Backtrack – Hatékonyság
gyakorlatok

Szlávi Péter

2008-2009

TARTALOM

0. CÉLOK

Kettős célt tűzünk ki. Egyrészt a backtrack alapú keresés logikájából kiindulva a **megszámolás** (vagy ami ehhez igen közel áll: a **kiválogatás kiírással**) algoritmusának összeállítása; másrészt az algoritmus időbeli **hatékonyság**ának vizsgálata.

Azért éppen a megszámolást (és nem pl. a kiválasztást) célozzuk meg, mert ez az a fajta feladat, amely során legjobban érvényesülhet a backtrack hatékonyságnövelő szerepe. Hiszen ennél a típusú feladatnál a **teljes állapotfát be kell járni**, és ekkor válik érdekessé, hogy az állapotfa hány pontját sikerül a backtrack „filozófiájának" köszönhetően elhagyni…

Építeni fogunk a backtrack-es keresés ismeretére, és az ennek kipróbálásra elkészített N-vezéres programra.

1. MEGSZÁMOLÁS „BACKTRACK MÓDRA”

Kapcsolatot keresünk a backtrack keresés és a lineáris keresés algoritmusa közt. E kapcsolatot arra használjuk föl, hogy kiindulva a kiválogatás tétel ismert algoritmusából megsejtsük a backtrack-alapú kiválogatás algoritmusát.

1.1. A lineáris keresés lényegkiemelése – általánosítás

A lineáris keresés	***Az absztrakt keresés***
`i:=1` [X 1. eleménél kezdődik a keresés]	`L:=X`
Ciklus amíg `i≤N` [van még hol keresni]	**Ciklus amíg** `‖L‖>0`
és nem `T(X(i))` [az i. olyan-e]	**és nem** `T(X(ε(L)))`
`i:+i` [a következőnél kezdődik a sorozat]	`L:=σ(L)`
Ciklus vége	**Ciklus vége**
`Van:=N-i+1>0`	`Van:=‖L‖>0`
Ha `Van` **akkor** `Sorsz:=i` [a keresett]	**Ha** `Van` **akkor** `Sorsz:=ε(L)`

ahol bevezettük az alábbi fogalmakat:

- $L \in H^*$ ***lehetségesek sorozata*** — az eredeti X egy alkalmasan választott (algoritmus meghatározta) rész-sorozata ($L \subseteq X$)
- $\|\cdot\|: H^* \to \mathbb{N}$ ***hossz*** — a sorozat hossza (elemszáma)
- $\sigma: H^* \to H^*$ ***szűkítés*** — $L \in H^*$: $(x \in \sigma(L) \Rightarrow x \in L) \wedge \|\sigma(L)\| < \|L\|$, azaz a σ L-ből elhagy valahány elemet
- $\varepsilon: H^* \to \mathbb{N}$ ***elemkijelölés*** — $L \in H^*$: $\varepsilon(L) \in [1..\|X\|]$, azaz ε L egy lehetséges elemének indexét adja

1.2. A visszalépéses keresés algoritmusa

Csak a legfelsőbb szint érdekel most minket. Mindjárt illesszük mellé az általánosított keresést!

A visszalépéses keresés	***Az absztrakt keresés***
`i:=1; X(1..K):=0`	`L:=X`
Ciklus amíg `i≥1`	**Ciklus amíg** `‖L‖>0`
és `i≤K`	**és nem** `T(X(ε(L)))`
`ker:=JóElem(i)`	`L:=σ(L)`
Ha `ker.van` **akkor**	
`X(i):=ker.melyik; i:+1`	
különben	
`X(i):=0; i:-1`	
Elágazás vége	
Ciklus vége	**Ciklus vége**
`Van:=i≥1`	`Van:=‖L‖>0`
~~**Ha** `Van` **akkor** …~~ [X maga a keresett]	**Ha** `Van` **akkor** `Sorsz:=ε(L)`

Hasonlítsuk össze őket! Az alábbi fogalmi kapcsolatok fedezhetők föl:

Absztrakt fogalom	*Backtrack „fogalom”*
L:=X ‖L‖>0 **nem** T(X(ε(L))) L:= σ(L)	i:=1; X(1..K):=0 i≥1 i≤K ker:=JóElem(i) **Ha** ker.van **akkor** X(i):=ker.melyik; i:+1 **különben** X(i):=0; i:-1 **Elágazás vége**

1.3. Megszámolás és általánosítása

Két lépésben végezzük el az általánosítást. Elsőként átírjuk ’amíg’-os ciklussá, hogy világosabb legyen a fentiekkel a kapcsolat. Majd általánosítunk a korábban bevezetett fogalmak felhasználásával.

Az első lépés:

A megszámolás	*Az ’amíg’-os megszámolás*
Db:=0 **Ciklus** i=1-től N-ig **Ha** T(X(i)) **akkor** Db:+1 **Elágazás vége** **Ciklus vége**	Db:=0; i:=1 **Ciklus amíg** i≤N **Ha** T(X(i)) **akkor** Db:+1 **Elágazás vége** i:+1 **Ciklus vége**

A második:

Az ’amíg’-os megszámolás	*Az absztrakt megszámolás*
Db:=0; i:=1 **Ciklus amíg** i≤N **Ha** T(X(i)) **akkor** Db:+1 **Elágazás vége** i:+1 **Ciklus vége**	Db:=0; L:=X **Ciklus amíg** ‖L‖>0 **Ha** T(X(ε(L))) **akkor** Db:+1 **Elágazás vége** L:=σ(L) **Ciklus vége**

1.4. Backtrack megszámolás algoritmusa

Az absztrakt megszámolásból az alábbi backtrack-es megszámolást kaphatjuk, miközben alkalmazzuk az 1.2.-ben föltárt fogalmi kapcsolatokat.

Egyetlen „furcsaságot” kell csak hozzátenni: **amikor egy megoldást a backtrack megtalál**, úgy kell folytassa, hogy az utolsó komponensnél folytassa a továbbkeresést, azaz egy **visszalépés**t is kell tenni.

Az absztrakt megszámolás	***A backtrack-es megszámolás***
Db:=0; L:=X	Db:=0; i:=1; X(1..K):=0
Ciklus amíg ‖L‖>0	**Ciklus amíg** i≥1
Ha T(X(ε(L))) **akkor**	**Ha** i>K **akkor**
Db:+1	Db:+1 ~~X(i):=0;~~ i:-1 [visszalépés!!!]
Elágazás vége	**Elágazás vége**
L:=σ(L)	Ker:=JóElem(i) **Ha** ker.van **akkor** X(i):=ker.melyik; i:+1 **különben** X(i):=0; i:-1 **Elágazás vége**
Ciklus vége	**Ciklus vége**

2. AZ N-VEZÉR PROBLÉMA MEGOLDÁSOK SZÁMA – HATÉKONYSÁG

2.1. Alapelvek

Az időbeli viselkedés mérésére alkalmas a **vezérek ütésszáma**. Ehhez mindössze arra van csak szükség, hogy az ütést vizsgáló függvénybe beleillesszünk egy *számláló* inkrementálását. Ez a számláló célszerűen *globális változó*, hasonlóan a *megoldás-szám változó*hoz (hogy ne kelljen paraméterezéssel rontani a hatékonyságot), amelyet állítani kell kezdetben (inicializáláskor), és el kell tudni érni a végeredmény megjelenítésénél is.

Kiindulópontul szolgál az előző gyakorlaton készült, N-vezér problémára írt megoldás kereső program. Ez tartalmaz egy „vizuális” betétet is az algoritmus működés mélyebb megértése, könnyebb „nyomon követhetősége” céljából. Erre azonban a végső programban, amelyet a megoldás-szám meghatározásra (vagy az összmegoldások kiírására) írunk, kevés szüksége lesz, ezért majd kihagyható, kihagyandó.

2.2. A megoldás első változata

Lényege –túl a visszalépést megszervező részek algoritmizálásán–: a program a megoldás **számlálás** és a **hatékonysági mérés** megszervezése.

A cél, hogy megkapjuk az adott N-hez mennyi

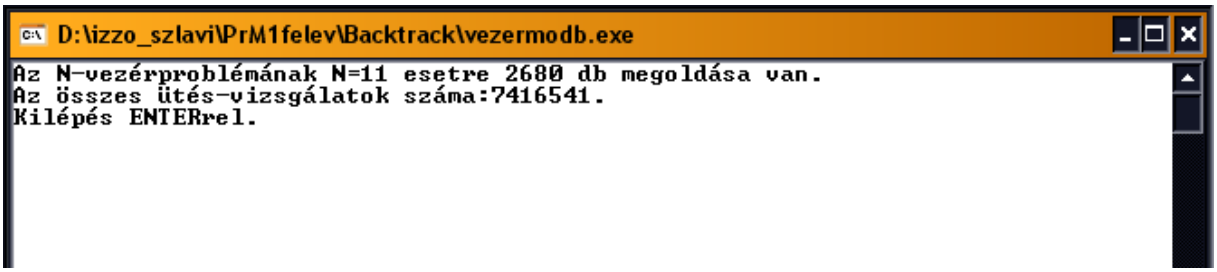
- a megoldások száma, és
- az ütésvizsgálatok száma.

A programban pirossal jelöltük az új részeket. (A kód kipróbálható: vezermodb.exe.)

```
Program N_Vezer_Osszes_Megoldasok_Szama;
  Uses
     Crt;
  Const
     MaxN  = 16;
  Type
     Index = 0..MaxN+1;
     Hol   = Array [1..MaxN] of Index;
  Var
     Vezer : Hol;
     N,i,j,
     hova  : Index;
     lehet : Boolean;
   { megjelenítési paraméterek: }
     utVizsDb,       { ütés-vizsgálatok száma }
     moDb: LongInt; { megoldás-szám }
  Procedure Inicializalas(Var n: Index);
    Var
      i,j: Integer;
  Begin
    Window(1,1, 80,25);
    TextColor(White); TextBackGround(Black); ClrScr;
    Writeln('N-vezérprobléma':48);
```

```
    Repeat
      GotoXY(1,5); Write('Vezérszám (3<N<',MaxN:2,'):');
      Readln(n)
    Until n in [4..MaxN-1];
  End; {Inicializalas}
  Procedure OsszMegoldas(n: Index);
    Var
      c: Char;
  Begin
    Window(1,1, 80,25);
    TextColor(White); TextBackGround(Black); ClrScr;
    Writeln('Az N-vezérproblémának N=',n,' esetre ',moDb,
            ' db megoldása van.');
    Writeln('Az összes ütés-vizsgálatok száma:',utVizsDb,'.');
    Writeln('Kilépés ENTERrel.');
    Repeat
      c:=ReadKey;
    Until c=#13;
  End; {OsszMegoldas}

  Function Uti(vx1,vy1, vx2,vy2: Index): Boolean;
  Begin
    Inc(utVizsDb);
    Uti:=(vy1=vy2) or (Abs(vy1-vy2)=vx1-vx2)
  End; {Uti}

  Function RosszEset(x,y: Index): Boolean;
    Var
       j: Index;
  Begin
    j:=1;
    While (j<=x-1) and not uti(x,y, j,Vezer[j]) do Inc(j);
    RosszEset:=j<=x-1
  End; {RosszEset}

  Procedure VanJoEset(i: Index; Var talan: Boolean; Var ide: Index);
  Begin
    ide:=Vezer[i]+1;
    While (ide<=N) and RosszEset(i,ide) do Inc(ide);
    talan:=ide<=N;
  End; {VanJoEset}

Begin {Főprogram}
  Inicializalas(N);
  utVizsDb:=0; moDb:=0;
  i:=1;
  For j:=1 to N do Vezer[j]:=0;
  While i>=1 do
  Begin
    If i>N then {megoldás}
    Begin
      Inc(moDb);
      VanMegoldas(N);
      {Visszalépés:}
      {Vezer[i]:=0;} Dec(i);
    End;
    VanJoEset(i,lehet,hova);
    If lehet then
    Begin
      Vezer[i]:=hova; Inc(i);
    End
```

```
      else
    Begin
      Vezer[i]:=0; Dec(i);
    End;
  End;
  OsszMegoldas(N);
End.
```



2.2.–1. ábra. A futás végeredményét mutató ablak képe, N=11-re.

2.3. A megoldás második változata – folytatható futtatás

E változat is egy adott N-hez tartozó vezérelhelyezések (és ütés-vizsgálatok) számát igyekszik meghatározni. A program újdonsága, hogy futás közben minden egyes megoldás megtalálásakor egy *időleges* (szöveges) **fájlba írja a pillanatnyi jellemző állapotát**. Mégpedig az N-et, az aktuálisan talált megoldást (Vezer[1..N] tömbelemeket), az eddig talált megoldások számát (moDb-t) és az eddigi ütésvizsgálatok számát (utVizsDb-t). A kiírás után lezárja a fájlt, így az megmarad, mégha a program valamilyen oknál fogva meg is áll, pl. végére ér a számításnak, vagy a felhasználó megszakítja a program futását. A program indításakor megkérdezi, hogy továbbfolytassa-e a korábbi állapotból vagy új számításba kezdjen.

A program az elvi hatékonysági mutató (utVizsDb/moDb) mellett méri a tényleges futási időt is. De –sajnos– ez csak pontatlanul jellemzi a szükséges időket, így aztán: nem alkalmas a hatékonyság tendenciájának vizsgálatára, hiszen függ a számítógép hardver képességeitől (pl. a processzor sebességétől) és pillanatnyi szoftverállapotától (pl. egy időben futó szálaktól). (A kód kipróbálható: vezermodb2.exe.)

```
Program N_Vezer_Osszes_Megoldasok_Szama;
(*
  Gyorsított, menet közben csendes változat;
  Az egyes megoldások megtalálásakor szöveges fájlba (VezMoDb.tmp)
  rögzíti az állapotot.
  Így bármikor ebből az állapotból folytatható.
  (A mért idő a "tisztán" számításra fordított idő, század másodpercekben.)
*)
  Uses
     Crt,Dos{Get/SetTime-hoz};
  Const
     MaxN  = 16;
     fN    = 'VezMoDb.tmp'; {a munkafájl neve}
  Type
     Index = 0..MaxN+1;
     Hol   = Array [1..MaxN] of Index;
  Var
     Vezer : Hol;
     N,i,j,
```

```
    hova  : Index;
    lehet : Boolean;
  { megjelenítési paraméterek: }
    moDb: LongInt;     { megoldás-szám }
    utVizsDb: Comp;    { ütés-vizsgálatok száma }
    folyt: Boolean;    { folytatás-e vagy újrakezdés }
    Kezdet,Veg,
    ElteltIdo: LongInt;

Procedure OraIndul;
  Var
     o,p,mp,szmp: Word;
     ol: LongInt;
Begin
  GetTime(o,p,mp,szmp); ol:=o; {konverzió LongInt-tá}
  Kezdet:=szmp+100*(mp+60*(p+60*ol{konverzió LongInt-tá}))
End; {OraIndul}

Procedure OraAll;
  Var
     o,p,mp,szmp: Word;
     ol: LongInt;
Begin
  GetTime(o,p,mp,szmp); ol:=o; {konverzió LongInt-tá}
  Veg:=szmp+100*(mp+60*(p+60*ol{konverzió LongInt-tá}));
End; {OraAll}

Procedure Inicializalas(Var n: Index);
  Var
    i,j: Integer;
    f  : Text;
    c  : Char;
Begin
  Window(1,1, 80,25); TextColor(White); TextBackGround(Black); ClrScr;
  Writeln('N-vezérprobléma':48);

  Write('Folytatás, fájlból (I/.)?:'); c:=ReadKey;
  folyt:=UpCase(c)='I';
  If folyt then
  Begin
    {$i-}
    writeln; writeln('Fájlból folytatom:');
    assign(f,fN);
    reset(f);
    {$i+}
    If IOResult<>0 then
    Begin
      Writeln('Nincs meg a folytatáshoz szükséges paraméterfájl.');
      ReadKey; Halt;
    End;
    readln(f,n); writeln(' N:',n);
    readln(f,moDb); writeln(' moDb:',moDb);
    readln(f,utvizsDb); writeln(' utvizsDb:',utvizsDb:12:0);
    writeln(' Utolsó megoldás:');
    for i:=1 to n do
    Begin
      read(f,Vezer[i]); write(Vezer[i]:3);
    End;
    writeln;
    readln(f,ElteltIdo);
    writeln(' ElteltIdo:',ElteltIdo);
```

```
    close(f);
  End
    else
  Begin
    Repeat
      GotoXY(1,5); Write('Vezérszám (3<N<',MaxN:2,'):'); Readln(n)
    Until n in [4..MaxN-1];
  End;
End; {Inicializalas}

Procedure VanMegoldas(n: Index);
  Var
    i,j: Index;
    f  : Text;
Begin
  OraAll;
  assign(f,fN); rewrite(f);
  writeln(f,n,' -- n');
  writeln(f,moDb, ' -- moDb');
  writeln(f,utvizsDb:12:0, ' -- utvizsDb');
  For i:=1 to n do
  Begin
    write(f,Vezer[i]:3);
  End;
  writeln(f);
  ElteltIdo:=ElteltIdo+Veg-Kezdet;
  writeln(f,ElteltIdo, ' -- ElteltIdo');
  close(f);
  OraIndul;
End; {VanMegoldas}

Procedure OsszMegoldas(n: Index);
  Var
    c: Char;
Begin
  Window(1,1, 80,25); TextColor(White); TextBackGround(Black); ClrScr;
  Writeln('Az N-vezérproblémának N=',n,' esetre ',moDb,
          ' db megoldása van.');
  Writeln('Az összes ütés-vizsgálatok száma:',utVizsDb:12:0,'.');
  ElteltIdo:=ElteltIdo+Veg-Kezdet;
  writeln('Eltelt idő:',ElteltIdo/100:0:1,'mp.');
  Writeln('Kilépés ENTERrel.');
  Repeat
    c:=ReadKey;
  Until c=#13;
End; {OsszMegoldas}

Function Uti(vx1,vy1, vx2,vy2: Index): Boolean;
Begin
  utVizsDb:=utVizsDb+1; {ütésvizsgálat-számlálás}
  Uti:=(vy1=vy2) or (Abs(vy1-vy2)=vx1-vx2)
End; {Uti}

Function RosszEset(x,y: Index): Boolean;
  Var
     j: Index;
Begin
  j:=1;
  While (j<=x-1) and not uti(x,y, j,Vezer[j]) do Inc(j);
  RosszEset:=j<=x-1
End;
```

```
  Procedure VanJoEset(i: Index; Var talan: Boolean; Var ide: Index);
  Begin
    ide:=Vezer[i]+1;
    While (ide<=N) and RosszEset(i,ide) do Inc(ide);
    talan:=ide<=N;
  End; {RosszEset}

Begin {Főprogram}
  Inicializalas(N);
  If folyt then
  Begin
    i:=N;
  End
    else
  Begin
    utVizsDb:=0; moDb:=0;
    i:=1;
    For j:=1 to N do Vezer[j]:=0;
  End;
  OraIndul;
  While i>=1 do
  Begin
    If i>N then {újabb megoldás}
    Begin
      Inc(moDb);
      VanMegoldas(N);
      {Visszalépés:}
      {Vezer[i]:=0;} Dec(i);
    End;
    VanJoEset(i,lehet,hova);
    If lehet then
    Begin
      Vezer[i]:=hova; Inc(i);
    End
      else
    Begin
      Vezer[i]:=0; Dec(i);
    End;
  End;
  OraAll;
  OsszMegoldas(N);
End.
```

Megjegyzés:

Az persze előfordulhat, hogy rossz „lélektani pillanatban” állítjuk le a programot, éppen akkor, amikor a fájlbaírásnál tart, ekkor ui. már a korábbi változat nem létezik, az új még nem készült el.

Ez a probléma úgy orvosolható, hogy mielőtt a munkafájlt megnyitná, átnevezi. Így az utolsó egy vagy kettő állapot megmarad.

```
12 - n
4956 - moDb
    17148907 - utvizsDb
  5  9 12 10  2  6  1 11  8  3  7  4
142 - ElteltIdo
```

2.3.–1. ábra. A `VezMoDb.tmp` fájl `N=12`-höz tartozó, egy közbülső állapota.

```
12 - n
14200 - moDb
    45385225 - utvizsDb
 12 10  8  5  3  1  7  2 11  6  4  9
358 - ElteltIdo
```

2.3.–2. ábra. A `VezMoDb.tmp` fájl N=12-höz tartozó végállapota.

2.4. A hatékonysági vizsgálat előkészítése

Annak érdekében, hogy csak később és viszonylag kényelmesen foglalkozhassunk annak elemzésével, hogy az N függvényében miként alakul a megoldás-szám, valamint az ütközés-szám, egy **szöveges fájlba írjuk az eredményeket** (itt tehát nem a folytathatóság kedvéért használjuk a fájlt!).

Az eredmény kiírása fájlba kétféle koncepció szerint történhet.

1. A program kezdő paramétereként a maximális N-et (MaxN) kéri be. Egy ciklusban az addigi „érdekes” N-ekre (4..MaxN) meghatározza a *megoldások számát*, valamint az ehhez szükséges *ütés-vizsgálatok számát*. Majd ezeket (az aktuális N-nel együtt) a fájlba írja, N-enként külön sorba. Ez a megoldás nagy futási időket kíván. N=12-től érezhetően „lelassul” a számítás. Így ki kell várjuk MaxN-ig az összes N-re a számítást. Ezen igyekszik segíteni a második megoldás, amely lehetővé teszi a „részeredmények” felhasználását is.
2. Egy megadott nevű fájlt –ha létezik– folytat, ha még nem léteik, akkor „újrakezd”. A folytatás azt jelenti, hogy a fájlvégén új sorral bővít. Ehhez a Pascal speciális megnyitását kell használni (Append), az újrakezdéshez a szokásos megnyitást (Rewrite) használjuk. A megoldás a következőt jelenti: ha létezik az adatfájl, akkor végigolvassuk azt, megkeresve az utolsó sor N-jét. Ezt inkrementálva indítjuk a megoldáskeresést. A fájl lezárása, majd Append-del újranyitása után a legújabb eredménnyel bővítjük a fájlt.

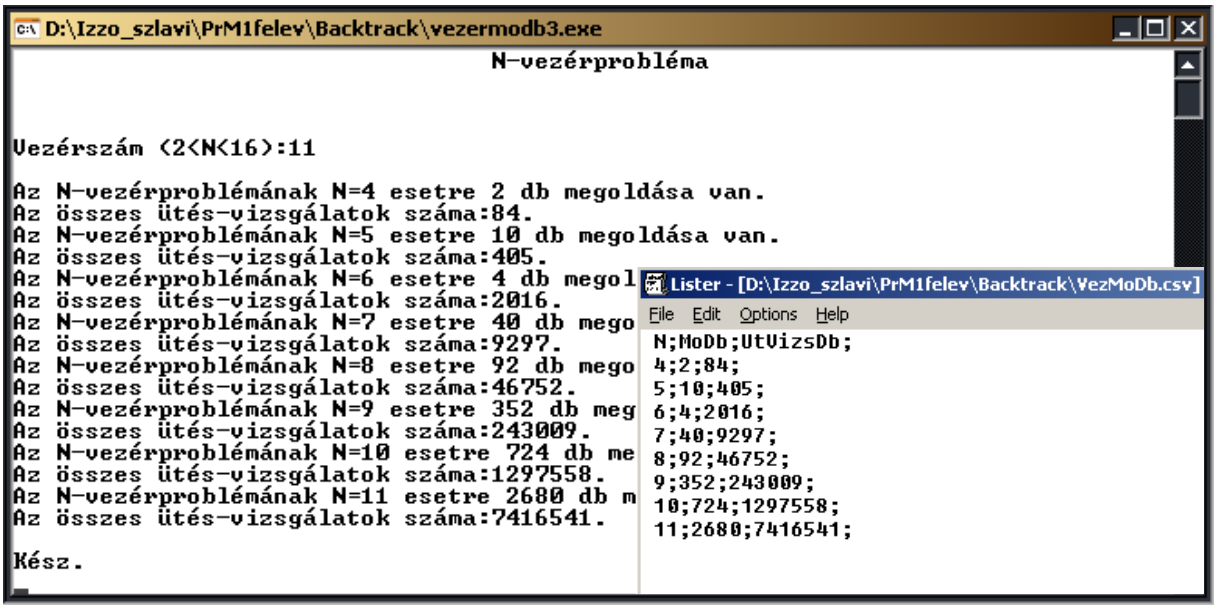
Ügyelni kell, hogy a fájl egyes soraiba elhelyezett 3 adat felismerhető legyen, azaz valamilyen **elválasztójel**et kell közéjük tenni. Erre alkalmas a *szóköz*, de megfelelő pl. a **pontosvessző**, vagy a *tab* is. (Olyan elválasztó jelet érdemes választani, ami a táblázatkezelők számára emészthető fájlokban előfordulhat mező-elválasztójelként. Ilyen pl. a pontosvessző, ui. ez a „formátuma” a CSV[♣]-nek.) (A kód kipróbálható: vezermodb3.exe.)

```
Program N_Vezer_Osszes_Megoldasok_szama;
(*
  Szöveges fájlba írja adott N-ig az N-vezér feladat:
  * megoldásainak és
  * szükséges ütközésvizsgálatainak
  a számát.
  A fájl CSV formátumú. (Alkalmas arra, hogy a táblázatkezelő programokkal
  feldolgozzák.
  Szerkezete:
  1. sor: fejléc szövegek.
  i. sor: N;MoDb;utDb;
*)
```

[♣] Comma separated value.

```
Uses
   Crt;
Const
   MaxN  = 16;
   fN    = 'VezMoDb.csv'; { statisztikafájl neve }
Type
   Index = 0..MaxN+1;
   Hol   = Array [1..MaxN] of Index;
Var
   Vezer : Hol;
   mN,N,i,j,
   hova  : Index;
   lehet : Boolean;
 { megjelenítési paraméterek: }
   utVizsDb,          { ütés-vizsgálatok száma }
   moDb  : LongInt;   { megoldás-szám }
   f     : Text;      { statisztikafájl }

Procedure Inicializalas(Var n: Index);
  Var
    i,j: Integer;
Begin
  Window(1,1, 80,25);
  TextColor(White); TextBackGround(Black); ClrScr;
  Writeln('N-vezérprobléma':48);
  Repeat
    GotoXY(1,5); Write('Vezérszám (3<N<',MaxN:2,'):');
    Readln(n)
  Until n in [4..MaxN-1];
  Writeln;
End; {Inicializalas}

Procedure OsszMegoldas(n: Index);
  Var
    c: Char;
Begin
  Writeln('Az N-vezérproblémának N=',n,' esetre ',moDb,
          ' db megoldása van.');
  Writeln('Az összes ütés-vizsgálatok száma:',utVizsDb,'.');
End; {OsszMegoldas}

Function Uti(vx1,vy1, vx2,vy2: Index): Boolean;
Begin
  Inc(utVizsDb);
  Uti:=(vy1=vy2) or (Abs(vy1-vy2)=vx1-vx2)
End; {Uti}

Function RosszEset(x,y: Index): Boolean;
  Var
     j: Index;
Begin
  j:=1;
  While (j<=x-1) and not uti(x,y, j,Vezer[j]) do Inc(j);
  RosszEset:=j<=x-1
End; {RosszEset}

Procedure VanJoEset(i: Index; Var talan: Boolean; Var ide: Index);
Begin
  ide:=Vezer[i]+1;
  While (ide<=N) and RosszEset(i,ide) do Inc(ide);
  talan:=ide<=N;
```

```
  End; {VanJoEset}

Begin {Főprogram}
  {statisztikafájl megnyitása:}
  Assign(f,fN); Rewrite(f);
  {a fájlbeli fejléc-sor kiírása:}
  Writeln(f,'N;MoDb;UtVizsDb;');
  Inicializalas(mN);
  For N:=4 to mN do
  Begin
    utVizsDb:=0; moDb:=0;
    i:=1;
    For j:=1 to N do Vezer[j]:=0;
    While i>=1 do
    Begin
      If i>N then {újabb megoldás}
      Begin
        Inc(moDb);
        {Visszalépés:}
        {Vezer[i]:=0;} Dec(i);
      End;
      VanJoEset(i,lehet,hova);
      If lehet then
      Begin
        Vezer[i]:=hova; Inc(i);
      End
        else
      Begin
        Vezer[i]:=0; Dec(i);
      End;
    End;
    OsszMegoldas(N);
    {a következő adatsor kiírása a fájlba:}
    Writeln(f,N,';',moDb,';',utVizsDb,';');
  End;
  {statisztikafájl lezárása:}
  Close(f);
  Writeln(#10#13{=CrLf,soremelés},'Kész. '); ReadKey;
End.
```



2.4.–1. ábra. A futás végeredményét mutató ablak képe, valamint a generált fájl tartalma, N=11-re.

2.5. A hatékonyságvizsgálat táblázatkezelővel

Az eredményfájl birtokában elemezzük a **backtrack** (BT) és a teljes fa átvizsgálásával (TF) dolgozó, legegyszerűbb, **lineáris keresés tétel** alapján készíthető megoldás viszonyát.

Az utóbbi, egyszerű TF algoritmus vizsgálatával kezdjük! Az algoritmus az összes variációt generálva *eldönti* mindegyikről, hogy az megoldást jelent-e. Vagyis a megoldás egy N-dimenziós térben dolgozó keresés lesz, amelyben szereplő *T-tulajdonság*ot az eldöntés tétel sablonja alapján implementáljuk.

Az *összes variációk száma* N vezér esetén: N^N. Annak eldöntése egy konkrét N-esről, hogy megoldást ír-e le, legrosszabb (értsd: leghosszabb) esetben N alatt a 2 darab ütközés-vizsgálatra van szükség. A legkedvezőbb esetben elég egyetlen egy is. Az *átlag* (=*átl*(N)) e kettő között van.♥ Most eltekintünk ennek pontos megállapításától, mivel minket úgyis csak a növekedés üteme izgat: ez az érték vagy $\Theta(N)$, vagy $\Theta(N^2)$ lehet. Végülis az algoritmus időbeli hatékonysága (az ütközésvizsgálatok léptékében mérve):

(TF1) $\Theta(N^{N+1})$, vagy

(TF2) $\Theta(N^{N+2})$, vagy

a precíz, de még „kifejtésre váró” *átl*(N) függvénnyel kifejezve:

(TFátl) $\Theta(\textit{átl}(N)\cdot N^N)$, ahol $1 \le \textit{átl}(N) \le N\cdot(N-1)/2$.

Lássuk a két algoritmikus „hozzáállás” összevetését! A BT hatékonyságát jellemző adatsorozatot *importáljuk* egy táblázatkezelőbe. Felvetődik a kérdés, miként ragadható meg az egyes algoritmusok időbeli hatékonysága. A hatékonyság önmagában is érdekes lehet, azaz változik-e; s ha igen, N függvényében hogyan? De –és ez az igazi kérdés– egymáshoz képest

♥ Ennek kísérleti alapú meghatározását lásd a függelékben.

hogyan: vagyis a BT mennyivel hatékonyabban talál rá a megoldásokra, mint a TF, és a hatékonyságnövekedés mi módon függ N-től (növekszik N-nel vagy csökken, esetleg stagnál)?

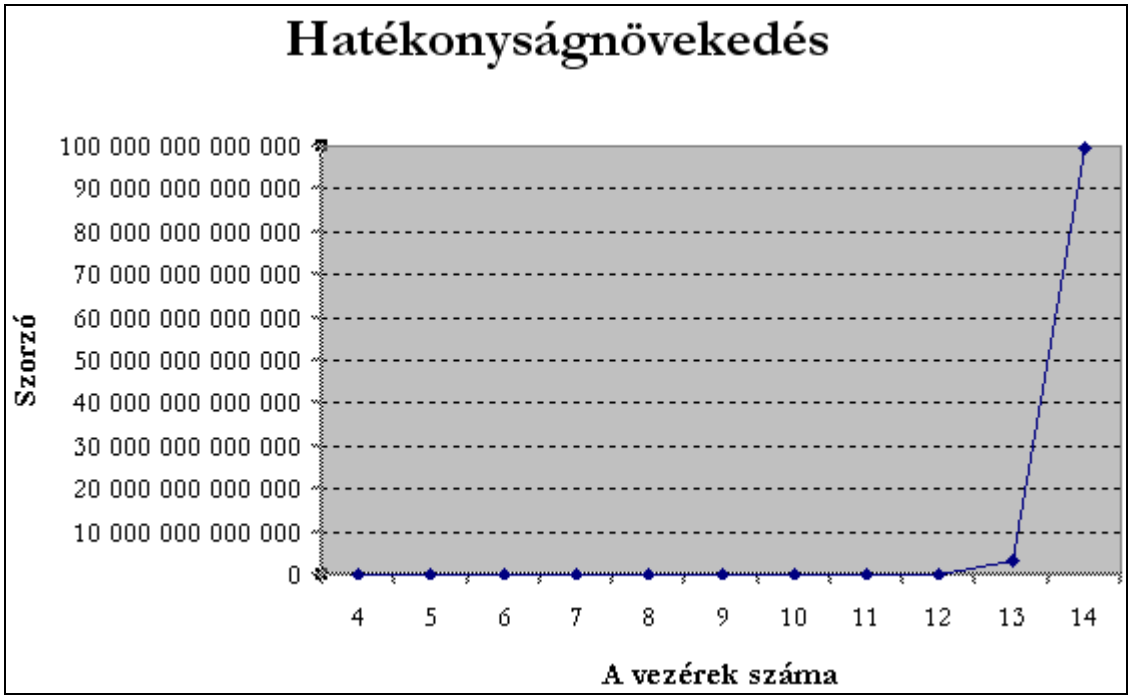
Az első kérdés kedvéért a táblázatot bővítsük ki: az egy megoldásra eső, *fajlagosan szükséges ütközésvizsgálatok számát* (ϕ_x=UtVizsDb$_x$/MoDb$_x$, x=BT,TF) tartalmazó oszloppal! Az egymáshoz viszonyított „*módszer-relatív*” *hatékonyságnövekedés*ét pedig az előbbi *fajlagos értékek arányá*val jellemezzük (ϕ_{TF}/ϕ_{BT}).

A TF módszer ütésvizsgálatait a „pesszimista” átlagként megfogalmazott (TF2)-höz hű „N/2 alatt a 2” értékkel vesszük figyelembe az „átlagos” esetet.

	Backtrack			Teljes fa-vizsgálat			
Vezérszám N	Megoldás-szám MoDb	Ütésvizsgálat-szám UtVizsDb	UtVizsDb/MoDb	levélszám N"	Ütésvizsgálat-szám (N/2 alatt a 2)*N"	UtVizsDb/MoDb	Hatékonyságnövekedés
4	2	84	42	256	2048	1024	**24**
5	10	405	41	3125	234375	23437,5	**579**
6	4	2016	504	46656	1679616	419904	**833**
7	40	9297	232	823543	46184080	11529602	**49606**
8	92	46752	508	16777216	24696061952	268435456	**528235**
9	352	243009	690	387420489	3,06837E+12	8716961003	**12626571**
10	724	1297558	1792	10000000000	1,81E+14	2,5E+11	**139492801**
11	2680	7416541	2767	2,85312E+11	2,5233E+16	9,41529E+12	**3402255061**
12	14200	45396914	3197	8,9161E+12	4,55791E+18	3,2098E+14	**100401330125**
13	73712	292182579	3964	3,02875E+14	1,01581E+21	1,37808E+16	**3476633042175**
14	365596	1995957888	5459	1,1112E+16	1,99063E+23	5,44488E+17	**99732944426951**

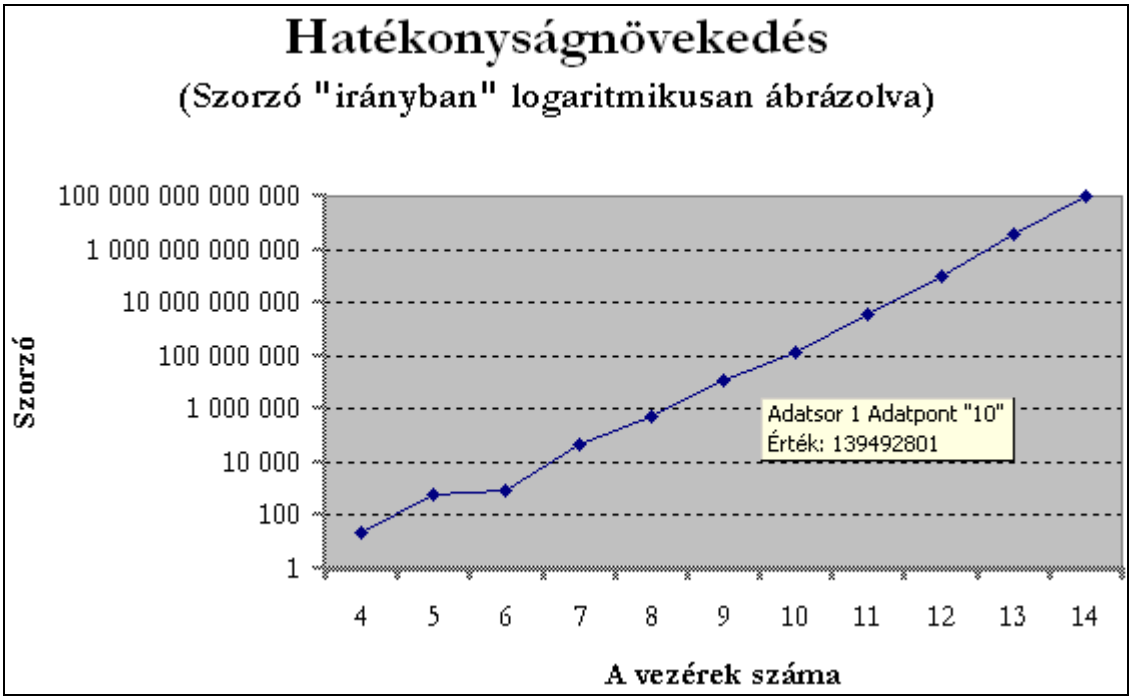
2.5.–1. ábra. A táblázat az importálás és oszlopbővítés után. (VezMoDb.xls)

Sokat mondó a látvány. Megállapítható, hogy N növekedtével egyre nagyobb a fajlagos ütésvizsgálat-szám. A BT esetében $\phi_{BT}(4)$= 42, …, $\phi_{BT}(14)$= 5459, azaz kb. 100-szorosára növekedett 11 lépés alatt, pontosabban: $\phi_{BT}(N+1)\approx\sqrt[11]{5459/42}\cdot\phi_{BT}(N)\approx 1{,}55\cdot\phi_{BT}(N)$. Elgondolkodtató a tendencia a TF esetében is, hiszen azt látjuk, hogy eggyel nagyobb N-re közel 10-szeresére nő a ϕ_{TF} értéke: $\phi_{TF}(N+1)\approx 10\cdot\phi_{TF}(N)$. Ez az „aránytalan” tendenciakülönbség olvasható le az utolsó oszlopból (ϕ_{TF}/ϕ_{BT}) is. „Szemre” is látszik rajta a hatékonyságnövekedés üteme: eggyel nagyobb N-re nagyjából egy számjeggyel nagyobb. Hogy még felfoghatóbb legyen ez a „globális hatékonyságnövekedés”, ábrázoljuk grafikusan a ϕ_{TF}/ϕ_{BT}-ket tartalmazó oszlop adatait!



2.5.–2. ábra. A ϕ_{TF}/ϕ_{BT}-k alakulása, diagrammal.

Hát ez nem a remélt módon alakult! A növekedés túl drasztikus ahhoz, hogy az egész görbe valamit mondjon. Térjünk át az értékek logaritmusára, és így ábrázoljuk! Íme:



2.5.–3. ábra. A ϕ_{TF}/ϕ_{BT}-k alakulása diagramon, az értéket logaritmikus skálán ábrázolva.

Ez a diagram már inspirál a következtetések levonására! A függvénygörbére rápillantva kijelenthető, hogy –nagyjából– lineárisan növekedő. Ebből kiindulva a BT hatékonyságára formulát is kiokoskodhatunk. Ekképpen:

Jelöljük a BT és a TF hatékonyságát N függvényében bt(N)-nel és tf(N)-nel! A diagram alapján föltesszük, hogy

(BT1) $\lg(tf(N)/bt(N))=c*N$

A c meghatározása: N=14-nél kb. 10^{14} az érték, így $\lg(10^{14})=14$, így iránytangensként az 1 választható. (1)-ből következik, hogy

(BT2) $tf(N)/bt(N)=10^N$.

Ebből kapjuk bt(N)-re:

(BT3) $bt(N)=tf(N)/10^N$.

Ebbe belekalkulálva (TF2)-őt vagy (TFátl)-t jön a végső konklúzió:

(BT4) $bt(N)=\Theta(N^{N+2}/10^N)= \Theta((N/10)^N \cdot N^2)$,

vagy

(BT5) $bt(N)=\Theta(\textit{átl}(N)\cdot N^N/10^N)= \Theta(\textit{átl}(N)\cdot(N/10)^N)$.

Annyi mindenesetre látszik, hogy a backtrack-es megszámolásban is egy exponenciális „gyors” algoritmust köszönthetünk, bár nem lebecsülendő az exponenciálisan növekedő nyereség, amit (BT3) fejez ki legtömörebben.

FÜGGELÉK

F1. Az ütésvizsgálatok számának meghatározása

A teljes keresés (TF) algoritmusa az N-szintű, N^N levél-elemű „állapotfa” összes ágát megvizsgálja, hogy egy megoldás-állapothoz tartozik-e. Az ág és az N-vezér elhelyezés kapcsolata világos: ha az ág i. pontja az i–1. pont j. irányú leágazása (i,j∈[1..N]), akkor ez azt jelenti, hogy az i. vezér a saját (i,) oszlopának a j. mezőjén van; az ág 0. pontja a fa gyökéreleme.

Egy N-vezér elhelyezés megoldásként elfogadható, ha bárhogyan kiválasztott (különböző) vezérpár nem ütik egymást. Ennek eldöntése legkevesebb 1, legtöbb „N alatt a 2” (azaz N·(N–1)/2) ütésvizsgálatot kíván. De vajon milyen átlagos értékkel számolhatunk a (TF2) és a (BT5) formulákban? Az előzőekben feltettük, hogy ez az érték $c{\cdot}N^2$ függvénnyel (is) jellemezhető, amelyben szereplő c N-től független pozitív konstans.

Hogy így van-e, próbáljunk empirikusan választ találni. Úgy írjuk meg a TF programot♠ (vezermodb0.exe), hogy –hasonlóan a (BT)-hez– számláljuk a szükséges ütésvizsgálatok számát. Ezzel ugyan a 2.5.–1. ábra-n szereplő táblázat 6. oszlopának tartalmát közvetlenül megkapjuk, s így már semmi szükség a frissen feltett kérdésre, mégis a probléma kihívása miatt találjuk meg a keresett, korábban *átl*(N)-nel jelölt függvényt.

A programmal fájlba generáljuk az egyes N-ekhez a megoldás- és az ütésvizsgálatok számát, amit egy táblázatkezelőben dolgozzuk fel. A fájl-import után kibővítjük a korábbi táblázatban is meglévő N^N és átlagként az (N/2 alatt 2)N^N formulákkal definiált oszlopokkal. Így összevethetők lesznek a becsült és a mért értékek.

Vezérszám	Teljes fa-vizsgálat (a VezerMoDb0.PAS programmal)				
	Megoldás-szám	levélszám	Ütésvizsgálat-szám		
N	MoDb	N^N	(N/2 alatt a 2)·N^N	méréssel	Idő (mp)
4	2	256	2 048	443	0,0
5	10	3 125	234 375	6 730	0,0
6	4	46 656	1 679 616	120 047	0,0
7	40	823 543	461 184 080	2 463 326	0,1
8	92	16 777 216	24 696 061 952	57 314 107	3,0
9	352	387 420 489	3 068 370 272 880	1 464 411 970	57,0
10	724	10 000 000 000	181 000 000 000 000	42 199 972 040	1898,3

F1.–1. ábra. A táblázat az importálás és oszlopbővítés után.

> Megjegyzés: a program kizárólag „tájékoztatólag” megjeleníti a (csak számításra, s nem a fájlba vitelre!) felhasznált időt is. Ebből is jól látszik az, ami várható volt, hogy a szükséges idő (egy adott N-től kezdődően) igen meredeken nőni kezd. Az N=10-re már olyan jelentős a futási idő, hogy nagyobb N-ekre már nem is kísérletezünk. Vagyis a pusztán kihívásból valós szükségszerűséggé vált a keresett formula megtalálása.

A különbség elszomorító: pl. 2 048 helyett ténylegesen csak 443 (N=4 esetén), 234 375 helyett csak 6 730 (N=5 esetén). Ezek (és a többi N-re is) súlyosan eltérnek. Tehát komolyan kell vegyük a kihívást!

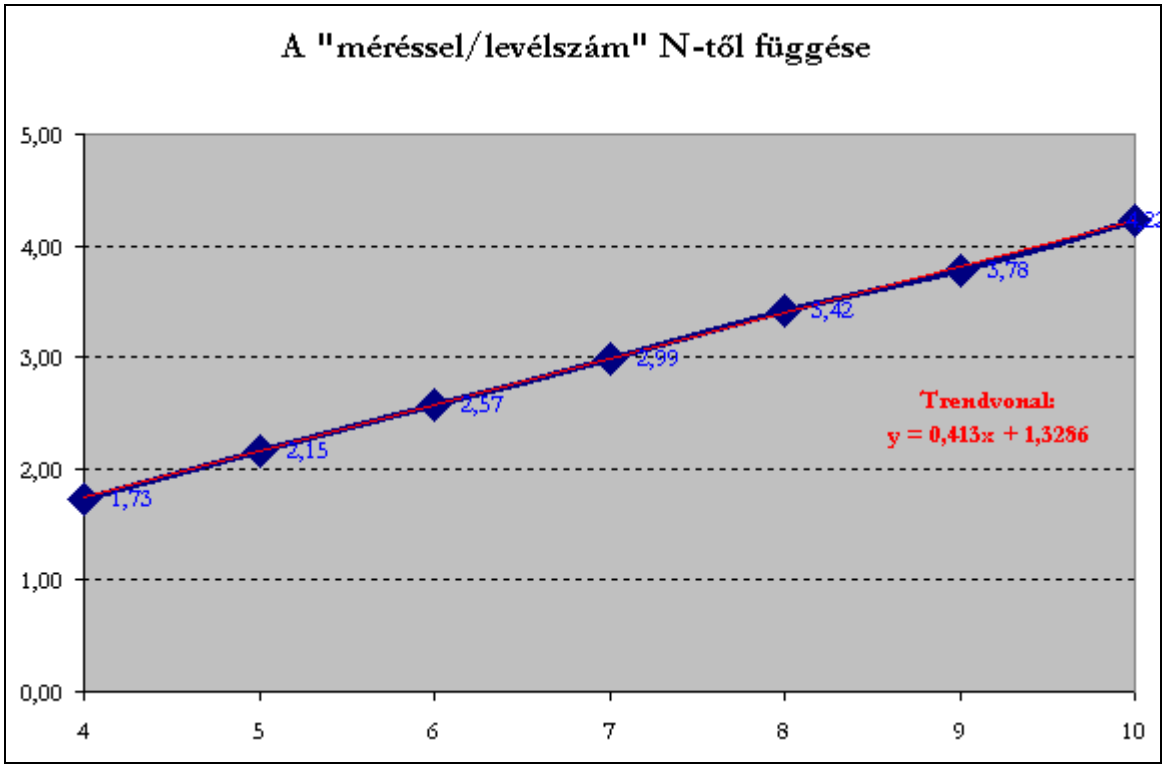
♠ Ennek forráskódja az F2 függelékbe található.

Első kérdésünk, hogy legalább a mért érték és a levélszám között van-e könnyen érzékelhető kapcsolat. Ennek megválaszolására kiegészítjük egy oszloppal, amelyben a „méréssel/levélszám” formula szerepel. Ez megnyugtatóan „egyenletesnek” látszik.

Egy vonaldiagramon ábrázolva csak erősödik a gyanúnk, hogy lineáris a kapcsolat. A táblázatkezelő „trendvonal”-szolgáltatását felhasználva, kapunk is egy formulát az egyeneshez (y=0,413x+1,3286). Az ebben szereplő független változót kicserélve az N-re, és igazítva ehhez a formulát, jutunk a

(átl) $\quad \textit{átl}(N)=0{,}4134\cdot N+0{,}0884$

összefüggéshez. Hát ez jelentősen (függvény-kategóriában is!) eltér az általunk feltételezettől.



F1.–2. ábra. A méréssel/levélszám hányados alakulása, diagrammal.

A korábbi 14 vezérig a táblázatunkat kiegészítjük, most már a formula „jóslásával”.

Vezérszám N	Teljes fa-vizsgálat (a VezerMoDb0.PAS programmal) Megoldás-szám MoDb	levélszám N*	Ütésvizsgálat-szám (N/2 alatt a 2)*N*	méréssel	Idő (mp)	A "méréssel/levélszám" N-től függése méréssel/levélszám	0,4134*N+0,0884	(0,4134*N+0,0884)*levélszám
4	2	256	2 048	443	0,0	1,7305	1,7420	446
5	10	3 125	234 375	6 730	0,0	2,1536	2,1554	6 736
6	4	46 656	1 679 616	120 047	0,0	2,5730	2,5688	119 860
7	40	823 543	461 184 080	2 463 326	0,1	2,9911	2,9822	2 455 970
8	92	16 777 216	24 696 061 952	57 314 107	3,0	3,4162	3,3956	56 968 715
9	352	387 420 489	3 068 370 272 880	1 488 228 276	54,6	3,8414	3,8090	1 475 684 643
10	724	10 000 000 000	181 000 000 000 000	42 686 094 229	2869,3	4,2686	4,2224	42 224 000 000
11	2680	285 311 670 611	25 232 964 148 836 800				4,6358	1 322 647 842 618
12	14200	8 916 100 448 256	4 557 910 549 148 470 000				5,0492	45 019 174 383 334
13	73712	302 875 106 592 253	1 015 811 608 499 330 000 000				5,4626	1 654 485 557 270 840
14	366596	11 112 006 825 558 000	199 062 757 122 439 000 000 000				5,8760	65 294 152 106 978 900

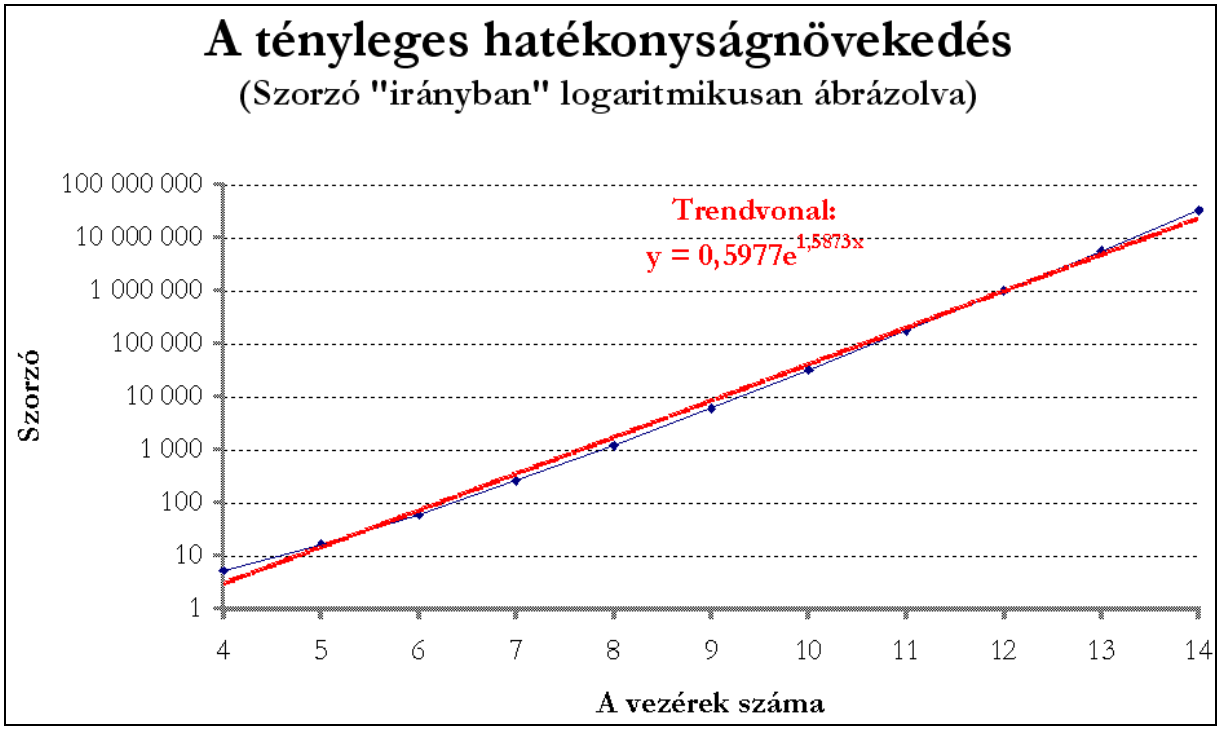
F1.–3. ábra. A táblázat újabb oszlopbővítés, valamint extrapolált sorok beillesztése után.

Az „új” oszloppal újraszámoljuk a tényleges hatékonyságnövekedést. Ebből is arra következtethetünk, hogy N növekedtével exponenciálisan nő a hatékonyság. Végülis a BT globális hatékonysága továbbra is exponenciálisnak tűnik. A (BT5)-ből kapjuk a végső hatékonysági összefüggést:

(BT6) $bt(N)=\Theta((0{,}4134\cdot N+0{,}0884)\cdot(N/10)^N)= \Theta(N\cdot(N/10)^N)$.

(0,4134*N+0,0884)*levélszám	Hatékonyságnövekedés
446	5
6 736	17
119 850	59
2 455 970	264
56 968 715	1 219
1 475 684 643	6 073
42 224 000 000	32 541
1 322 647 842 618	178 338
45 019 174 383 334	991 679
1 654 485 557 270 840	5 662 506
65 294 152 106 978 900	32 713 191

F1.–4. ábra. A táblázat újabb oszlopbővítés, valamint extrapolált sorok beillesztése után.



F1.–5. ábra. A tényleges hatékonyságnövekedés alakulása diagramon, az értéket logaritmikus skálán ábrázolva.

F2. A TF-program, hatékonysági mérésekkel

A program elvi „újdonságait”, algoritmikusa szempontból érdekes részeit pirossal emeltük ki.

```
Program N_Vezer_Osszes_Megoldasok_Szama;{FP-változat}
(*
   Teljes kereséses változat,
   hatékonysági mérésekkel.
   (A VezerMoDb2.pas "rokona":
   azaz egy megoldás megtalálásakor az aktuális állapotot fájlba rögzíti,
```

```
   majd -esetleg később- újra indítva a fájlbeli állapottól folytatható.)
*)
  Uses
     Crt,Dos{Get/SetTime-hoz};
  Const
     MaxN  = 16;
     fN    = 'VezMoDb0.tmp'; {a munkafájl neve}
  Type
     Index = 0..MaxN+1;
     Hol   = Array [1..MaxN] of Index;
  Var
     Vezer : Hol;
     N,i,j,
     hova  : Index;
     lehet : Boolean;
   { megjelenítési paraméterek: }
     moDb: LongInt;     { megoldás-szám }
     utVizsDb: Comp;    { ütés-vizsgálatok száma }
     folyt: Boolean;    { folytatás-e vagy újrakezdés }
     Kezdet,Veg,
     ElteltIdo: LongInt;

  Procedure OraIndul;
    Var
       o,p,mp,szmp: Word;
       ol: LongInt;
  Begin
    GetTime(o,p,mp,szmp); ol:=o; {konverzió LongInt-tá}
    Kezdet:=szmp+100*(mp+60*(p+60*ol{konverzió LongInt-tá}))
  End; {OraIndul}

  Procedure OraAll;
    Var
       o,p,mp,szmp: Word;
       ol: LongInt;
  Begin
    GetTime(o,p,mp,szmp); ol:=o; {konverzió LongInt-tá}
    Veg:=szmp+100*(mp+60*(p+60*ol{konverzió LongInt-tá}));
  End; {OraAll}

  Procedure Inicializalas(Var n: Index);
    Var
      i,j: Integer;
      f  : Text;
      c  : Char;
  Begin
    Window(1,1, 80,25); TextColor(White); TextBackGround(Black); ClrScr;
    Writeln('N-vezérprobléma':48);

    Write('Folytatás, fájlból (I/.)?:'); c:=ReadKey;
    folyt:=UpCase(c)='I';
    If folyt then
    Begin
      {$i-}
      writeln; writeln('Fájlból folytatom:');
      assign(f,fN);
      reset(f);
      {$i+}
```

```
      If IOResult<>0 then
      Begin
        Writeln('Nincs meg a folytatáshoz szükséges paraméterfájl.');
        ReadKey; Halt;
      End;
      readln(f,n); writeln(' N:',n);
      readln(f,moDb); writeln(' moDb:',moDb);
      readln(f,utvizsDb); writeln(' utvizsDb:',utvizsDb:12:0);
      writeln(' Utolsó megoldás:');
      for i:=1 to n do
      Begin
        read(f,Vezer[i]); write(Vezer[i]:3);
      End;
      writeln;
      readln(f,ElteltIdo);
      writeln(' ElteltIdo:',ElteltIdo);
      close(f);
    End
      else
    Begin
      Repeat
        GotoXY(1,5); Write('Vezérszám (3<N<',MaxN:2,'):'); Readln(n)
      Until n in [4..MaxN-1];
    End;
  End; {Inicializal}

  Procedure VanMegoldas(n: Index);
    Var
      i,j: Index;
      f  : Text;
  Begin
    OraAll;
    {$i-}
{
  TP-ben így műxik, FP-ben az törlés+átnevezés nem megy:
  ezért jelenik meg megoldásonként a "még nincs fájl" uzenet
}
    assign(f,'_'+fN); Erase(f);
    If IOResult<>0 then
      Writeln('Még nincs backup tmp-fájl.');
    assign(f,fN); Rename(f,'_'+fN);
    If IOResult<>0 then
      Writeln('Még nincs tmp-fájl.');
    assign(f,fN);
    Rewrite(f);
    If IOResult<>0 then {nem írható a fájl, pl. nyitva van}
    Begin
      Writeln('Fájlíráshiba.');
      assign(f,'_'+fN);
      Rename(f,fN);
      ReadKey;
      Halt(1);
    End;
    {$i+}
    writeln(f,n,' -- n');
    writeln(f,moDb, ' -- moDb');
    writeln(f,utvizsDb:12:0, ' -- utvizsDb');
    For i:=1 to n do
    Begin
      write(f,Vezer[i]:3);
    End;
```

```
    writeln(f);
    ElteltIdo:=ElteltIdo+Veg-Kezdet;
    writeln(f,ElteltIdo, ' -- ElteltIdo');
    close(f);
    OraIndul;
  End; {VanMegoldas}

  Procedure OsszMegoldas(n: Index);
    Var
      c: Char;
  Begin
    Window(1,1, 80,25); TextColor(White); TextBackGround(Black); ClrScr;
    Writeln('Az N-vezérproblémának N=',n,' esetre ',moDb,
            ' db megoldása van.');
    Writeln('Az összes ütés-vizsgálatok száma:',utVizsDb:12:0,'.');
    ElteltIdo:=ElteltIdo+Veg-Kezdet;
    writeln('Eltelt idő:',ElteltIdo/100:5:2,'mp.');
    Writeln('Kilépés ENTERrel.');
    Repeat
      c:=ReadKey;
    Until c=#13;
  End; {OsszMegoldas}

  Function JoElhelyezes: Boolean;
    Var
      i,j:Integer;
      OK:Boolean;
  Begin
    OK:=True;
    i:=1;
    While (i<N) and OK do
    Begin
      j:=i+1;
      While (j<=N) and OK do
      Begin
        OK:=(Vezer[i]<>Vezer[j]) and (Abs(Vezer[i]-Vezer[j])<>j-i);
        Inc(j);
        utVizsDb:=utVizsDb+1; {ütésvizsgálat-számlálás}
      End;
      Inc(i);
    End;
    JoElhelyezes:=OK
  End; {JoElhelyezes}

Begin {Főprogram}
  Inicializalas(N);
  If not folyt then
  Begin
    utVizsDb:=0; moDb:=0;
    For j:=1 to N do Vezer[j]:=1;
  End;
  OraIndul;
  i:=1;
  While i>0 do
  Begin
    j:=N;
    While (j>0) and (Vezer[j]=N) do
    Begin
      Vezer[j]:=1;
      Dec(j);
    End;
```

```
    i:=j;
    If j>0 then
    Begin
      Inc(Vezer[j]);
      If JoElhelyezes then
      Begin
        Inc(moDb);
        VanMegoldas(N);
      End;
    End;
  End;
  OraAll;
  OsszMegoldas(N);
End.
```