NEWLY AVAILABLE SUPPLEMENT TO THE CLASSIC WORK

The MMIX Supplement

Supplement to
The Art of Computer Programming
Volumes 1, 2, 3
by Donald E. Knuth

MARTIN RUCKERT

THE MMIX SUPPLEMENT

Supplement to *The Art of Computer Programming* Volumes 1, 2, 3 by Donald E. Knuth

MARTIN RUCKERT *Munich University of Applied Sciences*

ADDISON–WESLEY

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montréal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

```
        GETA   q+3,:Neg
        PUSHJ  q,:Tree1        Q ← Tree1(Q,·,"−").
        JMP    D4
:Add    LDOU   t,q1,:INFO
        PBNZ   t,1F            Jump unless INFO(Q1) = 0.
        SET    t+1,q1
        PUSHJ  t,:Free         AVAIL ⇐ Q1.
        JMP    D3
1H      LDOU   t,q,:INFO
        PBNZ   t,1F            Jump unless INFO(Q) = 0.
2H      SET    t+1,q
        PUSHJ  t,:Free         AVAIL ⇐ Q.
        SET    q,q1            Q ← Q1.
        JMP    D3
1H      GETA   q+3,:Add
3H      SET    q+1,q1
        SET    q+2,q
        PUSHJ  q,:Tree2        Q ← Tree2(Q1,Q,"+").
        JMP    D3
:Sub    LDOU   t,q,:INFO
        BZ     t,2B            If INFO(Q) = 0, then −Q = +Q.
        GETA   q+3,:Sub        Prepare for Q ← Tree2(Q1,Q,"−").
        LDOU   t,q1,:INFO
        PBNZ   t,3B
        SET    t+1,q1
        PUSHJ  t,:Free         AVAIL ⇐ Q1.
        SET    q+1,q
        GETA   q+3,:Neg
        PUSHJ  q,:Tree1        Q ← Tree1(Q,·,"−").
        JMP    D3
:Mul    LDOU   t,q1,:INFO
        BZ     t,1F            Jump if INFO(Q1) = 0.
        SET    t+1,q1
        SET    t+3,p2
        PUSHJ  t+2,:Copy
        PUSHJ  t,:Mult
        SET    q1,t            Q1 ← Mult(Q1,Copy(P2)).
1H      LDOU   t,q,:INFO
        BZ     t,:Add          Jump if INFO(Q) = 0.
        SET    q+2,p1
        PUSHJ  q+1,:Copy
        SET    q+2,q
        PUSHJ  q,:Mult         Q ← Mult(Copy(P1),Q).
        JMP    :Add            ▮
```

Mult expects two parameters u and v; it returns an optimized representation of u × v.

```
:Mult   GET    rJ,:rJ
        SETMH  info,1          The constant "1" has INFO = 1 and DIFF = 0.
```

```
153          LDO    t,u,:INFO
154          CMP    t,info,t     Test if U is the constant “1”;
155          BZ     t,1F           jump if so.
156          LDO    t,v,:INFO    Otherwise,
157          CMP    t,info,t       test if V is the constant “1”,
158          GETA   v+1,:Mul       prepare third parameter,
159          BNZ    t,:Tree2       and if not so, return Tree2(U,V,“×”);
160          SET    t+1,v          else, pass V to Free.
161          JMP    2F
162    1H    SET    t+1,u        Pass U to Free.
163          SET    u,v          U ← V.
164    2H    PUSHJ  t,:Free      Free one parameter
165          PUT    :rJ,rJ         and return U.
166          POP    1,0          ▮
```

The last two routines `Div` and `Pwr` are similar and they have been left as exercises (see exercises 15 and 16).

EXERCISES

[*347*]

▸ **13.** [*26*] Write an `MMIX` program for the `Copy` subroutine. [*Hint:* Adapt Algorithm 2.3.1C to the case of a right-threaded binary tree, with suitable initial conditions.]

▸ **14.** [*M21*] How long does it take the program of exercise 13 to copy a tree with n nodes?

15. [*23*] Write an `MMIX` program for the `Div` routine, corresponding to `DIFF[7]` as specified in the text. (This program should be added to the program in the text after line 166.)

16. [*24*] Write an `MMIX` program for the `Pwr` routine, corresponding to `DIFF[8]` as specified in exercise 12. (This program should be added to the program in the text after the solution to exercise 15.)

2.3.3. Other Representations of Trees

[*357*]

Nodes have six fields, which in the case of `MMIX` might fit in three octabytes. A compact representation may use the fact that either the `VALUE` field is used to represent a constant or the `NAME` and `DOWN` fields are used to represent a polynomial g_j. So two kinds of nodes are possible:

RIGHT	LEFT
UP	EXP
VALUE	

or

RIGHT	LEFT
UP	EXP
NAME	DOWN

. (17)

Here `RIGHT`, `LEFT`, `UP`, and `DOWN` are relative links; `EXP` is an integer representing an exponent; `VALUE` contains a 64-bit floating point constant; and the `NAME` field contains the variable name. To distinguish between the two types of nodes, the low-order bit in a link field can be used. There are two essentially different choices: Either one of the link fields within the node is used or all the links that point to the node are marked. The first choice makes it easy to change a node from one type to the other (as is possible in step A9); the second choice makes searching for a constant (as in step A1) simpler.

2.3.5. Lists and Garbage Collection

[*411*]

1) … Therefore each node generally contains tag bits that tell what kind of information the node represents. The tag bits can occupy a separate `TYPE` field that can also be used to distinguish between various types of atoms (for example, between alphabetic, integer, or floating point quantities, for use when manipulating or displaying the data), or the tag bits can be placed in the low-order bits of the link fields, where they are ignored when using link fields as addresses of other `OCTA`-aligned nodes.

2) The format of nodes for general List manipulation with the `MMIX` computer might be designed in many different ways. For example, consider the following two ways.

a) Compact one-word format, assuming that all `INFO` appears in atoms:

`REF`	`RLINK`	`HMA`

(9)

This format uses 32-bit relative addresses to nodes from a common storage pool; the short addresses imply a limit of 4GByte on its maximum size. `RLINK` is such a pointer for straight or circular linkage as in (8). Limiting addresses to `OCTA`-aligned data, the three least significant bits `H`, `M`, and `A` are freely available as tag bits.

The `M` bit, normally zero, is used as a mark bit in garbage collection (see below).

The `A` bit indicates an atomic node. If `A` $= 1$, all the bits of the node, except `A` and `M`, can be used to represent the atom. If `A` $= 0$, the `H` bit can be used to distinguish between List heads and List elements. If `H` $= 1$, the node is a List head, and `REF` is a reference count (see below); otherwise, `REF` points to the List head of the sub-List in question.

b) Simple three-word format: A straightforward modification of (9) yields three-word nodes using absolute addresses. For example:

`RLINK`	`HMA`
`LLINK`	
`INFO`	

(10)

The `H`, `M`, and `A` bits are as in (9). `RLINK` and `LLINK` are the usual pointers for double linkage as in (8). `INFO` is a full word of information associated with this node; for a header node this may include a reference count, a running pointer to the interior of the List to facilitate linear traversal, an alphabetic name, and so on. If $\mathtt{H} = 0$, this field contains the `DLINK`.

[*420*]

Of all the marking algorithms we have discussed, only Algorithm D is directly applicable if atomic nodes must use all the node bits except a single bit, the mark bit. For example, Lists could be represented as in (9) using only the least significant bit for `M`. The other algorithms all test whether or not a given node `P` is an atom; they will need the `A` bit. However, each of the other algorithms can be modified so that they will work when atomic data is distinguished from pointer data in the word that links to it instead of by looking at the word itself. ... The adaptation of Algorithm E is almost as simple; both `ALINK` and `BLINK` can even accommodate two more tag bits in addition to the mark bit.

EXERCISES

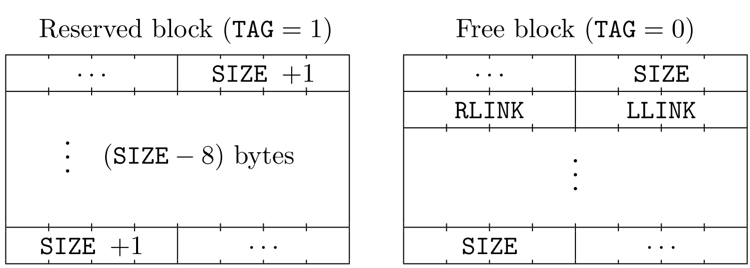
[*422*]

4. [*28*] Write an `MMIX` program for Algorithm E, assuming that the nodes are represented as two octabytes, with `ALINK` the first octabyte and `BLINK` the second octabyte. The least significant bits of `ALINK` and `BLINK` can be used for `MARK` and `ATOM`. Also determine the execution time of your program in terms of relevant parameters.

2.5. DYNAMIC STORAGE ALLOCATION

[*440*]

The method we will describe assumes that each block has the following form:



(7)

Note that the `SIZE` − 8 bytes reserved for use by an application are `OCTA`-aligned, while the node itself starts and ends with a `SIZE` field that is only `TETRA`-aligned.

The idea in the following algorithm is to maintain a doubly linked `AVAIL` list, so that entries may conveniently be deleted from random parts of the list. The `TAG` bit at either end of a block — the least significant bit in the `SIZE` field — can be used to control the collapsing process, since we can tell easily whether or not both adjacent blocks are available.

To save space, links are stored as relative addresses in a `TETRA`. As base address, we use `LOC(AVAIL)`, the address of the list head, which conveniently makes the relative address of the list head zero.

Unfortunately, a notation such as '`LINK(P + 1)`' does not work well in the world of `MMIX`, where addresses refer to bytes and links are stored as tetrabytes or octabytes. Therefore, we use the familiar `RLINK` and `LLINK` instead of '`LINK(P)`' and '`LINK(P + 1)`', but we do not rephrase Algorithm C. Double linking is achieved in a familiar way—by letting `RLINK` point to the next free block in the list, and letting `LLINK` point back to the previous block; thus, if `P` is the address of an available block, we always have

$$\mathtt{LLINK(RLINK(P))} = \mathtt{P} = \mathtt{RLINK(LLINK(P))}. \tag{8}$$

To ensure proper "boundary conditions," the list head is set up as follows:

`LOC(AVAIL)` − 4:	···	0
`LOC(AVAIL)` + 4:	`RLINK`	`LLINK`
`LOC(AVAIL)` + 12:	0	···

(9)

Here `RLINK` points to the first block and `LLINK` to the last block in the available space list. Further, a tagged tetrabyte should occur before and after the memory area used to limit the activities of Algorithm C.

[*449*]

Here are the approximate results:

	Time for reservation	Time for liberation
Boundary tag system:	$24 + 5A$	18, 22, 27, or 28
Buddy system:	$26 + 26R$	$36.5 + 24S$

...

This shows that both methods are quite fast, with the buddy system reservation faster and liberation slower by a factor of approximately 1.5 in `MMIX`'s case. Remember that the buddy system requires about 44 percent more space when block sizes are not constrained to be powers of 2.

A corresponding time estimate for the garbage collection and compacting algorithm of exercise 33 is about $98v$ to locate a free node, assuming that garbage collection occurs when the memory is approximately half full, and assuming that nodes have an average length of 5 octabytes with two links per node.

EXERCISES

[*453*]

4. [*22*] Write an `MMIX` program for Algorithm A, paying special attention to making the inner loop fast. Assume that the `SIZE` and the `LINK` fields are stored in the high and low `TETRA` of an octabyte. To make links fit in a tetrabyte, use addresses relative to the base address in the global register `base`. If successful, return an *absolute* address.

Use $\Lambda = -1$ if dealing with relative addresses, but for absolute addresses (the return value) use $\Lambda = 0$.

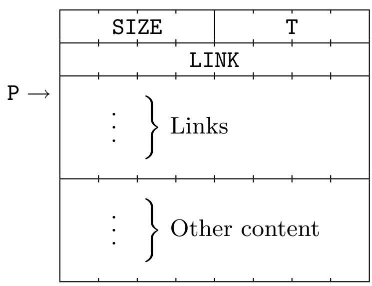
13. [*21*] Write an `MMIX` subroutine using the algorithm of exercise 12. Assume that the only parameter `N` is the size of the requested memory in bytes and that the return value is an `OCTA`-aligned absolute address where these `N` bytes are available. In case of overflow, the return value should be zero.

16. [*24*] Write an `MMIX` subroutine for Algorithm C that complements the program of exercise 13, incorporating the ideas of exercise 15.

27. [*24*] Write an `MMIX` program for Algorithm R, and determine its running time.

28. [*25*] Write an `MMIX` program for Algorithm S, and determine its running time.

▸ **33.** [*28*] (*Garbage collection and compacting.*) Assume a storage pool of nodes of varying sizes, each one having the following form:



`SIZE(P)` = number of bytes in `NODE(P)`;
`T(P)` = number of bytes used for links, `T(P)` < `SIZE(P)`;
`LINK(P)` = special link field for use only during garbage collection.

The node at address `P` starts with two octabytes *preceding* the address `P`; these contain special data for use during garbage collection only. The node immediately following `NODE(P)` in memory is the node at address `P` + `SIZE(P)`. The nodes populate a memory area starting at `BASE` − 16 up to `AVAIL` − 16. Assume that the only fields in `NODE(P)` that are used as links to other nodes are the octabytes `LINK(P)` + 8, `LINK(P)` + 16, …, `LINK(P)` + `T(P)`, and that each of these link fields is either Λ or the absolute address of another node. Finally, assume that there is one further link variable in the program, called `USE`, and it points to one of the nodes.

34. [*29*] Write an `MMIX` program for the algorithm of exercise 33, and determine its running time.