2016级算法第一次上机解题报告

16211017 仇善召

## 总结

本次上机主要考查了递推、简单数论、分治思想。难度不高，能不能做出主要在于有没有推断出这个题目的模型。

Stupid owls和AlvinZH去图书馆两题主要考察了递推思想，根据基础的结果来推断更复杂的情况，实质上是递归。这两题均采用多组输入，因此可以先将范围内的所有结果储存在数组里，这样可以节省时间。ModricWang和数论考察了基础数论，可以通过找规律解决。水水的Horner Rule主要考察了进制转换。ModricWang's QuickSort和D&C--玲珑数考察了对分治思想中的快排和归并的理解。

## 解题报告

A The stupid owls

**思路及算法分析：**

主要是错排递推式的推导。错排的定义：一段序列中一共有n个元素，那么可知这些元素一共n!种排列方法。假如在进行排列时，原来所有的元素都不在原来的位置，那么称这个排列为错排。而错排数所指的就是在一段有n个元素的序列中，有多少种排列方式是错排。

递归关系：D(n)=(n-1)(D(n-1)+D(n-2)) 特别地有D(1)=0,D(2)=1。

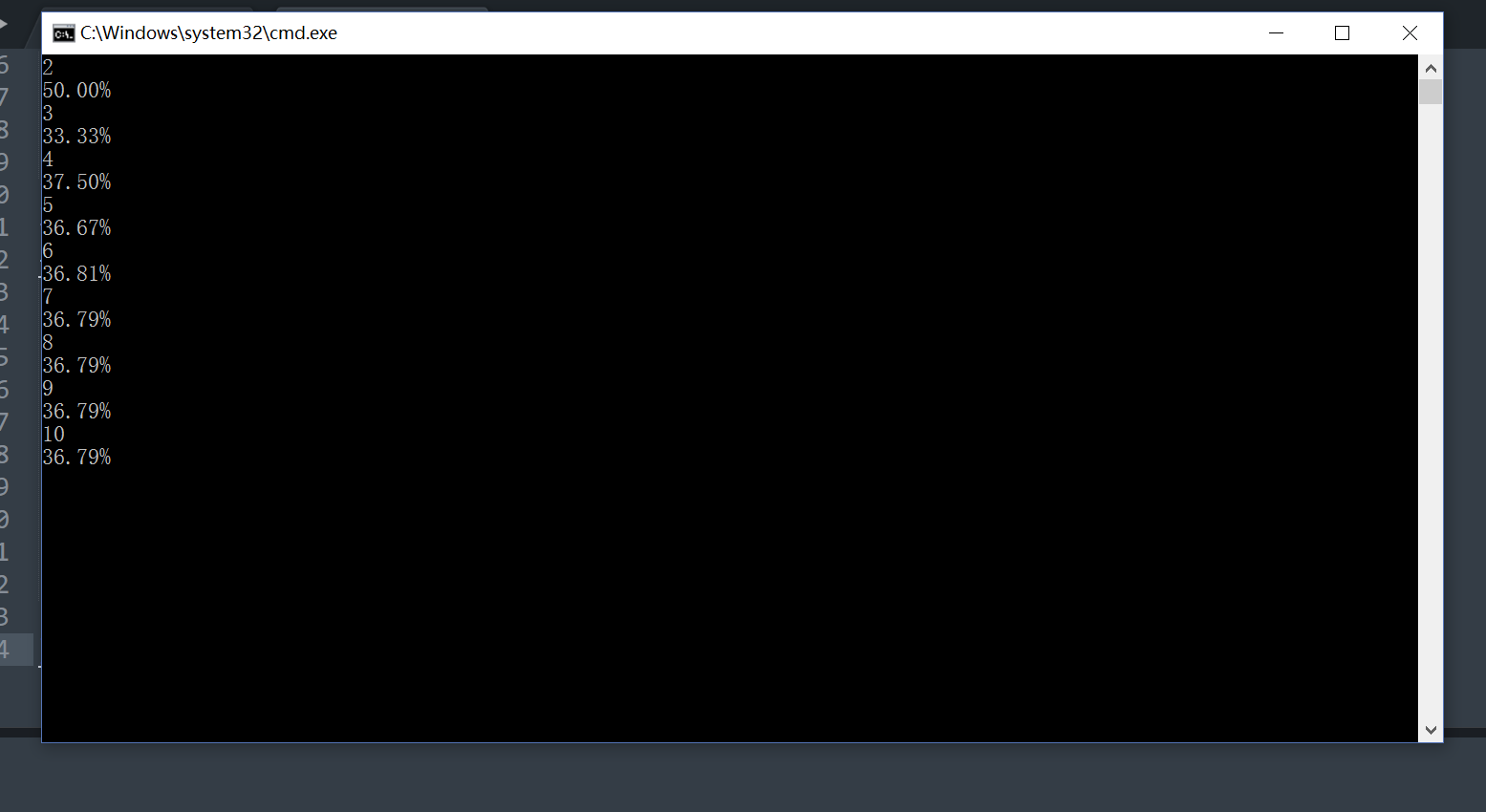
下面以自己的理解给出递推式的简明推导：

第一步：取这n个元素的第一个元素，它放错的方式有n-1种。放错那个位置的原来的元素记为k。

第二步：下面错排剩余的n-1个元素。因为K是特殊的，k的放置有两种可能。（1）假如k恰好错排在第一个元素的位置，那么此时可以将第一个位置和k原来所在的位置移除，将问题的规模减为D（n-2），变成了n-2个数的错排问题。（2）假如k放置在除第一个之外的其他位置，那么先将放在k位置的原本第一个元素剔除，这样只剩n-1个元素。把k放在第一个位置，由于这种情况k不在第一个位置，因此问题退化为D(n-1),变成了n-1个数的错排问题。

由我们学过的乘法公式和加法公式得到错排数计算的递推式：D(n)=(n-1)(D(n-1)+D(n-2)) 特别地有D(1)=0,D(2)=1。

参考资料：<http://blog.csdn.net/passion_acmer/article/details/51023275>

除此之外，由于题目精度的特殊性，只有两位精度，当输入越过某个值，答案固定，因此也可以打表输出。。

参考代码:

#include <cstdio>

#include <algorithm>

#include <queue>

#include <math.h>

double f(int n)

{

if ( n==1 )

return 0;

if ( n==2 )

return 1;

else

return (n-1)\*(f(n-1)+f(n-2));

}

double frab(double n)

{

double sum=1;

for ( double i=n; i>=1; i--)

sum=sum\*i;

return sum;

}

int main()

{

int n;

double a[26];

for ( int i=1; i<=25; i++)

a[i]=f(i);

while (~scanf("%d",&n))

{

double m=a[n];

double b=frab(n);

printf("%.2lf%%\n",m/b\*100);

}

}

**B ModricWang和数论**

**思路及算法分析：**

对一个数取模的结果最多有多少种？

假设这个数为n，那么取模的结果不存在大于n/2（奇数偶数都一样）的除了他自己以外的数。例如对于100这个数，50，51，52…99均不可以作为它的取模结果。因为2\*（100-51）<100，即用51相对于100的补码——49取模，结果必不为51。而小于n/2的数字均可以作为取模结果，包括0。故可以推出规律，结果为（n+3）/2。算法复杂度为O（1）

**参考代码：**

**#include <cstdio>**

**int main()**

**{**

**long long n;**

**scanf("%lld",&n);**

**printf("%lld\n",(n+3)/2);**

**}**

**C AlvinZH去图书馆**

**思路及算法分析：**

递推的思想，特殊之处在于连跳三个的情况不能连续进行两次。

因此如果某一个位置是通过连跳三次达到的，那么上上次运动一定是通过单跳或者双跳。

因此用两个数组，a[n],b[n],

两个数组的含义分别是：a[i]的含义是到达i位置最后一次运动是单跳或者双跳的情况的数量。b[i]的含义是到达i位置最后一次运动是三跳的情况。显然b[i]=a[i-3],因为连跳三个不能连续进行。a[i]=a[i-1]+a[i-2]+b[i-1]+b[i-2], i可能由i-1得到也可能由i-2得到，而对于i-1和i-2来说，他们的方法数等于a与b之和。

因此得到递推关系b[i]=a[i-3]，a[i]=a[i-1]+a[i-2]+b[i-1]+b[i-2]，我们记a[0]=1 a[1]=1 a[2]=2 b[0]=b[1]=b[2]=0 定义递推基本情况。

参考代码：

#include <iostream>

#include <string>

#include <cstring>

#include <math.h>

using namespace std;

long long a[51],b[51];

int main()

{

a[0]=1;

a[1]=1;

a[2]=2;

b[0]=b[1]=b[2]=0;

for ( int i=3; i<=50; i++)

{

b[i]=a[i-3];

a[i]=a[i-1]+a[i-2]+b[i-1]+b[i-2];

}

int n;

while(cin>>n)

{

cout<<a[n]+b[n]<<endl;

}

}

递推拓展：

递推的定义：给定一个数的序列H0,H1,…,Hn,…若存在整数n0，使当n>n0时,可以用等号(或大于号、小于号)将Hn与其前面的某些项Hi(0<i<n)联系起来，这样的式子就叫做递推关系。

递推算法是一种简单的算法，即通过已知条件，利用特定关系得出中间推论，直至得到结果的算法。

递推算法分为顺推和逆推两种。

相对于递归算法,递推算法免除了数据进出栈的过程，也就是说,不需要函数不断的向边界值靠拢,而直接从边界出发,直到求出函数值.

递推算法的经典例子：

【案例】从原点出发，一步只能向右走、向上走或向左走。恰好走N步且不经过已走的点共有多少种走法？

样例输入：N=2

样例输出：result=7

样例输入：N=3

样例输出：result=17

与去图书馆这道题类似，通过分析，我们可以得到递推式：f(n)=3\*f(n-2)+2\*(f(n-1)-f(n-2) ) (n>=3)

参考资料：<http://blog.csdn.net/nuc_sheryl/article/details/51344141>

**D 水水的Horner Rule**

**思路及算法分析：**

本题思路较简单，考查由任意进制转化成十进制，难点在于输入的x为十进制的int范围，换算成二进制最多有31位，超出int和long long的数值范围，因此要采用string读入。复杂度O（n）

参考代码：

#include <iostream>

#include <string>

#include <cstring>

#include <math.h>

using namespace std;

long long convet(int H, string a)

{

long long y=0;

for ( int i=0; i<a.length(); i++)

{

y=y+pow(H,a.length()-i-1)\*(a[i]-'0');

}

return y;

}

int main()

{

int n;

cin>>n;

while ( n-- )

{

int a1;

string a;

int b1;

string b;

cin>>a1>>a;

cin>>b1>>b;

cout<<convet(a1,a)+convet(b1,b)<<endl;

}

}

**E ModricWang's QuickSort**

思路及算法分析：

考查对分治思想——快排的理解。此题与普通快排不一样之处在于基准元素选择的是中间，如果盲目套板子可能会崩。。

只需要将快排的划分函数按照题目要求写好，为了加深自己对快排的理解我要将快排的思路口述下来。普通快排首先要进行一次划分，在这次划分中，先选取最低位作为基准元素，我们要做的是将所有比基准元素小的数字放在它的左边，把所有比基准元素大的数字放在它的右边。首先，设置两个指针，我们先从高位开始寻找，直到找到比基准元素小的数字，swap交换一次，基准元素的位置改变。我们再从低位开始寻找，直到找到比基准元素大的数字停止，swap交换一次。循环的条件式low<high，当这次划分结束时，low=high=基准元素位置。然后依次为界递归求解。

本题只需要调用两次划分函数，得到两个中界。

参考代码：

#include <iostream>

#include <string>

#include <cstring>

#include <math.h>

using namespace std;

int n;

int a[1000005];

int Partition(int a[],int low,int high)

{

//a[0]=a[low];

int privotkey=a[(high+1)/2];

while(low<high){

while ( low<high && a[low]<=privotkey)

++low;

while ( low<high && a[high]>privotkey)

--high;

if ( low < high )

swap(a[low],a[high]);

}

//a[low]=a[0];

return low;

}

void QSort(int a[],int s,int t)

{

if ( s<t )

{

int midloc=Partition(a,s,t);

QSort(a,s,midloc-1);

QSort(a,midloc+1,t);

}

}

int main()

{

cin>>n;

for ( int i=0; i<n; i++)

{

cin>>a[i];

}

int mid=Partition(a,0,n-1);

int mid2=Partition(a,0,mid);

for ( int i=mid2; i<mid; i++)

{

cout<<a[i]<<" ";

}

// cout<<m;

}

拓展：

int Partition(int a[],int low,int high)

{

int privotkey=a[low];

while(low<high)

{

while ( low<high && a[high]>=privotkey)

--high;

swap(a[low],a[high]);

while ( low<high && a[low]<=privotkey)

++low;

swap(a[high],a[low]);

}

//a[low]=a[0];

return low;

}

普通快排以最低位为基准元素划分。

**G D&C--玲珑数**

**思路及算法分析：**

Hint已经提示，采用暴力n^2做会超时，采用nlg（n）

在说这道题之前，先讲一下算法热身G题模式寻对，即寻找逆序对的数量，两题很类似，均采用了归并排序进行求解。（循环不变式）主要思路是，在归并划分到底进行合并时，两个序列都已经是有序的序列了，用i，mid，j分别表示第一个序列的指针，中间位置，和第二个序列的指针。

（1）当i<j&&a[i]<=a[j]时，a[i]出，i++，没有逆序数产生；

（2）当i<j&&a[i]>a[j]时，a[j]出，j++，有逆序数产生，因为两个子序列均已经有序，对a[j]来说，i-mid中间所有的元素都比它大，因此产生逆序数，mid-i+1.

注意的是每次查询都会改变数组元素的位置，因此需用第二个数组来当“替身”。

对于这道题情况高度类似，只是产生逆序的情况变为当i<j&&a[i]>2\*a[j]

我们的做法是将原数组扩展，例如3，2，1扩展为3，6，2，4，2，1

原数字的二倍紧跟在该数字的后边。用结构体进行储存，原数字flag=1，2倍数字flag=0

同样，在归并划分到底进行合并时，两个序列都已经是有序的序列了，用i，mid，j分别表示第一个序列的指针，中间位置，和第二个序列的指针。

此时逆序数的产生只能是flag==1的元素来产生，因此我们需要先一遍循环统计出前一个序列标记为1的元素的个数count，然后在后续的比较中维护count的值，一旦产生如题目所示的逆序，这是增加的值不为mid-i+1了而是count，因为我们需要的仅仅是原数组的数字的个数。

这里的b数组作用还是替身。

参考代码：

#include <cstdio>

struct data

{

long long da;

int flag;

}a[20002],b[20002];

long long Check(data source[],int left,int mid,int right)

{

int i=left;

int j=mid+1;

long long num=0;

data \*temp = new data[right-left+1];

int index=0;

int count=0;

for ( int m=left; m<=mid; m++)

{

if ( source[m].flag==1 )

count++;

}

//printf("%d\n",count);

while ( i<=mid && j<=right){

if ( source[i].da<=source[j].da)

{

if ( source[i].flag==1 )

count--;

temp[index++]=source[i++];

}

else if ( source[i].da>source[j].da )

{

if ( source[j].flag==0 )

num+=count;

temp[index++]=source[j++];

}

}

while ( i<=mid ){

temp[index++]=source[i++];

}

while ( j<=right){

temp[index++]=source[j++];

}

for ( int k=0; k<=right-left; k++)

{

source[k+left]=temp[k];

}

delete []temp;

return num;

}

long long ReserveNum( data source[],int left,int right)

{

if ( left<right)

{

int mid=(left+right)/2;

long long num1=ReserveNum(source,left,mid);

long long num2=ReserveNum(source,mid+1,right);

return num1+num2+Check(source,left,mid,right);

}

return 0;

}

int main(int argc, char const \*argv[])

{

int n,m;

while ( ~scanf("%d",&n))

{

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

scanf("%lld",&a[i].da);

a[i].flag=1;

a[i+n].da=2\*a[i].da;

a[i+n].flag=0;

}

//for ( int z=0; z<2\*n; z++)

// printf("%lld ",a[z].da );

// printf("\n");

scanf("%d",&m);

while ( m-- ){

//int b[n];

int x,y;

scanf("%d%d",&x,&y);

if ( x>y )

{

int t=x;

x=y;

y=t;

}

int i=0;

int j=x;

for ( j; j<=y; i=i+2,j++){

b[i]=a[j];

b[i+1]=a[j+n];

}

/\* printf("%d\n",i-1);

for ( int m=0; m<i; m++)

printf("%lld %d ",b[m].da,b[m].flag );

printf("\n");\*/

printf("%lld\n", ReserveNum(b,0,i-1));

}

}

return 0;

}