ISE-TR-80-18

# A COLLISION RESOLUTION TECHNIQUE BY USING PREDICTORS

by

Seiichi Nishihara

August 15, 1980

INSTITUTE
OF
INFORMATION SCIENCES AND ELECTRONICS

UNIVERSITY OF TSUKUBA

ISE-TR-80-18

# A COLLISION RESOLUTION TECHNIQUE BY USING PREDICTORS

Seiichi Nishihara

August 15, 1980

Institute of Information Sciences and Electronics
University of Tsukuba
Sakura-mura, Niihari-gun, Ibaraki 305 Japan

## 1. Introduction

The most remarkable feature of hash addressing is that the average number of probes depends just on the fraction $\alpha$ of the table that is occupied; it is not affected by the total number of keys. Since any key-to-address transformation generally makes a many-to-one mapping, it will probably happen that more than one distinct keys are hashed to the same address. Those keys having the same home address are called synonyms. Such an occurrence, called a collision, causes many kinds of clustering phenomena[1].

Many techniques of resolving collisions have been proposed. They are classified mainly into two categories: open addressing and chaining[3,4]. In open addressing, in addition to the hash function H, which determines the home address H(k) of a key k to be stored, a collision-resolution function h is necessary for tracing through the table until an empty cell is encountered. The probe-sequence generated by the function h for a key k is expressed as h(i,k), $i=0,1,\dots,M-1$, where M is the table size. Here, $h(0,k)=H(k)$ and $0 \le h(i,k) \le M-1$, for $i=1,2,\dots$. The other approach to collision resolution is the direct chaining method[4], in which all the keys transformed to the same address are kept in a chain using simple list processing techniques.

Both hashing techniques and many variations are surveyed in references [3] and [4], and details are omitted here. Assuming equal usage of cells, the theoretical approximation of the average number of probes necessary to retrieve a key in a hash table has been given for each method[3,4]:

$-(1/\alpha)\ln(1-\alpha)$, for open addressing eliminating primary and secondary clusterings[1],

$1+\alpha/2$, for direct chaining, (1)

where $\alpha$ is the load factor of the table. In general, the average number of probes needed in open addressing cannot be less than that needed in chaining. In a recent paper[2], a method named pseudochaining which combines characteristics of open addressing and chaining was proposed. The performance of pseudochaining lies between those of the other two methods.

In this paper, another combined method is proposed, whose performance in terms of the average number of probes is essentially equal to that of chaining. In the following sections, a new method using a predictor, a several bit field assigned to each cell, and an extension of this method are described. Then the retrieval efficiencies of the predictor method and its extension, called the multiple predictor method, are estimated theoretically and verified by experiments. In conclusion, it is proved that a predictor of more than four or five bit length is always preferable to chaining from the viewpoint of efficient use of memory, and furthermore that the multiple predictor breaks through the limitation $1+\alpha/2$ of direct chaining without expending extra space.

## 2. Description of the methods

### 2.1. The single predictor method

Our technique is applied to the open addressing method in which secondary clustering may occur. A hash table of size M is a set of M successive cells addressed from 0 to M-1. Each cell contains not only an item space(key field) but also a p bit field as a predictor. It holds a nonnegative integer q, which is used for the purpose of tracing only synonyms, i.e. keys in the same cluster, where $0 \le q \le (2^p-1)$.

Assume that the search for key k is now being performed at the address h(i,k), i.e. none of the cells h(0,k),...,h(i,k) contains the key k. In the usual open addressing method, the next search address is h(i+1,k). However, if the key in the h(i+1,k)-th location is not a synonym of k, there is no need to check this location. In this case, the predictor value q is used to indicate the number of probes to be skipped until the next address containing a synonym is encountered. In other words, the next synonym is found in the h(i+q,k)-th location. Note that the function h(i,k) is assumed to be directly computable; it takes a key value, k, and a nonnegative integer, i, as arguments and returns an address in the table. The predictor of the last cell of a cluster is set to zero. It means that no more synonyms exist in the table, which is the natural extension of the interpretation of predictors and is very effective for reducing reject time. In some cases the number of probes that should be skipped in the search for another synonym is greater than the maximum predictor value max($=2^p-1$). When this occurs, after checking the h(i+max,k)-th location, we must repeat probing operations one by one until a synonym is encountered following the probe sequence. This phenomenon is the only factor that makes the average number of probes greater than that of the direct chaining method. The additional cost is estimated in the following section.

The algorithms for storing and retrieving keys may easily be derived and so are not formulated here. The detailed algorithm given later for the multiple predictor method includes the above algorithm as a special case.

## 2.2. The multiple predictor method

The multiple predictor method employs more than one predictor and a predictor-selecting function g in addition to the collision-resolution function h. The difference from the single predictor method is the number N of predictors reserved in each cell. The function g is used to determine which predictor should maintain the synonym cluster. Synonym here means keys having the same value of g as well as of the hash function H. Assume that the predictor number g(k) for a key k is determined independently of H(k), and $1 \leq g(k) \leq N$ holds. The basic idea of the algorithm is much the same as that using a single predictor, except that the multiple predictor algorithm uses the g(k)-th predictor of each cell rather than the single one. Let us introduce a function h' using the collision-resolution function h as

$$h'(i,k)=\begin{cases} h(0,k)\ (=H(k)), & \text{for } i=0, \\ h(i+\Delta,k), & \text{for } i\geq 1, \end{cases}$$

where $\Delta$ is the bias determined by g(k) as

$$\Delta = M\cdot(g(k)-1)/N.$$

First the home address h'(0,k)(=H(k)) of the key k to be searched for is computed. If h'(0,k) does not contain the key k, then the second address to be checked is h'(L(h'(0,k),g(k)),k), where L(a,n) indicates the value of the n-th predictor in the a-th cell. In general, the next candidate address containing a synonym after checking the h'(i,k)-th cell is given as h'(i+L(h'(i,k),g(k)),k). It may occasionally happen that even the maximum predictor value cannot represent the number of probes to be skipped to reach the next candidate address, in which case the search must continue with on-by-one probing.

Now we give algorithms to store or search for key k by using PASCAL-like expressions. In the following, T, L, M, N and max are non-local variables denoting the hash table, the table for multiple predictors, the table size, the number of predictors reserved per cell and the maximum value of a predictor, respectively. T[a] and L[a,n] mean the key and the n-th predictor corresponding to the a-th cell.

### 2.2.1. The storing algorithm

Initially the elements of T and L are all empty. The function h' and g, assumed to be defined outside the procedure, give the probe sequence and the predictor identifier for a given key k to be stored. The algorithm is as follows, where the variables a, kw, aw, i, n, b, q and qw mean the home address of k, the key occupying the home address of k, the home address of kw, the position in the probe sequence, the predictor identifier($\leq$N) for k, a candidate address of an empty cell, a predictor value and the temporary record of a predictor value used by 'updatepredictor' procedure, respectively.

```
procedure store(k:integer);
   label 1,2;
   const empty=0 {means empty};
   a,kw,aw,i,n,b,q,qw:integer;
   procedure updatepredictor;
      w:integer;
   begin if q>max then w:=max else w:=q;
         if w<>qw then L[a,n]:=w
   end;
begin {main procedure}
   a:=h'(0,k);
   if T[a]=empty then storeitem(a,k) {completed}
   else
      begin kw:=T[a]; aw:=h'(0,kw);
         if aw<>a then {displace the non-synonym key kw}
                    begin L[a,g(kw)]:=0; storeitem(a,k);
                          k:=kw; a:=aw
                    end;
{hereafter, k:the key to be stored, a:the home address of k}
        n:=g(k); b:=a; i:=0;
        1:q:=L[b,n]; qw:=q;
        if q=0
        then {search empty cell}
           repeat q:=q+1; i:=i+1;
                  if i>M then table-full else b:=h'(i,k)
```

```
          until T[b]=empty
       else {trace the cluster}
          2:if i+q>M then table-full
            else begin b:=h'(i+q,k);
                    if h'(0,T[b])=a and g(T[b])=n
                    then begin i:=i+q; updatepredictor; goto 1
                         end
                    else begin q:=q+1;
                               if T[b]<>empty then goto 2
                         end
                 end;
       storeitem(b,k); updatepredictor {completed}
    end
end.
```

As the basic rule, starting from the home address the cluster for the key k to be stored is traced through by using the g(k)-th predictor till an empty cell is encountered. If the home address is occupied by some key whose home address is different, then that key is moved to another location(item displacement), and the predictor that pointed to the item displaced must be corrected. The procedure 'updatepredictor' is used to change the incorrect predictor.

## 2.2.2. The search algorithm

```
procedure search(k:integer);
   a,b,n,i:integer;
begin
   a:=h'(0,k); b:=a;  n:=g(k);  i:=0;
   while T[b]<>k and L[b,n]<>0
   {i.e. not equal to k, but synonyms are not exhausted}
   do begin q:=L[b,n];  i:=i+q;  b:=h'(i,k);
         if q>=max
         then while h'(0,T[b])<>a or g(T[b])<>n
              {search a synonym one-by-one}
              do begin i:=i+1;  b:=h'(i,k) end;
      end;
   if T[b]=k then found else not-found
end.
```

Searching is much simpler than storing. The program includes two while-statements. The second one is executed only if $q \geq max$ holds. However, if the length of the predictor field is chosen to be more than 4 or 5 bits, such cases will be very rare. Probing is caused in evaluating the logical expression in the first of these while-statements. As noted earlier, the absence of the key to be retrieved is effectively treated by the final statement.

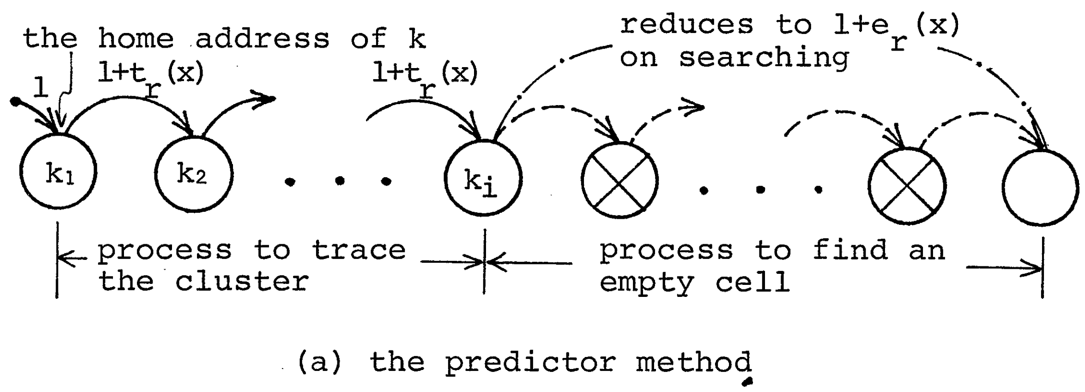
## 3. Searching efficiency of the algorithms

In this section, the searching efficiency of the two methods proposed in the preceding section is analyzed in terms of the mean number of probes.

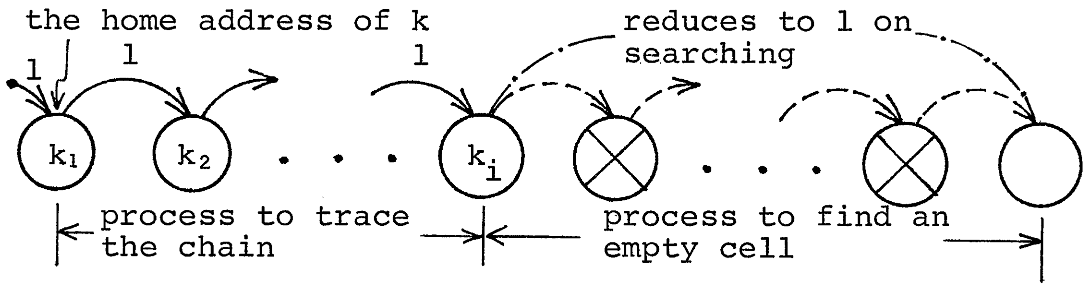
First we consider the basic method using just one predictor. Let p and x be the bit length of a predictor and the load factor respectively. Then the maximum value r of a predictor, denoted 'max' in the procedures, is $2^p-1$. Assume that each cell in the table is hit as frequently as any other. Then, the probability that i keys are hashed to any one cell may be given by the Poisson approximation $e^{-x}\cdot x^i/i!$.

Figure 1 shows the storing process for key k when the number of synonyms already stored is i, i.e. the hash addresses of $k_1,\ldots,k_i$ and k are all identical. First the tracing process of the cluster takes place, as shown by solid arrows. Then the scanning process to find an empty cell follows, as indicated by dashed arrows.

Let us estimate the excess cost caused by those two processes shown in Figure 1 over the direct chaining method. Starting from the last cell of a cluster, the probability that j probes are needed to find an empty cell is $x^{j-1}\cdot(1-x)$. Whenever the number j does not exceed the maximum value r, the number of probes needed to access this key on searching is reduced to one by using the predictor. But if $j>r$, then


the home address of k
1
$1+t_r(x)$
$1+t_r(x)$
reduces to $1+e_r(x)$ on searching
$k_1$
$k_2$
$k_i$
process to trace the cluster
process to find an empty cell

(a) the predictor method


the home address of k
1
1
1
reduces to 1 on searching
$k_1$
$k_2$
$k_i$
process to trace the chain
process to find an empty cell

(b) the chaining method

⊗ : full ◯ : empty

Fig.1. Storing process when the load factor is x.

the number of probes becomes 1+j-r. Therefore, the average probe number is estimated as

$$\sum_{j=0}^{r} x^j \cdot (1-x) + \sum_{j=r+1}^{\infty} (1+j-r) \cdot x^j \cdot (1-x)$$
$$= 1 + \frac{x^r}{1-x} .$$

Let $e_r(x)$ be the excess cost needed to traverse the gap between the two keys $k_i$ and k, as compared with the cost using the chaining method. Then we have:

$$e_r(x) = \frac{x^r}{1-x} . \qquad (2)$$

Next, let $1+t_r(x)$ be the average number of probes needed to traverse between two synonym cells adjoining each other in a cluster. Since the excess cost of traversing between two keys is equal to $e_r(y)$, where y is the load factor when the second key was stored, the average excess cost $t_r(x)$ is given by integrating and averaging $e_r(y)$ as

$$t_r(x) = \frac{1}{x} \int_0^x e_r(y)\,dy$$
$$= -\frac{1}{x} \ln(1-x) - \sum_{i=1}^{r} \frac{x^{i-1}}{i} . \qquad (3)$$

The average traversing cost for keys placed earlier in a cluster is usually less than that for keys placed later; we do not, however, take this into consideration. The average excess cost to trace a cluster is $(i-1) \cdot t_r(x)$, where i is the length of a cluster. Let $s_r(x)$ be the total excess cost to search a key which is stored when the load factor is x. Then, from the results (2) and (3), and by the assumption of Poisson approximation, it follows that

$$s_r(x) = e_r(x) + \sum_{i=1}^{\infty}(i-1)\cdot t_r(x)\cdot P(i,x)$$

$$= -\ln(1-x) - \sum_{i=1}^{r}\frac{x^i}{i} + \frac{x^r}{1-x} - t_r(x)\cdot(1-e^{-x}). \quad (4)$$

Note that we do not consider the effect of key displacement for simplicity. Instead, the excess cost $e_r(x)$ is taken into account even if the key to be stored is the first key of a cluster, to compensate for the primary effect of key displacement. Let $E(p,\alpha)$ denote the average number of probes needed to retrieve a key in the table when the load factor is $\alpha$. Then, from (1) and (4),

$$E(p,\alpha) = 1+\frac{\alpha}{2} + \frac{1}{\alpha}\int_0^{\alpha} s_r(x)\,dx$$

$$= 1+\frac{\alpha}{2} + \frac{1}{\alpha}\int_0^{\alpha} e_r(x)\,dx + \frac{1}{\alpha}\sum_{i=1}^{\infty}(i-1)\int_0^{\alpha}\frac{P(i,x)}{x}\int_0^{x} e_r(y)\,dy\,dx$$

$$= 2+\frac{\alpha}{2} - \ln(1-\alpha) - \sum_{i=1}^{r}\frac{\alpha^{i-1}}{i}\cdot\left(1+\frac{\alpha}{i+1}\right)$$

$$- \frac{1}{\alpha}\int_0^{\alpha} t_r(x)\cdot(1-e^{-x})\,dx, \quad (5)$$

where $r=2^p-1$.

Now we turn to the case of multiple predictors. Before estimating the efficiency of multiple predictors, let us consider an extended chaining method, called the multiple chaining method, which uses more than one link field per cell.

Let N indicate the number of link fields associated with each cell. Each link field is used as a pointer to the next synonym. Consider the case of storing a key into the home cell, into which the storing algorithm has already attempted to store $i(\geq 1)$ keys(i.e. synonyms). Those synonyms, except for the one stored in the home cell, have been scattered again to the N lists by using another hashing function like g used in the multiple predictor method. The average length of each list is $(i-1)/N$. Since a new key is stored after visiting the home address and all elements in a proper list selected by the secondary hashing function, the number of probes needed to retrieve this key later is $1+(i-1)/N+1$ for $i\geq 1$. Assuming that keys are scattered to random locations of the table, the average search length for a key stored when the load factor is x is given as

$$c^N(x)=\sum_{i=1}^{\infty}\left(1+\frac{i-1}{N}+1\right)\cdot P(i,x)+P(0,x)$$

$$=2-\frac{1}{N}+\left(\frac{1}{N}-1\right)\cdot e^{-x}+\frac{x}{N}\ .$$

Therefore, the average number of probes $E^N(\alpha)$ for a successful search is

$$E^N(\alpha)=\frac{1}{\alpha}\int_0^{\alpha} c^N(x)\,dx$$

$$=2-\frac{1}{N}+\frac{1}{\alpha}\left(\frac{1}{N}-1\right)\cdot(1-e^{-\alpha})+\frac{\alpha}{2N}\ , \qquad (6)$$

where $\alpha$ is the load factor. Note that when N=1, $E^N(\alpha)$ is reduced to (1), which is as expected. Note further that

$$\lim_{N\to\infty} E^N(\alpha)=2-\frac{1}{\alpha}(1-e^{-\alpha})\ , \qquad (7)$$

which gives the boundary of improvement by multiple chaining.

The efficiency of the multiple predictor method will be proved by estimating the excess cost over the multiple chaining method, similar to the method used for estimating the performance of the single predictor method in comparison with the direct chaining method. Assume each cell has N associated predictors. The excess cost of finding an empty cell is equal to (2). By using the results of (3) and the discussion of multiple chaining, the averaged extra cost of scanning the final key in a cluster is approximated as $t_r(x)\cdot(i-1)/N$, where x is the load factor. Thus the total excess cost $s_r^N(x)$ to search a key which was stored when the load factor was x is given as

$$s_r^N(x)=e_r(x)+\sum_{i=1}^{\infty}\frac{i-1}{N}\cdot t_r(x)\cdot P(i,x)$$

$$=-\frac{1}{N}\ln(1-x)-\frac{1}{N}\sum_{i=1}^{r}\frac{x^i}{i}+\frac{x^r}{1-x}-\frac{1}{N}t_r(x)\cdot(1-e^{-x}). \qquad (8)$$

Let $E^N(p,\alpha)$ be the average number of probes needed for a successful search when the load factor is $\alpha$. Then, from (6) and (8),

$$E^N(p,\alpha)=E^N(\alpha)+\frac{1}{\alpha}\int_0^{\alpha}s_r^N(x)\,dx$$

$$=2-\frac{1}{N}+\frac{1}{\alpha}\left(\frac{1}{N}-1\right)\cdot(1-e^{-\alpha})+\frac{\alpha}{2N}-\frac{1}{N\alpha}\int_0^{\alpha}\ln(1-x)\,dx$$

$$-\frac{1}{N\alpha}\sum_{i=1}^{r}\frac{1}{i}\int_0^{\alpha}x^i dx+\frac{1}{\alpha}\int_0^{\alpha}\frac{x^r}{1-x}dx-\frac{1}{N\alpha}\int_0^{\alpha}t_r(x)\cdot(1-e^{-x})\,dx$$

$$=2+\frac{1}{\alpha}\left(\frac{1}{N}-1\right)\cdot(1-e^{-\alpha})+\frac{\alpha}{2N}+\frac{1}{\alpha}\left(\frac{1-\alpha}{N}-1\right)\cdot\ln(1-\alpha)$$

$$-\frac{1}{N}\sum_{i=1}^{r}\frac{\alpha^i}{i(i+1)}-\sum_{i=1}^{r}\frac{\alpha^{i-1}}{i}-\frac{1}{N\alpha}\int_0^{\alpha}t_r(x)\cdot(1-e^{-x})\,dx, \qquad (9)$$

where $r=2^p-1$. If we let N approach infinity, $E^N(p,\alpha)$ gives the boundary of improvement by multiple predictors whose size is p, as:

$$\lim_{N\to\infty}E^N(p,\alpha)=2-\frac{1-e^{-\alpha}}{\alpha}-\frac{1}{\alpha}\ln(1-\alpha)-\sum_{i=1}^{r}\frac{\alpha^{i-1}}{i}. \qquad (10)$$

When N=1, $E^N(p,\alpha)$ naturally reduces to (5).

The integral in the last term of (5) or (9) is easily

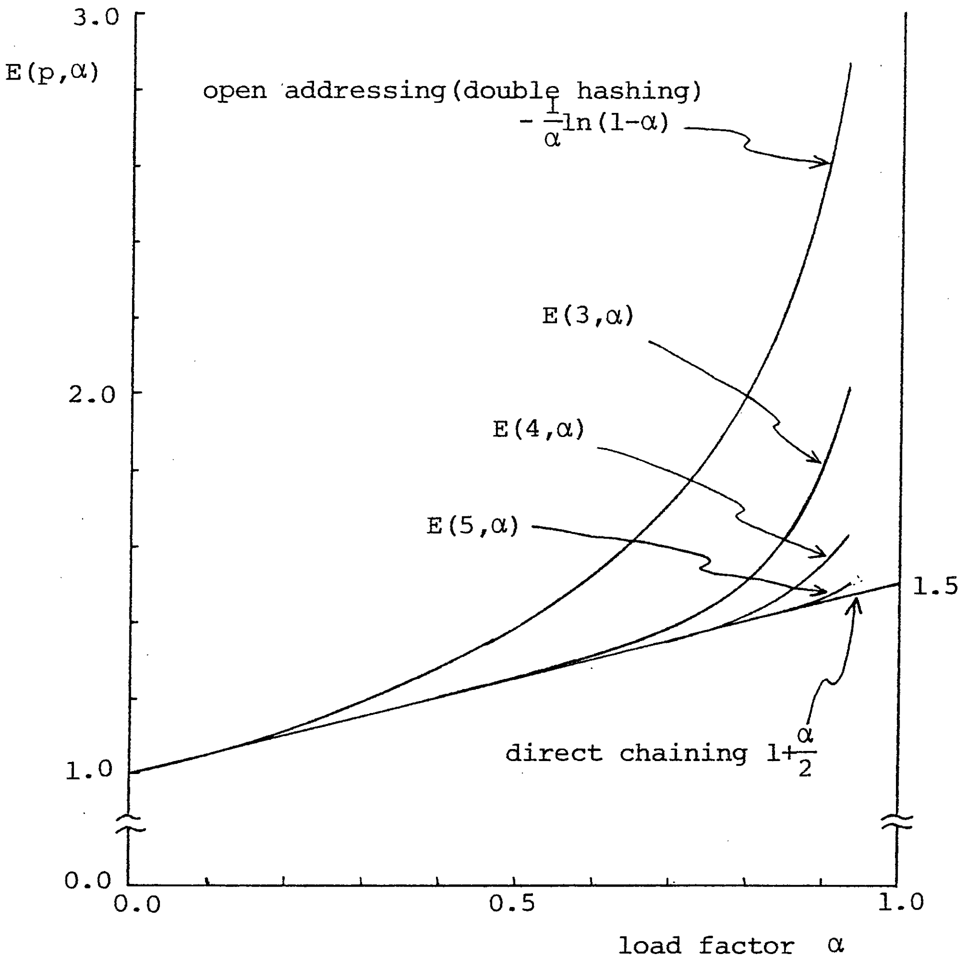


Fig.2 E(p,α) of the single predictor method.

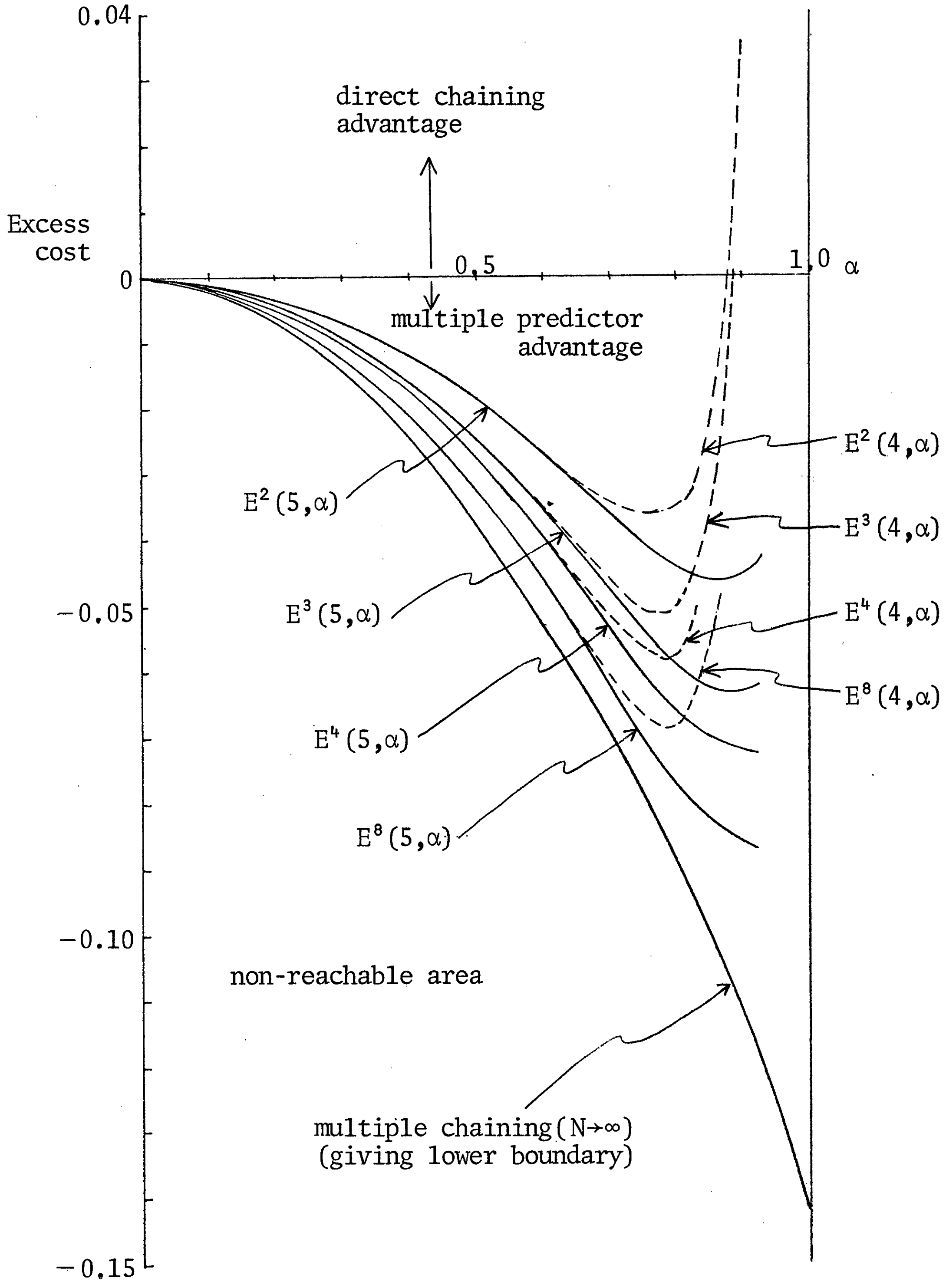


Fig.3 The excess cost of the multiple predictor method over the direct chaining method, i.e. $E^N(p,\alpha)-(1+\alpha/2)$.

evaluated by a standard numerical integration method. Figure 2 and 3 show the estimated average probe numbers for successful searching by the single predictor method and the multiple predictor method, respectively.

## 4. Experimental verification

Applying our methods to the quadratic search method, which is a typical open addressing method eliminating primary clustering, we made the following set of experiments.

Many cases of the size of a predictor field and the number of predictors were tested. Each simulation run was repeated 10 times and averaged for a table of length 2048 using pseudorandom keys. The results obtained for each case are compared with the theoretical values i.e. $E(p,\alpha)$ or $E^N(p,\alpha)$ in Table 1. Estimated efficiencies $E^N(\alpha)$ of the multiple chaining method and the ultimate values when N approaches infinity are also listed in Table 1. It is seen that the experiments give results very close to the expected values.

The greater the bit length p of each predictor field is chosen, the closer the value of $E^N(p,\alpha)$ becomes to that of chaining, i.e. $E^N(\alpha)$. In the chaining method, the length of

Table 1 Summary of results of simulations and theoretical values $E(p,\alpha)$ or $E^N(p,\alpha)$.

| N: no.of preds. | α: load fact. | predictor |  |  |  |  |  | chain |
|---|---|---|---|---|---|---|---|---|
|  |  | p=3 |  | p=4 |  | p=5 |  |  |
|  |  | $E^N(3,\alpha)$ | observed | $E^N(4,\alpha)$ | observed | $E^N(5,\alpha)$ | observed | $E^N(\alpha)$ |
| N=1 single | 0.1 | 1.050 | 1.049 | 1.050 | 1.049 | 1.050 | 1.049 | 1.050 |
|  | 0.2 | 1.100 | 1.099 | 1.100 | 1.099 | 1.100 | 1.099 | 1.100 |
|  | 0.3 | 1.150 | 1.154 | 1.150 | 1.154 | 1.150 | 1.154 | 1.150 |
|  | 0.4 | 1.200 | 1.203 | 1.200 | 1.203 | 1.200 | 1.203 | 1.200 |
|  | 0.5 | 1.252 | 1.253 | 1.250 | 1.252 | 1.250 | 1.252 | 1.250 |
|  | 0.6 | 1.308 | 1.312 | 1.300 | 1.304 | 1.300 | 1.303 | 1.300 |
|  | 0.7 | 1.379 | 1.389 | 1.351 | 1.354 | 1.350 | 1.351 | 1.350 |
|  | 0.8 | 1.498 | 1.521 | 1.409 | 1.412 | 1.400 | 1.398 | 1.400 |
|  | 0.9 | 1.809 | 1.832 | 1.543 | 1.545 | 1.460 | 1.457 | 1.450 |
| N=2 | 0.5 | 1.233 | 1.235 | 1.232 | 1.234 | 1.232 | 1.234 | 1.232 |
|  | 0.6 | 1.282 | 1.282 | 1.274 | 1.275 | 1.274 | 1.275 | 1.274 |
|  | 0.7 | 1.344 | 1.346 | 1.316 | 1.315 | 1.315 | 1.314 | 1.315 |
|  | 0.8 | 1.453 | 1.462 | 1.365 | 1.365 | 1.356 | 1.351 | 1.356 |
|  | 0.9 | 1.750 | 1.785 | 1.487 | 1.504 | 1.405 | 1.409 | 1.395 |
| N=3 | 0.5 | 1.227 | 1.229 | 1.225 | 1.228 | 1.225 | 1.228 | 1.225 |
|  | 0.6 | 1.273 | 1.274 | 1.265 | 1.267 | 1.265 | 1.267 | 1.265 |
|  | 0.7 | 1.332 | 1.329 | 1.305 | 1.303 | 1.304 | 1.303 | 1.304 |
|  | 0.8 | 1.438 | 1.434 | 1.350 | 1.384 | 1.341 | 1.336 | 1.341 |
|  | 0.9 | 1.730 | 1.760 | 1.469 | 1.489 | 1.387 | 1.387 | 1.377 |
| N=4 | 0.5 | 1.224 | 1.226 | 1.222 | 1.225 | 1.222 | 1.225 | 1.222 |
|  | 0.6 | 1.269 | 1.271 | 1.261 | 1.263 | 1.261 | 1.263 | 1.261 |
|  | 0.7 | 1.326 | 1.330 | 1.299 | 1.299 | 1.298 | 1.297 | 1.298 |
|  | 0.8 | 1.431 | 1.446 | 1.343 | 1.345 | 1.334 | 1.331 | 1.334 |
|  | 0.9 | 1.721 | 1.744 | 1.460 | 1.474 | 1.378 | 1.377 | 1.368 |
| N=6 | 0.5 | 1.221 | 1.223 | 1.219 | 1.221 | 1.219 | 1.221 | 1.219 |
|  | 0.6 | 1.264 | 1.266 | 1.257 | 1.259 | 1.257 | 1.259 | 1.257 |
|  | 0.7 | 1.320 | 1.321 | 1.293 | 1.292 | 1.292 | 1.291 | 1.287 |
|  | 0.8 | 1.423 | 1.425 | 1.336 | 1.335 | 1.327 | 1.322 | 1.326 |
|  | 0.9 | 1.711 | 1.745 | 1.450 | 1.477 | 1.369 | 1.375 | 1.359 |
| N=8 | 0.5 | 1.219 | 1.221 | 1.218 | 1.220 | 1.218 | 1.220 | 1.218 |
|  | 0.6 | 1.262 | 1.263 | 1.255 | 1.257 | 1.255 | 1.256 | 1.255 |
|  | 0.7 | 1.318 | 1.319 | 1.290 | 1.290 | 1.289 | 1.289 | 1.289 |
|  | 0.8 | 1.419 | 1.417 | 1.332 | 1.330 | 1.323 | 1.319 | 1.323 |
|  | 0.9 | 1.706 | 1.716 | 1.446 | 1.455 | 1.364 | 1.366 | 1.354 |
| $N\to\infty$ | 0.1 | 1.048 |  | 1.048 |  | 1.048 |  | 1.048 |
|  | 0.2 | 1.094 |  | 1.094 |  | 1.094 |  | 1.094 |
|  | 0.3 | 1.136 |  | 1.136 |  | 1.136 |  | 1.136 |
|  | 0.4 | 1.176 |  | 1.176 |  | 1.176 |  | 1.176 |
|  | 0.5 | 1.215 |  | 1.213 |  | 1.213 |  | 1.213 |
|  | 0.6 | 1.256 |  | 1.248 |  | 1.248 |  | 1.248 |
|  | 0.7 | 1.309 |  | 1.282 |  | 1.281 |  | 1.281 |
|  | 0.8 | 1.408 |  | 1.321 |  | 1.312 |  | 1.312 |
|  | 0.9 | 1.691 |  | 1.432 |  | 1.350 |  | 1.341 |
|  | 1.0 | $\infty$ |  | $\infty$ |  | $\infty$ |  | 1.368 |

Note: $E^1(p,\alpha)\equiv E(p,\alpha)$

a link field must be at least $\log_2 M$ bits, where M is the table size. In general, the size of a cell of the predictor method is less than that of the chaining method. In practical usage, with respect to the space/time trade-offs, the predictor method is always preferable to the other as long as the size of each predictor field is chosen to be more than 4 or 5 bits.

In particular, when the table size is very large and the entire required bit length of the link field is used instead for multiple predictors, the expected number of probes to look up a key of the multiple predictor method becomes less than that of the direct chaining method, i.e. $1+\alpha/2$.

## 5. Conclusion

We have proposed two methods, the single predictor method and the multiple predictor method, which use several bit fields as predictors, to reduce the average number of probes necessary to search a key in a hash table. The efficiency of each method was analyzed theoretically and verified experimentally.

The single predictor method whose predictor size is more than 4 or 5 bits is in practice preferable to the chaining method with respect to space/time trade-offs. Further, when the table size is great, the multiple predictor method gives a smaller average number of probes than that of the chaining method, i.e. $1+\alpha/2$. The multiple predictor method is in a sense an extension of the single predictor method, similar to the double hashing method[1,3] which is an improved open addressing method that eliminates secondary clusterings.

Finally, note that the two proposed methods are also effective to reduce the reject time when the key to be retrieved does not exist in the table.

Acknowledgment: The author would like to thank Professors K. Ikeda, M. Mori and J. W. Higgins for their many suggestions and review of the manuscript.

[1] J.R.Bell, The quadratic quotient method: A hash code eliminating secondary clustering, Comm.ACM,13,2(1970), pp.107-109.

[2] C.Halatsis and G.Philokyprou, Pseudochaining in hash tables, Comm.ACM,21,7(1978), pp.554-557.

[3] D.E.Knuth, The Art of Computer Programming, Vol.3: Sorting and Searching, Addison-Wesley, Reading, Mass.,1973.

[4] R.Morris, Scatter storage techniques, Comm.ACM, 11,1(1968), pp.38-44.

## APPENDIX A: Evaluation of theoretical values $E^N(p,\alpha)$.

```
COMPUTATION OF THEORETICAL VALUES, JUNE 11, 1980.
C   MULTIPLE PREDICTORS METHOD.
C   NK=NO. OF PREDICTORS,
C   IJ=PREDICTOR FIELD SIZE, I.E. IR=MAX OF PRED.,
C   IA= AL= LOAD FACTOR.
      COMMON IR
      COMMON/COMDEF/N,X0,W0,X1(160),X2(160),W(160)
      DOUBLE PRECISION F,EPS,V,A,AL
      EXTERNAL F
      DO 1 NK=1,8
      WRITE(6,1000) NK
 1000 FORMAT(1H ,30X,22H****************** N=,I2,6H *****)
      FN=NK
      DO 10 IJ=2,5
      IR=2**IJ-1
      WRITE(6,1001) IR
 1001 FORMAT(4H IR=,I4)
      A=0.0D0
      EPS=1.0D-10
C
      DO 20 IA=1,9
      AL=0.1D0*DFLOAT(IA)
          CALL FOMULA(A,AL)
          CALL DEFINT(F,EPS,V)
          VW=V/(AL*FN)
C          WRITE(6,1002) V,VW
 1002     FORMAT(1H ,20X,2HV=,D22.15,6H,  VW=,F15.10)
      WRK=0.0
          DO 30 IW=1,IR
          WW=IW
          WRK=WRK+AL**IW*(1.+AL/(FN*(WW+1.)))/(WW*AL)
   30     CONTINUE
      AW=1.0-AL
      BW=2.-1./FN+(1./FN-1.)*(1.-DEXP(-AL))/AL+AL/(2.*FN)
      AV=BW+ALOG(AW)/AL*(AW/FN-1.)+1./FN-WRK+VW
      WRITE(6,1003) AL,AV,AV,BW
 1003 FORMAT(8H      AL=,F4.2,F10.3,F10.5,5X,9H***** BW=,F7.4)
   20 CONTINUE
   10 CONTINUE
    1 CONTINUE
      STOP
      END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE FOMULA(A,B)
C     ** POINTS AND WEIGHTS OF       **
C     ** DOUBLE EXPONENTIAL FORMULA **
      COMMON IR
      COMMON/COMDEF/N,X0,W0,X1(160),X2(160),W(160)
      DOUBLE PRECISION X0,W0,X1,X2,W,A,B
      DOUBLE PRECISION EP,HP,TMAX,H,EHI,ENI,EN,S1,C1,E1,E1I,C2
      DOUBLE PRECISION P,Q
C
      K=6
      EP=1.0D-18
      HP=DATAN(1.0D0)*2.0D0
      TMAX=DLOG(-DLOG(EP)/HP)
      H=1.0D0/DFLOAT(2**(K-1))
      N=TMAX/H
C      WRITE(6,2000) A,B,TMAX,H,N
 2000 FORMAT(27H0DOUBLE EXPONENTIAL FORMULA,
     1 3X,2HA=,D13.5,3X,2HB=,D13.5/
     2 3X,5HTMAX=,D11.3,2X,2HH=,D13.5,2X,2HN=,I3)
      X0=0.0D0
      W0=HP
      EHI=DEXP(-H)
      ENI=0.5D0
      DO 10 I=1,N
      ENI=EHI*ENI
      EN=0.25D0/ENI
      S1=HP*(EN-ENI)
      C1=EN+ENI
      E1=DEXP(S1)
      E1I=1.0D0/E1
      C2=2.0D0/(E1+E1I)
      X2(I)=0.5D0*(E1-E1I)*C2
      X1(I)=-X2(I)
      W(I)=C1*C2*C2*HP
      IF(X2(I).LT.1.0D0) GO TO 10
      N=I-1
      WRITE(6,2001) N
 2001 FORMAT(4H X2(,I2,23H) IS NOT LESS THAN 1.0.)
      GO TO 100
   10 CONTINUE
  100 CONTINUE
C
      IF(A.NE.-1.0D0) GO TO 200
      IF(B.EQ. 1.0D0) GO TO 999
  200 CONTINUE
      P=0.5D0*(B-A)
      Q=0.5D0*(B+A)
      X0=P*X0+Q
      W0=P*W0
      DO 20 J=1,N
      X1(J)=P*X1(J)+Q
      X2(J)=P*X2(J)+Q
      W(J)=P*W(J)
   20 CONTINUE
  999 RETURN
      END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE DEFINT(FUNC,EPS,V)
C     ** INTEGRATION BY           **
C     ** DOUBLE EXPONENTIAL FORMULA **
      COMMON IR
      COMMON/COMDEF/N,X0,W0,X1(160),X2(160),W(160)
      DOUBLE PRECISION X0,W0,X1,X2,W
      DOUBLE PRECISION FUNC,EPS,V
      DOUBLE PRECISION H,U
      K=6
      H=1.0D0
      M=2**(K-1)
      U=W0*FUNC(X0)
      DO 10 I=M,N,M
      U=U+W(I)*(FUNC(X1(I))+FUNC(X2(I)))
   10 CONTINUE
C      WRITE(6,7000)
 7000 FORMAT(13H0** DEFINT **)
      U=U*H
      DO 20 J=2,K
      H=0.5D0*H
      V=0.0D0
      M1=2**(K-J)
      M2=2*M1
      DO 30 I=M1,N,M2
      V=V+W(I)*(FUNC(X1(I))+FUNC(X2(I)))
   30 CONTINUE
      V=0.5D0*U+H*V
C      WRITE(6,7001) J,H,V
 7001 FORMAT(3H J=I1,3H H=F8.5,3H V=D22.15)
      IF(DABS(V-U).LE.EPS) GO TO 999
      U=V
   20 CONTINUE
      WRITE(6,2001)
 2001 FORMAT(16H CONVERGENCE BAD)
  999 RETURN
      END


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      DOUBLE PRECISION FUNCTION F(X)
      COMMON IR
      COMMON/COMDEF/N,X0,W0,X1(160),X2(160),W(160)
      DOUBLE PRECISION X,G
      IF(X .EQ. 0.0D0) GO TO 1
      F=(1.0D0-DEXP(-X))/X*DLOG(1.0D0-X)
      G=0.0D0
      DO 10 I=1,IR
      G=G+X**(I-1)/DFLOAT(I)
   10 CONTINUE
      G=G*(1.0D0-DEXP(-X))
      F=F+G
      RETURN
    1 F=0.0D0
      RETURN
      END
```

APPENDIX B: The simulation program of the single predictor method.

```
C OPEN HASH METHOD USING A PREDICTOR.
C MAY 30, 1980.
      COMMON KY(4096),IPR(4096),IR,JPRB,ITAB
      DIMENSION FAV(9,12)
      ITAB=2048
      DO 1 L=2,5
      IR=2**L-1
      INIT=584287
        DO 10 KURI=1,12
        MEMO=INIT
        WRITE(6,1000) IR,KURI
 1000   FORMAT(10H ***** IR=,I4,3X,5HKURI=,I4,10H *********)
CC  CLEAR  CCCCCCCCCCC
          DO 20 KW=1,ITAB
          KY(KW)=0
          IPR(KW)=0
   20     CONTINUE
CC  STORE AND SEARCH  CCCCCCCCCCCCCCCCC
        INIT=MEMO
        JCNT=1
        DO 30 JJ=1,9
CC  STORE
        JJ1=FLOAT(ITAB)*FLOAT(JJ)/10.0+0.5
        DO 40 JJ2=JCNT,JJ1
        CALL KEY(INIT,NEXT)
        CALL STORE(NEXT)
        INIT=NEXT
   40   CONTINUE
CC  SEARCH
        INIT=MEMO
        JPRB=0
        DO 50 JJ3=1,JJ1
        CALL KEY(INIT,NEXT)
        CALL SEARCH(NEXT)
        INIT=NEXT
   50   CONTINUE
        AV=FLOAT(JPRB)/FLOAT(JJ1)
        WRITE(6,1001) JJ,JJ1,JPRB,AV
 1001   FORMAT(1H ,2I6,I10,F10.3)
        FAV(JJ,KURI)=AV
        JCNT=JJ1+1
   30   CONTINUE
        INIT=NEXT
   10   CONTINUE
CCCC  MATOME  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      WRITE(6,1002) L,IR
 1002 FORMAT(22H AVERAGE*** PREDICTOR=,I1,5H BITS,3X,6HIRMAX=,I3)
      DO 2 LL=1,9
      FMAX=0.0
      FMIN=3000.0
      FTOTAL=0.0
      DO 3 LW=1,12
      FW=FAV(LL,LW)
      FTOTAL=FTOTAL+FW
      IF(FW .GT. FMAX) FMAX=FW
      IF(FW .LT. FMIN) FMIN=FW
    3 CONTINUE
      FTOTAL=(FTOTAL-FMAX-FMIN)/10.0
      WRITE(6,1003) LL,FTOTAL
 1003 FORMAT(1H ,I5,F8.4)
    2 CONTINUE
      WRITE(6,1004)
      WRITE(6,1004)
 1004 FORMAT(40H ***************************************)
    1 CONTINUE
      STOP
      END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE STORE(K)
      COMMON KY(4096),IPR(4096),IR,JPRB,ITAB
      KW=K
      CALL HASH(K,0,IA)
      IF(KY(IA) .EQ. 0) GOTO 3
      K1=KY(IA)
      CALL HASH(K1,0,IA1)
      IF(IA1 .EQ. IA) GOTO 8
      KY(IA)=K
      IPR(IA)=0
      K=K1
      IA=IA1
C
    8 I=0
    9 IP=IPR(IA1)
      IP1=IP
      IF(IP1 .EQ. 0) GOTO 19
   11 IW=I+IP1
      CALL HASH(K,IW,IA2)
      IW=KY(IA2)
      IF(IW .EQ. 0) GOTO 22
      CALL HASH(IW,0,IW2)
      IF(IW2 .EQ. IA) GOTO 15
      IP1=IP1+1
      GOTO 11
C
   15 I=I+IP1
      IF(IR .LT. IP1) IP1=IR
      IF(IP1 .EQ. IP) GOTO 18
      IPR(IA1)=IP1
   18 IA1=IA2
      GO TO 9
```

```
C
   19 IP1=IP1+1
      I=I+1
      CALL HASH(K,I,IA2)
      IW=KY(IA2)
      IF(IW .EQ. 0) GOTO 22
      GO TO 19
C
   22 IF(IR .LT. IP1) IP1=IR
      IF(IP1 .EQ. IP) GOTO 25
      IPR(IA1)=IP1
   25 KY(IA2)=K
      IPR(IA2)=0
      K=KW
      RETURN
C
    3 KY(IA)=K
      IPR(IA)=0
      RETURN
      END
```

```
CCCCCCCCCC
      SUBROUTINE SEARCH(K)
      COMMON KY(4096),IPR(4096),IR,JPRB,ITAB
      CALL HASH(K,0,IA)
      IA1=IA
      I=0
    2 JPRB=JPRB+1
      IF(KY(IA1) .EQ. K) RETURN
      IF(IPR(IA1) .EQ. 0) STOP 9999
      IP=IPR(IA1)
      I=I+IP
      CALL HASH(K,I,IA1)
      IF(IP .LT. IR) GO TO 2
    7 IW=KY(IA1)
      CALL HASH(IW,0,IW1)
      IF(IW1 .EQ. IA) GO TO 2
      JPRB=JPRB+1
      I=I+1
      CALL HASH(K,I,IA1)
      GO TO 7
      END
```

```
CCCCCCCCCCC
      SUBROUTINE HASH(K,IBAN,IX)
      COMMON KY(4096),IPR(4096),IR,JPRB,ITAB
      IW= K/3+K/7+K/11+K/23+K/119
      IW=MOD(IW,ITAB)
      IQ=IW*2+1
      IX=IW+IQ*IBAN*(IBAN+1)/2
      IX=MOD(IX,ITAB)+1
      RETURN
      END
```

```
CCCCCCCCCCCCC
      SUBROUTINE KEY(K1,K2)
      K2=K1*48828125
      IF(K2 .LT. 0) K2=K2+2147483647+1
      RETURN
      END
```

APPENDIX C: The simulation program of the multiple predictor method.

```
C OPEN HASH METHOD USING MULTIPLE-PREDICTORS.
C MAY 30, 1980.
      COMMON KY(2048),IPR(2048,8),IR,JPRB,ITAB
      COMMON NPR,ISMAX
      DIMENSION FAV(9,12)
      ITAB=2048
      DO 4 NPR=2,8
      WRITE(6,1005) NPR
 1005 FORMAT(15H *************,10X,10H***** NPR=,I2,15H *************)
      DO 1 L=2,5
      IR=2**L-1
      INIT=584287
        DO 10 KURI=1,12
        MEMO=INIT
C        WRITE(6,1000) IR,KURI
 1000   FORMAT(10H ***** IR=,I4,3X,5HKURI=,I4,10H *********)
CC  CLEAR  CCCCCCCCCC
          DO 20 KW=1,ITAB
          KY(KW)=0
          DO 21 KW1=1,NPR
          IPR(KW,KW1)=0
   21     CONTINUE
   20     CONTINUE
CC  STORE AND SEARCH  CCCCCCCCCCCCCCCCCC
        INIT=MEMO
        JCNT=1
        DO 30 JJ=1,9
        ISMAX=0
CC  STORE
        JJ1=FLOAT(ITAB)*FLOAT(JJ)/10.0+0.5
        DO 40 JJ2=JCNT,JJ1
        CALL KEY(INIT,NEXT)
        CALL STORE(NEXT)
        INIT=NEXT
   40   CONTINUE
CC  SEARCH
        INIT=MEMO
        JPRB=0
        DO 50 JJ3=1,JJ1
        CALL KEY(INIT,NEXT)
        CALL SEARCH(NEXT)
        INIT=NEXT
   50   CONTINUE
        AV=FLOAT(JPRB)/FLOAT(JJ1)
C        WRITE(6,1001) JJ,JJ1,JPRB,AV,ISMAX
 1001   FORMAT(1H ,2I6,I10,F10.3,5X,13HMAX S-LENGTH=,I5)
        FAV(JJ,KURI)=AV
        JCNT=JJ1+1
   30   CONTINUE
        INIT=NEXT
   10   CONTINUE
CCCC  MATOME  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      WRITE(6,1002) L,IR
 1002 FORMAT(22H AVERAGE*** PREDICTOR=,I1,5H BITS,3X,6HIRMAX=,I3)
      DO 2 LL=1,9
      FMAX=0.0
      FMIN=3000.0
      FTOTAL=0.0
      DO 3 LW=1,12
      FW=FAV(LL,LW)
      FTOTAL=FTOTAL+FW
      IF(FW .GT. FMAX) FMAX=FW
      IF(FW .LT. FMIN) FMIN=FW
    3 CONTINUE
      FTOTAL=(FTOTAL-FMAX-FMIN)/10.0
      WRITE(6,1003) LL,FTOTAL
 1003 FORMAT(1H ,I5,F8.4)
    2 CONTINUE
C      WRITE(6,1004)
 1004 FORMAT(40H ****************************************)
    1 CONTINUE
    4 CONTINUE
      STOP
      END




CCCCCCCCCC
      SUBROUTINE STORE(K)
      COMMON KY(2048),IPR(2048,8),IR,JPRB,ITAB
      COMMON NPR,ISMAX
      KW=K
      CALL HASH(K,0,IA,NPS)
      IF(KY(IA) .EQ. 0) GOTO 3
      K1=KY(IA)
      CALL HASH(K1,0,IA1,NPS1)
      IF(IA1 .EQ. IA) GOTO 8
      KY(IA)=K
      IPR(IA,NPS)=0
      IPR(IA,NPS1)=0
      K=K1
      IA=IA1
      NPS=NPS1
C
    8 I=0
    9 IP=IPR(IA1,NPS)
      IP1=IP
      IF(IP1 .EQ. 0) GOTO 19
   11 IW=I+IP1
      CALL HASH(K,IW,IA2,NPS2)
      IW=KY(IA2)
      IF(IW .EQ. 0) GOTO 22
      CALL HASH(IW,0,IWA,IWB)
      IF(IWA .EQ. IA .AND. IWB .EQ. NPS) GOTO 15
      IP1=IP1+1
      GOTO 11
```

```
C
   15 I=I+IP1
      IF(IR .LT. IP1) IP1=IR
      IF(IP1 .EQ. IP) GOTO 18
      IPR(IA1,NPS)=IP1
   18 IA1=IA2
      GO TO 9
C
   19 IP1=IP1+1
      I=I+1
      CALL HASH(K,I,IA2,NPS2)
      IW=KY(IA2)
      IF(IW .EQ. 0) GOTO 22
      GO TO 19
C
   22 IF(IR .LT. IP1) IP1=IR
      IF(IP1 .EQ. IP) GOTO 25
      IPR(IA1,NPS)=IP1
   25 KY(IA2)=K
      IPR(IA2,NPS)=0
      K=KW
      RETURN
C
    3 KY(IA)=K
      IPR(IA,NPS)=0
      RETURN
      END


CCCCCCCCCC
      SUBROUTINE SEARCH(K)
      COMMON KY(2048),IPR(2048,8),IR,JPRB,ITAB
      COMMON NPR,ISMAX
      CALL HASH(K,0,IA,NPS)
      IA1=IA
      MAXW=0
      I=0
    2 JPRB=JPRB+1
      MAXW=MAXW+1
      IF(KY(IA1) .EQ. K) GOTO 1
      IF(IPR(IA1,NPS) .EQ. 0) STOP 9999
      IP=IPR(IA1,NPS)
      I=I+IP
      CALL HASH(K,I,IA1,NPW)
      IF(IP .LT. IR) GO TO 2
    7 IW=KY(IA1)
      CALL HASH(IW,0,IW1,NPS1)
      IF(IW1 .EQ. IA .AND. NPS1 .EQ. NPS) GOTO 2
      JPRB=JPRB+1
      MAXW=MAXW+1
      I=I+1
      CALL HASH(K,I,IA1,NPW)
      GO TO 7
    1 IF(MAXW .GT. ISMAX) ISMAX=MAXW
      RETURN
      END
```

```
CCCCCCCCCCC
      SUBROUTINE HASH(K,IBAN,IX,NPS)
      COMMON KY(2048),IPR(2048,8),IR,JPRB,ITAB
      COMMON NPR,ISMAX
      IW= K/3+K/7+K/11+K/23+K/119
      NPS=MOD(IW/31+K/13+K/29+K/137,NPR)+1
      IW=MOD(IW,ITAB)
      IF(IBAN .EQ. 0) GO TO 1
      IBAN1=IBAN+ITAB*(NPS-1)/NPR
      IWW=IBAN1*(IBAN1+1)/2
      IWW=MOD(IWW,ITAB)
      IQ=IW*2+1
      IX=IW+IQ*IWW
      IX=MOD(IX,ITAB)+1
      RETURN
    1 IX=IW+1
      RETURN
      END
```

```
CCCCCCCCCCCCC
      SUBROUTINE KEY(K1,K2)
      K2=K1*48828125
      IF(K2 .LT. 0) K2=K2+2147483647+1
      RETURN
      END
```

INSTITUTE OF INFORMATION SCIENCES AND ELECTRONICS
UNIVERSITY OF TSUKUBA
SAKURA-MURA, NIIHARI-GUN, IBARAKI 305 JAPAN

| | |
|---|---|
| REPORT DOCUMENTATION PAGE | REPORT NUMBER<br>ISE-TR-80-18 |

TITLE

# A COLLISION RESOLUTION TECHNIQUE BY USING PREDICTORS

AUTHOR(s)

Seiichi Nishihara

| | |
|---|---|
| REPORT DATE<br>August 15, 1980 | NUMBER OF PAGES<br>34 |
| MAIN CATEGORY<br>Information Retrieval | CR CATEGORIES<br>3.72, 3.74, 4.34 |

KEY WORDS
hashing, scatter storage, open addressing, chaining,
collision, clustering, predictor

ABSTRACT

In hashing techniques, many methods of resolving collisions have been proposed. Those are classified into two main categories, i.e. open addressing and chaining. In this paper, other methods are presented which are intermediate between those two categories. The basic idea of our methods is the use of one or more predictors reserved per cell instead of a link field as in the chaining method. The predictors are used to maintain loose synonym chains. After describing the methods, the efficiencies are estimated theoretically and verified experimentally. In comparison with the chaining method, it is proved that our methods significantly reduce the average number of probes nécessary to retrieve a key without expending extra space.

SUPPLEMENTARY NOTES