INFORMATION TO USERS

This dissertation was produced from a microfilm copy of the original document. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the original submitted.

The following explanation of techniques is provided to help you understand markings or patterns which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting thru an image and duplicating adjacent pages to insure you complete continuity.

2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.

3. When a map, drawing or chart, etc., was part of the material being photographed the photographer followed a definite method in "sectioning" the material. It is customary to begin photoing at the upper left hand corner of a large sheet and to continue photoing from left to right in equal sections with a small overlap. If necessary, sectioning is continued again — beginning below the first row and continuing on until complete.

4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from "photographs" if essential to the understanding of the dissertation. Silver prints of "photographs" may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.

University Microfilms

300 North Zeeb Road
Ann Arbor, Michigan 48106

A Xerox Education Company

LINEAR LISTS AND PRIORITY QUEUES AS BALANCED BINARY TREES

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Clark Allan Crane

March 1972

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

(Principal Adviser)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

PREFACE

In the Spring of 1970 I approached Professor Knuth about becoming his advisee. Adopting a pragmatic attitude, I enumerated my criteria for a thesis topic: its results should be potentially useful to a reasonably broad body of computer users; the work should be completable in a year or so; and the subject should be currently under investigation elsewhere by preferably no one. Would he please suggest a few? What I got was one, along with an admonition to the effect that good thesis topics are on the same endangered species list as the golden-egg-laying goose.

Chapters One and Three present two related but distinct data structures, one for lists and the other for priority queues. They both are based on triply-linked binary trees; but while a perfectly balanced tree presents the best case for lists, it is the absolute worst for priority queues. Chapter Four contains ALGOL W implementations of the algorithms for both data structures, plus a routine for displaying trees on the line printer.

Chapter Two is partly a tutorial on measures of the amount of effort required to search or enlarge trees, and how many trees of various kinds there are. Then, using some new terminology, Section 2.5 discusses local search-order-preserving transformations and their relative ability to change the shape of binary trees.

All definitions have been collected by chapter, arranged by their semantic relations, and offered as Chapter Zero. This approach presumes that a thorough study of the paper consists of several passes through it, the first a fast scan and each succeeding one a more detailed and critical investigation. The material hopefully flows more smoothly without the interruptions of parenthetical definitions.

No solutions for the problems of Chapters One, Two, and Three have been provided, not even for the odd-numbered ones. The difficulty rating, a mime of Knuth's successful scheme, loosely classifies the problems: 0-25, the student can verify his own solution; 26-35, I haven't worked the problem myself; 36-50, the problem is a suitable term project or is included in lieu of a statement to the effect, "The proof is beyond the scope of this author."

My sincere gratitude goes to Professor Donald E. Knuth, who saved me a week's to a year's work with each consultation; to Professor Edward J. McCluskey, who served as my course adviser for two years; to Associate Professor Harold S. Stone and Professor Robert W. Floyd, who completed my reading committee; to Provost William F. Miller, who gave me my first programming job, at SLAC; to Miss Eileen Kennedy at Hughes and Mrs. Phyllis Winkler at Stanford, able secretaries who prepared the first and final drafts, respectively; to Hughes Aircraft Company, for support and gainful summer employment; to the people of the SPIRES/BALLOTS project, for experience, stimulating contacts, and support; to the Fannie and John Hertz Foundation and the Veterans Administration for support; and to my wife, Valerie, who had her own homework to do in addition to caring for me and keeping up the house.

This exercise in arboriculture is dedicated:

To Don Knuth and all my other teachers,
for the seeds and tools;

To HAC, the Hertz Foundation, SPIRES/BALLOTS, and
the VA, for the fertilizer;

And to Valerie, for the sunshine.

(I did the tilling.)

Table of Contents

List of Tables

List of Illustrations

Abstract

Any representation of a list in high speed computer memory is a compromise among competing measures of efficiency: compactness; speed of the desired list operations; and simplicity of algorithm. Most representations optimize the speed of a few operations at the expense of others. A linked data structure given here, an outgrowth of the balanced trees of Adel'son-Velskiĭ and Landis, allows the following common linear list operations to be performed in worst-case times which grow only as the logarithm of list size: insertion, retrieval, and deletion of an item by position in the list or by key value; concatenation of two lists; splitting a list in two at a certain position or after a certain key value; and finding the predecessor or successor of a given item. The construction, traversal, copying, and merger operations require times which grow linearly with list size. The logarithmic bounds result from making local changes, when necessary, to assure that no two sibling subtrees differ in height by more than one level. Items may be either atoms or lists.

Local transformations which involve only two pivotal nodes suffice to change any binary tree of n nodes into any other in no more than $2n-2$ steps. Transformations of no more than five pivotal nodes suffice to change the balance of a node in a balanced tree from positive to non-positive while preserving tree balance.

A (non-preemptive) priority queue obeys a best-in-first-out discipline. Stacks and simple queues are special cases of a priority queue. By representing priority queues as linked binary trees which impose only a partial ordering on the items, the item with the earliest priority in each subtree appearing at the root of the subtree, it is possible to exploit the restriction that only the best item need be accessible. Maintaining in each node a field which indicates the distance to the nearest empty subtree provides the basis for algorithms which require worst-case times which grow as the logarithm of priority queue size for the following operations: enqueuing an item by its priority; the merger of two queues; removing the next item from a queue; and purging a given arbitrary item from a queue.

The paper presents detailed non-recursive algorithms for the linear list and priority queue operations, both informally and as implemented in ALGOL W. Historical background of the theory and use of binary trees for lists and priority queues supplements the present contribution. Numerous exercises, rated by difficulty, confirm the reader's grasp of the material and suggest areas for further research.

Key words and phrases:

balanced tree
binary tree
content-addressable memory
data structure
heap
linear list
list processing
priority queue
queue
searching
sorting
stack

LINEAR LISTS AND PRIORITY QUEUES AS BALANCED BINARY TREES

by

Clark A. Crane

Corrigenda

Negative line numbers are counted from the bottom of the page.

Page

ii Line -3, The support citation refers to the costs of final typing and reproduction of the report. The research itself was supported by Hughes Aircraft Company, the SPIRES/BALLOTS Project, the Fannie and John Hertz Foundation, and the Veterans Administration.

10 Lines -7,-6. Replace with the reverse order:

choice: (0.19%-Riemer, 1.39%-Chaffin, 5.99%-Debs, 23.15%-Taft, 27.42%-Roosevelt, 41.85%-Wilson)

36 Line -2. A comma should follow "ALGOL W".

54 Line -6. "(c1*i1+i2)" should read "(c1*i1+c2)".

56 Line -6. The equality should read "O(2*m-1) = O(m)".

63 Line 6. The "k^{i}" should appear centered to the right of the summation sign, not as a superscript.

68 Line 8. The "t^{-1}" should appear centered to the right of the summation sigh, not as a superscript.

97 Figure 3.6.1. There should be an additional arrow from "purge" to "submerge".

104 Line -16. "Wourld" should read "would".

126 Some copies have ink blots obscuring parts of three lines.
Line 4 starts out "STR:=INTBASE10(SIDE(T));"
Line 5 starts out "L3(J+9|1):=STR(1|1);"
Line 8 is "L4(I|1):="|" " .

129 Line -11. Underline "SIAM J. Appl. Math.": Line -7. Underline "Proc. IRE".

130 Line 12. "Jaunuary" should read "January".

Chapter Zero

CONTEXTUAL GLOSSARY

It is convenient for reference and review purposes to assemble all the special vocabulary used in this paper. Although the reader need not dwell on this chapter during the first reading, he should peruse Section 0.0 and sections pertaining to chapters of interest to him. No illustrations are found here; they appear in the appropriate positive-numbered chapters with their descriptions.

This terminology for trees has been selected without regard to race, creed, sex, age, or national origin.

0.0 General Terms

The following GENERAL TERMS apply throughout the paper.

A node is an entity comprised of a finite number $k \geq 1$ of named fields (f1,...,fk) , each of which can hold a fixed amount of information. A class is a set of nodes which have the same fields arranged in the same spatial relationship, i.e., the same format. A class may be implemented in such a way that each node occupies a block of contiguous memory locations; in this case a template, e.g., a dummy control section, specifies the locations of the various fields in relation to an addressable reference point in the node. Alternatively, each field name of a class may define a block of contiguous cells, with an integral number of cells forming the field for each node; then the k-th field of node i is simply the i-th element of array fk . The choice of representation depends on: the addressability and other access characteristics of the storage medium; storage and execution time constraints; the method of storage allocation and deallocation; the instruction set or programming language used; and programmer preference.

A node itself has no name, and its physical location is irrelevant. It is known by its contents or by the relationships it has with other nodes in the data structure in which it appears.

A pointer, or link, is a value which designates (points to) a particular node. Together with knowledge of the node's class, a pointer

to a node suffices to inspect or modify the contents of any field of the node. A pointer designating no node, or an empty structure, equals the distinguished value null. Pointers may be consistently encoded in any desired way: as absolute addresses, array subscript values, base-displacement pairs, etc. If a pointer variable p points to a node which has a field info then that field is referenced by the expression " info(p) ". In this paper the name of a pointer variable may variously refer to its value (as in comparisons for equality), a node to which it points, or to the entire data structure emanating from the node to which it points. The meaning in each instance should be clear from the context. A linked data structure is empty, or is a non-empty finite collection of nodes connected by appropriate link fields in the nodes, such that from some node in the collection there are link-node sequences leading to all other nodes in the collection. In other words, if a node cannot be accessed it does not belong to the data structure.

A binary subtree, or subtree, is a linked data structure which is empty, or consists of a subtree node p called the root and two disjoint subtrees linked to p by link fields left(p) and right(p) . If p has a non-empty subtree then p is non-terminal; otherwise p is terminal. Let q = left(p) and r = right(p) be non-empty. The integer field side is defined for subtree nodes by side(q) = -1 and side(r) = +1 . If p is part of no other subtree then side(p) = 0 , p is called the principal root, and is the root of the principal subtree. Nodes q and r are siblings, the children of p ; p is their parent. The root of a subtree is an ancestor of all other nodes in the subtree; a node is a descendant of its ancestors. Link field up is defined for subtree nodes by up(q) = up(r) = p . Note that up and side carry redundant information about a subtree.

A binary tree, or tree, consists of a distinguished node t called the header, and a principal subtree p linked to t by link field up(t) . Fields right(t) and left(t) are not defined. If p is non-empty then up(p) = t . Hereafter, subtree nodes will be simply called nodes, and header nodes will be called headers; phrases such as "number of nodes" will implicitly exclude headers. The header is the ancestor of all nodes in the tree and the descendant of none.

A binary tree is depicted with the header at the top, the principal root immediately below it, etc. Hence, higher means nearer to the header; down and lower mean toward a terminal node. The height h of a subtree p is 0 if p is empty; otherwise, h(p) = 1+max(h(left(p)),h(right(p))) . The height of a tree is the height of its principal subtree. The size of a tree is the number of nodes in the tree.

To allow shortening the statement of algorithms by taking advantage of symmetry, some equivalences are defined for node p in terms of link: link(p,-1) = left(p) , link(p,0) = up(p) , and link(p,+1) = right(p) . If t is a header then only link(t,0) = up(t) is defined. Note that for node p , if the absolute value abs(c) = 1 and link(p,c) ≠ null then side(link(p,c)) = c . Further, link(up(p),side(p)) = p , illustrating the redundancy of the fields up and side. The side of a header is undefined.

The (internal unweighted) path length is the number of ancestors of each node, summed over all nodes in the tree.

If t is a (sub)tree each of whose nodes has two subtrees of the same size, then t is a perfect (sub)tree, and size(t)+1 is a power of 2 , i.e., size(t) = 2↑h(t)-1 .

The distance to null is a recursively defined attribute dist of a tree or subtree t . If t is an empty subtree then dist(t) = -1 . If t is a node (non-empty subtree) then dist(t) = 1+min(dist(left(t)),dist(right(t))) . If t is a header (tree) then dist(t) = 1+dist(up(t)) . For example, dist of an empty tree is 0 , dist of a subtree of size 1 or 2 is 0 , and dist of a subtree of size 3 is either 0 or 1 .

If t is a subtree which is empty or which satisfies h(t) - dist(t) ≤ 1 then t is a complete subtree, or minimum path length subtree. If t is a tree whose principal subtree is complete then t is a complete tree, or minimum path length tree.

If two (sub)trees t1 and t2 can be exactly superimposed on one another (disregarding node contents), after suitable nodes of t1 have had their subtrees interchanged left for right, then t1 is homeomorphic to t2 . Homeomorphism is an equivalence relation. A perfect (sub)tree is homeomorphic only to itself.

A (sub)tree which has no non-empty right (left) subtree is right-(left-)degenerate. If every node of a (sub)tree has an empty subtree then the (sub)tree is degenerate. .

To say $f(n) = O(g(n))$, read " f of n is order of g of n ," means that there are positive constants c1 and c2 such that the relation $|f(n)| \leq c1 * |g(n)|$ holds for all $n \geq c2$. To say that an algorithm is $O(f(n))$ means that the number of steps required to execute it is $O(f(n))$, and each of these steps takes $O(1)$ units of time to complete on a conventional computer; thus, the total time required is $O(f(n))$ units.

0.1 Linear Lists and Balanced Trees

The following are TERMS APPLYING TO BALANCED TREE REPRESENTATION OF LINEAR LISTS.

A linear list is a finite set of $k \geq 0$ items arranged in such a way that it is possible to identify and access uniquely the i-th item for $1 \leq i \leq k$. Examples are: a vector, a one-dimensional array, a string of characters, a LISP list (ignoring the presence of structure in any non-atomic list elements). An ordered linear list is a linear list in which each item is associated with a value called its key, and the items are arranged in strictly increasing order of their keys. (We require all keys to be distinct, in order to simplify discussion of the algorithms.) In the sequel, "linear list" and "ordered linear list" will be abbreviated "list" and "ordered list", respectively.

The symmetric order of nodes in a subtree s is the order in which the nodes are visited in postorder traversal, described recursively: if s = null then do nothing; otherwise traverse left(s) , visit s , and traverse right(s) . The first node thus visited is called the leftmost and the last node visited is called the rightmost. The symmetric order of a tree t of size k is circular: 1st,2nd,...,kth,1st,... , etc. The successor of node p in tree t , denoted p$, is the next node or header in symmetric order; the predecessor, $p , is the previous node or header. If tree t is empty then successor(t) = predecessor(t) = t . The cessor relationship

is defined by the equivalences cessor(p,-1) = $p and cessor(p,+1) = p$. The extended cessor relationship allows equivalences such as cessor(p,0) = p , cessor(p,-2) = $$p , etc.

To insert an item p in a linear list after positio: i , $0 \leq i \leq k$, is to increase by 1 the ordinal numbers of the (i+1)-st through k-th items and to make p the (i+1)-st item. To delete the i-th item is to remove it from the list and decrease by 1 the ordinal numbers of items i+1 through k ; the list is shortened, and no vacancy is introduced. Searching a list t1 of size k1 on position i means setting a pointer to the i-th item unless $i < 1$ or $i > k1$, in which case the pointer is set to null. Searching an ordered list by value on key k means setting a pointer to the item whose key is k and setting a real variable to the item's position in the list. If no item's key is k then the pointer is set to null and the real variable is set to a fractional value indicating where the item would lie, e.g., 0.5 for "before the first item", or 2.5 for "after the second item". Concatenating list t1 to the left of list t2 to produce list t3 is the same as appending t2 to t1 , calling the result t3 , and setting t1 and t2 to empty. If t1 and t2 had k1 and k2 items, respectively, then the first (leftmost) k1 items of t3 are from t1 and the last k2 items are from t2 . Splitting a list t1 of k1 nodes into lists t2 and t3 on position i is making t2 the list composed of the first i items of t1 , making t3 the list composed of the last k1-i items of t1 , and setting t1 empty. Merging ordered lists t1 and t2 to produce ordered list t3 means interleaving the items of t1 and t2 in such a way as to result in a single ordered list t3 , and setting t1 and t2 to empty.

A linear list of $k \geq 0$ items can be represented in a tree t of size k by putting the first item in the leftmost node t$, the second item in t$$, etc. They key becomes an integer or real field containing the key of the item in the node; key(t) is not defined.

The key space is totally ordered. Examples of key spaces are: the non-negative integers less than 2↑18 ; all ANSI character strings of length 1 through 10 ; all short-precision floating-point numbers; and payroll numbers of all current employees.

The balance factor of a node p is an integer field bal whose value is h(left(p))-h(right(p)) . If t is a header then bal(t) is defined to be 0 .

The left-subtree size of node p is an integer field less, containing the number of nodes in left(p) ; less(t) = "minus infinity" for header t .

A (sub)tree t is balanced if, for every node p in t , |bal(p)| < 2 . A (sub)tree t is well-balanced if it is balanced and, for every node p in t , bal(p) = 0 or p has an empty subtree. A (sub)tree t is perfectly-balanced if bal(p) = 0 for every node p in t . It follows that a well-balanced tree is a complete tree, and a perfectly-balanced tree is a perfect tree. If bal(p) = -1 for every non-terminal node p in tree t then t is a Fibonacci tree. Trees homeomorphic to a Fibonacci tree achieve the greatest height attainable by balanced trees whose size is less than that of the next larger Fibonacci tree. A perfectly-balanced tree has a size greater than any other tree of the same height. A degenerate tree has a height equal to its size, the height greater than all non-degenerate trees of the same size. A threaded tree is a tree whose nodes are augmented with two logical fields ltag and rtag , used as follows. For each node p , if the left subtree is empty then ltag = true and left(p) = $p is a thread. Otherwise, ltag = false and left(p) points to the left subtree of p as in an unthreaded tree. If the right subtree is empty then rtag = true and right(p) = p$ is a thread. Otherwise, rtag = false and right(p) points to the right subtree of p . If t is a header of a threaded tree then left(t) is defined to be t$, and right(t) is $t , making it very easy to locate the 1-st and k-th nodes. Threads improve tree traversal speed by utilizing otherwise empty links, at the expense of two bits per node storage and a bit comparison per level on search.

0.2 Mathematical Aspects of Trees

The following are TERMS DEALING WITH TRANSFORMATIONS ON BALANCED TREES.

A set of m nodes in a tree t are adjacent if the cutting of all links (arcs) not connecting two of the m nodes results in a (connected) subtree containing exactly the m nodes.

A rotation of degree m is a transformation of a tree T_1 into a tree T_2 satisfying the following conditions.

1. No nodes are added to or deleted from T_1.
2. There is a node, called the founder, whose parent, if the founder is not the header, does not change in the transformation. The founder has a subtree S which is the smallest subtree containing the set M of m adjacent nodes called the participants. Subtree S becomes subtree S' as a result of the transformation.
3. Neither subtree of any node of S may change in the transformation unless the node is a participant; the change in any subtree of a participant is limited to the upward link of the root of the subtree, unless the subtree itself contains nodes of M. The sibling subtree of S, R, does not change.
4. The symmetric traversal order (postorder) of S' is the same as that of S.

A rotation of degree m is of strict degree m if each of the m adjacent nodes has its parent and one or both of its subtrees changed.

The only rotation of degree 1 is the identity rotation. Since only the subtrees of one node may change, the only possibility is to interchange them; this interchange preserves symmetric order only if both down-links are null. There is no rotation of strict degree 1. The identity rotation does not involve the change of any parents, violating the definition of strict order.

The root r of S in a rotation belongs to M. Suppose r does not belong to M. Then either M is divided between the left and right subtrees of r or M is entirely contained in one of the subtrees of r. The former contradicts adjacency of M and the latter contradicts the requirement that S is the smallest subtree containing M.

Two trees T and S of size n are neighbors (1-neighbors) if a single rotation of degree 2 suffices to transform one into the other. Trees T and S are k-neighbors if the minimum number of degree 2 rotations required to transform one into the other is exactly k .

ARE YOU STILL THERE?

0.3 Priority Queues and Trees

The following are TERMS APPLYING TO TREE REPRESENTATION OF PRIORITY QUEUES.

A priority queue is a finite set of $k \geq 0$ items awaiting service, each of which is associated with a distinct value called its priority. Items are served (removed from the priority queue) in order of precedence; low values of priority are served early and high values late. The order in which the items are enqueued is irrelevant, unless the item of earliest priority need be accessible. Thus, a priority queue obeys a "best-in-first-out" discipline. If items are enqueued in order of strictly increasing priority value then the priority queue behaves exactly as a queue (first-in-first-out); the reverse order results in stack behavior (last-in-first-out).

The distance of a header (node) is an integer field dist whose value is the distance to null of the header (node).

The priority queue balance factor is an integer field balp which indicates the path to the nearest empty subtree. If t is a header then balp(t) = 0 , since either up(t) is null or up(t) is part of the path to the nearest null link. If p is a node with an empty subtree (dist(p) = 0) , then balp(p) is -1 or +1 , meaning that left(p) = null or right(p) = null , respectively. If p has no empty subtree then balp(p) is -1 or +1 , depending on whether left(p) or right(p) is nearer to an empty subtree; specifically, dist(link(p,balp(p))) = dist(p)-1 . If both p's subtrees are empty or dist(left(p)) = dist(right(p)) then balp(p) may be either +1 or -1 , arbitrarily.

The priority of a node is integer or real field prio , which contains the priority value of the item in the node. If t is a header then prio(t) is "minus infinity".

Let t be a tree with a distinct value of prio in each node. Further, for every pair of nodes p and q in t assume that if p is an ancestor of q then prio(p) $\leq$ prio(q) . Then t is partially-ordered on prio, and t satisfies the tree partial-order criterion. (Equality is not achieved since we are assuming distinct priorities.)

A priority queue of $k \geq 0$ items can be represented in a tree t of size k by putting the items into the nodes of t , one item per node, in such a way that t satisfies the tree partial-order criterion. Thus, if $k > 0$, the principal root of t contains the item with the earliest priority (smallest prio value).

To remove any item from a priority queue is to purge that item from the queue. Removing the item of earliest priority is serving that item, and is servicing the queue. Enqueuing an item is placing it in a priority queue to await service according to its priority. Merging priority queues q1 and q2 to form q3 is combining the items into a single valid priority queue q3 .

Thank you for your attention.

Chapter One

LINEAR LISTS

1.1 Introduction

(At this point, please read Sections 0.0 and 0.1 of the Contextual Glossary if you have not already done so.) A linear list is an abstract entity which is considered to contain items of information arranged in separate sequential positions so that one may speak of the first, i-th, or last item without ambiguity. The term "linear" means that the items are presumed to be atomic, or indivisible. Thus, any phenomena introduced by nested lists (items which are themselves lists) or recursive lists (lists which are nested within themselves) are at the same time permitted and without consequence to linear list operations, which ignore list substructure. An example of a list might be one called "journal", giving a housewife's activities for the day.

journal: (washing, cookdinner, grocery, callmother)

The list has four items, but any of those items in fact may designate other lists of activities.

An ordered linear list is a linear list each of whose items has an attribute called the key; the items are positioned in the list in order of ascending keys, and no two items have the same key. For example, the following list of 6 items, ordered on percentage, gives the result of the 1912 U.S. presidential election:

choice: (41.85% - Wilson, 27.42% - Roosevelt, 23.15% - Taft,
5.99% - Debs, 1.39% - Chaffin, 0.19% - Riemer)

The concept of linear list is such a simple and widely applied scheme of arranging information that it is difficult to say anything very scholarly about it. As anyone who has been asked about his driving record by a traffic judge can verify, even an empty list can have meaning. It is nonetheless useful to list the more common linear list operations.