de később szüleivel átköltöztek Amerikába, ahol Fahrenheit fokokban fejezik ki a hőmérsékleti értékeket (mind a mai napig). A 102°F csak 38,8°C testhőmérsékletet jelent, amibe viszont nem lehet belehalni.

A relativitáselmélet alapjai G. Gamow*: Mr. Tomkins Csodaországban* elbeszélésével hozható „emberközelbe". Képzeletbeli történet egy úrról, aki egy olyan világban jár álmában, ahol a határsebesség mindössze 20 mérföld. Ilyen körülmények között a relativisztikus változások meghökkentő hétköznapi tapasztalatokkal járnak, mármint azok számára, akik idegenek ebben az „országban".

Hárs László, József Attila-díjas költő *Miértek és hogyanok* című verse számos fizikával kapcsolatos kérdést felvet: „Hogyha nyár van, hol a tél?", „Mikor nem fúj, hol a szél?", „Fényes délben hol a Hold?", „Miért folyik a folyó?", „Az eső mért esik le? Mért nem esik soha fel?". A fenti versben a tanítás lényegét is megfogalmazza a költő az egyik szakaszban. Azt, hogy tanítani nem azt jelenti, hogy a tanár megmondja, hogy úgy van, ahogy van, és a tanuló meg kell(ene) azt úgy tanulja, hanem azt, hogy hozzá kell segíteni őt ahhoz, hogy magától fedezze fel a dolgokat. „Mondják: néhány év alatt/nagyra növök biztosan,/s mind az összes titkokat/megfejthetem egymagam."

Számomra a legnagyobb kérdés pedig az, hogy hová tűnt a tanulókból a titkok megfejtésének a vágya, a gyermeki kíváncsiság, miért adják fel egy idő után a kérdésfeltevést, miért akadozik ennyire a gondolkodás folyamata, és sokuknál miért került egyenlőség jel a tanulás és a biflázás közé.

Próbálgatom tehát továbbra is, hogy „megfogjam" őket egy szép verssel vagy írással, de az is bevált, ha a váltakozó áram szinuszai mellé becsempészem az *AC/DC* (Alternativ Curent/Direct Curent) valamelyik ismert dalát. Ha pedig Edison felfedezéseiről mesélünk, akkor jól jön a Fonográf együttes *Edison Magyarországon* című száma, de, ha van kedvünk, meghallgatjuk (akár le is fordítjuk) a *Mary Had a Little Lamb* című angol gyermekdalt, melynek szöveges változata volt az első, Edison által 1877-ben rögzített hanganyag. Mára ezt már megvétózták, mert talán egy Martinville nevű nyomdásznak sikerült 17 évvel korábban készítenie egy mindössze tíz másodperces hangfelvétel, egy fonoautográf nevű készülékkel. Sebaj, nem ez az első legenda, ami szertefoszlik.

Száva Ildikó, tanár

Backtracking és greedy kéz a kézben

Ahhoz, hogy két dolog jól kiegészíthesse egymást, szükséges hogy eléggé hasonlítsanak egymásra, de kellő mértékben különbözzenek is. Szép példa erre, ahogy a férfi és a nő ki tudja egészíteni egymást a házasságban.

Mind a backtracking, mind a mohó stratégiák mélységükben viszonyulnak a feladatok szerkezetét ábrázoló fákhoz. Mindkét módszer gyökér-levél irányba építkezik: a backtracking megoldás *utakat*, a greedy pedig optimális levélhez vezető *utat* adja meg. Optimalizálási problémák esetében az alapvető különbség köztük az, hogy amíg a

backtracking a teljes fa vagy ennek egy jelentős részfája, mélységi bejárása révén potenciális megoldások között válogatva keres, addig a mohó módszer egyetlen gyökér-levél úton szalad le.

Továbbá egy megoldott feladaton keresztül mutatjuk be, miként növelhető a backtracking és mohó stratégiák eredményessége a kombinálásuk révén.

Hátizsák-probléma: Egy üzletben *n* tárgy (áru) található, amelyeknek ismert az áruk és a súlyuk. Az árakat a *t* bejegyzés típusú tömb elemei *a* mezőiben, a súlyokat pedig az elemek *g* mezőiben tároljuk, ahol *t[i].g* (*i* = 1,*n*) természetes számok. Állapítsuk meg, hogy mely tárgyakat fogja magával vinni egy tolvaj ahhoz, hogy a lehető legnagyobb nyereséggel távozzon (a hátizsákja legtöbb *G* súlyt bír meg).

A feladat szövege két változatban is ismert:

a) a tárgyak elvághatók (folytonos változat),

b) a tárgyak nem vághatók el (diszkrét változat).

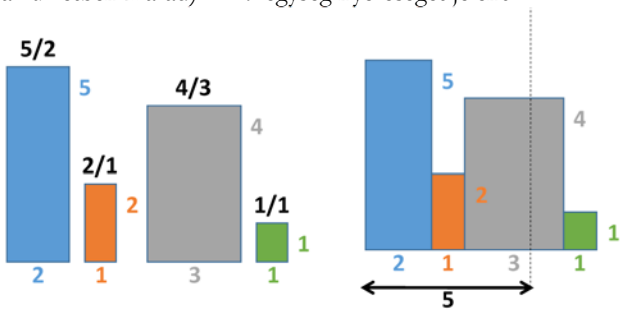
Példa:

Bemenet: n = 4, G = 5, t[1..4].g = {2, 1, 3, 1}, t[1..4].a = {5, 2, 4, 1}

Kimenet:

a) (1, 1, 2/3, 0) – jelentése: az első és második tárgy egészében, a harmadiknak pedig 2/3 része kerül a hátizsákba (a negyedik áru az üzletben marad). Ez 29/3=9.66 egység nyereséget jelent.

b) (1, 0, 1, 0) – jelentése: az első és harmadik tárgy kerül a hátizsákba (a második és negyedik áru az üzletben marad). Ez 9 egység nyereséget jelent.



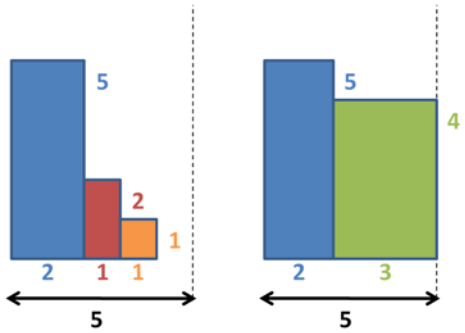
1. ábra

(a) A tárgyak vízszintes irányú mérete a súlyukkal arányos, a függőleges irányú pedig az árukkal. Mindenik tárgy fölé az egységnyi értékét írtuk.
(b) A mohó megoldást szemlélteti a példafeladat folytonos változatára.

Megoldás: A feladat a) változata megoldható mohó stratégiával. A tárgyakat érték (ár/súly) szerint csökkenő sorrendben próbáljuk betenni a hátizsákba (a példabemenet éppen ebben a sorrendben tartalmazza a tárgyakat; ha nem így lenne, akkor a tárgyak megfelelő rendezésével tudjuk biztosítani a mohó sorrendet). Az első áruból, amelyik

már nem fér egészében a hátizsákba, levágunk annyit, hogy azzal teljesen megteljen. Bizonyítható, hogy ez a stratégia mindig az optimális megoldáshoz vezet.

Ha a feladat b) változatát próbáljuk mohó algoritmussal megoldani, a fenti megközelítés nem mindig vezet optimális megoldáshoz. A fenti példa esetében is az (1, 1, 0, 1) kódú megoldást találnánk, holott az optimálisnak a kódja, amint láttuk, az (1, 0, 1, 0).



2. ábra

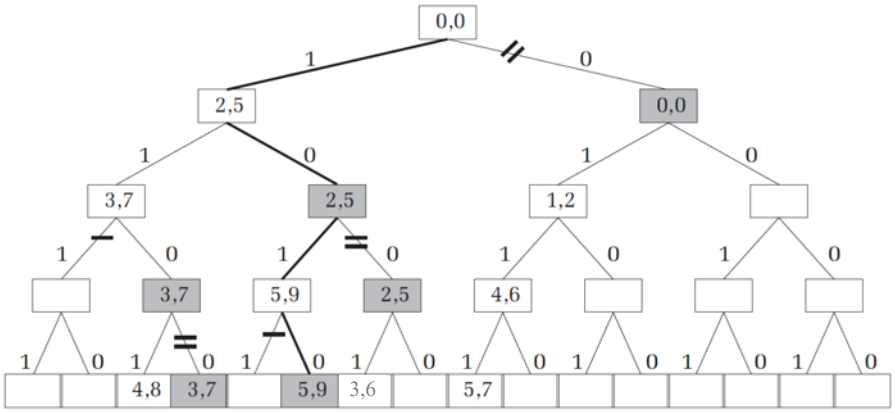
(a) A mohó megoldást szemlélteti a példafeladat diszkrét változatára.
(b) A példafeladat diszkrét változatának optimális megoldását szemlélteti

Hogyan közelítené meg ezt a feladatot a backtracking stratégia? Mivel mind az n tárgy esetén két lehetőség közül választhatunk: vagy beletesszük a tárgyat a hátizsákba, vagy nem, a feladat keresési tere egy n+1 szintes bináris fa lesz. Ezt a fát mutatja be a 3. ábra a példánkra felrajzolva. A fa gyökér-levél útjai a tárgyak halmazának részhalmazait ábrázolják. Nevezzük optimális gyökér-levél útnak azt, amelyik a feladat *optimális* megoldását képviseli, és optimális levélnek azt, amelyikhez ez az út vezet. A nyers erő módszere az lenne, hogy generáljuk a tárgyak halmazának összes részhalmazát, kiválasztva közülük először azokat, amelyek belefférnek a hátizsákba, majd pedig azt, amelyik a legtöbb nyereséggel jár (maximumkeresést végzünk a *potenciális* megoldások között). Mivel az n elemű halmaz részhalmazainak száma 2^n (mindenik részhalmazhoz rendelhető egy n elemű bináris kód), ez a megoldás 2^n bonyolultságú algoritmust jelent. Mindez megvalósítható a teljes fa mélységi bejárásával, ami felfogható egy olyan backtracking algoritmusként, amely csak akkor lép vissza, ha már nincs további bepakolható tárgy. Hogyan lehetne javítani ezen az algoritmuson? Csak olyan részhalmazokat generálunk, amelyek belefférnek a hátizsákba, azaz nem folytatjuk az építkezést olyan irányokba, amelyek túlterhelt hátizsákot eredményeznének. Az „backtracking ollót", amely ebből a szempontból metszi meg a fát, vastagított szimpla vonalkával jelöltük (3. ábra).

Mindezek után is úgy találhatjuk, hogy míg a mohó stratégia nem volt kielégítő, a backtracking túl időigényes. Egy lehetséges (és jobb) megoldáshoz vezet a két módszer kombinálása.

Hogyan lehetne még hatékonyabban optimalizálni a fenti backtracking algoritmust a mohó stratégia segítségével? Foglalkozzon a backtracking is mohó sorrendben (rendezzük a tárgyakat az értékük szerint csökkenő sorrendbe) a tárgyakkal, és először mindig azt a lehetőséget próbáljuk ki, hogy a tárgyat beletesszük (nyilván csak akkor, ha még

belefér) a hátizsákba. Ez azt jelenti, hogy a bináris fában a bal ágakat kódoljuk 1-essel (beletesszük), és a jobb ágakat 0-val (nem tesszük bele) (3. ábra).



3. ábra

A hátizsák feladat diszkrét változatának keresési terét ábrázoló bináris fa.

Ily módon a backtracking algoritmus elsőnek éppen a mohó megoldást találja meg, amely ha nem is garantáltan optimális, de egy elég jó megoldásnak számít. Egy viszonylag jó megoldás korai megtalálása javít az alábbiakban bemutatott optimalizálás hatékonyságán.

Tartsuk nyilván, hogy minden csomópont képviselte állapotban mekkora összsúly van már a hátizsákban (a 3. ábrán ezt az értéket írtuk az egyes csomópontokba, a hozzájuk tartozó nyereséggel; `akt_g` és `akt_ny` az alábbiakban bemutatásra kerülő algoritmusban). Legyen továbbá egy globális változó (`max_ny`), amely a *kurrens optimum* értéket, az addig megtalált legjobb megoldás értékét tárolja. Ha ezt kezdetben nullára is állítjuk, az első megoldás megtalálásakor frissül a mohó megoldás értékére (lásd az előbbi bekezdést). Ahhoz, hogy konstans időben tudjuk ellenőrizni, hogy adott pillanatban a hátralevő tárgyak nem férnek-e mind bele a hátizsákba, célszerű előre feltölteni az a[1..n] és b[1..n] tömböket úgy, hogy a[i] = $\sum_{j=i}^{n}$ t[j].g, b[i] = $\sum_{j=i}^{n}$ t[j].a, (ahol j=i..n). Ha a[1] < G, akkor a feladat triviális, hiszen ez azt jelenti, hogy az összes tárgy belefér a hátizsákba, és a maximális nyereség b[1]. Ellenkező esetben a feladat úgy is átfogalmazható, hogy mely tárgyakat hagyja a tolvaj az üzletben, szem előtt tartva, hogy a maximális nyereséggel szeretne távozni. Mivel valamely tárgy elvetésének esetét a megfelelő *jobb* fiúrészfa képviseli, ezért a következőkben vázolt optimalizálás ezekre fókuszál.

Milyen esetekben kerülhető el a kurrens részfa *jobb* fiúrészfájának bejárása? Tegyük fel, hogy a teljes fa gyökerétől a kurrens részfa gyökeréhez vezető út az 1..k tárgyakra vonatkozó, már meghozott döntéseket képviseli. A kurrens részfa *jobb* fia nyilván azt az esetet ábrázolja, hogy a (k+1)-edik tárgy nem kerül bele a hátizsákba. Amennyiben a hátralevő tárgyak (k+2 … n) mind belefférnek a hátizsákba, akkor úgymond „gondolkozás nélkül" (konstans időben eldönthető: `akt_g + a[k+2]` ≤ `G`) az összest beletesszük (és frissítjük a kurrens optimumot, amennyiben indokolt). Mivel a szóbanforgó

jobb fiúrészfa e legbaloldaliabb levele garantált legjobb (az illető részfára nézve), ezért ennek bejárása jobb levelek reményében, értelmetlenné vált.

Ha a fenti logikával nem kerülhető el az illető jobb fiúrészfa bejárása, akkor esetleges terméketlenségét (nem tartalmazza az optimális levelet) az alábbi módon próbálhatjuk meg kideríteni. Vizsgálat céljából folytassuk a bejárást egy olyan mohó algoritmussal, amely elvághatja a tárgyakat. Ezt valósítja meg az alábbi algoritmus `supremum` függvénye, amely a szóbanforgó jobb fiúrészfa egyetlen gyökér-levél útján „szalad le", tehát lineáris bonyolultságú. Az így kapott nyereség nagyobb (vagy egyenlő) lesz, mint a teljes fa gyökeréből az illető részfa bármelyik leveléhez vezető út nyeresége. A kapcsolódó optimalizálás alapötlete a következő: ha ez a supremum érték sem nagyobb, mint a kurrens optimum, akkor az illető jobb fiúrészfa biztosan nem tartalmazza az optimális levelet, tehát értelmetlen lenne bejárni (így teljesen lemetszhető a fáról).

A 3. ábrán besatíroztuk azokat a csomópontokat, amelyekhez tartozó részfákra a példánk esetében meghívódik a `supremum` függvény. Vastagított dupla vonalkával jelöltük ezen optimalizálás „ollóját". A „dupla ollóval" lemetszett részfákban bejelöltük a vizsgálati mohó utat. Az optimális gyökér-levél utat, amelynek kódja 1010, a vastagított vonal jelzi. Üresen hagytuk azokat a csomópontokat, amelyeknek bejárását sikerült elkerülni.

A `hátizsák` rekurzív backtracking eljárás k-adik szinti meghívása az `akt_g` és `akt_ny` paraméterekben megkapja, hogy az aktuális állapotban mekkora súly van már a hátizsákban, és mennyi nyereséggel. Ezt a két értéket fogja érték szerint átadni a `supremum` függvénynek is, amely – amint már említettük – vizsgálat céljából mohó módon (folytonos változatban) folytatja a (k + 2), . . ., n tárgyak bepakolását (mivel jobb fiúról van szó, a (k + 1)-edik tárgy nem kerül a hátizsákba). A `max_ny` és `opt_x[]` cím szerint átadott paraméterekben tárolódik a maximális nyereség és az optimális megoldás kódja.

```
supremum(t[],n,G,akt_g,akt_ny,k)
minden i=k,n végezd
    ha akt_g + t[i].g ≤ G akkor
      akt_g = akt_g + t[i].g
      akt_ny = akt_ny + t[i].ár
    különben
      akt_ny = akt_ny + (G - akt_g) * t[i].ár / t[i].g
      return akt_ny
    vége ha
vége minden
return akt_ny
vége supremum

hátizsák(x[],t[],n,G,akt_g,akt_ny,k,max_ny,opt_x[])
ha k == n akkor
    ha akt_ny > max_ny akkor // frissítjük a kurrens optimumot
      max_ny = akt_ny
      opt_x[1..n] = x[1..n]
    vége ha
különben
```

```
    // bal fiú
    ha akt_g + t[k+1].g <= G akkor // a (k+1). tárgy még belefér a hátizsákba
      x[k+1] = 1
      háti-
      zsák(x,t,n,G,akt_g+t[k+1].g,akt_ny+t[k+1].ár,k+1,max_ny,
      opt_x)
    vége ha
    // jobb fiú
    ha akt_g + a[k+2] <= G akkor // a maradék (k+2)..n tárgyak mind belefér-
    nek
      ha akt_ny + b[k+2] > max_ny akkor // frissítjük a kurrens optimumot
        max_ny = akt_ny + b[k+2]
        opt_x[1..n] = x[1..k]+[0]+[1..1] // a + jel utána-fűzést jelent
      vége ha
    különben
      sup_ny = supremum(t,n,G,akt_g,akt_ny,k+2)
      ha sup_ny > max_ny akkor // a jobb fiúrészfa esélyes optimális levélre
        x[k+1] = 0
        hátizsák(x,t,n,G,akt_g,akt_ny,k+1,max_ny,opt_x)
      vége ha
      // hiányzik a különben-ág: a jobb fiúrészfa esélytelen optimális levélre
    vége ha
  vége ha
  vége hátizsák
```

Az alábbiakban közöljük azt az algoritmusrészletet, amely tartalmazza a hátizsák eljárás meghívását és az optimális megoldás kiíratását:

```
...
max_ny = 0
hátizsák(x,t,n,G,0,0,0,max_ny,opt_x)
ki: max_ny, opt_x[1..n]
...
```

A dupla olló, úgymond Branch and bound szellemben metszi meg a fát. Egy következő cikkben részletesebben tárgyaljuk majd e programozási technikát. Előzetesként annyit, hogy a Branch and bound ollóval olyan részfákat metszünk le, amelyekről kideríthető, hogy garantáltan nem tartalmaznak jobb megoldás-leveleket, mint az addig ismert legjobb megoldás (ha nem jobbak az addigi legjobbnál, akkor nyilván esélytelenek az optimális-levél státusra). A `supremum` függvény egy felső korlátot térít vissza a kurrens csomópont jobb fiú-részfája tartalmazta esetleges megoldás levelekre vonatkozóan. Ha e felső korlát sem nagyobb az addigi legjobb megoldás értékénél (melyet a `max_ny` változó képvisel), akkor a szóbanforgó részfa lemetszhető a keresési fáról (bejárása átugorható; a (`sup_ny>max_ny`) feltételű `ha`-utasításnak hiányzik a `különben`-ága).

Kátai Zoltán