"BABEŞ - BOLYAI" TUDOMÁNYEGYETEM
KOLOZSVÁR
INFORMATIKAI TÁVOKTATÁS

ÁLLAMVIZSGA DOLGOZAT

DISZKRÉT MATEMATIKA MAPLE-BEN

Témavezető
Prof. dr. Kása Zoltán

Végzős hallgató
Egri Edith

Csíkszereda,2003

Tartalomjegyzék

Bevezetés

A számítógépek programozása, az algoritmusok elemzése matematikai előismeretekre épül. A rekurzív sorozatok tulajdonságait, a generátor függvények módszerét feltétlenül ismernie kell annak, aki összetett feladatokat akar önállóan megoldani számítógép segítségével.

A modern számítástudomány korszakában a technika alapját a diszkrét matematika alkotja. Ma már nélkülözhetetlenek egy informatikus számára a logika, a kombinatorika eszközei.

A diszkrét matematika elkülönült részekből álló, nem folytonos objektumokat tanulmányoz. Egy olyan tárgy , amely egyesíti a matematika különböző tradicionális területeit, úgy mint kombinatorika, aritmetika, halmazelmélet, gráfelmélet, stb. Ezek pedig a számítástudomány valamely területén alkalmazhatók.

A dolgozat hat fejezetet ölel fel, nem meríti ki teljesen a diszkrét matematika összes területét.

A fejezetek a következőképpen épülnek fel:

1. A Maple, mint számítógép-algebrai rendszer
2. Alapvető adattípusok
3. Kombinatorika
4. Halmazok generálása
5. Gráfok és fák
6. Lineáris rekurziós egyenletek

A dolgozatom végén található tárgymutató azokat a Maple eljárásokat tartalmazza, amelyeket itt említettem, felhasználtam.

1. A Maple, mint számítógép-algebrai rendszer

Az első szimbólikus számításokat végző számítógépes rendszerek mindössze a hetvenes évek elején jelentek meg. Eleinte ezek a programok valamilyen konkrét matematikai vagy fizikai probléma megoldását támogatták. Később aztán általánosították őket az igényeknek megfelelően. Szimbólikus algebrai algoritmusok születtek és időközben egy új elmélet jött létre, a számítógép-algebra elmélete.

Egy ilyen számítógép-algebrai rendszer a Maple - jelentése juharlevél -, amely a nagypontosságú matematikai számítások elvégzése mellett képes szimbolikus képletkezelésre, feldolgozásra, és a legkülönbözobb két- és háromdimenziós ábrák készítésére.

A Maple szoftver-csomagot 1980-ban kezdték el fejleszteni a kanadai Waterloo Egyetemen, Moreven Gentleman, Michael Malcolm és Frank Tompa informatikusok közreműködésével . Ma már nagyon sok felhasználója van. Sikerét nem utolsó sorban nyitott architektúrájának köszönheti, a rendszer lehetőségeit és "tudását" szinte korlátlanul lehet bővíteni. Ugyanis a Maple kernele és a hozzá kapcsolódó segédprogramok, algoritmusok nagy része C-ben íródott, míg a könyvtári eljárások a Maple saját (C és Pascal keverékére emlékeztető) programnyelvén készültek. Lévén a Maple nyitott rendszer, ezek elolvashatók, és tetszés szerint átírhatók, ha egy speciális probléma ezt úgy kívánja. Ám egy átlagos felhasználónak erre többnyire nincs szüksége, ők a Maple nyelvén írhatnak programokat. Így a Maple lehetőséget ad bárkinek a rendszer könyvtári eljárásainak megváltoztatására, vagy akár új algoritmusok, könyvtárak létrehozására. Nem kell beépített függvényekre szorítkoznunk, amikor a megoldandó probléma matematikai modelljét készítjük. Megírhatjuk saját eljárásainkat, kiterjesztve ezzel a rendszer lehetőségeit feladatok bővebb halmazára.

Külön fontossággal bír a Maple esetében az a tényező, hogy a rendszer dokumentációiban az általános felhasználói dokumentumok mellett, matematikai elméleti összefoglalókat, magyarázatokat is találunk.

A Maple rendszer alkalmazható a matematika legkülönbözőbb ágaiban, az oktatásban, ezenkívül a statisztikában, a mérnöki, üzleti és gazdasági életben modellalkotásra, szimulációra, mérnöki tervezésre, tudományos alkalmazásfejlesztésre.

Olyan felülmúlhatatlan pontosságot és rugalmasságot biztosít, amely korábban elképzelhetetlen volt egy egyszerű számítási környezetben.

A Maple-lel egy munkalapon keresztül lehet kommunikálni, az utasításokat a munkalap parancssorából tudjuk kiadni. Az aktuális parancssor alá írja ki a válaszait a Maple (számítási eredményeket, hibaüzeneteket, eljárásokat), és egy külön ablakba kerülnek az ábrák, animációk.

A Maple összekapcsolható a legfontosabb szoftverekkel, beleértve az Excel 2000-et a Mathlabot, a legtöbb szövegszerkesztő rendszert és nem utolsó sorban a Webet. Ezek a járulékos funkciók még inkább hozzájárulnak a problémamegoldás produktivitásának és hatékonyságának növeléséhez.

1.1. A Maple rendszer alapvető funkciói

A Maple a matematika alkalmazásaival foglalkozó emberek régi álmát valósítja meg, az intelligens rendszert, amely nem csak számokat, hanem képleteket és formulákat is képes kezelni.

Habár a felsőbb matematika szinte minden ágában jól helyt áll, leginkább szimbolikus számítási képességeinek köszönheti hírnevét, pozícióját a matematikai kutatásban. Ismeri a szimbolikus műveletek összes szabályát. A többnapi kézi számolás és munka helyett másodpercek alatt végzi el a feladatokat. Egyértelműen könnyebbé teszi a technikai számításokat, gyorsabbá és hatékonyabbá a modellalkotás folyamatát. A Maple-t alkalmazva nehézségek nélkül értelmezhetjük, megoldhatjuk, tanulmányozhatjuk és optimalizálhatjuk a matematikai problémákat vagy a technikai jellegű adatokat.

Két- és háromdimenziós grafikonjai és animációi segítenek ábrázolni egyes problémákat, például fraktálokat, vektormezőket vagy dinamikus rendszereket. Magas szintű programozási nyelve lehetővé teszi, hogy kiterjesszük és kombináljuk ezeket a vizualizációs eszközöket.

Alapvető funkciói közé sorolhatjuk: numerikus számítások, algebrai számítások, differenciál- és integrálszámítás, lineáris algebra, approximáció, grafika, csoportelmélet, geometria, gráfelmélet, általános relativitáselmélet, statisztikai számítások, animáció.

A szoftvercsomag alkalmas *aritmetikai kifejezések kezelésé*re, így ismeri az alapműveletek mellett a gyökvonást (az n-edik gyököt is), tudja kezelni a komplex alapfüggvényeket, a Gauss-egészeket, a közelítéseket, nem folytonos függvényeket (pl. Dirac-delta függvény). Azt is meg lehet adni, hogy a függvények szakadási helyeit, pólusait hogy kezelje.

A határozott *integrálok* elvégzése gyakorlatilag tetszőleges pontosságig lehetséges, a határozatlan integrálok kiszámításánál felhasználja a beépített függvényeit (Bessel, Fresnel, elliptikus , hipergeometrikus függvények). Olyan esetekben is ad megoldást, amikor "elemi" függvényekkel esetleg nem írható fel a határozatlan integrál. Improprius, paraméteres és több dimenziós integrálokat is tud kezelni.

Tetszőleges, maximum 4-ed fokú polinom zérushelyeit meg tudja találni pontosan, speciális esetekben magasabb rendűekét is. Transzcendens egyenletek, magasabbfokú polinomok esetén a megoldást formálisan kezeli, mint az adott egyenlet gyökeit, amelyek a későbbi számolásokban felhasználhatók. Numerikus megoldásra természetesen mindig van lehetőség.

A komolyabb matematikai problémák megoldásakor szinte kikerülhetetlen, hogy ne kelljen differenciálni, vagy integrálni. A Maple beépített funkciói ezeket az akadályokat is könnyedén veszi, megszabadítva a felhasználót a hosszú és időigényes

számításoktól. Természetesen a *közönséges differenciálegyenletek megoldása* sem hiányzik a program kelléktárából. Általában sikeresen bírkózik meg az inhomogén egyenletekkel is. Ha nem talál zárt alakban kifejezhető megoldást, a rendelkezésre álló numerikus megoldási módszerekre támaszkodik. a megoldást pedig grafikusan ábrázolni tudja. A legújabb verziókban már *parciális differenciálegyenlete*ket is meg lehet oldani és ugyancsak ábrázolni.

A *lineáris algebra* terén a vektorok és mátrixok kezelése is adott a Maple-ben. Mind az egyszerűbb (összeadás, szorzás, determináns vagy inverzmátrix-számítás, skalárszorzat, vektorszorzat, vektor hossza, két vektor szöge), mind pedig a bonyolultabb műveletek (sajátérték vagy sajátvektor-számítás, karakterisztikus polinom felírása, képterek, magterek meghatározása, Jordán felbontás, Hermite normál alak, Gram-Schmidt ortogonalizáció) rendelkezésre állnak.

A Maple ismeri a következő *közelítő eljárások*at: polinom-illesztés (Lagrange), spline interpoláció, Padé-approximáció, Csebisev-Padé approximáció, vagy minimax közelítés. Minden esetben megadja a közelítések pontosságát, relatív hibáját és szórását. Egy függvényt Taylor-, Laurent-, aszimptotikus- és ortogonális polinomok szerint haladó sorba tud fejteni.

Maple-ben a legegyszerűbb *függvényábrázolás*i lehetőségeken kívül könnyen lehet tetszőleges alakzatot kirajzolni. A függvényeket megadhatjuk Descartes-féle koordinátákban, polár-koordinátákban és paraméteres alakban is. A grafikonokat az egér segítségével elforgathatjuk, ha nem lennénk megelégedve a nézőponttal.

Külön utasítás van poligonok, síkbeli vektormezők, implicit függvények kirajzolására. Lehetőség van kétváltozós függvények szintvonalainak meghatározására. Külön opciókat használhatunk a színek kezelésére, vonalak vastagságának megadására, nyilak, tengelyek pontos kinézetére, az ábrák feliratozására. A kész grafikákat el lehet menteni post-script vagy gif formátumban.

*Három dimenzió*ban ismeri a Maple a felületek, térgörbék Descartes-koordinátás, polár-koordinátás, henger-koordinátás és paraméteres megadását. A kirajzolt felület vagy térgörbe színét, árnyékolását, kirajzolási módját (hálós, teli, szintvonalas, stb.), rálátási szögét, nyújtási arányait meglehet változtatni. A térgörbéket körül lehet venni akár változó vastagságú csővel, így téve őket térhatásúvá. Egyetlen utasítással előhívhatók a gyakran használt testek: tetraéder, hexaéder, oktaéder, dodekaéder és ikozaéder.

Három dimenzióban is lehet implicit függvényeket, vektormezőket ábrázolni. Egy ábrán tetszőleges számú felületet, görbét, vektormezőt vagy testet lehet elhelyezni.

Több lehetőségünk van diszkrét csoportok definiálására. Megadhatjuk konkrétan a csoportszorzási, inverzképzési műveleteket, és az egységelemet. Egy másik lehetőség a csoport generátorelemekkel és relációkkal való megadása. Tetszőleges (diszkrét) csoportra elvégzi a következőket: megkeresi egy elem rangját, eldönti, hogy egy részcsoport normálosztó-e, megkeresi egy részcsoport normalizátorát, meghatározza egy elem orbitját, reprezentálja a csoportot, mint permutációcsoportot, stb. Ezen kívül további operációkat ismer permutációcsoportokra: eldönti, hogy két

permutáció egymásnak konjugáltja-e, megkeresi a permutációcsoport centrumát, permutációk egy halmazának megkeresi a centralizátorát, permutációcsoportok metszetét határozza meg, eldönti, hogy a permutációcsoport Abel-féle-e.

Maple-ben *kétdimenziós euklideszi geometriá*val lehet dolgozni. A következő objektumokat ismeri: pont, szakasz, irányított szakasz, egyenes, háromszög, négyzet, kör, ellipszis, parabola és hiperbola. El lehet dönteni egyenesek párhuzamosságát, merőlegességét, síkidomok egybevágóságát, hasonlóságát. Többek közt a következő transzformációkat ismeri: eltolás, forgatás, nyújtás, nyújtva forgatás, inverzió. Viszont nem tud dolgozni végtelen távoli ponttal és végtelen távoli egyenessel.

Tetszőleges *gráf*ot lehet definiálni a Maple-ben akár többszörös-, irányítottvagy hurokéllel is. Egyetlen utasítással előhívhatók a nevezetesebb gráfok. Lehetőség van véletlen gráf értelmezésére. Ismeri az alapvető gráfelméleti fogalmakat , egyszerűen megoldhatók az alábbi problémák: komplementer gráf megadása, csúcsok összekötése, adott gráfhoz tartozó mátrixok felírása, a legrövidebb kör megkeresése, síkbarajzolhatóság ellenőrzése.

Az általános *relativitáselmélet*ben használt tenzorokat könnyen kezelhetjük.A Maple ismeri az alapvető fogalmakat,mint a metrika, görbület, kovariáns deriválás, párhuzamos eltolás, geodetikus, koordináta transzformáció.

A statikus függvények ábrázolása mellett képes még adathalmazok megjelenítésére vagy *animáció*ra, alig változó képek egy sorozatának készítésére. Külön ablakban lejátszható a kész animációt, amely lehet 2, vagy 3 dimenziós is. Ennek köszönhetően figyelemmel kísérhetők az időben és térben egyaránt változó folyamatok.

1.2. Függvényeket tartalmazó csomagok

Az előbbi részben említett problémák megoldásához különböző függvényeket, más néven eljárásokat használunk.

A Maple rendszer eljárásainak egy része a rendszer betöltésével egyidejűleg rendelkezésre áll. Más részük, a nem könyvtári eljárások, ún. csomagokban (package) helyezkedik el, és ezen eljárások esetén nemcsak a függvény nevét, hanem a csomag nevét is meg kell adni a rendszernek. Példa csomagokra:

combinat kombinatorikai függvény csomag
combstruct kombinatorikai struktúra csomag
DEtools differenciálegyenletek függvény csomag
GaussInt a Gauss egészek csomagja
geometry geometria csomag
group csoport csomag
linalg lineáris algebra csomag
logic logikai csomag
networks hálózatok csomag
numapprox approximációs számítások csomagja
stats statisztika csomag

2. Alapvető adattípusok

```
alma, szilva, eper
[1,3,5,2,1,x,x,3,a]
    {x,y,z,a,b}
```

A Maple nyelv lehetőséget ad a diszkrét matematika széleskörű problémáinak tanulmányozására.

A rendszerrel való munka során gyakran találkozunk a sorozat, lista ,halmaz vagy tömb objektumokkal.

2.1. Sorozat, lista, halmaz

A sorozatok gyakran használt struktúrák az információ ábrázolására. Amikor utasí-tást adunk ki oly módon, hogy a meghívandó eljárás neve mögé zárójelbe felsoroljuk az aktuális paramétereket, vesszővel elválasztva, akkor a paraméterek egy sorozatát adjuk át az eljárásnak. Például:

```
> sorozat1:=2,1,4,4,6,2,4,0;
```

$$sorozat1 := 2, 1, 4, 4, 6, 2, 4, 0$$

```
> sorozat2:=seq(2*i-1,i=1..10) ;
```

$$sorozat2 := 1, 3, 5, 7, 9, 11, 13, 15, 17, 19$$

Lekérdezhetjük a sorozat egy adott elemét (vagy elemeit):

```
> sorozat1[5] ;
```

$$6$$

Ha egy sorozatot szögletes zárójelek közé zárunk, akkor listát, ha pedig kapcsos zárójelek közé zárunk, akkor halmazt kapunk.

A Maple a halmaz típusú objektumot elemei rendezetlen összességének tekinti, az elemek ismétlődését nem veszi figyelembe, a sorrendet pedig a rendszer határozza meg. Ezzel szemben a lista rendezett elemek összessége, ahol bármelyik elem ismétlődését a Maple megkülönbözteti. A listának lehetnek azonos elemei, a halmazban egy objektum legfennebb egyszer fordul elő.

Az üres lista jele [ ], az üres halmazé pedig {}.

2.2. Tábla

A rendszer gyakran használja a tábla adattípust, sajátos esetben pedig ennek valamelyik speciális esetét:a tömböt, a mátrixot vagy a vektort.

Táblát létrehozni a $T := table([index1 = elem1, index2 = elem2, ...])$ utasítással lehet. Ha nem adunk meg paramétert, akkor üres táblát kapunk, ha pedig egyenlőségek helyett kifejezések listája a table() paramétere, akkor a rendszer az 1,2,... természetes számokat rendeli hozzá az elemekhez indexként.

A tömb specializált tábla, olyan tábla, ami kiegészül dimenzióinak határaival. Létrehozására az array eljárást használjuk.Először a dimenziók határait, majd a tömb elemeit kell megadnunk a függvényben. Ha kétdimenziós a tömb, és alsó határa 1, mátrixot kapunk, ha viszont egydimenziós, akkor vektorról beszélünk.

```
> A:=array(1..2,1..3,[[1,3,0],[2,0,9]]);
```

$$A := \begin{bmatrix} 1 & 3 & 0 \\ 2 & 0 & 9 \end{bmatrix}$$

Mátrix létrehozható a linalg csomag matrix eljárásával is.

2.3. Típuskonverziók

Sorozatból listát és halmazt könnyen hozhatunk létre.

Tekintsük az alábbi objektumokat:

```
> lista:=[sorozat1];
```

$$lista := [2,\ 1,\ 4,\ 4,\ 6,\ 2,\ 4,\ 0]$$

```
> halmaz:={sorozat1};
```

$$halmaz := \{0,\ 1,\ 2,\ 4,\ 6\}$$

Hogyan konvertáljunk listát és halmazt sorozattá, illetve listát halmazzá, valamint a halmazt listává? Az op() és a convert() eljárásokkal. Az op() eljárásnak egy argumentuma van, ennek állítja elő az összetevőit, a convert() első paramétere az átalakítandó struktúra, a második paramétere pedig az a forma, amivé az első paramétert konvertálni kell.

Végezzünk a fenti objektumokkal átalakításokat egyik típusból a másikba.

```
> lista; op(lista); convert(lista,set);
```

$$[2,\ 1,\ 4,\ 4,\ 6,\ 2,\ 4,\ 0]$$

$$2,\ 1,\ 4,\ 4,\ 6,\ 2,\ 4,\ 0$$

$$\{0,\ 1,\ 2,\ 4,\ 6\}$$

```
> halmaz; op(halmaz); convert(halmaz,list);
```

$$\{0,\ 1,\ 2,\ 4,\ 6\}$$

$$0,\ 1,\ 2,\ 4,\ 6$$

$$[0,\ 1,\ 2,\ 4,\ 6]$$

A convert() eljárással természetesen vektort, tömböt, vagy táblát is alakíthatunk akár halmazzá, akár listává.

3. Kombinatorika

```
                    1
                  1,1
                1,2,1
              1,3,3,1
            1,4,6,4,1
          1,5,10,10,5,1
        1,6,15,20,15,6,1
      1,7,21,35,35,21,7,1
```

A Maple rendszer a kombinatorika terén elvégzendő számításokhoz szükséges eljárásokat a combstruct és a combinat csomagokban őrzi. A csomagok betöltése a következő utasítássorral történik és minden munkalap elején el kell végezni:

```
> with(combinat);
```

```
Warning, new definition for Chi
```

[Chi, *bell*, binomial, *cartprod*, *character*, *choose*, *composition*, *conjpart*, *decodepart*, *encodepart*, *fibonacci*, *firstpart*, *graycode*, *inttovec*, *lastpart*, *multinomial*, *nextpart*, *numbcomb*, *numbcomp*, *numbpart*, *numbperm*, *partition*, *permute*, *powerset*, *prevpart*, *randcomb*, *randpart*, *randperm*, *stirling1*, *stirling2*, *subsets*, *vectoint*]

```
> with(combstruct);
```

[*allstructs*, *count*, *draw*, *finished*, *gfeqns*, *gfseries*, *gfsolve*, *iterstructs*, *nextstruct*]

3.1. Permutációk

A kombinatorika leggyakoribb feladatai közé tartoznak a sorba rendezési problémák. A gyakorlatban előforduló sorba rendezési típusoknak külön nevet szoktak adni, két ilyen alaptípust képviselnek a permutációk és a variációk.

1. Értelmezés. *n különböző elem egy bizonyos sorrendjét az n elem egy permutációjának nevezzük.*

Az elemek különbözőségére gyakran az ismétlés nélküli permutációk elnevezéssel utalunk.

n elem összes permutációinak száma $n!$.

2. Értelmezés. *Legyen adott n elem, melyek közül $k_1, k_2, \ldots, k_r$ rendre egyforma($k_1 + k_2 + k_3 + ...k_r = n$). Ezen elemek egy rögzített sorrendjét egy ismétléses permutációnak nevezzük.*

n elem ismétléses permutációinak száma

$$\frac{n!}{k_1!.k_2!.k_3!...k_r!}$$

.

Hasznos, ha ismerünk néhány dolgot a faktoriálisról, mint például azt a tényt, hogy 10! körülbelül három és fél milliónál több választást jelent, valamint azt, hogy n! jegyeinek a száma meghaladja n-et, ha n=25.

n elem permutációinak számát Maple-ben a numbperm() eljárással határozhatjuk meg.

```
> numbperm(6);
```

720

A numbperm() függvény argumentuma halmaz is lehet:

```
> numbperm({A,B,C,D,E,F,G,H,J,K});
```

3628800

Észrevehető, hogy a numbperm eljárásban egy n elemű halmaz helyettesíthető magával az n természetes számmal.

Megszerkeszthetjük egy halmaz, vagy lista elemeinek összes permutációit a permute() eljárás segítségével.

```
> permute({1,2,3});
```

[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]

Vegyük észre, hogy a Maple kimenetele egy sorozat.

A permute() paramétere lehet ismétlődő elemeket tartalmazó lista is.

```
> permute([k,r,o,k,o,d,i,l]): numbperm([k,r,o,k,o,d,i,l]);
```

10080

Hogy meggyőződjünk az eredmény helyességéről, felhasználjuk az ismétléses permutációk számát.

```
> 8!/(2!*2!);
```

10080

Egy n elemű halmaz véletlenszerűen generált permutációját a randperm() függvény adja meg.

```
> randperm({m,a,p,l,e});
```

$[p, l, m, e, a]$

3.2. Kombinációk

3. Értelmezés. *n különböző elem egy k-ad osztályú kombinációján valamely k elem egyszerre történő kiválasztását értjük (a kiválasztás sorrendje nem számít).*

n elem k-ad osztályú kombinációinak száma C_n^k.

4. Értelmezés. *Ha az n elem közül oly módon választunk ki k darabot, hogy egy elem többször is szerepelhet és a sorrendre nem vagyunk tekintettel, akkor az n elem egy k-ad osztályú ismétléses kombinációjáról beszélünk.*

n-féle elem k-ad osztályú ismétléses kombinációinak száma C^k_{n+k-1}.

A numbcomb() eljárást használhatjuk két paraméterrel is. Ekkor n elem k-ad osztályú kombinációinak számát adja meg a Maple.

```
> numbcomb( {Maria,Eva,Istvan,Peter,Janos,Zoltan},2);
```

15

3.3. Variációk

5. Értelmezés. *n különböző elem közül kiválasztott rendezett k elemet ismétlés nélküli k-ad osztályú variációnak nevezzük.*

n elem k-ad osztályú variációinak száma $\frac{n!}{(n-k)!}$.
Az ismétlés nélküli variációknál természetesen $k \leq n$.

6. Értelmezés. *Ha n különböző elemből k elemet oly módon választunk ki, hogy egy elemet többször is választhatunk és a sorrend számít, akkor n elem k-ad osztályú ismétléses variációjáról beszélünk.*

n-féle elem k-ad osztályú ismétléses variációinak száma n^k.

A Maple nem rendelkezik külön eljárásokkal a variációkat illetően, ugyanis a már említett permute() és numbperm () függvények erre kiválóan alkalmasak, azzal a kikötéssel, hogy most két opciót kell megadnunk: az első egy lista, halmaz, vagy természetes szám lesz, a második pedig meghatározza, hogy hányad osztályú variációt tekintsen.

```
> permute([alpha,beta,gamma,delta],3);
```

[[α, β, γ], [α, β, δ], [α, γ, β], [α, γ, δ], [α, δ, β], [α, δ, γ], [β, α, γ], [β, α, δ], [β, γ, α], [β, γ, δ], [β, δ, α], [β, δ, γ], [γ, α, β], [γ, α, δ], [γ, β, α], [γ, β, δ], [γ, δ, α], [γ, δ, β], [δ, α, β], [δ, α, γ], [δ, β, α], [δ, β, γ], [δ, γ, α], [δ, γ, β]]

Az alábbi kis eljárások az ismétlés nélküli, illetve ismétléses variációk számát határozzák meg.

```
> varszam:=proc(m,n)
local s,i:
if n>m then s:=0 else s:=1:
for i from 0 to n-1 do s:=s*(m-i): od:
fi: s end:
> varszam(5,3);
```

60

```
> ismvarszam:=(m,n)->m^n;
```

$$ismvarszam := (m, n) \rightarrow m^n$$

```
> ismvarszam(3,5);
```

243

Eljárás az összes variáció szemléltetésére:

```
> variaciok:=proc(A,n)
local C,D,i:
C:=[op(choose(A,n))]:
D:={}:
for i from 1 to nops(C) do
D:=D union {op(permute([op(C[i])]))}: od:
D end:
> variaciok({a,b,c,d},3);
```

{[*d*, *a*, *c*], [*b*, *a*, *c*], [*b*, *c*, *a*], [*c*, *a*, *b*], [*c*, *b*, *a*], [*a*, *b*, *c*], [*a*, *b*, *d*], [*a*, *c*, *d*], [*c*, *a*, *d*], [*c*, *d*, *a*], [*b*, *c*, *d*], [*a*, *c*, *b*], [*a*, *d*, *b*], [*b*, *a*, *d*], [*b*, *d*, *a*], [*d*, *a*, *b*], [*d*, *b*, *a*], [*a*, *d*, *c*], [*d*, *c*, *a*], [*b*, *d*, *c*], [*c*, *b*, *d*], [*c*, *d*, *b*], [*d*, *b*, *c*], [*d*, *c*, *b*]}

3.4. Binomiális és polinomiális tétel

1. Tétel (polinomiális). *Legyen* $\forall a_1, a_2, \ldots, a_k \in \mathbf{R}$*, ahol* $\mathbf{R}$ *kommutatív gyűrű és n egynél nagyobb természetes szám. Ekkor*

$$(a_1 + a_2 + \cdots + a_k)^n = \sum_{s_1+s_2\cdots+s_k=n} \frac{n!}{s_1!s_2!\ldots s_k!} {a_1}^{s_1}{a_2}^{s_2}\ldots{a_k}^{s_k}.$$

2. Tétel (binomiális). *Legyen* $\forall a_1, a_2 \in \mathbf{R}$*, ahol* $\mathbf{R}$ *kommutatív gyűrű és n egynél nagyobb természetes szám. Ekkor*

$$(a_1 + a_2)^n = \sum_{s_1=0}^{n} \binom{n}{s_1} {a_1}^{s_1}{a_2}^{n-s_1}.$$

Az $\binom{n}{k}$ szimbólumot binomiális együtthatónak is nevezzük.

Maple-ben a binomial() függvény határozza meg a binomiális együtthatókat, amint a neve is mutatja.

```
> binomial(6,2);
```

15

Faktoriális segítségével is felírhatjuk a binomiális együtthatót. Ehhez a convert() függvényt kell használnunk.

```
> convert(binomial(m,n),factorial);
```

$$\frac{m!}{n!\,(m-n)!}$$

A polinomiális együtthatót, amely a tétel szerint az ismétléses permutációk számával egyezik meg a multinomial() eljárás adja meg.

```
> multinomial(8,2,2,1,1,1,1);
```

10080

A convert() függvénnyel szemléltetni lehet a polinomiális formulát is.

```
> convert(multinomial(n,a,b,c,d),factorial);
```

$$\frac{n!}{a!\,b!\,c!\,d!}$$

Részhalmaz-kiválasztási problémák a choose() függvénnyel oldhatók meg .

```
> choose({Maria,Eva,Istvan,Peter,Janos,Zoltan },2);
```

$\{\{Eva, Maria\}, \{Zoltan, Maria\}, \{Zoltan, Eva\}, \{Janos, Maria\}, \{Janos, Eva\}, \{Janos, Zoltan\}, \{Peter, Maria\}, \{Peter, Eva\}, \{Peter, Zoltan\}, \{Peter, Janos\}, \{Istvan, Maria\}, \{Istvan, Eva\}, \{Istvan, Zoltan\}, \{Istvan, Janos\}, \{Istvan, Peter\}\}$

Véletlenszerűen is generálhatunk részhalmazt. Ezt a randcomb() eljárás teszi lehetővé.

```
> randcomb({alma,korte,szilva,banan},2);
```

$$\{alma, korte\}$$

```
> randcomb(5,3);
```

$$\{1, 3, 5\}$$

Ismétléses kombináció esetén a choose() első argumentuma a lista kell hogy legyen.

```
> choose([a,b,a],2);
```

$$[[a, a], [a, b]]$$

Ebben az esetben az ismétléses kombinációk számát a numcomb() eljárással kérdezhetjük le, szintén listának megadva az első paramétert.

```
> numbcomb([a,b,a],2);
```

$$2$$

3.5. A combstruct csomag használata

A combstruct csomagot kombinatórikai struktúrák értelmezésére használjuk. Segít-ségével kombinatorikai osztályok, objektumok definiálhatók.

A combstruct csomag a combinat csomag függvényeinek kiterjesztését teszi lehetővé.

3.5.1. Kombinatorikai struktúrák nyelvtani leírásai

A rendszer erőssége abban áll, hogy saját kombinatórikai struktúrák megadását teszi lehetővé.

Egy kombinatorikai osztályt nyelvtani leírással (specifikációval) adunk meg. A specifikáció produkciók halmaza, formája: **A=jobb**, ahol **A** az osztály neve, **jobb** pedig egy kifejezés,amely elemi osztályokat (Atom, Epsilon), konstruktorokat (Union, Prod, Set, Sequence, Cycle, Subst) és egyéb osztályokat foglal magába.

A Maple alkalmazni tud reguláris-, környezetfüggetlen nyelvtanokat, bináris fákat értelmező nyelvtanokat, általános fákat, nyakláncokat, kifejezési fákat, stb.

Miután értelmeztünk egy osztályt, a draw() paranccsal véletlenszerűen generálhatunk is egy adott méretű objektumot, illetve megkaphatjuk az összes objektum számát a count() függvénnyel.

Most pedig értelmezzünk egy bináris fát. A Union és Prod konstruktorok segítségével a következőképpen járunk el:

```
> binfa := {B = Union(Z, Prod(B,B))};
```

$$binfa := \{B = \text{Union}(Z, \text{Prod}(B, B))\}$$

A bináris fa állhat csupán egy levélből, ekkor a mérete 1 (size=1), vagy (a Union jelentése itt vagy) két bináris fából, amelyeket a Prod kapcsol a gyökérhez.

Generáljunk 5 méretű bináris fát.

```
> draw([B, binfa], size=5);
```

$$\text{Prod}(\text{Prod}(\text{Prod}(Z, Z), Z), \text{Prod}(Z, Z))$$

Az atomokat minősíthetjük is, azaz megkülönböztethetjük egymástól. Ekkor a **labelled** opciót kell használnunk. Az előbbi példánk esetén kapjuk:

```
> draw([B, binfa, labelled], size=5);
```

$$\text{Prod}(\text{Prod}(Z_3, \text{Prod}(Z_5, \text{Prod}(Z_4, Z_1))), Z_2)$$

```
> count([B, binfa, labelled], size=5);
```

$$1680$$

```
> count([B, binfa, unlabelled], size=5);
```

$$14$$

A rendszerbe beépített atom jele Z, de saját magunk is értelmezhetünk atomokat.

Felvetődhet a következő megszámlálási probléma: n különböző gyöngyünk van, és ki akarjuk számítani, hogy hány különböző módon fűzhetjük fel őket ahhoz, hogy m hosszúságú kör alakú nyakláncot kapjunk.

A következő struktúra például egy olyan nyakláncot definiál, amely három különböző színből áll.

```
> nyaklanc:= {N=Cycle(Union(piros,kek,zold)),piros=Atom,kek=Atom,
zold=Atom}:
> draw([N,nyaklanc,unlabelled], size=10);
```

$$\text{Cycle}(piros, piros, zold, piros, piros, zold, zold, zold, zold, kek)$$

Minden atomnak van súlya. Epsilon (jele E) súlya 0, a Z atom súlya pedig 1.

Az előbb adott értelmzésünk a bináris fára nem engedte meg az üres fát. Az alábbi definíció ezt lehetővé teszi.

```
> binfa2:= {T = Union(Epsilon, B), B=Union(Z, Prod(Z,Z))}:
draw([T, binfa2, unlabelled], size=0);
```

$$E$$

A Set, Cycle és Sequence konstruktorok használatakor megadhatjuk a számosságukat is. Néhány példa:

```
> count([M, {M=Set(Z, card > 8)}, labelled], size=7);
```

$$0$$

```
> draw([A, {A=Cycle(Z, card=4)},labelled],size=4);
```

$$\text{Cycle}(Z_1, Z_2, Z_4, Z_3)$$

```
> count([A, {A=Cycle(Z, card=4)},labelled],size=3);
```

$$0$$

```
> draw([S, {S=Sequence(Set(Z, card>0), card <=10)}, labelled],
size=6);
```

$$\text{Sequence}(\text{Set}(Z_5, Z_2, Z_1), \text{Set}(Z_3), \text{Set}(Z_6, Z_4))$$

```
> count([S, {S=Sequence(Z, card <=10)}, labelled], size=13);
```

$$0$$

A Subst egy mechanizmus, amely egy objektum összes atomjának helyettesítését teszi lehetővé egy más objektummal. Subst(A,B) jelentése:végy egy B-objektumot és bármely atomját helyettesítsd egy A-objektummal.

A Subst konstruktor használható 2-3 fák rekurzív értelmezésére is.

7. Értelmezés. *Egy 2-3 fa olyan fa, amelynek végződései (levelei) egy szinten találhatók, és bármely közbelső csúcs pontosan két vagy három fiút tartalmaz.*

Úgy fogjuk értelmezni a fánkat, hogy vagy egyetlen levélből áll, vagy 2-3 fa, amelynek bármely levelét olyan közbelső szögponttal helyettesítjük, ami két, illetve három fiút tartalmaz.

```
> fa23 := {T = Union(Z, Subst(Union(Prod(Z,Z), Prod(Z,Z,Z)), T))
}:
> draw([T, fa23], size=11);
```

Prod(Prod(Prod(Z, Z), Prod(Z, Z)), Prod(Prod(Z, Z), Prod(Z, Z), Prod(Z, Z, Z)))

3.5.2. Előre-definiált struktúrák

Bizonyos kombinatorikai struktúrák annyira használatosak, hogy speciális algoritmusokat érdemelnek a sokkal általánosabb nyelvtani módszerek helyett.

A combstruct csomag a következő előre-definiált struktúrákkal rendelkezik:

Combination - elemek kombinációi (Subset nevet is viseli)

Permutation - elemek permutációi és variációi

Partition - egészek partíciói (sorrendtől függetlenül)

Composition - egészek kompozíciói (számít a sorrend)

A függvények, amelyeket velük kapcsolatosan használhatunk, a következők: draw(), count(), allstructs(), iterstructs().

Akárcsak a nyelvtan által értelmezett struktúrák, szemléltethetők és megszámolhatók.

```
> draw(Combination({a,b,c,d,e,f,g}), size=5);
```

$$\{a, b, f, c, d\}$$

```
> count(Partition(95), size=40);
```

$$450768$$

Az összes függvény, amely a fent említett struktúrákkal dolgozik, ugyanazzal a szintaxissal rendelkezik: az első argumentum az értelmezett struktúra, a második a mérete. A **Partition** és a **Composition** struktúrák egészeket, míg a **Combination** és **Permutation** listákat, halmazokat vagy egészeket fogad el argumentumként. Utóbbi esetben egy n egész szám a **Combination** esetén az {1,...,n}

halmazt, a **Permutation** esetében pedig az [1,...,n] listát fogja jelenteni. A **size** opció el is hagyható (ekkor a **default** értéket rendeli hozzá a rendszer), vagy **allsizes** értéket is felvehet.

Az elmondottak szemléltetésére lássunk néhány példát:

```
> draw(Combination({a,b,c,d,e,f,g}));
```

$$\{a, b, c, e\}$$

```
> draw(Permutation(42));
```

[26, 3, 6, 16, 31, 30, 8, 29, 22, 13, 32, 37, 5, 20, 28, 9, 42, 17, 33, 24, 7, 2, 36, 12, 4, 10, 15, 19, 41, 25, 18, 11, 14, 38, 35, 34, 21, 27, 40, 23, 1, 39]

```
> draw(Permutation(16),size='allsizes');
```

[13, 6, 3, 2, 14, 4, 5, 15, 8, 9, 7, 12]

```
> draw(Composition(32));
```

[1, 3, 1, 1, 1, 3, 2, 6, 1, 4, 3, 2, 1, 1, 1, 1]

Az allstructs() függvény adott méretű struktúrák generálására használatos.

```
> allstructs(Combination(4));
```

{{}, {1}, {2, 3, 4}, {3, 4}, {1, 3, 4}, {4}, {1, 4}, {2, 4}, {1, 2, 4}, {2}, {1, 2}, {3}, {1, 3}, {2, 3}, {1, 2, 3}, {1, 2, 3, 4}}

```
> allstructs(Permutation([a,a,b,c]),size=3);
```

[[a, a, b], [a, a, c], [a, b, a], [a, b, c], [a, c, a], [a, c, b], [b, a, a], [b, a, c], [b, c, a], [c, a, a], [c, a, b], [c, b, a]]

```
> allstructs(Partition(4));
```

[[1, 1, 1, 1], [1, 1, 2], [2, 2], [1, 3], [4]]

```
> allstructs(Composition(6));
```

[[1, 1, 2, 2], [1, 1, 1, 2, 1], [1, 2, 1, 1, 1], [2, 1, 1, 1, 1], [1, 1, 2, 1, 1], [1, 1, 1, 1, 2], [1, 1, 1, 3], [1, 1, 1, 1, 1, 1], [2, 1, 1, 2], [3, 1, 2], [2, 2, 2], [1, 3, 2], [1, 2, 3], [2, 1, 3], [1, 1, 4], [3, 1, 1, 1], [2, 2, 1, 1], [1, 3, 1, 1], [1, 2, 2, 1], [2, 1, 2, 1], [1, 1, 3, 1], [1, 2, 1, 2], [3, 3], [2, 4], [1, 5], [3, 2, 1], [2, 3, 1], [1, 4, 1], [4, 1, 1], [6], [5, 1], [4, 2]]

Az iterstructs() függvény létrehoz egy mechanizmust, amely az összes adott méretű struktúrát megadja, egyenként. A nextstruct() és finished() eljárásokat akkor használjuk, ha az iterstructs() által visszaadott tábla elemeivel szeretnénk dolgozni.

```
> it := iterstructs(Combination([a, o, b]), size=2):
while not finished(it) do nextstruct(it) od;
```

$$[a, b]$$
$$[a, o]$$
$$[b, o]$$

```
> it := iterstructs(Permutation(3), size='allsizes'):
while not finished(it) do nextstruct(it) od;
```

[]
[1]
[2]
[3]
[1, 2]
[2, 1]
[1, 3]
[3, 1]
[2, 3]
[3, 2]
[1, 2, 3]
[2, 1, 3]
[3, 1, 2]
[1, 3, 2]
[2, 3, 1]
[3, 2, 1]

A **PowerSet** konstruktor ismétlődő elemeket nem tartalmazó halmaz a **Set**-tel ellentétben, amelyben az elemek ismétlődhetnek.

A **PowerSet** alkalmazása egészek nem egyenlő számokból alkotott particionálására:

```
> s := {L = PowerSet(Sequence(Z,card>=1)) },unlabelled;
```

$s := \{L = \text{PowerSet}(\text{Sequence}(Z,\ 1 \leq \mathit{card}))\},\ \mathit{unlabelled}$

```
> draw([L,s],size=5);
```

PowerSet(Sequence(Z, Z, Z, Z, Z))

```
> seq(count([L,s],size=i),i=1..10);
```

1, 1, 2, 2, 3, 4, 5, 6, 8, 10

3.5.3. Generátorfüggvények

8. Értelmezés. *Egy végtelen* $(a_n)_{n\geq 0} =< a_0, a_1, a_2, \ldots, a_n, \ldots >$ *sorozat egy* z *segédváltozó segítségével hatványsorként ábrázolható:*

$$A(z) = a_0 + a_1 z + a_2 z^2 + \cdots = \sum_{k\geq 0} a_k z^k,$$

amelyet az $(a_n)_{n\geq 0}$ *sorozat generátorfüggvényének nevezünk.*

A generátorfüggvény azért nagyon hasznos, mert egyetlen mennyiség, aminek segítségével egy egész végtelen sorozatot ábrázolni lehet.

Nyelvtan által leírt objektumok számának meghatározására három generátorfüggvény-eljárás áll rendelkezésünkre. Ha adott egy z változó, bármely A nemterminális az A(z) generátorfüggvénnyel lesz társítva, ahol

```
> A(z)=Sum(a[n]*z^n,n=0..infinity):
```

a[n] az A-ban levő n méretű objektumok száma.

Tekintsük az alább értelmezett nyakláncot:

```
> lanc := {N=Cycle(gyongy),gyongy=Union(piros,kek,zold),piros=
Atom, kek=Atom,zold=Atom };
```

$lanc := \{gyongy = \mathrm{Union}(piros,\ kek,\ zold),\ piros = Atom,\ N = \mathrm{Cycle}(gyongy),$
$kek = Atom, zold = Atom\}$

A gfeqns() függvény visszaadja egy nyelvtan generátorfüggvényeinek a rendszerét, anélkül, hogy megoldaná. Ebben az esetben:

```
> gfeqns(lanc,unlabelled,z);
```

$$\left[\mathrm{N}(z) = \sum_{j_1=1}^{\infty} \frac{numtheory_{\phi}(j_1)\ln(\frac{1}{1-\mathrm{gyongy}(z^{j_1})})}{j_1},\ \mathrm{kek}(z) = z,\ \mathrm{piros}(z) = z,\right.$$

$$\left.\mathrm{zold}(z) = z, \mathrm{gyongy}(z) = \mathrm{piros}(z) + \mathrm{kek}(z) + \mathrm{zold}(z)\right]$$

A gfsolve() segítségével meg is oldhatjuk.

```
> gfsolve(lanc,unlabelled,z);
```

$$\left\{\mathrm{blue}(z) = z,\ \mathrm{piros}(z) = z,\ \mathrm{zold}(z) = z,\ \mathrm{N}(z) = \sum_{j_1=1}^{\infty} \frac{numtheory_{\phi}(j_1)\ln(-\frac{1}{-1+3\,z^{j_1}})}{j_1},\right.$$

$$\left.\mathrm{gyongy}(z) = 3\,z\right\}$$

Mindenik generátorfüggvényre megtalálhatjuk a hozzá rendelt sort a gfseries() eljárással. Eredményül egy tábla adattípust kapunk.

```
> gfseries(necklace,unlabelled,z);
```

table([
$\mathrm{piros}(z) = z$
$\mathrm{gyongy}(z) = 3\,z$
$\mathrm{zold}(z) = z$
$\mathrm{N}(z) = 3\,z + 6\,z^2 + 11\,z^3 + 24\,z^4 + 51\,z^5 + \mathrm{O}(z^6)$
$\mathrm{kek}(z) = z$
])

A gfsolve() nem mindig talál választ.Ilyenkor FAIL-t ad eredményül.

3.5.4. Alkalmazás

Feladat: Határozzuk meg az n+2 oldalú sokszög n darab háromszögre való szétdarabolását az illető sokszög nem metsző n-1 átlója mentén.

Megoldás: Rendeljünk 1-től n+2-ig terjedő számokat a sokszög csúcsaihoz.

A háromszögelés lépései : Kiválasztunk egy alap háromszöget, amely létezése egyértelmű olyan értelemben, hogy a legkisebb számokkal illetett csúcsokat tartalmazza (kezdetben 1,2). Az alap-háromszöghöz képest bal- illetve jobb- háromszögelést végzünk.

A sokszög háromszögelését a combstruct csomag segítségével a következőképpen oldhatjuk meg: A Z atom az alap-háromszöget fogja jelenteni, T pedig egy tetszőleges háromszögelést. Figyelembe kell vennünk, hogy a bal-, illetve a jobb- háromszögelés üres is lehet.

A nyelvtanunk tehát így néz ki:

```
> haromszog:=[T, {T=Union(Z,Prod(Epsilon,Z,T),Prod(T,Z,Epsilon),
Prod(T,Z,T)) },unlabelled]:
```

Ez a leírás emlékeztet a bináris keresési fákra.

Megszámlálhatjuk, hogy a 102-szöget hányféleképpen tudjuk háromszögekre bontani a fent említett módon.

```
> count(haromszog,size=100);
```

896519947090131496687170070074100632420837521538745909320

Kilistázható az első 20 érték:

```
> seq(count(haromszog,size=i),i=0..20);
```

0, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420

Ezek a számok a jól ismert **Catalan számok**, nevüket a francia matematikus nevéről kapták, aki az 1850-es években kidolgozta a főbb tulajdonságukat.

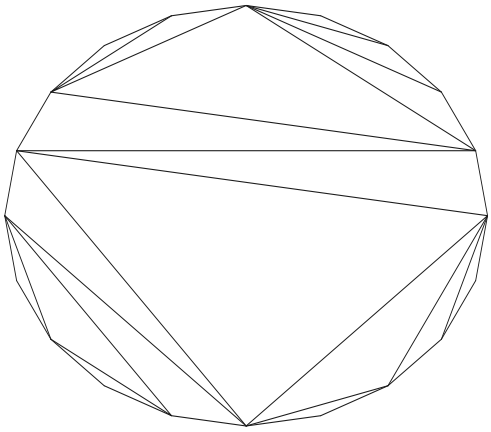
Lássunk egy adott méret esetén véletlenszerűen generált háromszögelést. A dodekaédernek (12 oldalú sokszögnek) háromszögekre való bontása a következő lehet:

```
> draw(haromszog,size=10);
```

$\mathrm{Prod}(E, Z, \mathrm{Prod}(\mathrm{Prod}(\mathrm{Prod}(\mathrm{Prod}(\mathrm{Prod}(\mathrm{Prod}(Z, Z, E), Z, E), Z, E), Z, E), Z, E), Z, Z), Z, E))$

Rajzzal is szemléltethetjük a háromszögelést. Az alábbi eljárások a meghúzott élek (átlók) listáját szerkesztik meg.

```
> size:=proc(t) convert(map(size,t),`+`) end: size(Epsilon):=0:
size(Z):=1:
> elek:=proc(fa,org,el) local se1,se2;
if fa=Epsilon then org,org,el union {[org,org+1]}
elif fa=Z then org,org+1,el union
{[org,org+1],[org+1,org+2],[org+2,org]}
else
```



1. ábra. A 20-szög egy lehetséges háromszögelése

```
se1:=elek(op(1,fa),org,el);
se2:=elek(op(3,fa),se1[2]+1,se1[3]);
se1[1],se2[2],se2[3] union {[se1[1],se2[2]+1]}
fi
end:
```

A kirajzolást megvalósító eljárás:

```
> rajz:=proc(n) plot([op(map2(map,[cos,sin],expand(map(‘*‘,elek
(draw(haromszog,size= n-2),0,{})[3],2*Pi/n))))],color=blue,
axes=NONE) end:
```

```
> rajz(20);
```

Most listázzuk ki az allstructs() függvénnyel a hatszög összes háromszögelését.

```
> harom4:=allstructs(haromszog,size=4);
```

$harom4 :=$ [Prod(Prod(Z, Z, Z), Z, E), Prod(E, Z, Prod(Z, Z, Z)),
Prod(E, Z, Prod(Prod(E, Z, Z), Z, E)), Prod(E, Z, Prod(E, Z, Prod(E, Z, Z))),
Prod(Z, Z, Prod(Z, Z, E)), Prod(Prod(E, Z, Prod(E, Z, Z)), Z, E),
Prod(E, Z, Prod(E, Z, Prod(Z, Z, E))), Prod(Prod(Z, Z, E), Z, Z),
Prod(E, Z, Prod(Prod(Z, Z, E), Z, E)), Prod(Prod(Prod(Z, Z, E), Z, E), Z, E),
Prod(Prod(Prod(E, Z, Z), Z, E), Z, E), Prod(Prod(E, Z, Z), Z, Z),
Prod(Prod(E, Z, Prod(Z, Z, E)), Z, E), Prod(Z, Z, Prod(E, Z, Z))]

14 objektumot találtunk.

Generátorfüggvény megadása ez esetben is lehetséges:

```
> gfeqns(op(2,haromszog),unlabelled,z);
```

$$[\mathrm{T}(z) = \mathrm{Z}(z) + 2\,\mathrm{Z}(z)\,\mathrm{T}(z) + \mathrm{T}(z)^2\,\mathrm{Z}(z),\ \mathrm{Z}(z) = z]$$

```
> gf:=gfsolve(op(2,haromszog),unlabelled,z);
```

$$gf := \{\mathrm{Z}(z) = z,\ \mathrm{T}(z) = \frac{1}{2}\,\frac{1 - 2\,z - \sqrt{1 - 4\,z}}{z}\}$$

```
> op([1,2],%); series(%,z=0,12);
```

$$\frac{1}{2}\,\frac{1 - 2\,z - \sqrt{1 - 4\,z}}{z}$$

$$z+2\,z^2+5\,z^3+14\,z^4+42\,z^5+132\,z^6+429\,z^7+1430\,z^8+4862\,z^9+16796\,z^{10}+\mathrm{O}(z^{11})$$

```
> seq(count(haromszog,size=n),n=0..10);
```

0, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796

```
> gfsor(op(2..3,haromszog),z,[[z]],6);
```

table([
$\mathrm{T}(z) = z + 2\,z^2 + 5\,z^3 + 14\,z^4 + 42\,z^5 + \mathrm{O}(z^6)$
$\mathrm{Z}(z) = z$
])

3.5.5. Természetes számok partíciói

9. Értelmezés. *Az n szám partícióinak az n-nek pozitív egész számok összegeként való előállításait nevezzük.*

Két partíciót nem tekintünk különbözőnek, ha csak a tagok sorrendjében térnek el egymástól.

Határozzuk meg például 10 partícióit. Ehhez a combinat csomag partition() ejárását használjuk.

```
> partition(10);
```

[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 2], [1, 1, 1, 1, 1, 1, 2, 2],
[1, 1, 1, 1, 2, 2, 2], [1, 1, 2, 2, 2, 2], [2, 2, 2, 2, 2], [1, 1, 1, 1, 1, 1, 1, 3],
[1, 1, 1, 1, 1, 2, 3], [1, 1, 1, 2, 2, 3], [1, 2, 2, 2, 3], [1, 1, 1, 1, 3, 3], [1, 1, 2, 3, 3],
[2, 2, 3, 3], [1, 3, 3, 3], [1, 1, 1, 1, 1, 1, 4], [1, 1, 1, 1, 2, 4], [1, 1, 2, 2, 4],
[2, 2, 2, 4], [1, 1, 1, 3, 4], [1, 2, 3, 4], [3, 3, 4], [1, 1, 4, 4], [2, 4, 4],
[1, 1, 1, 1, 1, 5], [1, 1, 1, 2, 5], [1, 2, 2, 5], [1, 1, 3, 5], [2, 3, 5], [1, 4, 5], [5, 5],
[1, 1, 1, 1, 6], [1, 1, 2, 6], [2, 2, 6], [1, 3, 6], [4, 6], [1, 1, 1, 7], [1, 2, 7], [3, 7],
[1, 1, 8], [2, 8], [1, 9], [10]]

Ha az összes partíciók száma érdekel, Mapleben adott a numbpart() eljárás.

```
> numbpart(10);
```

42

A randpart() függvény véletlenszerű partíciót generál.

```
> randpart(10);
```

[1, 2, 2, 2, 3]

A combstruct csomag egy, előbb már említett struktúrája, a **Partition**, szintén segítségünkre lehet.

Egy adott szám particionálásához megadhatjuk az egyes partíciókat alkotó számok mennyiségét az allstructs eljárásban.

```
> allstructs(Partition(6),size=3);
```

[[1, 1, 4], [1, 2, 3], [2, 2, 2]]

Egy kis munkával a partition() eljárással is meghatározható ez.

```
> for part in partition(6) do if nops(part)=3 then print(part); fi;
od;
```

[2, 2, 2]

[1, 2, 3]

[1, 1, 4]

Ha a partíciók sorrendjére tekintettel vagyunk, vagyis ha egy adott szám kompozíciói érdekelnek, erre az esetre is megoldást kínál a Maple. Ekkor a **Composition** struktúrára kell hivatkoznunk.

```
> allstructs(Composition(6),size=3);
```

[[1, 2, 3], [1, 3, 2], [2, 1, 3], [3, 1, 2], [2, 2, 2], [1, 1, 4], [1, 4, 1], [4, 1, 1], [3, 2, 1], [2, 3, 1]]

A count() függvényt ez esetben is alkalmazhatjuk.

```
> count(Partition(6),size=3);
```

3

```
> count(Composition(6),size=3);
```

10

Egy másik alternatíva a numbperm(), illetve a numbcomp() eljárások alkalmazása lehet. Például:

```
> numbcomp(6,3);
```

10

Könnyen észrevehető, hogy fennáll a következő reláció:

numbcomp(n,m) = binomial(n-1, m-1).

Szintén két választási lehetőségünk van véletlenszerű partíció generálására, attól függően, hogy milyen csomagot töltöttünk be.

```
> draw(Partition(6),size=3);
```

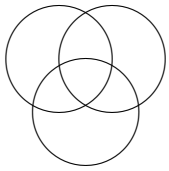
[1, 1, 4]

```
> randpart(6);
```

[2, 4]

Előbbi előnye, hogy a méretet is megadhatjuk, míg az utóbbinál ez nem lehetséges.

4. Halmazok generálása



4.1. Descartes szorzat

Maple-ben rendelkezésünkre állnak a megszokott halmazműveletek: egyesítés (union), metszet (intersect), különbség (minus). Ezeken kívül még adott egy member() eljárás, amely két paramétert használ, az első paraméter az objektum, melynek jelenlétét a második objektumban a rendszer ellenőrzi. A visszaadott érték igaz (true), vagy hamis (false).

Az alábbi eljárás két halmaz Descartes-szorzatát generálja.

```
> descartes1:=proc(X,Y)
local Z,x,y;
Z:={};
for x in X do
for y in Y do
Z:=Z union {[x,y]};
od;
od;
Z end:
```

Alkalmazva két halmazra:

```
> descartes1({2,5,8,3},{1,3,5,9,4});
```

{[3, 4], [3, 5], [2, 1], [8, 5], [8, 9], [8, 1], [2, 3], [2, 4], [2, 5], [2, 9], [3, 9], [3, 1], [5, 3], [5, 4], [5, 5], [5, 9], [5, 1], [8, 3], [8, 4], [3, 3]}

Egy másik, rekurzív eljárás, amely tetszőleges számú halmaz Descartes-szorzatát adja meg.

```
> descartes2:=proc()
local Z,k,x,y;
option remember;
if nargs=0 then Z:={};
elif nargs=1 then Z:=args[1];
else Z:={};
for x in descartes2( seq(args[k], k=1..nargs-1) ) do
for y in args[nargs] do
Z:= Z union {[op(x),y]};
od; od; fi;
RETURN (Z);
end:
```

Tesztelése:

```
> B1:={1,2,3}; B2:={b,a}; B3:={a,c,b};
```

$$B1 := \{3, 1, 2\}$$
$$B2 := \{b, 6\}$$
$$B3 := \{b, c, 6\}$$

```
> descartes2(B1,B2,B3);
```

$\{[1, a, a], [1, a, b], [1, a, c], [1, b, a], [1, b, b], [1, b, c]\, [2, a, a], [2, a, b], [2, a, c],$
$[2, b, a], [2, b, b], [2, b, c]\, [3, a, a], [3, a, b], [3, a, c], [3, b, a], [3, b, b], [3, b, c]\}$

Alkalmazhatjuk a Maple beépített függvényét is, a cartprod() eljárást.

```
> T:=cartprod([[1,2,3], [a,b]]):
> while not T[finished] do T[nextvalue]() od;
```

$$[1, 6]$$
$$[1, b]$$
$$[2, 6]$$
$$[2, b]$$
$$[3, 6]$$
$$[3, b]$$

4.2. Részhalmazok

A Maple maga is rendelkezik olyan eljárásokkal, amelyek segítségével adott halmaz részhalmazait határozhatjuk vagy számlálhatjuk meg. Ilyen például a combstruct csomag allstructs() eljárása, amelyet a Subset konstruktorral kell használni.

```
> allstructs(Subset({a,b,c}),size=2);
```

$$\{\{b, 6\}, \{c, 6\}, \{b, c\}\}$$

```
> allstructs(Subset({a,b,c,c }),size=allsizes);
```

$$\{\{b, 6\}, \{b, c, 6\}, \{c, 6\}, \{b, c\}, \{b\}, \{c\}, \{6\}, \{\}\}$$

```
> count(Subset({a,b,c,c}),size=allsizes);
```

$$8$$

A combinat csomag powerset() függvénye szintén részhalmazokat származtat.

```
> powerset({a,b,c});
```

$$\{\{b, 6\}, \{b, c, 6\}, \{c, 6\}, \{b, c\}, \{b\}, \{c\}, \{6\}, \{\}\}$$

Paramétere ugyanakkor lista is lehet.

```
> powerset([a,b,c]);
```

$$[[b, c, 6], [c, 6], [6], [c], [b, 6], [b, c], [b], []]$$

Csak akkor van különbség a két adattípus alkalmazásakor, ha ismétlődő elemeket tartalmaznak.

```
> powerset({a,b,c,c});
```

$$\{\{b, 6\}, \{b, c, 6\}, \{c, 6\}, \{b, c\}, \{b\}, \{c\}, \{6\}, \{\}\}$$

```
> powerset([a,b,c,c]);
```

$[[], [c], [6], [c, 6], [c, c], [c, c, 6], [b], [b, c], [b, 6], [b, c, 6], [b, c, c], [b, c, c, 6]]$

4.2.1. Részhalmazokat meghatározó eljárások

Saját magunk is írhatunk függvényeket hasonló problémák megoldására.

Az alábbi eljárás egy lista i-edik elemétől kezdődő allistáját (részhalmazát) határozza meg. Meghívásakor(allista(lista,i)) figyelembe kell venni, hogy az első argumentum nem üres lista, a második paraméternek pedig teljesítenie kell az $1 <= ini <= nops(lista)$ összefüggést.

```
> allista := proc(list, ini) [ seq(list[i],i=ini..nops(list)) ]:
end:

Warning, ‘i‘ in call to ‘seq‘ is not local
> allista([a,b,c,d],1);
```

$$[a, b, c, d]$$

```
> allista([a,b,c,d],2);
```

$$[b, c, d]$$

```
> allista([a,b,c,d],4);
```

$$[d]$$

Egy lista listáihoz adott elemet rendel a következő eljárás.

```
> restart;
> Rendel := proc(elt, list) local temp, i: temp:=[]: for i from 1
to nops(list) do temp:=[ op(temp), [elt, op(list[i])] ]: od: temp:
end:
> Rendel(m, [[1,a],[2],[3,b,c],[4,d,x,y,x],[]]);
```

$$[[m, 1, a], [m, 2], [m, 3, b, c], [m, 4, d, x, y, x], [m]]$$

```
> Rendel(a, [[b,c],[b,d],[c,d]]);
```

$$[[a, b, c], [a, b, d], [a, c, d]]$$

Az Ielemu_ reszhalm(L,i) eljárás meghívása esetén visszaadja az L lista összes allistáját (részhalmazát) amely i elemet tartalmaz. Adott tehát egy L nem üres lista és i, ahol $1 <= i <= nops(L)$.

```
> Ielemu_reszhalm := proc(L, i) local n, j, eredm, temp:
n := nops(L):
if i=1 then
eredm:=[]:
for j from 1 to n do
eredm:=[op(eredm),[L[j]]]:
od:
else
eredm:=[]:
for j from 1 to n-i+1 do
temp:=Ielemu_reszhalm(allista(L,j+1),i-1):
temp:=Rendel(L[j],temp):
eredm:=[op(eredm),op(temp)]:
od:
fi:
eredm:
end:
```

```
> Ielemu_reszhalm([a,b,c,d,e,f,g],1);
```

$$[[a], [b], [c], [d], [e], [f], [g]]$$

Az összes részhalmaz kilistázására szolgál az alábbi eljárás.

Megjegyzendő, hogy L ismét egy nem üres lista.

```
> Reszhalmazok := proc(L)
local n, i, temp, eredm:
n := nops(L):
eredm := []:
for i from 1 to n do
eredm:=[op(eredm),op(Ielemu_reszhalm(L,i))]:
od:
eredm:
end:
> Reszhalmazok([x,y,z]);
```

$$[[x], [y], [z], [x, y,], [x, z], [x, y, z]]$$

4.3. Halmazok partíciói

10. Értelmezés. *Halmazfelbontáson azt értjük, hogy a halmaz minden eleme a szóban forgó részhalmazok közül pontosan egyhez tartozik.*

Néha szeretnénk egy kifejtett listát kapni egy halmaz összes partícióiról, hogy tudjunk bizonyos műveletet végezni velük (azaz a lista elemeivel, amelyek partíciók). Ezt teszi lehetővé az alábbi eljárás, amely az $1, 2, 3, ..., n$ partícióit listázza ki.

```
> particiok:=proc(n)
local P,A,B,C; option remember;
if n=1 then P:=[{1}];
elif n=2 then P:=[ {{1},{2}}, {{1,2}} ];
elif n>2 then P:=[];
for A in particiok(n-1) do
P:= [ op(P), A union {{n}} ];
for B in A do
C:= ( A minus {B} ) union { B union {n} };
P:= [op(P), C ];
od;
od;
elif n=0 then P:= []; else P:=FAIL;
fi; RETURN(P) end:
```

Kipróbáljuk, hogy működik:

```
> for A in particiok(3) do print(A); od;
```

$$\{\{2\}, \{3\}, \{1\}\}$$
$$\{\{2, 3\}, \{1\}\}$$
$$\{\{2\}, \{1, 3\}\}$$
$$\{\{1, 2\}, \{3\}\}$$
$$\{\{1, 2, 3\}\}$$

4.3.1. Stirling- és Bell-féle számok

A Stirling-számok, amelyeket James Stirling (1692-1770) matematikus tiszteletére neveztek el, kapcsolatban áll a binomiális együtthatókkal. Két típust különböztetünk meg és ezeket első- és másodfajú Stirling-számoknak nevezzük.

11. Értelmezés. *Az elsőfajú Stirling-szám ($s(n,m)$)azon lehetőségek számát jelöli, ahogyan n elemet m ciklusba lehet elrendezni.*

12. Értelmezés. *A másodfajú Stirling-szám, amelyre az $S(n,m)$jelölést használjuk, azon esetek számát jelenti,ahányféleképpen egy n elemű halmazt m számú nemüres részhalmazra fel tudunk bontani.*

13. Értelmezés. *A Bell-szám, B_n, azon esetek számát jelöli, ahányféleképpen n elemet részhalmazokba lehet szétosztani.*

Legyen A egy n elemű halmaz. Az A halmaz összes nem üres partícióinak száma tehát a Bell-féle szám.

Maple-ben az első-, illetve a másodfajú Stirling-féle számok kiszámítására rendre a stirling1() és a stirling2() függvényt használjuk. Például egy hat elemű halmazból képezett négy elemű részhalmazok száma a következőképpen is megadható:

```
> stirling2(6,4);
```

$$65$$

A Bell-féle számokat a bell() eljárással kapjuk meg.

```
> bell(4);
```

$$15$$

Maple-ben könnyen szemléltethető, hogy a Bell-féle számok nagyon gyorsan nőnek n egyre nagyobb értékeire.

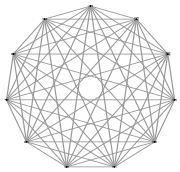
```
> 'bell(8)' = bell(8); 'bell(10)' = bell(10); 'bell(20)' = bell(20);
```

$$\text{bell}(8) = 4140$$

$$\text{bell}(10) = 115975$$

$$\text{bell}(20) = 51724158235372$$

5. Gráfok és fák



14. Értelmezés. *A $G = (V, E)$ rendezett párt gráfnak nevezzük, ha V egy nem üres halmaz, E pedig e halmazból képezhető párok egy halmaza.*
A V halmaz elemeit a gráf csúcsainak (csomópontjainak, szögpontjainak), az E halmaz elemeit a gráf éleinek nevezzük.

A Maple külön csomaggal rendelkezik véges gráfok megszerkesztéséhez: a network csomaggal. Ezt kell betöltenünk amikor gráfokkal akarunk dolgozni.

```
> with(networks);
```

[*acycpoly*, *addedge*, *addvertex*, *adjacency*, *allpairs*, *ancestor*, *arrivals*, *bicomponents*, *charpoly*, *chrompoly*, *complement*, *complete*, *components*, *connect*, *connectivity*, *contract*, *countcuts*, *counttrees*, *cube*, *cycle*, *cyclebase*, *daughter*, *degreeseq*, *delete*, *departures*, *diameter*, *dinic*, *djspantree*, *dodecahedron*, *draw*, *duplicate*, *edges*, *ends*, *eweight*, *flow*, *flowpoly*, *fundcyc*, *getlabel*, *girth*, *graph*, *graphical*, *gsimp*, *gunion*, *head*, *icosahedron*, *incidence*, *incident*, *indegree*, *induce*, *isplanar*, *maxdegree*, *mincut*, *mindegree*, *neighbors*, *new*, *octahedron*, *outdegree*, *path*, *petersen*, *random*, *rank*, *rankpoly*, *shortpathtree*, *show*, *shrink*, *span*, *spanpoly*, *spantree*, *tail*, *tetrahedron*, *tuttepoly*, *vdegree*, *vertices*, *void*, *vweight*]

5.1. Gráfok létrehozása

Gráfokat megjeleníteni többféle utasítással lehet. Erre léteznek a graph(), new(), void(), complete(), cube(), cycle() és random() parancsok.

A new() eljárás olyan gráfot hoz létre, amelynek nincs egy csúcsa és egy éle sem.

```
> G1:=new():
```

Az előbbi gráfhoz csúcsot az addvertex(), élet pedig az addedge()függvénnyel rendelünk.

```
> csucsok:={A,B,C,D,E};
```

$$csucsok := \{E,\ \mathrm{D},\ 4,\ B,\ C\}$$

```
> addvertex(csucsok,G1);
```

$$E,\ \mathrm{D},\ 4,\ B,\ C$$

```
> addedge({{A,B},{A,C},{B,C},{C,D}, {B,E},{D,E},{B,D}},G1);
```

$$e1,\ e2,\ e3,\ e4,\ e5,\ e6,\ e7$$

A hurokél megadása, például annak a ténynek az értelmezése, hogy a gráf A-ban hurokélet tartalmaz, a {A} jelöléssel történik. Sajnos ez nem lesz feltűntetve a rajzon.

Irányított gráfokat szintén nem lehet szemléltetni rajzon, viszont az irányított élek megadásánál az összekötendő csúcsokat szögletes zárójelbe tesszük. Például a [B,D] jelölést használjuk a B-ből D-be tartó irányított él értelmezésére.

Az előbbi gráf esetén lekérdezhetjük az élek végcsúcsait az ends()függvénnyel,illetve az élek neveit az edges()eljárással.

```
> ends(G1); edges(G1);
```

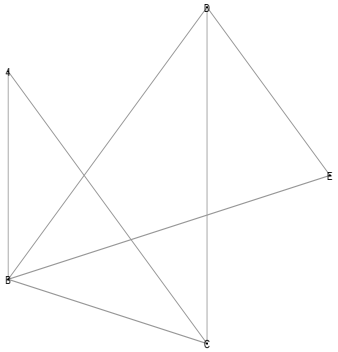
$$\{\{E, \mathrm{D}\}, \{B, C\}, \{\mathrm{D}, C\}, \{E, B\}, \{\mathrm{D}, B\}, \{4, B\}, \{4, C\}\}$$

$$\{e2, e3, e4, e5, e6, e7, e1\}$$

Gráfot kirajzolni a draw() utasítással tudunk.

Esetünkben a G1 gráf a következőképpen szemléltethető:

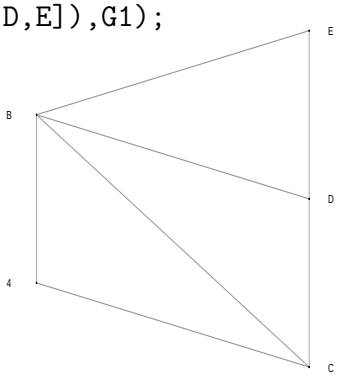
```
> draw(G1);
```



2. ábra. A G1 gráf

Bizonyos mértékig saját magunk is szabályozhatjuk az ábra kinézetelét, ha a draw() utasításban a **Concentric** vagy **Linear** opciót használjuk.

```
> draw(Linear([A,B],[C,D,E]),G1);
```

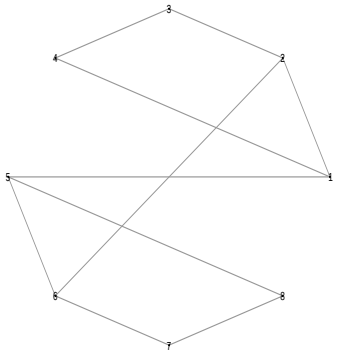


3. ábra. A G1 gráf a csúcsok átrendezésével

Gráf rajzolása a graph() segítségével:

```
> G2 := graph( {1,2,3,4,5,6,7,8},{{1,2},{2,3},{3,4},{4,1},{5,6},{6,7},
{7,8},{8,5},{1,5},{2,6}}):
```

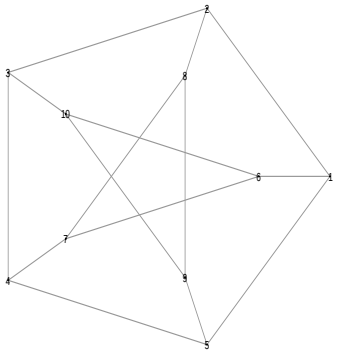
```
> draw(G2);
```



4. ábra. A G2 gráf

Előre definiált gráfok is vannak a networks csomagban, ilyenek például a Petersen gráf, a szabályos sokszöggráfok (szabályos tetraéder, kocka, oktaéder, dodekaéder, ikozaéder gráfjai), vagy a teljes gráfok. Ez utóbbit a complete()függvény hozza létre.

A Petersen gráfot a petersen() függvény értelmezi és a következőképpen néz ki:



5. ábra. A Petersen gráf

5.2. Mátrix-reprezentációk

A networks csomag adjacency() és incidence() eljárásai hozzárendelik egy adott gráfhoz szomszédsági és illeszkedési mátrixait. A szomszédsági mátrix minden sorának, illetve minden oszlopának egy-egy csúcs felel meg, 1-gyel jelölve, ha két csúcs szomszédos (össze van kötve éllel), 0-val, ha nem szomszédos. Az illeszkedési mátrix oszlopai éleket, sorai pedig csúcsokat reprezentálnak, -1-gyel jelölve az irányított élek kezdőpontját, 1-gyel a végpontot, a többi esetben pedig 0 az értéküket.

A G1 gráf szomszédsági és illeszkedési mátrixa:

```
> adjacency(G1);
```

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Ha a gráf nem irányított, a szomszédsági mátrixa szimmetrikus lesz.

```
> incidence(G1);
```

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Ha adott egy A négyzetes mátrix, amelynek elemei nullákból és egyesekből állanak, megrajzolható-e a gráf, amelynek a szomszédsági mátrixa A-val megegyezik?

Az alábbi eljárás létrehozza az A matrix ismeretében a csúcsok halmazát. Ha n*n-es a mátrix, a csúcsok halmazát $1, 2, \ldots, n$-nel fogjuk jelölni. A linalg csomag a mátrixokkal való műveleteket tartalmazza. Benne található a matrix() függvény is, amely létrehoz egy kétdimenziós tömböt, vagyis egy mátrixot. A mátrix sorainak számát a rowdim, oszlopainak számát a coldim eljárással határozhatjuk meg.

```
> with(linalg):
> szogpontok:=proc(A::matrix)
local L,i:
L:={}:
for i from 1 to rowdim(A) do L:=L union {i}: od: L; end:
> szogpontok(matrix(5,5,[0,1,1,0,0,1,0,1,1,1,1,1,0,1,0,0,1,1,0,1,
0,1,0,1,0]));
```

$$\{1, 2, 3, 4, 5\}$$

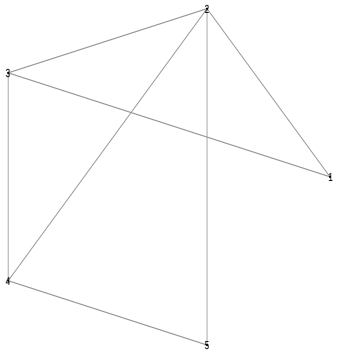
Az élek meghatározása végett megkeressük a mátrix azon elemeit, amelyek 1-gyel egyenlők. Ha például $A(i, j) = 1$, akkor az i,j élet hozzávesszük az L, kezdetben üres halmazhoz.

```
> elek:=proc(A::matrix)
local L,i,j:
L:={}:
for i from 1 to rowdim(A) do
for j from 1 to coldim(A) do
if A[i,j]=1 then L:=L union {{i,j}}: fi:
od: od: L; end:
```

```
> elek(matrix(5,5,[0,1,1,0,0,1,0,1,1,1,1,1,0,1,0,0,1,1,0,1,0,1,0,
1,0]));
```

$$\{\{2, 4\}, \{1, 3\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{2, 5\}, \{1, 2\}\}$$

```
> graf:=proc(A::matrix)
local G:
G:=new():
addvertex(szogpontok(A),G):
addedge(elek(A),G):
G; end:
> draw(graf(matrix(5,5,[0,1,1,0,0,1,0,1,1,1,1,1,0,1,0,0,1,1,0,1,0,
1,0,1,0])));
```



6. ábra. Az A mátrix ismeretében kapott gráf

A szomszédsági mátrix, mint ismeretes, egy gráf i-edik csúcsától a j-edik csúcsig tartó n hosszúságú utak meghatározására is alkalmazható.

Például az előbbiekben említett G1 gráf 20 hosszúságú (20 élből álló) B és C szögpontjai közötti utak száma (a Maple szerint) 592841526 (a harmadik sor, második oszlop eleme), még ha szabad szemmel az összest nem is látjuk.

```
> evalm(szomszed^20);
```

$$\begin{bmatrix} 277284196 & 442198913 & 371738687 & 371745452 & 277280015 \\ 442198913 & 705206824 & 592841526 & 592841526 & 442198913 \\ 371738687 & 592841526 & 498387035 & 498376089 & 371745452 \\ 371745452 & 592841526 & 498376089 & 498387035 & 371738687 \\ 277280015 & 442198913 & 371745452 & 371738687 & 277284196 \end{bmatrix}$$

A gráfok mátrixszal való reprezentálása előnyös lehet, ha a gráf bonyolult, sok éllel rendelkezik.

A grafikus ábrázolásmódhoz képest egy másik előny, hogy a hurokél vagy a többszörös él is reprezentálható, a mátrixokból kiolvasható.

5.3. Gráfok összefüggősége

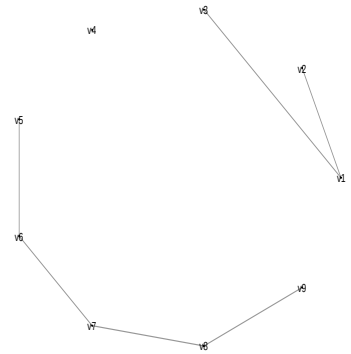
15. Értelmezés. *Azt mondjuk egy gráfról, hogy konex (összefüggő), ha bármely két csúcsa között létezik út. Ellenkező esetben azt mondjuk, hogy konex komponensek alkotják.*

A components() parancs segít megtalálni egy gráf összefüggő komponenseit.

```
> G2:=new():
> addvertex({v1,v2,v3,v4,v5,v6,v7,v8,v9},G2):
> addedge({{v1,v2},{v1,v3},{v5,v6}, {v6,v7},{v7,v8},{v8,v9}},G2):
> components(G2);
```

$$\{\{v5,\ v6,\ v7,\ v8,\ v9\},\ \{v1,\ v3,\ v2\},\ \{v4\}\}$$

```
> draw(G2);
```



7. ábra. A G2 gráf

A szomszédsági mátrixról is leolvasható, hogy egy gráf összefüggő vagy sem.

```
> adjacency(G2);
```

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Látható, hogy blokkátlós mátrixot kaptunk. Ez azt jelenti, hogy a gráfot összefüggő komponensek alkotják.

5.4. Euler-gráfok

16. Értelmezés. *Az Euler-kör a gráfnak olyan zárt vonala, mely minden élét és minden szögpontját tartalmazza (a csúcsokat többször, az éleket csak egyszer érintheti). Azt a gráfot, amely Euler-kört tartalmaz, Euler-gráfnak nevezzük.*

Az alábbi eljárás (Euler) azon a tulajdonságon alapszik, hogy egy izolált pontot nem tartalmazó gráf akkor és csak akkor Euler-gráf, ha összefüggő és minden csúcsa páros fokszámú.

Ha adott egy gráf, a degreeseq() eljárás kilistázza a szögpontok fokszámát növekvő sorrendben. Például:

```
> degreeseq(G2);
```

[0, 1, 1, 1, 1, 2, 2, 2, 2]

Az eljárás Maple kódja:

```
> Euler:=proc(G::graph)
local c,i,r:
c:=degreeseq(G):
i:=1:
while i<nops(c)+1 do
if (c[i] mod 2)<>0 then i:=nops(c)+2:else i:=i+1:
fi:
od:
if i=nops(c)+2 then r:="Nem Euler-graf":
else r:="Euler-graf":
fi:
r
end:
```

Tesztelés:

```
> Euler(cycle(5));
```

„Euler-graf”

```
> Euler(complete((4)));
```

„Nem Euler-graf”

```
> Euler(G1);
```

„Nem Euler-graf”

Megjegyzés: Ahhoz, hogy a szögpontok sorrendjében írja ki a rendszer egy gráf fokszámait, így járhatunk el:

```
> L:=[op(vertices(G2))]; vdegree(L[1],G2);
```

$L := [v4, v5, v6, v7, v8, v9, v1, v3, v2]$

0

```
> for i from 1 to nops(L) do
c[i]:=vdegree(L[i],G2):
od:
seq(c[j],j=1..nops(L));
```

0, 1, 2, 2, 2, 1, 2, 1, 1

5.5. Súlyozott gráfok

Ha a gráf éleihez súlyokat akarunk rendelni (időt, távolságot vagy költséget akarunk megfeleltetni az adott éleknek), ez megoldható úgy, hogy az addedge() parancs **weights** opcióját használjuk.

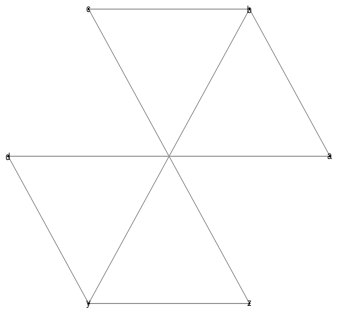
```
> new(G3):
> addvertex({a,b,c,d,y,z},G3);
```

$$d,\ a,\ b,\ c,\ y,\ z$$

```
> addedge([{a,b},{a,d},{b,c},{b,y}, {c,z},{d,y},{y,z}],
weights=[4,2,3,3,2,3,1],G3);
```

$$e1,\ e2,\ e3,\ e4,\ e5,\ e6,\ e7$$

```
> draw(G3);
```



8. ábra. A G3 gráf

Az eweight() függvény visszaadja egy meghatározott él súlyát.

```
> eweight(e5,G3);
```

$$2$$

5.6. Feszítőfák, minimális hosszúságú utak

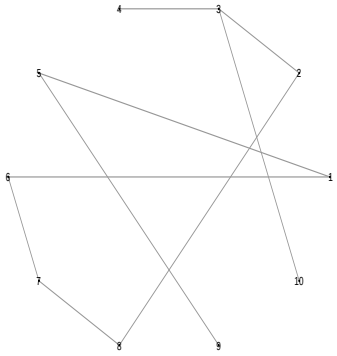
17. Értelmezés. *Egy összefüggő, körmentes egyszerű (huroél- és többszörösélnélküli) gráfot fának nevezünk.*

18. Értelmezés. *Egy G gráf feszítőfája egy olyan F fa, amely szögpontjainak halmaza megegyezik g szögpontjainak halmazával, élei pedig G élei közül valók.*

19. Értelmezés. *A gráf feszítőfáját minimálisnak nevezzük, ha az éleihez tartozó súlyok összege minimális.*

A feszítőfa meghatározására a Maple a spantree() eljárást adja.
A Petersen-gráf egy feszítőfája a következőképpen néz ki:

```
> draw(spantree(petersen()));
```



9. ábra. A Petersen gráf egy feszítőfája

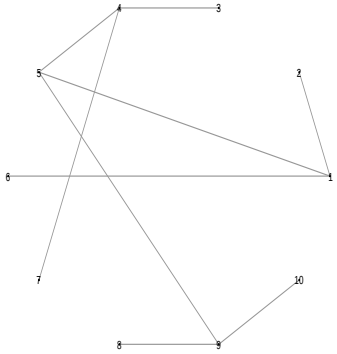
Megtudhatjuk egy adott gráf feszítőfáinak számát, ha a counttrees() függvényre hivatkozunk. Például:

```
> counttrees(petersen());
```

2000

A networks csomagban található a shortpathtree() eljárás. Eredményeképp megkapjuk egy adott csúcstól a minimális hosszúságú utakat a többi csúcshoz.

```
> draw(shortpathtree(petersen(),5));
```



10. ábra. Minimális utak a Petersen gráf 5-ös csúcsától

Ha egy gráf tetszőleges két szögpontja közötti (az élek súlyai szerinti) legrövidebb út hosszát szeretnénk meghatározni, akkor az allpairs() utasítást kell alkalmaznunk.

6. Lineáris rekurziós egyenletek

$$\mathbf{a_{n+4} - 4a_{n+3} + 6a_{n+2} - 4a_{n+1} + a_n = 0}$$

20. Értelmezés. *Azt az egyenletet, amelyben az ismeretlenek egy adott számsorozat $x_n, , x_{n+k}$ tajai, k-ad rendű rekurzív relációnak nevezzük.*

A rekurzió megoldásához kezdőértéket kell megadnunk.

Maple-ben az rsolve() eljárást használjuk a lineáris rekurziós eljárások megoldásához.

Egyik legismertebb másod-rendű rekurziós reláció a következő,amelynek megoldása a Fibonacci-sorozat:

```
> fibonacci:=f(n)=f(n-1)+f(n-2);
```

$$\textit{fibonacci} := \mathrm{f}(n) = \mathrm{f}(n-1) + \mathrm{f}(n-2)$$

Kezdőérteket adunk meg:

```
> kezdeti1:=f(0)=0,f(1)=1;
```

$$\textit{kezdeti1} := \mathrm{f}(0) = 0, \mathrm{f}(1) = 1$$

A megoldás zárt formában:

```
> megoldas1:=rsolve({fibonacci,kezdeti1},f);
```

$$\textit{megoldas1} := \frac{(-1 + \frac{1}{5}\sqrt{5})\,(-2\,\frac{1}{1-\sqrt{5}})^n}{1-\sqrt{5}} + \frac{(-\frac{1}{5}\sqrt{5} - 1)\,(-2\,\frac{1}{1+\sqrt{5}})^n}{1+\sqrt{5}}$$

A sorozat generátorfüggvényét is egyszerű megkapni.

```
> generator1:=rsolve({fibonacci,kezdeti1 },f,'genfunc'(z));
```

$$\textit{generator1} := -\frac{z}{-1 + z + z^2}$$

A Maple rendszer egy jó sajátsága, hogy akár értelmezhetjük is az eljárást, amely kiszámítja $a(n)$-et.

```
> fuggveny1:=rsolve({fibonacci,kezdeti1 },f,'makeproc'):
```

Kipróbálása:

```
> fuggveny1(3);fuggveny1(4);fuggveny1(5);fuggveny1(6);
fuggveny1(7);fuggveny1(8);
```

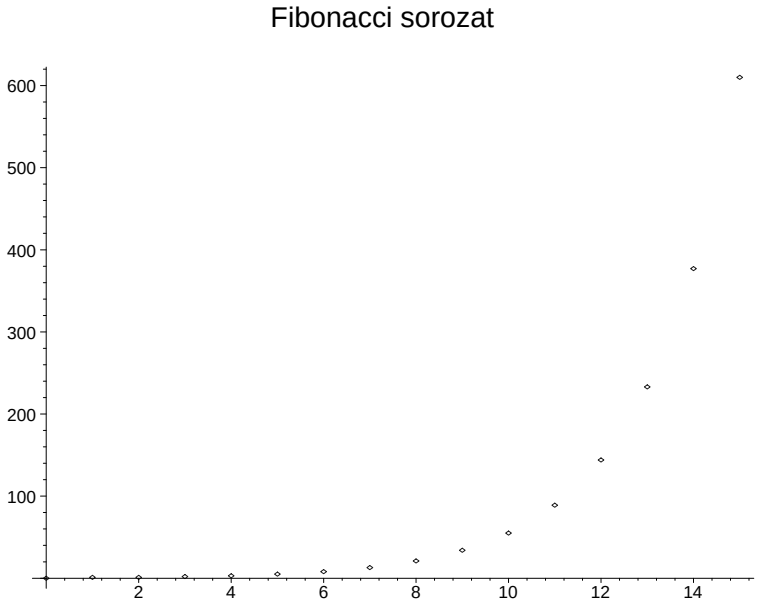
2
3
5
8
13
21

A fuggvény1 eljárás az a(n) sorozat ábrázolására is felhasználható.

```
> sorozat1:=seq([n,fuggveny1(n)],n=0..15);
```

sorozat1 := [0, 0], [1, 1], [2, 1], [3, 2], [4, 3], [5, 5], [6, 8], [7, 13], [8, 21], [9, 34], [10, 55], [11, 89], [12, 144], [13, 233], [14, 377], [15, 610]

```
> plot([sorozat1],style=point,symbol=diamond,title="Fibonacci
sorozat",color=blue);
```



A lineáris,homogén rekurziós egyenlethez karakterisztikus polinomot rendelhetünk.

Az alábbi eljárás ezt a polinomot határozza meg.

```
> karpol:=proc(egyenlet,a,n,rang,valtozo)
local k,sor;
sor:=seq(a(n-k)=valtozo^(rang-k),k=0..rang);
subs(sor,egyenlet);
end:
```

Ellenorizzük le a fibonacci nevu egyenleten.

```
> karpol(fibonacci,f,n,2,z);
```

$$z^2 = z + 1$$

Akár valós,akár komplex gyökkel rendelkezik a rekurziós egyenlet, a fent említett módon határozzuk meg a megoldását.

Irodalomjegyzék

[**1**] **Ronald L. Graham, Donald E. Knuth**:*Konkrét matematika*, Műszaki Könyvkiadó, 1998.

[**2**] **Klincsik Mihály, Maróti György**: *Maple 8 tételben*, Novadat, 1995.

[**3**] *First Leaves, A Tutorial Introduction to Maple*, Waterloo Maple Publishing, 1990.

[**4**] **Kása Zoltán, Bege Antal**: *Matematică discretă*, curs universitar, Cluj-Napoca, 2002.

[**5**] **Szendrei Ágnes**:*Diszkrét matematika*, POLYGON, szeged, 1998.

[**6**] **Márton Éva**: *Gráfelmélet Maple-ben*, Államvizsgadolgozat, 2003

[**7**] http://www.mapleapps.com

[**8**] http://www.mathforum.org/advanced/robertd/catalan.html

Tárgymutató