201311211914_赵帅帅_实验 6

1、 实验目的

通过编写和调试存储管理的模拟程序以加深对存储管理方案的理解。

2、 实验任务

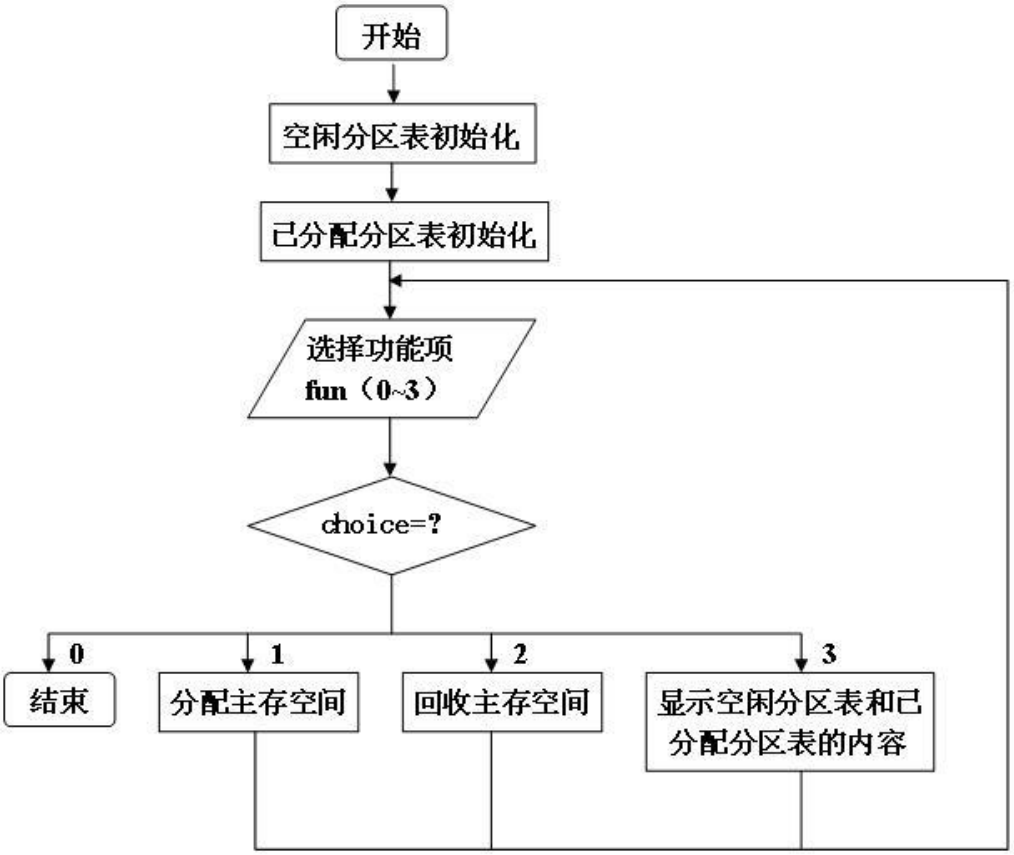
编写并调试一个可变式分区分配的存储管理方案。并模拟实现分区的分配和回收过程。

对分区的分配算法可以是下面三种算法之一：

（1） 首次适应算法

（2） 循环首次适应算法

（3） 最佳适应算法



3、 代码

```
/*
Author zhaossbnu
*/
#include<stdio.h>
#include<stdlib.h>
struct block
{
 int start;
 int len;
 int id;
 struct block *last;
 struct block *next;
}*blank = NULL, *allocated = NULL;

//初始化未分配表和已分配表
void init()
{
 blank = (struct block *)malloc(sizeof(struct block));
 struct block* chushi = (struct block *)malloc(sizeof(struct block));
 blank->start = -1;
 blank->len = -1;
 blank->id = -1;
 chushi->start = 10240;
 chushi->len = 102400;
 chushi->id = 1;
 blank->last = NULL;
 blank->next = chushi;
 chushi->last = blank;
 chushi->next = NULL;

 allocated = (struct block *)malloc(sizeof(struct block));
 allocated->start = -1;
 allocated->len = -1;
 allocated->id = -1;
 allocated->last = NULL;
 allocated->next = NULL;
}

//退出后 释放所有链表内容
void clear()
{
 struct block * current = blank;
```

```
while(current != NULL)
{
    struct block *p = current;
    current = current->next;
    free(p);
}
current = allocated;
while(current != NULL)
{
    struct block *p = current;
    current = current->next;
    free(p);
}
}

//外部菜单显示
int welcome1()
{
int choose = 0;
do
{
    printf("请选择功能(0-退出 1-首次适应 2-循环首次适应 3-最佳适应):");
    scanf("%d", &choose);
    if(choose != 0 && choose != 1 && choose != 2 && choose != 3)
        {
            printf("Error input!!!\n");
        }
}while(choose != 0 && choose != 1 && choose != 2 && choose != 3);
printf("\n");
return choose;
}

//内部菜单显示
int welcome2()
{
int choose =0;
do
{
    printf("选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):");
    scanf("%d", &choose);
}while(choose != 0 && choose != 1 && choose != 2 && choose != 3);
printf("\n");
return choose;
}
```

```
//显示内存分配情况
void print_mem()
{
 printf("空闲分区表如下：\n");
 printf("起始地址 分区长度 标志\n");
 struct block *current = blank->next;
 while(current != NULL)
 {
    printf("%d\t %d\t  %d\n", current->start,current->len, current->id);
     current = current->next;
 }
 printf("\n");
 printf("已分配区表：\n");
 printf("起始地址 分区长度 标志\n");
 current = allocated->next;
 while(current != NULL)
 {
    printf("%d\t %d\t  %d\n", current->start,current->len, current->id);
     current = current->next;
 }
 printf("\n");
}

//回收内存
void recycle()
{
 int id = 0;
 int len = 0;
 printf("输入作业名(请输入数字):");
 scanf("%d", &id);
 struct block *current = allocated->next;
 bool in = false;
 while(current != NULL)
 {
     if(current->id == id)
     {
         in = true;
         break;
     }
     current = current->next;
 }
 if(!in)
 {
```

```
        printf("找不到作业! \n\n");
        return ;
    }
    else
    {
        current->last->next = current->next;  //将当前节点移出已分配表
        if(current ->next != NULL)
        {
            current->next->last = current->last;
        }
        struct block* p = blank->next;
        if(p == NULL)           //未分配表为空时
        {
            current->id = 1;
            p = current;
            current->last = blank;
            blank->next = p;
            current->next = NULL;
            printf("\n");
            return ;
        }
        while(p != NULL)
        {
            //两边都空白
            if(p->start + p->len == current->start && p ->next != NULL &&
current->start + current->len == p->next->start)
            {
                p->len = p->len + current->len + p->next->len;
                struct block * p_a = p->next;
                if(p->next->next != NULL)
                {
                    p->next->next->last = p;
                }
                p->next = p->next->next;
                free(current);
                free(p_a);
                printf("\n");
                return;
            }
            //前面有空白
            if(p->start + p->len == current->start)
            {
                p->len = p->len + current->len;
                free(current);
```

```
            printf("\n");
            return;
        }
        //后面有空白
        if(current->start + current->len == p->start)
        {
            p->start = p->start - current->len;
            p->len += current->len;
            free(current);
            printf("\n");
            return;
        }
        //无空白区与之相邻且在最靠近节点的前面
        if(current->start + current->len < p->start)
        {
            current->id = 1;
            current->next = p;
            current->last = p->last;
            p->last->next = current;
            p->last = current;
            printf("\n");
            return;
        }
        //无空白区与之相邻且在最靠近节点的后面
        if(p->start + p->len < current->start && p->next != NULL &&
current->start + current->len < p->next->start)
        {
            current->id = 1;
            current->last = p;
            current->next = p->next;
            p->next->last = current;
            p->next = current;
            printf("\n");
            return;
        }
        //已经走到表尾
        if(p->next == NULL)
        {
            current->id = 1;
            p->next = current ;
            current->last = p;
            current->next = NULL;
            printf("\n");
            return;
```

```
        }
        p = p->next;
    }
}
}

//首次适应
void first_adapt()
{
init();
int choose = welcome2();
while(true)
{
    switch(choose)
    {
    case 0:
        clear();
        return ;
    case 1:
        {
            int id = 0;
            int len = 0;
            printf("输入作业名(请输入数字):");
            scanf("%d", &id);
            struct block *current = allocated->next;
            bool can_in = true;   //看看作业号是否重复
            while(current != NULL)
            {
                if(current->id == id)
                {
                    can_in = false;
                    break;
                }
                current = current->next;
            }
            if(!can_in)
            {
                printf("分配块 id 重复! \n\n");
            }
            else
            {
                printf("输入作业所需长度 xk:");
                scanf("%d", &len);
                current = blank->next;
```

```
                    //设置分配的块的祥细信息
                    struct block * p = (struct block *)malloc(sizeof(struct
block));
                    p->id = id;
                    p->len = len;
                    p->last = NULL;
                    p->next = NULL;
                    if(current == NULL)
                    {
                        printf("内存已被瓜分完毕! \n\n");
                        break ;
                    }
                    while(current != NULL)
                    {
                        if(current->len == len)     //当前块刚好够分配
                        {
                            //进行分配
                            p->start = current->start;
                            current->last->next = current->next;
                            if(current ->next != NULL)
                            {
                                current->next->last = current->last;
                            }
                            else
                            {
                                current->next = NULL;
                            }
                            free(current);
                            //插入到分配表中
                            struct block *a_current = allocated;
                            while(a_current->next != NULL)
                            {
                                a_current = a_current->next;

                            }
                            a_current->next = p;
                            p->last = a_current;
                            break;
                        }
                        else if(current->len > len) //当前节点大于需要的大
小
                        {
                            //开始分配并修改当前未分配表项的相关信息
                            p->start = current->start + current->len -len ;
```

```
                            current->len -= len;
                            //插入到已分配表中
                            struct block *current = allocated;
                            while(current->next != NULL)
                            {
                                current = current->next;
                            }
                            current->next = p;
                            p->last = current;
                            break;
                        }
                        else
                        {
                            //继续向下寻找
                            current = current->next;
                            if(current == NULL)          //找到表尾仍未找到
说明无法分配
                            {
                                printf("没有这么大的块！\n");
                                break ;
                            }
                        }
                    }
                    printf("\n");
                }
            }
            break;
        case 2:
            recycle();
            break;
        case 3:
            print_mem();
            break;
        default:
            printf("Input Error!\n");
            break;
        }
        choose = welcome2();
    }
    }

    //循环首次适应
    void rotation_first_adapt()
    {
```

```
    init();
    int choose = welcome2();
    struct block* next_rotation = blank->next;   //下一次分配时循环的开始
    while(true)
    {
        switch(choose)
        {
        case 0:
            clear();
            return ;
        case 1:
            {
                int id = 0;
                int len = 0;
                printf("输入作业名(请输入数字):");
                scanf("%d", &id);
                struct block *current = allocated->next;
                bool can_in = true;   //看看作业号是否重复
                while(current != NULL)
                {
                    if(current->id == id)
                    {
                        can_in = false;
                        break;
                    }
                    current = current->next;
                }
                if(!can_in)
                {
                    printf("分配块 id 重复! \n\n");
                }
                else
                {
                    printf("输入作业所需长度 xk:");
                    scanf("%d", &len);
                    //如果上次内存被分完了!!!!!!
                    if(next_rotation == NULL)
                    {
                        next_rotation = current = blank->next;
                    }
                    else
                    {
                        current = next_rotation;     //将当前节点设置为上次
运行的位置
```

```
                    }
                    //设置分配的块的祥细信息
                    struct block * p = (struct block *)malloc(sizeof(struct
block));
                    p->id = id;
                    p->len = len;
                    p->last = NULL;
                    p->next = NULL;
                    if(current == NULL)
                    {
                        printf("内存已被瓜分玩毕! \n\n");
                        break ;
                    }
                    while(current != NULL)
                    {
                        if(current->len == len)     //当前块刚好够分配
                        {
                            //进行分配
                            p->start = current->start;
                            current->last->next = current->next;
                            if(current ->next != NULL)
                            {
                                current->next->last = current->last;
                                //若未到链表结尾,设置下一次的循环节点为下一
节点
                                next_rotation = current->next;
                            }
                            else
                            {
                                //若已到达链表结尾,设置下一次的循环节点为链
表开始
                                next_rotation = blank->next;
                                current->next = NULL;
                            }
                            free(current);
                            //插入到分配表中
                            struct block *a_current = allocated;
                            while(a_current->next != NULL)
                            {
                                a_current = a_current->next;

                            }
                            a_current->next = p;
                            p->last = a_current;
```

```
                printf("\n");
                break;
            }
            else if(current->len > len) //当前节点大于需要的大小
            {
                //开始分配并修改当前未分配表项的相关信息
                p->start = current->start + current->len -len ;
                current->len -= len;
                if(current ->next != NULL)
                {
                    //若未到链表结尾,设置下一次的循环节点为下一节点
                    next_rotation = current->next;
                }
                else
                {
                    //若已到达链表结尾,设置下一次的循环节点为链表开始
                    next_rotation = blank->next;
                }
                //插入到已分配表中
                struct block *current = allocated;
                while(current->next != NULL)
                {
                    current = current->next;
                }
                current->next = p;
                p->last = current;
                printf("\n");
                break;
            }
            else
            {
                //继续向下寻找
                current = current->next;
                if(current == next_rotation)         //到上一次分配的地方仍找不到 说明无法分配
                {
                    printf("没有这么大的块! \n\n");
                    break ;
                }
            }
        }
```

```
            }
        }
        break;
    case 2:
        recycle();
        break;
    case 3:
        print_mem();
        break;
    default:
        printf("Input Error!\n");
        break;
    }
    choose = welcome2();
}
}

/*
最佳适应算法要求未分配空间按从小到大的顺序排列，
但是这会加大编程难度，故而仍采用按地址排序的链表
*/
//最佳适应
void best_adapt()
{
 init();
 int choose = welcome2();
 while(true)
 {
    switch(choose)
    {
    case 0:
        clear();
        return ;
    case 1:
        {
            int id = 0;
            int len = 0;
            printf("输入作业名(请输入数字):");
            scanf("%d", &id);
            struct block *current = allocated->next;
            bool can_in = true;   //看看作业号是否重复
            while(current != NULL)
            {
                if(current->id == id)
```

```
                    {
                        can_in = false;
                        break;
                    }
                    current = current->next;
                }
                if(!can_in)
                {
                    printf("分配块 id 重复! \n\n");
                }
                else
                {
                    printf("输入作业所需长度 xk:");
                    scanf("%d", &len);
                    current = blank->next;
                    //设置分配的块的祥细信息
                    struct block * p = (struct block *)malloc(sizeof(struct
block));
                    p->id = id;
                    p->len = len;
                    p->last = NULL;
                    p->next = NULL;
                    if(current == NULL)
                    {
                        printf("内存已被瓜分完毕! \n\n");
                        break ;
                    }
                    struct block * min = blank->next;
                    while(min != NULL)
                    {
                        if(min->len >= len)
                        {
                            break;
                        }
                        if(min == NULL)
                        {
                            printf("没有那么大的块! \n\n");
                            break;
                        }
                        min = min->next;
                    }
                    if(min == NULL)
                    {
                        break;
```

```
                    }
                    current = min;
                    while(current != NULL)
                    {
                        if(current->len == len)     //当前块刚好够分配
                        {
                            //进行分配
                            p->start = current->start;
                            current->last->next = current->next;
                            if(current ->next != NULL)
                            {
                                current->next->last = current->last;
                            }

                            free(current);
                            //插入到分配表中
                            struct block *a_current = allocated;
                            while(a_current->next != NULL)
                            {
                                a_current = a_current->next;
                            }
                            a_current->next = p;
                            p->last = a_current;
                            printf("\n");
                            break;
                        }
                        else
                        {
                            //继续向下寻找
                            current = current->next;
                            if(current != NULL && current->len >= len &&
current->len < min->len)
                            {
                                min = current;
                                    continue;
                            }
                            if(current == NULL)
                            {
                                //开始分配并修改当前未分配表项的相关信息
                                p->start = min->start + min->len -len ;
                                min->len -= len;
                                //插入到已分配表中
                                struct block *current = allocated;
                                while(current->next != NULL)
```

```
                            {
                                current = current->next;
                            }
                            current->next = p;
                            p->last = current;
                            printf("\n");
                            break;
                        }
                    }
                }
            }
        }
        break;
    case 2:
        recycle();
        break;
    case 3:
        print_mem();
        break;
    default:
        printf("Input Error!\n");
        break;
    }
    choose = welcome2();
}

}

int main(void)
{
 int choose = welcome1();
 while(true)
 {
    switch(choose)
    {
    case 0:
        printf("谢谢使用~\n");
        return 0;
    case 1:
        first_adapt();
        break;
    case 2:
        rotation_first_adapt();
        break;
```

```
        case 3:
            best_adapt();
            break;
        default :
            printf("Error input!!!\n");
            break;
        }
        choose = welcome1();
    }
    return 0;
}
```

4、 运行结果

首次适应：

请选择功能(0-退出 1-首次适应 2-循环首次适应 3-最佳适应):1

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):1
输入作业所需长度 xk:1000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):2
输入作业所需长度 xk:2000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):3
输入作业所需长度 xk:3000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):4
输入作业所需长度 xk:96400

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):2

输入作业名(请输入数字):1

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志

111640 1000 1

已分配区表：
起始地址 分区长度 标志
109640 2000 2
106640 3000 3
10240 96400 4

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):2

输入作业名(请输入数字):3

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志
106640 3000 1
111640 1000 1

已分配区表：
起始地址 分区长度 标志
109640 2000 2
10240 96400 4

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志
106640 3000 1
111640 1000 1

已分配区表：
起始地址 分区长度 标志
109640 2000 2
10240 96400 4

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):2

输入作业名(请输入数字):4

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志

10240 99400 1
111640 1000 1

已分配区表：
起始地址 分区长度 标志
109640 2000 2

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):2

输入作业名(请输入数字):2

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志
10240 102400 1

已分配区表：
起始地址 分区长度 标志

循环首次适应：

请选择功能(0-退出 1-首次适应 2-循环首次适应 3-最佳适应):2

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):1
输入作业所需长度 xk:1000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):2
输入作业所需长度 xk:2000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):3
输入作业所需长度 xk:3000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):4
选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):4
输入作业所需长度 xk:4000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):5
输入作业所需长度 xk:92400

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志

已分配区表：
起始地址 分区长度 标志
111640 1000 1
109640 2000 2
106640 3000 3
102640 4000 4
10240 92400 5

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):2

输入作业名(请输入数字):5

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):2

输入作业名(请输入数字):3

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志
10240 92400 1
106640 3000 1

已分配区表：
起始地址 分区长度 标志
111640 1000 1
109640 2000 2
102640 4000 4

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):3
输入作业所需长度 xk:1000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志
10240 91400 1
106640 3000 1

已分配区表：
起始地址 分区长度 标志
111640 1000 1
109640 2000 2
102640 4000 4
101640 1000 3

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):5
输入作业所需长度 xk:2000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志
10240 91400 1
106640 1000 1

已分配区表：
起始地址 分区长度 标志
111640 1000 1
109640 2000 2
102640 4000 4
101640 1000 3
107640 2000 5

最佳适应：

请选择功能(0-退出 1-首次适应 2-循环首次适应 3-最佳适应):3

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):1
输入作业所需长度 xk:1000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):2
输入作业所需长度 xk:2000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):3
输入作业所需长度 xk:3000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):4
输入作业所需长度 xk:4000

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):5
输入作业所需长度 xk:92400

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志

已分配区表：
起始地址 分区长度 标志
111640 1000 1
109640 2000 2
106640 3000 3
102640 4000 4
10240 92400 5

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):2

输入作业名(请输入数字):1

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):2

输入作业名(请输入数字):3

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):2

输入作业名(请输入数字):5

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：

起始地址 分区长度 标志
10240 92400 1
106640 3000 1
111640 1000 1

已分配区表：
起始地址 分区长度 标志
109640 2000 2
102640 4000 4

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):3
输入作业所需长度 xk:500

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志
10240 92400 1
106640 3000 1
111640 500 1

已分配区表：
起始地址 分区长度 标志
109640 2000 2
102640 4000 4
112140 500 3

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):1

输入作业名(请输入数字):1
输入作业所需长度 xk:500

选择功能项(0-退出 1-分配主存 2-回收主存 3-显示主存):3

空闲分区表如下：
起始地址 分区长度 标志
10240 92400 1
106640 3000 1

已分配区表：
起始地址 分区长度 标志
109640 2000 2

102640	4000	4
112140	500	3
111640	500	1

5、 实验感想

实验的算法没有什么困难，若是用数组写的话代码很短，但是用链表来写的话代码确实是有点长。