# 8、其他

**程序员面试题精选100题(58)-八皇后问题[算法]**

题目：在8×8的国际象棋上摆放八个皇后，使其不能相互攻击，即任意两个皇后不得处在同一行、同一列或者同一对角斜线上。下图中的每个黑色格子表示一个皇后，这就是一种符合条件的摆放方法。请求出总共有多少种摆法。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

这就是有名的八皇后问题。解决这个问题通常需要用递归，而递归对编程能力的要求比较高。因此有不少面试官青睐这个题目，用来考察应聘者的分析复杂问题的能力以及编程的能力。

由于八个皇后的任意两个不能处在同一行，那么这肯定是每一个皇后占据一行。于是我们可以定义一个数组ColumnIndex[8]，数组中第i个数字表示位于第i行的皇后的列号。先把ColumnIndex的八个数字分别用0-7初始化，接下来我们要做的事情就是对数组ColumnIndex做全排列。由于我们是用不同的数字初始化数组中的数字，因此任意两个皇后肯定不同列。我们只需要判断得到的每一个排列对应的八个皇后是不是在同一对角斜线上，也就是数组的两个下标i和j，是不是i-j==ColumnIndex[i]-Column[j]或者j-i==ColumnIndex[i]-ColumnIndex[j]。

关于排列的详细讨论，详见本系列博客的第28篇，《[字符串的排列](http://zhedahht.blog.163.com/blog/static/254111742007499363479/)》，这里不再赘述。

接下来就是写代码了。思路想清楚之后，编码并不是很难的事情。下面是一段参考代码：

int g\_number = 0;

void EightQueen()

{

    const int queens = 8;

    int ColumnIndex[queens];

    for(int i = 0; i < queens; ++ i)

        ColumnIndex[i] = i;

    Permutation(ColumnIndex, queens, 0);

}

void Permutation(int ColumnIndex[], int length, int index)

{

    if(index == length)

    {

        if(Check(ColumnIndex, length))

        {

            ++ g\_number;

            PrintQueen(ColumnIndex, length);

        }

    }

    else

    {

        for(int i = index; i < length; ++ i)

        {

            int temp = ColumnIndex[i];

            ColumnIndex[i] = ColumnIndex[index];

            ColumnIndex[index] = temp;

            Permutation(ColumnIndex, length, index + 1);

            temp = ColumnIndex[index];

            ColumnIndex[index] = ColumnIndex[i];

            ColumnIndex[i] = temp;

        }

    }

}

bool Check(int ColumnIndex[], int length)

{

    for(int i = 0; i < length; ++ i)

    {

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

        {

            if((i - j == ColumnIndex[i] - ColumnIndex[j])

                || (j - i == ColumnIndex[i] - ColumnIndex[j]))

            return false;

        }

    }

    return true;

}

void PrintQueen(int ColumnIndex[], int length)

{

    printf("Solution %d\n", g\_number);

    for(int i = 0; i < length; ++i)

        printf("%d\t", ColumnIndex[i]);

    printf("\n");

}

**程序员面试题精选**（39）：一道autodesk笔试题求解

去年5月参加Autodesk实习生招聘时的算法题，当时时间紧，题量相当大，没有来得及做这道题，之后也因为懒一直没有做，小崔的答案发给我也没看，今天突然想起来，就做了一下，比想像的要简单一些：）

/\*================================================================================

题目：给定一个M×N矩阵，从左上角元素开始，按照顺时针螺旋状打印所有矩阵元素

关键点：遍历过程的方向控制和左右界控制、上下界控制

作者：sunnyrain

日期：2007.9.3下午

运行环境：vc++ 6.0

==================================================================================\*/

#include<iostream>

#include<vector>

using namespace std;

//定义方向

enum Direction{ToRight,Down,ToLeft,Up};

//改变方向函数

void turnDirect(Direction& cur)

{

switch(cur)

{

case ToRight:

cur = Down;

break;

case Down:

cur = ToLeft;

break;

case ToLeft:

cur = Up;

break;

case Up:

cur = ToRight;

break;

}

}

//遍历矩阵

void traverse(vector<vector<int>>& vv)

{

int m = vv[0].size();

int n = vv.size();

int up=0,down=n,left=0,right=m;

int i=0,j=0;

Direction curD = ToRight;

for(;up !=down || left != right;)

{

switch(curD)

{

case ToRight: //从左向右遍历

for(j=left;j<right;j++)

cout<<vv[i][j]<<" ";

cout<<endl;

--j; //列坐标出界回退

up++; //遍历完一行上界下移

turnDirect(curD); //改变遍历方向

break;

case Down: //从上向下遍历

for(i=up;i<down;i++)

cout<<vv[i][j]<<" ";

cout<<endl;

--i; //出界回退

right--; //遍历完一列右界左移

turnDirect(curD);

break;

case ToLeft: //从右向左遍历

for(j=right-1;j>=left;--j)

cout<<vv[i][j]<<" ";

cout<<endl;

++j; //出界回退

down--; //下界上移

turnDirect(curD);

break;

case Up: //从下向上遍历

for(i=down-1;i>=up;--i)

cout<<vv[i][j]<<" ";

cout<<endl;

++i; //出界回退

left++; //左界右移

turnDirect(curD);

break;

}

}

}

int main()

{

int m,n,i,j,k=0;

cout<<"please input the m and n of the matrix(m\*n):"<<endl;

cin>>m>>n;

vector<vector<int>> vec(n);

//初始化矩阵

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

for(j=0;j<m;j++)

vec[i].push\_back(++k);

//正常遍历矩阵

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

{

for(j=0;j<m;j++)

cout<<vec[i][j]<<"\t";

cout<<endl;

}

//顺时针螺旋遍历矩阵

traverse(vec);

return 0;

}

**程序员面试题精选**（40）：一道SPSS笔试题求解

/\*================================================================================

题目：输入四个点的坐标，求证四个点是不是一个矩形

关键点：

1.相邻两边斜率之积等于-1，

2.矩形边与坐标系平行的情况下，斜率无穷大不能用积判断。

3.输入四点可能不按顺序，需要对四点排序

作者：sunnyrain

日期：2007.9.2 & 2007.9.3

运行环境：vc++ 6.0

==================================================================================\*/

#include<iostream>

#include<limits>

using namespace std;

const double MAX = numeric\_limits<double>::max(); //斜率最大值

class Point

{

float x;

float y;

public:

Point(float \_x = 0, float \_y = 0):x(\_x),y(\_y){}

float getX() const

{

return x;

}

float getY() const

{

return y;

}

//为点重新设置坐标

void set(float \_x,float \_y)

{

x = \_x;

y = \_y;

}

//重载 == 成员操作符

bool operator == (Point& p)

{

return (this->x == p.x && this->y == p.y);

}

//重载流插入操作符

friend istream & operator >> (istream& is, Point & p)

{

return is>>p.x>>p.y;

}

};

class Line //两点形成一条直线/线段

{

Point start;

Point end;

double k; //斜率

public:

Line(){}

Line(Point s,Point e):start(s),end(e)

{

if(start.getX() - end.getX() != 0)

k = (start.getY() - end.getY())/(start.getX() - end.getX()) ;

else

k = MAX; //两点x坐标相等则斜率无穷大

}

double getK()

{

return k;

}

};

//查找数组pp中是否存在点p，是返回数组序号，否返回-1

int findPoint(Point \*pp,int size,Point &p)

{

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

{

if(pp[i] == p)

return i;

}

return -1;

}

//主函数

int main()

{

Point p[4];

Point \*s[4];

int i;

for(i=0; i<4; i++)

{

cout<<"Please input the coordinates of the "<<i+1<<" point in format \"x.xx y.yy\""<<endl;

cin>>p[i];

}

/\* p[0].set(0,0);

p[1].set(1,1);

p[2].set(2,0);

p[3].set(1,-1);\*/

float left,up,right,down;

//获取四点x坐标最大值和最小值

left = p[0].getX(); //left为四点x坐标最小值

right = p[0].getX(); //right为四点x坐标最大值

for(i=1;i<4;i++)

{

if(left>p[i].getX())

left = p[i].getX();

if(right<p[i].getX())

right = p[i].getX();

}

//获取四点y坐标最大值和最小值

up = p[0].getY(); //up为四点y坐标最大值

down = p[0].getY(); //四点y坐标最小值

for(i=1;i<4;i++)

{

if(up<p[i].getY())

up = p[i].getY();

if(down > p[i].getY())

down = p[i].getY();

}

//判断矩形与坐标系平行情况

Point P1(left,up),P2(right,up),P3(right,down),P4(left,down);

if(findPoint(p,4,P1) != -1 && findPoint(p,4,P2) != -1 && findPoint(p,4,P3) != -1 && findPoint(p,4,P4) != -1)

{

cout<<"是矩形"<<endl;

return 0;

}

//按照顺时针方向对四点排序

for(i=0;i<4;i++)

{

if(p[i].getX() == left)

s[0] = &p[i];

else if(p[i].getY() == up)

s[1] = &p[i];

else if(p[i].getX() == right)

s[2] = &p[i];

else if(p[i].getY() == down)

s[3] = &p[i];

}

//排序后的四点顺时针相连组成矩形边

Line one(\*s[0],\*s[1]),two(\*s[1],\*s[2]),three(\*s[2],\*s[3]),four(\*s[3],\*s[0]);

cout<<"k1 = "<<one.getK()<<endl;

cout<<"k2 = "<<two.getK()<<endl;

cout<<"k3 = "<<three.getK()<<endl;

cout<<"k4 = "<<four.getK()<<endl;

//判断相邻边斜率之积是否都等于0

if(one.getK()\*two.getK() == -1 || (one.getK() == 0 && two.getK() == MAX) || (one.getK() == MAX && two.getK() == 0) )

if(two.getK()\*three.getK() == -1 || (two.getK() == 0 && three.getK() == MAX) || (two.getK() == MAX && three.getK() == 0) )

if(three.getK()\*four.getK() == -1 || (three.getK() == 0 && four.getK() == MAX) || (three.getK() == MAX && four.getK() == 0) )

if(four.getK()\*one.getK() == -1 || (four.getK() == 0 && one.getK() == MAX) || (four.getK() == MAX && one.getK() == 0) )

{

cout<<"是矩形!"<<endl;

return 0;

}

cout<<"不是矩形"<<endl;

return 0;

}

**程序员面试题精选** 一道百度面试题题解

问： A厂有1万个工人，编号0-9999，( EE[10000] ), 1个厂长( GG )分派任务, 1个监工( MM )管理工人.厂子忙的时间不确定,可能突然很忙,1天接到任务5000多个,1个任务只能分配给1个工人做, 也可能好几十天没新任务.厂长分配任务给这1万个工人干，按工人编号一个一个来，到最后一个工人就又从头开始，任务完成时间各不相同，可能一个工人在分配 任务的时候手里还有任务， 就得换下一个。

　　但是这1万个工人都很懒，领到了任务先不做，需要监工1个1个去问，如果工人有任务，就 做，如果工人没任务，则不做。厂长只管分任务，1个1个来，可能几天也没新任务，不累; 但是监工很累，监工每天都要看所有工人的情况，即使这些工人都没有任务, 实际上每天工人(80%左右)是没任务的，请问，怎么让监工的工作轻松下来. 比如说每天只问1小半工人.

　　Peak Wong:

　　分析如下：

　　因为“任务完成时间各不相同” ，所以有可能a，b，c某天都有任务，但b的任务最先完成，那么当b的任务完成后，有任务的人的工号可能是不连续的;

　　用一个数组表示1万个工人是否有任务，并保存最后被分配任务的人的工号;

　　1)从前一天“最后被分配任务的人的工号”开始，依次问下一个工号的人，置对应的工作状态，直到碰到前一天无工作，且当天也无工作的人; 并更新当步最后有工作的人的工号为当天的“最后被分配任务的人的工号”;

　　2)从前一天“最后被分配任务的人的工号”开始，依次问上一个工号且前一天有工作的人;

　　问题是监工可以知道那些信息，否则还不是一个一个接着去问。

　　还有就是tailzhou的步骤1消耗的时间T1， 工人完成的时间T2,如果T2

　　所以很多条件都没有限制。

　　可是仔细想想，就是监工要记录所有工人的工作状态，然后每天只查询在工作的工人就可以了。(并且记录谁还在工作中)

　　其实最根本的解决之道是在每次厂长分配任务时，监工也被通知谁被分配了任务。而现在题目的假设是厂长太忙了，来不及通知监工(其实让监工分配就行了)。

　　解决这个问题很简单，监工只要记录(也可能是他猜测)上次厂长分配任务最后分配到那个人就可以了。

　 　然后每天查询时，除了监督前一天来在工作的人外，还要查看从上次分配到任务的编号的下一个编号开始的工人，向后依次查询，知道遇到一个没有被分配工作的 人(这个工人后面不会再有人被分配任务了).同时监工还要记录或猜测哪个工人是当天最后被分配任务的。在“查看从上次分配到任务的编号的下一个编号开始的 工人，向后依次查询”过程中，如果工人的话可信，询问他们就可以了。如果不可信，那么可以猜测为最后一个昨天空闲，今天忙的工人就可以了。

　　其实，监工既然可以监督，他总得有渠道知道当天那个工人还有活要做(不然所有工人都可以说今天我没有任务呀)，所以没有这么复杂的问题。

　　解决问题：

　　有一问：监工是不是问了以后，工人就会一直把工作做完?

　　如果这样的话，最简单的一个考虑，他第一次问的时候记住这个工人今天能否做完，不能做完的话，哪天才能做完。

　　监工实际上只需要在工人做完了以后的第二天去问就可以了。因为不做完，厂长不会给他分新任务。

　　但问题说：但是监工很累，监工每天都要看所有工人的情况，即使这些工人都没有任务那又不一样了，就是说监工必须每天问，不然工人不会开始工作。

　　这时候就是没有工作的工人不需要问。

　　首先工作队列，每个人问一遍，今天能做完的移到空闲队列。

　　空闲队列，二分查找，找到非空闲的，往前处理完，今天能做完的不动，不能做完的移到工作队列。

　　很简单的问题阿。二分查找需要问的人非常少的。

　　绝对能满足楼主说的：比如说每天只问1小半工人，用我的方法厂长分任务的时候不需要通知监工。

　　实际上通知也没有意义，因为是按顺序分配，监工实际上需要知道的是今天有多少新任务。

　　就算不知道，用二分查找10000人也最多需要10几次。

　　这10几次，再根据实际上每天工人(80%左右)是没任务的，所以实际上每天问的人只有20%左右。

　　1.问题分析

　 　现在的情况是监工每天要查看所有的工人，催他们工作，因为不催他们不开工，即要访问EE[10000]的每个元素一次。目标是每天只问一小半的工人，实 际上没有工作的工人是不需要问的，最理想的情况就是监工只问有工作的工人，或者尽可能少地问没有工作的工人，即要尽可能少地访问EE[10000]的元 素。

　　怎么办呢?监工想了一个办法，他做了一万张卡片，每张卡片上写着工人的编号，从0-9999，恰好和数组EE[10000]的下标对应。

　 　监工拿着他的秘密武器上阵了，0号，有工作没?没有。好，放右边口袋。1号，有工作没?有。今天能干完吗?能。好，放右边口袋(并且放在0号后面)。2 号，有工作没?有，今天能干完吗?不能。嗯，放左边。3号，没有。放右边(1号后面)。4号，今天干不完，放左边。……第二天，先看右边(昨天没事的)， 0号，有工作没，有，今天能干完吗，能，好，不动。1号有工作没，有，干不完，好，放左边(接着昨天后面放)。3号，没有，哦，厂长GG还没有分配到这里 阿，那明天检查空的从这里开始就可以了(记住)，但今天仍然轻松不了，因为可能是从后面的号码过来，并且分配到前面来了。右边全部查完了。再查左边。2 号，今天能干完，放右边(并且放在0号后面)。阿，又碰到了1号了，今天的检查结束。第三天，总算可以轻松了。从3号开始，先查右边。今天做得完，不动， 做不完，移到左边后面，碰到没有任务的，或者碰到3号，右边检查完。再看左边，做得完的，移到右边，并且按顺序插入其它卡片中间，做不完的，不动，直到碰 到今天新加入的做不完的，或者整个左边的卡片都检查完。

　　说了这么多，实际上就是左边的是工作的队列，右边的是没工作的队列，左边和右 边的区别就是右边的要保持按编号排序(因为厂长GG分配任务是按顺序分的)。再拿支铅笔，在有的卡片上做做记号。工作轻松不少。以上都是假设工人必须每天 催才会工作，并且监工每天都是在厂长分配任务之后才去催。如果工人催一次就会一直工作，那么简单，监工只需要在卡片上记下还要几天才去问就行了。如果监工 是每天早上去催，厂长去可能在清早(问之前)或下午(问之后，但工人还没工作完)去分任务。

　　如果是前者，当然没问题，如果是后者问题来了，因为3号是今天才能完工的，但也是放在右边，如果厂长刚好分了2号和4号，那么按上面的逻辑，4号就催不到了。

　　所以为了避免这种情况，当天能完的还不能放在空闲里。嗯，只要再准备一个口袋就行了。好在监工的衣服上面有两个口袋，下面也有两个，用了下面两个，上面还有两个没有被使用。拿一个来用就行了。

　 　检查下面的左右口袋时，凡是当天能做完的，都放在左上。OK，先右下，再左下，当天能做完的这会儿放右上，再把左上的按顺序插入右下。应该没问题了吧， 不管厂长何时分任务，监工只需要看自己的口袋就可以了。右下需要访问的是从昨天访问的没工作的开始，再到一个新的没工作的。左下都要访问。右上或左上的卡 片只需要整理。因为实际上每天工人(80%左右)是没任务的，都在右下，并且也可能好几十天没新任务，这下轻松了，只用问小部分工人就能保证工作正常进行 了。

　　好了，如果想先模拟一遍怎么办，用大脑模拟，10000人太多想不过来，累。写个程序吧。要写程序得先有算法。

　　2.算法

　　以下算法模拟最一般的情况，即监工不知道厂长何时分任务，厂长在一天的任何时候都可以分任务，工人每天都要问才工作，监工只在早上去催(为了简化工时的计算，即工时以天为单位)。

　　完全的随机起点模拟，即此方法可从任何监工想要采用此方法的时间点开始。

　　假设一个任务完成的时间是1-N天，厂子一天接到的新任务数是0-M个。用T分钟模拟一天。定时器是精度毫秒级。

　　A.用0-N之间的数初始化EE[10000]，模拟当前工人的工作状态。EE[i]表示工人还要多少天完成这个任务。EE[i]=0，表示没任务。

　　B.设置定时器，厂长分任务定时器为1-T\*60\*1000毫秒之间某个时间，监工定时器设为T\*60\*1000毫秒。

　　C.厂长定时器到，厂长分任务。用c记录厂长从哪里开始。第一次时有个随机初始化的工程，随机一个0-9999之间的数，然后找到第一个EE[i]=0的i，从这个c=i开始分配。

　　随机产生0-M，如果=0，则c=i不变，如果是1-M之间的值，则一个个查，碰到EE[i]=0的，给他随机一个1-N的值，直到分完这些任务，并且c=最后分到的+1。这个题目是研究监工的问题是，厂长比较轻松，下次让他继续从这里找下去。

　 　重新设定厂长定时器，定时为T\*60\*1000-上次定的时，再加上1-T\*60\*1000毫秒之间某个时间,因为厂长也只会一天分一次，所以先要把今 天的时间用完，再加上下一天的某个时间，(从前面可以看出，厂长的定时器设成T也是一样的，只要考虑一下访问共享数据的问题，这里先不考虑这个问题)。

　　D.监工定时器到，监工问工人。

　　新建四个链表。a(今天还不能做完的),b(没有工作的),c(今天可以做完的),d(今天可以做完的)，初始化为空。但b为有序链表。c和d轮流使用。

　　第一次定时到，访问EE[10000]，今天不能做完的(EE[i]＞ 1)接到a的尾巴上，能做完的(EE[i]=1)接到c的尾巴上，没有工作的(EE[i]=0)接b，d空。

　　第二次定时到，访问b，今天不能做完的接到a的尾巴上，能做完的接到d的尾巴上，并且记录出现有工作的人后再出现没有工作的人的结点指针p。如果没有这样的人，那就是链表第一个人或者为空(大家都有工作)。但不管怎样必须把整个链表b访问一遍。

　　访问a,不能做完的不动，能做完的接到d的尾巴上，最后将c中的元素插入b。注意，链表中元素唯一，也就是说移到另一个链表的时候，也意味着从原链表删除。

　　第三次定时到，从p开始访问b中的元素，今天不能做完的接到a的尾巴上，能做完的接到c的尾巴上，直到找到一个没有工作的或者已经全部找了一遍(找 的时候调整p到新的位置)。到链表最后一个后，可能要从头找。

　　访问a,不能做完的不动，能做完的接到c的尾巴上，最后将d中的元素插入b。

　　第四次定时到，和第3次类似，只是c和d的位置对调了一下。到这时，监工的工作已经轻松了，整个系统将按这个新的方式一直运行下去。

　　监工的定时器不需要重新设置。

　　a,b,c,d中的元素内容为工人编号，访问时语法类似if(EE[p-＞ index]＞ 1)。

腾讯部分笔试面试题

1、请定义一个宏，比较两个数a、b的大小，不能使用大于、小于、if语句

2、如何输出源文件的标题和目前执行行的行数

3、两个数相乘，小数点后位数没有限制，请写一个高精度算法

4、写一个病毒

5、有A、B、C、D四个人，要在夜里过一座桥。他们通过这座桥分别需要耗时1、2、5、10分钟，只有一支手电，并且同时最多只能两个人一起过桥。请问，如何安排，能够在17分钟内这四个人都过桥？

2008年腾讯招聘

选择题(60)

c/c++ os linux 方面的基础知识 c的Sizeof函数有好几个!

程序填空(40)

1.(20) 4空x5

不使用额外空间,将 A,B两链表的元素交叉归并

2.(20) 4空x5

MFC 将树序列化 转存在数组或 链表中!

1, 计算 a^b << 2 (运算符优先级问题)

2 根据先序中序求后序

3 a[3][4]哪个不能表示 a[1][1]: \*(&a[0][0]) \*(\*(a+1)+1) \*(&a[1]+1) \*(&a[0][0]+4)

4 for(int i...)

for(int j...)

printf(i,j);

printf(j)

会出现什么问题

5 for(i=0;i<10;++i,sum+=i);的运行结果

6 10个数顺序插入查找二叉树，元素62的比较次数

7 10个数放入模10hash链表，最大长度是多少

8 fun((exp1,exp2),(exp3,exp4,exp5))有几个实参

9 希尔 冒泡 快速 插入 哪个平均速度最快

10 二分查找是 顺序存储 链存储 按value有序中的哪些

11 顺序查找的平均时间

12 \*p=NULL \*p=new char[100] sizeof(p)各为多少

13 频繁的插入删除操作使用什么结构比较合适，链表还是数组

14 enum的声明方式

15 1-20的两个数把和告诉A,积告诉B，A说不知道是多少，

B也说不知道，这时A说我知道了，B接着说我也知道了，问这两个数是多少

大题：

1 把字符串转换为小写，不成功返回NULL,成功返回新串

char\* toLower(char\* sSrcStr)

{

char\* sDest= NULL;

if( \_\_1\_\_\_)

{

int j;

sLen = strlen(sSrcStr);

sDest = new [\_\_\_\_\_\_\_2\_\_\_\_\_];

if(\*sDest == NULL)

return NULL;

sDest[sLen] = '\0';

while(\_\_\_\_\_3\_\_\_\_)

sDest[sLen] = toLowerChar(sSrcStr[sLen]);

}

return sDest;

}

2 把字符串转换为整数 例如："-123" -> -123

main()

{

.....

if( \*string == '-' )

n = \_\_\_\_1\_\_\_\_\_\_;

else

n = num(string);

.....

}

int num(char\* string)

{

for(;!(\*string==0);string++)

{

int k;

k = \_\_2\_\_\_\_\_;

j = --sLen;

while( \_\_3\_\_)

k = k \* 10;

num = num + k;

}

return num;

}

附加题：

1 linux下调试core的命令，察看堆栈状态命令

2 写出socks套接字 服务端 客户端 通讯程序

3 填空补全程序，按照我的理解是添入：win32调入dll的函数名

查找函数入口的函数名 找到函数的调用形式

把formView加到singledoc的声明 将singledoc加到app的声明

4 有关系 s(sno,sname) c(cno,cname) sc(sno,cno,grade)

1 问上课程 "db"的学生no

2 成绩最高的学生号

3 每科大于90分的人数

主要是c/c++、数据结构、操作系统等方面的基础知识。好像有sizeof、树等选择题。填空题是补充完整程序。附加题有写算法的、编程的、数据库sql语句查询的。还有一张开放性问题。

请定义一个宏，比较两个数a、b的大小，不能使用大于、小于、if语句

#define Max(a,b) ( a/b)?a:b

如何输出源文件的标题和目前执行行的行数

int line = \_\_LINE\_\_;

char \*file = \_\_FILE\_\_;

cout<<"file name is "<<(file)<<",line is "<<line<<endl;

两个数相乘，小数点后位数没有限制，请写一个高精度算法

写一个病毒

while (1)

{

int \*p = new int[10000000];

}

不使用额外空间,将 A,B两链表的元素交叉归并

将树序列化 转存在数组或 链表中

struct st{

int i;

short s;

char c;

};

sizeof(struct st);

8

char \* p1;

void \* p2;

int p3;

char p4[10];

sizeof(p1...p4) =?

4，4，4，10

二分查找

快速排序

双向链表的删除结点

有12个小球,外形相同,其中一个小球的质量与其他11个不同

给一个天平,问如何用3次把这个小球找出来

并且求出这个小球是比其他的轻还是重

解答:

哈哈，据说这是微软前几年的一个面试题。很经典滴啊！三次一定能求出来，而且能确定是重还是轻。

数据结构的知识还没怎么学透，不过这个题我到是自己研究过，可以分析下。

将12个球分别编号为a1,a2,a3.......a10,a11,a12.

第一步：将12球分开3拨，每拨4个，a1~a4第一拨，记为b1， a5~a6第2拨，记为b2，其余第3拨，记为b3；

第二步：将b1和b2放到天平两盘上，记左盘为c1，右为c2；这时候分两中情况：

1.c1和c2平衡，此时可以确定从a1到a8都是常球；然后把c2拿空，并从c1上拿下a4，从a9到a12四球里随便取三球，假设为a9到a11，放到c2上。此时c1上是a1到a3，c2上是a9到a11。从这里又分三种情况：

A：天平平衡，很简单，说明没有放上去的a12就是异球，而到此步一共称了两次，所以将a12随便跟11个常球再称一次，也就是第三次，马上就可以确定a12是重还是轻；

B： 若c1上升，则这次称说明异球为a9到a11三球中的一个，而且是比常球重。取下c1所有的球，并将a8放到c1上，将a9取下，比较a8和a11（第三 次称），如果平衡则说明从c2上取下的a9是偏重异球，如果不平衡，则偏向哪盘则哪盘里放的就是偏重异球；

C：若c1下降，说明a9到a11里有一个是偏轻异球。次种情况和B类似，所以接下来的步骤照搬B就是；

2.c1和c2不平衡，这时候又分两种情况，c1上升和c1下降，但是不管哪种情况都能说明a9到a12是常球。这步是解题的关键。也是这个题最妙的地方。

A：c1上升，此时不能判断异球在哪盘也不能判断是轻还是重。取下c1中的a2到a4三球放一边，将c2中的a5和a6放到c1上，然后将常球a9放到c2上。至此，c1上是a1，a5和a6，c2上是a7，a8和a9。此时又分三中情况：

1） 如果平衡，说明天平上所有的球都是常球，异球在从c1上取下a2到a4中。而且可以断定异球轻重。因为a5到a8都是常球，而第2次称的时候c1是上升 的，所以a2到a4里必然有一个轻球。那么第三次称就用来从a2到a4中找到轻球。这很简单，随便拿两球放到c1和c2，平衡则剩余的为要找球，不平衡则 哪边低则哪个为要找球；

2）c1仍然保持上升，则说明要么a1是要找的轻球， 要么a7和a8两球中有一个是重球（这步懂吧？好好想想，很简单的。因为a9是常球，而取下的a2到a4肯定也是常球，还可以推出换盘放置的a5和a6也 是常球。所以要么a1轻，要么a7或a8重）。至此，还剩一次称的机会。只需把a7和a8放上两盘，平衡则说明a1是要找的偏轻异球，如果不平衡，则哪边 高说明哪个是偏重异球；

3）如果换球称第2次后天平平衡打破，并且c1降低了，这说明异球肯定在换过来的a5和a6两求中，并且异球偏重，否则天平要么平衡要么保持c1上升。确定要找球是偏重之后，将a5和a6放到两盘上称第3次根据哪边高可以判定a5和a6哪个是重球；

B： 第1次称后c1是下降的，此时可以将c1看成c2，其实以后的步骤都同A，所以就不必要再重复叙述了。至此，不管情况如何，用且只用三次就能称出12个外 观手感一模一样的小球中有质量不同于其他11球的偏常的球。而且在称的过程中可以判定其是偏轻还是偏重。

给一个奇数阶N幻方，填入数字1，2，3...N\*N,使得横竖斜方向上的和都相同

答案:

#include<iostream>

#include<iomanip>

#include<cmath>

usingnamespace std;

int main()

{

int n;

cin>>n;

int i;

int \*\*Matr=newint\*[n];//动态分配二维数组

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

Matr[ i ]=newint[n];//动态分配二维数组

//j=n/2代表首行中间数作为起点，即1所在位置

int j=n/2,num=1;//初始值

i=0;

while(num!=n\*n+1)

{

//往右上角延升，若超出则用%转移到左下角

Matr[(i%n+n)%n][(j%n+n)%n]=num;

//斜行的长度和n是相等的，超出则转至下一斜行

if(num%n==0)

i++;

else

{

i--;

j++;

}

num++;

}

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

{

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

cout<<setw((int)log10(n\*n)+4)<<Matr[ i][ j ];//格式控制

cout<<endl<<endl;//格式控制

}

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

delete [ ]Matr[ i ];

return1;

}