2015级算法第四次上机解题报告

15211041 朱辉

# 一、总结

本次上机主要考察的是分治思想、递推思想，和简单的数论知识。题目难度整体上比较容易，但是后几道题有一些坑点。感觉这次的题目质量很高，涉及面比较广，而且比较基础。

# 二、解题报告

### A The stupid owls

### 思路分析

这道题的核心是错排。错排的种类数是有公式的：对于n封信，它的错排总数是

f(n)=n\*f(n-1)+(-1)^n

推导过程如下：

(1)假设对n-1封信，错排的总数是f(n-1)。

(2)把编号为n的数放在编号为k的位置，共有n-1种情况(k!=n)。

(3)此时k位置被占据，于是开始放置k号元素，自然也有n-1种放置方法。对其进行分类：1、k号元素放置在n位置，则剩下的n-2种元素其实仍然是错排，即共有f(n-2)种情况；2、k号元素没有放在n位置，那么包括k在内的n-1个元素构成错排，共有情况f(n-1)。

(4)于是有f(n)=(n-1)\*(f(n-1)+f(n-2))，其中f(1)=0,f(2)=1，由数列公式可得如上结论。

该题除了计算这个种类数以外，还要算概率。由于是古典概型，其计算方法很简单，故不再赘述。但是这其中还要考虑范围问题。由于涉及阶乘，即使用long long类型也并不能保证0<n<25中所有的情况都能计算。因此要寻找规律。由于让保留两位小数，可以发现，当n>10（不严格）时，答案稳定在36.79%，因此无需计算，直接输出即可。

### 算法分析

该题就是普通的递推，只要找到递推关系即可。其实可以不用化简公式直接用原始的f(n)=(n-1)\*(f(n-1)+f(n-2))计算。当然要注意范围问题和输出%时的转义等细节问题。

### 参考代码（A316967）

#include<cstdio>

using namespace std;

int main(){

long long all[25];

long long x[25];

x[1]=1;

all[1]=0;

for(int i=2;i<25;i++){

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

if(i%2==0){

all[i]+=1;

}else{

all[i]-=1;

}

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

}

int n;

while(scanf("%d",&n)!=EOF){

if(n<=10)printf("%.2f%%\n",all[n]/(double)x[n]\*100);

else printf("36.79%%\n");

}

}

### ModricWang和数论

### 思路分析

这是一道非常简单的数论问题。求a%b可能的答案数量。

思路如下：

1. 先令mid=a/2，此处为整除，即有mid<=a/2.0。
2. 以此为界，首先论证对于任意的0<n<mid，都存在b，使得a%b=n：

令b=a-n，可知b>a-a/2>=a/2=mid>n，故有a%b=a-b=n

1. 再证mid<n<a，不存在b，使得a%b=n：

假设存在b，则存在正整数k，使得a-bk=n，已知n>mid，则有bk=a-n<a-a/2<a/2+1<=n，即bk<n，又k是正整数，有b<n。数学上，余数必然比除数大，故此情况矛盾，可知假设不成立

1. 考虑特殊情况：1、当n=mid时，若a是偶数，则mid=a/2，若余数为n，除数必小于等于n（证明方法类似于(3)），舍去；2、当n=a时，显然令b>a皆可。
2. 考虑n<=a恒成立，故全部情况已经考虑，只要综合上述结论即可。

### 算法分析

该题属简单数论，只要知道以下几个性质即可轻松过关：

1. a%b=n定义为存在正整数k，使得a-bk=n
2. n<b恒成立
3. 0<=n<=a恒成立

### 参考代码（B317131）

#include<cstdio>

using namespace std;

int main(){

long long a;

long long b;

while(scanf("%lld",&a)!=EOF){

b=a/2;

if(a%2==0){

printf("%lld\n",b+1);

}else{

b+=1;

b=a-b;

b+=2;

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

}

}

}

### AlvinZH去图书馆

### 思路分析

本题仍然找递推关系。类似练习赛中的两行路问题，这道题主要考虑到达对应位置可能的走法，只是要考虑更多内容。

首先，对于题目的理解要到位。如果有n个砖块，其实是要走n步，而不是n-1步，这里很容易出错。

由于一次跨两块砖有特殊要求，所以这种但考虑。为了方便起见，我把正常走(1步)、跨一块砖(2步)、跨两块砖(3步)都分开。

设三个数组all\_1,all\_2,all\_3，分别代表这三种情况，即跨1，2，3步到达该砖块的情况总数。对于第n块砖来说，all\_1[n]是走一步到达n的，那么显然，all\_1[n]=all\_1[n-1]+all\_2[n-1]+all\_3[n-1]，也就是等于所有到达n-1砖块的情况总数；同理all\_2[n]=all\_1[n-2]+all\_2[n-2]+all\_3[n-2]。第三中情况略有不同，由于不能对身体不好，所以all\_3[n]不能考虑也是跨3步到达n-3的情况，即all\_3[n]=all\_1[n-3]+all\_2[n-3]。

最终，情况数即对应砖块的三个分量之和。

### 算法分析

该题只是在一般的路径问题基础上多加了限定条件，最简单的方法就是把加有约束的情况单分出来，即all\_3数组，之后只要仔细考虑每个数组的递推关系就可以了，稍微复杂，但还是比较简单。

### 参考代码（C317734）

#include<cstdio>

using namespace std;

long long all\_1[52];

long long all\_2[52];

long long all\_3[52];

int main(){

all\_1[1]=1;

all\_1[2]=1;

all\_1[3]=2;

all\_2[1]=0;

all\_2[2]=1;

all\_2[3]=1;

all\_3[1]=all\_3[2]=0;

all\_3[3]=1;

for(int i=4;i<=51;i++){

all\_1[i]=all\_1[i-1]+all\_2[i-1]+all\_3[i-1];

all\_2[i]=all\_2[i-2]+all\_1[i-2]+all\_3[i-2];

all\_3[i]=all\_1[i-3]+all\_2[i-3];

}

int n;

while(scanf("%d",&n)!=EOF){

printf("%lld\n",all\_1[n]+all\_3[n]+all\_2[n]);

}

}

### 水水的Horner Rule

### 思路分析

这道题的算法已经给出了，并没有什么难度。只是输入的时候可能会造成溢出，所以直接用字符串输入会更方便一些，而且也容易提取每个位。

输入以后首先将每位由ASCII转换成数字，之后清零结果，每次结果乘对应进制之后加下一位，这样就将数字转换成了10进制，然后两个十进制数相加，注意使用long long 即可。

### 算法分析

这道题的算法已经在题目中有详尽说明了，只要按照说明写就可以了。

### 参考代码一（D317980）

#include<cstdio>

using namespace std;

char all[32];

int main(){

int n;

int h;

long long result1,result2;

while(scanf("%d",&n)!=EOF){

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

result1=result2=0;

scanf("%d%s",&h,all);

for(int i=0;all[i]!='\0';i++){

all[i]-='0';

result1\*=h;

result1+=all[i];

}

scanf("%d%s",&h,all);

for(int i=0;all[i]!='\0';i++){

all[i]-='0';

result2\*=h;

result2+=all[i];

}

printf("%lld\n",result1+result2);

}

}

}

### ModricWang's QuickSort

### 思路分析

本题基本思路是快排。但是本题的快排和数据结构课上学的快排有一定的区别。数据结构课上的快排思路是：

1. 选定最前端的数作为中位数，提取出来；
2. 移动后面的指针tail，并且判断对应数是否大于中位数，若大于则重复(2)，否则把tail指向的数据直接复制到head；
3. Head后移，判断当前值是否小于中位数，若是则重复(3)，否则将该数复制到tail
4. 重复(2)(3)直到tail==head，将中位数复制回head

这题的快速排序没有第(4)步，原因是使用tail和head交换的方法并不会破坏数据组成。最终，tail和head（代码中使用i，j）夹出一个分割，分割前后两个串。

这道题由于不是问最终排序结果，所以可以简化：对全体数据应用一次上述过程，找到该过程中“左侧的”数据串，之后对这个串应用过程，找到“右侧的”串即可，不需要递归操作。

### 算法分析

这题的核心算法就是快速排序，只是由于题目并不要求最后排序的结果，所以可以简化操作，不进行递归操作。这道题其实并不难，我没能做出来主要还是局限在了以前学的内容上，没能正确理解题意。

### 参考代码（E319144）

#include <cstdio>

using namespace std;

int all[1000000];

int did(int last){

int mid,temp;

if(last%2==0){

mid=all[last/2];

}else{

mid=all[last/2+1];

}

int i=0,j=last;

while(i<j){

while(all[i]<=mid) i++;

while(all[j]>mid) j--;

if(i<j) {

temp=all[i];

all[i]=all[j];

all[j]=temp;

}

}

return j;

}

int main(){

int n;

int mid1,mid2;

while(scanf("%d",&n)!=EOF){

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

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

}

mid1=did(n-1);

//printf("%d\n",all[mid1]);

mid2=did(mid1);

//printf("%d %d\n",mid1,mid2);

for(int i=mid2+1;i<=mid1;i++){

printf("%d ",all[i]);

}

printf("\n");

}

}

### AlvinZH的儿时梦想——木匠篇

### 思路分析

这道题我还是没能找到O(n)的解决方法。我的思路是暴力求解加剪枝。暴力求解即对于原点左侧每根木棍，都遍历原点右侧的所有木棍，求其围成的容积。剪枝的话，对于原点左侧的木棍，如果它的左侧木棍（如果存在）比它高，那么这个木棍围成的容积一定没有那根木棍围成的大，因为圆柱体的底面积、高都更小。对右侧同理。这种方法可以把大量的乘法计算省略。

### 算法分析

这道题，我用的算法就是剪枝，通过先验信息（相邻木棍的高度）将一部分计算省略。我还在想O(n)的算法。目前的思路是通过先验给每个木棍赋权值，同时通过单调队列直接舍去某些木棍，以达到简化计算的目的。

### 参考代码（F320770）

#include<cstdio>

#include<cmath>

#include<algorithm>

using namespace std;

double pi=acos(-1);

struct woods{

int x;

int h;

}all[105];

int ptr=0;

bool compare(woods a,woods b){

return a.x<b.x;

}

double run(woods a,woods b){

double mid=b.x-(a.x+b.x)/2.0;

double h;

if(a.h>b.h) h=b.h;

else h=a.h;

return pi\*mid\*mid\*h;

}

int main(){

int n,x,h;

double maximum,temp;

while(scanf("%d",&n)!=EOF){

ptr=0;

maximum=0;

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

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

all[ptr].h=h;

all[ptr].x=x;

ptr++;

}

sort(all,all+ptr,compare);

for(int i=0;all[i].x<=0 && i<ptr ;i++){

if(i>0 && all[i].h<all[i-1].h) continue;

for(int j=ptr-1;all[j].x>=0 && j>i;j--){

if(j<ptr-1 && all[j].h<all[j+1].h) continue;

temp=run(all[i],all[j]);

if(temp>maximum) maximum=temp;

}

}

printf("%.3f\n",maximum);

}

}

### D&C--玲珑数

### 思路分析

这道题是这次上机中最难的一道题。思路是寻找最优子序列。

类似于寻找最长单调子序列的方法，进行分治。假设函数f(a,b)完成对原始序列在区间[a,b]上的排序，并返回在[a,b]上的玲珑数个数，我们要考虑对于两个相连的区间上的合并问题。

之所以要排序，是因为如果a>2\*b，那么对于任意的c<b，必有a>2\*c，也就是说，排好序的两个数列在合并的时候可以省去大量的判断过程(比如不用判断a>2\*c)。

由于总是要用分治法，所以排序本身使用归并排序。在归并之前，先行判断是否满足玲珑数条件，将全部玲珑数找到。具体方法如下：两个相邻区间，对于第一区间任意一个a[i]，找到恰不满足玲珑数条件的第二区间的a[j]，那么a[i]对应的玲珑数个数就是j-startOfSecond，其中startOfSecond代表第二区间的起始位置。

将全部玲珑数加和后返回即可。

注意使用long long，对于p<q的情况认为区间范围是[q,p]。

### 算法分析

这道题灵活使用了分治策略，综合了最优子序列问题和归并排序，非常有价值。

### 参考代码（G320293）

#include<cstdio>

#include<algorithm>

using namespace std;

long long all[10000];

long long part[10000];

long long temp[10000];

int ptr\_temp=0;

int run(int start,int ends){

int left,right,result=0;

if(start==ends) return 0;

else if(start==ends-1){

if(part[start]>part[ends]){

swap(part[start],part[ends]);

if(part[start]\*2<part[ends]) return 1;

}

return 0;

}

left=run(start,(start+ends)/2);

right=run((start+ends)/2+1,ends);

int i=start,j=(start+ends)/2+1;

while(i<=(start+ends)/2 && j<=ends){

if(part[i]>2\*part[j]){

j++;

}else{

i++;

result+=(j-(start+ends)/2-1);

}

}

while(i<=(start+ends)/2){

i++;

result+=(j-(start+ends)/2-1);

}

ptr\_temp=0;

i=start,j=(start+ends)/2+1;

while(i<=(start+ends)/2 && j<=ends){

if(part[i]<part[j]){

temp[ptr\_temp]=part[i];

ptr\_temp++,i++;

}else{

temp[ptr\_temp]=part[j];

ptr\_temp++,j++;

}

//for(int i=start;i<=ends;i++){

//printf("%d ",part[i]);

//}

//printf("\n");

}

while(i<=(start+ends)/2){

temp[ptr\_temp]=part[i];

ptr\_temp++,i++;

}

while(j<=ends){

temp[ptr\_temp]=part[j];

ptr\_temp++,j++;

}

for(i=start,ptr\_temp=0;i<=ends;i++,ptr\_temp++){

part[i]=temp[ptr\_temp];

}

return result+left+right;

}

int main(){

int n,t,p,q;

while(scanf("%d",&n)!=EOF){

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

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

}

scanf("%d",&t);

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

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

if(p>q) swap(p,q);

for(int j=0,k=p;k<=q;j++,k++){

part[j]=all[k];

}

printf("%d\n",run(0,q-p));

//for(int i=0;i<=q-p;i++){

//printf("%d ",part[i]);

//}

//printf("\n");

}

}

}