# 作业3.1

## 题目描述 1

求 末尾有多少个 ,

## 算法描述

对于一个 , 考虑所有小于 的正整数的因子, 的数量远超 的数量. 而对于 , 其末尾每个零的来源都是一个因子 , 即 , 所以只需要统计 ~ 中所有数的因子中, 有多少个因子

## 代码

//golang  
var N int  
var cnt int = 0  
  
func main() {  
 fmt.Scanln(&N)  
 for i := 1; i <= N; i++ {  
 n := i  
 for n%5 == 0 {  
 cnt++  
 n /= 5  
 }  
 }  
 fmt.Println("Ans =", cnt)  
}

## 运行结果

1000  
Ans = 249

10000  
Ans = 2499

## 题目描述 2

构造行列的逆转矩阵

## 算法描述

以当前数字、当前位置和行进方向四元组为状态, 编写一状态机, 当时就完成了矩阵构造

## 代码

//golang  
var graph [][]int  
var M, N int  
  
var incr = []int{1, 0, -1, 0}  
var incc = []int{0, 1, 0, -1}  
var inc = []int{1, 1, -1, -1}  
var gap []int  
  
func move(d int, r int, c int, stat int) {  
 //fmt.Println(r, c, d)  
 stat %= 4  
 graph[r][c] = d  
 if d == M\*N {  
 return  
 }  
  
 var oper \*int  
 if incr[stat] != 0 {  
 oper = &r  
 } else {  
 oper = &c  
 }  
 \*oper += inc[stat]  
 if \*oper >= gap[stat] || \*oper < 0 || graph[r][c] != 0 {  
 \*oper -= inc[stat]  
 stat++  
 move(d, r, c, stat)  
 } else {  
 move(d+1, r, c, stat)  
 }  
}  
func main() {  
 //var N int  
 fmt.Scanln(&M, &N)  
 gap = []int{M, N, M, N}  
 for i := 0; i < M; i++ {  
 var t []int = make([]int, N)  
 graph = append(graph, t)  
 }  
 move(1, 0, 0, 0)  
 for i := 0; i < M; i++ {  
 for j := 0; j < N; j++ {  
 fmt.Printf("%3d", graph[i][j])  
 }  
 fmt.Println()  
 }  
}

## 运行结果

4 5  
 1 14 13 12 11  
 2 15 20 19 10  
 3 16 17 18 9  
 4 5 6 7 8

7 7  
 1 24 23 22 21 20 19  
 2 25 40 39 38 37 18  
 3 26 41 48 47 36 17  
 4 27 42 49 46 35 16  
 5 28 43 44 45 34 15  
 6 29 30 31 32 33 14  
 7 8 9 10 11 12 13

6 3  
 1 14 13  
 2 15 12  
 3 16 11  
 4 17 10  
 5 18 9  
 6 7 8

## 题目描述 3

两个有三元集合 与 , 不与 匹配, 不与 和 匹配,求两个集合间的一个一对一映射

## 算法描述

可以抽象为一个二分图, 两个集合间的一一映射关系就是这个二分图的最大匹配, 两个集合的元素抽象为点, 集合间的关系抽象为边, 不允许匹配的边删除后, 使用二分图匹配的匈牙利算法解决,

## 代码

//c++  
int graph[MAXN][MAXN];  
int match[MAXN];  
int vis[MAXN];  
  
const char out[] = {'A', 'B', 'C', 'X', 'Y', 'Z'};  
bool find(int p) {  
 for(int i=0;i<6;++i) {  
 if(!graph[p][i])  
 continue;  
 if(vis[i])  
 continue;  
 vis[i] = 1;  
 if(match[i] == -1 || find(match[i])) {  
 match[p] = i;  
 match[i] = p;  
 return true;  
 } else {  
 return false;  
 }  
 }  
 return false;  
}  
int main() {  
 memset(match, -1, sizeof(match));  
 //cin >> N;  
 for(int i=0;i<3;++i)  
 for(int j=3;j<6;++j)  
 graph[i][j] = graph[j][i] = 1;  
 graph[0][3] = graph[3][0] = 0;  
 graph[2][3] = graph[3][2] = 0;  
 graph[2][5] = graph[5][2] = 0;  
  
 for(int i=0;i<3;i++) {  
 memset(vis, 0, sizeof(vis));  
 find(i);  
 }  
 for(int i=0;i<3;++i)  
 cout << out[i] << ' ' << out[match[i]] << endl;  
 return 0;  
}

## 运行结果

A Z  
B X  
C Y

# 作业3.2

## 题目描述 1

求

## 算法描述

算式中, 每个加数作为数列的一项, 则数列递推式为

时, 设 , 有, 求出 <br>  
即数列 为首项为, 公比为的等比数列, 通项公式为 <br>  
求和, 得到  
编写程序计算公式即可

## 代码

//golang  
func qpow(n float64, k int) float64 {  
 var ans float64 = 1  
 for k > 0 {  
 if k%2 == 1 {  
 ans = ans \* n  
 }  
 n = n \* n  
 k = k >> 1  
 }  
 return ans  
}  
func main() {  
 var N int  
 fmt.Scanln(&N)  
 var ans float64  
  
 ans = (2.0 / 9.0) \* ((qpow(10.0, (N+1))-10)/9.0 - float64(N))  
  
 fmt.Println("ANS =", ans)  
}

## 运行结果

6  
ANS = 246912

20  
ANS = 2.4691358024691356e+19

## 题目描述 2

给定正整数N, 求比N大的最小"不重复数", 不重复数指没有两个相等的相邻位的整数

## 算法描述

为了保证生成的数大于, 在的基础上生成答案.<br>  
设表示整数N + 1从高位往低位数第i位, 先求, 令, 令, 如果发生进位, 则计算进位后, 再找到新的, 保证, 进行相同操作, 重复直到不发生进位.<br>  
操作完成后,设最后一次操作中为,对于所有满足, 不会影响最后的数与的大小关系; 对于所有满足, 不存在相邻位相同.<br>  
此时, 可以直接使用序列填充所有满足, 得到最后的答案

## 代码

//golang  
func main() {  
 var n int  
 fmt.Scanln(&n)  
 n++  
 var N string  
 N = fmt.Sprintf("%d", n)  
  
 var oper []int  
 for \_, x := range N {  
 oper = append(oper, int(x-'0'))  
 }  
 //fmt.Println(oper)  
 var flag int = -1  
 L := len(oper)  
 for i, \_ := range oper {  
 if i == L-1 {  
 break  
 }  
 if oper[i] == oper[i+1] {  
 flag = i + 1  
 break  
 }  
 }  
 fmt.Print("ANS = ")  
 if flag < 0 {  
 fmt.Println(n)  
 return  
 }  
 for flag >= 0 {  
 oper[flag]++  
 if flag == 0 {  
 if oper[flag] > 9 {  
 fmt.Print("1")  
 oper[flag] = 0  
 flag = -1  
 }  
 break  
 }  
 if oper[flag] > 9 {  
 flag--  
 continue  
 }  
 if oper[flag] == oper[flag-1] {  
 flag--  
 continue  
 }  
 break  
 }  
  
 for i := 0; i <= flag; i++ {  
 fmt.Print(oper[i])  
 }  
 flag++  
 for i := 0; i+flag < L; i++ {  
 fmt.Print(i % 2)  
 }  
 fmt.Print("\n")  
}

## 运行结果

1222  
ANS = 1230

9899  
ANS = 10101

1234  
ANS = 1235

## 题目描述 3

给定数组 , 求数组 , 令

## 算法描述

对于 , 直接求出 和 , 分别存储在数组 和数组 中, 再求出需要的数组   
时间复杂度为 , 空间复杂度为

## 代码

//golang  
func main() {  
 var N int  
 fmt.Scanf("%d\n", &N)  
 var A, B []int  
 A = make([]int, N)  
 B = make([]int, N)  
 for i, \_ := range A {  
 fmt.Scanf("%d", &A[i])  
 }  
 //fmt.Println(A)  
 for i, \_ := range A {  
 if i != 0 {  
 B[i] = B[i-1] \* A[i-1]  
 } else {  
 B[i] = 1  
 }  
 }  
 L := len(A)  
 for i, \_ := range A {  
 if i != 0 {  
 A[L-i-1] \*= A[L-i]  
 }  
 }  
  
 for i, \_ := range B {  
 if i != L-1 {  
 B[i] = B[i] \* A[i+1]  
 }  
 }  
  
 fmt.Println(B)  
}

## 运行结果

5  
1 2 3 4 5  
[120 60 40 30 24]

6  
1 2 3 4 5 6  
[720 360 240 180 144 120]