算法设计与分析 E3-C

21377206 阮阳栋

题目描述

你正在努力地打怪升级。初始时你的等级为 `1` ，当你的等级为 `i` 时，你能够打败所有等级小于等于你的怪物。当你打败等级为 `k` 的怪物时，你的等级能够提升 `k` 。在你的世界中有无限只所有等级的怪物，请你求出最少需要击败多少只怪物，能够将自身的等级提升到 `n` 。

题目分析

因为怪物是无限制的，可以采用贪心的策略去打怪升级：

$$1 \to 2 \to 4 \to ... \to 2^m$$

现在就将问题转化成了寻找最小的整数m_0使得$2^{m_0} \geq n$，即$m_0 = \lceil \log_2 n \rceil$，就是最后的答案。

题目求解

第一反应是$O(1)$的做法：

```
result = log(n)/log(2);
```

但题目中 n 的取值范围为$1 \leq n \leq 10^{1000}$，不能直接当成数字，必须转化为字符串来进行运算。

当成字符串，可以有两种思路来实现对数运算：起始是 `1` ，不断乘 `2` 直到大于等于 n ；起始是 n ，不断除以 `2` 直至为 `1` 。显然字符串做乘法是更为容易的。

C++中的字符串 `string` 相比于C语言的 `char*` 方便了很多，代码可以更加灵活：

比较两个字符串整数的大小时，在长度不等时，长度小的整数值更小；在长度相等时，字典序小的整数值更小（C++中的 `string` 的字典序比较直接用 `<` 、 `>` 、 `==` ）。比较代码如下：

```
int numl = num.length(), sl = s0.length();
//比较s0对应整数是否比num对应整数大
if (sl > numl) ...
else if (sl == numl && s0 >= num) ...
```

在进行乘法时，如果整体发生进位，整数前补 1 。乘 2 代码如下：

```
int sp = 0; //进位标志
for (int i=sl-1;i>=0;i--){
    int u = s0[i]-'0'; //第i位对应数字
    s0[i] = (2*u)%10+sp+'0'; //乘2
    if (u>=5) sp = 1; //发生进位
    else sp = 0; //不进位
}
if (sp) s0 = "1" + s0; //整体进位
```

时间复杂度

数字 n 的位数为 $\lg n$，进行了 $\log_2 n$ 次乘法，一共 T 组数据，时间复杂度 $O(T\cdot \log n\cdot \log n)$。

核心代码

```
int t;
string num, s0;

void solve(){
    s0 = "1";
    int result = 0, numl = num.length();
    while (1){
        int sl = s0.length();
        if (sl > numl) break;
        else if (sl == numl && s0 >= num) break;
        int sp = 0;
        for (int i=sl-1;i>=0;i--){
            int u = s0[i]-'0';
            s0[i] = (2*u)%10+sp+'0';
            if (u>=5) sp = 1;
            else sp = 0;
        }
        if (sp) s0 = "1" + s0;
        result++;
    }
    cout << result << endl;
}
```