SEND+MORE=MONEY in J

Joachim Hoffmann DSK-536\\SENDMM

Neue Lösungswege für das in PCNEWS edit-49 vorgestellte Problem zur Lösung eines Rechenrätsels. Dort, in PCNEWS edit-49 diente es als Beispiel für die Anwendung einer Rekursion. Heute dient es uns als Demonstration einer Implementierung in der Sprache J.

Gleichungssystem mit Überträgen

```
  SEND
  MORE
  ----
 MONEY
```

als homogenes Gleichungssystem, mit G, H, I, J als Überträgen.

```
    S   E   N   D   M   O   R   Y   G   H   I   J

1   0   1   0   1   0   0   0  _1 _10   0   0   0    (D+E  )=Y+10*G
2   0  _1   1   0   0   0   1   0   1 _10   0   0    (N+R+G)=E+10*H
3   0   1 _ 1   0   0   1   0   0   0   1 _10   0    (E+O+H)=N+10*I
4   1   0   0   0   1  _1   0   0   0   0   1 _10    (S+M+I)=O+10*J
5   0   0   0   0  _1   0   0   0   0   0   0   1         J =M
```

Damit man in diesem unterbestimmten Gleichungssystem die unabhängigen Variablen von den abhängigen unterscheiden kann, normiert man es mittels Gauß'scher Elimination. Die führenden Einser in jeder Zeile zeigen auf die unabhängigen (S, E, N, O, M), D, R, Y und G, H, I, J sind die abhängigen.

```
    S   E   N   O   M   D   R   Y   G   H   I   J

1   1   0   0  _1   1   0   0   0   0   0   1 _10
2   0   1  _1   1   0   0   0   0   0   1 _10   0
3   0   0   1   0   0   1   1  _1  _9 _10   0   0
4   0   0   0   1   0   0   1   0   1  _9 _10   0
5   0   0   0   0   1   0   0   0   0   0   0  _1
```

Indem man für die abhängigen Variablen Werte auswählt und einsetzt, können die unabhängigen bestimmt werden. Der Vorteil dieser Methode ist, daß die möglichen Kombinationen auf ein Minimum eingeschränkt werden. D, R, Y können jeweils Werte von 0 bis 9 einnehmen, aber G, H, I, J jeweils nur 0 oder 1.

Es sind daher 16000=(10^3)*(4^2) Möglichkeiten zu überprüfen, im Gegensatz zu 3628794=(!10)-(!10-7) unter der Berücksichtigung, daß M=1 ist, wovon ich hier keinen Gebrauch mache.

Ein weiterer Vorteil dieser Lösung ist, daß man ALLE möglichen Lösungen geliefert bekommt. Die Lösungen, bei denen keine Ziffer zweimal verwendet wird, können leicht selektiert werden.

```
    _10 TAKE result   NB. Die letzen 10 Loesungen
09221=8229+0992
09332=8339+0993
09443=8449+0994
09554=8559+0995
09665=8669+0996
09776=8779+0997
09887=8889+0998
09998=8999+0999
00109=0019+0090
10109=9019+1090

   EXECUTE _10 TAKE result  NB. alle richtig?
1 1 1 1 1 1 1 1 1 1         NB. nur 1'er => alles OK

      OPEN _1 FROM result
10652=9567+1085            NB. Maximum ohne doppelte Ziffern
11099=9900+1199            NB. Maximum mit doppelten Ziffern

 OPEN COUNT each result
  25   NB. Loesungen ohne doppelte Ziffern
1130   NB. Loesungen mit doppelten Ziffern
   2   NB. ist die Abzahl der Maxima, trivial
```

```
NB. ====================== Beginn des J Scripts ======================

charsub=:  3 : 0
NB. characterpairs charsub data
NB. For example:
NB.    '-_$ ' charsub '$123 -456 -789'
NB.  123 _456 _789
NB. Use <rplc> for arbitrary string replacement.
:
'ft'=. ((#x.)$0 1)<@,&a./.x.
t {~ f i. y.
)

NB.  Hier werden Einzeiler definiert und Namen fuer die Symbole der Stammverben
NB.  vergeben.

X  =: +/ . *                           NB. Matrizenprodukt
f  =: {:@]*{.@] { [
F  =: [:+/(1:,{:@])*(}:@] { [)
g  =: {.@]
TAU=: E1=: <@] C. [                NB. AusTAUschen
MUL=: E2=: f`g`[}                  NB. MULtiplizieren mit Konstante
AMU=: E3=: F`g`[}                  NB. Zeile(1) + Zeike(2)*Konstante(3)
CCO=: (<@] C. i.@{:@$@[)  {"1  [   NB. ChangeColumns of Matrix
 TAKE =: {.                        NB. Nehmen
 DROP =: }.                        NB. Loeschen
CTAKE =: {."1                         NB. ColumnTake=Spalten nehmen
CDROP =: }."1                           NB. ColumnDrop=Spalten loeschen
PositiveDigits=: */@(0&<: *. <:&9)"1 # ]  NB. positive Ziffern selektieren
PAIRS  =: ,@,.        NB. Paaren= aufreihen nach annaehen
TEXTYFY=: -.&' '@:":  NB. ohne_Leerzeichen nach deaktivieren
EXECUTE=: ".          NB. Ausfueren (eines Textes)
COUNT=: #      NB. Anzahl der Elemente im rechten Argument
COPY =: #      NB. Einser in links nehmen entsprechende Elemente aus rechts
NOT  =: -.     NB. logisch nicht
NUB  =: ~.     NB. Kernmenge= keine Doppelten= nur Unikate
MATCH=: -:     NB. Test auf Identitaet
LINK =: ;      NB. Zwei Nomen verbinden => geschachtelte Liste
rowwise  =: "1 NB. Adverb: etwas zeilenweise anwenden
ANTIBASE2=: #: NB. konvertiert Dezimalzahl in Binaerzahl
TRANSPOSE=: |: NB. Zeilen mit Spalten vertauschen=Transponieren
OPEN  =: >     NB. oeffnet geschachtelte Liste zur Matrix
MAX   =: >./   NB. Maximum einer Liste
LEFT  =: [     NB. Identitaetsverb: Ergebnis ist linkes  Argument
RIGHT =: ]     NB. Identitaetsverb: Ergebnis ist rechtes Argument
INDEXOF =: i.  NB. Indizes von rechts in links
FROM=: {       NB. Index links aus Array rechts

NB.........................................................................
NB. SOLVE ist das Hauptprogramm, das wichtigste Verb in SOLVE ist TRY.


SOLVE=: 3 : 0
txt =: 'MONEY=SEND+MORE'
cols=:'SENDMORYGHIJ'
NB. Eingabe des Gleichungssystems als geschachtelte Liste
NB. sollte eigentlich automatisch aus txt generiert werden
mat=: 0 0 0 1 0; 1 _1 1 0 0; 0 1 _1 0 0; 1 0 0 0 0; 0 0 0 1 _1
mat=: mat,0 0 1 _1 0; 0 1 0 0 0; _1 0 0 0 0; _10 1 0 0 0; 0 _10 1 0 0
mat=: mat,0 0 _10 1 0; 0 0 0 _10 1
mat=: TRANSPOSE OPEN mat

NB. "haendisches" Normieren der Matrix, Gausssche Elimination
mat =: TAU&3 4 AMU&3 2 _1  AMU&3 1 _1  AMU&2 3 1 TAU&1 2 TAU&0 3 mat
mat =: CCO&5 3  TAU&3 4 MUL&4 _1 MUL&3 _1 mat
cols=:   (<5 3)C. cols        NB. Vertausche Variable 5. mit 3.

dep  =: 5 CDROP  mat          NB. abhaengiger Teil des Gleichungsystems
indep=: 5 CTAKE  mat          NB. Unabhaengiger Teil des Gleichungsystems(5x5)
TRY  =: %.&indep@:(-@(dep&X)) NB. Verb TRY setzt abhaengige Variablen ein und
                              NB. rechnet die unabhaengigen () aus
                              NB. %. ist der Gleichungsloeser
all2p4=: ANTIBASE2 i. 2^4     NB. ghij: all arrangements of 0 1 in groups of 4
                              NB. #: uebersetzt 0 1 2 ... 14 15 ins Binaersystem
all2  =: ,/ ,"0/~ i.10        NB.  RY: alle Ziffern-Arrangements in 2-er Gruppen
all3  =: ,/ all2 ,"1 0/ i.10  NB. DRY: alle Ziffern-Arrangements in 2-er Gruppen
all   =: ,/ all3 ,"1/ all2p4  NB. alle 1600 Kombiantionen der abhaengigen Variablen
                              NB. Das Wichtigste kommt jetzt!!!
res=: TRY"1  all              NB. Einsetzen und unabhaengige Vars. bestimmen
                              NB. 1600x5 Matrix mit allen Loesungen fuer die
unabhaengigen

NB. Jetzt werden die Zeilen die nur positive Ziffern enthalten selektiert
NB. Als Ergebnis erhaelt man alle Kombinationen fuer alle acht Buchstaben
res0 =: PositiveDigits res ,. 3{."1 all

res  =: TEXTYFY"1 res0       NB. Zahlen in Texte ohne Leerzeichen umwandeln
chars=: _4 DROP cols         NB. ohne Hilsvariablen GHIJ
res  =: chars PAIRS"1 res    NB. Buchstaben-Zahlen-Paare fuer's austauschen
res  =: res charsub"1 txt    NB. Ersetzen der Buchstaben durch Zahlen

boo=.(COUNT@NUB MATCH COUNT)rowwise res0
uniques=.boo COPY res        NB. Loesungen ohne doppelte Ziffern
doubles=.((NOT boo)COPY res) NB. Loesungen mit doppelten Ziffern

NB. selektiert groesste Loesung aus Matrix mit den Loesungen in den Zeilen
MAXIMUM=. (RIGHT INDEXOF MAX)@:EXECUTE@(5&CTAKE) FROM RIGHT

result=. uniques ( LEFT ; RIGHT ; MAXIMUM@LEFT ,: MAXIMUM@RIGHT) doubles
NB. Das Ergebnis ist eine geschachtelte Liste mit 3 Schachteln
NB. 1.) Loesungen ohne doppelte Ziffern
NB. 2.) Loesungen mit doppelten Ziffern
NB. 3.) die maximalen Loesungen aus 1.) und 2.)
)

NB.........................................................................
NB. Aufruf des Verbs SOLVE mit leerem Argument

result=: SOLVE''

NB. The End
```

Rekursive Pascal-Lösung

Nachfolgend eine J und eine Pascal Version der rekursiven Lösung für das SEND+MORE=MONEY Problem.

Ein wesentlicher Unterschied zum Original ist, daß zum Testen eine Lösung die Buchstaben im Text 'MONEY=SEND+MORE' durch Text-Ziffern ersetzt werden, und dann der Text mit Ziffern ('56328=1234+5672') ausgeführt wird.

Das Programm kann auch andere Gleichungen lösen.

```
   (2 to 7) solve_rec_p 'CC=BA+AB'     NB. pascal style J verb
+--------+-+-+
|CC=BA+AB|8|1|
|55=32+23| | |
|66=42+24| | |
|77=52+25| | |
|55=23+32| | |
|77=43+34| | |
|66=24+42| | |
|77=34+43| | |
|77=25+52| | |
+--------+-+-+

   (2 to 5) solve_rec_j 'CC=BA+AB'     NB. partly functional defined J verb
+--------+-+-+
|CC=BA+AB|2|1|
|55=32+23| | |
|55=23+32| | |
+--------+-+-+

NB.    A        B        C
   ((2 to 5);(4 to 5);(6 to 9)) solve_rec_j 'CC=BA+AB'
+--------+-+-+
|CC=BA+AB|6|1|
|66=42+24| | |
|77=52+25| | |
|77=43+34| | |
|88=53+35| | |
|99=54+45| | |
|99=45+54| | |
+--------+-+-+

   statistics each 'permute 0';'permute_rec_pascal 0'
+-------------------------+-----------------------------+
|+--------+--------------+|+--------+-------------------+|
||        |   permute 0  |||        |permute_rec_pascal 0||
|+--------+--------------+|+--------+-------------------+|
||usedtime|     53.89    |||usedtime|       45.75        ||
|+--------+--------------+|+--------+-------------------+|
|| result |+---------+-+-+||| result |  +---------+-+-+   ||
||        ||CCD=AB+BA|6|1|||||        |  |CCD=AB+BA|6|1|   ||
||        ||110=28+82| | ||||        |  |110=28+82| | |   ||
||        ||110=37+73| | ||||        |  |110=37+73| | |   ||
||        ||110=46+64| | ||||        |  |110=46+64| | |   ||
||        ||110=64+46| | ||||        |  |110=64+46| | |   ||
||        ||110=73+37| | ||||        |  |110=73+37| | |   ||
||        ||110=82+28| | ||||        |  |110=82+28| | |   ||
||        |+---------+-+-+|||        |  +---------+-+-+   ||
|+--------+--------------+|+--------+-------------------+|
+-------------------------+-----------------------------+

======================================================================
NB. Script MONEYREC3.JS
NB. Version 3
NB. .........................................................
NB. Rekursiv:
NB. MONEY=SEND+MORE
NB. 10652=9567+1085            NB. Maximum ohne doppelte Ziffern
NB. 11099=9900+1199            NB. Maximum mit doppelten Ziffern
NB. .........................................................

to   =: [ + i.@>:@-~      NB. 2 to 6 => 2 3 4 5 6
check=: */@:".@:}.      NB. drop first line and execute each line
show =: (1!:2)&2          NB. show on screen
fmt  =: (0 1 1)&":

define_globals=: 3 : 0
NB. defined global variables for
NB. permute and permute_rec_pascal
NB. the right argument y. is 'MONEY=SEND+MORE'
NB. the optional left argument is a boxed list with allowed values
(<0 to 9) define_globals y.
:
txt=: y.                        NB.
variables=: /:~ (~.txt)-.'=*+-%^'  NB. extract variables and sort them
permutations=: (#variables)#0  NB. this is the global permutation
variable
used  =: 10#0         NB. this boolean list indicates, if a digit is used
result=: ,:txt              NB. initialize result as matrix with one row
max   =: # variables          NB. max is tally of variables
permusets=: max $ x.          NB.  allowed values
to=: [ + i.@>:@-~          NB. 2 to 6 => 2 3 4 5 6
NB.      allowed values for S   E   N   D   M   0   R   Y
NB. permusets  =: to/each 1 9;0 9;0 9;0 9; 1 ;0 9;0 9;0 9
NB. permusets  =: to/each 5 9;5 7;3 7;6 9; 1 ;0 2;6 9;0 3
getpermuset=: >@{&permusets    NB. get values for variable i
NB. find all indices of x. in y.
NB. result is a boxed list, each box contains all occurences
NB. of each atom in x., a box may also be empty
NB.  inxin=: Decrement@nozero each@Box rowwise@:Transpose@Find"0 1
inxin     =:(<:@(-.&0)&.>)@(<"1)@:|:@:(* >:@i.@#)@E."0 1
partitions=: ;@:(# each)@] # i.@#@]

NB. boxed list with multiple indices of each variable in txt
NB. , e.g. E has 3 occurrences in txt
indices=: variables inxin txt
blowup =: partitions indices      NB. how the boxed list is partitioned
indices=: ; indices               NB. indices is now a simple list

replace=: indices}&txt@(blowup&{) NB. replace var. in txt by text-digits
execute=: ".           NB. execute a text,e.g.: execute 56328=1234+5672'
textify=: -.&' '@":              NB. list of digits=>text
                                 NB. and delete the blanks
empty  =: (0 0$0)"_              NB. returns empty matrix => no effect
quote  =: ('''''&,)@(,&'''')     NB. append left and right singl quotes

NB. append a solution to global result
solution=: execute@:('result=: result,'&,)@quote
testit=:(empty`solution@.execute)@:replace@:textify@:".@('permutations'"_)
permute =: testit`(] substitute"0 getpermuset)@. (max&>)
empty''                          NB. return empty result
)

NB. substitutes digits for i-th char and calls recursively permute
NB. x. substitute y. => value y. is written at postion x. (i) of
NB.  global list permutations (is b in original Pascal program)
NB. We are forced to use direct definition because of global assignments.
substitute=: 4 : 0
i=. x.                           NB. because
if. -.y.{used do.
 permutations=: y. i}permutations NB. write digit to i-th pos of
permutations
 used        =: 1 y.}used         NB. mark this digit as used
 permute i+1                      NB. !!! the recursive call !!!
 used=:0 y.}used                  NB. unuse digit y.
end.
empty''
)

NB. solve equatation using partly functional J style
NB. e.g.: (0 to 9) solve_rec_j 'CC=BA+AB'

solve_rec_j=: 4 : 0
(boxopen x.) define_globals y.
permute  0 NB.<:#variables             NB. permute positions 7 to 0
(];<:@{.@$;check) result
)

NB. ===========================================================
NB. === Pascal-style loop-recursion of send+more=money problem ========
NB. e.g. permute_rec_pascal 0  start is 0 because of 0-origin

permute_rec_pascal=: 3 : 0             NB. x. is i.th variable
i=. y.
vals=. getpermuset i                NB. get all allowed digits for i
 whilst. 0~:#vals=.}.vals do.       NB. loop over all digits for i
    val=.{.vals                     NB. current value for variable
    if. -.val{used do.              NB. exit if digit is already used
    permutations=: val i}permutations  NB. set i-th position to val
    used        =: 1 val}used          NB. mark this digit as used
      if. i < max-1 do.                NB. if not last variable,
                                       NB. 0-origin!
      permute_rec_pascal i+1         NB.   recursive call
      else.                          NB. else
      NB. replace variables in txt by digits => '10652=9567+1085'
      sol=. replace textify permutations

          if. execute sol do.     NB. test if permutation is solution
          NB. show sol            NB. display number on screen
          result=:result,sol      NB. global result
          end.
      end.                        NB. end if
    used=: 0 val}used     NB. now we have all permuted=>free digit i
    end.
 end.                               NB. end whilst
)

NB. solve equatation using pascal style recursion
NB. e.g.: (0 to 9) solve_rec_p 'CC=BA+AB'

solve_rec_p=: 4 : 0
(boxopen x.) define_globals y.
permute_rec_pascal 0               NB.
(];<:@{.@$;check) result
)

NB. verb to compare two vers
NB. eg.: fmt statistics each 'permute 0';'permute_rec_pascal 0'

NB.==========================================================

statistics=:3 : 0
(<0 to 9) define_globals 'CCD=AB+BA'     NB. 'MONEY=SEND+MORE'
usedtime=:timex y.
boo=. */ ". }. result
description=.  ;: 'usedtime result'
fmt ('';y.), description,. usedtime;< (];<:@{.@$;check) result
)
```