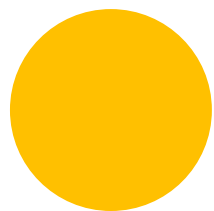
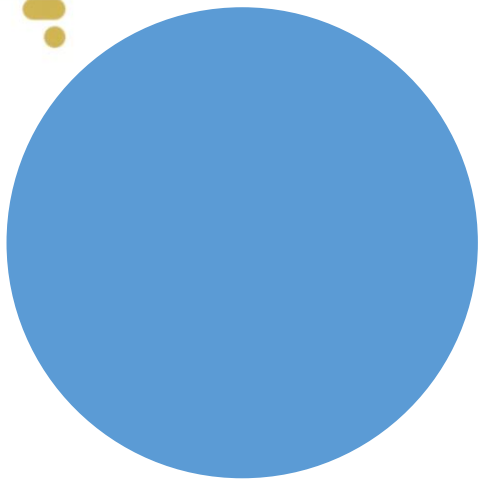


# 马上开始

35tang-C++竞赛系列三阶课程





# 《 35tang-C++竞赛系列三阶课程 》



## 本节目标

---

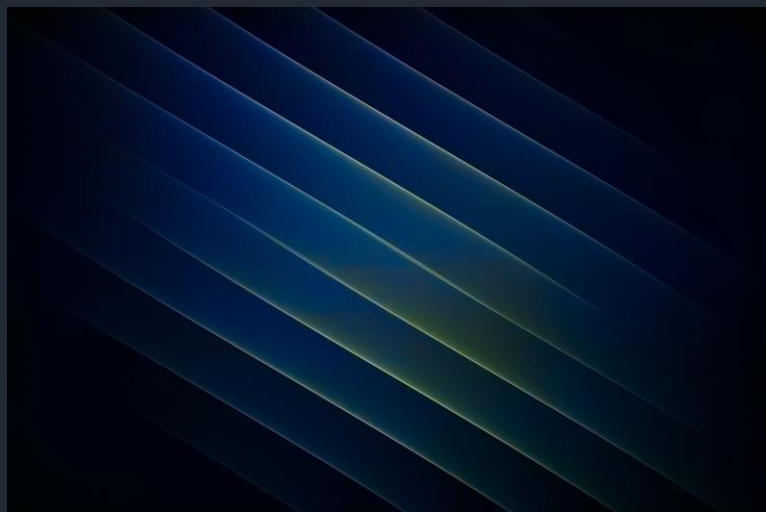
暴力枚举就是枚举所有可能出来

---

多重循环或者DFS都可以用来枚举所有可能

---

无论是哪一种枚举，同时都可以去考虑：找规律，优化，剪枝等手段，降低时间复杂度。



USACO training Section 1.5

PROB Arithmetic Progressions

<https://www.luogu.com.cn/problem/P1214>

先研究示例数据

给定一个数字 ( $\leq 250$ ) 比如  $m$ , 找出小于  $m$  的所有两个数字的平方和所组成的一定长度  $n$  ( $\leq 25$ ) 的等差数列, 输出首项和等差。

TIME LIMIT: 5 secs

输入

5

7

输出:

1 4

37 4

2 8

29 8

1 12

5 12

13 12

17 12

5 20

2 24

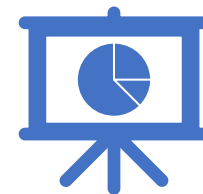
# DFS?

- $m$ 个数的平方和用双循环很容易全部算出来，放到输入数组。问题就变成了从输入数组找 $n$ 个数，他们是一个等差数列。很容易想到和上节课讲的方法类似，用DFS枚举所有的 $n$ 个数的有序排列（不是全排列）并且看他们是不是等差数列。
- 例如，输入5，那么可以得到所有的两个小于5的数的平方和如下（重复不算多）：

0 1 2 4 5 8 9 10 13 16 17 18 20 25 25 26 29 32 34 36 37 40 41 45 49 50 50 52 53 58 61 65  
72 74 85 98

```
int N,M;
vector<int> squarearr;//用set也可以，可以去掉重复，但是麻烦，必要性不是很大
int squarelen=0;
bool noneflag=true;
vector <int> outputnum;
vector<pair<int,int> > ans;
int main() {
    cin >> N >> M;
    //生成输入数组，就是所有的小于M的数两两平方的组合
    for (int i=0;i<=M;i++)
        for (int j=i;j<=M;j++)
            squarearr.push_back(i*i+j*j);
    sort(squarearr.begin(),squarearr.end());//一定要排序

    dfs(-1);
    sort(ans.begin(),ans.end());//答案也要排序，缺省按照first排序，输出先输出second
    for (int i=0;i<ans.size();i++) cout<<ans[i].second<<" "<<ans[i].first<<endl;
    if (noneflag) cout<<"NONE"<<endl;
    return 0;
}
```



```
void dfs(int i)
//i表示输入数组处理到哪里了
{
    if (outputnum.size()==N)
    {
        noneflag=false;
        ans.push_back(make_pair(outputnum[1]-outputnum[0],outputnum[0]));
        //根个数的输据N出数组，把首项和方差插入答案vector数组
        return;
    }

    for (int l=i+1;l<squarearr.size();l++)
    {
        if (l>0 && squarearr[l]==squarearr[l-1]) continue;//去重复，实际上不去重复逻辑也是正确的
        if (outputnum.size()>1 && squarearr[l]-outputnum[outputnum.size()-1]!=outputnum[1]-outputnum[0])
            continue;
        //上面是判断下一个可选的数，必须是和前两项的方差一样。
        outputnum.push_back(squarearr[l]);
        dfs(l);
        outputnum.pop_back();//回溯，因为一个数会被多个排列选择
    }
}
```

# DFS超时了，一定要DFS么？

- 循环暴力枚举，就是从最小的方差开始，从最小的平方数的和作为首项开始，一个个的去尝试。每一次构造不同的等差数列，然后验证后面是否有N-1项。怎么验证？**生成每一项然后去遍历看这些生成的是否都是平方数的和？**
- 注意，前面讲过flag计数方式的标记，有点像visited标记数组，当我们检查一个状态是否存在，或者一个数字是否存在，最好用flag方式时间，判断一个数字是否存在复杂度是 $O(1)$ ，实在用不了flag方式就用set或者map，判断一个数字是否存在时间复杂度是 $(\log n)$



枚举不一定要DFS，可以写成循环。

时间函数就大约是 $O(m^4)$ ，大约10的9次方，但实际上每一次很难都找到n个数，而且逻辑比较简单，所以5s是ok的。

如果红色部分不用flag标记？而是去数组里面找，爆掉。

```
cin >> N >> M;

for (int i=0;i<=M;i++)
    for (int j=i;j<=M;j++)        {
        squarearr.push_back(i*i+j*j);
        flagssquare[i*i+j*j]=1; //设置flag数组
    }
sort(squarearr.begin(),squarearr.end());//一定要排序

int bmax=2*M*M/(N-1)+1;//最大的可能方差
for (int b=1;b<=bmax;b++)    //尝试每一个方差
    for (int i=0;i<squarearr.size()-N+1;i++)//尝试第一项
    {
        if (i<squarearr.size()-1 && squarearr[i+1]!=squarearr[i])//避免重复
        {
            if (squarearr[i]+b*(N-1)>squarearr[squarearr.size()-1]) break;//这句话优化效果有限
            int k=1;
            for (;k<N;k++) if (!flagssquare[squarearr[i]+b*k]) break;
            if(k==N) {
                cout<<squarearr[i]<<" "<<b<<endl;
                noneflag=false;
            }
        }
    }
}
```

USACO training  
Section 1.6  
PROB Number  
Triangles  
luogu: P1216

- 金字塔数字，从顶点（第1行）到最后一行（第R行），每一个数字可以选择下面的1个数字或者右下的1个数字，求途经所有数字和的最大值。  
下列示例数据输出多少？

**5**  
**7**  
**3 8**  
**8 1 0**  
**2 7 4 4**  
**4 5 2 6 5**

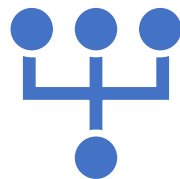
注意：虽然不是“方”的，但还是一个矩阵。只不过每一行的列数不同。有点像“棋盘”例题，但是选择相邻节点的方式不同。

# DFS

- 枚举所有可能。
- 循环不好枚举。
- 联想前面的选数，迷宫等，都是枚举所有可能。选数？每一步选择一个数，然后去选下一个，每次选择根据上次的选择可能都是多个可能的选择（相邻节点）。
- 从上到下，每一行根据前面的选择选不同的数，组成不同的路线，走到底就是一条路线，这个路线经过了每一行，或者说每行选择了一个数。

# 思路-DFS

- 从上往下，每一个数字后面可以选择最多两个下一行的数字（两个分支）之一，这样的话可以构造出从第一行到最后一行的一个分支路线，每一个分支路线代表了一种选择，一个排列，也就是R个数字。核心思路就是枚举所有的可能路线，打擂台找最大。
- DFS搜索，从最上面开始一行一行往下，每一次找下面的和右下的，然后分别再找下一层，每一条路径走到最底下一行都有一个和。所以，这里的相邻节点就是下面和右下两个位置。
- 所以，参数是什么？和上节课的“棋盘”例题有点像。什么状态需要传递给下一步或者说下一行？当前位置的行，列（需要这个来判断下一步往哪里走），还有就是当前已经走过的位置累加的和sum。
- 如果这个累加的和sum作为全局变量，就需要手工回溯：在进入下一个分支的时候，需要把**当前的和**还原回去，因为一条路不可能同时把下面左右两步的数字都加上，只能1个。所以作为参数会好一些。



## DFS

不需要visited数组

因为有序的选择，不会走回去

```
int main() {  
  
    cin >> R;  
    for (int i=1; i<=R; i++)  
    {  
        for (int j=1; j<=i; j++)  
            //列是1到i，不是到R  
            cin >> Rarray[i][j];  
    }  
    findr(1,1,Rarray[1][1]);  
    cout << ans << endl;  
    return 0;  
}
```

```
int R;//行数  
int Rarray[1001][1001];//输入数组  
int ans=INT_MIN;//最终答案  
void findr(int depth,int col,int sumr)  
//depth是深度其实就是行号，col是列，sumr是当前选择的和  
{  
    if (depth==R)//到底了，打擂  
    {  
        ans=max(ans,sumr);  
        return;  
    }  
    //往左下走  
    findr(depth+1,col,sumr+Rarray[depth+1][col]);  
    //往右下走  
    findr(depth+1,col+1,sumr+Rarray[depth+1][col+1]);  
}  
  
findr(1,1,Rarray[1][1]);//从1，1位置开始调用  
cout << ans << endl;//输出结果
```

# 记忆化搜索优化

## 还是超时!

倒着看，比如我们选择第3行的1的时候，只可能从上面的3，8之一过来，从3过来的一条路，肯定没有从8过来的走下去最终结果大，所以一个路线走到1这个数字的时候如果从8过来的已经尝试过了，从3过来的一条路就不需要走了：记忆化搜索就是加入一个数组，记录每一个位置当前走过的最大的值。每一次算出一个sumr的时候都要取判断：

```
if (sumr < flag[depth][col]) return;
```

上面if条件成立表示当前路线到达的这个位置（depth,col）在其他路线已经尝试过，而且其他路线肯定比当前这个路线最后结果要大，当前路线没有必要继续尝试下去了，就return函数。反之，继续逻辑，并且把当前ans赋值给flag数组对应元素。

```
flag[depth][col] = sumr;
```

5

7

3 8

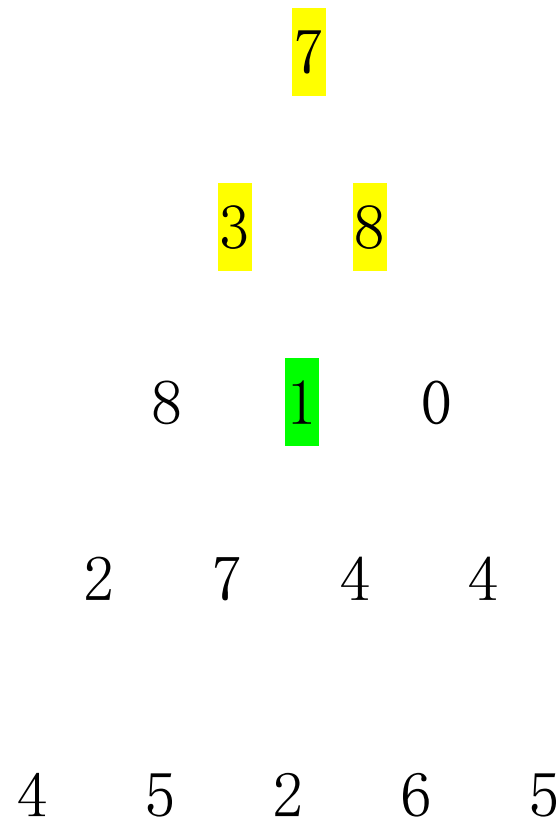
8 1 0

2 7 4 4

4 5 2 6 5

# 为什么超时？

从右图不难看出，到达每一个位置只有上面的两个位置，那么我们的记忆化搜索剪枝的优化其实在每一步也就是减少了一次，减的太少了。比如7->3->1不需要往下走了，但是7->8->1还是要继续。而如果我们先搜索到的7->3->1，后到的7->8->1，那么两个都要往下走。实际上，我们想一下，1无论是从7->3来的还是7->8来的，实际上后面走的路线是一样的，好像没有必要走两次？



先修改一下DFS，  
方便修改记忆化  
搜索。

直接把每一个位  
置到最后一行的  
最大值通过函数  
返回值返回

```
int findr(int depth,int col,int sumr)
{
    //depth is the row number
    if (depth==R)//到底了， 返回当前位置的值
    {
        return Rarray[depth][col];
    }

    //左下位置和右下位置的最大值中的最大加上当前位置的值
    //就是从当前位置走到底的最大
    return Rarray[depth][col]+
    max(findr(depth+1,col,sumr+Rarray[depth+1][col]),
    findr(depth+1,col+1,sumr+Rarray[depth+1][col+1]) );
}

cout<<findr(1,1,Rarray[1][1]);
```



# 记忆化搜索

我们看`max(findr(depth+1,col,sumr+Rarray[depth+1][col]),  
findr(depth+1,col+1,sumr+Rarray[depth+1][col+1]))`;

如果我们把`findr(depth+1,col,sumr+Rarray[depth+1][col])`的结果保存起来呢？从上面过来就不需要再往后走了，这比前面的剪枝少了很多。我们用`int flag[1001][1001];`来保存当前位置到最后一行的最大和。注意和前面剪枝的区别（前面表示从开始到当前位置的最大和）

一起来改写一下。

# 记忆化搜索

```
int flag[1001][1001]; //保存当前位置到最后的最大值。
```

```
int findr(int depth, int col, int sumr)
```

```
{  
    if(flag[depth][col] != -1) return flag[depth][col];  
    //如果计算过最大值就不要再计算了  
    if (depth == R) //到底了， 返回  
    {  
        return flag[depth][col] = Rarray[depth][col];  
    }  
}
```

```
//左下和右下的最大值加上当前位置就是从当前位置走到底的最大
```

```
return flag[depth][col] = Rarray[depth][col] +  
    max(findr(depth+1, col, sumr + Rarray[depth+1][col]),  
        findr(depth+1, col+1, sumr + Rarray[depth+1][col+1]) );  
}
```

```
int main() {  
    cin >> R;  
    for (int i = 1; i <= R; i++)  
        for (int j = 1; j <= i; j++)  
        {  
            cin >> Rarray[i][j];  
            flag[i][j] = -1;  
            //初始化表示没有填充过  
        }  
    cout << findr(1, 1, Rarray[1][1]);  
    return 0;  
}
```

**int findr(int depth, int col, int sumr) 这个 sumr 有必要么？**

sumr没有必要，因为值都在flag数组里面，最后return并没有和sumr相关。

```
int findr(int depth,int col)
{
    if(flag[depth][col]!=-1) return flag[depth][col];//如果计算过最大值就不要再计算了
    if (depth==R)//到底了
    {
        return flag[depth][col]=Rarray[depth][col];
    }

    //往左下走和往右下走的最大值加上当前位置就是从当前位置走到底的最大
    return flag[depth][col]=Rarray[depth][col]+
        max(findr(depth+1,col),
            findr(depth+1,col+1) );
}
```

注意：return flag[depth][col]=Rarray[depth][col];  
等价于 flag[depth][col]=Rarray[depth][col]; return flag[depth][col];

# 反过来从最后一行往前找呢？

- 反过来，这次我们定义dfs函数返回当前传入位置到起点(1,1)的最大值。对应的，记忆化搜索定义int flag[1001][1001];保存每一个位置从当前到起点(1,1)位置的最大值？

- 刚才的

```
return flag[depth][col]=Rarray[depth][col]+max(findr(depth+1,col),findr(depth+1,col+1) );
```

- 就成为了

```
return flag[depth][col]=Rarray[depth][col]+max(findr(depth-1,col),findr(depth-1,col-1) );
```

//当前位置到起点的最大和，就是当前位置的值+ max(左上位置到起点的最大和，正上方到起点的最大和)

- 这个时候我们主程序就需要从第R行的每一个位置去调用dfs并且返回结果打擂台找最大了

# 倒过来的记忆化搜索

```
int findr(int depth,int col)
{
    if(flag[depth][col]!=-1) return flag[depth][col];
    //如果计算过最大值就不要再计算了
    if (depth==1)//到最上面了
    {
        return flag[depth][col]=Rarray[depth][col];
    }

    //往左上走和往上走的最大值加上当前位置就是从当前位置走到头的最大
    return flag[depth][col]=Rarray[depth][col]+
        max(findr(depth-1,col),findr(depth-1,col-1) );
}
```

```
int main() {
    cin >>R;
    for (int i=1;i<=R;i++)
        for (int j=1;j<=i;j++)
        {
            cin>>Rarray[i][j];
            flag[i][j]=-1;//初始化没有填充过
        }
    int ans=-1;
    for (int i=1;i<=R;i++)
        ans=max(ans,findr(R,i,Rarray[R][i]));
    cout<<ans;
    return 0;
}
```


# 固定位置 排序

- 给出一些位置的规则，或者占用一些位置，然后求其他位置如何排列。DFS分支搜索。尝试填空，检测冲突。

# USACO 2019 December Contest, Bronze

## Livestock Lineup

- 八个奶牛，限定了一些奶牛的两两先后顺序（相邻），求可能的最小顺序。
- 基本思路就是枚举出各种可能的顺序（全排列），然后和限定条件做比较。
- 枚举出所有的可能的顺序（全排列），然后去检查是否冲突。
  - 枚举所有可能就是前面讲过的例题，列出各种排列。或者说每一个位置选择一个，然后往下走选择下一个位置，直到所有位置都选好了。非常类似选数。DFS就可以用来枚举所有的可能排列（路线）。
  - 和上节课例题“选数的另外一种做法”非常类似：设置visited数组（例程序叫做findflag数组），尝试填充输出order数组的每一个位置orderindex，从第一个位置开始尝试，把orderindex这个位置依次尝试放入0-7号没有放过位置的奶牛，并且标记这个奶牛放过了，然后去找第orderindex + 1个位置。几个位置都找过了，看看和限定条件是否冲突，不冲突就找到了答案。



# 最后如何判断该排列满足条件

---

- 找到了一个排列`order[]`，比如7 4 3 2 1 5 6 0。第一个位置是奶牛7，第二个奶牛4..... 如何判断是否满足给定的相邻的条件？
- 如果给定了奶牛2 和3必须相邻，设置`flag[2][3],flag[3][2]`都为1。最后我们遍历`order`数组，如果if `(flag[order[i]][order[i+1]]) count++`；也就是`flag[3][2]`为1，那么`count`就是1了。如果给出了`n`个固定要求，那么最后`count==n`的时候就说明条件都满足了。



目标就是一条路选择出一个order数组来，这个order数组表示了排好序的每一个位置是哪一个奶牛

DFS函数（参数：orderindex）

//orderindex表示当前order数组选第几个

{

判断当前是否已经选好了8个位置，如果选好了，判断这个选择是否合适（也就是看是否和规则冲突），不冲突就搞定了；

order数组从0到7（8个奶牛）选择一个没有选择过的作为当前元素也就是

填充order[orderindex];

}

一起看



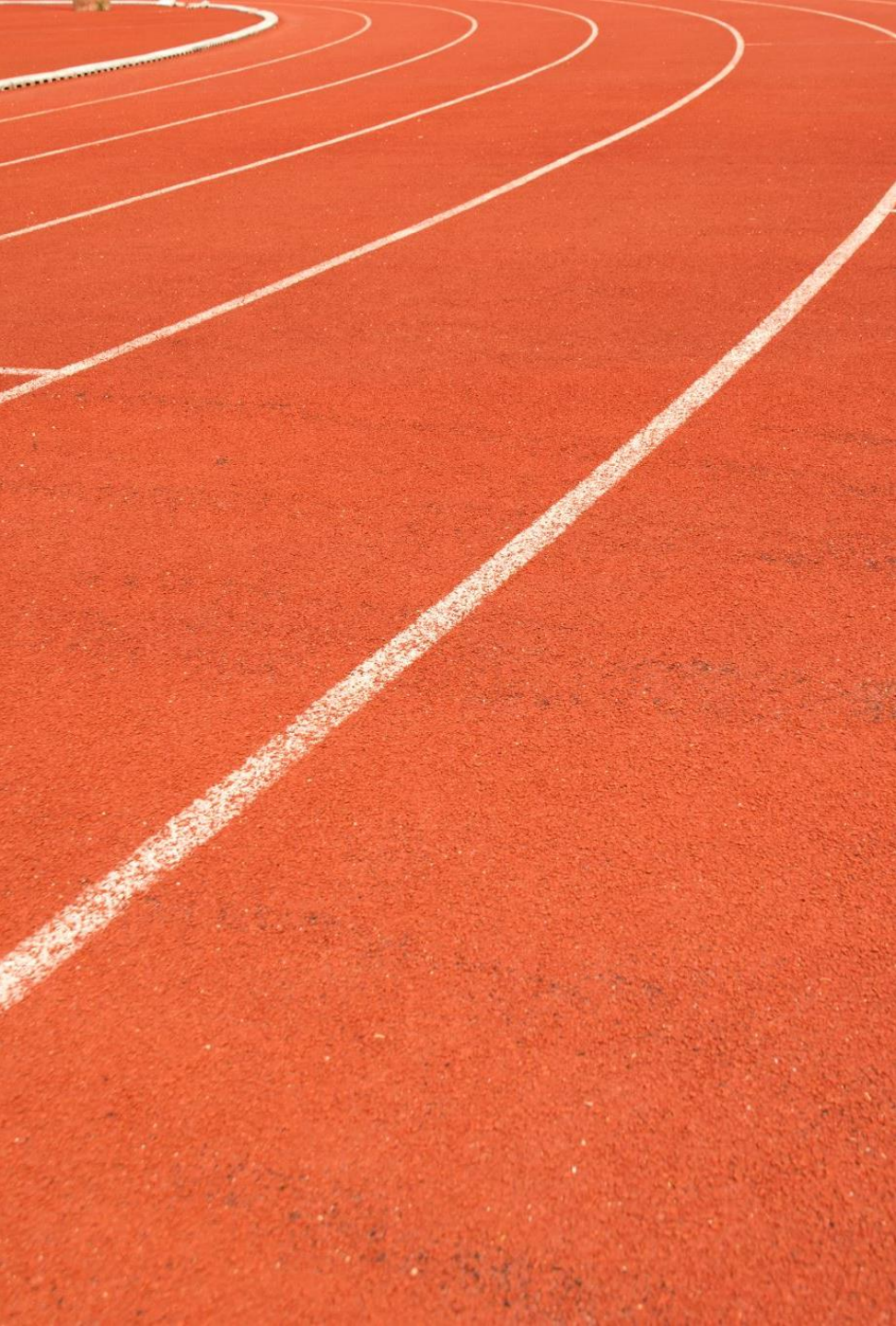


## 小tips

- 有个同学要实现 $i$ 的4次方的累加，比如1的4次方+2的4次方+3的4次方+...+ $n$ 的四次方。 $n$ 最大2000。
- 他发现结果很大，需要用long long。程序如下：

```
int n; cin>>n;  
long long sum=0;  
for (int i=1;i<=n;i++) sum+=i*i*i*i;  
cout<<sum<<endl;
```

结果不对，为什么？



# 作业1 迷宫mig.cpp

---

- <https://www.luogu.com.cn/problem/P1605>

# 作业2:sequence.cpp

全排列问题

<https://www.luogu.com.cn/problem/P1706>

提示: `cout<<setw(5)<<a;` `a`是int变量, 这里就会输出: 数字保留 5 个场宽。你可以输出几个数试一下看看屏幕上什么样子?

由易到难，思维体系训练  
实战结合，创新协作培养  
兴趣导向，未来职业引领

<https://www.35tang.com>



扫码关注公众号

<https://www.三五堂.com>



添加辅导老师