**2016级算法第一次上机解题报告**

16211130 董开宇

# 一、总结

本次上机主要考察知识点为分治、递推、简单数论算法知识。

关于题目难度，前四道题都比较简单，都是一些简单的推公式和找规律的题。对于这类题目，一开始看到题干可能有点懵不知道该怎么做，我认为这种时候不能干坐着看着题目空想，而要动手敲一敲动笔写一写，就跟做数学题的方法一样。而往往在这个思考加实践的过程中便能拨开云雾见天日，收获AC。

第五题如果不和题目较真，回归最简单的理解方法不考虑其他因素，只按照题目里所给出的伪代码来做的话也很简单。然而我因为对这道题想的太多而没能理解题目的本意，最终只做出了前四道题，没能AK，也算是个教训吧，如果自己以后遇到不理解的题目一定会及时求助助教。

最后为辛苦出题并附上HINT和详细样例解释还延长了一小时上机时间的助教们疯狂打call~

# 二、解题报告

### A．The stupid owls

### 1. 思路与算法分析

这道题是一道错排问题，求n个元素随机排列的所有排列中排成错排的概率，运用古典概型的公式求解。

基本事件的个数为n个元素的全排列：。

下面推导n个元素排列时错排事件的个数：

显然，，，则时：

1. 首先错排a号元素，那么它可以排在中除去a号位置的剩余个位置中的任意一个，可能的方式个数为。
2. 再错排其他n-1个元素，根据Ⅰ的结果，若a排在b号位置，那么此时需要错排b号元素。
3. 如果b号元素排在了a号位置，则相当于a号元素、b号元素互相交换了位置，且它们均处于错排状态，所以可以将这两个元素除外，再对其余个元素进行错排，此时得到的错排方式个数为。
4. 如果b号元素没有排在a号位置，此时排在b号位置的a号元素处于错排状态，可先将此元素除外，并将a号位置视为b号元素的原本位置。此时还剩下包括k在内一共个元素，对这个元素进行错排，得到错排方法数。

根据分步乘法原理，。

根据分类加法原理，。

综上，得到错排事件个数D(n)的递推公式 。

下面解出这个递推式的通项：

设

代入递推式可得

左右同时约去得到

移项得

同理

······

上式相乘得

代入，得

解得

所以

利用上面求得的的公式可轻松得到所求概率为

这道题的数据范围为，而unsigned long long所能表示的最大正整数也远小于，且输出结果要求保留两位小数的百分数。所以猜测在题目所给的精度要求下，随着数据的不断增大，最终结果将稳定于某个值不再变化。

通过在本地计算机测试输出的结果来看，当n增加到8以后最终结果稳定于36.79%。

### 2. 参考代码：317382

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

unsigned long long jiecheng[25];

double inverse[25];

double result[25];

jiecheng[1]=1;

inverse[1]=1.0;

result[1]=0.0;

for(int i=2;i<19;i++)

{

jiecheng[i]=i\*jiecheng[i-1];

inverse[i]=1.0/jiecheng[i];

}

for(int i=2;i<19;i++)

{

if(i%2==0)

{

result[i]=result[i-1]+inverse[i];

}

else

{

result[i]=result[i-1]-inverse[i];

}

}

for(int i=1;i<19;i++)

{

result[i]\*=100;

}

int n;

while(cin>>n)

{

if(n<=7)

cout<<fixed<<setprecision(2)<<result[n]<<"%\n";

else

cout<<"36.79%\n";

}

return 0;

}

### B．ModricWang和数论

### 1. 思路与算法分析

这是一道数论题，如果不看数据范围，一开始的一般想法可能是：对于这个给定的正整数a，让正整数b取遍从1到a+1的所有数（对大于a的数取模结果都为a故只需要计算至a+1即可），计算a%b的值并存入一个数组中，最后用sort函数对这个数组的前a+1个数快速排序，从第一个数扫描到第a+1个从而统计其中不同数字的个数。

然而这道题的数据范围非常之大，用上面的无脑做法可能会超时超内存，所以猜测对于此问题是否有某种公式或规律存在。果不其然，在使用上面的做法计算从1开始递增的较小的a时，可得到a%b的结果个数依次为2,2,3,3,4,4,5,5,6,6,7,7…

可以发现这是分奇偶数的某种规律，归纳总结得到表达式为：

下面给出上述表达式的证明：

给定正整数a，对任意正整数b：

1. 时，余数仅有一种情况，为a。
2. 时，必定存在自然数b，g，r满足，下面对a进行分类讨论：
3. a为奇数：
4. 时，显然所得结果各不相同，且依次分别为0 ，共个数。
5. 时，由，可得此时余数r的取值均包含于a)中。
6. a为偶数：
7. 时，显然所得结果各不相同，且依次分别为0 ，共个数。
8. 时，由，可得此时余数r的取值均包含于a)中。

综上所述：a为奇数时，为种。

a为偶数时，为种。

小坑：因为数据范围较大，所以申明变量时使用的数据类型均为unsigned long long。

### 2. 参考代码：316946

#include <iostream>

using namespace std;

int main()

{

unsigned long long a;

cin>>a;

unsigned long long b=a/2;

if(a%2==1)

cout<<b+2;

else

cout<<b+1;

return 0;

}

### C．AlvinZH去图书馆

### 1. 思路与算法分析

这道题是一道考察递推的题目，需要逆向从终点开始考虑。

设从起点到达第k块砖的安全的走法有种。逆向考虑，走最后一步时一共有三种方法，分别是走一块砖，两块砖和三块砖。可分为如下两类：

1. 对于只走一块砖或走两块砖，无论上一步是何种走法，均不会产生题目中所说的不安全的操作，一共有种方法。
2. 而对于最后一步走三块砖的走法，可能会产生不安全的操作。为避免出现这种情况，需要保证上一步走一块砖或两块砖，由①知，一共有种方法。

由分部加法原理可得，到达第k块砖的方法一共有种。

由于在递推过程中涉及到连续三项之间的关系，所以在使用这个递推式时，需要知道前三项的初始值，显然易得，。

### 2. 参考代码：321767

#include <iostream>

using namespace std;

unsigned long long sum[51];

unsigned long long a[51];

unsigned long long b[51];

int main()

{

sum[0]=a[0]=1;

sum[1]=a[1]=1;

sum[2]=a[2]=2;

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

for(int i=3; i<51; i++)

{

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

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

sum[i]=a[i]+b[i];

}

int n;

while(cin>>n)

{

cout<<sum[n]<<"\n";

}

return 0;

}

### D．水水的Horner Rule

### 1. 思路与算法分析

这道题是一道进制转换题，其实就是n次多项式求值。

多项式求值有两种方法：直接计算和题干中所给的霍纳规则。

对两种算法的分析如下：

1. 直接计算

伪代码：

计算时间：（2n-1次乘法和n次加法）

时间复杂度：

1. 霍纳规则

伪代码：

计算时间：（n次乘法和n次加法）

时间复杂度：

在霍纳算法中，计算时是从H进制下的最高位开始依次向低位取数字，所以可以使用string字符串数据类型（需要添加头文件<string>），这样可以使用下标运算方便快捷的从0开始向后取到字符串的最后一个非空字符（STL中string使用s.size()可获取字符串s长度），刚好与H进制下的从最高位向低位取数字是一个方向。

这里要注意的一个小的易错点是，string字符串类型使用下标访问到的每一个元素取出是char字符类型，字符数字的ASCII码与该数字的数学数值并不相同，需要减去字符’0’才可以作为数学数值进行运算。我自己在这里就出错了，将样例的结果算出了两百多。

### 2. 参考代码：317783

#include <iostream>

#include <string>

using namespace std;

int main()

{

int n;

cin>>n;

while(n--)

{

long long y1=0,y2=0;

int h1,h2;

string x1,x2;

cin>>h1>>x1>>h2>>x2;

for(int i=0;i<x1.size();i++)

{

y1=x1[i]-'0'+h1\*y1;

}

for(int i=0;i<x2.size();i++)

{

y2=x2[i]-'0'+h2\*y2;

}

long long sum=y1+y2;

cout<<sum<<"\n";

}

return 0;

}

### E．ModricWang's QuickSort

### 1. 思路与算法分析

这道题的本意是根据题目中伪代码所描述的算法建立一种划分规则，然后使用这种规则对所给元素进行划分。根据题干中所给伪代码，将元素从0开始存入数组arr中（数组长度较大，在main函数外开全局数组）。设哨兵分隔元素如果，进入划分函数：当i小于j时循环执行如下操作：如果循环执行直到。如果循环执行直到。如果，交换，此时仍然满足，继续从头执行循环直至，退出此循环。

此时完成了第一遍划分，题目要求输出第二层递归时从左往右的第二部分元素，所以对第一遍划分的左半部分继续进行第二次划分操作，更新，设right为输出时的右界并赋值为i，重置哨兵，分隔元素。此时再次进入划分函数，退出后设left为输出时的左界并赋值为i，此时从下标left到right即为第二层递归时从左往右的第二部分元素，循环依次输出即可。

至此这道题便做完了，这道题目中所谓快速排序其实并不是真正的快速排序，下面是快速排序算法的一点介绍：

快速排序算法采用的是分治算法，对于一个待排序的数组A[p…r]，快速排序的三个分治法的步骤具体为：

1. 分解：数组A [p...r]被划分为两个（可能为空）子数组A[p...q-1]和A[q+1...r]，给定一个排序基准，使得A[p...q-1]中的每个元素小于等于A[q]，A[q+1...r]中的每个元素大于等于A[q]，q下标是在划分过程中计算得出的。
2. 解决：通过递归调用快速排序，对子数组A[p...q-1]和A[q+1...r]进行排序。
3. 合并：子数组排序完成即整个数组排序完成，无需合并。

查阅《算法导论》书中关于快速排序的伪代码如下：

划分过程：

//将最后一个元素作为主元素

//从第一个元素开始到倒数第二个元素结束，比较确定主元的位置

//最终确定主元的位置

//返回主元的位置

快速排序：

QUICKSORT(A,p,r)

if p<r

q = PARTITION(A,p,r) //确定划分位置

QUICKSORT(A,p,q-1) //子数组A[p...q-1]

QUICKSORT(Q,q+1,r) //子数组A[q+1...r]

end

快速排序算法的平均时间复杂度为。

### 2. 参考代码：320083

#include <iostream>

#include <algorithm>

using namespace std;

int arr[1000002];

void divide(int &i, int &j, int &mid)

{

while(i<j)

{

while(arr[i]<=mid)

{

i++;

}

while(arr[j]>mid)

{

j--;

}

if(i<j)

{

swap(arr[i],arr[j]);

}

}

}

int main()

{

int n;

cin>>n;

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

{

cin>>arr[k];

}

int i=0;

int j=n-1;

int mid=arr[n/2];

divide(i,j,mid);

n=i+1;

int right=i;//要输出的角标上界

i=0;

j=n-1;

mid=arr[n/2];

divide(i,j,mid);

int left=i;//要输出的角标下界

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

{

cout<<arr[k]<<" ";

}

return 0;

}