第一次上机

地址

[第一次上机]

题目列表

解题报告

871 The stupid owls

分析

emmm怕英语表述的不到位，决定给Hint...但是
看了大家的解题报告，看来我的Hint给的还是太直白了
这的确是个完全错排的问题，但希望大家不只是百度了一下直接用了人家的公式而是真的明白这个递推公式的含义
设n封信完全错排的方式有f[n]种。 这样想，第一封信是送错的，那么有(n-1)种选择，假设放在了位置k；信k也是送错的，这是它有两类选择：如果它正好在第一封信本该到的位置，那么剩下的n-2封信就是新的完全错排问题有f[n-2]种方法；如果没有放在第一封信的位置，那这就是(n-1)封信的错排（可以理解为此时第k封新本应送到第1个位置但是为了错排不能送，也就是n-1封信情况下的完全错排）。所以有 f[n]=

(n-1)*(f[n-1]+f[n-2]); 有了这个推导公式后就可以自底向上的推导。记得要同时记录总的排列数，n封信的话有n!种排列方式，相除即可。记得用long long来存储。
上机的时候有同学不太理解这个概率是怎么回事，f[n]算的是完全错排有多少种选择，应当再除以n!也就是送n封信全部的可能

代码样例

```
#include<cstdio>
#include<iostream>
#include<algorithm>
using namespace std;
long long f[30];
long long b[30];
int main()
{

        int c,n;
        f[1]=0;  b[1]=1;
        f[2]= 1;  b[2]= 2;
        f[3]=2;  b[3]=6;
        for(int i= 4;i<=20;i++)
            {
                    b[i]=b[i-1]*i;
                f[i]=(i-1)*(f[i-1]+f[i-2]);
            }

                    while(~scanf("%d",&n)){
          double ans = (double)f[n]/b[n];
            ans = ans*100;

            printf("%.2lf%%\n",ans);
                    }


}
```

算法分析

用递归的方法也可以做，但是会慢很多； 建议用递推的方式自底向上，$O(n)$的时间内就可以了

843 ModricWang和数论

思路

设$m = a\%b$, 分为3种情况：

1. $b < a$时，m一定在 $1..\lfloor\frac{a-1}{2}\rfloor$ 之间，共 $\lfloor\frac{a-1}{2}\rfloor$个数
2. $b = a$时，$m = 0$
3. $b > a$时，$m = a$

因此，共 $\lfloor\frac{a-1}{2}\rfloor + 2$ 个 m，也就是 $\lfloor\frac{a+1}{2}\rfloor + 1$

时间复杂度$O(1)$，空间复杂度$O(1)$

代码

```
#include <iostream>

using namespace std;

int main() {
        long long a;
        cin >> a;
        cout << (a + 1) / 2 + 1;
        return 0;
}
```

860 AlvinZH去图书馆

前言：由于助教的错误操作，此题的错误数据给大家上机造成影响，在此表示歉意；同时感谢李明同学和郭镕昊同学，在上机时勇于怀疑题目问题，在此提出鼓励。

错误的思路

递推，像斐波那契数列那样递推。 本题可以先转化成从第0块砖走到第n块砖的方法数，令 $F[i]$ 代表走到第 i 块砖的方法数。

错误的分析

求走到第 i 块砖的方法数：①从第 $i-1$ 块砖走来，$F[i-1]$；②从第 $i-2$ 块砖走来，$F[i-2]$；③从第 $i-3$ 块砖走来，$F[i-3]$，这里面，得减去走到 $i-3$ 的前一步也是骚操作的情况，那不就是 $F(i-6)$ 么。 最后得到：
$F[i] = F[i-1] + F[i-2] + F[i-3] - F[i-6]$。

纠错

错在哪呢？错在减去的那个部分！

我们要减去的就是走到 $i-3$ 的前一步也是骚操作的情况吗？

仔细想想，你别忘了，$F[i-6]$ 里面也有最后一步是骚操作的情况（从 $i-9$ 直接骚操作），这些情况不应该被减去。

所以，这样算出来的答案比真实答案小。这种情况从 $n \geq 9$ 开始就会体现，这也是一部分同学从AC到0.5分的原因。

假设走一次骚操作会导致脚疼，那么我们要减去的是**在 $i-6$ 位置脚不疼的情况**。

所以，也可以退出上面的错误方法多减去的是 $i-6$ 位置脚疼的情况。

正确的解决方案

如何避免上述多减的情况？我们可以开两个一维数组:

$f[52]$：$f[i]$ 同上 $F[i]$，表示走到 i 位置的方法数。

$g[52]$：$g[i]$ 表示走到 i 位置**脚不疼**的方法数。

这样的话，递推公式变成 $f[i]=f[i-1]+f[i-2]+g[i-3]$，而 $g[i]=f[i-1]+f[i-2]$，请参考参考代码一。

还有一种思路，其实是异曲同工，在此做简单解释。

二维数组 $step[52][2]$：其中第二维为状态位，为0表示脚不疼，为1表示脚疼。

递推公式：
$step[n][0]=step[n-1][0]+step[n-2][0]+step[n-1][1]+step[n-2][1]$，而 $step[n][1]=step[n-3][0]$，请参考参考代码二。

上面二维数组变成两个一维数组效果相同，多数同学使用这种方法。

参考代码一

```
//感谢@Author:  骆嘉航

#include <cstdio>

long long f[52];
long long g[52];//走到第i块砖不疼的方法数
```

```
int n;

int main()
{
	f[0] = 1;
	f[1] = 1;
	g[0] = 1;
	g[1] = 1;
	for(int i = 2;i <= 50; i++)
	{
		f[i] = f[i - 1] + f[i - 2] + g[i - 3];
		g[i] = f[i - 1] + f[i - 2];
	}
	while(~scanf("%d", &n))
	{
		printf("%lld\n", f[n]);
	}
	return 0;
}
```

参考代码二

```
//感谢@Author:李明

#include <iostream>
using namespace std;

long long step[51][2];
int n;

int main()
{
	step[1][0] = 1;
	step[1][1] = 0;
	step[2][0] = 2;
	step[2][1] = 0;
	step[3][0] = 3;
	step[3][1] = 1;
	for(n = 4; n < 51; ++n)
	{
		step[n][0] = step[n-1][0] + step[n-2][0] + step[n-1][1] + step[n-2][1];
		step[n][1] = step[n-3][0];
	}
	while(cin >> n)
	{
		cout << step[n][1] + step[n][0] << endl;
	}
	return 0;
}
```

867 水水的Horner Rule

思路

经鉴定，这道题是真水题。题目的意思很简单，给你两个不同进制的数，把他们加起来用十进制表示。

方法：先把两个不同进制的数都转化成十进制，再相加就ok。

分析

注意两个简单问题，由于两个数都是在int范围内的正整数（降低了难度），题目已经说明是实际大小在int范围内，也就是说这个数很可能有31位，不过没有关系，我们才不管它多长，用 **字符串读入** 一点问题也没有！而且使用字符串还有一个有点就是我不用去操作这个数，除法or求模什么的。

第二个问题是转化十进制，题目描述中也提供了伪代码方法，采用最小乘法策略，其中的x就是我们的进制数，可以快速转化，另外两个int相加有可能超过int哦！

思考题： 本题降低了难度，想一想，如果有可能是负数又该如何处理？超过十进制的进制又如何转换呢？

参考代码

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#define MaxSize 100005

using namespace std;

long long ConvertToD(char *s, int H)
{
	long long result = 0;
	for (int i = 0; i < strlen(s); i++) {
		result = result * H + (s[i] - '0');
	}
	return result;
}

int main()
{
	int n;
	while (~scanf("%d", &n))
	{
		int H1, H2;
		char x[32], y[32];
		while (n--)
		{
			scanf("%d %s %d %s", &H1, &x, &H2, &y);

			printf("%lld\n",ConvertToD(x, H1) + ConvertToD(y, H2));
		}
	}
}
```

873 ModricWang’s QuickSort

思路

首先需要向大家道歉，题目描述中的划分函数并没有实现快排的划分函数的完整功能，给大家带来了困扰。在思考此题时，将其看作一种与快排无关的独特的划分方式即可。题目中已经提供了伪代码，只需要将其写成实际代码即可。以后会放出一个此题修复以后的版本给大家。

需要注意的是，后一层递归时的位置划分取决于前一层递归时mid元素最终的位置，因此在划分函数中需要以返回值或其他形式来提供前一次递归时mid元素的位置。同时也要注意，i和j作为两个位置指示符，需要严格管理相对大小，不可以出现划分的两个区域出现重叠的情况。

由于数据较为简单，要求的层数也较浅，实现划分函数后手工调用即可。

时间复杂度$O(n)$，空间复杂度$O(n)$

代码

```
#include <iostream>

using namespace std;

const int MaxN = (int) 1e7 + 10;

int n;
int nums[MaxN];

int partition(int *arr, int n) {
        int mid = arr[n / 2];
        int i = 0, j = n - 1;
        while (i < j) {
                while (arr[i] <= mid && i <= j) i++;
                while (arr[j] > mid && i <= j) j--;
                if (i < j) {
                        swap(arr[i], arr[j]);
                }
        }
        return i;

}


int main() {
#ifdef ONLINE_JUDGE
        ios_base::sync_with_stdio(false);
        cin.tie(0);
        cout.tie(0);
#endif

        cin >> n;
        for (int i = 0; i < n; i++) {
                cin >> nums[i];
        }
        int l, r;
        r = partition(nums, n);
        l = partition(nums, r);

        for (int i = l; i < r; i++) {
```

```
            cout << nums[i] << " ";
        }
        cout << "\n";
}
```

861 AlvinZH的儿时梦想--------木匠篇

前言

由于助教的失误，本题数据的精度出现严重问题，以为取10位小数完全没问题，考虑不周，望见谅。

本题当中，PI最好取值的方法是 $PI = acos(-1.0)$。

思路

中等题。先知道题目要求的是什么：$V = PI_(x_1 - x_2)^2/4_min(h_1, h_2)$ 。PI我们可以先不管，其实就是在找（$x_1 - x_2)^2 * min(h_1, h_2)$ 的最大值。

本题数据很弱，数据范围也都很小，所以有多种方法解题，甚至可以试试暴力求解。

分析

方法一：暴力解法 本题由于n、x和h都很小，所以暴力解法（直接遍历所有组合）没问题，虽然不是助教的初衷，助教表示不会出题，啊啊啊~

方法二：从位置x的角度出发，移动指针法

描述：采用两个变量分别代表最左和最右的木条，由两边向中间靠拢，移动较短的一边，直至靠拢到圆心处。

证明：当左端线段L小于右端线段R时，我们把L右移，这时舍弃的是L与右端其他线段（R-1, R-2, ...）组成的木桶，这些木桶是没必要判断的因为这些木桶的容积肯定都没有L和R组成的木桶容积大。

具体做法：可以有两种做法，一是开一个数组 $H[205]$，$H[i]$ 代表 $i-100$ 位置木条的高度，注意这种下标的处理方式，在题目"四和归零"中也有用到。左右指针初始化为0、200，向100靠拢。另一种做法是使用STL容器map，$m[i]$ 表示位置i木条的高度，这样可以不用考虑高度为0的地方。

具体可参考参考代码一。

方法三：从高度的角度出发，遍历高度。

描述：利用两个数组，下标代表高度，值分别为左右两边同一高度离圆心最远的位置。遍历一次高度数组，即可求得最大值。

具体可参考参考代码二（感谢郭镕昊dalao）。

参考代码一

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <map>
#include <algorithm>

using namespace std;
const double PI = acos(-1.0);

int n, pos, h;
map<int, int> m;

int main()
{
        //freopen("in2.txt", "r", stdin);
        //freopen("outme.txt", "w", stdout);
        while (~scanf("%d", &n))
        {
                m.clear();
                for (int i = 0; i < n; i++)
                {
                        scanf("%d %d", &pos, &h);
                        m[pos] = h;
                }

                double ans = 0.0;
                map<int, int>::iterator m_Head = m.begin();//最左边木条
                map<int, int>::iterator m_Tail = prev(m.end());//最右边木条

                while(m_Head != m_Tail && m_Head->first <= 0 && m_Tail->first >= 0)
                {
                        double r = (m_Head->first - m_Tail->first) / 2.0;//半径
                        int h = min(m_Head->second, m_Tail->second);

                        ans = max(ans, r * r * h);

                        if(m_Head->second <= m_Tail->second && m_Head->first < 0)
                                advance(m_Head, 1);
                        else if (m_Tail->second <= m_Head->second && m_Tail->first > 0)
                                advance(m_Tail, -1);
                        else if (m_Head->first == 0 && m_Tail->first > 0)
                                advance(m_Tail, -1);
                        else if (m_Tail->first == 0 && m_Head->first < 0)
                                advance(m_Head, 1);
```

```
                }
                printf("%.3lf\n", ans * PI);
        }
}
```

参考代码二

```
/*
  Author: 郭镕昊(12622)
  Result: AC      Submission_id: 318166
  Created at: Fri Oct 13 2017 20:14:13 GMT+0800 (CST)
  Problem: 861      Time: 79      Memory: 2696
*/

#include<bits/stdc++.h>

const double pi = std::acos(-1.0);
const int N = 104;
inline void Max(int &a, int b) {
        if(b > a) a = b;
}
int n, a[N], b[N];

int main() {
        while(~scanf("%d", &n)) {
                memset(a, -1, sizeof a);
                memset(b, -1, sizeof b);
                int tmp = -1;
                for(int i = 1, x, h; i <= n; ++i) {
                        scanf("%d%d", &x, &h);
                        if(!x) Max(tmp, h);//记录0位置高度
                        if(x > 0) Max(a[h], x);//右边
                        if(x < 0) Max(b[h], -x);//左边
                }
                int x = -1, y = -1, ans = 0;
                for(int h = 100; h; --h) {
                        Max(x, a[h]);
                        Max(y, b[h]);
                        if(x > 0 && y > 0) Max(ans, h * (x + y) * (x + y));
                        if(tmp > 0 && x > 0) Max(ans, std::min(h, tmp) * x * x);
                        if(tmp > 0 && y > 0) Max(ans, std::min(h, tmp) * y * y);
                }
                printf("%.3lf\n", pi * ans / 4.0);
        }
        return 0;
}
```

869 D&C–玲珑数

分析

D&C想表达的是divide and conquer的意思，不知大家get到了没
这道题与经典的求逆序数的问题很像。逆序数是数组中，i<j,a[i]>a[j]的对数。最一般

的暴力思路在数据量稍大时就会很慢，所以解决逆序数多是采用分治思想，借助归并排序 逆序数代码：

```
void mergee(int A[], int p, int q, int r)
{
        int n0 = r-p+1, n1 = q-p+1,n2 = r-q;
        int L[(n0+1)/2+1], R[(n0+1)/2+1];
        int i,j;
        for(i=1;i<=n1;i++)
                L[i] = A[p+i-1];
        for(j = 1;j<=n2;j++)
                R[j] = A[q+j];
        L[n1+1] = mm; R[n2+1] = mm;
        i = 1; j = 1; int k;
        for(k = p; k<=r;k++)
        {
                if(L[i]<=R[j])
                {
                        A[k] = L[i]; i++;
                }
                else
                {       coun += n1-i+1;//最关键的就是添加了这句
                        A[k]= R[j];j++;
                }
        }
}
```

其实只在二路合并的时候添加一句即可，因为在合并时，两个子数组内部已然有序，所以不存在逆序数，只可能发生在两个数组之间，当左边的l[i]>r[j]时构成了逆序数，同时l[i]后面属于左边的数组的数自然也都比r[j]大，可以一并加进去。
注意到这个关键的计算部分是合在合并数组的for循环中的

接下来说这道题的玲珑数，表述上非常的近似，但是再采用计算逆序数时的边计算边排序就会有问题，因为a[i]>a[j]是可以直接借助排序中的大小比较的，但是两倍情况下就会有问题，可能在排序的过程中丢掉一些玲珑数。（可以试一下3 2 1 分别求逆序数和玲珑数的情况）。所以求玲珑数时还是在Merge Sort的基础上，只是**先计数然后再排序**。但因为毕竟还是分治法，所以是比暴力法快很多。这种方法同样适用于计算逆序数，只是可能不如上面的方法计算的快。 另外，担心大家注意不到的坑点已经在Hint中给出了，希望大家都看到了。注意已经在Int范围内的数是有可能*2 的时候超出int范围的，不过看到大部分的ac代码直接选择了用long long

代码样例

```
#include <iostream>
#include <string.h>
#include <algorithm>
#include<cstdio>
#include<vector>
#include<fstream>
using namespace std;
int a[10005];
int coun;
void Merge(int a[], int p, int q, int r)
{
        int n1 = q - p + 1;
        int n2 = r - q;
```

```
		int *L = new int[n1 + 2];
		int *R = new int[n2 + 2];
		for (int i = 0; i<n1; i++)
			L[i] = a[p + i];
		for (int j = 0; j<n2; j++)
			R[j] = a[q + j + 1];
		int i = 0, j = 0;
		for (int k = p; k <= r; k++)
		{
			if (j >= n2 || (i<n1&&L[i] <= R[j]))
				a[k] = L[i++];
			else
				a[k] = R[j++];
		}
		delete []L;
		delete []R;
}
int MergeAndCount(int a[], int p, int q)
{
		if (p<q)
		{
			int mid = (p + q) / 2;

			MergeAndCount(a, p, mid) ;
			  MergeAndCount(a, mid + 1, q);
//int coun = MergeAndCount(a, p, mid)
// + MergeAndCount(a, mid + 1, q);
//上面这行表述方式等价于两次递归调用,
//只不过不用全局的coun变量了
			int j = mid + 1;
			for (int i = p; i <= mid; i++)
			{
				while (j <= q && a[i]>(a[j]*2LL))
					j++;
				coun += j - (mid + 1);
			}
/*上面这个关键的for循环部分专门用来计算玲珑数,
放在Merge里也可以,但是是不同于Merge中
本来就有的合并用的循环的*/
			Merge(a, p, mid, q);
			return coun;
		}
		else
			return 0;
}
int b[10005];
int main(){
		int n, t, p, q;
		while(scanf("%d",&n)!=EOF)
		{
			for (int i = 0; i < n; i++)
				cin >> a[i];

			cin >> t;
			int ans;
			while (t--)
			{
				coun = 0;
				cin >> p >> q;
				if (q < p)swap(p, q);
				for (int i = 0; i<n; i++)b[i] = a[i];
				ans = MergeAndCount(b, p, q);
				cout << ans << "\n";
			}
		}
		return 0;
}
```