

《数据结构》PA1辅导课

2018年10月24日

6C300

主要内容: PA1辅导

❖CST2018 1-1 A+B problem

❖CST2018 1-2 Graphics

❖CST2018 1-3 filename

❖CST2018 1-4 Stock

❖CST2018 1-5 Zuma

CST2018 1-1 A+B problem

计73 郑林楷

问题描述

- ❖ A+B problem
- ❖ 每个 Online Judge 的第一道题，帮助你熟悉使用 OJ 的方法
- ❖ 输入两个数字，输出它们之和
- ❖ 「抱歉，这题实际是 A*B problem」

问题描述

- ❖ 邓俊辉老师的作业常常过于简单，数据类型只需使用 `int`。
 - ❖ 助教们一致认为，向同学们介绍 Python 中自带的长整型是十分有必要的。
 - ❖ 例如，它可以计算几百位的整数乘法。
 - ❖ 但是，在介绍长整型之前，助教决定让你自己实现一遍长整型乘法。
-
- ❖ $1 \leq n \leq 500$
 - ❖ $0 \leq a, b < 10^{5000}$
 - ❖ Time Limit: 1 sec

问题分析

❖先看题目的提示。

例：

```
12345678910111213 + 11111111111111111111
```

使用两个数组存储：

```
a[] = {3, 1, 2, 1, 1, 1, 0, 1, 9, 8, 7, 6, 5, 4, 3, 2, 1};  
b[] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
```

两个数组分别把数值倒存，逐位相加，每位加后判断是否大于 10，再进行进位即可。

问题分析：算法 1

- ❖ 不思考照着做一遍？模拟竖式计算
- ❖ 设 a 在十进制下有 L_a 位， b 有 L_b 位，则其乘法运算次数为
$$L_a \times L_b$$
- ❖ 复杂度 $O(L_a L_b)$
- ❖ 最坏情况计算次数： $5000 * 5000 * 500 = 12500000000$
- ❖ Boom !

问题分析

- ❖ 题目后面不是有思考题么？
- ❖ No. 1
- ❖ 对于这两个整数应当如何存储？
- ❖ 2 进制？10 进制？（或者其它）
- ❖ 低位在前？高位在前？（或者其它）
- ❖ 从进制入手可能可以减小计算复杂度！

问题分析：算法 2

◆改成 100 进制如何？

◆相当于压 2 位

例：

```
12345678910111213 + 11111111111111111111
```

使用两个数组存储：

```
a[] = {3, 12, 11, 10, 19, 87, 65, 43, 21};  
b[] = {1, 11, 11, 11, 11, 11, 11, 11, 11};
```

两个数组分别把数值倒存，逐位相加，每位加后判断是否大于 100，再进行进位即可。

问题分析：算法 2

- ❖ 来分析下其乘法运算次数：
- ❖ 由于是十进制，所以 a 变成 $L_a/2$ 位， b 变成 $L_b/2$ 位
- ❖ 则其乘法运算次数为：
$$(L_a/2) \times (L_b/2)$$
- ❖ 复杂度 $O(L_a L_b / 4)$
- ❖ 排除掉常数因数，复杂度好像还是没减小啊
- ❖ 最坏情况计算次数： $2500 * 2500 * 500 = 3125000000$
- ❖ 至少它变小了啊！

问题分析：算法 2

问题分析：算法 2

- ❖ 两种模拟竖式计算的写法：
- ❖ 加完直接处理进位 vs 先无视进位加完后处理进位
- ❖ 处理进位次数： $O(L_a L_b)$ vs $O(L_a + L_b)$
- ❖ 压位极限（long long）：9 位 vs 6 位
- ❖ 选哪个？

问题分析

- ❖ 继续看思考题.....
- ❖ No. 2
- ❖ 大整数乘法本身有什么算法（可以自行查阅相应资料）？
- ❖ 什么样的算法能够较好地适应这个问题的数据规模？
- ❖ 此处应有 FFT NTT.....

问题分析：算法 3

❖ 讲点干货：Karatsuba 算法

$$\diamond a = a_0 \times 10^k + a_1$$

$$\diamond b = b_0 \times 10^k + b_1$$

$$\diamond a \times b = (a_0 b_0) \times 10^{2k} + (a_0 b_1 + a_1 b_0) \times 10^k + a_1 b_1$$

❖ 4 次乘法

问题分析：算法 3

$$\diamond a \times b = (a_0 b_0) \times 10^{2k} + (a_0 b_1 + a_1 b_0) \times 10^k + a_1 b_1$$

$$\diamond a_0 b_1 + a_1 b_0 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1$$

\diamond 3 次乘法！

\diamond 递归调用

$$\diamond T(n) = 3 * T(n/2)$$

\diamond 算法复杂度为 $O(3n \log_2 3)$

问题分析：算法 4

- ❖ 助教们一致认为，向同学们介绍 Python 中自带的长整型是十分有必要的。
- ❖ 研究下 Python 如何计算长整型乘法？
- ❖ 内部为 int 数组，进制为 2^{15} ，（无需乘、除和取模）
- ❖ 当然还需要一个高精度转进制
 - `PyObject * PyLong_FromString(const char *str, char **pend, int base)`

小结

- ❖ 直接模拟？会爆炸
- ❖ 加进制！压位！
- ❖ 换算法！
- ❖ 也可以看源码！
- ❖ 其实真的就是一道非常简单的高精度乘法，只是需要压下常数.....

CST2018 1-2 Graphics

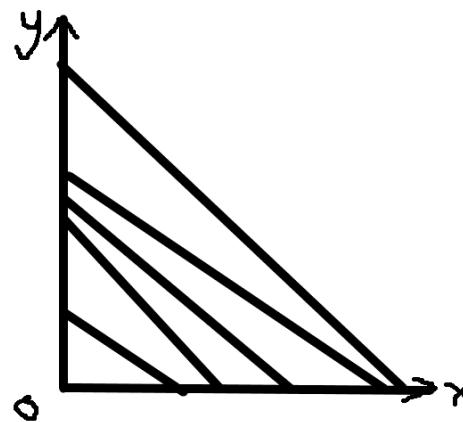
计71 吴一非

问题描述

- ❖ 给定一个平面直角坐标系。
- ❖ 在第一象限中有 n 条线段，这些线段两端分别在x正半轴和y正半轴上。
- ❖ 并且它们两两不会相交且没有公共点。
- ❖ 接下来我们有 m 个询问，每次询问都会给定在第一象限的一个点P。
- ❖ 我们需要对每一个询问回答：
- ❖ 线段OP与之前给定的 n 条线段会产生多少个交点？
- ❖ 其中 n 和 m 都是200000以内的正整数，坐标是int范围内的正整数。

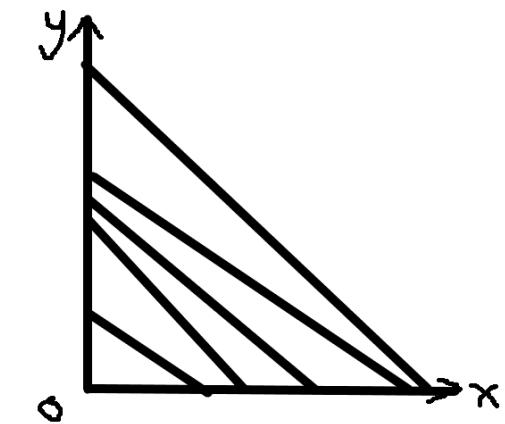
问题分析

- ❖ 首先我们可以很轻松的观察出，
- ❖ 满足任意两条线段不相交的连线方式只能是如下方案：
- ❖ x轴和y轴按照坐标从小到大一一对应相连
- ❖ 也就是长下面这个样子（画的丑大家意会一下就好qwq）



问题分析

- ❖ 而给定一个点P
- ❖ OP与这些直线的交点数量很显然就是点P左下方的直线数量
- ❖ 那么我们问题就变成了求有多少条直线在P点的左下方。
- ❖ 这个问题我们如果一个一个枚举判断，时间复杂度是 $O(n)$ 的。
- ❖ 总时间复杂度就是 $O(nm)$
- ❖ 显然是无法接受的！

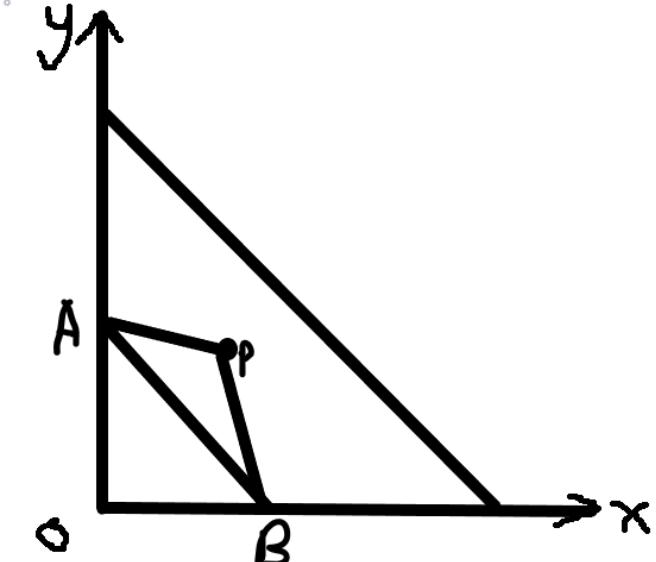


问题分析

- ❖ 那怎么改进呢？
- ❖ 显然如果我们按照x或者y坐标排序后，
- ❖ 满足条件的线段一定是前若干条线段。
- ❖ 那么我们就可以利用二分答案的方法，
- ❖ 将单次查询的时间复杂度从 $O(n)$ 缩小到 $O(\log n)$
- ❖ 这样我们的总时间复杂度就降低到了 $O(m \log n)$
- ❖ 属于可以接受的范围。

计算几何

- ❖ 这题就这么解决了？
- ❖ 还有最后一个问题：
- ❖ 如何判断一条直线在一个点的左下还是右上呢？
- ❖ 假设点A在y轴，点B在x轴。
- ❖ 那么P在AB直线的左下还是右上就取决于向量AP在向量AB的顺时针方向还是逆时针方向。
- ❖ 而判断顺时针还是逆时针方向我们可以使用向量叉积来实现。



计算几何

❖ 假设 $A(x_1, y_1), B(x_2, y_2)$ 。

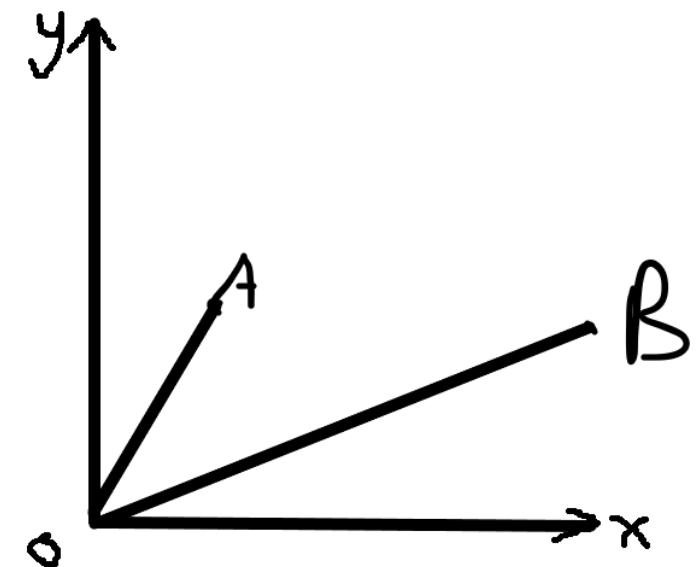
❖ 那么向量 OA 叉乘向量 OB

❖ 就是 $x_1y_2 - x_2y_1$

❖ 结果为正就代表 OB 在 OA 左手边（逆时针方向）

❖ 反之则说明 OB 在 OA 右手边（顺时针方向）

❖ 通过这个方法我们就可以 $O(1)$ 计算出点 P 到底在直线的哪一边啦。



小结

- ❖ 总结一下本题：
- ❖ 首先需要给坐标排序
- ❖ 然后对于每个询问
- ❖ 我们用二分答案来判断点P左下有多少条直线。
- ❖ 使用叉积帮助在二分答案的过程中判断。
- ❖ 本题写程序时有一个需要注意的地方就是叉积判断不要爆int

CST2018 1-3 filename

计73 王雨田

题目大意

- 求两个串的最长公共子序列
- 答案是 $n+m-\text{LCS}(n, m)$

题目分析

- 朴素的LCS过不了，怎么办？

时间复杂度压缩

- 每次只动归距离当前位置小于k的位置
- 时间复杂度为 $O(n*k)$

空间复杂度压缩

- 第一维使用滚动数组
- 第二维只存 $2k+1$ 个位置
- 空间复杂度为 $O(n+k)$

CST2018 1-4 Stock

计72 陈嘉杰

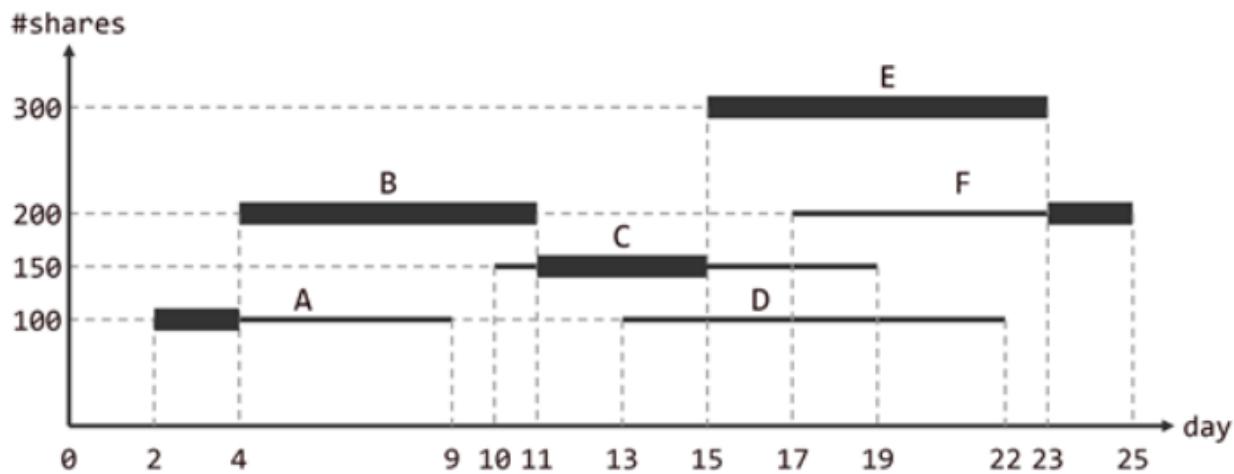
问题描述

描述

“天下大势，分久必合，合久必分”，股市亦是如此：新股不断发行，旧股相继退出。从发行到退出之间的时段，称作股票的活跃期。

为简化起见，这里不妨假定活跃期均以交易日为基本单位，并以股市开市之（第零）日作为统一起点。比如，若股票A“在第123天发行并于第130天退出”，则其活跃期为第123至129天共计7天，记作[123, 130)。另外，不妨假定各股票在其活动期内价格不变——否则，可等效于原股票退出，同时发行一支定价更新的股票。Q王国的股市虽不能脱俗，却亦有其与众不同之处。在该国的股市中，禁止不同股票的活跃期相互包含。比如，无论是[100, 150)和[120, 130)、[120, 130)和[120, 150)、[100, 130)和[120, 130)，甚至是[120, 130)和[120, 130)，都不会同时作为股票的活跃期。也就是说，每天发行和退市的股票累计不超过一支。

M先生是Q王国公认的股神，他的策略既稳妥也简单。所谓稳妥是指，任何时候的投资总额都保持固定，如此可保证风险（risk）有限。所谓简单是指，每天都将所有的投资集中用于购入当日价格最低的一支股票（为此可能需要先卖出原持有股票，交易所需时间忽略不计），如此他的持股量每天都能保持最大——在Q王国，这可是评估股市收益的首要指标。



TL;DR

不过，尽管M先生十分希望精确地知道自己在一段时期内的日均持股量，但对于此类计算他却毫无自信。于是终于有一天，他揣着长长的一叠股市历史记录，敲响了你的家门。

问题描述

- ❖ 好...长...啊...我又不炒股为啥要帮他炒股，贫穷限制了我的想象力
- ❖ 活跃期禁止互相包含？？什么鬼
- ❖ 每天发行和退市的股票累计不超过一支？
- ❖ 超过，就互相包含了。
- ❖ 每天买最便宜的一只股票 - 能买得最多的一只股票
- ❖ 日均持股量 乘以 天数 ？？？？ 这不就是一共可以买的股份吗
- ❖ 提示里说好的考队 (yu) 列 (wen) 呢

问题分析

- ❖ 我们再看看输入格式：D天后发行可以买x的股票/D天后退市
- ❖ 没有指定是哪只股票退市？不能互相包含 => 先发行先退市
- ❖ 先进先出？队列！啊，队列终于出现了。。
- ❖ 那么，怎么算最大持股量？找到每天队列里最大的那个，加起来
- ❖ 问题转化为 队列 + 最大值查询 这才是提示里的意思
- ❖ 关键词：单调队列

难点要点

- ❖ 题目好长，不想看了
- ❖ 题目描述故意绕弯子，考验语文理解水平
- ❖ 找到题目的考点，如这里是考可以查询最大值的队列
- ❖ 善用搜索引擎：Queue AND Maximum
- ❖ 学习单调队列
- ❖ 注意题目输入格式，如何判断输入的这一行是一个数字还是俩

小结

❖ 题意转化

❖ 单调队列

CST2018 1-5 Zuma

计72 谢兴宇

问题描述

- ❖ 维护一个珠子序列，每个珠子有颜色。
- ❖ 支持在位置k插入颜色为c的珠子。
- ❖ 如果插入一个新珠子之后有3个的连续的珠子同色，长度不小于3的极大同色珠子段会消失，并引起连锁反应，直到不存在3个连续的同色珠子为止。
- ❖ 求最终的珠子序列。
- ❖ 最初有n个珠子，有m次插入操作。
- ❖ $n, m \leq 5 \times 10^5$

问题分析

- ❖ 还原回维护线性序列的原子操作：
 - ❖ 1. 在位置 k 插入一个珠子
 - ❖ 2. 查询某一个珠子的前驱/后继
 - ❖ 3. 删 除(一段)珠子
-
- ❖ 典型的动态序列维护问题

基础数据结构

基础数据结构	数组	链表
查询前驱/后继	$O(1)$	$O(1)$
在指定位置插入	$O(1)+O(n)$	$O(n)+O(1)$
删除一段珠子	$O(n)$	$O(1)$

- ❖ 分块系列: unrolled linked list, B-tree...
- ❖ 链表系列: unrolled linked list, skip list...
- ❖ 平衡树系列: splay, treap, AVL, Red-black, Scapegoat...
- ❖ Anything goes!
- ❖ 大部分都可在邓老师的讲义里找到: AVL(P711-731),
Splay(P732-751), B-tree(P752-792), Red-black(P793-831),
Skip List(P1021-1039).

数据结构选择层面的常数优化

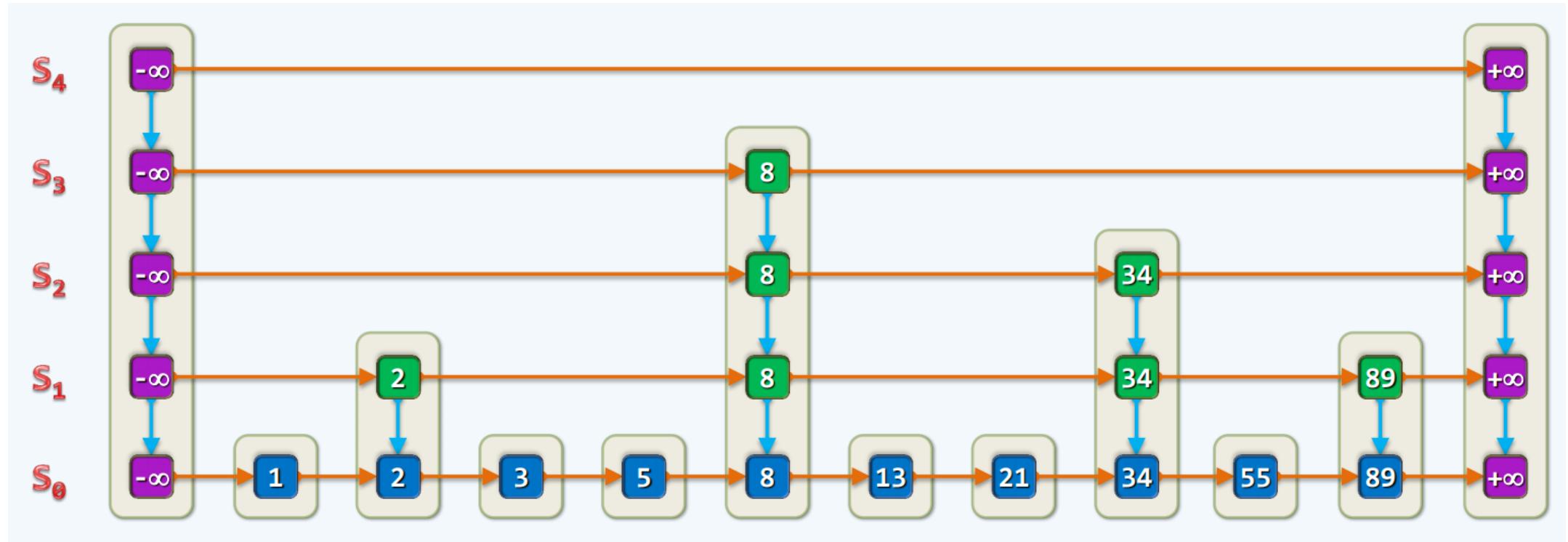
- ❖ 哪一个数据结构更加适合本题?
- ❖ 对3种操作作更加细致的分析:
- ❖ 1. 在位置 k 插入一个珠子: 不超过 m 次
- ❖ 2. 查询某一个珠子前/后面的珠子: 不超过 $n+5m$ 次
- ❖ 3. 删除(一段)珠子: 删除的珠子数量不超过 $n+m$ 个
- ❖ 需要较快地支持查询前驱/后继.

Splay

- ❖ 经典平衡树，可以在均摊 $O(\log n)$ 的时间内插入，删除，查询。
- ❖ 注意到对于前驱后继的查询是连续的。
- ❖ Dynamic Finger Theorem: 相邻两次查询的rank分别为 x 和 y ，
 $O(m + n + \sum_{x,y} \log(|y - x| + 1))$.

Skip List

- ◆ 链表变体，可以在期望 $O(\log n)$ 的时间内做到插入，删除，查询
- ◆ 特别地，可以严格 $O(1)$ 地查询前驱后继



块状链表

- ❖ 出现在神秘提示中的算法：链表和数组的结合。
- ❖ 在严格 $O(\sqrt{n})$ 的时间内支持插入，删除。
- ❖ 特别地，可以 $O(1)$ 查询前驱后继。
- ❖ 相比于跳表，对缓存更加友好。
- ❖ 将块链嵌套，即得到一棵 B-tree。

块链：深入讨论

❖ $O(n \sqrt{n}) = 10^9$

❖ 这不是随便卡？

❖ 本机对一组随机极限数据测试，结果如下

❖ Skip list: 0.578s

❖ Unrolled linked list: 0.172s

❖ 块的大小 = 10000

❖ memmove总字节数: 3.7×10^9

❖ 速度大概是手动移动内存的1/100.

小结

- ❖ 经典的维护动态序列的问题
- ❖ 学习/发明一种你喜爱的数据结构
- ❖ ~~memmove~~ 太快啦