7. Dinamikus programozás

7.1. Rekurzió memorizálással.

Láttuk, hogy a partíció probléma rekurzív algoritmusa $\Omega(2^{\sqrt{n}})$ eljáráshívást végez. pedig a lehetséges részproblémák száma csak n^2 (vagy $n(n+1)/2$, ha csak az $n \leq k$ eseteket vesszük.) Ennek az az oka, hogy ugyanazon részprobléma megoldása több más részpobléma megoldásához kell, és az algoritmus ezeket mindig újra kiszámítja. Tehát egyerűen gyorsíthatjuk a számítást, ha minden részprobléma (azaz $P2(n,k)$) megoldását tároljuk egy tömbben. Ha hivatkozunk egy részprobléma megoldására, akkor először ellenőrizzük, hogy kiszámítottuk-e már, és ha igen, akkor kiolvassuk az értéket a táblázatból, egyebként rekurzívan számítunk, és utána tároljuk az értéket a táblázatban.
A táblázat inicializálásához válasszunk egy olyan értéket, amely nem lehet egyetlen részprobléma megoldása sem. Esetünkben ez lehet a 0.

```
Program ParticoiRM;
 {A partíció probléma megoldása.
  Módszer: rekurzió memorizálással}
  Function P(N:Word):Int64;
  Const
    MaxN=500;                          {a táblázat mérete MaxN*MaxN}
  Var
    T2:array[1..MaxN,1..MaxN] of Int64;{a részproblémák táblázata}
    i,j:Word;

 Function P2(n,k:Word):Int64;
    Var E:Int64;
    Begin{P2}
      If T2[n,k]<>0 Then            {P2(n,k) értékét már kiszámítottuk}
        P2:=T2[n,k]
      Else Begin                    {P2(n,k) értékét még nem számítottuk ki}
        if (n = 1) Or (k = 1) Then  {rekurzív számítás}
          E := 1
        Else If n <= k Then
          E := 1 + P2(n,n-1)
        Else
          E :=P2(n,k-1) + P2(n-k,k);
        T2[n,k]:=E;                 {memorizálás}
        P2:=E;
      End;
    End{P2};
  Begin{P}
    For i:=1 To N Do                {a táblázat inicializálása}
      For j:=1 To N Do T2[i,j]:=0;
     P:=P2(n,n);
   End{P};
Begin{program}
  WriteLn(P(100));
End{program}.
```

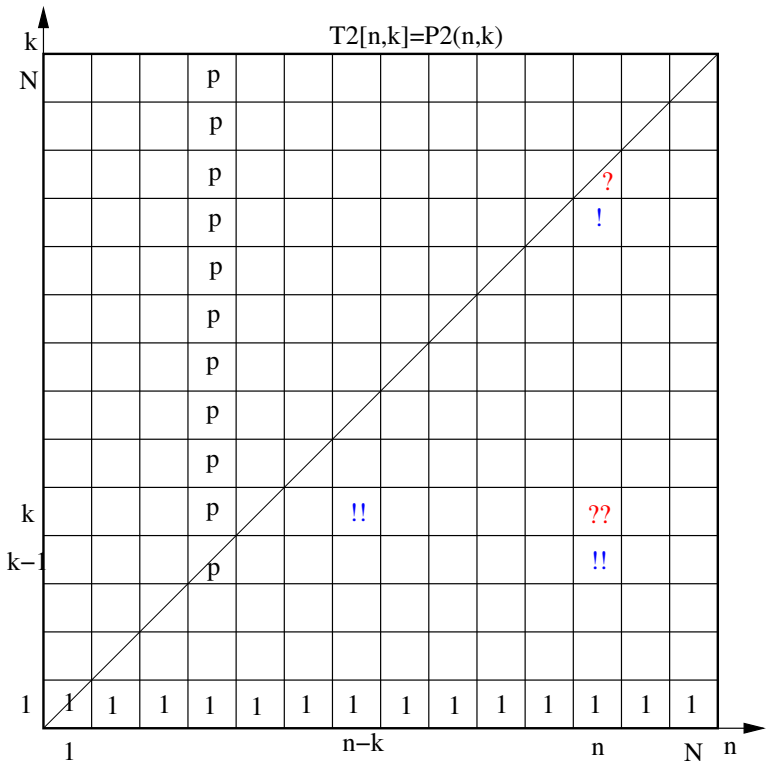
Nyilvánvaló, hogy az algoritmus futási ideje $\Theta(n^2)$, és a tárigénye is $\Theta(n^2)$ lesz, ha csak n^2 méretű táblázatnak foglalunk memóriát dinamikusan az aktuális paraméter függvényében.

7.2. A partíció probéma megoldása táblázatkitöltéssel.

A rekurziót teljesen kiküszöbölhetjük táblázatkitöltéssel. Az 1. árán szemléltetett táblázatot használjunk a részproblémák megoldásainak tárolására. Tehát a $T2[n,k]$ táblázatelem tartalmazza a $P2(n,k)$ részprobléma megoldását. A táblázat első sora azonnal kitölthető, mert $P2(n,1)=1$. Olyan kitöltési sorrendet keresünk, hogy minden $(n,k), k>1$ részprobléma kiszámítása esetén azok a részproblémák, amelyek szükségesek $P2(n,k)$ kiszámításához, már korábban kiszámítottak legyenek.

Általánosan, rekurzív összefüggéssel definiált problémamegoldás esetén egy *r* (rész)probléma *összetevői* azok a részproblémák, amelyek megoldásától *r* megoldása függ. Tehát a táblázatkitöltés alkalmazásához meg kell állapítani a részproblémáknak egy olyan sorrendjét, hogy minden *r* részprobléma minden összetevője előbb álljon a sorrendben, mint *r*. A

1. $P2(1,k)=1, P2(n,1)=1$,
2. $P2(n,n)=1+P2(n,n-1)$,
3. $P2(n,k)=P2(n,n)$ ha $n<k$,
4. $P2(n,k)=P2(n,k-1)+P2(n-k,k)$ ha $k<n$.



1. ábra. Táblázat a Partíció probléma megoldásához.

rekurzív összefüggések megadják az összetevőket:

1. $P2(1,k)$-nak és $P2(n,1)$-nek nincs összetevője,
2. $P2(n,n)$ összetevője $P2(n,n-1)$,
3. $P2(n,k)$ összetevője $P2(n,n)$, ha ($n<k$),
4. $P2(n,k)$ összetevői: $P2(n,k-1)$ és $P2(n-k,k)$, ha ($k<n$).

Tehát a táblázat kitöltése (k-szerint) soronként balról jobbra haladó lehet.
Az algoritmus futási ideje és tárigénye is $\Theta(n^2)$.

```
Program ParticT2; {A partíció probléma megoldása.
 Módszer: négyzetes táblázatkitöltés}

  Function P(N:Word):Int64;
  Const
    MaxN=500; {a táblázat mérete MaxN*MaxN}
  Var
    T2:array[1..MaxN,1..MaxN] of Int64;
    ki,ni,n1:Word;
  Begin{P}
    For ni:=1 To N Do T2[ni,1]:=1;          {az első sor kitöltése}
    For ki:=2 To N Do Begin                 {az ki. sor kitöltése }
      T2[ki,ki]:=T2[ki,ki-1]+1;             {P2(n,n)=P2(n,n-1)+1  }
      For ni:=ki+1 To n Do Begin            {P2(ni,ki)=T2[ni,ki] számítása}
        n1:=ki;                             {P2(n,k)=P2(n,k-1)+P2(n-k,k)  }
        If ni-ki<ki Then n1:=ni-ki;         {P2(n,k)=P2(n,n), ha k>n }
        T2[ni,ki]:=T2[ni,ki-1]+T2[ni-ki,n1];{P2(n,k)=P2(n,k-1)+P2(n-k,k)  }
      End{for ni};
    End{for ki};
    P:=T2[n,n];
  End{P};
Begin{Program}
  WriteLn(P(100));
End{Program}.
```

7.3. A partíció probéma megoldása lineáris táblázatkitöltéssel.

Látható, hogy elegendő lenne a táblázatnak csak két sorát tárolni, mert minden (n,k) részprobléma összetevői vagy a k-adik, vagy a $k-1$-edik sorban vannak. Sőt, elég egy sort tárolni balról-jobbra (növekvő n-szerint) haladó kitöltésnél, mert amelyik részproblémát felülírjuk (($n-k,k$)), annak később éppen az új értéke kell összetevőként.

```
Program ParticD; { A partíciószám probléma megoldása.
  Módszer: dinamikus programozás lineáris táblázatkitöltés helyben }

Function P(N:Word):Int64;
  Const
    MaxN=5000 ;(* a táblázat mérete  *)
  Var
    T : Array[1..MaxN] Of Int64;
    ki,ni : Word;
  Begin{P}
    For ni:=1 To N Do T[ni]:=1; {az első sor kitöltése       }
    For ki:=2 To N Do Begin     {az ki. sor kitöltése        }
      T[ki]:=T[ki]+1;           {P2(n,n)=P2(n,n-1)+1         }
      For ni:=ki+1 To N Do      {P2(n,k)=P2(n,k-1)+P2(n-k,k) }
        T[ni]:=T[ni] + T[ni-ki];
    End{for ki};
    P:=T[N];
  End{P};

Begin{program}
  WriteLn('P(405)= ',P(405));
End{program}.
```

P(405)= 9147679068859117602

7.4. A pénzváltás probléma.

Probléma: Pénzváltás
Bemenet: $P = \{p_1, ..., p_n\}$ pozitív egészek halmaza, és E pozitív egész szám.
Kimenet: Olyan $S \subseteq P$, hogy $\sum_{p \in S} = E$.
Megjegyzés: A pénzek tetszőleges címletek lehetnek, nem csak a szokásos 1, 2, 5 10, 20, stb., és minden pénz csak egyszer használható a felváltásban.
Először azt határozzuk meg, hogy van-e megoldás.
A megoldás szerkezetének elemzése.
Tegyük fel, hogy

$$E = p_{i_1} + \ldots + p_{i_k},\ i_1 < \ldots < i_k$$

egy megoldása a feladatnak. Ekkor

$$E - p_{i_k} = p_{i_1} + \ldots + p_{i_{k-1}}$$

megoldása lesz annak a feladatnak, amelynek bemenete a felváltandó $E - p_{i_k}$ érték, és a felváltáshoz legfeljebb a első $i_k - 1$ ($p_1, \ldots, p_{i_k - 1}$) pénzeket használhatjuk.
Részproblémákra bontás.
Bontsuk részproblémákra a kiindulási problémát: Minden $(X,i)(1 \leq X \leq E, 1 \leq i \leq N)$ számpárra vegyük azt a részproblémát, hogy az X érték felváltható-e legfeljebb az első $p_1, \ldots, p_i$ pénzzel. Jelölje $V(X,i)$ az (X,i) részprobléma megoldását, ami logikai érték; $V(X,i) = Igaz$, ha az X összeg előállítható legfeljebb az első i pénzzel, egyébként $Hamis$.

Összefüggések a részproblémák és megoldásaik között.
Nyilvánvaló, hogy az alábbi összefüggések teljesülnek a részproblémák megoldásaira:
1. $V(X,i) = (X = p_i)$, ha $i = 1$
2. $V(X,i) = V(X,i-1) \vee (X > p_i) \wedge V(X - p_i, i-1)$ ha $i > 1$

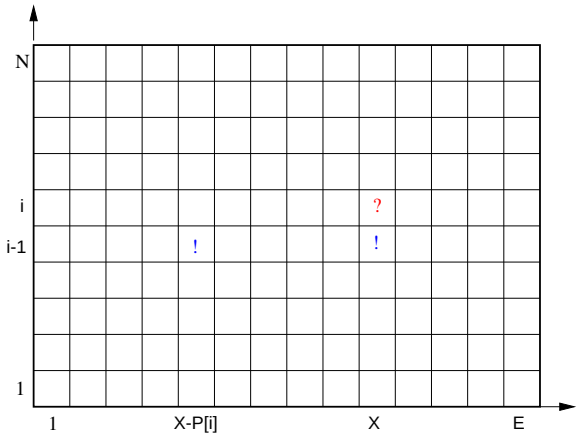
Rekurzív megoldás.
Mivel a megoldás kifejezhető egy V(X,i) logikai értékű függvénnyel, ezért a felírt összefüggések alapján azonnal tudunk adni egy rekurzív függvényeljárást, amely a pénzváltás probléma megoldását adja.

```
Function V(X,i:Word):Boolean;
{Globális:P}
{Módszer: Rekurzív megoldás }
Begin
  V:=(X=P[i])Or
     (i>1) And V(X,i-1) Or
     (i>1) And (X>P[i]) And V(X-P[i],i-1);
End{V};
```

Ez a megoldás azonban igen lassú, legrosszabb esetben a futási idő $\Omega(2^n)$.
Megoldás a részproblémák megoldásainak táblázatos tárolásával.
Vegyünk egy VT táblázatot, amelyben minden lehetséges részprobléma megoldását tároljuk. Mivel minden részproblémát két érték határoz meg, X és i, ezért téglalap alakú táblázat kell. $VT[X,i]$ az (X,i) részprobléma megoldását tartalmazza.

A részproblémák kiszámítási sorrendje. Olyan kiszámítási sorrendet kell megállapítani, amelyre teljesül, hogy amikor az (X,i) részproblémát számítjuk, akkor ennek összetevőit már korábban kiszámítottuk. Mivel az $(X,1)$ részproblémáknak nincs összetevőjük, ezért közvetlenül kiszámíthatóak, azaz a táblázat első sorát számíthatjuk először. Ha $i > 1$, akkor az (X,i) részprobléma összetevői az $(X,i-1)$ és $(X - p_i, i-1)$, ezért az i-edik sor bármely elemét ki tudjuk számítani, ha már kiszámítottuk az $i-1$-edik sor minden elemét. Tehát a táblázatkitöltés sorrendje: soronként (alulról felfelé), balról-jobbra haladó lehet.

```
Function V(E,N:Word):Boolean;
 { Pénzváltás négyzetes táblázatkitöltéssel }
 {Globál: P:Penzek}
Const
  MaxE=100; {a max. felváltandó összeg}
  MaxN=200; {a pénzek max. száma}
Var
  VT:Array[1..MaxE,1..MaxN] Of Boolean;
  i,x:Word;
```



2. ábra. A pénzváltás táblázata

```
Begin{VT}
  For x:=1 To E Do
    VT[x,1]:=False;                {az első sor, azaz V(x,1) számítása}
  VT[P[1],1]:= P[1]<=E;
  For i:=2 To N Do                 {az i-edik sor,azaz V(x,i) számítása}
    For x:=1 To E Do
      VT[x,i]:=(P[i]=X) Or
               VT[x,i-1] Or
               (x>P[i]) And VT[x-P[i],i-1];
  V:=VT[E,N];
End{V};
```

Egy megoldás előállítása a megoldás visszafejtésével. Akkor és csak akkor van megoldása a problémának, ha a VT táblázat kitöltése után $VT[E,N]$ értéke igaz. Ekkor az (1-2.) képletek szerint a legnagyobb i indexű p_i pénz, amely szerepelhet E előállításában, az a legnagyobb index, amelyre

$$VT[E,i] = True \wedge (VT[E,i-1] = False)$$

De ekkor $VT[E-P[i],i-1]$ igaz, tehát $E-p_i$ előállítható az első $i-1$ pénz felhasználásával. Tehát a fenti eljárást folytatni kell $E := E-p_i, i := i-1$-re mindaddig, amíg E 0 lesz.

```
Procedure PenzValt1(    E:Word; Const P:Penzek; N: Word;
                    Var Db:Word; Var C :Megoldas);
Const
  MaxE=300;{a max. felváltható érték}
Var
  VT:Array[0..MaxE, 0..MaxN] Of Boolean;
  i,X:Integer;
Begin{PenzValt1}
  For X:=1 To E Do
    VT[X,1]:=False;                {az első sor, azaz V(X,1) számítása}
  VT[P[1],1]:= P[1]<=E;
  For i:=2 To N Do                {az i-edik sor,azaz V(X,i) számítása}
    For X:=1 To E Do
      VT[X,i]:=(P[i]=X) Or
               VT[X,i-1] Or
               (X>P[i]) And VT[X-P[i],i-1];
  Db:=0; X:=E; i:=N                        { egy megoldás előállítása}
  If Not VT[E,N] Then Exit;                          {nincs megoldás}
```

```
    Repeat
      While (i>0) And VT[X,i] Do Dec(i);
      Inc(Db); C[Db]:=i+1;                {i+1 bejegyzése a megoldásba}
      X:=X-P[i+1];                  {X-P[i+1] felváltásával folytatjuk}
    Until X=0;
End{PenzValt1};
```

Ha csak arra kell valszolni, hogy létezi-e megoldása a problémának, akkor elég a táblázat egy sorát tárolni, mert soronként visszafelé (x-szerint csökkenő sorrendben) haladó kitöltést alkalmazhatunk.

```
Function PenzValt1L(E:Word; Const P:Penzek; N: Word):Boolean;
 { Lineáris táblázatkitöltéssel }
Const
  MaxE=60000;
Var
  T:Array[0..MaxE] Of Boolean;
  i,x:Word;
Begin{PenzValt1L}
  For x:=1 To E Do T[x]:=False;
  T[0]:=True;
  If P[1]<=E Then T[P[1]]:=True;
  For i:=2 To N Do
    For x:=E DownTo 1 Do
     T[x]:=T[x] Or (x>=P[i]) And T[x-P[i]];
  PenzValt1L:=T[E];
End{PenzValt1L};
```

7.5. Az optimális pénzváltás probléma.

Probléma: Optimális pénzváltás
Bemenet: $P = \{p_1, ..., p_n\}$ pozitív egészek halmaza, és E pozitív egész szám.
Kimenet: Olyan $S \subseteq P$, hogy $\sum_{p \in S} = E$ és $|S| \rightarrow$ minimális
Először is lássuk be, hogy az a mohó stratégia, amely mindig a lehető legnagyobb pénzt választja, nem vezet optimális megoldáshoz. Legyen $E = 8$ és a pénzek halmaza legyen $\{5,4,4,1,1,1\}$. A mohó módszer a $8 = 5+1+1+1$ megoldást adja, míg az optimális a $8 = 4+4$.
Az optimális megoldás szerkezetének elemzése.
Tegyük fel, hogy

$$E = p_{i_1} + \ldots + p_{i_k},\ i_1 < \ldots < i_k$$

egy optimális megoldása a feladatnak. Ekkor

$$E - p_{i_k} = p_{i_1} + \ldots + p_{i_{k-1}}$$

optimális megoldása lesz annak a feladatnak, amelynek bemenete a felváltandó $E - p_{i_k}$ érték, és a felváltáshoz legfeljebb a első $i_k - 1$ $(p_1, \ldots, p_{i_k - 1})$ pénzeket használhatjuk. Ugyanis, ha lenne kevesebb pénzből álló felváltása $E - p_{i_k}$-nak, akkor E-nek is lenne k-nál kevesebb pénzből álló felváltása.
Részproblémákra és összetevőkre bontás.
A részproblémák legyenek ugyanazok, mint az előző esetben. Minden (X,i) $(1 \leq X \leq E, 1 \leq i \leq N)$ számpárra vegyük azt a részproblémát, hogy legkevesebb hány pénz összegeként lehet az X értéket előállítani legfeljebb az első i $\{p_1, \ldots, p_i\}$ pénz felhasználásával. Ha nincs megoldás, akkor legyen ez az érték $N+1$. Jelölje az (X,i) részprobléma optimális megoldásának értékét $Opt(X,i)$. Definiáljuk az optimális megoldás értékét $X = 0$-ra és $i = 0$-ra is, azaz legyen $Opt(X,0) = N+1$ és $Opt(0,i) = 0$. Így $Opt(X,i)$-re az alábbi rekurzív összefüggés írható fel. **A részproblémák optimális megoldásának kifejezése az összetevők optimális megoldásaival.**

$$Opt(X,i) = \begin{cases} N+1 & \text{ha} \quad i = 0 \wedge X > 0 \\ 0 & ha \quad X = 0 \\ Opt(X, i-1) & \text{ha} \quad X < p_i \\ \min(Opt(X,i-1), 1 + Opt(X - p_i, i-1)) & \text{ha} \quad X \geq p_i \end{cases} \tag{1}$$

```
Procedure OptValto(Const P :Penzek; E :Word; N:Word;
                   Var   Db:Word; Var C:Megoldas );
Const  MaxE=300;
Var
  Opt:Array[0..MaxE] Of 0..MaxN+1; {az opt. mego.értéke}
  V:Array[0..MaxE,0..MaxN] Of 0..MaxN+1;
  i,x,Rop:Word;
Begin{OptValto}
  For x:=1 To E Do Begin                {inicializálás}
    Opt[x]:=N+1; V[x,0]:=N+1  End;
  Opt[0]:=0;
  For i:=1 To N Do                   { táblázatkitöltés}
    For x:=E DownTo 1 Do Begin
      If (x>=P[i]) Then
        Rop:=Opt[x-P[i]]+1     {x-P[i] opt.felváltása+1}
      Else
        Rop:=N+1;
      If Rop<Opt[x] Then Begin     {Opt[x]=1+Opt(x,i-1)}
        Opt[x]:=Rop;             {az i. pénz szerpel x }
        V[x,i]:=i;             {optimális felváltásában}
      End Else                       {Opt[x]=Opt(x,i-1)}
        V[x,i]:=V[x,i-1];      {nincs jobb, mint i-1-re}
    End{for x};

  Db:=0; x:=E; i:=N;
  If Opt[E]<=N Then                      { van megoldás}
    Repeat         { egy optimális megoldás előállítása}
      i:=V[x,i];      {i. szerepel x opt. felváltásában}
      Inc(Db); C[Db]:=i;     {i bejegyzése a megoldásba}
      x:=x-P[i];        {x-P[i] opt. felváltását nézzük}
      Dec(i);                     { az 1..i-1 pénzekkel}
    Until x=0;
End{OptValto};
```

A dinamikus programozás stratégiája.

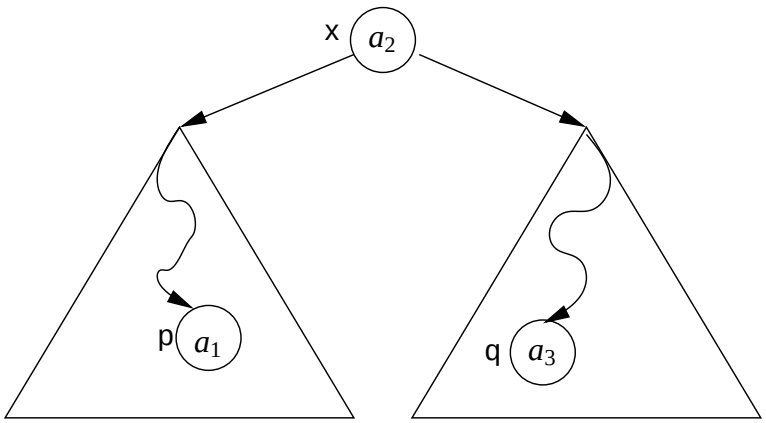
A dinamikus programozás, mint probléma-megoldási stratégia az alábbi öt lépés végrehajtását jelenti.

1. Az [optimális] megoldás szerkezetének elemzése.
2. Részproblémákra és összetevőkre bontás úgy, hogy:
   - a) Az összetevőktől való függés körmentes legyen.
   - b) Minden részprobléma [optimális] megoldása kifejezhető legyen (rekurzívan) az összetevők [optimális] megoldásaival.
3. Részproblémák [optimális] megoldásának kifejezése (rekurzívan) az összetevők [optimális] megoldásaiból.
4. Részproblémák [optimális] megoldásának kiszámítása alulról-felfelé haladva:
   - a) 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.
   - b) A részproblémák kiszámítása alulról-felfelé haladva, azaz táblázatkitöltéssel.
5. Egy [optimális] megoldás előállítása a 4. lépésben kiszámított (és tárolt) információkból.

7.6. Optimális bináris keresőfa előállítása

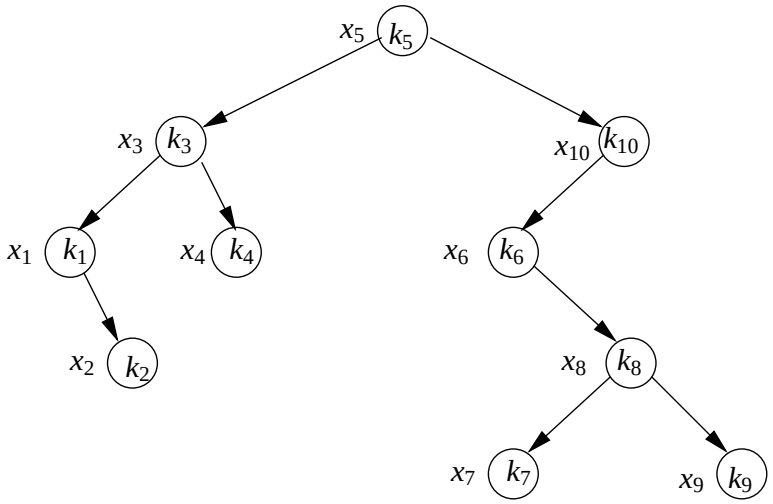
A $F = (M, R, Adat)$ absztrakt adatszerkezetet *bináris keresőfának* nevezzük, ha
1. F bináris fa,

2. $Adat : M \to Elemtip$ és $Elemtip$-on értelmezett egy $\leq$ lineáris rendezési reláció,
3. $(\forall x \in M)(\forall p \in F_{bal(x)})(\forall q \in F_{jobb(x)})(Adat(p) \leq Adat(x) \leq Adat(q))$
A BINKERFAKERES függvényeljárás egy nyilvánvaló megoldása a fában keresése feladatnak.



3. ábra. Bináris keresőfa

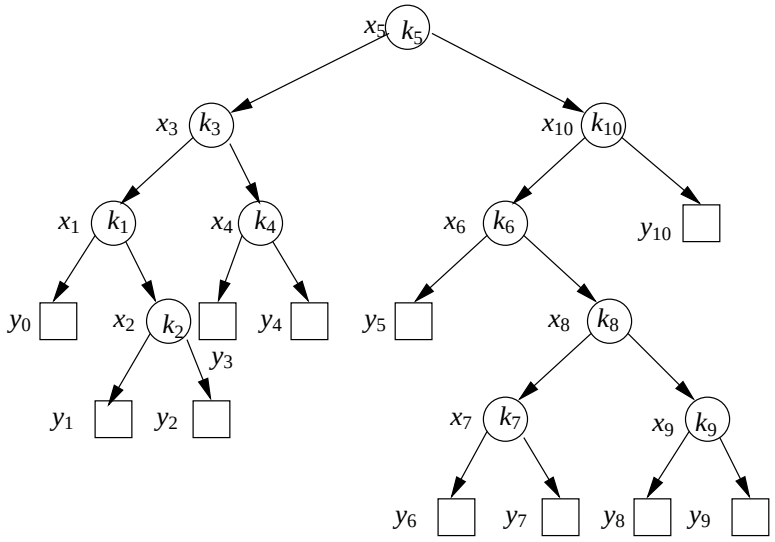
```
Function BinKerFaKeres(a:Adat; F:BinFA):BinFa;
Begin
 While (F<>Nil) And (a<>F^.adat) Do
  If a<F^.adat Then
    F:=F^.bal
  Else
    F:=F^.jobb;
  BinKerFaKeres:=F;
End;
```



4. ábra. 10 adatot (kulcsot) tartalmazó bináris keresőfa

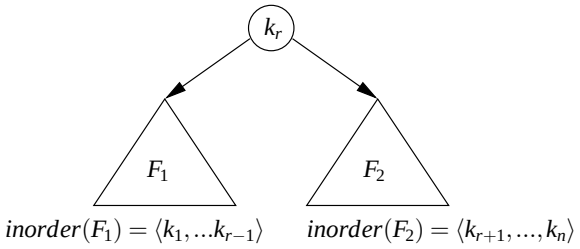
Tegyük fel, hogy ismerjük minden k_i kulcs keresési gyakoriságát, ami p_i ($i = 1, \ldots, n$) Továbbá ismert azon k kulcsok (sikertelen) keresési gyakorisága, amelyre $k_i < k < k_{i+1}$, ami q_i (i=1,. . . ,n), és q_0 a $k < k_1$ kulcsok keresési gyakorisága.
Átlagos keresési idő (költség):

$$V(F) = \sum_{i=1}^{n} p_i \, d_F(x_i) + \sum_{i=0}^{n} q_i \, d_F(y_i)$$



5. ábra. Bináris keresőfa kiegészíteve sikertelen keresési pontokkal

, ahol $d_F(p)$ a p pont mélysége az F fában.
Probléma: Optimális bináris keresőfa előállítása.
Bemenet: $P = \langle p_1, \ldots, p_n \rangle$ sikeres és $Q = \langle q_0, \ldots, q_n \rangle$ sikertelen keresési gyakoriságok.
Kimenet: Olyan F bináris keresőfa, amelynek a $V(F)$ költsége minimális.
Az optimális megoldás szerkezete.
Tegyük fel, hogy a $\langle k_1, \ldots, k_n \rangle$ kulcsokat tartalmazó F bináris keresőfa optimális, azaz $V(F)$ minimális. Jelölje x_r a fa gyökerét. Ekkor az $F_1 = F_{bal(x_r)}$ fa a $\langle k_1, \ldots, k_{r-1} \rangle$ kulcsokat, az $F_2 = F_{jobb(x_r)}$ fa pedig a $\langle k_{r+1}, \ldots, k_n \rangle$ kulcsokat tartalmazza. Mivel



6. ábra. Ha az optimális fa gyökerében a k_r kulcs van.

$d_{F_1}(p) = d_F(p) + 1$ és $d_{F_2}(p) = d_F(p) + 1$.

$$
\begin{aligned}
V(F) &= \sum_{i=1}^{n} p_i\, d_F(x_i) + \sum_{i=0}^{n} q_i\, d_F(y_i) \\
&= \sum_{i=1}^{r-1} p_i\, d_F(x_i) + \sum_{i=0}^{r-1} q_i\, d_F(y_i) + p_r + \sum_{i=r+1}^{n} p_i\, d_F(x_i) + \sum_{i=r}^{n} q_i\, d_F(y_i) \\
&= \sum_{i=1}^{r-1} p_i\,(d_{F_1}(x_i)+1) + \sum_{i=0}^{r-1} q_i\,(d_{F_1}(y_i)+1) + p_r \\
&+ \sum_{i=r+1}^{n} p_i\,(d_{F_2}(x_i)+1) + \sum_{i=r+1}^{n} q_i\,(d_{F_2}(y_i)+1) \\
&= \sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i + \sum_{i=1}^{r-1} p_i\, d_{F_1}(x_i) + \sum_{i=0}^{r-1} q_i\, d_{F_1}(y_i) \\
&+ \sum_{i=r+1}^{n} p_i\, d_{F_2}(x_i) + \sum_{i=r}^{n} q_i\, d_{F_2}(y_i) \\
&= \sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i + V(F_1) + V(F_2)
\end{aligned}
$$

Tehát

$$
V(F) = \sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i + V(F_1) + V(F_2) \tag{2}
$$

Az F_1 fa a $\langle k_1, \ldots, k_{r-1}\rangle$ kulcsokat tartalmazó optimális bináris keresőfa a $\langle p_1, \ldots, p_{r-1}\rangle$ sikeres és $\langle q_0, \ldots, q_{r-1}\rangle$ sikertelen keresési gyakoriságokra, az F_2 fa pedig $\langle k_{r+1}, \ldots, k_n\rangle$ kulcsokat tartalmazó optimális bináris keresőfa a $\langle p_{r+1}, \ldots, p_n\rangle$ sikeres és $\langle q_r, \ldots, q_n\rangle$ sikertelen keresési gyakoriságokra. A bizonyítás a kivágás-és-beillesztés módszerrel végezhető. Ha lenne olyan $\overline{F}_1$ bináris keresőfa a $\langle p_1, \ldots, p_{r-1}\rangle$ sikeres és $\langle q_0, \ldots, q_{r-1}\rangle$ sikertelen keresési gyakoriságokra, hogy $V(\overline{F}_1) < V(F_1)$, akkor az F fában F_1 helyett az $\overline{F}_1$ részfát véve olyan fát kapnánk a $\langle p_1, \ldots, p_n\rangle$ sikeres és $\langle q_0, \ldots, q_n\rangle$ sikertelen keresési gyakoriságokra, amelynek költsége $\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i + V(\overline{F}_1) + V(F_2) < V(F)$. Ugyanígy bizonyítható, hogy F_2 is optimális fa a $\langle k_{r+1}, \ldots, k_n\rangle$ kulcsokra a $\langle p_{r+1}, \ldots, p_n\rangle$ sikeres és $\langle q_r, \ldots, q_n\rangle$ sikertelen keresési gyakoriságokra.

Részproblémákra bontás.

Minden (i, j) indexpárra $0 \le i \le j \le n$ tekintsük azt a részproblémát hogy mi az optimális bináris keresőfa az $\langle p_{i+1}, \ldots, p_j\rangle$ sikeres és $\langle q_i, \ldots, q_j\rangle$ sikertelen keresési gyakoriságokra. Jelölje $Opt(i, j)$ az optimális fa költségét az (i, j) részproblémára.

Az optimális megoldás értékének rekurzív kiszámítása.

Vezessük be a

$$
W(i, j) = \sum_{u=i+1}^{j} p_u + \sum_{u=i}^{j} q_u
$$

jelölést.

Minden (i, j)-re a (2) képlet miatt biztosan létezik olyan $i < r \le j$, hogy
$Opt(i, j) = W(i, j) + Opt(i, r-1) + Opt(r, j)$, csak azt nem tudjuk, hogy melyik r-re. Tehát azt az r-et keressük, amelyre a fenti összeg minimális lesz. Tehát $Opt(i, j)$ a következő rekurzív összefüggéssel számítható.

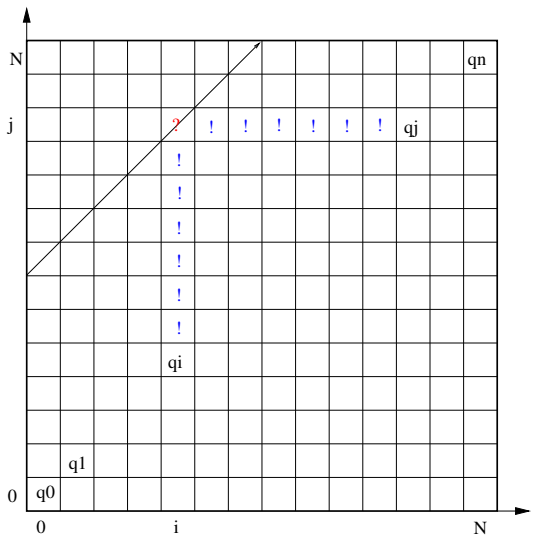
$$
Opt(i, j) = \begin{cases} q_i & \text{ha} \quad i = j \\ W(i, j) + \min\limits_{i<r\le j} (Opt(i, r-1) + Opt(r, j)) & \text{ha} \quad i < j \end{cases} \tag{3}
$$

Az összetevők és a kiszámítási sorrend meghatározása.

Az (i, i) részproblémáknak nincs összetevőjük, mert $Opt(i, i) = q_i$.
Az (i, j), $i < j$ részprobléma összetevői az $(i, r-1)$ és (r, j), $r = i+1, \ldots, j$ részproblémák.
Tehát a táblázatot ki tudjuk tölteni átlósan haladva, a m-edik átlóban $m = 1, \ldots, n$ azon
(i, j)részproblémákat számítjuk, amelyekre $j - i = m$.

Kiszámítás alulról-felfelé haladva (táblázatkitöltés).

Ahhoz, hogy egy optimális megoldást elő tujunk állítani, minden (i, j) részproblémára tároljuk egy G táblázat $G[i, j]$-elemében



7. ábra. Táblázatkitöltési sorrend

azt az r értéket, amelyre a (3) képletben az minimum előáll. Ez az r lesz a $\langle k_{i+1}, \ldots, k_j \rangle$ kulcsokat tartalmazó optimális bináris keresőfa gyökere. A $G[i, j]$ értékeket felhasználva a FASIT rekurzív eljárás állítja elő ténylegesen az algoritmus kimenetét jelentő keresőfát.

```
{ Globális programelemek az OptBKfa eljáráshoz :}
Const
  MaxN = ???    ;                  { a kulcsok max. száma }
Type
  Kulcstip = ???;                  { a kulcsok típusa }
  Index = 1..MaxN;
  Vektor = Array[Index] Of Real;    {a sikeres keresési gyakoriságok}
  Vektor1 = Array[0..MaxN] Of Real; {a sikertelen keresési gyakoriságok}
  Fa = Array[Index] Of Record       {a bináris keresőfa ábrázolása}
                         bal, jobb : 0..MaxN;
                         kulcs : kulcstip;
                         {egyéb mezők}
                       End;

Procedure OptBKfa(Const    P : Vektor;
                  Const    Q : Vektor1;
                           N : Index;
                  Var Gyoker : Index;
                  Var      F : Fa       );
 { P[i]:az i-edik halmazelem keresési gyakorisága          }
 { Q[i]:az i-edik es az i+1-edik almazelem közé eső elemek }
 {      keresési gyakorisága                               }
 { Gyoker:az optimális keresőfa gyökérpontjának indexe     }
 { F     :az optimális bináris keresőfa}
```

```
Var
  Opt: Array[0..MaxN, 0..MaxN] Of Real;
  { Opt[i,j] az i+1..j elemeket tartalmazó OBK költsége }
  W: Array[0..MaxN, 0..MaxN] Of Real;
  { W[i,j]= Q[i]+Sum(P[k]+Q[k]: k:=i+1..j }
  G : Array[0..MaxN, 0..MaxN] Of 0..MaxN;
  {G[i,j] az i+1..j elemeket tartalmazó optimális bináris keresőfa
   gyökérpontjának indexe }
  i,j,r,m,optr : Integer; optV, V : Real;

  Procedure Fasit(Apa, i, j : Integer);
  { Előallítja az i+1..j elemek OB keresőfáját a G értékekből}
  { Globális: G, F}
    Begin{Fasit}
      If Apa <> 0 Then Begin
        F[Apa].bal := G[i, Apa-1];
        F[Apa].jobb := G[Apa, j];
        Fasit(G[i, Apa-1], i, Apa-1);
        Fasit(G[Apa, j], Apa, j)
      End
    End{Fasit};

Begin{OptBKfa}
  For i := 0 To N-1 Do Begin    { inicializálás }
    W[i,i]:=Q[i];
    G[i,i]:=0;
    Opt[i,i]:=Q[i];
  End;
  W[N,N]:=Q[N]; G[N,N]:=0; Opt[N,N]:=Q[N];
  For m := 1 To N Do {m=j-i}
    For i := 0 To N-m Do Begin{ Opt(i,j) számítása }
      j := i+m;
      W[i,j] := W[i,j-1]+P[j]+Q[j];
      optr := j; optV := Opt[i,j-1]+Q[j]{=Opt[j,j]};
      For r := i+1 To j-1 Do Begin
        V := Opt[i,r-1]+Opt[r,j];
        If V < optV Then Begin
          optV := V; optr := r
        End
      End{for r};
      Opt[i,j] := W[i,j]+optV; G[i,j] := optr
    End{i};
  Gyoker := G[0,N];
  Fasit(Gyoker, 0, N)
End{OptBKfa};
```

A OPTBKFA algoritmus futási ideje $\Theta(n^3)$.
Bizonyítás nélkül megjegyezzük, hogy a

```
      For r := i+1 To j-1 Do Begin
```

ciklusban elegendő lenne az `r` ciklusváltozót `G[i,j-1]+1` -től `G[i+1,j]`-ig futtani, és ezzel az algoritmus futási ideje $\Theta(n^2)$ lenne.