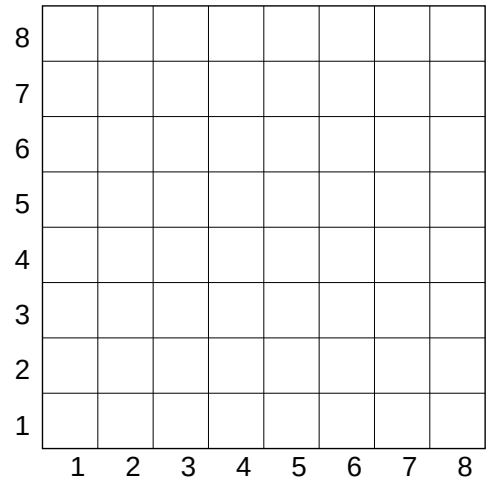
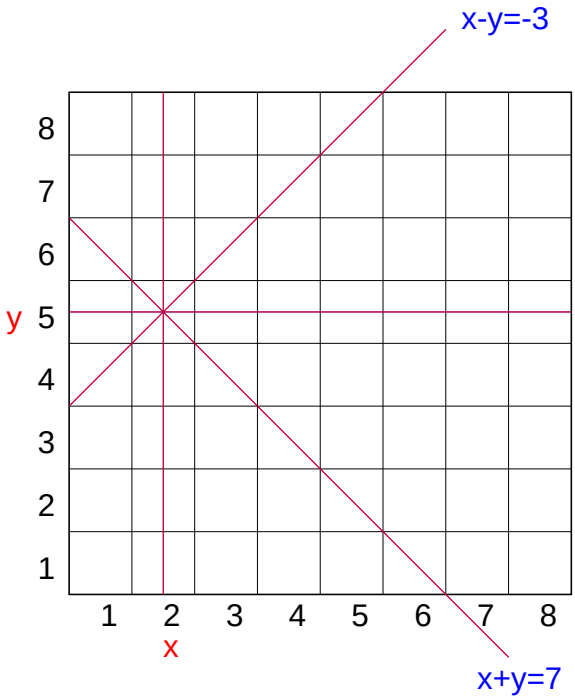
9. Megoldás keresése visszalépéssel (backtracking)

Probléma: *n*-királynő
Helyezzünk el az *nxn*-es sakktáblán *n* királynőt, hogy egyik se üsse a másikat!



1. ábra. Üres tábla

A megoldás megadható annak az *n* mezőnek a koordinátáival, amelyekre királynőket helyezzük: $M = \{(x_1,y_1),\cdots,(x_n,y_n)\}$. Tehát az (x_i,y_i) és (x_j,y_j) mezőkre helyezett két királynő akkor és csak akkor nem üti egymást, ha



2. ábra. Az (x,y) mezőn lévő királynő ütési mezői.

$$x_i \neq x_j \tag{1}$$
$$y_i \neq y_j \tag{2}$$
$$x_i - y_i \neq x_j - y_j \tag{3}$$
$$x_i + y_i \neq x_j + y_j \tag{4}$$

Tehát egy ilyen M halmaz akkor és csak akkor megoldása a feladatnak, ha $(\forall i, j)(1 \leq i < j \leq n)$teljesül a fenti 4 feltétel. Minden sorban, minden oszlopban pontosan egy királynőnek kell lenni, továbbá minden főátlóban és minden mellékátlóban legfeljebb egy királynő lehet. Tehát minden megoldás megadható egy $X = \langle x_1, \ldots, x_n \rangle$ vektorral, ami azt jelenti, a királynőket az $\{(x_1, 1), \cdots, (x_n, n)\}$ mezökre helyezünk királynőtket. Ekkor a megoldás feltétele:$(\forall i, j)(1 \leq i < j \leq n)$

$$x_i \neq x_j \tag{5}$$
$$x_i - i \neq x_j - j \tag{6}$$
$$x_i + i \neq x_j + j \tag{7}$$

9.1. Kimerítő keresés (nyers erő) módszere

Elvi algoritmus:
KIMERITOKERESES

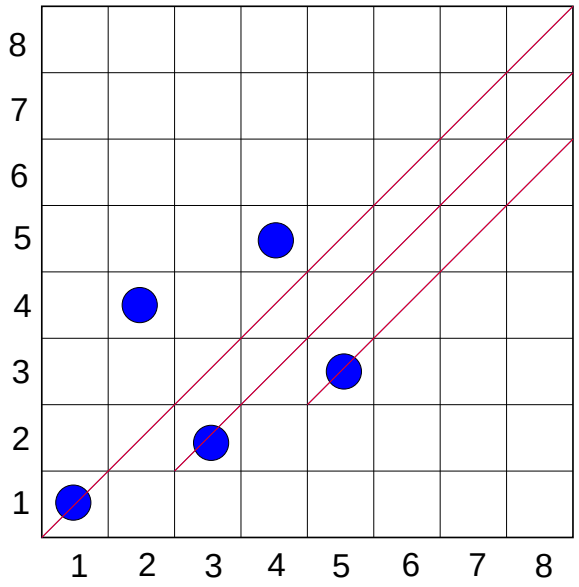
$\forall X = \langle x_1, \ldots, x_n \rangle \in [n] \times \ldots \times [n]$ do
    if Megoldás(X) then
        KiIr(X)

A KIMERITOKERESES algoritmus megvalósítása:

```
Const
  MaxN=100;
Type
  Index=1..MaxN;
  Vektor=Array[Index] of Index;
Var
  N:Index;
  X:Array[Index] of Index;

Function Megoldas(X:Vektor):Boolean;
Begin End;
Procedure KiIr(X:Vektor);
Begin End;

Procedure KimeritoKereses(k:Integer);
 {Globál: X, N}
 {A megoldásvektor <X[1],...,X[k-1]>komponenseit már beállítottuk}
Var
  i:Index;
Begin{KimeritoKereses}
  For i:=1 To N Do Begin
    X[k]:=i;
    If k=N Then Begin
      If Megoldas(X) Then
        KiIr(X)
    End Else {k<N}
      KimeritoKereses(k+1);
  End{for i};
End{KimeritoKereses};
```

```
Begin
 {Beolvasás}
 KimeritoKereses(0);
End.
```

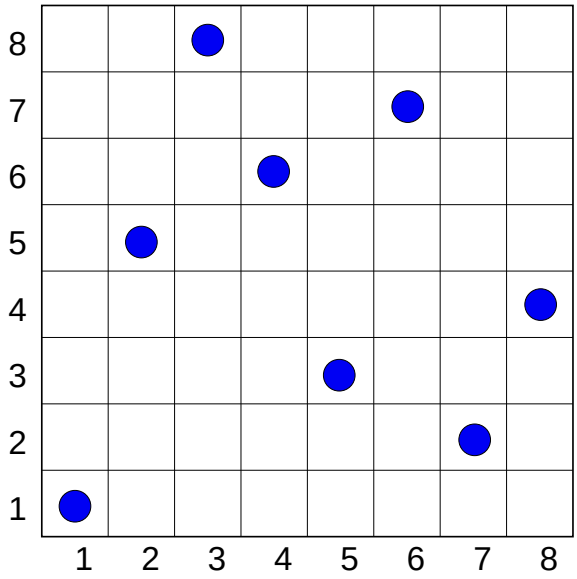
Nyilvánvaló, hogy ez a módszer sok vektort feleslegesen vizsgál, hiszen ha $x_1 = x_2$, akkor X biztosan nem lehet megoldás. Azaz, elegendő lenne csak a permutációkat vizsgálni.
Ötlet: probáljuk a megoldást lépésenként előállítani úgy, hogy ha már elhelyeztünk a tábla első $k-1$ sorában királynőket úgy, hogy egyik sem üti a másikat, akkor a következő lépésben a k-adik sorba próbáljunk rakni egy királynőt.
Megoldáskezdemény:
$X = \langle x_1, \ldots, x_k \rangle$, $0 \leq k \leq n, 1 \leq x_i \leq n$
X jó megoldáskezdemény, ha kezdőszelete lehet egy megoldásnak, tehát nem üti egymást a táblára már elhelyezett k darab királynő, azaz:
$(\forall i, j)(1 \leq i < j \leq k)$
$(x_i \neq x_j) \wedge (x_i - i \neq x_j - j) \wedge (x_i + i \neq x_j + j)$



3. ábra. Nem folytatható állás (megoldáskezdemény)

$V = \langle 1,4,2,5,3 \rangle$ jó megoldáskezdemény, de nem folytatható, mert a 6. sorba nem helyezhető királynő, hogy ne üsse a már táblán lévők egyikét sem.
Visszalépést kell végezni: az 5. sorban lévőt más helyre kell rakni.
Ez az a pont, ahol ez a módszer különbözik a mohó stratégiáttól. Ott a megoldáskezdemény mindig folytatható volt, és meg tudtuk mondani, hogy melyik lépéssel.
Itt azonban nem tudjuk megmondani, hogy egy megoldáskezdemény folytathaztó-e, és ha igen milyen lépéssel.

```
Program Kiralynokr; {Az N-királynő probléma rekurzív megoldása}
  Const
    N  = 8       ;{ a tábla mérete }
    N1 = N-1     ;{ a mellékátlok indexei -N1..N1 }
    N2 = 2*N     ;{ a főátlók indexei 2..N2 }
  Type
    Index = 1..N;
    Vektor=Array[Index] of Index;
  Var
    X : Vektor;
    Oszlop : Array [Index] Of Boolean;
    Fatlo  : Array [-N1..N1] Of Boolean;
```



4. ábra. Az első megtalált megoldás

```
    Matlo  : Array [2..N2] Of Boolean;
    i,j : Integer;

Procedure KiIr(V:Vektor);
  Var i:Integer;
  Begin
   For i:=1 To N Do Write(V[i]:3);
   WriteLn;
  End;

Function Szabad(i,k : Index) : Boolean;
 {Az (i,k) mezőre helyezhető-e királynő?}
Begin
  If Oszlop[i]  And
     Fatlo[i-k] And
     Matlo[i+k] Then
  Begin {az elhelyezés bejegyzése}
    Szabad := True;
    Oszlop[i] := False;  {i. oszlop foglalt}
    Fatlo [i-k] := False;{i-k. főátló foglalt}
    Matlo [i+k] := False {i+k. melléátló foglalt}
  End Else
    Szabad := False
End {Szabad};

Procedure Torol(i,k : Index); Begin
  Oszlop[i] := True;
  Fatlo [i-k] := True;
  Matlo [i+k] := True
End{Torol};

Procedure Lepes(k : Index); {Global: X, N}
 Var i : Integer;
 Begin{<X[1],...,X[k-1]> jó megoldáskezdemény}
  For i := 1 To N Do            {miden lehetséges választásra ... }
```

```
    If Szabad(i,k) Then Begin   {rakható-e az (i,k) mezőre királynő?}
      X[k]:=i;                  {a lépés bejegyzése}
      If k = N Then             {találtunk egy megoldás}
        KiIr(X)                 {kiíratjuk}
      Else Begin
        Lepes(k+1);             {továbblépés}
        Torol(i,k)              {visszalépés: a bejegyzés törlése}
      End{else:k<N}
    End{if}
End{Lepes};
Begin{program}
  For i := 1 To N Do Begin      {inicializálása}
    Oszlop[i] := True;          {minden oszlop szabad}
    For j := 1 To N Do Begin
      Fatlo[i-j] := True;       {minden főátló szabad}
      Matlo[i+j] := True        {minden mellékátló szabad}
    End{for j}
  End{for i};
  Lepes(1)                      {a keresés indítása}
End.
```

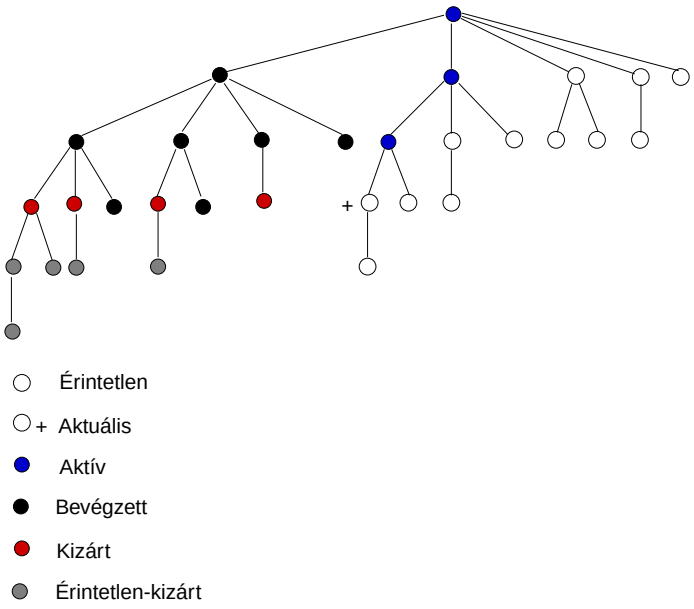
Nemrekurzív algoritmus

```
Program Kiralynokr;
{Az N-királynő probléma nemrekurzív megoldása}
  Const
    N  = 8       ;{ a tábla mérete }
    N1 = N-1     ;{ a mellékátlok indexei -N1..N1 }
    N2 = 2*N     ;{ a főátlók indexei 2..N2 }
  Type
    Index = 1..N;
    Vektor=Array[Index] of Index;
  Var
    X : Vektor;
    Oszlop : Array [Index] Of Boolean;
    Fatlo  : Array [-N1..N1] Of Boolean;
    Matlo  : Array [2..N2] Of Boolean;
    i,j : Integer;

Procedure KiIr(V:Vektor);
  Var i:Integer;
  Begin
   For i:=1 To N Do Write(V[i]:3);
   WriteLn;
  End;

Function Szabad(i,k : Index) : Boolean;
 {Az (i,k) mezőre helyezhető-e királynő?}
Begin
  If Oszlop[i]  And
     Fatlo[i-k] And
     Matlo[i+k] Then
  Begin {az elhelyezés bejegyzése}
    Szabad := True;
    Oszlop[i] := False;  {i. oszlop foglalt}
```

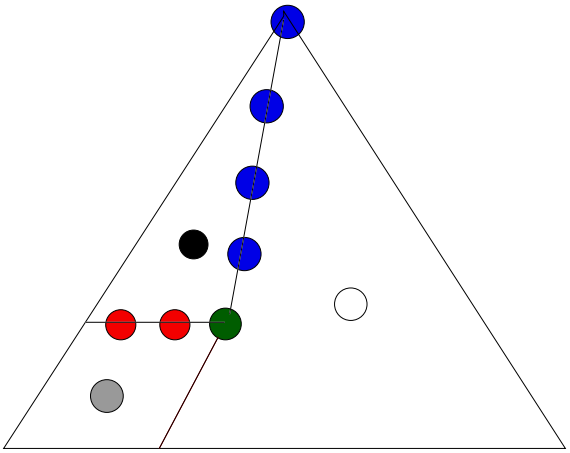
```
    Fatlo [i-k] := False;{i-k. főátló foglalt}
    Matlo [i+k] := False {i+k. melléátló foglalt}
  End Else
    Szabad := False
End {Szabad};

Procedure Torol(i,k : Index); Begin
  Oszlop[i] := True;
  Fatlo [i-k] := True;
  Matlo [i+k] := True
End{Torol};

Procedure Keres(Var X:Vektor; Var Van:Boolean); {Global: N}
 Type Irany=(Ujpont, Tovabb, Vissza);
 Var  k : Integer; Merre:Irany;
Begin{Keres}
  k:=1; X[1]:=1; Merre:=Ujpont; Van:=False;
  While True Do
    Case Merre of
      Ujpont: If Szabad(X[k], k) Then Begin
                If k=N Then Break;  { megoldást találtunk, vége}
                Inc(k);             { átlépés a következő sorba}
                X[k]:=1             { az első mezővel próbálkozunk}
              End Else              { X nem jó kezdemény }
                Merre:=Vissza;
      Tovabb: Begin                 { jó kezdeményről kell továbblépni}
                Torol(X[k], k);     { a bejegyzés törlése}
                If X[k]<N Then Begin{ ha nem sorvégén vagyunk}
                  Inc(X[k]);        { továbblépés a sorban }
                  Merre:=Ujpont
                End Else If k>1 Then{ visszalépés az előző sorba}
                  Dec(k)            { Merre nem változik}
                Else
                  Exit              { nem lehet visszalépni, vége}
              End;

      Vissza: If  X[k]<N Then Begin { ha van sorvégén vagyunk}
                Inc(X[k]);          { továbblépés a sorban }
                Merre:=Ujpont
              End Else If k>1 Then Begin{ visszalépés az előző sorba}
                Dec(k);
                Merre:=Tovabb       { X[1..k] jó kezdemény}
              End Else              { nem lehet visszalépni, vége}
                Exit
    End{case};
  Van:=True;
End{Keres};
Var Van:Boolean; Begin
  For i := 1 To N Do Begin         {inicializálása}
    Oszlop[i] := True;
    For j := 1 To N Do Begin
      Fatlo[i-j] := True;
      Matlo[i+j] := True
    End{for j}
  End{for i};
  Keres(X,Van);
```

```
  If Van Then KiIr(X);
End.
```

Vegyük észre, hogy a megoldáskezdemények fát alkotnak. Egy $X = \langle x_1, \ldots, x_k \rangle$ megoldáskezdemény lehetséges közvetlen folytatásai, azaz X fiai az $Y = \langle x_1, \ldots, x_k, i \rangle$ $i = 1, \ldots, n$ lehetséges megoldáskezdemények.
Tehát minden fabejáró algoritmus alkalmazható megoldás keresésére, azzal a módosítással, hogy ha az aktuális X pont (megoldáskezdemény) nem választható, azaz kizárt lesz, akkor X-et úgy kell tekinteni a bejárás során, mint ha levél pont lenne.
A megoldás keresését meg tudjuk fogalmazni olyan általános formában, hogy az algoritmus érdemi része, azaz a megoldástér bejárása csak néhány problémaspecifikus műveletet alkalmaz.
Ezt módszert "application framework" módszernek is nevezik. Adott problémára nem kell újraírni az érdmi részt, csak a problémaspecifikus műveletek megvalósítását kell megadni.



5. ábra. A megoldástér pontjainak osztályozása visszalépéses keresésnél



6. ábra. A megoldástér sematikus képe visszalépéses keresésnél

```
{ Probléma-specifikus műveletek: }
  Procedure UresX(Var X:MTer);Forward;
    { X az üres megoldáskezdemény lesz }
```

```
  Function EFiu(Var X: MTer): Boolean; Forward;
    { Ha van X-nek fia, akkor X az első fiúra változik és a
     függvényhívás értéke True, egyébként False és X nem változik. }
  Function Testver(Var X: MTer): Boolean; Forward;
   { Ha van X-nek még benemjárt testvére, akkor X a következő testvér lesz
     és a függvényhívás értéke True, egyébként False és X nem változik.}
  Function Apa(Var X: MTer):Boolean;  Forward;
    { Ha van X-nek apja, akkor X az apjára változik és a
     függvényhívás értéke True, egyébként False és X nem változik. }
  Procedure VisszaAllit(Var X:MTer);Forward;
    { Törli a az aktívvá tételkor tett bejegyzéseket,
      felszabadítja az esetleg foglalt memóriát }

  Function Megoldas (Var X: MTer): Boolean; Forward;
    { akkor és csak akkor ad True értéket, ha X megoldása a problémának.}
  Function LehetMego(Var X: MTer): Boolean; Forward;
    { Ha LehetMego(X) hamis, akkor nincs megoldás az X gyökerű részfában. }
    { Ha LehetMego(X) igaz, abból nem következik, hogy van is megoldás.   }
    { Olyan bejegyzéseket is tehet, amelyek a további LehetMego és Megoldás
      műveletek gyorsabb elvégzését segítik. }

Procedure RKeres(X:MTer);
 {Megoldás rekurziv keresése az X-gyökerű megoldástér részfában }
 {Global: X0; egy megoldás, Van=True<=>talált megoldást }
 {X biztosan jó megoldáskezdemény, LehetMego(X) igaz}
Begin{RKeres}
  If Megoldas(X) Then Begin
    X0:=X;                     { KiIr(X) ha az összes megoldást keressük}
    Van:=True; Exit            { egy megoldás keresése esetén}
  End;
  If Not EFiu(X) Then Exit;    { átlépés az első fiúra, ha van}
  Repeat                       { a fiúk bejárása }
    If Not LehetMego(X) Then   { X kizárt pont-e? }
      Continue;                { ha igen, kihagyjuk }
    RKeres(X);                 { az X gyökerű részfa rekurzív bejárása}
    VisszaAllit(X);            { visszalépés: a bejegyzés törlése}
    If Van Then                { egy megoldás keresése esetén kilépés, ha}
      Exit;                    { volt megoldás az X-gyökerű részfában }
  Until Not Testver(X)         { átlépés a következő testvérre, ha van }
End{RKeres};
{************************************************************************}

Procedure Keres(Var X:MTer; Var Van:Boolean);
 { Bemenet: X a megoldástér fájának gyökere}
 { Kimenet: Van=True<=>ha van megoldás és X egy megoldás lesz}
 Type
  Paletta=(Feher, Kek, Piros);
Var
 Szin: Paletta;
Begin{Keres}
  Szin:=Feher; Van:=False;

  While True Do
    Case Szin of
      Feher: If LehetMego(X) Then Begin  { X jó megoldáskezdemény?}
               If Megoldas(X) Then Break;{ egy megoldás keresése esetén}
```

```
              If Not EFiu(X) Then       { majd a testvérére kell átlépni}
                Szin:=Kek               { X még aktív marad}
            End Else                    { X kizárt pont lett}
              Szin:=Piros;
      Kek:  Begin
              VisszaAllit(X);           { a bejegyzések törlése}
              If Testver(X) Then        { átlépés a testvére}
                Szin:=Feher
              Else If Not Apa(X) Then   { visszalépés az apára}
                Exit                    { X a megoldástér gyökere, kész}
            End;                        { egyébként X:=Apa(X) és marad Kék}
      Piros: If Testver(X) Then         { a testvér lesz az új akt.pont}
              Szin:=Feher
            Else If Apa(X) Then         { visszalépés az apára, ha van}
              Szin:=Kek                 { X apja mindig aktív}
            Else                        { X a gyökér, vége a keresésnek}
              Exit
    End{case};
  Van:=True;
End{Keres};
```

A probléma-specifikus műveletek megvalósítása az n-királynő problémához.

```
Const MaxN=32; Type
  Index = 1..MaxN;
  Vektor=Array[Index] of 0..MaxN;
  MTer=Record {a megoldástér pontjainak ábrázolása}
         N:Index;
         V:Vektor;
         k:Integer;
         Oszlop : Array [Index] Of Boolean;
         Fatlo  : Array [-(MaxN-1)..MaxN-1] Of Boolean;
         Matlo  : Array [2..2+MaxN] Of Boolean;
       End;
Procedure UresX(Var X:MTer); Var i,j:Integer;
Begin
  With X Do Begin
    k:=0;
    For j:=1 To N Do             {a tabla inicializalasa }
      Oszlop[j]:=True;           {minden sor szabad}
    For j:=-(N-1) To N-1 Do      {minden foatlo szabad}
      Fatlo[j]:=True;
    For j:=2 To 2*N Do           {minden mellekatlo szabad}
      Matlo[j]:=True;
  End;
End{UresX};

Function EFiu(Var X: MTer): Boolean;
Begin
  If X.k<X.N Then Begin
    Inc(X.k);
    X.V[X.k]:=1;
    EFiu:=True
  End Else
    EFiu:=False;
End{EFiu};
```

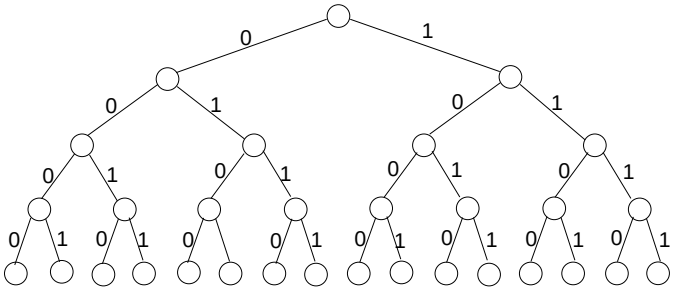
```
Function Testver(Var X: MTer): Boolean;
Begin
  If (X.k>0)And(X.V[X.k]<X.N) Then Begin
    Testver:=True;
    Inc(X.V[X.k]);
  End Else
    Testver:=False;
End{Testver};

Function Apa(Var X: MTer):Boolean;
Begin
  If X.k>0 Then Begin
    Apa:=True;
    Dec(X.k);
  End Else
    Apa:=False;
End{Apa};

Procedure VisszaAllit(Var X:MTer); Begin
  With X Do Begin
    If k=0 Then Exit;
    Oszlop[V[k]]  := True;
    Fatlo [V[k]-k]:= True;
    Matlo [V[k]+k]:= True
  End;
End{VisszaAllit};

Function LehetMego(Var X:MTer) : Boolean; Begin
  LehetMego:= True;
  With X Do Begin
    If k=0 Then Exit;
    If Oszlop[V[k]]   And
       Fatlo[V[k]-k] And
       Matlo[V[k]+k] Then
    Begin
      Oszlop[V[k]] := False;
      Fatlo [V[k]-k] := False;
      Matlo [V[k]+k] := False
    End Else
      LehetMego:= False
  End
End {LehetMego};

Function Megoldas(Var X: MTer): Boolean; Begin
  Megoldas:=X.k=X.N
End{Megoldas};
```

9.2. A visszalépéses keresés alkalmazása a pénzváltás problémára.

Probléma: Pénzváltás
Bemenet: $P = \{p_1, ..., p_n\}$ pozitív egészek halmaza, és E pozitív egész szám.
Kimenet: Olyan $S \subseteq P$, hogy $E = \sum_{p \in S}$
A megoldást kifejezhetjük és kereshetjük bitvektor formában, tehát olyan $X = \langle x_1, \ldots, x_n \rangle$ vektort keresünk, amelyre

$$E = \sum_{i=1}^{n} x_i \, p_i$$

Ekkor a megoldástér fája bináris fa lesz. A megoldást kifejezhetjük és kereshetjük mint a pénzek indexeinek olyan $S \subseteq \{1,\ldots,n\}$

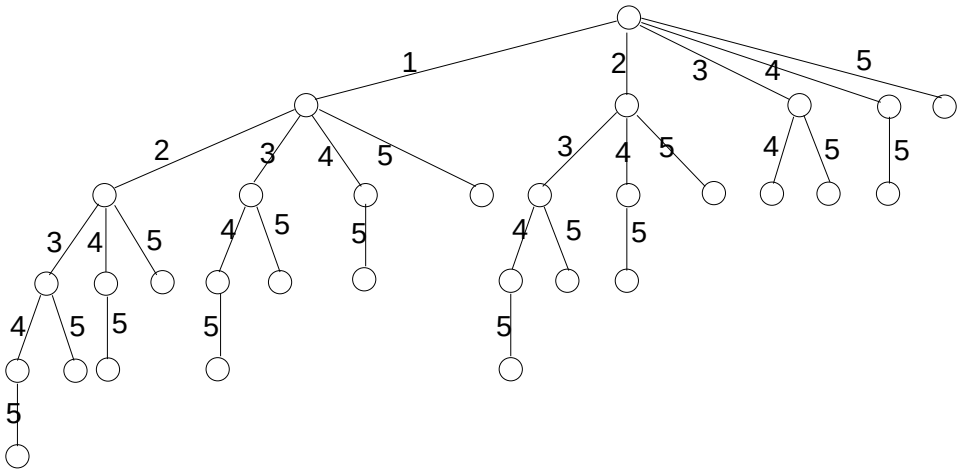


7. ábra. Bináris megoldástér a pénzváltás probléma $n = 4$ esetében

halmazának $X = \langle i_k,\ldots,i_m\rangle$ növekvő felsorolásáként is, azaz $i_1 < i_2 < \ldots < i_m$, hogy .

$$E = \sum_{k=1}^{m} p_{i_k}$$

Ekkor a megoldástér formája a 8. ábrán látható $n = 5$ esetére. A pénzváltás probléma megoldásához elegendő megadni a



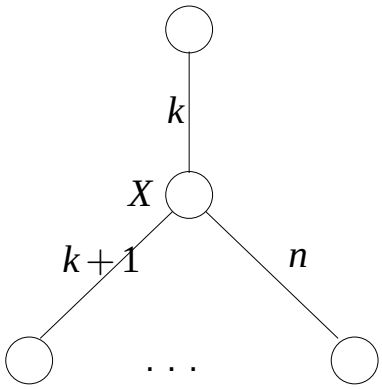
8. ábra. Nem bináris megoldástér a pénzváltás probléma $n = 5$ esetében

probléma-specifikus URESX, EFIU, TESTVER, (APA,) VISSZAALLIT, LEHETMEGO, MEGOLDAS műveletek megvalósítását, és az RKERES (KERES) eljárás változtatás nélkül alkalmazhaztó egy megoldás előállítására.
(Az APA művelet csak a nemrekurzív keresés esetén kell.)
Legyen $X = \langle i_k,\ldots,i_m\rangle$ tetszőleges megoldáskezdemény.
EFIU(X) $= \langle i_1,\ldots,i_m,i_m+1\rangle$, ha $i_m < n$
TESTVER(X) $= \langle i_1,\ldots,i_{m-1},i_m+1\rangle$, ha $i_m < n$
APA(X) $= \langle i_1,\ldots,i_{m-1}\rangle$, ha $m > 0$
LEHETMEGO(X) akkor és csak akkor adjon igaz értéket, ha

$$\sum_{k=1}^{m} p_{i_k} \le E \wedge \sum_{k=1}^{m} p_{i_k} + \sum_{j=i_m+1}^{n} p_j \ge E$$

MEGOLDAS(X) akkor és csak akkor adjon igaz értéket, ha

$$E = \sum_{k=1}^{m} p_{i_k}$$



9. ábra. A fa pontjai

A VISSZAALLIT művelet megvalósítása előtt dönteni kell, hogy milyen segéd információt tárolunk egy megoldástérpontban. Célszerű tárolni a

$$Resz = \sum_{k=1}^{m} p_{i_k}$$

$$Maradt = \sum_{j=i_m+1}^{n} p_j$$

összegeket, hogy a LEHETMEGO(X) és MEGOLDAS(X) műveleteket konstans időben ki tudjuk számítani.

```
Const
  MaxN=100; {a pénzek max. száma}
Type
  Index = 1..MaxN;
  MTer=Record
        E:Longint;
        P:Array[Index] of Word; {a pénzek}
        N:Index;       {az összes pénz száma}
        V:Array[0..MaxN] of Word; {a megoldáskezdemény}
        k:Integer;     {a megoldáskezdemény pénzeinek száma}
        Resz:Longint;  {a megoldáskezdemény pénzeinek összege}
        Maradt:Longint;{a még választható pénzek összege}
      End;

Procedure UresX(Var X:MTer);
Var i,j:Integer;
Begin
  With X Do Begin
    k:=0;
    V[0]:=0;
    Maradt:=0;
    Resz:=0;
    For i:=1 To N Do Maradt:=Maradt+P[i];
  End;
End;

Function EFiu(Var X: MTer): Boolean;
Begin
  If X.V[X.k]<X.N Then Begin
    Inc(X.k);
    X.V[X.k]:=X.V[X.k-1]+1;
```

```
    EFiu:=True
  End Else
    EFiu:=False;
End;

Function Testver(Var X: MTer): Boolean;
Begin
  If (X.k>0)And(X.V[X.k]<X.N) Then Begin
    Testver:=True;
    Inc(X.V[X.k]);
  End Else
    Testver:=False;
End;

Function Apa(Var X: MTer):Boolean;
Begin
  If X.k>0 Then Begin
    Apa:=True;
    Dec(X.k);
  End Else
    Apa:=False;
End;
Procedure VisszaAllit(Var X:MTer);
Begin
  With X Do Begin
    If k=0 Then Exit;
    Resz := Resz-P[V[k]];
    Maradt := Maradt+P[V[k]];
  End;
End{VisszaAllit};

Function LehetMego(Var X:MTer) : Boolean;
Begin
  LehetMego:= True;
  With X Do Begin
    If k=0 Then Exit;
    If (Resz+V[k] <= E) And (Resz+Maradt>=E)
    Then Begin
      Resz:=Resz+P[V[k]];
      Maradt:=Maradt-P[V[k]];
    End Else
      LehetMego:= False
  End
End {LehetMego};

Function Megoldas (Var X: MTer): Boolean;
Begin
  Megoldas:=X.Resz=X.E
End;
```

Visszalépésses keresési algoritmusok futási ideje

Legrosszabb esetben a kersés a megoldástér-fa minden pontját bejárja, ami exponenciális. **Visszalépésses keresési algoritmusok tárigénye**

A rekurzív megvalósítás során minden aktív pont tárolódik.

Nemrekurzív megvalósítás során elég csak csak az aktuális pontot tárolni.

A visszalépéses keresés alkamazható optimális megoldás előállítására is.

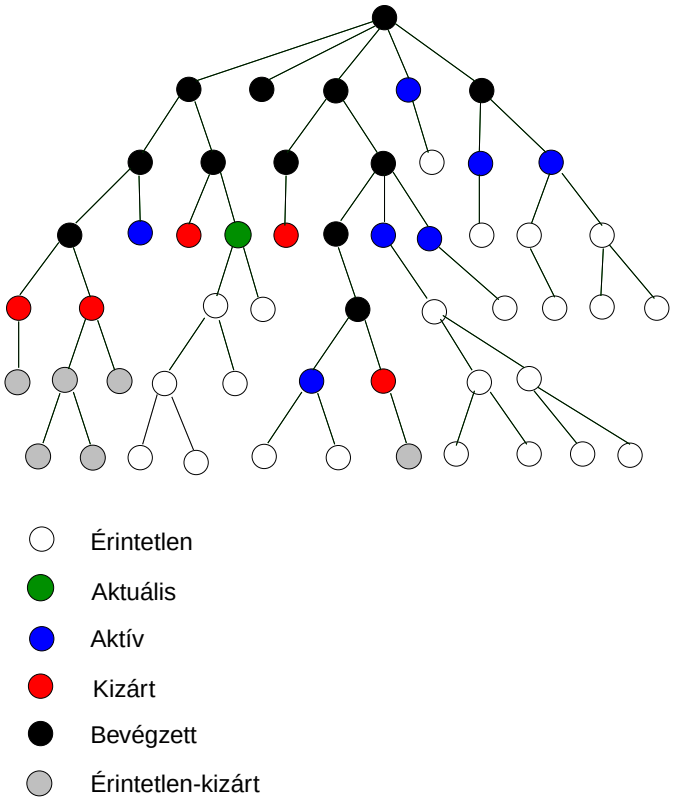
Legyen $C(X)$ a célfüggvény, tehát olyan X-et keresünk, amelyre: MEGOLDAS(X) és $C(X)$ minimális.

```
If Megoldas(X) and (C(X)<OptC) Then Begin
  X0:=X;
  OptC:=C(X)
End
```

10. Elágazás-korlátozás módszere (branch and bound)

```
{ Megoldás keresése a megoldástér adagolóval történő bejárásával }
  Type
    MTer   = ???; (* a megoldástér típusa *)
    Adagolo= ???; (* az adagoló típusa *)
{------------------------ Adagoló műveletek: --------------------------------}
  Procedure Letesit(Var A:Adagolo);
    Begin  End;
  Procedure BeTesz(Var A:Adagolo; X:MTer);
  (* {} Be(A,X) {A=Pre(A) U {X}} *)
    Begin  End{BeTesz};
  Procedure KiVesz(Var A:Adagolo; Var X:MTer);
  (* {Not Ures(A)} Ki(A,X) {Pre(A)=A U {X}} *)
    Begin  End{Kivesz};
  Function Ures(A:Adagolo):Boolean;
    Begin  End{Ures};
  Procedure Megszuntet(Var A:Adagolo);
    Begin  End{Megszuntet};

{--------------------- Probléma-specifikus műveletek:-----------------------}
  { A megoldástér bejárásához használt műveletek:}
  Procedure UresX(Var X:MTer);Forward;
    { X az üres megoldáskezdemény lesz }
  Function EFiu(Var X: MTer): Boolean; Forward;
    { Ha van X-nek fia, akkor X az első fiúra változik és a
     függvényhívás értéke True, egyébként False és X nem változik. }
  Function Testver(Var X: MTer): Boolean; Forward;
   { Ha van X-nek még benemjárt testvére, akkor X a következő testvér lesz
     és a függvényhívás értéke True, egyébként False és X nem változik.}

  Function Megoldas (Var X: MTer): Boolean; Forward;
    { akkor és csak akkor ad True értéket, ha X megoldása a problémának.}
  Function LehetMego(Var X: MTer): Boolean; Forward;
    { Ha LehetMego(X) hamis, akkor nincs megoldás az X gyökerű részfában. }
    { Ha LehetMego(X) igaz, abból nem következik, hogy van is megoldás.   }
  Function  C(Var X: MTer): Real; Forward;
   { Ha Megoldas(X) akkor C(X) az X megoldás célfüggvény értéke }
{*****************************************************************************}

Procedure AKeres(X:MTer; Var OptC:Real; Var X0:MTer);
  { A megoldás keresése az X gyökerű megoldástér-fában }
  { A megoldástér bejárása adagolóval                  }
  Var
    A:Adagolo;
  Begin (* Keres *)
    If Not LehetMego(X) Then
      Exit;                     { nem létezik megoldás }
```

```
  OptC:=Inf;
  Letesit(A);                 { üres adagoló létesítése}
  BeTesz(A,X);                { az A adagolóban csak az X pont van }
  While Not Ures(A) Do Begin  { amíg van aktív pont az adagolóban }
    KiVesz(A,X);              { egy aktív pontot kiveszünk az adagolóból }
    If Megoldas(X) And (C(X)<OptC) Then Begin { jobb megoldást talaltunk }
      OptC:=C(X);
      X0:=X;                  { a megoldás feljegyzése }
    End;
    If EFiu(X) Then           { átlépés az első fiúra, ha van; }
      Repeat                  { X összes fiának kigenerálása  }
        If LehetMego(X) Then  { X kizárt pont-e? }
          BeTesz(A,X);        { X-et betesszük az aktív pontok közé }
      Until Not Testver(X);   { átlépés a következő testvérre, ha van }
  End{while};
End (* AKeres *);
```

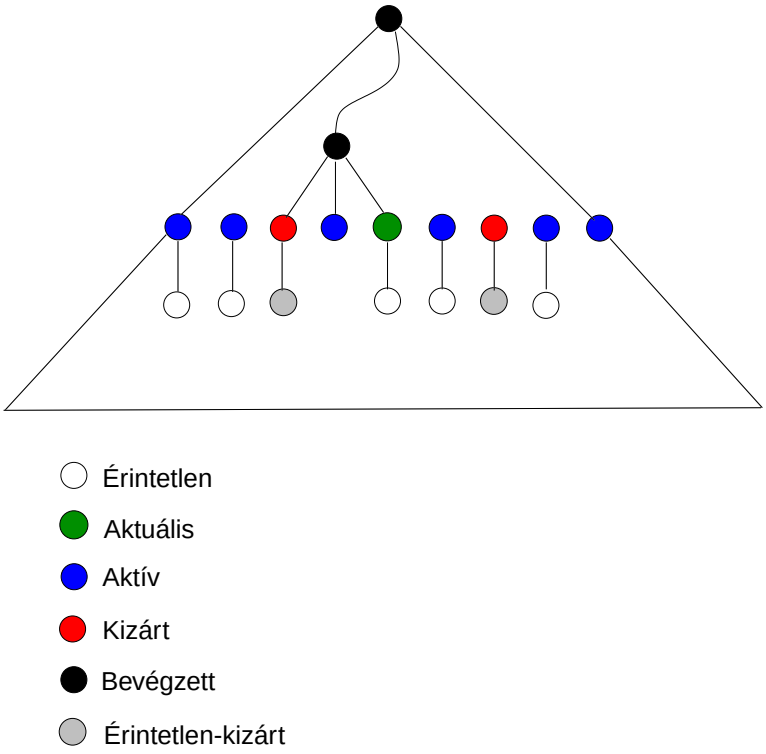


10. ábra. A megoldástér pontjainak osztályozása adagolóval történő keresés esetén

Adagolós keresés esetén mindig teljesül, hogy bármely Y érintetlen ponthoz pontosan egy olyan X aktív (vagy aktuális) pont van, hogy Y leszármazottja X-nek.
Ez a feltétel ciklusinvariáns, tehát az algortimus helyes.
Az algoritmus futási ideje sok esetben erősen függ az aktuális pont választásától. Ezen kívül további kizárásokat is tehetünk.
Tegyük fel, hogy a célfüggvényre meg tudunk fogalmazni olyan $AK(X)$ *alsókorlát* és $FK(X)$ *felsőkorlát* függvényeket, amelyekre teljesülnek az alábbi egyenlőtlenségek.
Bármely X megoldáskezdeményre és minden olyan Y megoldásra, amely leszármazottja X-nek:

$$AK(X) \leq C(Y) \leq FK(X)$$

Ekkor az adagoló lehet olyan prioritási sor, amely akár az AK alsó korlát, akár az FK felső korlát szerinti minimumos prioritási sor.
Tekintsük először azt az esetet, amikor az adagoló AK szerinti minimumos prioritási sor.



11. ábra. A megoldástér pontjainak sematikus ábrázolása adagolós keresés esetén

```
  Const
    Inf=10.0E10;       (* a végtelen reprezentánsa *)
  Type
    MTer  = ???;(* a megoldástér típusa *)
    PriSor= ???;(* a minimumos prioritási sor típusa *)
{--------------------------- Prioritási sor műveletek: ---------------------}
  Procedure Letesit(Var S:PriSor);
    Begin  End{Letesit};
  Procedure SorBa(Var S:PriSor; X:MTer);
    Begin  End{SorBa};
  Procedure SorBol(Var S:PriSor; Var X:MTer);
    Begin  End{SorBol};
  Procedure Megszuntet(Var S:PriSor);
    Begin  End{Megszuntet};
  Function Elemszam(S:PriSor):Word;
    Begin  End{Elemszam};

{--------------------- Probléma-specifikus műveletek:-------------------------}
  { A megoldástér bejarásához használt műveletek:}
  Procedure UresX(Var X:MTer);Forward;
    { X az üres megoldáskezdemény lesz }
  Function EFiu(Var X: MTer): Boolean; Forward;
    { Ha van X-nek fia, akkor X az első fiúra változik és a
     függvényhívás értéke True, egyébként False és X nem változik. }
  Function Testver(Var X: MTer): Boolean; Forward;
   { Ha van X-nek még benemjárt testvére, akkor X a következő testvér lesz
     és a függvényhívás értéke True, egyébként False és X nem változik.}
```

```
  Function Megoldas (Var X: MTer): Boolean; Forward;
    { akkor és csak akkor ad True értéket, ha X megoldása a problémának.}
  Function LehetMego(Var X: MTer): Boolean; Forward;
    { Ha LehetMego(X) hamis, akkor nincs megoldás az X gyökerű részfában. }
    { Ha LehetMego(X) igaz, abból nem következik, hogy van is megoldás.   }
  Function  C(Var X: MTer): Real; Forward;
   { Ha Megoldas(X) akkor C(X) az X megoldás célfüggvény értéke }
  Function AK(Const X: MTer): Real;
    { - Az X gyökerű részfában minden Y-ra, ha Megoldas(Y) akkor AK(X)<=C(Y).}
  Begin End;
  Function FK(Const X: MTer): Real;
    { Az X gyökerű részfaban minden Y-ra, ha Megoldas(Y) akkor C(Y)<=FK(X) }
    { Megjegyzés: FK(X)<Inf-bol nem következik, hogy egyáltalan létezik
      megoldás az X-gyökerű részfában!}
    Begin End;

Procedure Keres11(X:MTer; Var X0:MTer);
  { Optimalis megoldás keresése az X gyökerű megoldástér-fában. }
  { Ha az Y megoldás folytatasa X-nek, akkor AK(X)<=C(Y),
    de nem biztos, hogy AK(X)=C(X) ha X megoldás }
  { Aktiv pont választása a also korlát szerinti mini. prioritási sorral }
Var
  S:PriSor;  { AK-szerinti minimumos prioritási sor }
  G_F_K:Real;{ Globalis felső korlát: az eddigi legjobb Y megoldás,
               C(Y) célfüggvény értéke              }

Begin (* Keres11 *)
  UresX(X0);
  G_F_K:=Inf;                        { még nincs megoldásunk}
  If Not LehetMego(X) Then Exit;     { nem létezik megoldás}
  Letesit(S);                        { az S prisor létesítése}
  SorBa(S,X);                        { X az első aktiv pont}
  While Elemszam(S)>0 Do Begin       { amíg van aktiv pont S-ben}
    SorBol(S,X);                     { új aktív pontot a prisorbol}
    If G_F_K<=AK(X) Then Exit;       { egyetlen aktiv pontbol sem
                                       kapható jobb megoldás}
    If EFiu(X) Then                  { átlépés X első fiára, ha van,}
    Repeat                           { X összes fiának kigenerálása}
      If Not LehetMego(X) Then       { X kizárt pont lett}
        Continue;
      If AK(X)>=G_F_K Then Continue;{ X-nek nincs jobb folytátasa}
      If Megoldas(X) And             { új megoldást találtunk }
         (C(X)<G_F_K) Then  Begin    { az új megoldás az eddigi}
        G_F_K:=C(X);                 { legjobb:G_F_K aktualizálása }
        X0:=X;                       { feljegyezzük a jobb megoldást}
      End;
      SorBa(S,X);                    { X-et be az aktív pontok közé}
    Until Not Testver(X);            { átlépés a következő fiúra, ha van }
  End{while};
End (* Keres11 *);
```

Az adagoló a felső korlát szerinti minimumos prioritási sor.

```
Procedure Keres12(X:MTer; Var X0:MTer);
  { Optimális megoldás keresese az X gyökerű megoldástér-fában. }
```

```
  { Ha az Y megoldás folytatasa X-nek, akkor AK(X)<=C(Y)<=FK(X),
    de nem biztos, hogy AK(X)=C(X) ha X megoldás               }
  { Aktiv pont választása a felső korlát szerinti mini. prioritási sorral }
Var
  S:PriSor;  { FK-szerinti minimumos prioritási sor }
  G_F_K:Real;{ Globalis felső korlat: az eddigi legjobb Y megoldás,
               C(Y) célfüggvény értéke              }

Begin (* Keres12 *)
  UresX(X0);
  G_F_K:=Inf;                        { még nincs megoldásunk }
  If Not LehetMego(X) Then Exit;     { nem léetezik megoldás }
  Letesit(S);                        { az S prisor létesítése }
  SorBa(S,X);                        { X az első aktív pont }
  While Elemszam(S)>0 Do Begin       { amíg van aktiív pont S-ben }
    SorBol(S,X);                     { uj aktiv pontot a prisorbol }

    If EFiu(X) Then                  { átlépés X első fiára, ha van, }
    Repeat                           { X összes fiának kigenerálása  }
      If Not LehetMego(X) Then       { X kizárt pont lett}
        Continue;
      If AK(X)>=G_F_K Then Continue;{ X-nek nincs jobb folytátasa }
      If Megoldas(X) And             { új megoldást találtunk }
         (C(X)<G_F_K) Then  Begin    { az új megoldás az eddigi legjobb:}
        G_F_K:=C(X);                 { G_F_K aktualizálása }
        X0:=X;                       { feljegyezzük a jobb megoldást }
      End;
      SorBa(S,X);                    { X-et be az aktív pontok közé }
    Until Not Testver(X);            { átlépés a következő fiúra}
  End{while};
End (* Keres12 *);
```

Az elágazás-korlátozás módszer alkalmazása az optimális pénzváltás probléma megoldására.

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.

Tegyük fel, hogy a pénzek nagyság szerint nemcsökkenő sorrendbe rendezettek: $p_1 \geq \ldots \geq p_n$. A megoldást keressük $X = \langle i_1, \ldots, i_m \rangle$, $i_1 < i_2 < \ldots < i_m$ alakú vektor formában. jelölje $Resz = \sum_{k=1}^{m} p_{i_k}$ és $Maradt = \sum_{j=i_m+1}^{n} p_j$ összegeket.

$$AK(X) = m + \lceil (E - Resz)/p_{i_m+1} \rceil$$

$$FK(X) = m + \lceil (E - Resz)/p_n \rceil$$

10.1. Erős felső korlát

Az $FK(X)$ felső korlátot erős felső korlátnak nevezzük, ha bármely X megoldáslezdeményre:

$$FK(X) < \infty \Rightarrow (\exists Y)(Y \trianglelefteq X \wedge \textsc{Megoldas}(Y) \wedge C(Y) \leq FK(X))$$

```
Procedure Keres21(X:MTer; Var X0:MTer; Var C0:Real);
  { Optimalis megoldás keresése az X gyökerű megoldástér-fában. }
  { Aktiv pont választása a also korlát szerinti mini. prioritási sorral }
Var
  S:PriSor;  { AK-szerinti minimumos prioritási sor       }
  G_F_K:Real;{ Létexik olyan X megoldás, hogy C(X)<=G_F_K }
```

```
Begin (* Keres21 *)
  UresX(X0); G_F_K:=FK(X);C0:=G_F_K;
  If Not LehetMego(X) Then Exit;   { Nincs megoldás }
  Letesit(S); SorBa(S,X);          { üres prisor létesítése }
  While Elemszam(S)>0 Do Begin     { amíg van aktív pont }
    SorBol(S,X);                   { új aktív pontot a prisorból}
    If AK(X)>G_F_K Then Begin      { egyetlen aktiv pontbol sem}
      Megszuntet(S); Exit          { kapható már jobb megoldás!}
    End;                           { vége a keresésnek}
    If EFiu(X) Then Repeat         { átlépés az első fiúra, ha van }
      If Not LehetMego(X) Then Continue;{ X kizárt pont lett}
      If G_F_K<AK(X) Then Continue;{ X-nek nincs jobb folytatása }
      If Megoldas(X) And (C(X)<=G_F_K)
      Then Begin                   { új, jobb megoldást kaptunk,}
        X0:=X;                     { feljegyezzük }
        G_F_K:=C(X)                { G_F_K aktualizálása }
      End Else If FK(X)<G_F_K Then
        G_F_K:=FK(X);              { G_F_K aktualizálása }
      SorBa(S,X);                  { X-et be az aktív pontok közé}
    Until Not Testver(X);          { átlépés a követ. testvérre}
  End{while};
  C0:=G_F_K;
End (* Keres21 *);

Procedure Keres22(X:MTer; Var X0:Mter; Var C0:Real);
 { Aktív pont választasa a első korlat szerinti minimumos
  prioritási sorral, FK erős felső korlát }
Var  S:PriSor; { FK-szerinti minimumos prioritási sor }
  G_F_K:Real;  { létezik olyan X: C(X)<=G_F_K}

Begin (* Keres22 *)
  C0:=Inf;
  If Not LehetMego(X) Then Exit;       {nincs megoldás}
  G_F_K:=FK(X);                        {inicializálás}
  Letesit(S);                          {üres prisor létesítése}
  SorBa(S,X);
  While Elemszam(S)>0 Do Begin         {amíg van aktív pont}
    SorBol(S,X);                       {új aktiv pontot a prisorból}
    If G_F_K<AK(X) Then                {X-nek nincs jobb folytatása,}
       Continue;                       {kizárt pont lesz}
    If EFiu(X) Then                    {átlépés az első fiúra}
     Repeat                            {X összes fiának kigenerálása}
        If Not LehetMego(X) Then Continue;{ X kizárt lett}
        If G_F_K<AK(X) Then Continue;  {X-nek nincs jobb folytatása}
        If Megoldas(X) And (C(X)<=G_F_K) Then Begin
          X0:=X;                       {új, jobb megoldást kaptunk}
           G_F_K:=C(X);                {G_F_K aktualizálása}
       End Else If FK(X)<G_F_K Then    {létezik jobb megoldás, amely}
           G_F_K:=FK(X);               {folytatása X-nek }
        SorBa(S,X);                    {X-et be az aktív pontok közé}
    Until Not Testver(X);              {átlépés a következő testvérre}
  End{while};
  C0:=G_F_K;
End (* Keres22 *);
```

10.2. Ütemezési probléma

Bemenet:
$M = \{m_1, \ldots, m_n\}$ munkák halmaza
$m[i].idotartam \geq 0$ egész
$m[i].hatarido \geq 0$ egész
$m[i].haszon \geq 0$ valós
Kimenet:
$H \subseteq 1..n$
1. A H-beli munkák beoszthatók határidőt nem sértő módon.
2.

$$\overline{C}(H) = \sum_{i \in H} m[i].haszon \rightarrow maxi \tag{8}$$

H elemeinek egy $\langle i_1, \ldots, i_k \rangle$ felsorolása határidőt nem sértő, ha $\forall 1 \leq j \leq k$

$$\sum_{u=1}^{j} m[i_u].idotartam \leq m[i_j].hatarido \tag{9}$$

Állítás: H-nak akkor és csak akkor van határidőt nem sértő beosztása, ha elemeinek határidő szerinti felsorolása határidőt nem sértő.
$\Leftarrow$ trivi.
$\Rightarrow$ Tfh. H-nak van határidőt nem sértő beosztása, de ebben van olyan egymást követő u és $u+1$, hogy $m[i_u].hatarido > m[i_{u+1}].hatarido$. Ekkor u és $u+1$ felcserélhető a sorban.
Visszavezetés minimalizációs feladatra.

$$\begin{aligned} C(H) &= \sum_{i \notin H} m[i].haszon \\ &= \sum_{i=1}^{n} m[i].haszon - \overline{C}(H) \rightarrow mini \end{aligned}$$

$\overline{C}(H) \rightarrow maxi \Leftrightarrow C(H) \rightarrow mini$
Tegyük fel, hogy a munkák határidő szerint nemcsökkenő sorrendben vannak felsorolva. Ekkor a megoldás kifejezhető
$X = \langle i_1, \ldots, i_k \rangle$ vektorral, ahol $i_1 < i_2 < \cdots < i_k$
Minden X megoldáskezdeményre def. a problémaspecifikus műveleteket.
$LehetMego(X) = igaz \Leftrightarrow$ ha a felsorolás határidőt nem sértő.
$Megoldas(X) = LehetMego(X)$

$$AK(X) = \sum_{j < i_k, j \notin X} m[j].haszon \tag{10}$$

$$FK(X) = \sum_{j \notin X} m[j].haszon \tag{11}$$

Tehát, ha $LehetMego(X)$, akkor $Megoldas(X)$ és $C(X) = FK(X)$, ezért FK erős felső korlát.