# OS_LAB4 概要设计

## 一、最佳置换算法

### 实验目标：

通过算法模拟：选择永不使用或是在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。计算缺页率。

### 数据结构

1. 页表结构体
2. 工作集
3. 访问序列

### 主要函数

1. 初始化函数：初始化工作集，构造访问序列。
2. 访问序列生成函数：生成有局部特征的随机访问序列。
3. 工作集页权重计算函数：计算工作集中每页最近的访问距离。
4. 最佳置换控制函数：按照访问序列访问，计算缺页率。

### 设计与实现

#### 1. 访问序列生成函数

- 确定虚拟内存的尺寸N，工作集的起始位置p，工作集中包含的页数e，工作集移动率m（每处理m个页面访问则将起始位置p +1），以及一个范围在0和1之间的值t；
- 生成m个取值范围在p和p + e间的随机数，并记录到页面访问序列串中；
- 生成一个随机数r，0 ≤ r ≤ 1；
- 如果r < t，则为p生成一个新值，否则p = (p + 1) mod N；
- 如果想继续加大页面访问序列串的长度，请返回第2步，否则结束。

```
//随机访问序列生成函数
void create_visilist(int size)
{
    int N = 20;
    int p = 0;
    int m = 6;
    double t = 0.7;
    int choose = 0;
    cout<<endl <<"是否要改变访问序列生成时的其他参数[0(否)/1(是)]: ";
    cin >> choose;
    if (choose == 1)
    {
```

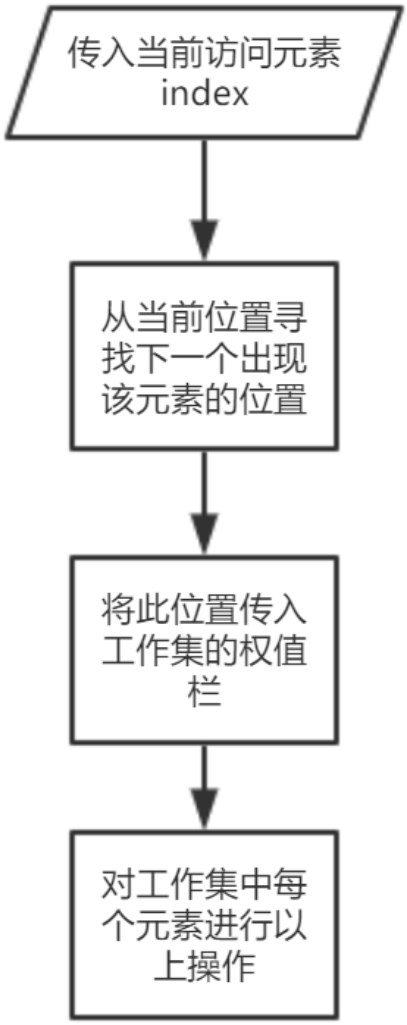
```
        cout << endl << "请输入虚拟内存尺寸N: ";
        cin >> N;
        cout << endl << "请输入起始位置p: ";
        cin >> p;
        cout << endl << "请输入工作集移动率m: ";
        cin >> m;
        cout << endl << "请输入访问调变率t: ";
        cin >> t;
    }

    visitlist = new int[size];
    int temp = 0;
    visitlist[0] = 0;
    for (int  i = 1; i < size; i++)
    {
        temp++;
        if (temp%m == 0)
        {
            p = p + 1;
        }
        int temp = rand() % 6 + p;
        double r = rand() % 100 / 100.0;
        if (r<t)
        {
            visitlist[i] = temp;
        }

        else
        {
            visitlist[i] = (visitlist[i - 1] + 1) % N;
        }
    }
    cout << "随机访问序列为: " << endl;
    for (int i = 0; i < size; i++)
    {
        cout << visitlist[i]<<" ";
    }

}
```

**2. 工作集页权重计算函数**

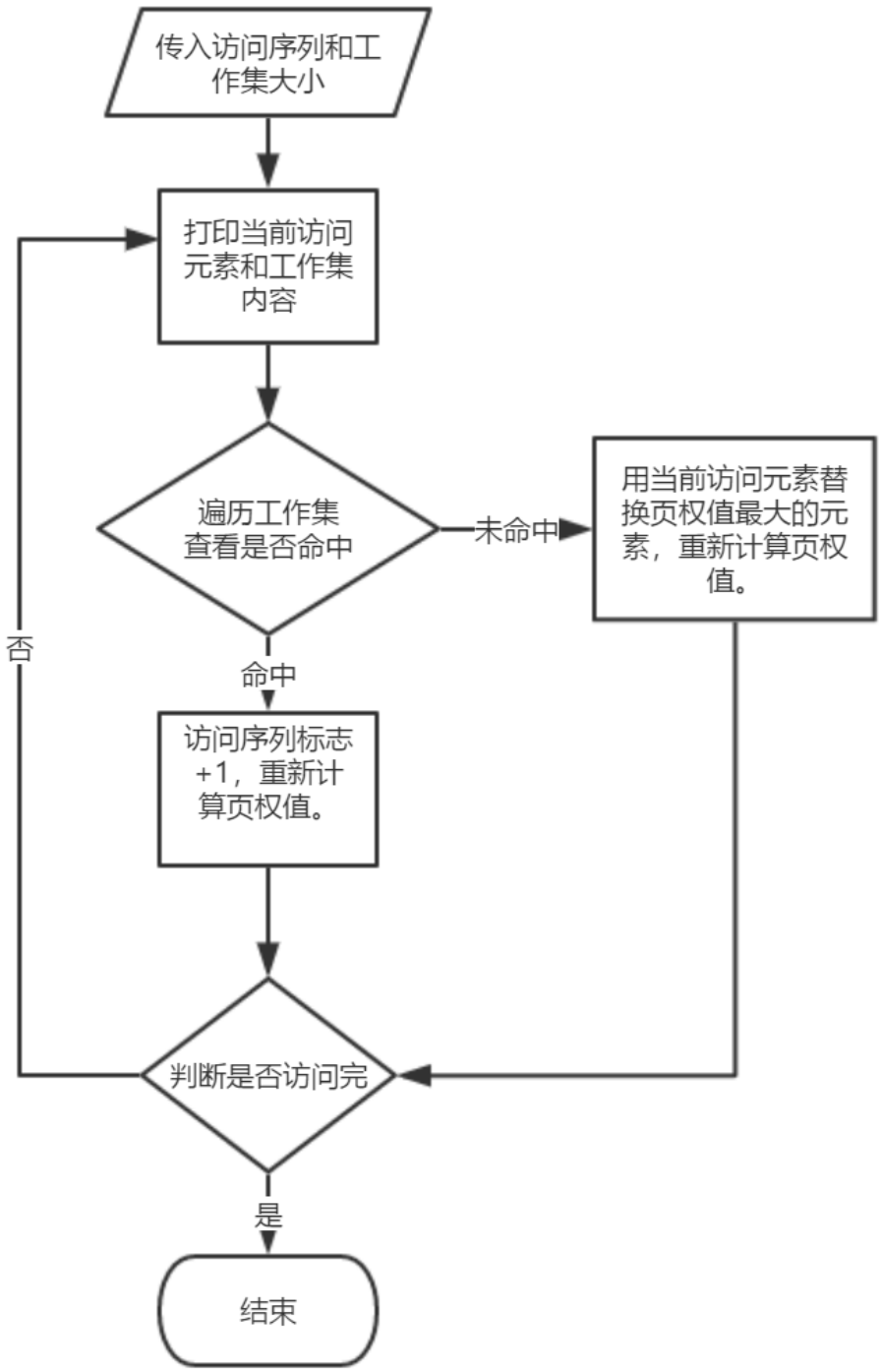



遍历访问序列获取工作集中每个元素的权值，并装入权值表。在最佳置换算法中的权值为最近一次调用的位置距离，如果不再调用则赋一个非常大的数表示不再调用。

```
//工作集中每个元素最近访问位置计算
void beat_value(int wl, int vl, int index) {
    for (int j = 0; j < wl; j++) {
        for (int i = index; i < vl; i++)
        {
            if (worklist[j][0] == visitlist[i]) {
                worklist[j][1] = i-index;
                break;
            }
            worklist[j][1] = 214748364;
        }
    }


}
```

**3. 最佳置换控制函数**




每次访问页面时查看是否命中，若命中则直接访问工作集内页面，若没有命中则需要将工作集中元素进行替换，替换的原则是最远不被访问的界面被替换，如上图。

```
//最佳置换算法
void best_change(int wl, int vl) {
    int right_count = 0;
    for (int i = 0; i < vl; i++)
```

```
    {

        cout<<endl << "访问元素: " << visitlist[i];
        print_worklist(wl);
        int temp = 0;
        for (int j = 0; j < wl; j++)
        {
            if (visitlist[i]==worklist[j][0])
            {
                temp = 1;
                break;
            }
        }
        if (temp==1)
        {
            right_count++;
            beat_value(wl, vl, i + 1);
            continue;
        }
        else
        {
            int max_index = 0;
            int max_value = worklist[0][1];
            for (int k = 0; k < wl; k++)
            {
                if (worklist[k][1]>max_value)
                {
                    max_index = k;
                }
            }
            worklist[max_index][0] = visitlist[i];
            beat_value(wl, vl, i+1);
            cout<<endl << "置换位置: " << max_index;
        }
    }
}
```

**运行结果:**

**运行结果:**

工作集大小: 4

访问序列大小: 50

访问序列生成参数如下

```
int N = 20;
int p = 0;
int m = 6;
double t = 0.7;
```

运行结果（部分）如下图：

```
请输入工作集大小：4

请输入访问序列大小：50

是否要改变访问序列生成时的其他参数[0(否)/1(是)]：0
随机访问序列为：
0 3 2 4 5 3 5 6 6 5 6 6 7 6 3 6 5 2 7 5 7 8 4 5 7 6 6 4 8 9 5 6 7 9 7 8 9 8 9 10 10 6 7 8 10 10 11 9 9 13
工作集初始化结果：
0
3
2
4
```

```
**************************************
访问元素：9
工作集：
9   0
6   214748364
7   214748364
8   6

**************************************
访问元素：10
工作集：
9   8
6   214748364
7   214748364
8   5

置换位置：3
**************************************
访问元素：11
工作集：
9   7
6   214748364
10   5
8   4

置换位置：2
**************************************
访问元素：12
工作集：
9   6
11   2
10   4
8   3

置换位置：1
**************************************
访问元素：13
工作集：
12   6
11   1
10   3
8   2

置换位置：1
```

```
************************************
访问元素：8
工作集：
10  214748364
8  0
4  214748364
9  214748364
7  214748364

************************************
访问元素：12
工作集：
10  214748364
8  214748364
4  214748364
9  214748364
7  214748364

置换位置：1
************************************
访问元素：12
工作集：
12  0
8  214748364
4  214748364
9  214748364
7  214748364

页面置换次数：13
缺页率0.26
Total time:432ms
```

## 二、先进先出置换算法（FIFO）

### 实验目标：

通过算法模拟：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进入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。计算缺页率。

### 数据结构

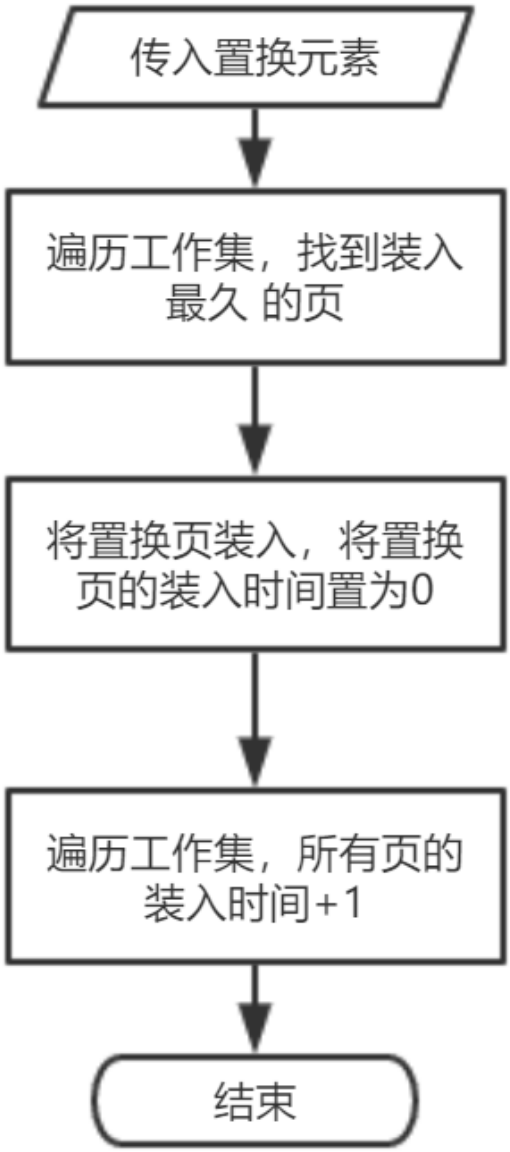
1. 页表结构体
2. 工作集
3. 访问序列

### 主要函数

1. 初始化函数：初始化工作集，构造访问序列。
2. 访问序列生成函数：生成有局部特征的随机访问序列。
3. 工作集元素置换函数：将最早装入的页换出，并对其余页的访问时间+1
4. 最佳置换控制函数：按照访问序列进行先进先出的访问，并计算缺页率。

### 设计与实现

**工作集元素置换函数：**

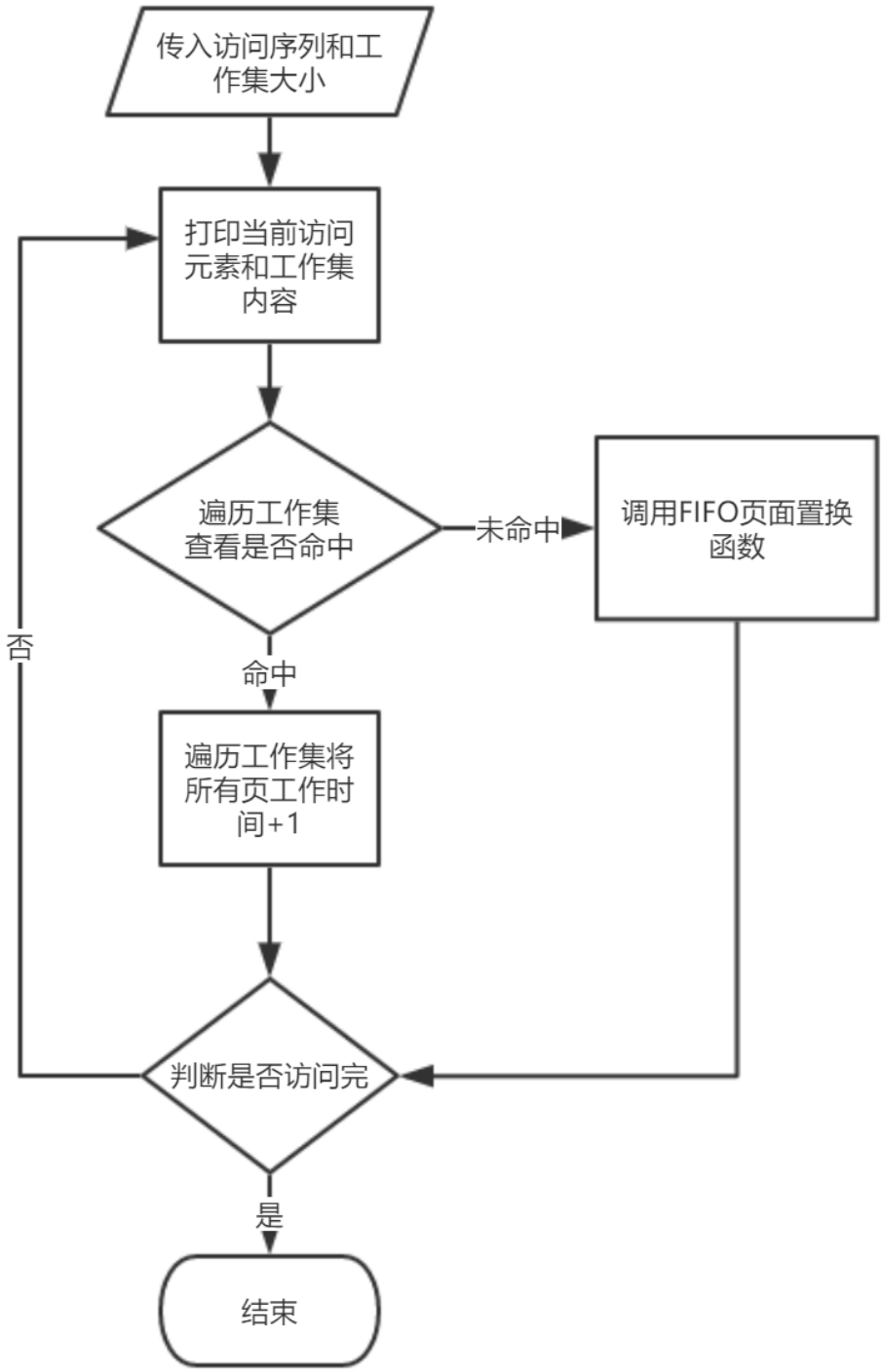



代码：

```
//先进先出元素替换函数
void first_in_outchangge(int now,int wl) {
    int max_index = 0;
    int max_value = worklist[0][1];
    for (int k = 0; k < wl; k++)
    {
        if (worklist[k][1] > max_value)
        {
            max_index = k;
        }
    }
    worklist[max_index][0] = now;
    worklist[max_index][1] = 0;
    cout << endl << "置换位置: " << max_index;
```

```
    for (int k = 0; k < wl; k++)
    {
        worklist[k][1]++;
    }
}
```

**最佳置换控制函数：**




代码：

```
//先进先出置换算法
void FIFO(int wl, int vl) {
```

```
    for (int k = 0; k < wl; k++)
    {
        worklist[k][1] = 0;
    }
    int right_count = 0;
    for (int i = 0; i < vl; i++)
    {

        cout << endl << "访问元素: " << visitlist[i];
        print_worklist(wl);
        int temp = 0;
        for (int j = 0; j < wl; j++)
        {
            if (visitlist[i] == worklist[j][0])
            {
                temp = 1;
                break;
            }
        }
        if (temp == 1)
        {
            for (int k = 0; k < wl; k++)
            {
                worklist[k][1]++;
            }
            continue;
        }
        else
        {
            first_in_outchangge(visitlist[i], wl);

        }
    }
}
```

**运行结果：**

工作集大小：4

访问序列大小：50

访问序列生成参数如下

```
int N = 20;
int p = 0;
int m = 6;
double t = 0.7;
```

运行结果（部分）如下图：

```
请输入工作集大小：4

请输入访问序列大小：50

是否要改变访问序列生成时的其他参数[0(否)/1(是)]：0
随机访问序列为：
0 3 3 0 3 4 5 5 3 4 2 3 4 2 3 3 4 5 4 3 4 3 5 6 8 9 10 11 9 10 11 12 5 6 7 9 8 9 10 7 8 6 7 8 8 10 11 10 12 13
工作集初始化结果：
0
3
3
0
```

```
**************************************
访问元素：3
工作集：
4  3
3  8
3  8
5  2

**************************************
访问元素：4
工作集：
4  4
3  9
3  9
5  3

**************************************
访问元素：2
工作集：
4  5
3  10
3  10
5  4

置换位置：3

**************************************
访问元素：3
工作集：
4  6
3  11
2  1
5  5
```

```
************************************
访问元素：10
工作集：
7  5
6  6
10  9
11  1

************************************
访问元素：12
工作集：
7  6
6  7
10  10
11  2

置换位置：3

************************************
访问元素：13
工作集：
7  7
6  8
12  1
11  3

置换位置：2

页面置换次数：20
缺页率0.4
Total time:326ms
```

# 三、LRU置换算法

## 实验目标：

通过算法模拟：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计算缺页率。

## 数据结构

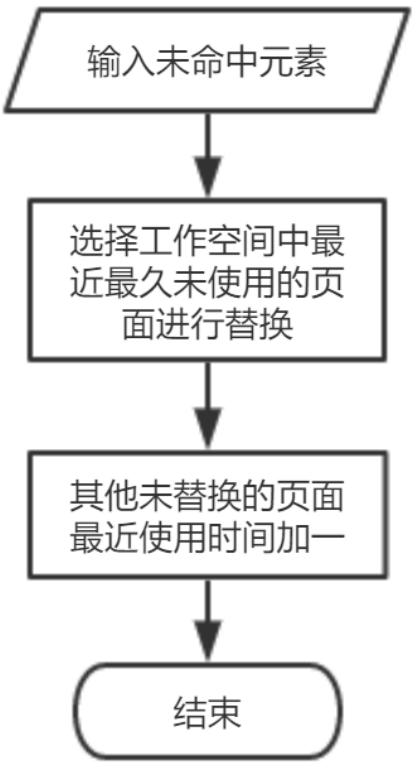
1. 页表结构体
2. 工作集
3. 访问序列

## 主要函数

1. 初始化函数：初始化工作集，构造访问序列。
2. 访问序列生成函数：生成有局部特征的随机访问序列。
3. LRU工作集页面替换函数：若出现缺页时则将最近最久未使用的页换出，并重新计算工作集中每页的最近调用时间。
4. LRU置换算法：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，并计算缺页率和系统开销。

## 设计与实现

**LRU工作元素置换函数：**

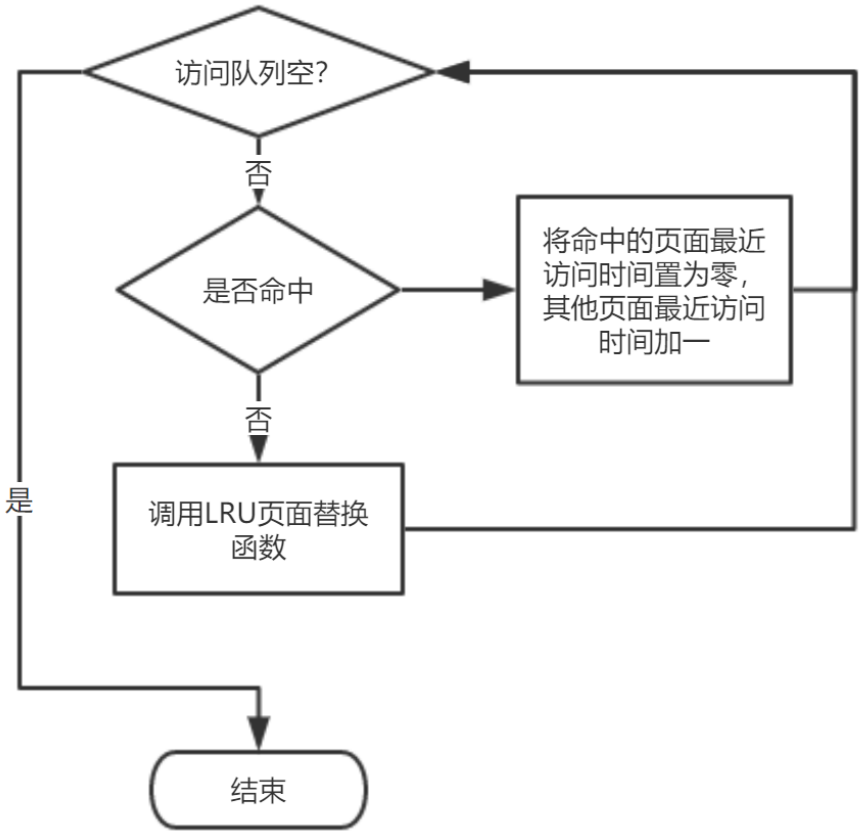



```
//LRU元素置换函数
void LRU_value( int  now,int wl,double *con) {
    int temp = 0;
    for (int j = 0; j < wl; j++)
    {
        if (now == worklist[j][0])
        {
            temp = 1;
            worklist[j][1]=0;
            break;
        }
    }
    if (temp == 1)
    {
        cout << "命中" << endl;
        *con = *con+1;
        for (int k = 0; k < wl; k++)
        {
            worklist[k][1]++;
        }
    }
    else
    {
        cout << "未命中" ;
        first_in_outchangge(now, wl);
```

```
    }

}
```

**LRU置换主控函数：**




```
//LRU置换算法
void LRU(int wl, int vl) {
    for (int k = 0; k < wl; k++)
    {
        worklist[k][1] = 0;
    }
    double right_count = 0;

    for (int i = 0; i < vl; i++)
    {

        cout << endl << "访问元素: " << visitlist[i];
        print_worklist(wl);
        LRU_value(visitlist[i], wl,&right_count);
    }
    cout << endl << "页面置换次数: " << vl - right_count;
    cout << endl << "缺页率" << 1 - double(right_count / vl) << endl;
}
```

## 运行结果：

工作集大小：4

访问序列大小：50

访问序列生成参数如下

```
int N = 20;
int p = 0;
int m = 6;
double t = 0.7;
```

运行结果如下图：

```
请输入工作集大小：4

请输入访问序列大小：50

是否要改变访问序列生成时的其他参数[0(否)/1(是)]：0
随机访问序列为：
0 1 2 1 2 2 4 6 2 5 3 4 7 2 5 6 4 5 6 7 7 3 3 4 6 7 5 6 7 4 5 7 5 6 7 9 10 6 10 11 9 10 9 10 10 7 8 8 10 11
工作集初始化结果：
0
1
2
1

**************************************
访问元素：0
工作集：
0  0
1  0
2  0
1  0
命中

**************************************
访问元素：1
工作集：
0  1
1  1
2  1
1  1
命中
```

```
************************************
访问元素：5
工作集：
5  3
2  4
4  1
6  2
命中

************************************
访问元素：6
工作集：
5  1
2  5
4  2
6  3
命中

************************************
访问元素：7
工作集：
5  2
2  6
4  3
6  1
未命中
置换位置：3

************************************
访问元素：7
工作集：
5  3
2  7
7  1
6  2
命中
```

```
************************************
访问元素：10
工作集：
9  6
8  1
7  3
10  4
命中

************************************
访问元素：11
工作集：
9  7
8  2
7  4
10  1
未命中
置换位置：1

页面置换次数：21
缺页率0.42
Total time:351ms
```

## 四、改进clock置换算法

### 实验目标：

通过算法模拟：改进型的Clock算法需要综合考虑某一内存页面的访问位和修改位来判断是否置换该页面。在实际编写算法过程中，同样可以用一个等长的整型数组来标识每个内存块的修改状态，通过不同的状态决定替换的页面，模拟算法运行并计算缺页率。

## 数据结构

1. 页表结构体
2. 工作集
3. 访问序列

## 主要函数

1. 初始化函数：初始化工作集，构造访问序列。
2. 访问序列生成函数：生成有局部特征的随机访问序列。
3. LRU工作集元素置换函数：若出现缺页时则将最近最久未使用的页换出，并重新计算工作集中每页的最近调用时间。
4. 最佳置换控制函数：按照访问序列进行先进先出的访问，并计算缺页率。

## 设计与实现

**一类页面扫描和二类页面扫描函数：**

实现功能：扫描一类页面和二类页面是否存在，并返回对应index

```
//判断工作空间是否存在一类页面，若有，返回坐标，若不存在返回-1
int isexit1(int wl) {
    for (int j = 0; j < wl; j++)
    {
        if (worklist[j][1] == 0 && worklist[j][2] == 0)
        {
            return j;
            break;
        }
    }
    return -1;
}

//判断工作空间是否存在二类页面，若有，返回坐标，若不存在返回-1
int isexit2(int wl) {

    for (int j = 0; j < wl; j++)
    {
        if (worklist[j][1] == 0 && worklist[j][2] == 1)
        {
            return j;
            break;
        }
        worklist[j][1] = 0;
    }
    return -1;
```

```
}
```

**改进clock置换主控函数：**

改进型的Clock算法需要综合考虑某一内存页面的访问位和修改位来判断是否置换该页面。在实际编写算法过程中，同样可以用一个等长的整型数组来标识每个内存块的修改状态。访问位和修改位可以组成一下四种类型的页面。

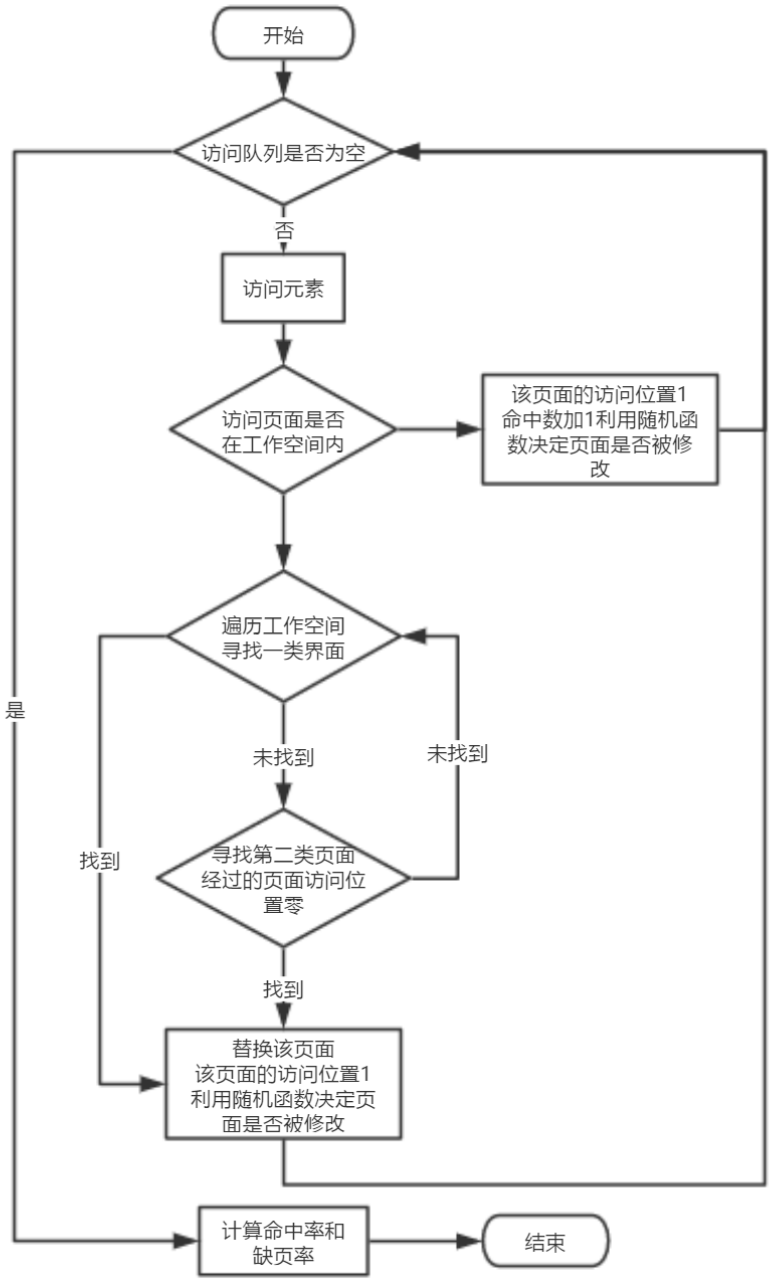
1类(0, 0)：表示该页面最近既未被访问，又未被修改，淘汰等级1。

2类(0, 1)：表示该页面最近未被访问，但已被修改，淘汰等级2。

3类(1, 0)：表示该页面最近已被访问，但未被修改，该页有可能再被访问。

4类(1, 1)：表示该页最近已被访问且被修改，该页可能再被访问。

执行步骤如下：当未命中时，循环扫描队列，寻找第一类页面，将所遇到的第一个一类页面作为所选中的淘汰页。在第一次扫描期间不改变访问位.如果第一步没有找到一类页面，即查找一周后未遇到第一类页面，则开始第二轮扫描，寻找第二类页面，将所遇到的第一个这类页面作为淘汰页。在第二轮扫描期间，将所有扫描过的页面的访问位都置0.如果第二步也失败，亦即未找到第二类页面，则将指针返回到开始位置，并将所有访问位复0.返回第一步。流程如下：




```
void Sclock()
{
    double right_count =0;
    cout << endl << "请输入改进clock工作集大小: ";
    int wl = 0;
    cin >> wl;
    worklist = new int* [wl];
    for (int i = 0; i < wl; i++) {
        worklist[i] = new int[3];
        worklist[i][0] = 0;
        worklist[i][1] = 0;
        worklist[i][2] = 0;
    }
```

```
cout << endl << "请输入访问序列大小: ";
int vl = 0;
cin >> vl;
create_visilist(vl);
cout << endl << "工作集初始化结果: " << endl;
for (int i = 0; i < wl; i++)
{
    worklist[i][0] = visitlist[i];
}
print_cwork(wl);
start1 = clock();
for (int i = 0; i < vl; i++)
{
    cout <<endl<<"************************************"<< endl;
    cout << endl << "访问元素: " << visitlist[i];
    print_cwork(wl);
    int temp = 0;
    for (int j = 0; j < wl; j++)
    {
        if (visitlist[i] == worklist[j][0])
        {
            temp = 1;
            ischange(j);
            break;
        }
    }
    if (temp == 1)
    {
        right_count++;
        continue;
    }
    else
    {
        int index1 = 0;
        if (((index1 = isexit1(wl)))== -1) {
            if ( (index1 = isexit2(wl)) == -1) {
                if ( (index1 = isexit1(wl)) == -1) {
                    (index1 = isexit2(wl));
                    worklist[index1][0] = visitlist[i];
                    worklist[index1][1] = 1;
                    ischange(index1);
                    cout << endl << "置换位置: " << index1 + 1;
                }
                else
                {
                    worklist[index1][0] = visitlist[i];
                    worklist[index1][1] = 1;
                    ischange(index1);
                    cout << endl << "置换位置: " << index1 + 1;
                }
            }
            else {
```

```
                    worklist[index1][0] = visitlist[i];
                    worklist[index1][1] = 1;
                    ischange(index1);
                    cout << endl << "置换位置: " << index1 + 1;
                }
            }
            else
            {
                worklist[index1][0] = visitlist[i];
                worklist[index1][1] = 1;
                ischange(index1);
                cout << endl << "置换位置: " << index1 + 1;
            }

            continue;
        }

    }
    cout << endl << "页面置换次数: " << vl - right_count;
    cout << endl << "缺页率" << 1 - double(right_count / vl) << endl;
    end1 = clock();
    double endtime = (double)(end1 - start1) / CLOCKS_PER_SEC;
    cout << "Total time:" << endtime * 1000 << "ms" << endl;
    return;
}
```

### 运行结果：

工作集大小：4

访问序列大小：50

访问序列生成参数如下

```
int N = 20;
int p = 0;
int m = 6;
double t = 0.7;
```

运行结果如下图：

```
请输入改进clock工作集大小：4

请输入访问序列大小：50

是否要改变访问序列生成时的其他参数[0(否)/1(是)]：0
随机访问序列为：
0 0 3 4 5 6 2 2 1 4 5 5 2 5 4 6 7 4 3 5 4 5 3 8 9 7 5 4 5 5 6 6 8 9 10 11 12 10 11 8 9 11 9 10 11 7 8 12 11 9
工作集初始化结果：

改进clock工作集：
0  0  0
0  0  0
3  0  0
4  0  0
```

```
************************************

访问元素：7
改进clock工作集：
6  1  1
5  0  1
4  1  1
2  0  1

置换位置：2
************************************

访问元素：4
改进clock工作集：
6  0  1
7  1  1
4  1  1
2  0  1

************************************

访问元素：3
改进clock工作集：
6  0  1
7  1  1
4  1  1
2  0  1

置换位置：1
************************************

访问元素：5
改进clock工作集：
3  1  1
7  1  1
4  1  1
2  0  1
4号页面被修改

置换位置：4
------------------------------------

访问元素：11
改进clock工作集：
13  0  1
10  1  1
11  0  1
12  1  1
3  0  1

************************************

访问元素：12
改进clock工作集：
13  0  1
10  1  1
11  0  1
12  1  1
3  0  1

页面置换次数：16
缺页率0.32
Total time:446ms
```

## 五、页面缓冲算法PBA

### 实验目标：

设立空闲页面链表和已修改页面链表采用可变分配和基于先进先出的局部置换策略，并规定被淘汰页先不做物理移动，而是依据是否修改分别挂到**空闲页面链表**或**已修改页面链表**的末尾空闲页面链表同时用于物理块分配。当已修改页面链表达到一定长度如Z个页面时，一起将所有已修改页面写回磁盘，故可显著减少磁盘I/O操作次数

## 数据结构

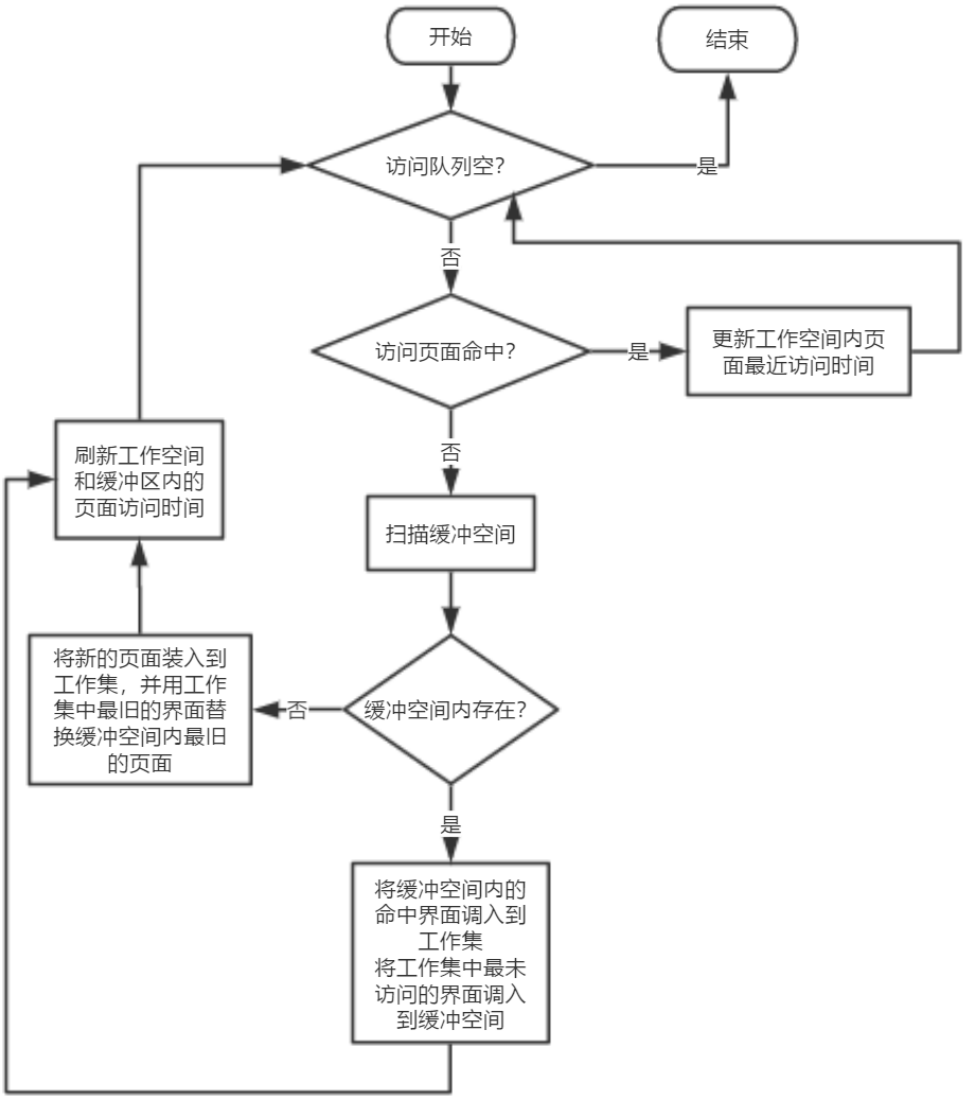
1. 工作集
2. 页面缓冲集
3. 访问序列

## 主要函数

1. 初始化函数：初始化工作集，构造访问序列。
2. 访问序列生成函数：生成有局部特征的随机访问序列。
3. PBA工作集页面修改函数：修改一级工作空间并返回修改页面页号，供缓冲空间页面修改函数调用
4. 缓冲空间页面修改函数：修改二级工作空间并返回修改页面页号，供PBA置换函数调用
5. PBA置换算法：完成PBA置换的主要功能。

## 设计与实现

**PBA置换算法：**




```
//PBA置换算法
void PBA(int wl, int vl) {
    cout << endl << "请输入二级缓存队列大小：";
    int tempw2 = 0;
    cin >> tempw2;
    worklist2 = new int* [tempw2];
    for (int i = 0; i < tempw2; i++) {
        worklist2[i] = new int[2];
        worklist2[i][0] = 0;
        worklist2[i][1] = 0;
    }

    for (int k = 0; k < wl; k++)
    {
        worklist2[k][1] = 214748364;
    }
    double right_count = 0;
    for (int i = 0; i < vl; i++)
    {
```

```
        cout << endl << "访问元素: " << visitlist[i];
        print_worklist(wl);
        int temp = 0;
        for (int j = 0; j < wl; j++)
        {
            if (visitlist[i] == worklist[j][0])
            {
                temp = 1;
                break;
            }
        }
        if (temp == 1)
        {
            cout << endl << "命中! " << endl;
            right_count++;
            for (int k = 0; k < wl; k++)
            {
                worklist[k][1]++;
            }
            continue;
        }
        else
        {
            int temp2 = 0;
            int tempvalue =0;
            for (int j = 0; j < tempw2; j++)
            {
                if (visitlist[i] == worklist2[j][0])
                {
                    temp2 = 1;
                    break;
                }
            }
            if (temp2==1)
            {
                right_count++;
                cout << "未命中但二级队列中存在! " << endl;
                tempvalue = PBA_changew1(visitlist[i], wl);
                PBA_changew2(tempvalue, tempw2);
            }
            else
            {
                cout << "未命中且二级队列中不存在! " << endl;
                tempvalue = PBA_changew1(visitlist[i], wl);
                PBA_changew2(tempvalue, tempw2);
            }

        }
    }
    cout << endl << "页面置换次数: " << vl - right_count;
    cout << endl << "缺页率" << 1 - double(right_count / vl) << endl;
}
```

## 运行结果：

工作集大小：4

访问序列大小：50

二级缓冲空间大小：4

访问序列生成参数如下

```
int N = 20;
int p = 0;
int m = 6;
double t = 0.7;
```

运行结果（部分）如下图：

```
请输入工作集大小：4

请输入访问序列大小：50

是否要改变访问序列生成时的其他参数[0(否)/1(是)]：0
随机访问序列为：
0 5 1 2 0 4 2 5 1 6 7 1 2 4 5 6 3 4 3 7 7 3 4 4 6 9 8 5 9 10 11 12 9 10 5 6 7 10 10 10 11 11 12 8 11 7 7 11 12 9
工作集初始化结果：
0
5
1
2
```

```
************************************
访问元素：5
工作集：
2  2
1  3
7  4
4  1
未命中但二级队列中存在！

置换位置：3
************************************
访问元素：6
工作集：
2  3
1  4
5  1
4  2
未命中但二级队列中存在！

置换位置：2
************************************
访问元素：3
工作集：
2  4
6  1
5  2
4  3
未命中且二级队列中不存在！

置换位置：1
************************************
访问元素：4
工作集：
3  1
6  2
5  3
4  4

命中！

************************************
访问元素：3
工作集：
3  2
```

```
****************************************
访问元素：11
工作集：
8  4
12  5
11  7
7  2

命中！

****************************************
访问元素：12
工作集：
8  5
12  6
11  8
7  3

命中！

****************************************
访问元素：9
工作集：
8  6
12  7
11  9
7  4
未命中且二级队列中不存在！

置换位置：3
页面置换次数：13
缺页率0.26
Total time:2196ms
```

# 六、总结

## 运行结果展示

在同样的参数下

各个置换算法的开销、置换次数、缺页率如下：

| | 置换次数 | 缺页率 | 系统开销 |
|---|---|---|---|
| 最佳置换 | 13 | 26% | 432ms |
| FIFO | 20 | 40% | 326ms |
| LRU | 21 | 42% | 351ms |
| 改进的clock | 25 | 32% | 391ms |
| PBA | 13 | 26% | 2196ms |

## 运行结果分析

从多次的实验中发现：

系统开销最大的是：PBA算法，毕竟PBA算法每次要扫描两次，一次是工作空间，一次是缓存空间。

系统开销最小的是：FIFO置换算法,该算法实现起来最简单，逻辑简单，时间复杂度低，因此系统开销较小。

缺页率最高的是：FIFO和LRU基本持平，都在40%左右。两者考虑的因素都比较少，因此命中率都会偏低一些

缺页率最低的是：最佳置换和PBA算法，实际的操作中PBA通常会比最佳置换多一些，原因是PBA中的二级缓存空间就是一个作弊器，相当于把工作空间扩大了一倍，自然就更容易命中。同样也是因为如此PBA的系统开销要比其他的方式大的多。