一、实验内容

结合数据结构的相关知识，使用 LRU 的策略，对一组访问序列进行内部的 LRU 置换算法是选择最近最久未使用的页面予以置换。

二、实验目的

了解和掌握寄存器分配和内存分配的有关技术。

三、实验要求

结合数据结构的相关知识，使用 LRU 的策略，对一组访问序列进行内部的 LRU 置换算法是选择最近最久未使用的页面予以置换。该算法赋予每个页面一个访问字段，用来记录一个页面自上次被访问以来经历的时间 T，当须淘汰一个页面时，选择现有页面中 T 值最大的，即最近最久没有访问的页面。这是一个比较合理的置换算法。

举例说明此问题，例如：

有一个 CACHE 采用组相连映象方式。每组有四块，为了实现 LRU 置换算法，在快表中为每块设置一个 2 位计数器。我们假设访问序列为“1,1,2,4,3,5,2,1,6,7,1,3”。在访问 CACHE 的过程中，块的装入，置换及命中时，具体情况如下表所示：

	1	1	2	4	3	5	2	1	6	7	1	3
Cache块 0	1	1	1	1	1	5	5	5	5	7	7	7
Cache块 1			2	2	2	2	2	2	2	2	2	3
Cache块 2				4	4	4	4	1	1	1	1	1
Cache块 3				3		3	3	3	6	6	6	6
	装入	命中	装入	装入	装入	置换	命中	置换	置换	置换	命中	置换

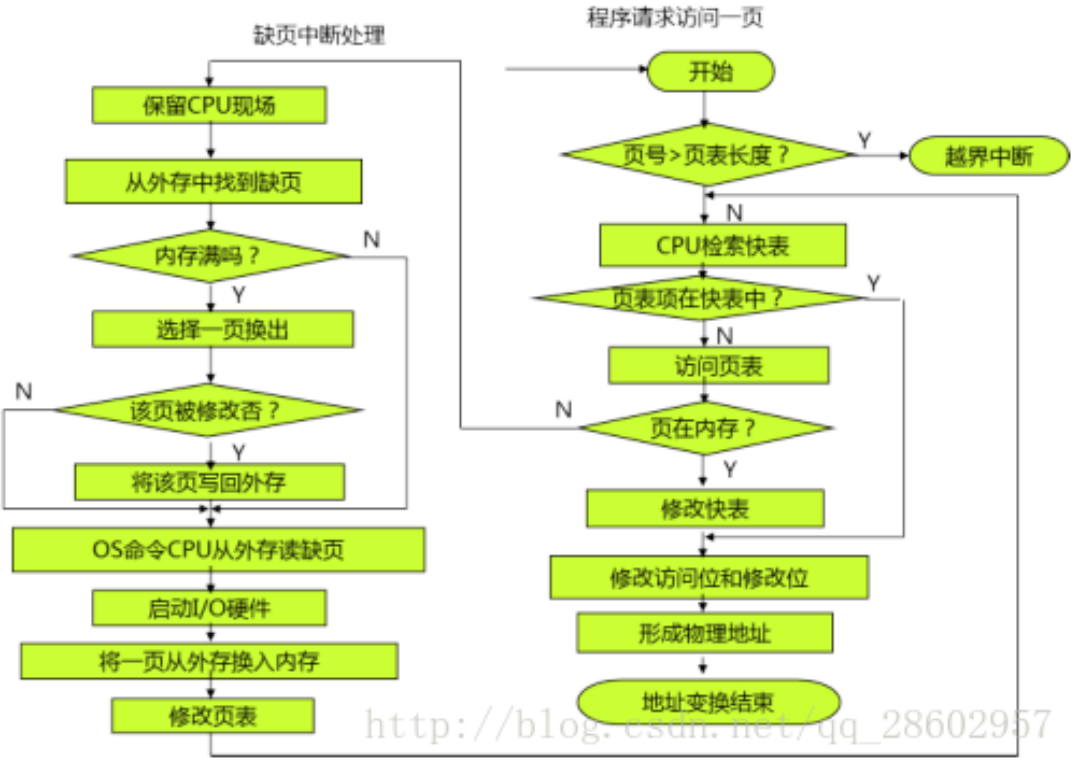
四、实验分析与设计

1. 虚地址生成与映射

（1）页式存储管理把内存分割成大小相等位置固定的若干区域，叫内存页面，内存的分配以“页”为单位，一个程序可以占用不连续的页面，逻辑页面的大小和内存页面的大小相同，内外存的交换也以页为单位进行，页面交换时，先查询快表，若快表中找不到所需页面再去查询页表，若页表中仍未找到说明发生了缺页中断，需先将所需页面调入内存再进行存取。

（2）实验中的页面大小设定为 3。response()函数的主要目的是发现缺页中断，若在页面调度过程中，出现了缺页情况，则该函数需要将缺页中断响应，使其发现缺页的情况，再跟据不同的调页算法来调度。

（3）采用随机数表示每次生成的页面的虚地址，除以页面大小的到相应的页号，采用页面置换算法完成实验



2. 最近最久未使用（LRU）算法

这种算法的基本思想是：利用局部性原理，根据一个作业在执行过程中过去的页面访问

历史来推测未来的行为。它认为过去一段时间里不曾被访问过的页面，在最近的将来可能也不会再被访问。所以，这种算法的实质是：当需要淘汰一个页面时，总是选择在最近一段时间内最久不用的页面予以淘汰。

再对上面的实例采用 LRU 算法进行页面置换，如图 3-29 所示。进程第一次对页面 2 访问时，将最近最久未被访问的页面 7 置换出去。然后访问页面 3 时，将最近最久未使用的页面 1 换出。

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块 1	7	7	7	2		2		4	4	4	0			1		1		1		
物理块 2		0	0	0		0		0	0	3	3			3		0		0		
物理块 3			1	1		3		3	2	2	2			2		2		7		
缺页否	√	√	√	√		√		√	√	√	√			√		√		√		

实际上，LRU 算法根据各页以前的情况，是“向前看”的，而最佳置换算法则根据各页以后的使用情况，是“向后看”的。

LRU 性能较好，但需要寄存器和栈的硬件支持。LRU 是堆栈类的算法。理论上可以证明，堆栈类算法不可能出现 Belady 异常。FIFO 算法基于队列实现，不是堆栈类算法。

3．最佳置换算法（OPT）（理想置换算法）

从主存中移出永远不再需要的页面；如无这样的页面存在，则选择最长时间不需要访问的页面。于所选择的被淘汰页面将是以后永不使用的，或者是在最长时间内不再被访问的页面，这样可以保证获得最低的缺页率。

最佳置换算法可以用来评价其他算法。假定系统为某进程分配了三个物理块，并考虑有以下页面号引用串：

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

进程运行时，先将 7, 0, 1 三个页面依次装入内存。进程要访问页面 2 时，产生缺页中断，根据最佳置换算法，选择第 18 次访问才需调入的页面 7 予以淘汰。然后，访问页面 0 时，因为已在内存中所以不必产生缺页中断。访问页面 3 时又会根据最佳置换算法将页面 1 淘汰……依此类推，如图 3-26 所示。从图中可以看出采用最佳置换算法时的情况。

可以看到，发生缺页中断的次数为 9，页面置换的次数为 6。

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块 1	7	7	7	2		2		2			2			2				7		
物理块 2		0	0	0		0		4			0			0				0		
物理块 3			1	1		3		3			3			1				1		
缺页否	√		√	√		√		√			√			√				√		

4．先进先出置换算法（FIFO）

是最简单的页面置换算法。

这种算法的基本思想是：当需要淘汰一个页面时，总是选择驻留主存时间最长的页面进行淘汰，即先进入主存的页面先淘汰。其理由是：最早调入主存的页面不再被使用的可能性最大。

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块 1	7	7	7	2		2	2	4	4	4	0			0	0			7	7	7
物理块 2		0	0	0		3	3	3	2	2	2			1	1			1	0	0
物理块 3			1	1		1	0	0	0	3	3			3	2			2	2	1
缺页否	√	√	√	√		√	√	√	√	√	√			√	√			√	√	√

这里仍用上面的实例，采用 FIFO 算法进行页面置换。进程访问页面 2 时，把最早进入内存的页面 7 换出。然后访问页面 3 时，再把 2, 0, 1 中最先进入内存的页换出。由图 3-27 可以看出，利用 FIFO 算法时进行了 12 次页面置换，比最佳置换算法正好多一倍。

FIFO 算法还会产生当所分配的物理块数增大而页故障数不减反增的异常现象，这是由 Belady 于 1969 年发现，故称为 Belady 异常，如图 3-28 所示。只有 FIFO 算法可能出现 Belady 异常，而 LRU 和 OPT 算法永远不会出现 Belady 异常。

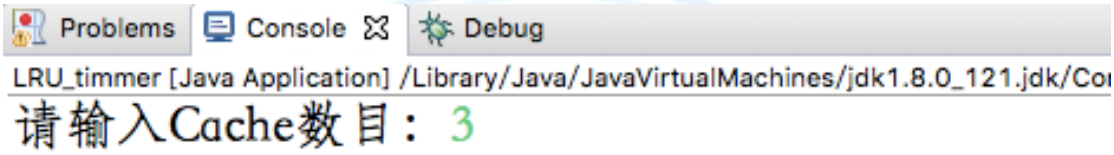
访问页面	1	2	3	4	1	2	5	1	2	3	4	5
物理块 1	1	1	1	4	4	4	5			5	5	
物理块 2		2	2	2	1	1	1			3	3	
物理块 3			3	3	3	2	2			2	4	
缺页否	√	√	√	√	√	√	√			√	√	
		1	1	1			5	5	5	5	4	4
物理块 2*		2	2	2			2	1	1	1	1	5
物理块 3*			3	3			3	3	2	2	2	2
物理块 4*				4			4	4	4	3	3	3
缺页否	√	√	√				√	√	√	√	√	√

内存的页面中“最老“的页面，会被新的网页直接覆盖，而不是“最老“的页面先出队，然后新的网页从队尾入队。

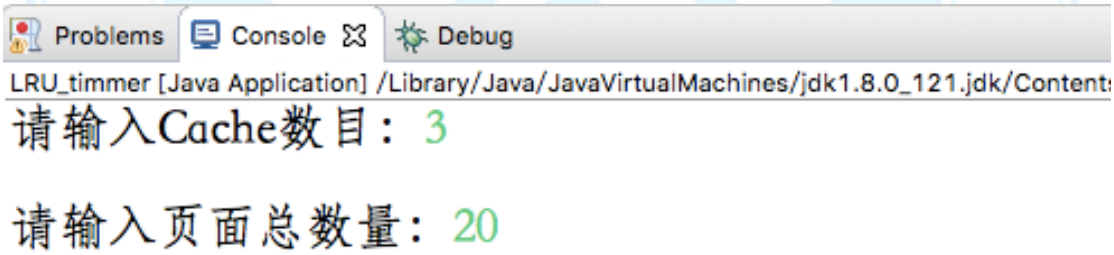
五、实验运行结果和相关源代码

1.实验结果

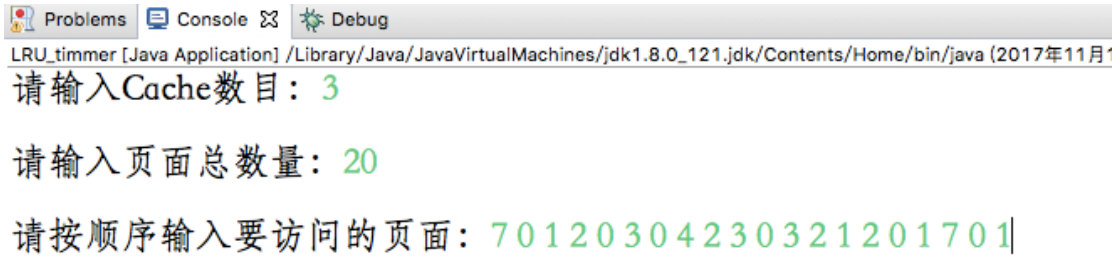
首先输入Cache的数目：



在输入请求访问的页面数量：



然后按顺序输入依次访问的页面编号：



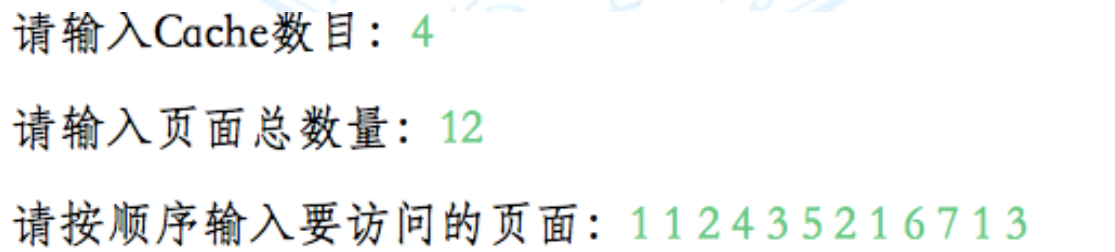
然后可以得到页面替换的结果：
其中，最后一行出现标志则意味着发生缺页，需要替换。

```
--------------------------------------------------------------------------------
Pages:     7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
--------------------------------------------------------------------------------
Cache 1:   7 7 7 2 2 2 2 4 4 4 0 0 0 1 1 1 1 1 1 1
Cache 2:     0 0 0 0 0 0 0 0 3 3 3 3 3 3 0 0 0 0 0
Cache 3:       1 1 1 3 3 3 2 2 2 2 2 2 2 2 2 7 7 7
--------------------------------------------------------------------------------
             $ $ $ $   $   $ $ $ $     $   $   $
--------------------------------------------------------------------------------
```

与前面提到的相符：

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块 1	7	7	7	2		2		4	4	4	0			1		1		1		
物理块 2		0	0	0		0		0	0	3	3			3		0		0		
物理块 3			1	1		3		3	2	2	2			2		2		7		
缺页否	√	√	√	√		√		√	√	√	√			√		√		√		

用题目给的样例测试：



结果如下：

```
-----------------------------------------------
Pages:      1 1 2 4 3 5 2 1 6 7 1 3
-----------------------------------------------
Cache 1:    1 1 1 1 1 2 4 3 5 2 2 6
Cache 2:      2 2 2 4 3 5 2 1 6 7
Cache 3:        4 4 3 5 2 1 6 7 1
Cache 4:          3 5 2 1 6 7 1 3
-----------------------------------------------
                $   $ $ $ $   $ $ $   $
-----------------------------------------------
```

	1	1	2	4	3	5	2	1	6	7	1	3
Cache块 0	1	1	1	1	1	5	5	5	5	7	7	7
Cache块 1			2	2	2	2	2	2	2	2	2	3
Cache块 2				4	4	4	4	1	1	1	1	1
Cache块 3				3		3	3	3	6	6	6	6
	装入	命中	装入	装入	装入	置换	命中	置换	置换	置换	命中	置换

结果相符。