

DINAMIKUS PROGRAMOZÁS

(ESETTANULMÁNY)

1. FELADAT:

Mohó Marci malacperselyben gyűjti a pénzét. Csak fém pénzeket rakott a perselybe, de nem jegyezte föl, hogy milyeneket. Felírta azonban az üres persely súlyát, így meg tudja állapítani a perselyben levő pénzek összsúlyát. Ismeri továbbá az egyes pénzürmék egyedi súlyát és értékét. Szeretné kiszámítani, hogy mennyi az a legkisebb összeg, amelyet a perselye biztosan tartalmaz. Egy adott típusú pénzürméből több is lehet a perselyben.

Írjon programot a perselybeli minimális pénzösszeg meghatározására.

Példa:

MALAC . BE		MALAC . KI	
15	– a persely súlya	8	– a minimális összeg
4	– érmefélék száma	1 1 1 1 1 1 2	– az érmék értéke
1 2	– 1. érmeféle értéke , súlya		
2 3	– 2. érmeféle értéke , súlya		
5 6	– 3. érmeféle értéke , súlya		
10 4	– 4. érmeféle értéke , súlya		

1. FELADAT – 1. MEGOLDÁS

Keressük a $E(X) = \min \left\{ \sum_i \text{érme}_i . \text{érték} \mid \sum_i \text{érme}_i . \text{súly} = X \right\}$,

azaz az X súly „érmekre-bontásának” (érme-súlyösszeg=X) értékminimumát.

E rekurzívan kiszámítható az alábbiak szerint (N az érmefélék száma):

$E(x) := 0$, ha $x=0$

$E(x) := \min_{i=1}^N (E(x - \text{érme}_i . \text{súly}) + \text{érme}_i . \text{érték} \mid \text{érme}_i . \text{súly} \leq x)$, egyébként.

Demó. (A nyomkövető kiírás magyarázata: „X, érme_i.érték(érme_i.súly)”. Látható a nagyszámú, „fölsleges” rekurzív hívás.)

```

Program MohoMarciperselye;
{
  1. megoldás: rekurzív
  Keressük a Min( Szumma(i=1..N) Erme(i).ertek |
                  Szumma(i=1..N) Erme(i).suly=X )
  Rekurzív összefüggés a keresett értékre:
  E(x):=0,      ha x=0
  E(x):=Min(i=1..N) (E(x-Erme(i).suly)+Erme(i).ertek |
                  Erme(i).suly<=x), egyébként
}
Uses
  Newdelay,Crt;
Const
  MaxErmeDb=10;
  MaxPenz=MaxInt;
  MaxX=100;
Type
  TErme=Record suly,ertek:Word End;
  TErmek=Array [1..MaxErmeDb] of TErme;
  TMinik=Array [0..MaxX] of Integer;
Var
  N:Word;
  erme:TErmek;
  X:Word;
  min,i:Word;
  valaszt:TMinik; {X-hez hozzáválasztott érme sorszáma}

  {$i AltRutin.inc}

Function E(X:Word):Word;
  {X súlyú pénzérték legkisebb értéke}
  Var
    i,m:Word;
Begin
  If
    x=0 then
      Begin
        E:=0
      End
    Else
      Begin
        m:=MaxPenz;
        For i:=1 to N do
          Begin
            If erme[i].suly<=X then
              If E(X-erme[i].suly)+erme[i].ertek<m then
                Begin
                  m:=E(X-erme[i].suly)+erme[i].ertek;
                  {csak a rekurzív próbálkozások követéséhez:}
                  Write(X:4,': ',erme[i].suly:3,'(',
                        erme[i].ertek:3,') ');
                  valaszt[X]:=i {az X-hez jónak talált érme
                                sorszáma, az „X-erme[i].suly”
                                folytatásaként}
                End;
          End;
        End;
      End;

```

```

        E:=m
    End
End;

Begin
    UjLap('Mohó Marci perselye',0);
    {Ef: X az tényleges érmefelbontás legyen, azaz  $\exists$  megoldás}
    {beolvasás helyett, csak most az egyszer;}
    N:=4;
    erme[1].suly:=2; erme[1].ertek:=1;
    erme[2].suly:=3; erme[2].ertek:=2;
    erme[3].suly:=6; erme[3].ertek:=5;
    erme[4].suly:=4; erme[4].ertek:=10;
    X:=15;

    valaszt[0]:=0;
    For i:=1 to MaxX do
    Begin
        valaszt[i]:=0;
    End;

    min:=E(X);

    Write(CrLf,min,' ');
    While X>0 do
    Begin
        i:=valaszt[X];
        Write(erm[i].ertek:3);
        X:=X-erm[i].suly;
    End;
    BillreVar;
End.

```

1. FELADAT – 2. MEGOLDÁS

Keressük a $E(X)=\text{Min}\left\{\sum_i \text{érme}_i.\text{érték} \mid \sum_i \text{érme}_i.\text{súly} = X\right\}$.

E rekurzívan kiszámítható az alábbiak szerint is: **építsünk a már kiszámolt minimális részösszegekre. Ezeket tároljuk az ETömb-ben, amelyet kezdetben -1-ekkel töltünk föl.**

$E(x):=0$, ha $x=0$

$E(x):=ETömb_x$, ha $ETömb_x \geq 0$

$(ETömb_x)=E(x):=\underset{i=1}{\overset{N}{Min}} (E(x-\text{érme}_i.\text{súly})+\text{érme}_i.\text{érték} \mid \text{érme}_i.\text{súly} \leq x)$, egyébként.

[Demó.](#) (Drasztikus rekurzív hívás csökkenés.)

```

Program MohoMarciPerselye;
{
  2. megoldás: fél-rekurzív
  Keressük a Min( Szumma(i=1..N) Erme(i).ertek |
                    Szumma(i=1..N) Ereme(i).suly=X )
  A számításnál már fölhasználjuk a korábban kiszámolt
  részösszeg-minimumot, amit az ETömb-ben gyűjtjük.
  E(x):=0,          ha x=0
  E(x):=ETömb(x),  ha ETömb(x)>=0
  E(x):=Min(i=1..N) (E(x-Erme(i).suly)+Erme(i).ertek |
                    Erme(i).suly<=x), egyébként
}
Uses
  Newdelay,Crt;
Const
  MaxErmeDb=10;
  MaxX=100;
  MaxPenz=MaxInt;
Type
  TErme=Record suLy,ertek:Word End;
  TErmek=Array [1..MaxErmeDb] of TErme;
  TMinik=Array [0..MaxX] of Integer;
Var
  N:Word;
  erme:TErmek;
  X,min:Word;
  i:Word;
  ETomb:TMinik;
  valaszt:TMinik; {X-hez hozzáválasztott érme sorszáma}

  {$i AltRutin.inc}

Function Mini(X:Word):Word;
  {X súlyú pénzérték legkisebb értéke}
  Var
    i,m,Mx:Word;
Begin
  If ETomb[X]>=0 then
    Begin
      Mini:=ETOMB[X]
    End
  Else

```

```

Begin
  m:=MaxPenz;
  For i:=1 to N do
    Begin
      If erme[i].suly<=X then
        Begin
          Mx:=Mini(X-erme[i].suly);
          If Mx+erme[i].ertek<m then
            Begin
              m:=Mx+erme[i].ertek;
              {csak a rekurzív próbálkozások követéséhez:}
              Write(X:4,':',erme[i].suly:3,'(',
                  erme[i].ertek:3,')');
              valaszt[X]:=i; {az X-hez jónak talált érme
                           sorszáma, az „X-erme[i].suly”
                           folytatásaként}

            End;
          End;
        End;
      ETomb[X]:=m;
      Mini:=m
    End;
  End;

Begin
  UjLap('Mohó Marci perselye',0);
  {Ef: X az tényleges ermefelbontás legyen}
  {beolvasás helyett, csak most az egyszer:}
  N:=4;
  erme[1].suly:=2; erme[1].ertek:=1;
  erme[2].suly:=3; erme[2].ertek:=2;
  erme[3].suly:=6; erme[3].ertek:=5;
  erme[4].suly:=4; erme[4].ertek:=10;
  X:=15;

  ETomb[0]:=0;      {ETomb[X]:=0, ha X=0}

  valaszt[0]:=0;
  For i:=1 to MaxX do
    Begin
      ETomb[i]:=-1; valaszt[i]:=0;
    End;

  min:=Mini(X);

  Write(CrLf,min,' ');
  While X>0 do
    Begin
      i:=valaszt[X];
      Write(erme[i].ertek:3);
      X:=X-erme[i].suly;
    End;
  BillreVar;
End.

```

2. FELADAT:

Egy szót tükörszónak nevezünk, ha balról és jobbról kiolvasva betűről betűre megegyezik. (Tehát minden egybetűs szó tükörszó.) Minden szó felbontható részekre úgy, hogy minden rész tükörszó legyen. Minimálisnak nevezzük az olyan felbontást, amely egy szót a lehető legkevesebb tükörszóra szed szét.

Írjon programot a minimális tükörszó partíciók számának meghatározására.

Példa:

TUKOR.BE		TUKOR.KI
abbakabadara –a felbontandó szó	5	– minimális felbontásszám (abbakabadara)

2. FELADAT – 1. MEGOLDÁS

Keressük az $R(\text{Hossz}(S))$ értékét, ahol

$$R(i) = \underset{n}{\text{Min}} \{ n \mid S(1..i) = S(1..i_1) + S(i_1+1..i_2) \dots + S(i_{n-1}+1..i) \}$$

$$\wedge T(1..i_1) \wedge T(i_1+1..i_2) \wedge \dots \wedge T(i_{n-1}+1..i) \},$$

azaz az $S(1..i)$ legkevesebb tükörszóra bontását.

Jelölések:

- $S(i..j) := S$ szó i - j . betűi,
- „+” a *konkatenáció* műveleti jele,
- $T(i..j) := S(i..j)$ tükörszó-e.

R rekurzívan kiszámítható az alábbiak szerint:

$R(i) := i$, ha $i=0$ vagy $i=1$

$R(i) := \text{Min} (R(i-1)+1, \underset{j=i-1}{\overset{1}{\text{Min}}} R(j-1)+1 \mid T(j..i))$, egyébként.

(Magyarázat az R -def rekurzív ágához: az $S(1..i-1)$ -hez az $S(i)$ betűt hozzáveszem **1-betűs tükörszóként**, vagy $S(1..i-2)$ -höz hozzávéve **2-**, $S(1..i-3)$ -hez **3-**, ... $S(1..i-i)$ -hez **i -betűs tükörszót** alkot...)

[Demó.](#)

```

Program MinimalisTukorSzoraBontas;
{
  1. megoldás: rekurzív
  Keressük a  $R(i) = \text{Min}(n \mid S(1..i) = S(1..i1) + \dots + S(..in) \text{ ES } in=i$ 
    ES  $T(1.i1) \text{ ES } \dots \text{ ES } T(..in)$  ),
  azaz az  $S(1..i)$  szó minimális tükörszó-felbontása,
  ahol
  *  $S(j..i)$  az  $S$  szó  $j-i$ . betüi,
  * a "+" a konkatenáció műveleti jele,
  *  $T(i..j) := S(i..j)$  tükörszó-e.
   $R(x) := 0$ , ha  $x=0$ , vagy 1
   $R(x) := \text{Min}(R(i-1)+1$  ,
     $\text{Min}(j=i-1..1) (R(j-1)+1 \mid T(j..i))$  ), egyébként.
}

Uses
  Newdelay, Crt;

Const
  MaxHossz=100;

Type
  TMinFelbont=Array [0..MaxHossz] of Integer;
  {i. eleme=innentől visszafelé ennyi betűs tükörszót
   választottunk}

Var
  S, {a felbontandó szó}
  SKi:String; {a tükörszavakra bontott output szöveg}
  min,i:Word;
  valaszt:TMinFelbont; {S-hez hozzáválasztott felbontás
   betű-száma}

{$i AltRutin.inc}

Function T(i,j:Word):Boolean;
  {Ef:  $i \leq j$  ES  $i, j \text{ ELEME } [1..Hossz(S)]$ 
   Uf:  $T(i,j) = S(i..j)$  tükörszó-e}

Begin
  While (i<=j) and (S[j]=S[i]) do
    Begin
      Inc(i); Dec(j);
    End;
  T:=i>j
End;

Function R(i:Word):Word;
  {Uf:  $R(i)=i$ . jelleg a minimum tükörfelbontás-beli
   tükörszavak száma}

Var
  j,min:Word;

```

```

Begin
  If
    (i=0) or (i=1) then
      Begin
        R:=i; valaszt[i]:=i
      End
    Else
      Begin
        min:=R(i-1)+1;
        For j:=i-1 downto 1 do
          Begin
            If T(j,i) then
              If R(j-1)+1<min then
                Begin
                  min:=R(j-1)+1;
                  Write(j:2,'..',i:2,' ',i-j+1:3,'-betüs(',
                    Copy(Copy(S,j,i-j+1)+')-
                      1,23));
                  valaszt[i]:=i-j+1
                End;
              End;
            R:=min
          End
        End;
      End;
End;

Begin
  UjLap('Minimális tükörszavakra bontás',0);
  S:='abbakabadara';

  valaszt[0]:=0;
  For i:=1 to MaxHossz do
    Begin
      valaszt[i]:=1; {minden egybetűs szó tükörszónak is jó}
    End;

    min:=R(Length(S));

    Write(CrLf,min,'=');
    SKi:='';

    While Length(S)>0 do
      Begin
        i:=valaszt[Length(S)];
        SKi:=Copy(S,Length(S)-i+1,i)+' '+SKi;
        S:=Copy(S,1,Length(S)-i);
      End;

      Writeln(SKi);
      BillreVar;
    End.

```

2. FELADAT – 2. MEGOLDÁS

Keressük az $R(\text{Hossz}(S))$ értékét, ahol

$$R(i) = \underset{n}{Min} \{ n \mid S(1..i) = S(1..i_1) + S(i_1..i_2) \dots + S(i_{n-1}..i) \\ \wedge T(1..i_1) \wedge T(i_1..i_2) \wedge \dots \wedge T(i_{n-1}..i) \},$$

azaz az $S(1..i)$ legkevesebb tükörszóra bontását.

Jelölések:

- $S(i..j)$:= S szó i - j . betűi,
- „+” a *konkatenáció* műveleti jele,
- $T(i..j)$:= $S(i..j)$ tükörszó-e.

R rekurzívan kiszámítható az alábbiak szerint:

$R(i) := i$, ha $i=0$ vagy $i=1$

$R(i) := \underset{j=i-1}{Min} (R(i-1)+1, \underset{j=i-1}{Min} (R(j-1)+1 \mid T(j..i)))$, egyébként.

(Magyarázat az R -def rekurzív ágához: az $S(1..i-1)$ -hez az $S(i)$ betűt hozzáveszem 1-betűs tükörszóként, vagy $S(1..i-2)$ -höz hozzávéve 2-, $S(1..i-3)$ -hez 3-, ... $S(1..i-i)$ -hez i -betűs tükörszót alkot...)

Az R iteratíván is könnyen meghatározható. Mivel $R(i)$ -hez csak a kisebb indexű $R(j)$ -k ($j < i$) szükségesek, ezért a rekurzív definíció *nem rekurzív* sorára támaszkodva *alulról-felfelé iterálva* kiszámítható.

Demó. (A rekurzív hívások száma lényegesen csökkent. Jól követhetők a később optimálisnak nem bizonyult próbálkozásai is.)

Program MinimalisTukorSzoraBontas;

```
{
  2. megoldás: rekurzív
  Keressük a  $R(i) = \underset{n}{Min} ( n \mid S(1..i) = S(1..i_1) + \dots + S(i_{n-1}..i) \text{ ES } i_n = i \\ \text{ES } T(1..i_1) \text{ ES } \dots \text{ ES } T(i_{n-1}..i) )$ ,
  azaz az  $S(1..i)$  szó minimális tükörszó-felbontása,
  ahol
  *  $S(j..i)$  az  $S$  szó  $j$ - $i$ . betűi,
  * a "+" a konkatenáció műveleti jele,
  *  $T(i..j) := S(i..j)$  tükörszó-e.
   $R(x) := 0$ , ha  $x=0$ , vagy 1
   $R(x) := \underset{j=i-1}{Min} (R(i-1)+1, \underset{j=i-1}{Min} (R(j-1)+1 \mid T(j..i)))$ , egyébként.
  Rekurzió helyett iteráció, amire az ad módot,
  hogy  $R(i)$  meghatározásához csak a megelőző  $R(j)$ -k szükségesek,
  így alulról-felfelé számíthatók.
  A  $T(,)$ -tesztelést is rábízunk egy tömbre,
  így elegendő egyszer kiszámolni.
}
```

Uses

Newdelay, Crt;

Const

MaxHossz=20;

```

Type
  TMinFelbont=Array [0..MaxHossz] of Integer;
    {i. eleme=innentől visszafelé ennyi betűs tükörszót
     választottunk}
  TTukorE=Array [0..MaxHossz,0..MaxHossz] of Boolean;
Var
  S,           {a felbontandó szó}
  SKi:String; {a tükörszavakra bontott output szöveg}
  min,i,j:Word;
  valaszt:TMinFelbont; {S-hez hozzáválasztott felbontás
                        betű-száma}
  R:TMinFelbont;       {S-hez hozzáválasztott min.
                        felbontás-szám}
  T:TTukorE;           {S-hez tartozó 'S(i..j) tükörszó-e'
                        mátrix}

  {$i AltRutin.inc}

Function TInic(i,j:Word):Boolean;
  {Ef: i<=j ES i,j ELEME [1..Hossz(S)]
   Uf: T(i,j)=S(i..j) tükörszó-e}
Begin
  While (i<=j) and (S[j]=S[i]) do
  Begin
    Inc(i); Dec(j);
  End;
  TInic:=i>j
End;

Function RIter(N:Word):Word;
  {Uf: RIter(N)=N. jelleg a minimum tükörfelbontás-beli
   tükörszavak száma}

  Var
    j,min:Word;
Begin
  R[0]:=0; R[1]:=1; {Az R-def. nem rekurzív ága}
  For i:=2 to N do {Az R-def rekurzív ág megfelelője}
  Begin
    min:=R[i-1]+1;
    For j:=i-1 downto 1 do
    Begin
      If T[j,i] then
        If R[j-1]+1<min then
          Begin
            min:=R[j-1]+1;
            Write(j:2, '..',i:2, ' ',i-j+1:3, '-betüs(',
                  Copy(Copy(S,j,i-j+1)+')-
                        1,23));
            valaszt[i]:=i-j+1
          End;
        End;
      R[i]:=min
    End;
  RIter:=R[N]
End;

```

```

Begin
  {Ef: Hossz(S) <= MaxHossz}
  UjLap('Minimális tükörszavakra bontás', 0);
  S := 'abbakabadara';

  valaszt[0] := 0;
  For i:=1 to MaxHossz do
    Begin
      valaszt[i] := 1; {minden egybetűs szó tükörszónak is jó}
    End;

    For i:=1 to Length(S) do
      Begin
        T[i,i] := True;
        For j:=i+1 to Length(S) do
          Begin
            T[i,j] := TInic(i,j); {van ennél gyorsabb mód is!!!}
          End;
        End;
      End;

      min := RIter(Length(S));

      Write(CrLf, min, '=');
      SKi := '';
      While Length(S) > 0 do
        Begin
          i := valaszt[Length(S)];
          SKi := Copy(S, Length(S) - i + 1, i) + ' ' + SKi;
          S := Copy(S, 1, Length(S) - i);
        End;
        Writeln(SKi);
        BillreVar;
      End.

```

Egy hatékonyabb és „stílszerűbb” T-kiszámítás:

```

For i:=1 to Length(S)-1 do
  Begin
    T[i,i] := True; {1-hosszú szó tükörszó}
    T[i,i+1] := S[i] = S[i+1]; {2-hosszú szó tükörszó,
                                ha jelismétlődés}
  End;
  T[Length(S), Length(S)] := True; {1-hosszú szó tükörszó}
  For k:=2 to Length(S) do {k=hosszúság}
    Begin
      For i:=i+1 to Length(S)-k do
        Begin
          T[i,i+k] := S[i] = S[i+k] and T[i+1,i+k-1];
          {eleje, vége azonos, és közte tükörszó van}
        End;
      End;
    End;
  End;

```

3. FELADAT:

Egy szót tükörszónak nevezünk, ha balról és jobbról kiolvasva betűről betűre megegyezik. (Tehát minden egybetűs szó tükörszó.) Minden szóban található tükörszó, amin azt értjük, hogy ha kitörlünk belőle betűket, akkor tükörszót kapunk.

Írjon programot, amely meghatározza egy adott szóban található leghosszabb tükörszó hosszát!

Példa:

TUKOR.BE		TUKOR.KI
abbakabadara – az elemzendő szó	7	– leghosszabb tükörszava (a b b a k a b a d a r a)

3. FELADAT – 1. MEGOLDÁS

Keressük a $H(1, \text{Hossz}(S))$ értékét, ahol

$$H(i,j) = \max_n \{n \mid \text{Hossz}(S(k..l))=n \wedge k \geq i \wedge l \leq j \wedge T(k..l) \},$$

azaz az $S(i..j)$ szóban $H(i,j)$ hosszúságú tükörszó a leghosszabb.

Jelölések:

- $S(i..j) := S$ szó i - j . betűi,
- $T(i..j) := S(i..j)$ tükörszó-e.

R rekurzívan kiszámítható az alábbiak szerint:

$$H(i,i) := 1$$

$$H(i,i+1) := 2, \text{ ha } S(i) = S(i+1)$$

$$H(i,j) := \max (H(i,j-1), H(i+1,j), H(i+1,j-1)+2), \text{ ha } S(i) = S(j)$$

$$H(i,j) := \max (H(i,j-1), H(i+1,j)), \text{ egyébként.}$$

(Magyarázat az **H**-def rekurzív ágaihoz: az $S(i..j)$ -beli leghosszabb vagy az $S(1..i-1)$ -beli, vagy az $S(i+1..j)$ vagy –feltéve, hogy $S(i) = S(j)$ – $S(i+1..j-1)$ -belinél 2-vel –az elhagyott szélsőkkel– hosszabb.)

Demó. (Irdatlan sok nyomkövető kiírás, amely sok-sok „hiábavaló” próbálkozásra utal.)

```

Program LeghosszabbTukorszo;
{
  1. megoldás: rekurzív
  Keressük a  $H(1, \text{Hossz}(S))$  értékét, vagyis
     $H(i, j) = \text{Max}(n \mid \text{Hossz}(S(k..l)) = n \text{ ES } k \geq i \text{ ES } l \leq j \text{ ES } T(k..l)),$ 
    azaz az  $S(i..j)$  szóban  $n$  hosszúságú tükörszó a
    leghosszabb, ahol
    *  $S(i..j)$  az  $S$  szó  $i$ - $j$ . betüi,
    *  $T(i..j) := S(i..j)$  tükörszó-e.
     $H(i, j) := 1,$  ha  $j = i$ 
     $H(i, j) := 2,$  ha  $j = i + 1$  ES  $S(i) = S(j)$ 
     $H(i, j) := \text{Max} (H(i, j-1), H(i+1, j),$ 
 $H(i+1, j-1) + 2$ ), ha  $S(i) = S(j)$ 
     $H(i, j) := \text{Max} (H(i, j-1), H(i+1, j)),$  egyébként.
}
Uses
  Newdelay, Crt;
Const
  MaxHossz = 100;
Type
  TMaxTukorSzo = Array [1..MaxHossz, 1..MaxHossz] of
    Record max, eset: Integer; End;
    {(i, j). eleme = i..j között ennyi betűs tükörszó a
    leghosszabb; és a 0..3 (0=OK; 1=j.elhagyva;
    2=i.elhagyva; 3=i. és j. elhagyva) eset valamelyikét
    választottuk}
Var
  S, {a felbontandó szó}
  SKi: String; {a tükörszavakra bontott output szöveg}
  maxH, i, j: Word;
  valaszt: TMaxTukorSzo; {S-hez max.tükörszó-hosszak}

  {$i AltRutin.inc}

Function T(i, j: Word): Boolean;
  {Ef:  $i \leq j$  ES  $i, j$  ELEME [1..Hossz(S)]
  Uf:  $T(i, j) = S(i..j)$  tükörszó-e}
Begin
  While (i <= j) and (S[j] = S[i]) do
    Begin
      Inc(i); Dec(j);
    End;
    T := i > j
End;

Function H(i, j: Word): Word;
  {i-j. közötti leghosszabb tükörszó hossza}
Var
  max: Word;

```

```

Begin
  If
    (i=j) then
      Begin
        H:=1;
        valaszt[i,j].max:=1; valaszt[i,j].eset:=0;
      End Else if
    (j=i+1) and (S[i]=S[j]) then
      Begin
        H:=2;
        valaszt[i,j].max:=2; valaszt[i,j].eset:=0;
      End
    Else
      Begin
        max:=H(i,j-1);
        valaszt[i,j].max:=max; valaszt[i,j].eset:=1;
        If max<H(i+1,j) then
          Begin
            max:=H(i+1,j);
            valaszt[i,j].max:=max; valaszt[i,j].eset:=2;
          End;
        If (S[i]=S[j]) and (max<H(i+1,j-1)+2) then
          Begin
            max:=H(i+1,j-1)+2;
            valaszt[i,j].max:=max; valaszt[i,j].eset:=3;
          End;
        H:=max;
        Write('      H(' ,i:2, ', ',j:2, ')=' ,max:3);
      End;
    End;
  End;

Begin
  UjLap('Maximális hosszúságú tükörszó keresése',0);
  S:='abbakabadara';

  maxH:=H(1,Length(S));

  Write(CrLf,maxH,'=');
  i:=1; j:=Length(S);
  While i<j do
    Begin
      Case valaszt[i,j].eset of
        0: Begin Inc(i); Dec(j); End;
        1: Begin S[j]:='#'; Dec(j) End;
        2: Begin S[i]:='#'; Inc(i) End;
        3: Begin Inc(i); Dec(j) End;
      End;
    End;
  End;
  Writeln(S);
  BillreVar;
End.

```

3. FELADAT – 2. MEGOLDÁS

Keressük a $H(1, \text{Hossz}(S))$ értékét, ahol

$$H(i,j) = \underset{n}{\text{Max}} \{ n \mid \text{Hossz}(S(k..l)) = n \wedge k \geq i \wedge l \leq j \wedge T(k..l) \},$$

azaz az $S(i..j)$ szóban $H(i,j)$ hosszúságú tükörszó a leghosszabb.

Jelölések:

- $S(i..j) := S$ szó i - j . betűi,
- $T(i..j) := S(i..j)$ tükörszó-e.

R rekurzívan kiszámítható az alábbiak szerint:

$$H(i,i) := 1$$

$$H(i,i+1) := 2, \text{ ha } S(i) = S(j)$$

$$H(i,j) := \text{Max} (H(i,j-1), H(i+1,j), H(i+1,j-1)+2), \text{ ha } S(i) = S(j).$$

$$H(i,j) := \text{Max} (H(i,j-1), H(i+1,j)), \text{ egyébként.}$$

(Magyarázat az **H**-def rekurzív ágaihoz: az $S(i..j)$ -beli leghosszabb vagy az $S(1..i-1)$ -beli, vagy az $S(i+1..j)$ vagy –feltéve, hogy $S(i) = S(j)$ – $S(i+1,j-1)$ -belinél 2-vel –az elhagyott szélsőkkel– hosszabb.)

A H iteratíván is könnyen meghatározható. Mivel $H(i,j)$ -hez csak a kisebb indexű $H(k,l)$ -k ($k < i$ vagy/és $l < j$) szükségesek, ezért a rekurzív definíció **nem rekurzív sorára támaszkodva **alulról-felfelé iterálva** kiszámítható.**

Demó. (Minimális számítás.)

Program LeghosszabbTukorszo;

```
{
  2. megoldás: iteratív
  Keressük a  $H(1, \text{Hossz}(S))$  értékét, vagyis
     $H(i,j) = \text{Max} ( n \mid \text{Hossz}(S(k..l)) = n \text{ ES } k \geq i \text{ ES } l \leq j \text{ ES } T(k..l) ),$ 
  azaz az  $S(i..j)$  szóban  $n$  hosszúságú tükörszó a
  leghosszabb, ahol
  *  $S(i..j)$  az  $S$  szó  $i$ - $j$ . betűi,
  *  $T(i..j) := S(i..j)$  tükörszó-e.
   $H(i,j) := 1,$  ha  $j=i$ 
   $H(i,j) := 2,$  ha  $j=i+1$  ES  $S(i) = S(j)$ 
   $H(i,j) := \text{Max} (H(i,j-1), H(i+1,j),$  ha  $S(i) = S(j)$ 
     $H(i+1,j-1)+2),$  
   $H(i,j) := \text{Max} (H(i,j-1), H(i+1,j)),$  egyébként.

  A hatékonyság lehetőségét az adja, hogy tömbben alulról
  felfelé sorrendben is kiszámíthatók a  $H(i,j)$  értékei.
}

Uses
  Newdelay, Crt;
Const
  MaxHossz=20;
```

Type

```

TMaxTukorSzo=Array [1..MaxHossz,1..MaxHossz] of
    Record max,eset:Integer; End;
    {(i,j). eleme=i..j között ennyi betűs tükörszó a
    leghosszabb; és a 0..3 (0=OK; 1=j.elhagyva;
    2=i.elhagyva; 3=i. és j. elhagyva) eset valamelyikét
    választottuk}
TH=Array [1..MaxHossz,1..MaxHossz] of Integer;
    {A H hatékony kiszámtásához}
TT=Array [1..MaxHossz,1..MaxHossz] of Boolean;
    {A T-"tükörség" hatékony kiszámtásához}

```

Var

```

S,                {a felbontandó szó}
SKi:String;      {a tükörszavakra bontott output szöveg}
maxH,i,j:Word;
valaszt:TMaxTukorSzo; {S-hez max.tükörszó-hosszak}
HTomb:TH;
TTomb:TT;

```

```
{$i AltRutin.inc}
```

Procedure TInic;

```

    Var
        i,j,k:Integer;
    Begin
        {Triviális esetek:}
        For i:=1 to Length(S) do
            Begin
                TTomb[i,i]:=True;                {1-hosszúságú tükörszavak}
                TTomb[i,i+1]:=S[i]=S[i+1]        {2-hosszúságú tükörszavak?}
            End;
            {Visszavezetés már számoltakra:}
            For k:=2 to Length(S)-1 do{k=a vizsgált rész-szó hossza}
                Begin
                    For i:=1 to Length(S)-k do
                        Begin
                            TTomb[i,i+k]:=(S[i]=S[i+k]) and TTomb[i+1,i+k-1]
                        End;
                    End;
            End;
    End;

```

Procedure HInic;

```

    Var
        i,j:Integer;
    Begin
        For i:=1 to MaxHossz do For j:=1 to MaxHossz do
            HTomb[i,j]:=-1;
        HTomb[Length(S),Length(S)]:=1; {1-hosszú tükörszavak}
        For i:=1 to Length(S)-1 do
            Begin
                HTomb[i,i]:=1;                {1-hosszú tükörszavak}
                If S[i]=S[i+1] then HTomb[i,i+1]:=2
                    {2-hosszú tükörszavak}
                else HTomb[i,j]:=1;
            End;
    End;

```

```

    For j:=i+2 to Length(S) do
    Begin
        If TTomb[i,j] then HTomb[i,j]:=j-i+1;
    End;
    End;
End;

Function H(i,j:Word):Word;
    {i-j. közötti leghosszabb tükörszó hossza}
    Var
        max:Word;
    Begin
        If
            (i=j) then
                Begin
                    H:=1;
                    valaszt[i,j].max:=1; valaszt[i,j].eset:=0;
                End Else if
                    (j=i+1) and (S[i]=S[j]) then
                        Begin
                            H:=2;
                            valaszt[i,j].max:=2; valaszt[i,j].eset:=0;
                        End
                    Else
                        Begin
                            If HTomb[i,j]>-1 then
                                Begin {már ismert H-érték}
                                    H:=HTomb[i,j];
                                    {Nem kell már, mert ki lett töltve:
                                        valaszt[i,j].max:=HTomb[i,j];
                                        valaszt[i,j].eset:=0;}
                                End
                            Else
                                Begin {még nem ismert H-érték}
                                    max:=H(i,j-1);
                                    valaszt[i,j].max:=max; valaszt[i,j].eset:=1;
                                    If max<H(i+1,j) then
                                        Begin
                                            max:=H(i+1,j);
                                            valaszt[i,j].max:=max; valaszt[i,j].eset:=2;
                                        End;
                                    If (S[i]=S[j]) and (max<H(i+1,j-1)+2) then
                                        Begin
                                            max:=H(i+1,j-1)+2;
                                            valaszt[i,j].max:=max; valaszt[i,j].eset:=3;
                                        End;
                                    H:=max;
                                    HTomb[i,j]:=max;
                                    Write('      H(' ,i:2, ', ',j:2, ')=' ,max:3);
                                End;
                            End;
                        End;
                    End;
        End;
    End;

Begin
    UjLap('Maximális hosszúságú tükörszó keresése',0);
    S:='abbakabadara';

```

```

TInic;
HInic;

maxH:=H(1,Length(S));

Write(CrLf,maxH,'=');
i:=1; j:=Length(S);
While i<j do
Begin
    Case valaszt[i,j].eset of
        0: Begin Inc(i); Dec(j); End;
        1: Begin S[j]:='#'; Dec(j) End;
        2: Begin S[i]:='#'; Inc(i) End;
        3: Begin Inc(i); Dec(j) End;
    End;
End;
Writeln(S);
BillreVar;

End.

```

4. FELADAT

Egy fazekas műhelyében sorban várakoznak az égetésre váró tárgyak. Minden tárgyról tudjuk, hogy mennyi az a legkevesebb idő ($idő_{1..N}$), ami a kiégetéshez kell. Az égetésre váró tárgyakat érkezésük sorrendjében kell kiégetni. Egyszerre legfeljebb K darabot égethetünk. Az égetésidő egy menetben mindig a kemencébe rakott tárgyak (minimális) égetési idejének maximum ($Max\ idő_{j..k}$) kell legyen.

Írjon programot, amely meghatározza, mennyi a legkevesebb idő, ami szükséges az összes tárgy kiégetéséhez, továbbá megadja azt is, hogy ezen idő eléréséhez mely tárgyakat kell egy-egy menetben a kemencében égetni.

Példa:

FAZEKAS . BE		FAZEKAS . KI	
7 3	– égetendő /beférő tárgyak száma	75	– a minimális égetési idő
10	– az 1. tárgy égetési ideje	1 2	– az egyszerre égetendők
8	– a 2. tárgy égetési ideje	3 4	– az egyszerre égetendők
20	– a 3. tárgy égetési ideje	5 7	– az egyszerre égetendők
25	– a 4. tárgy égetési ideje		
30	– az 5. tárgy égetési ideje		
12	– a 6. tárgy égetési ideje		
40	– a 7. tárgy égetési ideje		

Megoldás:

A problémával való barátkozásként elemezzük az alábbi eseteket!

1. Biztos kedvezőtlen megoldás, de jó viszonyítási pont: *pontosan* egyet rakunk a kemencébe.

$$Idő(1..5)=(10, 8, 25, 20, 9)$$

$$\text{Ekkor a szükséges idő: } 10+8+25+20+9=72.$$

2. *Párosával* égetjük ki ($K \geq 2$).

$$Idő(1..5)=(10, 8, 25, 20, 9)$$

$$\text{Ekkor a szükséges idő: } \text{Max}(10,8)+\text{Max}(25,20)+\text{Max}(9)=10+25+9=44.$$

3. *Hármasával* ($K \geq 3$; reményeink szerint továbbjavul a helyzet a többszörös párhuzamosodás miatt)

$$Idő(1..5)=(10, 8, 25, 20, 9)$$

$$\text{Ekkor a szükséges idő: } \text{Max}(10,8,25)+\text{Max}(20,9)=25+20=45.$$

A szomorú következtetés: nem vezet garantáltan helyes eredményre a mohó (a pillanatnyilag legjobbnak látszó) „mindig a lehető legtöbbet égetni” elv. Megoldás: meg kell találni azokat a „vágáspontokat”, ahol az együtt égetendők csoportjainak a határai vannak. Azaz egy –az elsőtől indulva– optimális(j)-hez ($Opt(j)$) vegyük hozzá azt az 1, 2,... K

cserepet, amelynél a legkedvezőbb alakul az össződ. A probléma forrása, hogy egy adott i esetén a hozzá tartozó legkisebb idő megtalálásához visszafelé lépdelve kell meghatározni a lehetséges $(i-1, i-2, \dots, i-K+1)$ vágások idejét.

Következzék az algoritmus előrevetítése manuálisan. Legyen az idősorozat: (10,8,12,18,6). Haladjunk előre felé, a magasabb indexek felé, úgy, hogy kipróbáljuk: mennyi lenne az össz égetési idő, ha az aktuális elemet egykeként hozzávennénk, majd, ha az előzővel alkotna párt, és így tovább.

index	1	1 2	1 2 3	1 2 3 4	1 2 3 4 5
idő	10	10, 8	10, 8, 12	10, 8, 12, 18	10, 8, 12, 18, 6
opt	10	10 10	10 10 12	10 10 12 28	10 10 12 28 28
<i>Próbálkozás az (aktuális) i. helyen: csoportosítva a j.-kkel kezdve, majd közülük minimumot választva:</i>					
j=i		10+8=18	10+12=22	12+18=30	28+6=34
j=i-1		0+10=10	10+12=22	10+18=28	12+18=30
j=i-2			0+12=12	10+18=28	10+18=28

Keressük az **Opt(N)** értékét, ahol

$$\text{Opt}(0)=0$$

$$\text{Opt}(i)=\text{Min} \left\{ \text{Opt}(i-1)+\text{idő}_i, \text{Opt}(j-1)+\text{Max}_{m=j}^i \{\text{idő}_m\} \forall j: i-j+1 \leq K \right\},$$

azaz az N.-ig minimálisan szükséges égetési idő.

Ahol

- idő_i az i . tárgy égetési ideje;
- K az egyszerre égethető tárgyak maximális száma.

Magyarázat:

- $\text{Opt}(i-1)+\text{idő}_i$
– egyedülként égetni, s hozzávenni az addigi optimális időhöz;
- $\text{Opt}(j-1)+\text{Max}_{m=j}^i \{\text{idő}_m\} \forall j: i-j+1 \leq K$
– az összes legfeljebb K -val előbbi helyen „vágva” (így legfeljebb K darabból álló csoportot alkotva) kiszámolni a számukra szükséges égetési időt, s hozzá véve az összesen addig szükséges optimális időhöz.

[Demó.](#)

```

Program FazekakEgeteseMinimalisIdoAlatt;
  (*
    Opt(i)--az i. fazéig szükséges optimális (minimális idő

    Opt(0):=0
      ( Opt(i-1)+Ido(i)          -- (i)-t egyedül égetni
    Opt(i):=Min{
      \ Opt(j-1)+Max{Ido(j..i)} MINDEN j: i-j+1<=K
                                     -- (j..i)-ket együtt égetni

    CsopDbk(i)--az i-edikkel együtt egy csoportba tartozó fazekak
      száma
    A kötött sorrend miatt igaz, hogy
    i-dikkel együtt égetendő: i-CsopDbk(i)+1..i
  *)
Uses
  Newdelay,Crt;
  {$i AltRutin.inc}
Const
  MaxN=100;
Type
  TIdok=Array [0..MaxN] of Word; {szükséges égetési idők}
  TCsop=Array [1..MaxN] of Byte; {egy csoportba tartozó
                                     fazekak száma, vagy lehetne
                                     a csoport első elemének
                                     indexe is}
Var
  N,K:Word;
  ido:TIdok;
  opt:TIdok;
  CsopDbk:TCsop;

Procedure OptSzamitas;
  Var
    opti,optVagas,maxIdo,
    i,j:Integer;
Begin
  opt[0]:=0;
  For i:=1 to N do
    {CiklusInvariáns: opt(1..i-1) minimális égetési idő}
    Begin
      opti:=opt[i-1]+ido[i]; {az i egyedül megy a kemencébe}
      optVagas:=i;
      {maximum/optimum-kiválasztás:}
      maxIdo:=ido[i]; j:=i-1; {a vágás j mögött}
      While (j>0) {vágás legfeljebb az első mögött} and
        (i-j+1<=K) {vágás legfeljebb K-nyival előbbre} do
        Begin
          If maxIdo<ido[j] then
            {meghosszabbodik a csoport égetési ideje}
            Begin
              maxIdo:=ido[j];
            End;
        End;
  End;

```

```

    If opt[j-1]+maxIdo<opti then
      {j-nél jobb vágni => következő csoport: j..i}
    Begin
      opti:=opt[j-1]+maxIdo; optVagas:=j;
    End;
    Dec(j)
  End;
  opt[i]:=opti;
  csopDbk[i]:=i-optVagas+1; {i.-kel együtt égetendők:
                             optVagas..i; lehetne egyszerűen:
                             csopDbk[i]:=optVagas is}

End;
End;

{munkaváltozók:}
Var
  tol,ig:Integer;
Begin
  UjLap('Fazekak minimális égetési ideje',0);
  {paraméterbeolvasás helyett:}
  N:=7; K:=3;
  ido[1]:=10; ido[2]:=8; ido[3]:=20; ido[4]:=25;
  ido[5]:=30; ido[6]:=12; ido[7]:=40;
  {a lényeg:}
  OptSzamitas;
  {megjelenítés:}
  UjLap('Fazekak minimális égetési ideje',0);
  Writeln('Minimális égetési idő:',opt[N]);
  ig:=N;
  While ig>=1 do
    Begin
      tol:=ig-CsopDbk[ig]+1;
      Writeln(tol,' ',ig);
      ig:=tol-1;
    End;
  BillreVar;
End.

```

5. FELADAT

Egy fazekas műhelyében sorban várakoznak az égetésre váró tárgyak. Minden tárgyról tudjuk, hogy mennyi a súlya és azt a legkevesebb időt ($idő_1 \dots N$), ami a kiégetéséhez kell. Az égetésre váró tárgyakat érkezésük sorrendjében kell kiégetni. Egyszerre legfeljebb több tárgyat is rakhatunk a kemencébe, azonban összsúlyuk nem haladhatja meg a K kemence kapacitását. Az égetésidő egy menetben mindig a kemencébe rakott tárgyak (minimális) égetési idejének maximum ($\max idő_j \dots k$) kell legyen.

Írjon programot, amely meghatározza, mennyi a legkevesebb idő, ami szükséges az összes tárgy kiégetéséhez, továbbá megadja azt is, hogy ezen idő eléréséhez mely tárgyakat kell egy-egy menetben a kemencében égetni.

Példa:

FAZEKAS . BE	FAZEKAS . KI
6 10 – égetendő /beférő tárgyak száma	4 6 – a minimális égetési idő
3 10 – az 1. tárgy súlya és égetési ideje	1 1 – az egyszerre égetendő
2 12 – a 2. tárgy súlya és égetési ideje	2 3 – az egyszerre égetendő
7 20 – a 3. tárgy súlya és égetési ideje	4 6 – az egyszerre égetendő
5 15 – a 4. tárgy súlya és égetési ideje	
3 11 – az 5. tárgy súlya és égetési ideje	
2 16 – a 6. tárgy súlya és égetési ideje	

Megoldás:

Az előzőhöz nagyban hasonlít. Abból lépünk tovább.

Keressük az **Opt(N)** értékét, ahol

Opt(0)=0

$$\mathbf{Opt(i)=Min \{ Opt(i-1)+idő_i, \quad Opt(j-1)+\underset{m=j}{\overset{i}{Max}} \{idő_m\} \sum_{k=j}^i súly_k \leq K \},}$$

azaz az N.-ig minimálisan szükséges égetési idő.

Ahol

- $idő_i$ az i. tárgy égetési ideje;
- $súly_i$ az i. tárgy súlya;
- K az egyszerre égethető tárgyak maximális súlya.

Magyarázat:

- $Opt(i-1)+idő_i$
– egyedülként égetni, s hozzávenni az addigi optimális időhöz;

- $Opt(j-1) + \underset{m=j}{\overset{i}{Max}} \{ idő_m \mid \sum_{k=j}^i súly_k \leq K \}$

– az összes valahány tárgy előtti helyen „vágva” (így legfeljebb K-súlyú csoportot alkotva) kiszámolni a számukra szükséges égetési időt, s hozzá véve az összesen addig szükséges optimális időhöz.

Demó.

Program FazekakEgeteseMinimalisIdoAlatt;

```
(*
  Opt(i) -- az i. fazéig szükséges optimális (minimális) idő

  Opt(0) := 0
  ( Opt(i-1) + Ido(i)          -- (i)-t egyedül égetni
  Opt(i) := Min {
    ( Opt(j-1) + Max { Ido(j..i) MINDEN j: SZUM(k=j..i)
                      súly(k) <= K }
    -- (j..i)-ket együtt égetni
  CsopDbk(i) -- az i-edikkel együtt egy csoportba tartozó fazekak
               száma
  A kötött sorrend miatt igaz, hogy
  i-dikkel együtt égetendő: i - CsopDbk(i) + 1..i
*)
```

Uses

```
Newdelay, Crt;
{$i AltRutin.inc}
```

Const

```
MaxN = 100;
```

Type

```
TFazekak = Array [0..MaxN] of
  Record
    ido: Word; {szükséges égetési idők}
    súly: Word
  End;

TIdok = Array [0..MaxN] of Word;
TCsop = Array [1..MaxN] of Byte; {egy csoportba tartozó fazekak
                                   száma}
```

Var

```
N, K: Word;
fazek: TFazekak;
opt: TIdok;
CsopDbk: TCsop;
```

Procedure OptSzamitas;

Var

```
opti, optVagas, maxIdo, súly,
i, j: Integer;
```

```

Begin
  opt[0]:=0;
  For i:=1 to N do
    {CiklusInvariáns: opt(1..i-1) minimális égetési idő}
    Begin
      opti:=opt[i-1]+fazek[i].ido; {az i egyedül megy a kemencébe}
      optVagas:=i;
      {maximum/optimum-kiválasztás:}
      maxIdo:=fazek[i].ido; suly:=fazek[i].suly;
      j:=i-1; {a vágás j mögött}
      While (j>0) {vágás legfeljebb az első mögött} and
        (suly+fazek[j].suly<=K) {vágás legfeljebb K súlynyi} do
        Begin
          If maxIdo<fazek[j].ido then
            {meghosszabbodik a csoport égetési ideje}
            Begin
              maxIdo:=fazek[j].ido;
            End;
            suly:=suly+fazek[j].suly;
            If opt[j-1]+maxIdo<opti then
              {j-nél jobb vágni => következő csoport: j..i}
              Begin
                opti:=opt[j-1]+maxIdo; optVagas:=j;
              End;
              Dec(j)
            End;
          opt[i]:=opti;
          csopDbk[i]:=i-optVagas+1; {i.-kel együtt égetendők:
                                   optVagas..i}

        End;
      End;

  {munkaváltozók:}
  Var
    tol,ig:Integer;
Begin
  UjLap('Fazekak minimális égetési ideje',0);
  {paraméterbeolvasás helyett:}
  N:=6; K:=10;
  fazek[1].ido:=10; fazek[1].suly:=3; fazek[2].ido:=12;
  fazek[2].suly:=2; fazek[3].ido:=20; fazek[3].suly:=7;
  fazek[4].ido:=15; fazek[4].suly:=5; fazek[5].ido:=11;
  fazek[5].suly:=3; fazek[6].ido:=16; fazek[6].suly:=2;
  {a lényeg:}
  OptSzamitas;
  {megjelenítés:}
  UjLap('Fazekak minimális égetési ideje',0);
  Writeln('Minimális égetési idő:',opt[N]);

```

```
ig:=N;  
While ig>=1 do  
  Begin  
    tol:=ig-CsopDbk[ig]+1;  
    Writeln(tol,' ',ig);  
    ig:=tol-1;  
  End;  
  BillreVar;  
End.
```