《操作系统分析与设计实习》成绩单

开设时间：2019学年第二学期

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学号 | test | 姓名 | Herry Wu | | 成绩 | 100 |
| 实验题目 | 题目一：页面置换算法的模拟实现及命中率对比  模拟实现 OPT（最佳置换）、FIFO 和 LRU 算法，并计算缺页率。 | | | | | |
| 自我评价 | 通过这次实验，我更加深入地理解了三大页面置换算法——OPT（最佳置换）、FIFO 和 LRU 算法，在编辑算法代码以及debug的过程中，更加理解了置换算法的核心思路。  在完成题目算法要求的情况下，我还用图形界面实现了输出显示，模拟页面置换算法及命中率对比，最后并根据对比情况输出文字总结，以达到在操作系统的级别上真实模拟。 | | | | | |
| 教师评语 | 评价指标：  题目内容和要求完成情况 优 □ 良 □ 中 □ 差 □  对算法原理的理解程度 优 □ 良 □ 中 □ 差 □  程序设计水平 优 □ 良 □ 中 □ 差 □  程序运行效果及正确性 优 □ 良 □ 中 □ 差 □  课程设计报告结构清晰 优 □ 良 □ 中 □ 差 □  报告中总结和分析详尽 优 □ 良 □ 中 □ 差 □ | | | | | |
|  |  | | 教师签名 |  | | |

### 需求分析

明确陈述说明程序设计的任务，强调的是程序要做什么，主要包括：

1、输入的形式和输入值的范围；

①程序的输入值是执行指令的地址流以及其对应的页号流，该指令的地址流包含400 个随机产生的地址数据，页号流也包含400个页号，页号流有地址流转换得到。其中 地 址流中50%的地址访问是顺序执行的，另外 50%就是非顺序执行，且地址在前半部地 址空间和后半部地址空间均匀分布.

②400个地址流输入值的范围在[0,399]之间，其中50%在[0,199],50%在[200,399].

③400个页号流输入值的范围在[0,40]之间，页号流由地址流转换而来.

2、输出的形式；

①输出值是三个页面置换算法的从4-40个页框的命中率.

②输出形式是用html、css、JavaScript搭建的一个图形用户界面，并将数据以表格的形 式动态地输出显示在图形界面上.

3、程序所能达到的功能；

①程序能按要求随机产生符合要求的地址流，并将其转换为对应页号流.

②程序利用产生的页号流，模拟页面置换的三个算法——FIFO、LUR、OPT算法，计算 三个置换算法在页框号在4-40的范围内的命中率，并将其以表格的形式输出显示到图 形用户界面上.

4、测试数据：包括正确的输入及其输出结果和含有错误的输入及其输出结果。

①正确的测试数据：

当随机生成的数据符合地址流以及页号流的数据范围时，便会正确地模拟三个置换算法 的执行过程，并将正确的命中率输出显示在图形用户界面上

②错误的测试数据：

若随机产生的输入数据不符合地址流以及页号流的要去范围，那么则不会正确地模拟置 换算法的执行过程，并产生范围不在[0,1]之间的错误命中率数据.

1. 概要设计

说明本程序中用到的所有抽象数据类型的定义、主程序的流程以及各程序模块之间的层次 (调用)关系。

2.1、本程序的主要开发语言是web前端开发语言的html、css、JavaScript以及jQuery，程序主题框架分为三大部分：simulation.html、simulation.css以及simulation.js。其中simulation.html实现的是图形用户界面的内容，simulation.css实现的是图形用户界面的样式，simulation.js是程序的主体，实现的是程序的核心算法部分，实现操作系统的三大页面置换算法——FIFO算法、LRU算法以及OPT算法；以及实现图形用户界面的动作，与程序界面使用者的交互，例如单击按钮产生响应等页面动作。

2.2、simulation.html程序：实现了程序的图形用户界面的内容部分，即需要显示输出哪些内容。将地址流与页号流、4-40个页框调用下的三大置换算法——FIFO算法、LRU算法以及OPT算法分别命中率显示输出在图形用户界面上。

2.2.1、

2.3、simulation.css程序：实现对图形用户界面的内容布局以及样式的美化，即输出显示的内容分美化、合理地显示出来，给使用者更好的使用体验，增加程序的美感。

2.3.1、

2.4、simulation.js程序主要使用的数据结构是数组和队列，定义了5个数组和3个队列，simulation.js还定义了7个函数，用于实现三大页面置换算法——FIFO算法、LRU算法以及OPT算法的算法逻辑以及实现页面动作交互逻辑，分别是

2.4.1、5个数组分别是：

addressArr[] : 地址流数组，用于存放随机生成的400个范围在0-399的指令地址

pageArr[] : 页号流数组，用于存放400个由地址流转换过来的范围在0-39的页号

fifoArr[] : FIFO算法的页框数组，用于存放FIFO算法置换过程中的页框

lruArr[] : LRU算法的页框数组，用于存放LRU算法置换过程中的页框

optArr[] : OPT算法的页框数组，用于存放OPT算法置换过程中的页框

2.4.2、3个队列分别是：

fifoQueue : 页号在页框中的模拟队列，用于模拟FIFO算法的页号在页框中的置换队列

lruQueue : 页号在页框中的模拟队列，用于模拟LRU算法的页号在页框中的置换队列

optQueue : 页号在页框中的模拟队列，用于模拟OPT算法的页号在页框中的置换队列

2.4.3、7个函数分别是：

function random( ) { } : 用于生成400个范围在0-399的地址流，并将其转换位400个范围在0-39的页号流

function FIFO( ) { } : 实现FIFO页面置换算法

function LRU( ) { } : 实现LRU页面置换算法

function OPT( ) { } : 实现OPT页面置换算法

function replace( ) { } : simulation.js程序的主函数，调用FIFO()函数、LRU()函数以及OPT()函数，实现从4-40页框的三大页面置换算法——FIFO算法、LRU算法以及OPT算法的页面置换，与此同时将4-40个页框对应的FIFO算法、LRU算法以及OPT算法的命中率以表格的形式显示在图形用户界面上。

$(".btn").click( ) : 为class = “btn”的按钮绑定一个单击响应函数，当单击按钮时调用random()函数以及replace()函数。单击时随机生成400个范围在0-399的地址流，并将其转换为400个范围在0-39的页号流，与此同时将地址流与页号流显示在图形用户界面的方框中；然后调用replace()函数，实现三大页面置换算法——FIFO算法、LRU算法以及OPT算法，并分别计算它们的命中率；最后总结三个算法的规律与特点，将总结的文字输出显示在图形用户界面上。

$(document).ready( ) : 页面加载函数，将单击响应函数$(".btn").click(function ( ) { }放在其中，因为只有但页面文档节点对象DOM加载完成后才能对页面节点对象进行操作。

2.4.4、程序整体实现流程图：

simulation.css

实现页面图形用户界面样式

simulation.js

程序主体部分，实现FIFO、LRU以及OPT逻辑，以及实现程序的行为

simulation.html

实现页面图形用户界面内容

2.4.5、主函数流程图：

随机产生地址流

转换为页号流

OPT()函数

LRU()函数

FIFO()函数

random()函数

$(".btn").click( )函数：

单击响应函数，调用random()、FIFO()、LUR()、OPT()

$(document).ready( )函数：

主函数调用$(".btn").click( )函数，页面加载函数，让整个页面加载显示

### 详细设计

实现概要设计中定义的所有数据类型，对每个操作只需要写出伪码算法；对主程序和其他模 块也都需要写出伪码算法(伪码算法达到的详细程度应能够按照伪码算法在计算机键盘上直接输 入高级程序设计语言程序)；画出函数的调用关系图。

3.1、function random( ) { } : 用于生成400个范围在0-399的地址流，并将其转换位400个范围在0-39的页号流.

3.1.1、算法思路：

①定义变量：

定义两个数组数组：地址流数组和页号流数组addressArr[], pageArr[]

②通过随机数产生一个地址序列，共产生 400 条。其中 50%的地址访问是顺序执 行的，另外 50%就是非 顺序执行。且地址在前半部地址空间和后半部地址空间均匀 分布。具体产生方法如下：

1) 在前半部地址空间，即[0，199]中随机选一数 m，记录到地址流数组中（这是

非顺序执行）；

1. 接着“顺序执行一条指令”，即执行地址为 m+1 的指令，把 m+1 记录下来

3) 在后半部地址空间，[200，399]中随机选一数 m’，作为新指令地址；

4) 顺序执行一条指令，其地址为 m’+1；

5) 重复步骤 1~4 100次，直到产生 400 个指令地址。

③得到地址流数组后再将数组中的各个元素都除以10取其商的整数部分，得到的数组 便是页号流数组

3.1.2、random()函数流程图：

开始

定义地址流数组和页号流数组变量

for循环100次生成400个地址流

一次循环产生4个，i : 0-100

①随机生成一个[0,199]的整数m

②将m、（m+1）推进地址流数组中

③随机生成一个[200,399]的整数m’

④将m’、（m’+1）推进地址流数组中

i<100

将地址流转换为页号流

结束

是

否

3.1.3、function random( ) { }函数伪代码如下：

var addressArr, pageArr;

function random() {

addressArr = [];

var randomNum;

for(let i = 0; i < 100; i++) { //将数组按要求填满400个数

randomNum = Math.floor(Math.random() \* 200); //随机生成一个[0,200)之间的整数

addressArr.push(randomNum); //将m推进队列里

if(randomNum == 199) { //如果是199，则加1就溢出了

addressArr.push(0); //将0推进队列里

} else {

addressArr.push(randomNum + 1); //将m+1推进队列里

}

randomNum = Math.floor(Math.random() \* 200 + 200); //随机生成一个[200,400)之间的整数

addressArr.push(randomNum); //将m’推进队列里

if(randomNum == 399) { //如果是399，则加1就溢出了

addressArr.push(200); //将200推进队列里

} else {

addressArr.push(randomNum + 1); //将m’+1推进队列里

}

}

pageArr = addressArr.map(

function(x) {

return(Math.floor(x / 10));

}

);

}

3.2、function FIFO( ) { } : 实现FIFO页面置换算法

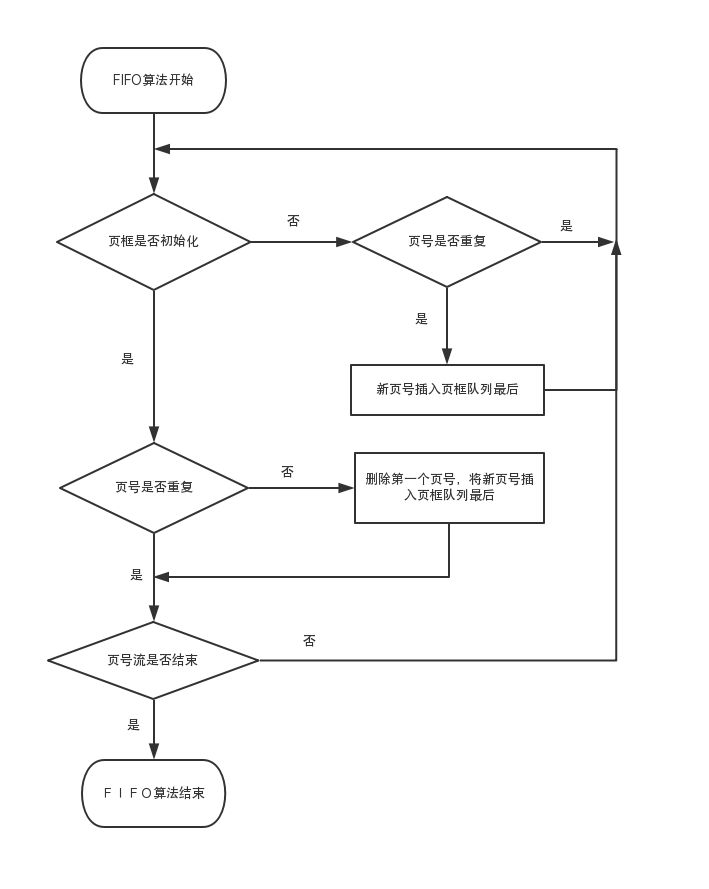
3.2.1、算法思路：

建立两个数组，一个表示页地址流数组，一个表示页框数组；

循环页地址流数组里的元素，将页框数组当成队列来使用，达到先进先出的效果；

相关的数组类的方法：push()、shift()、indexOf()

3.2.2、FIFO()函数流程图



3.2.2、伪代码如下：

/\*\*\*\*\*\*fifo\*\*\*\*\*\*/

function FIFO(k, fifoArr, pageArr) {

for(let i = 0; i < pageArr.length; i++){

if(fifoArr.length < k){ //页框初始化

if((fifoArr.indexOf(pageArr[i])) == -1){ //若页框数组里没有该页号了，则加入

fifoArr.push(pageArr[i]); //进队列

} else { //若页框数组里有该页号了，则不用再加入了，循环页地址流中的下一个页号

continue;

}

} else { //页框初始化后

if((fifoArr.indexOf(pageArr[i])) != -1){ //若页框数组里有该页号了，则循环页地址流中的下一个页号

continue;

} else { //若页框数组里没有该页号，则删掉第一个元素（因为它最早进来），再从后面插入一个元素

fifoArr.shift(); //出队列

fifoArr.push(pageArr[i]); //进队列

num2++; //同时缺页数加1

}

}

}

}

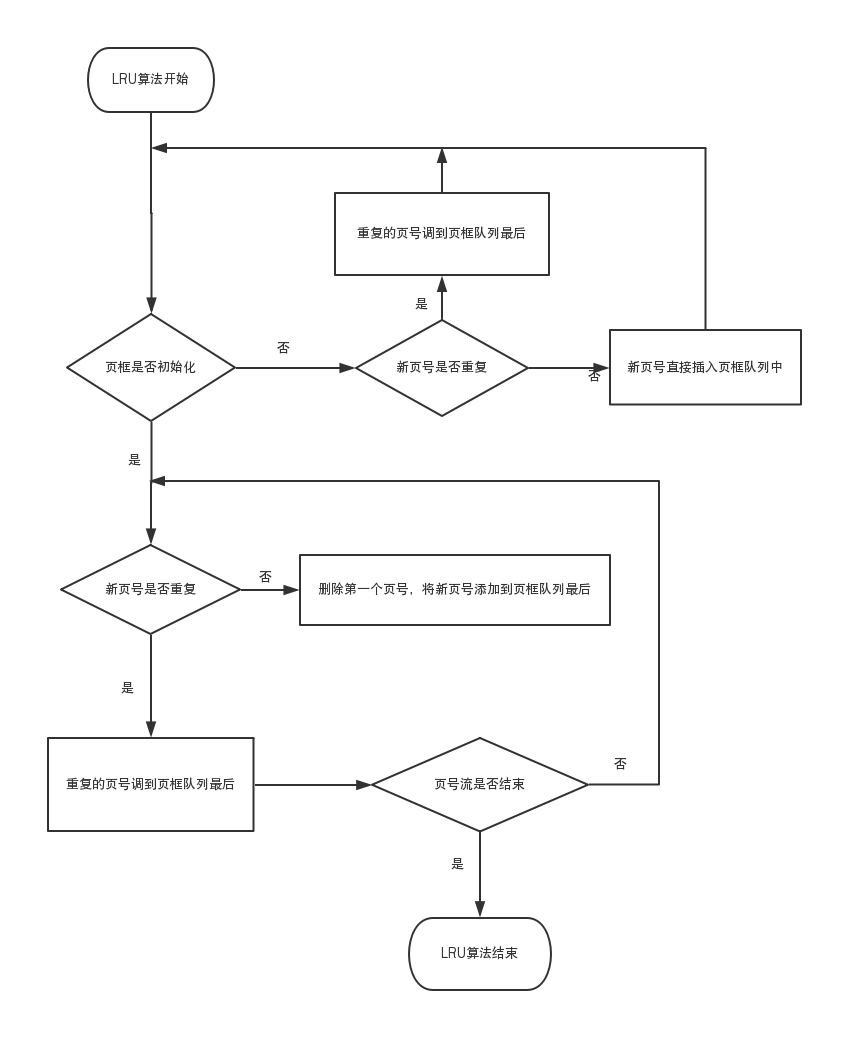
3.3、function LRU( ) { } : 实现LRU页面置换算法

3.3.1、算法思路：

建立两个数组，并且将第二个数组当队列使用，即将每次进来的从后面进入，删除的从前面删除，并且关键是将不缺页时重新访问的那个元素调出来重新插到最后一个

相关的数组类的方法：splice():指定数组下标删除某元素

3.3.2、LRU()函数流程图：



3.3.2、伪代码如下：

/\*\*\*\*\*lru\*\*\*\*\*\*/

function LRU(k, lruArr, pageArr) {

for(let i = 0; i < pageArr.length; i++) { //循环页地址流数组

if(lruArr.length < k) { //页框初始化时

if((lruArr.indexOf(pageArr[i])) == -1) { //页框队列里的元素都不与页地址流的页号不重复时

lruArr.push(pageArr[i]); //新的页号直接插入页框队列尾部

} else { //页框队列里有元素与页地址流的页号重复时，将该页号调出插到队列尾

lruArr.splice(lruArr.indexOf(pageArr[i]), 1); //删除该元素

lruArr.push(pageArr[i]); //插到队列尾

}

} else { //页框初始化后

if((lruArr.indexOf(pageArr[i])) == -1) { //不重复时,删除队头页号，新的页号插入队列尾

lruArr.shift(); //删除队头页号

lruArr.push(pageArr[i]); //新来的页号插到队列尾

num3++; //缺页数加1

} else { //重复时，与初始化时的有重复时的处理一样,将该重复的页号调到队列尾

lruArr.splice(lruArr.indexOf(pageArr[i]), 1); //删除

lruArr.push(pageArr[i]); //重新插到队列尾

}

}

}

}

3.4、function OPT( ) { } : 实现OPT页面置换算法

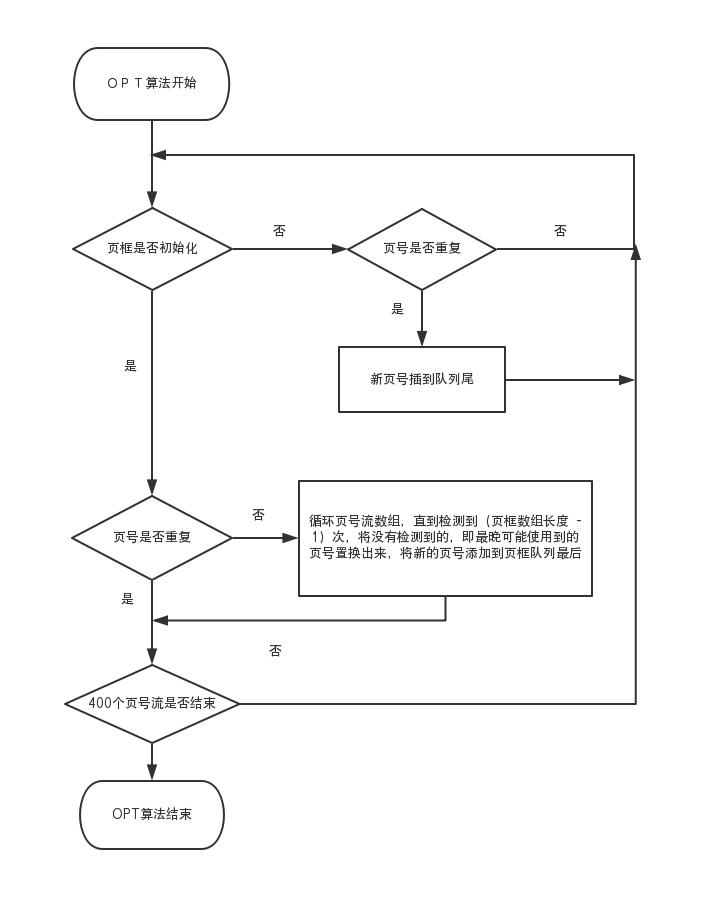
3.4.1、算法思路：

建立两个数组，并且将第二个数组当队列使用，即将每次进来的从后面进入，删除的从 前面删除；并且关键是当缺页且页框队列里没有相同的页号时，从页号流当前位置出发 往后循环页号流数组，找到接下来最不可能使用到的页号，将其从页框数组中置换出来， 并将新的页号插入到页框队列的尾部

相关的数组类的方法：相关的数组类的方法：push()、shift()、indexOf()

splice():指定数组下标删除某元素

3.4.2、OPT()函数算法流程图：



3.4.2、伪代码如下：

/\*\*\*\*\*\*\*\*opt算法\*\*\*\*\*\*\*/

function OPT(k, optArr, pageArr) {

// var pageArr = [2,3,2,1,5,2,4,5,3,2,5,2]; //页地址流数组

let num5; //当缺页时判断是否可以置换了的标志

for(let i = 0; i < pageArr.length; i++) {

if(optArr.length < k) { //页框初始化前

if((optArr.indexOf(pageArr[i])) != -1) { //有重复，则跳出该循环

continue;

} else { //无重复，则新页号直接插入到队列尾

optArr.push(pageArr[i]);

}

} else { //页框初始化后

if((optArr.indexOf(pageArr[i])) != -1) { //有重复，则也是直接跳出该循环

continue;

} else { //无重复，

num4++; //缺页数加1

num5 = 0;

//从i开始往后循环数组1,两种情况下跳出循环

for(let j = i + 1; (num5 < k - 1) && (j < pageArr.length); j++) { //注意这里的循环跳出条件，贼坑，看我下面注释掉的代码，应该是要取反的，因为是满足才进入，不满足跳出

if((optArr.indexOf(pageArr[j])) != -1) { //pageArr后面有跟pageArr数组重复时，num2++，并将该元素调到队列尾

num5++;

optArr.splice(optArr.indexOf(pageArr[j]), 1);

optArr.push(pageArr[j]);

} else {

continue;

}

}

//跳出循环后，删除队头元素，将新元素加到队列尾

optArr.shift();

optArr.push(pageArr[i]);

}

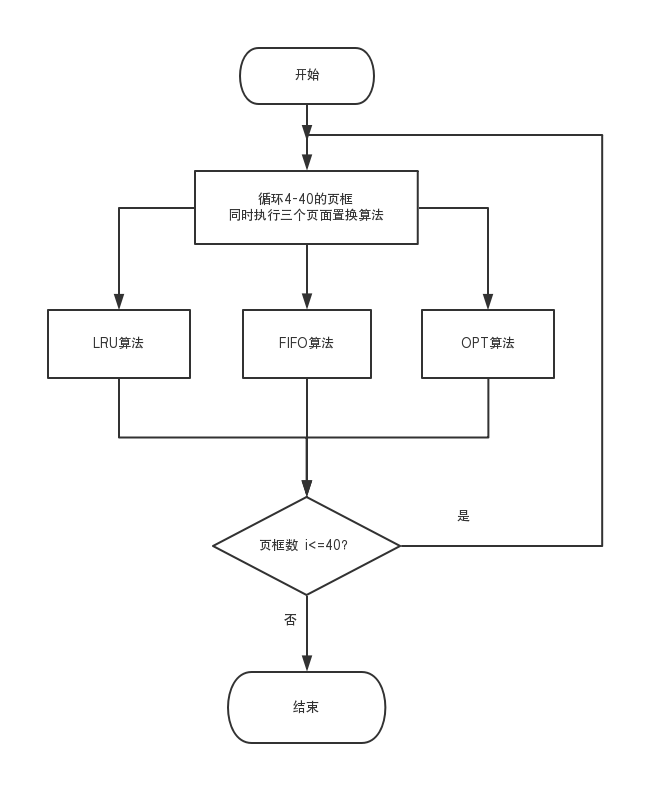
}

}

}

3.5、function replace( ) { } : simulation.js程序的主函数，调用FIFO()函数、LRU()函数以及OPT()函数，实现从4-40页框的三大页面置换算法——FIFO算法、LRU算法以及OPT算法的页面置换，并将命中率以表格的形式打印出来

3.5.1、函数执行流程图：



3.5.2、伪代码如下：

function replace() {

for(let k = 4; k <= 40; k++) {

var fifoArr = [], lruArr = [], optArr = []; //页框数组

var num2 = 0, num3 = 0, num4 = 0; //缺页次数

/\*\*\*\*\*\*fifo\*\*\*\*\*\*/

FIFO(k, fifoArr, pageArr);

/\*\*\*\*\*lru\*\*\*\*\*\*/

LRU(k, fifoArr, pageArr);

/\*\*\*\*\*\*\*\*opt算法\*\*\*\*\*\*\*/

OPT(k, fifoArr, pageArr);

/\*\*\*动态添加命中率数据到表格中\*\*\*/

num2 = 1 - (num2 / 400);

num3 = 1 - (num3 / 400);

num4 = 1 - (num4 / 400);

num2 = num2.toFixed(4);

num3 = num3.toFixed(4);

num4 = num4.toFixed(4);

var $txt2 = $(".content").html();

$(".content").html(`

${$txt2}

<tr>

<td>${k}</td>

<td>${num4}</td>

<td>${num2}</td>

<td>${num3}</td>

</tr>

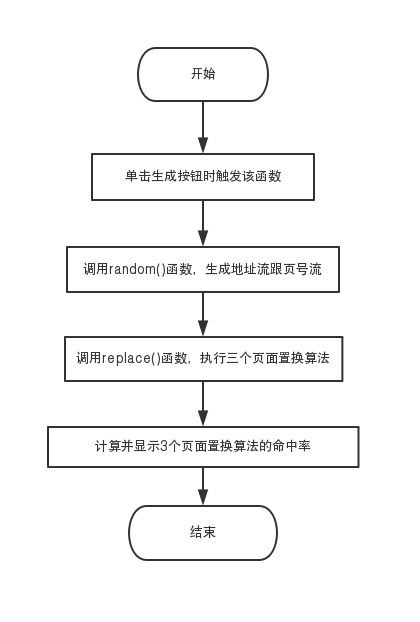
`);

}

}

3.6、$(".btn").click( ) : 为class = “btn”的按钮绑定一个单击响应函数，当单击按钮时调用random()函数以及replace()函数。单击时随机生成400个范围在0-399的地址流，并将其转换为400个范围在0-39的页号流，与此同时将地址流与页号流显示在图形用户界面的方框中；然后调用replace()函数，实现三大页面置换算法——FIFO算法、LRU算法以及OPT算法，并分别计算它们的命中率，与此同时将4-40个页框对应的FIFO算法、LRU算法以及OPT算法的命中率显示在图形用户界面上；最后总结三个算法的规律与特点，将总结的文字输出显示在图形用户界面上。

3.6.1、函数执行流程图：



3.6.2、伪代码如下：

$(".btn").click(function () {

var addressArr, pageArr; //定义变量

random(); //生成随机数函数

$(".addressTxt").attr("placeholder", addressArr); //地址流显示到屏幕上

$(".pageNumTxt").attr("placeholder", pageArr); //页号流显示到屏幕上

$(".a").attr("placeholder", pageArr);

//每次点击时又重新赋值为刚开始时的html代码

$(".content").html("$txt1"); //jq选择器跟html()方法是的html代码都得加"",被坑了好几次，而且好像也没有什么错误提醒？

replace(); //页面置换算法函数

$(".summary").html(`

<h4>

文字总结内容

</h4>

`);

});

7、$(document).ready( ) : 页面加载函数，将单击响应函数$(".btn").click(function ( ) { }放在其中，因为只有但页面文档节点对象DOM加载完成后才能对页面节点对象进行操作。

var $txt1 = $(".content").html(); //单击函数开始前保存刚开始的$(".content")节点的html代码

$(document).ready(function() {

$(".btn").click(function () );

});

### 调试分析

内容包括：

1. 调试过程中遇到的问题是如何解决的以及对设计与实现的讨论和分析；

①本程序的编写设计思路是先实现底层的逻辑算法，即第一步是先实现随机产生地址 流，并将其转换为页号流，并对数据的正确性进行验证与调试；第二步是先单独地实现 3个页面置换算法——FIFO、LRU、OPT算法的逻辑代码部分并逐个对其进行测试与维 护；第三步便模拟三个置换算法的置换过程，并计算它们对应的命中率；第四步便是将 模拟的页面置换过程以及输入输出等数据显示到图形用户界面上，并验证调试正确性； 第五步是实现页面的交互动作以及页面的样式风格；最后再进行整体的多次整合和调试

②在第一步时遇到很多困难，比如前期对三个页面置换算法的不熟悉，核心思路掌握 得不牢靠，导致实现起来的逻辑思路比较乱，后来重新认真地看了各个算法的定义以及 对应着各个算法走一遍流程.

③后来在实现置换算法时一个致命的错误便是一开始一直没有考虑到页框还未初始化 的情况，走了几遍算法以及反复调试后最后才意识到这点，页框数组在初始化前后对页 号的操作方法完全不同，不应该只是单单地考虑核心算法部分而忽略了基础的部分，细 节是十分重要的，有时甚至直接影响了整个程序.

④三个页面置换算法中最难实现、错误率最多的无疑是OPT最佳置换算法，它作为最 佳置换算法，便有它的优势与难点。我用的方法是当遇到缺页且无重复页号时从当前的 地址流数组开始，往后循环直到找到那个在接下来的置换过程中最晚用到的甚至可能都 用不到的。思路一开始虽然可行，但在调试的过程中我发现有一种情况下回一直报错， 就是当页框数组中有两个页号的情况条件是完全一致的情况下，即算法也无法确定是两 个哪个先被置换出来，后来调试了多次，拿了多组数据进行反复测试后才找到这个错误， 最后从二者中选一来进行置换.

⑤当在进行实现图形用户界面时，当我把单击响应函数绑定到按钮上，以该按钮来进 行交互，但当我反复测试点击按钮都没有任何反应，便想可能是逻辑层出现了问题，所 以一直调试逻辑层，最后在反复否测试中发现，当我点击按钮时，页面的DOM节点还 没加载完成，即相当于我还没定义变量前便开始使用该变量，结果当然会报错。所以最 后发现是这个问题后，我便将按钮的单击响应函数放到了jQuery的页面加载函数： $(document).ready( ) 中执行，便解决了这个问题.

⑥在调试的过程中还遇到了许多大大小小的问题，有对JavaScript语言语法本身的不熟 悉，也有对算法逻辑的错误理解，最后在自己的努力与思考之下成功地实现了该程序， 收获了很多，更加深入地理解了软件开发的流程以及实现过程.

1. 算法的时间复杂性(包括基本操作和其他算法的时间复杂性的分析)和改进设想；

①FIFO算法的基本操作以及其复杂性分析：

建立两个数组，一个表示页地址流数组，一个表示页框数组；

循环页地址流数组里的元素，将页框数组当成队列来使用，达到先进先出的效果；

相关的数组类的方法：push()、shift()、indexOf()

因此FIFO算法的时间复杂性为：O()

②LRU算法的基本操作以及其复杂性分析：

建立两个数组，并且将第二个数组当队列使用，即将每次进来的从后面进入，删除 的从前面删除，并且关键是将不缺页时重新访问的那个元素调出来重新插到最后一个

相关的数组类的方法：splice():指定数组下标删除某元素

因此LRU算法的时间复杂性为：O()

③OPT算法的基本操作以及其复杂性分析：

建立两个数组，并且将第二个数组当队列使用，即将每次进来的从后面进入，删除的从 前面删除；并且关键是当缺页且页框队列里没有相同的页号时，从页号流当前位置出发 往后循环页号流数组，找到接下来最不可能使用到的页号，将其从页框数组中置换出来， 并将新的页号插入到页框队列的尾部

相关的数组类的方法：相关的数组类的方法：push()、shift()、indexOf()

splice():指定数组下标删除某元素

因此OPT算法的时间复杂性为：O()

④改进设想：

结合其他的页面置换算法，除了计算其命中率一以外，我觉得还可以研究几个的页面置 换规律，找到它们之间的异同，这样也更有利于理解内存与进程之间的调度关系.

1. 设计过程的经验和体会；

①本程序的编写设计思路是先实现底层的逻辑算法，即第一步是先实现随机产生地址 流，并将其转换为页号流，并对数据的正确性进行验证与调试；第二步是先单独地实现 3个页面置换算法——FIFO、LRU、OPT算法的逻辑代码部分并逐个对其进行测试与维 护；第三步便模拟三个置换算法的置换过程，并计算它们对应的命中率；第四步便是将 模拟的页面置换过程以及输入输出等数据显示到图形用户界面上，并验证调试正确性； 第五步是实现页面的交互动作以及页面的样式风格；最后再进行整体的多次整合和调试

②在调试的过程中还遇到了许多大大小小的问题，有对JavaScript语言语法本身的不熟 悉，也有对算法逻辑的错误理解，最后在自己的努力与思考之下成功地实现了该程序， 收获了很多，更加深入地理解了软件开发的流程以及实现过程.

③所以编程还是应该多多上机实现以下，毕竟机器设备相对我们而言在代码的语法容 错性上还是更加严谨的。不过在这基础上更应该打好基础，有更加丰富的知识才能编写 出更加完善的程序.

1. 实现过程中出现的主要问题及解决方法。

主要问题有几个方面，一个是自身本身对语言的语法知识掌握不牢靠二导致，二是对算 法的思维理解不透彻，三是在实现程序时遇到的逻辑层面的错误：

①本程序的编写设计思路是先实现底层的逻辑算法，即第一步是先实现随机产生地址 流，并将其转换为页号流，并对数据的正确性进行验证与调试；第二步是先单独地实现 3个页面置换算法——FIFO、LRU、OPT算法的逻辑代码部分并逐个对其进行测试与维 护；第三步便模拟三个置换算法的置换过程，并计算它们对应的命中率；第四步便是将 模拟的页面置换过程以及输入输出等数据显示到图形用户界面上，并验证调试正确性； 第五步是实现页面的交互动作以及页面的样式风格；最后再进行整体的多次整合和调试

②在第一步时遇到很多困难，比如前期对三个页面置换算法的不熟悉，核心思路掌握 得不牢靠，导致实现起来的逻辑思路比较乱，后来重新认真地看了各个算法的定义以及 对应着各个算法走一遍流程.

③后来在实现置换算法时一个致命的错误便是一开始一直没有考虑到页框还未初始化 的情况，走了几遍算法以及反复调试后最后才意识到这点，页框数组在初始化前后对页 号的操作方法完全不同，不应该只是单单地考虑核心算法部分而忽略了基础的部分，细 节是十分重要的，有时甚至直接影响了整个程序.

④三个页面置换算法中最难实现、错误率最多的无疑是OPT最佳置换算法，它作为最 佳置换算法，便有它的优势与难点。我用的方法是当遇到缺页且无重复页号时从当前的 地址流数组开始，往后循环直到找到那个在接下来的置换过程中最晚用到的甚至可能都 用不到的。思路一开始虽然可行，但在调试的过程中我发现有一种情况下回一直报错， 就是当页框数组中有两个页号的情况条件是完全一致的情况下，即算法也无法确定是两 个哪个先被置换出来，后来调试了多次，拿了多组数据进行反复测试后才找到这个错误， 最后从二者中选一来进行置换.

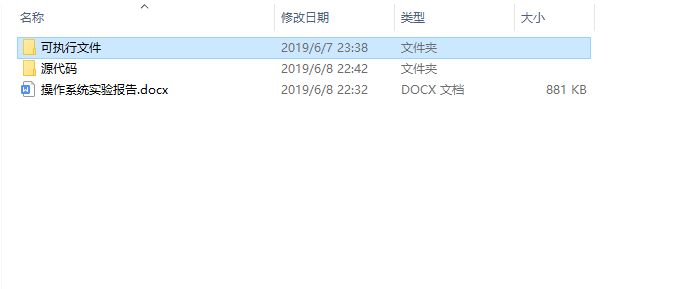
⑤当在进行实现图形用户界面时，当我把单击响应函数绑定到按钮上，以该按钮来进 行交互，但当我反复测试点击按钮都没有任何反应，便想可能是逻辑层出现了问题，所 以一直调试逻辑层，最后在反复否测试中发现，当我点击按钮时，页面的DOM节点还 没加载完成，即相当于我还没定义变量前便开始使用该变量，结果当然会报错。所以最 后发现是这个问题后，我便将按钮的单击响应函数放到了jQuery的页面加载函数： $(document).ready( ) 中执行，便解决了这个问题.

⑥在调试的过程中还遇到了许多大大小小的问题，有对JavaScript语言语法本身的不熟 悉，也有对算法逻辑的错误理解，最后在自己的努力与思考之下成功地实现了该程序， 收获了很多，更加深入地理解了软件开发的流程以及实现过程.

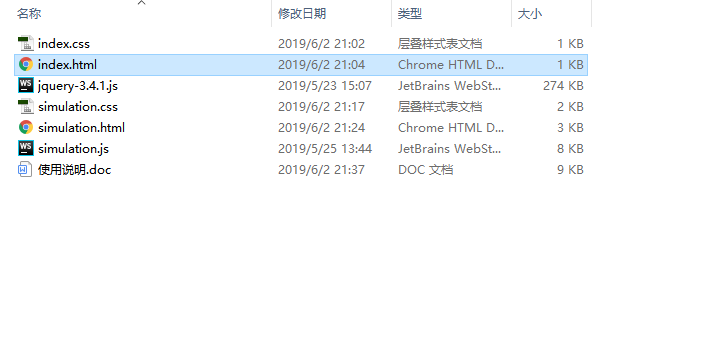
### 用户使用说明

说明如何使用你编写的程序，详细列出每一步的操作步骤。

①打开可执行文件：



②双击用浏览器打开“index.html”文件：**（注意是“index.html”文件）**



③点击按钮进入模拟系统：



④点击“一键生成”按钮，生成地址流与页号流：



⑤页面上方为随机生成的400个地址流以及转换得到的页号流：



⑥滑动页面可以查看三大页面置换算法在0-40页框分别的命中率：



⑦页面最底部下面的文字是页面置换算法规律的文字总结：



1. 测试与运行结果

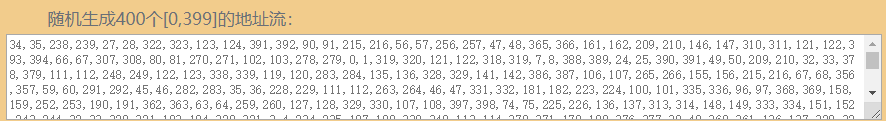
列出你的测试结果和运行情况（即运行时的关键画面），包括输入和输出。

①运行时的界面：

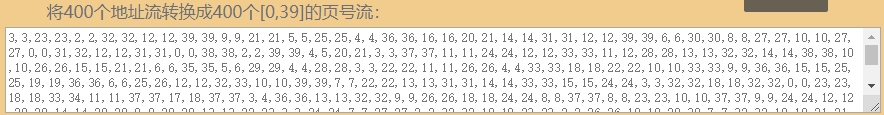




②数据输入——地址流输入：

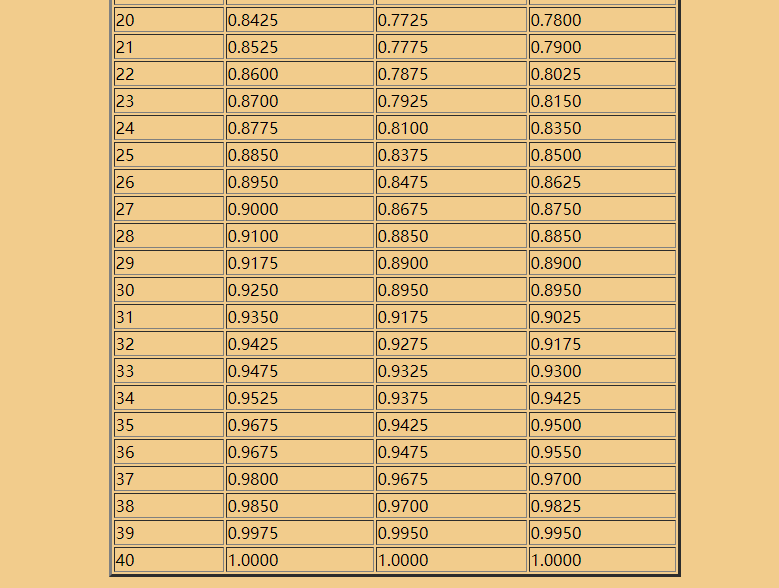


③数据输入——页号流输入：



④数据输出——三大页面置换算法4-40页框的命中率显示：





⑤数据输出——结合置换规律输出总结文字：

