Dr Strauber Györgyi – Sóti Lászlóné

A számítástudomány alapjai. II.

T a r t a l o m j e g y z é k

Bevezetés

A Számítástudomány alapjai II. tantárgy két nagy témakörbe nyújt bevezetést.

Az első témakör az adatszerkezetek és algoritmusok területe. Az előadás anyaga, amely a jelen jegyzetben kapott helyet, az alapvető adatszerkezetekkel (verem, sor, halmaz, zsák, gráf, fa) ismerteti meg a hallgatót és bemutat néhány ezekhez az adatszerkezetekhez kapcsolódó algoritmust. Az 1. fejezet foglalkozik ezen a témakörön belül az adatszerkezetek általános leírásával, valamint a verem adatszerkezethez kapcsolható néhány algoritmussal, a 2. fejezet a gráfelméletbe és a gráfokhoz kapcsolódó algoritmusokba nyújt betekintést, a 3. fejezet az algoritmusok egy nagy csoportját, az alapvető keresési ill. rendezési algoritmusokat ismerteti. A gyakorlaton az előadás ezen részének elmélyítése a cél, még több algoritmus megismerésével és ezek megvalósításával mondatszerű leíró nyelven. A gyakorlat anyaga a „Számítástudomány alapjai II., Programozási feladatok, feladatsorok, megoldások" című munkafüzetben kapott helyet.

Az előadás második nagy feldolgozott témaköre a formális nyelvek és véges automaták elmélete. Ezen belül szó esik a generatív grammatikákról, ezek osztályozásáról, a Chomsky-féle hierarchiáról, a véges determinisztikus- és nemdeterminisztikus automatákról, a környezetfüggetlen nyelvekről, ezek kapcsolatáról a programozási nyelvek szintaktikájával, valamint felismerőikről, a veremautomatákról. Ez a teljes témakör a jelen jegyzet 4. fejezetében került kifejtésre.

A Számítástudomány alapjai II. tantárgy és így ez a jegyzet is feltételezi a Számítástudomány alapjai I. és a Bevezetés az informatikába tantárgyak tananyagának ismeretét.

1.Adatszerkezetek

1.1 Adattípusok

Ez a tárgy feltételezi a ***Bevezetés a programozásba I*** tárgy elsajátítását, ahol részletesen sor kerül az adatok és adattípusok definiálására és osztályozására. Ezen ismeretek alapján az ***elemi adattípusokkal*** mint például az ***egész, valós, logikai(boolean), karakter, felsorolás, illetve mutató típus*** stb. nem kívánunk foglalkozni. A hangsúlyt az összetett adattípusokra, ezek és a műveleteik megvalósítására, valamint a tárban való ábrázolásukra helyezzük. Bemutatunk továbbá néhány algoritmust mely ezen adat szerkezetek használatán alapul.

Összetett adattípusok

Az adatelemek sorrendi, szerkezeti összefüggéssel, elemi típusok alkalmazásával hozhatóak létre. A típus attól függ, milyen összetett, új struktúrát hozunk létre az elemiekből.
Megkülönböztethetők:

- Azonos típusú elemek sokasága. (pld. tömb, szöveg, verem, sor, lista, fa, halmaz)
- Különböző típusú részek egy szerkezetben (rekord).
- Különböző típusú részek egy szerkezetben, de valamilyen feltételtől függően különböző adatokkal (alternatív szerkezet).

Az összetett adattípusok többsége az első kategóriába tartozik, amelyeket a rajtuk végzett műveletek alapján különböztetünk meg.
Ilyen lehetséges műveletek:

- Elem értékének lekérdezése, megváltoztatása (tetszőleges, első, utolsó, következő).
- Elemszám meghatározása.
- Új elem felvétele (tetszőleges, első, utolsó, következő helyre).
- Egy elem törlése.
- Üresség vizsgálat.
- Részsorozat felhasználása, megváltoztatása.

Elemek sorrendje lehet: - tárbeli – azaz fizikai,
- valódi – azaz logikai.

Ezek különbözők lehetnek, ezért a szerkezeti összefüggéseket is tárolni kell.
Az összetett típusokat a *struktúrájuk szerint* három fő csoportba osztjuk:

- ***Direkt szorzat típus***: Értékeit akár *n*, akár különböző típus értékhalmazából veszi. Az *n* halmaz lehet azonos vagy különböző. Jelölése: $T = T_1 \times T_2 \times \ \ldots \ \times T_n$

- ***Unió*** vagy alternatív ***típus***: Értékeit *n* különböző típus valamelyikéből veszi.
  Jelölése: $T = T_1 \cup T_2 \cup \ \ldots \ \cup T_n$

- ***Sokaság típus:*** Értékeit több, de véges sok azonos típusú elemből képzett egységek alkotják. Alapvető csoportosítási elv e típusnál az elemek sorrendiségére vonatkozik.
  - ***Halmaz típus***: Itt az elemek között nincs semmiféle sorrendiség. Ilyenek a *halmaz, multihalmaz, táblázat, file*, stb.

- ***Sorozat típus***: Az elemek között egyértelmű rákövetkezőségi reláció van, az utolsó kivételével mindegyik elemet egy elem követi és az első kivételével mindegyiket egy előzi meg. Tehát megkülönböztetett első és utolsó elemről is beszélhetünk. A memóriában elfoglalt helyük (fizikai sorrend) és a sorozat definíciója szerinti sorrend (logikai sorrend) alapján többféle ábrázolásmódról beszélhetünk. Ilyenek:

  tömb
  Az elemszám rögzített, ezen adatszerkezetnél, de bármelyik elemére egy vagy több indexszel hivatkozhatunk, a tömb dimenziójától függően.

  szöveg
  Ez esetben az elemszám nem minden nyelvben rögzített, valamint egyszerre több elemmel is dolgozhatunk.

  verem
  Ez az adattípus egy olyan sorozat, melynek egyik végével tudunk csak műveletet végezni. (Az új elem a többi "tetejére" kerül, míg kivenni vagy elvenni elemet csak a "tetejéről" lehet. Olyan mint egy zsák, vagyis „stack”.)
  Angol rövidítése: LIFO: Last In First Out

  sor
  Ez az adattípus egy olyan sorozat, melynek mindkét végével tudunk műveletet végezni, de csak egyfajtát. (Az új elem a többi után kerül, míg elvenni elemet csak a sor elejéről lehet. Olyan mint egy várakozási sor, vagyis „queue”.)
  Angol rövidítése: FIFO: First In First Out

  lista
  Ez az adattípus egy olyan sorozat, melynek minden elemével tudunk műveletet végezni, az elemek fizikai sorrendjétől eltérő logikai sorrendet tudunk kijelölni (mutatók segítségével). Bárhová lehet új elemet beszúrni vagy elemet törölni. Egy elem mutatója a logikai sorrendben őt követő elem címe.
  Beszélhetünk egyirányú vagy kétirányú kapcsolt listától.
- ***Hierarchikus típus***: Ezen típusnál minden elemet egy elem előz meg közvetlenül (melyet az adott elem szülőjének is szokás nevezni), és minden elemnek több rákövetkezője (gyereke) lehet. Ilyenek a *fák, bináris fák*, stb.
- ***Hálós típusok***: Ez esetben minden elemet több elem is megelőzhet közvetlenül, és minden elemnek több rákövetkező eleme is lehet. Ilyenek az *irányított és irányítatlan gráfok, hálók,* stb.

1.2. Adatszerkezetek absztrakt analízise

Definíció: Absztrakt elemi adat: Olyan adategység, mellyel a programozás adott szintjén végezhetők műveletek, de annak részeivel nem.

Definíció: ***Elemi adattípus (objektum)***: egy (A,M) kettős, ahol A az adattípusba tartozó absztrakt adatok nem üres halmaza, M pedig az m: $A \times A \times ... \times A \rightarrow A$ alakú műveletek véges halmaza.

Példa:

1., Egész típus absztrakciója:
A:={...,-1,0,1,...} (egész számok)
M:={+,-}
Szintaktika: + : $A \times A \rightarrow A$
- : $A \times A \rightarrow A$

2., Boolean (vagy logikai) típus:
A:= { ↑,↓ } M:= {∧, ∨, ¬ },
ahol ↑ az igaz, ↓ a hamis értéket jelöli.
Szintaktika: ∧ : $A \times A \rightarrow A$
∨ : $A \times A \rightarrow A$
¬ : $A \rightarrow A$

Definíció: ***Összetett adattípus*** absztrakciója egy (V,F) kettős,
ahol $V=\{V_1, V_2,...,V_m\}$, $m \geq 2$ és
V_1 az összetett adatok (vagy adatszerkezetek) nem üres halmaza,
V_2 az összetett adatokat alkotó elemi adatok nem üres halmaza,
V_3... V_m segédhalmazok (meglétük opcionális).

F a következő f_i műveletek véges halmaza:
$f_i : V_{i1} \times V_{i2} \times ... \times V_{ik} \rightarrow V_{in}$ $(k \geq 0)$

és az f_i műveletek között létezik a V_1 halmazba képező, azaz az
$f_j : V_{j1} \times V_{j2} \times ... \times V_{jm} \rightarrow V_1$ $(m \geq 0)$
alakú műveleteknek egy olyan részhalmaza, melyek egymás utáni alkalmazásával a V_1 halmaz minden eleme előállítható.

Példa: Halmaz (set)

V = { H, HE, B }	H:	halmazok halmaza
↑ ↑ ↑	HE:	halmaz elemei (elemi adatok, pl. egész számok)
V_1 V_2 V_3	B:	{↑, ↓}, segédhalmaz.

F= { empty, insert, has, delete }
↑ ↑ ↑ ↑
f_1 f_2 f_3 f_4

A műveletek szintaktikája:

empty: $\rightarrow H$: $f_1 : \rightarrow (V_1)$ az üres halmazt hozza létre,

insert: $H \times HE \rightarrow H$: $f_2 : V_1 \times V_2 \rightarrow (V_1)$ betesz egy elemet a halmazba,

has: $H \times HE \rightarrow B$: $f_3 : V_1 \times V_2 \rightarrow V_3$ megvizsgálja, hogy egy elem eleme-e a halmaznak,

delete: $H \times HE \rightarrow H$: $f_4 : V_1 \times V_2 \rightarrow V_1$ töröl egy elemet a halmazból.

A fenti műveletek közül az empty (f_1), az insert (f_2) és a delete (f_4) művelet felel meg annak a feltételnek, hogy a halmazok halmazába képezzen, azaz egy halmazt hozzon létre. Ezen műveletek segítségével V_1 minden eleme, azaz minden halmaz előállítható, sőt ahhoz, hogy V_1 minden elemet előállítsunk, már az empty és az insert művelet is elég.

Definíció: Tekintsük a fenti

$f_j : V_{j1} \times V_{j2} \times ... \times V_{jn} \rightarrow V_1$ ($n \geq 0$),

alakú műveleteket. Ezek közül a minimális számút, melyek egymás utáni alkalmazásával V_1 minden eleme előállítható, ***konstrukciós művelet***eknek nevezzük.
A többi művelet: szelekciós művelet.

Példa 1: Halmaz:
empty, insert: konstrukciós műveletek,
has, delete: szelekciós műveletek

Definíció: Ha $f_j : V_{j1} \times V_{j2} \times ... \times V_{jn} \rightarrow V_1$ ($n \geq 0$), ahol ($\forall$ i, $1 \leq i \leq n$) ($j_i \neq 1$), akkor a $v = f_j (v_{j1}, v_{j2},...,v_{jm})$ $v_{ji} \in V_{j1}$ i=1,...,n adatszerkezeteket ***generátor vagy konstruktor elem***eknek nevezzük. (Tehát konstruktor elem egy olyan összetett adat, amelynek létrehozásához nem használtunk fel már létező összetett adatot.)

Megjegyzés: Általában egy generátor elem van, melyet a $f: \rightarrow V_1$ konstans függvény állít elő.

Példa 2: Halmaz:
empty: $\rightarrow$ H generátor függvény, mely az üres halmazt (generátorelem) hozza létre.

Példa 3.: Zsák (bag), mely annyiban különbözik a halmaztól, hogy benne egy elem többször is szerepelhet.

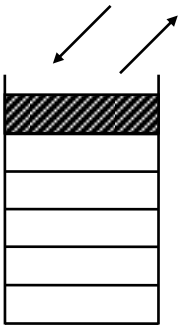
V = { B, BE, I }, ahol B: bag (nem az előbbi B(oolean)!!) BE: zsák elemei (pl. egész számok, karakterek) I: egész vagy lehet Bool(ean)
F={ bempty, binsert, bmany (bhas), bdelete }

Szintaxis:
bempty: $\rightarrow$ B,
binsert: $B \times BE \rightarrow B$,
bmany: $B \times BE \rightarrow I$ (helyette bevezethető: bhas: B x BE $\rightarrow$ Bool),
bdelete: $B \times BE \rightarrow B$,

ahol bempty, binsert, bhas, bdelete művelet jelentése megegyezik a halmazoknál leírtakkal, bmany függvény visszaadja, hogy egy elem hányszor fordul elő a zsákban.

Konstrukciós műveletek: bempty, binsert
Szelekciós műveletek: bmany, bdelete

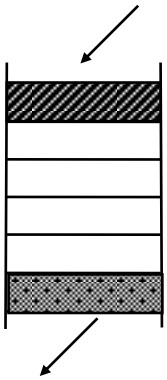
Példa 4.: Verem (LIFO)
V = { V (vermek halmaza), VE (verem-elemek halmaza) }
F = { create, push, top, pop }



Szintaxis:	create:		→ V
	push:	V × VE	→ V
	pop:	V	→ V
	top:	V	→ VE
Konstrukciós:	create, push		
Szelekciós:	pop, top		

(Itt create az üres vermet hozza létre, push betesz egy elemet a verem tetejére, top a verem tetején levő elemet, míg pop a tetejétől megfosztott maradék vermet adja vissza.)

Példa 5.: Sor (FIFO)
V = { S (sorok halmaza), SE (sor-elemek halmaza)}
F = { new, add, front, remove }



Szintaxis:	new:		→ S
	add:	S × SE	→ S
	remove:	S	→ S
	front:	S	→ SE
Konstrukciós:	new, add		
Szelekciós:	remove, front		

(Itt new az üres sort hozza létre, add betesz egy elemet a sor végére, front a sor elején levő elemet, míg remove az elejétől megfosztott maradék sort adja vissza.)

1.3.Adatszerkezetek szemantikája

Definíció: ***Szemantika***: a műveletek jelentését adja meg.

A műveletek jelentését olyan módon írjuk le, hogy megadjuk a szelekciós műveleteknek a konstrukciós műveletekkel előállított adatszerkezetekre gyakorolt hatását. Ezeket az azonosságokat szemantika-axiómáknak nevezzük.

Példa 1.: Halmaz: Legyen $h \in H$ (ahol H a halmazok halmaza, azaz legyen h egy halmaz) és $e_1, e_2 \in HE$ (ahol HE a halmaz lehetséges elemeinek halmaza)

A szemantika axiómák halmaz esetén:
delete (empty,e) = empty
delete (insert(h, e_1), e_2) =
ha $e_1 = e_2$, akkor delete(h, e_1)
különben insert(delete(h, e_2), e_1)

ugyanis:

e:= e_1= e_2
ha e ∈ h , akkor insert(h,e)=h és így delete(insert(h,e),e) = delete(h,e)
ha e ∉ h , akkor delete(h,e)=h és így delete(insert(h,e),e) = h = delete(h,e)

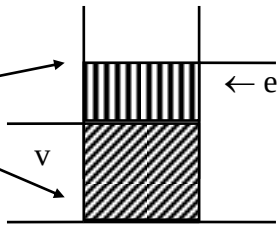
has(empty,e)=↓
has(insert(h, e_1), e_2)=
ha e_1= e_2 , akkor ↑
különben: has(h, e_2)

Példa 2.: Zsák: Legyen b ∈ B (B a zsákok halmaza) és e_1, e_2 ∈ BE (ahol BE a lehetséges zsák-elemek halmaza)
A szemantika axiómák zsák esetén:
bdelete(bempty,e)=bempty
bdelete(binsert(b, e_1), e_2)=
ha e_1= e_2 , akkor b
különben: binsert(bdelete(b, e_2), e_1)
bhas(bempty,e)=↓
bhas(binsert(b, e_1), e_2)=
ha e_1= e_2 , akkor ↑
különben: bhas (b, e_2)

Példa 3.: Verem: v ∈ V (vermek halmaza) és e∈VE (vermet alkotó elemek halmaza)

A szemantika axiómák:

pop(push(v,e))=v
top(push(v,e))=e
pop(create)=create
top(create)=error

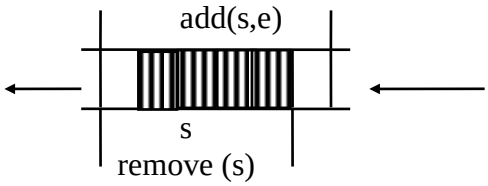


push (v,e)

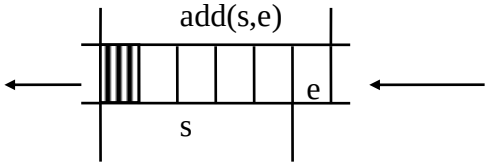
Példa 4.: Sor: s ∈ S (sorok halmaza)és e ∈ SE (sor alkotó elemek halmaza)

A szemantika axiómák:

remove(add(s,e))=
ha s=new, akkor new
különben: add(remove(s),e)



front(add(s,e))=
ha s=new, akkor e
különben: front(s)



remove(new)=new
front(new)=error

Általában egy axiómarendszerrel kapcsolatban az a kérdés merül fel, hogy:

- teljes-e, azaz minden egyéb tulajdonság levezethető-e az axiómákból?
- konzisztens-e, azaz nem vezethetők-e le egymásnak ellentmondó tulajdonságok az axiómákból?

Egy adott összetett adatszerkezetre vonatkozóan a szemantika axiómák rendszerével kapcsolatban is megfogalmazhatjuk azt az elvárást, hogy teljes és konzisztens legyen, azaz az adatszerkezetek összes további tulajdonsága levezethető legyen az axiómákból és ezek a tulajdonságok ne legyenek ellentmondóak. Ahhoz azonban, hogy vizsgálni tudjuk az adatszerkezetek tulajdonságait, szükségünk van annak definiálására, hogy mikor tekintünk két összetett adatszerkezetet egyenlőnek, azaz meg kell adnunk az adatszerkezetre vonatkozóan ez egyenlőség axiómáját.

Definíció: Két összetett adatszerkezetet akkor nevezünk ***egyenlő***nek, ha a szelekciós műveletekkel kiválasztott megfelelő részeik egyenlőek.

Példa 1.: Verem

Tegyük fel, hogy VE elemei, azaz a vermet alkotó elemek már ismert tulajdonságúak, így itt az egyenlőség eldönthető.

Legyen $v_1, v_2 \in V$, azaz v_1, v_2 két verem.

Szelekciós műveletek: pop, top

Ekkor definíció szerint:

$v_1 = v_2 \quad \equiv \quad (pop(v_1)=pop(v_2) \wedge top(v_1)=top(v_2))$

Ez az egyenlőség axiómája vermekre vonatkozóan.

Példa 2.: Halmaz

Legyen $h_1, h_2 \in H$, azaz h_1, h_2 két halmaz.

Az egyenlőség axiómája ebben az esetben:

$$h_1 = h_2 \quad \equiv \quad ((\forall e, e \in HE)(has(h_1,e)=has(h_2,e) \wedge delete(h_1,e)=delete(h_2,e))) \vee (h_1=empty \wedge h_2=empty)$$

De az általunk elképzelt halmazfogalom esetében két halmaz akkor egyenlő, ha az elemei egyenlőek.

Azaz:

$$h_1 = h_2 \quad \equiv \quad ((\forall e, e \in HE)(has(h_1,e)=has(h_2,e))) \vee (h_1=empty \wedge h_2=empty)$$

Belátható, hogy a fenti követelmény elégséges a két halmaz egyenlőségéhez, azaz igaz az alábbi

Állítás: $((\forall e, e \in HE)(has(h_1,e)=has(h_2,e))) \Rightarrow ((\forall e, e \in HE)(delete(h_1,e)=delete(h_2,e)))$

Bizonyítás: Teljes indukcióval történhet a konstrukciós műveletek alapján.

1.4.Adatszerkezetek tulajdonságai

Vizsgáljuk az adatszerkezetek tulajdonságait!
Az alábbiakban felsorolunk néhány tételt, melyek a műveletek közötti azon kapcsolatokat írják le, melyeket az axiómák nem rögzítenek. Ha az axiómarendszerünk teljes, ezeket a tételeket – melyek a műveletek tulajdonságait írják le – már bizonyítanunk kell tudni az axiómák alapján:

Példa: Halmazok esetén:

1., insert(insert(h,e_1),e_2)=
insert(insert(h,e_2),e_1)

2., delete(delete(h,e_1),e_2)=
delete(delete(h,e_2),e_1)

3., has(delete(h,e_1),e_2)= ha $e_1=e_2$, akkor ↓
különben: has(h,e_2)

4., insert(delete(h,e_1),e_2)=
ha $e_1=e_2$, akkor insert(h, e_2)
különben: delete(insert(h, e_2), e_1)

5., has(h, e) ⇒ insert(h,e)=h

6., ¬ has(h, e) ⇒ delete(h,e)=h

Tételek bizonyítása:

<u>1. módszer:</u> az egyenlőség mindkét oldalát behelyettesítjük az egyenlőség axiómájába.

Példa : Fenti 1. állítás.
Legyen e' tetszőleges

has(insert(insert(h,e_1), e_2), e')= ← szemantika axiómája miatt
| h^* |

ha $e_2=e'$, akkor ↑
különben: has(h^*,e')=has(insert(h,e_1),e')

has(h^*,e')=has(insert(h,e_1),e')= ←
ha $e_1=e'$, akkor ↑
különben: has(h,e')

<u>Tehát összefoglalva:</u>
has(insert(insert(h,e_1), e_2), e')=
| h_1 |

ha e'= e_1 ∨ e'= e_2 , akkor ↑
különben: has(h,e')

Mivel a végeredmény független az indextől, így azok felcserélésével ugyanez kapható:

has(insert(insert(h,e_2), e_1), e')=
| h_2 |

ha e'= $e_1 \vee$ e'= e_2 , akkor ↑
különben.: has(h,e')

Tehát h_1 és h_2 halmaz ugyanaz.

2. módszer:
A konstrukciós műveletek száma szerinti teljes indukció.

Példa: Fenti 2. állítás:

1., Legyen h=empty Ekkor
delete(delete(empty,e_1), e_2) = delete(empty, e_2)=empty
↑ szemantika ↑
Az indexek felcserélésével ugyanez kapható.

2., Tegyük fel, hogy h'-re igaz az állítás.
3., Belátjuk: igaz h=insert(h',e)-re is.
Az egyenlet bal oldala:
delete(delete(insert(h',e), e_1), e_2)=

(Mivel delete(insert(h',e), e_1), =
ha e = e_1 , akkor delete(h', e_1)
kül.: insert(delete(h', e_1),e))

= ha e = e_1 , akkor delete(delete(h', e_1), e_2)
kül.: delete(insert(delete(h', e_1),e), e_2)
| h^* |

delete(insert(h^*,e), e_2)=
ha e= e_2 , akkor delete(h^*,e_2)= delete(delete(h', e_1), e_2)
kül.: insert(delete(h^*, e_2),e)=insert(delete(delete(h', e_1), e_2),e)

Tehát összefoglalva:
delete(delete(h, e_1), e_2)=
ha e= $e_1 \vee$ e= e_2 , akkor delete(delete(h', e_1), e_2)
kül.: insert(delete(delete(h', e_1), e_2),e)

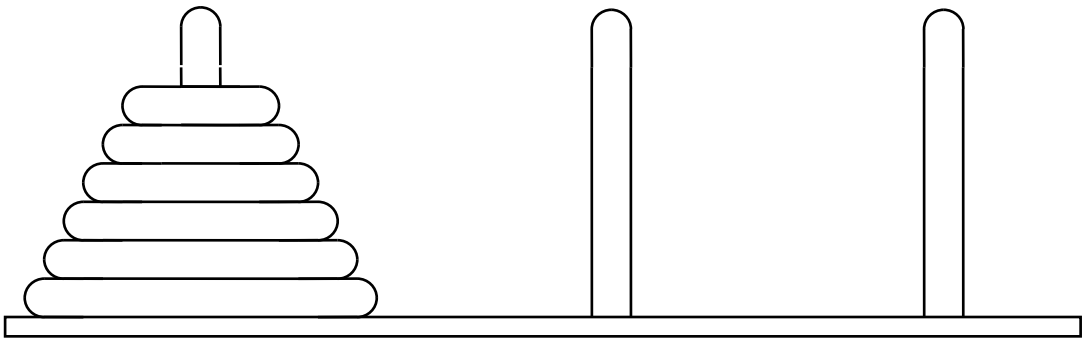
Az indexek felcserélésével kapható:
Az egyenlet jobb oldala
delete(delete(h, e_2), e_1) =
ha e= $e_1 \vee$ e= e_2 ,akkor delete(delete(h',e_2), e_1)
kül.: insert(delete(delete(h', e_2), e_1),e)

Tehát h'-re vonatkoztatott teljes indukciós feltevésből következik, hogy az egyenlet két oldala azonos.

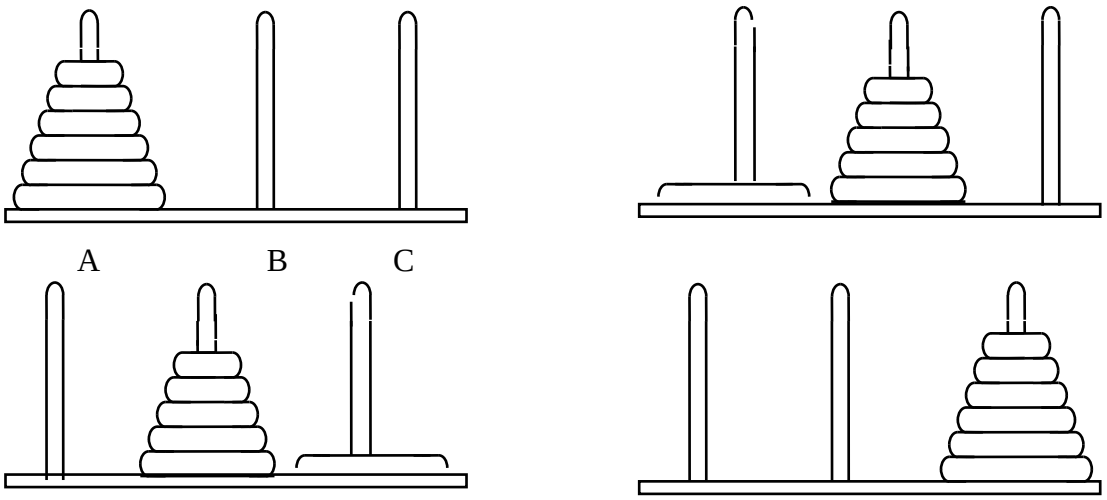
1.5. Vermek felhasználása

Pl. rekurzív algoritmusok esetében veremben tároljuk a
- paramétereket,
- a lokális változókat,
- a visszatérési címeket

Példa rekurzív algoritmusra a Hanoi tornyai feladat.
A Hanoi tornyai néven ismert feladat kiinduló helyzete n=6 korong esetén



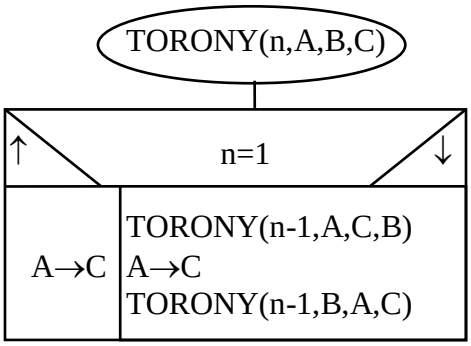
Cél: a korongok áthelyezése egyenként az első rúdról (A rúd) a harmadikra (C rúd) úgy, hogy a középső rudat (B rúd) használhatjuk segítségként, de minden esetben csak nagyobb korongra kisebb korongot rakhatunk.
A megoldás fő lépései n korong esetén:



1. Az A rúd legfelső n-1 darab korongjának áthelyezése B rúdra. (több lépésben – rekurzióval megoldva)
2. Az A rúd legfelső korongjának áthelyezése C rúdra.
   Jelölése a továbbiakban: A $\rightarrow$ C (egy lépés)
3. A B rúd legfelső n-1 darab korongjának áthelyezése a C rúdra. (több lépésben – rekurzióval megoldva)

A rekurzív algoritmus 4 paramétere a következő:
n: korongok száma,
A: kezdőrúd,
B: segédrúd,
C: célrúd.

TORONY(n,A,B,C): eljárás, mely n korongot áttesz A rúdról C rúdra B rúd segítségével. (Ekkor TORONY(1,A,B,C) a fenti jelöléssel: A $\rightarrow$ C, azaz a megadott 1 korong áthelyezése A rúdról C rúdra.)
Az eljárás struktogramja a következő:

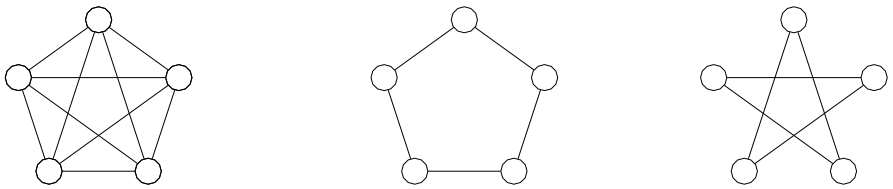


2. Gráfelmélet

2.1. Irányítatlan gráfok

Legyen V egy véges halmaz, E pedig egy V-beli rendezetlen elempárok véges halmaza, azaz E $\subset V \times V$ (rendezetlen párok). Ekkor a G=(V, E) párt ***irányítatlan g r á f*** nak nevezzük, ahol V a ***csúcsok***, E pedig az ***élek*** halmaza. (Rendezetlen az elempár, ha (v_1,v_2) és (v_2,v_1) között nem teszünk különbséget, $v_1, v_2 \in V$)

Ha $e=(v_1,v_2) \subset V \times V$ egy él, akkor azt mondjuk, hogy az ***é l*** a v_1 és v_2 csúcsokat köti össze, és ez a két végpontja az e$\subset$ E élnek. Ha két csúcs között vezet él, a csúcsokat ***szomszédos***nak nevezzük.
Definíció: Egy gráf egy csúcsa ***i z o l á l t c s ú c s ,*** ha nem indul belőle él.
Definíció: ***H u r o k é l*** az önmagába visszatérő él, azaz ha $v_1=v_2$.
Definíció: Ha két különböző nem hurok élnek a végpontjai azonosak, a két élet párhuzamos vagy **többszörös él**nek nevezzük, azaz két pontot több él is összeköthet.
Definíció: Az ***ü r e s g r á f*** csupa izolált pontból álló gráf.
Definíció: Az ***e g y s z e r ű g r á f*** nem tartalmaz sem hurokélet, sem többszörös élt.
Definíció: A ***t e l j e s g r á f*** olyan egyszerű gráf, amelyben bármely 2 különböző csúcs között vezet él, azaz bármely két pontja szomszédos a gráfnak.
Definíció: Egy G gráf ***k o m p l e m e n t e r e*** a G' gráf, amely G-t teljes gráffá egészíti ki. (ld.2.1. ábra)



Teljes gráf G G komplementere

2.1. ábra

Definíció: Egy $G_1=(V_1,E_1)$ gráf a G(V,E) gráf ***r é s z g r á f j a*** , ha $E_1 \subset E$, és és $V_1 \subset V$ tehát G1-et G-ből néhány él és/vagy csúcs elhagyásával kapjuk.

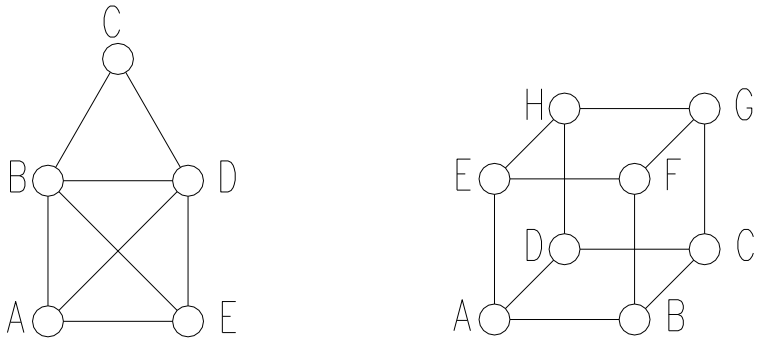
Definíció: A G_1 és G_2 gráfok ***i z o m o r f*** ak, ha létezik a csúcsok között olyan bijekció, hogy két G_1-beli csúcs között pontosan akkor vezet él, ha a bijekcióval megfeleltetett két G_2- beli csúcs is össze van kötve éllel.
Definíció: Egy csúcs ***f o k s z á m a*** a belőle kiinduló élek száma: ***d(x)***, $x \in V$.
Megjegyzés: Az a csúcs, amelyre $d(x) = 0$ izolált csúcs.
Tétel: Egy n csúcsú teljes gráfban minden csúcs fokszáma *n-1* és összesen $\binom{n}{2}$ élet tartalmaz.

Állítás: Minden gráfra igaz, hogy a fokszámok összege az élek számának kétszerese:

$$\sum_{v \in V(G)_i} d(v_i) = 2e(G)$$

Megjegyzés: A fenti állításból következik, hogy a fokszámok összege páros, azaz igaz az alábbi tétel is.
Tétel: A gráf páratlan fokú csúcsainak száma páros!
Definíció: ***Élsorozat:*** (x_1,x_2), (x_2,x_3), (x_3,x_4),… , (x_{n-1},x_n), tetszőleges élek sorozata, ahol $(x_i,x_{i+1}) \in E$, *i=1,2,… n-1*.
Definíció: ***S é t a*** : Két tetszőleges csúcsot összekötő élsorozat
Definíció: ***V o n a l*** : Olyan séta, melyben minden él legfeljebb egyszer szerepel (csúcsok többször is lehetnek).

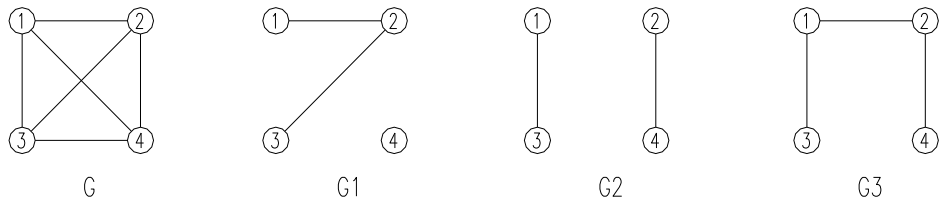
Definíció: Út : Olyan nyílt séta (kezdő és végpontja különböző) mely a sétában szereplő minden csúcsot legfeljebb egyszer érint.
Definíció: Kör : Olyan zárt séta (kezdő és a végpontja azonos), amely a többi, a sétában szereplő csúcsot legfeljebb egyszer érinti.
Tétel: Ha x_1 és x_2 csúcsok között létezik élsorozat, akkor létezik út is.



2.2. ábra

Példa: Az 2.2 ábrán látható G gráf esetén:
Élsorozat (de nem vonal): AB, BD, DA, AE, ED, DA (Mert az élek nem különbözők.)
Vonal (de nem út): BD, DA, AB, BE (Mert a csúcsok nem különbözők)
Kör: BE, EA, AB
Út: AB, BC,CD,DE vagy AD, DC vagy AE, ED, DC

Definíció: A gráf ***összefüggő*** , ha bármely két csúcs között vezet út.



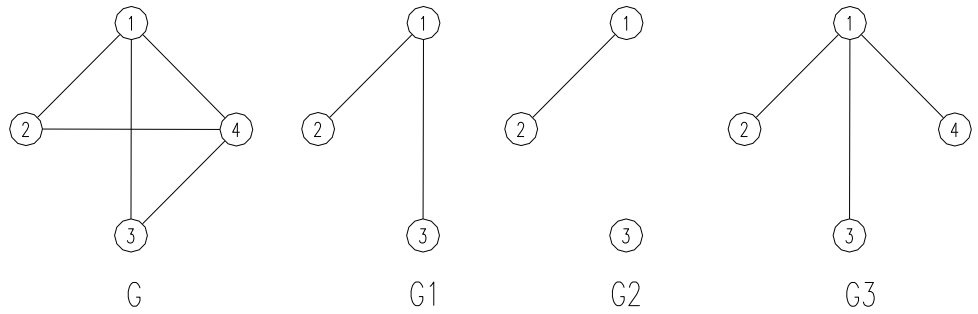
2.3. ábra

Példa:
Az 2.3 ábrán látható G1 gráf nem összefüggő, G2 gráf szintén nem összefüggő, G3 gráf összefüggő, G összefüggő teljes gráf.
Állítás: Ha a G gráf összefüggő, akkor *n* csúcs esetén az élek száma >= *n-1*

Definíció: Euler vonal : olyan vonal melyben a gráf minden éle pontosan egyszer szerepel! Van zárt és nyílt Euler vonal. A zárt Euler vonalat Euler körnek, a nem feltétlen zártat Euler –útnak nevezzük.
Megjegyzés: Az Euler út és kör nem út és kör valójában, hiszen egy csúcsot többször is tartalmazhat.
Tétel: Összefüggő gráfban akkor és csak akkor létezik zárt Euler vonal, ha minden csúcs fokszáma páros.
(Egy összefüggő G gráfban akkor és csak akkor van Euler út, ha G-ben a páratlan fokú pontok száma 0 vagy 2.)
Az 2.2 ábrán látható baloldali gráf Euler vonala: AB, BD, DC, CB, BE, ED, DA, AE.

Megjegyzés: Egy G gráfban ***Hamilton kör*** nek nevezünk egy H kört, ha G minden pontját (pontosan egyszer) tartalmazza. Egy utat pedig Hamilton útnak nevezzük, ha G minden pontját pontosan egyszer tartalmazza.
(A Hamilton kör és Hamilton út egy speciális kör ill. út a gráfban, ellentétben az Euler körrel és úttal.)
Az 2.2 ábrán látható jobboldali gráf Hamilton köre: AB, BC, CD, DH, HG, GF, FE, EA.
Az 2.2 ábrán látható baloldali gráf Hamilton köre: AB, BC, CD, DE, EA.

A Hamilton kör létezésének kérdése speciális esete a széles körben felmerülő Utazó ügynök problémának: Egy ügynöknek meg kell látogatnia útja során bizonyos városokat (n db) és végül haza kell mennie. Adott, hogy valamelyik városból egy másik városba milyen költséggel tud eljutni (repülőjegy, autóút ára). Természetesen szeretné az utak összköltségét minimalizálni. Ez a feladat sok alkalmazás során felmerül, és csak bizonyos speciális esetekben ismeretesek jó algoritmusok a megoldására.
Ha feltesszük, hogy bizonyos városokból nem lehet közvetlenül eljutni egyes másik városokba, míg a többi városba egységnyi költséggel lehet eljutni, és az ügynöknek, minden várost meg kell látogatnia, akkor a feladat a Hamilton kör létezésére redukálódik. Hiszen vegyük azt a gráfot, melynek pontjai a városoknak megfelelő n pont, és amelyben két pont akkor van összekötve, ha a nekik megfelelő városok között közvetlen összeköttetés van. Ebben a gráfban akkor és csak akkor van Hamilton kör, ha az ügynök n összköltséggel meg tud látogatni minden várost.
Hamilton kör létezésének nem ismert egyszerű, egyszerre szükséges és elégséges feltétele, s ugyancsak nincs gyors algoritmus sem a Hamilton kör keresésére. Külön-külön szükséges és elégséges feltételek [2]-ben találhatóak.



2.4. ábra

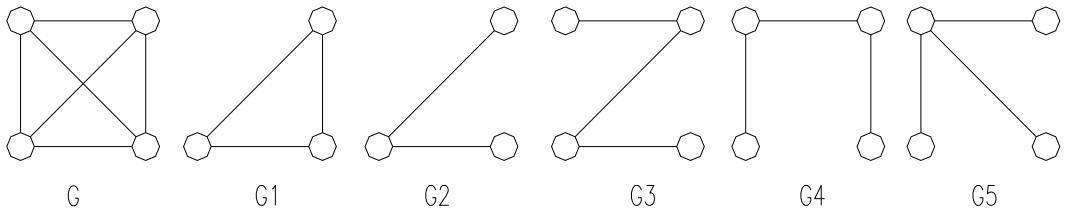
Definíció: G1 ***telített részgráfja*** G -nek ha $E_1 = V_1 \times V_1 \cap E$, ahol G=(V,E), $G_1=(V_1,E_1)$.

Definíció: G1 ***feszítő részgráfja*** G -nek ha $V_1 = V$.
***Példa*:** Az 2.4 ábrán látható G gráfnak G_1 telített, G_2 nem telített részgráfja, G3 feszítő részgráfja G-nek.

2.2. Fák és tulajdonságaik

Definíció: ***F á*** nak nevezzük az olyan összefüggő gráfokat, amelyikben nincs kör.

Állítás: Egy n pontú fa éleinek száma n-1.
A fa tetszőleges élét elhagyva már nem lesz összefüggő, és egy tetszőleges élet hozzávéve már lesz benne kör.(Azaz nem lesz fa.)
Definíció: Legyen G egyszerű összefüggő gráf. Az F fa a G gráf ***f e s z í t ő f á j a*** , vagy faváza, ha F feszítő részgráfja G-nek.
Tehát az F gráf a G gráf feszítőfája, ha F fa, és feszítő részgráfja G-nek.
Megjegyzés: Mivel a feszítőfa feszítő részgráf, ezért tartalmazza az eredeti gráf összes csúcsát.

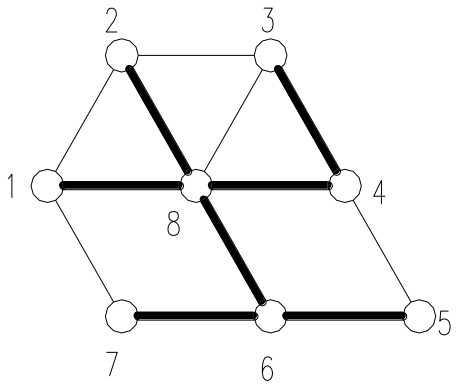


2.5. ábra

Az 2.5-s ábrán látható gráfok közül G teljes gráf, G_1 nem fa, G_2 fa, de nem faváz, mivel nem tartalmazza G összes csúcsát, G_3 feszítőfa azaz faváz, hasonlóan G_4 és G_5 is faváz.

Megjegyzés: Az *n* csúcsú gráf esetén a feszítőfához *n–1* élet kell kiválasztani.

***Állítás*:** G-nek akkor és csak akkor létezik feszítőfája, ha G összefüggő.



2.6 ábra.
Egy gráf egy lehetséges feszítőfája.

Biz.: => Ha G-nek létezik feszítőfája, akkor összefüggő. Ez az irány triviális, hiszen a feszítőfa maga is összefüggő.
<= Minden összefüggő G gráf tartalmaz feszítőfát.
Ha G-ben van kör, akkor hagyjuk el a kör egy tetszőleges élét. Ha a maradék gráfban megint van kör, ismét hagyjuk el ennek egy élét, és ezt az eljárást folytassuk egészen addig, amíg találunk kört. Ha már nincs több kör, akkor nézzük meg, mit kaptunk. Az eljárás során soha nem sérült meg az összefüggőség, hiszen mindig egy kör egyik élét hagytuk el. Nem hagytuk el a gráfnak egy pontját sem. Így a maradék gráf láthatóan G egy feszítőfája.
Legyen a csúcsok száma =*n*, az élek száma =*m*. Mivel a feszítőfa éleinek száma *n-1,* ezért a fenti algoritmus során *m-(n-1)* élet hagyunk el.

Definíció: Ciklikus szám :(Ciklometrikus szám, nullitás)
υ(G) = *m –(n-1) = m – n +1,* ahol *m* az élek, *n* pedig a csúcsok száma.
Összefüggő gráf esetén:

Ha υ(G) = 0 akkor a G fa.

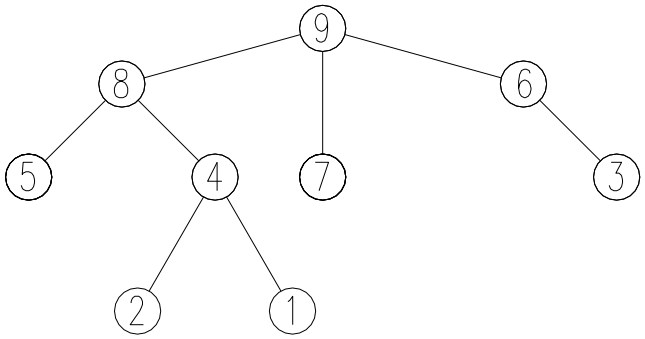
Ha υ(G) =1 akkor egy kör van a gráfban.

Ha υ (G) >= 1 akkor legalább υ(G) kör van a gráfban.

***Példa*:** Az 2.4 ábrán látható G gráf esetén: υ (G) = 5 – 4 +1 = 2 , Körök száma: 3

Definíció: A körmentes (nem feltétlen összefüggő) gráfokat ***erdő***nek nevezzük. Egy F gráf a G gráf ***feszítőerdeje***, ha F erdő és minden komponense feszítőfája G megfelelő komponensének.

Könnyen látható, hogy egy erdő összefüggő komponensei fák. Így teljesen hasonlóan a fentiekhez belátható, hogy ha az F erdő pontjainak száma n, komponenseinek száma k, akkor F –nek pontosan *n–k* éle van. Ennek speciális esete, amikor F fa, hiszen ekkor *1* komponensből áll.
Cayley bebizonyította, hogy az {1,2,…,n} pontokon – ha most különbözőknek tekintjük az egyébként izomorf gráfokat – pontosan n^{n-2} darab különböző fa adható meg. Ennek a tételnek a bizonyításához szükséges a Prüfer kód. A bizonyítást nem részletezzük, (lsd [2]) de a Prüfer – kód egyébként is hasznos, így ezt az alábbiakban bemutatjuk.
Az {1,2,…,n} pontokon adott fához rendeljünk egy számsorozatot a következőképpen. Hagyjuk el a fa elsőfokú pontjai közül a legkisebb indexűt, és jegyezzük fel a szomszédja (a vele összekötött egyetlen pont) indexét. Legyen ez v_1.
Ismételjük az eljárást a maradék fára, majd folytassuk egészen addig, amíg csak egy pont marad. Világos, hogy az utolsó pont az *n* sorszámú. Ugyanis ezt biztosan nem hagytuk el soha, hiszen mindig legalább két elsőfokú pont volt, és nyilván nem lehetett a kisebb sorszám az *n.* Ezért nem is kell, hogy a számsorozat végén feltüntessük.
Az így kapott $v_1,v_2,\ldots v_{n-2}$ sorozatot a fa ***Prüfer-kódjának*** nevezzük.
Például az 2.7 ábrán látható fa Prüfer –kódja: 4, 4, 6, 8, 8, 9, 9
E kódból bármikor visszaállítható a fa.

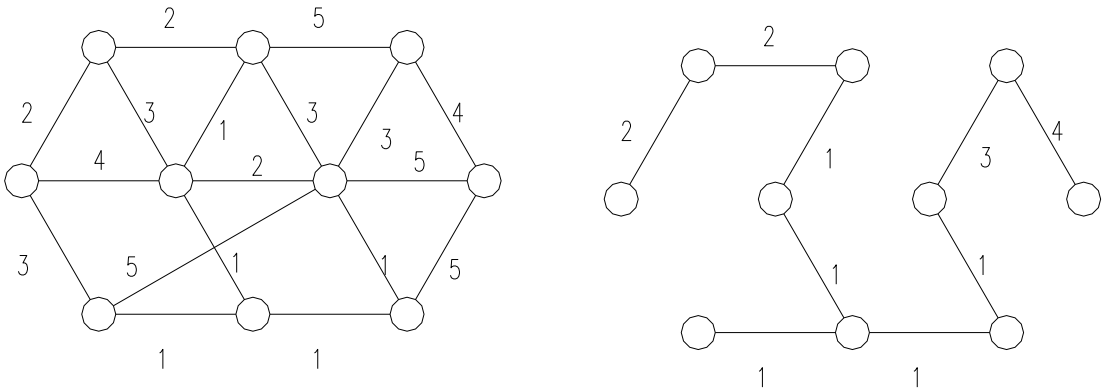


2.7. ábra

2.3. Minimális súlyú feszítőfa keresés

(A mohó algoritmus.)

Legyen adott n csúcsú egyszerű összefüggő gráf, valamint az élekhez rendelt valós nem negatív számok, melyek az élek hosszát vagy egyéb fizikai jellemzőjét írják le. Ezt a számot általában az él „súlyának” szokás nevezni. Keressük azt a kifeszítő fát, amelyben az élek összes súlya minimális.
Az 2.8 ábrán a fa minimális súlyú feszítőfája látható. Ennek összsúlya: 16.
Egy gráfnak több feszítőfája létezik, és ezek közül a minimális feszítőfa kiválasztása sem feltétlenül egyértelmű, de bármelyiket választjuk is, a minimális összsúlyuk azonos.



2.8. ábra

Gráf és a minimális súlyú feszítőfája.

Adjunk algoritmust, amely megkeresi a ***m i n i m á l i s s ú l y ú f e s z í t ő f á t*** !

Nyilván úgy kell az éleket kiválogatni a feszítőfába, hogy összességében a legkisebb súlyúak kerüljenek be. A használt algoritmus az ún. mohó algoritmus, azaz végrehajtása során minden lépésben a lehető legkisebb élet választjuk ki, feltéve, hogy ezzel az éllel nem alakítunk ki kört az eddigi fa szerkezetű gráfban. Ebben az esetben ez az él ki kell hogy maradjon. Azt is figyelni kell, hogy minden csúcs szerepeljen a fában. Ezeket az elveket fogalmazza meg a következő algoritmus.

KRUSKAL – féle MOHÓ algoritmus.

- Rendezzük súlyuk szerint növekvő sorrendbe az éleket.
- Válasszuk ki sorban az éleket, de olyat ne válasszunk ki, amelynek kiválasztásával kör keletkezne.
- Az előző pontot ismételjük *n-1* szer.

Az algoritmus pontosabban:

1. Az élek sorba rendezése a hozzájuk rendelt értékek növekvő sorrendje alapján.
2. Tekintsük a sorrendben következő *e* élet.
3. Ha az eddig kialakult R részgráfban *e* hozzáillesztésével kör keletkezne, akkor menjünk a 2. lépésre, azaz vegyük a következő *e* élet. Egyébként pedig *e*-t vegyük hozzá R-hez.
4. Ha R éleinek száma < *n-1*-nél akkor menjünk a 2. lépésre, különben vége az algoritmusnak.

Megjegyzés: Általában egy algoritmust ***mohó*** algoritmusnak nevezünk, ha végrehajtása folyamán minden lépésben az éppen a legjobbnak tűnő lehetőséget választjuk, és nem törődünk azzal, hogy esetleg egy most rosszabbnak tűnő választással végül jobb eredményt kaphatnánk.
A Kruskal algoritmus *mohó algoritmus* a legkisebb feszítőfa megkeresésére.
A mohó algoritmus más feladatokra is használható, de nem minden esetben ad feltétlenül jó megoldást!

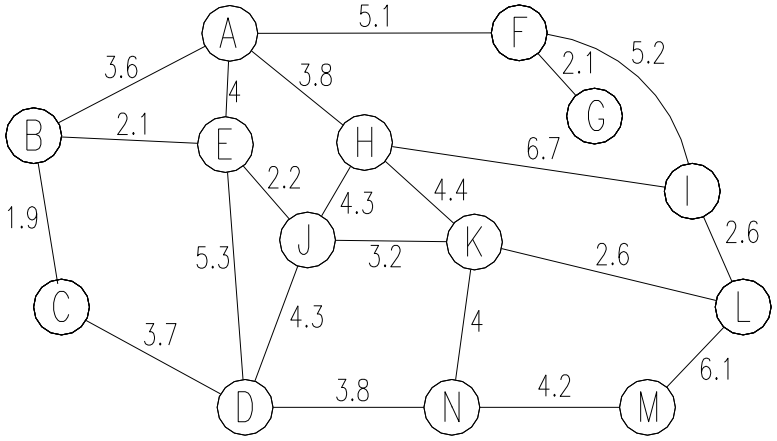
Számítógépes megvalósítás esetén felmerülő problémák:

- Az első lépéshez vannak jó sorba rendező algoritmusok. Ha több él értéke egyenlő, akkor mindegy, hogy melyiket vesszük előbbre, a minimális költség nem fog változni, esetleg a faváz.
- A 3. lépésnél felmerül, hogyan lehet eldönteni egy élről, hogy beválasztásával kört kapunk, vagy nem. Ennek a kérdésnek az eldöntésére szolgál az alábbi algoritmus:

A már kiválasztott csúcsokat soroljuk halmazokba aszerint, hogy az épülő feszítőfa melyik összefüggő részgráfjához tartozik. A most sorra kerülő új él 2 végpontját vizsgáljuk meg. 4 eset lesz:

1. Ha a 2 végpont azonos halmazban van, akkor a vizsgált él nem kerül be az épülő favázba, mert különben kört kapnánk.
2. Ha a 2 végpont nem ugyanabban a halmazban van, az új élet vegyük hozzá a feszítőfához, és egyesítsük a két halmazt.
3. Ha csak az egyik végpont szerepel valamelyik halmazban, akkor a másik végpontot tegyük be az első végpont halmazába, az új él pedig a feszítőfába kerül.
4. Ha egyik végpont sem szerepelt eddig a nyilvántartásban, akkor a 2 csúccsal kezdjünk újabb halmazt, a köztük lévő élet illesszük a favázba.

Lássuk ennek megvalósítását a 2.9 –es ábrán látható gráf példáján:



2.9. ábra

Növekvő sorrend:

BC	1.9
BE,FG	2.1
EJ	2.2
IL,KL	2.6
JK	3.2
AB	3.6
CD	3.7
AH,DN	3.8
AE,KN	4
MN	4.2

stb

Élek beválogatása:		Beválasztott él sorszáma:	hossza:
B-C	H_1={B,C}	1	1.9
B-E	H_1={B,C,E}	2	2.1
F-G	H_1, H_2{F,G}	3	2.1
E-J	$H_{1=}$={B,C,E,J}, H_2	4	2.2
I-L	H_1, H_2, H_3={I,L}	5	2.6
K-L	H_1, H_2, H_3={I,K,L}	6	2.6
J-K	H_1= $H_1 \cup H_3$={B,C,E,I,J,K,L}, H_2,	7	3.2
A-B	H_1={A,B,C,E,I,J,K,L}, H_2,	8	3.6
C-D	H_1={A,B,C,D,E,I,J,K,L}, H_2,	9	3.7
A-H	H_1={A,B,C,D,E,H,I,J,K,L}, H_2,	10	3.8
D-N	H_1={A,B,C,D,E,H,I,J,K,L,N}, H_2,	11	3.8
A-E	A$\in$ H_1, E$\in$ H_1	nincs	-
K-N	K$\in$ H_1, N$\in$ H_1	nincs	-
M-N	H_1={A,B,C,D,E,H,I,J,K,L,M,N}, H_2,	12	4.2
D-J	D$\in$ H_1, J$\in$ H_1	nincs	-
H-J	D$\in$ H_1, J$\in$ H_1	nincs	-
H-K	H$\in$ H_1, K$\in$ H_1	nincs	-
A-F	H_1= $H_1 \cup H_2$	13	5.1

Készen van a faváz. Hossza: 40.9

2.4. Irányított gráfok

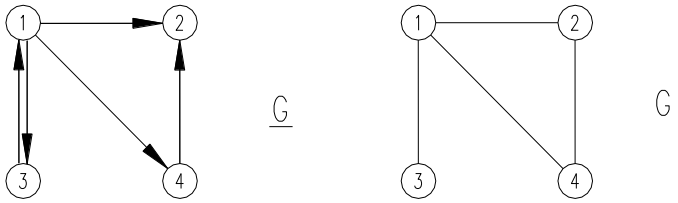
Definíció: Egyszerű irányított gráf a G = (V,E) kettős, ahol V a csúcsok, E az élek halmaza, E ={<x,y>, x≠y, rendezett párok, x,y ∈ V } ; (<x,y> és <y,x> megengedett, de hurok és többszörös él nem)

Megjegyzés: G-hoz hozzárendelhető egy G irányítatlan gráf az alábbi módon:

Ha G = (V,E), akkor G=(V,E)

V(G)=V(G), E = { [x,y] | <x,y> ∈ E vagy <y,x> ∈ E }, azaz az élek irányítását elhagyjuk, ha két csúcs között mindkét irányban vezetett él, akkor csak egyet tartunk meg. (2.10. ábra)

Példa



2.10. ábra

Definíció: Csúcsok fokszáma: Irányított gráfok esetén kétféle fokszám értelmezhető, a csúcsok „be” foka a csúcsba bemutató élek száma, a csúcsok „ki” foka a csúcsból induló élek száma.

$d_{be}(x)$: e= <y,x> élek száma

$d_{ki}(x)$: e= <x,y> élek száma

d(x) jelöli egy x∈V csúcs esetén a csúcsba bemutató és kimutató élek összességét.

$d(x)= d_{be}(x) + d_{ki}(x)$

Megjegyzés:

$\sum_x d_{be}(x) = \sum_x d_{ki}(x) = e(G)$, a gráf éleinek száma.

Példa.

Az 2.10. –es ábrán látható G gráfra $d_{be}(1)=1$; $d_{be}(3)=1$; $d_{ki}(1)=3$; $d_{ki}(3)=1$;

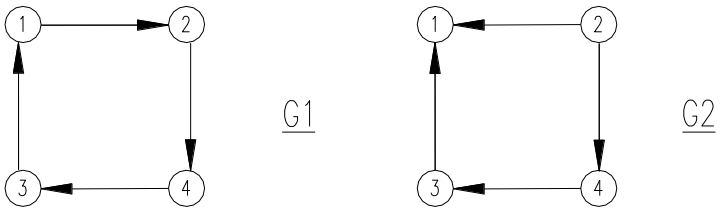
Tehát d(1)=4 ; d(3)=2;

Definíció: Irányított élsorozat:

$<x_0,x_1><x_1,x_2> ... <x_{k-1},x_k> \rightarrow <x_i,x_{i+1}> \in E$ tetszőleges élek sorozata.

Definíció: Irányított vonal, út, kör: hasonlóan értelmezhető, mint az irányítatlan gráfok esetén, természetesen az élek irányítottságát figyelembe véve.

Példa Az 2.11 ábrán látható G1 gráf tartalmaz irányított kört, míg a G2 nem.

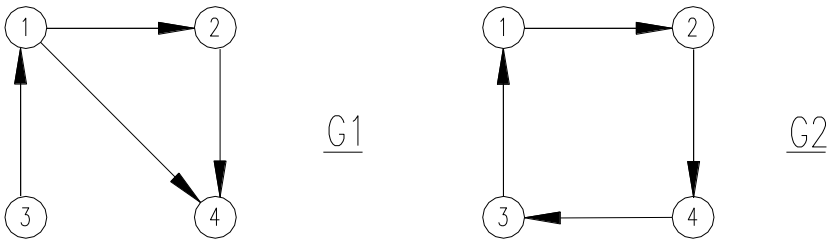


2.11. ábra

Definíció: Gyenge összefüggőség: Egy G gráf gyengén összefüggő, ha G összefüggő, ahol G a G –hez rendelt irányítatlan gráf.

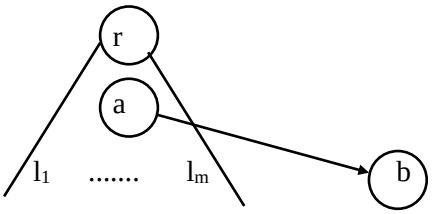
Definíció: Erős összefüggőség: G erősen összefüggő , ha bármely két x és y ∈ V csúcs esetén létezik $x \rightarrow y$ út és létezik $y \rightarrow x$ út is.

Példa: Az 2.12. ábra szerinti G1 gyengén összefüggő, de erősen nem. (Hiszen pld nem létezik az $1 \rightarrow 3$ út)
A G2 erősen és gyengén is összefüggő.



2.12. ábra

Megjegyzés: Ha egy irányított gráf erősen összefüggő, akkor gyengén is az. Visszafele az állítás nem igaz.
Irányított gráf esetén a fa gráf fogalma és definíciója lényegesen eltér az irányítatlan gráf esetén használttól.
Irányított fa gráf minden olyan gráf (és csak az), mely az alábbi rekurzív definícióval adható meg:
Definíció: Irányított fa (rekurzív definíció)
1., Legyen az 1 csúcspontból álló gráf egy irányított fa. Ennek gyökere és levele is adott 1 csúcspont.
2., Legyen G=(V,E) egy már előállított *k-1* csúccsal rendelkező irányított fa, melynek gyökere az *r* csúcspont, levelei az $l_1, \ldots l_m$ csúcsok.
3., A fenti fából egy új *b* csúcs ($b \notin V$) hozzá vételével az alábbi módon kapható irányított fa:
Legyen $a \in V$ egy már létező csúcs. Ekkor *b* hozzá vételével egy *b* új $a \rightarrow b$ élet is hozzá kell venni a fához.
Ekkor az új gráf: $\underline{G'}=(\underline{V'},\underline{E'})$, $\underline{V'}=\underline{V} \cup b$, $\underline{E'}=\underline{E} \cup \langle a,b \rangle$

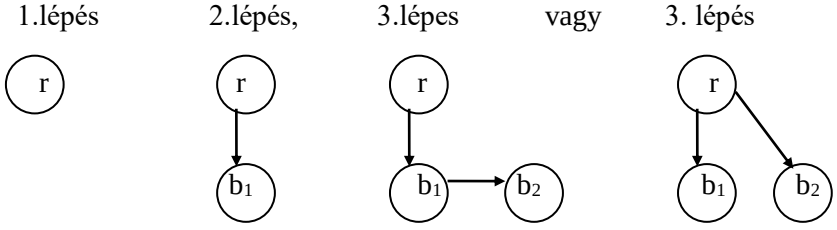


2.13. ábra

Ekkor az új fa gyökere marad r. Ha a csúcs levél volt ($a = l_j$), akkor az új levelek:
$l_1,\ldots, l_{j-1}, b, l_{j+1}, \ldots l_m, b$ különben az új levelek: $l_1, \ldots l_m, b$

Fa tehát az, és csak az ami ezzel a konstrukcióval előállítható.

Példa:

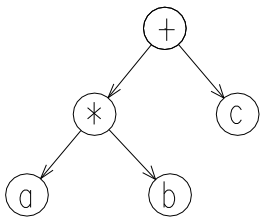


2.14. ábra

Állítás: A gyökértől minden csúcsba pontosan 1 út vezet.

2.5. Bináris fák

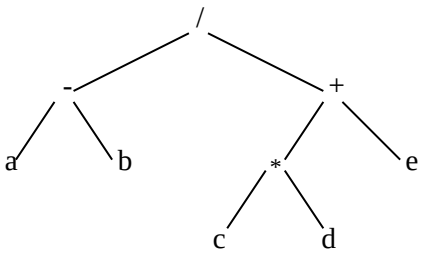
A számítástechnikában fontos szerepet játszanak azok az adatszerkezetek, amelyeknek elemei között a szerkezeti összefüggések irányított fával ábrázolhatók. Nézzük az alábbi egyszerű aritmetikai kifejezést: a * b + c.
Ennek kiszámítási szabályát fával ábrázolhatjuk. A fa irányítottságát mindig a gyökértől a levelek irányában képzeljük el, és a továbbiakban nem ábrázoljuk.



2.15. ábra

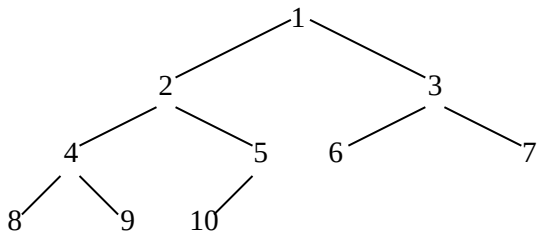
A továbbiakban olyan fákkal foglalkozunk, amelyeknél egy csomópontból legfeljebb két él indul ki; ez a bináris fa. Gyökérelemnek nevezzük azt az elemet, amelybe nem fut él. Bináris fa esetén minden csomópontnak 0,1 vagy 2 rákövetkezője (gyereke) lehet.
Egy fa esetén beszélünk gyökér (szülő) elemről, ami mindig a 0-dik szinten van. Az első szinten van maximum két gyereke, a bal és a jobboldali gyerek. A második szinten már legfeljebb 4 elem található mindkét gyerek bal és jobboldali gyereke. A baloldali gyerek és az ő két gyereke alkotja a gyökér baloldali részfáját, míg a jobboldali gyerek az ő gyerekeivel a jobboldali részfát.

Példa Legyen egy kétoperandusos műveleteket tartalmazó algebrai kifejezés:
E=(a-b)/((c*d)+e) Ábrázolása a 2.16 ábrán látható.



2.16. ábra

Definíció: Teljes bináris fa akkor keletkezik, ha minden csomópontnak megvan feltétlen mind a két gyereke, kivéve az utolsó szintet, ahol baloldalról vannak a gyerekek, de nem feltétlen töltik ki a teljes szintet.
Tehát az utolsó szintet kivéve minden szinten a csomópontok száma maximális, az utolsó szinten baloldalról helyezkednek el a csomópontok.
Példa: Az 2.17-es ábrán egy T_{10} teljes bináris fa látható.

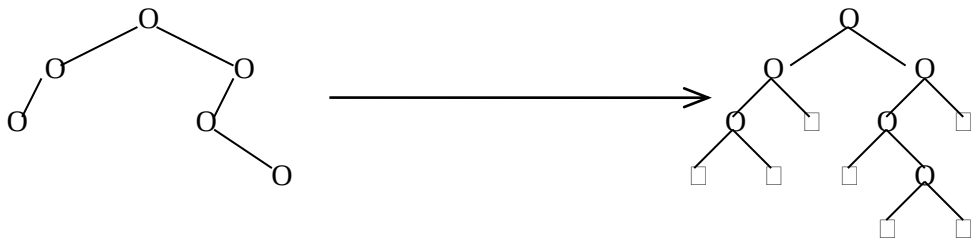


2.17. ábra

Megjegyzés: Az 2.16. ábrán látható fa nem teljes.

Definíció: Kiterjesztett bináris fa (kettes fa): minden csomópontnak 0 vagy 2 gyereke van.
2 gyerekes csomópont: belső csomópont (O)
0 gyerekes csomópont: külső csomópont, vagy levél (□)

1.Példa



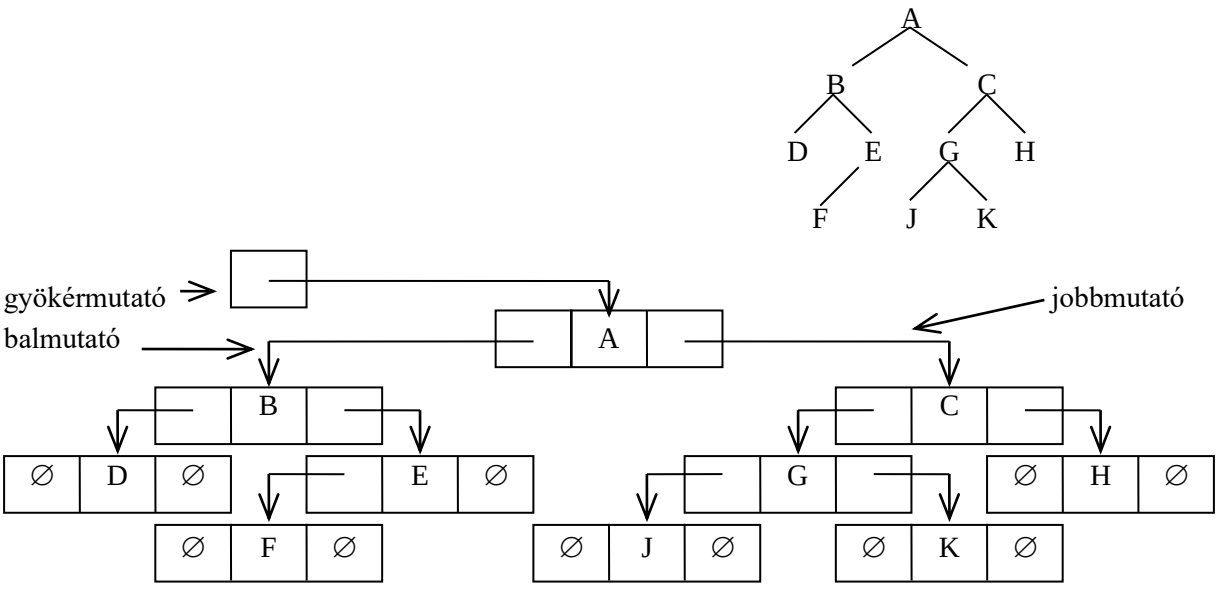
2.18. ábra

2. Példa Két operandusos műveletek szemléltetése:
operandus: külső csomópont
operátor: belső csomópont
Látható az 2.16 –os ábrán a -, /, +, *, operátorok mind belső csomópontok, az a, b, c, d, e, operandusok pedig mind külső csomópontok (levelek).

2.5.1. A bináris fák ábrázolása.

Ahhoz, hogy alkalmazni tudjuk a fa adatszerkezeteket matematikai problémák számítástechnikai feldolgozása során, elsőként ábrázolnunk kell tudni egy adott programnyelv eszközeivel ezt ez adatszerkezetet. Az ábrázolásra kétféle alapvető lehetőség nyílik.

1. módszer: láncolt listákkal történő ábrázolás.
Az ábrázolandó fa a következő:



2.19. ábra

Minden adatelemet (ADAT(N)) két mutatóval egészítünk ki (BAL(N)), JOBB(N)), amelyek megadják az elemtől balra, illetve jobbra elhelyezkedő elem, azaz az adott elem bal és jobb gyerekének címét. A GYÖKÉR nevű változó fogja tartalmazni a gyökérelem címét. Ha egy mutató értéke 0, az azt jelenti, hogy a fa arra nem folytatódik, az adott elemnek nincs bal ill. jobb gyereke. (Ha a gyerekek felől a szülő csomópont felé szeretnénk megkönnyíteni a haladást, használhatunk kétirányú listát is, azaz kiegészíthetjük az elemeket egy harmadik mutatóval, mely a szülőre mutat.)

2. módszer: vektorban történő ábrázolás. (szekvenciális ábrázolás)

Az adott fa elemeit helyezzük el egy vektorban úgy, hogy a gyökérelem legyen a vektor 1. eleme, őt kövesse a bal, majd a jobb gyereke, azaz a fa 1. szintjének elemei sorfolytonosan, majd a 2. szint elemei sorfolytonosan és így tovább. Ha egy adott helyen a fa nem tartalmaz elemet, a vektorba nullelem (0) kerül.

A vektorból könnyen kiolvasható egy adott csomópont gyerekeinek ill. szülőjének indexe az alábbi összefüggés alapján:

A K-dik csomópont bal rákövetkezője: 2*K-dik elem
A K-dik csomópont jobb rákövetkezője: 2*K+1-dik elem
A K-dik csomópont szülője a K/2 alsó egészrészeként számítható elem.

Az előbbi gráf ábrázolása vektorban:

A	B	C	D	E	G	H	∅	∅	F	∅	J	K	∅	∅

Megjegyzés: A szekvenciális ábrázolás hatékony, ha a fa teljes vagy csaknem teljes, hiszen ekkor nullelemek csak a vektor végén helyezkednek el.

2.5.2. A bináris fák bejárása.

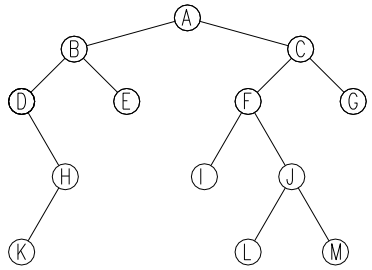
Egyes algoritmusok estén szükség lehet a bináris fák bejárására, azaz az összes csúcs egy adott sorrendben történő végigjárására.
A bináris fa bejárásakor három stratégiát alkalmazhatunk:

Preorder *bejárás*: azaz a gyökér elem, majd a bal oldali részfa preorder bejárása, végül a jobboldali részfa preorder bejárása.

Inorder *bejárás*: azaz először a bal részfa inorder bejárása, majd a gyökérelem, végül a jobboldali részfa inorder bejárása.

Postorder *bejárás*: azaz először a bal részfa postorder bejárása, majd a jobboldali részfa postorder bejárása, végül a gyökérelem feldolgozása.

Példa:



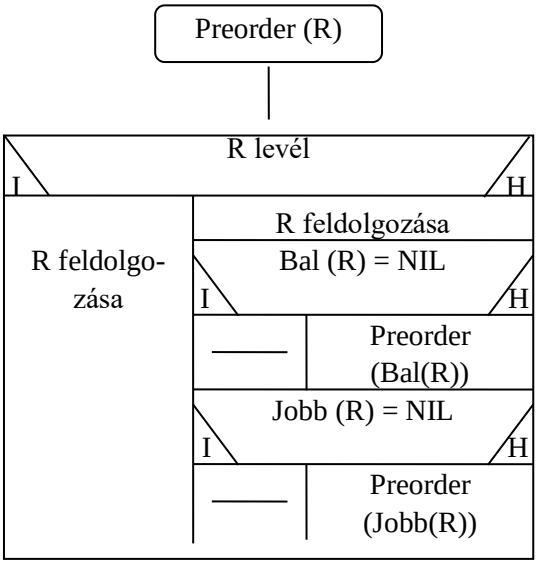
Preorder: ABDHKECFIJLMG
Inorder: DKHBEAIFLJMCG
Postorder: KHDEBILMJFGCA

2.20 ábra

A preorder bejárás rekurzív algoritmusa:
(mindegyik eljárást a Gyökér címmel kell meghívni.)

I. Preorder bejárás (R gyökerű fáé)

1. gyökér feldolgozása
2. R bal oldali részfájának preorder bejárása
3. R jobb oldali részfájának preorder bejárása



2.21 ábra

PREORDER(MUT):

Ha MUT <> 0 akkor Ki: ADAT (MUT)
PREORDER (BAL (MUT))
PREORDER (JOBB (MUT))
Elágazás vége
Eljárás vége

II. Inorder bejárás

1. R bal oldali részfájának inorder bejárása
2. R gyökér feldolgozása
3. R jobb oldali részfájának inorder bejárása

INODER(MUT):

Ha MUT ≠ 0 akkor INORDER (BAL (MUT))
Ki: ADAT (MUT)
INORDER (JOBB (MUT))
Elágazás vége
Eljárás vége

III. Postorder bejárás

1. R bal oldali részfájának postorder bejárása
2. R jobb oldali részfájának postorder bejárása
3. R gyökér feldolgozása

POSTORDER(MUT):

Ha MUT ≠ 0 akkor POSTORDER (BAL (MUT))
POSTORDER (JOBB (MUT))
Ki: ADAT (MUT)
Elágazás vége
Eljárás vége

Ezek az eljárások, bár megadják a különböző módon bejáró algoritmusokat, ***rekurzív*** eljárások, melyek több programnyelvben nehezen valósíthatóak meg. Ezért elkészítjük a nem rekurzív változatukat is.
Felhasználjuk a már megismert veremhasználó algoritmusokat, melyek a második kötetben megtalálhatóak.
(Az eljárásoknál V a verem, X az elemérték, VM az aktuális mutató(az első szabad helyre mutat). Ha üres a verem akkor VM=1)

VEREMBE(V,X):

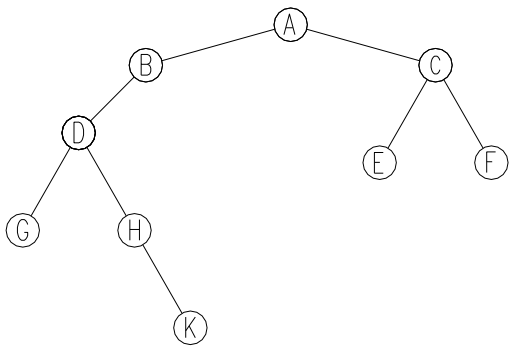
VEREMBŐL(V,X):

A fa bejáró algoritmusok:
A MUT egy mutató ami a feldolgozandó csomópontra mutat. A veremben a még feldolgozandó jobb csomópontokat tároljuk. Ha nincs további bal vagy jobb levél, akkor a megfelelő mutató 0. A V_j végjel jele.

PREORDER(GYÖKÉR):

VEREMBE(V, Vj)
MUT:=GYÖKÉR
Ciklus amíg MUT ≠ Vj
Ki: ADAT(MUT)
Ha JOBB(MUT) ≠0 akkor
VEREMBE(V,JOBB(MUT))
Ha BAL(MUT) ≠0 akkor
MUT:=BAL(MUT)
Különben:
VEREMBŐL(V,MUT)
Ciklus vége
Eljárás vége

Példa:



2.22. ábra

A program képzeletbeli lefuttatása során a verem és a kifejezés tartalmának alakulása a 2.22 ábrán látható fára az alábbiakban látható:(Vj a végjel):

Input	verem tartalma	Mutató	Bejárási sorrend alakulása
	Vj	A	A
	Vj,C	B	A,B
		D	A,B,D
	Vj,C,H	G	A,B,D,G
	Vj,C	H	A,B,D,G,H
	Vj,C,K		
	Vj,C,		A,B,D,G,H,K
		C	A,B,D,G,H,K,C
	Vj,F,		A,B,D,G,H,K,C
		E	A,B,D,G,H,K,C,E
		F	A,B,D,G,H,K,C,E,F
		Vj	eljárás vége

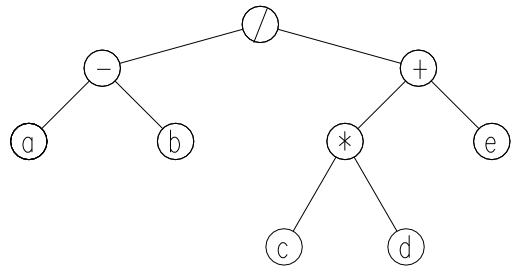
INORDER(GYÖKÉR):

```
VEREMBE(V,Vj)
MUT:=GYÖKÉR
Ciklus amíg MUT ≠ Vj
    Ciklus amíg MUT ≠0
        VEREMBE(V,MUT)
        MUT:=BAL(MUT)
    Ciklus vége
    VEREMBŐL(V,MUT)
    HA MUT≠0 akkor Ki: ADAT(MUT)
    MUT:=JOBB(MUT)
Ciklus vége
Eljárás vége
```

A következő eljárás megírását a hallgatóra bízzuk.

POSTORDER(GYÖKÉR):

Példa 1:. Adjuk meg az alábbi infix kifejezés gráfját és bejárásait: (a – b) / (c*d + e)



A bejárások:
Preorder: /-ab+*cde
Inorder: a-b/c*d+e
Postorder: ab-cd*e+/

2.23. ábra

A három bejárás eredménye a kifejezés megfelelő prefix, infix, illetve postfix alakja.

2.5.3. Aritmetikai kifejezések lengyelformára hozása, és kiértékelése.[4]

Ma már természetes, hogy egy matematikai kifejezés kiértékelését automatikusan meg tudja oldani bármely programnyelv. De vajon hogyan? Alapvető probléma, hogy az általánosan használt infix kifejezésekben, ahol a műveleti jel az operandusok között áll (pl. 1+2*3) a kiértékelési és a felírási sorrend nem azonos. Olyan formára kell tehát hozni a kifejezést, ami egyértelműen kifejezi a végrehajtási sorrendet. A kifejezésnek ezt az átírását a kifejezés ***l e n g y e l f o r m á***jának vagy posztfix kifejezésnek nevezik (Ugyanis egy lengyel matematikus használta először a formulákat ilyen formában). A lengyel forma lényege, hogy a műveleti jelek nem a műveletben szereplő adatok között, hanem mögöttük találhatók, a műveleti jelek sorrendje egyben végrehajtási sorrendjüket is jelenti. (A fenti pl. esetén 123*+)
Vegyük észre, hogy a kifejezés ilyen felírási formájában nincs szükség zárójelezésre (hiszen az a végrehajtási sorrend megváltoztatására szolgált, itt pedig ezt a felírási sorrend egyértelműen eldönti.)!
Belátható, hogy a kifejezést ábrázoló fa postorder bejárása ugyanazt a kifejezést adja, mint a lengyel forma, azaz a posztfix kifejezést, míg inorder bejárása a hagyományos infix kifejezést. Lehet értelmezni egy kifejezés prefix alakját is, mely az őt ábrázoló fa preorder bejárásával kapható meg. (2.23. ábra.)
Kiértékelés szempontjából a posztfix alak a megfelelő forma, csak éppen nem követelhetjük meg senkitől, hogy ilyen formában adja meg a kifejezést a programjában. Tehát meg kell oldani az infix módon felírt kifejezés ilyen formára való hozását, majd a formula kiértékelését.

2.5.3.1. Infix kifejezés átalakítása postfixé.(algoritmizálható forma.)(Lengyelformára hozás)

Olvassuk el az infix kifejezést balról jobbra haladva, és az olvasott érték típusától függően járjunk el a következő módon:

1. Ha számot olvasunk (általánosabban: operandust, mely lehet szám vagy változó), akkor az a postfix kifejezésbe kerül
2. Ha (nyitó zárójelet, az a verembe kerül.
3. Ha operátort (műveleti jelet), akkor az a verembe kerül, de ha az alatta lévő operátor(ok) magasabb prioritású(ak), akkor ki kell venni őt (őket) a veremből, a postfix kifejezésbe áttenni és csak ekkor tehető az új operátor a verembe.
4. Ha) csukó zárójelet, akkor mindaddig vesszük ki a veremből az elemeket és helyezzük a postfix kifejezésbe, amíg meg nem találjuk a csukó zárójelhez tartozó nyitó zárójelet, és azt eldobjuk a csukóval együtt.
5. Ha elfogyott a kifejezés, de van még a veremben elem, akkor ezeket egyenként kivesszük (a verem szabályainak megfelelően) és áttesszük a postfix kifejezésbe.

Példa 1: **(5 * (6 + 2) – 8 / 4)**

Ssz	Input	Műveletek:	Verem tartalma.	Postfix kifejezés:
1.	(	verembe	(	
2.	5	postfixbe	(	5
3.	*	verembe	(*	
4.	(	verembe	(* (	
5.	6	postfixbe		5 6
6.	+	verembe	(* (+	
7.	2	postfixbe		5 6 2
8.	)	veremből + postfixbe	(* (	5 6 2 +
9.		veremből)	(*	nyitó (eldobom
10.	-	verembe	(* -	prioritás miatt nem jó! Tehát ezt nem lépjük meg.
11.	*	veremből * postfixba	(	5 6 2 + *

12.		verembe	(-		
13.	8	postfixbe		5 6 2 + * 8	
14.	/	verembe	(- /		
15.	4	postfixbe		5 6 2 + * 8 4	
16.	)	veremből postfixbe	(-	5 6 2 + * 8 4 /	
17.		veremből postfixbe	(	5 6 2 + * 8 4 / -	
18.		veremből postfixbe			nyitó (eldobom

tehát:***5 * (6 + 2) – 8 / 4 → 5 6 2 + * 8 4 / -***

Példa 2*:** Másik kifejezés: ***3 ^2 / (5 + 3) (Az átalakítás menete rövidebb magyarázattal:)

Input	Verem	Postfix	tevékenység
3		3	Sorból – sorba
^	^	3	Verembe
2	^	3 2	Sorból – sorba
/	/	3 2 ^	Prioritás? Sorba, verembe
(	/ (	3 2 ^	Verembe
5	/ (	32^5	Sorból – sorba
+	/ (+	32^5	Prioritás? Sorba, verembe
3	/ (+	32^53	Sorból - sorba
)	/	32^53+	Veremürítés) ig
		32^53+/	Veremürítés teljesen

A fenti eljárás algoritmusa a következőkben látható.
Felhasználjuk a már ismert verem feldolgozó algoritmusokat. Leírásuk a másik kötetben megtalálhatók.

VEREMBE(V,X)	Az X elhelyezése a V verembe (PUSH(X))
VEREMBŐL(V,X)	A V verem legfelső elemének kivétele X –be.
TETŐ(V)	A verem legfelső elemének értéke
N	infix kifejezés hossza

Az A-beli kifejezés B-beli lengyel formára hozása, V verem felhasználásával.

Lengyel formára hozás

```
Eljárás
        J=1;
         VEREMBE(V,kif.végjel)
        Ciklus I = 1-től N-ig
                X=A(I)
                Elágazás
                        X = adat        esetén B(J):=X
                                                J=J+1
                        X = „(„         esetén VEREMBE(V,X)
                        X = „)„         esetén zárójel feldolgozás
                        X = műv.jel esetén művelet feldolgozás
                Elágazás vége
        Ciklus vége
        Veremürítés
Eljárás vége
```

Műveletfeldolgozás (X-ben van a műveleti jel)

```
Eljárás
    Ciklus amíg prioritás(X) < = prioritás (TETŐ(V))
        VEREMBŐL(V,B(J))
        J=J+1
    Ciklus vége
    Verembe(V,X)
Eljárás vége
```

Veremürítés

```
Eljárás
    Ciklus amíg TETŐ(V) <> kifejezés végjel
        VEREMBŐL(V,B(J))
        J=J+1
    Ciklus vége
Eljárás vége
```

Zárójel feldolgozás (X-ben van a csukó zárójel)

```
Eljárás
    Ciklus amíg TETŐ(V) <> „(„
        VEREMBŐL(V, B(J))
        J=J+1
    Ciklus vége
    VEREMBŐL(V,X)
Eljárás vége
```

2.5.3.2. Postfix kifejezés kiértékelése veremmel.

A következő lépés a kapott posztfix kifejezés kiértékelése. Az algoritmus a következő:
Olvassuk végig a posztfix kifejezést balról jobbra haladva és a kiolvasott érték típusától függően járjunk el az alábbi módon:

1. Ha operandust olvasunk, akkor ezt tegyük be a verembe.
2. Ha műveleti jelet olvasunk, akkor vegyük ki a veremből a két legfelső elemet, végezzük el velük a műveletet, (a sorrend fontos!) és az eredményt tegyük vissza a verembe.

Példa 1: (5 * (6 + 2) – 8 / 4) → 5 6 2 + * 8 4 / -

A kiértékelés folyamata:

	Input	Tevékenység:	Verem tartalma.
1.	5	verembe	5
2.	6	verembe	5 6
3.	2	verembe	5 6 2
4.	+	veremből[1] 2+6=8 verembe	5 8
5.	*	veremből 8*5=40 verembe	40
6.	8	verembe	40 8
7.	4	verembe	40 8 4
8.	/	veremből 8/4=2 verembe	40 2
9.	-	veremből 40-2=38 verembe	38

[1] sorrend: ki op1, ki op2 és op2 művelet op1= eredmény!!

Példa 2: (2 * (5 + 2)) / 2 + 5 * 6 → 2 5 2 + * 2 / 5 6 * +

Input	Tevékenység:	Verem tartalma.
1. 2	verembe	2
2. 5	verembe	2 5
3. 2	verembe	2 5 2
4. +	veremből 2+5=7 verembe	2 7
5. *	veremből 2*7=14 verembe	14
6. 2	verembe	14 2
7. /	veremből 14/2=7 verembe	7
8. 5	verembe	7 5
10. 6	verembe	7 5 6
11. *	veremből 5*6=30 verembe	7 30
12. +	veremből 7+30=37 verembe	37

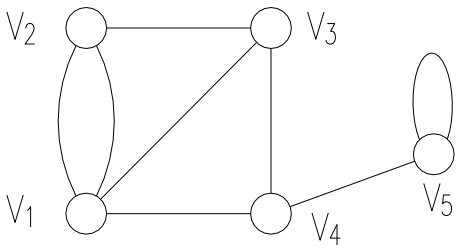
2.6. Irányított gráfok ábrázolása.

Tetszőleges irányított gráf ábrázolása egy adott programnyelvben – hasonlóan a bináris fákhoz – történhet statikusan, a szomszédsági mátrix vagy a szomszédsági tömbök segítségével avagy történhet láncolt listák használatával.

2.6.1.Szomszédsági mátrix [2]

Egy n pontú gráf mátrixszal való reprezentálásának egyik legtermészetesebb módja (feltéve, hogy az esetleges párhuzamos éleket nem kell megkülönböztetni): ha definiálunk egy n x n-es $A(G) = (a_{ij})$ mátrixot az alábbi módon:

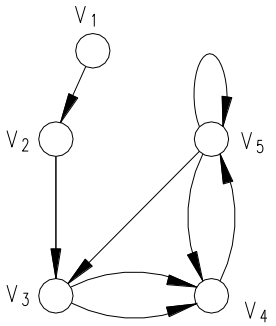
$$a_{ij} = \begin{cases} 0, & \text{ha az } i\text{-edik és } j\text{-edik pont nem szomszédos} \\ k, & \text{ha az } i\text{-edik és } j\text{-edik pont között } k \text{ darab párhuzamos él halad} \\ l, & \text{ha } i = j \text{ és az } i\text{-edik ponthoz } l \text{ darab hurokél illeszkedik.} \end{cases}$$



$$A = \begin{pmatrix} 0 & 2 & 1 & 1 & 0 \\ 2 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

2.24. ábra

Irányított gráfokat is megadhatunk ilyen módon, csak ott a_{ij} az i-edik pontból a j-edik pontba vezető élek száma. Ezt az $A(G)$ mátrixot a G gráf **szomszédsági mátrixának** nevezzük.



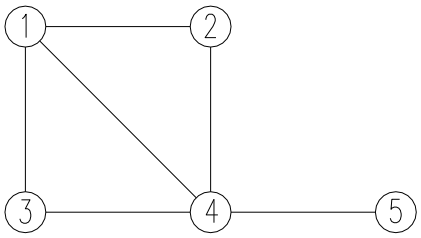
$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

2.25. ábra

Ennek a tárolási módnak előnye az egyszerűsége. Egyes algoritmusok, mint a későbbiekben bemutatásra kerülő útmátrixot kezelő algoritmusok (a szomszédsági mátrix hatványait használó algoritmus ill. a Warshall algoritmus) egyaránt támaszkodnak erre a tárolási módra.
Hátránya, hogy szükségtelenül nagy tárigényt követel, különösen „ritka” gráfok esetén, azaz ha az élek száma lényegesen kevesebb, mint a csúcsok számának négyzete, azaz a mátrix elemszáma.

2.6.2.Szomszédossági tömbök.

Gyakran hasznos a gráfot úgy tárolni, hogy minden pontjához felsoroljuk a szomszédjait. Pl. az 2.26. ábrán látható gráfra ez az ábra melletti táblázatot adná. A mátrix *k.* sora a *k.* csúcs szomszédjait sorolja fel.



2 3 4
1 4
1 4
1 2 3 5
4

2.26. ábra

Mivel ezek a szomszédossági listák általában különböző hosszúságúak, érdemes őket egy nagy közös tömbben tárolni a tárigény csökkentése érdekében, és egy külön tömbben tárolni a „mutatókat” (pointer), hogy honnét kezdve kell olvasni egy adott pont szomszédjait. Így a fenti táblázat az alábbi két tömbbel helyettesíthető:

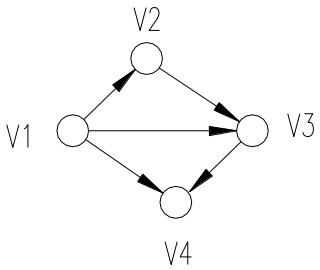
Élek: *|1* *|4* *|6* *|8* *|12* *címek:*
2 3 4 1 4 1 4 1 2 3 5 4 és 1 4 6 8 12

A „címek” vektor *l.* eleme azt az indexet adja meg, ahonnan az élek vektorban az *l.* csúcsa szomszédjainak felsorolása kezdődik.
Ezt a két tömböt nevezzük röviden szomszédossági tömbnek.

Előnye ennek az ábrázolási módnak a tömörsége, hátránya azonban, hogy nehézkes (és nagy műveletigényű) a gráfhoz egy-egy újabb él vagy csomópont hozzávétele vagy elhagyása.

2.6.3. Láncolt listás adatszerkezet.

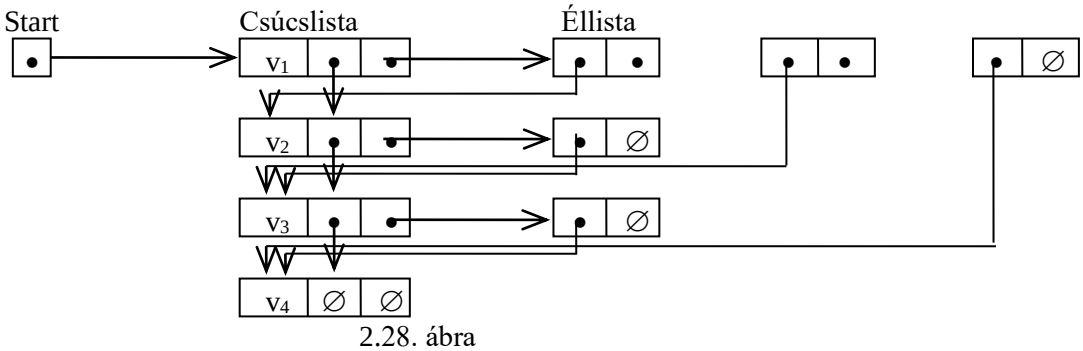
A láncolt listákkal való reprezentáció során a gráf csúcsait felfűzzük egy dinamikus listára (csúcslista). Emellett minden csúcsból indul egy másik lista, az adott csúcsból induló élek listája (éllista). Minden él – elem tartalmaz egy mutatót, mely az él végpontjára mutat.(2.27 – 2.28. ábra.)
Példa



Csúcs	Szomszédsági lista
v_1	v_2, v_3, v_4
v_2	v_3
v_3	v_4
v_4	

2.27. ábra

A kapcsolt adatszerkezet:



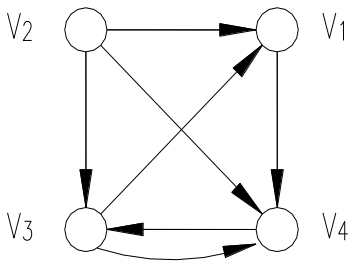
2.28. ábra

2.7. Útmátrix keresés.

Legyenek adottak a G gráf A szomszédsági mátrixának A, A^2, A^3,… hatványai, valamint legyen az $a_k(i,j)$ az A^k mátrixban az *ij*-edik elem!
Figyeljük meg, hogy az $a_2(ij)$ a v_i pontból a v_j pontba vezető 2 hosszúságú utak számát adja. Az ebből levonható általános következtetés:

Tétel: Legyen az A a G gráf szomszédsági mátrixa! Ebben az esetben $a_k(i,j)$, vagyis az A^k mátrix ij-edik eleme v_i-ből a v_j-be vezető *K* hosszúságú utak számát adja.

Legyen adott a 2.29.-es ábrán látható gráf. A gráf szomszédsági mátrixa az A mátrix:.



2.29. ábra $A=\begin{pmatrix}0&0&0&1\\1&0&1&1\\1&0&0&1\\0&0&1&0\end{pmatrix}$

Az A mátrix A^2, A^3, és A^4 hatványai a következők:

$A^2=\begin{pmatrix}0&0&1&0\\1&0&1&2\\0&0&1&1\\1&0&0&1\end{pmatrix}$ $A^3=\begin{pmatrix}1&0&0&1\\1&0&2&2\\1&0&1&1\\0&0&1&1\end{pmatrix}$ $A^4=\begin{pmatrix}0&0&1&1\\2&0&2&3\\1&0&1&2\\1&0&1&1\end{pmatrix}$

Ennek megfelelően tehát a v_4-ből a v_1-be egy út létezik, amely 2 hosszúságú, a v_2-ből a v_3-ba két olyan út létezik, amely 3 hosszúságú, a v_2-ből a v_4-be pedig három olyan út létezik, amely 4 hosszúságú. Definiáljuk most a B_r mátrixot a következőképpen:

$$B_r = A + A^2 + A^3 + \ldots + A^r$$

Ebben az esetben a B_r mátrix ij-edik eleme a v_i-ből a v_j-be vezető *r*, vagy ennél rövidebb utak számát adja. Ha csak olyan utakra vagyunk kíváncsiak, melyek nem tartalmaznak kört, elég az $r = m$ hatványig számolnunk. (ahol *m* a csúcsok száma) Az ennél több élt tartalmazó utak ugyanis mindenképpen tartalmaznak kört.

Definíció: Legyen a G egyszerű irányított gráf, amely a $v_1, v_2, \ldots, v_m$ pontokat tartalmazza.

P=(p_{ij}) mátrix a *G* irányított gráf ***útmátrix***a vagy *elérhetőségi mátrix*a:

p_{ij}=1 minden olyan esetben, amikor v_i-ből vezet út v_j-be. (akárhány élek sorozatán) és p_{ij}=0 különben.

A P útmátrix és az A szomszédsági mátrix között az alábbi összefüggés áll fenn:

A P = (p_{ij}) útmátrixban $p_{ij} \neq 1$ akkor és csak akkor, ha $B_r = A + A^2 + A^3 + \ldots + A^r$ mátrix *(i, j)*-ik eleme $\neq 0$, ahol A a G gráf szomszédsági mátrixa, *n* a csúcsok száma.

Az 2.29-es ábrán látható gráf $n = 4$ pontot tartalmazó G gráf. Az A, A^2, A^3, A^4 mátrix összeadása után az alábbi B_4 mátrixot kaptuk:

$$B_4 = \begin{pmatrix}1&0&2&3\\5&0&6&8\\3&0&3&5\\2&0&3&3\end{pmatrix}$$

Ha ebben a mátrixban 1-re cseréljük a nem zérus elemeket, akkor a gráf P útmátrixát kapjuk:

$$P = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

A P mátrixot megvizsgálva látható, hogy a v_2 pontot egyetlen más pontból sem érhetjük el.

Ezzel az algoritmussal az útmátrix megkeresése $O(n^4)$ műveletigényű, hiszen egy mátrix – hatvány kiszámításának műveletigénye $O(n^3)$, és itt m hatványra van szükségünk.

Az útmátrix megkeresésére ennél léteznek nagyságrenddel hatékonyabb eljárások is, mint például a Warshall algoritmus.

2.8. A Warshall algoritmus

2.8.1. Warshall algoritmus az útmátrix megkeresésére.

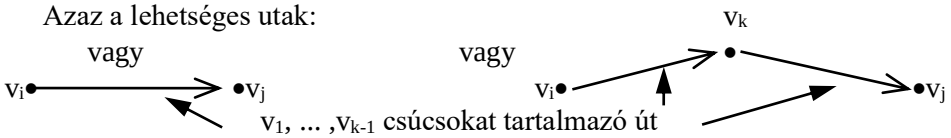
Cél: G mátrix esetén P útmátrix meghatározása.

Jelölés: $P_k(i,j):=1$, ha létezik olyan $v_i \rightarrow v_j$ út, amelyik csak a $v_1, v_2, \ldots, v_k$ közbeeső csúcsokat tartalmazza, és $P_k(i,j):=0$ különben.

Megjegyzés: $P_0(i,j)=1$, ha létezik $v_i \rightarrow v_j$ él, azaz $P_0=A$ szomszédsági mátrix. Továbbá $P_m=P$, a keresett útmátrix.

Állítás: $P_k(i,j) = 1$, ha
- vagy $P_{k-1}(i,j) = 1$
- vagy $P_{k-1}(i,k) = 1$ és $P_{k-1}(k,j) = 1$

Azaz a lehetséges utak:



Tehát $P_k(i,j) = P_{k-1}(i,j)$ vagy ($P_{k-1}(i,k)$ és $P_{k-1}(k,j)$), ahol:

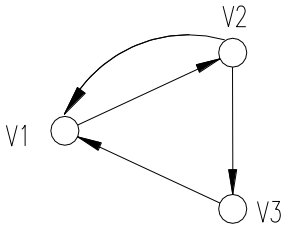
és	0	1
0	0	0
1	0	1

vagy	0	1
0	0	1
1	1	1

Az algoritmus:

1. $P_0:=A$
2. FOR K=1 TO M
    3. FOR I=1 TO M
    4. FOR J=1 TO M
    $P_k(i,j) := P_{k-1}(i,j)$ vagy ($P_{k-1}(i,k)$ és $P_{k-1}(k,j)$)

Példa



2.30. ábra

$$P_0=\begin{pmatrix}0&1&0\\1&0&1\\1&0&0\end{pmatrix} \qquad P_1=\begin{pmatrix}0&1&0\\1&1&1\\1&1&0\end{pmatrix} \qquad P_2=\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}$$

2.8.2. Warshall algoritmus a legrövidebb út megkeresésére.

(Ez az algoritmus egy módosított Warshall-algoritmus)
G súlyozott irányított gráf, $W:=(w_{ij})$ súlymátrix [$w_{ij} = 0$, ha nem létezik $v_i \rightarrow v_j$ él]

Cél: $Q=(q_{ij})$ mátrix megkeresése, ahol q(i,j) az $v_i \rightarrow v_j$ legrövidebb út hossza.

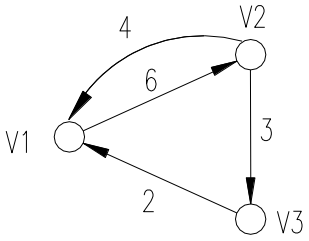
Jelölés: $Q_k(i,j) := \min(Q_{k-1}(i,j));Q_{k-1}(i,k)+Q_{k-1}(k,j))$

A csak $v_1,...,v_{k-1}$ közbülső csúcsokat felhasználó legrövidebb út hossza

Az algoritmus:
1. $Q_0(i,j):=W(i,j)$, ha $W(i,j) \neq 0$ és $Q_0(i,j):= \infty$ különben
2. FOR K:=1 to M
3. FOR I:=1 to M
4. FOR J:=1 to M
$Q_k(i,j):= \min(Q_{k-1}(i,j); Q_{k-1}(i,k)+Q_{k-1}(k,j))$

ahol $Q_k(i,j)$ a csak $v_1,...,v_{k-1}$ közbülső csúcsokat felhasználó legrövidebb út hossza

Példa.



2.31. ábra

$$W=\begin{pmatrix}0&6&0\\4&0&3\\2&0&0\end{pmatrix}$$

$$Q_0 = \begin{pmatrix} \infty & 6 & \infty \\ 4 & \infty & 3 \\ 2 & \infty & \infty \end{pmatrix} \quad Q_1 = \begin{pmatrix} \infty & 6 & \infty \\ 4 & 10 & 3 \\ 2 & 8 & \infty \end{pmatrix} \quad Q_2 = \begin{pmatrix} 10 & 6 & 9 \\ 4 & 10 & 3 \\ 2 & 8 & 11 \end{pmatrix} \quad Q_3 = \begin{pmatrix} 10 & 6 & 9 \\ 4 & 10 & 3 \\ 2 & 8 & 11 \end{pmatrix}$$

v_1 v_1, v_2 v_1, v_2, v_3

Ahol a Q_j mátrix a v_j csomópontot veszi figyelembe.

Megjegyzés: A legrövidebb utak megkeresésére egyéb algoritmusok is ismeretesek megtalálhatóak a [6], [3] irodalmakban.

2.9. Szélességi és mélységi bejárások. [2]

2.9.1. A szélességi és mélységi bejárások megvalósítása sor és verem adatszerkezettel.

Keresztirányú bejárás:

Cél: Egy tetszőleges gráf összes csomópontjának végigjárása (bejárása) keresztirányú sorrendben.

Keresztirányú sorrend:

1. A kiindulópontot járjuk be.
2. A kiindulópont összes szomszédjait járjuk be.
3. A szomszédok szomszédjait járjuk be, stb.

<u>*Megvalósítás:*</u> sor adatszerkezet segítségével.

Az algoritmus során jelöljük, hogy egy csomópontot már bejártunk-e, avagy sem. Ennek megfelelően egy csomópont státusza (ST) lehet az algoritmus során:

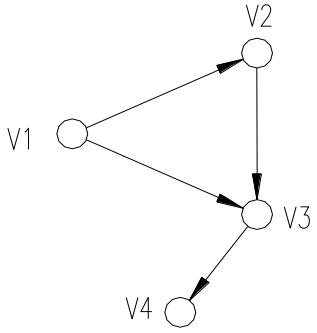
1. készenléti állapot (még nem vizsgált)
2. várakozó állapot (a sorban van)
3 feldolgozott állapot

Az algoritmus:

1. Minden pont státusza legyen 1.
2. A kiindulópont (A) elhelyezése a sorban ST(A):=2.
3. Ismétlés, amíg a sor üres nem lesz:
4. A sor első elemének (N) eltávolítása, N feldolgozása, ST(N):=3.
5. N készenléti állapotú szomszédjainak hozzácsatolása a sorhoz, státuszuk 2-re változtatása.

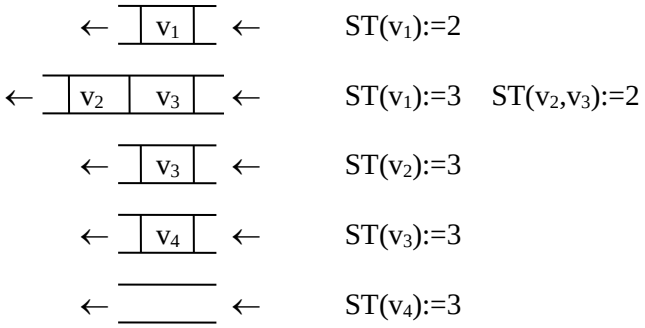
Megjegyzés: Ha maradnak feldolgozandó pontok, melyek A-ból nem érhetők el, egy ilyen ponttal újra kell kezdeni az algoritmust.

Példa



2.32 ábra

A sor alakulása: ST(v_1, v_2, v_3):=



Hosszirányú bejárás:

Cél: Egy tetszőleges gráf összes csomópontjának végigjárása (bejárása) hosszirányú sorrendben.

Hosszirányú sorrend:

1. A kiindulópont feldolgozása.
2. Egy A-ból kiinduló P út teljes feldolgozása.
3. Visszalépés P-n, az első elágazástól ismét egy teljes út végigjárása

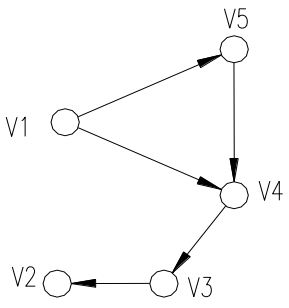
(Hasonlít a bináris fa inorder bejárásához.)

Megvalósítás: verem

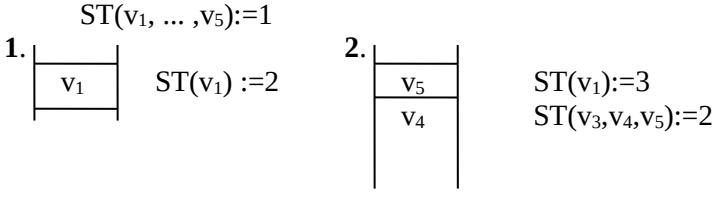
Az algoritmus:

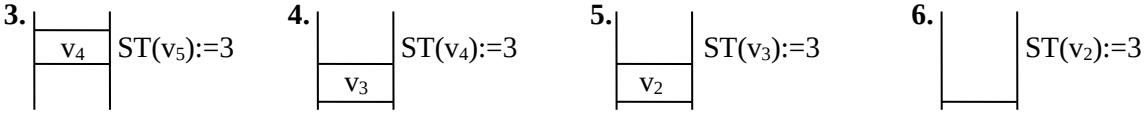
1. Minden csomópont státusza legyen 1.
2. A kiindulópont (A) ráhelyezése a veremre, ST(A) :=2.
3. Ismétlés, amíg a verem ki nem ürül.
4. A verem felső elemének (N) leemelése, N feldolgozása, ST(N):=3.
5. N készenléti állapotú szomszédjainak ráhelyezése a veremre, státuszuk legyen 2.

Példa



2.33. ábra



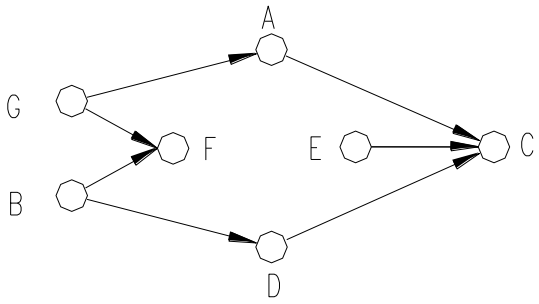


2.10. Gráfok topológiai rendezése

Tegyük fel hogy. G gráf nem tartalmaz kört

Definíció.: G gráf topológiai rendezése a G csúcsainak olyan lineáris sorbarendezése, melyre, ha létezik v→u út G-ben, akkor v megelőzi u-t a sorban.

Példa:



2.34. ábra

Két topológiai rendezése a gráfnak:
B D G A F E C

vagy

E G B A D F C

Tétel: Legyen G véges irányított gráf, mely nem tartalmaz köröket. Ekkor G-nek létezik topológiai sorba rendezése.

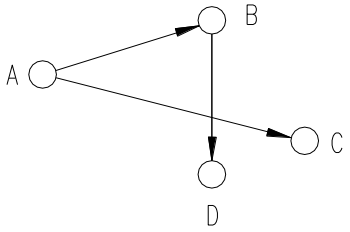
Cél: G topológiai rendezésének megvalósítása algoritmussal.

<u>*Megvalósítás:*</u> sor adatszerkezet segítségével.

Az algoritmus:

1. A G gráf minden pontjának be-fokának meghatározása.
2. A zérus be-fokú pontok elhelyezése a sorban.
3. Ismétlés, míg a sor üres nem lesz.
4. Sor első elemének (N) eltávolítása, és a topológiai sorba helyezése.
5. Az N pont minden M szomszédjára: BE(M):=BE(M)-1
   Ha BE(M)=0, akkor M hozzácsatolása a sorhoz.

Példa:



2.35. ábra.

A sor alakulása:
BE(A):=0 BE(B):=1
BE(C):=1 BE(D):=1

	1.,	← [A] ←		
A	2.,	← [ ] ←	BE(B):=1-1=0	BE(C):=1-1=0
	3.,	← [B C] ←		
AB	4.,	← [C D] ←	BE(D):=1-1=0	
ABC	5.,	← [D] ←	BE(D):=1-1=0	
ABCD	6.,	← [ ] ←		

↑
a topológiai rendezés

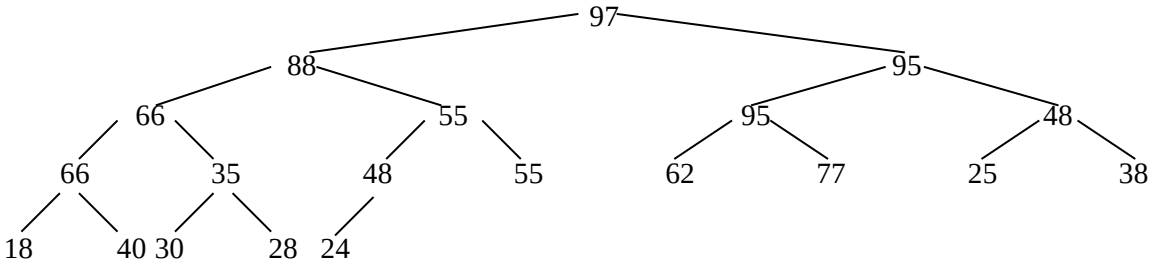
3. Rendezések

Az egyszerűbb rendezések leírása a programozási tételek keretén belül a gyakorlati anyagban megtalálható, a továbbiakban az összetett adatszerkezeteket használó rendezéseket mutatjuk be.

3.1.Halom, halomrendezés

Definíció: Halom(piramis) az olyan teljes bináris fa, melynek minden csomópontjára igaz, hogy a csomópont értéke nagyobb vagy nagyobb egyenlő, mint a csomópont gyermekeinek értéke.

Példa: *Piramis vagy halom*



Megjegyzés: Szekvenciálisan, vektorban tároljuk.

A halomrendezés lépései:

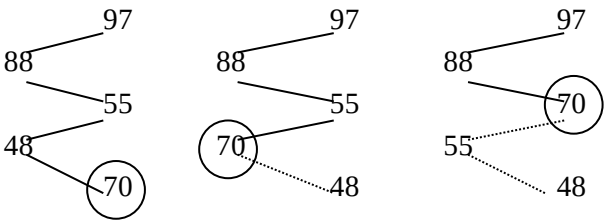
Célunk: „A" tömb elemeinek rendezése.
1. lépés: H halom felépítése A elemeiből, az elemek egymás utáni beszúrásával a készülő halomba.
2. *lépés*: H gyökerének (az aktuálisan legnagyobb elemnek) ismételt törlése. (Más vektorba, vagy A vektor végére)

Beszúrás halomba

A halomrendezés első lépéseként a rendezetlen sorozatból felépítünk egy halmot, méghozzá elemenként két lépésben:

1.lépés: Új elemet a halom végéhez csatoljuk, hogy a teljes fa szerkezet megmaradjon (a halomszerkezet nem feltétlenül marad meg).
2.lépés: Visszaállítjuk a halomszerkezetet.

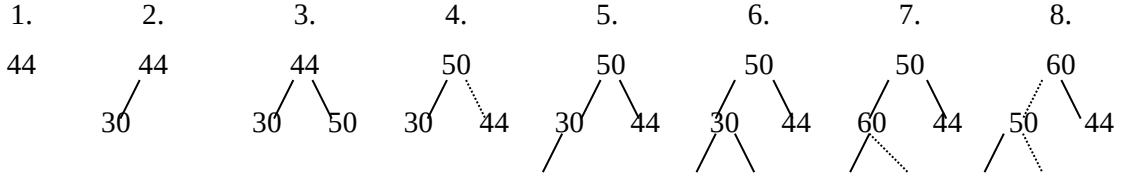
Példa Az előbbi halomba beszúrjuk a 70-es elemet.



1. - halom végére tétel
2., 3. - beemelés a halom megfelelő helyére

Megjegyzés: Sorozatból halmot beszúrások sorozatával építhetünk.

Példa: Legyen a rendezetlen sorozat: 44,30,50,22,60. A halom felépítésének lépései:



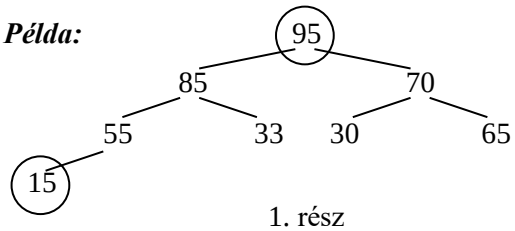
22 22 60 22 30 22 30

Halom gyökerének törlése

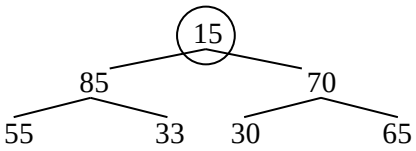
A halomrendezés második lépéseként a felépített halom gyökerét „töröljük”, azaz kivesszük a halomból és elhelyezzük a rendezett sorozatban, hiszen tudjuk (a halomszerkezet tulajdonságai alapján), hogy a gyökérelem a legnagyobb elem a halomban. Ezek után a halom utolsó elemét a gyökér helyére tesszük (azért, hogy a teljes bináris fa szerkezet megmaradjon), majd visszaállítjuk a halomszerkezetet.
Tehát minden egyes elem esetén a következő három lépést hajtjuk végre:

1. lépés: R gyökeret töröljük (értékül adjuk egy változónak, azaz elhelyezzük a rendezett sorozatban)
2. lépés: a gyökér helyére tesszük az utolsó elemet (megmarad a teljes fa szerkezet, a halom nem feltétlenül)
3. lépés: az új gyökeret a megfelelő helyre illesztjük a fában, olyan módon, hogy a nagyobbik gyermekével kicseréljük addig, míg nagyobb vagy egyenlő nem lesz mindkét gyermekénél.

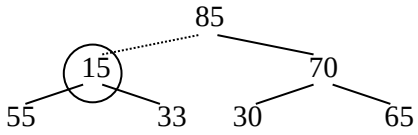
Példa:



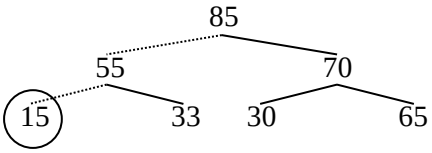
1. rész



2. rész



3.rész

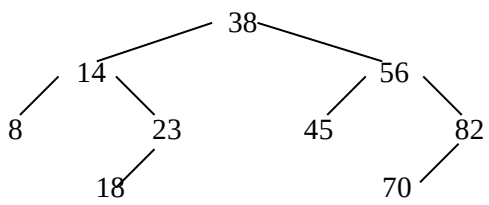


4.rész

3.2. Bináris rendezőfák

Definíció: Bináris keresési (vagy rendező-) fa az olyan bináris fa, melynek minden csomópontjára igaz, hogy a csomópont értéke nagyobb, mint a csomópont bal gyermekének értéke és kisebb vagy egyenlő, mint a csomópont jobb gyermekének értéke.

Példa: *Rendezőfa*



Megjegyzés: A fa inorder bejárása az elemeket növekvő sorrendben adja vissza.

Megjegyzés: Dinamikusan, láncolt listában tároljuk.

A rendezőfával történő rendezés lépései:

Célunk: „A” tömb elemeinek rendezése.
1. lépés: R rendezőfa felépítése A elemeiből, az elemek egymás utáni beszúrásával a készülő rendezőfába.
2. lépés: A fa inorder bejárása.

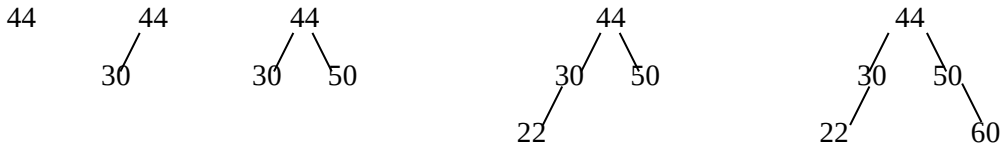
Beszúrás rendezőfába

A beszúrandó elemet (E) összehasonlítjuk a már felépített fa gyökérelemével (R).
- Ha nagyobb, akkor a jobb oldali részfa irányában haladunk tovább és a jobb gyermekkel hasonlítjuk össze a beszúrandó elemet.
- Ha kisebb, akkor a bal oldali részfa irányában haladunk tovább és a bal gyermekkel hasonlítjuk össze a beszúrandó elemet.

A fenti eljárást folytatjuk, míg levélhez nem jutunk, itt elvégezzük a beszúrást.
Azaz az algoritmus:

<table>
<tr><td colspan="3">R<>E és R<> null</td></tr>
<tr><td></td><td colspan="2">↑ R>E ↓</td></tr>
<tr><td></td><td>R:=bal(R)</td><td>R:=jobb(R)</td></tr>
<tr><td colspan="3">↑ R=E ↓</td></tr>
<tr><td colspan="2">Az elem már létezik</td><td>Beszúrás</td></tr>
</table>

Példa: Legyen a rendezetlen sorozat: 44,30,50,22,60. A rendezőfa felépítésének lépései:



A fenti fát inorder módon bejárva megkapjuk a növekvő sorozatot.

3.3. Gyorsrendezés (Quick sort)

Oszd meg és oldd meg típusú rendezés.
Célunk: „A” tömb elemeinek rendezése.
1. lépés: Az első elem helyének megkeresése a rendezetlen sorozatban úgy, hogy az elem előtt csak nála kisebb (nem feltétlenül rendezett sorrendű) elemek legyenek, az elem mögött csak nála nagyobbak. (több lépéses folyamat)
2. *lépés*: A sorozat részekre bontása: a megtalált helyű elem előtti ill mögötti rendezetlen részsorozatok rendezése az előbbi módon (rekurzió).

Példa: Legyen a rendezetlen sorozat: 44,30,50,22,60, 57. A rendezés lépései:

Az első elem (44) helyének megkeresése:
1. *lépés:* Hátulról előre haladunk a sorozatban és keressük az első 44-nél kisebb elemet, ha megtaláltuk, ezzel 44 helyet cserél:

44, 30 , 50 , 22 , 60 , 57
←

22 , 30 , 50 , 44 , 60 , 57

2. *lépés:* Elölről hátra haladunk a sorozatban és keressük az első 44-nél nagyobb elemet, ha megtaláltuk, ezzel 44 helyet cserél:

22 , 30 , ***50*** , 44 , 60, 57
→
22 , 30 , 44 , 50 , 60 , 57

Ezzel az első elem (44) helyét megtaláltuk.

A rendezetlen sorozatunk két még rendezetlen részsorozatot eredményezett:

22 , 30 | 44 | ***50 , 60 , 57***

Ezeket külön-külön a fenti rendezés megismétlésével rendezzük. Jelen esetben csak az alábbi sorozatot kell rendezni:

50 , 60 , 57

A rendezés lépései:

50, 60 , 57
←
50 | 60 , 57

50 | 60 , ***57***
←
50 | 57 , 60

3.4. Összefésüléses rendezés

Célunk: „A” tömb elemeinek rendezése.
1. lépés: Rendezett elempárok képzése a rendezetlen tömbből.
2. *lépés*: A rendezett elempárok összefésülése (összefuttatása) rendezett négyesekké.
….
k. *lépés*: A rendezett 2^{k-1} elemű tömbök összefésülése (összefuttatása) rendezett 2^k elemű tömbökké.

Példa: Legyen a rendezetlen sorozat: 44,30,50,22,60, 57. A rendezés lépései:

44 | 30 | 50 | 22 | 60 | 57
∪ ∪ ∪
30, 44 | 22, 50 | 57, 60
∪
22, 30, 44, 50 | 57, 60
∪
22, 30, 44, 50 , 57, 60

3.5.A rendező algoritmusok bonyolultságának összehasonlítása

A rendező algoritmusok megtalálhatóak ebben a jegyzetben, a *-gal jelöltek pedig az ehhez tartozó programozási feladatok kötetben.

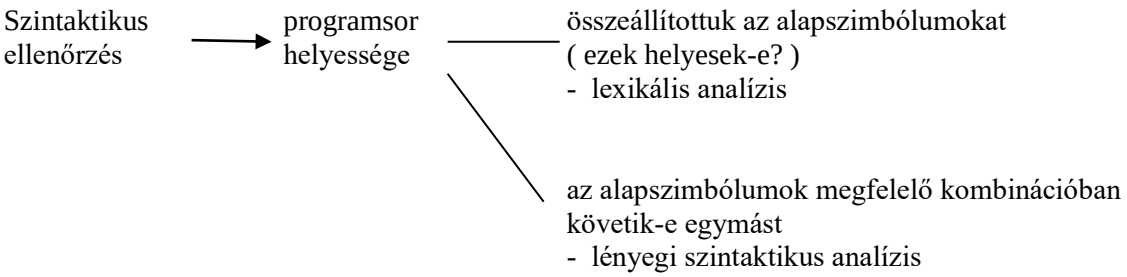
Algoritmus	Legrosszabb eset	Átlagos eset	+memóriaigény
Rendezés közvetlen kiválasztással*	$O(n^2)$	$O(n^2)$	
Rendezés minimum kiválasztással*	$O(n^2)$	$O(n^2)$	
Egyszerű beillesztéses rendezés.*	$O(n^2)$	$O(n^2)$	
Buborékrendezés*	$O(n^2)$	$O(n^2)$	-
Halomrendezés	$O(n \log_2 n)$	$O(n \log_2 n)$	-
Rendezés rendezőfával	$O(n^2)$	$O(n \log_2 n)$	-
Gyorsrendezés	$O(n^2)$	$O(n \log_2 n)$	-
Összefésüléses rendezés	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n)$

4. AUTOMATÁK ÉS FORMÁLIS NYELVEK

4.1. Programozási nyelvek szintaktikája

Amikor egy adott program szintaktikus ellenőrzését végezzük, két dolgot kell megvizsgálnunk:
Először is fel kell ismernünk a programozási nyelvben használható alapszavakat, szimbólumokat (if, then, else, begin, end, stb.), majd meg kell vizsgálnunk, hogy ezek, valamint a programban szereplő egyéb, szavak, karakterek, kifejezések helyes sorrendben követik-e egymást, abban az értelemben, hogy megfelelnek-e az adott programnyelv szabályainak. Az első vizsgálatot lexikális analízisnek, míg a másodikat lényegi szintaktikus ellenőrzésnek nevezzük.
(A lexikális analízist többnyire az adott programnyelvhez tartozó szövegszerkesztők azonnal elvégzik.)

Azaz:



A lényegi szintaktikai analízishez tehát szükséges azon szabályok megadása, mellyel szabályos programokat lehet létrehozni, maga az analízis pedig annak vizsgálata, hogy az általunk létrehozott program megfelel-e ezen szabályoknak.

A szabályok formális leírására úgynevezett metanyelvi módszert dolgoztak ki.

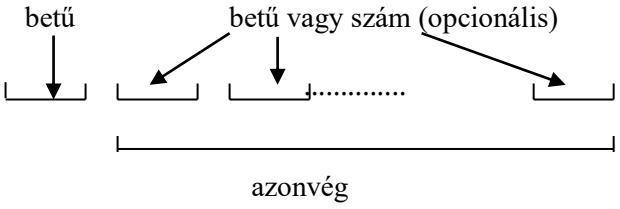
A metanyelvek egyik legkorábbi változata az ún. Backus-féle normálforma, vagy Backus-Naur forma (BNF). Az első magasszintű nyelvek egyikének, az ALGOL-nak is létezik leírása Backus-féle normálformában.

Amikor egy programnyelv leírását (szabályait, grammatikáját) szeretnénk BNF-ben megadni, valójában véges sok szabályt kell megadnunk. Ezeket a szabályokat a későbbiekben nyelvtani szabályoknak fogjuk nevezni. Minden szabály egy-egy metanyelvi egyenlőség, melynek két oldalát a ::= szimbólum választja el. A bal oldalon egy szöveg áll metazárójelek közé zárva. Ez a szöveg többnyire a nyelv valamely szintaktikus egységének neve. A későbbiekben az ilyen egységeket az adott nyelv nemterminális- vagy nyelvtani jeleinek fogjuk nevezni. A metanyelvi egyenlőség jobb oldalán szintén hasonló nyelvtani jelek vagy a programnyelv alapegységei, ill. ezek véges sorozata állhat, vagy több ilyen sorozat függőleges vonalakkal elválasztva. A függőleges vonal a „vagy” jele, azt jelenti, hogy a felsorolt sorozatok (később ezeket szavaknak nevezzük majd) mindegyike szóba jöhet az egyenlőségnél, mint lehetőség:

A metanyelvi szimbólum	Jelentése	Használata
::=	„Definíció szerint egyenlő”	A definiálandó fogalmat választja el a meghatározástól.
<szöveg>	Nemterminális vagy nyelvtani jel. A zárójelben levő szöveg egy szintaktikai egység neve.	Azt jelöli, hogy a két metazárójel közé zárt sorozatot egységes jelként kell kezelni.
|	„Vagy”	A definiálandó szintaktikai egység lehetséges meghatározásait választja el egymástól.

Továbbá a szabályok leírásának részét képezik a tényleges jelek (terminális jelek), például a programozásban szereplő alapszimbólumok, betűk, számok.

Példa 1.:
Egy olyan azonosító leírása, amely véges sok számjegyből vagy betűből állhat, de az első karaktere mindenképpen betű:



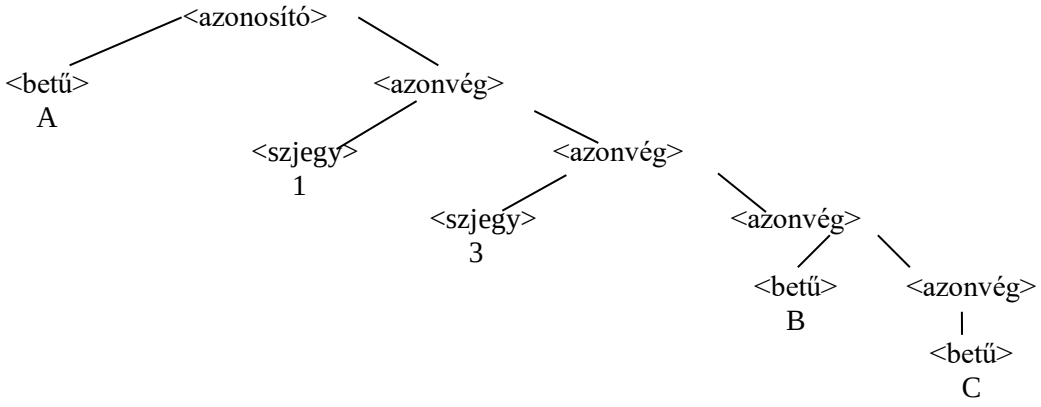
<azonosító>::=<betű> | <betű><azonvég>

nemterminális jel

<azonvég>::=<betű> | <számjegy> | <betű><azonvég> | <számjegy><azonvég>
<betű>::= A | B | C | ... | z ← terminális jel
<számjegy>::= 0 | 1 | 2 | 3 | ... | 9

A fenti szabályokkal definiáltuk az azonosító fogalmát a fent megfogalmazott elképzeléseink szerint. A szintaktikus elemzés során a szabályok ismeretében egy fordítóprogramnak ill. szintaktikus elemzőnek célja eldönteni, hogy egy adott konkrét karaktersorozat helyes azonosító-e a definíció szerint.

Tekintsük példaként az alábbi karaktersorozatot: **A13BC**
Cél: Eldönteni, hogy helyes azonosító-e.
Felépítjük a karaktersorozathoz tartozó „szintaxisfát”, melyet úgy kapunk, hogy a gyökér helyére az <azonosító> metanyelvi szimbólumot helyezzük, majd minden további szintet úgy képzünk, hogy ha egy adott csúcs szerepelt valamely szabály bal oldalán, akkor közvetlen leszármazottai balról jobbra haladva ezen szabály jobb oldalán található valamely szó elemei.
Jelen példánkban:

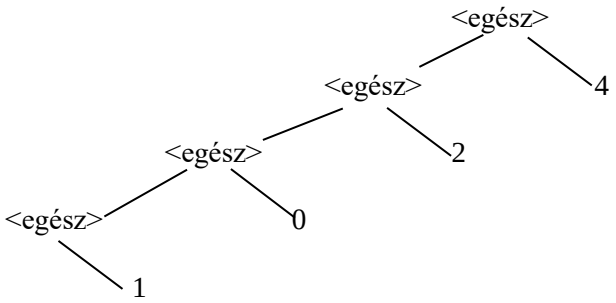


4.1. ábra

Ha tudunk olyan fát építeni, melyben a leveleket balról jobbra haladva összeolvassuk és így a keresett azonosítót kapjuk, akkor ez az azonosító megfelel a szintaxisnak. Ezt úgy is mondjuk, hogy ennek az azonosítónak létezik szintaxisfája.
Általában azt mondhatjuk, hogy egy azonosító akkor helyes, akkor felel meg a szabályoknak, ha létezik szintaxisfája, vagy más kifejezéssel élve: levezethető.

Példa 2.: pozitív egész számok definiálása:
<egész>::= 1 | 2 | 3 | ... | 9 | <egész> 0 | <egész>1 | ..... | <egész>9

Példa: 1024 levezetése:



4.2. ábra

Példa 3.: Egyszerű kifejezés leírása:

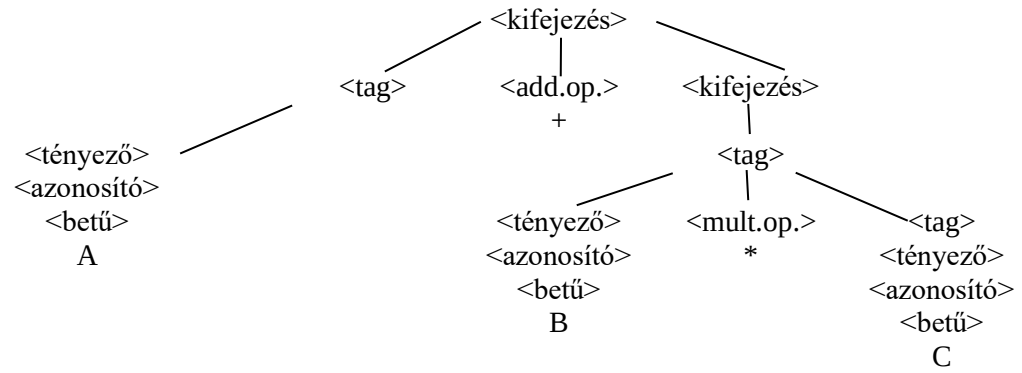
A + B * C / (+3) - D * E + A * D / (-5.3)

tag tag tényező konstans
additív operátor
multiplikatív operátor

<kifejezés>::=<tag> | <tag><add.op.><kifejezés>
<tag>::=<tényező> | <tényező><mult.op.><tag>
<tényező>::=<azonosító> | <konstans>
<add.op.>::= + | -
<mult.op.>::= * | /
<konstans>: :=<szám> | <előjel><szám>
<előjel>::= + | -
<szám>::=<szjegy> | <szjegy><szám> | <szjegy> • <tizedes>

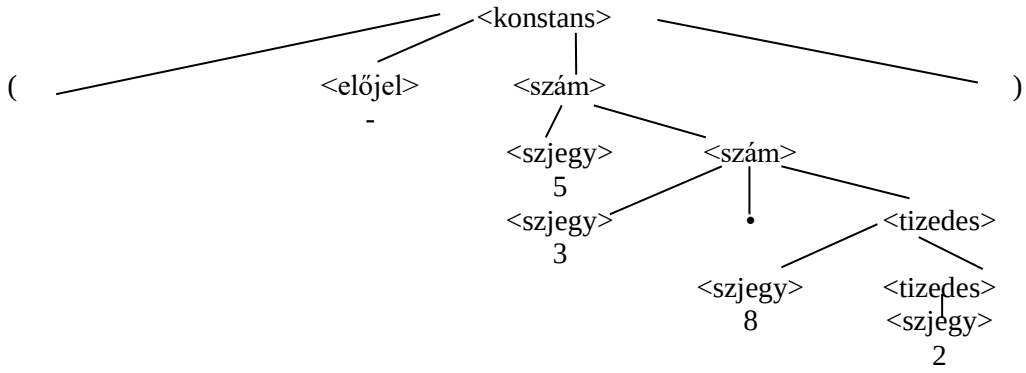
<tizedes>::=<szjegy> | <szjegy><tizedes>

Példa: A + B * C Cél: eldönteni, hogy helyes kifejezés-e?
Felépítjük a szintaxisfáját:



4.3. ábra

Példa (-53.82) Cél: Eldönteni, helyes konstans-e?
A szintaxisfa:



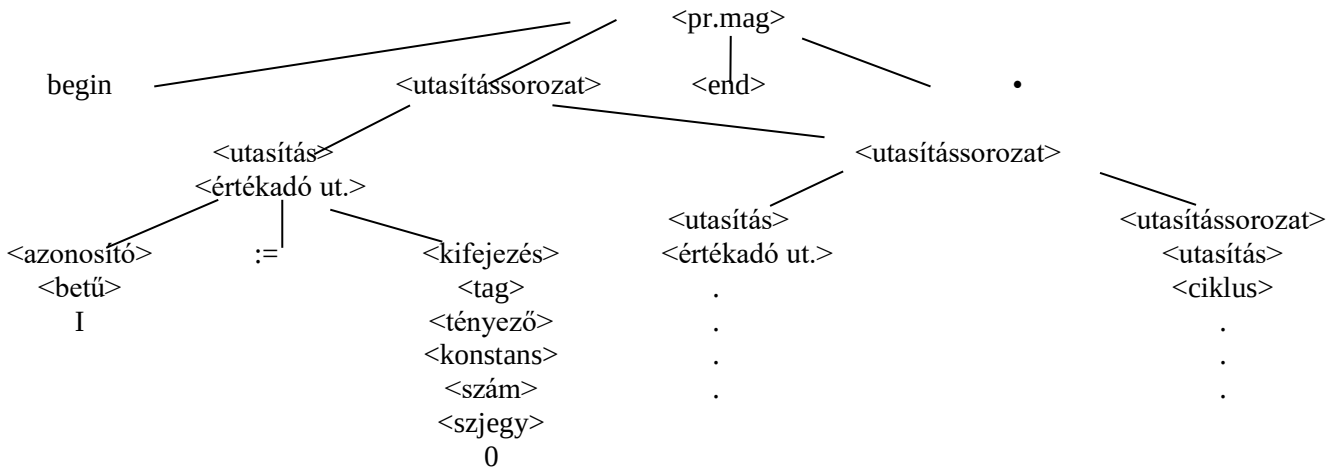
4.4. ábra

Példa 4.: A továbbiakban egy egyszerű, strukturált, eljárás nélküli Pascal pr. Magját írjuk le szabályokkal, a következő megszorításokat alkalmazva: programnév, deklaráció nincs, csak értékadó utasítások vannak, elágazásban csak begin...end szerkezet engedhető meg.

<pr.mag>::=begin<utasítássorozat>end •
<utasítássorozat>::=<utasítás> | <utasítás><utasítássorozat>
<utasítás>::=<blokk> | <elág.> | <ciklus> | <értékadó utasítás>
<blokk>::=begin < utasítássorozat > end **;**
<blokk1>::=begin < utasítássorozat >end
<elág.>::=if <log.kif.> then <blokk1> else <blokk> | if <log.kif.> then <blokk>
<ciklus>::= while <log.kif.> do <utasítás>
<értékadó utasítás >::= <azonosító> := <kifejezés> **;**
<log.kif.>::=<kifejezés><reláció><kifejezés> | <log.kif.><log.op.><logif.> | ¬ (<log.kif.>)
<reláció>::= < | > | <= | >= | = | <>
<log.op.>::= ∧ | ∨

Példa: begin
```
    I:=0;
    S:=0;
    while I <= N do
        begin
            S:=S + I;
            I:=I + 1;
        end;
end.
```

A szintaxisfa kezdő részlete:



4.5. ábra

4.2. Formális nyelvek - véges automaták

Az előző fejezetben találkoztunk néhány fogalommal: szabályok, nyelvtan, terminális– és nemterminális jelek, szintaxisfa, levezetés…

A további fejezetek célja pontosítani és általánosítani ezeket a fogalmakat és módszereket adni a levezetések megkonstruálására.

Definíció: Ábécé: jelek, szimbólumok nem üres halmaza.
Példa:

$$T_1 : \{a, b, c\}$$

$$T_2 : \{if\ , then, else,\ for, do, a, b, c\}$$

Definíció: Szó: Az ábécé betűiből alkotott véges sorozat.
Példa.:

T_1 esetén: aabbaa =: u_1
T_2 esetén: if b then for c do if b then a else c =: u_2

Definíció: Szó hossza: betűinek a száma.
Példa: $|u_1| = 6$, $|u_2| = 12$

Definíció: Üres szó: olyan szó, amely nem tartalmaz betűt.
Jele: ε Az üres szó hossza: |ε|=0

Jelölés: T abc

1. T* := {T-ből alkotott összes szavak halmaza}
2. T^+ :=T* \ {ε}

Példa: $T_1^* = \{\varepsilon, a, b, c, ab, ac, bc, aa, bb, cc, abc, ...\}$

Definíció: Formális nyelv:
Legyen T egy abc. L ⊆ T* formális nyelv a T abc felett.
Azaz képezzük egy abc betűiből az összes lehetséges szót (ez nyilván végtelen halmaz), majd vesszük ennek a halmaznak egy tetszőleges (véges vagy végtelen) részhalmazát. Az így kapott részhalmaz egy formális nyelv.

Példa:

1. T_1 abc feletti lehetséges formális nyelv: { ab,ba }
   vagy { ab,aabb, aaabbb, aaaabbbb, …, a^ib^i, …. }

2. T_2 Pascal alapszimbólumok
   E feletti nyelv: a helyesen megírt Pascal programok halmaza := L_{Pascal}

Tehát az, hogy egy Pascal-ban megírt program (p) szintaktikailag helyes, megfogalmazható úgy is, hogy p ∈ L_{Pascal}. Feladatunk majd a továbbiakban ennek eldöntése lesz.

Hogyan definiálhatunk formális nyelveket?

- Véges nyelveket: elemeik megadásával
  Pl: L = {aa,ab,ba,bb }

- Végtelen nyelveket:

  - Halmaz megadásával:
    Pl: L= {a^i b^i, i ≥ 0}

  - Szabályok segítségével: (BNF általánosítása)

A továbbiakban csak ezzel az utóbbi esettel foglakozunk, azaz formális nyelveket hozunk létre (generálunk) szabályok (nyelvtanok) segítségével.

4.3. Formális nyelvek generátorai – generatív grammatikák

Definíció: G generatív grammatika a következő rendezett négyes:

G = <T, N, S, P>, ahol

- T: véges abc, terminális jelek halmaza
- N: véges halmaz – nemterminális (vagy más szóval nyelvtani) jelek, ahol $T \cap N = 0$
- $S \in N$ kezdőjel, vagy kiinduló szimbólum
- P: helyettesítési szabályok véges halmaza

A szabályok $p \rightarrow q$ alakúak, ahol

- $p \in (T \cup N)^*$ és p-ben van nyelvtani jel
- $q \in (T \cup N)^+$

Példa1: A BNF során tárgyalt azonosító definíciójának megadása ebben a terminológiában:

T N S

G_1:={ {A,B,...,Z,0,1,...,9} , { <azonosító>,<azonvég>,<betű>,<szjegy> } , <azonosító>, P }

ahol P:

<azonosító>	→	<betű>
<azonosító>	→	<betű><azonvég>
<azonvég>	→	<betű> | <szjegy> | <betű><azonvég> | <szjegy><azonvég>
<betű>	→	A | B | ... | Z
<szjegy>	→	0 | 1 | ... | 9

Példa 2.:

G_2= < {a,b} , {S} , S, P >

P: {S → aSb

S → ab}

(Rövid jelölés: S → aSb | ab)

Definíció: Levezetés egy grammatikában az alábbi: kiindulunk a kezdőszimbólumból, és választunk egy olyan szabályt, melynek bal oldalán a kezdőszimbólum áll. A kezdőszimbólumot helyettesítjük az ezen szabály jobb oldalán álló szóval. Amennyiben az így kapott szóban van még nyelvtani jel, ezt is helyettesítjük a megfelelő szabályok alapján, egészen addig, míg a kapott szó már csak terminális jeleket tartalmaz. Az így kapott α szóra azt mondjuk, hogy S-ből levezethető.

Példa: Egy levezetés a 2. példában leírt grammatikában:
S → aSb→aaSbb→aaabbb,
Azaz az aaabbb szó levezethető az S kezdőszimbólumból.

Más megfogalmazással:
Definíció: G = < T, N, S, P > nyelvtan generálja az $\alpha \in T^*$ szót, ha: S-ből levezethető az α. ($S \neq \alpha$)

Definíció: G = < T, N, S, P > grammatika generálja az $L \subseteq T^*$ nyelvet, ha: L= {α | $\alpha \in T^*$ és α levezethető S-ből}
Azaz a G nyelvtan által generált formális nyelv nem más, mint a nyelv kezdőszimbólumából levezethető terminális jelsorozatok összessége.

Jelölés: L(G)

Példa1: $L(G_1)$ = { azonosítók } (a fenti 1. példának megfelelően)

Példa2: $L(G_2) = \{ a^i b^i , i \geq 1 \}$
A fenti 2. példában leírt nyelvtan által generált nyelv azon szavakból áll, melyek a^i b^i alakúak i ≥1 (azaz tartalmaznak valamennyi „a” betűt és utána ugyanannyi „b” betűt).

Definíció: G’ és G” nyelvtanok ekvivalensek, ha L(G’) = L(G”), azaz két nyelvtan akkor ekvivalens, ha az általuk generált nyelv megegyezik (tehát szabályaiknak és nyelvtani jeleiknek nem feltétlenül kell megegyezniük).

További példák:

Példa3: Természetes számok felírása:
$G_3 = < \{0,..,9\} , \{s\}, s, P >$
P: $s \rightarrow s0 \mid s1 \mid ... \mid s9 \mid 1 \mid 2 \mid ... \mid 9$

Példa4: 3-mal osztható természetes számok (009, 000, stb. is a nyelvhez tartoznak):

A nem terminális jelek a 3-mal való osztás maradékosztályai lesznek:
A_0 azon számok, melyek 3-mal osztva 0-t adnak maradékul,
A_1 azon számok, melyek 3-mal osztva 1-t adnak maradékul, végül
A_2 azon számok, melyek 3-mal osztva 2-t adnak maradékul.

$G_4 = < \{0,..,9\} , \{A_0, A_1, A_2 \} , A_0, P >$
P: $A_0 \rightarrow 0 \mid 3 \mid 6 \mid 9 \mid 3A_0 \mid 6A_0 \mid 9A_0 \mid 2A_1 \mid 5A_1 \mid 8A_1 \mid 1A_2 \mid 4A_2 \mid 7A_2$
$A_1 \rightarrow 1 \mid 4 \mid 7 \mid 1A_0 \mid 4A_0 \mid 7A_0 \mid 0A_1 \mid 3A_1 \mid 6A_1 \mid 9A_1 \mid 2A_2 \mid 5A_2 \mid 8A_2$
$A_2 \rightarrow 2 \mid 5 \mid 8 \mid 2A_0 \mid 5A_0 \mid 8A_0 \mid 1A_1 \mid 4A_1 \mid 7A_1 \mid 0A_2 \mid 3A_2 \mid 6A_2 \mid 9A_2$

Levezethető-e ebben a grammatikában a **893172** szó, azaz osztható-e 3-mal?
A levezetés:
$A_0 \rightarrow 8\ A_1 \rightarrow 89A_1 \rightarrow 893\ A_1 \rightarrow 8931\ A_0 \rightarrow 89317\ A_2 \rightarrow 893172$

4.4.Nyelvek osztályozása (Chomsky-féle hierarchia)

A nyelvtanokat szabályaik alakja alapján Noam Chomsky, amerikai nyelvész (1928.) 4 osztályba sorolta. Ezek az alábbiak:
a 0. típusú vagy általános nyelvtanok,
az 1. típusú vagy környezetfüggő nyelvtanok,
a 2. típusú vagy környezetfüggetlen nyelvtanok és
a 3. típusú vagy reguláris nyelvtanok.

Definíció: 0. típusú nyelvtan:

Ebben az esetben a szabályokra nincs megkötés, azaz a szabályok alakja: $\alpha A \beta \rightarrow \omega$ alakú, ahol
$\alpha, \beta, \omega \in (T \cup N)^*$ és $A \in N$.
Tehát az egyetlen megkötés, hogy a szabály bal oldalán legalább egy nyelvtani jelnek szerepelnie kell, ezen túl mind a szabályok bal mind a jobb oldalán állhat terminális és nemterminális jelek véges sorozata.

Definíció: $G = < T,N,S,P >$ 1-es típusú nyelvtan, ha a szabályok alakja:

$\alpha A \beta \rightarrow \alpha \omega \beta$, ahol
$\alpha, \beta, \omega \in (T \cup N)^*$ és $A \in N$, $\omega \neq \varepsilon$

Tehát A nyelvtani jel csak bizonyos környezetben helyettesíthető ω-val, ahol ω terminális és nemterminális jelek tetszőleges véges sorozata. α és β az A nyelvtani jel bal- ill. jobboldali környezete, mely a szabály jobb oldalán is „közreveszi" ω-t.

Megjegyzés: Egy nyelvtan csak akkor tekinthető 1-es típusúnak, ha valamennyi szabálya megfelel az 1-es típusú követelményeknek. Ha csak 1 szabálya is 0. típusú, akkor a nyelvtan már 0. típusú.

Megjegyzés: Azért, hogy az üres szót is levezethessük, megengedjük az $S \rightarrow \varepsilon$ szabályt, de ekkor S nem állhat szabály jobb oldalán. (Így a jobb oldal soha nem rövidebb, mint a bal.)

Példa: $G_5 = < \{a,b,c\} , \{S,A,B,C\} , S, P >$

P:	$S \rightarrow aSAC \mid abC$	itt S jobb- és baloldali környezete: ε
	$CA \rightarrow BA$	itt C jobboldali környezete: A, balodali környezete: ε
	$BacA \rightarrow BacC$	itt A baloldali környezete: Bac, jobbodali környezete: ε
	$BCCa \rightarrow ACCa$	itt B jobboldali környezete: CCa, balodali környezete: ε
	$bA \rightarrow bb$	itt A baloldali kömyezete : b, jobbodali környezete: ε
	$C \rightarrow c$	itt C jobb- és baloldali környezete: ε

Belátható: $L(G_5) = \{ a^n b^n c^n \mid n \geq 1 \}$.

Definíció: $G = < T,N,S,P >$ 2-es típusú nyelvtan, ha a szabályok alakja:

$A \rightarrow \omega$, ahol $A \in N$ és $\omega \in (T \cup N)^*$, $\omega \neq \varepsilon$
Tehát szabály bal oldalán már nem állhat tetszőleges sorozat, hanem csak egy nyelvtani jel (A), a jobboldalon pedig terminális és nemterminális jelek tetszőleges véges sorozata, ω.

Megjegyzés: G tartalmazhatja az $S \rightarrow \varepsilon$ szabályt, S ekkor nem állhat szabály jobb oldalán.

Megjegyzés: Számunkra a legfontosabbak a 2. típusú nyelvtanok lesznek, mert a programozási nyelveket leíró szabályok ebbe a csoportba sorolhatók. Ha megnézzük a BNF segítségével leírt szabályokat, felismerhetjük bennük a 2. típusú szabályok jellemzőit!

Példa: $G_6 := < \{0,1\} , \{S\} , S, P >$
P: $S \rightarrow SS \mid 0S1 \mid 1S0 \mid 10 \mid 01$
$L(G_6) = \{\alpha \in (0,1)^* \mid d(\alpha) \geq 2$, α-ban ugyanannyi 1-es van, mint 0$\}$

Definíció: G 3-as típusú nyelvtan, ha a szabályok alakja:
$A \rightarrow aB$, vagy $A \rightarrow a$, ahol $A,B \in N$, $a \in T$.
Tehát szabály bal oldalán továbbra is már csak egy nyelvtani jel (A) állhat, de itt már a jobboldalon sem állhat tetszőleges sorozat, hanem csak vagy egy terminális jel vagy egy terminális és egy nemterminális jel.

Megjegyzés: G tartalmazhatja az $S \rightarrow \varepsilon$ szabályt, S ekkor nem állhat szabály jobb oldalán.

Példa: $G_7 := < \{1\} , \{ S,A,B\} , S, P >$
P: $S \rightarrow 1A \mid \varepsilon$
$A \rightarrow 1B \mid 1$
$B \rightarrow 1A$
$L(G_7) = \{ 1^{2n} \mid n \geq 0 \}$

Definíció: $G_i := \{ L \mid \exists$ G i. típusú nyelvtan, olyan, hogy $L = L(G) \}$, azaz G_i azon nyelvek összessége, melyekhez létezik őket generáló i. típusú nyelvtan (itt $0 <= i <= 3$).

Definíció: L egy i. típusú nyelv, ha $L \in G_i$, azaz létezik őt generáló i. típusú nyelvtan.
Ilyen értelemben beszélünk általános, környezetfüggő, környezetfüggetlen, reguláris nyelvekről.

Tétel: (Chomsky-féle hierarchia)
$G_0 \supseteq G_1 \supseteq G_2 \supseteq G_3$

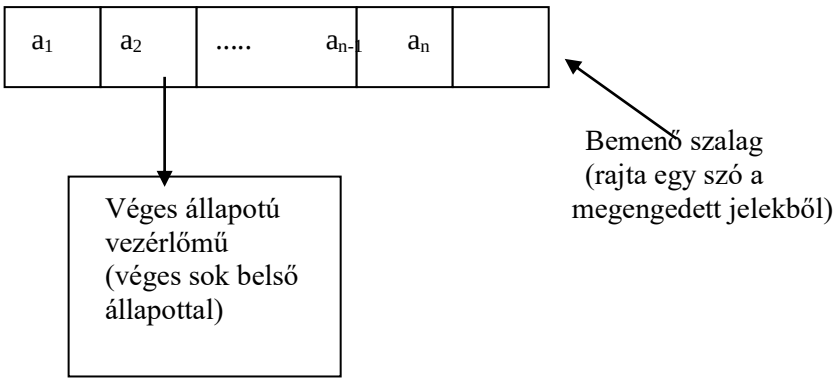
Tehát legszűkebb a reguláris nyelvek osztálya, ennél bővebb a környezetfüggetlen majd a környezetfüggő nyelvek osztálya, míg a legbővebb az általános nyelvek osztálya.

Megjegyzés: A szigorú tartalmazási reláció is belátható, azaz pl. létezik olyan 2. típusú nyelv, mely nem 3. típusú is egyben, stb.

4.5. Véges automaták, mint felismerők

A véges automaták a formális nyelvek (ezen belül reguláris nyelvek) felismerőinek viszonylag egyszerű matematikai modelljei.

A véges automaták szemléltetése:



4.6. ábra

Működésük:

Az automata rendelkezik egy bemenő szalaggal. Erre egy olyan szó van felírva, amelyet a megengedett bemenő jelekből (a bemenő abc jeleiből) alkottunk. Emellett az automata rendelkezik egy „vezérlőművel", melynek véges sok ún. belső állapota lehet.

A belső állapotok között van egy kitüntetett állapot, a kezdőállapot, az automata indulásakor ebben az állapotban található. Továbbá az automatának bizonyos állapotai végállapotok.

Működése során az automata beolvas egy jelet balról jobbra a bemenő szalagról. Ettől és az aktuális belső állapotától függően egy adott másik belső állapotba kerül. (Determinisztikus automata esetén ez az állapot egyértelműen meghatározott.) Ezután újra beolvas egy jelet a bemenő szalagról és ennek valamint a jelenlegi állapotának függvényében egy újabb állapotba kerül.

Az automata akkor fejezi be a működését, ha kiürült a bemenő szalag.

Ha az automata befejezéskor végállapotba kerül, akkor azt mondjuk, hogy felismerte a bemenő szalagra írt szót.

Definíció: Az automata által felismert vagy elfogadott nyelv: az automata által felismert szavak halmaza. Azaz: az automata ($\mathcal{A}$) által elfogadott nyelv:

$L(\mathcal{A}) := \{u \mid u \in T^*$, u-t $\mathcal{A}$ elfogadja $\}$

***Definíció*:** Két automata ekvivalens, ha az általuk elfogadott nyelv megegyezik.

Tehát a véges determinisztikus automatákat ($\mathcal{A}$) 5 jellemzővel lehet leírni:

$\mathcal{A} = < A,T,a_0,F,\delta >$

$|A| < \infty$: belső állapotok véges halmaza

$|T| < \infty$: bemenő jelek véges halmaza (bemenő abc)

$a_0 \in A$: kezdő állapot

$F \subseteq A$: végállapotok halmaza

$\delta: AxT \rightarrow A$:átmenetfüggvény, amely leírja, hogy egy adott bemenő jel hatására egy adott belső állapotból melyik belső állapotba kerüljön az automata.

Az automaták megadása történhet:

1. Jellemzőinek felsorolásával, ahol az átmenet függvény táblázatos formában megadható.
2. Irányított gráffal, ahol a belső állapotok a gráf csúcsai, melyeket körrel jelzünk, beleírva az állapot megnevezését. A kezdőállapotot a neki megfelelő csúcsba bemutató él jelzi, a végállapotokat dupla körrel jelöljük. Az átmenet függvény értékei az éleken találhatók, ha pl. „a” bemenő jel hatására „A” belső állapotból „B” állapotba juthatunk, akkor „A” és „B” csúcsok közé elhelyezünk egy „a”-val feliratozott (színezett) élt.

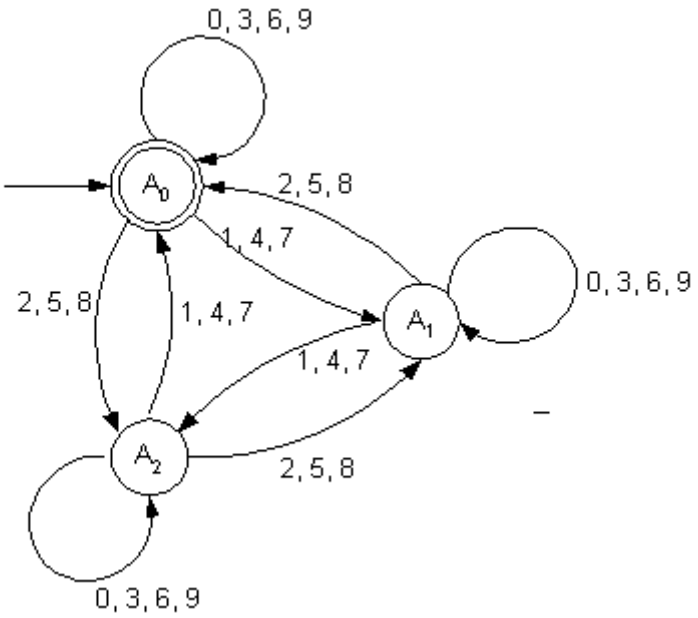
Példa1: Az alábbi automata a 3-mal osztható számokat ismeri fel:

1. Megadás a jellemzők segítségével:

$\mathcal{A}_1 := < \{A_0, A_1, A_2\} , \{ 0,1,2,3,4,5,6,7,8,9 \} , A_0 , \{ A_0\} , \delta >$, azaz
három lehetséges állapot van: A_0, A_1, A_2,
a bemenő jelek: 0,1,2,3,4,5,6,7,8,9,
kezdőállapot A_0,
egy végállapot van: A_0,
az átmenetfüggvény, δ pedig az alábbi módon adható meg táblázattal:

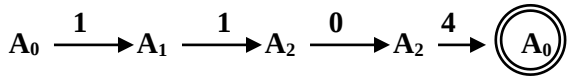
δ	0	1	…	9
A_0	A_0	A_1	…	A_0
A_1	A_1	A_2	…	A_1
A_2	A_2	A_0	…	A_2

2. Megadás gráffal:



4.7. ábra

Az automata működése a bemenő szalagra írt $\alpha = 1104$ szó esetén:



Azaz az automata induláskor az A_0 állapotban van, onnan az 1-es jel beolvasásának hatására az A_1 –es állapotba kerül, majd az újabb 1-es beolvasása után A_2 állapotba, stb.
A szó végigolvasása után az A_0 állapotban található.
Mivel ebben az esetben A_0 végállapot, megállapítható, hogy α szót az automata elfogadta, felismerte, ami ebben az esetben azt jelenti, hogy α 3-mal osztható.

Példa2:

1. Megadás a jellemzők segítségével:

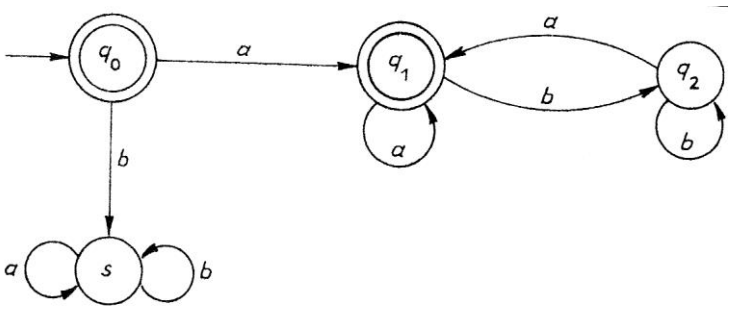
$\mathcal{A}_2 := < \{q_0, q_1, q_2, s\}, \{a,b\}, q_0, \{q_0, q_1\}, \delta >$

$\delta(q_0,a)= q_1$ $\quad\quad$ $\delta(q_0,b)= s$
$\delta(q_1,a)= q_1$ $\quad\quad$ $\delta(q_1,b)= q_2$
$\delta(q_2,a)= q_1$ $\quad\quad$ $\delta(q_2,b)= q_2$
$\delta(s,a)= s$ $\quad\quad$ $\delta(s,b)= s,$

azaz táblázattal leírva:

δ	a	b
q_0	q_1	s
q_1	q_1	q_2
q_2	q_1	q_2
s	s	s

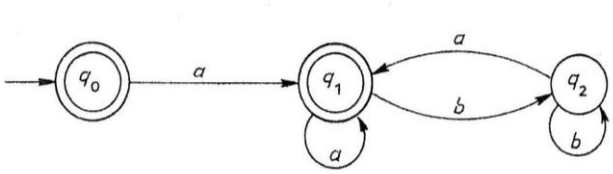
2. Megadás gráffal:



4.8. ábra

Mely szavakat fogadja el ez az automata?
- az üres szót,
- az a-val kezdődő és a-val végződő szavakat.

A fenti automatával ekvivalens automata:



4.9. ábra

4.6. Nemdeterminisztikus véges automaták

A nemdeterminisztikus automaták abban különböznek a determinisztikus automatáktól, hogy egy belső állapotból egy adott bemenő jel hatására több lehetséges állapotba is kerülhet az automata, azaz működése nem előre meghatározott, determinált.

A nemdeterminisztikus automatát – hasonlóan a determinisztikushoz -, 5 jellemző írja le:

$\mathcal{A} = < A, T , a_0, F, \delta >$

A:	belső állapotok véges, nem üres halmaza,
T:	bemenő abc,
a_0:	kezdőállapot ($a_0 \in A$),
$F \subseteq A$	végállapotok halmaza,
$\delta: A \times T \rightarrow 2^A$	állapotátmenet függvény, mely a belső állapotok és a bemenő jelek halmazából a belső állapotok részhalmazaiba képez.

Szó felismerése: u szót az automata felismeri, ha betűit balról jobbra végigolvasva létezik olyan működése, mellyel végállapotba kerül, azaz az elérhető állapotai között van végállapot.

A nemdeterminisztikus automatát a determinisztikus automatához hasonlóan ábrázolhatjuk.

Példa:

$\mathcal{A} := < \{q_0, q_1, q_2\} , \{ 0,1\} , q_0 , \{ q_1, q_2\} , \delta >$, ahol

δ	0	1
q_0	$\{q_1,q_2\}$	$\{q_0\}$
q_1	$\{q_0,q_1\}$	$\varnothing$
q_2	$\{q_0,q_2\}$	$\{q_1\}$

Működése a 0101 szóra:

1., $\delta(q_0,0) = \{q_1,q_2\}$
2., $\delta'(q_0,01) = \delta(q_1,1) \cup \delta(q_2,1) = \varnothing \cup \{q_1\} = \{q_1\}$
3., $\delta'(q_0,010) = \delta(q_1, 0) = \{q_0, q_1\}$
4., $\delta'(q_0,0101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \varnothing = \{q_0\}$

Mivel $q_0 \notin F$, a szót az automata nem fogadja el.

Definíció: G_A := { L(A) | A determinisztikus automata}, azaz azon nyelvek osztálya, melyekhez létezik a nyelvet felismerő véges determinisztikus automata.

G_{ND} := { L(A) | A nem determinisztikus automata}, azaz azon nyelvek osztálya, melyekhez létezik a nyelvet felismerő véges nemdeterminisztikus automata.

Tétel: $G_A = G_{ND}$, azaz a véges determinisztikus automatákkal felismerhető nyelvek osztálya megegyezik a véges nemdeterminisztikus automatákkal felismerhető nyelvek osztályával.

Bizonyítás:

$G_A \subseteq G_{ND}$

Triviális, hiszen minden determinisztikus automata felfogható speciális nemdeterminisztikus automataként.

$G_{ND} \subseteq G_A$

Az állítást konstruktív módon bizonyítjuk, azaz egy adott nemdeterminisztikus automatához megszerkesztjük a vele ekvivalens determinisztikus automatát.

Legyen A:= < A,T,a_0,F,δ > véges, nemdeterminisztikus automata.
Ehhez konstruálunk egy vele ekvivalens véges determinisztikus automatát az alábbi módon:

A' := < Q,T,t_0,F',δ' > a véges determinisztikus automata, ahol:

Q := 2^A , azaz A részhalmazai. Tehát a determinisztikus automata állapotainak száma, annyi, ahány részhalmaz képezhető az eredeti nemdeterminisztikus automata állapotaiból.

T ugyanaz, mint az eredeti automata esetén.

t_0 := { a_0 }, azaz az eredeti kezdőállapotot tartalmazó egyelemű részhalmaz.

F' := { H | H∈Q , H ∩ F ≠∅ }, azaz minden olyan részhalmaz végállapotnak tekintendő, amely tartalmaz elemet az eredtei végállapotok közül.

δ': δ'(H,a)=H', ha H,H' ∈ Q, a ∈ T és H = {p_1, ...p_1} p_i ∈A és H' {r_1, ... r_m} r_i ∈A és {r_1, ... r_m} = δ(p_1,a)∪ δ(p_2,a)∪...∪ δ(p_1,a)

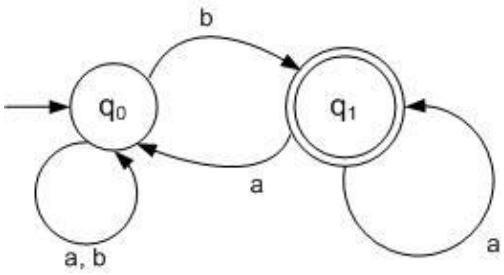
Azaz δ'(H,a):= $\bigcup_{p_i \in H} \delta(p_i, a)$

Példa :

$\mathcal{A}_{ND}$:= < {q_0, q_1} , {a,b} ,q_0, {q_1}, δ > véges nemdeterminisztikus automata, ahol

δ	a	b
q_0	{q_0}	{q_0,q_1}
q_1	{q_0,q_1}	∅

Az automata gráffal szemléltetve:



4.10. ábra

A hozzá tartozó determinisztikus automata:
$\mathcal{A}_D$:
Az eredeti 2 állapotból 4 részhalmaz képezhető, mindegyiknek megfeleltetünk egy állapotot, ezeket t-vel jelöljük, alsó indexben utalva a részhalmazra:

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

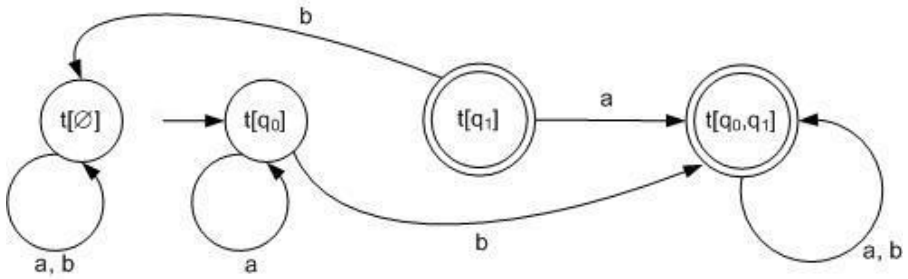
$t[\varnothing]$ $t[q_0]$ $t[q_1]$ $t[q_0,q_1]$

$T= \{a,b\}$,
$t_0 = \{q_0\}$, melynek a $t[q_0]$ állapot felel meg,
$F' = \{H \in Q \mid H \cap \{q_1\} \neq \varnothing\} = \{\{q_1\},\{q_0,q_1\}\}$, melyeknek a $\{t[q_1],t[q_0,q_1]\}$ állapotok felelnek meg.

Azaz $\mathcal{A}_D = < \{t[\varnothing],t[q_1],t[q_0],t[q_0,q_1]\},\{a,b\},t[q_0],\{t[q_1],t[q_0,q_1]\}, \delta' >$, ahol

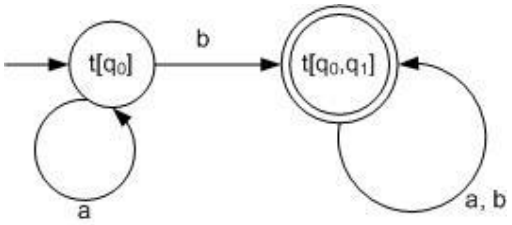
δ'	a	b
$t[\varnothing]$	$t[\varnothing]$	$t[\varnothing]$
$t[q_0]$	$\delta(q_0,a)=\{q_0\}\rightarrow t[q_0]$	$\delta(q_0,b)=\{q_0,q_1\}\rightarrow t[q_0,q_1]$
$t[q_1]$	$\delta(q_1,a)=\{q_0,q_1\}\rightarrow t[q_0,q_1]$	$\delta(q_1,b)=0\rightarrow t[\varnothing]$
$t[q_0,q_1]$	$\delta(q_0,a)\cup\delta(q_1,a)=\{q_0\}\cup\{q_0,q_1\}\rightarrow t[q_0,q_1]$	$\delta(q_0,b)\cup\delta(q_1,b)=\{q_0,q_1\}\cup\varnothing\rightarrow t[q_0,q_1]$

A kapott automata gráffal ábrázolva:



4.11. ábra

Egyszerűsítve:



4.12. ábra

Az automata minden szót felismer, amely tartalmaz b betűt.

Tétel: $G_3 = G_A$, azaz a véges nemdeterminisztikus (ill.: a véges determinisztikus) automaták által elfogadott nyelvek osztálya megegyezik a reguláris nyelvek osztályával. Vagyis a véges nemdeterminisztikus (ill.: a véges determinisztikus) automaták a reguláris (3. típusú) nyelvek felismerői.

Bizonyítás:
$G_3 \subseteq G_{ND}(=G_A)$, azaz belátjuk, hogy ha L reguláris nyelv, akkor található olyan véges nemdeterminisztikus automata , amely az L nyelvet ismeri fel.

A bizonyítás konstruktív, azaz L reguláris nyelvhez megkonstruáljuk az őt felismerő véges nemdeterminisztikus automatát. A továbbiakban csak magát a konstrukciót mutatjuk be.

Legyen L reguláris nyelv, melyet a G = < T,N,S,P > nyelvtan generál.
A hozzá tartozó A nemdeterminisztikus automata definíciója:

$\mathcal{A}_{ND}$:= < A,T,a_0,F,δ >, ahol
A:= N∪{E} , E ∉ T∪N,
a_0:= S,
F={E}, ha P-ben nem szerepel az S→ε szabály, F={E,S} különben,
δ(B,a)= {D_i}, ahol B, $D_i \in A$, a∈T, ha P-ben szerepel az alábbi szabály: B → aD_i
továbbá E∈{D_i} , ha P-ben szerepel a B → a szabály.
δ(E,a)= ∅ ∀ a∈T.

Példa: G:=< {a,b}, {S,A,B} ,S,P >, ahol a szabályok a következők:
P: S → aS | bS | aA
A → bB

$B \rightarrow a$

A hozzá tartozó automata:

$\mathcal{A}_{ND}$:=<{S,A,B,E},{a,b},S,{E},δ >, ahol

δ	a	b
S	{S,A}	{S}
A	∅	{B}
B	{E}	∅
E	∅	∅

Bizonyítás (folytatás):
$G_A \subseteq G_3$, azaz minden L nyelvhez, melyet egy véges determinisztikus automata felismer, létezik olyan reguláris grammatika, mely az L nyelvet generálja.

Bizonyítás:
A bizonyítás itt is konstruktív, azaz egy adott determinisztikus automatához megkeressük azt az L reguláris nyelvet, melyet az automata felismer.

Legyen $\mathcal{A}_D := < A, T, a_0, F, \delta >$ véges determinisztikus automata, mely L-t felismeri. Ehhez konstruáljuk a következő G:= <T, N, S, P> reguláris grammatikát, ahol:

T:=T, azaz a terminális jelek halmaza megegyezik az automata bemenő abc-jével,
N:=A, azaz a nyelvtani jelek halmaza megegyezik az automata belső állapotaival,
S:=a_0 azaz a kezdő nyelvtani jel megegyezik az automata kezdőállapotával,
A szabályok pedig az alábbi módon képezhetők:

P: - minden δ(q,a) = p-hez rendelhető egy $q \rightarrow ap$ szabály. (q,p∈A, a∈T)
- ha δ(q,a) = p esetén p∈F, akkor rendelnünk kell még egy $q \rightarrow a$ szabályt is.

Példa: $\mathcal{A}_D := < \{q_0,q_1,q_2\}, \{a,b\}, q_0, \{q_1,q_2\}, \delta >$, ahol

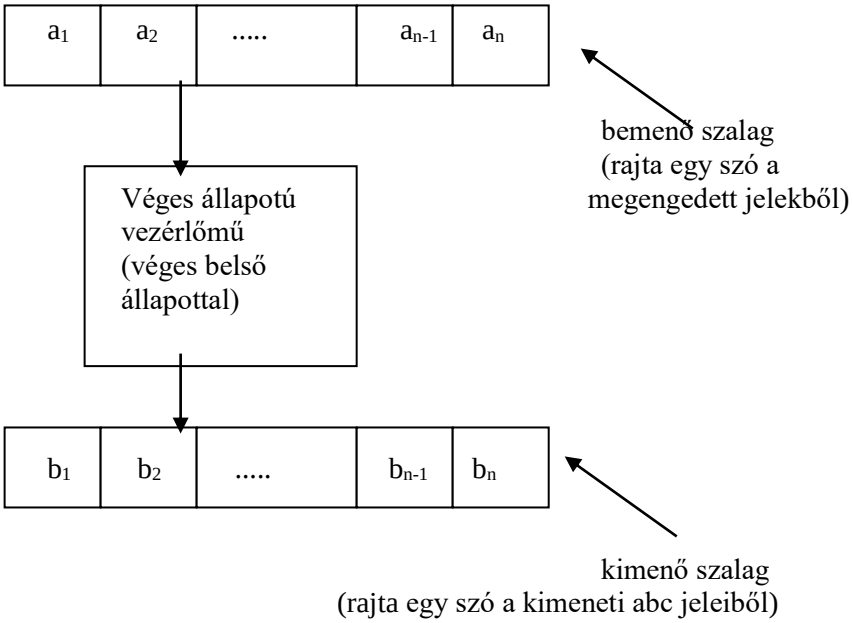
δ	a	b
q_0	q_1	q_2
q_1	q_0	q_2
q_2	q_1	q_0

A grammatika, mely ezt a nyelvet generálja:

G:= < {a,b}, { q_0,q_1,q_2} ,q_0, P >, ahol P szabályok a következők:

P: $q_0 \rightarrow aq_1$ $q_0 \rightarrow a$
$q_0 \rightarrow bq_2$ $q_0 \rightarrow b$
$q_1 \rightarrow aq_0$
$q_1 \rightarrow bq_2$ $q_1 \rightarrow b$
$q_2 \rightarrow aq_1$ $q_2 \rightarrow a$
$q_2 \rightarrow bq_0$

4.7. Véges automaták, mint formális nyelvek átalakítói



4.13. ábra

Az átalakító automaták működése:

- kezdőállapotból indul
- elolvas egy jelet a bemenő szalagról (balról jobbra)
- új belső állapotba kerül és leír egy jelet a kimenő szalagra (balról jobbra)

A működés befejeződik, ha

- nincs definiált következő lépés vagy
- elolvasta a bemenő szót

A működés eredménye:

- a kimenő szalagra írt szó

Kétféle átalakító automatáról tanulunk: Mealy-automatákról ill. Moore-automatákról.

Mealy – automata

Jellegzetessége, hogy a kimenő jel függ a belső állapottól és a beolvasott jeltől egyaránt. Megadásához 6 jellemzőre van szükség:

$\mathcal{M}$: = <A, V, W, δ, λ, q_0 >, ahol:

- A: belső állapotok véges, nem üres halmaza
- V: bemenő ábécé (V ≠ ∅, véges halmaz)
- W: kimenő ábécé (W ≠ ∅, véges halmaz)
- δ: átmenetfüggvény δ:AxV→A
- λ: kimeneti függvény λ: AxV→W
- q_0: kezdőállapot $q_0 \in$ A.

Működés: Az $a_1a_2.....a_k$ bemenő szó esetén:

1. lépés:	$q_1 := \delta(q_0,a_1)$	új belső állapot meghatározása
	$b_1 := \lambda(q_0,a_1)$	első kimenő jel meghatározása
2. lépés:	$q_2 := \delta(q_1,a_2)$	új belső állapot meghatározása
	$b_2 := \lambda(q_1,a_2)$	második kimenő jel meghatározása
…		

Eredmény:

$b_1b_2.....b_k=\lambda(q_0a_1)\lambda(q_1,a_2).....\lambda(q_{k-1},a_k) \in W^*$ szó, amely ugyanolyan hosszú, mint a beolvasott szó.

Definíció: M automata az $\alpha = a_1....a_k$ bemeneti szót $\beta = b_1....b_k$ kimeneti szóvá fordította le.
Jelölése: $\beta=\mathcal{M}(\alpha)$.

Definíció: Az M automata az L_1 formális nyelvet L_2-vé alakítja (fordítja le), ha
$L_2 = \{\omega|\omega=\mathcal{M}(\alpha),\alpha\in L_1\}$.

Az automata megadása – a véges determinisztikus automatákhoz hasonlóan történhet

- jellemzőinek megadásával, ahol az átmenetfüggvényt és a kimeneti függvényt egy táblázatban ábrázoljuk vagy
- gráffal, ahol az éleken a beolvasott jel mellett a kiírt jelet is feltüntetjük.

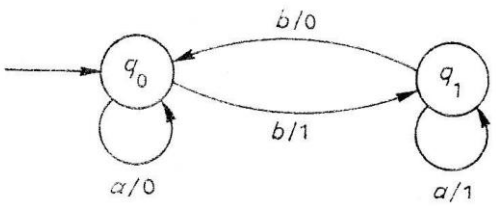
Példa: $\mathcal{M}_1 := <\{q_0,q_1\}, \{a,b\}, \{0,1\}, \delta, \lambda, q_0>$

$\delta(q_0,a)= q_0$	$\lambda(q_0,a)=0$
$\delta(q_0,b)= q_1$	$\lambda(q_0,b)=1$
$\delta(q_1,a)= q_1$	$\lambda(q_1,a)=1$
$\delta(q_1,b)= q_0$	$\lambda(q_1,b)=0$

Az átmenetfüggvény és a kimeneti függvény megadása táblázattal:

δ,λ	a	b
q_0	q_0,0	q_1,1
q_1	q_1,1	q_0,0

Az automata szemléltetése gráffal:



4.14. ábra

Hogyan működik ez az automata?

A kimeneti szó n. betűje 1 lesz, ha a bemeneti szóban az első n betű között páratlan b szerepel, egyébként 0.

Azaz pl. a baabbab bemenő szót a 1110110 kimenő szóra fordítja le.

Moore – automata

Jellemzője, hogy a kimeneti jel csak a belső állapottól függ.
Megadása a Mealy automatákhoz hasonlóan 6 jellemző segítségével történik:

$\mathcal{M}$:=< A, V, W, δ, λ, q_0>, ahol:
- A: belső állapotok véges, nem üres halmaza
- V: bemenő ábécé
- W: kimenő ábécé
- δ: átmenetfüggvény δ:AxV →A
- λ: kimeneti függvény λ: A →W
- q_0: kezdőállapot $q_0 \in A$

Működés: Az $a_1....a_k \in V^*$ bemeneti szóra:

0. lépés:	$b_0:=\lambda(q_0)$	1. kimeneti jel meghatározása
1. lépés:	$q_l:=\delta(q_0,a_1)$	új belső állapot meghatározása
	$b_1:=\lambda(q_l)$	2. kimeneti jel meghatározása
:		
:		
k. lépés:	$q_k:=\delta(q_{k-1},a_k)$	új belső állapot meghatározása
	$b_k:=\lambda(q_k)$	k. kimeneti jel meghatározása

Eredmény:
$b_0b_1b_2.....b_k=\lambda(q_0)\lambda(q_1)\lambda(q_2).....\lambda(q_k) \in W^*$ szó, azaz itt a kimeneti szóban egy betűvel több van, mint a bemeneti szóban.

Jelölés: $\mathcal{M}(\omega)$: ω bemenő szóból átalakított kimeneti szó.

Megadás: A Mealy automatákhoz hasonlóan
- jellemzőkkel vagy
- gráffal

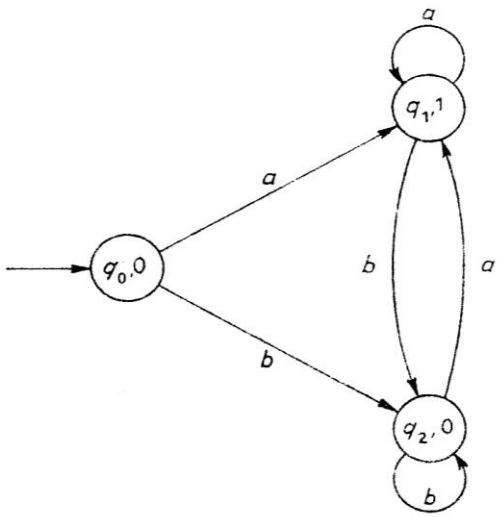
Példa $\mathcal{M}_1$:=<{q_0,q_1,q_2}, {a,b}, {0,1}, δ, λ, q_0>

$\delta(q_0,a) = q_1$	
$\delta(q_0,b) = q_2$	$\lambda(q_0) = 0$
$\delta(q_1,a) = q_1$	
$\delta(q_1,b) = q_2$	$\lambda(q_1) = 1$
$\delta(q_2,a) = q_1$	
$\delta(q_2,b) = q_2$	$\lambda(q_2) = 0$

Az átmenetfüggvény és a kimeneti függvény megadása táblázattal:

δ,λ	a	b
q_0	q_1,0	q_2,0
q_1	q_1,1	q_2,1
q_2	q_1,0	q_2,0

Az automata szemléltetése gráffal:



4.15. ábra

Hogyan működik ez az automata?

Kimeneti szó első betűje 0 lesz, a többi betűnél a bemenő a-t egyesre, a b-t nullára cseréljük.

Definíció: *Legyen* $\mathcal{M}$:=<A, V, W, δ, λ, q_0> egy Mea1y – automata, és
$\mathcal{N}$:=<A', V, W, δ', λ' q_0'> egy Moore - automata.
$\mathcal{M}$ és $\mathcal{N}$ ekvivalensek, ha minden ω bemeneti szóra: b$\mathcal{M}$(ω) = $\mathcal{N}$(ω), ahol b:=λ(q_0) a Moore automata kezdőállapotához tartozó kimenő jel. Azaz a két automata ekvivalens, ha ugyanazt a bemenő szót a Moore automata első kimenő jelétől eltekintve ugyanarra a kimenő szóra fordítják le.

Tétel: Tetszőleges Mealy-automatához létezik vele ekvivalens Moore-automata.

Tétel: Tetszőleges Moore-automatához létezik vele ekvivalens Mea1y-automata.

Tétel: Ha L reguláris nyelv, $\mathcal{M}$ egy tetszőleges Mea1y – automata, akkor az $\mathcal{M}$ által lefordított nyelv ($\mathcal{M}$(L)) szintén reguláris.

Tétel: Ha L reguláris nyelv, $\mathcal{N}$ egy tetszőleges Moore – automata, akkor az $\mathcal{N}$ által lefordított nyelv ($\mathcal{N}$(L)) szintén reguláris.

4.8. Környezetfüggetlen nyelvek

A továbbiakban a 2. típusú, azaz környezetfüggetlen nyelvekkel és az ő felismerőikkel, a veremautomatákkal foglalkozunk.

Mint tudjuk, a környezetfüggetlen nyelvek szabályainak alakja: $A \rightarrow \omega$, ahol $A \in N$ nyelvtani jel, $\omega \in (T \cup N)^*$, $\omega \neq \varepsilon$ nyelvtani és terminális jelek véges nem üres sorozata.

A környezetfüggetlen nyelvek szabályainak megfelelő levezetéseket irányított fagráfokkal, a szintaxisfával szemléltetjük, mint ezt a BNF tárgyalása során már láttuk.
A továbbiakban a szintaxisfa ill. a levezetés fogalmát pontosítjuk:

Definíció: Legyen G= <T, N, S, P> grammatika.
G-hez tartozó levezetési fa vagy szintaxisfa egy irányított fagráf, ha
- csúcsai $T \cup N$ elemei,
- gyökere S,
- és ha $A \in N$ esetén az A csúcs közvetlen leszármazottai balról jobbra $B_1, ..., B_k \in T \cup N$, akkor létezik P-ben $A \rightarrow B_1 ... B_k$ alakú szabály.

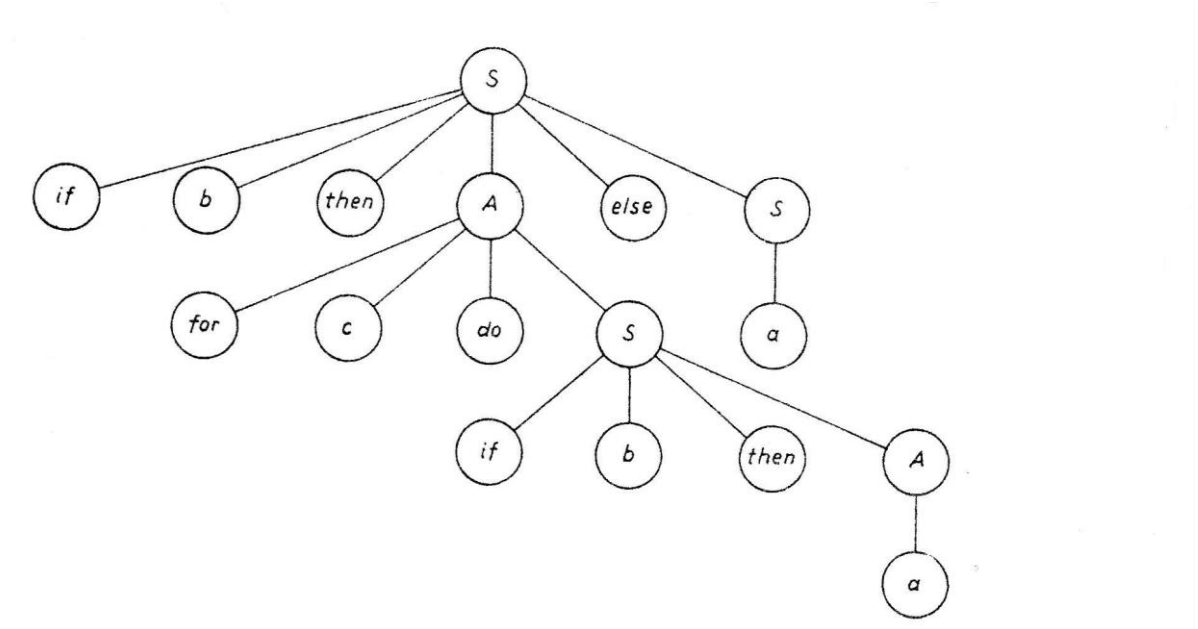
Definíció: A levezetési fa eredménye egy szó, melyet balról jobbra haladva a fa leveleiről olvasunk le.

Példa: G = <{if, then, else, for, do, a, b, c}, {S, A}, S, P>, ahol a szabályok az alábbiak:
P : $S \rightarrow$ if b then A | if b then A else S | a
$A \rightarrow$ for c do S | a

Cél az alábbi szó levezetése: if b then for c do if b then a else a

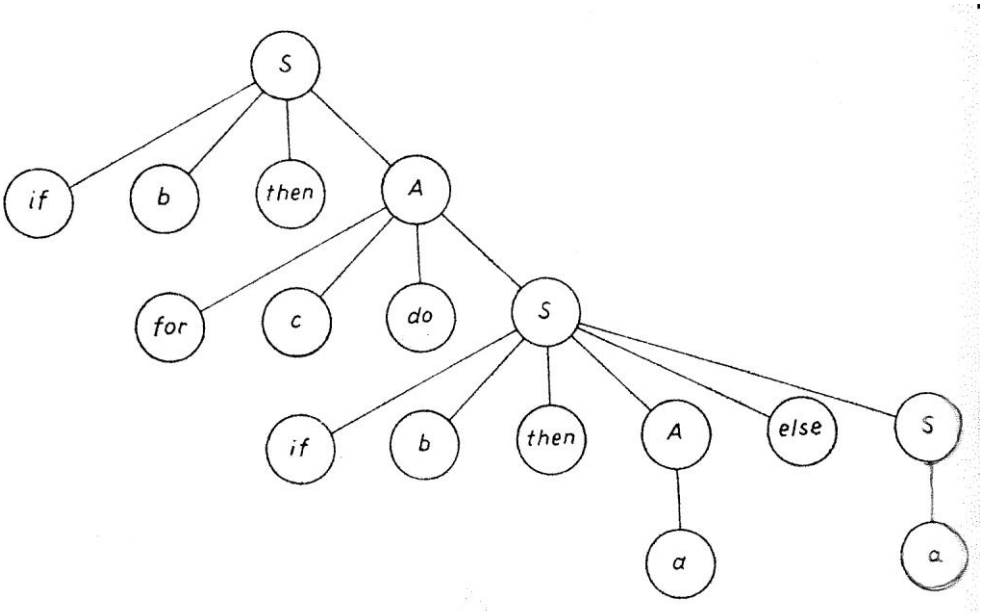
Tudjuk, hogy egy szó akkor vezethető le a szabályok segítségével, ha létezik szintaxisfája. A fenti esetben két különböző levezetési fát is konstruálhatunk, melynek eredménye az adott szó:

Az első levezetési fa:



4.16. ábra

A második levezetési fa:



4.17. ábra

Definíció: Legyen G=<T, N, S, P> környezetfüggetlen grammatika. Egy levezetés ebben a nyelvtanban legbaloldalibb levezetés, ha a levezetés lépéseinek során több nemterminális jel megléte esetén mindig a legbaloldalibb nemterminális jelet helyettesítjük valamely szabály alapján.

Példa:
A fenti két megadott szintaxisfa az alábbi két különböző legbaloldalibb levezetésnek felel meg:

1. S⊢ if b then A else S⊢ if b then for c do S else S ⊢ if b then for c do if b then A else S ⊢ if b then for c do if b then a else S⊢ if b then for c do if b then a else a.

2. S⊢ if b then A ⊢ if b then for c do S ⊢ if b then for c do if b then A else S ⊢ if b then for c do if b then a else S ⊢ if b then for c do if b then a else a.

Definíció: G többértelmű környezetfüggetlen grammatika, ha létezik benne olyan szó, melynek van legalább két különböző legbaloldalibb levezetése.

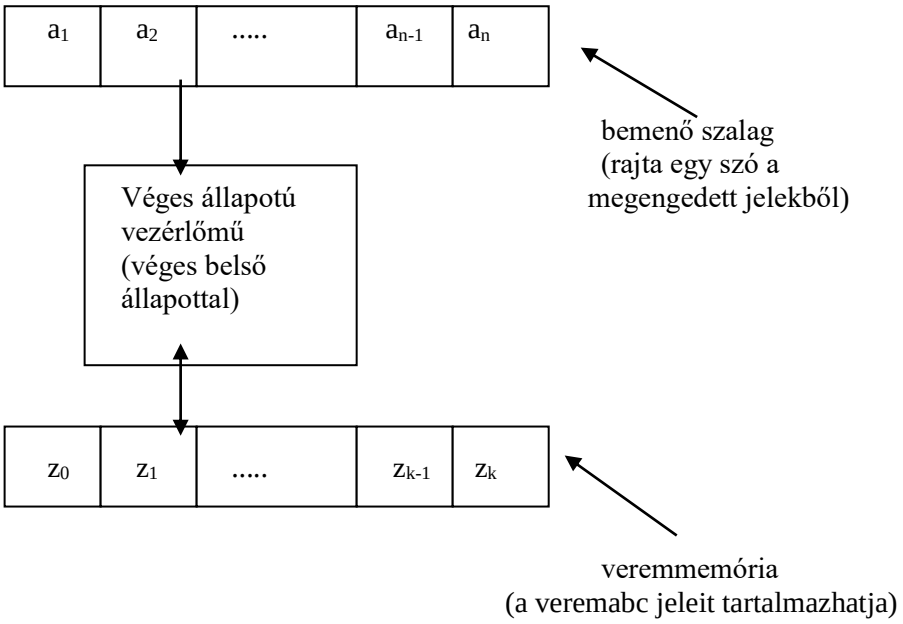
Megjegyzés: Programozási nyelvek szempontjából érdekes: ahhoz, hogy egy program olvasata egyértelmű legyen, kell, hogy az őt generáló grammatika egyértelmű legyen.
A többértelműség a grammatikák sajátja. Egy nyelvet elképzelhető, hogy le lehet írni egyértelmű és nemegyértelmű grammatikákkal is.

Definíció: Egy nyelv örökletesen többértelmű, ha nem létezik őt generáló egyértelmű grammatika.

Állítás: Ilyen nyelv létezik.

A továbbiakban algoritmust keresünk annak eldöntésére, hogy egy adott szó eleme-e egy környezetfüggetlen nyelvtannak, azaz azt vizsgáljuk, hogy $u \in L(G)$ avagy sem, ahol G környezetfüggetlen nyelvtan.
Ennek a kérdésnek az eldöntésére megpróbálunk építeni egy olyan szintaxisfát, melynek gyökere a nyelvtan kezdőjele (S), eredménye pedig u.
Ha sikerül ilyen szintaxisfát építenünk, akkor $u \in L(G)$, ha nem, akkor $u \notin L(G)$.
A szintaxisfa építésére a veremautomaták lesznek alkalmasak.

4.9. Veremautomaták



4.18. ábra

A veremautomata működése:

- Bekapcsoláskor a vezérlőmű egy rögzített kezdőállapotba kerül, a veremmemóriában egy rögzített kezdőjel van a veremabc jelei közül.
- Az automata lehetséges lépései:
    - beolvas egy bemenő jelet. Ettől és az aktuális belső állapotától függően, valamint attól függően, hogy milyen jel van a veremmemória tetején, új állapotba kerül és beír egy, a veremabc jeleiből álló szót, vagy az üres szót a veremmemóriába a beolvasott felső jel helyére. Ezt a lépést nevezzük normál lépésnek.
    - bemeneti jel beolvasása nélkül, az aktuális belső állapotától és a memória legfelső jelétől függően kerül új állapotba, és ír egy szót (akár itt is az üres szót, ezzel gyakorlatilag törölve a verem legfelső elemét) a veremmemóriába, szintén a beolvasott felső jel helyére. Ezt a lépést nevezzük ε-lépésnek.
- Az automata befejezi működését, ha
    - nincs definiált lépése vagy
    - kiürült a verem.

A veremautomata kétféle módon ismerhet fel egy szót:

- <u>*Végállapottal való felismerés:*</u> ha egy bemenő szó hatására létezik olyan működése, mellyel a szót végig tudja olvasni és a szó elolvasása után végállapotba jut. (A veremautomata működése nemdeterminisztikus, tehát egy adott helyzetben több lehetséges lépés is elképzelhető.)

- <u>*Üres veremmel való felismerés:*</u> ha egy bemenő szó hatására létezik olyan működése, mellyel a szót végig tudja olvasni és a szó elolvasása után a verem kiürül.

Egy $\mathcal{M}$ nemdeterminisztikus veremautomatát 7 jellemző határoz meg:

$\mathcal{M} = <A, V, W, \delta, q_0, z_0, F>$, ahol

A	a belső állapotok véges, nem üres halmaza
V	a bemenő ábécé
W	a veremábécé
δ	az átmenetfüggvény, ahol $\delta: A \times (V \cup \{\varepsilon\}) \times W \rightarrow A \times W^*$
$q_0 \in A$	a kezdőállapot
$z_0 \in W$	a kezdőjel a veremmemóriában
$F \subseteq A$	a végállapotok halmaza

Az $\mathcal{M}$ nemdeterminisztikus veremautomata konfigurációi írják le a veremautomata pillanatnyi aktuális állapotát. Ezt 3 dolog határozza meg: a belső állapot, amelyben az automata éppen van, a bemenő szónak az a maradéka, melyet az automatának még el kell olvasnia, valamint a verem pillanatnyi tartalma. Azaz a veremautomata konfigurációi az olyan $<q, \alpha, \gamma>$ hármasok, ahol $q \in A$ belső állapot, $\alpha \in V^*$ bemenő szó, $\gamma \in W^*$ a verem pillanatnyi tartalma.

A veremautomata működése nem más, mint ezen konfigurációk váltakozása, azaz a konfigurációk átvitele egymásba. Ez kétféle lehet annak megfelelően, hogy normál- vagy ε-lépést hajtottunk-e végre.

Legyen a bemenő szalagra felírt szó: $\alpha = a_1, a_2 ..... a_k$,
a veremmemória tartalma: $\gamma = x_1, ..., x_m$;
és $\gamma_i \in W^*$ a veremabc elemeiből összeállított szó.

Ekkor a konfiguráció-átmenet normál-lépés esetén:

Ha $(p_i, \gamma_i) \in \delta(q, a_1, x_m)$, akkor $<q, a_1 a_2 ... a_k, x_1 ... x_{m-1} x_m> \rightarrow <p_i, a_2 a_3 ... a_k, x_1 ... x_{m-1} \gamma_i>$, azaz a bemenő szalagról töröljük az első jelet, a verem legfelső elemét, x_m –et pedig helyettesítjük γ_i

–vel.

A konfiguráció-átmenet ε-lépés esetén:

Ha $(p_i,\gamma_i)\in\delta(q,\varepsilon,x_m)$, akkor $<q, a_1a_2...a_k, x_1...x_{m-1}x_m> \rightarrow <p_i, a_1...a_k, x_1...x_{m-1}\gamma_i>$, azaz a bemenő szalagról nem törlünk egy jelet sem, csak a verem legfelső elemét, x_m –et pedig helyettesítjük γ_i –vel.

Hogyan működik az automata a $b_1b_2...b_l$ bemenő szó hatására?

1. Kezdeti konfiguráció: $<q_0, b_1...b_l, z_0>$
2. δ-val meghatározzuk az összes lehetséges konfigurációt, amibe az eredeti konfiguráció átvihető.
3. Az összes új konfigurációkra meghatározzuk a lehetséges új konfigurációkat, stb. ...

Az automata felismer egy szót végállapottal: ha a sorozat utolsó elemei között létezik $<p,\varepsilon,\gamma>$ típusú, ahol $p\in F$.

Az automata felismer egy szót üres veremmel: ha a sorozat utolsó elemei között létezik $<p,\varepsilon,\varepsilon>$ típusú. (p nem feltétlen eleme F-nek).

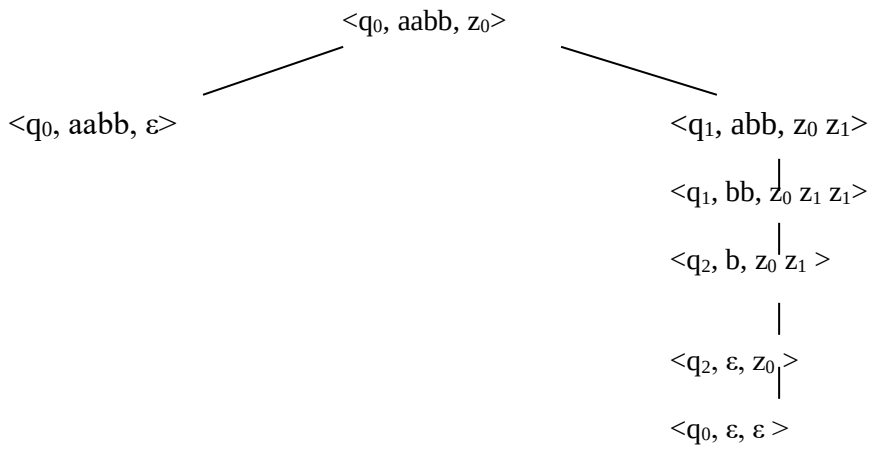
Definíció: Az $\mathcal{M}$ által végállapottal felismert nyelv - $T(\mathcal{M})$ - azon szavak halmaza, melyeket $\mathcal{M}$ végállapottal felismer.

Definíció: Az $\mathcal{M}$ által üres veremmel felismert nyelv - $N(\mathcal{M})$ - azon szavak halmaza, melyet $\mathcal{M}$ üres veremmel ismer fel.

Példa: $\mathcal{M}_1 := <\{q_0,q_1,q_2\}, \{a,b\}, \{z_0,z_1\}, \delta, q_0, z_0, \{q_0\}>$

$\delta(q_0,\varepsilon,z_0)= \{<q_0,\varepsilon>\}$
$\delta(q_0,a,z_0)= \{<q_1,z_0z_1>\}$
$\delta(q_1,a,z_1)= \{<q_1,z_1z_1>\}$
$\delta(q_1,b,z_1)= \{<q_2,\varepsilon>\}$
$\delta(q_2,b,z_1)= \{<q_2,\varepsilon>\}$
$\delta(q_2,\varepsilon,z_0)= \{<q_0,\varepsilon>\}$

A konfigurációk átmenetei az aabb bemeneti szó hatására:



4.19. ábra

Az automata végállapottal és üres veremmel is felismerte az aabb szót.
Az automata által felismert nyelv: $T(\mathcal{M})=N(\mathcal{M})=\{a^n b^n,\ n\geq 0\}$, vagyis azok a szavak amelyek elején „a” betűk állnak, majd ugyanannyi „b” betű.

Tétel: Egy L nyelvhez akkor és csak akkor létezik őt üres veremmel felismerő $\mathcal{M}_1$ nemdeterminisztikus veremautomata, ha létezik őt végállapottal felismerő $\mathcal{M}_2$ nemdeterminisztikus veremautomata is.

Definíció: Azt a folyamatot, mellyel megállapítjuk egy adott szóról, hogy egy adott grammatika generálja-e, szintaktikus elemzésnek nevezzük.

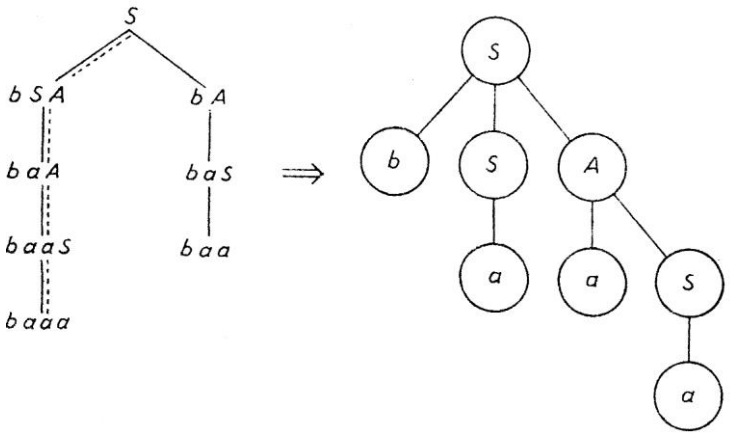
Definíció: Legyen G egy tetszőleges környezetfüggetlen grammatika. G szintaktikus elemzői az olyan automaták, melyek tetszőleges ω-ról eldöntik, hogy $\omega\in L(G)$ avagy sem, és megadják ω szó G-beli levezetési fáit is.

Megjegyzés: A továbbiakban olyan második típusú nyelvtanokkal foglalkozunk, ahol a szabályok alakja: $X\rightarrow dX_1 \ldots X_k$, ahol $d\in T$ terminális jel; $X,X_1, \ldots, X_k\in N$ nyelvtani jelek.

Állítás: Minden második típusú nyelvtanhoz létezik vele ekvivalens, a fenti alakú második típusú nyelvtan.

Példa: $G:= <\{a,b\},\{S,A\}, S,\{S \rightarrow a \mid bSA \mid bA,\quad A \rightarrow b \mid aS\}>$

Szintaktikus elemzés a „baaa” szó esetén, és a levezetési fa:



4.20. ábra

Tétel: Legyen G=<T, N, S, P> tetszőleges környezetfüggetlen grammatika. Ekkor létezik olyan $\mathcal{M}$ nemdeterminisztikus veremautomata, amelyik üres veremmel felismeri az L(G) nyelvet. Azaz a környezetfüggetlen (2. típusú) nyelvek felismerői a veremautomaták.

Bizonyítás: Konstruktív, azaz adott környezetfüggetlen nyelvtanhoz megkonstruáljuk az őt felismerő veremautomatát.

Tegyük fel, hogy $\varepsilon\notin L(G)$.
A veremautomata legyen a következő: $\mathcal{M}=<\{q\}, T, T\cup N, \delta, q, S, \varnothing>$, ahol $q\notin T\cup N$ és δ a következő:

- minden P-beli $A\rightarrow\alpha$ szabályra $<q,\alpha^{-1}>$ pár legyen eleme $\delta(q,\varepsilon,A)$-nak.

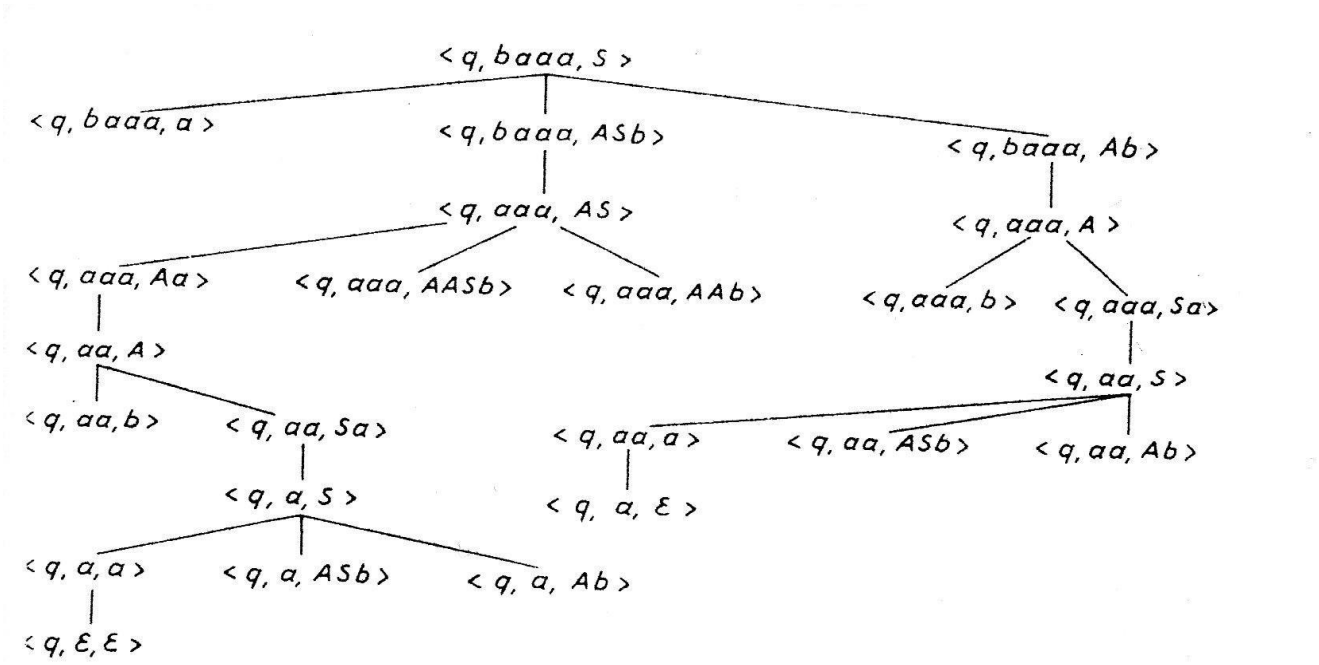
(α^{-1} az α tükörképe),
- minden a $\in$T-re $\delta(q,a,a)=\{<q,\varepsilon>\}$.

Példa: Az előbbi grammatikához tartozó $\mathcal{M}$ nemdeterminisztikus veremautomata, mely üres veremmel felismeri az L(G) nyelvet:

$\mathcal{M} = <\{q\}, \{a,b\}, \{a,b,S,A\}, \delta, q, S, \varnothing>$, ahol

$\delta(q,\varepsilon,S)=\{<q,a>, <q,ASb>, <q,Ab>\}$,
$\delta(q,\varepsilon,A)=\{<q,b>, <q,Sa>\}$,
$\delta(q,a,a)=\{<q,\varepsilon>\}$,
$\delta(q,b,b)=\{<q,\varepsilon>\}$,

A „baaa” szó szintaktikus elemzése:



4.21. ábra

Tehát $\mathcal{M}$ üres veremmel felismeri a „baaa” szót.

A veremautomata működése tehát általában egy ω szó esetén:
1. ε-lépéssel kicseréli S kezdőjelet α^{-1}-gyel, ha létezik $S\rightarrow\alpha$ szabály.
2. ω-t olvasva egyenként törli α^{-1} utolsó elemeit, ha azok megegyeznek ω beolvasott jeleivel.
3. Ha nemdeterminisztikus jelhez (pl. A) ér, helyettesíti α'^{-1} -gyel, ha létezik $A\rightarrow\alpha' \in P$ szabály.
4. Ha nem tud továbblépni, a fának ez az ága nem folytatható, ekkor visszalép és új helyettesítéssel próbálkozik.
5. Ha a veremmemória kiürül a szó elolvasása végén, akkor $\mathcal{M}$ felismeri ω-*t.*

Irodalomjegyzék

[1] ***Demetrovics*** – *Denev – Pavlov:* A számítástudomány matematikai alapjai.
Nemzeti Tankönyvkiadó

[2] ***Katona*** – *Recski - Szabó*: A számítástudomány alapjai.
Typotext kiadó

[3] *T.H.**Cormen**, C. E. Leiserson, R. L. Rivest*: Algoritmusok
Szerk:Iványi Antal
Műszaki Könyvkiadó

[4] ***μlogia34***: *Papné– Szlávi* - Zsakó: Módszeres programozás.
ELTE TTK Informatikai Tanszékcsoport 1998

[5] ***μlogia2***: *Szlávi – Zsakó*: Programozási forgácsok.
ELTE TTK Informatikai Tanszékcsoport

[6] ***Lipschutz***: Adatszerkezetek.
Schaum könyvek. Panem Kft 1993.