2016级算法第一次上机解题报告

15041091 唐博文 162114班

1. 总结

本次上机难度确实如助教考前所说，难度不是很大。不过我这个学期刚转入软件学院进行学习还是感受到自身代码能力与其他同学相比的不足，一方面确实训练太少，不能够熟练迅速地找到思路并进行代码实现，另一方面对于很多细节以及相关知识掌握程度远远不及。

至于本次考查的内容就是递推、分治、剪枝、和数论吧，尤其递推还比较多，递推式的推导，数论的分析很考验思维能力，分治和剪枝则可以看出算法的掌握水平吧？

1. 解题报告
2. The Stupid Owls
3. 思路分析

投机取巧从HINT中获得了思路，之前总是想会不会有两个人原本要寄给同一个人，题意就很复杂搞不清楚，错排提示之后，考虑递推式是啥子就好了，然后由于要求概率，再写一个阶乘把全排列算出来即可。

1. 参考代码

#include<stdio.h>

int main()

{

int n[1000], i = 0, j, k;

long long statistics[25] = {0, 0, 1}, sum;

while (scanf("%d", &n[i]) == 1) //输入的控制，每行只要有一个数据成功输入就继续下去

i++;

for (j = 3; j <= 25; j++)

statistics[j] = (statistics[j - 1] + statistics[j - 2]) \* (j - 1); //错排的递推式

for (j = 0; j < i; j++){

sum = 1;

for (k = 1; k <= n[j]; k++)

sum \*= k; //循环求全排列（阶乘）

printf("%.2lf%%\n", (double) statistics[n[j]] / sum \* 100); //变成百分数时\*100一开始忘掉了就一直WA

}

return 0;

}

1. 算法分析

关键就是错排数的递推式怎么出来的吧，就是设n个元素的错排数为D(n)，然后现在将第n个元素错放到第k个位置，此时分两种情况：第k个元素如果放到第n个位置的话，还剩n-2个元素有D(n-2)种错排方法；第k个元素如果没放到第n个位置，相当于包括它在内n-1个元素有D(n-1)种错排方法，加之k的选取有n-1种可能，递推式为D(n) = ( D(n-1) + D(n-2) ) \* (n-1)。时间渐进复杂度也就是O(n)吧。

1. 心得体会

就还好吧，错排的递推说实话还没有特别清楚，不过有一个问题一直弄不明白，就是为什么这个输入操作不能在自己的电脑上进行，很烦。

1. ModricWang和数论
2. 思路分析

这个题我认为关键就在与数学的分析能力吧，其实从数学上来说也很容易理解，觉得没做出来的问题是当时就是没能静下心来好好用笔写写找找规律。

事实上任何给定一个数字作为被除数，考虑它被一个整数除得到的余数时，除以1余0，除以比它大的数余本身，这是最基本的一定会出现的两种情况，基于这两种情况我们可以特殊地分析0、1、2。之外呢，我们分奇数和偶数考虑。如果是一个偶数a，小于它的二分之一的b作为除数时一定会和2\*b作为除数得到的余数相同，所以我们只要考虑大于它的二分之一的除数，这个时候a-b都是可能的余数，而它的二分之一作为除数时余数为0这是已经在特殊情况中考虑过的，因此偶数a可能的余数就是0到a/2-1再加上本身共a/2+1个；如果是一个奇数a，类似偶数的分析，它可能的余数就是0到(a-1)/2外加它本身共(a+1)/2个余数。

有了以上分析，代码的实现只需要简单地用条件语句控制一下输出就好了。

1. 参考代码

#include<stdio.h>

int main()

{

long long a, sum;

scanf("%lld", &a);

if (a <= 2)

sum = 2;

else if (a % 2 ==0)

sum = a / 2 + 1;

else

sum = (a + 1) / 2 + 1;

printf("%lld", sum);

return 0;

}

1. 算法分析

这叫什么算法啊，我有些说不上，就叫数论吧，感觉都是在纸上做出来的，至于时间渐进复杂度呢，我写的看上去是O(1)呢。

1. 心得体会

下笔如有神，规律是写着写着找来的，脑子想的话，像我这种比较笨的，还是以后不要尝试了。还有就是别畏难吧，第一念头总是逃跑的话以后一个题都搞不定啊！

1. AlvinZH去图书馆
2. 思路分析

在练习赛中我们做过一个题叫“群鸦的盛宴”，这个题目和那个题目几乎一样，所以我们的思路还是集中在找出正确的递推式，并且按理来说肯定是十分相似的。

那么我们想，你最后一定是踩在第n块砖上，你是如何到达第n块砖上的呢？题目中的描述给出了三种方法，你可以跨一步到达第n个位置，也可以跨两步，还可以跨三步。所以到达第n块砖处的方法数可以由到达第n-1处的方法数、到达第n-2处的方法数和到达第n-3处的方法数相加得到（联系加法原理）。值得一提的是，从n-3处到达n处有限制，因为你这一次迈了三步，说明你上一次必定没有迈3步，所以我们用于求解n处方法总数的n-3方法数不是到达n-3时的全部方法，而应该去掉迈三步到达n-3位置的情况，也就是说用于求解n处的n-3只包括了n-4与n-5之和。

到此我们写出递推式method[i] = method[i-1] + method[i-2] + method[i-4] + method[i-5]。利用这个递推式和循环，我们可以方便地求解题目范围内各个n对应的方法总数。

1. 参考代码

#include<stdio.h>

int main()

{

long long method[50] = {0, 1, 2, 4, 7, 13}, ans[100000] = {0}; //前面的几种情况在找规律的时候先算出来直接赋值，省得后面再用条件结构，不过我感觉显得很蠢

int i, j, n;

for (i = 6; i <= 50; i++)

method[i] = method[i-1] + method[i-2] + method[i-4] + method[i-5]; //递推

i = 1;

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

ans[i] = method[n]; //只要输入没有结束就按顺序存出这一输入对应的输出结果

i++; //这样i就可以存出到底输入了几组数据也就知道总共有几组答案了

}

//i = 0;

for (j = 1; j < i; j++)

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

/\*while (1){

if (ans[i] != 0){

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

i++;

}

else

break;

}\*/ 这里是之前用的输入方式，应该也没错，不过我实在找不出错误的时候以为这里出错了就注释掉了。

return 0;

}

1. 算法分析

先谈一下我的实现方式，我是先用递推把题目范围内可能涉及到的里程对应的方法数都求出来，用一个数组method保存，method[i]表示到第i个位置处的方法总数。之后每次输入都根据所输入的数作为下标找到method里的对应结果按顺序保存到ans结果数组中依次输出即可。

所以这就是一个递推的题啊，然后应该就必然可以用递归做，时间渐进复杂度的话，O(n)吧，里程的范围已经限制在50以内了没什么影响，影响的是数据规模，到底总共多少组数据，时间和他成正比的。

1. 心得体会

感觉自己对于各种基本的算法还不太熟悉吧，然后思维上也有点跟不上需要多训练。一个很重要的点是我在做这个题的时候发现自己打的代码少出现很多不应该出现的问题应该在以后多多注意。

首先上面我给出的参考代码其实有一些小瑕疵，我定义的n是int类型但是在scanf的说明串中用了lld格式进行输入，这导致我在调试的时候发生了一个诡异的问题，就是ans[0]的值在赋值以后进入下一次循环会被重置为0，所以我只好从ans[1]开始使用这个数组。虽然不知道什么原因这是很基础的问题，不应该犯错，以后要时刻提醒自己，不要将时间荒废了。

1. 水水的Horner Rule
2. 思路分析

这个题我感觉没什么思路可言，就是根据题目描述进行代码实现吧，甚至连标题所说的霍纳规则也可以不使用，简单地用多项式做进制转换再求和。不过我感觉用霍纳规则其实更简单一点也说不定？因为用单项式还要求幂，霍纳规则降次后就直接循环了。

最后再多说一句处理上的想法吧，其实输入的数都是可以用整型存储的，不过我还是选择了字符串，因为运用霍纳规则时，最里层的是数字的最高位，如果用整型处理的时候要先从最低位提取存入一个数组在使用，并不方便，直接用字符串更简洁。

1. 参考代码

#include<stdio.h>

int main()

{

long long y, n, sum, ai;

int h, i, k;

char x[100];

scanf("%lld", &n);

while (n--){

sum = 0;

for (i = 1; i <= 2; i++){

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

y = 0;

k = 0;

while (x[k] != '\0'){

ai = x[k] - '0'; //字符转成数字

y = ai + h \* y; //霍纳规则的循环，照抄无误233

k++;

}

sum += y;

}

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

}

return 0;

}

1. 算法分析

可以用math.h文件中的pow函数求幂来进行进制转换。代码的渐进时间复杂度O(n)吧感觉，因为处理每一组数据都是常数级别的执行次数，应该就只与数据规模相关了。

1. 心得体会

这个题真的要说一下，说实话我大部分时间都浪费在上面了，为什么呢，因为觉得这个题真的好简单啊，但是就是过不了心里生气啊！虽然一开始没用题目给的霍纳规则，后来用了AC了，不过我发现并不是这个的问题，真正的问题在于输出没搞清楚。题目中对于输出的描述我理解成了先把所有的输入进行完，然后把结果一次性都输出，这种脑残的错误为什么会犯？做题太少了经验不足呀老铁，以前只是上了自己学院开的不作数的C语言课，上了上机，不知道OJ的存在也没刷过，所以现在真的是太菜了，好好学习，在此自勉。

1. ModricWang’s QuickSort
2. 思路分析

说实话，题目的描述导向性好强啊，我一开始想到的就是先用快速排序把数组排好，然后在它的递归过程中想办法判断何时是我们需要的结果存出来。不过再细细想一下，还递归干啥，直接手动控制他调用两次qsort函数就ok了的呀，所以这个题还挺简单有趣的。

所以要提一下快速排序，题目中所描述的规则其实已经非常详尽，但是要解决本题还是必须要吃透快排的整个过程。依题目所述确实每一趟都会把比mid小的放在一边，大的放在另一边，但是要注意的是两个游标i和j，真正分隔这两段的不是mid这个数，也就是说mid并不一定在这两段的交界处，他只不过处于其中一段之中罢了，真正能够让你区分这两段的标志是一趟过后游标所处的位置。

因此我们第一次调用，返回左游标i的值就可以知道我们所要输出的左段第二部分的右边界了。而左段第二部分的左边界也很显而易见，我们第二次调用返回左游标i的值就可以了（可能有一些什么加一减一的细节就不care了，具体情况思索一下就好了吧应该这样）。

1. 参考代码

#include<stdio.h>

#define LEN 1000100

int main()

{

int l, r, k, n, arr[LEN];

int qsort(int arr0[LEN], int left, int right, int m); //传递的参数有

scanf("%d", &n);

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

scanf("%d", &arr[k]);

r = qsort(arr, 0, n-1, n / 2); //调用快排，传入整段，寻找左段右边界（也即左段第二部分右边界）

l = qsort(arr, 0, r-1, r/2); //调用快排，传入左段，寻找左段第二部分左边界

for (k = l; k < r; k++){

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

if (k != r)

printf(" ");

}

printf("\n");

return 0;

}

int qsort(int arr0[LEN], int left, int right, int m) //实参的数组将地址传过来了，可以直接在这里修改引起主函数里的数组存储数据变化

{

int i, j, mid, t;

if (left < right){

i = left;

j = right;

mid = arr0[m];

while (i < j){

while (i <= j && arr0[i] <= mid) //左游标向右遍历寻找

i++;

while (j >= i && arr0[j] > mid) //右游标向左遍历寻找

j--;

if (i < j){ //交换两个位置不合理的元素

t = arr0[i];

arr0[i] = arr0[j];

arr0[j] = t;

}

}

}

return i; //返回一波左游标位置美滋滋

}

1. 算法分析

递归都没有吧，应该就是一个对于快速排序算法的理解和灵活运用，时间渐进复杂度的话感觉是和数据规模成正比O(n)吧。

1. 心得体会

在自己电脑上调试的时候由于数组定义在函数内，1.0e+7的大小可能太大了就运行不了，询问了同学得知OJ是可以运行的，而且在自己电脑上只要在函数外定义就ok了，还是学到了，这些小细节可以总结记录下来。

1. AlvinZH的儿时梦想——木匠篇
2. 思路分析

比较笨拙，虽然提示中提到可以剪枝降低复杂度，但是我没能想出来怎么操作，只好笨拙地枚举所有情况，对每一组数据在枚举的过程中不断更新最大储水量，最后得到的就是结果了。

1. 参考代码

#include<stdio.h>

#include<math.h>

int main()

{

const double pi = acos(-1);

double s[10000] = {0.0}, radius, height;

int n, i, j, k, x, h, count, u, v;

int minus[10000], positive[10000], minush[10000], positiveh[10000];

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

scanf("%d", &n);

j = 0;

k = 0;

count = 0;

for (i = 1; i <= n; i++){

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

//printf("%d %d\n", x, h);

if (x < 0){

minus[j] = x;

minush[j] = h;

j++;

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

}

else if (x == 0){

minus[j] = x;

minush[j] = h;

j++;

positive[k] = x;

positiveh[k] = h;

k++;

}

else{

positive[k] = x;

positiveh[k] = h;

k++;

}

}

//printf("%d %d\n", j, k);

for (u = 0; u < j; u++){

for (v = 0; v < k; v++){

radius = (double) (positive[v] - minus[u]) / 2;

//printf("%lf\n", radius);

if (positiveh[v] >= minush[u]){

height = (double) minush[u];

//printf("%lf\n", height);

}

else{

height = (double) positiveh[v];

//printf("%lf\n", height);

}

if (s[count] < (pi \* radius \* radius \* height))

s[count] = pi \* radius \* radius \* height;

//printf("%d %lf\n", count, s[count]);

}

}

count++;

//}

int c;

for (c = 0; c < count; c++)

printf("%.3lf", s[c]);

return 0;

}

1. 算法分析

很无脑的枚举吧，可能本来想考查的应该是如何剪枝，但是心慈手软暴力也可以过。时间渐进复杂度O(n2)。开数组的时候根据题目要求我是将负位置的相关参数和正位置的相关参数分开两个数组存了的。

1. 心得体会

特意搜索了剪枝算法，还询问了同学听他说什么“单调队列”，最后依然没能想出来，脑子好笨啊。

1. D&C——玲珑数
2. 思路分析

暴力写过一次超时了，然而一开始并没能想到如何解决这个问题，后来从同学那里得到启发可以借鉴归并排序的思路采用分治法。很遗憾时间不够了代码还没能写出来不过我有了一些想法正在努力尝试。

1. 参考代码

暂无

1. 算法分析

其实我可能在瞎BB。

归并排序（二分）的划分应该是可以直接利用的，不过如果是排序，它在归并的时候是两个中不断取最小的出来，对于这一题可以变成循环判断左段的数和右段的数是否是玲珑对。

1. 心得体会

心情复杂。