**实验四：页面置换**

**安全1601 16281221 邓子轩**

1. **需求分析**

设计和实现最佳置换算法、先进先出换算法、最近最久未使用置换算法、页面缓冲置换算法、改进Clock置换算法；通过页面访问序列随机发生器实现对上述算法的测试及性能比较。

1. **概要设计**

工作集

多数程序都显示出高度的局部性，也就是说，在一个时间段内，一组页面被反复引用。这组被反复引用的页面随着时间的推移，其成员也会发生变化。有时这种变化是剧烈的，有时这种变化则是渐进的。我们把这组页面的集合称为工作集。

缺页率

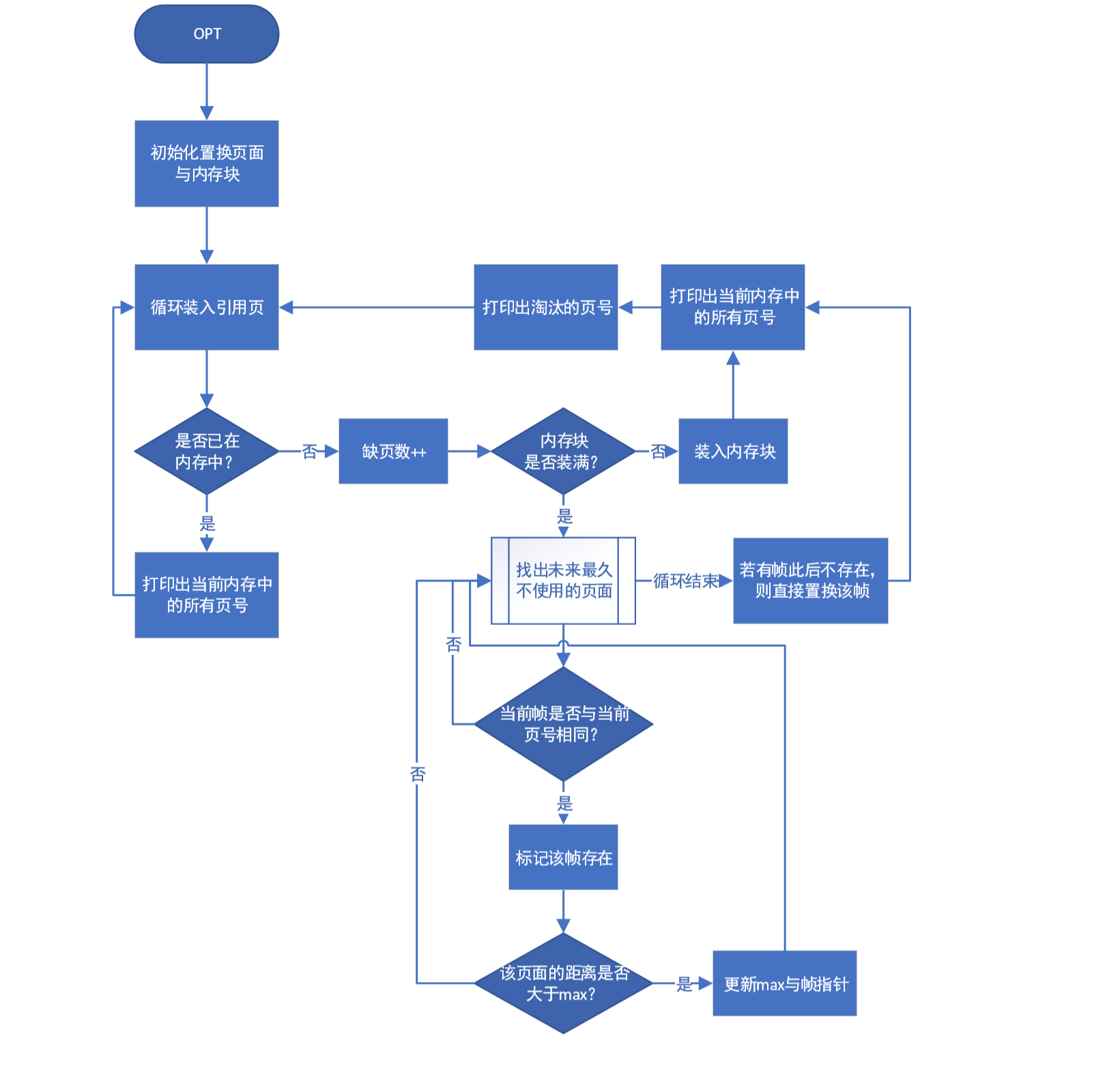
缺页率 = 缺页中断次数/页面访问次数

**1.最佳置换算法**

**基本思想：**

选择缓冲区内永不使用或是最长时间内不再被访问（即距现在最长时间才会被访问）的页面淘汰出内存。

**流程图：**



**评价：**

最佳置换算法是一个理想的化的算法，具有最好的性能(对于固定分配页面方式，本方法可以保证获得最低的缺页率)。但是人们目前还无法预测，一个进程在内存的若干个页面中，哪一个页面是未来最长时间内不再被访问的，因而该算法是无法实现的，但是可以利用该算法取评价其他的算法。故主要用于算法评价参照。

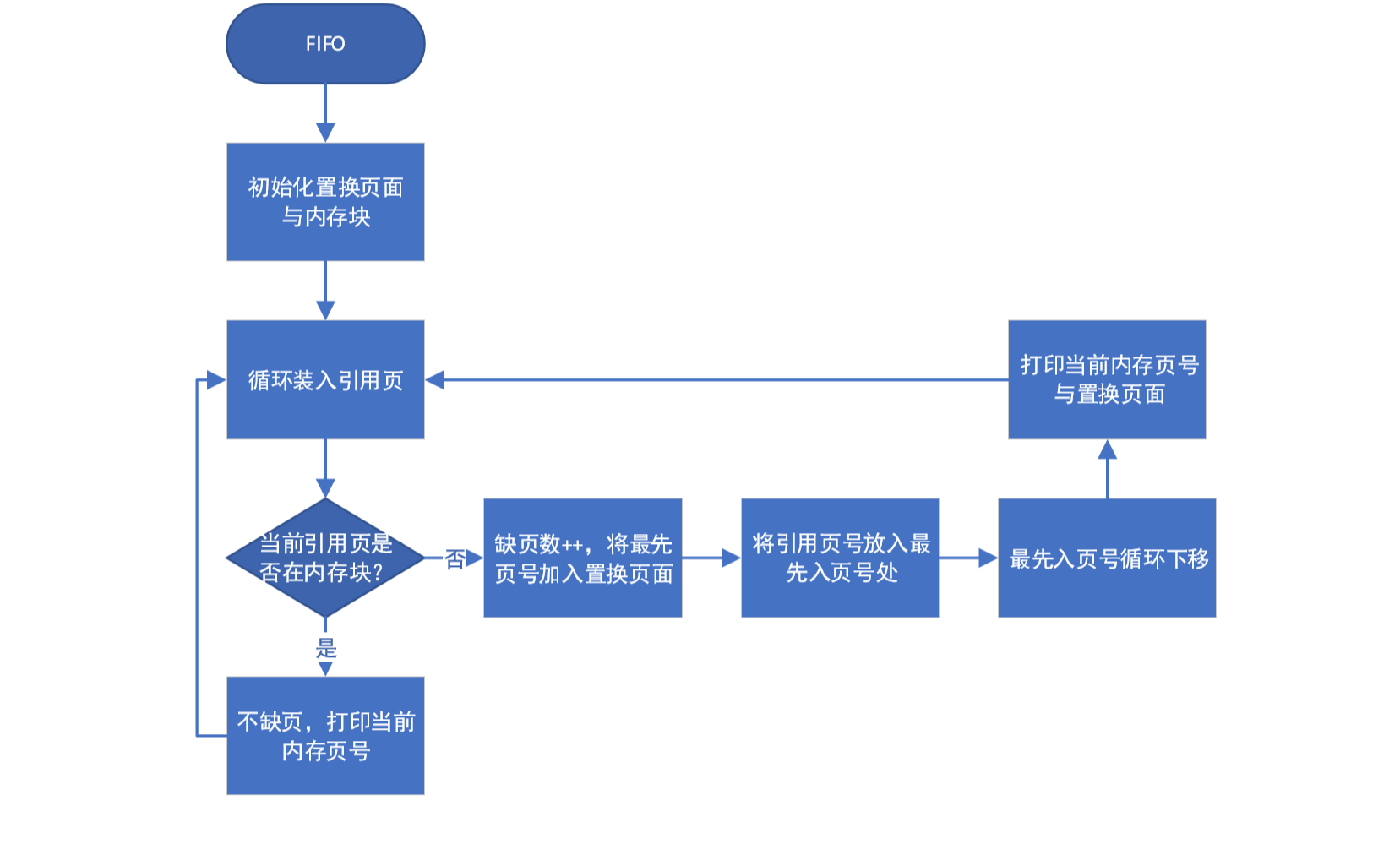
**2.先进先出置换算法**

**基本思想：**

选择最先进入内存即在内存驻留时间最久的页面换出到外存

进程已调入内存的页面按照先后次序链接成一个队列，并设置替换指针以指向最老页面

**流程图：**



**评价：**

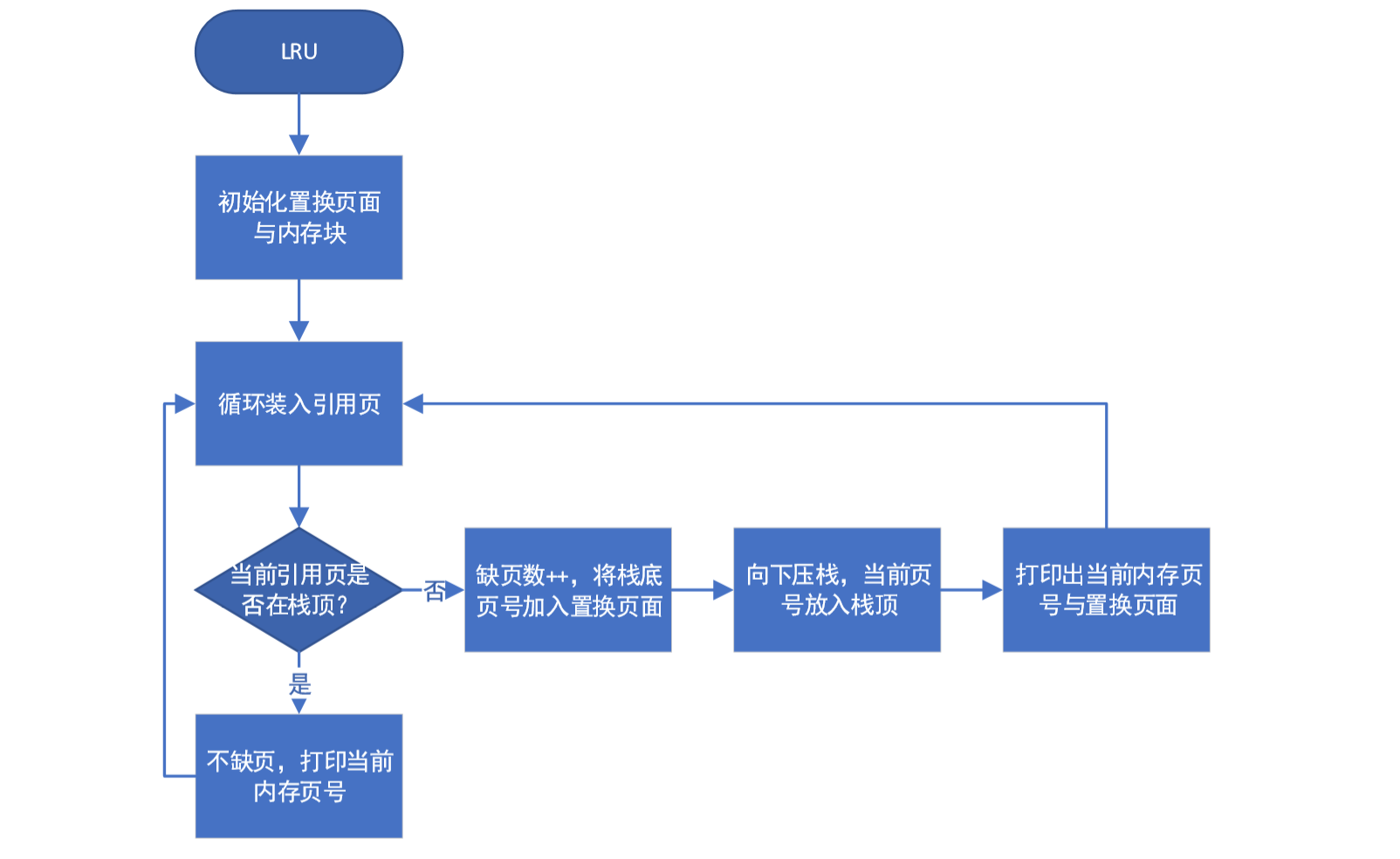
先进先出算法简单直观，但是不符合进程实际的运行规律，性能较差，故实际应用极少。

**3.最近最久未使用置换算法**

**基本思想：**

以”最近的过去”作为”最近的将来”的近似，选择最近一段时间最长时间未被访问的页面淘汰出内存。

**流程图：**



**评价：**

LRU算法适用于各种类型的程序，性能较好，但是需要较多的硬件支持。

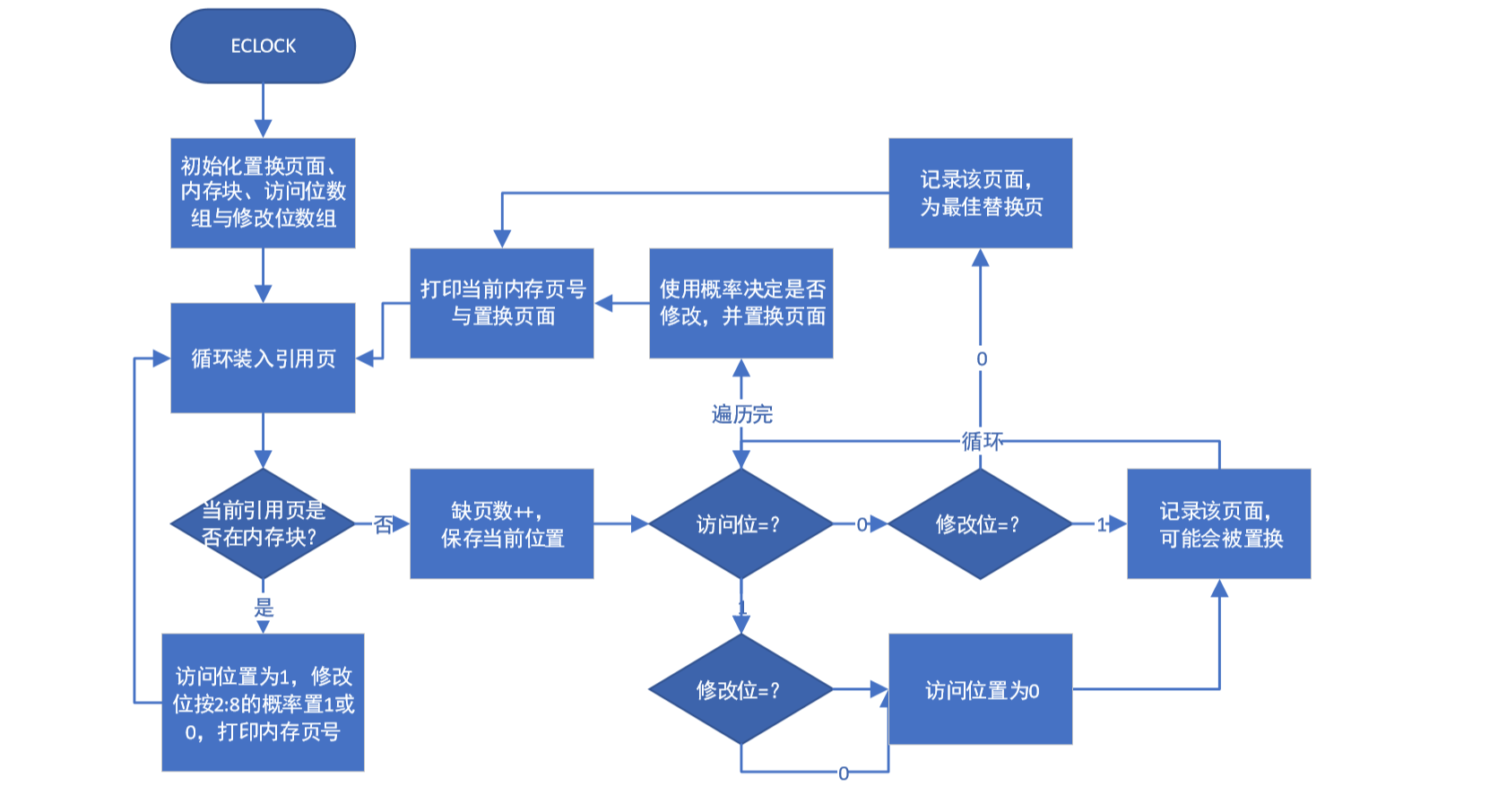
**4.改进型Clock置换法**

**基本思想：**

① 从查寻指针当前位置起扫描内存分页循环队列，选择A=0且M=0的第一个页面淘汰；若未找到，转②

② 开始第二轮扫描，选择A=0且M=1的第一个页面淘汰，同时将经过的所有页面访问位置0；若不能找到，转①

**流程图：**



**评价：**

与简单Clock算法相比，可减少磁盘的I/O操作次数，但淘汰页的选择可能经历多次扫描，故实现算法自身的开销增大。

1. **详细设计，程序代码及运行截图**
2. **最佳置换算法**

**//最佳置换算法**

**void Replace::Opt(void)**

**{**

**int i, j, k, l, next;**

**int max = 0, out = 0;**

**bool exist = false;**

**InitSpace("OPT");**

**//循环装入引用页**

**for (k = 0, j = l = 0; k < PageNumber; k++) {**

**next = ReferencePage[k];**

**//如果引用页已在实存中,报告实存页号**

**for (i = 0; i < FrameNumber; i++)**

**if (next == PageFrames[i])**

**break;**

**if (i < FrameNumber) {**

**for (i = 0; i < FrameNumber; i++)**

**if (PageFrames[i] >= 0)**

**cout << PageFrames[i] << " ";**

**cout << endl;**

**continue;// 继续引用下一页**

**}**

**//引用页不在实存中,缺页数加1**

**FaultNumber++;**

**if (PageFrames[j] == -1) {//内存块未装满时**

**EliminatePage[l] = PageFrames[j];//最先入页号记入淘汰页数组**

**PageFrames[j] = next;//引用页号放最先入页号处**

**j = (j + 1) % FrameNumber;//最先入页号循环下移**

**}**

**else { //找到未来最久不使用的页面**

**for (int m = 0; m < FrameNumber; m++) {**

**exist = false;**

**for (int n = k; n < PageNumber; n++) {**

**if (PageFrames[m] == ReferencePage[n]) {**

**exist = true;**

**if (n > max) {**

**max = n;**

**out = m;**

**}**

**break;**

**}**

**}**

**if (!exist) { //若此后都不用该页面，则直接置换**

**out = m;**

**break;**

**}**

**}**

**max = 0;**

**EliminatePage[l] = PageFrames[out];**

**PageFrames[out] = next;//引用页号放入置换位置**

**}**

**//报告当前实存页号和淘汰页号**

**for (i = 0; i < FrameNumber; i++)**

**if (PageFrames[i] >= 0)**

**cout << PageFrames[i] << " ";**

**if (EliminatePage[l] >= 0)**

**cout << "->" << EliminatePage[l++] << endl;**

**else**

**cout << endl;**

**}**

**//分析统计选择的算法对于当前引用的页面走向的性能**

**Report();**

**}**

1. **先进先出置换算法**

**//最近最久未用置换算法**

**void Replace::Lru(void)**

**{**

**int i, j, k, l, next;**

**InitSpace("LRU");**

**//循环装入引用页**

**for (k = 0, l = 0; k < PageNumber; k++) {**

**next = ReferencePage[k];**

**//检测引用页当前是否已在实存**

**for (i = 0; i < FrameNumber; i++) {**

**if (next == PageFrames[i]) {**

**//引用页已在实存将其调整到页记录栈顶**

**for (j = i; j > 0; j--)**

**PageFrames[j] = PageFrames[j - 1];**

**PageFrames[0] = next;**

**break;**

**}**

**}**

**if (PageFrames[0] == next) {**

**//如果引用页已放栈顶,则为不缺页,报告当前内存页号**

**for (j = 0; j < FrameNumber; j++)**

**if (PageFrames[j] >= 0)**

**cout << PageFrames[j] << " ";**

**cout << endl;**

**continue;//继续装入下一页**

**}**

**else**

**//如果引用页还未放栈顶,则为缺页,缺页数加1**

**FaultNumber++;**

**//栈底页号记入淘汰页数组中**

**EliminatePage[l] = PageFrames[FrameNumber - 1];**

**//向下压栈**

**for (j = FrameNumber - 1; j > 0; j--)**

**PageFrames[j] = PageFrames[j - 1];**

**PageFrames[0] = next;//引用页放栈顶**

**//报告当前实存中页号**

**for (j = 0; j < FrameNumber; j++)**

**if (PageFrames[j] >= 0)**

**cout << PageFrames[j] << " ";**

**//报告当前淘汰的页号**

**if (EliminatePage[l] >= 0)**

**cout << "->" << EliminatePage[l++] << endl;**

**else**

**cout << endl;**

**}**

**//分析统计选择的算法对于当前引用的页面走向的性能**

**Report();**

**}**

1. **最近最久未使用置换算法**

**//先进先出置换算法**

**void Replace::Fifo(void)**

**{**

**int i, j, k, l, next;**

**InitSpace("FIFO");**

**//循环装入引用页**

**for (k = 0, j = l = 0; k < PageNumber; k++) {**

**next = ReferencePage[k];**

**//如果引用页已在实存中,报告实存页号**

**for (i = 0; i < FrameNumber; i++)**

**if (next == PageFrames[i])**

**break;**

**if (i < FrameNumber) {**

**for (i = 0; i < FrameNumber; i++)**

**if (PageFrames[i] >= 0)**

**cout << PageFrames[i] << " ";**

**cout << endl;**

**continue;// 继续引用下一页**

**}**

**//引用页不在实存中,缺页数加1**

**FaultNumber++;**

**EliminatePage[l] = PageFrames[j];//最先入页号记入淘汰页数组**

**PageFrames[j] = next;//引用页号放最先入页号处**

**j = (j + 1) % FrameNumber;//最先入页号循环下移**

**//报告当前实存页号和淘汰页号**

**for (i = 0; i < FrameNumber; i++)**

**if (PageFrames[i] >= 0)**

**cout << PageFrames[i] << " ";**

**if (EliminatePage[l] >= 0)**

**cout << "->" << EliminatePage[l++] << endl;**

**else**

**cout << endl;**

**}**

**//分析统计选择的算法对于当前引用的页面走向的性能**

**Report();**

**}**

1. **改进型Clock置换算法**

**//改进的时钟置换算法**

**void Replace::Eclock(void)**

**{**

**int i, j, k, l, next;**

**InitSpace("Eclock");**

**//访问位数组**

**bool\* refbit = new bool[sizeof(bool) \* FrameNumber];**

**bool\* modbit = new bool[sizeof(bool) \* FrameNumber];**

**for (i = 0; i < FrameNumber; i++) {//将所有访问位和修改位初始化为0**

**refbit[i] = false;**

**modbit[i] = false;**

**}**

**//循环装入引用页**

**for (k = 0, j = l = 0; k < PageNumber; k++) {**

**next = ReferencePage[k];**

**//如果引用页已在实存中,报告实存页号**

**for (i = 0; i < FrameNumber; i++)**

**if (next == PageFrames[i]) {**

**refbit[i] = true; //当引用页被访问时，访问位置为1**

**if (rand() % 100 < 20) {**

**modbit[i] = 1; //20%概率修改为1**

**break;**

**}**

**else {**

**modbit[i] = 0; //80%概率修改为0**

**break;**

**}**

**}**

**if (i < FrameNumber) {**

**for (i = 0; i < FrameNumber; i++)**

**if (PageFrames[i] >= 0)**

**cout << PageFrames[i] << " ";**

**cout << endl;**

**}**

**else {**

**//引用页不在实存中,缺页数加1**

**FaultNumber++;**

**i = j; //保存当前位置，检验是否将队列遍历完毕**

**int replacePageNum = -1; //保存会被置换的页号**

**do {**

**if (refbit[j]) {//如果当前页被访问过，给予二次机会**

**if (!modbit[j])**

**replacePageNum = j; //如果未被修改，可能被替换**

**refbit[j] = false; //把访问位置为0**

**}**

**else if (modbit[j])**

**replacePageNum = j; //未被访问但被修改了，可能被替换**

**else { //未被访问也未被修改，即找到了最佳替换页**

**replacePageNum = j;**

**break;//立刻跳出**

**}**

**j = (j + 1) % FrameNumber;//再从下一帧开始找未访问页**

**} while (i != j);**

**if (-1 != replacePageNum) { //替换选中的被置换页**

**EliminatePage[l] = PageFrames[replacePageNum];**

**PageFrames[replacePageNum] = next;**

**refbit[replacePageNum] = true;**

**modbit[replacePageNum] = rand() % 2; //用随机数模拟这一页是否会被修改**

**}**

**else { //队列完全被遍历1遍，且一直没有找到可选置换页面**

**EliminatePage[l] = PageFrames[j];**

**PageFrames[j] = next;**

**refbit[j] = true;**

**modbit[j] = rand() % 2; //用随机数模拟这一页是否会被修改**

**}**

**//报告当前实存页号和淘汰页号**

**for (i = 0; i < FrameNumber; i++)**

**if (PageFrames[i] >= 0)**

**cout << PageFrames[i] << " ";**

**if (EliminatePage[l] >= 0)**

**cout << "->" << EliminatePage[l++] << endl;**

**else**

**cout << endl;**

**}**

**for (i = 0; i < FrameNumber; i++)**

**cout << "(" << refbit[i] << ',' << modbit[i] << ") ";**

**cout << endl;**

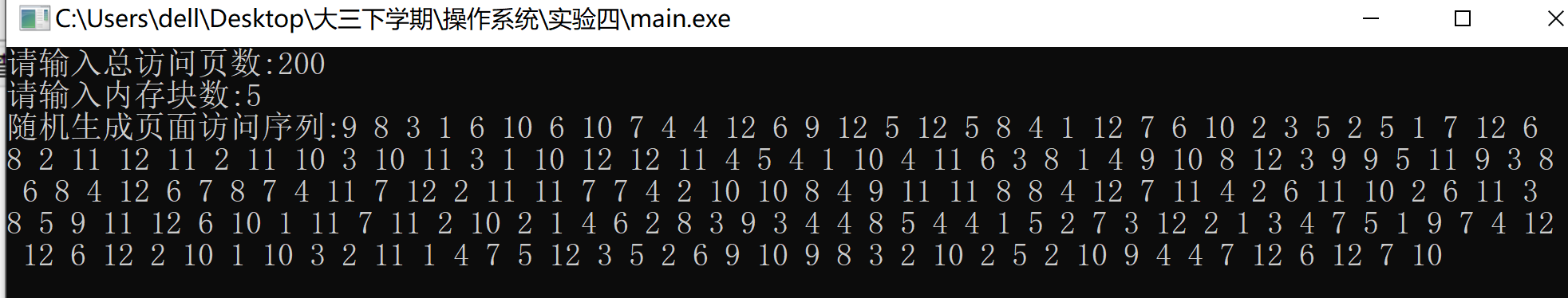
**}**

**//分析统计选择的算法对于当前引用的页面走向的性能**

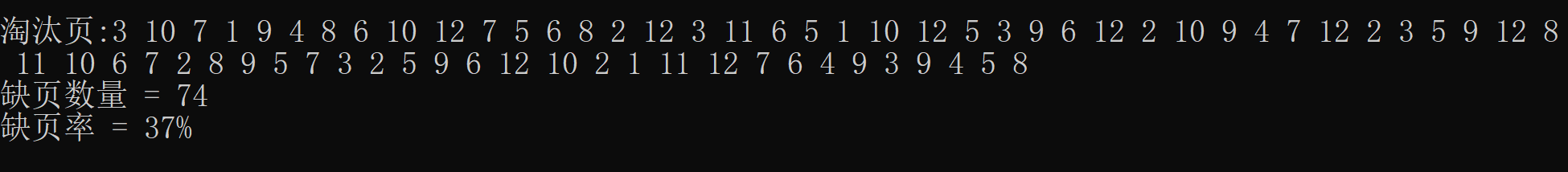
**Report();**

**}**

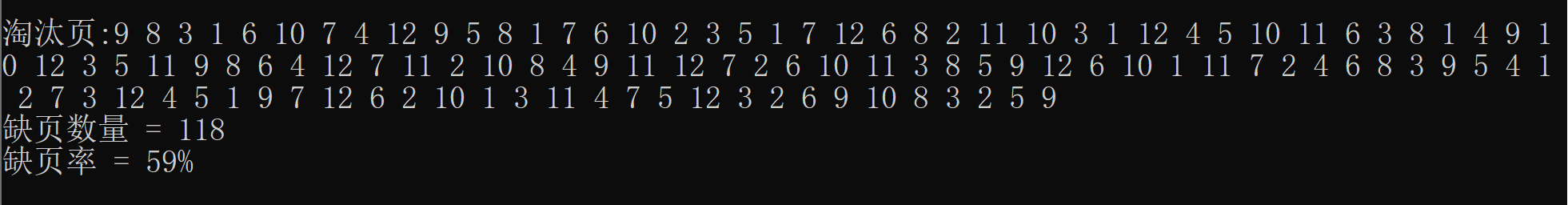
截图：



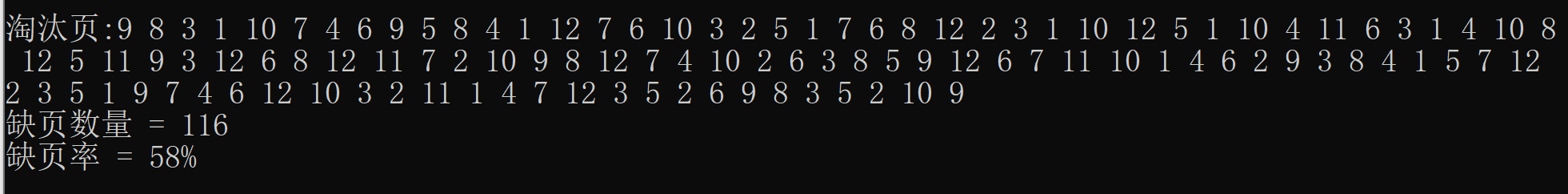
OPT：



FIFO：



LRU：



改进的Clock：

