SZAKDOLGOZAT

Fodor Zsolt

Debrecen
2011

Debreceni Egyetem
Informatika Kar

DINAMIKUS PROGRAMOZÁSRÓL

KÖZÉPISKOLAI SZAKKÖRÖN

Témavezető:

Dr. Papp Zoltán Lajos

egyetemi adjunktus

Készítette:

Fodor Zsolt

informatika szakvizsga

Debrecen
2011

TARTALOMJEGYZÉK

BEVEZETÉS

Úgy emlékszem, dinamikus programozásról tanárként hallottam először. Ez nem azt jelenti, hogy ne ismertem volna dinamikus programozás eszközeivel megoldható feladatokat, de nem azonosítottam azokat külön kategóriaként. Ha jobban meggondolom, már tanárrá válásom küszöbén, a matematika államvizsgán is ilyen feladatot vettem elő, amikor kedvenc kombinatorikai problémámat kérdezték. A probléma megoldása az ismétlés nélküli kombinációk számának meghatározása volt rekurzív összefüggés segítségével. Van, aki ezt a problémát nem tekinti igazi dinamikus programozási feladatnak. Vitatkozni nem akarok senkivel, azonban tapasztalatom azt mutatja, hogy az ismétlés nélküli kombinációk kapcsán megismerhető a dinamikus programozás teljes eszközkészlete. Külön szépsége a dolognak, hogy megtárgyalása közben szinte magától adódnak azok a felismerések is, amelyeknek egy részét bizonyítás nélkül, „készen” kapják meg matematika órán.

A hétköznapokban sok optimalizálási probléma adódik, amelyek megoldására ragyogó eszköz a dinamikus programozás. Ezért nem véletlen, hogy az eltelt évek során sok nagyszerű dinamikus programozási probléma került elő a programozási versenyeken is. Ezek hatására nem csak a versenyeken, hanem az arra készítő szakkörökön is külön kategóriává nőtte ki magát. Ez a feladattípus tapasztalatom szerint nem számít könnyűnek a diákok számára, ezért fontos vele kiemelten foglalkozni. Ha megfelelő sorrendben, kellően sok feladattal találkoznak a versenyzők, éles helyzetben is jó eséllyel felismerik az ide tartozókat, ha pedig sikerült felismerni, akkor megfelelő rutin birtokában meg is tudják azokat oldani.

Aki az előző bekezdés alapján arra gondol, hogy a feladatok megoldása csak azok számára lehetséges, akik a programozásban jártasak, téved. A feladatok általában önmagukban is érdekes problémákat hordoznak, amelyek gyakran egy táblázatkezelő programmal is leküzdhetők.

A szakdolgozatban az volt a célom, hogy részben magamnak, részben érdeklődő kollégáimnak és diákjaimnak megmutassam a témakör egy lehetséges feldolgozási módját. Az ismereteket példák megoldása során igyekeztem átadni. *A feladatokat gyakran nem eredeti formájukban közlöm, hanem céljaimnak megfelelően módosítva, részben a könnyebb szövegezés, részben a felesleges technikai részletek elhagyása miatt.* A feladatok forrására sajnos nem minden esetben emlékszem, így fordulhat elő, hogy van, amikor elmarad azok

megjelölése.

A dolgozat olvasója hiányolhatja, hogy nem mellékeltem tényleges megoldásokat (futtatható programokat, táblázatokat). E kérdésben döntésem tudatos. A feladatok megtárgyalási módját, segítő kérdéseket, ötleteket, algoritmusokat akartam rögzíteni. Úgy vélem, hogy minden további „magánügy”, az adott diáknak, kollégának kell megküzdenie azért, hogy az elvi elképzelésekből működő program szülessen.

1. A DINAMIKUS PROGRAMOZÁS HELYE A SZAKKÖRÖN

A gimnáziumi informatika tananyagban érdemi módon nem szerepel az algoritmizálás. Az a néhány óra, ami arra fordítható, nem feltétlenül elegendő arra, hogy az érdeklődést felkeltse, arra még kevésbé, hogy az ezen területen igazán tehetségesek felismerését lehetővé tegye. Algoritmizálással komolyabb formában csak önkéntes jelentkezés alapján szervezett szakkörön találkoznak a diákok. A szakköri jelentkezők száma sajnos nem teszi lehetővé a korosztály szerinti bontást. Ez a feldolgozás sorrendjét és mélységét is érinti, hiszen az újak miatt évről-évre elő kell vennem ugyanazokat a témákat. A jobbak számára ez kicsit unalmas lehet, ezt próbálom feloldani azzal, hogy ők magyarázzanak el egyes összefüggéseket az újonnan csatlakozóknak.

A szakkör tehát spirális felépítésű (4 évnyi ciklust szemlélve), adott éven belül pedig – feladattípusaiban – a Nemes Tihamér verseny és az OKTV fordulóihoz igazodik.

Főbb témakörök

- Matematikai logika
- Programozási tételek
- Mohó algoritmusok
- Dinamikus programozás
- Gráfok
- Visszalépéses algoritmusok

2. DINAMIKUS PROGRAMOZÁS – TECHNIKAI MEGALAPOZÁS

A diákokat nem lehet előzmények nélkül megismertetni a dinamikus programozással. Ha lehetséges, akkor olyan módon kell a kérdéshez közelíteni, hogy lehetőség szerint ne direkt módon oktassuk, hanem maguk fedezzék fel azt. Induljunk ki a többség számára ismert tudáselemekből és az ismert eszközök alkalmazásából, majd a maga természetességében tegyük meg a megfelelő előrelépést.

A cél természetesen a bevezető feladatokban is az, hogy eljussunk a megoldást adó rekurzív összefüggés megfogalmazásáig, majd annak közvetlen alkalmazásával vagy – adott esetben – a táblázatkitöltés módszerének felhasználásával a probléma által igényelt érték megadásáig.

2.1. FAKTORIÁLIS

A szakkörre járó diákok – tapasztalatom szerint – már általános iskolában megismerkednek a faktoriális fogalmával, ezért van mire építenünk. A kiszámítás egyik módját is ismerik, sőt, számukra általában csak egy jelölésről és az ahhoz tartozó számítási módszerről van szó.

Feladat: számítsuk ki 7! értékét!

Hagyományos megközelítéssel össze kell szorozzuk a számokat 1-től 7-ig. Végig is mondják a diákok: 1, 2, 6, 24, 120, 720, 5040.

A másik megközelítés már szokatlan számukra, amely a

$$n! = \begin{cases} n(n-1)!, \text{ ha } n > 0 \\ 1, \text{ ha } n = 0 \end{cases}$$

meghatározáson alapul. Ezt a kiszámítási módot nem kell elárulni a diákoknak, el is játszható a következő formában: ha nem tudod 7! értékét, kérdezd meg a szomszédodtól 6! értékét és szorozd meg 7-tel! Ha ő nem tudja 6! értékét, megkérdezi 5! értékét a szomszédjától és így tovább, egészen addig, amíg el nem jutunk a 0! kifejezésig. Azt a legtöbben nem ismerik, mivel: „Hogy is lehetne az első 0 darab pozitív egész számot összeszorozni?” „Nincs is értelme.” E kijelentésekkel természetesen nem érthetünk egyet, de

bővebb magyarázatba sem érdemes fogni. Egyszerűen azt mondjuk a diákoknak, hogy valahol véget kell vetni ezeknek a kérdéseknek, ezért a 0! kapcsán kinyilatkoztathatjuk, hogy annak értéke 1. Természetesen matematika órán vagy későbbi feladatok megoldásánál megtapasztalhatjuk, hogy szükségszerű volt ez a definiálás.

A későbbi feladatok miatt célszerű itt megismerni egy függvényszerű jelölési módot, mely szerint:

a) $fakt(n) = 1 \cdot 2 \cdot ... \cdot n$

b) $fakt(n) = n \cdot fakt(n-1),\ ha\ n > 0,\ fakt(0) = 1$

A fentiek megértéséhez és kipróbálásához nincs szükség programozási eszközre, lényegében egy táblázatkezelő programmal átültethetők a gyakorlatba.

Amikor szép sorban kiszámítjuk 1!, 2!, 3!, … 7! értékét, az a táblázatkitöltő módszer előfutára, amikor pedig megkérdezzük mástól a korábbi értéket (a rekurzív definíciót használjuk), az a rekurzív megoldás felé visz.

2.2. FIBONACCI SZÁMOK

A Fibonacci számokat sok diák ismeri, a hozzá tartozó klasszikus történetet azonban az utóbbi években kevesebben. Feladatként tálalva a mese előadása közben kideríthető, hogy mekkora az ismertsége. Ha nem ismerik, akkor célszerű a problémát lépésről lépésre leküzdeni, mivel nem megoldhatatlan a 14-16 éves korosztály számára. Megfelelő tanári irányítással biztosan célba lehet érni. Javaslom a klasszikus változat használatát, mivel az módot ad a teljes végiggondolásra és a használt összefüggést szisztematikus munkával tudjuk felépíteni.

2.2.1. FIBONACCI SZÁMOK – ALAPFELADAT

A feladat: elhatározzuk, hogy nyulakat fogunk tenyészteni, ezért vásárolunk egy újszülött nyúlpárt. A nyulak két hónap alatt válnak termékennyé. A termékeny nyúlpároktól minden hónapban egy újabb nyúlpár születik. Határozzuk meg, hogy az első év végén hány nyúlpárunk lesz, ha feltételezzük, hogy egy sem pusztul el a vizsgált idő alatt! [3]

A megoldást praktikus a nyúlpárok számát rögzítő táblázattal megkeresni.

	1. hó	2. hó	3. hó	4. hó	5. hó	6. hó	7. hó	8. hó	9. hó	10. hó	11. hó	12. hó
fiatal	1	0	1	1	2	3	5	8	13	21	34	55
nemzőképes	0	1	1	2	3	5	8	13	21	34	55	89
összes	1	1	2	3	5	8	13	21	34	55	89	144

A fenti táblázatból leolvasható, hogy fiatal nyulak a nemzőképes nyulaktól születnek, míg nemzőképes nyulak az előző hónap fiatal és nemzőképes nyulainak összegeként állnak elő.

Ha a megoldást táblázatkezelővel adjuk meg, akkor könnyedén kiszámítható az egymást követő Fibonacci számok hányadosa, amely az úgynevezett aranymetszés értékéhez (kb. 1,618) közelít. Érdemes ennek kapcsán néhány mondat erejéig a művészetre utalni, de a természeti és építészeti vonatkozásokat is meg lehet említeni. Ha a szakköri diákok fogékonyak erre, akkor házi feladatként is utánajárhatnak ennek.

Ne mulasszuk el megemlíteni, hogy a Fibonacci számok meghatározására zárt formulát is találtak, ez az Binet-formula: $Fib(n) = \frac{\left(1+\sqrt{5}\right)^n - \left(1-\sqrt{5}\right)^n}{\sqrt{5}\cdot 2^n}$. A formula helyességét – a diákok tudásszintjének megfelelően akár egy program írásával, akár táblázatkezelő program használatával igazolhatjuk. Programírás esetén érdemes az n-dik hatvány függvény megírásával egy kis időt tölteni, hiszen a triviális (n-szer választom szorzótényezőnek az alapot) helyett hatékonyabban is meg lehet oldani a feladatot. A megoldás közben megbeszélhető a háttér számrendszerváltás (tízesből kettesbe) is.

2.2.2. VÁLTOZATOK FIBONACCI SZÁMOKRA

A Fibonacci számos problémának sok változata ismert, hiszen még csak nem is Fibonacci találta ki azt. Először két indiai matematikus írta le: „Hányféleképpen lehet rövid és hosszú szótagokkal kitölteni egy adott időtartamot, ha egy hosszú szótag két rövidnek felel meg?” Dolgozatomban több változatot is bemutatok.

2.2.2.1. BACILUSOK

1993/1994 NTOKSZTV. 2. kategória, 2. forduló, 3. feladat [4][5.1]

Egy bacilusfajta a következő jellegzetességekkel rendelkezik:

- keletkezése után egy órával szaporodóképessé válik;
- minden (szaporodóképes) óra végén osztódással szaporodik; a keletkező két új bacilus közül az egyik 'öreg' – ez megőrzi korát, a másik 'fiatal' – ez csak egy óra múlva lesz szaporodóképes;
- a szaporodás elhanyagolható idő alatt megy végbe.

Ha adott pillanatban egy addig csíramentes környezetbe kerül egy újszülött bacilus, akkor hány bacilus lesz a tenyészetben N (N>=1) óra múlva? A program olvassa be a billentyűzetről N értékét, majd írja ki óránként a bacilusok számát!

Ebben a feladatban a klasszikus megfogalmazásnak egyszerű átiratával találkozunk. Ha otthoni munkának adjuk ki, a diákok fel fogják ismerni, hogy a Fibonacci számokat rejti a megfogalmazás.

2.2.2.2. LÉPCSŐFOKOK

Egy n lépcsőfokból álló lépcsősor tetejére szeretnénk feljutni. Egy lépéssel egy vagy két lépcsőfokkal kerülhetünk feljebb. Adjuk meg, hogy hány különböző módon juthatunk fel a lépcsősor tetejére!

Érdekes megfigyelni, hogy bár ez a feladat jobban tetszik a diákoknak, hiszen kézzel fogható, természetes megfogalmazás – mégis több nehézséget okoz a megoldása. Ennél a változatnál célszerű már a megbeszélés elején a rekurzív összefüggés felé terelni a gondolataikat. A megoldás alapötlete az a kérdés, hogy a lépcső tetejére mely lépcsőfokokról léphetünk? Mivel egy lépésben egy vagy két lépcsőfokkal juthatunk feljebb, ezért a közvetlen alatta levő, vagy a kettővel alatta levő lépcsőfokokról. Legyen $L(n)$ az a függvény, amely megadja, hogy az *n*-edik lépcsőfokra hányféleképpen juthatunk fel! Akkor $L(n) = L(n-1) + L(n-2)$. Érdemes visszautalni a faktoriális kiszámításának rekurzív

módjára és tisztázni azokat a feltételeket, amelynél ez az összefüggés alkalmazható. Nyilvánvaló, hogy $n \geq 2$ esetén nincs gond az alkalmazással, hiszen a *0*. lépcsőfok a lépcső legalja. Ebből azonnal adódik, hogy mely n értékekre kell külön meghatározni a függvény értékét. Kezdeni egyféleképpen tudjuk, ezért $L(0) = 1$, de az első lépcsőfokra is csak egyféleképpen juthatunk fel (egyet lépünk a legaljáról), ezért $L(1) = 1$. Ha megvizsgáljuk az összefüggést, akkor valóban ugyanahhoz jutottunk, mint amely a Fibonacci számokat határozta meg.

2.2.3. MÓDSZEREKRŐL A FIBONACCI SZÁMOK KAPCSÁN

2.2.3.1. MELYIK A JOBB MÓDSZER?

A Fibonacci számok meghatározásánál úgy látszik, hogy a táblázat-kitöltéses (sorkitöltéses) megoldások nem különböznek egymástól, hiszen mindkét módszer alkalmazása során minden érték meghatározására szükség van és természetesen meg is határozzuk azokat. Érdemes megnézni alaposabban a két megoldási módot.

Táblázatkitöltés

```
// n. szám meghatározása
Fib[1]:=1; Fib[2]:=1
Ciklus i:=3..n
     Fib[i]:=Fib[i-1]+Fib[i-2]
Ciklus vége
```

Rekurzió

```
Függvény Fib(n)
     Ha n>2 akkor Fib:=Fib(n-1)+Fib(n-2)
     Különben Fib:=1
Függvény vége
```

Ha alaposabban megnézzük a két fenti számítási módot, a diákok lényegi különbséget nem ismernek fel. Alakítsuk át az elsőt, hogy formailag is megegyezzen a kettő és egészítsük ki egy olyan sorral, amely az algoritmus végrehajtása során megszámolja az elvégzett összeadások számát.

Táblázatkitöltés

```
// n. szám meghatározása
osszeadas:=0
Ciklus i:=1..n
     Ha i>2 akkor
          Fib[i]:=Fib[i-1]+Fib[i-2]
          osszeadas:=osszeadas+1
     Különben Fib[i]:=1
Ciklus vége
// az összeadások száma n=10 esetén 8 lesz
```

Rekurzió

```
// az osszeadas globális változó, 0 kezdőértékkel
Függvény Fib(n)
     Ha n>2 akkor
          Fib:=Fib(n-1)+Fib(n-2)
          osszeadas:=osszeadas+1
     Különben Fib:=1
Függvény vége
// az összeadások száma n=10 esetén 54 lesz
```

A jelentős különbség okára nagyon könnyen rá lehet világítani, csak azt kell észrevenni, hogy az első esetben minden egyes összeadás egy újabb értéket számít ki a sorozatból, míg a másodikban mindig csak két részre bontjuk az aktuálisan kiszámítandó összeget. A két részre bontás az 1 értéknél fog befejeződni, tehát az 55 értéket 54 összeadással lehet előállítani, ha az 1 érték szolgál alapként.

2.2.3.2. VALÓBAN A „JOBB" MÓDSZER A JOBB?

Az előző pontban megállapítottuk, hogy a táblázatkitöltő módszer gazdaságosabb. Érdemes kitűzni egy olyan feladatot a diákok számára, amely bizonyos szempontból megingathatja az előző megállapításba vetett hitüket.

Egy n lépcsőfokból álló lépcsősor tetejére szeretnénk feljutni. Egy lépéssel **két** vagy **három** lépcsőfokkal kerülhetünk feljebb. Adjuk meg, hogy hány különböző módon juthatunk fel a lépcsősor tetejére!

A korábbi feladat leírását csak annyiban módosítottuk, hogy egy helyett három lépcsőfokot ugorhatunk. A diákok gyorsan felismerik a különbséget és megállapítják a $L(n) = L(n-2) + L(n-3)$ összefüggést. Természetesen csak $n > 2$ esetén alkalmazható, így az első három értéket nekünk kell megadni. Tehát $L(0) = 1$, $L(1) = 0$, $L(2) = 1$. A táblázat-kitöltéses módszernél használt képletet alig változtatva gyorsan át tudjuk ültetni azt a

gyakorlatba táblázatkezelővel. Amit észre kell vennünk, hogy van – a végeredmény szempontjából – feleslegesen kiszámított cella, az utolsó előtti. Mondhatjuk, hogy ez a megoldás pazarló. Ugyan csak egy érték volt felesleges, de könnyen tudunk olyan feladatot konstruálni, amelynél több ilyen lesz. Ráadásul bonyolult esetben nem is látjuk előre, hogy melyek lesznek feleslegesek.

Ha a rekurzív összefüggést nézzük, akkor – természeténél fogva – nincs olyan érték, amelyet feleslegesen határoznánk meg. Ha tudnánk olyan megoldást találni, amelyik rekurzív módon dolgozik, ugyanakkor magában foglalja a táblázatkitöltő módszer előnyét, az lenne az ideális, különösen azért, mert sok feladatnál könnyebb megfogalmazni a megoldást egy rekurzív függvénnyel.

Könnyen lehet, hogy ezen a ponton lesz diák, aki önállóan kimondja a lényeget: „Jegyezzük meg azokat az értékeket, amelyeket kiszámoltunk!” „Csak azokat számoljuk ki, amelyeket még nem ismerünk!”

Célszerű a korábban használt, az összeadások számát meghatározó algoritmust módosítni, hiszen a cél az, hogy lássuk, hogy az ott tapasztalt problémát hatékonyan orvosolja ötletünk.

```
osszeadas:=0 // az osszeadas globális változó, 0 kezdőértékkel
Fib[]:=-1
/*
globális tömb, a -1 jelzi, hogy még nem határoztuk meg az adott elem
értékét
TFib[1]:=1; TFib[2]:=1
*/

Függvény Fib(n)
     Ha TFib[n]=-1 akkor
          TFib[n]:=Fib(n-1)+Fib(n-2)
          osszeadas:=osszeadas+1
     Elágazás vége
     Fib:=TFib[n]
Függvény vége
// az összeadások száma n=10 esetén 8 lesz
```

Ugyan ekkor is mondhatjuk, hogy többet kell dolgoznia a programnak, hiszen a tömbelemek értékének beállítása pluszmunkát jelent, de még ennek beleszámításával sincs akkora különbség a két módszer között, hogy gazdaságossági okokból elvessük a rekurzív megoldást. Ezt a módosított formában megfogalmazott rekurzív eljárást tárolt rekurziónak nevezhetjük.

Vegyük észre, ezáltal már három módszer van a kezünkben, a táblázatkitöltés, a rekurzív és a tárolt rekurzív módszer. Hogy melyiket választjuk egy feladat megoldása során, azt a feladat jellege határozhatja meg azáltal, hogy melyik formában tudjuk könnyebben megadni az algoritmust. A rekurzió és a tárolt rekurzió között pedig az adatmennyiség segíthet dönteni. Ha kevés adatról van szó, nem biztos, hogy érdemes a tárolással foglalkozni.

2.2.3.3. FIBONACCI SZÁMOK A VERSENYFELADATOKBAN

A Fibonacci jellegű algoritmusokra látható szép példa az NTOKSZTV-n a 2009/2010-es tanév első és második fordulójában.

2009/2010. NTOKSZTV 1. kategória, 1. forduló, 3. feladat: **Járda** [2.1]

Egy N egység méretű járdát 1 és 2 méretű lapokkal szeretnénk kikövezni.

A. Add meg, hányféleképpen lehet kikövezni az N=1, N=2, N=3 hosszú járdákat!

B. Rajzold le az összes lehetséges kikövezést N=3 esetén!

C. Add meg, hányféleképpen lehet kikövezni az N=5 egység hosszú járdát!

D. Add meg, hányféleképpen lehet kikövezni az N=6 egység hosszú járdát!

E. Add meg, hányféleképpen lehet kikövezni az N=10 méretű járdát!

A legkisebb korcsoport feladatánál szó szerint alkalmazható a megismert algoritmus, ha felhívjuk a figyelmet arra, hogy az 1 méretű lap az 1 egységgel visz bennünket előre, a 2 méretű pedig 2 egységgel. Lényegében pont úgy, mint a 2.2.2.2 Lépcsőfokok alapfeladatnál.

A nagyobbak feladata már több, mint az ismeretek mechanikus alkalmazása. (Ezzel természetesen nem azt mondom, hogy elvárható egy 8. osztályos diáktól az algoritmus ismerete, bár az érettebb gondolkodásúak számára ez nem jelentene problémát.)

2009/2010 NTOKSZTV 2. kategória, 1. forduló, 5. feladat: **Járda** [2.2]

Egy N egység méretű járdát 1, 2 és 3 méretű lapokkal szeretnénk kikövezni.

A. Add meg, hányféleképpen lehet kikövezni az N=1, N=2, N=3 hosszú járdákat!

B. Rajzold le az összes lehetséges kikövezést N=3 esetén!

C. Add meg, hányféleképpen lehet kikövezni az N=5 egység hosszú járdát!

D. Add meg, hányféleképpen lehet kikövezni az N=6 egység hosszú járdát!

E. Add meg, hányféleképpen lehet kikövezni az N=10 méretű járdát!

A $J(n) = J(n-1) + J(n-2) + J(n-3)$ összefüggés felismerése a jobbaktól elvárható előismeretek nélkül is, de a korábbi feladatokat feldolgozó diákok mindegyike képes kell legyen a megoldásra.

Ugyanebben az évben a második forduló tartalmazta ezen feladatok „gépesített" változatát is.

2009/2010. NTOKSZTV 1. kategória, 2. forduló, 3. feladat: **Járdakövezés** [2.3]

Egy N $(1 \le N \le 80)$ egység hosszú járdát 1 és 2 méretű lapokkal szeretnénk kikövezni. Hányféleképpen lehet ezt megtenni?

Készíts programot (`lapok.pas`, …), amely kiszámítja, hogy egy N egység hosszú járdát hányféleképpen lehet kikövezni 1 és 2 méretű lapokkal!

A feladat látszólag nem nehezebb, mint a papíros változat, azonban a feladatban foglalt méretbeli korlátok elgondolkodtatók. Ennek kapcsán sokkal inkább az eredmény nagyságrendje ($N = 80$ esetén 10^{16}) által okozott technikai jellegű probléma jelenti az igaz nehézséget, mint maga a feladat.

2009/2010. NTOKSZTV 2. kategória, 2. forduló, 4. feladat: **Járdakövezés** [2.4]

Egy N egység hosszú járdát 1, 2 és 3 méretű lapokkal szeretnénk kikövezni. Hányféleképpen lehet ezt megtenni?

Készíts programot (`kovezes.pas`, …), amely kiszámítja, hogy egy N egység hosszú járdát hányféleképpen lehet kikövezni 1, 2 és 3 méretű lapokkal!

A 2. kategóriában is a megfelelő típus kiválasztása a legnehezebb feladat, mivel a $N = 70$-hez tartozó eredmény nagyságrendje 10^{18}.

2.3. ISMÉTLÉS NÉLKÜLI KOMBINÁCIÓ

Természetesen nem pőrén kell a címbeli fogalommal, az $\binom{n}{k}$ kifejezéssel, értékének kiszámításával, a binomiális együtthatókkal, azok tulajdonságaival foglalkozni, de nem érdemes elhallgatni a matematikai vonatkozásokat sem az érdeklődő diákok elől.

2.3.1. ISMÉTLÉS NÉLKÜLI KOMBINÁCIÓRA VEZETŐ FELADATOK

Tekintsünk egy sakktáblát! A tábla bal alsó sarkában helyezzünk el egy királyt! A sakktábla egy adott pontjába hány különböző úton juthat el a király, ha egy lépésben csak jobbra és felfelé léphet?

A fenti feladat könnyen érthető, kisebb diákok számára is kézzel fogható. Praktikus okokból másképpen szoktam kitűzni a feladatot.

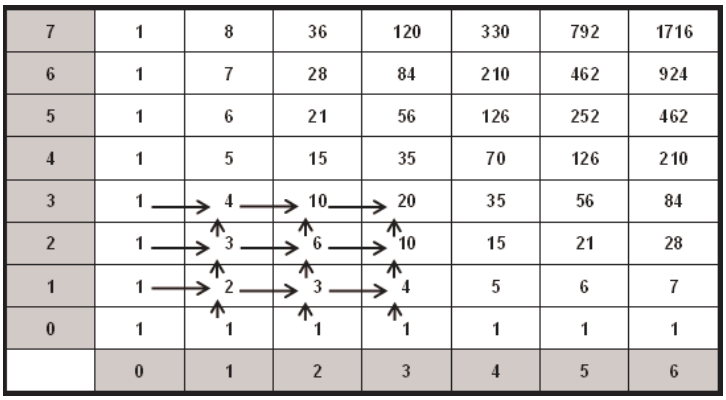
Tekintsük a koordinátarendszer első síknegyedét! Az origóból elindul egy hangya, amely csak jobbra és felfelé tud haladni a rácsvonalakon. Hány különböző úton juthat el az (x, y) pontba?

Utóbbi változat valamivel nagyobb előzetes tudást kíván, ugyanakkor a megoldás több lehetőséget nyújt a következtetések levonására.

Javaslom, hogy a diákok próbálják önállóan meghatározni az értéket a (2, 3), majd a (3, 5) pontok esetén. Megfigyelhető, hogy egyszerűbb esetben fejben vagy az útvonalak tényleges berajzolásával próbálkoznak, a nagyobb koordinátájú pontoknál a jobb diákok már megpróbálják kiszámítani a megjelölt pozícióba való eljutások számát. A kiszámítás lényegében a táblázatkitöltős módszert jelenti. Nem biztos, hogy tervszerűen, a kiszámítási módot tudatosan alkalmazva dolgoznak, de csak egy kicsi hiányzik ahhoz, hogy a lényeget kimondják.

A kiszámítási mód táblázatkitöltéssel:

A 0. sor és 0. oszlop minden helyére az 1 érték kerül, hiszen csak folyamatosan jobbra, illetve folyamatosan felfelé haladva érhetjük el ezeket az elemeket. Ezt követően soronként lentről felfelé, azon belül balról jobbra haladva a bal és az alsó szomszéd értékének felhasználásával (azok összegeként) számíthatjuk ki egy-egy mező értékét.



Kiszámítási mód rekurzióval:

Legyen az F függvény az, amely megadja az eltérő utak számát. Ez a függvény nyilvánvalóan függ a cél sor és oszlopszámától. Tehát az $F(o,s)$ értékét kell meghatározni. Az előző számítási módnál használt összefüggést alkalmazva adódik a képlet:

$$F(o,s) = F(o-1,s) + F(o,s-1)$$

A rekurzív függvény alkalmazásához szükséges, hogy valahol megállítsuk ezt a visszafelé hivatkozást. Ha elértük a 0. sort vagy a 0. oszlopot, akkor a függvény értéke 1 lesz.

$$F(o,s) = \begin{cases} F(o-1,s) + F(o,s-1), \text{ha } s \cdot o > 0 \\ 1, \text{ha } s \cdot o = 0 \end{cases}$$

A problémát megoldhatjuk mind a táblázatkitöltő módszerrel (táblázatkezelővel és valamely programozási nyelven), de érdemes programot írni tárolt rekurzióra is.

```
TF[_,_]:=-1
/*
globális tömb, a -1 jelzi, hogy még nem határoztuk meg az adott elem
értékét
*/
Függvény F(s, o)
     Ha TF[s,o]=-1 akkor
          Ha s*o=0, akkor TF[s,o]:=1
          különben TF[s,o]:=F(s-1,o)+F(s,o-1)
     Elágazás vége
     TF:=TF[s,o]
Függvény vége
```

Tegyük meg azokat a megállapításokat, amelyek a matematika oldalán hasznosak lehetnek.

Egy adott (s,o) pontba annyiféle úton juthatunk el, ahogyan az s+o darab útszakaszból ki tudjuk választani azt az s darab szakaszt, amelyen felfelé haladunk. Az ilyen választást a matematika nyelve ismétlés nélküli kombinációnak nevezi és – a mi elnevezéseinket használva – $\binom{s+o}{s}$ alakban jelöli. Ez természetesen ugyanaz az érték, mint az ugyanannyi lépésből vízszintesen megtett szakaszok száma, tehát $\binom{s+o}{s}=\binom{s+o}{o}$

A kiszámítási szabály alapján újabb megállapítást tehetünk: $\binom{s+o}{s}=\binom{s+o-1}{s}+\binom{s+o-1}{o}$

Ezen megállapításoknak a diákok akkor is szoktak örülni, ha már ismerik matematikából, de akkor is, ha még nem tanulták.

Kiíratva a táblázatot, felismerhető benne a Pascal háromszög, amelyet sok általános iskolás diák ismer.

3. TÁBLÁZATKITÖLTÉSRE KITALÁLT FELADATOK

3.1. VÉGTELEN MÉRETEK

E részben olyan jellegű feladatokat érintek, amelyekben az óriási adatmennyiség meglepi a diákokat, ezért ha előzmény nélkül kerülnek szembe ezekkel a problémákkal, ritkán adnak jó megoldást. Ha a korábbi feladatokat részletesen átbeszéltük, akkor érzik, hogy dinamikus programozási feladatról lehet szó. Innen már csak egy lépés, hogy a két megoldási mód közül a rekurzió nélküli táblázatkitöltés mellett tegyék le a voksukat.

3.1.1. OSZTHATÓSÁG

ELTE, ACM helyi forduló, 2000 [5.2]

Adott pozitív egész számok meghatározott sorozata ($s_1, s_2, ..., s_n$) és egy pozitív egész szám (K). A fenti számsor egyes elemei közé írhatunk-e olyan módon összeadás és kivonás jeleket, hogy az így előálló kifejezés értéke osztható legyen K-val?

A feladatot előzmény nélkül tárgyalva, egy példaként felírt, 7-8 számból álló sorozatot látva, az első ötlet szinte kivétel nélkül az összes lehetséges eset kipróbálására vonatkozik. Ekkor célszerű kiszámoltatni velük, hány eset megvizsgálására van szükség. Ez az érték 2^n, amely már kis n érték esetén is óriási esetszámhoz vezet, így ez a javaslat könnyen elvethető.

A következő hasznos megállapítás, mely szerint $s_i \geq K$ számok helyett használjuk az $s_i^* = s_i \bmod K$ számokat, hiszen az oszthatóság szempontjából egyenértékűek. Innen természetes módon következik a lépés, egyetlen közbülső lépés, tehát egyetlen részösszeg $r_i = \sum_{j=1}^{i} \pm s_j$ se haladja meg a K értékét, sőt, teljesüljön, hogy $-K < r_i < K$. Az első látásra meglepő $-K$ érték azért kerül elő, mert a kivonások eredményeként negatív részösszegek is előállnak. Általában van diák, aki felismeri, hogy elegendő $0 \leq r_i < K$ részösszegeket figyelni, mivel a negatív számokhoz a K értékét hozzáadva ebbe az intervallumba jutunk.

Itt megfogalmazható, hogy a kérdésre akkor adhatunk igen választ, ha a minden számot

tartalmazó – $\bmod K$ -val számolt – összeg 0 értéket ad.

K=	7					
eredeti	23	41	32	5	51	23
maradék	2	6	4	5	2	2
6			3-4>-1>7	4-5>-1>6		
5			1+4>5	0+5>5		
4			1-4>-3>4	6+5>11>4		
3		2-6>-4>3		5+5>10>3		
2	2			0-5>-5>2		
1		2+6>8>1		6-5>1		
0			3+4>7>0	5-5>0		

Ha nincs önálló javaslat a folytatásra, akkor arra érdemes irányítani a figyelmet, hogy ne pusztán teljes számsor esetén adjunk választ a kérdésre, hanem minden egyes szám után állapítsuk meg, hogy az addigi lehetséges összegek között van-e K -val osztható. Készítsünk egy táblázatot, amelyben minden egyes lépést tüntessünk fel! A táblázatban a kialakult részösszeget elegendő lehet akár az $\times$ karakterrel jelölni, azonban az utólagos ellenőrizhetőség miatt praktikus részletesebben feljegyezni, hogy miért is töltöttünk ki egy sort. A megoldás során észrevehetjük, hogy a helyes válaszadáshoz nem minden esetben kell a teljes számsorozaton végighaladni, megállhatunk az első olyan oszlopnál, amelynek minden sorába eljuthatunk. Ebből automatikusan következik, hogy a 0 összeg sora (ez a táblázat legalsó sora) az utolsó oszlopban sem lesz üres. Ha nem alakul ki ilyen oszlop, akkor az utolsó szám oszlopában a 0 sora fogja megadni a választ.

A diákok felvethetik, hogy itt is minden részösszeget „továbbszámoltunk”, azaz az újabb számot hozzá is adtuk és ki is vontuk, azaz minden lépésben duplázódik az esetszám, tehát mégis megvizsgáltunk minden lehetséges esetet. (Ha a diákok nem vetik fel, akkor mi magunk vegyük elő a kérdést.) Felületes szemlélő igazat adhat a felvetőnek, azonban nézzük meg: minden egyes oszlopban legfeljebb K helyről folytathatjuk két irányban. Tehát egy lépés legfeljebb $2K$ műveletet igényel, míg a táblázat teljes kitöltése ennek n -szeresét, azaz, $n2K$ -t. Ha összevetjük ezt a feladat elején felvetett 2^n értékkel, akkor láthatjuk, hogy a számok számának (n) növelésével egyre inkább eltörpül a $n2K$ értéke a 2^n mellett. Ezt konkrét K érték esetén érdemes kiszámítani. Az előbb vizsgált esetre vonatkozóan tartalmazza az itt látható táblázat az értékeket.

K=	7								
n=	1	2	3	4	5	6	7	8	9
n2K=	14	28	42	56	70	84	98	112	126
2^n=	2	4	8	16	32	64	128	256	512

Mekkora méretű tömböt / tömböket kell ezen feladat megoldása során használni? Egyértelműnek látszik, hogy K sorból és n oszlopból kell állnia a tömbnek, tehát összesen nK értéket kell tartalmaznia. Azonban láthatjuk, hogy minden egyes lépésben meghatározunk egy újabb oszlopot, és ennek az oszlopnak az értéke csak az őt megelőző oszloptól függ. Ezért elegendő összesen $2K$ értékkel dolgoznunk.

```
uj[]:=false // az éppen aktuális adatokat tartalmazó, K elemű tömb
regi[]:=false
/* az előző lépésben meghatározott adatokat tartalmazó, K elemű tömb
a tömbök elemeit false-ra állítjuk */

s[] // n elemű tömb, értékei a számok
uj[s[1] mod K]:=true
Ciklus i:=2..n
     regi:=uj;
     uj[]:=false
     Ciklus j:=1..K
          Ha regi[j], akkor
               uj[(j+s[i]) mod K]:=true
               uj[(j-s[i]) mod K]:=true
          Elágazás vége
     Ciklus vége
Ciklis vége
Ha uj[0], akkor Ki: 'Lehetséges' különben 'Nem lehetséges'
/* ez az algoritmus nem vizsgálja, hogy korábban megállapítható-e a
válasz *
```

Módosítsuk egy kicsit ezt a feladatot!

Adott pozitív egész számok meghatározott sorozata $(s_1, s_2, ..., s_n)$ és egy pozitív egész szám (K). A fenti számsor egyes elemei közé írhatunk-e olyan módon összeadás és kivonás jeleket, hogy az így előálló kifejezés értéke osztható legyen K-val? Ha lehetséges, akkor adjunk is meg egy ilyen műveletsorozatot!

Ebben az esetben nem állhatunk meg az első teljesen kitöltött oszlopnál, hiszen akkor még nem tudjuk, hogy a továbbiakban hogyan kell a nem vizsgált számok elé elhelyezni a műveleti jelet.

Vegyük észre, hogy ebben az esetben már szükségünk van a nK érték eltárolására, hiszen az egyes számoknál még nem tudjuk, hogy melyikből folytatva érjük el a 0 maradékot.

Fontos, hogy erről csak akkor van értelme beszélni, ha az utolsó oszlopot is kitöltöttük. Tehát a „megoldástól”, pontosabban az előző feladatra adott válasz megadását követően, visszafelé haladva tudjuk majd meghatározni, hogy miképpen kell a műveleti jeleket kitenni. Ebből az adódik, hogy az egyes tömbelemekben olyan értéket kell eltárolni, amelyből meghatározható, honnan jutottunk oda.

Kis gondolkodás után a diákok akár önállóan is megállapíthatják, hogy az egyes pontokon elegendő műveleti jelet kell eltárolni. Rövid megbeszélést igényel, hogy melyiket, ugyanis a korábbi példát megvizsgálva láthatjuk, hogy előbb-utóbb több művelet is ugyanazon pozícióba visz. Természetesen szinte azonnal adódik a válasz, hogy ezek közül bármelyik megfelel, hiszen csak egy lehetséges utat kell megadnunk. Praktikus okokból az alábbi algoritmusban az utolsó bejegyzést fogjuk rögzíteni.

```
m[][]:=’ ’
/* műveleti jelek tömbje, n oszlopból és K sorból áll, kezdeti értéke
a szóköz karakter */

s[] // n elemű tömb, értékei a számok
m[1][s[1]]:=’+’
// az első szám előtt pozitív előjel van, technikai okokból szükséges

Ciklus i:=2..n
     Ciklus j:=1..K
          Ha m[i-1][j]!=’ ’, akkor
               m[i][(j+s[i]) mod K]:=’+’
               m[i] [(j-s[i]) mod K]:=’-’
          Elágazás vége
     Ciklus vége
Ciklus vége

Ha m[n][0]!=’ ’, akkor
     // visszakeresés
     osszeg:=0
     Ciklus i:=n..2
          e[n]:=m[i][osszeg]
          Ha e[n]=’+’ akkor osszeg:=(osszeg-s[n]) mod K
          Különben osszeg:=(osszeg+s[n]) mod K
     Ciklus vége
különben ’Nem lehetséges’
Elágazás vége
/* az e[] n elemű tömb i-edik eleme az i-edik szám elé írandó műveleti
jelet fogja tartalmazni. */
```

3.1.2. IGAZSÁGOS OSZTOZKODÁS

Ha az alábbi példát [6] az előző után házi feladatként feladjuk, a diákok minden bizonnyal az Oszthatóság részben foglaltak szerint igyekeznek megadni a megoldást.

Péter és Pál ikertestvérek. Sajnos mindenben vetélkednek és semmin sem osztoznak, így nehéz helyzetben van, aki meg akarja ajándékozni őket. Legyen születésnap, névnap vagy éppen Karácsony, csak akkor örülnek igazán, ha tudják, a másik sem kapott nagyobb értékű ajándékot, mint ők. Az ajándékok értékének ismertében határozzuk meg, hogy

- eloszthatók-e „igazságosan az ajándékok?
- ha igen, akkor hogyan?

Mit vehetünk át az előző feladat megoldásából? A műveleti jelek jelölhetik, hogy ki kapja az ajándékot, a ’+’ jelet Péterhez, a ’-’ jelet Pálhoz kapcsolhatjuk. Vajon itt is leszoríthatjuk-e a vizsgált intervallumot? Vajon itt is elegendő pozitív számokat vizsgálni? Megállapíthatjuk, hogy az intervallum az előzőhöz hasonló mértékben nem szűkíthető le, mivel itt az ajándékok teljes értékével kell dolgozni. Emiatt az első kérdés megválaszolásához is meg kell vizsgálni az összes ajándékot. A negatív számokat sem spórolhatjuk meg, mivel elképzelhető, hogy az ajándékok csak úgy oszthatók ki a feltételek szerint, hogy a közbülső lépések során a pozitív és a negatív tartományt is érintjük. Konstruáljunk ilyen példát! (Például az 1, 3, 3, 5 értékű ajándékok esetén használni kell a pozitív és negatív tartományokat is.)

Mekkora tömbre van szükség? Az előbbiek alapján n darab, összesen K értékű ajándék esetén $n(2K+1)$ a tömb mérete. A sorokat ugyanis $-K..K$ -val indexelhetjük. Kell-e ekkora? Ha belegondolunk abba, hogy mit jelent az egyenlő elosztás, akkor nyilvánvalóan nem, ugyanis a 0 értéktől az ajándékok összértékének felétől jobban eltávolodni felesleges, hiszen akkor „nem térhetünk vissza”. Ebből következik, hogy a sorindex $-(K\ div\ 2)..(K\ div\ 2)$ között változhat. Tehát pontosan $2(K\ div\ 2)+1$ sorra van szükség, mivel létezhet ezen kívül elhelyezkedő állapot, de az nem lehet kedvező számunkra.

A fenti meggondolások után elvárható a feladat „megoldása” a diákoktól, azonban érdemes rámutatni, hogy a megoldás – szemben az Oszthatóság feladattal – inkább az összes

eset megvizsgálásához közelít. Ha egy általános számsort kapunk az ajándékok értékével, akkor kevés esetben fogunk különböző utakon ugyanabba az állapotba elérni. Érdemi spórolást csak a sorok számának felére csökkentésével tudtunk elérni.

Ha azt is meg kell mondani, hogy miképpen alkotható ilyen elosztás, akkor mindegyik ajándékot követően szükségünk van egy ilyen oszlop tárolására, ami a szükséges tömbméretet $(2(K\ div\ 2)+1)N$ -re növeli.

A használt tömb meglehetősen ritkásan lesz kitöltve. Érdemes elgondolkodni azon, hogy valóban az-e a helyes megoldás, amely ilyen óriási tárigénnyel bír. (A választ később, az Egydimenziós, véges méretek című fejezetben kapjuk meg a Igazságos osztozkodás – másképp részben.)

3.1.3. DOMINÓK

1999/2000. NTOKSZTV 3. kategória, 3. forduló, 3. feladat [2.5][5.3]

Egy zsákban nagyon sok (akár több millió) dominó található. Ezekből a dominókból kell a lehető leghosszabb sort építeni a dominózás szabályainak megfelelően. A dominókat egymás után emeljük ki a zsákból és mindegyikről eldöntjük, hogy felhasználjuk-e a sorban van sem. Az utolsó dominó után meg kell mondanunk, hogy a legnagyobb hosszúságú dominósor hány elemből áll.

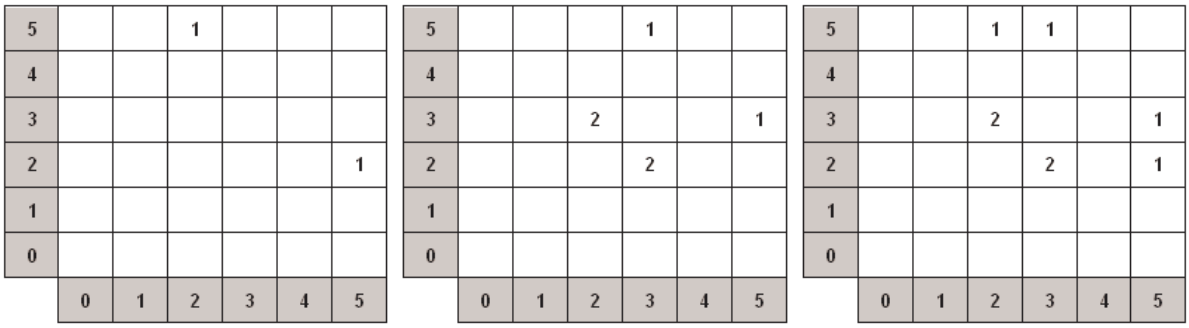
A feladat leírásában szereplő óriási dominószám orientálja a diákot, hogy nem próbálhatja ki az összes eshetőséget. Annak hangsúlyozása, hogy egy zsákból kiemelt dominóról azonnal döntenünk kell, azt sugallja, hogy – ha felismerjük, hogy ez dinamikus programozási feladat akkor – a táblázatkitöltés módszerével próbáljuk megoldani.

Vegyünk észre, hogy mi a közös egy dominóban és egy dominósorban! A továbbépíthetőség szempontjából tökéletesen megegyeznek, a sor szabad végei megfeleltethetők egyetlen dominó két felének. Tehát a (3,4), (4,1), (1,7) dominósor egyenértékű a (3,7) dominóval. Jelölésben azért megkülönböztethetjük, a fenti dominósort [3,7] módon írhatjuk le.

A feladat megoldásánál nyilván nem tudjuk, hogy melyik dominót érdemes felhasználni a folytatáshoz, ezért valamiképpen rögzítenünk kell az összes folytatásra érdemes állapotot. Ehhez minden dominó felhasználása után be kell jegyeznünk, hogy addig mely állapotok

alakulhattak ki és maximum milyen hosszúságú sorral lehetett elérni. Meg kell vizsgálni a kérdést, hogyan kaphatjuk meg a következő dominó utáni állapotot? Nyilván annak felhasználásával vagy eldobásával. Az eldobás változatlanul hagyja az eddigi értékeket, míg az (x,y) dominó például az [x,_] sorokhoz is csatlakoztatható, amelyet követően az előzőekhez képest eggyel hosszabb [y,_] sorok alakulnak ki. Nem biztos, hogy ez összességében is kedvezőbb hosszat eredményez.

A sorok, pontosabban sorok hossza egy kétdimenziós tömbbel leírható bármely dominó csatlakoztatását követően. A korábbi feladatok alapján csak azt kell eldönteni, hogy a dominószámmal egyező számú tömb kell-e vagy elegendő kevesebb. Mivel nem kell megmondani, hogy pontosan mely dominók kerültek a felhasználtak közé, ezért sejthető, hogy nem kell minden dominóhoz egy állapottömb, sőt, nyilvánvaló, hogy az előző állapottömbből megalkotható az új, tehát kettő biztosan elég. Azonban vajon egy állapottömb elég-e a válaszhoz? Ennek kimondásához, és az algoritmus rögzítéséhez érdemes néhány dominóval a gyakorlatban megcsinálni a feladat megoldását.




```
elozo[][]:=0
aktualis[][]:=0
/* a dominósorokat egy kétdimenziós tömb írja le, kezdetben minden
dominósor 0 hosszúságú */

Ciklus amíg van dominó
     Be: d // dominó beolvasása,
     Ciklus i:=0..9
          aktualis[i][d.y]:=max(elozo[i][d.x]+1, aktualis[i][d.y])
          aktualis[i][d.x]:=max(elozo[i][d.y]+1, aktualis[i][d.x])
          aktualis[d.y][i]:=max(elozo[d.x][i]+1, aktualis[d.y][i])
          aktualis[d.x][i]:=max(elozo[d.y][i]+1, aktualis[d.x][i])
     Ciklus vége
```

```
     /* az esetlegesen nagyobb értékeket az aktualis tömbből
     vissza kell másolni az előző tömbbe. */
     Ciklus i:=0..9
          Ciklus j:=0..9
               elozo[i][j]:=max(elozo[i][j], aktualis[i][j])
          Ciklus vége
     Ciklus vége
Ciklus vége
```

3.2. EGYDIMENZIÓS, VÉGES MÉRETEK

3.2.1. KIFIZETHETŐSÉG

Egy automatából szeretnénk vásárolni. Ismerjük a címleteket ($c_1, c_2, ..., c_n$), amelyeket az automata elfogad. Vajon ezen címletek felhasználásával előállítható-e az áruért járó F összeg?

Ha minden forgalomban lévő címletet elfogad, nyilván nem is lehet kérdés, a válasz igen.

Amennyiben nem szokványos címletekről van szó és a dinamikus programozás eszköztárának felhasználása nélkül kezdenénk hozzá a megoldáshoz, akkor meg kellene keresni azokat a $k_i \geq 0$ értékeket, amelyek felhasználásával a $k_1c_1 + k_2c_2 + ... + k_nc_n = F$ egyenlőség teljesül. Néhány egymásba ágyazott ciklussal találhatunk alkalmas szorzókat. Ilyen algoritmus készítése nem túl bonyolult, azonban most válasszunk más utat! Olyat, ahol a címletek többszörösei – bár bújtatva – de előfordulnak, egyúttal nem csupán azt kapjuk meg, hogy az F kifizethető-e, hanem azt is megtudjuk, hogy mely értékek fizethetők ki F -ig ilyen módon.

Az algoritmus lényege a következő. Tudjuk, hogy a 0 kifizethető. Nézzük az első címletet. Azonnal mondhatjuk, hogy a címlet és a többszörösei kifizethetők. Azonban ne így kezeljük! Ha a 0 kifizethető, akkor $0 + c_1$ is kifizethető. Nézzük meg, mi a következő kifizethető összeg! A következő a c_1. Ha az kifizethető, akkor a $c_1 + c_1$, tehát $2c_1$, majd a $2c_1 + c_1$ és így tovább. A következő címlettel is így járunk el, de ekkor már nem csak annak többszöröseit, hanem a c_1 többszöröseit is kiindulópontnak tekintjük. Sőt, általánosabban fogalmazva, minden kifizethető értékhez hozzáadjuk az aktuálisan vizsgált címlet értékét.

```
c[] // a címletek értékét tartalmazó n elemű tömb
lehet[]:=false
/* a tömb i-edik indexű elemének értéke megmutatja, hogy az i összeg
kifizethető-e; a feladat kérdésére igen lesz a válasz, ha
lehet[F]=true */

lehet[0]:=true // a 0 összeg kifizethető

Ciklus i:=1..n
     Ciklus j:=0..(F-c[i])
          Ha lehet[j] akkor lehet[j+c[i]]:=true
     Ciklus vége
Ciklus vége
Ha lehet[F], akkor Ki: 'kifizethető' különben 'nem fizethető ki'
/* mellékeredményként minden F-nél kisebb összegre is megkapjuk a
kifizethetőséget */
```

Egy automatából szeretnénk vásárolni. Ismerjük a címleteket, amelyeket az automata elfogad. Tudjuk, hogy az áru kifizethető úgy, hogy az automatának ne kelljen visszaadnia. Legkevesebb hány pénzérmét kell felhasználnunk a fizetés során?

A feladat nagyon hasonlít az előzőre, és a megoldása sem lesz lényegesen eltérő. Nyilvánvalóan nem elegendő a kifizethetőséget tárolni. Helyette azt kellene megőrizni az egyes összegek mellett, hogy az adott címlet vizsgálatáig vajon mennyi a szükséges pénzérmék számának minimuma.

```
c[] // a címletek értékét tartalmazó n elemű tömb

lehet[]:=-1
/* a tömb i-edik indexű elemének értéke megmutatja, hogy az i összeg
minimálisan hány pénzérmével fizethető ki; a feladat kérdésére a
lehet[F] értéke a válasz */

lehet[0]:=0 // a 0 összeg kifizethető 0 darab érmével
Ciklus i:=1..n
     Ciklus j:=0..(F-c[i])
          Ha lehet[j]>-1 akkor
               Ha lehet[j]=-1, akkor lehet[j+c[i]]:=lehet[j]+1
               Különben lehet[j+c[i]]:=min(lehet[j]+1, lehet[j+c[i]])
     Ciklus vége
Ciklus vége
Ki: 'a szükséges érmék minimális száma' lehet[F]
/* mellékeredményként minden F-nél kisebb összegre is megkapjuk az
érmeszám minimumát */
```

Egy automatából szeretnénk vásárolni. Ismerjük a címleteket, amelyeket az automata elfogad. Tudjuk, hogy az áru kifizethető úgy, hogy az automatának ne kelljen visszaadnia. Hogy válasszunk, hogy a lehető legkevesebb pénzérmét használjuk fel a fizetés során?

Egy kicsivel megint nehezebb válaszolni a kérdésre, mivel meg kell nevezni, hogy az egyes felhasznált címletekből hány darab szükséges. Nyilvánvaló, hogy a felhasznált darabszámot most is tárolni kell minden összegnél. Vajon mely címletekkel jutottunk oda? A kérdés megválaszolásához érdemes felidézni – ha erre mód van – az élsúlyozatlan gráfokban végzett minimális úthossz keresést. Először ott is a minimális lépésszámot határoztuk meg, majd az út megadásához megjegyeztük, hogy mely csúcsból jutottunk oda. Ennek mintájára itt is megjegyezhetjük, hogy mely összegből jutottunk az aktuálishoz, de ezzel egyenértékű a címlet sorszáma vagy maga a címlet is. Ha a memóriával akarunk spórolni, akkor a címlet sorszámát célszerű megjegyezni.

Az algoritmusnak az a része, amellyel a minimumot kerestük, érdemben nem változik, a szükséges címleteket nyilvánvalóan csak a minimális érmeszám meghatározása után, az eddigi algoritmust követően tudjuk kiíratni.

```
c[] // a címletek értékét tartalmazó n elemű tömb

lehet[].db:=-1
/* a tömb i-edik indexű elemének db mezője megmutatja, hogy az i
összeg minimálisan hány pénzérmével fizethető ki */

lehet[].cimlet:=0
/* a tömb i-edik indexű elemének címlet mezője adja meg, hogy utoljára
mely címletet használtuk fel, hogy ehhez az értékhez jussunk */

lehet[0].db:=0 // a 0 összeg kifizethető 0 darab érmével

Ciklus i:=1..n
     Ciklus j:=0..(F-c[i])
          Ha lehet[j].db>-1 akkor
               Ha (lehet[j+c[i]].db=-1)
                         vagy (lehet[j+c[i]].db>lehet[j].db+1)
               akkor
                    lehet[j+c[i]].db:=lehet[j].db+1
                    lehet[j+c[i]].cimlet:=i
               Elágazás vége
          Elágazás vége
     Ciklus vége
Ciklus vége
Ki: 'a szükséges érmék minimális száma' lehet[F].db

ertek:=F
Ciklus amíg ertek>0
     Ki: c[lehet[ertek].cimlet]
     ertek:=ertek-c[lehet[ertek].cimlet]
Ciklus vége
/* mellékeredményként minden F-nél kisebb összegre is megkapjuk az
érmeszám minimumát */
```

Egy automatából szeretnénk vásárolni. Ismerjük a címleteket, amelyeket az automata elfogad. Tudjuk, hogy az áru kifizethető úgy, hogy az automatának ne kelljen visszaadnia. Hogy válasszunk, hogy a lehető legkevesebb féle pénzérmét használjuk fel a fizetés során?

Most térjünk vissza a feladat bevezetőjében megfogalmazott kifejezéshez!

Ott a $k_1c_1 + k_2c_2 + ... + k_nc_n = F$ egyenlőség teljesüléséhez kell megkeresnünk a megfelelő $k_i \geq 0$ értékeket. Az első feladatnál a $k_i \geq 0$ értékek létezése elegendő a válaszhoz, a másodikban a $\sum_{i=1}^{n} k_i$ összeg minimuma, ennél pedig a $\sum_{i=1}^{n} \text{sgn}(k_i)$ összeg minimuma a kérdés. Utóbbira dinamikus programozással nem találnánk meg könnyen a megoldást, így célszerű a diákok figyelmét a táblázatkezelő program Solver eszközére irányítani.

3.2.2. BÉLYEGEK

1995. ACM döntő [5.4]

A postán különböző értékű bélyegek $(b_1, b_2, ..., b_n)$ vásárolhatók. Egy borítékra legfeljebb k darab bélyeg ragasztható fel. Melyik az a legkisebb összeg, amely értékben nem ragasztható bélyeg a borítékra?

A megoldáshoz eljuthatunk, ha visszavezetjük az előző feladatra. Összegként az $F = k \cdot \max(b_i)$ értéket tekintjük, hiszen ez az adott feltételekkel még biztosan kifizethető. Arra kell figyelnünk, hogy egyetlen összeg értékét előállító darabszám se lépje túl a maximálisan felhasználható bélyegszámot.

```
b[] // a címletek értékét tartalmazó n elemű tömb

lehet[]:=-1
/* a tömb i-edik indexű elemének értéke megmutatja, hogy az i összeg
minimálisan hány bélyeg formájában ragasztható fel */

lehet[0]:=0 // a 0 összeg kifizethető 0 darab bélyeggel

Ciklus i:=1..k*max(b[])
     Ciklus j:=0..(F-b[i])
          Ha lehet[j]>-1 és lehet[j]<k akkor
               Ha lehet[j]=-1, akkor lehet[j+b[i]]:=lehet[j]+1
               Különben lehet[j+b[i]]:=min(lehet[j]+1, lehet[j+b[i]])
     Ciklus vége
Ciklus vége

min:=1
```

```
Ciklus amíg lehet[min]!=-1
     min:=min+1
Ciklus vége
Ki: 'a legkisebb nem felragasztható érték', min
/* a min értéke lehet nagyobb is, mint a csupa maximális értékű
bélyegből készült összeállítás */
```

3.2.3. KERÍTÉSFESTÉS

1995/1996 NTOKSZTV. 3. kategória, 2. forduló, 2. feladat [2.6][5.5]

Egy h hosszúságú kerítést kell lefestenünk. A festéshez $d_1, d_2, ..., d_n$ kapacitású festékesdobozok állnak rendelkezésre. A festékesdoboz kapacitása a belőle lefesthető kerítéshossz értéke. Lefesthető-e a kerítés úgy, hogy minden kibontott festékesdobozt teljes egészében felhasználunk?

Az előző feladatokban az egyes elemek felhasználásának számát önmagában nem, csak összességében korlátoztuk. Így ez a módszer változtatás nélkül nem követhető. Érdemes ötleteket gyűjteni, hogyan biztosíthatjuk, hogy egy dobozt kétszer ne használjunk fel. A diákokban két ötlet szokott felmerülni.

Az egyik a dominó feladatnál is előkerülő megoldás, amelyben az aktuális állapotot leíró tömb mellett egy segédtömböt használunk, megakadályozva, hogy egy dobozt többször is használjunk.

```
d[] // a doboz kapacitását tároló n elemű tömb

lehet[]:=false
/* a tömb i-edik indexű elemének értéke megmutatja, hogy az i
hosszúságú kerítés lefesthető-e maradéktalanul felhaszált
festékesdobozokkal */

seged[]:=false
/* az aktuális elemmel az eddigi összegekből kiindulval elérhető
összegek */

lehet[0]:=true // a 0 összeg kifizethető

Ciklus i:=1..n
     Ciklus j:=0..(h-d[i])
          Ha lehet[j] akkor seged[j+d[i]]:=true
     Ciklus vége
     Ciklus j:=1..h
          lehet[j]:=lehet[j] OR seged[j]
     Ciklus vége
     seged[]:=false
Ciklus vége
```

```
Ha lehet[h], akkor Ki: 'lefesthető' különben 'nem festhető le'
/* mellékeredményként minden h-nál kisebb hosszra is megkapjuk a
lefesthetőséget */
```

A másik ötlet megvalósítása során nem logikai értékkel jelezzük, hogy lefesthető-e az adott hossz, hanem az eléréshez használt doboz sorszámával. Ha egy pozíciót más módon már elértünk, azt nem írjuk át, hogy később onnan folytathassuk.

```
d[] // a doboz kapacitását tároló n elemű tömb

mivel[]:=-1
/* a tömb i-edik indexű elemének értéke megmutatja, hogy az i
hosszúságú kerítés festésénél melyik volt az utoljára használt doboz
*/

mivel[0]:=0 // a 0 összeg kifizethető

Ciklus i:=1..n
     Ciklus j:=0..(h-d[i])
          Ha (mivel[j]>-1) és (mivel[j]!=i) és (mivel[j+d[i]]=-1)
          Akkor mivel[j+d[i]]:=i
     Ciklus vége
Ciklus vége
```

3.2.4. IGAZSÁGOS OSZTOZKODÁS – MÁSKÉPP

Most olvassuk el ismét az Igazságos osztozkodás című feladatot a 23. oldalon! Ott az Oszthatóság nevű feladatban alkalmazott módszerrel próbáltunk megoldást találni. Nézzük, hogy lesz-e más ötlet, ha a Kerítésfestés feladatot használjuk alapként!

Az ajándékok egyediek, mint ahogyan a festékes dobozok is. Az ajándékok értéke párba állítható a festékes doboz méretével. A festésnél a kerítés hossza az érdekes, annak elérése a cél, az osztozkodásnál az ajándékok összesített értékének felét kell elérni. Tehát a Kerítésfestés feladatnál írt algoritmus az előző mondatokban jelzett megfeleltetéssel az ajándékozás esetén is alkalmazható.

4. REKURZIÓRA KITALÁLT FELADATOK

4.1. PALINDROMMÁ ALAKÍTÁS

2005/2006. OKTV 1. forduló, 4. feladat [2.7]

Adott egy szó (értsd karaktersorozat), amelyet a lehető legkevesebb karakter elhagyásával palindrommá kell alakítani. Adjuk meg az elhagyni szükséges minimális karakterszámot!

A feladattal a rekurzióban nem jártas diákok érdemben nem tudnak mit kezdeni. Általában olyan javaslatok születnek, amelyek az egyes előforduló karakterek számával kapcsolatosak: A magában álló felesleges, a duplán szereplő része a palindromnak. Ezt könnyű megcáfolni például a KEREK, KEREKEBB karaktersorozatokkal.

Ha nincs továbbgondolásra alkalmas ötlet, akkor irányítsuk figyelmüket a karaktersor két széle felé. Ennek alapján kétféle karaktersorozat különböztethető meg: X[valami]Y, valamint X[valami]X karaktersorozat. Ez utóbbi tárgyalható egyszerűbben, az eredeti karaktersorozatból pontosan annyit kell elhagyni, mint a [valami] karaktersorozatból. Az első esetben nem tudjuk, hogy az X vagy az Y az, amire biztosan nincs szükségünk, hiszen nem lehet mindkettőt meghagyni elhelyezkedésük miatt. Tehát azt kell meghagyni, amelynél kedvezőbb érték születik a minimális elhagyásszámra.

Az alábbi „szabályok” írhatók fel:

```
// X, Y tetszőleges, nem egyező karakterek
// [valami] tetszőleges karaktersorozat
Elhagy([])=0
Elhagy(X)=0
Elhagy(X[valami]X)= Elhagy([valami])
Elhagy(X[valami]Y)= min(Elhagy(X[valami]), Elhagy([valami]Y))+1
```

A konkrét algoritmus

```
Függvény Elhagy(s)
     Ha hossz(s)<2,
     akkor Elhagy:=0
     különben
          Ha első(s)=utolsó(s)
          akkor Elhagy:=Elhagy(elsőnélkül(utolsónélkül(s)))
          különben Elhagy:=min(
                              Elhagy(elsőnélkül(s)),
```

```
                              Elhagy(utolsónélkül(s))
                              )+1
          Elágazás vége
     Elágazás vége
Függvény vége
```

Ha továbblépésként a fenti algoritmust tárolt rekurzióra írjuk át, akkor a függvénnyel párhuzamosan kell egy tömböt is kezelni. A rekurzív függvény paramétereként az s stringet használni meglehetősen gazdaságtalan, tömbindexként pedig néhány programozási nyelven nem is lehet. A rekurzió végrehajtása során minden lépésben az eredeti stringnek egy részletét használjuk, amely megadható a részlet első és utolsó karakterének sorszámával, a kiindulási stringet pedig globális változóként használjuk.

```
Függvény Elhagy(i,j)
     Ha j-i<2,
     akkor Elhagy:=0
     különben
          Ha s[i]=s[j]
          akkor Elhagy:=Elhagy(i+1, j-1)
          különben Elhagy:=min(
                              Elhagy(i+1, j),
                              Elhagy(i, j-1)
                              )+1
          Elágazás vége
     Elágazás vége
Függvény vége
```

Most pedig alakítsuk át tárolt rekurzióvá!

```
eh[_, _]:=-1 // a két pozíció közötti karakterelhagyások minimális
számát tároló tömb elemeinek még meg nem határozott értékét jelölje -1
Függvény Elhagy(i,j)
     Ha eh[i,j]=-1, akkor
          Ha j-i<2,
          akkor eh[i,j]:=0
          különben
               Ha s[i]=s[j]
               akkor eh[i,j]:=Elhagy(i+1, j-1)
               különben eh[i,j]:=min(
                                   Elhagy(i+1, j),
                                   Elhagy(i, j-1)
                                   )+1
               Elágazás vége
          Elágazás vége
     Elágazás vége
     Elhagy:=eh[i,j]
Függvény vége
```

Érdemes a feladatot úgy továbbgondolni, hogy miképpen járnánk el, ha meg kellene mondani azt is, hogy melyik az a leghosszabb palindrom, amely a karakterek elhagyásával marad.

4.2. ÖRÖKLÉS

2000. Diákolimpiai válogatóverseny 1. feladat [7.1]

Egy király – halálát közeledni érezvén – végrendelkezett. Országát fiai között szétosztani rendelte a következő módon. A felosztás megyék szerint történjen. Egy fiú nem kaphat kevesebb megyét, mint a nálánál fiatalabb. Minden fiú kapjon legalább egy megyét. Hány különböző módon történhet a szétosztás, ha a megyéket nem különböztetjük meg?

A feladat megoldását az $Eloszt(f, m)$ függvény szolgáltatja, ahol az érték az elosztások számát adja, f fiú és m megye esetén. Fel szokott vetődni az ötlet, hogy a kötelező megyét adjuk oda az örökösöknek és próbáljunk arra a problémára függvényt konstruálni, ahol a fiúk száma f, a megyék száma viszont csak $m-f$, de ebből nem mindenkinek kell kapnia.

Az így megadott függvényre $E(f, m-f) = Eloszt(f, m)$

Milyen módon lehet meghatározni $E(f, m)$ függvény értékét? Érdemes a legkisebb fiú felől kezdeni a vizsgálatot. Mennyi megyét kaphat? Legalább 0, legfeljebb $\left[\frac{m}{f}\right]$ megyét, hiszen ha több jutna neki, akkor nem teljesülne az a feltétel, hogy a fiatalabb legfeljebb annyit kaphat, mint az idősebb. Ha a legkisebb v megyét kapott, akkor a többit hogyan lehetne elosztani? Nyilván már csak $m-v$ megyét tudunk elosztani $f-1$ fiú között. Azonban nem igaz, hogy ezt $E(f-1, m-v)$ függvénnyel határozhatjuk meg, mivel – lévén a legkisebb megkapott v megyét. Ezért nem elegendő kétparaméteres függvényt használni, fel kell tüntetnünk, hogy a legkisebb legalább hány megyét kap. Így a kiszámítandó kifejezés a $e(f-1, m-v, v)$ alakba írandó, ahol v az f. fiúnak juttatott megyeszám. Vegyük észre, hogy ebben az alakban közvetlen kezelhető: $e(f, m, 1)$

A fentiek alapján az alábbi szabályokat tudjuk felírni.

```
e(f, m, i)=e(f, m-f*i, 0)= E(f, m-f*i), ha m/f>=i
e(f, m, i)=0, ha m/f<i
e(f, m, 0)=E(f, m)=
          e(f-1, m, 0)+
          e(f-1, m-1, 1)+
          …
          e(f-1, m-[m/f], [m/f]), ha m>0
e(1, m, _)=1
```

Ebből tárolt rekurzióval megoldást készítve:

```
Függvény e(f, m, i)
     Ha i<=m/f akkor e:=E(f,m-f*i)
     Különben e:=0
Függvény vége
Függvény E(f,m)
     Ha Etomb[f,m]>-1 akkor osszeg:= Etomb[f,m]
     különben
          Ha f=1 akkor osszeg:=1
          Különben
               osszeg:=0
               Ciklus i:=0..[m/f]
                    osszeg:=osszeg+e(f-1, m-i, i)
               Ciklus vége
          Eláagazás vége
     Elágazás vége
     E:=osszeg
Függvény vége
// a főprogram a feladatnak megfelelően
Ki: e(f, m, 1)
```

A diákok ritkán találkoznak a fenti eszközzel, ahol is két függvény kölcsönösen meghívja egymást. Ezt, mint érdekességet érdemes kicsit körüljárni, a technikai megvalósítást segíteni.

4.3. CSERÉPKIÉGETÉS

2000/2001. NTOKSZTV 3. kategória, 3. forduló, 2. feladat [2.8]

Egy műhelyben különböző cseréptárgyakat készítenek. A készítés utolsó fázisa a tárgyak kemencében való kiégetése. Minden tárgynak ismerjük a $t_1, t_2, ..., t_n$ kiégetési idejét, amennyi időt minimálisan a kemencében kell lennie. A kemencében egyszerre legfeljebb k darab tárgyat égethetünk. Az egyszerre égetett tárgyakat addig tartjuk bent, amennyi a leghosszabb égetési idejű tárgy elkészüléséhez szükséges. A tárgyakat a megadott sorrendben kell kiégetnünk. Mennyi idő szükséges az összes tárgy elkészítéséhez?

A megoldást adó algoritmusig könnyen eljutunk, ha többféle konkrét k értéket vizsgálunk. Ha az érintett diákok kevésbé jó képességűek, akkor konkrét égetési idősorozatokat is adjunk meg!

A feladat megoldása $k = 1$ esetén triviális, ekkor ugyanis a szükséges idő egyszerűen az égetési idők összege.

Mi a helyzet $k = 2$ esetén? Lehet, hogy úgy járunk jól, ha az egy tárgyat önmagában helyezzük a kemencébe, lehet, hogy úgy, ha az előzővel vagy a következővel együtt. A feldolgozást az elejéről vagy a végéről kell kezdeni. A feldolgozás iránya nyilván nem befolyásolja a végeredményt. A leírás egyszerűbb, ha a „szabályokat” hátulról előre haladva fogalmazzuk meg.

```
// Egido(x,y) az x. cseréptől az y. cserépig terjedő sorozat
kiégetéséhez szükséges idő
Egido(1,1)=t[1]
Egido(1,2)=max(1], t[2])
Egido(1,n)=min(
                    t[n]+Egido(1,n-1)
                    max(t[n], t[n-1])+Egido(1,n-2)
               )
```

Látható, hogy a függvény első paramétere felesleges, hiszen az első tárgy kiégetésére mindig szükség van. Az általános esetet már ennek megfelelően fogalmazzuk meg.

```
Egido(i)=max(t[1], …, t[i]), ha i<=k
Egido(i)=min  (
                    t[i]+Egido(i-1),
                    …,
                    max(t[i], …, t[i-k+1])+Egido(i-k+1)
               ), ha i>k
```

4.4. STRUCCTOJÁS

2000. Diákolimpiai válogatóverseny, 12. feladat [7.2]

Adott néhány tökéletesen egyforma, igen strapabíró strucctojás. A tojások sérülésmentesen leejthetők igen nagy magasságból is. Adott magasság felett mindegyik tojás összetörik, alatta mindegyik épen marad. Hány dobás szükséges a határt jelentő magasság megállapításához? (A toronyház e emelet magas, a felhasználható tojások száma t. Az épen maradt tojások ismét ledobhatók.)

Nézzük meg, hogy mitől függ az optimális dobásszám meghatározása! Nyilván az emeletek számától és a tojások számától. Alapesetben valóban az emeletek száma lesz a döntő, de a könnyebb végiggondolás érdekében inkább az adott intervallum kezdő és végpontját nevezzük meg a felhasználható tojások száma mellett.

Hogyan válasszunk emeletet? A választott emeletről való tojásledobás eredményezheti a tojás törését, ekkor a választott emelet alatt kell folytatni a próbálkozást eggyel kevesebb

tojással, ellenkező esetben pedig a megadott emelet felett változatlan tojásszám mellett. Felvetődő sejtés lehet, hogy az az ideális, ha a két érték egyenlő. Azonban ez csak egy sejtés, nekünk ki kell számolni az értéket, a fenti megállapítás csak mellékeredmény lehet. Végig kell nézni az összes emeletet a határok között, majd az így adódó próbálkozásszámok közül a legkisebbet kell kiválasztani.

Az alábbi összefüggésben az általános esetet tüntetjük fel:

```
DSz(alsó, felső, tojás)=
Min  // ki kell próbálnunk az összes lehetséges emeletre
     (
     …
     Max
          (
               DSz(alsó, e-1, tojás-1) // ha tört a tojás
               DSz(e+1, felső, tojás)  // a tojás maradt
          )
     …
     )+1
```

Meg kell határozni azokat a speciális eseteket, amelyek a rekurziónak határt szabnak. Ha a tojások száma 1, akkor minden lehetséges emeletet ki kell próbálni. Ha az alsó értéke túllépi a felsőt, akkor ott már nem kell vizsgálni semmit, hiszen 0 kísérletre van szükség.

Ha tárolt rekurzió módszerével adjuk meg a megoldást, akkor a felhasznált tömbnél már nem érdemes a konkrét emeletsorszámokkal dolgozni, helyettük az emeletintervallumok által meghatározott emeletszámot használjuk.

```
DbT[e,t]:=-1 // globális tömb, -1 kezdőértékkel, amely azt adja meg,
hogy e emeletből t tojás segítségével hány dobással lehet eldönteni,
hogy hol a határ
Függvény DSz(also, felso, tojas)
     Ha also>felso akkor DSz:=0
     különben
          Ha tojas=1 akkor DSz:=felso-also+1
          különben
               Ha DbT[felso-also+1, tojas]=-1 akkor
                    Min:=MaxInt;
                    Ciklus e:=also..felso
                         Ertek:=Max(
                                   DSz(also, e-1, tojas-1),
                                   DSz(e+1, felso, tojas)
                                   )
                         Ha Ertek<Min akkor Min:=Ertek
                    Ciklus vége
                    DbT[felso-also+1, tojas]:=Min+1;
               Elágazás vége
               DSz:=DbT[felso-also+1, tojas];
          Elágazás vége
     Elágazás vége
```

```
Függvény vége
```

5. AJÁNLOTT VERSENYFELADATOK

Ebben a fejezetben feltüntetek néhány konkrét versenyfeladatot, amelynek megoldásához csak ötleteket adok, a tényleges algoritmust nem tüntetem fel. Célszerű ezeket a példákat házi feladatként, önálló gyakorló példaként továbbadni a diákok felé. A jobbak remélhetőleg megbirkóznak majd velük, a többiekkel pedig közösen meg lehet tárgyalni.

5.1. SZÍNEZÉS

2005/2006. OKTV döntő, 4. feladat [2.9]

Egy N emeletes fehér épület bizonyos emeleteit a szépség kedvéért pirosra szeretnénk festeni. Csak olyan festést tartunk elfogadhatónak, amelynél szomszédos szinteket nem festünk pirosra. A színezéseket N+1 elemű 0-1 számsorozattal kódoljuk: 1-es jelöli a piros, 0-s pedig a fehér színű emeletet. Az első szám jelenti a földszint, az utolsó pedig az N. emelet színét.

Készíts programot (`SZIN.PAS`, `SZIN.C`, ...), amely megadja, hogy az épület hányféleképpen színezhető ki, valamint a lexikografikus (ábécé szerinti) K-adik színezést!

A `SZIN.BE` szöveges állomány egyetlen sorában az emeletek N száma ($0 \leq N \leq 40$) és K szám ($1 \leq K \leq 100000000$) van.

A `SZIN.KI` szöveges állomány első sorába a színezések lehetséges számát kell írni! A második sorba a K. színezést kell kiírni: az emeletek növekvő sorrendjében N+1 darab egész számot egy-egy szóközzel elválasztva, ahol 0 jelöli a fehér, 1 pedig a pirosra festett szintet!

Célszerű tisztázni egy értelmezésbeli problémát! A teljesen fehérre festett épület megfelel-e a feladat feltételeinek? Az alábbiakban úgy tekintem, hogy igen.

A feladat bonyolultnak látszik. Először is érdemes megnézni, hogy hányféle, szabályoknak megfelelő színezés létezik egyáltalán. Milyenek lehet a színezés? Lehet, hogy a földszintet pirosra festjük, lehet, hogy fehérre. Milyen lehet a többi? Ha a földszint piros, akkor utána fehér szín kell következzen, felette a színezés bármely, szabályoknak megfelelő lehet, ha fehér, akkor tetszőlegesen folytatható.

Rögzítsük ezt le:

```
// Színezés(N+1) legyen az a szám, amennyiféleképpen a földszint és N
emelet kiszínezhető
Színezés(N+1)=
Színezés(N-1)+ // Piros-Fehér kezdet, utána a maradék N-1 emelet
tetszőlegesen
Színezés(N) // Fehér kezdet, felette tetszőlegesen
```

Aki alaposabban megnézi a fenti összefüggést, bizonyára felismeri, hogy a színezések számának lehetséges értékei a Fibonacci számok.

A feladat a konkrét színezésre (K. színezés) is rákérdez. Ehhez meg kell határozni, hogy milyen sorrendben tekintjük az egyes színezéseket. (A lexikografikus rendezés akkor is helyes megfogalmazás, ha az F-P karaktereket használjuk és nem a 0-1 értékpárost.)

Tekintsünk egy N, K számpárt! A lehetséges színezésszám $Fib(N+1)$. A fentebbi meggondolások alapján ezek közül $Fib(N)$ darab kezdődik fehér színnel. Tehát ha $K \le Fib(N)$, akkor a földszint fehér, majd a K változatlan értéke mellett eggyel kevesebb emeletre kell folytatnunk. Amennyiben $K > Fib(N)$, akkor a földszint piros – következésképpen az 1. emelet fehér – és a 2. emelettől kell folytatni, azon belül a $K - Fib(N)$. sorszámú színezést keresve.

5.2. JEGY

2006/2007. NTOKSZTV döntő, 2. korcsoport, 3. feladat [2.10]

Egy jegyiroda nagyszabású koncertre árul jegyeket. Összesen *M* ülőhelyre lehet jegyeket igényelni, pontosabban minden igénylő két egymás melletti jegyet igényelhet, és meg kell adnia, hogy ezért mennyit fizetne. A jegyiroda a beérkezett igények közül ki akarja választani a legtöbb bevételt eredményező igényeket, amelyeket természetesen ki is tud elégíteni.

Készíts programot (`JEGY.PAS`, `JEGY.C`, …), amely kiszámítja az elérhető legnagyobb jegybevételt és meg is adja, hogy melyik igények kielégítése esetén érhető el a legnagyobb bevétel!

A `JEGY.BE` szöveges állomány első sora két egész számot tartalmaz, az ülőhelyek számát ($1 \le M \le 6000$), és az igények számát ($1 \le N \le 30000$). A további *N* sor mindegyike egy igényt leíró két egész számot tartalmaz. Az első szám az igényelt két ülőhely első székének *s*

sorszáma ($1 \leq s < M$), a második szám az az f pénzösszeg ($1 \leq f < 500$), amit az igénylő a két jegyért fizetne. Az igényeket a sorszámukkal azonosítjuk, az állomány $i+1$-edik sorában van az i-edik igény.

A `JEGY.KI` szöveges állomány első sorába az elérhető legnagyobb bevételt kell írni! A második sorba azoknak az igényeknek a sorszáma kerüljön (tetszőleges sorrendben), amelyek esetén a bevétel a legnagyobb lesz! Több megoldás esetén bármelyik megadható.

A feladat nem túl nehéz, akkor is komoly esély van a megoldásra, ha a diákok nem hallottak dinamikus programozásról.

Érdemes bizonyos egyszerűsítéseket tenni: az azonos helyre szóló igények közül csak a legmagasabbat tartjuk meg, valamint az így megmaradó igényeket az ülőhely sorszáma szerint növekvően rendezzük. (A rendezés feltétlen szükséges, a másik lépés elmaradása nem akadályozza a megoldást.)

A rendezés alatt sokan valamely rendezési algoritmus alkalmazását értik. Ennél a feladatnál azonban a rendezést és az azonos pozíción állók közül a legnagyobb értékű ajánlat meghagyását egyszerre megoldhatjuk. Vegyük észre, hogy ez a „rendezés” már a beolvasás során megvalósítható, ilyeténképpen lineáris időigényű.

```
HelyT[].ajanlat:=0
/*a HelyT tömb az ülőhelyekkel indexelt, elemei maximális ajánlatot és
az igény számát tartalmazó rekordból állnak */
// beolvasás
Be: helyszam, igenyszam
Ciklus i:=1..igenyszam
     be: hely, ajanlat
     Ha HelyT[hely].ajanlat<ajanlat, akkor
          HelyT[hely].ajanlat:=ajanlat
          HelyT[hely].igeny:=i
     Elágazás vége
Ciklus vége
```

A megoldásban alkalmazott dinamikus programozási rész ezt követően kerül sorra. A táblázatkitöltő módszerrel azt határozzuk meg, hogy az adott helyig mennyi maximális bevétel produkálható.

```
HelyT[].bevetel:=0
/* ebben a mezőben tároljuk az aktuális helyig terjedő ajánlatokból
származó maximális bevételt a tömböt a 0. elemtől indexeljük */
HelyT[1].bevetel:=HelyT[1].ajanlat
```

```
Ciklus i:=2..helyszam
     Ha HelyT[i-2].bevetel+HelyT[i].ajanlat>HelyT[i-1].bevetel,
     Akkor
          HelyT[i].kell:=True
          HelyT[i].bevetel:= HelyT[i-2].bevetel+HelyT[i].ajanlat
     Különben
          HelyT[i].kell:=False
          HelyT[i].bevetel:= Hely[i-1]
     Elágazás vége
Ciklus vége
```

Már csak meg kell adnunk a kért adatokat, tehát a maximális bevételt és a kielégített igényléseket:

```
Ki: HelyT[helyszam]
I:=helyszam
Ciklus amíg i>0
     Ha Hely[i].kell akkor ki: Hely[i].ajanlat
     i:=i-1
Ciklus vége
```

5.3. MUNKA

2008/2009. OKTV 2. forduló, 5. feladat [2.11]

Egy vállalkozó két azonos munkagépet üzemeltet, amelyeken speciális alkatrészeket gyárt. Sok megrendelést kapott alkatrészek gyártására. A megrendelésben különböző alkatrészek szerepelnek, de ismert, hogy az egyes alkatrészek legyártása mennyi időt igényel (percben kifejezve). A gépek folyamatosan dolgoznak. A vállalkozó el akarja osztani az alkatrészeket a két gép között, hogy a lehető legkorábban befejeződjön a legyártásuk, tehát ha az első gép a neki kiosztott alkatrészeket ***T1***, a második gép ***T2*** idő alatt gyártja le, akkor ***max(T1,T2)*** a lehető legkisebb legyen.

Készíts programot (`munka.pas`, …), amely kiszámítja, hogy legkevesebb mennyi idő alatt tudja a két gép legyártani az összes alkatrészt, és meg is ad egy megfelelő beosztást!

A `munka.be` szöveges állomány első sorában az alkatrészek ***(2≤N≤2000)*** száma van. A második sor pontosan ***N*** egész számot tartalmaz egy-egy szóközzel elválasztva, a legyártandó alkatrészek gyártási idejét, ami ***1*** és ***50*** közötti érték.

A `munka.ki` szöveges állomány első sora egyetlen egész számot tartalmazzon, azt a legkisebb ***T*** időt, amely alatt a két gép le tudja gyártani az összes alkatrészt! A második sor azon alkatrészek sorszámát tartalmazza (tetszőleges sorrendben, egy-egy szóközzel

elválasztva), amelyeket az első, a harmadik sor pedig azokat, amelyeket a második gép gyárt le. Több megoldás esetén bármelyik megadható.

Kis gondolkodás után azt mondhatjuk, hogy a $Max(T_1, T_2)$ értéke akkor a legkisebb, T_1 és T_2 eltérése minimális. Nyilván ideális esetben $T_1 = T_2$. Ennek alapján eszünkbe juthat az Igazságos osztozkodás – másképp feladat a 31. oldalon. Nézzük meg az abban foglaltakat!

Mivel itt nem biztos, hogy egyenlőséget kapunk, nem elegendő a $\frac{\sum_{i=1}^{N} t_i}{2}$ indexű tagig vizsgálódni, de a feldolgozás során megállhatunk az első annál nagyobb indexnél. Az összes munkadarab megvizsgálását követően az érintett indexek közül a fent említett indexnél nagyobbak közül a legkisebb adja a feladat első kérdésére a választ.

A második kérdésre Kerítésfestés feladatnál írt algoritmus egyszerű módosításával lehet válaszolni, a harmadik kérdésre adott válasz ebből már triviális módon következik.

6. MELLÉKLET

A mellékletben néhány, az előzőleg ismertetett feladat alapján megoldható feladatot idézek fel a versenyfeladatok közül.

6.1. PAKOLÁS

2007/2008. OKTV 2. forduló, 4. feladat [2.12]

Kamionnal kell elszállítani tárgyakat. Ismerjük a kamion kapacitását, tehát azt a súlyt, amelynél több nem rakható a kamionra, és ismerjük az elszállítandó tárgyak súlyát. Az a cél, hogy a kamiont úgy pakoljuk meg tárgyakkal, hogy az összsúly a lehető legnagyobb legyen.

Készíts programot (*PAKOL.PAS*, *PAKOL.C*, ...), amely kiszámítja, hogy mekkora az a legnagyobb összsúly, amit a kamionnal elszállíthatunk! A program adja meg, hogy mely tárgyak kamionra rakásával érhető ez el.

A *PAKOL.BE* szöveges állomány első sorában két egész szám van, a tárgyak ***N*** száma (***1≤N≤100***) és a kamion ***K*** kapacitása (***1≤K≤600***). A második sor pontosan ***N*** pozitív egész számot tartalmaz egy-egy szóközzel elválasztva. Az ***I***-edik szám az ***I***-edik tárgy súlya, ami nem nagyobb, mint a kamion ***K*** kapacitása.

A *PAKOL.KI* szöveges állomány első sorába a kamionnal elszállítható legnagyobb ***S*** összsúlyt kell írni! A második sorba az ***S*** összsúlyt adó pakolásban szereplő tárgyak ***M*** számát kell írni! A harmadik sorba a kamionra pakolt ***M*** tárgy sorszámát kell írni tetszőleges sorrendben, egy-egy szóközzel elválasztva! Több megoldás esetén bármelyik megadható.

6.2. CÍMLET

2007/2008. NTOKSZTV 2. kategória, 3. forduló, 1. feladat [2.13]

Egy országban *N*-féle pénzjegy címlet van (mindből tetszőleges darabszámú).

Készíts programot (`CIMLET.PAS`, `CIMLET.C`, …), amely megadja, hogy mely 1 és *M* közötti pénzösszegeket nem lehet a használható címletekkel pontosan kifizetni!

A `CIMLET.BE` szöveges állomány első sorában a pénzjegyek száma ($1 \le N \le 100$) és a

maximális összeg értéke ($1 \leq M \leq 1000000$) van egy szóközzel elválasztva. A második sorban N szám van, az N címlet értéke, egymástól egy-egy szóközzel elválasztva, érték szerint növekvő sorrendben.

A `CIMLET.KI` szöveges állomány első sorába a nem kifizethető összegek számát kell írni! A második sorban egy-egy szóközzel elválasztva a nem kifizethető összegek szerepeljenek, növekvő sorrendben!

6.3. GYÖNGYÖK

2006/2007. OKTV 2. forduló, 5. feladat [2.14]

Tekintsük azt az egyszemélyes játékot, amelyet egy ***N*** sorból és ***M*** oszlopból álló négyzetrácsos táblán lehet játszani! A tábla véletlenszerűen kiválasztott mezőin gyöngyöket helyeznek el. A táblán lehetnek csapda mezők, amelyekre nem lehet lépni. A játék célja az, hogy a játékos egy bábut mozgatva a tábla mezőin a lehető legtöbb gyöngyöt gyűjtse be. A játékszabály a következő:

- Kezdetben a bábu a tábla (1,1) koordinátájú bal felső sarkában áll.
- Egy lépésben a bábut csak szomszédos mezőre lehet mozgatni, vagy jobbra, vagy lefelé.
- Csapda mezőre nem lehet lépni.
- A játék akkor ér véget, ha a bábu a tábla (***N,M***) koordinátájú jobb alsó mezőjére, a célmezőre kerül.

A játékban szerzett pontszám azokon a mezőkön található gyöngyök számának összege, amelyekre a bábuval lépett a versenyző. Az (1, 1) nem csapdamező és az ott lévő gyöngyök is a játékosé lesznek.

Készíts programot (*`JATEK.PAS`, `JATEK.C`*, ...), amely kiszámít egy olyan játékmenetet, amely a legtöbb pontot eredményezi!

A `JATEK.BE` szöveges állomány első sora a tábla sorainak ***N***, és oszlopainak ***M*** számát tartalmazza ***(1≤N, M≤150)***, egy szóközzel elválasztva. Az állomány következő ***N*** sora a kezdeti játékállást tartalmazza. Minden sorban pontosan ***M*** pozitív egész szám van (egy-egy szóközzel elválasztva). Ha ***j***-edik szám ***-1***, akkor ott csapda mező van, egyébként azt adja meg, hogy az adott sorban a ***j***-edik mezőn hány gyöngy van. Minden szám értéke nem

nagyobb, mint ***500.***

A JATEK.KI szöveges állomány első sorába a szabályos játékkal elérhető legnagyobb pontszám értékét kell írni! Ha a célmező nem érhető el, akkor az első és egyetlen sorba a ***-1*** értéket kell írni! Ha el lehet jutni a célmezőre, akkor a második sor pontosan $N + M - 2$ karaktert tartalmazzon, egy olyan szabályos lépéssorozatot, amellyel elérhető a maximális pontszám! A jobbra lépés jele a **'J',** a lefelé lépés jele az **'L'** karakter. A karakterek között nem lehet szóköz, és az utolsó karakter után nem lehet szóköz! Több megoldás esetén bármelyik megadható.

6.4. MALACPERSELY

1999/2000. NTOKSZTV 3. kategória, 2. forduló, 3. feladat [2.15]

Mohó Marci malacperselyben gyűjti pénzét. Csak fémpénzeket rakott a perselybe, de nem jegyezte fel, hogy milyeneket. Felírta azonban az üres persely súlyát, így meg tudja állapítani a perselyben lévő pénzek összsúlyát. Ismeri továbbá az egyes pénzérmék egyedi súlyát és értékét. Szeretné kiszámítani, hogy mennyi az a legkisebb érték, amelyet a perselye biztosan tartalmaz. Egy adott típusú pénzérméből (címletből) több is lehet a perselyben.

Készíts programot (MALAC.PAS vagy MALAC.C), amely kiszámítja, hogy legkevesebb mekkora értéket tartalmaz a malacpersely!

A MALAC.BE bemeneti állomány első sorában van a perselyben lévő pénzek S ($1 \leq S \leq 10000$) összsúlya. A második sorban a pénzérme fajták (címletek) N ($1 \leq N \leq 100$) száma található. A további N sor mindegyike két pozitív egész számot tartalmaz egy szóközzel elválasztva. Az első szám egy pénzérme értéke (nem nagyobb, mint 200), a második szám pedig a pénzérme súlya (nem nagyobb, mint 1000).

A MALAC.KI állomány egyetlen sort tartalmazzon, azt a legkisebb értéket, amelyet a malacpersely biztosan tartalmaz!

6.5. KOCKÁK

1998/1999. NTOKSZTV 2. kategória, 2. forduló, 4. feladat [2.16]

Építőkockákból úgy lehet stabil tornyot építeni, hogy kisebb kockára nem lehet

nagyobbat, illetve könnyebb kockára nem lehet nehezebbet tenni.

Készíts programot (`KOCKA.PAS` vagy `KOCKA.C`), amely *N* kocka alapján megadja a belőlük építhető legmagasabb tornyot!

A `KOCKA.BE` állomány első sorában a kockák ($1 \leq N \leq 1000$) száma van, a további *N* sorban pedig az egyes kockák oldalhossza és súlya (mindkettő 20000-nél kisebb pozitív egész szám), egyetlen szóközzel elválasztva.

A `KOCKA.KI` állomány és a képernyő első sorába a legmagasabb torony *M* kockaszámát kell írni, a következő *M* sorba pedig az építés szerint alulról felfelé sorrendben a felhasznált kockák oldalhosszát és súlyát.

6.6. LICIT FELADATOK

2000/2001. NTOKSZTV 2. kategória, 2. forduló, 2. feladat [2.17]

Nevesincs város polgármestere elhatározta, hogy értékesíti a város mellett levő téglalap alakú földdarabot. A földet egyforma méretű parcellákra osztotta.

A polgármester úgy döntött, hogy a parcellákat nyilvános pályázat keretében adja el, azaz egy adott határidőig minden érdeklődő lezárt borítékban leadhatja ajánlatát. Egy pályázó csak egy ajánlatot nyújthat be, amelyben meg kell adnia, hogy melyik parcellától melyik parcelláig terjedő részt kívánja megvenni, és mennyiért.

A pályázat sikeres volt, a határidő lejártáig *N* pályázat érkezett. Ezek közül ki kell választani azokat az ajánlatokat, amelyek a legtöbb bevételt eredményezik, s persze úgy, hogy egyetlen parcellát sem ítélnek oda egynél több pályázónak. Egy-egy pályázó vagy az összes kért parcellát megkapja, vagy egyet sem kap meg. Előfordulhat, hogy a maximális bevétel eléréséhez nem kell eladni az összes parcellát.

Írj programot (`LICIT.PAS` vagy `LICIT.C`), amely megadja a választ a polgármester problémájára!

Bemenet:

A `LICIT.BE` állomány első sorában a pályázatok N száma ($1 \leq N \leq 100$) és a parcellák *M* száma ($1 \leq M \leq 100$) található, egy szóközzel elválasztva. A következő *N* sor az egyes pályázók adatait tartalmazza, a pályázókat ez a sorrend azonosítja. Mindegyik sorban 3 szám van egy-

egy szóközzel elválasztva: *A B FT*, ami azt jelenti, hogy a pályázó az *A* sorszámú parcellától ($1 \le A \le M$) a *B* sorszámú parcelláig ($A \le B \le M$) terjedő részért *FT* forintot fizetne ($1000 \le FT \le 1000000$).

Kimenet:

A `LICIT.KI` állomány első sorába az elérhető legnagyobb bevételt kell írni. A második sorba a nyertes pályázók sorszámai kerüljenek egy-egy szóközzel elválasztva. Ha több megoldás is van, csak egyet kell kiírni (bármelyiket).

2000/2001. NTOKSZTV 3. kategória, 2. forduló, 2. feladat [2.18]

Nevesincs sziget polgármestere elhatározta, hogy értékesíti a sziget tengerpartját. A partot egyforma méretű parcellákra osztotta.

A polgármester úgy döntött, hogy a parcellákat nyilvános pályázat keretében adja el, azaz egy adott határidőig minden érdeklődő lezárt borítékban leadhatja ajánlatát. Egy pályázó csak egy ajánlatot nyújthat be, amelyben meg kell adnia, hogy melyik parcellától melyik parcelláig terjedő részt kívánja megvenni, és mennyiért.

A pályázat sikeres volt, a határidő lejártáig *N* pályázat érkezett. Ezek közül ki kell választani azokat az ajánlatokat, amelyek a legtöbb bevételt eredményezik, s persze úgy, hogy egyetlen parcellát sem ítélünk oda egynél több pályázónak. Egy-egy pályázó vagy az összes kért parcellát megkapja, vagy egyet sem kap meg. Előfordulhat, hogy a maximális bevétel eléréséhez nem kell eladni az összes parcellát.

Írj programot (`LICIT.PAS` vagy `LICIT.C`), amely megadja a választ a polgármester problémájára!

Bemenet:

A `LICIT.BE` állomány első sorában a pályázatok *N* száma($1 \le N \le 100$) és a parcellák *M* száma ($1 \le M \le 100$) található, egy szóközzel elválasztva. A következő *N* sor az egyes pályázók adatait tartalmazza, a pályázókat ez a sorrend azonosítja. Mindegyik 3 számot tartalmaz egy-egy szóközzel elválasztva: *A B FT*, ami azt jelenti, hogy a pályázó az *A* sorszámú parcellától ($1 \le A \le M$) a *B* sorszámú parcelláig ($1 \le B \le M$) terjedő részért *FT* forintot fizetne ($1000 \le FT \le 1000000$). Ha $B < A$, akkor a pályázó *B*-től *M*-ig és 1-től *A*-ig szeretné megvásárolni a parcellákat.

6.7. JÁRDAKÖVEZÉS

2009/2010. OKTV 2. forduló, 5. feladat [2.19]

Írj programot (`JARDA.PAS`, …), amely kiszámítja, hogy hány féleképpen lehet kikövezni egy 2*N egység méretű járdát 1*1 és 1*2 méretű lapokkal!

A `JARDA.BE` szöveges állomány első sorában a járda méretét megadó N egész szám van ($1 \leq N \leq 36$)

A `JARDA.KI` szöveges állomány első és egyetlen sorába azt a számot kell írni, ahány féleképpen kikövezhető a 2*N méretű járda!

7. Összefoglalás

Célom az volt, hogy kollégák és diákok számára érthető művet alkossak. Jelenleg még csak reménykedem, hogy sikerült. A dokumentum szerkezete megfelel annak, amit korábban elképzeltem. Az érdeklődő olvasónak javaslom, hogy haladjon a leírtakkal egyező sorrendben, igyekezzen önállóan kitalálni a közölt problémák megoldását, és ne spórolja meg az algoritmusok programmá formálását.

Nem hittem volna, hogy a fejemben kavargó feladatok többségének meglelem az eredetijét, de majdnem mindet sikerült beazonosítanom. Úgy gondolom, a dolgozat törzsét képező feladatokhoz olyan megjegyzéseket tudtam fűzni, amelyek kellően segítik az olvasót azok megoldásában. Bízom abban, hogy a lejegyzett algoritmusokban nem követtem el hibát. A Melléklet fejezet tartalma csak válogatásnak tekinthető a bemutatott feladatokhoz hasonló példákból, ezt természetesen lehetne tovább bővíteni.

Köszönettel tartozom dr. Papp Zoltán adjunktus úrnak a türelméért, Danner Gábornak, volt tanítványomnak pedig azért, hogy figyelmembe ajánlott néhány, valaha közösen megbeszélt és megtárgyalt feladatot.

8. IRODALOMJEGYZÉK

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest: Algoritmusok
Műszaki Kiadó, 2003
IV/16. Dinamikus programozás fejezet

[2] Nemes Tihamér OKSZTV feladatarchívuma
http://nemes.inf.elte.hu/nemes_archivum.html

[2.1] http://tehetseg.inf.elte.hu/nemes/2010/nt10-1f1.doc

[2.2] http://tehetseg.inf.elte.hu/nemes/2010/nt10-1f2.doc

[2.3] http://tehetseg.inf.elte.hu/nemes/2010/nt10-2f1.doc

[2.4] http://tehetseg.inf.elte.hu/nemes/2010/nt10-2f2.doc

[2.5] http://tehetseg.inf.elte.hu/nemes/2000/Nt00-3f3.doc

[2.6] http://tehetseg.inf.elte.hu/nemes/1996/Nt96-2f3.doc

[2.7] http://tehetseg.inf.elte.hu/nemes/2006/nt06-1f3.doc

[2.8] http://tehetseg.inf.elte.hu/nemes/2001/Nt01-3f3.doc

[2.9] http://tehetseg.inf.elte.hu/nemes/2006/nt06-3f3.doc

[2.10] http://tehetseg.inf.elte.hu/nemes/2007/nt07-3f2.doc

[2.11] http://tehetseg.inf.elte.hu/nemes/2009/nt09-2f3.doc

[2.12] http://tehetseg.inf.elte.hu/nemes/2008/nt08-2f3.doc

[2.13] http://tehetseg.inf.elte.hu/nemes/2008/nt08-3f2.doc

[2.14] http://tehetseg.inf.elte.hu/nemes/2007/nt07-2f3.doc

[2.15] http://tehetseg.inf.elte.hu/nemes/2000/Nt00-2f3.doc

[2.16] http://tehetseg.inf.elte.hu/nemes/1999/Nt99-2f2.doc

[2.17] http://tehetseg.inf.elte.hu/nemes/2001/Nt01-2f2.doc

[2.18] http://tehetseg.inf.elte.hu/nemes/2001/Nt01-2f3.doc

[2.19] http://tehetseg.inf.elte.hu/nemes/2010/nt10-2f3.doc

[3] http://hu.wikipedia.org/wiki/Fibonacci-sz%C3%A1mok

[4] Hanák D. Péter, Zsakó László (szerkesztők): Programozási versenyfeladatok tára ’94
Neumann János Számítógéptudományi Társaság Budapest, 1994

[5] http://nttv.gyakg.u-szeged.hu/nttvfelm.htm
NTOKSZTV, ACM, IOI, CEOI feladatok nem teljes gyűjteménye,
a feladatszövegek mellett gyakran tesztesetek is megtalálhatók.
(Itt olyan korai Nemes Tihamér OKSZTV feladatok is elérhetők, amelyek a hivatalos archívumban nem)

[5.1] http://nttv.gyakg.u-szeged.hu/REGIO/Nttv94r2.HTM

[5.2] http://nttv.gyakg.u-szeged.hu/acm/elte2000.htm#oszthatosag

[5.3] http://nttv.gyakg.u-szeged.hu/donto/Nttv00d3.htm#3

[5.4] http://nttv.gyakg.u-szeged.hu/acm/acmd1995.htm#e

[5.5] http://nttv.gyakg.u-szeged.hu/REGIO/Nttv96r3.htm

[6] Középiskolai Matematikai Lapok

[7] http://tehetseg.inf.elte.hu/valogatok/valogatok_archivum.html
Diákolimpiai válogatóverseny feladatainak archívuma

[7.1] http://tehetseg.inf.elte.hu/valogatok/2000/f1.doc

[7.2] http://tehetseg.inf.elte.hu/valogatok/2000/f4.doc