Definice

Konečný automat budeme dále nazývat vyhledávací stroj. Naším cílem bude pro slovník K o celkové délce P písmen sestavit vyhledávací stroj v čase $O(P)$, který umožní vyhledání v textu délky n všech slov z K v čase $O(n)$.

Vyhledávací stroj pro abecedu S, konečnou množinu K slov v S definujme jako čtveřici $M = (Q, g, f, out)$:

- $Q = 0, ..., q$ je konečná množina stavů,
- $g : Q \times S \longrightarrow Q \cup \{\perp\}$ je tzv. přechodová funkce, která každé dvojici $< stav, písmeno >$ přiřadí buď nový stav, nebo symbol „$\perp$“, který znamená „nedefinováno“, přičemž platí, že $g(0, s)$ je definováno pro všechna $s \in S$.
- $f : Q \longrightarrow Q$ je tzv. zpětná funkce, pro kterou platí $f(0) = 0$
- $out : Q \longrightarrow P(K)$ je výstupní funkce, která každému stavu přiřadí podmnožinu K.

Činnost vyhledávacího stroje

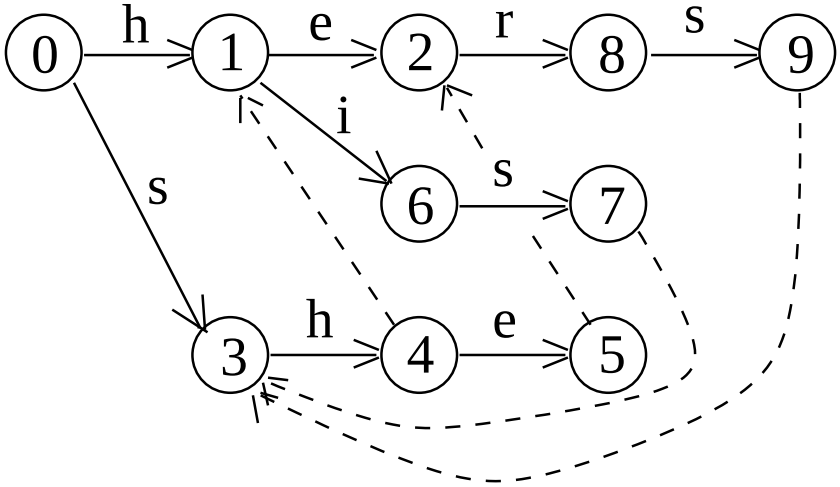
Algoritmus 1 (interpret vyhledávacího stroje)

VSTUP: text $x = x_1, x_2, x_3, ...x_n$, vyhledávací stroj $M = (Q, g, f, out)$ pro S a $K = y_1, y_2, ...y_m$

VÝSTUP: posloupnost dvojic $< i, y_p >$, kde $i = 1..n$, $p = 1..k$

```
begin
  state := 0;
  for i:= 1 to n do begin
    while  g(state,x[i]) nedefinován do state := f(state);
    state := g(state, x[i]);
    for y in out(state) do Report(i,y)
  end;
end.
```

Příklad: S je latinská abeceda, $K = \{\texttt{he}, \texttt{she}, \texttt{his}, \texttt{hers}\}$. Na následujícím obrázku je vyznačena stavová množina vyhledávacího stroje spolu se všemi přechody prostřednictvím fce g (plné šipky) i prostřednictvím funkce f (přerušované šipky), pro něž cílovým stavem není stav 0 (pro tento stav dodefinujeme $g(0, x_i) = 0$ pro všechna x_i různá od `h` a `s` a $f(0) = 0$). Funkce *out* je zadána tabulkou.



Stav	*out*
0	{}
1	{}
2	{`he`}
3	{}
4	{}
5	{`she`, `he`}
6	{}
7	{`his`}
8	{}
9	{`hers`}

Při reprezentaci stavu výpočtu dvojicí $< i, s >$, kde i je délka dosud přečteného úseku textu a s je okamžitý stav, bude historie zpracování textu $x =$ `ushers` vypadat takto: $< 0,0 >$, $< 1,0 >$, $< 2,3 >$, $< 3,4 >$, $< 4,5 >$ {vydá `she`,`he`}, $< 4,2 >$, $< 5,8 >$, $< 6,9 >$ {vydá `hers`}.

Vyhledávací stroj tedy pracuje tak, že sleduje shodu prohledávaného textu s co nejdelším vyhledávaným vzorkem a v okamžiku, kdy narazí na jeho konec nebo kdy dojde k neshodě, použije „záchrannou funkci“ f k tomu, aby nalezl stav, ze kterého již bude moci pokračovat. Okamžitý stav reprezentuje veškerou informaci, kterou vyhledávací stroj získal z dosud přečteného úseku textu a který je relevantní pro hledání výskytu vzorků. Při pohybu mezi stavy jsou v každém stavu hlášeny vzorky s tímto stavem spojené. Dále se samozřejmě budeme zajímat o korektní vyhledávací stroje, t.j. o ty, které pro každé $x = x_1x_2x_3....x_n$, vydají v i-té iteraci hlavního cyklu interpretu (t.j. po přečtení předpony $x_1x_2x_3....x_i$ prohledávaného textu) právě ty dvojice $< i, y_p >$, kde y_p je příponou $x_1x_2x_3....x_i$.

Správná funkce vyhledávacího stroje plyne z následujících faktů:

1) Každé vyhledávané slovo je v automatu reprezentováno nějakou cestou z počátečního stavu. Všimněte si, že stavy stroje spolu s přechodovou funkcí g tvoří strom, a tedy cesty se nemohou nijak spojovat.
2) Každá cesta z počátečního stavu je počátkem nějakého hledaného slova.
3) Zpětná funkce vede do stavu jehož odpovídající řetězec je zakončením řetězce stavu, ze kterého funkce vede. Navíc skáčeme do takového stavu, že jeho řetězec je nejdelší možný.
4) Vždy když je nějaké hledané slovo zakončením řetězce nějakého stavu, je toto slovo zahrnuto ve výstupní funkci pro daný stav.

Časová složitost interpretace tohoto automatu bude zřejmě $O(n)$. V každém kroku se totiž posuneme buď o stav dál, nebo se budeme vracet. Pokud se vracíme, museli jsme někdy předtím postupovat kupředu a to alespoň tolikrát, kolikrát se teď vracíme. Tedy celkový počet návratů je menší nebo roven počtu postupů kupředu, a tedy návraty nám nemohou zhoršit časovou složitost.

Konstrukce vyhledávacích strojů

Při konstrukci vyhledávacího stroje k dané množině vzorků K se přidržíme obrázku popisující fci g vyhledávacího stroje pro vzorky {`hers`, `she`, `he`, `his`}. Nejprve algoritmem 2 sestrojíme stavovou množinu $Q = 0, 1, ..., q$, přechodovou funkci g a „polotovar“ o výstupní funkce out. Poté algoritmem 3 sestrojíme zpětnou funkci f a rozšíříme funkci o na výstupní funkci out.

Algoritmus 2

VSTUP: Množina vzorků $K = \{y_1, y_2, ..., y_k\}$ neprázdných slov v S

VÝSTUP: $Q = \{0, ..., q\}$, $g : Q \times S \longrightarrow Q \cup \{\perp\}$, $o : Q \longrightarrow P(K)$

METODA: Založíme stavový strom s kořenem 0 a postupně k němu budeme připojovat cesty odpovídající jednotlivým prvkům K tak, aby Q a g měly požadované vlastnosti.

```
var q: integer;
procedure Enter (x[m]$); {procedura pro připojení slova y = x[1],x[2],x[3]...x[m]}
begin
  s := 0; j := 1;
  while  j < m  and  g(s, x[j]) definováno begin
    s := g(s, x[j]); j := j + 1;
  end;
  while  j < m do begin
    q := q + 1;              {přidej nový stav}
    g(s, x[j]) := q;         {rozšiř definici}
    s := q;                  {pokroč do nového stavu}
    j := j + 1;              {pokroč k dalšímu písmeni}
  end;
  o(s) := x;  {koncovému stavu s jako jediný výstup příslušné slovo}
end;
```

```
begin                                {tělo algoritmu}
  q := 0;                            {založ strom}
  for p := 1 to k do Enter(y[k]);    {připoj větve}
  for all x in S do                  {dodefinuj přechody z kořene}
    if g(0,x) nedefinováno then g(0,x) := 0;
end.
```

Není těžké se přesvědčit, že Algoritmus 2 pracuje v čase $O(P)$ a jeho výstupy mají vlastnosti popsané ve zdůvodnění správnosti.

Algoritmus 3

VSTUP: $Q = 0, ..., q$, $g : Q \times S \longrightarrow Q \cup \{\perp\}$, $o : Q \longrightarrow P(K)$

VÝSTUP: $f : Q \longrightarrow Q$, $out : Q \longrightarrow P(K)$

METODA: Začneme od kořene stavového stromu a postupně probíráme stavy v pořadí rostoucí hloubky. Každý stav, ve kterém definujeme funkce f a out, zařadíme do fronty, v níž si pamatujeme stavy, jejichž potomci ještě nemají tyto funkce definovány. Algoritmus končí vyčerpáním všech stavů.

```
begin
  vytvoř prázdnou frontu queue;
  definuj f(0) = 0; definuj out(0) = []; {prázdná množina}
  for all x in S begin                   {zpracuj potomky 0}
    s := g(0,x);
    if s <> 0 then begin
      definuj f(s) := 0;
      definuj out(s) := o(s);
      zařaď s na konec queue
    end;
  end;
  while queue neprázdná do begin
    r := první prvek queue; vyřaď r z queue;
    for all x in S do begin              {zpracuj potomky s}
      if g(r,x) definováno then begin
        s := g(r,x);                     {potomek}
        t := f(r);                       {má definováno f}
        while g(t,x) nedefinováno do t:= f(t);
        definuj f(s) := g(t,x);
        definuj out(s) := o(s) + out(f(s));
        zařaď s na konec queue;
      end;
    end;
  end;
end.
```

I zde se můžeme lehce přesvědčit, že Algoritmus 3 pracuje v čase $O(P)$ a jeho výstupy tvoří vyhledávací stroj korektně vzhledem k množině vzorků K.

Celková časová a paměťová složitost konstrukce vyhledávacího stroje algoritmem Aho-Corasickové je $O(P)$, přičemž na reprezentaci vyhledávacího stroj potřebujeme paměť o velikosti $O(P)$.