Optimal Table Lookup for Reserved Words in Ada

John A. Trono
Saint Michael's College
One Winooski Park
Colchester, VT 05439
jtrono@smcvt.edu

Preamble

Recently a colleague wanted to include a section on perfect hashing functions in the latest edition of a book [4], and after an on-line search brought up a reference of mine [8], he sent me an email asking if I could generate a minimal perfect hashing function (MPHF) for the set of reserved words in Ada 2005. This is a summary of the process that determined one.

Introduction

It has been stated that searching and sorting are two of the most prevalent activities that computers are asked to perform. Hashing is one of the fastest ways to search for information, assuming the specific hashing function, collision resolution strategy and hash table size are all reasonable. Perfect hashing occurs when each keyword maps uniquely to its place in the hash table, thereby eliminating all collisions. If the size of the hash table is exactly the same as the number of keywords to be stored, then the hashing function is also minimal.

Cichelli's Idea

One of the earliest papers on minimal perfect hashing functions (MPHF) was written by Cichelli [2]. He conjectured that a MPHF could be found fairly quickly, for a small, static set of keywords, by using a heuristically-guided, backtracking-based search to determine the array of weights in the following hashing function:

h(word) = weights[1st letter in word] + weights[last letter in word] + length(word);

Unfortunately, there are sets of words where the above hashing function is guaranteed to generate at least one collision because there are two words of the same length that share identical first and last letters, e.g., Arizona and Alabama. As it turns out, a MPHF for the fifty United States can be found if you use the first letter and the letter that is two before the final one [7]. This simple variation to Cichelli's hashing function was acceptable because the shortest state name is four letters, and once incorporated, it led to the discovery of a MPHF for this data set. Therefore, it seems reasonable to allow any program that will be trying to generate a MPFH to first search for a viable combination of indices that can be used to extract two letters from the keyword so that guaranteed collisions are avoided. Regrettably, there are still some cases where this modification is not sufficient to guarantee such a combination exists. (The hashing function in [7] also enclosed the entire arithmetic expression on the right hand side of the hashing function above in parentheses, and then added "mod TABLE_SIZE". This provides greater freedom for the program when it is determining the values to

be associated with letters, i.e., the array of weights.)

A Modified MPHF

A MPFH was constructed for the set of 63 reserved words in Ada83 [6] using a slightly more involved hashing function than Cichelli's original one because of the following three guaranteed collisions: raise/range, private/package and exit/type. (A MPHF for the 60 Ada reserved words, after removing the first of each previously listed pair, was found using Cichelli's original hashing function, and was included in [8] for comparison purposes.) The extra term that was added to Cichell's function in [6] – alphabetic position of the second to last letter in the word - alleviated those three guaranteed collisions.

One of the aforementioned simple variations to Cichelli's hashing function examined word[k][1] and word[length(name)-m-1], for all values of k and m between zero and the length of the shortest word -1, searching for a viable pair of letters to extract from each keyword that would avoid all guaranteed collisions [7]. Because Ada contains at least one two-letter reserved word, that really only leaves four viable choices for the pair. For the same reason as is given below for the two keyword pairs 'in' and 'is', and 'if' and 'of', only the first and last letters (or the last and first) might work, but we already know that these generate 3 guaranteed collisions.

There are two related modifications to the original Cichelli hashing function which made it possible to find a MPFH for the set of 72 reserved words in Ada 2005. The first one is concerned with which two letters will be extracted from each keyword. The idea is to find a pair (k,m) such that word[k mod length(word)] and word[m mod length(word)] yield unique, ordered letter pairs for keywords of the same length. To treat the letters as ordered pairs, and not as a set, was necessary; 72 reserved words is a fairly large data set, and it would generate 144 letter extractions, which would probably be divided across 20 letters or so. This would make it very difficult if not impossible to find assignments for the weights array that would generate a MPHF for this set. Because of this situation, the other modification was to use two different weight arrays: one for the first extracted letter, and another one for the second, which is why order now matters. This second modification provides much more flexibility when determining which value a letter would receive, since it could be assigned one value for the first, and a different value for the second extracted letter [1].

Ada 2005 has many reserved words which limit the possible choices for k and m. For example, 'access' and 'accept' are the same length and share the same first four letters. Therefore, to avoid any guaranteed collisions, k and m have to be of the form (6*n + 3, 6*n + 4), (6*n + 4, 6*n + 5), or (6*n + 5, 6*n + 6), where n >= 0, and the first letter of a keyword is assumed to be in position zero. (This choice of zero will be revisited a little later on.)

Two pairs of keywords - 'in' and 'is', and 'if' and 'of' - force the difference between k and m to be odd: if the difference was even, then both letters would come from position zero (or one), which would guarantee collisions for the first (or second) pair of the previously listed, two letter keywords. Therefore, a search for consecutive integral values for k and m was begun (though k and m could possibly differ by any odd integer).

[1] I use this notation here for convenience, assuming word is an array of characters accessed from [0] to [length-1].

After finding 11 more constraints that restrict the possible values for (k,m), where k+1 = m, the integer pairs, for values of k from 0 to 216 matching the three forms listed above, were written on a piece of paper. Pairs were then crossed out in a manner similar to applying the Sieve of Eratosthenes when searching for prime numbers. At first, it looked like more pairs would have to be generated since so many were being eliminated as each constraint was considered. Thankfully, one pair remained after all constraints had been applied – (107,108) – and this pair did select letters so that no guaranteed collisions would occur if two tables of weights, and the length of the keyword, are used in the hashing function.

Strangely enough, 108 = 2 * 2 * 3 * 3 * 3, and this means that word[108 mod length(word)] would be the first letter of those keywords of length 2, 3, 4, 6, 9 and 12, and word[107 mod length(word)] would be the last, just like in Cichelli's original function! Only four lengths would access different positions in word: lengths 5 and 7 would map to accessing positions 2 and 3, length 8 to positions 3 and 4, and length 10 to 7 and 8 (where all other keywords would map to the last and the first letter, using 107 and 108, in that order). Therefore, the hashing function for the 72 reserved words in Ada 2005 would be:

```
h(word)=(weights1[word[107 mod length(word)]] + weights2[word[108 mod length(word)]] +
                length(word)) mod 72;
```

Now that all guaranteed collisions have been eliminated, the frequency of the letters selected in both positions could be determined for this set of keywords, and the technique described in [7, 8] could be applied (by hand). There were sixteen letters that appeared only once as either the first or second one selected, and therefore fifteen keywords could be placed anywhere in the hash table because there were no restrictions as to what weight could be assigned to those "wildcard" letters. (Both selected letters in 'body' are "wildcards".)

Tentative weights were assigned to the most popular of the first and second letters selected: 5 being chosen in each category. After dividing 72 by 5, 0 was assigned to 't', 14 to 'n', 28 to 'd', 42 to '3' and 56 to 's' for weights1 in an attempt to distribute the keywords throughout the hash table. Likewise for weights2, except a different initial weight was used along with slightly staggered offsets, starting with 3 for 'a', 18 for 't', 32 for 'i', 51 for 'r' and 68 for 'e'. All 72 reserved words were then listed in descending order by the sum of the frequencies for the two letters extracted, along with the two letters and the word's length. Using the initial weight assignments above, the hash values that were already determined using those 10 weights were found to be unique for the 25 keywords they applied to. Judicious weight assignments were made, one at a time, to the letters which were associated with the most keywords still to be placed, looking ahead to avoid collisions. Before long, all 57 reserved words not containing a wildcard had been placed, and then it was straightforward to assign weights to the 16 wildcards, thereby completely filling up the hash table. (The two arrays - weights1 and weights2 - can be found in Appendix A, and information about where the reserved words are placed using this hashing function appears in Appendix B.)

A code excerpt from [4] can be found on the next page, where '+1' has been added to each pair of integers that are used to extract a letter from Name, since Strings in Ada begin at 1 (not zero, as was assumed in the hashing function shown above). This Ada code (provided by John McCormick) was

roughly a factor of three faster than the binary search implementation, for a sample run of 100,000 Ada 2005 reserved words.

```
Number_Of_Reserved_Words : constant := 72;

subtype Index_Range is Natural range 0 .. Number_Of_Reserved_Words - 1;
subtype Lowercase   is Character range 'a' .. 'z';
type    Table_Array is array (Lowercase) of Natural;

Table_1 : constant Table_Array :=
   (41, 33, 35, 28, 42,  9, 14, 1, 21,  0, 22, 32, 65,
    14, 11, 14,  2, 12, 56,  0, 0, 71, 30,  0,  0,  0);

Table_2 : constant Table_Array :=
   ( 3, 4,  8, 17, 68,  8, 34, 0, 32,  0, 26, 15, 35,
    42, 1, 60,  0, 51, 20, 18, 5, 42, 11,  7,  0,  0);

function Hash (Name: in String) return Index_Range is
-- Preconditions : Name contains only lowercase characters
--                 Name contains at least one character
begin
   return (Table_1 (Name(107 rem Name'Length + 1)) +
           Table_2 (Name(108 rem Name'Length + 1)) +
           Name'Length) rem Number_Of_Reserved_Words;
end Hash;
```

Conclusion

A minimal perfect hashing function (MPHF) for the 72 reserved words in Ada 2005 has been found, and this hashing function is very similar in nature to the style proposed by Cichelli back in the early 1980s [2]. More recent work has identified hashing functions which can be made perfect and minimal for very large data sets [5, 3], and so these strategies probably would determine a MPFH for this relatively small set of keywords quickly. However, since implementations of those were not available, finding this one by pencil and paper, rather than modifying the program(s) mentioned in the comparison study in [8] to use the modified hashing function outlined in this article, was quite exciting and rewarding.

References

[1] Brain and Tharp, Near-Perfect Hashing of Large Word Sets, *Software Practice and Experience*, Volume 19, #10, 1989.

[2] Cichelli, Minimal Perfect Hashing Functions Made Simple, *Communications of the ACM*, Volume 23, #1, 1980.

[3] Czech, Havas, and Majewski, An optimal algorithm for generating minimal perfect hash functions, *Information Processing Letters*, Volume 43, 1992.

[4] Dale and McCormick, *Ada 2005 Plus Data Structures: An Object-Oriented Approach*, Jones and Bartlett, 2nd edition (not yet released - expected 2006).

[5] Fox, Heath, Chen and Daoud, Practical Minimal Perfect Hash Functions for Large Databases, *Communications of the ACM*, Volume 35, #1, 1992.

[6] Sebesta and Taylor, Fast Identification of Ada and Modula-2 Reserved Words, *Journal of*

Pascal, Ada and Modula-2, March/April 1986.
[7] Trono, An Undergraduate Project to Compute Minimal Perfect Hashing Functions, *SIGCSE Bulletin*, Volume 24, #3, 1992.
[8] Trono, A Comparison of Three Strategies for Computing Minimal Perfect Hashing Functions, *SIGPLAN Notices*, Volume 27, #11, 1992.

Appendix A – weights assigned to letters

char	weight2[char]	Weight2[char]
a	41	3
b	33	4
c	35	8
d	28	17
e	42	68
f	9	8
g	14	34
h	1	0
i	21	32
j	0	0
k	22	26
l	32	15
m	65	35
n	14	42
o	11	1
p	14	60
q	2	0
r	12	51
s	56	20
t	0	18
u	0	5
v	71	42
w	30	11
x	0	7
y	0	0
z	0	0

Appendix B – where all 72 Ada 2005 reserved words are placed

The column denoted k[1st] holds the index and letter selected by (107 mod length of the keyword), where L denotes the last letter in the keyword and zero the first: k[2nd] denotes the index and letter selected by (108 mod length of the keyword); w1[1st] is the value for the letter at k[1st]; and w2[2nd] is the value for the letter at k[2nd].

[ ]	keyword	k[1st]	k[2nd]	w1[1st]	w2[2nd]	length
0	exit	L-t	0-e	0	68	4
1	overriding	7-i	8-n	21	42	10
2	reverse	2-v	3-e	71	68	7
3	new	L-w	0-n	30	42	3
4	out	L-t	0-o	0	1	3
5	at	L-t	0-a	0	3	2
6	null	L-l	0-n	32	42	4
7	digits	L-s	0-d	56	17	6
8	body	L-y	0-b	0	4	4
9	accept	L-t	0-a	0	3	6
10	constant	3-s	4-t	56	18	8
11	interface	L-e	0-i	42	32	9

12	of	L-f	0-o	9	1	2
13	record	L-d	0-r	28	51	6
14	requeue	2-q	3-u	2	5	7
15	or	L-r	0-o	12	1	2
16	with	L-h	0-w	1	11	4
17	generic	2-n	3-e	14	68	7
18	is	L-s	0-i	56	32	2
19	exception	L-n	0-e	14	68	9
20	array	2-r	3-a	12	3	5
21	elsif	2-s	3-i	56	32	5
22	xor	L-r	0-x	12	7	3
23	for	L-r	0-f	12	8	3
24	renames	2-n	3-a	14	3	7
25	protected	L-d	0-p	28	60	9
26	select	L-t	0-s	0	20	6
27	end	L-d	0-e	28	68	3
28	separate	3-a	4-r	41	51	8
29	when	L-n	0-w	14	11	4
30	do	L-o	0-d	11	17	2
31	aliased	2-i	3-a	21	3	7
32	limited	2-m	3-i	65	32	7
33	loop	L-p	0-l	14	15	4
34	and	L-d	0-a	28	3	3
35	pragma	L-a	0-p	41	60	6
36	then	L-n	0-t	14	18	4
37	until	2-t	3-i	0	32	5
38	all	L-l	0-a	32	3	3
39	procedure	L-e	0-p	42	60	9
40	delay	2-l	3-a	32	3	5
41	while	2-i	3-l	21	15	5
42	else	L-e	0-e	42	68	4
43	if	L-f	0-i	9	32	2
44	task	L-k	0-t	22	18	4
45	not	L-t	0-n	0	42	3
46	raise	2-i	3-s	21	20	5
47	rem	L-m	0-r	65	51	3
48	in	L-n	0-i	14	32	2
49	goto	L-o	0-g	11	34	4
50	use	L-e	0-u	42	5	3
51	begin	2-g	3-i	14	32	5
52	tagged	L-d	0-t	28	18	6
53	range	2-n	3-g	14	34	5
54	case	L-e	0-c	42	8	4
55	delta	2-l	3-t	32	18	5
56	entry	2-t	3-r	0	51	5
57	declare	2-c	3-l	35	15	7
58	subtype	2-b	3-t	33	18	7
59	abstract	3-t	4-r	0	51	8
60	synchronized	L-d	0-s	28	20	12
61	function	3-c	4-t	35	18	8
62	abs	L-s	0-a	56	3	3
63	others	L-s	0-o	56	1	6
64	type	L-e	0-t	42	18	4
65	access	L-s	0-a	56	3	6
66	mod	L-d	0-m	28	35	3
67	abort	2-o	3-r	11	51	5
68	package	2-c	3-k	35	26	7
69	terminate	L-e	0-t	42	18	9
70	private	2-i	3-v	21	42	7
71	return	L-n	0-r	14	51	6