Dinamikus programozás I.

Dinamikus programozás – kincs

Feladat:

Egy N*M-es téglalap alakú területen egy járművel szedhetjük össze az elrejtett kincseket, a bal felső sarokból a jobb alsó felé haladva. A terep jobbra és lefelé lejt, azaz a jármű csak jobbra és lefelé haladhat. Add meg, hogy maximum hány kincset gyűjthet be és ehhez merre kell mennie!

A megoldás elemzése:

Ha a megoldás egy $L_1, L_2, \ldots, L_j, \ldots, L_k$ úton érhető el, akkor az $L_1, L_2, \ldots, L_j$ egy olyan út, amin elért pontban az odáig begyűjthető legtöbb kincset kapjuk.

Kezdőpozíció: (1,1)

Végpozíció: (N,M)

Dinamikus programozás – kincs

Megoldás:

Számítsuk ki minden lehetséges mezőre, hogy addig eljutva mennyi a maximálisan begyűjthető kincsek száma!

$$Gy(i,j)=\begin{cases} -1 & \text{ha} \quad i=0 \quad \text{vagy} \quad j=0 \\ \max(Gy(i-1,j),Gy(i,j-1)) & \text{ha} \quad Kincs(i,j)="" \\ \max(Gy(i-1,j),Gy(i,j-1))+1 & \text{ha} \quad Kincs(i,j)="*" \end{cases}$$

Figyelni kell a szélekre!

		*					
		*					
*				*			
*							*
		*			*	*	

Dinamikus programozás – kincs

```
Kincsek:
  Gy(0,2..M):=-1; Gy(2..N,0):=-1
  Gy(0,1):=0; Gy(1,0):=0 {az (1,1)-et jó számolja}
  Ciklus i=1-től N-ig
    Ciklus j=1-től M-ig
      Ha Kincs(i,j)="*" akkor k:=1 különben k:=0
      Ha Gy(i,j-1)>Gy(i-1,j)
         akkor Gy(i,j):=Gy(i,j-1)+k
      különben Gy(i,j):=Gy(i-1,j)+k
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

A megoldás: Gy(N,M)

Dinamikus programozás – kincs

Ha a bejárt útra is szükségünk lenne, akkor a Gy mátrix alapján az út visszakövethető, amíg a kezdőponthoz nem érünk.

Azt kell figyelnünk, hogy minden helyre a nagyobb értékű szomszédból kellett jönnünk.

```
Útkiírás(i,j):
  Ha i≠1 vagy j≠1 akkor
     Ha Gy(i,j-1)>Gy(i-1,j)
        akkor Útkiírás(i,j-1); Ki: ”J”
     különben Útkiírás(i-1,j); Ki: ”L”
Eljárás vége.
```

A megoldás a számítások után:
Útkiírás(N,M)

Dinamikus programozás – kincs

Feladat:

Egy N*M-es téglalap alakú területen egy járművel szedhetjük össze az elrejtett kincseket. A terep jobbra és lefelé lejt, azaz a jármű csak jobbra és lefelé haladhat. Bizonyos helyeken akadályok vannak, ahova nem léphet! Add meg, hogy maximum hány kincset gyűjthet be!

A megoldás elemzése:

Ha a megoldás egy $L_1, L_2, \ldots, L_j, \ldots, L_k$ úton érhető el, akkor az $L_1, L_2, \ldots, L_j$ egy olyan út, amin elért pontban az odáig begyűjthető legtöbb kincset kapjuk.

Kezdőpozíció: (1,1)

Végpozíció: (N,M)

Dinamikus programozás – kincs

Megoldás:

Számítsuk ki minden lehetséges mezőre, hogy addig eljutva mennyi a maximálisan begyűjthető kincsek száma!

$$Gy(i,j)=\begin{cases} -1 & \text{ha} \quad i=0 \quad \text{vagy} \quad j=0 \\ -N*M & \text{ha} \quad Kincs(i,j)="+" \\ \max(Gy(i-1,j),Gy(i,j-1)) & \text{ha} \quad Kincs(i,j)="" \\ \max(Gy(i-1,j),Gy(i,j-1))+1 & \text{ha} \quad Kincs(i,j)="*" \end{cases}$$

A cél: akadályból ne legyen értelme kijönni!

		*					
		*					
*		+	+	*			
*		+	*		+	+	*
		*			*	*	
			+				

Dinamikus programozás – kincs

```
Kincsek:
  Gy(0,2..M):=-1; Gy(2..N,0):=-1
  Gy(0,1):=0; Gy(1,0):=0
  Ciklus i=1-től N-ig
    Ciklus j=1-től M-ig
      Ha Kincs(i,j)="*" akkor k:=1 különben k:=0
      Ha Kincs(i,j)="+" akkor Gy(i,j):=-N*M
      különben ha Gy(i,j-1)>Gy(i-1,j)
                  akkor Gy(i,j):=Gy(i,j-1)+k
      különben Gy(i,j):=Gy(i-1,j)+k
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

A megoldás: Gy(N,M)

Dinamikus programozás – leghosszabb közös rész

Feladat:

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.

Megoldás:

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

A megoldás elemzése:

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:

```
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.
```

Azaz, amit már egyszer kiszámoltunk, azt tároljuk és ne számoljuk ki újra! A H mátrixot kezdetben -1 értékekkel töltjük ki!

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

Dinamikus programozás – toronyépítés

Feladat:

Adott $M_1, M_2, \ldots M_n$ 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.

Megoldás:

Tegyük fel, hogy $M_{i_1}; M_{i_2}; \ldots; M_{i_k} \left(i_1 < \ldots < i_k\right)$ megoldás!
Ekkor $n=i_{k-1}$-re $M_{i_1}; M_{i_2}; \ldots; M_{i_{k-1}} \left(i_1 < \ldots < i_{k-1}\right)$ 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:

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

Kérdés: Mi változik, ha a legtöbb kockából álló torony kockaszámát kérdeznénk?

Dinamikus programozás – toronyépítés

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

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

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. Látható azonban, hogy 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))-edik, alatta a Mire(...).

Dinamikus programozás – toronyépítés

Ugyanez ciklussal és veremmel:

```
Toronykiírás(i):
  Ciklus amíg Mire(i)>0
    Verembe(Mire(i))
    i:=Mire(i)
  Ciklus vége
  Ciklus amíg nem üres(V)
    Veremből(V,k); Ki: k
  Ciklus vége
Eljárás vége.
```

Dinamikus programozás – kemence

Feladat:

Egy fazekasműhelyben N tárgy vár kiégetésre. A kemencébe 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!

Megoldás:

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

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(0)=0)! Legyen Éget(i) az i. tárgy kiégetéséhez szükséges minimális idő!

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

Kell majd egy maximumkiválasztás a j. tárgytól az i. tárgyig!

Dinamikus programozás – kemence

```
Kemence:
  Idő(0):=0
  Ciklus i=1-től N-ig
    H:=Idő(i-1)+Éget(i); max:=Éget(i); ki:=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; ki:=j
      j:=j-1
    Ciklus vége
    Idő(i):=H; Db(i):=i-ki+1; Kivel(i):=ki
  Ciklus vége
Eljárás vége.
```

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

Dinamikus programozás – kemence

```
Kemence_megoldás:
   Kemence
   Kemence_eredmény(N)
Eljárás vége.
Kemence_eredmény(i):
   Ha i-Db(i)>0 akkor Kemence_eredmény(i-Db(i))
   Ki: Idő(i),i-Db(i)+1
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 $((1,\ldots,i_1),\ldots,(i_{m-1}+1,\ldots,N))$ a megoldás!

Ekkor az N. tárgy vagy önmagában kerül a kemencébe, vagy legfeljebb S súlyú előző tárggyal 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 még szummát 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
  Ciklus vége
Eljárás vége.
```

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

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 I.