页面置换算法调研报告

ljt12138

2019 年 4 月 21 日

目录

1 概述

在页式虚拟存储计算机中，内存被看作是磁盘的缓存，操作系统通过

将常用的虚拟内存页放入物理内存页帧中，将不常用虚拟内存页放入磁盘中来提供更大、更自动化的虚拟存储。页面置换算法在上世纪曾是很热门的研究方向，Belady的 [1]中根据使用以往信息的多少对算法大致归类，并给出了离线下的最优算法；理论上，对于页面置换问题竞争性分析的研究有许多成果，Fiat等人的 [2]对于更一般的k-server问题证明了随机标记法的竞争比，并分析了随机算法竞争比的下界。todo

确定性页面置换算法的竞争比不会小于k，通过在算法中引入随机因素，可以期望将竞争比降低至$\log k$。可以证明，随机页面置换算法期望竞争比的下界是$\log k$。

2 竞争性分析

2.1 离线算法和竞争比

页面置换问题可以被这样描述：给定一台支持h个逻辑页面、拥有k个页面大小物理内存的机器，选择每次的置换页面使得总的置换次数最少。

页面置换算法可以分为离线算法（off-line）和在线算法（on-line）。离线算法是指整个访存序列已经预先给定，而在线算法是指访存序列是动态给出的。很显然，真实情景下的页面置换算法必须是在线的。

Belady在其 [1]中最早给出了称为`MIN`算法的最优离线算法。另一种简单的算法`FIF`（Further-in-future），通过总是换出下一次访问最晚的页面，算法可以做到最小化缺页次数 [9]。

虽然最优算法需要使用未来的信息，在实践中不可能实现，但其可以作为分析在线算法的一个标准。一个很显然的事实是，直接使用缺页数评价在线算法应对所有输入的能力是愚蠢的——如果连最优算法都无法给出令我们满意的命中率，那么在线算法也不可能做到更优。因此，我们选择使用在线算法缺页次数和最优算法缺页次数的比来评价在线算法的优劣。设一个算法A在访存序列S上的缺页次数为$A(S)$，如果对于任意访存序列S，都有：

$$A(S) \leq c \times OPT(S) + O(1)$$

我们可以称，算法A是c-Competitive的，c可以称为算法A的竞争比。

为了方便，我们有时不会考虑从内存为空到内存填满这一段的代价，因为无论任何算法在这里付出的代价都是相同的。

竞争比给在线算法的优劣提供了基本的评价标准。竞争比低的算法不一定总是提供更好的性能，但从某种程度上讲，其在最差情况下的性能是值得信赖的；反过来，竞争比高的算法虽然在某些情况下运行良好，但需要警惕可能出现的性能退化。当确实需要使用高近似比的算法时，最好提供一个“备选方案”。

2.2 确定性在线算法竞争比的下界

定理 2.1 *任何确定性在线算法C的竞争比至少是k。*

证明 2.1 *考虑仅有$k+1$个逻辑页面的长度为l的访存序列，每次访问的页面恰好是C上一次换出的页面，那么每一次访问都会发生缺页；在最优算法中，每次缺页发生之后，下一次发生缺页至少是在k次之后（因为换出的是下一次访问最晚的页面），因而总的缺页次数不会超过l/k。有：*

$$C(S) \geq k \times OPT(S) + O(1)$$

我们用一个例子来解释竞争比的价值。LRU算法通常被认为是性能最好的页面置换算法之一，但一个简单的情景就会导致其严重退化。设逻辑页面数为$k+1$，物理页面数为k。

$$\begin{aligned} &1, 2, 3, \ldots, k, k+1 \\ &1, 2, 3, \ldots, k, k+1 \\ &\ldots \end{aligned} \tag{1}$$

对于LRU算法而言，由于总是换出最近访问最早的页面，每次缺页都会恰好把下一次要访问的换出，因而每一次都会发生缺页。然而最优情况下每k次访存只会发生一次缺页。因此，许多系统中会使用其他算法作为LRU退化时的备选方案。

2.3 标记算法

标记算法（Marking Algorithm）是一类置换算法的总称。假设每个在内存中的页面都有一个标记位，初始时所有页面都是未标记的。未标记的页面会在访问时被标记，当所有页面都被标记时，将所有页面重新设为已标记。标记算法是那些只会将未标记的页面置换出去的算法。

许多根据访问排序的页面置换算法都是标记算法，例如最近未使用算法（NRU）与最近最久未使用算法（LRU）。

引理 2.1 *LRU*是标记算法。

证明 2.2 考虑使用反证法。如果*LRU*不是标记算法，那么在某个时刻其换出了一个未标记的页p，并且存在一个标记的页q没有被换出。因此上一次执行将所有页取消标记后，q被访问了至少一次，p没有被访问过，因而q上一次访问比p要近。根据*LRU*算法的定义，p不会被置换出，矛盾。

定理 2.2 任何标记算法都是k-*Competitive*的。

证明 2.3 将序列按顺序划分为t个段$b_1, b_2, \ldots, b_t$，每个段b_i中仅包含至多k种不同的页面，且b_1尽可能长，在此前提下b_2尽可能长，以此类推。即：对于一个以s_i开头的段b_i，b_{i+1}的开头s_j是最小的j，满足$\{s_i, s_i+1, \ldots, s_j\}$包含$k+1$个不同的位置。我们称这是访存序列的分割。

用归纳法可以证明，在每一段末尾，标记算法总会使得所有段被标记。那么在下一段开头所有段都被清除标记，且这些未标记的段要么被重新访问，要么被换出，这一段内的换出次数不会超过k。那么总的换出次数不超过$t \times k$。

接下来我们说明最优算法至少会发生t次缺页。考虑每一个段开头的页面$p_1, p_2, \ldots, p_t$。如果p_i没有发生缺页，说明p_i在内存中。根据划分段的定义，上一次p_i被引用一定不在b_{i-1}中，因而b_{i-1}可以使用的内存页面只有$k-1$个。那么除掉p_{i-1}发生的缺页，根据鸽巢原理，还会至少发生一次缺页。所以总的缺页次数至少是t。由于：

$$Marking(S) \leq t \times k \leq k \times OPT(S)$$

任何标记算法都是k-*Competitive*的。

一些确定性算法的竞争比结论如下 [11]。

- LRU算法是k-Competitive的；这是上面结论的平凡推论。
- FIFO，Clock算法是k-Competitive的；他们都是所谓Conservative algorithms [12]。
- LIFO，LFU算法不是Competitive的，即算法缺页次数和最优解的缺页次数之比可以无限大 [13]。

2.4 随机性的引入

在线算法应对输入的过程可以看成一个双人零和博弈。算法是“Defender”，为了让代价最小化；输入是“Hacker”，为了让代价最大化。由于面对的是“充满恶意的”输入序列，确定性算法的竞争比并不是很理想。一种自然的想法就是在决策中引入随机性。

使用类似之前的方法，我们定义随机算法R是fc-Competitive的，如果：

$$\mathbb{E}[R(S)] \leq c \times OPT(S) + O(1)$$

2.5 随机置换算法

随机置换算法（下面称为RAND）是最简单的随机置换策略：在需要换出页面时，始终从驻留在内存中的页面里等概率地选择一个进行换出。IBM的OS/390操作系统中曾使用随机置换算法作为LRU的补充，当LRU性能退化时使用RAND代替 [12]。

2.6 随机标记算法

一种有效的随机置换算法被称为随机标记算法（Random Marking），下文中我们简称其为RM。其思路类似于一般的标记算法，但每次需要换出时会从所有未被标记的页面中随机选出一个换出。Fiat首先证明了RM是$2H_k$-Competitive的 [2]，其中H_k是调和级数前k项的和，$H_k = \log k + O(1)$。

定理 2.3 *随机标记算法是$2H_k$-Competitive的。*

证明 2.4 *考虑访存序列的分割$b_1, b_2, \ldots, b_t$，考虑一个段b_i中的元素。我们称一个页面是**新的**，如果它在b_{i-1}中没有被引用，在b_i中还没有被引用；称一个页面是**旧的**，如果它在b_{i-1}中被引用过，在b_i中还没有被引用。不妨设b_i中对新页面的引用有l_i次，对旧页面的引用有$k - l_i$次。*

我们首先证明，最优算法在b_i执行的开销至少是$\sum l_i/2$。假设第b_i中对新页面的引用发生了p_i次缺页，那么在b_{i-1}中另外的$l_i - p_i$个页面都始终在内存中没有被换出，因而在b_{i-1}中可用的页面只有$k - (l_i - p_i)$个，至少发生了$l_i - p_i$次缺页。考虑将这两者相加得到：

$$\sum_i p_i + (l_i - p_i) = \sum_i l_i$$

一次缺页至多会被当前段计算一次，被下一段计算一次，因此总的缺页次数不会小于上面结果的一半，即$\sum l_i/2$。

接下来我们说明RM算法在段b_i的缺页次数不会超过$l_i H_k$，从而总的缺页次数不会超过$H_k \sum l_i$。其中H_k是调和级数前k项的和。对于任何标记算法，在一个段开始时，内存中总是存放上一个段的所有内容。因此，每一个新页面的引用都会导致缺页，我们只需要关心旧页面产生的缺页次数，即由于被b_i中导致的缺页换出的旧页面。很明显，最差情况下所有新页面的引用都在旧页面的引用之前。

如果一个页面P将页面Q换出，我们连接一条从P到Q的有向边，那么只有引用时入度为1的点会发生缺页。设旧页面依次是$O_0, O_1, \ldots, O_{k-l_i+1}$。当处理$O_j$时，发生缺页且可能换出$O_j$的，就是那些入度为1，出度为0的点，而这样的点恰好有l_i个。这l_i个点会在还未被换出的$k-j$个旧页面中等概率取出恰好l_i个，那么O_j被取出的概率就是$l_j/(k-j)$。所有旧页面的期望缺页次数就是：

$$\sum_{0 \le j < k - l_i} \frac{l_i}{k-j} = l_i \sum_{l_i < j \le k} \frac{1}{j} = l_i (H_k - H_j)$$

再加上新页面发生的缺页，RM总的开销不会超过：

$$RM(S) \le 2H_k \times OPT(S)$$

注意到上面的证明中给出了最优算法缺页次数一个很好的界，我们这里单独列出这个结论。

定理 2.4 对于任何访存序列S，设其分割是$b_1, b_2, \ldots, b_t$，其中b_i的新页面访问个数是l_i，则：

$$\frac{1}{2} \sum_i l_i \le OPT(S) \le \sum_i l_i$$

2.7 极大极小原理

证明一个问题随机算法竞争比下界的重要工具是极大极小原理（Minimax Principle）。我们可以将随机算法看成在一个确定性算法集合$\mathcal{A}$中以某种分布选出一个算法，设随机变量A为分布p下选出的算法；输入集合是$\mathcal{X}$，随机变量X为分布q下选出的输入，$c(a,x)$为算法a在输入x下的开销，那么：

$$\max_{x \in \mathcal{X}} \mathbb{E}[c(A,x)] \ge \min_{a \in \mathcal{A}} \mathbb{E}[c(a,X)]$$

证明 2.5

$$
\begin{aligned}
&\max_{x\in\mathcal{X}} \mathbb{E}[c(A,x)] \\
=&\sum_x q(x) \max_{x\in\mathcal{X}} \mathbb{E}[c(A,x)] \\
\geq&\sum_x q(x)\mathbb{E}[c(A,x)] \\
=&\sum_x q(x) \sum_a p(a)c(a,x) \\
=&\sum_x q(x)\mathbb{E}[c(A,x)] \\
\geq&\sum_x q(x) \min_{a\in\mathcal{A}} c(a,x) \\
=&\min_{a\in\mathcal{A}} \mathbb{E}[c(a,X)]
\end{aligned}
\tag{2}
$$

定理指出，随机算法A在最差输入下的开销，至少是某种分布的随机输入下最好程序的开销。那么我们要证明随机算法开销的下界，便可以构造某种概率分布，并证明最优的确定性算法的开销不小于随机算法的开销。利用这个极大极小原理可以证明下面这个定理 [5]。

2.8 随机算法竞争比的下界

定理 2.5 页面置换问题不存在c-*Competitive*的随机化算法，对于任何$c < \lceil\log(k+1)\rceil/2$。

证明 2.6 考虑构造一个逻辑页面个数为$k+1$访存序列。首先依次访问所有的位置。接下来我们按段生成一个无限长的序列，每一段都以所有页面被访问结束。不妨将$1,2,\ldots,k+1$摆放成一个环。在每一段中，我们总是不断地从当前位置出发，等概率地选择两条路径中的一条，到达未访问过位置的中间点。

在每一次游走中，对于任何一个确定性算法，都会以$1/2$的概率产生一次缺页，因此每一段中期望产生缺页的次数为$\lceil\log(k+1)\rceil/2$；由于每一段中最后访问的页面只会被访问一次，最优算法会在上一次缺页，即上一段的末尾将这个页面换出，从而在每一段只会产生一次缺页。那么任何确定算法的近似比都至少是$\lceil\log(k+1)\rceil/2$，根据极大极小原理，页面置换问题不存在小于这个近似比的随机算法。

因此我们可以认为，在竞争比的意义下，RM几乎已经是最优的算法了。使用RM可以保证在被某些访存序列的“恶意攻击”时，页面换入换出的次数不会超过最优算法的$2\log k$倍；与之相比，任何确定性算法至多只能做到不超过k倍。

2.9 LRU退化的情景

由于希望使用RM或RAND作为LRU退化时的候选，我们关注LRU退化的情景下各个算法的性能。实验的算法包括LRU、FIFO、Clock、RM和RAND。

Algorithm	n=129	n=130	n=150
LRU	126.046	64	6.76754
FIFO	126.046	64	6.76754
Clock	1.96935	64	6.76754
RM	5.35882	4.93054	3.18855
RAND	1.98533	1.96655	1.89022

表 1: 不同算法逻辑页面数与竞争比的关系

在第一个实验中，我们假定进程拥有的物理页面数为128，程序需要使用的页面数为129，且循环的访问所有页面。可以发现FIFO和LRU的竞争比都达到了k，而RAND、RM和Clock保持了比较良好的性能。

在第二个实验中，逻辑页面数变为了130。注意到Clock并没有像上面一样保持良好的性能，发生了严重的退化。但RM和RAND仍然保持着不错的性能。

在第三个实验中，逻辑页面数变为了150。和预期相符，RM和RAND仍然具有良好的性能。随着访问页面数量的增长，LRU的竞争比逐渐下降到了正常水平。

以LRU在$n = 129$时的性能为例分析退化情景下付出的代价。设单次访问内存的平均开销是M，处理一次缺页的平均开销为O，缺页率为P，平均访存开销C可以由下面的公式给出：

$$C = M + O \times P = M\left(1 + \frac{O}{M} \times P\right)$$

对于LRU而言，由于每次都会发生缺页，平均访存开销为：

$$C_{LRU} = M\left(1 + \frac{O}{M}\right) = M + O$$

而对于RM而言，缺页率只有LRU的约5/126，平均访存开销为：

$$C_{RM} = M\left(1 + \frac{O}{M} \times \frac{5}{126}\right) = M + \frac{5}{126} \times O$$

由于$M << O$，RM的平均访存性能比起LRU提升了25倍。通过实验我们可以看出，低竞争比的随机策略可以作为确定性算法的有效补充，防止可能出现的严重退化情景。

2.10 竞争性分析的局限性

必须指出，竞争性分析与实践经验之间的距离是巨大的。尽管LRU的竞争比达到了k，但在实际应用中，其竞争比通常在1到2之间；在实践中，LRU几乎总是比FIFO好，然而竞争性分析并不能给出他们之间的任何区别。[6]中指出了更多这样的例子。我们将在下一章详细描述一些为了弥补理论和实践之前差距而提出的一些模型。

3 页面置换与局部性

3.1 概述

在基于竞争性分析的页面置换算法研究中，访存序列被视为是“攻击者”，作为“防御者”的算法不得不考虑任何可能的访存序列。然而真实应用的访存序列既不会总是“针对”某种策略，也不是随机的。与之相反，程序访问内存具有所谓局部性（Locality）的特征。

内存访问的局部性主要可以分为两种。一是时间局部性，即一个被访问的地址很可能在近期再次被访问；二是空间局部性，即一个被访问的地址附近的地址很可能在近期被访问。将页面分页管理已经天然的利用了程序的空间局部性，那么能否良好利用程序的时间局部性便决定了页面置换算法的优劣。

对程序行为的刻画有着许多不同的模型，这里我们主要关注其中的两个。

- 基于访问图（Access Graph）的模型 [5]。模型认为，程序一段连续的访存行为应当是访问图上一个连续的路径。在很多情况下，程序的访存依赖于程序中的数据结构，访存行为有时是和数据结构保持一致

的。当数据结构是树甚至一般图时，空间局部性中的“相邻”变为了数据结构上的相邻。

- 基于工作集（Working Set）的模型 [4]。模型认为，程序在一段时间内总是会频繁的访问一些内存页面，这些内存页面被称为当前的工作集。工作集模型允许我们通过调整常驻集的大小来适应工作集，通过和进程调度子系统的整合，可以给并发环境下的程序提供更好的内存性能。

3.2 访存图模型

访存图是刻画访存局部性的模型之一。 [5]首先使用这个模型描述程序访存的局部性，下面的工作主要来源于这篇论文。其中的主要工作包括：分析了无向访存图上的页面置换问题，给出了在线算法竞争比更紧的下界，分析了访存图上LRU算法的竞争比并证明了两个有趣的结论：

1. LRU算法几乎总是比FIFO要好的。更严谨的，在任意访存图G上，LRU算法的竞争比总是小于两倍的FIFO算法的竞争比。即：

$$c_{LRU,k}(G) \leq 2c_{FIFO,k}(G)$$

   [7]则进一步证明了$c_{LRU,k} \leq c_{FIFO,k}(G)$。

2. 在访存图G是一棵树的情景下，无论树的形态如何，LRU总能达到确定性算法竞争比的下界。

访存图模型的优势和局限性都是明显的。由于程序几乎总是按照其数据结构的结构访问内存，使用图作为访问内存局部性的描述是合理的。更加一般的，程序的控制流图一定是程序的访存图。 [8]将这里的技术推广到了有向图上，并对最常见的控制流图——可规约流图做了讨论。

然而，我们无法寄希望让操作系统借助访存图来优化页面的换入换出——尽管这样的算法确实存在——因为这样做的成本太高了。只有当图具有某些很特殊的性质时，理论才有可能指导我们选用合适的算法来提高效率。

除去页面置换问题，访存图的另外一个作用在于所谓程序重组。程序重组发生在编译时，通过将程序的内存布局重新组织，使得其拥有更好的访存局部性。

3.3 一般图竞争比的一个下界

在访存图模型中，访存图$G=(V,E)$是一个无向图，程序的访存序列是图G上的一条路径$\{p_1,p_2,\ldots,p_m\}$。我们称当前位置是p_t，如果我们处理道到的最后一次访存是p_t。很明显，当$G=K_M$时模型退化为一般访存序列的情景。

首先定义一些记号。$c_{A,k}(G)$表示算法A在物理内存页面数为k时，在图G上的近似比。由于访存图模型是一般页面置换问题的严格加强，最优算法OPT仍然是最优算法，即$c_{OPT,k}(G)=1$。在不引起歧义的情况下，我们用n表示可能访问的页面数$|V|$，$\mathcal{T}_i$表示图G节点数为i的联通子树集合，$leaf(T)$表示树T的叶子集合。对于算法A和访存图G，在处理到访存序列第t项时，处于内存中的页面集合为$\texttt{ATree}(t)$。

首先我们给出在固定访存图下，在线算法近似比的一个下界。

引理 3.1 *对于给定的树T，OPTTree(t)是树上的一个联通块，且当前位置$p_t \in$ OPTTree(t)。*

证明 3.1 *对访存序列长度做归纳。当需要换出一个页面时，最优算法总是换出下一次访问最晚的页面，由于访问一个叶子节点总是需要先访问其父亲，被换出的节点一定是OPTTree(t)的叶子。因此在任何时候OPTTree(t)都是树上的一个联通块。*

定理 3.1 *对于给定的图G和k和任何确定性在线算法A，有：*

$$c_{A,k}(G) \geq \max_{T\in\mathcal{T}_{k+1}} |leaf(T)|-1$$

证明 3.2 *不妨设G中叶子最多的、包含$k+1$个节点的联通子树是T，我们仅在这棵树上构造访存路径。根据上面的引理，任何时候$OPT(T,t)$都是树上的一个联通块，那么唯一不在内存中的节点$P(t)$总是叶子节点。对于给定的确定性算法A，我们总是不断的构造走向$P(t)$的路径，设这样的游走进行了m次，算法A缺页的次数就是m。*

我们将这样的m次游走分割为若干段$b_1,b_2,\ldots,b_t$，每一段的结尾恰好访问了$|leaf(T)|-1$个不同的叶子节点。很明显，通过总是换出下一段中第一个访问的叶子节点，最优算法只会在每一段开头产生一次缺页，总的缺页次数为$c_{OPT,k}\leq m/(|leaf(T)|-1)$，因此：

$$c_{A,k} \geq |leaf(T)|-1$$

值得注意的是，这个下界并不总是足够紧的。在上一章对一般访存序列上确定性算法竞争比下界的证明中，访存图总是一个$k+1$个节点的环。对于环的情景，这里的下界只能说明$c_{A,k}(G) \geq 1$，但事实上我们已经说明了$c_{A,k}(G) = \Theta(\log n)$。

3.4 LRU在树上的情景

[5]中证明了，LRU在任意树上达到了上面给出的竞争比下界，不妨记下界为$a(G)$。这里我们详细讨论一下这个内容。

首先我们很容易说明，LRU在内存中的节点也总是构成一个联通块。我们称时刻t在LRU在内存中的节点构成了LRUTree(t)，OPT在内存中的节点构成了OPTTree(t)。我们称LRUTree(t)-OPTTree(t)中的节点是孤独的LRU节点，反过来OPTTree(t)-LRUTree(t)是孤独的OPT节点。

我们使用记账法，说明OPT的一次缺页至多会“导致”LRU之后的$a(G)$次缺页。一些在LRUTree内的节点上放有棋子。当一次OPT缺页发生时，设当前访问的节点是v。我们在所有LRUTree上所有叶子到v的唯一路径上，第一个没有棋子的节点上放一个棋子，如果不存在这样的节点就不放置。当一个节点被换出内存时，我们拿走这个节点上的棋子。

引理 3.2 *每个孤独的LRU节点上都放有一个棋子。*

证明 3.3 *我们只需说明在发生缺页时性质可以保持即可。设当前访问的页面是r_t，考虑两个算法发生缺页的三种情况：*

1. *OPT未发生缺页，LRU发生缺页：这时会将LRUTree的一个叶子换出，将r_t加入LRUTree。由于OPT未发生缺页，这个节点不会是孤独的LRU节点。*

2. *OPT发生缺页，LRU发生缺页。与上一种情况相比，只需说明OPTTree中换出的节点要么不在LRUTree中，要么放有一个棋子。OPTTree中换出的节点一定是一个叶子，记为l_t，其在OPTTree上的唯一父亲是p_t。如果$l_t \in$ LRUTree，在断开(p_t, l_t)这条边后l_t一侧的子树T'中，这次缺页发生之前只有l_t不是孤独的LRU节点，也只有l_t上没有棋子。那么根据放置棋子的方法，l_t上会被放置一个棋子，待证的性质得到保持。*

3. *OPT发生缺页，LRU未发生缺页。也只需要讨论上一类中的情景。*

引理 3.3 每一次*OPT*缺页至多会放置$a(G)$个棋子。

证明 3.4 考虑一个*OPT*缺页时引用的页r_t。如果$r_t \notin$ *LRUTree*，根据访存图的定义这个节点一定是*LRUTree*$\cup \{r_t\}$的叶子，这棵$k+1$个节点的树的除去r_t的叶子数一定不超过$a(G)$，因此放置的棋子也不超过$a(G)$；如果$r_t \in$ *LRUTree*，那么在这次访存之前r_t是一个孤独的*LRU*节点。考虑包含孤独的*LRU*节点构成的森林中r_t所在的子树T'。由于*OPTTree*构成了一个联通块，至少有一个T'中的叶子节点也是*LRUTree*的叶子节点，那么这个节点到r_t的路径上都是孤独的*LRU*节点，因此不会放置任何新的棋子。换言之，放置的棋子数不超过*LRUTree*的叶子数-1，因此也不超过$a(G)$。

引理 3.4 *LRU*总是换出那些放有棋子的节点。

证明 3.5 假设在时刻t_1时上一次访问最久的*LRUTree*的叶子节点v上没有棋子。不妨设上一次访问节点v的时刻为t_0，在t_1包含v且不包含其他*LRUTree*节点的子树为$T(v)$。我们说明下面的事实：

引理 3.5 在时刻t_1，所有孤独的*OPT*节点都在$T(v)-\{v\}$中。

如果有了上面的引理，我们立刻可以说明：在下一次*LRU*缺页发生时，要么同时出现了一次*OPT*缺页从而v获得棋子，要么在下一次访问之前，t_1被重新访问，不再是上一次访问最久的节点。因而就证明了*LRU*总是换出放有棋子的节点。

引理证明分为四步：

1. 在时刻t_0，$T(v)-\{v\}$中的*OPT*节点数，大于或等于$T(v)-\{v\}$中没有棋子的*LRU*节点数：这是由于所有孤独的*LRU*节点都有棋子。

2. 在时刻t_0，$T(v)-\{v\}$中没有棋子的*LRUTree*节点数，大于或等于在时间段$[t_0,t_1]$中放置在$T(v)-\{v\}$中的棋子数：这是由于每一个没有棋子的节点至多只会放置一次。

3. 在时间段$[t_0,t_1]$中放置在$T(v)-\{v\}$中的棋子数，大于或等于$[t_0,t_1]$时间段中发生*OPT*缺页的次数：这是由于v上没有棋子，所以每次*OPT*缺页至少会在$T(v)-\{v\}$中放置一个棋子。

4. $[t_0,t_1]$时间段中发生*OPT*缺页的次数，大于或等于t_1时刻孤独的*LRU*节点数，加上时间段$[t_0,t_1]$中$T(v)-\{v\}$内被换出的*OPT*节点数：

- t_1时刻孤独的*LRU*节点曾在$[t_0, t_1]$中被访问过，也曾是*OPT*节点。这个*OPT*节点被换出对应了一次*OPT*缺页。
- $[t_0, t_1]$中$T(v) - \{v\}$被换出的每一个*OPT*节点都对应了一次*OPT*缺页。

因此，在时刻t_0，$T(v) - \{v\}$中的*OPT*节点数，大于或等于t_1时刻孤独的*LRU*节点数，加上时间段$[t_0, t_1]$中$T(v)$内被换出的*OPT*节点数。换言之，t_1时刻$T(v) - \{v\}$中的*OPT*节点数大于等于孤独的*LRU*节点数。

由于孤独的*LRU*节点数等于孤独的*OPT*节点数，那么孤独的*OPT*节点数小于等于$T(v) - \{v\}$内的*OPT*节点数。因为t_1时刻v是一个叶子节点，$T(v) - \{v\}$中的每一个*OPT*节点都是孤独的。所以每一个孤独的*OPT*节点都在$T(v) - \{v\}$中，这就是我们要证明的。

由上面的三个定义立刻可以说明$c_{LRU,k} = a(G)$，因此我们可以断言下面的定理：

定理 3.2 *对于任意为树的访存图G，LRU是$a(G)$-Competitive的。在竞争比的意义下，LRU是最优的树上页面置换算法。*

树带给访存序列一个非常强的限制：一旦离开了某个子树$T(v)$，想要再次访问$T(v)$，一定要先访问节点v。在这个限制下，LRU“过去的信息”可以很好的预测“未来的信息”，从而达到了确定性算法的新下界。

参考文献

[1] Belady, L. A . A study of replacement algorithms for a virtual-storage computer[J]. IBM Systems Journal, 1966, 5(2):78-101.

[2] Fiat A , Karp R M , Luby M , et al. Competitive paging algorithms[J]. Journal of Algorithms, 1991, 12(4):685-699.

[3] Sleator D D , Tarjan R E . Amortized efficiency of list update and paging rules[J]. Communications of the Acm, 1985, 28(2):202-208.

[4] Denning P J. Working Sets Past and Present[J]. IEEE Transactions on Software Engineering, 1980, 6(1): 64-84.

[5] Borodin A, Raghavan P, Irani S, et al. Competitive paging with locality of reference[J]. symposium on the theory of computing, 1991, 50(2): 249-259.

[6] Albers S. Online algorithms: a survey[J]. Mathematical Programming, 2003, 97(1): 3-26.

[7] Chrobak M, Noga J. LRU is better than FIFO[J]. symposium on discrete algorithms, 1998, 23(2): 78-81.

[8] Irani S, Karlin A R, Phillips S J, et al. Strongly Competitive Algorithms for Paging with Locality of Reference[J]. SIAM Journal on Computing, 1996, 25(3): 477-497.

[9] https://piazza.com/class/i5j09fnsl7k5x0?cid=1296

[10] https://riptutorial.com/algorithm/topic/8022/online-algorithms

[11] https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-854j-advanced-algorithms-fall-2005/lecture-notes/n23online.pdf

[12] https://en.wikipedia.org/wiki/Page_replacement_algorithm

[13] https://riptutorial.com/algorithm/example/25916/paging–online-caching-