数据结构实验三

姓名：刘俊傲　　学号：**U201617047**　　班级：软工**1603**班

1. Josephus Problem

1. 问题描述:

   > 用游标方式的循环链表的方式实现Josephus(n, m)问题的求解过程

2. 问题分析与算法设计

   > 1. 问题分析：通过构建循环链表的方式，来不断的删除相应的节点而最终获得结果。
   > 2. 算法设计：通过计算链表的剩余元素个数并与M相比较，不停的循环累加，知道与M相等，然后清除该位置上面的节点元素。然后不停的重复上面的步骤。如果M=N-1,则最坏的算法复杂度为NlogN。

3. 实验步骤和方案

   > 1. 首先定义create_list(int size)函数来创建循环链表
   > 2. 定义int型变量n和m并通过scanf()函数对变量n和m赋值；
   > 3. 定义方法josephus(int n,int m)方法来求解问题；

4. 算法实现

```
#include<stdio.h>
#include<stdlib.h>

typedef struct nList
{
    int data;
    struct nList *next;
} LIST;

static void josephus(int n, int m);
static LIST *create_list(int size);

static LIST *create_list(int size)
{
    LIST *list, *ptr, *tmp = NULL;
    int i;

    list = (LIST *)malloc(sizeof(LIST));

    if (list == NULL)
    {
        printf("malloc () faile\n");
        return (NULL);
    }

    for (i = 0; i < size; i++)
    {
        ptr = (LIST *)malloc(sizeof(LIST));

        if (ptr == NULL)
        {
            printf("malloc () failed\n");
            return (NULL);
        }
        ptr->data = size - i;
        ptr->next = tmp;
        tmp = ptr;
        if (i == 0)
        {
            tmp->next = list;
        }
    }
    list->next = tmp;

    return (list);
}

void josephus(int n, int m)
{
    LIST *list = create_list(n);

    while (n > 1)
    {
```

```
        LIST *ptr;
        for (int i = 0;; i++)
        {
            ptr = list->next;
            if (i == m)
            {
                printf("%d->", list->data);
                free(list);
                n--;
                break;
            }
            list = ptr;
        }
    }
}

void main()
{
    int n = 0, m = 0;
    printf("please enter N: ");
    scanf("%d\n", &n);
    printf("please enter M: ");
    scanf("%d\n", &m);

    josephus(n, m);
    getchar();
}
```

5. 结果分析

> 通过不断的循环链表来删除相应的链表值可以最终达到最终结果。在相同的N值的情况下，M的值越接近N，则遍历链表的次数就相应越多，因而效率就越低，而最坏情况下的时间复杂度为NlogN;

2. 多项式乘法

1. 问题描述：

> 用链表表示多项式，分别在对指数排序和不排序的情况下，写出求两个给定多项式的乘法的函数

2. 问题分析与算法设计：

> 比较不对指数排序和对指数排序的算法复杂度，并对结果进行分析

3. 实验步骤和方案：

> 1. 首先首先定义 LIST* create()来创建相应的链表来存储相应的多项式
> 2. 接着定义sort(LIST* list)来对多项式进行指数排序
> 3. 定义int mutiply(int x,LIST* list)进行计算
> 4. 分别求出相应的时间复杂度

4. 算法实现：

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>

typedef struct nLIST
{
    int xishu;
    int power;
    struct nLIST *next;
}LIST;

static LIST* create();
static LIST* sort(LIST* list);
static int mutiply(int x,LIST* list);

//储存多项式的链表
LIST* create()
{
    int count;
    printf("please input the count of the list:");
    scanf("%d\n", &count);
    if (count < 1)
    {
        printf("the input is not invalid,please input again: ");
        scanf("%d\n", &count);
    }

    LIST *newList, *tmp = NULL;

    while (count > 0)
    {
        newList = (LIST*)malloc(sizeof(LIST));
        if (newList == NULL)
        {
            printf("malloc is faile\n");
            return NULL;
        }
        printf("please input the xishu: ");
        scanf("%d\n", &newList->xishu);
        printf("please input the power: ");
        scanf("%d\n", &newList->power);

        newList->next = tmp;
        tmp = newList;
        --count;
    }
    return tmp;
}

//多链表进行指数排序
LIST* sort(LIST* list)
{
    LIST *ptr = NULL,*tmp = NULL;
```

```
    for (int i = 0; list->next != NULL; i++)
    {
        for (int j = 0; list->next != NULL; j++)
        {
            ptr = list->next;

            if (list->power < ptr->power)
            {
                tmp = list;
                list = ptr;
                ptr = tmp;
            }
        }
        list = list->next;
    }

    return list;
}

//进行结果的计算
int mutiply(int x,LIST* list)
{
    int sum = 0;
    while (list != NULL)
    {
        int powerNum = 1;
        for (int i = list->power; i > 0; i--)
        {
            powerNum *= x;
        }
        sum += list->xishu * powerNum;

        list = list->next;
    }

    return sum;
}

void main()
{
    LIST *list = create();
    int sum = mutiply(2, list);
    printf("%d", sum);
    getchar();
}
```

5. 结果分析：

    1. 如果直接对多项式进行计算，则

    ```
    O(N) = O(power + 1) + O(N - 1) + O(N)
         = O(N)
    ```

> 2. 如果对指数进行排序，则
>
> ```
> O(N) += O(NlogN)
>      = O(NlogN)
> ```
>
> 因此用链表表示多项式并进行相应的计算时，不排序的时间复杂度更低，效率更高