Dinamikus programozás stratégiája

A dinamikus programozás stratégiája

1. Az [optimális] megoldás szerkezetének tanulmányozása.
2. Részproblémákra és összetevőkre bontás úgy, hogy:
   - az összetevőktől való függés körmentes legyen;
   - minden részprobléma [optimális] megoldása kifejezhető legyen (rekurzívan) az összetevők [optimális] megoldásaival.
3. Részproblémák [optimális] megoldásának kifejezése (rekurzívan) az összetevők [optimális] megoldásaiból.

Dinamikus programozás stratégiája

4. Részproblémák [optimális] megoldásának kiszámítása alulról-felfelé haladva:

- A részproblémák kiszámítási sorrendjének meghatározása. Olyan sorba kell rakni a részproblémákat, hogy minden p részprobléma minden összetevője (ha van) előbb szerepeljen a felsorolásban, mint p.
- A részproblémák kiszámítása alulról-felfelé haladva, azaz táblázatkitöltéssel.

5. Egy [optimális] megoldás előállítása a 4. lépésben kiszámított (és tárolt) információkból.

Dinamikus programozás – leghosszabb közös rész

Adjuk meg két sorozat $X=(x_1,x_2, \ldots, x_n)$ és $Y=(y_1,y_2, \ldots, y_m)$ leghosszabb közös részsorozatát!

Egy sorozat akkor részsorozata egy másiknak, ha abból elemek elhagyásával megkapható.

ALMA+KALAP → ALA

Dinamikus programozás – leghosszabb közös rész

Jelölje $XX_i=(x_1,x_2, \ldots, x_i)$ az X,

$YY_j=(y_1,y_2, \ldots, y_j)$ pedig az Y sorozat egy-egy prefixét.

Legyen $Z=(z_1,z_2, \ldots, z_k)$ egy megoldása a feladatnak.

ALMA+KALAP → ALA

Dinamikus programozás – leghosszabb közös rész

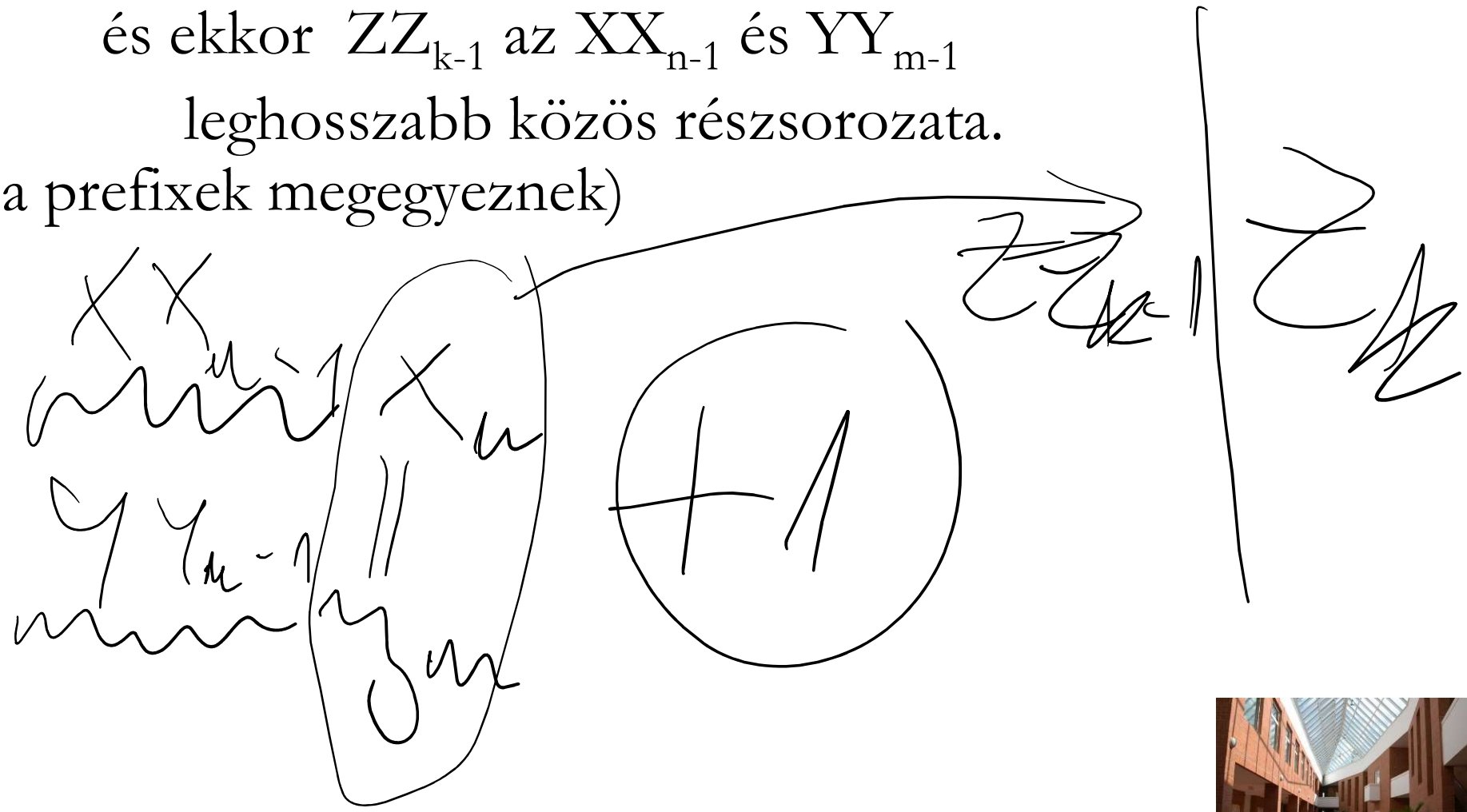
Ha $x_n=y_m$,

akkor $z_k=x_n=y_m$,

és ekkor ZZ_{k-1} az XX_{n-1} és YY_{m-1}

leghosszabb közös részsorozata.

(azaz a prefixek megegyeznek)



Dinamikus programozás – leghosszabb közös rész

Ha $x_n \neq y_m$,

akkor Z

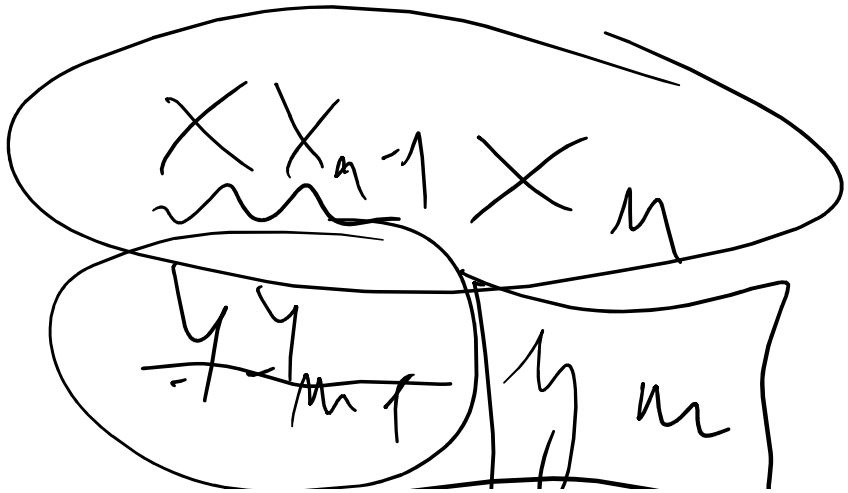
az XX_{n-1} és Y

vagy

az X és YY_{m-1}

leghosszabb közös ~~részsorozata~~.

(azaz vagy az x_n vagy az y_m nélküli lesz a jó)



Dinamikus programozás – leghosszabb közös rész

Ha $x_n=y_m$, akkor $z_k=x_n=y_m$, és ZZ_{k-1} az XX_{n-1} és YY_{m-1} leghosszabb közös részsorozata.

Ha $x_n \neq y_m$, akkor Z az XX_{n-1} és Y vagy az X és YY_{m-1} leghosszabb közös részsorozata.

Az XX_i és YY_j leghosszabb közös részsorozatának hossza:

$$h(i,j) = \begin{cases} 0 & ha \quad i = 0 \quad vagy \quad j = 0 \\ h(i-1, j-1) + 1 & ha \quad x_i = y_j \\ \max(h(i-1,j), h(i,j-1)) & egyébként \end{cases}$$

Dinamikus programozás – leghosszabb közös rész

A rekurzív algoritmus:

```
Hossz(i,j):
   Ha i=0 vagy j=0 akkor Hossz:=0
   különben
    ha X(i)=Y(j)
      akkor Hossz:=Hossz(i-1,j-1)+1
      különben Hossz:=max(Hossz(i-1,j),Hossz(i,j-1))
Függvény vége.
```

$$h(i,j)=\begin{cases} 0 & ha \quad i=0 \quad vagy \quad j=0 \\ h(i-1,j-1)+1 & ha \quad x_i = y_j \\ \max(h(i-1,j),h(i,j-1)) & egyébként \end{cases}$$

Dinamikus programozás – leghosszabb közös rész

A jó megoldás – rekurzió memorizálással:
A H mátrixot kezdetben -1 értékekkel töltjük ki!
Amit egyszer kiszámolunk, azt tároljuk és nem számoljuk ki újra!

```
Hossz(i,j):
  Ha H(i,j)=-1 akkor
    Ha i=0 vagy j=0 akkor H(i,j):=0
    különben
      ha X(i)=Y(j)
       akkor H(i,j):=Hossz(i-1,j-1)+1
       különben H(i,j):=max(Hossz(i-1,j),Hossz(i,j-1))
  Elágazás vége
  Hossz:=H(i,j)
Függvény vége.
```

$$h(i,j)=\begin{cases} 0 & ha \quad i=0 \quad vagy \quad j=0 \\ h(i-1,j-1)+1 & ha \quad x_i=y_j \\ \max(h(i-1,j),h(i,j-1)) & egyébként \end{cases}$$

Dinamikus programozás – toronyépítés

Adott M_1, M_2, …, M_n méretű kockákból (amelyek súly szerint csökkenő sorrendbe vannak rendezve) építsünk maximális magasságú stabil tornyot!

A stabil toronyban felfelé haladva a méret és a súly is csökken.

$$M_{i_1}; M_{i_2}; \ldots; M_{i_k} (i_1 < \ldots < i_k)$$

Dinamikus programozás – toronyépítés

Tegyük fel, hogy

$$M_{i_1}; M_{i_2}; \ldots; M_{i_k} \, (i_1 < \ldots < i_k)$$

megoldás!

Ekkor $n=i_{k-1}$-re

$$M_{i_1}; M_{i_2}; \ldots; M_{i_{k-1}} \, (i_1 < \ldots < i_{k-1})$$

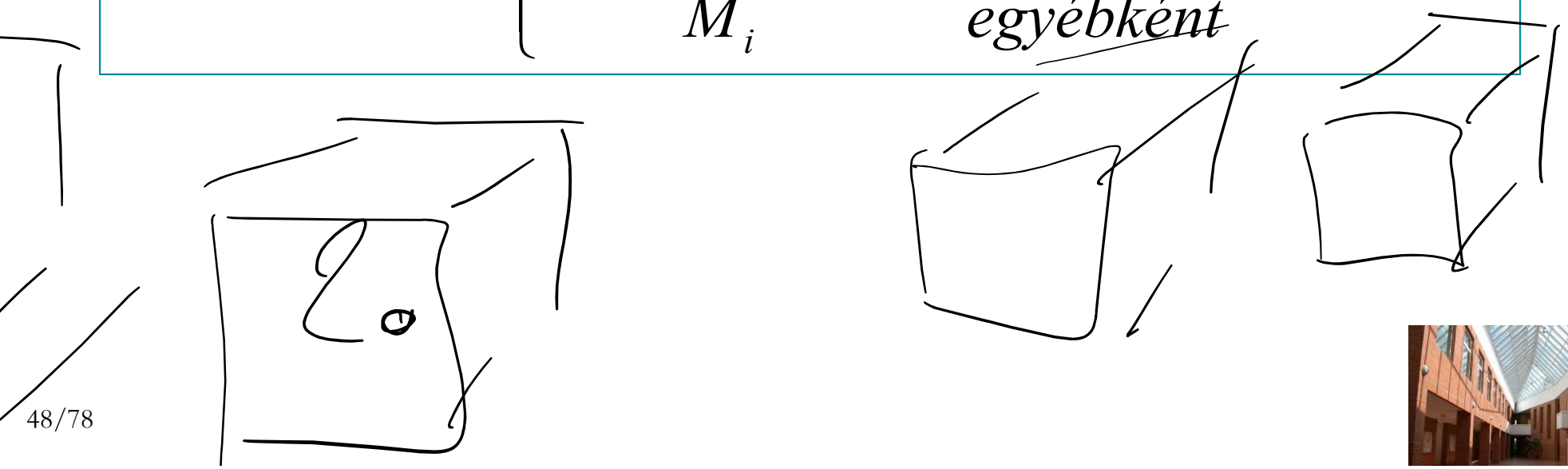
is megoldás,

azaz az adott részproblémának megoldása a megoldás megfelelő részlete.

Dinamikus programozás – toronyépítés

Tehát a feladat a legmagasabb olyan torony magasságának kiszámolása, ahol az i. kocka van legfelül:

$$Kocka(i) = \max \begin{cases} Kocka(1) + M_i & ha \quad M_i \le M_1 \\ Kocka(2) + M_i & ha \quad M_i \le M_2 \\ \dots & \\ M_i & egyébként \end{cases}$$



Dinamikus programozás – toronyépítés

A rekurzív megoldás:

```
Kocka(i):
   K:=M(i)
   Ciklus j=1-től i-1-ig
      Ha M(i)≤M(j) akkor
         Ha Kocka(j)+M(i)>K akkor K:=Kocka(j)+M(i)
   Ciklus vége
   Kocka:=K
Függvény vége.
```

A rengeteg rekurzív hívás miatt nagyon lassú.

Kocka(i) kiszámításához csak a korábbiakra van szükség.

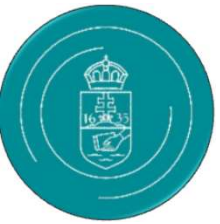
$$Kocka(i) = \max \begin{cases} Kocka(1) + M_i & ha \quad M_i \le M_1 \\ Kocka(2) + M_i & ha \quad M_i \le M_2 \\ \dots & \\ M_i & egyébként \end{cases}$$

Dinamikus programozás – toronyépítés

```
Toronyépítés:
  Kocka(1):=M(1)
  Ciklus i=2-től N-ig
    K:=M(i)
    Ciklus j=1-től i-1-ig
      Ha M(i)≤M(j) akkor
        Ha Kocka(j)+M(i)>K
          akkor K:=Kocka(j)+M(i)
    Ciklus vége
    Kocka(i):=K
  Ciklus vége
Eljárás vége.
```

$$Kocka(i) = \max \begin{cases} Kocka(1) + M_i & ha & M_i \le M_1 \\ Kocka(2) + M_i & ha & M_i \le M_2 \\ \dots & & \\ M_i & egyébként & \end{cases}$$

A táblázatkitöltős megoldás.



Dinamikus programozás – toronyépítés

Ezzel még nincs kész a megoldás, csak azt tudjuk minden i-re, hogy mekkora a legmagasabb torony, amikor az i-edik kocka van felül.

A megoldás ezen számok maximuma.

Ha bevezetnénk egy N+1., 0 méretű kockát, akkor a megoldás értéke Kocka(N+1) lenne.

Ha a tornyot fel is kellene építeni, akkor még azt is tárolni kell, hogy melyik kockát melyikre kell rátenni.

Dinamikus programozás – toronyépítés

```
Toronyépítés:
  Kocka(1):=M(1); Mire(1):=0; M(N+1):=0
  Ciklus i=2-től N+1-ig
    K:=M(i); Mire(i):=0
    Ciklus j=1-től i-1-ig
      Ha M(i)≤M(j) akkor
        Ha Kocka(j)+M(i)>K
          akkor K:=Kocka(j)+M(i); Mire(i):=j
    Ciklus vége
    Kocka(i):=K
  Ciklus vége
Eljárás vége.
```

Dinamikus programozás – toronyépítés

```
Torony:
   Toronyépítés
   Toronykiírás(N+1)
Eljárás vége.


Toronykiírás(i):
   Ha Mire(i)>0 akkor Toronykiírás(Mire(i))
                      Ki: Mire(i)
Eljárás vége.
```

Legfelül van a Mire(N+1). kocka, alatta a Mire(Mire(N+1)). , alatta a Mire(...).

Dinamikus programozás – kemence

Feladat:

Egy fazekasműhelyben N tárgy vár kiégetésre. A kemencébe a tárgyak a beérkezés sorrendjében tehetők be, egyszerre legfeljebb K darab.

Minden tárgynak ismerjük a minimális égetési idejét, amennyit legalább a kemencében kell töltenie.

Adjuk meg a minimális időt, ami alatt minden tárgy kiégethető!

Dinamikus programozás – kemence

Tegyük fel, hogy

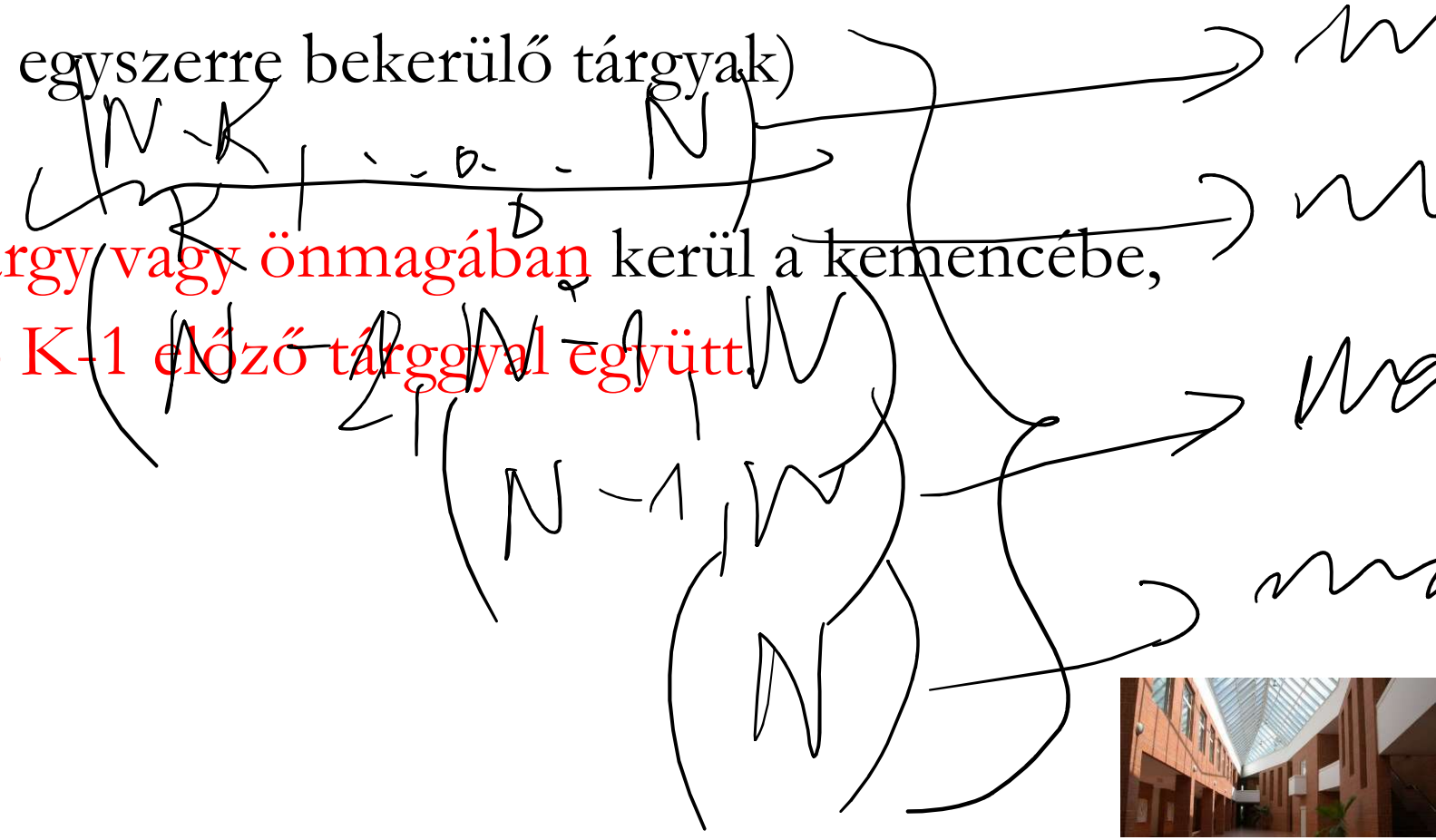
$$[(1,\ldots,i_1),\ldots(\ldots,i_{m-1}),(i_{m-1}+1,\ldots,N)]$$

a megoldás.

(zárójelezve az egyszerre bekerülő tárgyak)

Ekkor az N. tárgy vagy önmagában kerül a kemencébe,

vagy legfeljebb K-1 előző tárggyal együtt.



Dinamikus programozás – kemence

Számítsuk ki az első i tárgy kiégetéséhez szükséges időt!

$$Idő(i) = \min \begin{cases} Idő(i-1) + Éget(i) \\ Idő(j-1) + \max(Éget(j..i)) \quad ahol \quad i-(j-1) \le K \end{cases}$$

Idő(0)=0

Dinamikus programozás – kemence

```
Kemence:
  Idő(0):=0
  Ciklus i=1-től N-ig
    H:=Idő(i-1)+Éget(i); max:=Éget(i); G:=i
    j:=i-1
    Ciklus amíg j>0 és i+j-1≤K
      Ha max < Éget(j) akkor max:=Éget(j)
      Ha Idő(j-1)+max<H akkor H:=Idő(j-1)+max; G:=j
      j:=j-1
    Ciklus vége
    Idő(i):=H; Db(i):=i-G+1; Kivel(i):= G
  Ciklus vége
Eljárás vége.
```

Db(i) – az i. tárggyal hány tárgyat égetünk együtt?

Kivel(i) – melyik a legkisebb sorszámú, amivel együtt égetjük

$$Idő(i) = \min \begin{cases} Idő(i-1) + Éget(i) \\ Idő(j-1) + \max\left(Éget(j..i)\right) \quad ahol \quad i-(j-1) \le K \end{cases}$$

Dinamikus programozás – kemence

```
Kemence_megoldás:
        Kemence;
        Kemence_eredmény(N)
Eljárás vége.
```

Darabszámmal:

```
Kemence_eredmény(i):
  Ha i>Db(i) akkor Kemence_eredmény(i-Db(i))
  Ki: i-Db(i)+1,i,Idő(i)
Eljárás vége.
```

Kivel együtt:

```
Kemence_eredmény(i):
  Ha Kivel(i)>1 akkor Kemence_eredmény(Kivel(i)-1)
  Ki: Kivel(i), i, Idő(i)
Eljárás vége.
```


Dinamikus programozás – kemence

Feladat:

Egy fazekasműhelyben N tárgy vár kiégetésre.

A kemencébe a beérkezés sorrendjében tehetők be,

egyszerre legfeljebb S összsúlyú tárgy tehető.

Minden tárgynak ismerjük a minimális égetési idejét, amennyit legalább a kemencében kell töltenie.

Adjuk meg a minimális időt, ami alatt minden tárgy kiégethető!

Dinamikus programozás – kemence

Megoldás:

Tegyük fel, hogy $((1,\ldots,i_1), \ldots, (\ldots, i_{m-1}) , (i_{m-1}+1,\ldots,N))$ a megoldás.

Ekkor az i. tárgy vagy önmagában kerül a kemencébe,

vagy legfeljebb S súlyban, előző tárgyakkal együtt.

Számítsuk ki az első i tárgy kiégetéséhez szükséges időt!
(Idő(0)=0)

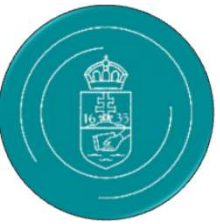
$$Idő(i) = \min\begin{cases} Idő(i-1) + Éget(i) \\ Idő(j-1) + \max(Éget(j..i)) \quad ahol \quad \sum Súly(j..i) \le S \end{cases}$$

A maximum mellett az összeget is kell számolnunk!

Dinamikus programozás – kemence

```
Kemence:
  Idő(0):=0
  Ciklus i=1-től N-ig
    H:=Idő(i-1)+Éget(i); max:=Éget(i); G:=i
    j:=i-1; Su:=Súly(i)
    Ciklus amíg j>0 és Su+Súly(j)≤S
      Ha max<Éget(j) akkor max:=Éget(j)
      Su:=Su+Súly(j)
      Ha Idő(j-1)+max<H akkor H:=Idő(j-1)+max; G:=j
      j:=j-1
    Ciklus vége
    Idő(i):=H; Db(i):=i-G+1; Kivel(i):=G
  Ciklus vége
Eljárás vége.
```

$$Idő(i) = \min\begin{cases} Idő(i-1) + Éget(i) \\ Idő(j-1) + \max(Éget(j..i)) \quad ahol \quad \sum Súly(j..i) \le S \end{cases}$$



Dinamikus programozás stratégiája

A dinamikus programozás stratégiája

1. Az [optimális] megoldás szerkezetének tanulmányozása.
2. Részproblémákra és összetevőkre bontás úgy, hogy:
   - az összetevőktől való függés körmentes legyen;
   - minden részprobléma [optimális] megoldása kifejezhető legyen (rekurzívan) az összetevők [optimális] megoldásaival.
3. Részproblémák [optimális] megoldásának kifejezése (rekurzívan) az összetevők [optimális] megoldásaiból.

Dinamikus programozás stratégiája

4. Részproblémák [optimális] megoldásának kiszámítása alulról-felfelé haladva:

- A részproblémák kiszámítási sorrendjének meghatározása. Olyan sorba kell rakni a részproblémákat, hogy minden p részprobléma minden összetevője (ha van) előbb szerepeljen a felsorolásban, mint p.
- A részproblémák kiszámítása alulról-felfelé haladva, azaz táblázatkitöltéssel.

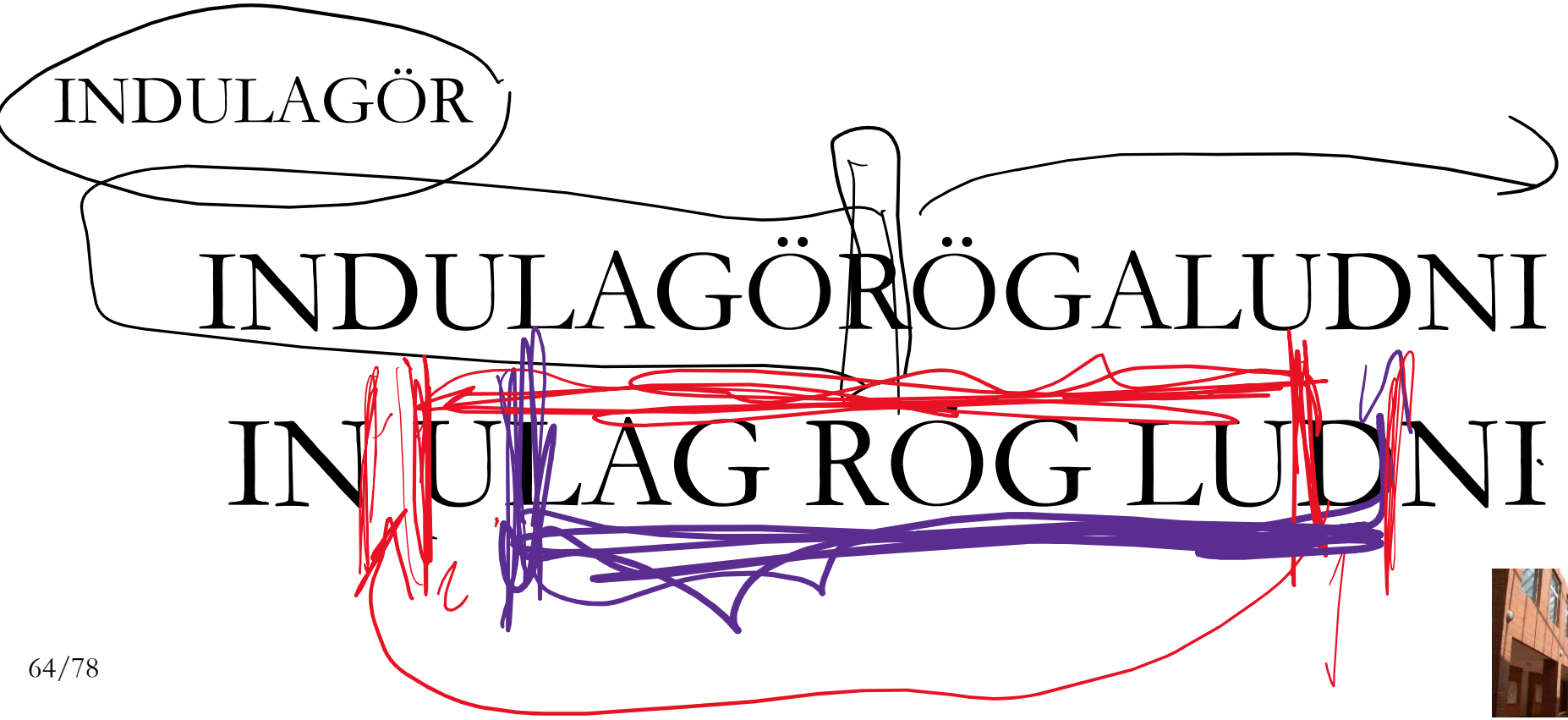
5. Egy [optimális] megoldás előállítása a 4. lépésben kiszámított (és tárolt) információkból.

Dinamikus programozás – Tükörszó

Feladat:

Egy szóba minimálisan hány karaktert kell beszúrni, hogy tükörszó legyen belőle

(azaz ugyanaz legyen balról-jobbra és jobbról balra olvasva is)!



Dinamikus programozás – Tükörszó

Feladat:

Egy szóba minimálisan hány karaktert kell beszúrni, hogy tükörszó legyen belőle

(azaz ugyanaz legyen balról-jobbra és jobbról balra olvasva is)!

Megoldás:

Tetszőleges S szóra az SS^T szó biztos tükörszó, tehát a feladatnak biztosan van megoldása!

Alternatív feladat: minimálisan hány betűt kell törölni?

Dinamikus programozás – Tükörszó

Az egybetűs szavak biztosan tükörszavak.

Az azonos kezdő- és végbetűjű szavak tükörszóvá alakításához elég a középső rész átalakítása.

Ha az első és az utolsó betű különbözik, akkor két lehetőségünk van a tükörszóvá alakításra:

- az első betűt szúrjuk be a szó végére;
- az utolsó betűt szúrjuk be a szó elejére.

Dinamikus programozás – Tükörszó

Megoldás:

M(i, j):

minimum hány karaktert kell beszúrni, hogy a szöveg i. és j. karaktere közötti részét tükörszóvá alakítsuk!

Három eset van:

- egybetűs részek
- a két szélső karakter egyforma
- az első és az utolsó karakter különbözik

Dinamikus programozás – Tükörszó

Három eset van:

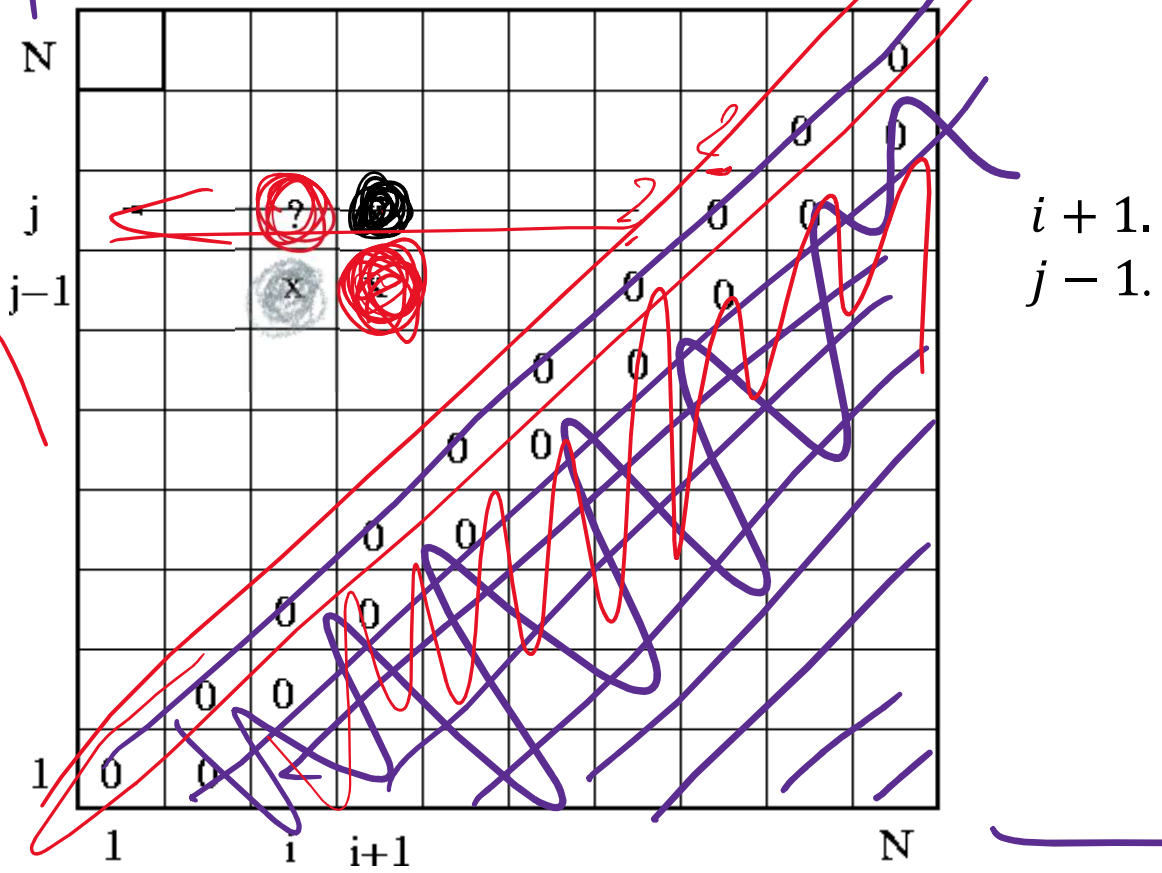
- egybetűs részek
- a két szélső karakter egyforma
- az első és az utolsó karakter különbözik

$$M(i,j) = \begin{cases} 0 & ha \quad i \geq j \\ M(i+1, j-1) & ha \quad i < j \quad és \quad S_i = S_j \\ 1 + \min(M(i+1,j), M(i,j-1)) & ha \quad i < j \quad és \quad S_i \neq S_j \end{cases}$$

Probléma: a rekurzió nem hatékony!

Dinamikus programozás – Tükörszó

Jó sorrendű táblázatkitöltés kell:



$i+1.$

$j-1.$

$$M(i,j)=\begin{cases}0 & ha \quad i \geq j \\ M(i+1,j-1) & ha \quad i<j \quad és \quad S_i = S_j \\ 1+\min(M(i+1,j),M(i,j-1)) & ha \quad i<j \quad és \quad S_i \neq S_j\end{cases}$$

Dinamikus programozás – Tükörszó

```
Tükörszó(S,M):
  M() kezdő feltöltése (0-val)
  Ciklus j=2-től N-ig
    M(j,j):=0
    Ciklus i=j-1-től 1-ig -1-esével
      Ha S(i)=S(j) akkor M(i,j):=M(i+1,j-1)
        különben M(i,j):=1+Min(M(i+1,j),M(i,j-1))
    Ciklus vége
  Ciklus vége
  Tükörszó:=M(1,N)
Függvény vége.
```

$$M(i,j)=\begin{cases} 0 & ha \quad i \geq j \\ M(i+1,j-1) & ha \quad i<j \quad és \quad S_i = S_j \\ 1+\min(M(i+1,j),M(i,j-1)) & ha \quad i<j \quad és \quad S_i \neq S_j \end{cases}$$

Dinamikus programozás – Tükörszó

```
Tükörszó kiírása:
  Ha Tükörszó(S,E)
     akkor i:=1; j:=N
           Ciklus amíg i<j
              Ha S(i)=S(j) akkor i:=i+1; j:=j-1
              különben Ha M(i,j)=M(i+1,j)
              akkor {S(i) a j. betű mögé}
              különben {S(j) az i. betű elé}
           Ciklus vége
Eljárás vége.
```

$$M(i,j)=\begin{cases} 0 & ha \quad i \geq j \\ M(i+1,j-1) & ha \quad i<j \quad és \quad S_i = S_j \\ 1+\min(M(i+1,j),M(i,j-1)) & ha \quad i<j \quad és \quad S_i \neq S_j \end{cases}$$

Dinamikus programozás – rúd darabolás

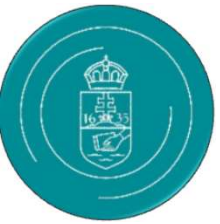
Feladat:

Egy fémrudat megadott számú darabra kell felvágni úgy, hogy a vágások pontos helyét is tudjuk.

A vágások tetszőleges sorrendben elvégezhetőek.

Egy vágás költsége megegyezik annak a darabnak a hosszával, amit éppen (két darabra) vágunk.

Adjuk meg a vágási műveletsor optimális összköltségét és egy olyan vágási sorrendet, amely optimális költséget eredményez.



Dinamikus programozás – rúd darabolás

Feladat:

Egy fémrudat megadott számú darabra kell felvágni úgy, hogy a vágások pontos helyét is tudjuk. A vágások tetszőleges sorrendben elvégezhetőek. Egy vágás költsége megegyezik annak a darabnak a hosszával, amit éppen (két darabra) vágunk. Adjuk meg a vágási műveletsor optimális összköltségét, és egy olyan vágási sorrendet, amely optimális költséget eredményez.

A megoldás elemzése:

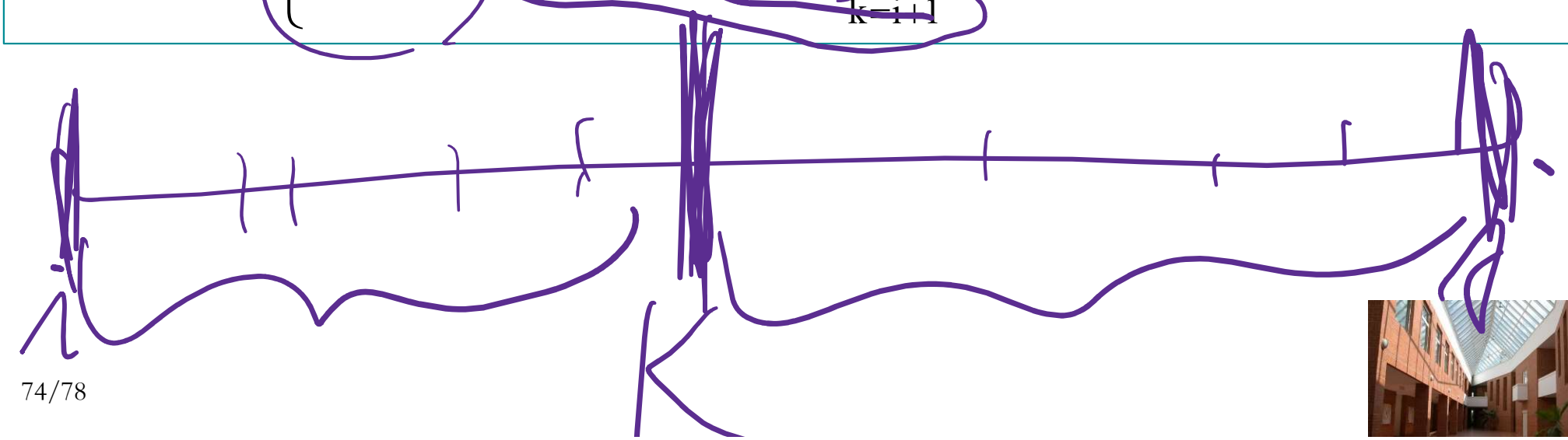
Ha az optimális vágássorozatban először a K. helyen történik a vágás, akkor a $V_0..V_k$ és a $V_k..V_{n+1}$ rúd vágássorozata is optimális lesz.

Dinamikus programozás – rúd darabolás

Megoldás:

Számítsuk ki minden (i,j) rúddarabra, hogy mi az optimális vágási költsége!

$$Opt(i,j) = \begin{cases} 0 & \text{ha} \quad j = i+1 \\ V_j - V_i + \min_{k=i+1}^{j-1}\left(Opt(i,k) + Opt(k,j)\right) & \text{ha} \quad i+1 < j \end{cases}$$

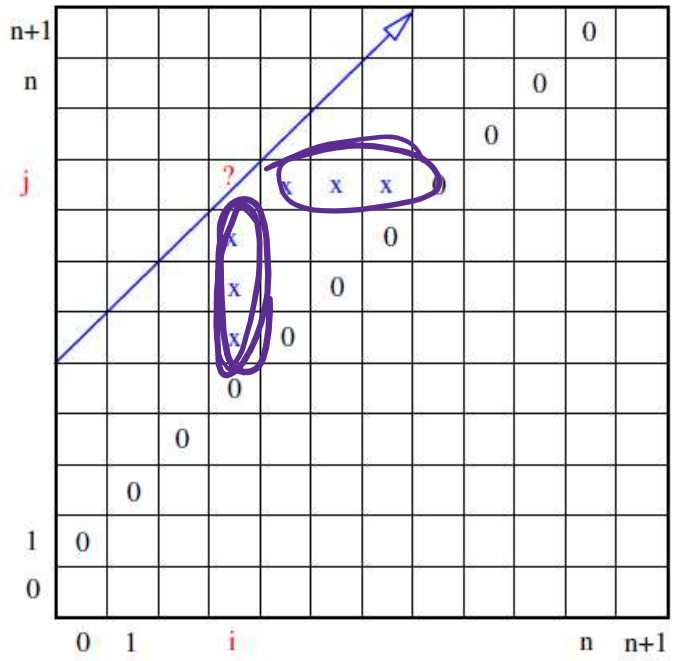


Dinamikus programozás – rúd darabolás

Megoldás:

Számítsuk ki minden (i,j) rúddarabra, hogy mi ennek az optimális vágási költsége!

$$Opt(i,j) = \begin{cases} 0 & \text{ha} \quad j = i+1 \\ V_j - V_i + \min\limits_{k=i+1}^{j-1}\left(Opt(i,k) + Opt(k,j)\right) & \text{ha} \quad i+1 < j \end{cases}$$



Dinamikus programozás – rúd darabolás

```
Rúd:
  Ciklus i=0-tól N-ig
    Opt(i,i+1):=0; S(i,i+1):=0
  Ciklus vége
  Ciklus u=2-től N+1-ig
    Ciklus i=0-től N-u+1-ig
      j:=i+u; Min:=+∞
      Ciklus k=i+1-től j-1-ig
        Ha Opt(i,k)+Opt(k,j)<Min
           akkor Min:=Opt(i,k)+Opt(k,j); Hol:=k
      Ciklus vége
      Opt(i,j):=V(j)-V(i)+Min; S(i,j):=Hol
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

A megoldás: Opt(0,N+1)

Dinamikus programozás stratégiája

A dinamikus programozás stratégiája

1. Az [optimális] megoldás szerkezetének tanulmányozása.
2. Részproblémákra és összetevőkre bontás úgy, hogy:
   - az összetevőktől való függés körmentes legyen;
   - minden részprobléma [optimális] megoldása kifejezhető legyen (rekurzívan) az összetevők [optimális] megoldásaival.
3. Részproblémák [optimális] megoldásának kifejezése (rekurzívan) az összetevők [optimális] megoldásaiból.

Dinamikus programozás stratégiája

4. Részproblémák [optimális] megoldásának kiszámítása alulról-felfelé haladva:

- A részproblémák kiszámítási sorrendjének meghatározása. Olyan sorba kell rakni a részproblémákat, hogy minden p részprobléma minden összetevője (ha van) előbb szerepeljen a felsorolásban, mint p.
- A részproblémák kiszámítása alulról-felfelé haladva, azaz táblázatkitöltéssel.

5. Egy [optimális] megoldás előállítása a 4. lépésben kiszámított (és tárolt) információkból.