Vyhledávání vzoru v řetězci

- **Řetězec** je *n*-tice prvků
- **Délka řetězce** – počet prvků v řetězci (*n*)
- **Znakový řetězec** – řetězec, jehož prvky jsou znaky
- **Shoda řetězců**
  - považujeme-li řetězce za *n*-tice prvků (znaků), pak dva řetězce jsou shodné, mají-li stejnou délku a jestliže jsou shodné i stejnolehlé prvky (znaky) obou řetězců
- **Def.:** Nechť **V** a **R** jsou řetězce znaků nad danou abecedou a nechť ***m*** je délka řetězce **V**, ***n*** je délka řetězce **R** a platí $n \geq m$. Označíme v_i *i*-tý znak řetězce **V** a r_j *j*-tý znak řetězce **R**.
  - Řetězec **V** nazveme **vzorem** a řešíme úlohu nalezení vzoru **V** v řetězci **R**.
  - **Nalezení vzoru** znamená nalezení **pozice *k*** v řetězci **R**, od které platí
    - znak $v_i = r_j$ pro $i = 1 \ldots m,\ j = k \ldots (k+m-1)$
    - $(k+m-1) \leq n$

Metody hledání vzoru v řetězci

Naivní algoritmus

- **Princip:**
  - začínáme od prvního znaku řetězce **R** (r_1)
  - porovnáme stejnolehlé znaky řetězce **R** a vzoru **V**
  - shodují-li se všechny znaky vzoru, hledání úspěšně končí
  - dojde-li k neshodě znaků, posuneme se v řetězci **R** o 1 znak a opakujeme postup
  - hledání končí neúspěchem, když do konce řetězce **R** zbývá méně znaků než ***m*** (délka vzoru)
- **Příklad:**

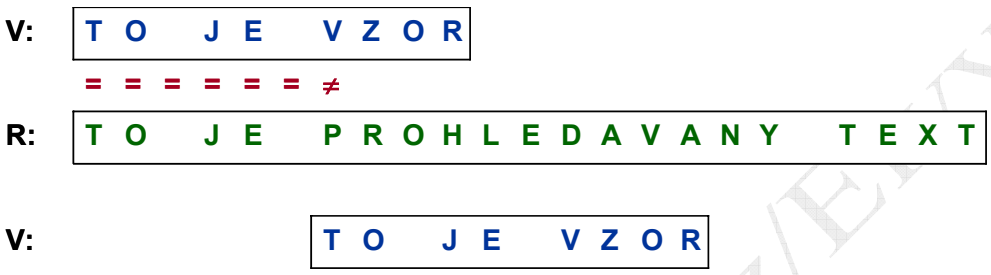
```
V:  T O   J E   V Z O R
    = = = = = = ≠
R:  T O   J E   P R O H L E D A V A N Y   T E X T

V:      T O   J E   V Z O R
        ≠
R:  T O   J E   P R O H L E D A V A N Y   T E X T
…
```

KMP algoritmus

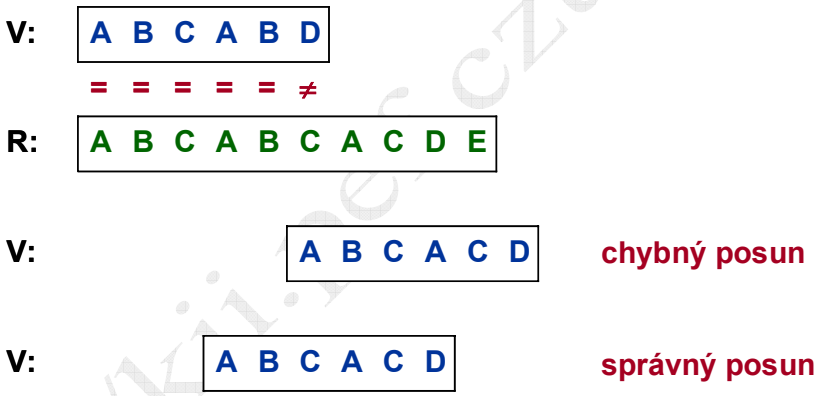
- Autoři **K**nuth, **M**orris a **P**ratt

- **Princip:**
  - využívá skutečnosti, že když se vzor shoduje s prohledávaným textem (**R**) v ***i*** znacích a ***i*+1** znak se liší, je možné v **R** pokračovat od pozice, kde došlo k neshodě

  - **Příklad:**



  - To platí pouze v případě, že se ve shodujících se ***i*** znacích vzoru znak nebo skupina znaků neopakuje.



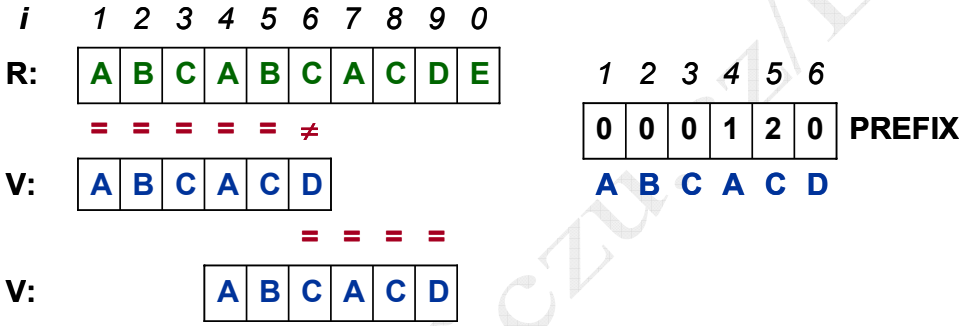
  - Opakující se posloupnosti znaků ve vzoru **V** se zjišťují **prefixovou funkcí** reprezentovanou polem **PREFIX[1..*m*]** – ***m*** je délka **V**

```
PREFIX[1]:=0;
k:=0;
for j:=2 to m do begin
     while (k>0 and V[k+1]<>V[j]) do k:=PREFIX[k];
     if (V[k+1]=V[j]) then k:=k+1;
     PREFIX[j]:=k
End;
```

- Hledání vzoru **V[1..*m*]** v textu **R[1..*n*]** probíhá podle následujícího algoritmu:

```
j:=0;
for i:= 1 to n do begin
     while (j>0 and V[j+1]<>R[i]) do j:=PREFIX[j];
     if V[j+1]=R[i] then j:=j+1;
     if j:=m then begin
          pozice:=i-m+1;
          j:=PREFIX[j];
     end;
end;
```

- **Příklad:**
  - **V**[1..*m*], ***m*** = **6**, **R**[1..*n*], ***n*** = **10**

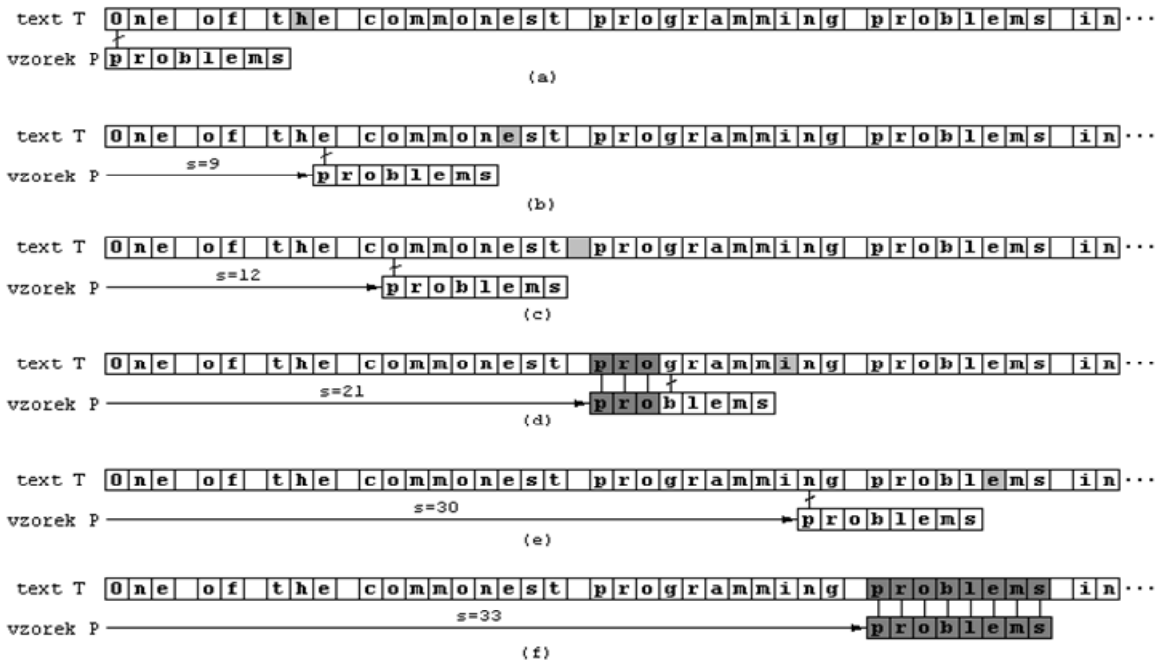


  - **pozice** := *i* – *m* + 1 = 9 – 6 + 1 = **4**

Metoda Quicksearch

- **Princip:**
  - vzor **V[1..*m*]** porovnáváme s textem **R[1..*n*]** od začátku znak po znaku jako u naivního algoritmu
  - objevíme-li neshodu znaků, vezmeme znak v **R** na pozici za koncem vzoru **V** – **testovací znak R[*m*+1]**
  - jestliže se **testovací znak nevyskytuje ve vzoru V**, můžeme vzor posunout za testovací znak v **R** – **posun o *m*+1 znaků**
  - jestliže se **testovací znak vyskytuje ve vzoru V**, posuneme vzor tak, aby se **testovací znak shodoval s prvním výskytem** znaku ve vzoru

- **Příklad:**



Další metody

- **Boyer-Mooreův algoritmus (BM algoritmus)**
  - prohledává text **R** odzadu
  - využívá ke zvýšení efektivnosti dvě heuristiky („*objevování založené na zkušenostech*“)
    - heuristika špatného znaku
    - heuristika dobré přípony
- **Baeza-Yates-Gonnetův algoritmus (BYG algoritmus)**
  - vyhledávání pomocí bitových masek výskytu jednotlivých znaků ve vzoru **V**
  - masky se postupně pokládají pod text **T** a hledá se jejich „průhlednost“

Porovnání metod

- Vzor **V[1..*m*]** a text **R[1..*n*]**
- **Naivní algoritmus** operační složitost – ***m.n***
- **KMP algoritmus** operační složitost – ***m + n***
- **Quicksearch** operační složitost – ***n / m***
- **BM algoritmus** operační složitost – ***m.n***
- **BYG algoritmus** operační složitost – ***m + n***