Program A (*Multiplication of permutations in cycle form*)

```
001 t          IS      $255
002 tt         GREG    0
003 _lpren     GREG    #20202028                  "␣␣␣("
004 _rpren     GREG    #20202029                  "␣␣␣)"
005 _tag       GREG    #80000000                  Set MSB of a tetra
006 _untag     GREG    #7FFFFFFF                  Turn off the MSB of a tetra
007 _nlnull    GREG    #0a000000                  Newline and a zero byte
008 ip         IS      $2                         Pointer for input permutation
009 op         IS      $3                         Pointer for output permutation
010 p          IS      $4                         A symbol of the permutation
011 size       IS      $5                         The size of the input permutation
012 start      IS      $6                         The variables of the algorithm
013 current    IS      $7
014            LOC     Data_Segment
015            GREG    @
016 NoArg      BYTE    "Missing argument: file with input permutation expected",#a,0
017 NoFile     BYTE    "Can't open the file given in first argument.",#a,0
018 BUFSIZE    IS      80+1+1                     80 Bytes plus newline can be read
019 INP        IS      3                          Handle for input file
020 ArgIn      OCTA    0,TextRead                 First octabyte is later filled with argument
021 ArgRead    OCTA    0,BUFSIZE                  Ditto
022 Perm       GREG    @                          Location to store the permutations
023            LOC     #100
024 Error1     LDA     t,NoArg
025            JMP     PrtAns
026 Error2     LDA     t,NoFile
027            JMP     PrtAns
028 Main       SET     tt,Perm
029            LDO     t,$1,8
030            BZ      t,Error1                   No argument: error
031            STO     t,ArgIn                    Otherwise use the argument.
032 OH         LDA     t,ArgIn                    Open input file.
033            TRAP    0,Fopen,INP
034            BN      t,Error2                   −1 indicates an error.
035 ReadLine   STO     tt,ArgRead                 Read the input.
036            LDA     t,ArgRead
037            TRAP    0,Fgets,INP
038            BN      t,EndRead
039            ADD     tt,tt,t
040            SUB     t,tt,t                     Output the input line.
041            TRAP    0,Fputs,StdOut
042            SUB     tt,tt,1                    Remove the newline byte.
043            JMP     ReadLine
044 EndRead    TRAP    0,Fclose,INP               Close the input file.
045            SUB     op,tt,Perm          1      Start output after the equal sign.
046            SUB     size,op,4           1
047            SET     ip,0                1      ip ← 0.
048 A1         LDT     p,Perm,ip           A      A1. First pass. Load symbol into p.
049            CMP     t,p,_lpren          A      Is p a left parenthesis?
050            PBNZ    t,0F                A      No, jump to test for right parenthesis.
051            OR      p,p,_tag            B      Yes, tag the left parenthesis.
052            STTU    p,Perm,ip           B
```

```
053             ADD     ip,ip,4                 B
054             LDT     p,Perm,ip               B   Get the next symbol and
055             OR      tt,p,_tag               B     add a tag to it.
056  0H         CMP     t,p,_rpren              C   Is it a right parenthesis?
057             PBNZ    t,0F                    C   No, test if the end is reached.
058             STTU    tt,Perm,ip              D   Yes, replace this parenthesis.
059  0H         ADD     ip,ip,4                 C
060             CMP     t,ip,size               C
061             PBNZ    t,A1                    C
062             JMP     A2                      1
063  A6         STT     _rpren,Perm,op          R   A6. Close. Output a right parenthesis.
064             ADD     op,op,4                 R
065             SUB     tt,op,3*4               R   Check for singleton cycle.
066             LDT     p,Perm,tt               R
067             CMP     t,p,_lpren              R   Appears a '(' two tetras earlier?
068             CSZ     op,t,tt                 R   Reset op if yes.
069  A2         SET     ip,0                    E   A2. Open. Set ip to the first element.
070  0H         ADD     ip,ip,4                 F   The leftist parenthesis is skipped.
071             CMP     t,ip,size               F
072             BZ      t,Done                  F   Exit at the end of input.
073             LDT     p,Perm,ip               G   Search untagged symbol.
074             PBN     p,0B                    G   Loop if tagged.
075             SET     start,p                 H   Set start.
076             STT     _lpren,Perm,op          H   Output a left parenthesis.
077             ADD     op,op,4                 H
078             STT     p,Perm,op               H   Output the element
079             ADD     op,op,4                 H
080             OR      p,p,_tag                H     and tag it.
081             STTU    p,Perm,ip               H
082  A3         ADD     ip,ip,4                 J   A3. Set CURRENT.
083             LDT     p,Perm,ip               J   Get next element and
084             AND     current,p,_untag        J     store it without a tag in current.
085             STT     current,Perm,size       J   Store it as sentinel.
086  A4         ADD     ip,ip,4                 K   A4. Scan formula.
087             LDT     p,Perm,ip               K   Load next symbol
088             AND     p,p,_untag              K     remove possible tag
089             CMP     t,p,current             K     and compare it to current.
090             PBNZ    t,A4                    K
091             CMP     t,ip,size               L
092             BNN     t,A5                    L   Branch if sentinel is reached.
093             OR      p,p,_tag                O   Element p equals current so tag it.
094             STTU    p,Perm,ip               O
095             JMP     A3                      O
096  A5         CMP     t,current,start         P   A5. CURRENT = START?
097             BZ      t,A6                    P   Yes, close the output cycle.
098             STT     current,Perm,op         Q   No, output current.
099             ADD     op,op,4                 Q
100             SET     ip,0                    Q   Start in A4 from the left.
101             JMP     A4                      Q
102  Done       ADD     size,size,4
103             LDA     t,Perm,size             Start output after the equal sign.
104             CMP     tt,op,size              Test if output is empty.
105             BNZ     tt,1F
```

```
106             STT     _lpren,t,0          Yes, so output the identity permutation.
107             STT     _rpren,t,4
108             ADD     op,size,8
109  1H         STT     _nlnull,Perm,op     Add newline and a null byte to output string.
110  PrtAns     TRAP    0,Fputs,StdOut
111             TRAP    0,Halt,0                                                    ▮
```

Analysis

This implementation in `MMIX` of Algorithm A has made some changes to the input format of the data compared to the `MIX` version. First the reader is replaced by a file and the output is sent to StdOut. It is assumed that StdOut can handle lines of arbitrary length. Next, the line length for the input is still 80 bytes but now 20 fields of 4 bytes each are formed. The length of an input line might be shorter so no fields with 4 spaces are considered. The equal sign is kept but it plays no role in the implementation. Its tetra in memory is used as a temporary storage. Third, all symbols are placed flush right in their fields.

Some aspects have not been changed. The MSB of the tetra that is build from the 4 bytes of a field is used to tag elements. So negative values indicate a tagged element as in the `MIX` implementation. No error checking for the input data is implemented. The frequency counts uses the same letters as in the old implementation if it was possible. The variable 'S' is no longer needed and a new frequency counter 'O' is introduced. And the implementation was done in a way that the loops in steps A2 and A4 use the same number of mems and oops. This allows to keep the analysis similar to the `MIX` program. Of course, a faster implementation is possible.

The algorithm needs without input and output $(A+2C+D+G+3H+2J+K+O+Q+2R)\mu + (8+3A+7B+5C+3D+E+3F+2G+9H+4J+5K+4L+4P+3O+4Q+8R)\upsilon$.

As in Eq. (8) of TAOCP V1, Section 1.3.3, the following equations hold:

$$A = C; \qquad E = R + 1; \qquad R = P - Q.$$

And we find the new equations

$$\begin{aligned} &F = E + G - H = G - H + P - Q + 1; \\ &G = F - 1; \\ &J = H + O \iff O = J - H; \end{aligned} \qquad \begin{aligned} &L = J + Q = H + O + Q; \\ &P = L - O = H + Q \iff Q = P - H. \end{aligned}$$

Applying these equations to the used mems and oops several variables are eliminated: $(2B+C+D+G+3H+3J+K+P)\mu + (7B+8C+3D+5G+7H+11J+5K+12P+12)\upsilon$.

Next, the following equations hold:

$$\begin{aligned} B + C &= \text{number of words of input;} \\ B &= \text{number of "(" in input;} \\ D &= \text{number of ")" in input;} \\ B = D &= \text{number of cycles in input;} \\ H &= \text{number of cycles in output (inclusive singletons);} \\ B + C = B + B + H + O &\iff J = C - B \text{ as all symbols get tagged once;} \\ P &= \text{number of distinct elements in input.} \end{aligned}$$

And the nontrivial relation is now $F + J + K = (B + C)(P + 1)$. The first left parenthesis is skipped but a sentinel is read.

With the variables of Eq. (19) the profile of the algorithm is $(NY + 4Y - 2M + N + 3U - 1)\mu + (5NY + 19Y - 10M + 12N + 7U + 7)\upsilon$.

Running the program with Eq. (6) as input the `MMIX`-simulator shows at the end: `1569 instructions, 330 mems, 1705 oops; 305 good guesses, 50 bad`. With this input it follows that $Y = 29$, $N = 7$, $M = 5$, and $U = 3$. The algorithm itself needs `1532 instructions, 324 mems, 1628 oops; 300 good guesses, 48 bad`. As expected the following equations hold: $11 * 29 - 10 + 7 + 9 - 1 = 324$ and $54 * 29 - 50 + 84 + 21 + 7 = 1628$.