FIFO (First In FIrst Out) 是最简单的算法，原理跟名字一样，**“如果一个数据最先进入缓存中，则应该最早淘汰掉”**。把缓存中的数据看成一个队列，最先加入的数据位于队列的头部，最后加入位于队列的尾部。当缓存空间不足需要执行缓存淘汰操作时，从队列的头部开始淘汰。 如下所示，假设我们的缓存可以缓存 3 对数据，1 加入时处于队列的头部，2 和 3 加入时缓存空间充足。当 4 加入时，执行缓存淘汰，由于 1 处于队列的头部，所以被淘汰。同理 5 加入时，2 被淘汰。  
Java 中有单独的队列 Queue ，可以使用 LinkedList。

| **1** | **2** | **3** | **4** | **5** |
| --- | --- | --- | --- | --- |
|  |  | 3=c | 4=e | 5=f |
|  | 2=b | 2=b | 3=c | 4=e |
| 1=a | 1=a | 1=a | 2=b | 3=c |

LRU (Least Recently Used) 的核心思想是基于**“如果数据最近被访问过，它在未来也极有可能访问过”**。同样把缓存看成一个队列，访问一个数据时，如果缓存中不存在，则插入到队列尾部；如果缓存中存在，则把该数据移动到队列尾部。当执行淘汰操作时，同样从队列的头部开始淘汰。 如下所示，1、2、3 加入时缓存空间充足，接下来 1 又被访问了一次，所以 1 被移动到队列尾部。当 4 加入时，执行缓存淘汰，2 位于队列头部被淘汰。 Java 中可以直接使用 LinkedHashMap 来实现。

| **1** | **2** | **3** | **1** | **4** |
| --- | --- | --- | --- | --- |
|  |  | 3=c | 1=a | 4=e |
|  | 2=b | 2=b | 3=c | 1=a |
| 1=a | 1=a | 1=a | 2=b | 3=c |

LFU （Least Frequently Used）的核心思想是**“如果一个数据在最近一段时间内使用次数很少，那么在将来一段时间内被使用的可能性也很小”**，会记录数据访问的次数，当需要进行淘汰操作时，淘汰掉访问次数最少的数据。 如下所示，一开始 1 被连续访问了两次，接下来 2 被访问一次，3 被访问一次，按照访问次数排序，访问次数少的处于队列头部。当 4 加入时，执行缓存淘汰，2 位于队列头部被淘汰。

| **1** | **1** | **2** | **3** | **4** |
| --- | --- | --- | --- | --- |
|  |  |  | 1=a (2) | 1=a (2) |
|  |  | 1=a (2) | 3=c (1) | 4=e(1) |
| 1=a (1) | 1=a (2) | 2=b (1) | 2=b (1) | 3=c (1) |