**操作系统实验**

**实验报告**

题目：[实验五 虚拟内存页面置换算法]

**专业：[软件工程]**

**年级：[2017级]**

**姓名：[陈思翔]**

**学号：[1725121003]**

目录

[一、需求分析 4](#_Toc27927114)

[1、程序设计的任务和目的 4](#_Toc27927115)

[2、输入的形式和输入值的范围 4](#_Toc27927116)

[3、输出的形式 4](#_Toc27927117)

[4、程序所能达到的功能 4](#_Toc27927118)

[5、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 4](#_Toc27927119)

[正确用例 4](#_Toc27927120)

[错误用例（输入中带有0页面，该页面号设置为状态量） 5](#_Toc27927121)

[二、概要设计 6](#_Toc27927122)

[1、抽象数据类型的定义 6](#_Toc27927123)

[2、主程序的流程 6](#_Toc27927124)

[3、各程序模块之间的层次(调用)关系 7](#_Toc27927125)

[三、详细设计 8](#_Toc27927126)

[1、先进先出(FIFO)页面置换算法 8](#_Toc27927127)

[2、最佳(OPI)页面置换算法 9](#_Toc27927128)

[3、最近最久未使用(LRU)页面置换算法 10](#_Toc27927129)

[四、调试分析 12](#_Toc27927130)

[调试过程中遇到的问题以及解决方法，设计与实现的回顾讨论和分析 12](#_Toc27927131)

[算法的性能分析(包括基本操作和其它算法的时间复杂度和空间复杂度的分析)及其改进设想 12](#_Toc27927132)

[性能分析 12](#_Toc27927133)

[改进设想 12](#_Toc27927134)

[五、用户使用说明 12](#_Toc27927135)

[使用说明 12](#_Toc27927136)

[六、测试结果 13](#_Toc27927137)

[测试结果，包括输入和输出 13](#_Toc27927138)

[七、附录 15](#_Toc27927139)

[带注释的源程序 15](#_Toc27927140)

# 一、需求分析

## 1、程序设计的任务和目的

通过这次实验，加深对虚拟内存页面置换概念的理解，进一步掌握先进先出FIFO、最佳置换OPI和最近最久未使用LRU页面置换算法的实现方法。

## 2、输入的形式和输入值的范围

最小物理块数m，页面个数n，页面访问序列P1, … ,Pn，算法选择1-FIFO，2-OPI，3-LRU。

## 3、输出的形式

每种算法的缺页次数和缺页率。

## 4、程序所能达到的功能

设计程序模拟先进先出FIFO、最佳置换OPI和最近最久未使用LRU页面置换算法的工作过程。假设内存中分配给每个进程的最小物理块数为m，在进程运行过程中要访问的页面个数为n，页面访问序列为P1, … ,Pn，分别利用不同的页面置换算法调度进程的页面访问序列，给出页面访问序列的置换过程，计算每种算法缺页次数和缺页率。

## 5、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

### 正确用例

#### 输入

请输入最小物理块数MinBlockNum：4  
请输入页面个数PageNum：12  
请输入页面1编号PageOrder[1]：4  
请输入页面2编号PageOrder[2]：3  
请输入页面3编号PageOrder[3]：2  
请输入页面4编号PageOrder[4]：1  
请输入页面5编号PageOrder[5]：4  
请输入页面6编号PageOrder[6]：3  
请输入页面7编号PageOrder[7]：5  
请输入页面8编号PageOrder[8]：4  
请输入页面9编号PageOrder[9]：3  
请输入页面10编号PageOrder[10]：2  
请输入页面11编号PageOrder[11]：1  
请输入页面12编号PageOrder[12]：5  
  
请选择想要先使用的算法（ 1-先进先出(FIFO)页面置换算法，2-最佳(OPI)页面置换算法，3-最近最久未使用(LRU)页面置换算法 ）：1

#### 输出

请选择想要先使用的算法（ 1-先进先出(FIFO)页面置换算法，2-最佳(OPI)页面置换算法，3-最近最久未使用(LRU)页面置换算法 ）：1  
  
您选择的是1-先进先出(FIFO)页面置换算法  
页面7换掉第1物理块中的页面  
页面8换掉第2物理块中的页面  
页面9换掉第3物理块中的页面  
页面10换掉第4物理块中的页面  
页面11换掉第1物理块中的页面  
页面12换掉第2物理块中的页面  
  
页面置换算法模拟过程如下：  
 Page 1 Page 2 Page 3 Page 4 Page 5 Page 6 Page 7 Page 8 Page 9 Page10 Page11 Page12  
BlockNum 1 4 4 4 4 4 4 5 5 5 5 1 1  
BlockNum 2 3 3 3 3 3 3 4 4 4 4 5  
BlockNum 3 2 2 2 2 2 2 3 3 3 3  
BlockNum 4 1 1 1 1 1 1 2 2 2  
页面置换算法缺页次数为：10  
页面置换算法缺页率为：83%

### 错误用例（输入中带有0页面，该页面号设置为状态量）

#### 输入

请输入最小物理块数MinBlockNum：4  
请输入页面个数PageNum：12  
请输入页面1编号PageOrder[1]：4  
请输入页面2编号PageOrder[2]：3  
请输入页面3编号PageOrder[3]：2  
请输入页面4编号PageOrder[4]：1  
请输入页面5编号PageOrder[5]：4  
请输入页面6编号PageOrder[6]：0  
请输入页面7编号PageOrder[7]：5  
请输入页面8编号PageOrder[8]：4  
请输入页面9编号PageOrder[9]：0  
请输入页面10编号PageOrder[10]：  
2  
请输入页面11编号PageOrder[11]：1  
请输入页面12编号PageOrder[12]：0  
  
请选择想要先使用的算法（ 1-先进先出(FIFO)页面置换算法，2-最佳(OPI)页面置换算法，3-最近最久未使用(LRU)页面置换算法 ）：2

#### 输出

您选择的是2-最佳(OPI)页面置换算法  
页面6于最远距离100换掉第2物理块中的页面  
页面9于最远距离100换掉第1物理块中的页面  
页面12于最远距离0换掉第0物理块中的页面  
  
页面置换算法模拟过程如下：  
 Page 1 Page 2 Page 3 Page 4 Page 5 Page 6 Page 7 Page 8 Page 9 Page10 Page11 Page12  
BlockNum 1 4 4 4 4 4 4 4 4 2 2 2  
BlockNum 2 3 3 3 3 5 5 5 5 5 5  
BlockNum 3 2 2 2 2 2 2 2 2 2 2  
BlockNum 4 1 1 1 1 1 1 1 1 1  
页面置换算法缺页次数为：9  
页面置换算法缺页率为：75%

# 二、概要设计

## 1、抽象数据类型的定义

int PageOrder[MaxNumber];//页面序列  
int Simulate[MaxNumber][MaxNumber];//模拟过程  
int PageNum;//页面数  
int MinBlockNum;//最小物理块数  
int LackNum;//缺页数  
double LackPageRate;//缺页率  
bool found;  
int isAlgorithm;//算法选择

## 2、主程序的流程

int main() {  
 virtualMemoryPageReplacementAlgorithm virtualMemoryPageReplacementAlgorithm{};  
  
 virtualMemoryPageReplacementAlgorithm.Input();  
  
 return 0;  
}

## 3、各程序模块之间的层次(调用)关系

int main() {  
 virtualMemoryPageReplacementAlgorithm virtualMemoryPageReplacementAlgorithm{};  
  
 virtualMemoryPageReplacementAlgorithm.Input();  
  
 return 0;  
}  
  
//输入函数调用InputAlgorithm函数选择输入函数  
void Input() {  
 ···  
 InputAlgorithm();  
}  
  
//调用IsAlgorithm函数进行算法存储确认  
void InputAlgorithm() {  
 ···  
 IsAlgorithm();  
}  
  
//算法确认后按照确认结果调用不同算法函数，若确认失败，要求重新输入  
void IsAlgorithm() {  
 ···  
 switch (isAlgorithm) {  
 case 1:  
 AlgorithmFIFO();  
 case 2:  
 AlgorithmOPI();  
 case 3:  
 AlgorithmLRU();  
 default:  
 InputAlgorithm();  
 }  
}  
  
//算法函数仅以FIFO为例，在完成算法处理后调用Print函数输出本算法运算结果，并调用NextAlgorithm函数询问下一算法使用  
void AlgorithmFIFO() {  
 ···  
  
 Print();  
  
 NextAlgorithm();  
}  
  
//询问后续，若有则重新调用算法确认函数循环流程，若触发终止条件结束进程  
void NextAlgorithm() {  
 ···  
 IsAlgorithm();  
}

# 三、详细设计

**实现程序模块的具体算法**

## 1、先进先出(FIFO)页面置换算法

//调用先进先出(FIFO)页面置换算法进行调度计算  
void AlgorithmFIFO() {  
 //初始化缺页次数  
 LackNum = 0;  
 //定义队列指针，指向下一个换出的页面  
 int pointer = 0;  
  
 for (int i = 1; i <= PageNum; i++) {  
 //初始化found  
 found = false;  
  
 //如果是第一个页面，直接添加页面进Simulate[1][i]，缺页数+1  
 if (i == 1) {  
 ···  
 continue;  
 }  
  
 for (int j = 1; j <= MinBlockNum; j++) {  
 //判断是否在物理块中，若存在，则此页面未缺页  
 if (Simulate[j][i - 1] == PageOrder[i]) {  
 ···  
 break;  
 }  
  
 //若缺页且有空物理块，则不置换，直接填入  
 if (Simulate[j][i - 1] == 0) {  
 ···  
 break;  
 }  
 }  
  
 //若正常缺页，进行置换且输出置换步骤  
 if (!found) {  
 ···  
 }  
 }  
  
 LackPageRate = (double) LackNum / (double) PageNum;  
  
 Print();  
  
 NextAlgorithm();  
}

## 2、最佳(OPI)页面置换算法

//调用最佳(OPI)页面置换算法进行调度计算  
void AlgorithmOPI() {  
 //初始化缺页次数  
 LackNum = 0;  
  
 for (int i = 1; i <= PageNum; i++) {  
 //初始化found  
 found = false;  
 //初始化队列指针，指向下一个换出的页面  
 int pointer = 0;  
 //初始化最远距离  
 int distance = 0;  
  
 //如果是第一个页面，直接添加页面，缺页数+1  
 if (i == 1) {  
 ···  
 continue;  
 }  
  
 for (int j = 1; j <= MinBlockNum; j++) {  
 //判断是否在物理块中，若存在，则此页面未缺页  
 ···  
 break;  
 }  
  
 //若缺页且有空物理块，则不置换，直接填入  
 if (Simulate[j][i - 1] == 0) {  
 ···  
 break;  
 }  
 }  
  
 if (!found) {  
 for (int j = 1; j <= MinBlockNum; j++) {  
 Simulate[j][i] = Simulate[j][i - 1];  
 //寻找最优置换位  
 for (int k = i + 1; k <= PageNum; k++) {  
 if (Simulate[j][i] == PageOrder[k]) {  
 if (k - i > distance) {  
 distance = k - i;  
 pointer = j;  
 }  
 break;  
 }  
 //判断页列表中无此需求，直接将此页面作为替换页面  
 if (k == PageNum && distance < MaxNumber) {  
 distance = MaxNumber;  
 pointer = j;  
 }  
 }  
 }  
   
 ···  
   
 }  
 }  
  
 LackPageRate = (double) LackNum / (double) PageNum;  
  
 Print();  
  
 NextAlgorithm();  
}

## 3、最近最久未使用(LRU)页面置换算法

//调用最近最久未使用(LRU)页面置换算法进行调度计算  
void AlgorithmLRU() {  
 //初始化缺页次数  
 LackNum = 0;  
  
 for (int i = 1; i <= PageNum; i++) {  
 //初始化found  
 found = false;  
 //初始化队列指针，指向下一个换出的页面  
 int pointer = 0;  
 //初始化最远距离  
 int distance = 0;  
  
 //如果是第一个页面，直接添加页面，缺页数+1  
 if (i == 1) {  
 ···  
 continue;  
 }  
  
 for (int j = 1; j <= MinBlockNum; j++) {  
 //判断是否在物理块中，若存在，则此页面未缺页  
 ···  
 break;  
 }  
  
 //若缺页且有空物理块，则不置换，直接填入  
 if (Simulate[j][i - 1] == 0) {  
 ···  
 break;  
 }  
 }  
  
 if (!found) {  
 for (int j = 1; j <= MinBlockNum; j++) {  
 Simulate[j][i] = Simulate[j][i - 1];  
 //寻找最优置换位  
 for (int k = i - 1; k > 0; k--) {  
 if (Simulate[j][i] == PageOrder[k]) {  
 if (i - k > distance) {  
 distance = i - k;  
 pointer = j;  
 }  
 break;  
 }  
 //判断页列表中无此需求，直接将此页面作为替换页面  
 if (k == 1 && distance < MaxNumber) {  
 distance = MaxNumber;  
 pointer = j;  
 }  
 }  
 }  
   
 ···  
   
 }  
 }  
  
 LackPageRate = (double) LackNum / (double) PageNum;  
  
 Print();  
  
 NextAlgorithm();  
}

# 四、调试分析

## 调试过程中遇到的问题以及解决方法，设计与实现的回顾讨论和分析

|  |  |  |
| --- | --- | --- |
| 问题 | 描述 | 解决方法 |
| OPI、LRU算法同优先级处理问题 | OPI、LRU算法中多个页面本页面序列中都不再使用时，因判定原因，会造成无限循环 | 使用if (k == 1 && distance < MaxNumber)条件语句解决本问题 |

## 算法的性能分析(包括基本操作和其它算法的时间复杂度和空间复杂度的分析)及其改进设想

### 性能分析

|  |  |  |
| --- | --- | --- |
| 算法 | 时间复杂度 | 空间复杂度 |
| 先进先出(FIFO)页面置换算法 | T(n) = O(n2) | S(n) = O(n2) |
| 最佳(OPI)页面置换算法 | T(n) = O(n2) | S(n) = O(n2) |
| 最近最久未使用(LRU)页面置换算法 | T(n) = O(n2) | S(n) = O(n2) |
| Print数据输出 | T(n) = O(n) | S(n) = O(n2) |
| NextAlgorithm算法 | T(n) = O(1) | S(n) = O(1) |

### 改进设想

设置页面0为状态符导致无法使用页面号为0的情况，可以考虑设计为数据结构

# 五、用户使用说明

## 使用说明

* 控制台会提示要求用户进行输入，按提示输入内容即可
* 不可输入页面0（该页面号设置为状态量）
* 选择算法输入完成后将会输出虚拟内存页面置换过程及模拟列表
* 可根据需要选择是否使用其他算法进行虚拟内存页面置换或退出

# 六、测试结果

## 测试结果，包括输入和输出

请输入最小物理块数MinBlockNum：4  
请输入页面个数PageNum：12  
请输入页面1编号PageOrder[1]：4  
请输入页面2编号PageOrder[2]：3  
请输入页面3编号PageOrder[3]：2  
请输入页面4编号PageOrder[4]：1  
请输入页面5编号PageOrder[5]：4  
请输入页面6编号PageOrder[6]：3  
请输入页面7编号PageOrder[7]：5  
请输入页面8编号PageOrder[8]：4  
请输入页面9编号PageOrder[9]：3  
请输入页面10编号PageOrder[10]：2  
请输入页面11编号PageOrder[11]：1  
请输入页面12编号PageOrder[12]：5  
  
请选择想要先使用的算法（ 1-先进先出(FIFO)页面置换算法，2-最佳(OPI)页面置换算法，3-最近最久未使用(LRU)页面置换算法 ）：1  
  
您选择的是1-先进先出(FIFO)页面置换算法  
页面7换掉第1物理块中的页面  
页面8换掉第2物理块中的页面  
页面9换掉第3物理块中的页面  
页面10换掉第4物理块中的页面  
页面11换掉第1物理块中的页面  
页面12换掉第2物理块中的页面  
  
页面置换算法模拟过程如下：  
 Page 1 Page 2 Page 3 Page 4 Page 5 Page 6 Page 7 Page 8 Page 9 Page10 Page11 Page12  
BlockNum 1 4 4 4 4 4 4 5 5 5 5 1 1  
BlockNum 2 3 3 3 3 3 3 4 4 4 4 5  
BlockNum 3 2 2 2 2 2 2 3 3 3 3  
BlockNum 4 1 1 1 1 1 1 2 2 2  
页面置换算法缺页次数为：10  
页面置换算法缺页率为：83%  
  
请问是否还要进行其余算法，若是，请输入（1-3值）；若否，请输入任意字符（ 1-先进先出(FIFO)页面置换算法，2-最佳(OPI)页面置换算法，3-最近最久未使用(LRU)页面置换算法 ）：2  
  
您选择的是2-最佳(OPI)页面置换算法  
页面7于最远距离4换掉第4物理块中的页面  
页面11于最远距离100换掉第1物理块中的页面  
  
页面置换算法模拟过程如下：  
 Page 1 Page 2 Page 3 Page 4 Page 5 Page 6 Page 7 Page 8 Page 9 Page10 Page11 Page12  
BlockNum 1 4 4 4 4 4 4 4 4 4 4 1 1  
BlockNum 2 3 3 3 3 3 3 3 3 3 3 3  
BlockNum 3 2 2 2 2 2 2 2 2 2 2  
BlockNum 4 1 1 1 5 5 5 5 5 5  
页面置换算法缺页次数为：6  
页面置换算法缺页率为：50%  
  
请问是否还要进行其余算法，若是，请输入（1-3值）；若否，请输入任意字符（ 1-先进先出(FIFO)页面置换算法，2-最佳(OPI)页面置换算法，3-最近最久未使用(LRU)页面置换算法 ）：3  
  
您选择的是3-最近最久未使用(LRU)页面置换算法  
页面7于最远距离4换掉第3物理块中的页面  
页面10于最远距离6换掉第4物理块中的页面  
页面11于最远距离4换掉第3物理块中的页面  
页面12于最远距离4换掉第1物理块中的页面  
  
页面置换算法模拟过程如下：  
 Page 1 Page 2 Page 3 Page 4 Page 5 Page 6 Page 7 Page 8 Page 9 Page10 Page11 Page12  
BlockNum 1 4 4 4 4 4 4 4 4 4 4 4 5  
BlockNum 2 3 3 3 3 3 3 3 3 3 3 3  
BlockNum 3 2 2 2 2 5 5 5 5 1 1  
BlockNum 4 1 1 1 1 1 1 2 2 2  
页面置换算法缺页次数为：8  
页面置换算法缺页率为：67%  
  
请问是否还要进行其余算法，若是，请输入（1-3值）；若否，请输入任意字符（ 1-先进先出(FIFO)页面置换算法，2-最佳(OPI)页面置换算法，3-最近最久未使用(LRU)页面置换算法 ）：0  
  
Process finished with exit code 0

# 七、附录

## 带注释的源程序

#include <iostream>  
#include <iomanip>  
  
using namespace std;  
#define MaxNumber 100  
  
class virtualMemoryPageReplacementAlgorithm {  
public:  
  
 int PageOrder[MaxNumber];//页面序列  
 int Simulate[MaxNumber][MaxNumber];//模拟过程  
// int PageCount[MaxNumber];//当前内存距离下一次出现的距离  
 int PageNum;//页面数  
 int MinBlockNum;//最小物理块数  
 int LackNum;//缺页数  
 double LackPageRate;//缺页率  
 bool found;  
 int isAlgorithm;//算法选择  
  
 //输入空闲分区数、空闲的分区大小、进程数、进程需要的分区大小  
 void Input() {  
 cout << "请输入最小物理块数MinBlockNum：";  
 cin >> MinBlockNum;  
 cout << "请输入页面个数PageNum：";  
 cin >> PageNum;  
  
 for (int i = 1; i <= PageNum; i++) {  
 cout << "请输入页面" << i << "编号" << "PageOrder[" << i << "]：";  
 cin >> PageOrder[i];  
 }  
  
 InputAlgorithm();  
 }  
  
 //获取算法选择输入  
 void InputAlgorithm() {  
 cout << endl  
 << "请选择想要先使用的算法（ 1-先进先出(FIFO)页面置换算法，2-最佳(OPI)页面置换算法，3-最近最久未使用(LRU)页面置换算法 ）：";  
 cin >> isAlgorithm;  
 IsAlgorithm();  
 }  
  
 //算法存储确认  
 void IsAlgorithm() {  
 switch (isAlgorithm) {  
 case 1:  
 cout << endl << "您选择的是1-先进先出(FIFO)页面置换算法" << endl;  
 AlgorithmFIFO();  
 break;  
 case 2:  
 cout << endl << "您选择的是2-最佳(OPI)页面置换算法" << endl;  
 AlgorithmOPI();  
 break;  
 case 3:  
 cout << endl << "您选择的是3-最近最久未使用(LRU)页面置换算法" << endl;  
 AlgorithmLRU();  
 break;  
 default:  
 cout << "算法值：" << isAlgorithm  
 << "有误,请重新输入正确的算法类型（ 1-先进先出(FIFO)页面置换算法，2-最佳(OPI)页面置换算法，3-最近最久未使用(LRU)页面置换算法 ）"  
 << endl;  
 InputAlgorithm();  
 }  
 }  
  
 //询问是否还要进行其余算法  
 void NextAlgorithm() {  
 cout << endl  
 << "请问是否还要进行其余算法，若是，请输入（1-3值）；若否，请输入任意字符（ 1-先进先出(FIFO)页面置换算法，2-最佳(OPI)页面置换算法，3-最近最久未使用(LRU)页面置换算法 ）：";  
 cin >> isAlgorithm;  
 if (isAlgorithm != 1 && isAlgorithm != 2 && isAlgorithm != 3) {  
 return;  
 }  
 IsAlgorithm();  
 }  
  
 //输出页面置换算法模拟过程及缺页次数与缺页率  
 void Print() {  
  
 cout << endl << "页面置换算法模拟过程如下：" << endl;  
  
 //模拟过程  
 cout << left << setw(10) << "";  
 for (int i = 1; i <= PageNum; i++) {  
 cout << right << setw(8) << "Page" << setw(2) << i;  
 }  
  
 for (int i = 1; i <= MinBlockNum; i++) {  
 cout << endl << left << setw(8) << "BlockNum" << right << setw(2) << i;  
 for (int j = 1; j <= PageNum; j++) {  
 if (Simulate[i][j] != 0) {  
 cout << right << setw(10) << Simulate[i][j];  
 } else {  
 cout << setw(10) << "";  
 }  
 }  
 }  
  
 cout << endl << "页面置换算法缺页次数为：" << LackNum << endl;  
  
 cout << "页面置换算法缺页率为：" << setprecision(2) << LackPageRate \* 100 << "%" << endl;  
  
 }  
  
 //调用先进先出(FIFO)页面置换算法进行调度计算  
 void AlgorithmFIFO() {  
 //初始化缺页次数  
 LackNum = 0;  
 //定义队列指针，指向下一个换出的页面  
 int pointer = 0;  
  
 for (int i = 1; i <= PageNum; i++) {  
 //初始化found  
 found = false;  
  
 //如果是第一个页面，直接添加页面，缺页数+1  
 if (i == 1) {  
 Simulate[1][i] = PageOrder[i];  
 LackNum++;  
 pointer = 1;  
 continue;  
 }  
  
 for (int j = 1; j <= MinBlockNum; j++) {  
 //判断是否在物理块中，若存在，则此页面未缺页  
 if (Simulate[j][i - 1] == PageOrder[i]) {  
 for (int k = 1; k <= MinBlockNum; k++) {  
 Simulate[k][i] = Simulate[k][i - 1];  
 }  
 found = true;  
 break;  
 }  
  
 //若缺页且有空物理块，则不置换，直接填入  
 if (Simulate[j][i - 1] == 0) {  
 for (int k = 1; k <= MinBlockNum; k++) {  
 Simulate[k][i] = Simulate[k][i - 1];  
 }  
 Simulate[j][i] = PageOrder[i];  
 LackNum++;  
 found = true;  
 break;  
 }  
 }  
  
 if (!found) {  
 for (int j = 1; j <= MinBlockNum; j++) {  
 Simulate[j][i] = Simulate[j][i - 1];  
 }  
 cout << "页面" << i << "换掉第" << pointer << "物理块中的页面" << endl;  
 Simulate[pointer][i] = PageOrder[i];  
 LackNum++;  
 if (pointer == MinBlockNum) {  
 pointer = 1;  
 } else {  
 pointer++;  
 }  
 }  
 }  
  
 LackPageRate = (double) LackNum / (double) PageNum;  
  
 Print();  
  
 NextAlgorithm();  
 }  
  
 //调用最佳(OPI)页面置换算法进行调度计算  
 void AlgorithmOPI() {  
 //初始化缺页次数  
 LackNum = 0;  
  
 for (int i = 1; i <= PageNum; i++) {  
 //初始化found  
 found = false;  
 //初始化队列指针，指向下一个换出的页面  
 int pointer = 0;  
 //初始化最远距离  
 int distance = 0;  
  
 //如果是第一个页面，直接添加页面，缺页数+1  
 if (i == 1) {  
 Simulate[1][i] = PageOrder[i];  
 LackNum++;  
 continue;  
 }  
  
 for (int j = 1; j <= MinBlockNum; j++) {  
 //判断是否在物理块中，若存在，则此页面未缺页  
 if (Simulate[j][i - 1] == PageOrder[i]) {  
 for (int k = 1; k <= MinBlockNum; k++) {  
 Simulate[k][i] = Simulate[k][i - 1];  
 }  
 found = true;  
 break;  
 }  
  
 //若缺页且有空物理块，则不置换，直接填入  
 if (Simulate[j][i - 1] == 0) {  
 for (int k = 1; k <= MinBlockNum; k++) {  
 Simulate[k][i] = Simulate[k][i - 1];  
 }  
 Simulate[j][i] = PageOrder[i];  
 LackNum++;  
 found = true;  
 break;  
 }  
 }  
  
 if (!found) {  
 for (int j = 1; j <= MinBlockNum; j++) {  
 Simulate[j][i] = Simulate[j][i - 1];  
 //寻找最优置换位  
 for (int k = i + 1; k <= PageNum; k++) {  
 if (Simulate[j][i] == PageOrder[k]) {  
 if (k - i > distance) {  
 distance = k - i;  
 pointer = j;  
 }  
 break;  
 }  
 //判断页列表中无此需求，直接将此页面作为替换页面  
 if (k == PageNum && distance < MaxNumber) {  
 distance = MaxNumber;  
 pointer = j;  
 }  
 }  
 }  
  
 cout << "页面" << i << "于最远距离" << distance << "换掉第" << pointer << "物理块中的页面" << endl;  
  
 Simulate[pointer][i] = PageOrder[i];  
 LackNum++;  
 }  
 }  
  
 LackPageRate = (double) LackNum / (double) PageNum;  
  
 Print();  
  
 NextAlgorithm();  
 }  
  
 //调用最近最久未使用(LRU)页面置换算法进行调度计算  
 void AlgorithmLRU() {  
 //初始化缺页次数  
 LackNum = 0;  
  
 for (int i = 1; i <= PageNum; i++) {  
 //初始化found  
 found = false;  
 //初始化队列指针，指向下一个换出的页面  
 int pointer = 0;  
 //初始化最远距离  
 int distance = 0;  
  
 //如果是第一个页面，直接添加页面，缺页数+1  
 if (i == 1) {  
 Simulate[1][i] = PageOrder[i];  
 LackNum++;  
 continue;  
 }  
  
 for (int j = 1; j <= MinBlockNum; j++) {  
 //判断是否在物理块中，若存在，则此页面未缺页  
 if (Simulate[j][i - 1] == PageOrder[i]) {  
 for (int k = 1; k <= MinBlockNum; k++) {  
 Simulate[k][i] = Simulate[k][i - 1];  
 }  
 found = true;  
 break;  
 }  
  
 //若缺页且有空物理块，则不置换，直接填入  
 if (Simulate[j][i - 1] == 0) {  
 for (int k = 1; k <= MinBlockNum; k++) {  
 Simulate[k][i] = Simulate[k][i - 1];  
 }  
 Simulate[j][i] = PageOrder[i];  
 LackNum++;  
 found = true;  
 break;  
 }  
 }  
  
 if (!found) {  
 for (int j = 1; j <= MinBlockNum; j++) {  
 Simulate[j][i] = Simulate[j][i - 1];  
 //寻找最优置换位  
 for (int k = i - 1; k > 0; k--) {  
 if (Simulate[j][i] == PageOrder[k]) {  
 if (i - k > distance) {  
 distance = i - k;  
 pointer = j;  
 }  
 break;  
 }  
 //判断页列表中无此需求，直接将此页面作为替换页面  
 if (k == 1 && distance < MaxNumber) {  
 distance = MaxNumber;  
 pointer = j;  
 }  
 }  
 }  
  
 cout << "页面" << i << "于最远距离" << distance << "换掉第" << pointer << "物理块中的页面" << endl;  
  
 Simulate[pointer][i] = PageOrder[i];  
 LackNum++;  
 }  
 }  
  
 LackPageRate = (double) LackNum / (double) PageNum;  
  
 Print();  
  
 NextAlgorithm();  
 }  
};  
  
int main() {  
 virtualMemoryPageReplacementAlgorithm virtualMemoryPageReplacementAlgorithm{};  
  
 virtualMemoryPageReplacementAlgorithm.Input();  
  
 return 0;  
}