

HUNAN UNIVERSITY

程序设计训练

报 告

学生姓名 侯茱元

学生学号 201926010214

专业班级 软件1902班

指导老师 陆绍飞

助 教 \_ \_刘园春、汪国庆、吴敏、陈冉龙\_\_\_\_\_\_

2020 年 9 月 5 日

目 录

[题1：数塔 3](#_Toc50323559)

[一、 问题描述 3](#_Toc50323560)

[二、 问题分析及边界条件 3](#_Toc50323561)

[三、 算法设计 3](#_Toc50323562)

[四、 详细设计（从算法到程序） 4](#_Toc50323563)

[五、 样例设计与测试 5](#_Toc50323564)

[六、 分析与总结 6](#_Toc50323565)

[题2：小希的数表 6](#_Toc50323566)

[一、 问题描述 6](#_Toc50323567)

[二、 问题分析及边界条件 7](#_Toc50323568)

[三、 算法设计 7](#_Toc50323569)

[四、 详细设计（从算法到程序） 8](#_Toc50323570)

[五、 样例设计与测试 9](#_Toc50323571)

[六、 分析与总结 10](#_Toc50323572)

[题3：二叉树遍历 10](#_Toc50323573)

[一、 问题描述 10](#_Toc50323574)

[二、 问题分析及边界条件 12](#_Toc50323575)

[三、 算法设计 12](#_Toc50323576)

[四、 详细设计（从算法到程序） 12](#_Toc50323577)

[五、 样例设计与测试 14](#_Toc50323578)

[六、 分析与总结 15](#_Toc50323579)

[题4：蛇形矩阵 15](#_Toc50323580)

[一、 问题描述 15](#_Toc50323581)

[二、 问题分析及边界条件 16](#_Toc50323582)

[三、 算法设计 16](#_Toc50323583)

[四、 详细设计（从算法到程序） 16](#_Toc50323584)

[五、 样例设计与测试 17](#_Toc50323585)

[六、 分析与总结 19](#_Toc50323586)

[题5：错误的里程表 19](#_Toc50323587)

[一、 问题描述 19](#_Toc50323588)

[二、 问题分析及边界条件 20](#_Toc50323589)

[三、 算法设计 20](#_Toc50323590)

[五、 样例设计与测试 21](#_Toc50323591)

[六、 分析与总结 22](#_Toc50323592)

# 题1：数塔

## 问题描述

【问题描述】

给定一个数塔，如下图所示。在此数塔中，从顶部出发，在每一节点可以选择走左下或右下，一直走到底层。请找出一条路径，使路径上的数值和最大。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 9 |  |  |  |  |
|  |  |  | 12 |  | 15 |  |  |  |
|  |  | 10 |  | 6 |  | 8 |  |  |
|  | 2 |  | 18 |  | 9 |  | 5 |  |
| 19 |  | 7 |  | 10 |  | 4 |  | 16 |

【输入形式】

输入时第一行一个整数n，表示该数塔的行数，其余n行表示该塔每行的数值。

【输出形式】

输出包含两行，第一行为最大路径上的数值之和， 第二行n个数字为从上而下最大路径数值。

## 问题分析及边界条件

该题看上去比较简单，逻辑很简单，找到路径最大的即可。但是实际操作却比较复杂。这是一道动态规划的题。因为逻辑较为简单，且每一次计算都具有相似性，所以采用递归的算法较好。

## 算法设计

当这个三角形只有1层时，我们输出这个数就可以了；  
当这个三角形只有2层时，我们输出上一层加上下一层的最大值就可以了。  
…  
这样可以推出一个结论：  
每个位置能够得到的最大值，是上方和左上方中最大值与自己的和。（三角形左对齐）以此类推到第n层，把第n层的数值遍历一遍，找出其中的最大值，输出即可

## 详细设计（从算法到程序）

#include <iostream>

#include <algorithm>

using namespace std;

const int N = 50;

int data[N][N];//存储数塔原始数据

int dp[N][N];//存储动态规划过程中的数据

int n;//塔的层数

/\*动态规划实现数塔求解\*/

void tower\_walk()

{

// dp初始化

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

{

dp[n - 1][i] = data[n - 1][i];

}

int temp\_max;

for (int i = n - 1; i >= 0; --i)

{

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

{

// 使用递推公式计算dp的值

temp\_max = max(dp[i + 1][j], dp[i + 1][j + 1]);

dp[i][j] = temp\_max + data[i][j];

}

}

}

/\*打印最终结果\*/

void print\_result()

{

cout<< dp[0][0] << '\n';

int node\_value;

// 首先输出塔顶元素

cout << data[0][0];

int j = 0;

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

{

node\_value = dp[i - 1][j] - data[i - 1][j];

/\* 如果node\_value == dp[i][j]则说明下一步应该是data[i][j]；如果node\_value == dp[i][j + 1]则说明下一步应该是data[i][j + 1]\*/

if (node\_value == dp[i][j + 1]) ++j;

cout << "->" << data[i][j];

}

cout << endl;

}

int main()

{

cin >> n;

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

{

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

{

cin >> data[i][j];

}

}

tower\_walk();

print\_result();

}

## 样例设计与测试

【样例输入1】

5

9

12 15

10 6 8

2 18 9 5

19 7 10 4 16

【样例输出1】

59

9 12 10 18 10

【样例输入2】

3

8

15 16

11 18 19

【样例输出2】

43

8 16 19

【样例输入3】

4

8

4 5

6 9 7

8 1 2 4

【样例输出3】

26

8 4 6 8

## 分析与总结

该题存在的遗憾就是还未想到在递归的方法下找出记录路径的方法。

通过该题，练习了解决动态规划题目的方法。并且还运用了一些离散数学的知识，很有收获。

# 题2：小希的数表

## 问题描述

【问题描述】

Gardon 昨天给小希布置了一道作业，即根据一张由不超过 5000 的 N(3<=N<=100)个正整数组成的数表两两相加得到 N\*(N-1)/2 个和，然后再将它们排序。例如，如果数表里含有四个数 1，3，4，9，那么正确答案是 4，5，7，10，12，13。小希做完作业以后出去玩了一阵，可是下午回家时发现原来的那张数表不见了，好在她做出的答案还在，你能帮助她根据她的答案计算出原来的数表么？

【输入形式】

包含多组数据，每组数据以一个 N 开头，接下来的一行有按照大小顺序排列的 N\*(N-1)/2 个数，是小希完成的答案。文件最后以一个 0 结束。  
假设输入保证解的存在性和唯一性。

【输出形式】

对于每组数据，输出原来的数表。它们也应当是按照顺序排列的。

## 问题分析及边界条件

该问题本质是一个非齐次线性方程组的求解问题。显然，输入的最小的一个数和倒数第二小的数，必定是a1和a2、a1和a3的和，如果找到a2和a3的和，就可以求出这三个数，从而求出剩下的a4、a5······an

## 算法设计

由以上分析可知，最小的一个数和倒数第二小的数，必定是a1和a2、a1和a3的和，但是a2和a3的和不确定，因为题目确定只有一个解，所以，必定只有一个数可以得出解，所以使用穷举的方法，在剩下的数里找出符合的，即a2与a3的和。之后便可求出a1、a2、a3。之后再从数表中将只包含a1、a2、a3的和去掉，那么剩下的最小的必定是a1与a4的和，然后再将只包含a1、a2、a3、a4的和从数表中去掉，剩下最小的就是a1与a5的和，然后以此类推，就可得到所有的数。

for(int i=2;i<M;i++)

{

for(int j=0;j!=M;j++)

mark[j]=false;

num[1]=(sum[0]-sum[1]+sum[i])/2; //三个不等式联立解得

num[0]=sum[0]-num[1];

num[2]=sum[1]-num[0];

if(num[1]+num[2]!=sum[i])

continue;

mark[0]=true;

mark[1]=true;

mark[i]=true; //将被加过的结果标记为true，此时num[0]num[1]num[2]已经确定

bool flag=true;

for(int j=3;j<N&&flag;j++)

{

int k=0;

while(mark[k]) //找到第一个没被标记的数，减去num[0]即为num[j]

k++;

num[j]=sum[k]-num[0];

mark[k]=true;

for(int l=1;l<j&&flag;l++)

{

flag=false;

for(int m=k+1;m<M;m++)

if(!mark[m]&&num[l]+num[j]==sum[m]){

flag=true;

mark[m] = true; //每确定一个num，将其与前面的num相加，并将结果在mark中标记，如果没有找到对应的sum，则说明不符合情况故break

break;

}

}

}

if(flag)

break;

}

## 详细设计（从算法到程序）

#include<iostream>

using namespace std;

int main()

{

int N;

while(cin>>N)

{

if(N==0)

break;

int M=N\*(N-1)/2;

int sum[M];

int num[N];

bool mark[M];

for(int i=0;i!=M;i++)

cin>>sum[i];

for(int i=2;i<M;i++)

{

for(int j=0;j!=M;j++)

mark[j]=false;

num[1]=(sum[0]-sum[1]+sum[i])/2; //三个不等式联立解得

num[0]=sum[0]-num[1];

num[2]=sum[1]-num[0];

if(num[1]+num[2]!=sum[i])

continue;

mark[0]=true;

mark[1]=true;

mark[i]=true; //将被加过的结果标记为true，此时num[0]num[1]num[2]已经确定

bool flag=true;

for(int j=3;j<N&&flag;j++)

{

int k=0;

while(mark[k]) //找到第一个没被标记的数，减去num[0]即为num[j]

k++;

num[j]=sum[k]-num[0];

mark[k]=true;

for(int l=1;l<j&&flag;l++)

{

flag=false;

for(int m=k+1;m<M;m++)

if(!mark[m]&&num[l]+num[j]==sum[m]){

flag=true;

mark[m] = true; //每确定一个num，将其与前面的num相加，并将结果在mark中标记，如果没有找到对应的sum，则说明不符合情况故break

break;

}

}

}

if(flag)

break;

}

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

cout<<num[i];

if(i!=N-1)

cout<<' ';

}

cout<<endl;

}

return 0;

}

## 样例设计与测试

【样例输入1】

4

4 5 7 10 12 13

4

5 6 7 8 9 10

0

【样例输出1】

1 3 4 9

2 3 4 6

【样例输入2】

3

5 6 9

0

【样例输出2】

1 4 5

【样例输入3】

4

4 6 8 10 12 14

0

【样例输出3】

1 3 5 9

## 分析与总结

该题考查了递归函数的使用，并不复杂，逻辑清楚之后就比较简单了

# 题3：二叉树遍历

## 问题描述

【问题描述】

二叉树是一种非常重要的[数据结构](http://lib.csdn.net/base/datastructure)，非常多其他数据结构都是基于二叉树的基础演变而来的。对于二叉树，深度遍历有前序、中序以及后序三种遍历方法。

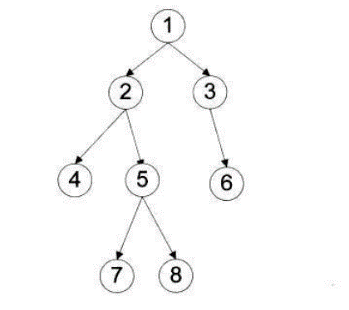
三种基本的遍历思想为：

前序遍历：根结点 ---> 左子树 ---> 右子树

中序遍历：左子树---> 根结点 ---> 右子树

后序遍历：左子树 ---> 右子树 ---> 根结点

比如，求以下二叉树的各种遍历



前序遍历：1  2  4  5  7  8  3  6

中序遍历：4  2  7  5  8  1  3  6

后序遍历：4  7  8  5  2  6  3  1

需要你编写程序解决的问题是：已知一个二叉树的前序遍历和中序遍历的结果，给出该二叉树的后序遍历的结果。

【输入形式】

有多组测试数据，每组测试数据三行，每组测试数据第一行只有一个正整数n，表示二叉树节点的数目，n=0意味着输入结束并且不需要处理。

每组测试数据第二行是二叉树的前序遍历的结果，是一个长度为n的字符串，每个节点由一个字符表示，字符是大小写英文字母及10个数字,不同的节点用不同的字符表示，也即无论前序遍历和中序遍历的字符串中没有重复的字符。

每组测试数据第二行是二叉树的中序遍历的结果，也是一个长度为n的字符串。

40%的测试数据1 ≤ n≤ 10；

30%的测试数据1 ≤ n≤ 20；

20%的测试数据1 ≤ n≤ 40；

10%的测试数据1 ≤ n≤ 62；

【输出形式】

对于每组测试数据，输出一行，是一个长度为n的字符串，表示二叉树后序遍历的结果。

## 问题分析及边界条件

由题目中给出的提示，根节点在前序遍历的第一个，在中序遍历的中间，在后序遍历的最后一个。而对于中序遍历来说，左边为左子树，右边为右子树，对于每个子树来说，他们的遍历同样有之前的规律。所以，可以运用递归的方法找到根节点来完成这个题目。

## 算法设计

由以上分析可知，重点是找到每棵树的根节点，把子树都看成一棵新的数即可。在只要在中序遍历中寻找到与前序遍历第一个数字相同的数字，即可确定根节点，然后在前序遍历中找到中序遍历根节点左边的数字，即可确定子树的根节点······以此类推，最后可以写出后序遍历。

## 详细设计（从算法到程序）

#include <iostream>

using namespace std ;

struct BiNode

{

char data ;

BiNode \*lchild , \*rchild ;

} ;

BiNode \*BiTree ;

int NodeID ;

BiNode \*CreateBiTree (char \*c , int n)

{

BiNode \*T ;

NodeID ++ ;

if (NodeID > n)

{

return (NULL) ;

}

if (c[NodeID] == '0')

{

return (NULL) ;

}

T = new BiNode ;

T -> data = c[NodeID] ;

T -> lchild = CreateBiTree (c , n) ;

T -> rchild = CreateBiTree (c , n) ;

return (T) ;

}

void PreOrderTraverse (BiNode \*T)

{

if (T)

{

cout << T -> data << " ";

PreOrderTraverse (T -> lchild) ;

PreOrderTraverse (T -> rchild) ;

}

}

void InOrderTraverse (BiNode \*T)

{

if (T)

{

InOrderTraverse (T -> lchild) ;

cout << T -> data << " ";

InOrderTraverse (T -> rchild) ;

}

}

void PostOrderTraverse (BiNode \*T)

{

if (T)

{

PostOrderTraverse (T -> lchild) ;

PostOrderTraverse (T -> rchild) ;

cout << T -> data << " ";

}

}

int main ()

{

int i , SampleNum ;

char c[100] ;

cin >> SampleNum ;

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

{

cin >> c[i] ;

}

NodeID = 0 ;

BiTree = CreateBiTree (c , SampleNum) ;

PreOrderTraverse (BiTree) ;

cout << endl ;

InOrderTraverse (BiTree) ;

cout << endl ;

PostOrderTraverse (BiTree) ;

return 0 ;

}

## 样例设计与测试

【样例输入1】

8

12457836

42758136

4

abcd

abcd

4

abcd

dcba

0

【样例输出1】

47852631

dcba

dcba

【样例输入2】

9

zsho7pwmG

so7hpzmwG

6

gNkoVO

kNVogO

0

【样例输出2】

7ophsmGwz

kVoNOg

【样例输入3】

1

i

i

7

0DYV4lJ

04VYDJl

0

【样例输出3】

i

4VYJlD0

## 分析与总结

该题需要有一定的离散数学知识。除此之外还需要熟练掌握递归函数。

# 题4：蛇形矩阵

## 问题描述

【问题描述】蛇形矩阵是由 1 开始的自然数依次排列成的一个矩阵上三角形

【输入形式】 正整数 N表示层数，N 不大于 100

【输出形式】输出一个 N 行的蛇形矩阵，矩阵三角中同一行的数字用一个空格分开，行尾不要多余的空格。

## 问题分析及边界条件

要实现以上程序需要定义int数据类型以及整型数组，用for循环和递归算法来实现蛇形矩阵，并且只用一个主函数来实现

## 算法设计

int start = 1; // 起始值

int temp = size;

for (int count = 0; count < size / 2; count++) // size阶的矩阵可以画size/2个圈

{

for (; col < temp - 1; col++)

matrix[row][col] = start++;

for (; row < temp - 1; row++)

matrix[row][col] = start++;

for (col = temp - 1; col > count; col--)

matrix[row][col] = start++;

for (row = temp - 1; row > count; row--)

matrix[row][col] = start++;

// 进入下一圈

temp--;

row++;

start -= 1; // 这里-1是因为在换圈的时候会多加1

}

if (0 != size % 2) // 如果size为奇数则最后会有一个数遍历不到，这里补上

matrix[row][col + 1] = start + 1;

。

## 四、 详细设计（从算法到程序）

#include <iostream>

#include <iomanip>

using namespace std;

// 输出螺旋矩阵

void Matrix()

{

const int size = 10; // 矩阵大小

int matrix[size][size] = { 0 };

int row = 0;

int col = 0;

int start = 1; // 起始值

int temp = size;

for (int count = 0; count < size / 2; count++) // size阶的矩阵可以画size/2个圈

{

for (; col < temp - 1; col++)

matrix[row][col] = start++;

for (; row < temp - 1; row++)

matrix[row][col] = start++;

for (col = temp - 1; col > count; col--)

matrix[row][col] = start++;

for (row = temp - 1; row > count; row--)

matrix[row][col] = start++;

// 进入下一圈

temp--;

row++;

start -= 1; // 这里-1是因为在换圈的时候会多加1

}

if (0 != size % 2) // 如果size为奇数则最后会有一个数遍历不到，这里补上

matrix[row][col + 1] = start + 1;

// 输出数组

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

{

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

{

cout << setw(5) << matrix[i][j];

}

cout << endl;

}

}

int main(int argc, char \*\*argv)

{

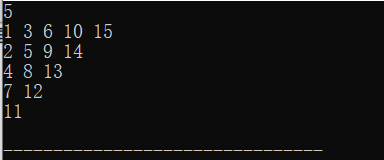
Matrix();

return 0;

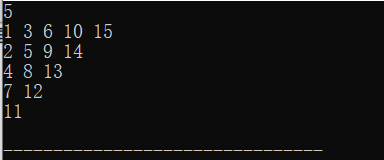
}

## 五、 样例设计与测试

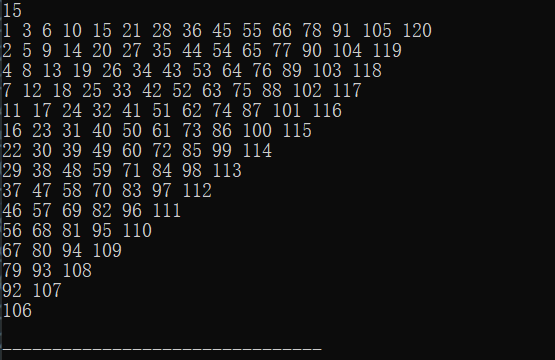
样例输入1：



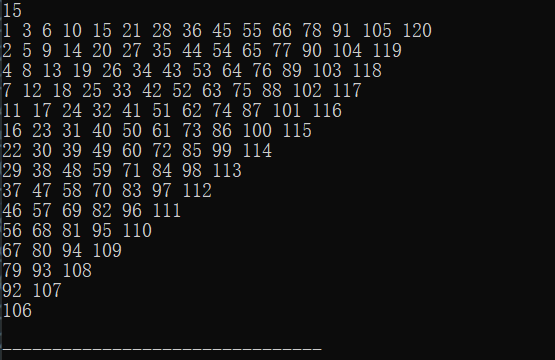
样例输出1：



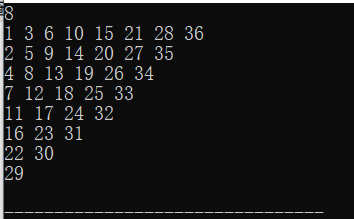
样例输入2：



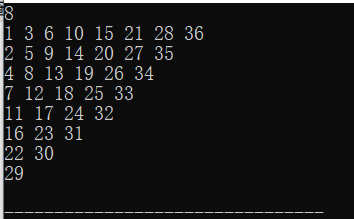
样例输出2



样例输入3：



样例输出3：



## 六、 分析与总结

1．程序的输入输出运用的都是整型，并且用for循环和数组来实现，当输入n(n大于0)的数程序开始执行1号for循环，知道i=n结束1号for循环。然后执行2号for循环，逐一把数打印出来当i=n时结束程序。

# 题5：错误的里程表

## 问题描述

【问题描述】

三月八日，小明买了台新车。但很快小明发现汽车的里程表有问题：里程表上每一位都不显示数字3和数字8，也就是说直接从数字2跳到数字4，直接从数字7跳到数字9。小明纳闷：这车到底行驶里程是多少。

现在，小明向你求助：根据里程表显示的数字，给出真实的行驶里程。

【输入形式】

输入有多组测试数据。

输入第一行正整数T，表示有多少组测试数据。

后面有T行，每行一个非负整数，表示里程表显示数字，里面不含有数字3和8。该数字不超过10位。

40%的测试数据组数T  10≤T≤ 102；

30%的测试数据组数T  102≤T≤ 103；

20%的测试数据组数T  103≤T≤ 104；

10%的测试数据组数T  104≤T≤ 105

【输出形式】

对于每组测试数据，输出一个整数占一行：真实的行程里程。

## 二、 问题分析及边界条件

该问题看上去比较复杂，但实际上就是一个进制转换问题，将八进制转化为十进制即可，需要注意的是4、5、6、7对应的是3、4、5、6而9对应的是7.

## 三、 算法设计

由上面的分析可知，只需要将这个八进制数转化为十进制即可，将这个数的每一位依次取出，乘对应的8的n次方，然后再加起来即可，需要注意4以后的数字的对应问题。

**四、详细设计（从算法到程序）**

#include<bits/stdc++.h>

using namespace std;

int return1\_index(int n) {

int s;

if(n>3&&n<8)

s=n-1;

else if(n>8)

s=n-2;

else s=n;

return s;

}

int main() {

int x;

cin>>x;

while(x–) {

long long int num,a,n,k=0,sum=0;

cin>>num;

n=num;

while(n>0) {

a=n%10;

a=return1\_index(a);

n=n/10;

sum=sum+a\*pow(8,k);

k++;

}

cout<<sum<<endl;

}

return 0;

}

## 五、 样例设计与测试

【样例输入1】

6

0

1

12

159

111224459

124567976

【样例输出1】

0

1

10

103

19212007

21913077

【样例输入2】

5

6

569

47661

121211

11001

【样例输出2】

5

303

15721

41609

4609

【样例输入3】

3

66666

555555

4444444

【样例输出3】

23405

149796

898779

## 六、 分析与总结

通过这道题复习了进制的转换问题，很有收获。