Alapvető algoritmusok

13. előadás

Dr. Pătcaș Csaba

BABEŞ-BOLYAI TUDOMÁNYEGYETEM
Matematika és Informatika Kar

Tartalom

Dinamikus programozás módszere
Palindrom
Edit distance
Hátizsák diszkrét feladata
Leghosszabb k páratlan számot tartalmazó növekvő részsorozat
Mátrixok optimális szorzása
Zárójelezések
Összefoglalás

Palindrom

Feladat

Adott egy n karakterből álló s karakterlánc. Határozzuk meg a leghosszabb olyan részsorozatát s karaktereinek, melyek az eredeti sorrendben írva tükörszót alkotnak!

Példa: $s = abecbd$

Megoldás: 3 (abecbd)

Palindrom

Megoldás

- Próbáljuk a megoldást s kisebb „darabjaiból” felépíteni.
- Egy ilyen „darabot” jellemezhetünk a kezdő és végső karakter indexével.
- Legyen `maxPal[i][j]` a leghosszabb palindrom részsorozat hossza, amely `s[i..j]`-ben található.
- A megoldás természetesen `maxPal[1][n]` lesz.
- Nyilván egy karakter önmagában palindrom, tehát `maxPal[i][i] = 1`
- Hasznos, ha kezdetben az üres részeket is lekezeljük ilyen módon: `maxPal[i][i - 1] = 0`

Palindrom

Megoldás

- Az előző feladat alapján könnyen beláthatjuk, hogy ha $s[i] = s[j]$, akkor `maxPal[i][j] = 2 + maxPal[i + 1][j - 1]`
- A leghosszabb közös részsorozat feladatában látott logika alapján, ha $s[i] \neq s[j]$, a két karakter közül valamelyik felesleges az optimális részmegoldás kialakításában, tehát
  `maxPal[i][j] = max(maxPal[i + 1][j], maxPal[i][j - 1])`

Palindrom

Megoldás

- Mennyi az algoritmus idő- és memóriabonyolultsága?

Palindrom

Megoldás

- Mennyi az algoritmus idő- és memóriabonyolultsága?
- Egy $n \times n$ méretű mátrix főátló feletti részére van szükségünk, tehát a memóriabonyolultság $\Theta(n^2)$.
- Minden elem kiszámítása konstans időben történik, tehát az időbonyolultság is $\Theta(n^2)$.
- Hogyan javíthatunk a memóriabonyolultságon?

Szerkesztési távolság (Edit distance)

Feladat

Hogyan alakíthatunk át egy s_1 karakterláncot egy s_2 karakterláncca minimális számú művelettel, ahol egy művelet egy karakter hozzáadása, törlése, vagy módosítása?

Példa: $s_1 =$`abcd`, $s_2 =$ `bxd`
Műveletek minimális száma: 2 (`abcd` $\rightarrow$ `bcd` $\rightarrow$ `bxd`)

Szerkesztési távolság (Edit distance)

Megoldás

- Az intuíció azt súgja, hogy a részfeladatok paraméterezésénél figyelembe kell vennünk s_1 és s_2 karaktereit is.
- Innen jön az ötlet, hogy jelöljük `tav[i][j]`-vel a műveletek minimális számát, melyek segítségével átalakíthatjuk s_1 első i karakterét s_2 első j karakterévé.
- Az eredmény `tav[`s_1`.hossz][`s_2`.hossz]` lesz, ahol `s.hossz` az `s` karakterláncban található karakterek száma.
- `tav[0][j] = j`, mivel az üres karakterláncból úgy kaphatjuk meg s_2 első j karakterét, hogy egyenként hozzáadjuk őket.
- Hasonló módon `tav[i][0] = i`, mivel s_1 első i karakteréből úgy kapunk üres karakterláncot, ha egyenként töröljük őket.

Szerkesztési távolság (Edit distance)

Megoldás

- Ideje megállapítani a rekurzív összefüggéseket!
- Ha $s_1[i] = s_2[j]$, akkor könnyű dolgunk van, nem kell semmilyen műveletet elvégeznünk, hanem arra hagyatkozhatunk, hogy az egyenlő karakter nélkül mekkora a szerkesztési távolság, vagyis ekkor
  `tav[i][j] = tav[i - 1][j - 1]`
- Ellenkező esetben valamelyik művelet elvégzésére lesz szükségünk, hogy az (i, j) állapotba jussunk.

Szerkesztési távolság (Edit distance)

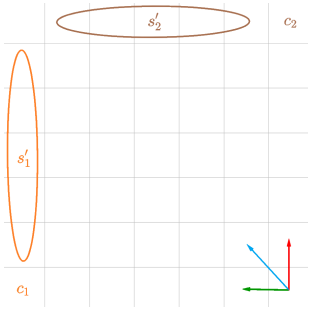
Megoldás

- Ha módosítunk, akkor s_1 i. karakteréből lesz s_2 j. karaktere és az előző esethez hasonlóan, leolvassuk, hogy mennyi a megoldás ezek nélkül a karakterek nélkül.
- Ha törlünk, azt jelenti, hogy eddig átalakítottuk $s_1[1..i-1]$-et $s_2[1..j]$-vé és $s_1[i]$-t eldobhatjuk.
- Hozzáadáskor eddig felépítettük s_1 első i karakteréből s_2 első $j-1$ karakterét, ehhez fűzzük hozzá $s_2[j]$-t.
- Nyilvánvalóan a három művelet közül azt választjuk, amelyik minimális lépésszámhoz vezet, vagyis

```
tav[i][j] =
1 + min(tav[i - 1][j - 1], tav[i][j - 1], tav[i - 1][j])
```

Szerkesztési távolság

Műveletek visszakeresése

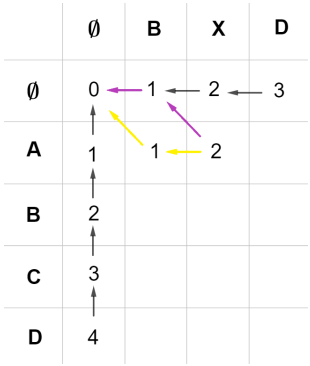


- Felfele nyíl: s'_1-et átalakítjuk s'_2c_2-vé, majd töröljük c_1-et
- Balra nyíl: s'_1c_1-et átalakítjuk s'_2-vé, majd hozzá adjuk c_2-t
- Átlós nyíl: ha $c_1 = c_2$ nem csinálunk semmit, egyébként átalakítjuk c_1-et c_2-vé

Milyen műveletekkel kezdődik a megoldás, ha a mátrix nulladik sorába, illetve oszlopába érünk a nyilakat követve?

Szerkesztési távolság

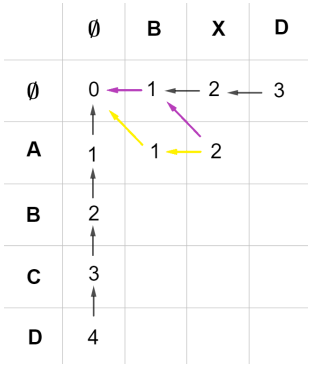
Műveletek visszakeresése



Milyen műveletek sorozatát jelöli a lila, illetve a sárga útvonal? (a $\rightarrow \ldots \rightarrow$ bx)

Szerkesztési távolság

Műveletek visszakeresése



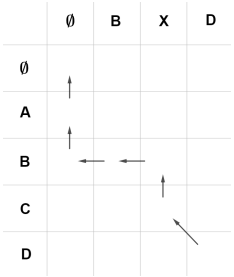
Milyen műveletek sorozatát jelöli a lila, illetve a sárga útvonal? (a $\rightarrow \ldots \rightarrow$ bx)

a $\rightarrow$ ba $\rightarrow$ bx

a $\rightarrow$ b $\rightarrow$ bx

Szerkesztési távolság

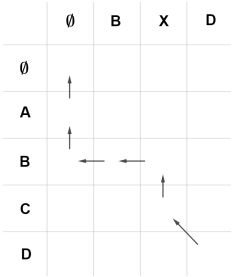
Műveletek visszakeresése



Milyen műveletek sorozatát jelöli az útvonal? (abcd $\rightarrow \ldots \rightarrow$ bxd)

Szerkesztési távolság

Műveletek visszakeresése



Milyen műveletek sorozatát jelöli az útvonal? (abcd $\rightarrow \ldots \rightarrow$ bxd)

Ahhoz, hogy könnyebben elképzeljük a műveleteket, felírhatjuk fordított sorrendben, hogy melyik karakterláncot alakítjuk melyikké minden lépésben:

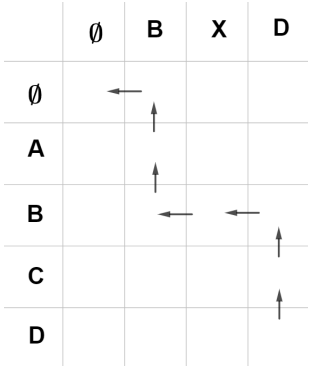
$(abcd, bxd) \leftarrow (abc, bx) \leftarrow (ab, bx) \leftarrow (ab, b) \leftarrow (ab, \emptyset) \leftarrow (a, \emptyset) \leftarrow (\emptyset, \emptyset)$

Ezek alapján a műveletek sorrendje:

$abcd \rightarrow bcd \rightarrow cd \rightarrow bcd \rightarrow bxcd \rightarrow bxd \rightarrow bxd$

Szerkesztési távolság

Műveletek visszakeresése



Hasonlóképpen itt a műveletek sorrendje:

$abcd \to babcd \to bbcd \to bcd \to bxcd \to bxdcd \to bxdd \to bxd$

Hátizsák diszkrét feladata

Feladat

Adott n tárgy, melyeknek ismerjük az értékét és térfogatát. Válasszuk ki a tárgyak egy olyan részhalmazát, melynek összértéke maximális és becsomagolható egy S összkapacitással rendelkező hátizsákba! A tárgyakat nem lehet darabolni.

- Láttuk, hogy a *greedy* módszer nem ad optimális eredményt a feladat ezen változatára.
- Egy megoldási lehetőség *backtracking*-gel generálni a tárgyak összes részhalmazát és ezek közül kiválasztani a megoldást, ez a megközelítés $O(2^n)$ időt igényelne.

Hátizsák diszkrét feladata

Megoldás

- A továbbiakban megoldjuk a feladatot a dinamikus programozás módszerével $\Theta(n \cdot S)$ időben.
- Mivel ez nem csak a bemenet n méretétől függ, hanem az S paramétertől is, nem tekinthetjük polinomiális futási időnek (mivel az S exponenciálisan nagy is lehet az n-hez képest), az ilyen futási időket pszeudopolinomiálisnak nevezzük.
- Legyen `maxÉrték[i][j]` a legnagyobb összérték, amelyet az első i tárgy felhasználásával kaphatunk úgy, hogy az össztérfogatuk j legyen.
- Ha az i. tárgy nem járulhat hozzá egy jobb megoldáshoz, akkor `maxÉrték[i][j] = maxÉrték[i - 1][j]`.
- Egyébként az i. tárgy berakása előtt a hátizsákba, ebben `j - térfogat[i]` összkapacitással rendelkező tárgyak voltak, tehát `maxÉrték[i][j] = maxÉrték[i - 1][j - térfogat[i]] + érték[i]`.
- Nyilván a két lehetőség közül azt választjuk, amelyik maximalizálja a kapott értéket.

Hátizsák diszkrét feladata

Megoldás

- A táblázat kitöltése közben megjegyezzük a `maxTárgy` és `maxTérfogat` változókban azt a pozíciót, ahol a legnagyobb kapott összérték található.
- Innen indulva építjük majd vissza a megoldáshoz tartozó tárgyak listáját.
- Ehhez felhasználjuk a `tárgy` tömböt, melynek az (i, j) pozícióján a legutolsó tárgynak az indexét jegyezzük meg, amelyet felhasználtunk a maximális összérték eléréséhez j összekapacitással az első i tárgy segítségével.
- Ez a megoldás $\Theta(n \cdot S)$ memóriabonyolultsággal rendelkezik.
- Számos módszer létezik a memóriafelhasználás csökkentésére, ezekre nem térünk ki.

Hátizsák diszkrét feladata

Pszeudokód

```
ALGORITMUS InitHátizsák(n, S, maxÉrték, tárgy, maxTárgy, maxTérfogat)
  MINDEN j = 0, S végezd el:
    maxÉrték[0][j] = 0
    tárgy[0][j] = 0
  VÉGE(Minden)
  maxTárgy = 0
  maxTérfogat = 0
VÉGE(Algoritmus)
```

Hátizsák diszkrét feladata

Pszeudokód

```
ALGORITMUS Másol(maxÉrték, tárgy, i, S)
  MINDEN j = 0, S végezd el:
    maxÉrték[i][j] = maxÉrték[i - 1][j]
    tárgy[i][j] = tárgy[i - 1][j]
  VÉGE(Minden)
VÉGE(Algoritmus)
```

Hátizsák diszkrét feladata

Pszeudokód

```
ALGORITMUS Frissít(maxTárgy, maxTérfogat, i, j, maxÉrték)
  HA (maxÉrték[i][j] > maxÉrték[maxTárgy][maxTérfogat]) akkor
    maxTárgy = i
    maxTérfogat = j
  VÉGE(Ha)
VÉGE(Algoritmus)
```

Hátizsák diszkrét feladata

Pszeudokód

```
ALGORITMUS Hátizsák(n, S, maxÉrték, tárgy, maxTárgy, maxTérfogat,
                    érték, térfogat)
  InitHátizsák(n, S, maxÉrték, tárgy, maxTárgy, maxTérfogat)
  MINDEN i = 1, n végezd el:
    Másol(maxÉrték, tárgy, i)
    MINDEN j = térfogat[i], S végezd el:
      segéd = maxÉrték[i - 1][j - térfogat[i]] + érték[i]
      HA (maxÉrték[i][j] < segéd)
        maxÉrték[i][j] = segéd
        tárgy[i][j] = i
        Frissít(maxTárgy, maxTérfogat, i, j, maxÉrték)
      VÉGE(Ha)
    VÉGE(Minden)
  VÉGE(Minden)
```

Hátizsák diszkrét feladata

Pszeudokód

```
  KiírMegoldás(maxTárgy, maxTérfogat, tárgy, térfogat)
  VISSZATÉRÍT: maxÉrték[maxTárgy][maxTérfogat]
VÉGE(Algoritmus)
```

Hátizsák diszkrét feladata

Pszeudokód

```
ALGORITMUS KiírMegoldás(maxTárgy, maxTérfogat, tárgy, térfogat)
  i = maxTárgy
  j = maxTérfogat
  AMÍG (j > 0) végezd el:
    segéd = tárgy[i][j]
    KI: segéd
    i = segéd - 1
    j = j - térfogat[segéd]
  VÉGE(Amíg)
VÉGE(Algoritmus)
```

Leghosszabb k páratlan számot tartalmazó növekvő részsorozat

Feladat

Adott az n elemű a sorozat és egy k természetes szám. Határozzuk meg a azon leghosszabb növekvő részsorozatát, amely pontosan k darab páratlan számot tartalmaz!

Példa: $n = 10, k = 2, a =$ [2 3 4 5 1 8 9 6 12 13]
Egy lehetséges megoldás: [2 3 4 5 8 12]
Ha $k = 4$ lett volna, a megoldás ez lenne: [2 3 4 5 8 9 12 13]

Leghosszabb k páratlan számot tartalmazó növekvő részsorozat

Megoldás

- Láthatjuk, hogy az optimális megoldás nem csak a elemeitől, hanem k értékétől is függ.
- Ebből arra következtetünk, hogy ismét több paraméterre lesz szükségünk.
- Legyen `hossz[i][p]` a leghosszab növekvő részsorozat hossza, amely az i. pozíción végződik és pontosan p páratlan számot tartalmaz.
- Innen a megoldás nagyon hasonló az eredeti feladatéhoz.
- Ha `a[i]` páros $\texttt{hossz[i][p]} = 1 + \max_{j=\overline{1,i-1}}\{\texttt{hossz[j][p]}, \text{ ahol } a[j] < a[i]\}$
- Ha `a[i]` páratlan
  $\texttt{hossz[i][p]} = 1 + \max_{j=\overline{1,i-1}}\{\texttt{hossz[j][p - 1]}, \text{ ahol } a[j] < a[i]\}$

Mátrixok optimális szorzása

Feladat

Adott n darab mátrix sorainak és oszlopainak a száma, úgy, hogy ezeket össze lehessen szorozni a megadott sorrendben, vagyis $oszlop_i = sor_{i+1}$ minden $i = \overline{1, n-1}$. Határozzuk meg a mátrixsorozat azon zárójelezését, amelyre az elvégzett skalárszorzások száma minimális!

- Emlékezzünk rá, hogy a mátrixok szorzása asszociatív, tehát bárhogyan csoportosítva a mátrixsorozatot, ugyanazt az eredményt kapjuk.
- Két mátrixot akkor tudunk összeszorozni, ha az első oszlopainak a száma megegyezik a második sorainak a számával.
- Egy $x \times y$ és egy $y \times z$ méretű mátrix szorzatához $x \cdot y \cdot z$ skalárszorzásra van szükségünk és az eredmény egy $x \times z$ méretű mátrix lesz.

Mátrixok optimális szorzása

Példa

$n = 3$, (10 100) (100 5) (5 50)

- A skalárszorzatok minimális száma 7500, amelyet úgy kapunk, hogy először összeszorozzuk az első mátrixot a másodikkal ($10 \cdot 100 \cdot 5 = 5000$), majd az így kapott 10×5 méretű mátrixot összeszorozzuk a harmadikkal ($10 \cdot 5 \cdot 50 = 2500$).
- Az optimális csoportosítás $((M_1 \times M_2) \times M_3)$
- Ha először a második és a harmadik mátrixot szoroztuk volna össze ($100 \cdot 5 \cdot 50 = 25000$), majd az első mátrixot az így kapott 100×50 méretűvel ($10 \cdot 100 \cdot 50 = 50000$), tízszer annyi skalárszorzásra lett volna szükségünk, mint a minimum.

Mátrixok optimális szorzása

Megoldás

- Jelöljük `minSzorzás[i][j]`-vel a skalárszorzások minimális számát, amellyel össze lehet szorozni az összes megadott mátrixot i és j között.
- `minSzorzás[i][i]` nyilván 0 lesz, mert egyetlen mátrixról lévén szó, nincs szükség szorzásra.
- $i < j$ esetén a zárójelezés valahol szétvágja a szorzatot, legyen ez a szétvágási pont a k. és $k+1$. mátrixok között.
- Ekkor az $i..j$ mátrixok összeszorzásához, először összeszorozzuk az $i..k$ mátrixokat, majd az $k+1..j$ mátrixokat, végül az eredményül kapott két mátrixot, melynek költsége $sor_i \cdot oszlop_k \cdot oszlop_j$.
- Mivel nem tudjuk előre, hogy mennyi k értéke, ezért végig kell próbálni az összes lehetőséget az $[i, j-1]$ intervallumból, és azt kiválasztani amelyikre `minSzorzás[i][j]` minimális lesz, vagyis `minSzorzás[i][j]` $= \min_{k=i..j-1} \{$`minSzorzás[i][k]` $+$ `minSzorzás[k + 1][j]` $+ sor_i \cdot oszlop_k \cdot oszlop_j\}$

Mátrixok optimális szorzása

Elemzés

- Mennyi lesz az így kapott algoritmus bonyolultsága?

Mátrixok optimális szorzása

Elemzés

- Mennyi lesz az így kapott algoritmus bonyolultsága?
- A memóriabonyolultság könnyen belátható, hogy $\Theta(n^2)$, mivel egy mátrix főátlóját és afeletti részét töltjük ki.
- Mivel egy-egy mező kitöltéséhez végig kell próbálni k lineáris nagyságrendű értékét, az időbonyolultság $O(n^3)$ és bizonyítható, hogy egyben $\Theta(n^3)$ is.
- Az implementáláshoz vegyük észre, hogy a mátrixot nem tölthetjük ki sorrol-sorra, mint a korábbi feladatok esetén.
- Egy (i, j) elem kiszámításához olyan elemekre van szükség, amelyek (i, j)-től balra vagy (i, j) alatt, de a főátlón vagy afelett helyezkednek el.
- Egy helyes kitöltési sorrend a főátlóval kezdeni, majd az ezzel párhuzamos átlókat egyenként kitölteni, míg el nem érjük az $(1, n)$ pozíción lévő megoldást.
- Segítségünkre lehet a memoizálás módszere, mivel ezt alkalmazva nem kell explicite kódolnunk a fenti sorrendet.

Zárójelezések

Feladat

Határozzuk meg, hogy hány olyan n zárójelet tartalmazó helyes zárójelezés létezik, mely pontosan k darab közvetlenül egymás után nyíló és záródó zárójelpárt tartalmaz.

Példa: $n = 6, k = 2$

A megoldás: 3

((()))
(()())
(())()
()(())
()()()

Zárójelezések

Megoldás

- Emlékezzünk a *backtracking* fejezetnél látott feladatra, melyben minden n hosszúságú zárójelezést generáltuk.
- Hogyan ellenőríztük, hogy egy zárójelezés helyes-e?
- Egy adott zárójelezést elképzelhetünk a síkban, felírhatjuk két dimenzióban.
- Az origóból indulunk, egy (i, j) pozícióból egy nyitott zárójellel az $(i + 1, j + 1)$, egy zárt zárójellel az $(i + 1, j - 1)$ pozícióra jutunk.
- Soha nem kerülhetünk az Ox tengely alá és az utolsó zárójellel az $(n, 0)$ pozícióra kell jussunk.

Zárójelezések

Megoldás

- Ez alapján elindulhatunk a következő megoldási szálon.
- Legyen `hany[i][j]`, hogy i darab zárójellel hányféleképpen juthatunk a j. szintre.
- Tekinthetjük úgy, hogy `hany[0][0] = 1`
- Ha az összes helyes zárójelezés számára lennénk kíváncsiak, ezt `hany[n][0]` értéke adná meg.
- A korábban látott gondolatmenet alapján `hany[i][j] = hany[i - 1][j - 1] + hany[i - 1][j + 1]`, ha $j > 0$, egyébként csak az összeg második tagjára van szükségünk.

Zárójelezések

Megoldás

- Nekünk viszont csak azon zárójelezések számára van szükségünk, melyek pontosan k darabszor haladnak felfele, majd azonnal lefele.
- Vagyis biztosan szükségünk lesz még egy paraméterre, amely azt számolja, hogy ez eddig hányszor fordult elő.
- Ezek alapján `hany[i][j][p]` jelentése az lenne, hogy hányféleképpen juthatunk i zárójel letevése után a j. szintre úgy, hogy eddig p-szer tettünk közvetlenül egymás után nyitott majd zárt zárójelet.
- Az nyilvánvaló, hogy p akkor növekszik, ha $(i-1, j+1)$-ből érkezünk, viszont csak akkor, ha oda $(i-2, j)$-ből érkeztünk, vagyis szükség van egy negyedik paraméterre, amely azt jelzi, hogy adott pozícióba honnan érkeztünk, ami ekvivalens azzal, hogy az utoljára letett zárójel nyitott, vagy zárt volt.

Zárójelezések

Megoldás

- Az eddigiek alapján a következő megoldás alakul ki.
- Legyen `hany[i][j][p][z]` azon i darab zárójelből álló zárójelezések száma, melyekben a nyitott és zárt zárójelek számának különbsége j, p darabszor fordul elő bennük közvetlenül egymás után egy nyitott és egy zárt zárójel és az i. zárójelet a z jelöli, ahol $z = 0$, ha az i. zárójel nyitott és $z = 1$, ha az i. zárójel zárt.
- Kezdetben `hany[0][0][0][0] = 1`, a többi érték 0.
- A megoldás `hany[n][0][k][1]`-ben lesz.
- Odafigyelve arra, hogy ne kérjünk le negatív j vagy p paramétert:

```
hany[i][j][p][0] =
hany[i - 1][j - 1][p][0] + hany[i - 1][j - 1][p][1]
hany[i][j][p][1] =
hany[i - 1][j + 1][p - 1][0] + hany[i - 1][j + 1][p][1]
```

Bónusz

Feladat

Határozzuk meg, hogy hány olyan n zárójelből álló zárójelezés létezik, amely legalább egyszer eljut a k. szintre!

Példa: $n = 6, k = 3$
Megoldás: 1
((()))
Meddig? Január 23., 23:59
Hova? Canvas privát üzenet
Mit? Forráskód, az ötlet rövid magyarázata, időbonyolultság
Mennyi? 2 pont
Hányszor? Diákonként egy beküldés

Összefoglalás

Jeff Erickson a következő lépéseket ajánlja egy feladat megoldásához a dinamikus programozás módszerével:

1. Fogalmazzuk meg a feladatot rekurzívan, rekurzív összefüggéseket használó képletek, vagy rekurzív hívásokat használó algoritmus formájában.
   1. Jelentsük ki a feladatot természetes nyelven, precíz és koherens magyar megfogalmazást használva. Nem azt, hogy *hogyan* akarjuk megoldani a feladatot, hanem, hogy mi a feladat.
   2. Adjuk meg a feladat megoldását rekurzívan, ugyanannak a feladatnak a kisebb méretű megoldásait felhasználva.

Összefoglalás

2. Építsük fel a megoldást lentről felfele (bottom-up). Kezdjük az alapesetekkel és ezekből jussunk el a köztes részfeladatokon keresztül a végső megoldásig.
   1. Határozzuk meg a részfeladatok paraméterezését.
   2. Válasszunk egy adatszerkezetet, amelyben minden részfeladat megoldását eltárolhatjuk.
   3. Határozzuk meg, hogy egy általános részfeladat melyik más részfeladatoktól függ. Ehhez készíthetünk egy rajzot is nyilakkal.
   4. Keressünk egy megfelelő sorrendet a részfeladatok megoldásához, ehhez használjuk az előző lépésben kapott eredményt. Kezdjük az alapesetekkel. Minden részfeladatot hamarabb kell megoldanunk, mint az összes tőle függő részfeladatot.
   5. Elemezzük az algoritmus memória- és időbonyolultságát.
   6. Írjuk le az algoritmust.