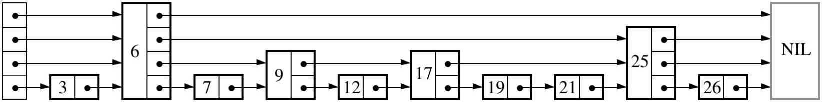
Skip List

Kvrmnks
tayyx2000@163.com

2020.11.11

Skip List 是什么



是链表，高度不一样，指针多了不少，是有序的，指针有的是跳着指的。

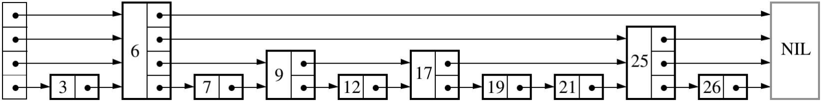
Skip List 的功能

1. 查找 期望 $O(\log n)$，最坏 $O(n)$
2. 插入 期望 $O(\log n)$，最坏 $O(n)$
3. 删除 期望 $O(\log n)$，最坏 $O(n)$
4. 前驱 期望 $O(\log n)$，最坏 $O(n)$
5. 后继 期望 $O(\log n)$，最坏 $O(n)$
6. 第 k 大 期望 $O(\log n)$，最坏 $O(n)$
7. *rank* 期望 $O(\log n)$，最坏 $O(n)$
8. 空间复杂度 期望 $O(n)$，最坏 $O(n \log n)$

Skip List 有哪些优势

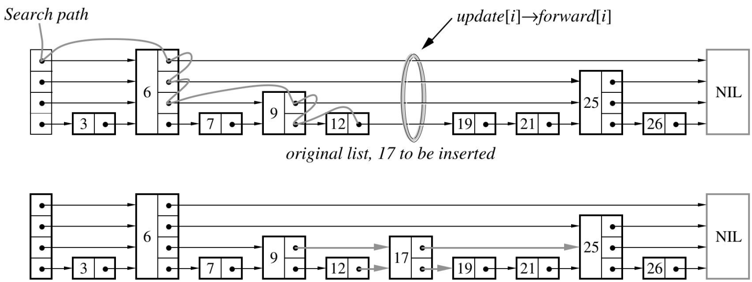
1. 实现简单
2. 期望下有和一般的平衡树一样的复杂度
3. 更好地支持并行
4. finger search

Skip List 是怎么实现的



每个元素都有一个高度
每个高度都有一个指针
每个指针向右指向第一阻挡到它的地方

插入

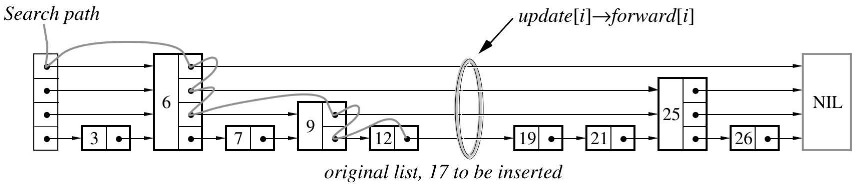


从起点的最高点出发，能往右走就往右走，不行就降低高度。
维护查找过程中每层最后一个，为了维护每个指针。
找到要插入元素的位置之后，直接放进去，更新指针。
不需要别的操作保持平衡！

每个元素的高度如何确定

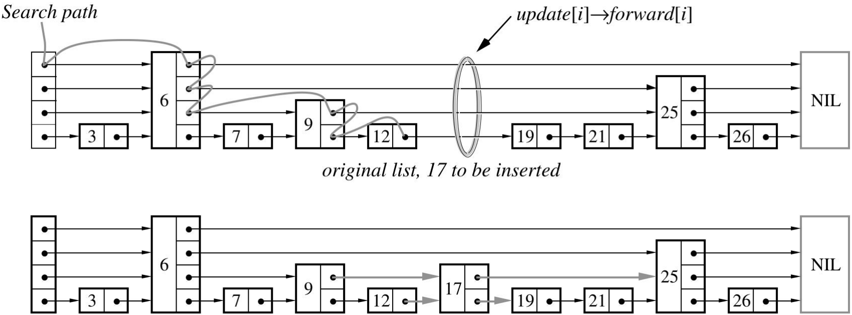
一个元素的高度是随机确定的，具体随机方式如下
有一个常数 p, $0 \leq p \leq 1$, 这个元素高度为 1 的概率是 p, 为 2 的概率是 $p^2 \cdots$ 为 m 的概率为 p^m
其中特别规定最大高度为 $\log n$

查找



从起点的最高点出发，能往右走就往右走，不行就降低高度。

删除



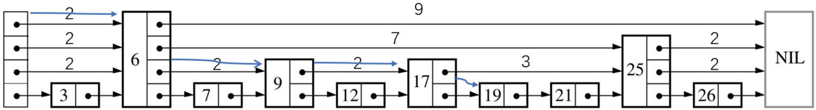
依旧要维护查找时每层最后一个元素，来维护指针。
不需要别的操作保持平衡！

求第 k 大

此处应有一张 rank tree 的图。

求第 k 大

给每条边加权



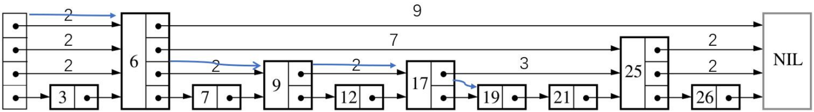
边权代表走了这条边之后排名增加了多少

求 rank

我可以二分呀 (X)
此处又应有一张 rank tree 的图。

求 rank

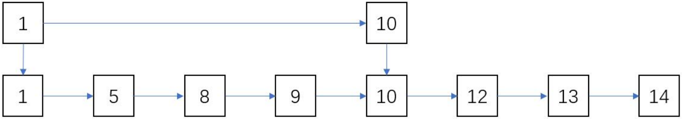
同上面的方法一样维护边权，查找过程中累加每条边的长度



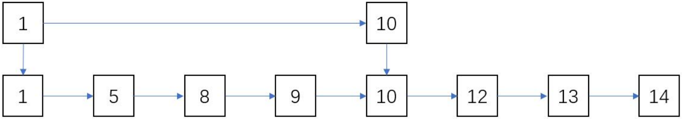
求前驱和后继

这…

怎样让链表快一点

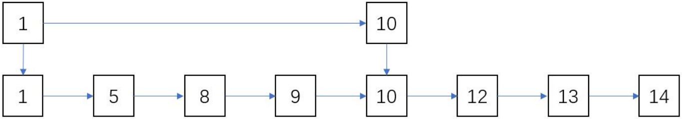


怎样让链表快一点



分块!

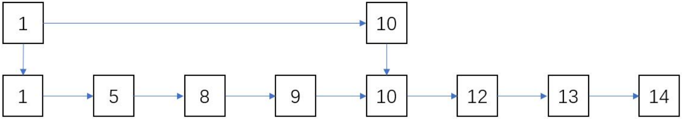
怎样让链表快一点



分块!

1. 怎么分最好？

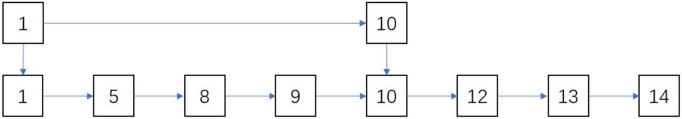
怎样让链表快一点



分块!

1. 怎么分最好？
2. 支持删除插入吗？

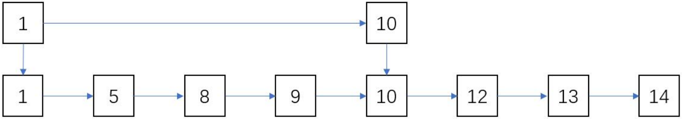
怎样让链表快一点



分块!

1. 怎么分最好？$\sqrt{n}$ 分块
2. 支持删除插入吗？

怎样让链表快一点



分块!

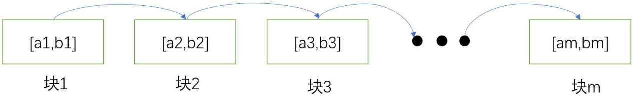
1. 怎么分最好？$\sqrt{n}$ 分块
2. 支持删除插入吗？同样是 $O(\sqrt{n})$ 的复杂度

块状链表

假设一共有 n 个数，假设将链表分成 $\frac{n}{a}$ 块，块的大小是 a
于是每次查找的复杂度是 $O(\frac{n}{a} + a)$，由均值不等式 $a = \sqrt{n}$ 时，
复杂度达到最小，这时复杂度是 $O(\sqrt{n})$

块状链表

考虑怎样维护插入和删除。



先规定每块的大小限制再考虑块数
如果插入的块比 $\sqrt{n}$，把块分裂成两个
删除直接在所在块中删除
将新相接的块尝试合并

块状链表

假设第 i 块的大小为 s_i，根据上面的规则有

$$s_i + s_{i-1} \geq \sqrt{n}$$

累加这个不等式，可以得到

$$s_1 + 2 * \sum_{i=2}^{m-1} s_i + s_m \geq m\sqrt{n}$$

放缩一下得到

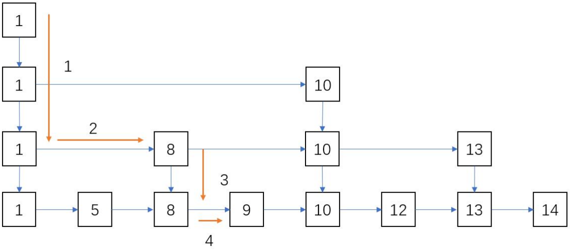
$$2n \geq m\sqrt{n}$$

$$m \leq 2\sqrt{n}$$

于是总复杂度为

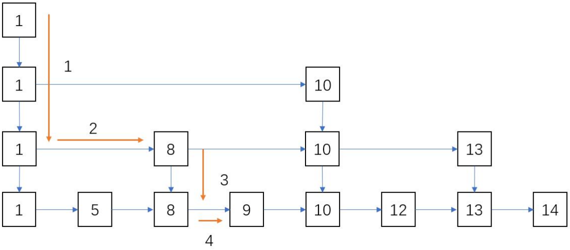
$$O(\sqrt{n})$$

再快一点?



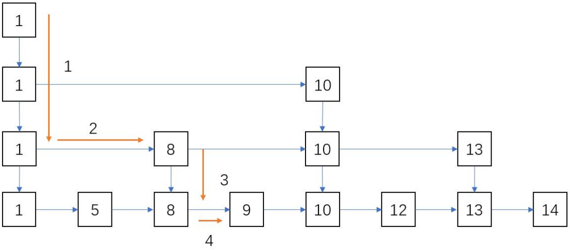
1. 查询是 $O(\log n)$ 了!

再快一点?



1. 查询是 $O(\log n)$ 了！　可以证明每层横着走最多一次

再快一点?



1. 查询是 $O(\log n)$ 了！　可以证明每层横着走最多一次
2. 删除和加入都很困难

各种复杂度

1. 什么是期望复杂度
2. 什么是最好复杂度
3. 什么是最坏复杂度
4. 什么是均摊复杂度

最好和最坏复杂度

1. 什么是最好复杂度
2. 什么是最坏复杂度

冒泡排序

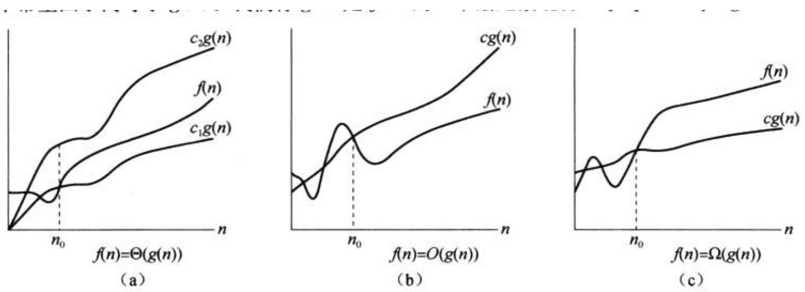
期望复杂度和均摊复杂度

1. 什么是期望复杂度
2. 什么是均摊复杂度

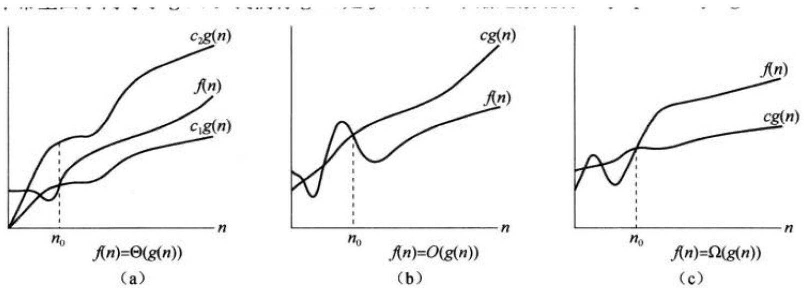
期望复杂度和均摊复杂度

为什么要关心最坏复杂度和最好复杂度？
参数化算法　缝合怪

Ω O Θ 的区别...


$c_2g(n)$
$f(n)$
$c_1g(n)$
n_0
n
$f(n)=\Theta(g(n))$
(a)
$cg(n)$
$f(n)$
n_0
n
$f(n)=O(g(n))$
(b)
$f(n)$
$cg(n)$
n_0
n
$f(n)=\Omega(g(n))$
(c)

Ω O Θ 的区别...

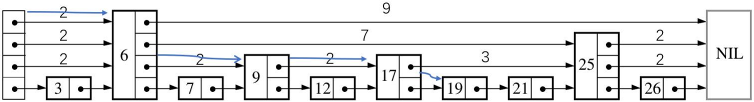


其实这个跟最好最坏情况没什么关系...

时间复杂度

可以证明查找的复杂度在期望情况下是 $\Theta(\log n)$
但这并没有什么卵用，大家其实并不在乎 Θ 和 O

区间求和



求 rank 实际上就是一种弱的区间求和，只需要再额外维护一个边权表示走了这条边，走过的元素的权值和增加了多少

它是个链表

区间移动，文本编辑器。

可以用来实现 ETT

1. 欧拉序
2. 动态树
3. link-cut
4. 维护子树信息
5. 维护到根信息

确定性 skip list

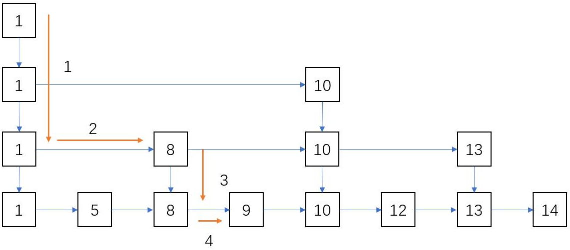
似乎能和 2-3 树对应...?

空间复杂度

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \sum_{j=1}^{\infty} jp^j = \sum_{i=1}^{n} \frac{p}{(p-1)^2} = \frac{np}{(p-1)^2}$$

取 $p = \frac{1}{2}$，$E[X] = 2n$，期望空间复杂度 $O(n)$
显然最坏情况下空间复杂度为 $O(n \log n)$

先感性理解一下



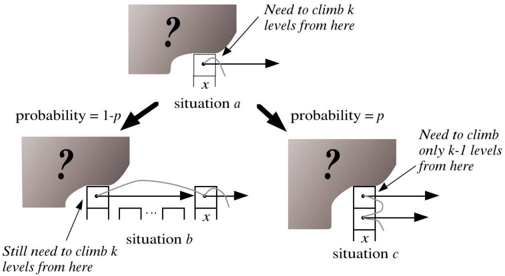
查找的复杂度

考虑反着思考复杂度

1. 跳到最高处的步数
2. 跳到起点的步数

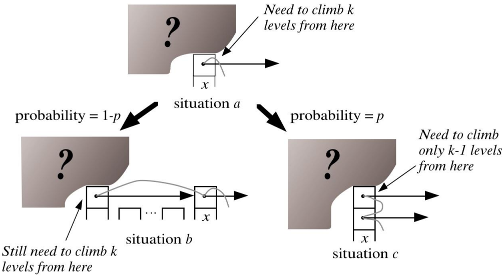
跳到最高处的步数

考虑一个无限长的 Skip List



跳到最高处的步数

考虑一个无限长的 Skip List



$$E[Y] = pE[Y] + (1-p)E[Y-1] + 1$$

跳到最高处的步数

$$E[Y] = (1-p)E[Y] + pE[Y-1] + 1$$

跳到最高处的步数

$$E[Y] = (1-p)E[Y] + pE[Y-1] + 1$$

化简得

$$E[Y] - E[Y-1] = \frac{1}{p}$$

跳到最高处的步数

$$E[Y] = (1-p)E[Y] + pE[Y-1] + 1$$

化简得

$$E[Y] - E[Y-1] = \frac{1}{p}$$

$$E[0] = 0$$

跳到最高处的步数

$$E[Y] = (1-p)E[Y] + pE[Y-1] + 1$$

化简得

$$E[Y] - E[Y-1] = \frac{1}{p}$$

$$E[0] = 0$$

于是 $E[Y] = \frac{Y}{p}$

跳到最高处的步数

$$E[Y] = (1-p)E[Y] + pE[Y-1] + 1$$

化简得

$$E[Y] - E[Y-1] = \frac{1}{p}$$

$$E[0] = 0$$

于是 $E[Y] = \frac{Y}{p}$ 当 $Y = [\log n] - 1, p = \frac{1}{2}$， 有 $E[Y] = 2[\log n] - 2$

跳到起点的步数

分析最高层有多少个数。

$$E[Z]=\sum_{i=1}^{n}\sum_{j=[\log n]}^{\infty}jp^j=\frac{np^{[\log n]}(p+[\log n]-p[\log n])}{(p-1)^2}$$

跳到起点的步数

分析最高层有多少个数。

$$E[Z]=\sum_{i=1}^{n}\sum_{j=[\log n]}^{\infty} jp^j=\frac{np^{[\log n]}(p+[\log n]-p[\log n])}{(p-1)^2}$$

取 $p=\frac{1}{2}$

$$E[Z]\leq 2[\log n]+2$$

时间复杂度

1. 跳到最高处的步数 $E[Y] = 2[\log n] - 2$
2. 跳到起点的步数 $E[Z] \leq 2[\log n] + 2$

期望复杂度为 $O(\log n)$

时间复杂度

1. 查找　期望 $O(\log n)$，最坏 $O(n)$
2. 插入　期望 $O(\log n)$，最坏 $O(n)$
3. 删除　期望 $O(\log n)$，最坏 $O(n)$
4. 前驱　期望 $O(\log n)$，最坏 $O(n)$
5. 后继　期望 $O(\log n)$，最坏 $O(n)$
6. 第 k 大　期望 $O(\log n)$，最坏 $O(n)$
7. *rank*　期望 $O(\log n)$，最坏 $O(n)$

1. 插入　查找时维护每个高度最后的一个结点
2. 删除　查找时维护每个高度最后的一个结点
3. 前驱　本质就是查找
4. 后继　查找之后跳到最底层再往前跳
5. 第 k 大　本质就是搜索
6. rank　本质就是搜索

感谢倾听！