Задача А2. НИЗОВЕ

Низът $\alpha = a_1 a_2 ... a_P$ е *начало* на низа $\beta = b_1 b_2 ... b_Q$, ако $0 < P < Q$ и $a_K = b_K$, $K = 1,2,\ldots,P$. На първия ред на стандартния вход е зададено цялото положително число N, $N \le 1000$. На всеки от следващите N реда е зададен по един низ, съставен от малки латински букви, с дължина L, $1 \le L \le 20$. Напишете програма **ORD**, която намира и извежда на стандартния изход дължината M на най-дългата редица $\alpha_1, \alpha_2, \ldots, \alpha_M$, съставена от някои от зададените низове такава, че α_K е начало на α_{K+1}, $K = 1,2,\ldots,M-1$.

ПРИМЕР

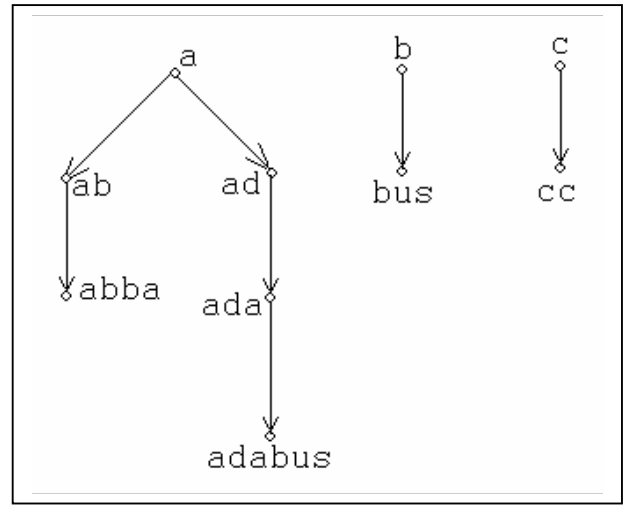
Вход	**Изход**
10	4
a	
b	
ad	
ab	
ada	
c	
adabus	
bus	
abba	
cc	

Решение:

Очевидно е, че релацията между низове „*е начало на*", дефинирана в задачата, е преднаредба или, което е едно и също, ориентиран ацикличен граф (*даг*) $G(V,E)$. Поради по-голямата популярност на второто понятие, ще си служим с него в това обяснение. Задачата, поставена в условието, сега може да се формулира така: по зададен даг да се намери броят на върховете на един негов най-дълъг път. Популярен подход за решаване на тази задача е да се използва динамично програмиране. За всеки връх v на дага означаваме с P_v дължината на максималния път, завършващ във v. За всеки връх v, който няма предшественици в дага, $P_v = 1$. Ако преките предшественици на v в дага са $u_1, u_2, \ldots, u_K$, тогава

$$P_v = \max \{ P_{u1}, P_{u2}, \ldots, P_{uK} \} + 1$$

За да се извърши пресмятането по-лесно, добре е предварително да се сортират топологически върховете на дага и пресмятанията да се извършват в реда на топологическото сортиране. Така, при пресмятане на стойността P_v за един връх v, стойностите на всички негови предшественици ще са вече пресметнати. Неприятното в това решение е необходимостта да се построи дага, което ще отнеме $O(N^2)$ стъпки. Топологическо сортиране, реализирано с обхождане на дага в дълбочина, ще добави още $O(M)$ стъпки, където M е броят на ребрата на дага. Самото изчисляване на стойностите и намиране на максималната също ще изисква $O(M)$ стъпки. Така сложността на алгоритъма ще бъде $O(N^2) + 2.O(M) = O(N^2)$, защото броят на ребрата M е $O(N^2)$.



Да опитаме нещо друго. В сила е следното

Твърдение. *Лексикографското сортиране на множество от низове е топологическо сортиране на върховете на съответния даг.*

доказателството на което оставяме на читателя за упражнение. Нещо повече, ако представим в даг само преките предшествания, ще получим гора от ориентирани дървета. За примера от условието, съответната гора е показана на фигурата.

Да сортираме лексикографски низовете (в приложената програма не е фиксирано как ще сортираме). Заблежете, че отделните дървета на гората заемат плътни участъци от лексикографското сортиране. Остава да направим обхождането в дълбочина на сортираните топологически низове (вж. Твърдението), да пресметнем необходимите ни стойности P_v, като попътно намерим и максималната от тези стойности. Но това вече става със сложност $O(N)$, защото всеки връх минава точно един път през стека (определящ за това е фактът, че графът на непосредствените прешественици е гора от дървета, а не произволен даг). Така сложността на алгоритъма е равна на сложноста на лексикографското сортиране. При използване на сортиране със сложност $O(N.\log N)$ ще получим много по-бърз алгоритъм. При алгоритъм за сортиране със сложност $O(N^2)$ порядъците на двата алгоритъма са еднакви, и все пак вторият ще бъде по-бърз. За нуждите на състезанието, „бързият" библиотечен `quick_sort` е напълно достататъчен.

```
#include <stdio.h>
#include <string.h>
char a[1001][21],t[21];
int s[1001],sp;
int prefix(int x,int y)
{
    int i=0;
    while(a[x][i]==a[y][i])i++;
    if(a[x][i]=='\0') return 1;
    else return 0;
}
main()
{
   int N,i,j,max;
   scanf("%d",&N);
   for(i=1;i<=N;i++) scanf("%s",a[i]);
   < тук е мястото на сортирането >
   s[1]=1;sp=1;max=1;
   for(i=2;i<=N;i++)
   {
      if(prefix(i-1,i))
      {s[i]=s[i-1]+1;if(max<s[i])max=s[i];}
      else
      {  j=i-2;
         while(j>=sp&&!prefix(j,i))j--;
         if(j<sp){s[i]=1;sp=i;}
         else {s[i]=s[j]+1;if(max<s[i])max=s[i];}
      }
   }
   printf("%d\n",max);
}
```