

**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： 林鑫泓 \_\_**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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** | **148** | **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：windows10操作系统

② 编译环境：CodeBlock（使用C++）

## 1.3 实验要求

① 问题描述

本次实验主要要求解决格雷码问题，给出一个数字n，要求你编写2^n长度的格雷码编码。格雷码的概念在下文中提。例如，当n为2时，需要输出：

0 0

0 1

1 1

1 0

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。

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2^n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。

③ 测试数据和结果

测试数据：

4

结果：

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

# 2. 实验记录

## 2.1 递归和格雷码的概念

### 2.1.1递归的概念

① 递归含义

递归的基本含义：递归是一种重要的程序设计方法。递归指算法自己调用自己, 有直接递归与间接递归两种。递归方法用于解决一类满足递归关系的问题。即：对原问题的求解可转化为对其性质相同的子问题的求解。在平时比赛中我通常会遇到很多递归类的题目，使用递归最大的优点是形式简单，易写，通常在和迭代复杂度差不多的情况下推荐使用迭代（例如并查集），一般情况下递归复杂度较高。

② 实现方式

递归是最常用的思想，在比赛中经常用到。比如dfs、辗转相除法、并查集都会用到递归的写法。递归一般通过自己调用自己，设置一个临界值返回来达到目的。在编写递归时一定要确定返回的情况，否则容易超时和爆栈。

递归的写法一般类似以下形式：

**int dfs(int u) {**

**if (u >= 10)**

**return u;**

**else return dfs(u + 1);**

**}**

### 2.1.2 格雷码概念

① 格雷码基本含义

在一组数的编码中，若任意两个相邻的代码只有一位二进制数不同，则称这种编码为格雷码（Gray Code），另外由于最大数与最小数之间也仅一位数不同，即“首尾相连”，因此又称循环码或反射码。

②格雷码的递归实现方法

格雷码有规律就是除去最高位，剩余位数上下对称。递归的思路就是n位各格雷码是由n-1位格雷码加0或1生成，比如求n=3的格雷码，首先知道n=2的格雷码是(00,01,11,10)，那么n=3的格雷码其实就是对n=2的格雷码首位添加0或1生成的，添加0后变成(000,001,011,010)，添加1后需要顺序反向就变成(110,111,101,100)，组合在一起就是n的格雷码(000,001,011,010,110,111,101,100)。

## 2.2 实验过程

### 2.2.1 实验思路

在上文中我们讲到了格雷码的递归方法。这里采用递归，当n不为1时，我们就进入下一个递归（因为此时我们不能确定格雷码，我们只知道1的格雷码是多少），直到n == 1。当n == 1时我们知道0表示为1,1表示为0，返回这两个字符串组。然后不断返回，在串前加上0、1，形成新的格雷码。由于格雷码的特性，我们可以从i-1位得到i位格雷码的编码，那么从1位一直返回到n位就是最终答案。

### 2.2.2 程序代码

Main.cpp

**#include<cstdio>**

**#include<vector>**

**#include<string>**

**#include<iostream>**

**#include<algorithm>**

**typedef long long ll;**

**using namespace std;**

**const int maxn = 500 + 10;**

**const int INF = 0x3f3f3f3f;**

**vector<string> g; //储存答案**

**vector<string> gray(int n) {**

**vector<string> gg;**

**if(n == 1){**

**gg.push\_back("0");**

**gg.push\_back("1");**

**return gg;**

**}**

**else{**

**vector<string> pre = gray(n - 1);**

**for(int i = 0; i < pre.size(); i++){//正着加0**

**gg.push\_back("0" + pre[i]);**

**}**

**for(int i = pre.size() - 1; i >= 0; i--){//反着加1**

**gg.push\_back("1" + pre[i]);**

**}**

**return gg;**

**}**

**}**

**int pmul(int a, int b){ //快速幂**

**int ret = 1;**

**while(b){**

**if(b & 1) ret \*= a;**

**a = a \* a;**

**b >>= 1;**

**}**

**return ret;**

**}**

**int main() {**

**int n;**

**while (~scanf("%d", &n)) {**

**g = gray(n);**

**int Size = pmul(2, n); //快速幂得到总个数2^n**

**for(int i = 0; i < Size; i++){**

**for(int len = 0; len < n; len++){ //输出答案**

**cout << (len == 0? "" : " ") << g[i][len];**

**}**

**cout << endl;**

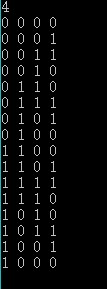
**}**

**}**

**return 0;**

**}**。

## 2.3 实现结果



答案经过验算符合试验预期。因为使用了C++的vector和string，所以不用担心数组越界等情况。

# 3. 实验总结

由于打了一年的ACM，因为vector和string的复杂度较高常常弃之不用，所以对vector和string的使用不是很熟悉，谷歌了一堆才了解用法。格雷码的概念很好理解，但是格雷码的编码规则不是很容易想，看了很久都没发现有什么规律。谷歌了一下知道了格雷码的一种做法：直接从一个数的二进制n转换为格雷码，把一个数二进制每一位与左边一位异或（最高位不变）得到新结果，例如010（2）转化为格雷码就是011。显然这样做算出一个格雷码的复杂度为O（logn），但是好像没用到递归。后来发现另一种做法，就是上文所提到的对称规律的做法，这里不过多赘述。

对于递归的写法其实很好写，在ACM中很多算法都是递归写出来，在写代码的过程中最大的问题在于找格雷码编码的规律和vector和string的使用（确实是因为一直在用邻接表所以就忘了vector的很多用法）。其余的似乎没什么难度。

在总结里我还想补充一种有一点点局限的方法（或者称为思想）供其他人参考。上文我储存每一位的做法是用string储存（网上的代码不论java还是c++几乎都是用string），保存在vector中，但是这样有时候还是很麻烦。其实在比赛里还有一种常见的做法就是直接用二进制保存，然后通过对1移位再相与操作进行判断每一位是1或是0，类似：

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

**if(ret & (1 << i)){**

**}**

**else{**

**}**

**}**

这种做法叫状态压缩，所以完全可以用这样的方法代替vecto<string>，只用一个数组就能表示。比如数字23的二进制为10111，那么我在递归的时候只要传入23，在读取的时候按位和1相与，就能读出10111，这样在有些情况下传参比较方便。但是我没有用这个方法，就像我刚才说的，这个方法有限制。因为long long只有2^64左右，所以最多只能表示63位，显然当n超过63时，程序会给出错误答案。格雷码这道题的描述很像比赛时候对题目的描述，但是实验要求中对n的范围没有具体规定，如果是OJ的题目可能还要去讨论版问出题人，所以这里就不用状态压缩的方法去做了。