Thus even if h_i is not already reduced modulo n, a subtraction of n (rather than a more expensive division by n) will make it so. Since even this subtraction is needed only about Q/n of the time, a small Q (as in the last paragraph) saves subtract time.

6. Conclusion

We have presented a new algorithm for hash coding. It has been shown to possess certain attributes that are desired in such algorithms. Specifically it is simple, efficient, exhaustive, needs little time per probe, and uses few probes per lookup.

RECEIVED JUNE, 1970

REFERENCES

1. MORRIS, R. Scatter storage techniques. *Comm. ACM 11*, 1 (Jan. 1968), 35–38.
2. MAURER, W. D. An improved hash code for scatter storage. *Comm. ACM 11*, 1 (Jan. 1968), 38–44.
3. BELL, J. R. The quadratic quotient method: A hash code eliminating secondary clustering. *Comm. ACM 13*, 2 (Feb. 1970), 107–109.
4. RADKE, E. E. The use of quadratic residue research. *Comm. ACM 13*, 2 (Feb. 1970), 103–105.

A Nonrecursive List Compacting Algorithm

C. J. CHENEY
University Mathematical Laboratory, Cambridge, England

A simple nonrecursive list structure compacting scheme or garbage collector suitable for both compact and LISP-like list structures is presented. The algorithm avoids the need for recursion by using the partial structure as it is built up to keep track of those lists that have been copied.

KEY WORDS AND PHRASES: list compacting, garbage collection, compact list, LISP
CR CATEGORIES: 4.19, 4.49

Hansen [1] and Fenichel and Yochelson [2] have presented two algorithms for compacting list structures. One feature of both algorithms is that, on finding a list pointer, recursion is needed to collect the sublist. The authors of the second paper suggest that a nonrecursive but complex scheme should be possible using the algorithm of Deutsch, Schorr, and Waite [3]. However a much simpler algorithm is possible.

This algorithm uses a function COPYLIST to copy a list in the CDR direction from the original list area (the old area) to a new list area. List pointers are copied without transformation. A linear scan of the new list area applying COPYLIST to any list pointers found will copy any sublists into the new area.

To demonstrate the algorithm, consider the structure in Figure 1(a). The cells shown with a pointer from them, drawn with a broken line, serve only to connect the list in the CDR direction—these cells will be called "nonitems." The structure is accessed via the pointer HEAD. This structure has two lists, the first consisting of three atoms A, B, C and a list pointer to the second list which consists solely of a list pointer to the atom A of the first list. Lists

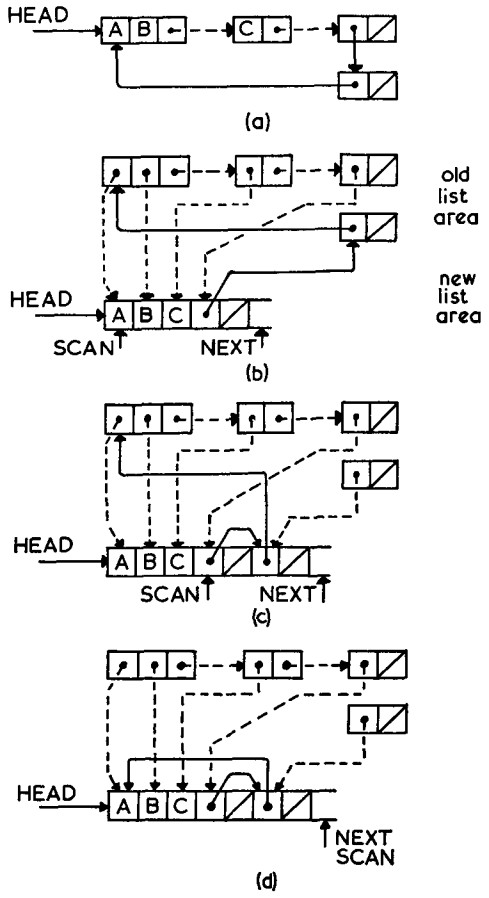

FIG. 1. Compacting a structure without looped lists: (a) initial structure; (b) and (c) during processing; (d) final structure.

are terminated by NIL cells. The algorithm with some modification can be applied to LISP-type structures.

To compact the structure, the list pointed to by HEAD is copied by COPYLIST into the new area—the contents of each item (not nonitems) are placed in consecutive cells in the new area, and the cells in the old area are changed to nonitems pointing to their equivalent cells in the new area. COPYLIST returns as a result the address of the first cell of the list in the new area. This value is used to update HEAD. A pointer NEXT is kept pointing to the next free cell in the new area. This intermediate structure is shown in Figure 1(b).

A further pointer SCAN now scans the new area from the beginning. If SCAN points to a NIL or an atom, it is moved on to the next cell. If SCAN points to a list pointer, COPYLIST is entered with this list pointer as its parameter. The sublist is therefore copied, updating NEXT, and the value returned is used to make the list pointer pointed at by SCAN point to the copied list in the new area. The structure resulting when SCAN has processed the first list pointer is shown in Figure 1(c). Note that, in copying, COPYLIST omits the nonitems of the original structure.

SCAN continues its traverse of the new area and will pass over the NIL to the second list to reach another list pointer. COPYLIST is again applied, but this time it finds the first item is already in the new area (e.g. by comparing core addresses), and the function returns with the address of this cell in the new area, which is used to update the list pointer, but without recopying the list.

The compact structure is complete when SCAN reaches the cell pointed to by NEXT. The procedure is capable of dealing with looped lists as COPYLIST will place a nonitem in the new area when necessary (see Figure 2).

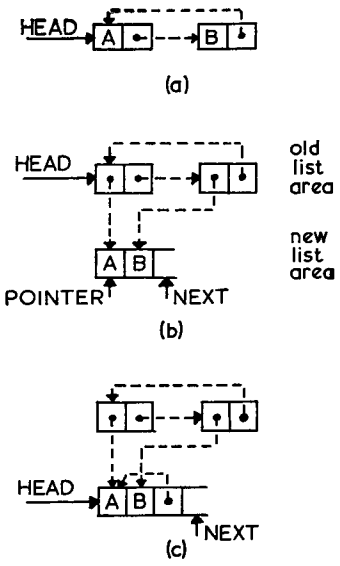

FIG. 2. Compacting a looped list

A version has been written in assembly language involving the execution of between 30 and 40 orders on an Atlas computer for each item of the structure transferred to the new area.

The algorithm is presented below in two parts—the "main program" first, and then the function COPYLIST.

Step 1. Initialize the pointers SCAN and NEXT to point to the beginning of the new list area.

Step 2. Apply the function COPYLIST—described below—to the pointer that points to the whole structure and assign the result of COPYLIST to that pointer.

Step 3. If SCAN points to a list-pointer, then apply COPYLIST to that list pointer, the result of COPYLIST replaces the contents of the cell pointed at by SCAN.

Step 4. Increment SCAN, unless SCAN now points to the same cell pointed at by NEXT, go to step 3 above. Otherwise the compacting is complete.

The function COPYLIST takes one parameter, POINTER, called by value; the function does the following:

Step 1. If POINTER points to a cell that is a nonitem, make POINTER point to the cell pointed at by the nonitem. Repeat this step while POINTER points to a nonitem.

Step 2. If POINTER is pointing to a cell in the new area, return with the value of POINTER as the result.

Step 3. Save the current value of NEXT in a variable V.

Step 4. If POINTER is pointing to a cell in the new area, make the cell pointed at by NEXT into a nonitem pointing to the cell that is pointed at by POINTER, and go to step 11 below.

Step 5. Copy the contents of the cell pointed at by POINTER to the cell pointed at by NEXT.

Step 6. If POINTER is pointing to a NIL, then go to step 11 below.

Step 7. Make the cell pointed at by POINTER into a nonitem that points to the cell pointed at by NEXT, i.e. the corresponding cell in the new area.

Step 8. Increment NEXT, increment POINTER.

Step 9. If POINTER points to a nonitem, make POINTER point to the cell pointed at by the nonitem, repeat this step while POINTER is pointing to a nonitem.

Step 10. Go to step 4 of the function COPYLIST.

Step 11. Increment NEXT and return with the value of V as the result.

Acknowledgments. The author is particularly grateful for many discussions with N. E. Wiseman and for the criticisms of the draft of this note from M. V. Wilkes.

RECEIVED MARCH, 1970; REVISED JUNE, 1970

REFERENCES

1. HANSEN, W. J. Compact list representation: definition, garbage collection and system implementation. *Comm. ACM 12*, 9 (Sept. 1969), 499–507.
2. FENICHEL, R. R., AND YOCHELSON, J. C. A LISP garbage-collector for virtual memory computer symstems. *Comm. ACM 12*, 11 (Nov. 1969), 611–612.
3. KNUTH, D. E. *The Art of Computer Programming Vol. 1 Fundamental Algorithms.* Addison-Wesley, Reading, Mass., 1968, pp. 417–419.