01 Jan 1984

An Improved Algorithm for Generating Minimal Perfect Hash Functions

Thomas J. Sager

An Improved Algorithm for Generating Minimal Perfect Hash Functions

Thomas J. Sager

CSc-84-1

Department of Computer Science
University of Missouri-Rolla
Rolla, MO 65401 (314)341-4491

Title: An Improved Algorithm for Generating Minimal Perfect Hash Functions.

Author: Thomas J. Sager

Address: Department of Computer Science
Univerisity of Missouri - Rolla
Rolla, Mo. 65401 USA
(314) 341- 4856

Abstract

A minimal perfect hash function (MPHF) is a function from a set of M objects to the first M non-negative integers. MPHF's are useful for the compact storage and fast retrieval of frequently used objects such as reserved words in a programming language or commonly employed words in a natural language. In this paper we improve on an earlier result and present an algorithm for generating MPHF's with an expected time complexity proportional to M^4. We also give a MPHF for the 256 most frequently used words in the English language.

Categories and Subject Discriptors:

E.2 [Data Storage Representation]
hash table representations

H.3.3 [Information Search and Retrieval]
retrieval models, search process, selection process.

I.2.7 [Natural Language Processing]

General Terms: Algorithms, Performance, Languages.

Additional Keywords and Phrases:
searching, hashing, minimal perfect hashing.

1. Introduction

A perfect hash function is an injection, F, from a set, W, of M objects to the first N non-negative integers where N >= M. If N = M then we say that F is a minimal perfect hash function. Minimal perfect hash functions are useful for compact storage and fast retrieval of frequently employed sets of objects such as reserved words in a programming language or commonly used words in a natural language.

Algorithms for generating perfect hash functions have been presented by Sprugnoli[5], CIchelli[1], Jaeschke[3] and Sager[4]. Whereas the algorithms presented in [1,3,5] have an expected execution time exponential in M, Sager's mincycle algorithm[4] has an expected execution time proportional to M^5.

In this paper we present an improvement on the mincycle algorithm which reduces its expected time complexity to M^4. We also give a minimal perfect hash function for the 256 most frequently used words in the English language as compiled by Dewey[2]. Dewey[2] found that these 256 words are used with a frequency of over 64.2%

The use of minimal perfect hash functions for looking up most frequently used words should speed up many natural language processing applications immensely. The technique presented here should be equally applicable to the Chinese language or any other natural language.

Given the limits of our computer resources, it did not seem feasible to increase M much beyond 256 without incurring a considerable expense in recoding and computer time. However, given sufficient resources, it should be feasible to find minimal perfect hash functions for sets of 1K or more words.

In section 2 we briefly review the mincycle algorithm. The interested reader should refer to [4] for a more complete discussion of the mincycle algorithm. Section 3 contains an improvement to the mincycle algorithm. Section 4 contains some concluding remarks. Appendix 1 gives a minimal perfect hash function for the 256 most frequently used words in the English language.

2. Mincycle Algorithm

The problem can be stated as:

"Given a set, W, of words and an integer N >= M = card(W), find a quickly computable injection F: W ---> [0..N-1]. For minimal perfect hash functions let N = M.

We break the problem down into two parts:

Part 1.

Let

R be the power of 2 closest to M,

$r = R/2$,

$V = [0..R-1]$,

h_0: W ---> [0..N-1] be defined by $h_0(w) =$

(length(w) + (Σord(w[i]), i := 1 to length(w) by 3)) mod N,

h_1: W ---> [0..r-1] be defined by $h_1(w) =$

(Σ ord(w[i]), i := 1 to length(w) by 2) mod r and

h_2: W ---> [r..R-1] be defined by $h_2(w) =$

(Σ ord(w[i]), i := 2 to length(w) by 2) mod r + r.

Note that h_0, h_1 and h_2 are quickly computable pseudo-random functions. It is important that h_0, h_1 and h_2 do not all agree on any pair of members of W. In the event we wish to find a perfect hash function for a set W on which h_0, h_1 and h_2 agree on some pair of members, we may substitute any three quickly computable pseudo-random functions with equivalent ranges for h_0, h_1 and h_2.

We now restate the problem as:

"find a function g: V ---> [0..N-1] such that

$F(w) = (h_0(w) + g \circ h_1(w) + g \circ h_2(w)) \bmod N$

has the desired properties.

Our method is to consider the sequence of graphs H_1, H_2, H_3... where each $H_i = \langle V_i, E_i \rangle$. Each V_i is a partition of V. $V_0 = \{\{v\} \mid v \in V\}$ and

$E_0 = \{(h_1(w), h_2(w)) \mid w \varepsilon W\}$. Each H_i is loop-free but may contain multi-edges (more than one edge connecting the same pair of vertices).

We construct each H_{i+1} from H_i in the following manner:

1: Choose e_i in E_i such that e_i lies on a maximal number of minimal length cycles of H_i. For our purposes we consider that two edges connecting the same pair of vertices form a cycle of length 2 and that an edge which lies on no cycles at all is on a cycle of length ∞.

2: Delete all edges in H_i connecting the two vertices connected by e_i and then merge these two vertices.

Let A and B be the two vertices of H_i connected by e_i. In constructing H_{i+1} from H_i, let

$$w_i = \{w \varepsilon W \mid (h_1(w) \varepsilon A \text{ and } h_2(w) \varepsilon B) \text{ or } (h_1(w) \varepsilon B \text{ and } h_2(w) \varepsilon A)\}$$ and

$u_i = \{h_1(w), h_2(w)\}$ for some $w \varepsilon w_i$.

We stop when E_i is empty. Let k be the number of iterations performed. Note that $k < R \le 3N/2$ necessarily.

The original mincycle algorithm found the edge lying on the maximum number of minimal length cycles through an exhaustive search, a process taking time proportional to R^4. In the following section we present an improved algorithm which takes time proportional to R^3.

Part 2.

Let

$U_0 = \emptyset$,

$U_j = \{u_i \mid i <= j\}$,

$W_0 = \emptyset$,

$W_j = \{ \cup w_i,\ i := 1 \text{ to } j\}$ and

G: U_k ---> [0..N-1] be defined by $G(u) = (\Sigma\, g(v),\ v \,\varepsilon\, u) \bmod N$.

Note that $\forall w \in w_j$, $F(w)$ is uniquely determined by the values of $G(u_1), G(u_2), \ldots, G(u_j)$ regardless of the function g. Also note that there always exists at least one function g consistent with G. These facts follow if we consider V as an orthogonal basis for a vector space and W as the set of vectors $\{h_1(w) \oplus h_2(w) \mid w \quad W\}$ over the space defined by the basis V. In choosing $u_1, u_2, \ldots, u_k$, we are attempting to maximize the subset of W in the subspace whose basis is $\{u_i, u_2, \ldots, u_j\}$, $\forall j \,\varepsilon\, [1..k]$. $\forall j \,\varepsilon\, [1..k]$, W_j is precisely this subset. This follows from:

Theorem 1: Let $X \subseteq W$. X considered as the edges of a graph is cycle free iff X considered as a set of vectors is linearly independent.

Theorem 1 has been proved in [4]. We do not give the proof here.

The algorithm for part 2 is given in figure 1. Note that it is a back-tracking algorithm and that its worst case time complexity is exponential in M. Also note that it is not guaranteed to succeed. We have found empirically, however, that when h_0, h_1 and h_2 are pseudo-random enough and $R > 2M/3$, part 1 tends to dominate and the expected time complexity of the entire algorithm is therefore proportional to M^4. We have found no example where, with minor manipulation of the functions h_0, h_1 and h_2, the algorithm can not be made to succeed. This is to be expected since when $R > 2M/3$, the graphs H_i are quite sparse.

3. An Improved Mincycle Algorithm

Our algorithm for finding the edge of a graph lying on a maximal number of minimal length cycles is given in figure 2. One should note its similarity to Warshall's algorithm for finding the transitive closure of a relation. Essentially we find the number of paths between each pair of vertices that are either of minimal length or one more than minimal length. Data about shortest paths is then combined to form data about shortest cycles.

Figure 1.

```
algorithm Part2;
   input     k: upper bound of u and w;
             u: array [1..k] of record
                   a, b: vertices of u[i] end;
             w: array [1..k] of set of words;
             R: number of vertices;
             M: number of words;
             N: size of hash table;
   output    success: boolean;
             g: array [0..R-1] of 0..N-1;
             F: array [0..M-1] of 0..N-1;
   var       G: array [1..k] of 0..N;
   { search for a function G which makes F a perfect hash }
   { function.  If found then compute g consistent with G.}
begin
   forall i in [1..k] do G[i] := N;
   i := 1;
   while i in [1..k] do
      G[u[i]] := (G[u[i]] + 1) mod (N + 1);
      conflict := true;
      while (G[u[i]] < N) and conflict do
         conflict := false;
         forall x in w[i] do compute F(x) from G;
         forall x in w[i], j <= i, y in w[j], x <> y do
            if F[x] = F[y] then conflict := true;
         if conflict then G[u[i]] := (G[u[i]] + 1) mod (N + 1);
      if conflict then i := i - 1 else i := i + 1;
   if i = 0 then success := false
   else success := true; compute g consistent with G
end;
```

Figure 2.

```
algorithm Bestedge;
   input     n: number of vertices in graph - 1;
             adj: adjacency matrix;
   output    a, b: 2 vertices of edge which is on a maximal
                   number of minimal length cycles;
   var paths: array [0..n, 0..n] of record
         minlnth:  length of shortest path;
         nminl:    number of shortest length paths;
         nminll:   number of paths of shortest length + 1 end;
      { assume input graph contains no multiple edges or loops }
      { assume input contains at least one cycle }
begin
   limit := maxint / 2;
   forall x, y in [0..n] do
      with paths[x,y] do
         if adj[x,y] then minlnth := 1; nminl := 1; nminll := 0
         else minlnth := limit; nminl := 0; nminll := 0;
   forall x in [0..n] do
      forall y, z in [0..n] such that x, y and z are distinct do
         w := paths[y,x].minlnth + paths[x,z].minlnth;
         if w <= limit then with paths[y,z] do
            if w = minlnth + 1 then
               nminll := nminll + 1; limit := w; even := false
            elsif w = minlnth then
               nminl := nminl + 1;
               if w < limit then
                  limit := w; even := true
            elsif w = minlnth - 1 then
               if w < limit then
                  nminll := nminl; limit := w + 1; even := false;
               minlnth := w; nminl := 1
            elsif w < minlnth - 1 then
               minlnth := w; nminl := 1; nminll := 0;
   maxncyc := 0;  { maximum number of cycles }
   case even of
      true:
         forall x, y in [0..n] such that x < y and adj[x,y] do
            ncyc := 0;
            forall z in [0..n] do
               if (path[x,z].minlnth = limit) and
                  (path[y,z].minlnth = limit - 1)
               then  ncyc := ncyc + path[x,z].nminl - 1;
            if ncyc > maxncyc then maxncyc := ncyc; a := x; b := y
      false:
         forall x, y in [0..n] such that x < y and adj[x,y] do
            ncyc := 0;
            forall z in [0..n] do
               if (path[x,z].minlnth = limit - 1) and
                  (path[y,z].minlnth = limit - 1)
               then  ncyc := ncyc + 1;
            if ncyc > maxncyc then maxncyc := ncyc; a := x; b := y;
end;
```

<u>Figure 3</u>: Example of application of mincycle algorithm

Given: W = {AA, AAD, AB, BAA, BB, FA}.

Choose: N = 6, R = 8 and ASCII character code.

Results:

	AA	AAD	AB	FA	BB	BAA
h_0	1	2	1	0	2	3
h_1	1	1	1	2	2	3
h_2	5	5	6	5	6	5
F	1	2	3	0	4	5

i	1	2	3	4	5	6	7	0
u_i	{1,5}	{1,6}	{2,5}	{3,5}				
w_i	{AA,AAD}	{AB}	{FA,BB}	{BAA}				
$G(u_i)$	0	2	0	2				
g(i)	0	0	2	0	0	2	0	0

4. Conclusion

In totality, the mincycle algorithm now has an expected time complexity proportional to M^4 and a space complexity proportional to M^2. Compiling on the PASCAL 8000 compiler under T- option and running on an Amdahl V8 in a partition of 5M, a minimal perfect hash function for the 256 most frequently used words in the English language was found in slightly more than 45 seconds of CPU time. With an optimizing compiler and a larger computer system, it should be feasible to find minimal perfect hash functions for wordsets of size 1K or more using the mincycle algorithm.

It is expected that such minimal perfect hash functions will prove useful in natural language processing and other applications.

References:

[1] Cichelli, R.J.: Minimal perfect hash functions made simple. Comm. ACM, 23, 1 (Jan. 1980), 17-19.

[2] Dewey, G.: Relativ frequency of English speech sounds. Harvard Univ. Press, 1923.

[3] Jaeschke, G.: Reciprocal hashing: A method for generating minimal perfect hashing functions. Comm. ACM, 24, 12, (Dec. 1981) 829-833.

[4] Sager, T.: A polynomial time generator for minimal perfect hash functions. 1983 (to appear soon)

[5] Sprugnoli, R.: Perfect hashing functions: a single probe retrieval method for static sets. Comm. ACM, 20, 11, (Nov. 1977) 841-850.

Appendix 1: Minimal Perfect Hash Function for 256 most commonly used English words. (using ASCII character code)

i	g(i)	$F^{-1}(i)$	i	g(i)	$F^{-1}(i)$
0	0	WAS	50	143	WHOSE
1	0	PAY	51	0	THEIR
2	0	MAN	52	0	DONE
3	0	MANY	53	0	MIGHT
4	0	GET	54	0	THESE
5	3	DAY	55	0	MADE
6	3	INTO	56	212	WE
7	140	FAR	57	0	THOSE
8	50	WAR	58	0	UNDER
9	241	PER	59	0	SUCH
10	47	MEN	60	0	GIVEN
11	0	GOT	61	0	WHERE
12	96	TOOK	62	108	GREAT
13	0	NEW	63	60	SAYS
14	0	NOT	64	0	ONCE
15	40	FEW	65	0	WENT
16	198	OWN	66	66	WHOLE
17	205	PART	67	4	FOOD
18	206	HOW	68	140	OUT
19	243	MUCH	69	0	HOME
20	35	NOW	70	0	HIS
21	219	NAME	71	136	MAY
22	232	FOR	72	105	CASE
23	228	LAST	73	245	SHE
24	182	LET	74	0	WAY
25	230	TWO	75	0	HER
26	240	SAY	76	0	YEARS
27	170	CAN	77	50	LIFE
28	224	SEE	78	127	THEM
29	182	GOING	79	245	THINK
30	92	WANT	80	83	YOUNG
31	176	MORE	81	0	GOOD
32	185	MAKE	82	71	FACT
33	234	TAKE	83	121	ORDER
34	166	THERE	84	119	SHALL
35	186	THIS	85	64	BOTH
36	115	WILL	86	0	ABOUT
37	172	AMONG	87	207	NIGHT
38	239	CAME	88	30	LEFT
39	95	WORLD	89	0	FIVE
40	104	TOO	90	69	MEANS
41	166	SAME	91	3	BEST
42	158	DAYS	92	157	WHILE
43	102	COME	93	0	GUN
44	188	RIGHT	94	200	THAN
45	88	TELL	95	128	STEEL
46	32	SOME	96	215	THING
47	185	TAKEN	97	224	SMALL
48	232	GIVE	98	35	STILL
49	0	WELL	99	45	LIKE

i	g(i)	$F^{-1}(i)$	i	g(i)	$F^{-1}(i)$
100	220	LONG	150	221	BE
101	219	BUT	151	223	BY
102	230	WHEN	152	157	IT
103	224	WERE	153	230	THEY
104	203	JUST	154	175	OF
105	236	THEN	155	194	AT
106	0	SIDE	156	14	ANOTHER
107	193	OTHER	157	180	MATTER
108	76	USED	158	209	SINCE
109	189	HAND	159	70	FIGHTING
110	231	MUST	160	243	MORNING
111	0	LINE	161	2	ENOUGH
112	115	SAID	162	241	HERE
113	70	TIME	163	151	BELIEVE
114	66	UNTIL	164	18	PLACE
115	0	WHAT	165	48	CANNOT
116	74	OUR	166	0	PEACE
117	240	CITY	167	118	COUNTRY
118	14	ANY	168	12	PURPOSE
119	0	HIM	169	16	BUSINESS
120	0	FRONT	170	197	SERVICE
121	70	ALSO	171	14	THROUGH
122	0	THE	172	195	ARMY
123	0	NEXT	173	0	ALL
124	0	DEAR	174	0	OLD
125	0	DOES	175	0	SOMETHING
126	0	YEAR	176	18	AWAY
127	192	WHY	177	0	BACK
128	76	THREE	178	0	YOURS
129	0	OFF	179	0	HAD
130	0	IN	180	0	HOUSE
131	0	OH	181	0	STAND
132	0	ITS	182	195	BEING
133	245	AN	183	0	ONE
134	212	WHO	184	0	FROM
135	65	BIG	185	0	POWER
136	14	ON	186	0	MONEY
137	8	AND	187	0	PUBLIC
138	33	IS	188	0	WOMEN
139	135	I	189	0	WOMAN
140	29	HIGH	190	0	TODAY
141	33	AS	191	0	ALWAYS
142	234	A	192	0	AGAIN
143	12	AGAINST	193	8	HAVE
144	222	BECAUSE	194	0	SITUATION
145	208	ME	195	0	HE
146	0	MY	196	0	YET
147	218	YOU	197	16	SET
148	217	IF	198	84	KNOW
149	195	THAT	199	0	NEVER

i	g(i)	$F^{-1}(i)$	i	g(i)	$F^{-1}(i)$
200	61	BETWEEN	228	22	TO
201	81	SHOULD	229	96	SO
202	0	DID	230	110	NO
203	0	EACH	231	16	WITHOUT
204	105	ONLY	232	57	GO
205	160	FOUND	233	74	DO
206	66	THINGS	234	0	PRESENT
207	23	DURING	235	254	UPON
208	75	THOUGHT	236	0	VERY
209	0	YOUR	237	0	BEFORE
210	153	FIND	238	0	INTEREST
211	74	NOTHING	239	182	MILITARY
212	88	OVER	240	0	LESS
213	56	EVERY	241	0	WHICH
214	0	EVER	242	0	AFTER
215	8	GOVERNMENT	243	204	COULD
216	0	EVEN	244	0	DOWN
217	17	WITH	245	0	MOST
218	119	HIMSELF	246	161	HALF
219	0	POSSIBLE	247	0	DON'T
220	0	CALL	248	188	FIRST
221	0	WOULD	249	0	LITTLE
222	181	PEOPLE	250	0	BEEN
223	0	OR	251	113	WORK
224	241	ARE	252	0	SAW
225	52	US	253	0	HAS
226	7	UP	254	0	PUT
227	55	AM	255	0	SOON