

**数学与信息学院学生实验报告**

**实验课程名称：** 算法分析与设计基础 **教师：\_\_**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验项目名称** | **实验一递归算法设计与应用** | | | **实验成绩** |  |
| **学生姓名** |  | **学 号** | **133** | **年级专业班级** |  |
| **小组成员** | **无** | | | **实验日期** | **2019年3 月** |

# 1. 实验目的和要求

## 1.1 实验目的

① 加深对**递归算法**的理解，并针对具体问题设计算法；

② 分析算法的复杂性，寻找比较高效的算法，并实现。

③ 分析**格雷码**问题，并设计递归算法求解之。

## 1.2 实验软硬件环境

① 操作系统

Win10系统

② 编译环境

Java

## 1.3 实验要求

① 问题描述

对于给定的正整数n，格雷码为满足如下条件的一个编码序列：

(1) 序列由2n个编码组成，每个编码都是长度为n的二进制位串。

(2) 序列中无相同的编码。

(3) 序列中位置相邻的两个编码恰有一位不同。

例如：n=2时的格雷码为：{00, 01, 11, 10}。

设计求格雷码的递归算法并实现。

② 具体要求

输入：输入的第一行是一个正整数m，表示测试例个数。接下来几行是m个测试例的数据，每个测试例的数据由一个正整数n组成。

输出：对于每个测试例n，输出2n个长度为n的格雷码。（为方便查看，在每个格雷码内，两个位之间用一个空格隔开，如，00输出为：0 0）。两个测试例的输出数据之间用一个空行隔开，最后一个测试例后无空行。

③ 测试数据和结果

输入：2

4

5

输出：

0 0 0 0

0 0 0 1

0 0 1 1

0 0 1 0

0 1 1 0

0 1 1 1

0 1 0 1

0 1 0 0

1 1 0 0

1 1 0 1

1 1 1 1

1 1 1 0

1 0 1 0

1 0 1 1

1 0 0 1

1 0 0 0

0 0 0 0 0

0 0 0 0 1

0 0 0 1 1

0 0 0 1 0

0 0 1 1 0

0 0 1 1 1

0 0 1 0 1

0 0 1 0 0

0 1 1 0 0

0 1 1 0 1

0 1 1 1 1

0 1 1 1 0

0 1 0 1 0

0 1 0 1 1

0 1 0 0 1

0 1 0 0 0

1 1 0 0 0

1 1 0 0 1

1 1 0 1 1

1 1 0 1 0

1 1 1 1 0

1 1 1 1 1

1 1 1 0 1

1 1 1 0 0

1 0 1 0 0

1 0 1 0 1

1 0 1 1 1

1 0 1 1 0

1 0 0 1 0

1 0 0 1 1

1 0 0 0 1

1 0 0 0 0

# 2. 实验记录

2.1 理解格雷码

首先通过题目分析格雷码的规律序列中无相同的编码。 序列中位置相邻的两个编码恰有一位不同。通过百度发现格雷码n位与n-1位之间的关系。

1.1位格雷码有两个码字

2.(n+1)位格雷码中的前2n个码字等于n位格雷码的码字，按顺序书写，加前缀0

3.(n+1)位格雷码中的后2n个码字等于n位格雷码的码字，按逆序书写，加前缀1 

4.n+1位格雷码的集合 = n位格雷码集合(顺序)加前缀0 + n位格雷码集合(逆序)加前缀1

## 2.2 实验过程

### 2.2.1 实验思路

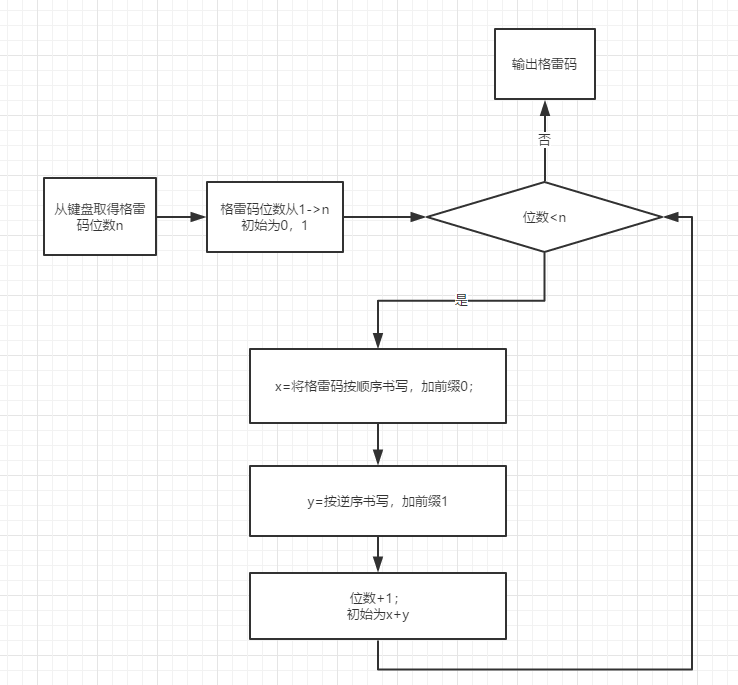


图1 程序框架

### 2.2.2 程序代码

**package** lab;

**import** java.util.Scanner;

**publicclass** lab {

String gray(**int**x,**int**y,String result){

String n=**new** String();

String m=**new** String();

**if**(x<y)

{

String[] a=result.split("/");

**for**(**int**i=0;i<Math.*pow*(2, x);i++)

{

n+="0 "+a[i]+"/";

m+="/"+a[i]+" 1";

}

**return** gray(x+1,y,n+**new** StringBuffer(m).reverse());

}

**else**

{

String[] a=result.split("/");

**for**(**int**i=0;i<Math.*pow*(2, x);i++)

{

System.***out***.println(a[i]);

}

**return**result;

}

}

**publicstaticvoid** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

**int**t = scan.nextInt();

System.***out***.println("测试次数："+t);

**int**n=1;

**while**(t!=0)

{ n = scan.nextInt();

System.***out***.println("输入数据："+n);

lab test =**new** lab ();

test.gray(1,n,"0/1");

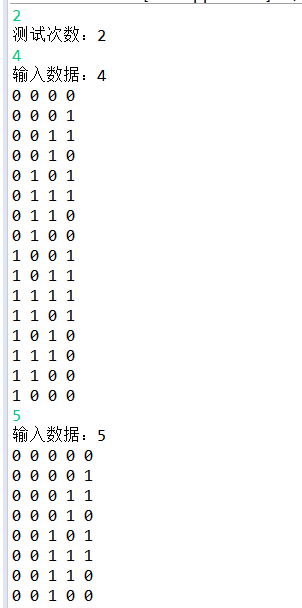
t--;

}

}

}

## 2.3 实现结果



顺利输出，符合实验预期。

# 3. 实验总结

直接找格雷码的规律有点蒙，通过百度发现规律。一开始选择用C语言来编程。发现数组的大小不能根据输入的位数来设定，且字符串操作很繁琐。后来改用JAVA。知道了格雷码n位与n-1位之间的关系，设计出了

String gray(**int**x,**int**y,String result)

x:格雷码当前位数；

y:格雷码目标位数；

result:格雷码当前结果

因为result是字符串，但是处理的时候又要对每一个码进行处理，这里用字符串的spilt方法。每个格雷码之间用/来分隔。

如：一位的格雷码 result是“0/1”

用spilt可以将result分成0和1，并且生成一个数组来分别装它们。

把数组进行处理后再加成串；

然后进行递归；

这次实验加深了对递归的理解，通过百度等各种途径，更深入的理解的递归。递归分为递推和回归两个阶段。可以在递推的阶段进行对问题的一步步的解决，也可以在回归的阶段进行解决。我的代码的在递推的时候解决的，递推到X==Y时，问题就解决了。然后从最里层不断的把答案进行回归。

看了别人的代码后，我发现他们设计的递归在回归的时候进行问题解决，让我对递归的理解更深刻了。他先一直调用自己把格雷码位数递推到1，然后让result是0 ，1。然后对result处理，处理完带着结果回到上一层。回到最外层的时候问题解决。