Задача B1. Две и три

Разполагаме с неограничен брой десетични цифри 2 и 3. С част от тях искаме да напишем десетично число, което да се дели на 2^n, където n е естествено число. При това търсим най-малкото такова число.

Ако например $n = 4$, една цифра не стига да напишем такова число: 2 е четно, но не се дели на $2^4 = 16$, 3 пък направо е нечетно. С две цифри от наличните могат да се напишат (по големина) 22, 23, 32 и 33. Числото 32 вече има нужното ни свойство (впрочем, то се дели даже и на $2^5 = 32$). Факт е, че например 33232 също изпълнява условието, но то е много по-голямо. Няма изискване и двете цифри да се срещат задължително в записа, нито пък да са с равен брой срещания.

Напишете програма **N23**, която намира желаното число или установява, че такова не съществува.

От стандартния вход се въвежда един ред, съдържащ естественото число n, $1 \le n \le 10000$.

На стандартния изход програмата трябва да изведе един ред с намереното най-малко естествено число, в десетичния запис на което има само цифрите 2 и 3 (по колкото е необходимо от всяка от тях) и което се дели на 2^n; или един ред със съобщението NO, ако такова число не съществува.

Пример. Вход:
```
11
```

Изход:
```
223232
```

Решение

Още от основния училищен курс за десетично записаното естествено число m е известно, че:

- дели се на 2, ако е четно (т.е. – последната му (*една*) цифра се дели на 2);
- дели се на 4, ако числото, образувано от последните му *две* цифри се дели на 4 (ако е едноцифрено, можем да считаме, че има водеща нула);
- дели се на 8, ако числото, образувано от последните му *три* цифри се дели на 8 (отново можем да дописваме водещи нули, ако няма достатъчно цифри).

Обобщението на тези правила се очаква и лесно се доказва: $\boldsymbol{m}$ се дели на 2^n тогава и само тогава, когато числото, образувано от последните му $\boldsymbol{n}$ цифри в същия ред се дели на 2^n (ако цифрите на $\boldsymbol{m}$ не стигат, можем да си мислим, че има водещи нули).

Като вземем предвид и елементарното съображение, че щом $\boldsymbol{m}$ се дели на 2^n, то се дели и на всички по-малки степени на двойката, заключаваме, че с ограничения ресурс от десетични цифри трябва да изграждаме „опашки" от $\boldsymbol{n}$ цифри, гарантиращи делимостта на 2^n. В нашия случай се оказва, впрочем, че такава „опашка" е еднозначно определена. Наистина – последната цифра задължително е 2, иначе пропада делимостта на 2; предпоследната – задължително 3, което осигурява делимост на 4; следващата по старшинство – задължително 2, иначе няма делимост на 8 и т. н.

Алгоритъмът за изграждане на тази редица се оказва елементарен, което еднозначно ни осигурява число, записано с общо n на брой цифри (двойки и тройки), което се дели на 2^n, т.е. задачата винаги има решение с не повече от n цифри. По-интересно е изискването да се намери най-малкото решение. Наистина, 23232 (което е опашката от пет цифри) се дели на 2^5=32, но още 32=00032 има това свойство (и се записва само с две цифри).

Алгоритъм за получаване на опашката. Да разгледаме сега по-подробно аритметиката на получаване на редицата $\{\boldsymbol{c_i}\}$ от цифри в „опашката", започвайки от най-младшата – $\boldsymbol{c_1}$. Единствената възможност, както споменахме, е $\boldsymbol{c_1}$=2. Всяка от следващите (по старшинство) цифри от „опашката" m трябва да осигурява делимост на още една степен на двойката. На i-тата стъпка ($i > 1$) ще имаме $\boldsymbol{c_i}$=2 или $\boldsymbol{c_i}$=3, т. е., ако означим с

$$m_i = \overline{c_i \underbrace{c_{i-1} c_{i-2} \dots c_1}_{m_{i-1}}}$$

естественото число, записано с тези цифри в този ред, имаме:

$$m_i = \begin{cases} 2.10^{i-1} + m_{i-1} \\ \textit{или} \\ 3.10^{i-1} + m_{i-1} \end{cases}$$

като изборът е еднозначно определен от това, дали „досегашната опашка" m_{i-1} вече се дели и на 2^i (на 2^{i-1} тя задължително се дели) или не. Наистина, ако $m_{i-1} = 2^i.k$, където $\boldsymbol{k}$ е естествено число, единствената възможност да получим още един делител на 2 е да вземем първата алтернатива:

$$m_i = 3.10^{i-1} + m_{i-1} = 3.2^{i-1}.5^{i-1} + 2^{i-1}.t = 2^{i-1}(3.5^{i-1} + t)$$

(не втората, защото тогава първото събираемо не се дели на 2^i, второто се дели → сумата не се дели). И точно напротив – при $m_{i-1} \neq 2^i.k$ ще имаме $m_{i-1} = 2^{i-1}.t$, където $\boldsymbol{t}$ е нечетно. Единствена възможност сега остава

$$m_i = 2.10^{i-1} + m_{i-1} = 2.2^{i-1}.5^{i-1} + 2^i.k = 2^i(5^{i-1} + k)$$

като числото в скобите е четно (сума от две нечетни) и, следователно, произведението се дели на 2^i. По аналогични на предишния случай причини другата алтернатива не може да бъде избрана. Това доказва еднозначността и показва алгоритъма за получаване на редицата от цифри $\{\boldsymbol{c_i}\}$.

Алгоритъм за решаване на задачата. Въз основа на горните разглеждания, естественият подход към атакуване на проблема е да изграждаме опашката по указания алгоритъм и да спрем в първия момент, когато получим делимост на 2^n. Малко запомняне на предишни резултати ще ускори изчислителния процес.

1. Въвежда се естественото число n.
2. Инициализираме променливи: резултат: res = 2; степен на петицата: deg5 = 1; изчислена база за пресмятане на новата стойност k = 1; осигурена степен на двойката, на която res се дели: deg = 1.
3. Проверяваме дали deg е по-малко от n. Ако да – към т. 4 иначе към т. 11.
4. Ако k е четно, долепваме към res водеща цифра 2, иначе – водеща цифра 3.
5. Минимално осигурената степен на делимост сега е равна на броя на цифрите на res, затова deg приема тази стойност.
6. Проверяваме дали deg все още е по-малко от n. Ако да – към т. 7 иначе към т. 11
7. Намираме следващата степен на петицата: deg5 := 5*deg5
8. k :=((новодобавената цифра)*deg5+k) / 2
9. Проверяваме каква степен на двойката дели k, като увеличаваме deg с толкова. Ако достигнем (или надхвърлим) **n**, към т. 11.
10. Към т. 4.
11. Извеждаме резултата res.

Примерна реализация

```
#include <stdio.h>
int n;
typedef struct {int count;
                char dig[10001];
               } Long;

Long res={1,{2}},deg5={1,{1}},k={1,{1}};

void Long_Add(Long *a, Long *b, Long *r)
{char carry=0;
 int i;
 r->count=(a->count>b->count)?a->count:b->count;
 for (i=0;i<r->count;i++)
 {if (i<a->count&&i<b->count) r->dig[i]=a->dig[i]+b->dig[i];
  else r->dig[i]=(i<a->count)?a->dig[i]:b->dig[i];
  r->dig[i]+=carry;
  if (r->dig[i]>9) {r->dig[i]-=10;carry=1;}
  else carry=0;
 }
 if (carry) r->dig[r->count++]=1;
}
```

```
void Long_MulDig(Long *a,char d,Long *r)
{char carry=0;
 int i,t;
 r->count=a->count;
 for(i=0;i<a->count;i++)
 {t=d*a->dig[i]+carry;
  r->dig[i]=t%10;
  carry=t/10;
 }
 if (carry) r->dig[r->count++]=carry;
}

int Long_Div2(Long *a,Long *r)
{int i,d=0;
 r->count=a->count;
 for (i=a->count-1;i>=0;i--)
 {d=10*d+a->dig[i];
  r->dig[i]=d>>1;
  d&=1;
 }
 if (!r->dig[r->count-1]) r->count--;
 return d;
}

void Long_Show(Long *a)
{int i;
 for (i=a->count-1;i>=0;i--) printf("%d",a->dig[i]);
 printf("\n");
}

int main(void)
{int deg=1;
 Long t;
 scanf("%d",&n);
 while (deg<n)
 {res.dig[res.count++]=2+(k.dig[0]&1);
  deg=res.count;
  if (deg==n) break;
  Long_MulDig(&deg5,5,&deg5);
  Long_MulDig(&deg5,res.dig[res.count-1],&t);
  Long_Add(&t,&k,&t);
  Long_Div2(&t,&k);
  t=k;
  while (deg<n&&!Long_Div2(&t,&t)) deg++;
 }
 Long_Show(&res);
 return 0;
}
```