Задача С1. ПОДДУМИ

Разглеждаме думи, образувани от главни латински букви.

Последователност от една или повече съседни букви в думата наричаме „поддума”.

Например, някои от думите, които са поддуми на думата ТАРАТОР са: Т, А, АРАТ, ТОР, ТАРАТОР.

Напишете програма **SUBWORDS**, която намира броя на ***различните*** поддуми, които съдържа дадена дума.

От стандартния вход се въвежда един ред, който съдържа една дума с дължина, не по-голяма от 100. На стандартния изход да се изведе един ред с едно естествено число – броя на различните поддуми във въведената дума.

ПРИМЕР

Вход	**Изход**
CTPACT	18

Обяснение: В думата СТРАСТ има 18 различни поддуми: С, Т, Р, А, СТ, ТР, РА, АС, СТР, ТРА, РАС, АСТ, СТРА, ТРАС, РАСТ, СТРАС, ТРАСТ и СТРАСТ.

Решение:

Да се намерят всички поддуми, като се запазят в масив (или по-добра структура) неповтарящите се от тях е едно стандартно решение, чиято коректна реализация решава проблема при тези ограничения. Тук предлагаме едно съвсем икономично на памет решение. При него от максималния възможен брой поддуми се изважда броя на тези, които се повтарят.

Решение (C):

```
#include <stdio.h>
#include <string.h>
char w[128];
// Функция, която определя дали масив от символи m с дължина l
// се среща в С-низа s.
int memstr(const char *m,int l,const char *s)
{char b[128];
 memcpy(b,m,l);
 b[l]=0;
 return strstr(s,b)?1:0;
}
//Функция, която преброява неповтарящите се поддуми в С-низа s.
int Count(const char *s)
{int c,i,j,l=strlen(s);
 c=l*(l+1)/2; //Максимален брой поддуми: една с дължина l,
// 2 с дължина l-1,…, l с дължина 1, т. е. 1+2+3+…+l=l*(l+1)/2.
 for (i=1;i<l;i++)
  for (j=0;j<=l-i;j++)
   if (memstr(&s[j],i,&s[j+1])) c--;//Ако се среща и по-надясно,
                                    // намаляваме броя.
 return c;
}
int main (void)
{fgets(w,102,stdin);
 w[strlen(w)-1]=0;
 printf("%d\n",Count(w));
 return 0;
}
```

Решение (Pascal):

```
VAR
 w:String;
Function memstr(const s,w:string):Boolean;
Begin
 memstr:=pos(s,w)>0;
End;
Function Count(const s:String):Integer;
Var c,i,j,l:Integer;
Begin
 l:=length(s);
 c:=l*(l+1) div 2;
 For i:=1 To l-1 Do
  For j:=1 To l-i Do
   If memstr(copy(s,j,i),copy(s,j+1,255)) Then Dec(c);
 Count:=c;
End;
BEGIN
 ReadLn(w);
 WriteLn(Count(w));
END.
```

Задача С2. ДЪЛБОЧИНА

Разглеждаме следния алгоритъм за преобразуване на естествено число ***n***.

Стъпка 1. Ако числото е 1 – край на алгоритъма.
Стъпка 2. Ако числото е четно – делим го на 2 и се връщаме към Стъпка 1.
Стъпка 3. Умножаваме числото по 3, прибавяме 1 и преминаваме към Стъпка 1.

Да приложим алгоритъма за ***n***=7.

Стъпка 1: 7 не е равно на 1, преминаваме към Стъпка 2
Стъпка 2: 7 не е четно, преминаваме към Стъпка 3
Стъпка 3: 3*7+1=22, преминаваме към Стъпка 1
Стъпка 1: 22 не е равно на 1, преминаваме към Стъпка 2
Стъпка 2: 22 е четно, 22/2=11, преминаваме към Стъпка 1
Стъпка 1: 11 не е равно на едно ... и т. н.

Така числото 7 преминава през следните „състояния”, докато се преобразува в 1:

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Под „**дълбочина**” на естественото число ***n*** ще разбираме броя на числата в тази редица.

Така дълбочината на 7 е 17. Лесно се пресмята, че дълбочината на 1 е 1, на 2 е 2, на 3 е 8 и т. н.

Напишете програма **DEPTH**, която намира какъв е максималният брой числа с равна дълбочина в зададен интервал от естествени числа, като въвежда от стандартния вход две естествени числа ***a*** и ***b*** ($1 \le a \le b \le 10000$) и извежда на стандартния изход един ред с едно естествено число – максималния брой естествени числа в интервала [***a***, ***b***] с една и съща дълбочина.