Dinamikus programozás

(Dr. Horváth Gyula és Dr. Szlávi Péter előadásai felhasználásával

Dr. Zsakó László diáival)

Mohó stratégia

Feladat:

N-féle pénzjegyünk van, P_1, P_2, …, P_n címletű ($P_i<P_{i+1}$). Add meg, hogy minimálisan melyek felhasználásával fizethető ki az F összeg!

Feltehetjük, hogy minden pénzjegyből tetszőleges számú van.

Mohó stratégia

Feladat:

N-féle pénzjegyünk van, P_1, P_2, …, P_n címletű ($P_i<P_{i+1}$). Add meg, hogy minimálisan melyek felhasználásával fizethető ki az F összeg! Feltehetjük, hogy minden pénzjegyből tetszőleges számú van.

➢Megoldás:

Vegyünk a legnagyobb címletű pénzjegyből, amennyi szükséges, majd a maradék összeget fizessük ki a nála kisebb pénz-jegyekkel!

Mohó stratégia

```
Pénzváltás(N,P,F,Db):
  i:=N
  Ciklus amíg i>0 és F>0
    db(i):=F div P(i); F:=F mod P(i)
    i:=i-1
  Ciklus vége
  Ha i=0 akkor ...
Eljárás vége.
```


Mohó stratégia

```
Pénzváltás(N,P,F,Db):
  i:=N
  Ciklus amíg i>0 és F>0
    db(i):=F div P(i); F:=F mod P(i)
    i:=i-1
  Ciklus vége
  Ha i=0 akkor ...
Eljárás vége.
```

Probléma:

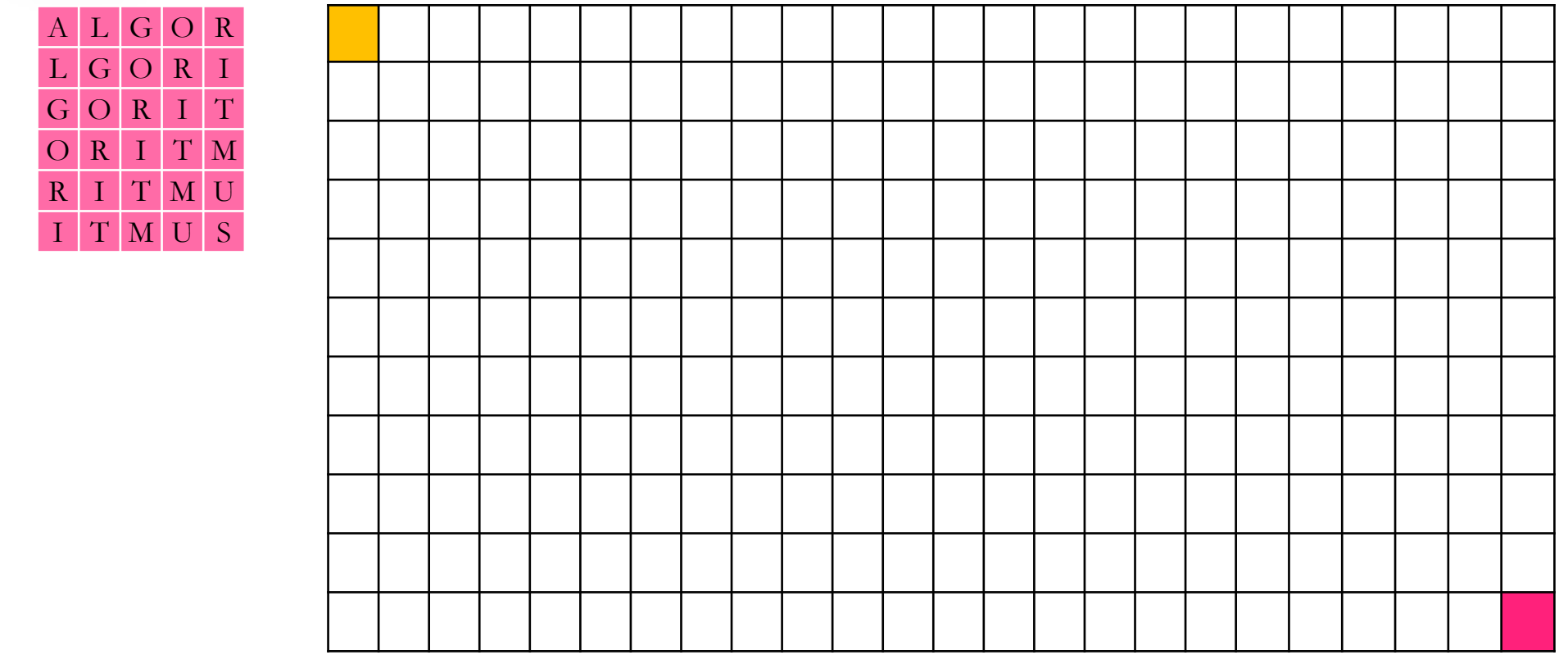
P=(1,3,4), F=6 esetén a megoldás (2,0,1), azaz 3 pénzjeggyel fizetnénk ki a 6 forintot,

pedig 6=3+3

→ Nagyon ritkán ad helyes eredményt, nem mohó stratégia!

Dinamikus programozás - számolás

A	L	G	O	R
L	G	O	R	I
G	O	R	I	T
O	R	I	T	M
R	I	T	M	U
I	T	M	U	S



Dinamikus programozás – kincs

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, a jármű csak jobbra és lefelé haladhat. Maximum hány kincset gyűjthet be?

		*					
		*					
*				*			
*							*
		*			*	*	

Kezdőpozíció: (1,1)

Végpozíció: (N,M)

Dinamikus programozás – kincs

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 az odáig begyűjthető legtöbb kincset kapjuk.

		*					
		*					
*				*			
*							*
		*			*	*	

Dinamikus programozás – kincs

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

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

		*					
		*					
*				*			
*							*
		*			*	*	

Dinamikus programozás – kincs

```
Kincsek:
  Gy(0,1..M):=-1; Gy(1..N,0):=-1; Gy(0,1):=0
  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)

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

Dinamikus programozás – kincs

Ha a bejárt útra is szükségünk van, akkor a Gy mátrix alapján az út visszakövethető. 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); Végére: "J"
     különben Útkiírás(i-1,j); Végére: "L"
Eljárás vége.
```

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

Dinamikus programozás – kincs2

Egy N*M-es téglalap alakú területen egy járművel szedhetjük össze az elrejtett kincseket. A jármű csak jobbra és lefelé haladhat.

Bizonyos helyeken akadályok vannak, ahova nem léphet!

Maximum hány kincset gyűjthet be?

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

Kezdőpozíció: (1,1)

Végpozíció: (N,M)

Dinamikus programozás – kincs2

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 az odáig begyűjthető legtöbb kincset kapjuk.

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

Kezdőpozíció: (1,1)

Végpozíció: (N,M)

Dinamikus programozás – kincs2

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

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

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

Dinamikus programozás – kincs

```
Kincsek:
  Gy(0,1..M):=-1; Gy(1..N,0):=-1; Gy(0,1):=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 – kincs

Egy N*M-es téglalap alakú területen egy járművel szedhetjük össze az elrejtett kincseket. A terep balról jobbra lejt, azaz a jármű csak jobbra, felfelé és lefelé haladhat. Maximum hány kincset gyűjthet be? Minden mezőre csak egyszer léphetünk!

		*					
		*					
*				*			
*							*
		*			*	*	

Kezdőpozíció: (1,1)

Végpozíció: (N,M)

Dinamikus programozás – kincs

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, de ide jöhetünk fentről és lentről is.

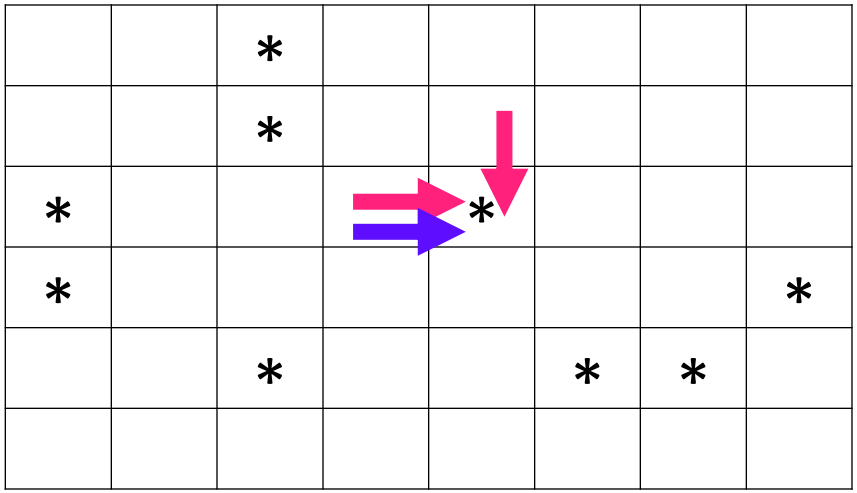
		*					
		*					
*				*			
*							*
		*			*	*	

Kezdőpozíció: (1,1)

Végpozíció: (N,M)

Dinamikus programozás – kincs

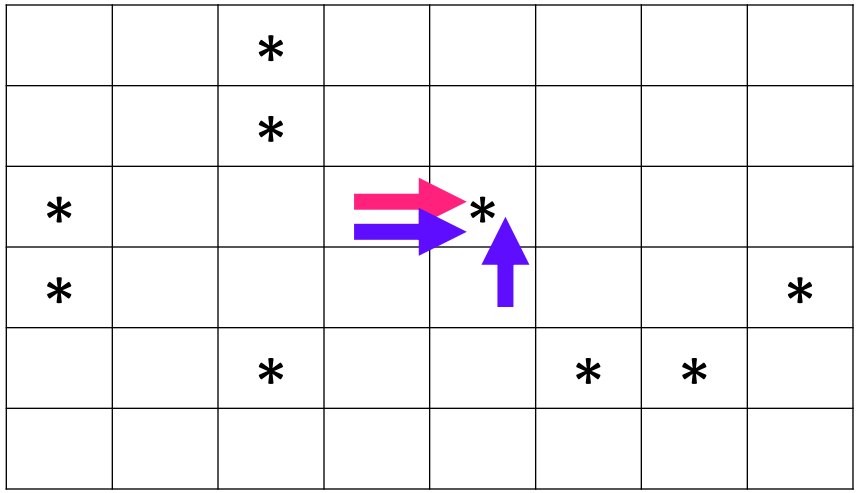
Fentről:



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

Dinamikus programozás – kincs

Lentről:



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

Dinamikus programozás – kincs

Számítsuk ki minden lehetséges mezőre, hogy addig eljutva mennyi a maximálisan begyűjthető kincsek száma, ha lentről, és ha fentről léptünk be!

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

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

Dinamikus programozás – kincs

```
Kincsek:
  L(N+1,1..M):=-1; L(1..N,0):=-1
  F(0,1..M):=-1; F(1..N,0):=-1
  Ciklus j=1-től M-ig
    Ciklus i=1-től N-ig
      Ha Kincs(i,j)="*" akkor k:=1 különben k:=0
      F(i,j):=max(F(i-1,j),L(i,j-1),F(i,j-1))+k
    Ciklus vége
    Ciklus i=N-től 1-ig -1-esével
      Ha Kincs(i,j)="*" akkor k:=1 különben k:=0
      L(i,j):=max(L(i+1,j),L(i,j-1),F(i,j-1))+k
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

A megoldás: F(N,M)

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 - pénzjegy

Adott $P_1, P_2, \ldots, P_n$ pénzjegyekkel kifizethető-e F forint?

$$F = P_{i_1} + P_{i_2} + \ldots + P_{i_m} \qquad (i_1 < \ldots < i_m)$$

Dinamikus programozás – pénzjegy

Adott $P_1, P_2, \ldots, P_n$ pénzjegyekkel kifizethető-e F forint?

Megoldás:

Tegyük fel, hogy !

Ekkor $$F = P_{i_1} + P_{i_2} + \ldots + P_{i_m} \left(i_1 < \ldots < i_m\right)$$

$$F - P_{i_m} = P_{i_1} + P_{i_2} + \ldots + P_{i_{m-1}} \left(i_1 < \ldots < i_m\right)$$

, azaz az adott részproblémának megoldása a végső részlete.

Számoljuk ki V(X,i) értékeket!

V(X,i) igaz, ha az X összeg kifizethető az első i pénzjeggyel.

Dinamikus programozás – pénzjegy

Adott $P_1, P_2, \ldots P_n$ pénzjegyekkel kifizethető-e F forint?

Három esetet vizsgálunk:

- az összeg megegyezik az i. pénzjeggyel;
- az i. pénzjegyet nem használjuk fel;
- az i. pénzjegyet felhasználjuk;

V(X, i) igaz, ha az X összeg kifizethető az első i pénzjeggyel.

Dinamikus programozás – pénzjegy

Adott $P_1, P_2, \ldots P_n$ pénzjegyekkel kifizethető-e F forint?

Három esetet vizsgálunk:

- az összeg megegyezik az i. pénzjeggyel;
- az i. pénzjegyet nem használjuk fel;
- az i. pénzjegyet felhasználjuk;

$$V(X,i) = igaz \quad \Leftrightarrow \quad \begin{cases} P_i = x \\ i > 1 \quad és \quad V(X, i-1) \\ i > 1 \quad és \quad X \geq P_i \quad és \quad V(X - P_i, i-1) \end{cases}$$

Dinamikus programozás

Feladat:

Adott $P_1, P_2, \ldots P_n$ pénzjegyekkel kifizethető-e F forint?

Megoldás:

$$V(X,i) = igaz \quad \Leftrightarrow \quad \begin{cases} P_i = x \\ i > 1 \quad és \quad V(X, i-1) \\ i > 1 \quad és \quad X \geq P_i \quad és \quad V(X - P_i, i-1) \end{cases}$$

```
V(X,i):
   V:=(X=P(i)) vagy
       ((i>1) és V(X,i-1)) vagy
       ((i>1) és (X≥P(i)) és V(X-P(i),i-1))
Függvény vége
```

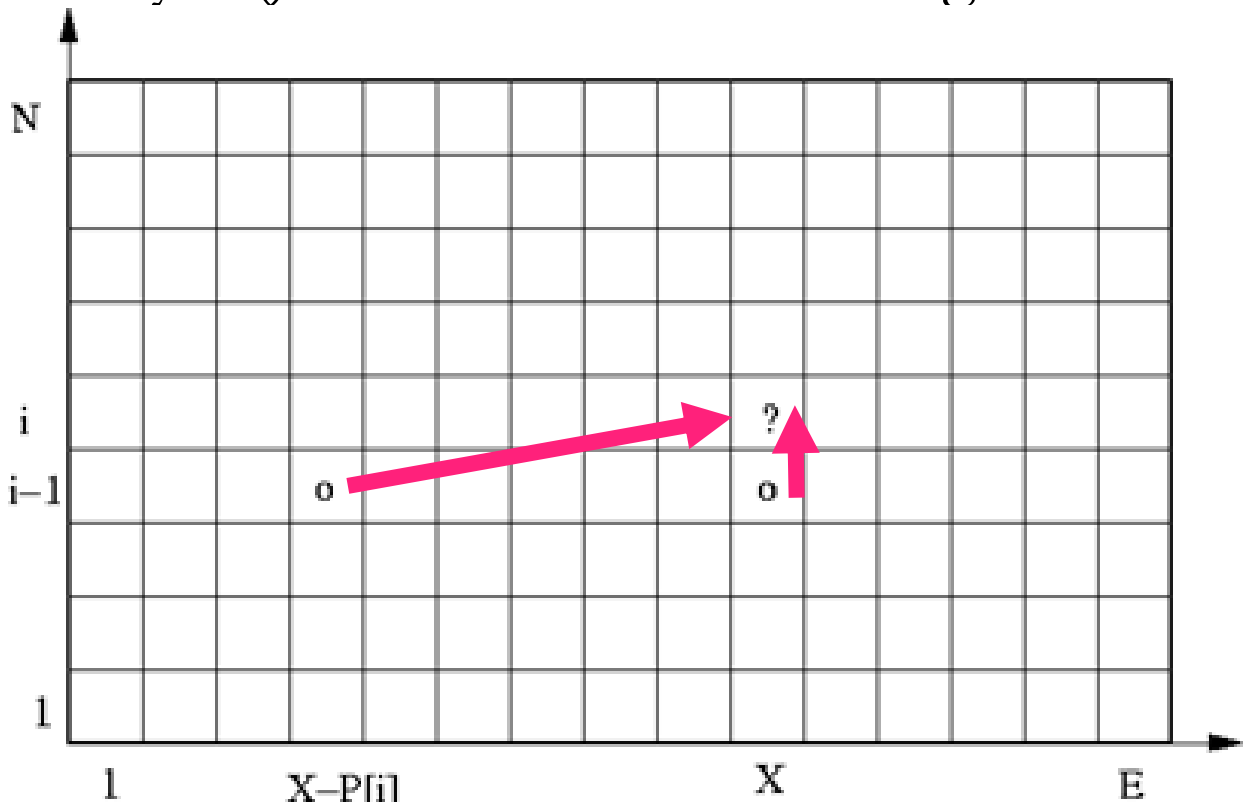
A megoldás: V(F,N) kiszámolása.

Probléma: a futási idő $O(2^N)$

Dinamikus programozás

V(X,i) kiszámításához mely V() értékekre lehet szükség?

Az ábra alapján V értékei alulról felfelé, azon belül balról jobbra számolhatók.



Dinamikus programozás

```
Pénzváltás(Lehet):
  V(1..F, 1):=hamis
  Ha P(1)≤F, akkor V(P(1),1):=igaz
  Ciklus i=2-től N-ig
    Ciklus X=1-től F-ig
      V(X,i):=(X=P(i)) vagy
              (V(X,i-1)) vagy
              ((X≥P(i)) és (V(X-P(i),i-1))
    Ciklus vége
  Ciklus vége
  Lehet:=V(F,N)
Függvény vége.
```

A futási idő O(N*F)

Memóriaigény: N*F

Dinamikus programozás – pénzcímlet

Feladat:

Adott P_1,P_2, …P_n pénzcímletek közül melyekkel fizethető ki F forint?

Dinamikus programozás – pénzcímlet

Feladat:

Adott $P_1, P_2, \ldots P_n$ pénzcímletek közül melyekkel fizethető ki F forint?

Megoldás:

Legyen V(X,i) a legutolsó olyan k index, amelyre igaz, hogy P_k előfordul X felváltásában!

Legyen V(X,i)=N+1, ha X nem váltható fel az első i pénzjeggyel!

Dinamikus programozás – pénzcímlet

Feladat:

Adott $P_1, P_2, \ldots P_n$ pénzcímletek közül melyekkel fizethető ki F forint?

Megoldás:

$$V(X,i) = \begin{cases} N+1 & ha \quad i = 0 \quad és \quad X > 0 \\ 0 & ha \quad X = 0 \\ V(X,i-1) & ha \quad V(X,i-1) \le N \\ i & ha \quad X \ge P_i \quad és \quad V(X-P_i,i-1) \le N \\ N+1 & egyébként \end{cases}$$

Dinamikus programozás – pénzcímlet

Pénzváltás: A V mátrix kiszámolása.

```
V(1..F,0):=N+1; V(0,0):=0
Ciklus i=1-től N-ig
  V(0,i):=0
  Ciklus X=1-től F-ig
   Ha V(X,i-1)≤N
    akkor V(X,i):=V(X,i-1)
    különben
      ha X≥P(i) és V(X-P(i),i-1)≤N
         akkor V(X,i):=i
         különben V(X,i):=N+1
  Ciklus vége
Ciklus vége
```

$$V(X,i)=\begin{cases} N+1 & ha & i=0 & és & X>0 \\ 0 & ha & X=0 & & \\ V(X,i-1) & ha & V(X,i-1)\le N & & \\ i & ha & X\ge P_i & és & V(X-P_i,i-1)\le N \\ N+1 & egyébként & & & \end{cases}$$

Dinamikus programozás – pénzcímlet

Csak akkor van megoldás, ha $V(F,N) \leq N$.

Ekkor $k=V(F,N)$ olyan pénz indexe, hogy $F-P(k)$ felváltható az első $k-1$ pénzzel.

Tehát a $V(F-P(k),k-1)$ probléma megoldásával kell folytatnunk, mindaddig, amíg a maradék pénz 0 nem lesz.

Dinamikus programozás

```
…
Db:=0; X:=F; i:=N
Ha V(F,N)≤N
    akkor Ciklus
            i:=V(x,i)
            Db:=Db+1; A(Db):=i
            X:=X-P(i) i:=i-1
          amíg X≠0
          Ciklus vége
Eljárás vége.
```

A megoldás előállítása.

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 ZZ_{k-1}
az XX_{n-1} és
YY_{m-1} leghosszabb közös részsorozata.

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.

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:

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

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.
```

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

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.
```

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

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

Dinamikus programozás – toronyépítés

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.

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

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:

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

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))-edik, alatta a Mire(...).

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!

Dinamikus programozás – kemence

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!

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

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):=ki
  Ciklus vége
Eljárás vége.
```

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

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

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

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 kiéget-hető!

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)

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

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; Kivel(i):=ki
  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)!

Megoldás:

Tetszőleges S szóra az SS^T szó biztos tükörszó, tehát a feladatnak kell legyen 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:

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

Három eset van:

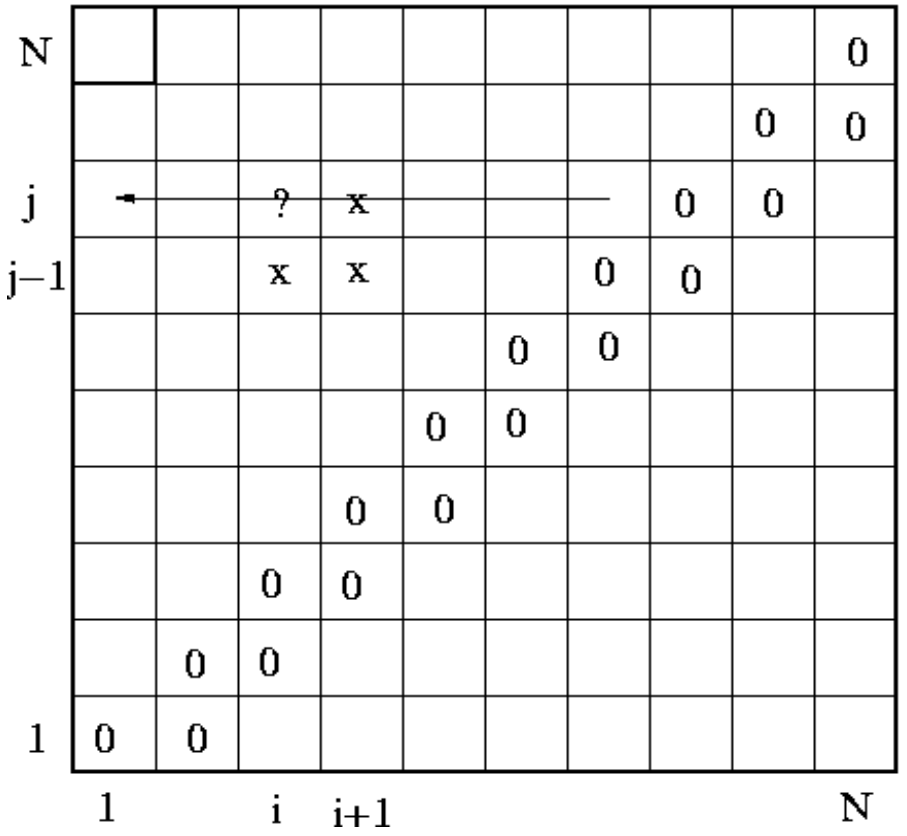
- egybetűs részek
- a két szélső karakter egyforma
- az első és az utolsó karakter különbözik

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

Probléma: a rekurzió nem hatékony!

Dinamikus programozás – Tükörszó

Jó sorrendű táblázatkitöltés kell:



Dinamikus programozás – Tükörszó

```
Tükörszó(S,M):
  M() kezdő feltöltése
  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.
```

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.
```

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:

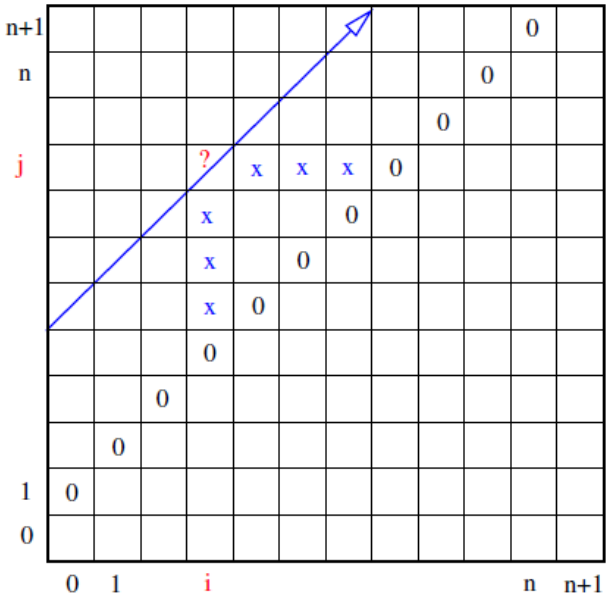
Ha az optimális vágássorozatban először a K. helyen történik a vágás, akkor a $V_0..V_k$ és a $V_k..V_{n+1}$ rúd vágássorozata is optimális lesz.

Dinamikus programozás – rúd darabolás

Megoldás:

Számítsuk ki minden (i,j) rúddarabra, hogy mi 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.

Dinamikus programozás

Feladat:

Adott P_1,P_2, …, P_n pénzjegyek közül a lehető legkevesebbet használva mennyivel fizethető ki F forint?

Megoldás:

Tegyük fel, hogy $F = P_{i_1} + P_{i_2} + \ldots + P_{i_m} \left(i_1 < \ldots < i_m\right)$ optimális felbontás!

Ekkor $F - P_{i_m} = P_{i_1} + P_{i_2} + \ldots + P_{i_{m-1}} \left(i_1 < \ldots < i_m\right)$ is optimális, azaz az adott részproblémának megoldása a megoldás megfelelő részlete.

Dinamikus programozás

Feladat:

Adott $P_1, P_2, \ldots P_n$ pénzjegyek közül a lehető legkevesebbet használva mennyivel fizethető ki F forint?

Megoldás:

$$O(X,i) = \begin{cases} N+1 & ha \quad i=0 \quad és \quad X>0 \\ 0 & ha \quad X=0 \\ O(X,i-1) & ha \quad i>0 \quad és \quad X<P_i \\ \min(O(X,i-1), 1+O(X-P_i,i-1)) & ha \quad i>0 \quad és \quad X \geq P_i \end{cases}$$

Dinamikus programozás

```
Pénzváltás(Minimum):
  O(1..F,0):=N+1; O(0,0):=0
  Ciklus i=1-től N-ig
    O(0,i):=0
    Ciklus X=1-től F-ig
      Ha X<P(i) akkor O(X,i):=O(X,i-1)
      különben O(X,i):=min(O(X,i-1),
                           1+O(X-P(i),i-1))
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

$$O(X,i)=\begin{cases} N+1 & ha \quad i=0 \quad és \quad X>0 \\ 0 & ha \quad X=0 \\ O(X,i-1) & ha \quad i>0 \quad és \quad X<P_i \\ \min(O(X,i-1),1+O(X-P_i,i-1)) & ha \quad i>0 \quad és \quad X\geq P_i \end{cases}$$

Dinamikus programozás

Feladat:

Adott P_1,P_2, …P_n pénzjegyek közül a lehető legkevesebbet használva melyekkel fizethető ki F forint?

$$F = P_{i_1} + P_{i_2} + \ldots + P_{i_m} \left(i_1 < \ldots < i_m\right)$$

Dinamikus programozás

Feladat:

Adott P_1,P_2, …P_n pénzjegyek közül a lehető legkevesebbet használva melyekkel fizethető ki F forint?

Megoldás:

Tegyük fel, hogy $F = P_{i_1} + P_{i_2} + \ldots + P_{i_m} \left(i_1 < \ldots < i_m\right)$ optimális felbontás!

Ekkor $F - P_{i_m} = P_{i_1} + P_{i_2} + \ldots + P_{i_{m-1}} \left(i_1 < \ldots < i_m\right)$ is optimális, azaz az adott részproblémának megoldása a megoldás megfelelő részlete.

Dinamikus programozás

Feladat:

Adott $P_1, P_2, \ldots P_n$ pénzjegyek közül a lehető legkevesebbet használva melyekkel fizethető ki F forint?

Megoldás:

$$V(X,i) = \begin{cases} N+1 & \textit{ha} \quad i = 0 \quad \textit{és} \quad X > 0 \\ 0 & \textit{ha} \quad X = 0 \\ i & \textit{ha} \quad X \geq P_i \quad \textit{és} \quad O(X,i) > 1 + O(X - P_i, i-1) \\ V(X, i-1) & \textit{egyébként} \end{cases}$$

Dinamikus programozás

```
Pénzváltás(O,V):
  O(1..F):=N+1; O(0):=0; V(1..F,0):=N+1
  Ciklus i=1-től N-ig
    O(0):=0; V(0,i):=0
    Ciklus X=F-től 1-ig -1-esével
      Ha X≥P(i) akkor S:=O(X-P(i))+1
                különben S:=N+1
      Ha S<O(X) akkor O(X):=S; V(X,i):=i
                különben V(X,i):=V(X,i-1)
    Ciklus vége
  Ciklus vége
Eljárás vége.
```

$$V(X,i)=\begin{cases} N+1 & ha \quad i=0 \quad és \quad X>0 \\ 0 & ha \quad X=0 \\ i & ha \quad X \geq P_i \quad és \quad O(X,i)>1+O(X-P_i,i-1) \\ V(X,i-1) & egyébként \end{cases}$$

Dinamikus programozás

Feladat:

Két testvér adott értékű ajándékokon osztozkodik. Minden egyes ajándékot pontosan ez egyik testvérnek kell adni. Készíts testvéries osztozkodást, azaz olyat, amelyben a két testvér által kapott ajándékok értéke különbségének abszolút értéke minimális!

Megoldás:

Ha az ajándékok összértéke E, akkor a feladat megoldása a legnagyobb olyan $A \leq E/2$ szám, amely érték az ajándékok értékeiből kétféleképpen is előállítható.

Tehát a megoldás visszavezethető a pénzváltás probléma megoldására.

Dinamikus programozás

Megoldás:

$$V(X,i) = igaz \quad \Leftrightarrow \quad \begin{cases} P_i = x \\ i > 1 \quad és \quad V(X, i-1) \\ i > 1 \quad és \quad X \geq P_i \quad és \quad V(X - P_i, i-1) \end{cases}$$

```
V(X,i):
   V:=X=P(i) vagy
      i>1 és V(X,i-1) vagy
      i>1 és X≥P(i) és V(X-P(i),i-1)
Függvény vége.
```

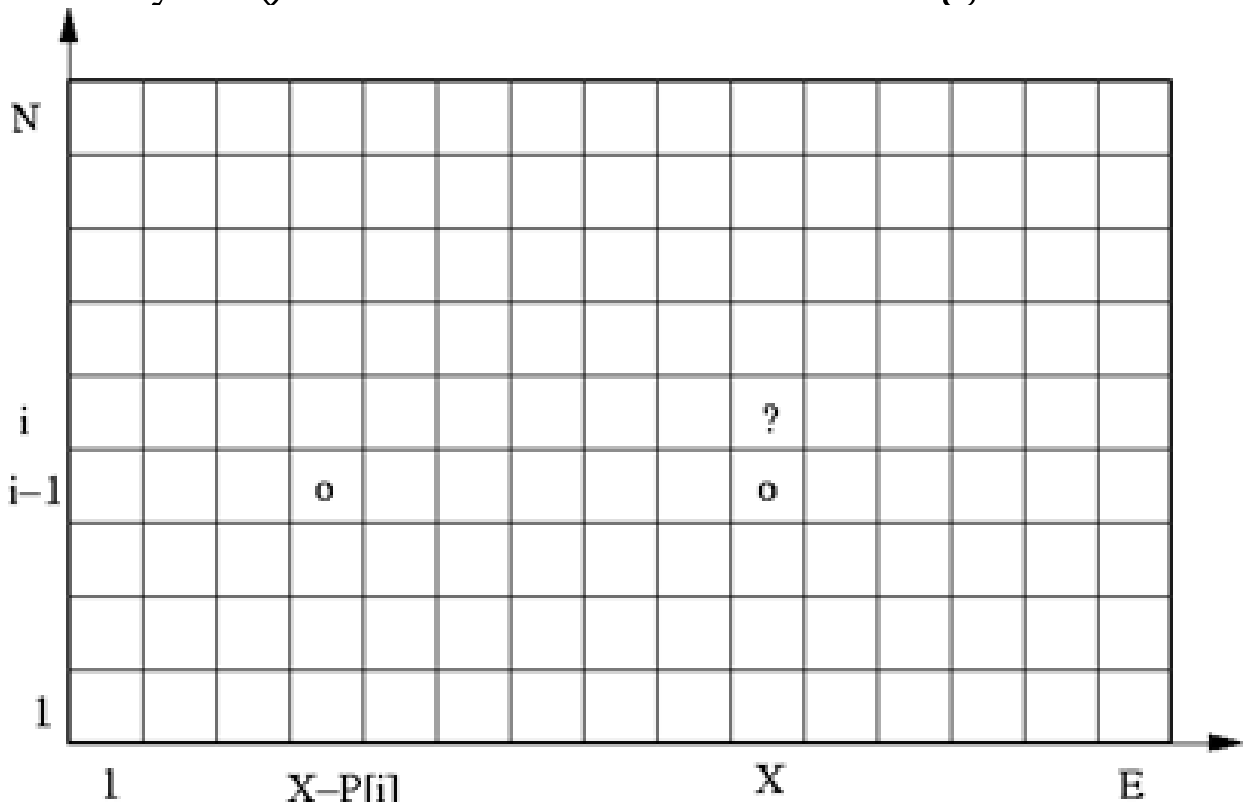
A megoldás: a legnagyobb olyan A kiválasztása (1 és E/2 között), amelyre V(A,N) igaz.

Probléma: a futási idő $O(2^N)$

Dinamikus programozás

V(X,i) kiszámításához mely V() értékekre lehet szükség?

Az ábra alapján V értékei alulról felfelé, azon belül balról jobbra számolhatók.



Dinamikus programozás

```
Testvéries osztozkodás(A):
  V(1..E,1):=hamis
  Ha P(1)≤E akkor V(P(1),1):=igaz
  Ciklus i=2-től N-ig
    Ciklus X=1-től E-ig
      V(X,i):=X=P(i) vagy
              V(X,i-1) vagy
              X≥P(i) és V(X-P(i),i-1)
    Ciklus vége
  Ciklus vége
  Eredmény megadása(A,V,E,N)
Függvény vége.
```

A futási idő O(N*F)

Memóriaigény: N*F

Dinamikus programozás

```
Eredmény megadása(A,V,E,N):
  A:=E/2
  Ciklus amíg nem V(A,N)
    A:=A-1
  Ciklus vége
Függvény vége.
```

Dinamikus programozás

Feladat:

Két testvér adott értékű ajándékokon osztozkodik. Minden egyes ajándékot pontosan ez egyik testvérnek kell adni. Készíts igazságos osztozkodást, azaz olyat, amelyben a két testvér által kapott ajándékok értéke egyforma, a maradék értéke pedig minimális!

Megoldás:

Ha az ajándékok összértéke E, mindkét testvér A értékű ajándékot kap. A feladat megoldása a legnagyobb olyan $A \leq E/2$ szám, amely érték az ajándékok értékeiből kétféleképpen előállítható.

Tehát a megoldás visszavezethető a pénzváltás probléma megoldására.

Dinamikus programozás

Megoldás:

Legyen V(X,Y,i) igaz, ha az első i ajándékból X és Y értékű is előállítható úgy, hogy mindegyik szám legfeljebb az egyik összegben szerepel! Legyen V(0,0,i)=igaz minden i-re, V(X,Y,0) hamis minden X,Y-ra!

V(X,Y,i)-re a következő rekurzív összefüggés írható fel i>0 esetén:

$$V(X,Y,i)=\text{igaz} \quad \Leftrightarrow \quad \begin{cases} V(X,Y,i-1) \\ P_i \le X \quad \text{és} \quad V(X-P_i,Y,i-1) \\ P_i \le Y \quad \text{és} \quad V(X,Y-P_i,i-1) \end{cases}$$

Dinamikus programozás

Megoldás:

Legyen V(X,Y,i) igaz, ha az első i ajándékból X és Y érték is előállítható úgy, hogy mindegyik legfeljebb az egyik összegben szerepel! Legyen V(0,0,i)=igaz minden i-re, V(X,Y,0) hamis minden X,Y-ra!

```
V(X,Y,i):
   Ha i=0 akkor V:=hamis
   különben ha X=0 és Y=0 akkor V:=igaz
   különben V:=V(X,Y,i-1) vagy
                P(i)≤X és V(X-P(i),Y,i-1) vagy
                P(i)≤Y és V(X,Y-P(i),i-1)
Függvény vége.
```

A megoldás: a legnagyobb olyan A kiválasztása (1 és E/2 között), amelyre V(A,A,N) igaz.

Dinamikus programozás

```
Igazságos osztozkodás(A):
  V(1..E,1..E,1):=hamis; V(0,0,1):=igaz
  Ha P(1)≤E akkor V(P(1),0,1):=igaz
                  V(0,P(1),1):=igaz
  Ciklus i=2-től N-ig
    Ciklus X=1-től E-ig
      Ciklus Y=1-től E-ig
        V(X,Y,i):=V(X,Y,i-1) vagy
                  P(i)≤X és V(X-P(i),Y,i-1) vagy
                  P(i)≤Y és V(X,Y-P(i),i-1)
    Ciklus vége
  Ciklus vége
  Eredmény megadása(A,V,E,N)
Függvény vége.
```

Dinamikus programozás

```
Eredmény megadása(A,V,E,N):
  A:=E/2
  Ciklus amíg nem V(A,A,N)
    A:=A-1
  Ciklus vége
Függvény vége.
```

Dinamikus programozás – Hátizsák feladat

Feladat:

Adott N tárgy (értékük F_i, súlyuk S_i). Egy hátizsákba maximum H súlyú tárgyat pakolhatunk. Mennyi a maximális elszállítható érték?

Megoldás:

Tegyük fel, hogy $H \geq S_{i_1} + S_{i_2} + \ldots + S_{i_m} \left(i_1 < \ldots < i_m\right)$ optimális megoldás, azaz $F_{i_1} + F_{i_2} + \ldots + F_{i_m}$ maximális!

Ekkor $H - S_{i_m} \geq S_{i_1} + S_{i_2} + \ldots + S_{i_{m-1}} \left(i_1 < \ldots < i_m\right)$ is optimális, azaz az adott részproblémának megoldása a megoldás megfelelő részlete.

Dinamikus programozás – Hátizsák feladat

Feladat:

Adott N tárgy (értékük F_i, súlyuk S_i). Egy hátizsákba maximum H súlyú tárgyat pakolhatunk. Mennyi a maximális elszállítható érték?

Megoldás:

Jelölje E(X,i) a legnagyobb elszállítható értéket, amely az első i tárgyból egy X kapacitású hátizsákba bepakolható.

$$E(X,i) = \begin{cases} F_1 & \text{ha } i = 1 \text{ és } S_1 \le X \\ 0 & \text{ha } i = 1 \text{ és } S_1 > X \\ E(X, i-1) & \text{ha } i > 1 \text{ és } S_i > X \\ \max\left(E(X,i-1), F_i + E(X - S_i, i-1)\right) & \text{egyébként} \end{cases}$$

Dinamikus programozás – Hátizsák feladat

```
Hátizsák():
  E(0..S(1)-1,1):=0; E(S(1)..H,1):=F(1)
  Ciklus i=2-től N-ig
    Ciklus X=0-tól H-ig
      E(X,i):=E(X,i-1)
      Ha S(i)≤X és E(X,i)<E(X-S(i),i-1)+F(i)
         akkor E(X,i):=E(X-S(i),i-1)+F(i)
    Ciklus vége
  Ciklus vége
  Kiírás
Eljárás vége.
```

$$E(X,i)=\begin{cases} F_1 & \text{ha } i=1 \text{ és } S_1 \le X \\ 0 & \text{ha } i=1 \text{ és } S_1 > X \\ E(X,i-1) & \text{ha } i>1 \text{ és } S_i > X \\ \max\left(E(X,i-1), F_i + E(X-S_i,i-1)\right) & \text{egyébként} \end{cases}$$

Dinamikus programozás – Hátizsák feladat

```
Kiírás:
  i:=N; X:=H
  Ciklus amíg E(X,i)>0
    Ciklus amíg i≥1 és E(X,i)=E(X,i-1)
      i:=i-1
    Ciklus vége
    Ki: i, X:=X-S(i); i:=i-1
  Ciklus vége
Eljárás vége.
```

Ha E(X,i)=E(X,i-1), akkor az i. tárgy nem kell a hátizsákba!

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 további feladatok

DNS

A biológiai szekvenciák, különösen a DNS szekvenciák vizsgálata nagyon fontos kutatási terület. Minden DNS szekvencia leírható olyan karaktersorozattal, amely csak az A, C, G és T karaktereket tartalmazhatja.

Számítsd ki egy DNS szekvenciának egy pontosan K betűből álló részsorozatát, amely a legtöbbször fordul elő a vizsgált szekvenciában!

Dinamikus programozás
további feladatok

LEFED

Azt mondjuk, hogy egész számok zárt intervallumainak egy H halmaza lefedi az [1,N] intervallumot, ha az intervallum minden x eleméhez van olyan intervallum H-ban, amelynek x eleme. Egy lefedés költségén a lefedéshez használt intervallumok hosszainak összegét értjük.

Számítsd ki, hogy adott [1,N] lefedendő intervallum és lefedéshez használható intervallumok egy H halmaza esetén mekkora a minimális lefedés költsége, ha létezik lefedés!

Dinamikus programozás
további feladatok

KÉPÁTLÓ

Adott egy N x N pixelből álló fekete-fehér kép. Szeretnénk a képen a bal felső saroktól a jobb alsó sarokig egy jobbra-lefele haladó határvonalat húzni úgy, hogy a vonaltól jobbra-felfele eső fekete, valamint a vonaltól balra-lefele eső fehér pixelek számának K összege a lehető legkevesebb legyen. A határvonalra eső pixelek nem számítanak bele.

Add meg a minimális ilyen K értéket!

Dinamikus programozás
további feladatok

CSOMAG

A csomagküldő szolgálat központjában a beérkezés sorrendjében várakoznak a csomagok továbbításra. Minden csomagnak ismert a súlya. A cégnek két kamionja van, mindegyik azonos K kapacitású. Minden csomag súlya legfeljebb K. A lehető legtöbb csomagot akarják továbbítani a két kamionnal.

Kiszámítandó az a legnagyobb M szám ($1 \leq M \leq N$), hogy a sorban első M csomag mindegyike felpakolható a két kamion valamelyikére!

Dinamikus programozás
további feladatok

PAKOLÁS

Egy raktárból konténereket kell elszállítani K kamionnal. A konténerek egy sorban egymás után helyezkednek el. Minden konténer súlyát ismerjük. Minden kamionra csak a sorban egymást követő konténerek pakolhatók. Azt szeretnék elérni, hogy a lehető legegyenletesebb legyen a kamionok terhelése, ami azt jelenti, hogy a maximálisan terhelt kamion terhelése a lehető legkisebb legyen. A kamionok súlykapacitása legalább akkora, hogy mindegyik biztosan elbírja a rárakandó konténerek súlyát. Minden kamionra legalább egy konténert kell rakni.

Számíts ki egy legegyenletesebb pakolást!

Dinamikus programozás előadás vége