ming topics. This is directly opposed to the often made assumption that a student's academic discipline is a good predictor of potential competency in programming. This result implies that if a course is suitably constructed and presented, there is no need to segregate students from different academic disciplines due to concerns based on learning ability or interdisciplinary competitiveness.

5.2 Gender Differences

No significant difference was found in academic performance between the genders. The only observable difference is that women appear to be more uniform in their academic performance than men.

5.3 Semester in School

The correlations found between semester in school and academic performance were very low. This clearly indicates that in a suitably constructed course, there is no reason to worry more about those beginning their university experience as opposed to those well along in their academic careers. They all seem to succeed or fail in much the same manner.

5.4 PAT Use

The results indicate that future programming skill is not predictable by the most commonly used written test (IBM's Programming Aptitude Test). The suggestion is that this type of test should not be administered to college level people as the results are unreliable.

6. Conclusions

It is not necessary to construct separate competing courses for those from differing disciplines and levels of academic experience as there is no apparent need to be concerned with unequal capability. It is also not possible to reliably predict success in learning programming on the basis of normally observable external personal attributes or standardized written tests when the people involved are of college level.

Received May 1978; revised September 1978; accepted September 1979

References

1. Bateman, C.R. Predicting performance in a basic computer course. Proc. of the Fifth Annual Meeting of the Amer. Inst. for Decision Sciences, Boston, Mass., 1973.
2. Newstead, P.R. Grade and ability predictions in an introductory programming course. SIGCSE Bull. 7, 2 (June 1975), pp. 87–91.

Programming and Data Structures
M.D. McIlroy Editor

Minimal Perfect Hash Functions Made Simple

Richard J. Cichelli
Software Consulting Services,
Allentown, Pennsylvania

A method is presented for computing machine independent, minimal perfect hash functions of the form: hash value ← key length + the associated value of the key's first character + the associated value of the key's last character. Such functions allow single probe retrieval from minimally sized tables of identifier lists. Application areas include table lookup for reserved words in compilers and filtering high frequency words in natural language processing. Functions for Pascal's reserved words, Pascal's predefined identifiers, frequently occurring English words, and month abbreviations are presented as examples.

Key Words and Phrases: hashing, hashing methods, hash coding, direct addressing, dictionary lookup, information retrieval, lexical analysis, identifier-to-address transformations, perfect hashing functions, perfect hash coding, scatter storage, searching, Pascal, Pascal reserved words, backtracking

CR Categories: 3.7, 3.74, 4.34, 5.25, 5.39

Introduction

In several recent articles [2, 3] it has been asserted that in general computing minimal perfect hash functions for static identifier lists (keys) is difficult. In [1], Knuth also notes the difficulty in computing perfect hash functions. He estimates that for the set of words used here in Example No. 3 a search for such a function might include the examination of 10^{43} possibilities. A program imple-

Author's present address: R.J. Cichelli, Software Consulting Services, 901 Whittier Drive, Allentown, PA 18103.

menting the procedure outlined in this paper finds a machine independent minimal perfect hash function for these words in less than one second on a DEC PDP-11/45 minicomputer. Other examples of such functions are shown and an effective method for computing them is described.

Although my algorithm requires exponential search of potentially large spaces, it is based on intelligent search of these spaces. The procedure has been shown to find solutions for many sets in a few seconds. For thirty-nine of Pascal's forty predefined identifiers (Example No. 2), the potential search space was 20^{39} nodes. Computation of the function required seven minutes on the PDP-11/45. For lookup tables that will be used many times, the method has proved quite practical. So far I have encountered no instance of an interminably long search.

The form of my hash function is:

Hash value ← key length +
associated value of the key's first character +
associated value of the key's last character.

Example No. 1: Pascal's Reserved Words

For Pascal's 36 reserved words, the following list defines the associated value for each letter:

A = 11, B = 15, C = 1, D = 0, E = 0, F = 15, G = 3, H = 15, I = 13, J = 0, K = 0, L = 15, M = 15, N = 13, O = 0, P = 15, Q = 0, R = 14, S = 6, T = 6, U = 14, V = 10, W = 6, X = 0, Y = 13, Z = 0.

(For lookup routines these values are stored in an integer array indexed by the letters. Note: Associated values need not be unique.)

The corresponding hash table with hash values running from 2 through 37 is as follows:

DO, END, ELSE, CASE, DOWNTO, GOTO, TO, OTHERWISE, TYPE, WHILE, CONST, DIV, AND, SET, OR, OF, MOD, FILE, RECORD, PACKED, NOT, THEN, PROCEDURE, WITH, REPEAT, VAR, IN, ARRAY, IF, NIL, FOR, BEGIN, UNTIL, LABEL, FUNCTION, PROGRAM.

As an example, consider the computation for "CASE":

(1 ← "C") + (0 ← "E") + (4 ← length("CASE")) = 5.

The advantage of hash functions of the above form is that they are simple, efficient, and machine (i.e., character representation) independent. It is also likely that any lexical scanning process will have, as a by-product of its identifier scanning logic, the identifier length and the values of the first and last characters. Two disadvantages of functions of this form are: (1) it requires that no two keys share length and first and last characters and (2) for lists with more than about 45 items, segmentation into sublists may be necessary. (This is a result of the limited range of hash values which the functions produce.)

The associated values for each of the letters are computed by the following procedure: (1) order the identifier list, and (2) search, by backtracking, for a solution.

The ordering process is twofold. First, order the keys by the sum of the frequencies of the occurrences of each key's first and last letter in the list. For example: "E" occurs 9 times as a first or last letter in the Pascal reserved word list. It is the most frequent letter and thus, "ELSE" is the first word in the search list. "D" is the next most frequent letter, and thus "END" is second. After the words have been put in order by character occurrence frequencies, modify the order of the list such that any word whose hash value is determined by assigning the associated character values already determined by previous words is placed next. Thus, after "OTHERWISE"[1] has been placed as the third element of the frequency ordered list, the hash value of the word "DO" is determined and so it is placed fourth. (For example, during search, after the placement of the word "END," a value will be associated with "D," and after the placement of the word "OTHERWISE," a value will be associated with "O.") The ordering process causes inevitable hash value conflicts to occur during search as early as possible, thus pruning the search tree and speeding the computation.

The completely ordered search list for Pascal's reserved words is:

ELSE, END, OTHERWISE, DO, DOWNTO, TYPE, TO, FILE, OF, THEN, NOT, FUNCTION, RECORD, REPEAT, OR, FOR, PROCEDURE, PACKED, WHILE, CASE, CONST, DIV, VAR, AND, MOD, PROGRAM, NIL, LABEL, SET, IN, IF, GOTO, BEGIN, UNTIL, ARRAY, WITH.

The backtracking search procedure then attempts to find a set of associated values which will permit the unique referencing of all members of the key word list. It does this by trying the words one at a time in order. The backtracking procedure is as follows: If both the first and last letter of the identifier already have associated values, try the word. If either the first or last letter has an associated value, vary the associated value of the unassigned character from zero to the maximum allowed associated value, trying each occurrence. If both letters are as yet unassociated, vary the first and then the second, trying each possible combination. (An exception test is required to catch situations in which the first and last letters are the same.) Each "try" tests whether the given hash value is already assigned and, if not, reserves the value and assigns the letters. If all identifiers have been selected, print the solution and halt. Otherwise, invoke the search procedure recursively to place the next word. If the "try" fails, remove the word by backtracking.

The search time for computing such functions is related to the number of identifiers to be placed, the maximum value which is allowed to be associated with a character, and the density of the resultant hash table.

[1] Inclusion of the word "OTHERWISE" in Pascal's reserved word list anticipates the acceptance by the Pascal Users Group of the recommendation for a revised CASE construct submitted by its International Working Group for Extensions.

If the table density is one (i.e., a minimal perfect hash) and the maximum associated value is allowed to be the count of distinct first and last letter occurrences (21 for Pascal's reserved words), then the above procedure finds a solution for Pascal's reserved words in about seven seconds on a DEC PDP-11/45 using a straightforward implementation of the algorithm in Pascal. (Without the second ordering, the search required 5.5 hours.) If the maximum associated value is limited to 15, as in the above list, the search requires about 40 minutes. (There is no solution with 14 as a maximum value.)

Incorporation of the above hash function into a Pascal cross-reference program yielded a 10 percent reduction in total run time for processing large programs. The method replaced a well-coded binary search which was used to exclude reserved words from cross-referencing.

Example No. 2

The second example is for the list of Pascal's predefined identifiers.

A = 15, B = 9, C = 11, D = 19, E = 5, F = 3, G = 0, H = 0, I = 3, J = 0, K = 16, L = 13, M = 1, N = 19, O = 0, P = 18, Q = 0, R = 0, S = 15, T = 0, U = 17, V = 0, W = 10, X = 0, Y = 0, Z = 0.

GET, TEXT, RESET, OUTPUT, MAXINT, INPUT, TRUE, INTEGER, EOF, REWRITE, FALSE, CHR, CHAR, TRUNC, REAL, SQR, SQRT, WRITE, PUT, ORD, READ, ROUND, READLN, EXP, PAGE, EOLN, COS, SUCC, DISPOSE, NEW, ABS, LN, BOOLEAN, WRITELN, SIN, PACK, UNPACK, ARCTAN, PRED.

Computation of this function required about seven minutes. Note: Since the predefined identifier "ODD" conflicts with "ORD," it was not included in the list.

Example No. 3: Frequently Occurring English Words

This example uses the word list of [1, 2]. Search time was less than one second.

A = 3, B = 15, C = 0, D = 7, E = 0, F = 15, G = 0, H = 10, I = 0, J = 0, K = 0, L = 0, M = 12, N = 13, O = 7, P = 0, Q = 0, R = 12, S = 6, T = 0, U = 15, V = 0, W = 14, X = 0, Y = 9, Z = 0.

I, it, the, that, at, are, a, is, to, this, as, he, and, have, in, not, be, but, his, had, or, on, was, of, her, by, you, with, which, for, from.

Example No. 4: Month Abbreviations

This example is from [3]. The function's form was modified slightly to:

Hash value ← associated value of the key's second character + associated value of the key's third character.

A = 4, B = 5, C = 2, D = 0, E = 0, F = 0, G = 3, H = 0, I = 0, J = 0, K = 0, L = 6, M = 0, N = 0, O = 5, P = 1, Q = 0, R = 6, S = 0, T = 6, U = 0, V = 6, W = 0, X = 0, Y = 5, Z = 0.

JUN, SEP, DEC, AUG, JAN, FEB, JUL, APR, OCT, MAY, MAR, NOV.

This form avoids the conflict between "JAN" and "JUN" and takes into account the constant key length. Search time was again well less than one second. Note: The method presented here is applicable to sets up to four times as large as those said to be feasible by the methods described in [3].

Moral

This article does not have a conclusion, but it does have a moral. In the words of the renowned chess programmer, J. Gillogly, author of the Technology Chess Program which was the prototype of the current generation of highly successful chess programs, "When all else fails, try brute force."

Received May 1979; revised and accepted October 1979

References

1. Knuth, D.E. *The Art of Computing Programming. Volume 3: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973, pp. 506–507.
2. Sheil, B.A. Median split trees: A fast lookup technique for frequently occurring keys. *Comm. ACM 21,* 11 (Nov. 1978), 947–958.
3. Sprugnoli, R. Perfect hashing functions: A single probe retrieving method for static sets. *Comm. ACM 20,* 11 (Nov. 1977), 841–850.