一．重复字符串(powerstr):

30 分做法：枚举

长度范围只有 1000 时，我们可以枚举 k， 取字符串第 1 个到第 k 个字符作为子串 T，然后去验证剩下的字符串是否都是 T 重复得来

时间复杂度 O(n^2)

100 分做法：KMP，Next 数组

假设字符串为 S，长度为 N，子串 T 重复 K 次后得到串 S，那么 T 的长度一定为 L = N/K(要整除)，则 T = S[1...L]，将 S 拆分成 K 份，每份长度为 L，则有

S[1...L] = S[L+1...2L] = S[2L+1...3L] = ... = S[(K-1)L+1...KL]

由于要保证 K 最大，势必 L 要取最小，所以根据 Next 函数的定义，有 Next[KL] = (K-1)L;

即 Next[N] = N - L，所以 L = N - Next[N];

但是得出的长度 L 还要保证能被 N 整除，所以如果不能整除说明 L = N，即 K = 1；而如果能整除，那么 K = N / (N - Next[N]);

因而我们只要对字符串 S 做一趟 KMP，对其求 Next 数组，剩下的就是上述结论

时间复杂度 O(n)

二．Fibonacci 进制(fib):

100 分做法：DP

N 很大，先尝试几个小数据。可以发现，每个 Fibonacci 表示的长度，和 Fibonacci 数大小有关（1,2,3,5,8,13,21……），这些值最高位上是 1，后面全是 0，即第一个 Fibonacci 表示长度为 i 的数是 fib[i]。因此按照长度对 Fibonacci 表示进行分类，长度为 i 的数有 fib[i-1]个，看做是第 i 组。那么第 i 组的总长度 len[i] = fib[i-1]*i。

接下来，看 1 出现的次数。长度为 i 的数的表示中，第 i-1 位肯定是 0。

Sum[i] 表示前 i 组的 1 的个数。可以得到如下式子：Sum[i]=sum[i-1]+fib[i-1]+sum[i-2]。第 i 组首位 1 的个数为 fib[i-1]，i-1 位为 0，那么最后的 i-2 位的情况，恰好为 1~i-2 组的所有情况。

整体算法也就明了了：

1） 求出 N 位所在的 Fibonacci 表示的数的长度 t
2） 求 1~t 中 Fibonacci 表示中 1 出现的个数。
3） 继续求解剩余字符的 1。

例如求解得到最后对应 Fibonacci 表示为 x=100100

1） 对于长度为 1~5 的 Fibonacci 表示中 1 的个数为 sum[5]，i<=100000 中 1 的个数即为 sum[5]+1。
2） 对于 100000<i<=100100,最高位都是 1，共有 fib[3]个，后三位 1 的个数为 sum[2]+1。
3） 1 的总个数为 sum[5]+1+fib[3]+sum[2]+1。

最后细节比较多，要实现的仔细一些。

三．发奖金(reword):

100 分做法：组合+质因数分解+逆元+中国剩余定理
题目相当于求 n 个数的和不超过 m 的方案数。
首先如果是恰好等于 m，那么就等价于求方程 x1 + x2 + ... + xn = m 的解的个数，利用插板法可得到公式：C(n + m - 1, m)
现在要求不大于 m 的，相当于对 i = 0 ... m 对 C(n + i - 1, i)求和，根据 pascal 递推式可以得到答案为 C(n + m, m)
现在就需要求 C(n + m, m) mod P

这里我们主要要解决如何快速计算 n! mod P
以及当分母有 m! mod P 的情况

1. 当 n,m 都比较小的时候，同时 P 为比较大的素数时，可以直接利用逆元求解，当 n,m 比较大的时候，见下面两种情况（以下参考魏铭 2011 年国家集训队作业）

2. 当 P 为素数的情况：
   我们发现 n! mod P 的计算过程是以 P 为周期的，举例如下：
   n = 10, P = 3
   n! = 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10
   = 1 * 2 *
   4 * 5 *
   7 * 8 *
   10 *
   3 * 6 * 9
   = $(1 * 2)^3$ *
   $3^{3\,*}$ (1 * 2 * 3)
   最后一步中的 1 * 2 *3 可递归处理。
   因为 P 的倍数与 P 不互质，所以 P 的倍数不能直接乘入答案，应当用一个计数器变量 cnt 来保存答案中因子 P 的个数。
   我们提前预处理出 fac[i] = 1 * 2 * 3 * … * (i – 1) * i mod P，函数 calcfac(n)返回 n! mod P 的值，power(a, b, c)返回 a^b mod c 的值，可用快速幂在 O(logb)的时间内完成。

```
typedef long long LL;
LL calcfac(LL n)
{
    if (n <P) return fac[n];
    LL seg = n / P, rem = n % P;
    LL ret = power(fac[P - 1], seg, P); //fac[P - 1]重复出现了 seg 次
    ret = ret * fac[rem] % P; //除去 seg 次 fac[P – 1]外，剩下的零头
    cnt += n / P; //提出 n / P 个因子 P
    ret = ret * calcfac(n / P) % P; //递归处理
    return ret;
}
```

   于是 n! mod p 的计算可在 O(logn)的时间内解决。

对于分母中的 n!，方法是相似的。若 a 为正整数，a * a’ = 1(mod P)，那么我们称 a’为 a 的逆元，记作 a^{-1}，并有 b / a(mod P) = b * a^{-1}(mod P)。这样我们只需要把预处理 fac[i] = 1 * 2 * 3 * … * (i – 1) * i mod P 更换为 inv[i] = 1^{-1} * 2^{-1} * 3^{-1}* … * $(i – 1)^{-1}$ * i^{-1} mod P，其计算方法与分子中的 n!计算相差无几，具体可参考我的代码。求逆元可以使用扩展欧几里得算法。

3. 当 p 为合数时

对于某些测试点，我们发现 P 分解后只有 2 个因子，并且不相同，所以我们可以对这两个因子分别运行算法 2，最后用中国剩余定理合并即可。

对于剩下的数据，可以对 P 进行质因数分解，得到 $P = p_1^{c1} * p_2^{c2} * … * p_t^{ct}$。

对于每个 1≤i≤t，以 p_i^{ci} 为模运行算法 2，最后用中国剩余定理合并。这里 p_i^{ci} 不一定为质数，不过只需对原算法稍加修改即可。令 $P = p_i^{ci}$，fac[i] = 除去 p_i 的倍数外 i 的阶乘。例如 pi = 3,ci = 2，那么 fac[10] = 1 * 2 * 4 * 5 * 7 * 8 * 10，除去了 3 的倍数 3、6 和 9。阶乘依然是以 P 为周期的，calcfac(n)与算法 2 主体相同，只是统计因子个数时，应使用 ret += n / p_i 而不是 ret += n / P，递归处理时也应该是 calcfac(n / p_i)而不是 calcfac(n / P)。

时间复杂度 O(t * n)