2016级算法第一次上机解题报告

16211109 周宏建

# 一、总结

主要考察知识点为**分治、递推、简单数论**等简单的算法知识。

关于题目难度，全部为简单题，C题和E题出现了坑点。本次上机发挥一般，主要原因是1对于快排的理解还是不够到位，2没有好好读题，因此在E题和C题上卡了相当长一段时间。

# 二、解题报告

### A The stupid owls

### 思路分析

本题考查了错排公式

题意为把所有人的信都送错的概率，很自然能够想到用所有人都送错的情况数除以所有送信情况数。因此计算出错排的情况除以总的情况即为概率。注意数据类型，需要转换为double。

### 算法分析

根据排列组合的知识可知，在有n个人的条件下，所有送信的情况是n！种。

错排公式：

当n个编号元素放在n个编号位置，元素编号与位置编号各不对应的方法数用D(n)表示，那么D(n-1)就表示n-1个编号元素放在n-1个编号位置，各不对应的方法数，其它类推.

第一步，把第n个元素放在一个位置，比如位置k，一共有n-1种方法；

第二步，放编号为k的元素，这时有两种情况：⑴把它放到位置n，那么，对于剩下的n-1个元素，由于第k个元素放到了位置n，剩下n-2个元素就有D(n-2)种方法；⑵第k个元素不把它放到位置n，这时，对于这n-1个元素，有D(n-1)种方法；

综上得到

D(n) = (n-1) [D(n-2) + D(n-1)]

特殊地，D(1) = 0, D(2) = 1.

### 参考代码 #include<stdio.h>

#include<iostream>

#include<iomanip>

using namespace std;

long long a[30];

long long b[30];

int main()

{

int n;

a[1]=0;a[2]=1;

for(long long i=3;i<=25;i++)

{

a[i]=(i-1)\*(a[i-2]+a[i-1]);

}

b[1]=1;

for(long long i=2;i<=25;i++)

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

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

{

double res=(double)a[n]/(double)b[n];

res\*=100;

cout<<fixed<<setprecision(2)<<res<<"\%\n";

}

### }

### B ModricWang和数论

### 思路分析

本题是一道数论题（废话）

能够出到10^18这么大的数说明一定是有通用的规律能解决而不是递归一顿乱算的…

暂时没有思路可以尝试暴力算出前30个数字找找规律，于是惊喜地发现了规律，设输入的数字为n，则答案为(n+1)/2+1

从理论上来推导有以下几个方面：

1. 对于整数数a，任取整数b且b>a，有a%b=a
2. 对于整数a，任取整数b，b>=1&&b<=a有0<=a%b<=[a/2]，此处[a/2]向下取整。并且当a为奇数时可取等号，a为偶数时不可取等号。证明：若存在整数c，a%b=c且c>a/2，则若要使a%b=c成立，必有b>c，否则等式不成立，所以b>a/2，c>a/2，b+c>a，而由a%b=c可知b+c<=a，矛盾，因此不存在整数c ，a%b=c且c>a/2
3. 对于2中的情况，设a%b=c，一般地有a%(a-c)=c

### 算法分析

读入一个long long型数n，输出(n+1)/2+1

### 参考代码

#include<stdio.h>

#include<cmath>

int main()

{

long long a;

while(~scanf("%lld",&a))

{

printf("%lld\n",(a+1)/2+1);

}

}

### C AlvinZH去图书馆

### 思路分析

考察了递推的思想。可以将直接大跨步的状态显式的开一个数组存下来，也可以只开一个数组存下所有方法然后再减去。

对于i小于6时因为不涉及连续大跨步的情况，所以直接a[n]=a[n-1]+a[n-2]+a[n-3]即可

但对于后面可能出现连续大跨步的情况需要多考虑，比如在i=7时，有可能是从1直接跨到4又跨到7，进行连续两次大跨步，所以a[7]=a[6]+a[5]+a[4]-a[1]，需要在加上从4直接大跨步到7的基础上扣除从1直接跨到4的方法，而进一步，对于i=10，a[10]=a[9]+a[8]+a[7]-a[4]+a[1]这时需要减去从4连续两次大跨步的情况，而从4过来又包括了从1到4大跨步此后并不大跨步的情况又被多减去了，于是需要加回来a[1]，以此类推，例如a[13]=a[12]+a[11]+a[10]-a[7]+a[4]-a[1]…

### 算法分析

从前往后递推……注意用long long

### 参考代码

#include<stdio.h>

int main()

{

long long a[55]={0};a[0]=1;

a[1]=1;a[2]=2;a[3]=4;

for(int i=4;i<=50;i++)

{

if(i>=6)

{

a[i]=a[i-3]+a[i-2]+a[i-1];

bool tmp=false;

int t=i-6;

while(t>=0)

{

if(!tmp)

{

a[i]-=a[t];

tmp=true;

}

else

{

a[i]+=a[t];

tmp=false;

}

t-=3;

}

}

else

{

a[i]=a[i-3]+a[i-2]+a[i-1];

}

}

int n;

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

{

printf("%lld\n",a[n]);

}

}

### D 水水的Horner Rule

### 思路分析

简单来讲这个东西在中国应该叫秦九韶算法。。

假设有n+1个实数a0，a1，…，an,和x的序列，要对多项式Pn(x)= anx ^n+a(n－1)x^(n-1)+…+a1x+a0求值，直接方法是对每一项分别求值，并把每一项求的值累加起来，这种方法十分低效，它需要进行n+(n－1)+…+1=n(n+1)/2次乘法运算和n次加法运算。通过如下变换我们可以得到一种快得多的算法，即Pn(x)= anx ^n+a(n－1)x^(n-1)+…+a1x+a0=((…(((anx +an－1)x+an－2)x+ an－3)…)x+a1)x+a0，这种求值的安排我们称为霍纳法则。

### 算法分析

读入一个字符串，A(x)=a0+a1x+a2x^2+...+an-1x^n-1+anx^n在x处的值，转化为A(x)=a0+x(a1+x(a2+...+x(an-1+xan)···))，利用霍纳法则进行如下操作。。

y = 0

for i = n downto 0

y = ai + x \* y

可能对于16进制数要分ABC和abc的大小写情况（没测试过）总之这里都写了。

另外记得用long long

### 参考代码一

#include<stdio.h>

#include<string.h>

int main()

{

int n,h1,h2;

char s1[70],s2[70];

scanf("%d",&n);

while(n--)

{

scanf("%d %s",&h1,s1);

scanf("%d %s",&h2,s2);

long long res1=0,res2=0;

int len1=strlen(s1);

for(int i=0;i<len1;i++)

{

res1\*=h1;

long long tmp;

if(s1[i]>='0'&&s1[i]<='9')

tmp=s1[i]-'0';

else if(s1[i]>='A'&&s1[i]<='Z')

tmp=s1[i]-'A'+10;

else

tmp=s1[i]-'a'+10;

res1+=tmp;

}

int len2=strlen(s2);

for(int i=0;i<len2;i++)

{

res2\*=h2;

long long tmp;

if(s2[i]>='0'&&s2[i]<='9')

tmp=s2[i]-'0';

else if(s2[i]>='A'&&s2[i]<='Z')

tmp=s2[i]-'A'+10;

else

tmp=s2[i]-'a'+10;

res2+=tmp;

}

printf("%lld\n",res1+res2);

}

}

### E ModricWang's QuickSort

### 思路分析

另一种partition操作，按照要求来写就行，也可以直接两次暴力解决。快排的实现的方法可参考算法导论。。对于不同的partition操作快排在效率上会有一定差异

### 算法分析

对于partition来讲，有：

1. 设数组为arr[n]，元素从0开始存储
2. 令i=0，j=n-1, mid=arr[n/2]
3. 如果 i < j，转到4，否则转到7
4. 如果arr[i] <= mid, i++ ，重复执行直到arr[i]>mid
5. 如果arr[j] > mid, j-- ，重复执行直到arr[j]<=mid
6. 如果i < j, 交换arr[i]和arr[j]，转到4
7. 退出

注意在第二层递归时输出从左往右的第二部分的元素。

### 参考代码

#include<stdio.h>

#include<ctime>

#include<cstdlib>

int cnt;

void swap(int &x,int &y)

{

int t=x;x=y;y=t;

}

int \_partition(int arr[],int l,int r)

{

++cnt;

int i=l,j=r-1;

int mid=arr[(r+1)/2];

while(i<j)

{

while(arr[i]<=mid)

++i;

while(arr[j]>mid)

--j;

if(i<j)

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

}

if(cnt==2)

{

for(int t=i;t<r;t++)

printf("%d ",arr[t]);

printf("\n");

}

return i;

}

void \_quicksort(int arr[],int l,int r)

{

if(l>=r)

return;

else

{

if(cnt<2)

{

int p=\_partition(arr,l,r);

if(cnt<2)

\_quicksort(arr,l,p);

}

}

}

void quicksort(int arr[],int n)

{

\_quicksort(arr,0,n);

}

int a[1000005];

int main()

{

int n;

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

{

cnt=0;

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

scanf("%d",&a[i]);

quicksort(a,n);

}

}

### F AlvinZH的儿时梦想——木匠篇

### 思路分析

本题有两种思路，一种是O(n^2)级别的暴力搜索法，其中剪枝之后能提高很多效率，也可以过。

另一种方法是O(n)级别的扫描方法

### 算法分析

圆柱体积计算公式为：V=pi\*d\*d\*h/4，其中pi为圆周率，d为底面直径，h为高

1. 对于O(n^2)级别的暴力搜索法，用两重for循环找出所有可能，设置答案初值ans=0，底面直径d=0，高度h=0。当计算出比答案大的值时将答案更新为新的结果，并且将d和h更新，在此后的搜索中，如果遇到了直径小于等于d并且高度小于等于h的情况可直接跳过
2. 对于O(n)级别的算法，首先将输入读入，并且开一个数组，用来将这些木条从左到右（按照在x轴上的顺序）存储好。这里由于木条的x绝对值小于100，因此可直接开一个数组存下来。或者也可以读入后再sort一次不过这样因为sort整体复杂度变为O（nlogn）。之后我们设置两个指针，一个指向最左边的木条，称为l，另一个指向最右边的木条，称为r，计算出此时的盛水体积，并记录底面直径d和较短木条高度h。此时我们注意到：**如果将较短的木条固定，将指向较长木条的另一指针向圆心移动，也就是选中其他更靠近圆心的木条，围成木桶的盛水体积必然小于当前体积**。这是因为在此时，是较短的木条在限制盛水体积。如果将另一指针向内移动，所得结果无非两种：**1.当前指向的木条长于较短木条，2 当前指向的木条长度小于等于较短木条。**此时对于情况1，围成木桶的高度不变，底面积变小，盛水体积必然变小，对于情况2，高度变小底面积变小必然不能围成更大木桶。因此我们得出结论，**必须移动指向较短木条的指针。**不断地将指向较短木条的指针向圆心移动，每次算出新结果并更新，注意在这里也可以用到如果直径和高度都小于上一结果直接跳过的方法。

### 参考代码1（O（n））

#include<stdio.h>

#include<algorithm>

#include<iostream>

#include<cmath>

#include<iomanip>

using namespace std;

typedef struct wood

{

int len,x;

}wood;

bool cmp(wood a,wood b)

{

return a.x<b.x;

}

int main()

{

int n;

wood a[105];

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

{

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

scanf("%d%d",&a[i].x,&a[i].len);

sort(a,a+n,cmp);

int lend=0,rstart;

for(rstart=0;rstart<n;rstart++)

{

if(a[rstart].x>=0)

break;

}

if(a[rstart].x==0)

lend=rstart;

else

lend=rstart-1;

double res=0;int ra=0,minlen=0;

int l=0,r=n-1;

while(l<=lend&&r>=rstart)

{

int tmp=min(a[l].len,a[r].len),tmpr=a[r].x-a[l].x;

if(!(ra>=tmpr&&tmp<=minlen))

{

double t=(double)acos(-1)\*(double)tmpr\*(double)tmpr\*(double)tmp/4;

if(res<t)

{

res=t;ra=tmpr;minlen=tmp;

}

}

if(tmp==a[l].len)

++l;

else

--r;

}

cout<<fixed<<setprecision(3)<<res<<"\n";

}

}

### 参考代码2（O（n^2））

#include<stdio.h>

#include<algorithm>

#include<iostream>

#include<cmath>

#include<iomanip>

using namespace std;

typedef struct wood

{

int len,x;

}wood;

bool cmp(wood a,wood b)

{

return a.x<b.x;

}

int min(int &a,int &b)

{

if(a>b)

return b;

return a;

}

int main()

{

int n;

wood a[105];

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

{

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

scanf("%d%d",&a[i].x,&a[i].len);

sort(a,a+n,cmp);

int lend=0,rstart;

for(rstart=0;rstart<n;rstart++)

{

if(a[rstart].x>=0)

break;

}

if(a[rstart].x==0)

lend=rstart;

else

lend=rstart-1;

double res=0;int r=0,minlen=0;

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

{

for(int j=rstart;j<n;++j)

{

int tmp=min(a[j].len,a[i].len),tmpr=a[j].x-a[i].x;

if(r>=tmpr&&tmp<=minlen)

continue;

double t=(double)acos(-1)\*(double)tmpr\*(double)tmpr\*(double)tmp/4;

if(res<t)

{

res=t;r=tmpr;minlen=tmp;

}

}

}

cout<<fixed<<setprecision(3)<<res<<"\n";

}

}

### G D&C--玲珑数

### 思路分析

O(n^2)算法必然超时，这时候我们就不得不求助于其他方法，比如一些有利的数据结构或者算法。在这里我们发现，题目和求逆序数十分相像，很自然地想到可以利用归并排序找到玲珑数。

此题有两个坑点：

1. 下标p和q并不保证q>=p
2. 注意数据范围，数组中的数\*2可能就超过int了。。。

### 算法分析

利用归并排序求逆序数曾经出现在以前的练习题，其主要思路是利用归并排序中merge操作中的两部分是已经排好顺序的，此时利用有序性可以大大简化我们的计算。本题相似于逆序数。

例如我们对两部分排好序的部分进行merge，两部分分别为：

5 6 7 8 9

1 2 3 4 5

第一部分为[left,mid)，第二部分为[mid,right)

由于这两部分已经是有序的了，所以可以极大简化计算。

这里我们设置两个指针pl指向mid-1,pr指向right-1

当pl指向的数字没有超过pr的两倍时，pr左移，直到找到合适的未知，如果始终找不到则在mid-1位置停下，并可以直接return

此后令pl不断前移，每次让pr根据位置上的数字进行调整，由于有序性，如果pl所指数字大于pr所指数字的两倍，则它也必然大于pr以前那些数字的两倍，因此可以直接让答案加上pr-mid+1。

### 代码实现

#include<stdio.h>

long long ans=0;

long long a[10005],b[10005];

void check(int left,int mid,int right)

{

int pl=mid-1,pr=right-1,k;

for(k=pr;k>=mid;k--)

{

if(a[pl]>2\*a[k])

break;

}

for(int i=pl;i>=left&&k>=mid;i--)

{

if(a[i]>a[k]\*2)

ans+=k-mid+1;

else

{

while(a[i]<=2\*a[k]&&k>=mid)

--k;

ans+=k-mid+1;

}

}

}

void merge(int left,int mid,int right)

{

long long helper[right-left];

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

{

helper[i]=a[i+left];

}

check(left,mid,right);

int pleft=0,pmid=mid-left,k=left;

while(pleft<mid-left&&pmid<right-left)

{

if(helper[pleft]<=helper[pmid])

{

a[k]=helper[pleft];++k;++pleft;

}

else if(helper[pleft]>helper[pmid])

{

a[k]=helper[pmid];++pmid;++k;

}

}

while(pleft<mid-left)

{

a[k]=helper[pleft];k++;pleft++;

}

while(pmid<right-left)

{

a[k]=helper[pmid];++k;++pmid;

}

}

void mergesort(int left,int right)

{

if(left<right-1)

{

int mid=(left+right)/2;

mergesort(left,mid);

mergesort(mid,right);

merge(left,mid,right);

}

}

int main()

{

int n;

int p,q,t;

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

{

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

scanf("%lld",&b[i]);

scanf("%d",&t);

while(t--)

{

ans=0;

scanf("%d%d",&p,&q);

if(p>q)

{

int t=p;

p=q;q=t;

}

for(int i=p;i<=q;++i)

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

mergesort(0,q-p+1);

printf("%lld\n",ans);

}

}

}