prof. *Marinel Şerban*
Liceul de Informatică "Gr. C. Moisil", Iaşi

Potrivirea şirurilor
(căutarea apariției unui subşir(*model-pattern*) într-un şir(*text*))

A. Introducere (punerea problemei)

Vom numi subşirul a cărui apariție se caută ***model*** iar şirul în care se caută ***text***.
Fie

`T[1..n]` – textul de lungime `n`

`P[1..m]` – modelul de lungime `m`, `m≤n`

Textul şi modelul sunt, de obicei şiruri de caractere.

Definiție. Spunem că `P` apare cu deplasamentul `s` în `T` dacă:

1. `0 ≤ s ≤ n-m`
2. `T[s+j]=P[j], ∀j, 1 ≤ j ≤ m`

Problemă. Să se determine toate deplasamentele `s` cu care `P` apare în `T`.

Exemplu.
Fie

`T ≡ abcabaabcabac`

`P ≡ abaa`

Pentru `s=3` se obține

```
abcabaabcabac
  abaa
```

B. Algoritmul elementer (naiv)

Presupune parcurgerea tuturor deplasărilor posibile şi verificarea potrivirii modelului în text.

Algoritm:

```
n ← lungime(T)
m ← lungime(P)
pentru s← 0, n-m execută
     dacă P[1..m] = T[s+1..s+m] atunci
          scrie "am găsit subşirul în poziția ", s+1
```

Timp de execuție:

`O((n-m+1)*m)` deoarece

- există `n-m+1` valori posibile pentru `s`
- pentru fiecare s testarea `P[1..m] = T[s+1..s+m]` implică un ciclu cu `m` paşi

Program simplu:

```
Program Elementar;

Var T, P: String;
    n, m, s, j: Byte;
    f, g: Text;
    OK: Boolean;
```

```
Begin
  Assign(f, 'NAIV.IN'); Reset(f);
  Assign(g, 'NAIV.OUT'); ReWrite(g);
  ReadLn(f, T);
  ReadLn(f, P);
  n := Length(T);
  m := Length(P);
  For s := 0 To n-m Do
    Begin
      OK := TRUE;
      j := 1;
      While j<=m Do
        Begin
          If P[j]<>T[s+j] Then OK := FALSE;
          Inc(j)
        End;
      If OK Then WriteLn(g, 'Am gasit subsirul in pozitia ', s+1)
    End;
  Close(g)
End.
```

Algoritmul este ineficient deoarece nu foloseşte deloc informaţiile obţinute prin prelucrare pentru o valoare oarecare a lui s în prelucrările ulterioare.

Exemplu:
Fie

T ≡ aaabaab...

P ≡ aaab

Se observă că pentru s=0 se obţine o căutare cu rezultat pozitiv, dar, în acelaşi timp, din studierea lui P se observă că nici unul din deplasamentele s=1, 2, 3 NU va furniza un rezultat pozitiv deoarece P[4]= 'b', deci nu are rost să fie testate aceste valori pentru deplasament.

C. Algoritmul Knuth-Morris-Pratt (KMP)

Ideea algoritmului este aceea de a defini o funcţie (numită "funcţie prefix") care să reţină informaţii depre ***potrivirea modelului cu deplasamentele lui***.
Funcţia este definită printr-un tablou cu m elemente, tablou care reţine în elementul de indice q o valoare care indică ce deplasament se va testa în continuare ştiind că primele q elemente din model "se potrivesc" cu q elemente din text pentru deplasamentul curent.

Formalizând:
Ştiind că

T[s+1..s+q] = P[1..q], deci q elemente se "potrivesc"

care este cel mai mic deplasament s'>s astfel încât

T[s'+1..s'+k] = P[1..k], cu s+q = s'+k

În cel mai bun caz vom avea s'=s+q, ceea ce indică faptul că deplasamentele

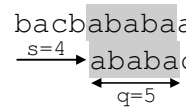
s+1, s+2, ..., s+q-1

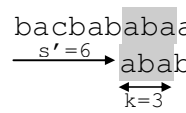
nu trebuie testate.

Tabloul care defineşte funcţia se obţine prin compararea modelului P cu el însuşi, astfel:

- deoarece T[s'+1..s'+k] face parte din porţiunea de text cunoscută, el este sufix al şirului P[1..q]
- urmărim exemplul:

T ≡ bacbababaabcbab
P ≡ ababaca





P[1..5] ababa
P[1..3] aba

P[1..3] este sufix a lui P[1..5]

adică cel mai lung prefix a lui P care este şi sufix a lui P[1..5] este P[1..3], deci în tabloul care reprezintă funcția vom avea

Urm[5]=3

cu interpretarea:

"*dacă la deplasamentul* s *s-au potrivit cu succes* 5 *caractere, următorul deplasament posibil corect este*

s' = s + (5 - Urm[5]) = s + (5 - 3) = s + 2"

- Deci:

Urm : {1, 2, ..., m} → {0, 1, ..., m-1}
Urm[q] = max{k, k<q şi P[1..k], care este prefix a lui P, este şi sufix a lui P[1..q]} = "*lungimea celui mai lung prefix a lui* P *care este sufix a lui* P[1..q]"

Algoritm:

```
n ← lungime(T)
m ← lungime(P)
Urmatorul(P)
q ← 0
pentru i ← 1, n execută
      cât_timp (q>0) şi (P[q+1]<>T[i]) execută
            q ← Urm[q]
      dacă P[q+1] = T[i] atunci
            q ← q + 1     //am mai gasit un caracter corect
      dacă q=m atunci
            scrie "am găsit subşirul în poziția ", i-m+1
            q ← Urm[q]

procedura Următorul(P)
m ← lungime(P)
Urm[1] ← 0
k ← 0
pentru q ← 2, m execută
      cât_timp (k>0) şi (P[k+1]<>P[q]) execută
            k ← Urm[k]
      dacă P[k+1] = P[q] atunci
            k ← k + 1
      Urm[q] ← k
```

prof. *Marinel Şerban*
Liceul de Informatică "Gr. C. Moisil", Iaşi

Exemplu:
Cosiderăm modelul

P ≡ abababababca

Să construim pas cu pas vectorul Urm

q	P	P[1..q]	k	Obs
1	ababababca	a	0	(!k<q)
2	ababababca	ab	0	
3	ababababca	aba	1	
4	ababababca	abab	2	
5	ababababca	ababa	3	
6	ababababca	abagab	4	
7	ababababca	abababa	5	
8	ababababca	abababab	6	(!k<q)
9	ababababca	abababab c	0	
10	ababababca	abababab ca	1	

Timp de execuţie:

O(n+m) deoarece

- calculul tabloului Urm (funcţia prefix) consumă O(m)
- determinarea "potrivirilor" consumă O(n)

Program simplu:

```
Program KMP;

Const nMax = 255;
      mMax = 255;

Var Urm: Array[1..mMax] Of Byte;
    T, P: String;
    n, m, q, i: Byte;
    f, g: Text;

Procedure Urmatorul(P: String);
Var k, m, q: Byte;
Begin
  m := Length(P);
  Urm[1] := 0;
  k := 0;
  For q := 2 To m Do
     {determina lungimea celui mai lung prefix al lui P
      care este si sufix al subsirului P[1..q] }
    Begin
      While (k>0) And (P[k+1]<>P[q]) Do k := Urm[k];
      If P[k+1]=P[q] Then Inc(k);
      Urm[q] := k
    End
End;

Begin
  Assign(f, 'KMP.IN'); Reset(f);
  Assign(g, 'KMP.OUT'); ReWrite(g);
  ReadLn(f, T);
  ReadLn(f, P);
  n := Length(T);
  m := Length(P);
```

```
  Urmatorul(P);
  q := 0;
  For i := 1 To n Do
    Begin
      While (q>0) And (P[q+1]<>T[i]) Do q := Urm[q];
      If P[q+1]=T[i] Then Inc(q);   {s-a mai gasit un caracter corect}
      If q=m Then
        Begin
          WriteLn(g, 'Am gasit subsirul in pozitia ', i-m+1);
          q := Urm[q]
        End
    End;
  Close(g)
End.
```

Două exemple de rulare pentru acelaşi text `T`, fişierul `KMPFIS.PAS`, care are `68` linii şi dimensiune `1453` bytes, şi două modele P (`' Do'` şi `'Do'`):

Model **`' Do'`**

```
Am gasit subsirul in linia 18 pozitia 18
Am gasit subsirul in linia 21 pozitia 37
Am gasit subsirul in linia 40 pozitia 19
Am gasit subsirul in linia 46 pozitia 20
Am gasit subsirul in linia 48 pozitia 39
Am facut 3453 comparatii
```

algoritm ***naiv***

```
Am gasit subsirul in linia 18 pozitia 18
Am gasit subsirul in linia 21 pozitia 37
Am gasit subsirul in linia 40 pozitia 19
Am gasit subsirul in linia 46 pozitia 20
Am gasit subsirul in linia 48 pozitia 39
Am facut 1708 comparatii
```

algoritm ***KMP***

Vectorul `Urm` este în acest caz:

`Urm: (0,0,0,0,0,0,0,...)`

Model **`'Do'`**

```
Am gasit subsirul in linia 18 pozitia 19
Am gasit subsirul in linia 21 pozitia 38
Am gasit subsirul in linia 40 pozitia 20
Am gasit subsirul in linia 46 pozitia 21
Am gasit subsirul in linia 48 pozitia 40
Am facut 2424 comparatii
```

```
Am gasit subsirul in linia 18 pozitia 19
Am gasit subsirul in linia 21 pozitia 38
Am gasit subsirul in linia 40 pozitia 20
Am gasit subsirul in linia 46 pozitia 21
Am gasit subsirul in linia 48 pozitia 40
Am facut 1274 comparatii
```

Vectorul `Urm` este în acest caz:

`Urm: (0,0,0,0,0,0,0,...)`

D. Temă (programe în care se poate utiliza acest algoritm)

prof. *Marinel Şerban*
Liceul de Informatică "Gr. C. Moisil", Iaşi

Problema **desert**, propusă la barajul pentru lotul național, 2001, Bacău, autor: **Ştefan Radu**, student Universitatea "Transilvania" Braşov

Deşert

Fişiere sursă**: `desert.pas`** sau **`desert.c`** sau **`desert.cpp`**

Fie o imagine alb-negru care reprezintă fotografia, realizată din satelit, a unei zone dintr-un deşert unde se caută o construcție secretă de formă dreptunghiulară. Imaginea porțiunii de deşert analizate a fost codificată într-o matrice având dimensiunile maxime de $N \times 255$ elemente. Imaginea construcției este codificată într-o matrice având $K \times 32$ elemente. Elementele celor două matrice pot fi numai caracterele `'.'` şi `'#'`.

Cerință

Determinați de câte ori se regăseşte fotografia construcției pe hartă.

Date de intrare

Fişier de intrare: **`DESERT.IN`**

Linia 1: `N K`

- două numere naturale nenule, *N* reprezentând numărul de linii al matricei care codifică fotografia *deşertului*, iar *K* numărul de linii al matricei pătratice care codifică fotografia *construcției*;

Următoarele *K* linii conțin în formă codificată matricea fotografiei construcției: (caractere `'#'` şi `'.'` **neseparate** prin spații)

C_{11} C_{12} ... $C_{1,32}$
C_{21} C_{22} ... $C_{2,32}$
. . . . .
C_{K1} C_{K2} ... $C_{K,32}$

Următoarele *N* linii conțin în formă codificată matricea fotografiei deşertului (caractere `'#'` şi `'.'` **neseparate** prin spații):

$D_{1,1}$ $D_{1,2}$ ... $D_{1,255}$
$D_{2,1}$ $D_{2,2}$ ... $D_{2,255}$
. . .
$D_{N,1}$ $D_{N,2}$ ... $D_{N,255}$

Date de ieşire

Fişier de ieşire: **`DESERT.OUT`**

Linia 1: `NR`

- număr natural, reprezentând numărul de potriviri al matricei `C` în matricea `D`.

Restricții şi precizări

- $3 \le N \le 1024$
- $1 < K < N$

Exemplu

`DESERT.IN`

```
3 2
#.....(în total 31 de puncte)...
#.....(în total 31 de puncte)...
#.....(în total 254 de puncte).....................
#.....(în total 254 de puncte).....................
#.....(în total 254 de puncte).....................
```

`DESERT.OUT`

```
2
```

Timp maxim de executare/test: 1 secundă

Programul sursă:

```
{ autor: Ştefan Radu, student Universitatea "Transilvania" Braşov }
{$S+,T-,E-,F-,A+,N+,R-,D+,V+,B-,G+,I+,L+,O-,P-,Q-,X+,Y+}
{$M 65000,0,655360}
var f: text;
```

prof. *Marinel Şerban*
Liceul de Informatică "Gr. C. Moisil", Iaşi

```
    q, x, y, z, n, nc: integer;
    v: array[0..1024] of longint;
    dyn: array[0..1024] of integer;
    poz: array[0..223] of integer;
    count, l1: longint;
    s: string;

function s2l(s: string): longint;
var l: longint;
    i: integer;
begin                           {codifica constructia pe biti}
   l := 0;
   for i := 1 to 32 do          {1 rand din constructie = 32 biti = 1 longint}
     begin
       l := l shl 1;
       if s[i]='#' then l := l or 1;
     end;
   s2l := l;
end;

begin
   assign(f, 'desert.in'); reset(f);
   readln(f, n, nc);
   for q := 1 to nc do          {citesc cele nc linii ale constructiei}
     begin
       readln(f, s);
       if s='' then break;
       v[q] := s2l(s);      {si le codific ca nc longint-ti in vectorul v}
     end;
   for x := 2 to nc do      {se construieste vectorul 'Urm' din teorie in vectorul dyn}
     begin
       y := dyn[x-1];
       if v[x]=v[y+1] then
         dyn[x] := dyn[x-1]+1
       else
         if v[x]=v[1] then dyn[x] := 1;
     end;

   for q := 1 to n do       {ia pe rand cele n linii ale fotografiei}
     begin
       readln(f, s);        {citesc o linie}
       if s='' then break; {am terminat}
       l1 := s2l(s);        {codific primele 32 caractere }
       inc(n);
       for x := 0 to 223 do{parcurge cele 255-32=223 caractere}
         begin
           y := poz[x];     {scoate in y pozitia caracterului, este si contor}
           if v[y+1]=l1 then {verific daca se potriveste l1 cu paternul din v}
             begin          {daca DA}
               inc(y);        {am mai gasit o potrivire - o numar}
               if y=nc then  {daca a gasit nc potriviri => am detectat o constructie}
                 begin
                   inc(count);  {o numar si incep sa caut}
                   y := dyn[y]; {de la urmatorul "deplasament posibil corect"}
                 end;
             end
           else             {daca NU}
             {caut urmatorul "deplasament posibil corect"}
             while (y>0) and (v[y+1]<>l1) do y := dyn[y];
           poz[x] := y;          {si il pun in vectorul poz}
           l1 := l1 shl 1;       {ma deplasez spre dreapta in s si}
           l1:=l1 or byte((x<223) and (s[x+33]='#')); {codific ultimul bit}
         end;
     end;
   close(f);
   assign(f, 'desert.out'); rewrite(f);
   writeln(f, count);
   close(f);
end.
```