# Érdekes informatika feladatok

XXIII. rész

**Véges determinisztikus automaták programozása**

Legyen $\Sigma$ egy véges, nem üres halmaz. *Absztrakt szimbólum*nak nevezünk egy $\Sigma$-beli elemet. A $\Sigma$ halmazt *véges ábécé*nek nevezzük. A $\Sigma$ elemeit általában *betűknek*, *jeleknek* vagy *szimbólumoknak* nevezzük. Egy szimbólumot általában az $s$ karakterrel fogunk jelölni. A $\Sigma$ elemeiből (szimbólumaiból) álló véges sorozatokat *szavaknak*, *jelsorozatoknak* vagy *szimbólumsorozatoknak* nevezzük, s általában a $p$ karakterrel jelöljük. A szimbólumsorozatok tehát szimbólumokból álló halmazok.

- A $\Sigma$ elemeiből álló szimbólumsorozatok összességét $\Sigma^*$-gal jelöljük.
- A $\Sigma^*$ elemének tekintjük az ún. *üres szimbólumsorozat*ot is, amelyet $\varepsilon$-al jelölünk, és nem tartalmaz egyetlen szimbólumot sem.
- A $\Sigma - \{\varepsilon\}$ szimbólumsorozat-halmazt $\Sigma^+$-al jelöljük.
- Egy $p$ szimbólumsorozat hosszán értjük a $p$ szimbólumsorozat szimbólumainak a számát, s ezt $|p|$-vel vel jelöljük. Eszerint $|\varepsilon| = 0$.

Két $\Sigma^*$-beli szimbólumsorozatnak a *konkatenációján* vagy *szorzatán* értjük azt a $\Sigma^*$-beli szimbólumsorozatot, amely az adott két szimbólumsorozatunk egymásután való leírásából adódik. Tehát, ha $v$ és $w$ két szimbólumsorozat, akkor $vw$ is szimbólumsorozat és $|vw| = |v| + |w|$. A konkatenáció általában nem kommutatív művelet. Az üres szó $\varepsilon$, a konkatenációra nézve a semleges elem szerepét tölti be: $\forall\, p \in \Sigma^*$ esetén $\varepsilon p = p\varepsilon = p$.

Egy $v$ szimbólumsorozatot a $w$ szimbólumsorozat *részszimbólumsorozatának* nevezünk, ha léteznek olyan $v_1$ és $v_2$ szimbólumsorozatok, amelyekkel a $w = v_1 v v_2$ egyenlőség fennáll. Amennyiben $v \neq \varepsilon$, akkor $v$ *valódi részszimbólumsorozata* $w$-nek. Ha $v_1 = \varepsilon$, akkor $v$ a $w$ *elejét*, ha $v_2 = \varepsilon$, akkor $v$ a $w$ *végét* képezi.

Két szimbólumsorozatot egyenlőnek nevezünk, ha azok szimbólumról szimbólumra megegyeznek.

Bármely $i$ pozitív egész számra értelmezhetjük bármely $p$ szimbólumsorozat $i$-edik hatványát, vagyis $i$-szer önmagával való konkatenációját, és ezt $p^i$-vel jelöljük. Minden $p$ szimbólumsorozatra $p^0 = \varepsilon$.

Egy $p$ szimbólumsorozat *tükörképén* értjük azt a szimbólumsorozatot, amelyben $p$ szimbólumai fordított sorrendben szerepelnek, és ezt $p^{-1}$-el jelöljük. $\varepsilon^{-1} = \varepsilon$.

A szimbólumsorozatoknak egy tetszőleges halmazát *nyelv*nek nevezzük, és általában $L$-el jelöljük. Minden nyelv tehát $\Sigma^*$-nak egy részhalmaza.

Az *üres nyelv*et, vagyis azt a nyelvet, amelynek egyetlen szimbólumsorozata sincs a $\varnothing$ szimbólummal jelöljük. Ez a nyelv nem tévesztendő össze a $\{\varepsilon\}$ nyelvvel, amely egyedül az üres szimbólumsorozatot tartalmazza.

Egy nem üres nyelv *véges*, ha csak végesen sok szimbólumsorozatot tartalmaz, különben *végtelen.*

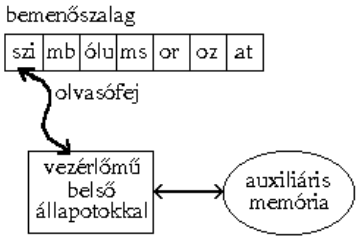
Az így bevezetett nyelvfogalom a *formális nyelv* fogalma.

Ha azt szeretnénk eldönteni, hogy egy szimbólumsorozat beletartozik-e egy nyelvbe vagy sem, vagyis a $p \in L$ reláció logikai értékét (igaz, hamis) szeretnénk megkapni, *automatá*kra van szükségünk.

Képzeljünk el egy olyan elemzőberendezést (*automatát*), amelybe egy tetszőleges szimbólumsorozatot beadva „IGEN" vagy „NEM" választ kapunk aszerint, hogy a kérdéses szimbólumsorozat beletartozik-e egy adott nyelvbe, vagy sem.

Egy ilyen automata *belső állapotok*kal rendelkezik, amelyek közül van egy kitüntetett állapot a *kezdőállapot,* és egy kitüntetett állapothalmaz, a *végállapotok halmaza.* Az automata megkapja a szimbólumsorozatot. Ezt úgy foghatjuk fel, hogy az automata egy *bemenőszalag*gal rendelkezik, a bemenőszalag *mezők*re van osztva és minden szimbólum egy-egy mezőbe kerül. Az automata továbbá egy *olvasófej*jel van ellátva. A bemenőszalagra felírunk egy $p$ szimbólumsorozatot úgy, hogy az első szimbólum éppen az olvasófej előtt legyen. Ezután az automatát a kezdőállapotból indítjuk. Minden belső állapotra és beolvasott szimbólumra az automata újabb állapotba megy át ugrásszerűen. Ha a $p$ szimbólumsorozat utolsó szimbólumának beolvasása után az automata végállapotba kerül, akkor az „IGEN" választ szolgáltatja, vagyis azt mondjuk, hogy „*az automata felismerte a szimbólumsorozatot*".

Az automatát így szemléltethetjük:




Ha létezik egy $A$ automata, amely felismer minden $p \in L$ szimbólumsorozatot, akkor azt mondjuk, hogy az $A$ automata felismeri az $L$ nyelvet és ezt $L(A)$-val jelöljük.

Ha az automata egy belső állapotra és egy beolvasott szimbólumra legfennebb egy újabb állapotba megy át, akkor azt mondjuk, hogy az automata *determinisztikus.*

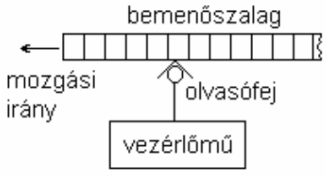
Egy *véges determinisztikus automatá*n az $A = (Q, \Sigma, \delta, q_0, F)$ rendezett ötöst értjük, ahol:

- $Q$ egy véges, nem üres halmaz, az automata belső állapotainak halmaza.
- $\Sigma$ egy véges ábécé, a bemeneti szimbólumok halmaza.
- $\delta$ a $Q\mathrm{x}\Sigma$ halmaznak egy leképezése a $Q$-ra, az átmenetfüggvény, véges determinisztikus automatáknál tehát:

$$\delta: Q\mathrm{x}\Sigma \rightarrow Q.$$

- $q_0 \in Q$ a kezdő állapot.
- $F \subseteq Q$ a végállapotok halmaza.

A véges automaták a legegyszerűbb automaták. Véges ábécével, belső állapotokkal rendelkeznek és az átmenetfüggvény értelmében minden beolvasott szimbólumra felvesznek egy új állapotot. A következő ábrán egy ilyen automatát szemléltetünk, amely el van látva egy olvasófejjel, és ez előtt halad el a mezőkre felosztott bemenőszalag, a mozgási iránynak megfelelően:




**Példa:** Adjunk meg egy véges determinisztikus automatát, amely felismeri a hárommal osztható, tízes számrendszerben ábrázolt számokat:

$A = (Q, \Sigma, \delta, q_0, F)$

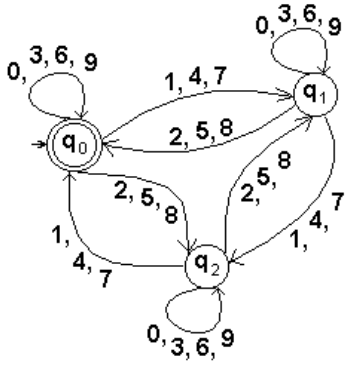
$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$F = \{q_0\}$

| | | | |
|---|---|---|---|
| $\delta(q_0, 0) = q_0$, | $\delta(q_0, 1) = q_1$, | $\delta(q_0, 2) = q_2$, | $\delta(q_0, 3) = q_0$, |
| $\delta(q_0, 4) = q_1$, | $\delta(q_0, 5) = q_2$, | $\delta(q_0, 6) = q_0$, | $\delta(q_0, 7) = q_1$, |
| $\delta(q_0, 8) = q_2$, | $\delta(q_0, 9) = q_0$, | $\delta(q_1, 0) = q_1$, | $\delta(q_1, 1) = q_2$, |
| $\delta(q_1, 2) = q_0$, | $\delta(q_1, 3) = q_1$, | $\delta(q_1, 4) = q_2$, | $\delta(q_1, 5) = q_0$, |
| $\delta(q_1, 6) = q_1$, | $\delta(q_1, 7) = q_2$, | $\delta(q_1, 8) = q_0$, | $\delta(q_1, 9) = q_1$, |
| $\delta(q_2, 0) = q_2$, | $\delta(q_2, 1) = q_0$, | $\delta(q_2, 2) = q_1$, | $\delta(q_2, 3) = q_2$, |
| $\delta(q_2, 4) = q_0$, | $\delta(q_2, 5) = q_1$, | $\delta(q_2, 6) = q_2$, | $\delta(q_2, 7) = q_0$, |
| $\delta(q_2, 8) = q_1$, | $\delta(q_2, 9) = q_2$ | | |

Egy ilyen megadás kényelmetlen és nem esztétikus, ezért az automatákat olyan irányított gráffal szokás megadni, amelynek a csúcspontjai az automata különböző állapotainak felelnek meg, az élei pedig az egyes bemenőjelek hatására történő állapotváltozásokat jelentik, vagy olyan táblázattal, amelynek oszlopai a szimbólumokat, sorai pedig az állapotokat jelentik. A sor és oszlop által meghatározott helyre pedig, az átmenetfüggvénynek megfelelően, az új állapot kerül:



ahol a  állapotot, a  kezdőállapotot, a  pedig végállapotot jelöl, vagy pedig megadhatjuk táblázattal a következőképpen:

| □ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $q_0$ | $q_0$ | $q_1$ | $q_2$ | $q_0$ | $q_1$ | $q_2$ | $q_0$ | $q_1$ | $q_2$ | $q_0$ |
| $q_1$ | $q_1$ | $q_2$ | $q_0$ | $q_1$ | $q_2$ | $q_0$ | $q_1$ | $q_2$ | $q_0$ | $q_1$ |
| $q_2$ | $q_2$ | $q_0$ | $q_1$ | $q_2$ | $q_0$ | $q_1$ | $q_2$ | $q_0$ | $q_1$ | $q_2$ |

A következő *Borland Delphi* program véges determinisztikus automaták működését szimulálja objektumorientáltan, a konkrét példa pedig a hárommal osztható számok ellenőrzése. A program beolvas egy legtöbb string-nyi hosszúságú (2 GB) tetszőleges egész számot, és az automata eldönti, hogy osztható-e hárommal vagy sem. Megjegyzendő, hogy a program jelen állapotában nem végez mindenre kiterjedő hibaellenőrzést.

```
program vdautomata;
{$APPTYPE CONSOLE}

uses SysUtils;

type
   TAutomata = class
   private
     fQ: array of byte;
     fSigma: array of char;
     fDelta: array of array of byte;
     fQo: byte;
     fF: array of byte;
     fInnerState: byte;
   public
     procedure SetQ(const Q: array of byte);
     procedure SetSigma(const Sigma: array of char);
     procedure SetQo(Qo: byte);
     procedure SetF(const F: array of byte);
     procedure SetDelta(q: byte; s: char; nq: byte);
     procedure NewState(s: char);
     function IsFinalState: boolean;
   end;

procedure TAutomata.SetQ;
var i: integer;
begin
  SetLength(fQ, Length(Q));
  for i := 0 to Length(Q)-1 do
    fQ[i] := Q[i];
end;

procedure TAutomata.SetSigma;
var i: integer;
begin
  SetLength(fSigma, Length(Sigma));
  for i := 0 to Length(Sigma)-1 do
    fSigma[i] := Sigma[i];
end;

procedure TAutomata.SetQo;
begin
  fQo := Qo;
  fInnerState := Qo;
end;
```

```
procedure TAutomata.SetF;
var i: integer;
begin
  SetLength(fF, Length(F));
  for i := 0 to Length(F)-1 do
    fF[i] := F[i];
end;

procedure TAutomata.SetDelta;
var i, x, y: integer;
begin
  if Length(fDelta) = 0 then
    begin
     SetLength(fDelta, Length(fQ));
     for i := 0 to Length(fQ)-1 do
       SetLength(fDelta[i], Length(fSigma))
    end;
  x := -1;
  for i := 0 to Length(fQ) do
    if fQ[i] = q then
     begin
      x := i;
      break;
     end;
  if x = -1 then
   begin
     writeln(q, ' - not a state!');
     readln;
     halt(1);
   end;
  y := -1;
  for i := 0 to Length(fSigma) do
    if fSigma[i] = s then
     begin
      y := i;
      break;
     end;
  if y = -1 then
   begin
     writeln(s, ' - not in alphabet!');
     readln;
     halt(1);
   end;
  fDelta[x, y] := nq;
end;

procedure TAutomata.NewState;
var i, x, y: integer;
begin
  x := fInnerState;
  y := -1;
  for i := 0 to Length(fSigma) do
    if fSigma[i] = s then
     begin
      y := i;
      break;
     end;
  if y = -1 then
   begin
     writeln(s, ' - not in alphabet!');
     readln;
     halt(1);
   end;
  fInnerState := fDelta[x, y];
end;
```

```
function TAutomata.IsFinalState;
var i: integer;
begin
  Result := false;
  for i := 0 to Length(fF)-1 do
   if fF[i] = fInnerState then
    begin
     Result := true;
     break;
    end;
end;

var
  h: TAutomata;
  szam: string;
  i: integer;
begin
  h := TAutomata.Create;
  h.SetQ([0, 1, 2]);
  h.SetSigma(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']);
  h.SetQo(0);
  h.SetF([0]);
  h.SetDelta(0, '0', 0);
  h.SetDelta(0, '1', 1);
  h.SetDelta(0, '2', 2);
  h.SetDelta(0, '3', 0);
  h.SetDelta(0, '4', 1);
  h.SetDelta(0, '5', 2);
  h.SetDelta(0, '6', 0);
  h.SetDelta(0, '7', 1);
  h.SetDelta(0, '8', 2);
  h.SetDelta(0, '9', 0);
  h.SetDelta(1, '0', 1);
  h.SetDelta(1, '1', 2);
  h.SetDelta(1, '2', 0);
  h.SetDelta(1, '3', 1);
  h.SetDelta(1, '4', 2);
  h.SetDelta(1, '5', 0);
  h.SetDelta(1, '6', 1);
  h.SetDelta(1, '7', 2);
  h.SetDelta(1, '8', 0);
  h.SetDelta(1, '9', 1);
  h.SetDelta(2, '0', 2);
  h.SetDelta(2, '1', 0);
  h.SetDelta(2, '2', 1);
  h.SetDelta(2, '3', 2);
  h.SetDelta(2, '4', 0);
  h.SetDelta(2, '5', 1);
  h.SetDelta(2, '6', 2);
  h.SetDelta(2, '7', 0);
  h.SetDelta(2, '8', 1);
  h.SetDelta(2, '9', 2);
  writeln('A szam: ');
  readln(szam);
  for i := 1 to Length(szam) do
   h.NewState(szam[i]);
  if h.IsFinalState then writeln('A szam oszthato 3-mal.')
                    else writeln('A szam nem oszthato 3-mal.');
  h.Free;
  readln;
end.
```

**Kovács Lehel István**