作业3.1

题目描述 1

求 $n!$ 末尾有多少个 0 ,$1000 < n < 10000$

算法描述

对于一个 n, 考虑所有小于 n 的正整数的因子, 2 的数量远超 5 的数量. 而对于 $n!$, 其末尾每个零的来源都是一个因子 10, 即 $2 * 5 = 10$, 所以只需要统计 $1 \sim n$ 中所有数的因子中, 有多少个因子 5

代码

```golang
//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M行N列的逆转矩阵

算法描述

以当前数字、当前位置和行进方向(d, r, c, s)四元组为状态, 编写一状态机, 当$rc = MN$时就完成了矩阵构造

代码

```golang
//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 $\{A,B,C\}$ 与 $\{X,Y,Z\}$, A 不与 X 匹配, C 不与 X 和 Z 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++
//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 $2+22+222+\ldots+222222\ldots$

算法描述

算式中, 每个加数作为数列$\{a_n\}$的一项, 则数列递推式为

$$a_n=\begin{cases}2 & n=1\\ 10a_{n-1}+2 & n>1\end{cases}$$

$n>1$时, 设 $a_n+b=10(a_{n-1}+b)$, 有$a_n=10a_{n-1}+9b$, 求出$b=10/9$

即数列$\{a_n+10/9\}$ 为首项为$2+10/9$, 公比为10的等比数列, 通项公式为 $a_n=2(10^n-1)/9$

求和, 得到$S_n=2((10^{n+1}-10)/9-n)/9$
编写程序计算公式即可

代码

```golang
//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N, 在$N+1$的基础上生成答案.

设a_i表示整数N + 1从高位往低位数第i位, 先求k, 令$a_k = max\{a_i | a_i = a_{i-1}\}$, 令$a_k \leftarrow a_k + 1$, 如果发生进位, 则计算进位后, 再找到新的$k'$, 保证$k' < k$, 进行相同操作, 重复直到不发生进位.

操作完成后,设最后一次操作中为k_λ,对于所有a_i满足$i > k_\lambda$, 不会影响最后的数与N的大小关系; 对于所有a_j满足$j \leq k_\lambda$, 不存在相邻位相同.

此时, 可以直接使用01序列填充所有a_i满足$i > k_\lambda$, 得到最后的答案

代码

```golang
//golang
```

```go
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 a, 求数组 b, 令 $b[i] = (\prod_{i=0}^{n-1} a[i])/a[i]$

算法描述

对于 i, 直接求出 $\prod_{j=0}^{i-1} a[j]$ 和 $\prod_{j=i+1}^{n-1} a[j]$, 分别存储在数组 a 和数组 b 中, 再求出需要的数组 b
时间复杂度为 $O(n)$, 空间复杂度为 $O(1)$

代码

```golang
//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]
```