请求调页存储管理方式模拟

项目旨在模拟请求调页存储管理方式，以更好地了解不同调页算法在实际模拟过程中的对比，熟悉调页算法的具体实现细节。

项目需求：

1. 首先对于模拟的内存管理，项目允许用户定义自己需要的内存大小与内存块大小，需要模拟的作业指令总数。
2. 对于模拟，项目将可视化地展示出内存块的状态与指令执行的顺序。
3. 项目模拟将采用随机模拟地模拟指令执行的顺序。
4. 项目也允许用户按照自己的需要输入所需要的指令。

随机模拟方法：

对于某个总共有n条指令未执行的作业

1. 首先在随机模拟一个指令m
2. 随机一个1-4的数k，然后执行指令m后k条指令
3. 然后随机模拟指令1-m-1中的某条指令m1，执行指令m1
4. 随机一个1-4的数k1，然后执行指令m1后k1条指令
5. 随机模拟m+1-n中的某条指令m2，执行m2
6. 随机一个1-4的数k2，然后执行指令m1后k2条指令
7. 重复执行上述步骤直至所有指令执行完成。

项目实现：

算法实现：

模拟除了采用随机模拟作为比较外，模拟将采用LRU和FIFO调页存储管理系统。

LRU算法优化与实现：

在LRU实现中，易分析得知，算法中最消耗时间的地方在于每次需要检索最久未使用的内存块和检索申请的内存块是否已经存在，而这两项检索，如果采用正常的遍历搜索，时间复杂度均为O（n），为了优化检索所花的时间，我选择构造了一个基于哈希表的链表用于检索，使得检索所需要的时间简化到了O（1）。

数据结构：

实现过程中，数据结构使用的是基于哈希表的链表。简而言之，就是在单向链表的基础上，使用哈希表维护的内存块进入顺序与是否存在于哈希表中。

而对于每个内存块，使用PageInfo类来维护内存块中，维护内存块中指令信息和内存块大小。而对于每个作业，将维护一个PageInfo表，作为类似PCB的表以保存所有内存的信息。

实现细节：

LRU具体实现：

对于某个申请的内存块，使用HashMap检索内存块是否存在

将内存块加入链表的最后

将链表头节点即最近最少被调用的从链表中去除，将新加入的内存块的信息加入链表尾。

FIFO具体实现：

对于某个申请的内存块，使用HashMap检索内存块是否存在

将链表头节点即最近最少被调用的从链表中去除，将新加入的内存块的信息加入链表尾。

Random具体实现：

对于某个申请的内存块，使用HashMap检索内存块是否存在

将任一内存块进行调页存储管理，将旧内存块除去，将新内存块加入