递归分治排序

特别注意：OJ里的输出最好都用long long，防止溢出。

递归与分治

汉诺塔问题

1. 基础汉诺塔问题

- 题目链接：基础汉诺塔问题
- 采用**递归**方式写汉诺塔即可，注意传入**参数的设置。**

  参考代码如下：

```
#include<stdio.h>
int n;
void hanoi(char A,char B,char C,int n)
{
    if(n == 1)
    {
        move(A,C);
    }
    else{
        hanoi(A,C,B,n-1);
        move(A,C);
        hanoi(B,A,C,n-1);
    }
```

```
}
void move(char x,char y)
{
    printf("%c to %c\n",x,y);
}
int main()
{
    while(~scanf("%d",&n))
    {
        hanoi('A','B','C',n);
        puts("");
    }
    return 0;
}
```

2. 汉诺塔变形

递归找规律问题

- 对于陌生的找规律问题，有步骤如下：
  1. 首先先**枚举**出前几项，在枚举出来的前几项中找规律
  2. 找到规律后，利用递归分析写出其**递归表达式**
  3. 利用**数组**计算递归结果，把结果存在相应的**数组下标**的数组中（如果直接递归求解会超时）
  4. 如果发现**查询的数值过大**（一般会说利用取模防止溢出），打印数组，找到循环节，再用**查询的数组对循环节取模**

1.斐波那契数列

- 题目链接：斐波那契数列

  $F(n) = F(n-1) + F(n-2)$

- **经验积累**：如果是斐波那契数列对100007取模，其**循环节为20188。**

  参考代码如下：

```
#include<stdio.h>
#define MOD 100007
int a[MOD*100];
long long n,i;
int main()
{
    a[0] = 1;
    a[1] = 2;
    a[2] = 3;
    a[3] = 5;
    for(i=4;i<=MOD*100;i++)
    {
        a[i] = ((a[i-1] % MOD) + (a[i-2] % MOD)) % MOD;
    }
    while(~scanf("%lld",&n))
    {
        printf("%lld\n",a[n%20188]);
```

```
    }
    return 0;
}
```

2.123数列

- 蒙德城吃鱼大赛
- 方法1：一维数组

  $F[n]=F[n-1]+F[n-2]+F[n-4]+F[n-5]$

```
#include<stdio.h>
int main() {
    long long n,a[10000];a[1]=1;a[2]=2;a[3]=4;a[4]=7;a[5]=13;a[6]=23;
    for(int i=7;i<=50;i++)  a[i]=a[i-1]+a[i-2]+a[i-4]+a[i-5];
    while(scanf("%lld",&n)!=EOF)
    printf("%lld\n",a[n]);
    return 0;
}
```

- 方法2：二维数组
- 要吃 n 条鱼的方法总数，就是吃 n−1条鱼的方法总数、吃 n−2条鱼的方法总数和吃 n−3条鱼且上一次没有吃 3 条鱼的方法总数的和。

  因此可以用一个二维数组分别保存吃n条鱼的时候上一次吃 3条鱼和没有吃 3 条鱼的方法数。

  注意：在 n 大于 45 的时候，方法总数会超过 `int` 的最大值，因此，需要设为 `long long` 型变量。

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

  int n;
  long long a[51][2];

  int main() {
    int i;

    a[1][0] = 1;
    a[2][0] = 2;
    a[3][0] = 3;
    a[1][1] = 0;
    a[2][1] = 0;
    a[3][1] = 1;
    for (i = 4; i <= 50; i++) {
        a[i][0] = a[i - 1][0] + a[i - 2][0] + a[i - 1][1] + a[i - 2][1];
        a[i][1] = a[i - 3][0];
    }
    while (~scanf("%d", &n)) {
        printf("%lld\n", a[n][0] + a[n][1]);
    }

    return 0;
  }
```

3. 其他递归找规律问题

逆序数问题

- 利用好**归并排序的基础代码**就可以了，只需要做稍微的改动，同时记住**输出最好用long long型**
- Inverse number：Reborn
- 模式寻对

  参考代码如下：

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>
int a[1000002],n;
long long num;
void merge(int *a,int p,int q,int r)
{
    int n1 = q-p+1;
    int n2 = r-q;
    int L[n1],R[n2],i,j,k;
    for(i=0;i<n1;i++)
        L[i] = a[p+i];
    for(i=0;i<n2;i++)
        R[i] = a[q+i+1];
    for(i = j = 0,k = p;k<=r;k++){
        if(i == n1)
            a[k] = R[j++];
        else if (j == n2)
            a[k] = L[i++];
        else if(L[i]<=R[j])
            a[k]=L[i++];
        else{
            a[k]=R[j++];
            num += n1 - i;/****关键位置****/
        }
    }
}
void merge_sort(int *a,int p, int r)
{
    if(p<r)
    {
        int q = (p + r) / 2;
        merge_sort(a,p,q);
        merge_sort(a,q+1,r);
        merge(a,p,q,r);
    }
}
int main()
{
    while(~scanf("%d",&n))
    {
        num = 0;
        memset(a,0,sizeof(a));
```

```
        for(int i = 0;i<n ;i++)
        {
            scanf("%d",&a[i]);
        }
        merge_sort(a,0,n-1);
        printf("%lld\n",num);
    }
    return 0;
}
```

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>
int a[1000002],n;
long long num;

void merge(int x[ ],int tmp[ ],int left,int leftend,int rightend)
{
    int i=left, j=leftend+1, q=left;
    while(i<=leftend && j<=rightend)
    {
        if(x[i]<=x[j])
            tmp[q++]=x[i++];
        else
        {
            tmp[q++]=x[j++];
            num += leftend  + 1 - i ;/****关键位置****/
        }

    }
        while(i<=leftend)
            tmp[q++]=x[i++];
        while(j<=rightend)
            tmp[q++]=x[j++];
        for(i=left; i<=rightend; i++)
           x[i]=tmp[i];

}
void mSort(int k[], int tmp[], int left, int right)
{
    int center;
    if(left < right){
        center = (left+right)/2;
        mSort(k, tmp, left, center);
        mSort(k, tmp, center+1, right);
        merge(k, tmp, left, center, right);
    }
}
void mergeSort(int k[ ],int n)
{
    int *tmp;
    tmp = (int *)malloc(sizeof(int) * n);
    if(tmp != NULL) {
        mSort(k, tmp, 0, n-1);
        free(tmp);
    }
```

```
}
int main()
{
    while(~scanf("%d",&n))
    {
        num = 0;
        memset(a,0,sizeof(a));
        for(int i = 0;i<n ;i++)
        {
            scanf("%d",&a[i]);
        }
        mergeSort(a,n);
        printf("%lld\n",num);
    }
    return 0;
}
```

多项式问题（霍纳法则Horner Rule)

霍纳法则递归解释

- **霍纳法则**(Horner Rule)是采用最少的乘法运算策略，来求多项式在x处的值 .
- 一般的求解公式如下图所示，看起来使用math.h中的pow很快，其实如果题目要求取模的时候，**pow不会自动取模，极大可能溢出。**如果**手写pow**确实可以实现加法乘法都取模防止溢出，但是运算的时间为O(n*n),**一般都会TLE。**

  $A(x) = a_n x^n + a_{n-1} x^{(n-1)} + \ldots + a_1 x + a_0$

  这时候采用**Horner Rule**，所有问题都会迎刃而解，该规则如下图所示：

  $A(x_0) = (\ldots((a_n x_0 + a_{n-1})x_0 + \ldots + a_1)x_0 + a_0)$

  痛过分解我们注意到，从右往左来看，每一个小式子都是如此：

  $something * x_0 + a_i$

  所以不难得到递归公式：

  $Horner(a, n, x) = a[0] + Horner(a + 1, n - 1, x)$

- 零崎的人间冒险Ⅲ
- 多项式计算器

  参考代码如下：

```
#include<stdio.h>
#include<math.h>
#define M 1000007
long long a[20000000];
long long n,x,i;
long long Horner(long long *a,long long n,long long x)
{
    if(n<1)
        return a[0] % M;
    else
        return (a[0] % M + (Horner(a+1,n-1,x) * x) % M) % M;
}
int main()
```

```
{
    while(~scanf("%lld%lld",&n,&x))
    {
        for(i=0;i<=n;i++)
        {
            scanf("%lld",&a[i]);
        }
        printf("%lld\n",Horner(a,n,x));
    }
    return 0;
}
```

博弈问题

基本博弈思想

- Let's play a game
- 博弈论题型的思想：
    1. 如果有一个能**使对手进入必败状态**的取数字方法，则**我方必胜**
    2. 当我方拿到的条件**上手就直接输掉**，或者**我方的下一步怎么走**，**对方都必胜的话**，则**我方必败**

参考代码如下：

```
#include<stdio.h>
#define win 1
#define lose 0
int a,b,cnt;
int judge(int a,int b)
{
    if(a<b){    //默认b是最小的那个
        int tmp = a;
        a = b;
        b = tmp;
    }
    if (b==0)   //如果此时这个人拿到了0，就说明这个人必然会输
        return lose;
    for(int i=a/b;i>0;i--)
    {           //看一下下一层对方会如何，如果对方有一种必败的可能性，则我方必胜
        if(judge(a - b * i,b)==lose)
            return win;
    }
    return lose;
}
int main()
{
    while(~scanf("%d%d",&a,&b))
    {
        if(judge(a,b))
            printf("Nova\n");
        else
            printf("LaoWang\n");
    }
    return 0;
}
```

威佐夫博弈根号5

- **威佐夫博弈**（Wythoff's game）是指的这样一个问题：有两堆各若干个物品，两个人轮流从任意一堆中取出至少一个或者同时从两堆中取出同样多的物品，规定每次至少取一个，至多不限，最后取光者胜利。
- 我们可以来找找那些先手必输局势的规律

  第一种（0，0）

  第二种（1，2）

  第三种（3，5）

  第四种（4，7）

  第五种（6，10）

  第六种（8，13）

  第七种（9， 15）

  第八种（11，18）

  第n种 （a[k]，b[k]）

  我们把这些局势称为“奇异局势”

  我们会发现他们的差值是递增的，分别是0,1,2,3,4,5,6,7.....n，同时**我们用数学方法分析发现这些局势的第一个值是未在前面出现过的最小的自然数。**
  所以结论为：

  $a[k] = (int)\ (\ (b[k] - a[k]) * (sqrt(5.0) + 1)/2)$

卡特兰数（Catalan数）

- n个节点的二叉树有多少种形态问题
- Zexal的二叉树（签到）
- 递归思路：只固定根节点，假设**左子树有x个结点，你们右子树则有n-x-1个结点**，设n个结点的二叉树有f(n)，种形态，不妨**定义f(0)=1**,则有递推关系：

$$f(n) = \sum_{x=0}^{n-1} f(x) * f(n-x-1)$$

- 递归关系的解为：

  $f(n) = (2n)!/n!(n+1)!$

  参考代码如下：

```
#include<stdio.h>
long long a[20000000];
long long i,j,n;
int main(){
    a[0]=1;
    a[1]=1;
    a[2]=2;
    a[3]=5;
    for(i=4;i<31;i++)
    {
        for(j=0;j<i;j++)
        {
```

```
            a[i]+=a[j]*a[i-1-j];
        }
    }
    while(~scanf("%lld",&n))
    {
        printf("%lld\n",a[n]);
    }
    return 0;
}
```

排序

堆排序

- 零崎的人间冒险IV
- 合理利用好STL里面的堆排序算法即可，注意要使用vector容器

```
#include <iostream>
#include<algorithm>
#include <vector>
using namespace std;
int n,tmp;
vector<int> v;
int main()
{
    while(~scanf("%d",&n))
    {
        for(int i=0;i<n;i++)
        {
            scanf("%d",&tmp);
            v.push_back(tmp);
        }
        make_heap(v.begin(),v.end());
        sort_heap(v.begin(),v.end());
        for(int i=0;i<v.size();i++)
        {
            printf("%d ",v[i]);
        }
        puts("");
        v.clear();
    }
}
```

优先队列

优先队列应用

Zexal的电影院

jhljx水水的补习班

- top 访问队头元素
- empty 队列是否为空

- size 返回队列内元素个数
- push 插入元素到队尾 (并排序)
- emplace 原地构造一个元素并插入队列
- pop 弹出队头元素
- swap 交换内容
- 升序队列 priority_queue <int,vector,greater > q;
- 降序队列 priority_queue <int,vector,less >q;
- 对结构体排序

```
struct cmp    //另辟struct，排序自定义
{
    bool operator () (const student & a,const student & b) const
    {
        if(a.sc != b.sc)    return b.sc > a.sc;
        else if(a.g != b.g) return a.g > b.g;
        else return a.s > b.s;
    }
};
```