序做？
- 对于一个 quest (x, d) ，如果想获得奖励，完成之前的经验值必须小于 $v*d$ ，而完成之后的经验值会小于 $c*x + v*d$ 。这些是做这个任务并获取奖励的前置条件 (precondition) 和后置条件 (postcondition)



解法

- 直观的看, $v*d$ 越小越应该先做, 但是有可能一个 $v*d$ 比较小的 quest, 做完之后经验值 $c*x$ 特别大, 一下子就导致另外一个 quest 得不到奖励了
- 所以说, 如果两个 quest 都能做, 应该按照 $c*x+v*d$ 从小到大的顺序做, 用 exchange argument 很容易证明。问题转化成了: 排序后选一些 quest 依次完成并获得奖励



解法

- 这下能想到 DP 了吧
- 依次考虑每个 quest，状态是“当前经验值”。虽然数值上可能很大，因为我们关注的是获取奖励的情况，这个值肯定是 c 的倍数，除以 c 之后不同的数值不超过 $\text{sum}\{x_i\}$ 种
- 然后我们只关心状态是否可达，位运算加速即可



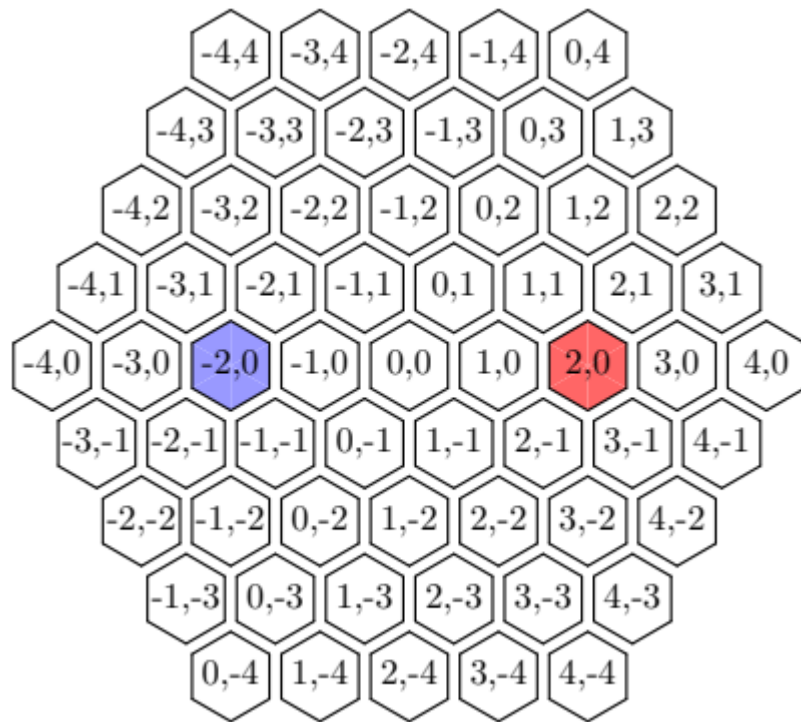
2.2 一些策略问题

- 例 5. Interactive Knockout
- 例 6. The Mind
- 例 7. Flipping Coin
- 例 8. Is it Rated?



Interactive Knockout (NERC 2020 OL)

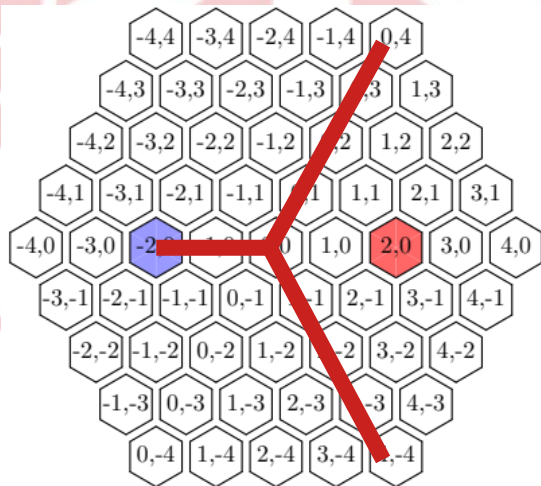
- 在一个坐标最大值为 n 的六边形棋盘上，你从蓝色格子出发，对手从红色格子出发，双方轮流走，你先走。每次可以走到相邻格子，但不能走到对手的格子，也不能走到曾经有人到过的格子。不能走算输
- 对手的策略纯随机（能走的各个方向等概率走）要求在最多 5000 场 $n=20$ 的比赛里**全胜**
- 右图是 $n=4$ ，不会出现在真实数据中





分析

- 设总格子数为 s ，很大概率下 $s/3$ 轮之内会自己输掉。我们先走到 $(0,0)$ ，如果对方在下半棋盘，就围住左上。如果它撞进来的话 ...





The Mind (WFI 2020)

- 玩一个双人合作游戏：每人发 5 张牌，然后玩 5 轮。每轮每个人**同时做出选择**：可以选择出他的最小牌，也可以不出。只要有人出牌，这一局立即结束，当且仅当最小牌被打出，且没有其他牌被打出时，游戏者获胜。
- 程序输入 $n=1000$ 个局面（5 张牌的面值，1~100），要求每读取一个局面后立即给出一个**策略** $p_1 \sim p_5$ ，其中 p_i 表示第 i 轮时出牌的概率。 $\sum\{p_i\} \leq 1$
- 自动测试这 n 个局面的所有 pair，赢 85% 就算通过



"pure strategy"

- 有效合作的两大手段：通讯和默契
- 题目没说要找最优策略，只要 85% 的胜率就行
- 直觉上，如果你抽到的数很大，最好不要先出。但是如果等了好几轮之后对方也没有出牌，说明对方的牌也挺大的，可以考虑出
- 如果出的轮数和牌的大小有某种确定的关系，那么双方可以形成默契，增加胜率



Flipping Coin (IPSC 2011)

- $n(1 \leq n \leq 8)$ 个人坐成一圈，每个人抛一个硬币，可以看到别人的硬币，但看不到自己的
- 每个人可以猜自己的结果 (H/T)，也可以 pass。如果至少有一人猜对，但没有人猜错（其他人都 pass 了），则整体获胜
- 求获胜概率最大的**确定性**策略。
- 整体策略由每个人的策略组成，每个人的策略是一个函数 $f(\text{outcome})$ ，其中 outcome 是所有 2^{n-1} 种可能的观察，函数值为这个人的决定 H/T/P(Pass)



初步分析

- 说是概率，实际上枚举所有 2^n 种可能的情况，数一数这个策略赢了多少次，就能知道获胜的概率了。每个人能观察到的信息很多，他只面临两种可能性，所以可以想到把可能性当做点，选择看成边
- 实际上挺难想到的。我在比赛的时候就没想到



继续分析

- 为了让问题更加直观，我们把一个决策进行“可视化”：对于一条边 $u-v$ ，如果我们选择猜 u ，就在结点 u 上画一个小蓝点，在 v 上画一个小红点。如果 Pass，就什么都不画。
- 有多少种情况会整体获胜呢？就是要数一下有多少结点里有至少一个小蓝点，但是没有小红点，我们称为 good node



空白结点？

- 可能有空白结点吗？
- 理论上有可能，但是我们可以选择不要
- 规范化！如果有，我们在里面画一个小蓝点，同时也得在某一个相邻点里画上小红点
- 顶多抵消，没有变差



数学模型

- 每个 good node 至少有一个相邻点里有小红点，所以带红点的结点 (bad node) 形成了一个支配集 (dominating set)。
- 不考虑空白结点，所以 good node 最多意味着 bad node 就得最少
- 所以最优策略对应了一个最小支配集。反过来呢？是不是每个支配集都对应一个可行策略？



给定支配集

- 对于每条边，如果其中一个端点在支配集中而另一个不在，那么就猜那个不在支配集的
- 其他情况下 pass
- 在这个策略下，所有不在支配集中的结点（局面）都会获胜
- 启发：——对应



Is It Rated? (NERC 2020)

- Izzy 和 n 个人做 m 道判断题 (答案是 1 或者 0) , 每次都是 Izzy 最后做, 可以看到其他 n 个人的答案。 Izzy 做完之后公布答案
- 现在你扮演 Izzy 的角色, 不过不告诉你题目, 只能靠瞎蒙。要求最多只能错 $1.3b+100$ 次, 其中 b 是题目全部做完后, 错的最少的人错误的次数
- $1 \leq n \leq 1000, 1 \leq m \leq 10000$



直觉

- 大神带我飞！
- 每次你都最后猜，可以数一数谁错得最少，然后跟着猜
- 可惜有时候并没有大神，大家都是瞎猜的，都是50% 的正确性，而你可能比瞎猜还差！



比瞎猜还差？

- 比如就两个人，0¹和¹0交替（红色代表正确答案）
- 奇数次，两个人打平，你有 50% 概率猜对；偶数次“大神”猜错了，你跟着错。总错误率 75%



加个权

- 比如有四个人，错误率是 20%，19%，18%，17%，分别猜 1, 0, 0, 0，是不是感觉 0 的可能性大一点？
- 那就加个权，相当于每个人投个票，错了 x 题的人投票的权重是 b^x 。最后看看 1 和 0 的（加权）票数哪个大，就猜哪个



还不够

- 得票最多的那个选项也不一定靠谱，还需要按概率来。根据票数比率计算概率，然后随机
- 如果不随机的话，可以构造数据使得每次总票数都差不多并且错误答案的得票数还要稍微高一点，然后不随机的算法的错误率比随机算法几乎高一倍！



机器学习理论

- 其实思路也不难想到，不过是否真的满足题目说的界，还是需要一番证明的
- Randomized Weighted Majority Algorithm (RWMA)
- https://en.wikipedia.org/wiki/Randomized_weighted_majority_algorithm

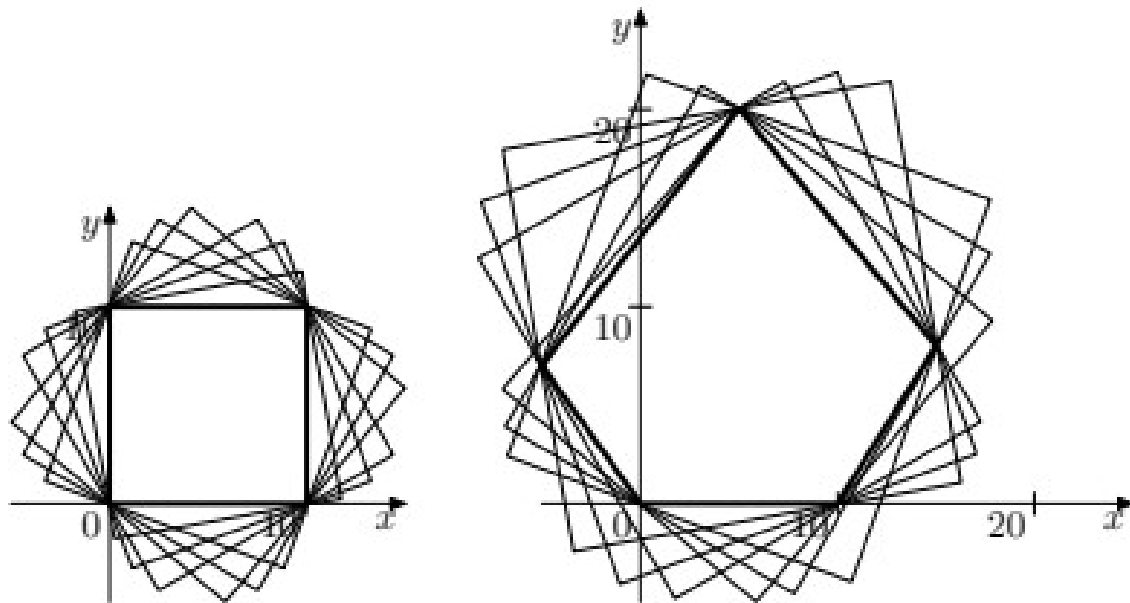


2.3 和几何有关的问题

- 例 9. Framing Pictures
- 例 10. Fiber Shape
- 例 11. Connecting Dots
- 例 12. Screammers in the Storm
- 例 13. Convex Polyhedron
- 例 14. New Flat

Framing Pictures (WF 2020 Inv.)

- 输入一个 n 个点的凸多边形，随机一个拍照角度，求照片面积（即该角度下的包围盒面积）的数学期望
- $n \leq 200,000$





满满的套路

- 因为很多人不太熟悉几何，先来一个套路题
- 因为随机的只有角度这一个参数，很自然的变成一个以角度为参数的定积分
- 麻烦之处在于，很难列出一个统一的式子。影响包围盒的顶点集不同，式子也不同



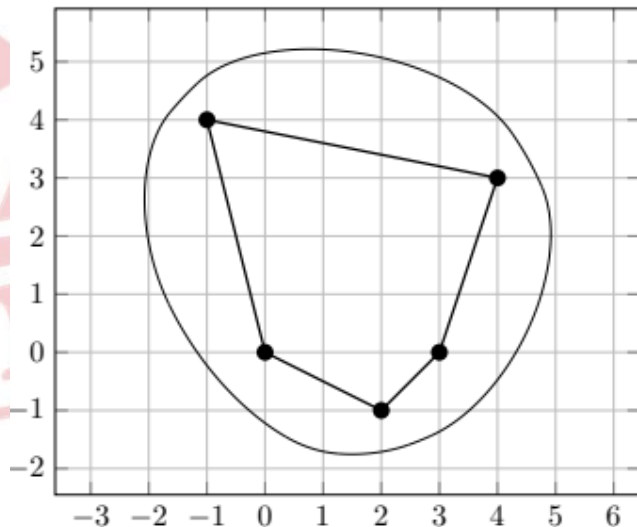
旋转卡壳

- 如果你只是背代码，可能并不知道旋转卡壳能用到这里
- 如果你知道旋转卡壳（具体来说，最小包围矩形）实际上遍历了包围盒的**所有形态**，或者理解为把**参数空间**分割 / 离散化得足够细（统一表达式，可直接积分），问题就解决了
- 积分部分不是重点，自己看官方题解吧



Fiber Shape (NERC 2020)

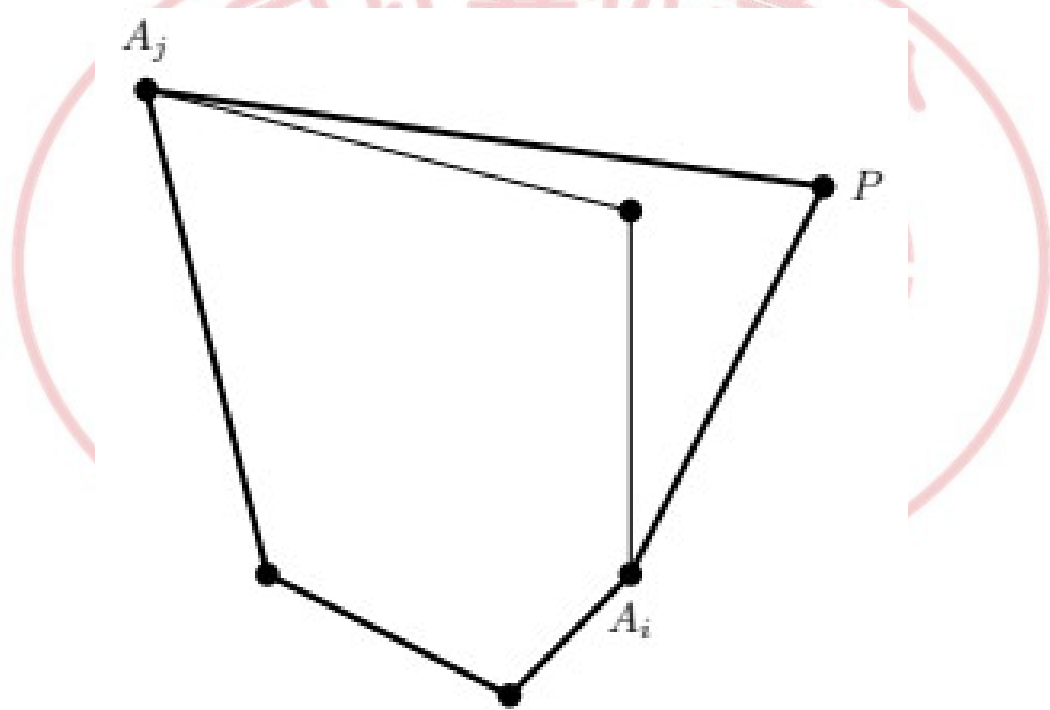
- 输入一个 n 个结点的凸多边形，你有一个长度为 L 的橡皮筋，绷住多边形后，把其中一个点拉得尽量远。求这个点的轨迹组成的图形的面积。 $n \leq 30000$





把一个点拉到尽量远

- 在拉 P 的过程中，橡皮筋总长不变，所以 ...





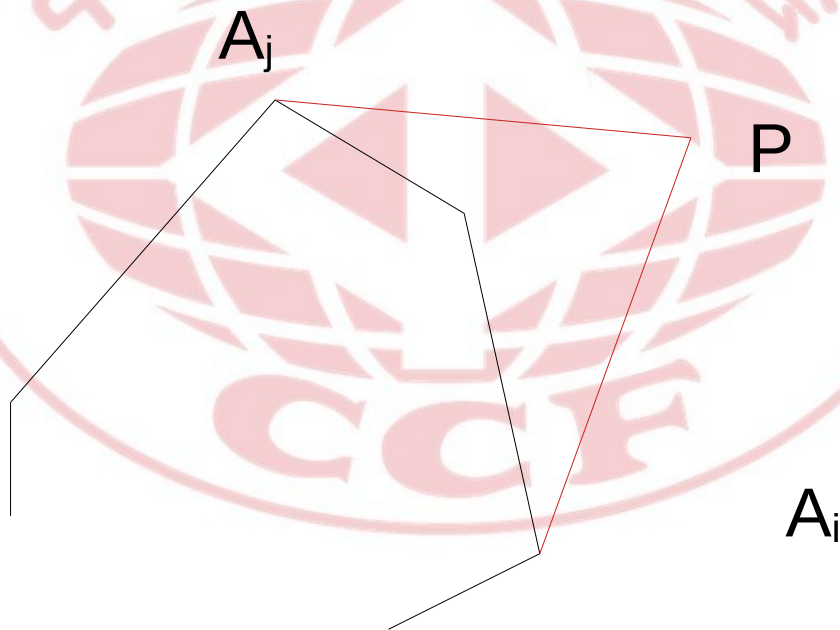
椭圆弧

- 在**整体形状不变**的前提下， P 的轨迹是以 A_i, A_j 为焦点的椭圆弧
- 假设我们逆时针继续拉 P ，拉到椭圆弧的边界之后再拉一点，会怎样？ $A_i A_{i+1}$ 绷住了，接下来焦点从 A_i 变为了 A_{i+1} 。



换一个图

- 如果改成这样呢？支撑点 A_j 会“滑到” A_{j+1} 。所以说后续有两种情况，看看哪种先发生即可





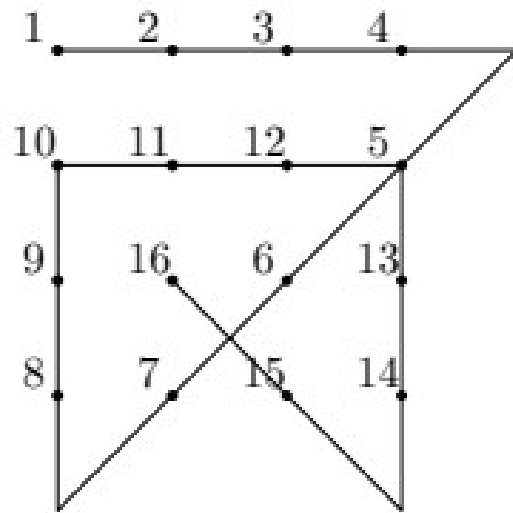
另类旋转卡壳

- 和旋转卡壳很像，从 A_0A_1 开始绕凸多边形一圈，每次打射线看看“绷住下一个点”和“滑到下一个点”哪个先发生
- 面积的计算可以用格林公式，但不是这里的重点，自己看官方题解吧



Connecting Dots(NWERC 2017)

- 1~16 各一次组成一个 4×4 点阵网格, 要用尽量少的转折次数**一笔画**穿过所有的点, 使得排列 $1, 2, 3, \dots, 16$ 是访问序列的子序列 (比如, 实际访问序列可以是 $1, 4, 2, 3, 2, 4, \dots$)





连续的决策

- 这道题目的讨厌之处是：决策是连续的。当你准备另起一条线段时，你有无穷多种选择。
- 对付“无穷多种选择”最常见的套路是离散化
- 上两道题目也可以说是广义的离散化



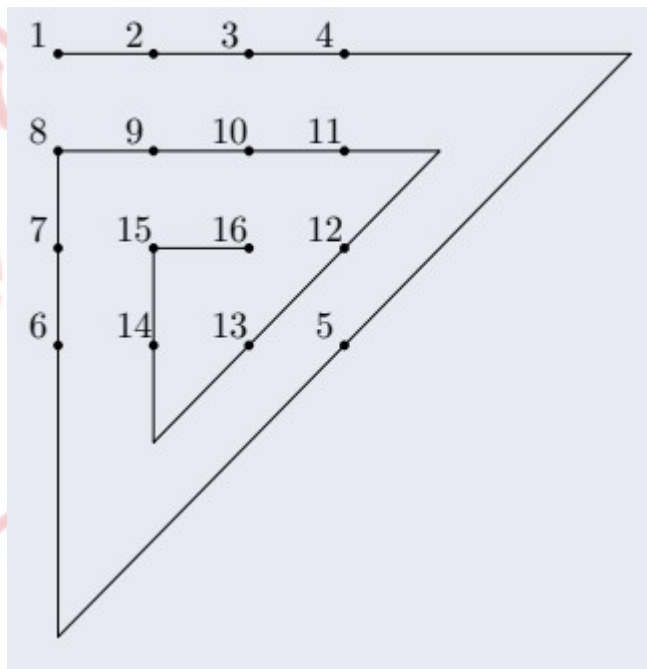
离散化？

- 不知道你们的直觉如何，反正我觉得不太好离散化 ...
- 离散化比较适合静态的，一拿到输入就可以离散化
- 处理动态物体的常见工具是离散事件模拟系统（比如扫描线），随着算法的执行，动态生成事件。
- 好像都不太适用本题
- 刚看到本题，我的直觉是利用“两点确定一条直线”，只要想办法固定两个点，就能确定一条直线。比如，可以在搜索的过程中确定？



良心样例

- 如果你手算了样例 2，会发现 ...
- 有的线段可能只经过一个点
- 那这条线段的**自由度**岂不是很大？
- 会有什么**约束**它吗？





换一个思路

- 如果当前状态只是“一条射线”，那么似乎很难决策。但如果我们保存了“所有可能的射线”会怎样？
- 用数学方法表示无穷集合，比如区间，然后在算法过程中加以维护
- 直觉上的动机：不要提前排除任何可能性，然后把决策延迟到信息足够时完成



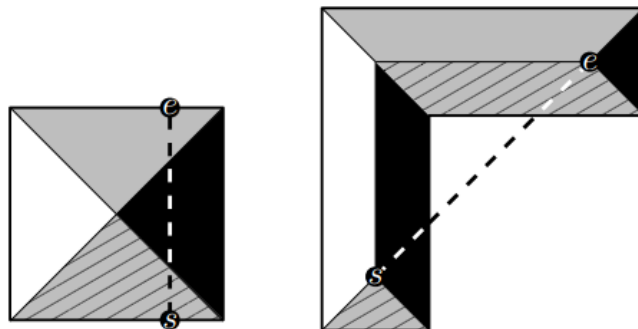
解法

- 设 $S(i)$ 表示从 1 到 i 的线段数量最少的前提下，进入点 i 的角度集合，感觉这个集合中间不会有间断点，所以暂时认为它是一个区间
- 已知 $S(i)$ ，如何计算 $S(i+1)$ ？有两种可能
 - 直接走过去，线段数不变
 - 继续走到无穷远处，然后顺着 $S(i)$ 的左 / 右端点绕回来
- 注意 $S(i)$ 可能是单点或者半开区间



Screamers in the Storm (CERC 2019)

- 有一个底面为 n 边形的屋顶，每条边都平行于坐标轴。输入边界上的两个点 s 和 e ，求一只鸽子从 s 飞到 e 的距离。路线在俯视图上看就是 s 到 e 的线段，只是在屋顶上的时候用脚走，悬空的时候沿直线飞。 $n \leq 400$





- 三维想象不清楚的话可以先理解俯视图
- 用 wedge 表示一条屋顶边界线段生成的（有可能无限大）梯形。想象在屋子里面仰望屋顶的反面，那么任意二维位置 (x,y) 处的高度就是所有 wedge 对应的斜面的最低高度





- 俯视图上的 s-e 线段对应空间里的一个平面（称为 travel 平面），相当于屋顶围住的空间的一个切面
- 在这个切面上，每个 wedge 就是一条射线，原题在概念上就是在 travel 平面求下包络。具体做的时候把 x 离散化后每个 mid_x 现求最近的 wedge 即可。最近的 wedge 对应点高度也最小
- 离散化的方法是求 s-e 和每个 wedge 在俯视图上的交点（最多两个点），然后映射到 travel 平面
- 启示：三维几何经常转化为二维几何，包括俯视和立面



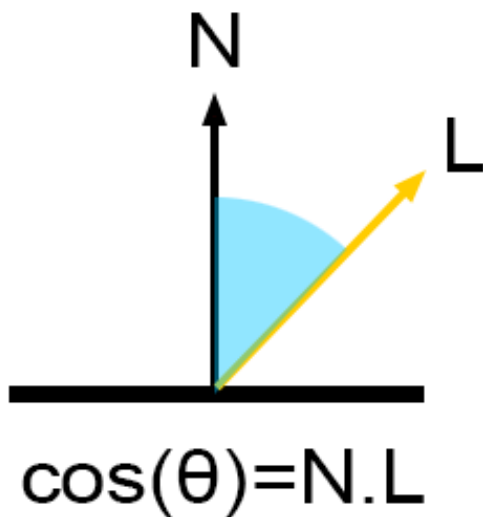
Convex Polyhedron (EC-Final 2015)

- 有一个三维凸多面体，求一个平面，使得多面体投影到平面上的多边形面积最大
- 输入多面体的 n 个顶点，输出最大面积。 $n \leq 50$, $|X_i|, |Y_i|, |Z_i| \leq 10^9$



如果你有图形学基础

- 假设有一束平行光照射在一个平面上，漫反射 (diffuse) 的强度和照射角度有关。直射的时候反射最强，而几乎平行的时候，只是“蹭到了一点”，反射就很弱，遵循 \cos 定律，或者理解成一个点积，或者投影面积
- 图片来源：www.scratchapixel.com





基础知识

- 用点法式表示平面，平面方程为 $\text{Dot}(\mathbf{n}, \mathbf{p} - \mathbf{p}_0) = 0$
- 一个平面上的三角形投影到另一个平面上，也遵守上述的 \cos 定律
- 三角形的面积和二维一样，注意求出叉积后要求长度，即 $|\text{Cross}(\mathbf{B} - \mathbf{A}, \mathbf{C} - \mathbf{A})|$
- 点积满足交换律和对加法的分配率



照亮部分的总投影面积

- 首先求三维凸包
- 对于凸包上的一个被照亮的三角形，设面积为 a ，法向量为 n ，照射向量为 d ，则投影面积等于 $a(n \cdot d)$
- 先把原始面积提到 Dot 里面去，然后根据 Dot 对加法的分配率，被照亮的三角形的投影面积之和等于 $(\sum a \cdot n) \cdot d$



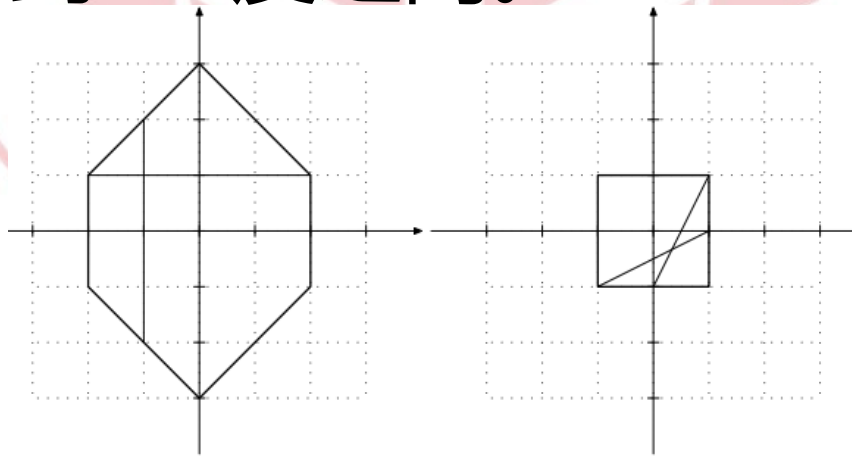
$$\bullet (\sum a^*n) \cdot d$$

- 而 a^*n 其实就是叉积的一半。因此只要知道了哪些三角形被照亮，我们让 d 等于那个加权和 $w = \sum a^*n$ 对应的单位向量，答案就是 w 本身
- 所以问题转化为：如何枚举“哪些面被照亮”
- 和 Framing Pictures 一样，空间分割！枚举三个点
- 其实还能随机照射方向计算出照亮三角形集，也能 AC



New Flat (NERC 2020 OL)

- 输入一个 n 个点的凸多边形，以及多边形内一条线段 AB （端点可以在边界上），要求把它平移旋转成一条新线段 CD ，使得二者夹角尽量大。夹角总是 0 到 90 度之间。 $n \leq 50$





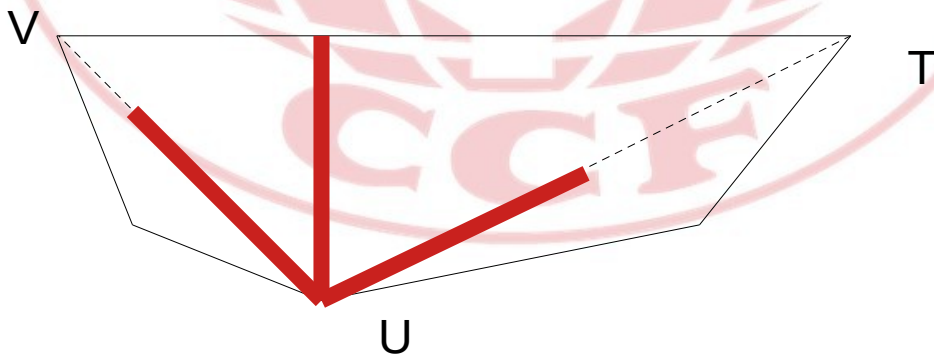
直觉

- 旋转的时候沙发的一端固定在房间的一个顶点
- 旋转最好在空旷的地方进行。又想到空间分割了？
- 定义多边形的边和对角线统称为 horde 。定义 horde 之间的等价关系如下：如果沙发初始时完全在线段 H_1 上，经过变换后完全在 H_2 上，我们说 H_1 和 H_2 等价。长度小于 AB 的只与自己等价



等价关系

- 考虑三个不同顶点 U, V, T ，其中 V 和 T 是多边形的两个连续顶点，且 $|UV| \geq |AB|$ ，且 $|UT| \geq |AB|$ 。如果 U 到线段 VT 的最小距离不小于 $|AB|$ ，则 UV 和 UT 等价， VU 和 TU 等价





解

- 如果所有长度不小于 $|AB|$ 的 horde 相互等价，说明完全随便移动和旋转，答案是 90 度
- 否则我们需要找一个端点 P 使得 AB 可以平移旋转后 A 或者 B 和 P 重合，然后旋转到它相邻的 horde（至少有一个能成功到达）。考虑它的所有等价 horde，尝试着在相邻三角形内旋转，记录下与 AB 形成的最大角（注意不能超过 90 度）



2.4 其他题目

- 例 15. Cow Photography
- 例 16. 圣诞礼物
- 例 17. Anti-Tetris
- 例 18. Access Points
- 例 19. Juggling Troupe
- 例 20. Better Productivity
- 例 21. Game of Chance



奶牛照相 (USACO 2011 Dec)

- FJ 给他的奶牛们排成一个特定的顺序照相。即将快门的一瞬间，有一些调皮的奶牛冲了出来，插到了队列的其他位置，然后照了一张顺序错误的照片。FJ 没办法，只好把奶牛重新排好队。正准备按快门，又有一些奶牛冲了出来 ... 一共照了 5 张相。虽然调皮，但是奶牛们还是有底线的，所以每头奶牛最多冲出来一次。
- 输入这 5 张相片的顺序，求 FJ 最初给奶牛安排的顺序。

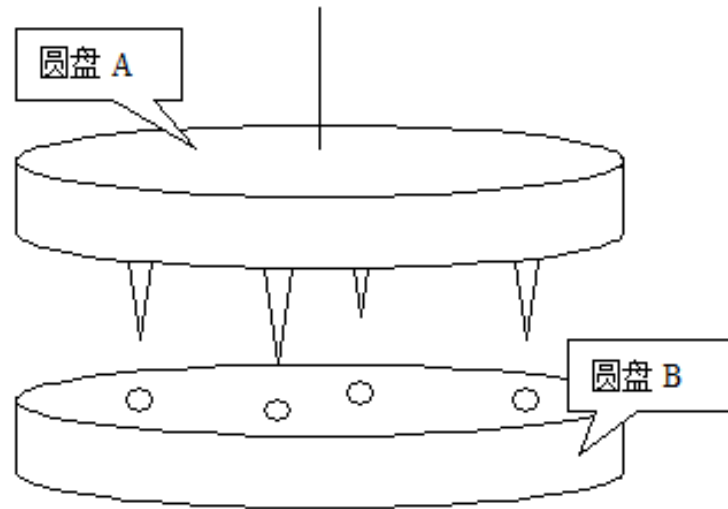


要点

- 每个牛最多只动一次
- 只考虑两个牛，那么 5 次照相中，至少其中三次**两头牛都没动**，而这三三次正好就是两头牛的真实排序
- 预处理之后，上述“排序两头牛”的操作可以在 $O(1)$ 时间内完成。把它传给 sort 就可以了
- 总结：由局部到整体

圣诞礼物 (NOI 冬令营 2003)

- 在圆盘 A 的圆周上，均匀的分布着 n 根直径相同的小银针，长度为 1, 2, ..., n 的针各一根，所有银针均垂直圆盘向下；在圆盘 B 的圆周（和 A 的圆周尺寸相同）上，均匀的分布着 n 个小孔，深度为 1, 2, ..., n 的孔各一个，孔的直径和针恰好一样，因此银针可以插到孔里。





- 佳佳可以把圆盘 A 往下压，直到有的银针碰到孔底。如果有的针长比孔深大，则针会有部分露在外面，圆盘 A 仍会离圆盘 B 有一定的距离。这个距离就是所有银针长度减去相应孔深的最大值。
- 输入每根银针的长度 ($1 \sim n$ 的一个排列)，可以调用 Rotate（旋转圆盘 A）和最多 5 次 Drop（交互库会返回两个圆盘的距离），使得两个圆盘完美贴合。 $3 \leq n \leq 100,000$



分析

- 比较容易想到的思路：筛
- 可是有两个障碍：次数够吗？速度够快吗？
- 一个看上去比较高大上的方法是用信息论，把各种 Rotate 之后 Drop 返回的信息量算出来
- 可惜太慢了



时间才是关键

- 就算是决策上不靠信息论，只是随便转一转，筛似乎也会很慢！怎么办？
- 只排除部分解！如果 Drop 返回值为 v ，比如是 n 对着 $n-v$ ，或者 $n-1$ 对着 $n-v-1$ 等等，一共 $n-v$ 种可能。如果 v 很大， $n-v$ 会很小；如果 v 很小，这些可能性往往有很多重复



这样就可以了

- 基本上都是 4 次以内，极少碰到需要 5 次的
- 平均不超过 3.5 次
- 但是并没有理论分析。有兴趣的同学可以试试看
- 启发：如果筛无法做到非常精确，可以用必要条件粗略的筛



Anti-Tetris (WF2020. Invitational)

- Sticky 俄罗斯方块游戏，每个 tile 是最多包含 7 个格子的四连块。一开始 tile 的第一行是棋盘的第一行，每一步可以往 left, right, down 移动一步，停下的时候可以悬空。给出最终状态，求一个可能的方案。每个 tile 输出第一行最左边格子的初始列编号和移动序列（前面是 LRD，最后一个字母是 S）

```
5 6
....dd
.ccccd
.cbbdd
.aab.a
aabbaa
5
2 DDDS
4 DDLS
6 DDDS
2 DS
5 S
```



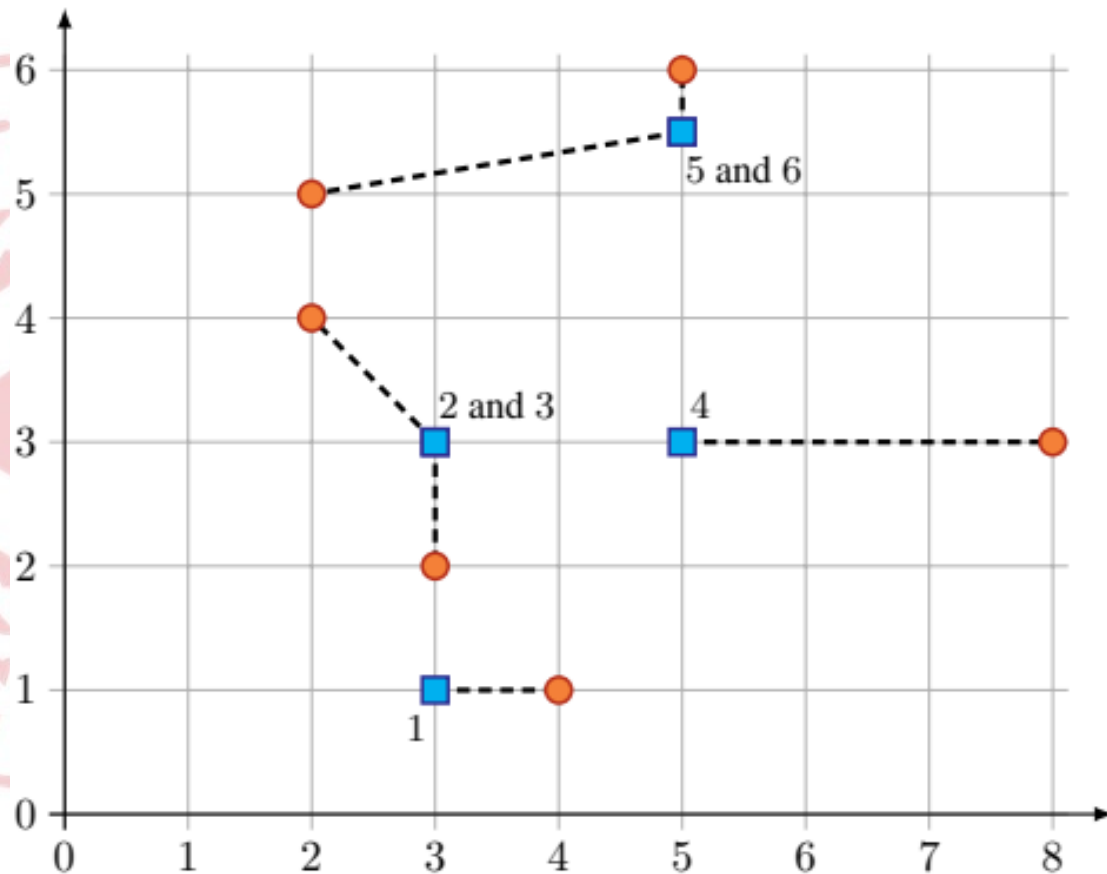

“正难则反”

- 每次贪心的找一个 tile 向上从棋盘里抽出来
- 如果多个能抽，那么先抽哪个都行，因为总不至于抽完某个 tile 之后原先可以直接抽出来的 tile 反而突然抽不出来了
- 证明？不需要说了吧



Access Points (NWERC 2018)

- 编程竞赛在一个很大的体育馆里进行。队伍 i 的网络端口位置固定为 (s_i, t_i) ，你需要设计每个队伍的位置，使得网线长度的平方和最小。对于两个队伍 $i < j$ ，坐标必须满足 $x_i \leq x_j$ 且 $y_i \leq y_j$ 。不同队伍可以在同一个位置





分析

- 套路性直觉：先把式子展开。两点距离的平方就是 x 之差的平方加上 y 之差的平方，所以 x 和 y 两个维度相互独立！
- 一维问题：给定序列 a_1, a_2, \dots, a_n ，求 n 个数 $x_1 \leq x_2 \leq \dots \leq x_n$ 使得 $x_i - a_i$ 的平方和最小



实践

- 如果用手做，大概会才用怎样的策略呢？一个一个的算。放 x_i 的时候，如果可以直接放到 a_i 的位置是最好的，如果不行的话，说明 $a_i < x_{i-1}$ 。这个时候可不可以“挤一挤”，把 x_{i-1} 往左挪一点？不行的话， x_i 就等于 x_{i-1} 了；如果可以挤一挤的话， x_i 肯定也要跟着一起移，否则 x_{i-1} 的移动就没有意义了，不管是哪种情况，都可以把 x_{i-1} 和 x_i 捆绑在一起看成一个 meta-item。

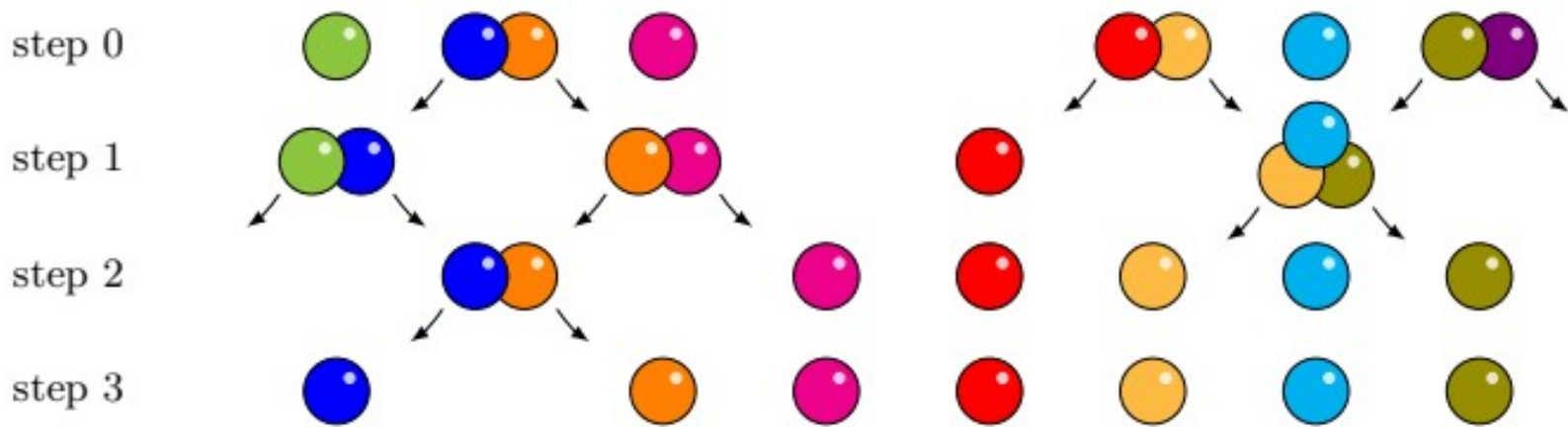


meta-item

- 不难发现，继续上述过程，meta-item 可能会越来越大，实际上代表着某个连续区间 x_j, \dots, x_i ，它的“等效 a_i ”就是让 $(x-a_j)^2 + \dots + (x-a_i)^2$ 最小化的值，就是 a_j, \dots, a_i 的平均值
- 套路：独立性、等效变换

Juggling Troupe (NWERC 2017)

- n 个菜鸟杂技演员在台上表演。每个人手里有 0 个 1 个或 2 个球。每个时刻，有 2 个球的人同时往左右扔球（如果没有人就直接扔到台下）





续

- 每个时刻，所有人同时动作，求最终（每个人手里最多一个球）每个人手里有几个球
- $n \leq 10^6$



2 是一个麻烦的东西

- 假定有一个 2，左边右边都是一些 1，然后两边各有一个 0（可以在最左边和最右边加两个虚拟的 0）

| L | | | | i | | R | | | |
|-----|---|---|---|-----|---|-----|---|---|-----|
| ... | 0 | 1 | 1 | 1 | 2 | 1 | 1 | 0 | ... |

| L | | | | $L + R - i$ | | | | R | |
|-----|---|---|---|-------------|---|---|---|-----|-----|
| ... | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | ... |



多个 2 ?

- 每个 2 单独算
- 为什么是独立的?
- 为了确保时间复杂度, 这个算法还需要数据结构维护, 不过不是重点



Better Productivity (NWERC 2015)

- 工厂里有 n 个工人和 p 条完全相同的生产线。第 i 个人每天上班的时刻为 a_i ，下班时刻为 b_i ，你的任务是给每个工人分配一条生产线，使得生产线的生产力之和最大。生产线的生产力等于每天该生产线所有工人都在的时间长度（不能为 0）
- $1 \leq n, p \leq 200$, $1 \leq a_i < b_i \leq 100,000$ 。输入保证有解



分析

- 每个工人的时间抽象成一条线段，那么本题就是要把线段分成若干个集合，每个集合的交集不为空，且交集的长度之和最大
- 一堆线段，很容易想到排序以后贪心或者动态规划，并且往往重点在于：如何处理相互包含的线段
- 仔细一想：越交越小，所以如果 A 包含 B ，那么 A 和 B 放在一组是很不划算的，不如一开始就单独拿出去



拼凑的算法

- 排除了包含别人的特殊线段，dp 就可以成立了：决策就是“要不要另开一个集合”
- 那么特殊线段怎么办？当然可以扔到被它包含的线段所在集合中，相当于直接浪费了。也可以单独构成一个集合
- 正好 dp 一次可以算出好多值，拼一下就行了



解法

- DP 状态: $d(i,j)$ 表示前 i 条线段划分成 j 个组
- 枚举 t , 表示 DP 只分成 t 个组, 剩下的 $p-t$ 个组 **每个组是单独的一条特殊线段**, 则最终答案为
- $\max\{dp(n_1,t) + \sum \text{最长的 } p-t \text{ 条特殊线段的总长}\}$



Game of Chance (WFI 2020)

- n 个人进行抛硬币比赛，第 i 个人的幸运值为 a_i ($1 \leq a_i \leq 10^9$)
- 幸运值分别为 x 和 y 的人比赛，前者胜利的概率为 $x/(x+y)$
- 第一轮先算出一个 k ($1 \leq k \leq n$) 使得 $n-k/2$ 为 2 的整数幂 (可以证明 k 唯一)，然后前 k 个人先打一轮，之后每一轮的参赛人数都是 2 的整数幂了
- 先按幸运值从小到大排序成 p_1, p_2, \dots ，然后 p_1 和 p_2 打， p_3 和 p_4 打，以此类推。求每个人最终获胜的概率。 $n \leq 300,000$



分析

- 概率计算的特点就是，根据定义和常见公式往往要算好多好多东西的各种搭配，经常得用卷积等数学工具
- 比如我们要算结点 v 的左子树 L 里的选手 A 获胜的概率。假设已知它的幸运值为 x ，在 L 胜出的概率为 p ，右子树的 m 个人，幸运值分别为 b_i ，概率分别为 q_i ，则 A 在 v 里胜出的概率为 $pxf(x)$ ，其中

$$f(x) = \sum_{i=1}^m \frac{q_i}{x + b_i}.$$



计算

- 直接计算的话，这个式子需要 $O(m)$ 时间。这还只是算一个选手 A 的。那怎么办？注意，对于子树 L 里的其他选手，这个 $f(x)$ 是完全一样的，所以我们实际上是要要求一个函数在多点取值的结果
- 多点取值有一些通用方法，但是这道题的话 ...



旗鼓相当才有意思

- 直觉上，两个幸运值差不多的选手比拼，胜率是差不多的；但如果幸运值差别比较大，一方的胜率会很大。题目说的 a_i 可以高达 10^9 ，如果一个幸运值为 1 的和幸运值为 10^9 的比 ...
- 所以感觉近似方法会很靠谱。而且是重点计算**幸运值接近**时的胜率。想到了：邻域、泰勒展开



泰勒展开

$$\begin{aligned} f(x_0 + \Delta x) &= \sum_{i=1}^m \frac{q_i}{x_0 + \Delta x + b_i} = \sum_{i=1}^m \frac{q_i}{x_0 + b_i} \cdot \frac{1}{1 + \frac{\Delta x}{x_0 + b_i}} = \sum_{i=1}^m \frac{q_i}{x_0 + b_i} \sum_{k=0}^{\infty} \left(\frac{-\Delta x}{x_0 + b_i} \right)^k \\ &= \sum_{i=1}^m \frac{q_i}{x_0 + b_i} \sum_{k=0}^{\infty} \left(\frac{x_0}{x_0 + b_i} \right)^k \left(\frac{-\Delta x}{x_0} \right)^k = \sum_{k=0}^{\infty} c_k \left(\frac{-\Delta x}{x_0} \right)^k, \end{aligned}$$

where

$$c_0 \geq c_1 \geq \dots \geq c_k = \sum_{i=1}^m \frac{q_i}{x_0 + b_i} \left(\frac{x_0}{x_0 + b_i} \right)^k$$



覆盖

- 可以看到，如果 $|\Delta x| \leq x_0/2$ ，则只要算 50 项，相对误差约等于 $(1/2)^{50} < 9 \cdot 10^{-16}$ 。
- 这很好，因为 x_0 很大的时候，我们计算一次 $f(x_0)$ 就能适用于一个很大的区间。如果取计算点 1, 3, 9, ...，可以覆盖所有 $x \geq 1$ 的值！
- 总时间复杂度为 $O(n \log n * \log(\text{eps}) * \log(\max_a))$



精度够吗

- 设 e_d 表示对于一棵深度为 d 的树，计算某个 player 能在树里夺冠的概率的最大相对误差，则 $(1+e_{d+1}) \leq (1+e_d)^2(1+\text{eps})$
- 因此

$$e_d \leq (1 + \text{eps})^{2^d - 1} - 1 = O(n \cdot \text{eps})$$



中国计算机学会
China Computer Federation



第三部分、细化：全面性、约束和冲突



题目

- 例 22. Chess Pieces
- 例 23. Kitesurfing
- 例 24. Date Pickup
- 例 25. Maximum enjoyment
- 例 26. Scenery
- 例 27. Pipe Stream
- 例 28. Son of Pipe Stream



Chess Pieces (IPSC 2015)

- 标准国际象棋规则（8x8 棋盘，标准初始位置，标准走子规则，包括吃过路兵和王车易位）
- 要求让棋盘上的车总数（包括黑白）最大

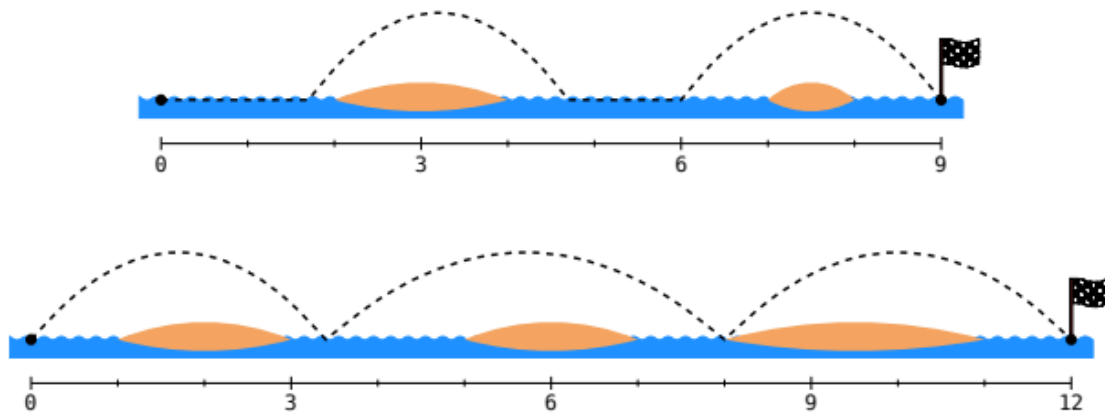
分析

- **最好情况**就是所有 4 个车都没被吃，然后所有 16 个兵都升变成车，一共就是 20 个
- 可能吗？



Kitesurfing (NWERC 2019)

- 你参加了一个风筝冲浪比赛。游泳的速度为 1 米 / 秒，跳跃的距离不超过 D 米，起点和终点都不能在岛上（边缘可以），且总是花 t 秒（和距离无关）。不能在岛上游泳。
- 求从起点到终点的最短时间。单座岛屿的长度不超过 D 米，因此一定有解。一共有 n 座岛屿，相邻岛屿没有公共点。 $n \leq 500$





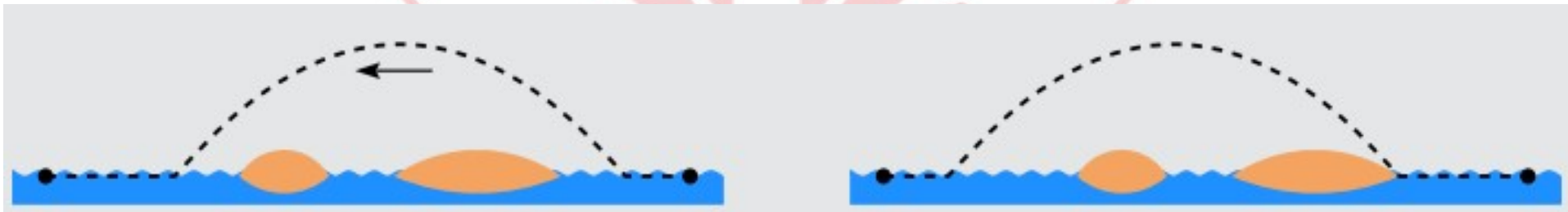
问题

- 难点依然是：决策无限，难以提前离散化
- 只关注岛屿的端点是不够的
- 我们需要全面的思考，不要漏掉任何情况
- 具体来说，跳跃的起点和终点需要满足什么性质？



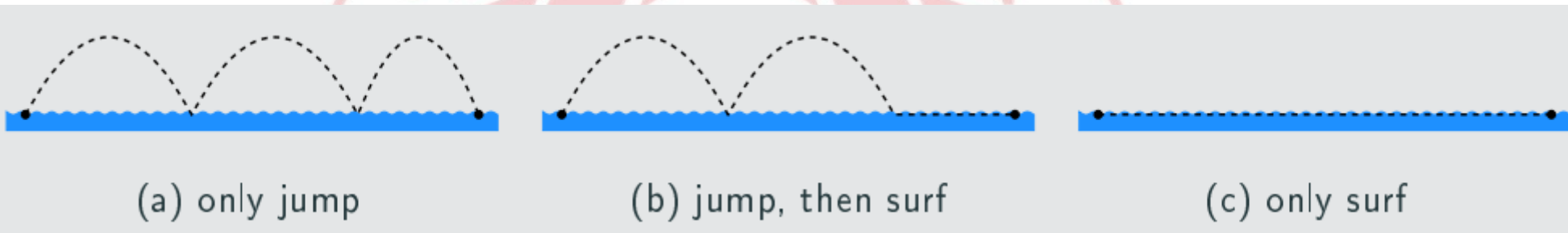
跳跃的起点和落点

- 会从什么地方起跳？跳到什么地方？
- **规范化**！如下图，可以往左移动，使得每一段 surf 的终止处要么在某个岛屿终点左边 D 米（称为**起跳点**），要么是终点线



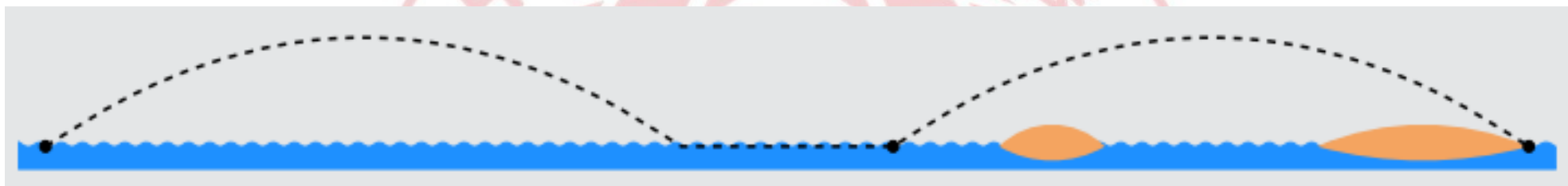
没有岛屿的时候呢？

- 跳和游泳的结合。再次使用规范化，所有的跳跃放到最前面。注意只有最后一步可能小于 D 米



决策

- 在一个位置有两种决策
 - 游到某个起跳点，然后跳，正好落到岛屿边缘



- 一直跳，跳过一个岛屿。落脚点可以在常数时间求出





动态规划

- 访问到的状态数为 $O(n^2)$
- 总时间复杂度 $O(n^3)$



Date Pickup (NWERC 2018)

- Richard 和 Janet 在一个 n 个结点和 m 条有向边组成的城市里约会。Richard 从结点 1 出发去往 Janet 的家（结点 n ），同时 Janet 打电话说她 $a \sim b$ 分钟（不一定是整数）之后就能下楼，**到时候会立刻给 Richard 打电话**。Richard 不喜欢原地等，所以他不会在任何点停留，而是会一直走，等 Janet 一给他打电话就**立刻赶往结点 n** 。
- 你的任务是为他设计一个策略，使得不管 Janet 什么时候准备好，最坏情况下等待的时间最短。



- 如果存在一个实数 w ，虽然 Janet 不需要等 w 分钟，但是对于任意小的正实数 ϵ ，都有可能需要等 $w-\epsilon$ 分钟，那么也算作需要等 w 分钟
- 如果 Richard 经过一个路口的同时 Janet 打电话过来，Richard 来得及重新选择接下来要走的边
- 保证结点 1 能走到结点 n ，并且从每个点出发都有路走。可以证明答案是整数。 $2 \leq n \leq m \leq 100,000$
- 题外话：Richard 的行为不值得提倡。不应该让女朋友等自己



预处理

- 不难想到预处理 s 到所有点的距离，以及所有点到 t 的距离，然后二分答案，判断 e 是否可行
- 首先根据之前提到的“分大类”思想，看看我们需要考虑一个结点 u 的必要条件。最起码，如果 Janet 在最早时刻 a 准备好的话，要来得及赶到。即 $d(s,u)+d(u,t)\leq a+e$ 。把所有这样的 u 放到一个集合 S 里待处理



闲逛舍不得离开

- 对于所有需要考虑的结点 u ，如果存在边 $u \rightarrow v$ ，且 $\text{len}(u,v) + d(v,t) \leq e$ ，则如果 Janet 准备好的时候正在走这条边（包括刚刚开始走），立马直奔目标是来得及的，也放到集合 S 里
- 这样我们就有了一个值得闲逛的子图。如果这个子图有环，那么可以一直在里面呆着等 Janet， e 可行！
- 如果是 DAG 的话用 DP 求出在子图里最多能呆多久。如果可以一直呆到 b 时刻，则 e 可行

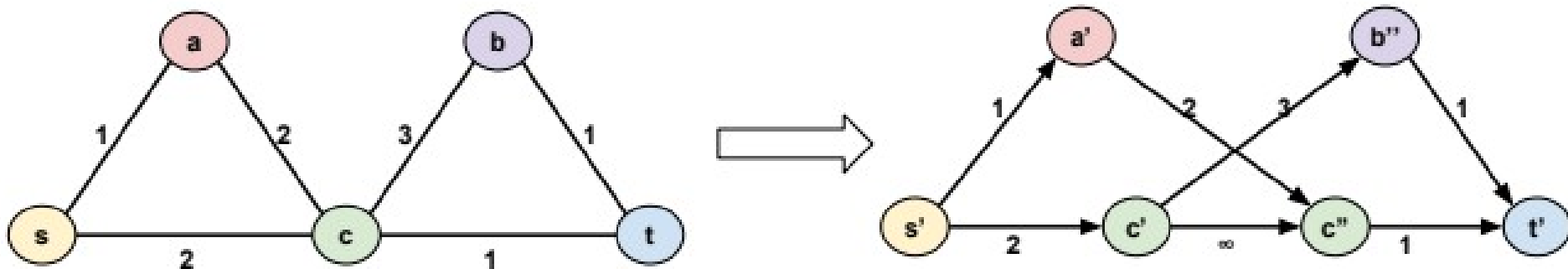


Maximum enjoyment (IPSC 2014)

- 求一个 N 个结点的无向网络的 s - t 最大流，但是流的路径长度不能超过 L 。输入一个 $N \times N$ 对称矩阵，其中 $c_{i,j}$ ($0 \leq c_{i,j} \leq 10000$) 表示边 i - j 的容量限制
- Easy: $L \leq 3$, $N \leq 100$
- Hard: $L \leq 6$, $N \leq 100$

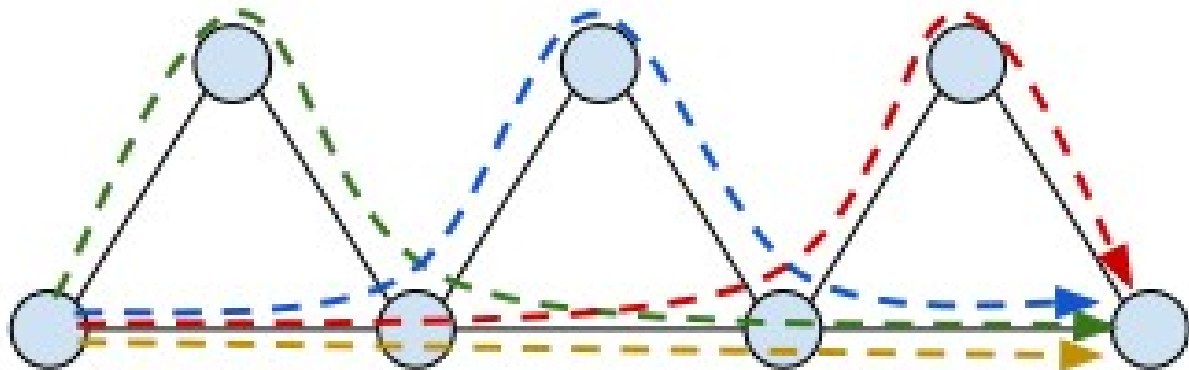
Easy 数据

- $L=1$ 和 $L=2$ 不用多说, $L=3$ 相当于路径上的结点要么是 s, t 要么是 s 或者 t 的邻居
- 拆点直接转化为传统的最大流问题



Hard 数据

- 如果仍然拆点， $L=6$ 会遇到的问题是：一些边会被很多路径共享！一种办法是枚举所有路径，每条路径作为一个变量，列出线性约束：



$$r + \quad + b + y \leq 1$$

$$r + g + \quad + y \leq 1$$

$$\quad + g + b + y \leq 1$$



变量太多

- 模型倒是没问题，变量太多了 ...
- 还是回到最初的拆点法，每个点拆成 $L+1$ 个，每条边都是从第 i 层到第 $i+1$ 层，容量限制变为一个线性约束，变为一个普通的线性规划问题
- 有些细节问题但不太难，这里略



Scenery (WF 2017)

- 你要拍 n 张照片，每张照片均需要 k 个单位的时间。因为不同时刻的景色不同，第 i 张照片必须在 $a_i \sim b_i$ 时刻之内（即开始时刻 s_i 满足 $a_i \leq s_i$, $s_i + k \leq b_i$ ）
- 判断是否有解，输出 yes/no
- $N \leq 10000$



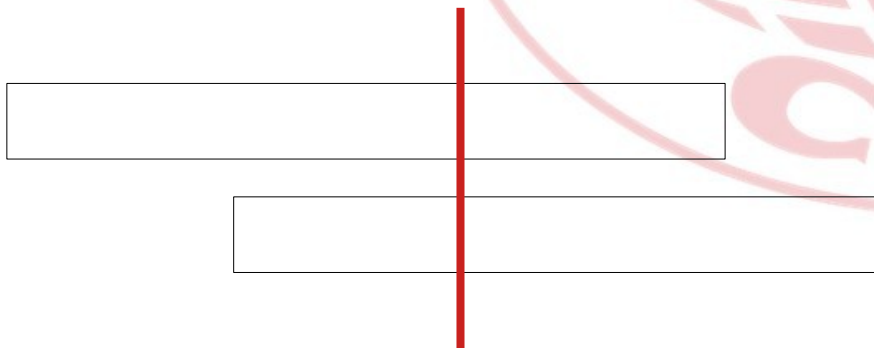
难题预警

- 不要有压力，能理解解法就不错了，赛场上并没有人做出来
- 出到比赛里其实有点不合适，但是平时学习却是一个非常好的思维锻炼



传统贪心好使不？

- 试着排一下序，然后按顺序执行某种贪心决策？
- 比如：对于每个时刻，如果没有照相正在进行，则选一个区间右端点 b_i 最早的照片开始照
- 试着证明一下：好像很容易证明？





WA

- 如果比赛的时候你这样写了，就会 WA
- 问题出在哪里？
- 用 exchange argument 证明的是，如果决定照相，则一定是选择结束时间最短的照，但问题在于，“**能照就照**”不一定是最优的！



照还是不照？

- 能照的时候该不该照相？这取决于尚未考虑过的东西
- 比如 $t=2$ ，第一个照片是 $[0,5]$ ，时刻 0 应该照相吗？
 - 如果照，但第二个照片是 $[1,3]$ ，就糟糕了
 - 如果不照，但第二个照片是 $[2,4]$ ，就糟糕了

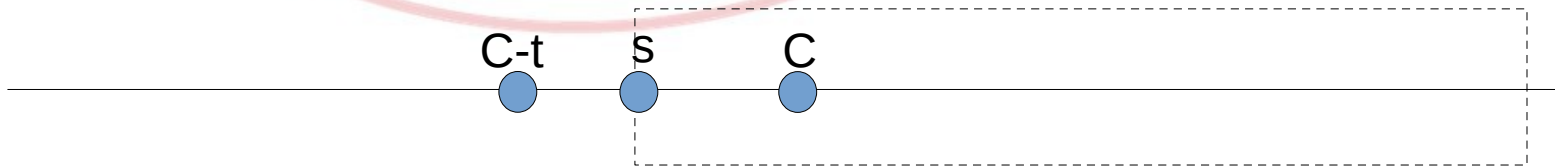


约束

- 这种情况有另外一个思考方式：照片 2 的约束性比较强，先考虑它。
- 比如，照片 2 是 $[1, 3]$ ，显然只有一种照法，因此其他所有照片都不能占用时刻 1，因此开始时间只能 ≤ -1 。反过来说，如果其他照片遵守这个新约束（开始时间 ≤ -1 ），那么照片 2 保证可以照

把这个约束一般化

- 考虑一个区间 $[s, e]$ 和可行区间完全被该区间包含在内的所有照片。在不考虑 $a_i \leq s_i, s_i + k \leq b_i$ 约束的情况下来一个“尽量往后排”的贪心，设排出来的第一个相片开始时间为 C ，如果 $C < s$ ，则原问题无解；如果 $C - s < t$ ，说明有可能有解，但不能受前面的影响。具体来说， $(C - t, s)$ 区间不能作为相片的开始时刻
- 这种放宽约束的方法在搜索剪枝等其他地方也很常用





主算法

- 这样，对于所有可能的 $[s, e]$ ，我们都能算出一个“禁止区间”，或者直接判定问题无解。注意，要按照 **s 从大到小** 的顺序计算这些子问题，这样后面的子问题在贪心的时候可以考虑之前算出来的禁止区域。
- **新贪心**：求所有区间禁止区间，然后依次考虑每个时刻，如果不在禁止区间内，并且空闲，则选一个区间右端点最小的照片开始照
- 这个算法是对的！惊不惊喜？意不意外？



证明

- 如果新贪心找到解，原问题当然有解；我们要证明的是新贪心无解的时候，原问题真的无解。
- 从证明的角度讲，我们可以直接假定不拍照的时刻都在禁止区间的，因为如果该时刻不是被禁止的，肯定是因为“没得拍”。这相当于我们找到了一条分界线，前面已经找到了解，后面是**完全独立**的子问题
- 如果想说的严密一点，就用数学归纳法好了



证明

- 假设失败的是照片 i ，deadline 是 e_i 。如果之前照过的相片的 deadline 都不晚于 e_i ，说明前 i 个照片对于子问题 $[s_0, e_i]$ 无解。因为新贪心一点也没浪费时间，而 i 直到最后都还有机会，可是最终还是没排下
- 如果有相片的 deadline 晚于 e_i ，设其中最晚开始照的是照片 j ，考虑从相片 j 结束之后开始拍的所有照片以及照片 i ，设 s 为这些相片中最小左端点，则子问题 $[s, e_i]$ 无解（待续）



证明 (续)

- 考虑从相片 j 结束之后开始拍的所有照片以及照片 i ，设为集合 PP 。设 s 为这些相片中最小左端点，则 j 开始照的时刻肯定在 s 之前，否则一旦到了时刻 s ，就再也轮不到 j 了
- 考虑子问题 $[s, e]$ ，它至少包含了 PP 。考虑求禁止区间时的贪心，设最后一个安排的相片（其实是最左）的开始时刻为 p ，那么相片 j 结束时间会比 p 早，否则 j 的开始时刻会在禁止区间里！但是如果这样的话，相片 j 并不会碍事，新贪心应该能排完相片 i 才对，矛盾



时间复杂度

- 目前一直没讲得一个问题是如何快速求出所有 $[s, e]$ 的禁止区间。如果暴力求解， s 和 e 的枚举量均为 $O(n)$ ，求禁止区间本身的贪心过程需要 $O(n)$ ，总时间复杂度 $O(n^3)$ ，太高了
- 假设已经做完了一次枚举 s ，所有禁止区间都是以 s 结尾的，显然可以合并起来，得到一个大区间 $(C-t, s)$
- 下一次枚举 $s' < s$ 的时候，可以递推得到。



另一个算法

- 官方解答里提到了另外一个解法：按照时间顺序**暴力求解**，状态是“当前在拍的照片啥时候能完”以及“能拍但还没开始拍”的照片集
- 状态集是指数级的，但是“没有照片正在拍”的状态只会有一个。因为一旦生成了一个新的，和原来的相比一定有一个严格更优，所以不需要保留另外一个。**可以利用这一点得到一个平方时间复杂度的算法**
- “细节留着读者”，我也留给你们吧 :)



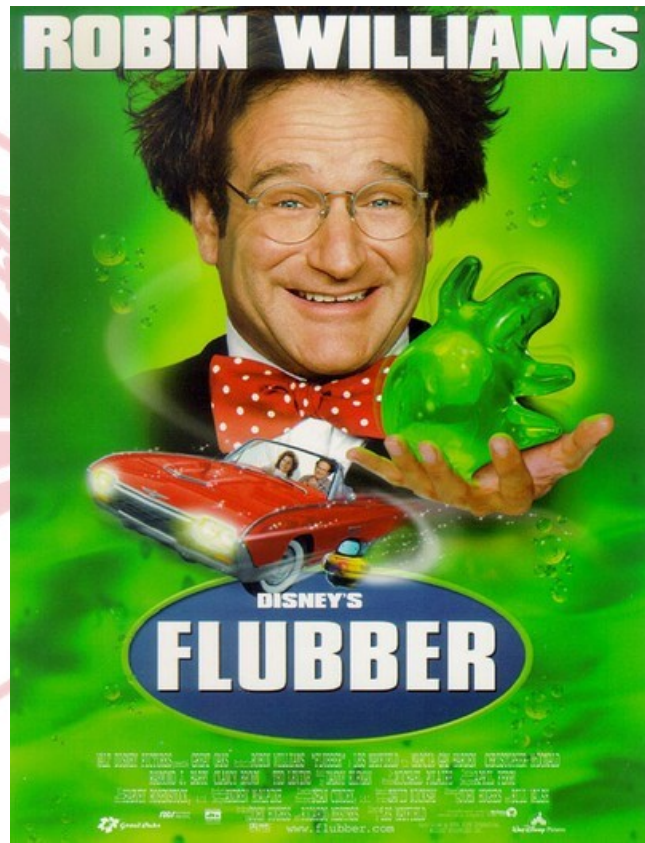
Pipe Stream (WF2015)

- 你想测量 Flubber 在管道里的流速。已知它在 $v_1 \sim v_2$ 之间（单位：m/s），但是我希望更精确的估计，使得绝对误差不超过 $t/2$
- 管道长为 L ，每次可以敲击管道的某个位置 $[0, L]$ ，根据声音确定 Flubber 有没有通过这个位置。每 s 秒可以敲一次，最少敲几次可以完成任务？无解输出 impossible
- $1 \leq L, v_1, v_2, t, s \leq 10^9, v_1 < v_2$.



Flubber 是啥

- 百度百科：《飞天法宝》是由迪斯尼公司发行的喜剧片，由莱斯·梅菲尔执导，罗宾·威廉斯和马西娅·盖伊·哈登等主演，于1997年上映。
- 该片讲述了终日沉浸于科学研究的菲利普博士发明了一种神奇的绿色橡胶，可以抵抗地心引力的故事。

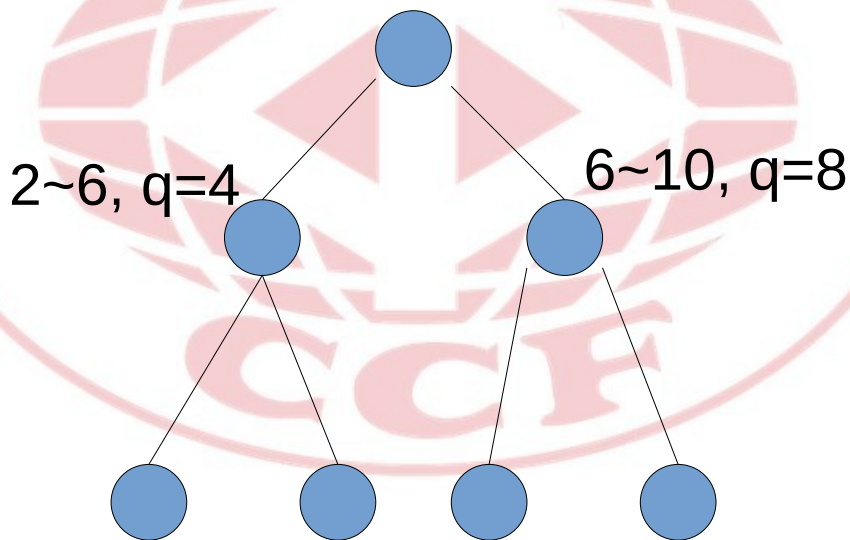




直觉

- 和“猜数字”有点类似，估计是某种形式的二分？
- 但是有一个重大的不同：时间是在流逝的，所以每次查询时所处的世界状态不一样！
- 比较复杂的情况是量子物理里的观测：观测本身改变了状态；本题相对简单一点：观测本身并没有副作用，只是每个询问 v 有一个最晚可能的时间 $\text{Latest}(v)$ 。太抽象？分析例子吧

- 后两个样例比较有意思。 $L=60$ $v_1=2$ $v_2=10$ $t=2$ $s=5$ 的答案是 3, 但 60 改成 59 就是 impossible
- 理想：分 4 个区间： $2\sim 4, 4\sim 6, 6\sim 8, 8\sim 10$, 询问两次即可





- **现实：**能否在第二次敲击时刻 $T=2*5=10$ 询问 $v=8$ ？询问位置 $x=vT=80>60$ ，超了。所以这棵树不合法。
- 怎么办呢？可不可以把询问的位置往左调到 60？反算出询问为 $v=6$ 。由于 6~10 还有两个 t 的范围，接下来的询问肯定比 60 还要大，又超了。
- 第二个可能的方法是把第一层询问往右移，使得右边的区间窄一点，但是面临同一个问题：只要区间不够窄，下一次询问最多只能到 $v=6$ 。因此，唯一的方法是：把询问往右挪到**第一层的右子树只剩一个结点！**



- 不难发现，这个结点只能是 $[8,10]$ ，所以根结点是询问是 8，接下来问题就变成了：已知速度区间是 $[2,8]$ ，当前时刻为 $T=5$ ，接下来应该怎么办？和刚才一样，先考虑理想情况下，询问中点 $v=5$ ，但是右子树区间 $[5,8]$ 不够小，还得分。不难算出第三次敲击 $T=15$ 最多问到 $v=60/15=4$ ，又失败了
- 所以第二次询问只能问 $v=6$ ，使得右子树 $[6,8]$ 是合法的叶子，剩下一个 $[2,6]$ ，询问中点 $v=4$ ，刚刚好！
- 所以这棵树是一棵往左边偏的树



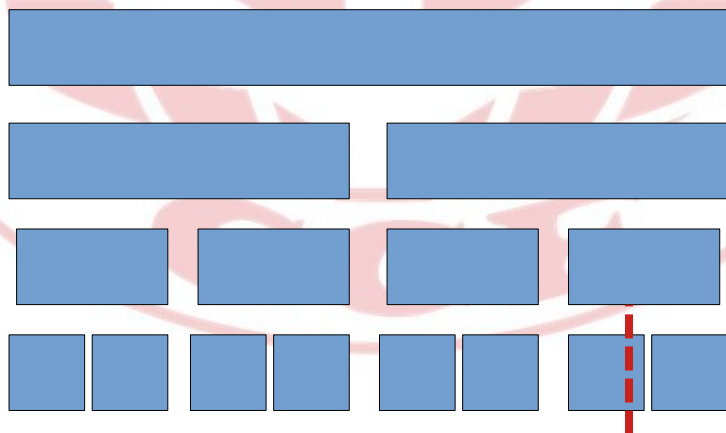
决策树?

- 总的来说，我们是在扩展一棵决策树。每一层都是区间 $[v1, v2]$ 的一个分解，所有的叶子对应的区间长度不能超过 t
- 但是这个决策树可以表达出任意决策吗？可以，因为对于任意一个具体结点，它的深度决定了世界状态，而状态决定了最优决策。所以对于任意结点，都有一个确定性的最优询问点，只不过不太容易计算罢了



一般情形

- 刚才那个例子比较极端，事实上可能前几层都比较平衡，直到有一层出现了一条虚线，虚线右边的询问都无法完成。所以虚线右边必须全是单位区间，个数可以算出来。比如 3 个，那么我们就得把右边 3 个结点往右压缩





压缩结点之后呢

- 压缩结点之后，这一层的最右边 3 个结点再也不能扩展了，但是也有一个好处就是虚线右边的速度都不用问了。我们扩展决策树的状态可以用三个数描述：已询问次数 k ，可扩展区间数 $n(k)$ ，尚未确定速度的右边界 $v(k)$ ，不难想象，每扩展一层，就是在递推 $n(k+1)$ 和 $v(k+1)$
- 初始时 $v(0)=v_2$, $n(0)=1$



- 第 $k+1$ 次敲击询问的最大速度为 $v_f(k)=L/(s(k+1))$
- 虚线位置在 $v_f(k)+t$, 右边单位区间个数
 $n_f(k)=\text{ceil}((v(k)-v_f(k)-t)/t)$
- 剩下的区间下次会一分为二, 因此
 $n(k+1)=2(n(k)-n_f(k))$, 然后 $v(k+1)=v(k)-n_f(k)*t$
- 从小到大枚举 k , 如果 $n(k)\geq(v(k)-v_1)/t$ 则输出 k (平均分割就行) ; 如果 $n(k)\leq 0$ 则输出无解



开始求解

- 首先进行归一化，以后就假定 $s=1$ 了
- 根据刚才的讨论，似乎重点是以 v_1 开头的区间。设 k 次敲击后的速度区间是 $[v_1, v(k)]$ ，并且分成了 $n(k) \leq 2^k$ 份
- 最简单的情况是
- 敲击一次以后会怎样？



Son of Pipe Stream (WF 2017)

- 有一个 n 个点， p 根管道（弧）的网络， Flubber 从点 1(Flubber factory) 流到点 3(Flubber Department, FD)， 水从点 2(水源) 流到点 3。
- 同一根管道里可以既有水又有 Flubber， 但方向必须一致， 并且有容量限制 $v \cdot f + w \leq c_i$ ， 其中 f 和 w 分别为该管道内 Flubber 和水的流速 (liters/second)， v 是 viscosity 系数（粘度）



- 要求流入 FD 的总流速 $F+W$ 最大，在此前提下 $F^a \cdot W^{1-a}$ 最大。要输出方案
- $n \leq 200, n-1 \leq p \leq n(n-1)/2$
- $1 \leq v \leq 10, 0.01 \leq a \leq 0.99$, v 和 a 为实数
- 每条管道用三个整数 j, k, c_i 表示, $1 \leq j < k \leq n, 1 \leq c_i \leq 10$
- 所有约束 (Flubber 和水不得反向, 流量平衡, 容量限制, 方案和最大值的一致性) 允许绝对误差 10^{-4}



简要分析

- 同一根管道里流两种液体，感觉很麻烦 ... 能统一处理吗？好像很困难，因为
 - 源不一样
 - 还有一个粘度系数
- 其实粘度系数是吓唬人的。设 $f' = v * f$ ，则容量限制为 $f' + w \leq c_i$ ，目标为 $F^a * W^{1-a} = (F'/v)^a * W^{1-a}$ ，它和 $F'^a W^{1-a}$ 同时取到最大值。以后直接认为 $v=1$



最大流?

- 假设我们已经拿到的最优解，那么从流量的角度讲， $F+W$ 就是忽略液体类型时的最大流吗？
- 感觉像是，但好像又不是那么显然。官方解答里没证，能搜到的网上的题解也都没证。
- 尝试证明一下吧。假定在最优解基础上，如果忽略液体类型时还可以增广，不失一般性地，我们假定增广路是从水源开始的



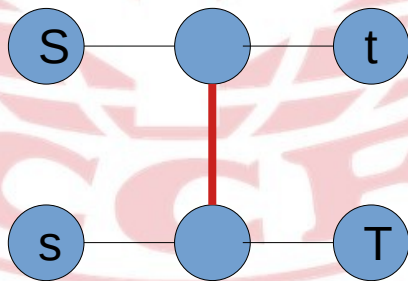
最大流

- 如果增广路都是正向边，那么直接把增广成水流就可以了，和最优解矛盾
- 如果增广路里有反向边呢？如果反向边是水流，那就退流，和普通的最大流算法一样。麻烦的情况是：遇到 Flubber 的反向流怎么办？直觉上，因为二者的目的地相同，似乎没有理由反向移动？



如果目的地不一样？

- 想到这里，我们不禁好奇（反正我是好奇了）：如果题目里二者的目的地不相同，结论仍然成立吗？很容易找到反例：





证明

- 这个反例也启发我们证明的思路：我们先临时允许两种液体在同一个管道 P 里反向移动，然后**交换后半段路径**！因为目的地一样，仍然是合法解。然后 P 里现在最多只有一种液体了



流量分配

- 现在我们已经求出了最大流 Z ，那么最优解 F 和 W 满足 $F+W=Z$ 。不难发现目标函数 $f(F)=F^a(Z-F)^{1-a}$ 在 $F=aZ$ 时取到最大值（实在不会算极值的话三分也行）。
- 遗憾的是， F 并不是随便取的。如果我们再求一下两种液体从各自源出发的最大流 F_{\max} 和 W_{\max} ，显然 F 不能超过 F_{\max} 。类似的， W 也不能超过 W_{\max} ，所以 F 的区间为 $[Z-W_{\max}, F_{\max}]$ 。



理想情况下

- 理想情况下，在这个区间里的 F 值都是合法解，所以三分还是对的。如果你数学比较好，算一算就会发现 F 如果取不到 az ，最优解应该是取两个区间端点中更接近 az 的那一个
- 不过这只是理想情况。我们还得证明：任意满足这个区间里的 F ，都能构造出合法的流！



直接构造

- 反正这题要输出方案，就不用特意证明了，直接找构造的方法吧。建一个超级源，分别向 Flubber 源和水源连一条容量为 F 和 W 的边，求最大流。这样求出的流对应最优解的“混合流”。如何把两种液体分开呢？再构造一个网络，每条边的容量等于混合流的流量，求 F 出发的最大流即可



题目讲完了

- 更多证明和问题求解的技巧可以参考 tricki
- “a Wiki-style site with a large store of useful mathematical problem-solving techniques”
- <http://www.tricki.org/>



中国计算机学会
China Computer Federation



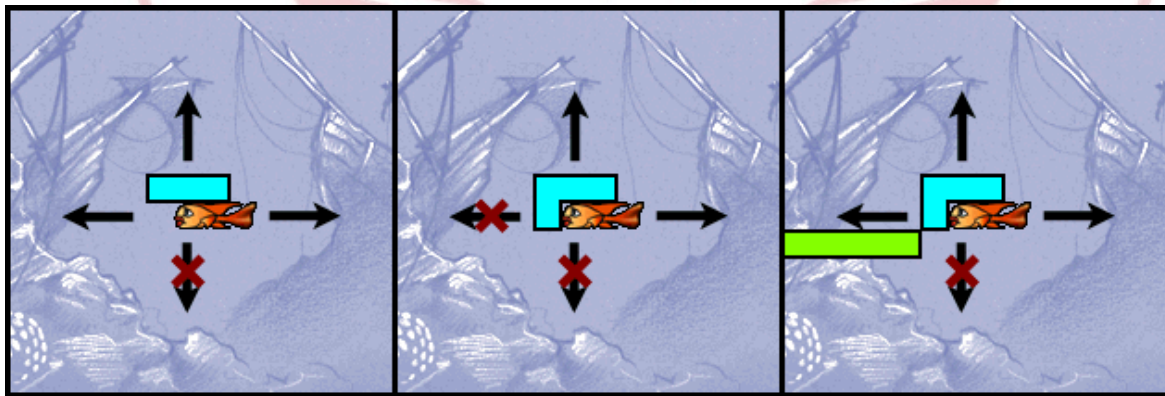
尾声 游戏和现实



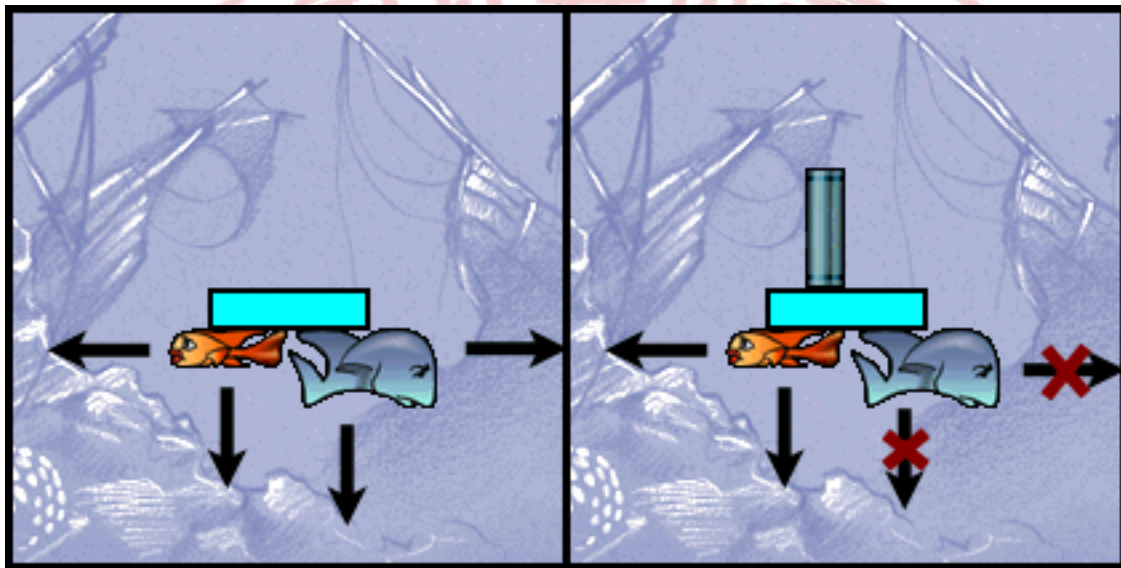
Fish Fillet (IPSC 2009)

- 玩一个智力游戏，任务是帮助大小两条鱼逃离一个网格房间
- 两条鱼都可以朝四个方向移动，每步可以选择移动一条鱼，移动之后它可能会承载 (carry) 物体或者推 (push) 物体，也有可能会有物体失去支撑和掉落。
- 要注意不要被下落的物体砸到
- 原题有三关，样例 (f0)，简单数据 (f1)，困难数据 (f2)

- 悬空的物体本**不会掉下来**（左图），但一旦鱼开始支撑 (support) 这个物体，它就变成活动的，**鱼一离开就会掉下来**。所以鱼不能推一个仅由它自己支撑的物体（中图）。但如果还有其他物体支撑，**就可以推**（右图）。小鱼不能直接或间接的支撑右边的钢条



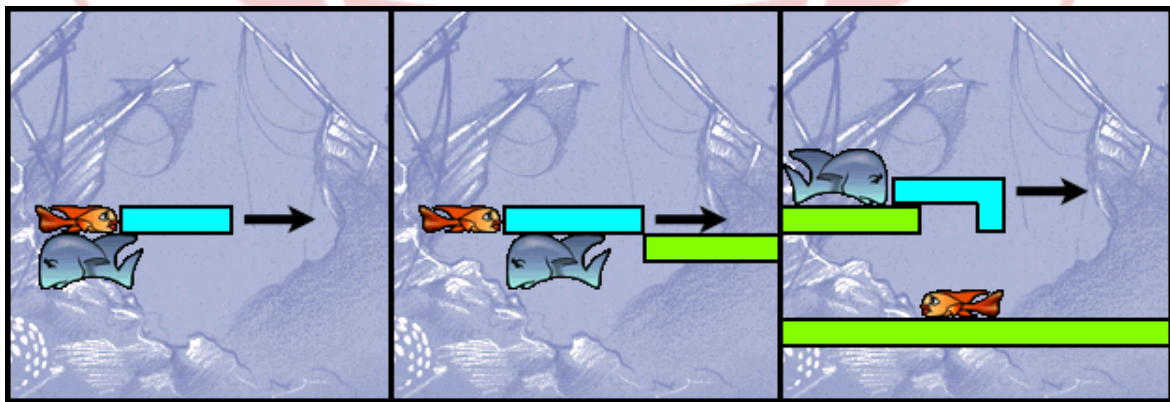
- 两条鱼可以交换支撑的物体。注意，有钢条的情况下可能只有小鱼能离开，大鱼不能离开



- 可以推没有被鱼支撑的物体（左图）
- 推完物体之后，这个物体不能直接落到同伴身上（中图、右图）。不过在中图中，小鱼可以绕到右边去往左推 L 形物体



- 不管多小的物体，只要往下落就很危险。只要它碰到鱼或者鱼支撑着的某个物体，你就输了
- 要想推一个被鱼支撑的物体，要么推完之后它会掉到空地，要么会被另外一个物体所支撑（左图、中图）。注意右图，那个其实没有砸到鱼





中国计算机学会
China Computer Federation



样例数据 (f0)



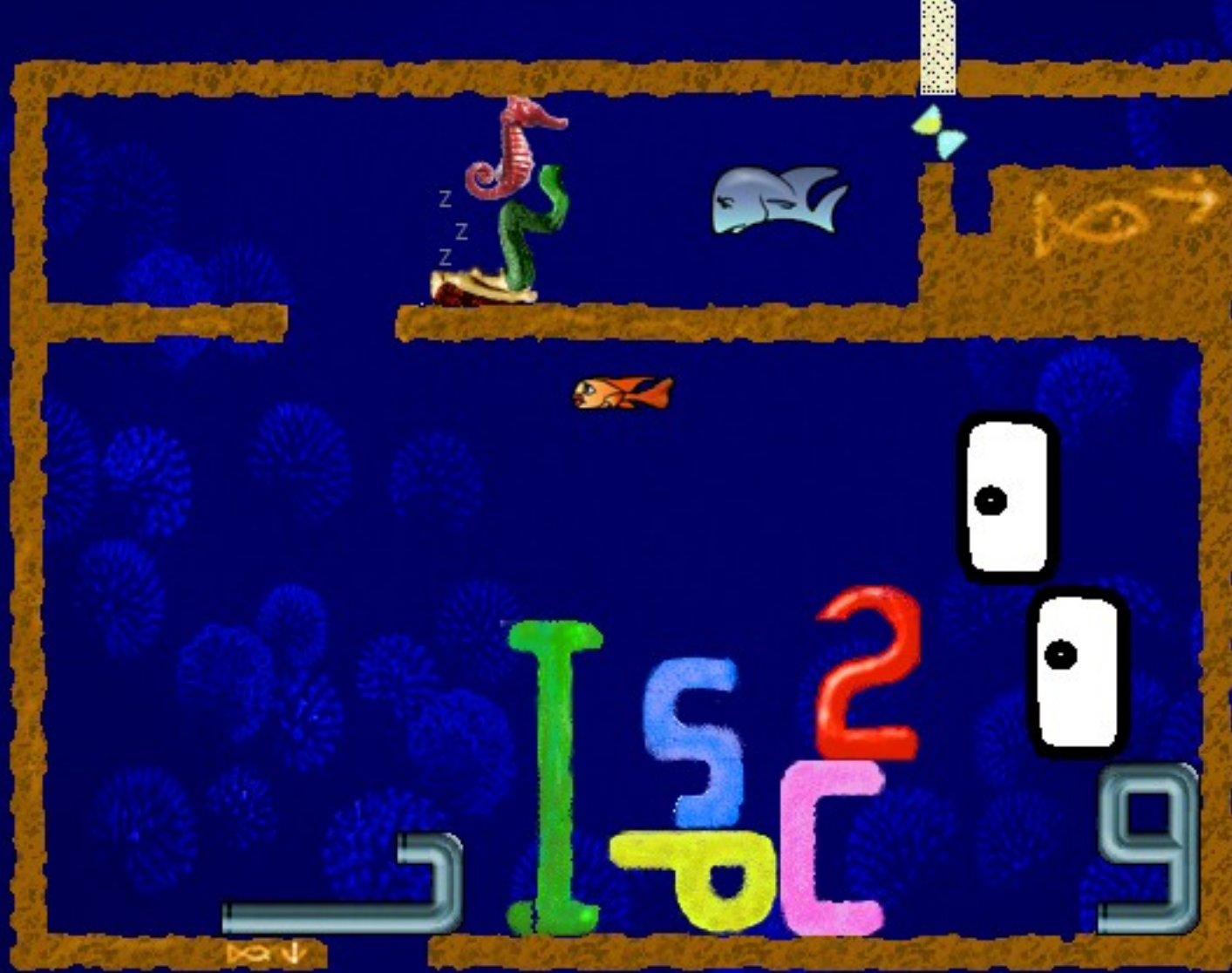


中国计算机学会
China Computer Federation



“简单”
数据 (f1)





困难数据 (f2)



不尽兴?

- Fish Fillets - Next Generation
- <http://fillets.sourceforge.net/>
- 还有一个变种 Fish Fillets Clone:
- http://www.olsak.net/mirek/ff-clone/index_en.html



“官方题解”

- Implementing even the simplest possible state space search for this game is quite a lot of work, and it is not even worth the effort – the number of possible states is **so huge that there is no chance** it will find a solution.
- Instead, the recommended way to solve this task was to use the applet to play, and to **use your head** to make the moves



规划问题

- Planning 是传统人工智能的重点研究对象
- 我从小就对这个很感兴趣
- 我的 IOI 国家集训队论文《搬运工问题的启示》就是研究了一个经典的 Planning 问题：Sokoban
- 可惜后来没时间再研究了



中国计算机学会
China Computer Federation



后来我看到一篇论文

Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)

Solving Hard AI Planning Instances Using Curriculum-Driven Deep Reinforcement Learning

Dieqiao Feng , Carla P. Gomes and Bart Selman

Department of Computer Science
Cornell University

{dqfeng, gomes, selman}@cs.cornell.edu



然后

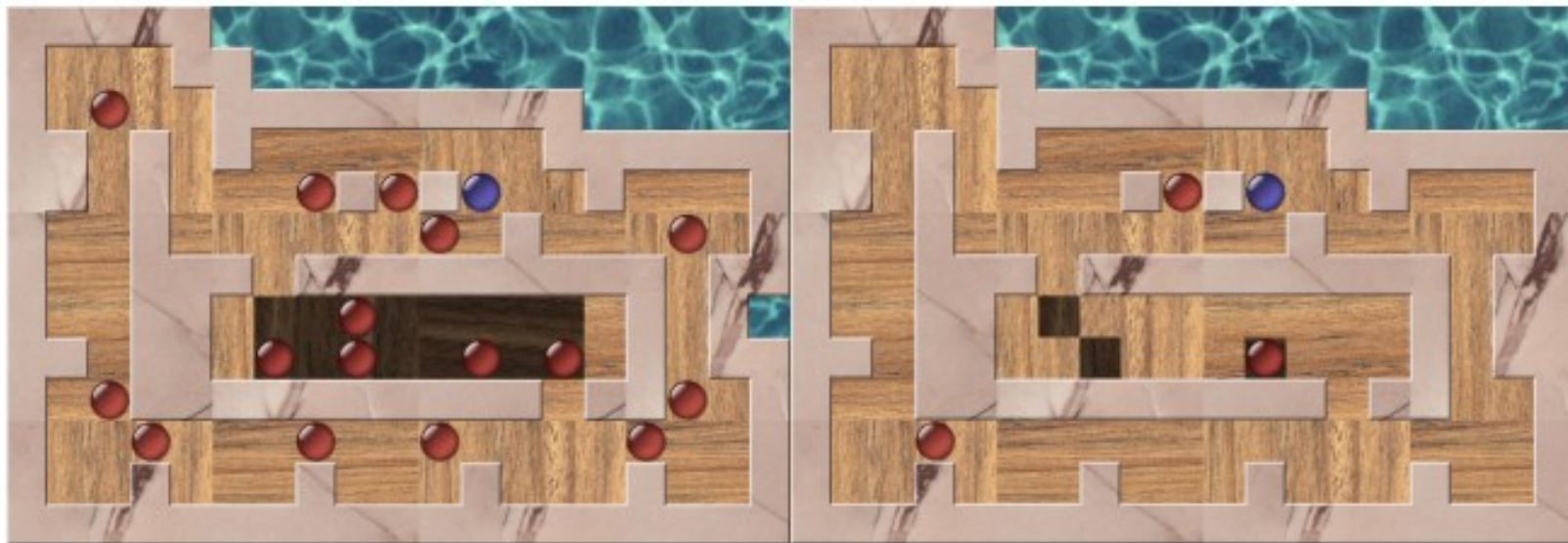
- 我和冯迭乔联系了一下，征得他的同意在冬令营上向大家介绍他的方法
- 这里只介绍这篇论文里的方法，后续还有一篇论文，有兴趣的同学自己搜吧
- 他的论文研究的正是我当年研究过的推箱子游戏，和本题有很多相似之处



从 SAT 获得的启发

- CDCL(**conflict**-driven clause learning) 框架
- “The key insight of their success is the ability of the algorithm to learn problem **invariants** and **reshape the search space** by avoiding entering subtrees which do not contain a solution.”
- 专心求解一个问题

- 核心算法是常用的 Monte Carlo tree search (MCTS)



- 和通用 planner 与专业 sokoban 求解器相比效果非常好
- 后续论文更精彩

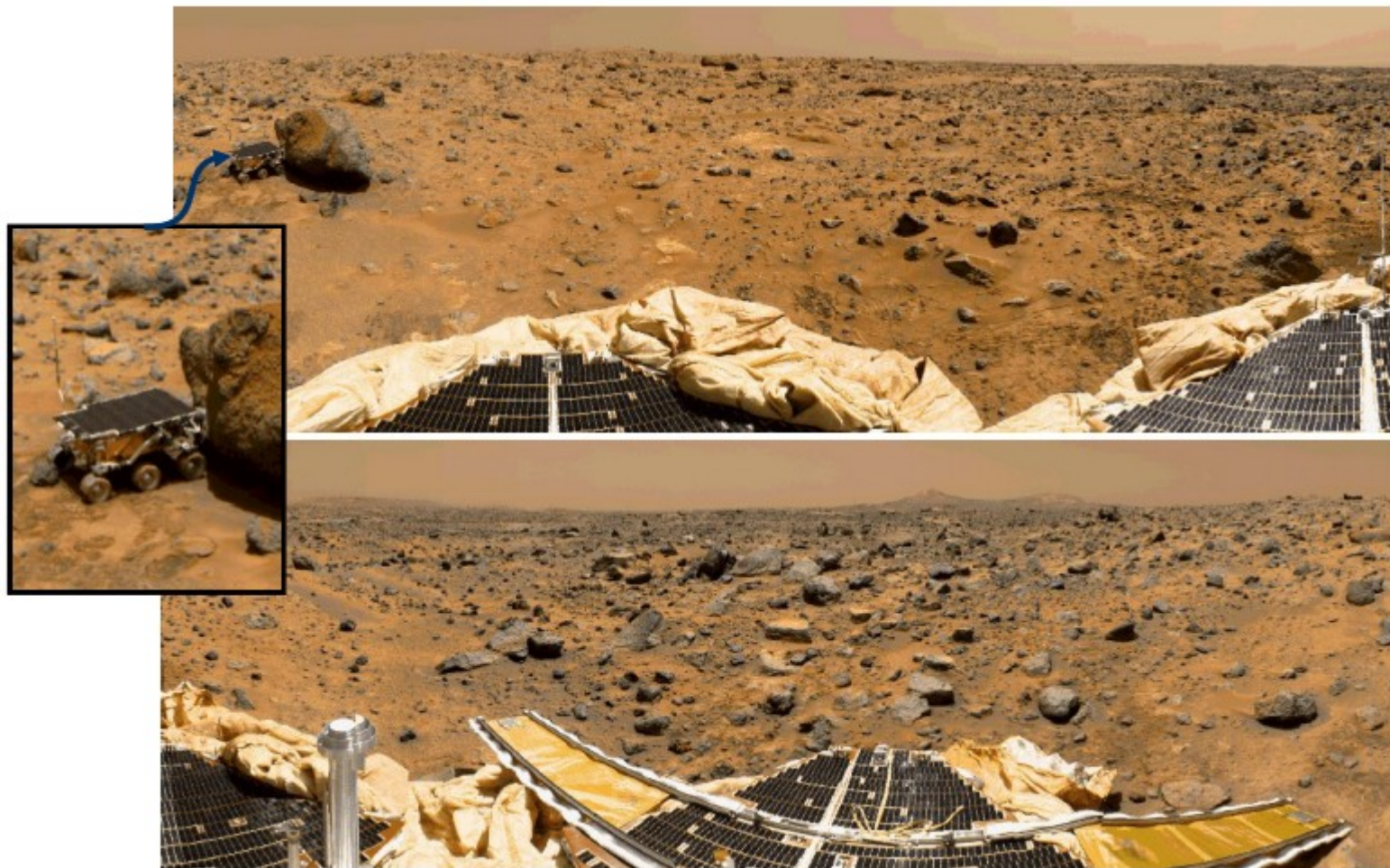


What Really Happened on Mars?

- 这是 ACM/ICPC 总决赛 2016 年的一道题目。题目本身只是一道模拟题，但是背景却不一般
- 网上能搜到的最早资料是 NASA 的 Glenn E Reeves 在 1997 年 12 月 15 日群发的邮件《What really happened on Mars?》，后面又有公司和大学写了文章和课件，很有教育意义
- 接下来的一些课件参考了原始群发邮件和 Jim Huang 的讲义《Priority Inversion on Mars》



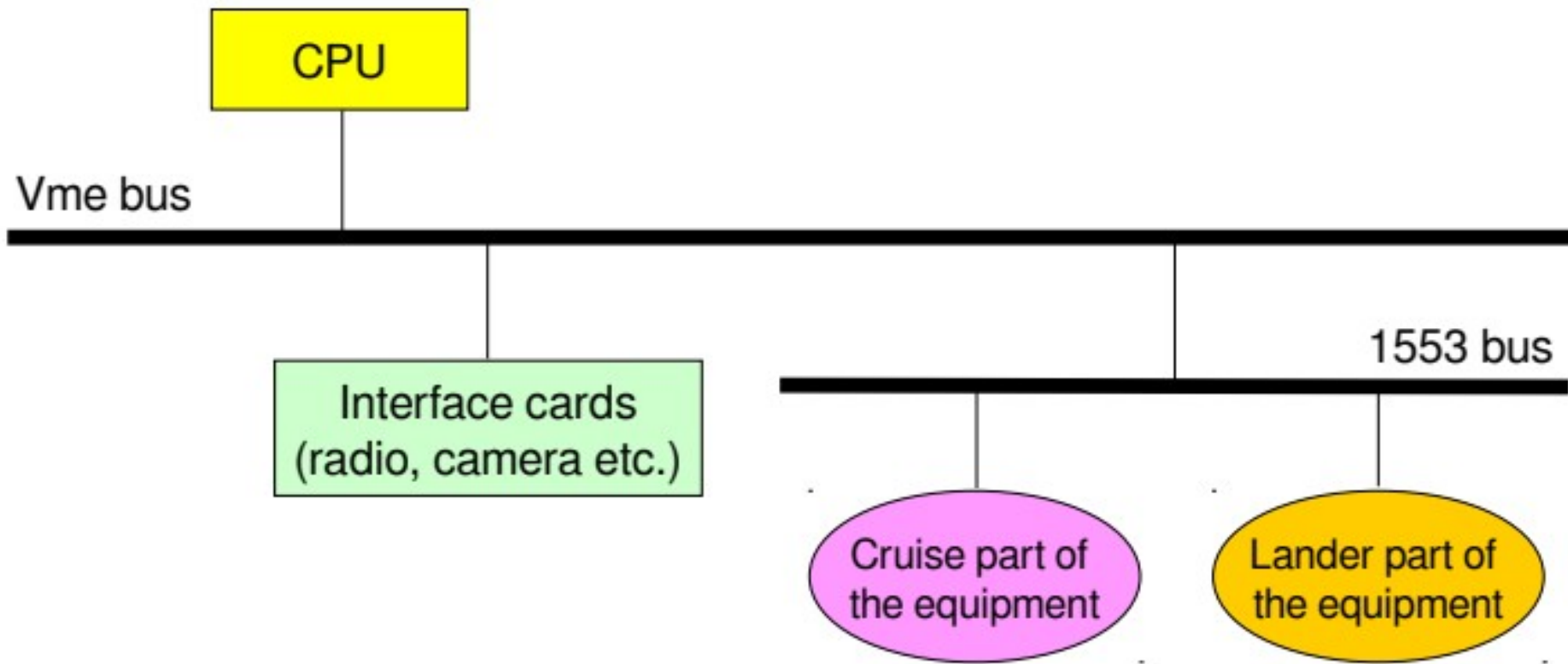
中国计算机学会
China Computer Federation





问题

- Pathfinder 在搜集气象 (meteorological) 数据时经历了几次系统重置 (total system resets), 导致数据丢失
- JPL 工程师尝试在实验室里重现问题。18 小时之后, 故障重现, 原因找到, 问题很快得到解决
- 到底什么原因? 为什么没在发射之前找到问题呢?





硬件简介

- 单个 CPU，控制 1553 总线和上面的设备。
- 软件 task 运行的频率**固定是 8Hz**。
- 总体顺序是 bus_sched 设置 transaction 然后 bc_dist 搜集数据。
- 如果下周期的 bc_sched 开始执行的时候**上周期的 bc_dist 还没结束**，系统就会重启

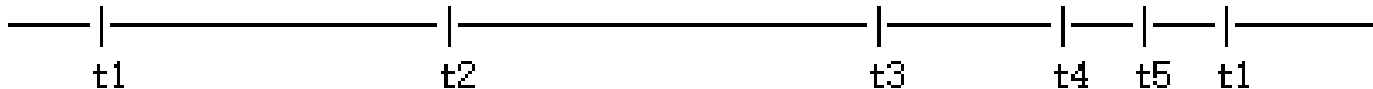


- t1: 硬件开始一个新的周期
- t2: 1553 traffic 结束, bc_dist 被唤醒
- t3: bc_dist 结束数据搜集任务
- t4: bc_sched 被唤醒, 开始设置下个周期的 transaction
- t5: bc_sched 结束

|< ————— .125 seconds —————>|

|<*****| *****| **>|

|<- bc_dist active ->| bc_sched active
|<-bus active ->| |<->|

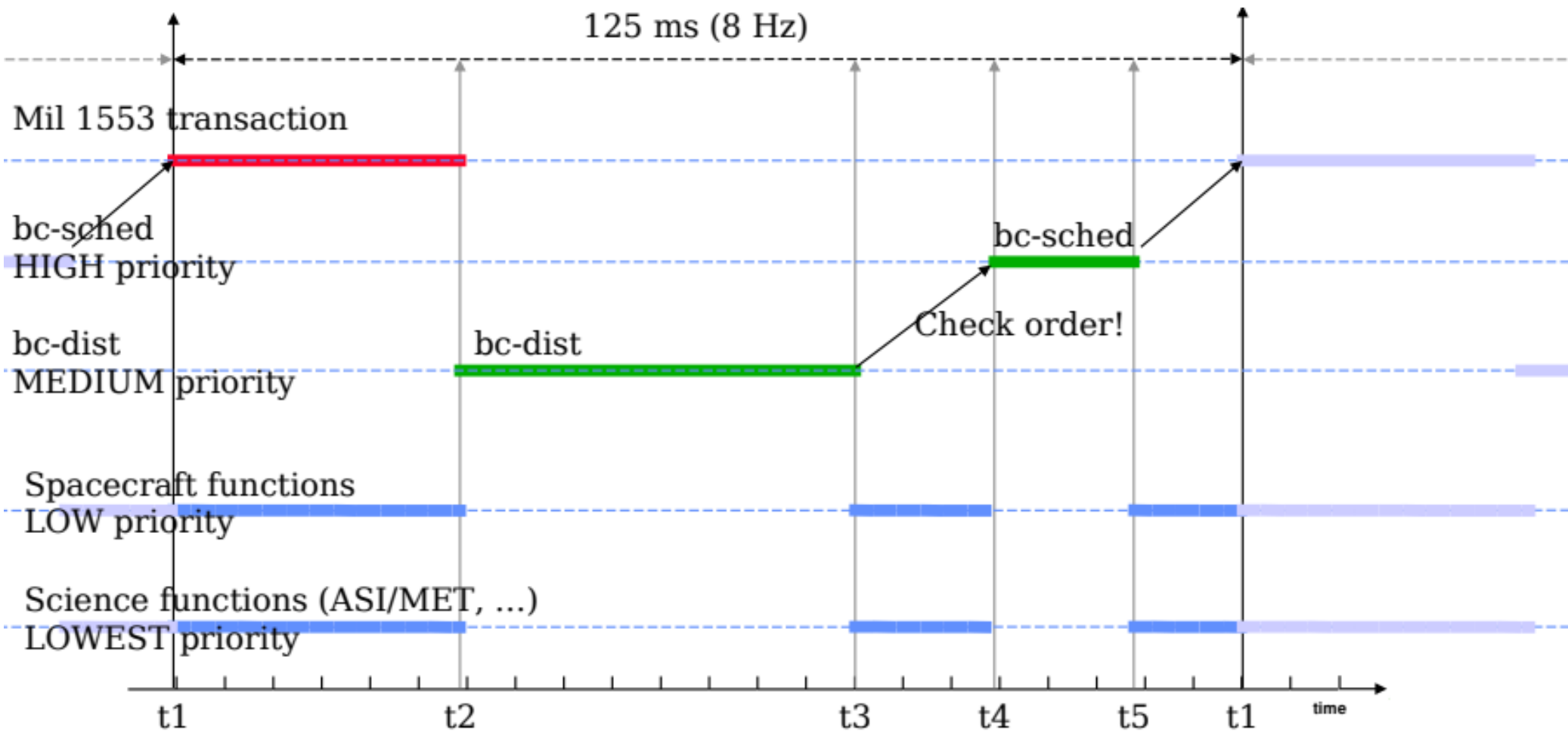


星号是指除了 bc_sched
和 bc_dist 的其他任务



优先级

- 抢占式多任务，在 RTOS 里很常见
- bc_sched task 是系统里最高优先级的（除了 vxWorks 的 tExec 任务），bc_dist 是第三高的（第二高的是控制 entry/landing 的）
- 其他控制 spacecraft 得功能优先级要低一点，而 science 功能（摄像、图像压缩，ASI/MET 任务）优先级更低





理想和现实

- 设计思路是这样的：bc_shed 和 bc_dist 需要在固定的时间点开始执行，执行时间也不能太长（否则会被 watchdog 发现，重置系统），设为高优先级
- 有些任务运行时间长，但是不需要 real-time，有空就运行一点，必要的时候被高优先级的任务抢占，两不误。很合理啊！



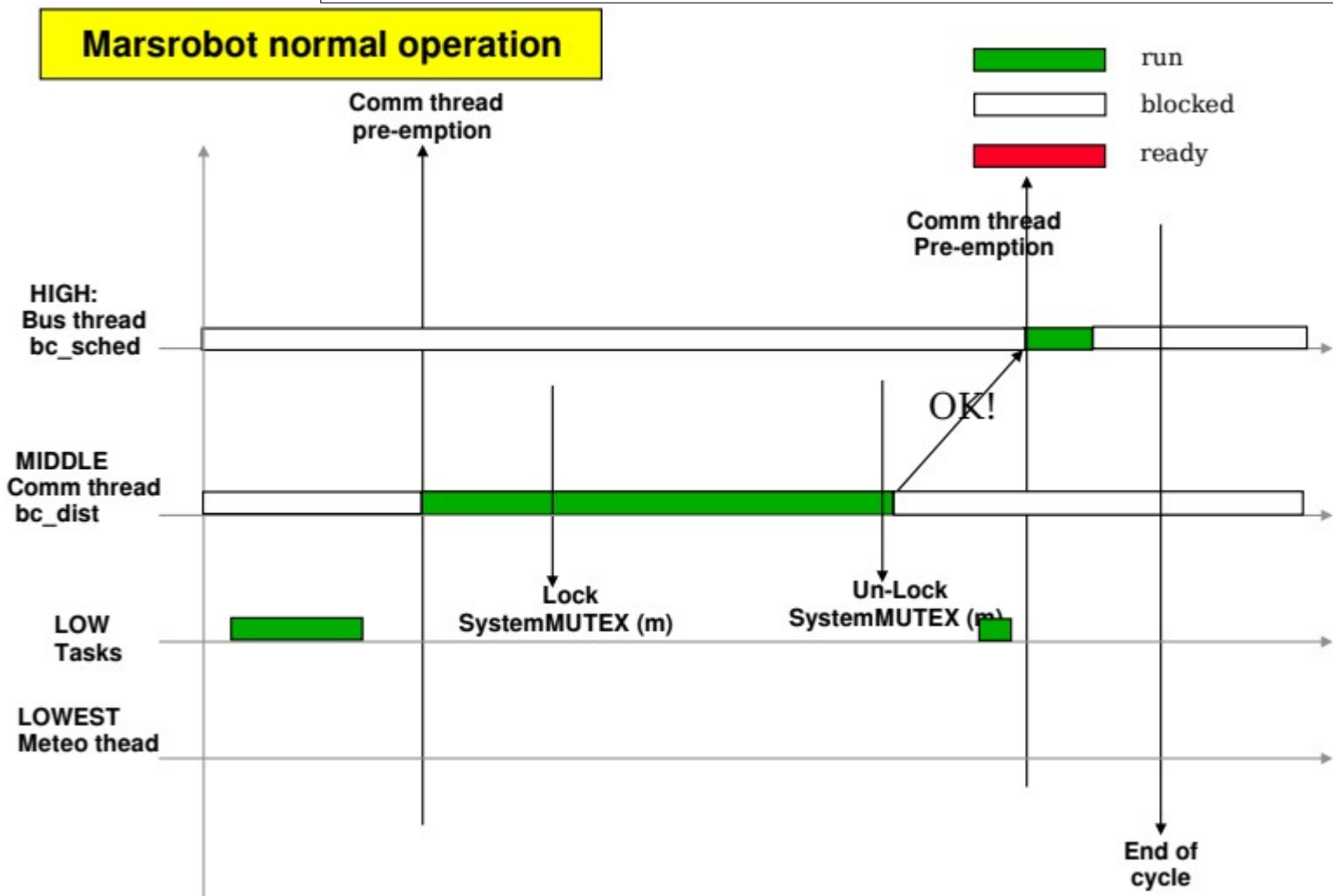
ASI/MET

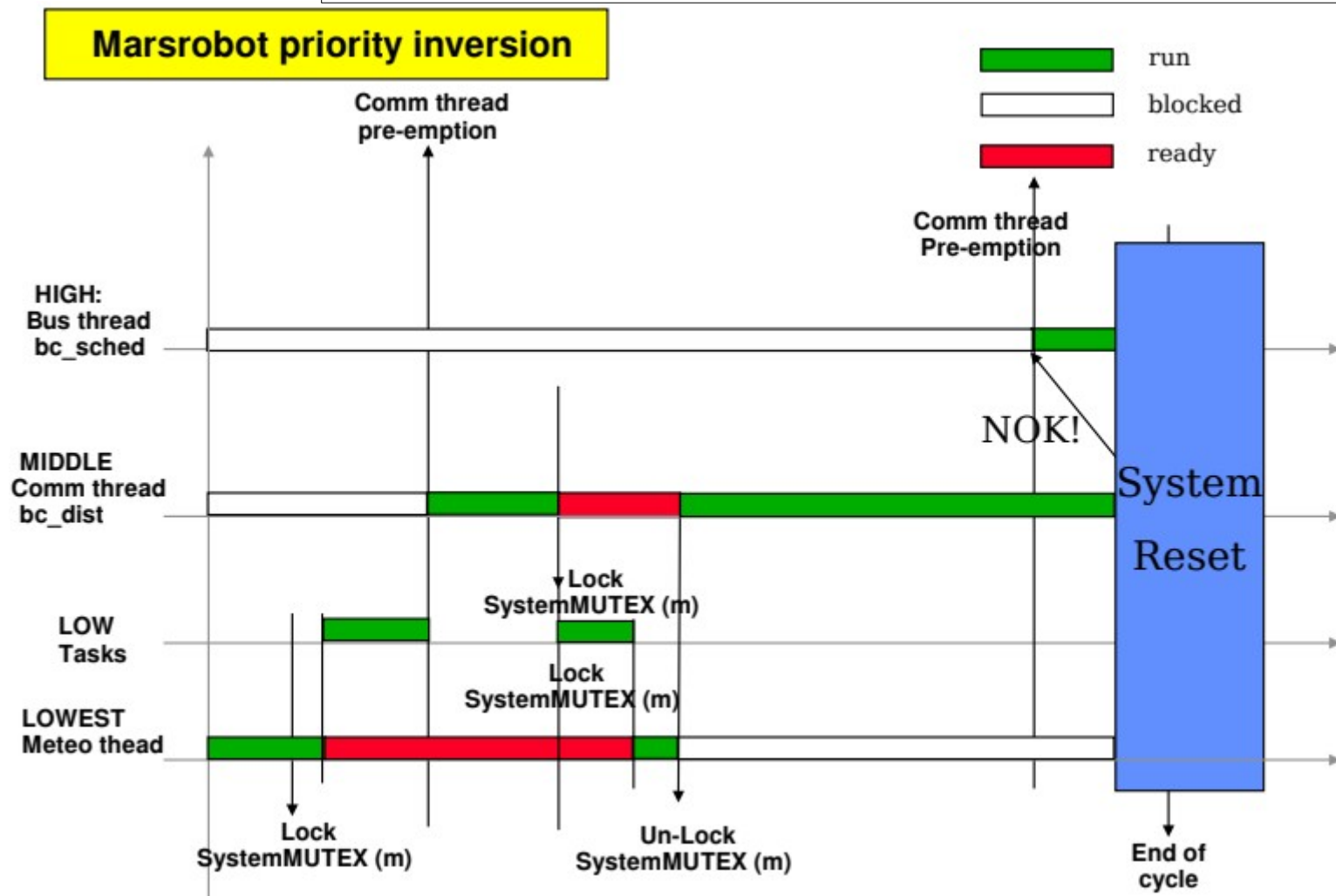
- 这个 ASI/MET 就是用于搜集气象数据的，它有什么特别之处呢？为什么这个理论上优先级最低任务，导致了优先级高的 bc_dist 没有在规定时间内运行完，从而引起系统重置？
- 大部分设备搜集到的数据通过共享内存的方式直接读取 bc_dist 写入双缓冲区。唯独 ASI/MET 使用 IPC。具体来说，它使用了 vxWorks 的 pipe() 机制，用 select() 等待消息（不用完全懂，有点印象就行）



关键词：优先级反转

- 本课程不是讲操作系统的，所以只需要理解：ASI/MET 因为用到了 `select()`，占用了一个互斥信号量 (mutual exclusion semaphore)，而 `bc_dist` 也需要这个东西（所以称它为 shared resource），只能等着
- 可是，由于 ASI/MET 的优先级太低，被一些中优先级的任务抢占了，于是 `bc_dist` 越等越久，直到 `bc_sched` 被唤醒，然后系统重置。







为什么没有事先发现

- The problem would only manifest itself when ASI/MET data was being collected and intermediate tasks were heavily loaded. Our before launch testing was **limited to the "best case"** high data rates and science activities. The fact that data rates from the surface were higher than anticipated and the amount of science activities proportionally greater served to **aggravate the problem**. We did **not expect nor test** the "better than we could have ever imagined" case.



真的没发现？一次都没有？

- We **did see the problem before landing** but could not get it to repeat when we tried to track it down. It was not forgotten nor was it deemed unimportant. Yes, we were concentrating heavily on the entry and landing software. Yes, we considered this problem lower priority. Yes, we would have liked to have everything perfect before landing. However, I don't see any problem here other than we ran out of time to get the lower priority issues completed.



回到 WF 的题目本身

- 根源：拥有 mutex 的时候被抢占，锁不被释放
- 想法：如果它占用的 mutex 被高优先级任务 wait，则继承它的优先级，这样就不会被抢占了
- 题目给了一个避免优先级反转的 Priority Ceiling Protocol，要求模拟一些任务的执行，输出每个任务的完成时刻
- 输入保证：每个任务只能释放最近占用的锁，不会重复锁，执行完毕的时候也不会拥有锁



输入格式

- 任务数 $t(1 \leq t \leq 20)$ ，共享资源数 $r(1 \leq r \leq 20)$ 。每个任务用三个整数 $s(1 \leq s \leq 10000)$ ， $b(1 \leq b \leq t)$ ， $a(1 \leq a \leq 100)$ 和一个长度为 a 的指令序列表示， s 是开始时间， b 是基础优先级，所有任务互不相同
- 指令有三种：
 - $C_n(1 \leq n \leq 100)$ 表示连续执行 n 条计算指令（各花费 1 个单位时间）
 - $L_k/U_k(1 \leq k \leq r)$ 表示 lock/unlock 第 k 个资源



Priority Ceiling Protocol

- 每个任务有一个动态的当前优先级
- 每个资源有一个固定的 priority ceiling，定义为所有“需要使用该资源的任务”的基础优先级的最大值
- 单个 CPU 执行所有任务。初始化时钟为 0，然后不断执行以下步骤



执行步骤

- Step 1. 识别运行态的任务。定义：开始时间小于或等于当前处理器时钟，且没有执行完毕
- Step 2. 计算所有运行态任务的当前优先级，以及其中哪些任务是阻塞态。任务 T 被阻塞的条件是：下一条指令是锁资源 k ，但是 k 要么已经被占用，要么存在另外一个任务拥有一个资源 l ，它的 priority ceiling 大于或等于 T 的当前优先级。我们说任务 T 被所有这样拥有上述资源 k 或 l 的任务阻塞。任务 T 的当前优先级就是 $\max\{T \text{ 的基础优先级, 阻塞 } T \text{ 的所有任务的当前优先级}\}$



执行步骤

- Step 3. 执行非阻塞态的运行态任务中当前优先级最高的那个任务的下一条指令。如果没有这样的任务，或者执行的是计算指令，则时钟加 1。如果执行的是 lock/unlock 指令，则时钟不变
- Priority Ceiling Protocol 拥有三个很棒的性质



性质

- 首先，当前优先级由自身和阻塞定义，而阻塞又有当前优先级定义，看上去是循环定义，但实际上会有一套唯一确定的当前优先级满足条件
- 所有任务最终都会终止
- 步骤 3 不会有 tie



模拟

- 步骤 1 比较简单。步骤 2 怎么做？有两种方法。一种是迭代。不断的循环两个任务，如果其中当前优先级较小的任务 $T1$ 阻塞了另一个任务 $T2$ ，把 $T1$ 的当前优先级修改为 $T2$
- 根据题目描述的性质，假定这样做会收敛到一个不动点吧。收敛时间？不清楚，不过对于做题来说足够了 ...



第二种模拟

- 按照基础优先级从高到底的顺序判断各个任务。如果基础优先级最高（设为 p_{\max} ）的任务没有“立即判定为”被阻塞，则它肯定可以执行，因为没有任务能把当前优先级提升到比它更高，然后题目说了不会有 tie
- 如果它被阻塞了，则把阻塞它的所有任务的当前优先级提升到 p_{\max} 。反复进行这样的检查，直到发现一个可以运行的任务。这种写法比较容易分析，需要 $O(t^2)$ 时间



性质的证明呢？

- 在这里：
- L. Sha, R. Rajkumar, and J. P. Lehoczky, Priority Inheritance Protocols: An Approach to Real-Time Synchronization. In IEEE Transactions on Computers, vol. 39, pp. 1175-1185, Sep. 1990.
- 可以在 <https://www.semanticscholar.org/> 下载



尾声

- WF 题目本身的解法不是重点
- 不过这个背景事件还是很有启发意义的
- 如果能事先进行形式化验证 (formal verification) 而不仅仅是传统测试，事情就会变得不一样
- 敬请期待本讲座下半部分：)



中国计算机学会
China Computer Federation



谢谢大家！

欢迎曹老师带来下半部分的精彩内容