ISE-TR-77-7

PERFORMING SET OPERATIONS BY USING HASHING TECHNIQUES

by

Seiichi Nishihara
Hiroshi Hagiwara

September 12, 1977

INSTITUTE
OF
INFORMATION SCIENCES AND ELECTRONICS

UNIVERSITY OF TSUKUBA

ISE-TR-77-7

PERFORMING SET OPERATIONS BY USING HASHING TECHNIQUES

Seiichi Nishihara

Institute of Information
Sciences and Electronics
University of Tsukuba
Niihari-gun, Ibaraki 300-31
Japan

Hiroshi Hagiwara

Department of Information
Science
Kyoto University
Sakyo-ku, Kyoto 606
Japan

September 12, 1977

Abstract

Performing set operations is one of the basic techniques in the fields of information retrieval, data structure and data base management.

In this paper, it is shown that hashing techniques can effectively be applied to performing set operations, where each set is a set of keys. Each entry of a hash table contains a key field, a pointer field and a match level indicator field. The last field is used to indicate how well the key satisfies the set formula under consideration.

Some algorithms to process set formulas containing no complementary set are given and the efficiency is proved by some experiments.

1. Introduction

One of the purposes of recent data management is the centralized control of many files, so that the redundancy and inconsistency in the stored data may be avoided. Further, queries concerning more than one files can be accepted by unifying files. Most of such operations basically contain set operations especially in information retrieval systems. For instance, when two sets of records satisfy different conditions, the intersection of the two sets is the set of records satisfying the both conditions.

In this paper, a method to perform set operations by using hashing techniques is proposed. First a method for set formulas in disjunctive normal form is described, and then the method is extended to general set formulas. Simple experiments are also executed to estimate the efficiency of the method.

2. Performing Set Operations

2.1 Definition of Terms

Before describing the method, we shall introduce the terms necessary for the algorithms. The sets appearing in expression of set operations (shortly set formula) are expressed as S or S_i $(i=1,2,\cdots)$. Each set is a finite set of keys. We assume the operation to get each key in a set one after another without repetition is available. Let card(S) be the cardinal number of set S. Intersection or union of two sets S_i and S_j are written as $S_i \cdot S_j$ or S_i+S_j, respectively. Further, elementary intersection or elementary union is defined as

$$\bigcap_{i=1}^{m} S_i = S_1 \cdot S_2 \cdot \cdots \cdot S_m \quad \text{or} \quad \bigcup_{i=1}^{m} S_i = S_1 + S_2 + \cdots + S_m ,$$

respectively. Then a set formula is called to be in disjunctive normal form if it is a union of elementary intersections, i.e.

$$\bigcup_{i=1}^{m} \bigcap_{j=1}^{n(i)} S_{ij} = S_{11} \cdot S_{12} \cdot \cdots \cdot S_{1n(1)} + \cdots + S_{m1} \cdot S_{m2} \cdot \cdots \cdot S_{m(n)} . \qquad (1)$$

In the formula (1), the first set of each elementary intersection (i.e. $S_{11}, S_{21}, \cdots, S_{m1}$) is called a candidate set. Conversely, the last one (i.e. $S_{1n(1)}, S_{2n(2)}, \cdots, S_{mn(m)}$) is called a determinating set. The keys in the resulting set of a given set formula are called the matched keys.

Up to now, several kinds of hash method have been proposed [1], whose detailed explanation is entirely omitted here. However, we just claim that the hash method adopted in our algorithms works correctly even if a given query key is not in the table. Thus a hash method such as the separate chaining method[1], the conflict flag method[2] or the predictor method [3] is preferable.

Each entry of the hash table contains at least three fields, as is shown in Fig.1. A key is hold in the key field.

match level	key	pointer

Fig.1 Structure of an entry.

The match level field(ML-field) is used to indicate how well the key in the key field agrees with the given set formula. The pointer field, holding a pointer of the chaining method, is of course replaced by the conflict flag or the predictor field in the case other method is adopted.

Now our problem is to get the matched keys of a given set formula by using a hash table.

[Example]

We give here a simple example. Assuming each entry to be initially empty, the algorithm to perform the set formula $S_1 \cdot S_2 \cdot S_3$ is described as follows:

Step 1. Store each element K_1 of set S_1 into the hash table, setting the ML-field to 1.

Step 2. For each element K_2 of set S_2, execute the following operation: Search the element K_2 in the table. If K_2 is found and its ML-field is equal to 1, then change the ML-field to 2.

Step 3. For each element K_3 of set S_3, execute the following operation: Search the element K_3 in the table. If K_3 is found and its ML-field is equal to 2, then change the ML-field to 3.

As the conclusion of the algorithm, the key in the entry whose ML-field is equal to 3 is a matched key of the set formula $S_1 \cdot S_2 \cdot S_3$. In this example the necessary and sufficient length of the ML-field is 2 bits.

2.2 Method for Disjunctive Normal Form

In this section we give a method to process set formulas in disjunctive normal form. The method consists of two phases as follows.

<u>Phase 1</u> (Preprocessing- assigning a match value to each set)

Assign serial numbers to all the sets in the set formula from left to right except the determinating sets. The number assigned to each set is called the <u>match value</u> of the set. Then assign to each determinating set a same value called the <u>final match value</u>, which is the least integer greater than any match value.

For example,

$$\begin{array}{ccccccccccccccccc} S_1 & \cdot & S_2 & \cdot & S_3 & \cdot & S_4 & + & S_5 & \cdot & S_6 & + & S_7 & \cdot & S_8 & \cdot & S_9 \\ 1 & & 2 & & 3 & & 7 & & 4 & & 7 & & 5 & & 6 & & 7 \end{array},$$

where the final match value is 7.

<u>Phase 2</u> (Execution)

Before giving the algorithm of phase 2, we define some wordings used throughout the paper.

First, "storing set S" means "to store each element of set S into the hash table while initializing the ML-field with the match value assigned to the set. But notice that the entry whose ML-field is not equal to the final match value is treated as empty." In this operation, if the key to be stored already exists in the table and its ML-field is equal to the final match value, then there is no need to store the key again.

Next, "filtering x-valued keys according to set S" means the following operation: "For each element key of set S, if the key exists in the hash table and the ML-field is greater than or equal to x and less than the match value (say y) of S, then update the ML-field by y. Otherwise, leave as it is."

Here we give the phase 2 algorithm to process the set formula (1):

Step 1. Set i=1;

Step 2. Store the i-th candidate set S_{i1};

Step 3. Set j=2;

Step 4. Set x equal to the match value of set S_{ij-1};

Step 5. Filter x-valued keys according to set S_{ij};

Step 6. Set j=j+1; Is j>n(i)? If so go to step 7, if not go back to step 4;

Step 7. Set i=i+1; Is i>m? If not go back to step 2, if so we are done.

As the result of the algorithm, the key in the entry whose ML-field is equal to the final match value is a mathced key of (1).

In short, the algorithm first stores the keys belonging to a candidate set as candidates of matched keys (step 2), and then reduces them gradually by checking with the sets following after the candidate set (step 5).

The irreducible minimum size of the hash table does not exceed card($\bigcup_{i=1}^{m} S_{i1}$).

Under the situation that the table size is fixed, the several ways to reduce the processing time are considered as,

i) When a set is stored in step 2, choose a set whose cardinality is as small as possible. In other words, place the smallest set at the first position of each elementary intersection in formula (1).

ii) Arrange the sets in each elementary intersection in set formula (1) in such a way that the number of remaining keys which passed the filtering process of step 5 is reduced as fast as possible.

iii) Arrange the elementary intersections of formula (1) in an ascending order of the size card($\bigcap_{j=1}^{n(i)} S_{ij}$), ($1 \le i \le n$).

Ingeneral, requirement ii) and iii) are hard to insight in advance. On the other hand, requirement i) is relatively easy to satisfy by modifying the algorithm. Further, the effect of requirement i) is greater than that of the rest, as is proved by experiments in the following section.

3. Some Experiments

3.1 Simulations of a Simple Intersection Operation

In the experiment, a basic set operation to get the intersection of three sets S_A, S_B and S_C is simulated and evaluated by employing the separate chaining method with overflow area [1]. The table size is 2000. Varying not only the cardinality of each set (, which influences the load factor[1]) but also set formula (, which influences the filtering sequence of sets), six cases(case1.1 – case2.3) shown in Table 1 are executed. Computer generated pseudorandom numbers are used as keys. Before using them, we made χ^2-test for Poisson distribution at the 5% significance level.

Table 1 The cases executed by simulations.

cardinal number of each set	set formula	case no.
card(S_A)=1000, card(S_B)=500, card(S_C)=200,	$S_A \cdot S_B \cdot S_C$	case 1.1
card($S_A \cdot S_B$)=200, card($S_B \cdot S_C$)=100,	$S_C \cdot S_B \cdot S_A$	case 1.2
card($S_C \cdot S_A$)=40, card($S_A \cdot S_B \cdot S_C$)=30	$S_C \cdot S_A \cdot S_B$	case 1.3
card(S_A)=1800, card(S_B)=900, card(S_C)=360,	$S_A \cdot S_B \cdot S_C$	case 2.1
card($S_A \cdot S_B$)=360, card($S_B \cdot S_C$)=180,	$S_C \cdot S_B \cdot S_A$	case 2.2
card($S_C \cdot S_A$)=72, card($S_A \cdot S_B \cdot S_C$)=54	$S_C \cdot S_A \cdot S_B$	case 2.3

The efficiency of the algorithm may be expressed in terms of the average number of table access operations (i.e. probes) that occur in hashing processes included in step 2 and step 5. Simulations were programmed and run ten times for each case. The results of the simulations are listed in Table 2, where 'average' columns indicate the values averaged by dividing by the total number of keys, i.e. $card(S_A)+card(S_B)+card(S_C)$.

3.2 Analysis of the Experiments

It is easy to estimate analytically the average number of probes needed to get the intersection of three sets. Here we estimate the number of probes needed to process the set formula $S_1 \cdot S_2 \cdot S_3$. Assume that the adopted hash method is the separate chaining method, and each entry in the table is hit as frequently as any other. Then, using Poisson approximation, we can expect that the probability $P(i,x)$ of a cluster of length i is $e^{-x} \cdot x^i / i!$, where x is the load factor[3].

Let M be the table size, and let

$$\left.\begin{array}{l} card(S_1)=k_1, \quad card(S_2)=k_2, \quad card(S_3)=k_3, \\ card(S_1 \cdot S_2)=k_2', \quad card(S_1 \cdot S_3)=k_3', \\ card(S_1 \cdot S_2 \cdot S_3)=k, \end{array}\right\} \qquad (2)$$

see Fig.2, where k is the number of matched keys. Let $\alpha=k_1/M$.

First estimate the number of probes to store set S_1. For an empty entry, probing occurs two times, i.e. to check and to store. Similarly for a chain of length ℓ, probing occurs $\ell+2$ times where ℓ indicates the number of probings to trace the chain. As described above, the probability of a

Table 2 Summary of results of simulations and analysis.

	observed value		theoretical value	
	total	average	total	average
case 1.1	3331	1.96	3289	1.94
case 1.2	2086	1.23	2054	1.21
case 1.3	2026	1.19	1994	1.17
case 2.1	6612	2.16	6532	2.14
case 2.2	3807	1.24	3746	1.22
case 2.3	3699	1.21	3638	1.19

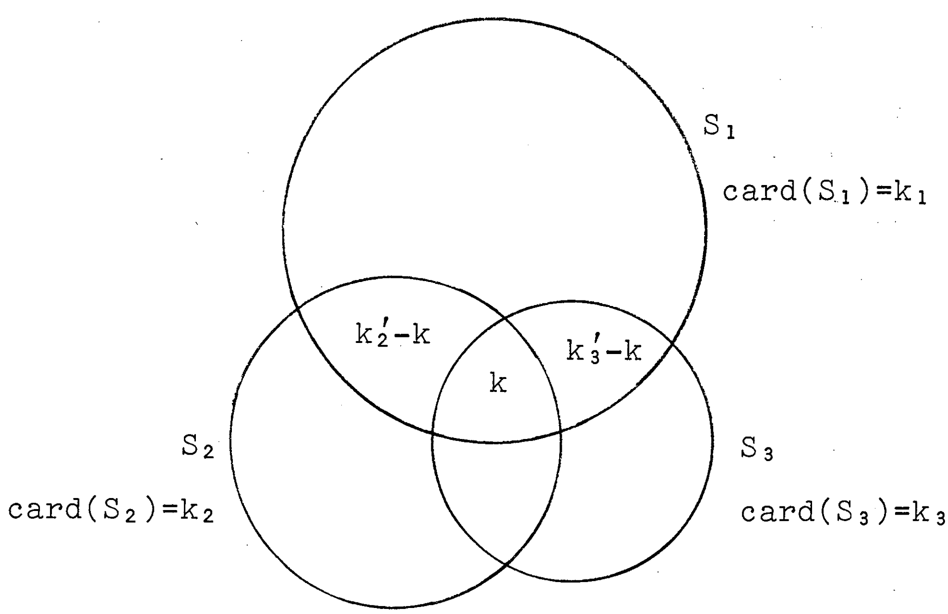

Fig.2 Venn diagram of $S_1 \cdot S_2 \cdot S_3$.

chain of length ℓ is $P(\ell,x)$, where x is the load factor. Thus the average number of probes needed to store a key when the load factor is x is given as

$$2 \cdot P(0,x) + \sum_{\ell=1}^{\infty}(\ell+2) \cdot P(\ell,x) = 2+x.$$

Let T_1 be the average number of probes needed to store each key of set S_1. Then, by integrating and averaging:

$$T_1 = \frac{1}{\alpha}\int_0^{\alpha}(2+x)dx = 2+\frac{\alpha}{2} \quad , \qquad (3)$$

where α is the load factor after storing process of set S_1.

Next consider the keys belonging to set S_2. For each key belonging to the intersection of S_2 and S_1, the average number of probes to search is $1+\alpha/2$, and further one more probing occurs to update the ML-field. Thus average number of probes is as

$$T_2 = 1+\alpha/2+1 = 2+\alpha/2 \ . \qquad (4)$$

On the other hand, the average number of probes T_3 for the keys belonging to S_2-S_1 is equal to the average number for the reject operation (i.e. unsuccessful search)[2]:

$$T_3 = P(0,\alpha)+ \sum_{\ell=1}^{\infty}\ell \cdot P(\ell,\alpha) = e^{-\alpha}+\alpha \ . \qquad (5)$$

Finally, consider the keys belonging to set S_3. For the keys in S_3 and in $S_1 \cdot S_2$ (i.e. matched keys), the average number of probes is equal to T_2. For the keys in S_1-S_2, however, the update operation is not necessary. Thus, average number of probes is as

$$T_4 = 1+\alpha/2 \ , \qquad (6)$$

which is given in [3]. For the keys in S_3-S_1, the average number of probes is equal to T_3.

From the definition (2) and the results (3), (4), (5) and (6), the total number of probing operations T is given as follows:

$$T = T_1 \cdot k_1 + T_2 \cdot (k_2' + k) + T_3 \cdot (k_2 - k_2' + k_3 - k_3') + T_4 \cdot (k_3' - k)$$
$$= k_1 \cdot (2 + \alpha/2) + (k_2 + k_3) \cdot (\alpha + e^{-\alpha})$$
$$+ (k_2' + k_3') \cdot (2 - \alpha/2 - e^{-\alpha}) + k - k_3' \; . \qquad (7)$$

Then the average number of probes E for each key is given as

$$E = T/(k_1 + k_2 + k_3) \; . \qquad (8)$$

The results of theoretical evaluation (7) and (8) are presented in Table 2.

Comparing case 1.1 or case 2.1 with case 1.2 or case 2.2 respectively, the effect of requirement i) is proved. The difference between case 1.2 and 1.3 or between case 2.2 and 2.3 indicates the effect of requirement ii).

4. Extending to General Set Formula

4.1 Necessity of Extension

Every set formula can be rewritten in an equivalent disjunctive normal form. Thus the algorithm given in section 2 is theoretically applicable to any set formula. Consider, however, an example set formula $S_1 \cdot (S_2 + S_3)$, which may be transformed to $S_1 \cdot S_2 + S_1 \cdot S_3$. Then the processing speed will be considerably slowed down, since set S_1 should be stored twice. Therefore, it is desirable that there is an algorithm to execute any set formula in the form as it is, which we call direct execution.

In the following section, we give a direct execution algorithm for general set formula containing no complementary set. The fundamental idea is similar to that of section 2.

Here we extend and redefine the term determinating set. When a given set formula contains parenthesized subformulas, assume each of them to be a single set. Then the original set formula can be regarded as a disjunctive normal form. Therefore, the determinating sets are determined by using the definition given in section 2.1. If the determinating set is a parenthesized subformula, then apply the above rule again recursively.

Similarly the term candidate set can also be extended and redefined, but the manner is omitted here.

For example, consider the set formula:

$$(S_1+S_2\cdot S_3)\cdot(S_4+S_5\cdot(S_6+S_7))+S_8 \ , \qquad (9)$$

where the determinating sets are S_4, S_6, S_7 and S_8, and the candidate sets are S_1, S_2 and S_8. Especially paying attention to subformula $(S_1+S_2\cdot S_3)$, the determinating sets are S_1 and S_3, and the candidate sets are S_1 and S_2.

4.2 Preprocessing of Set Formula (Phase 1)

The rule for assigning a match value to each set is similar to that given in section 2. Roughly speaking, assign serial number from left to right under the restriction that the determinating sets in each parenthesized subformula should be assigned the same value.

For example, the match values assigned to set formula (9) are as:

$$\begin{array}{cccccccc} (S_1+ & S_2\cdot & S_3)\cdot(& S_4+ & S_5\cdot(& S_6+ & S_7))+ & S_8 \\ 2 & 1 & 2 & 4 & 3 & 4 & 4 & 4 \end{array} \qquad (9')$$

where the final match value is 4.

In section 2, the match value of S_{ij-1} ($1\le i\le m$, $2\le j\le n(i)$) is used to filter candidate keys according to S_{ij} in step 5. In the case of general set formula, however, this does not hold. Therefore newly a value, called <u>check value</u>, is introduced, which is assigned to each set so that the filtering process may work correctly. The basic rule of assigning check values is as follows: with respect to each intersection operator (i.e. '•'), the final match value of the left-hand subformula of the operator becomes the check value of the candidate sets of the right-hand subformula. The set that cannot be assigned a check value by the basic rule must be a candidate set of the original set formula and is assigned zero.

For example, the match values and the check values of the set formula (9) are as:

$$\left.\begin{array}{lcccccccc} & (S_1+ & S_2\cdot & S_3)\cdot(& S_4+ & S_5\cdot(& S_6+ & S_7))+ & S_8 \\ \text{match value} & 2 & 1 & 2 & 4 & 3 & 4 & 4 & 4 \\ \text{check value} & 0 & 0 & 1 & 2 & 2 & 3 & 3 & 0 \end{array}\right\} \quad (9'')$$

In conclusion, what phase 1 should do is to assign a check value and a match value to each set of the given set formula. A concrete algorithm of phase 1 is presented in Appendix.

4.3 Execution by Using a Hash Table (Phase 2)

After the completion of phase 1, the main execution process performed on a hash table is started. Let S_i be the i-th set from left in the set formula and let check(S_i) be the check value assigned to set S_i. Let m be the number of sets appears in the set formula. Then the algorithm of phase 2 takes a simple form as follows:

[Algorithm of Phase 2]

Step 1. Set i=1;

Step 2. Set x=check(S_i);

Step 3. If $x \neq 0$, then go to step 4. Otherwise, store S_i and go to step 5;

Step 4. Filter x-valued keys according to set S_i;

Step 5. Set i=i+1; If $i \leq m$, then go back to step 2. Otherwise, we are done.

As the result of the algorithm, the key in the entry whose ML-field is equal to the final match value is a matched key of the given set formula.

Now let k be the number of intersection operator appearing in a set formula. Then, notice that the final match value is equal to k+1. Thus $\lceil \log_2(k+1) \rceil$ bits are needed for the ML-field to process the set formula.

5. Conclusion

We have proposed methods to perform set operations by using a hash table. Two algorithms for disjunctive normal form and general set formulas are presented.

In this note, the influence of complementary set to the algorithm has not been considered at all, which is the future problem.

References

1) Knuth,D.E. The Art of Computer Programming, Vol.3: Sorting and Searching, Addison-Wesley(1973).
2) Furukawa,K. Hash addressing with conflict flag, Information Processing in Japan, Vol.13(1973),pp.13-18.
3) Nishihara,S. & Hagiwara,H. An open hash method using predictors, ibid.,Vol.15(1975),pp.6-10.

APPENDIX. An Algorithm of Phase 1

A stack is used as the work area. Fig.A shows the structure of each entry of the stack, where the fields of set id., match and check are used to hold a set identifier, a match or final match value and a check value, respectively. The handling of perentheses is performed by using delimiter fields.

Let p indicate the position in the set formula where the process is in progress, and let a indicate the address of the stack. The position of the first ∇ is 0. The initial values of p, a and v are 0, 1 and 1, respectively.

The algorithm of phase 1 is shown in Table A. In the algorithm, if the symbols placed at the p-th and (p+1)-th positions agree with the symbols in the columns of 'present' and 'next' of Table A, then the corresponding operations in 'operation' column are applied.

As an example, the results of the processing of set formula (9) is shown in Fig.B, which coincide with (9'').

address	set id.	match	check	delimiter

Fig.A Structure of an entry of the stack.

Table A An algorithm of Phase 1.

next	present	operation
(	free	delimiter(a):=delimiter(a)+1; p:=p+1;
SET	free	setid(a):=SET; a:=a+1; p:=p+1;
•	SET	match(a-1):=v; check(a):=v; v:=v+1; p:=p+1;

	)	w:=a;
		L1:w:=w-1;
		if match(w)=0 then match(w):=v;
		if delimiter(w)=0 then go to L1;
		delimiter(w):=delimiter(w)-1;
		check(a):=match(a-1);
		v:=v+1; p:=p+1;

+	SET	w:=a;
		L2:w:=w-1;
		LL:if w=1 then L3:begin
		check(a):=check(w);
		p:=p+1;
		end
		else
		if delimiter(w)=0
		then go to L2 else go to L3;

	)	w:=a;
		L4:w:=w-1;
		if delimiter(w)=0 then go to L4;
		delimiter(w):=delimiter(w)-1;
		go to LL;

(continued)

	SET	p:=p+1;

)	)	w:=a;
		L5:w:=w-1;
		if delimiter(w)=0 then go to L5;
		delimiter(w):=delimiter(w)-1;
		p:=p+1;

∇	free	w:=a;
		L6:w:=w-1;
		if match(w)≠0 then go to L7;
		match(w):=v;
		L7:if w=1 then go to END
		else go to L6;

$$(S_1+S_2\cdot S_3)\cdot(S_4+S_5\cdot(S_6+S_7))+S_8$$

⇩

p=	0	1	2	3	4	5	6	·	·	·	·						·	·	·	·	20	21	22
	∇	(	S_1	+	S_2	·	S_3	)	·	(	S_4	+	S_5	·	(	S_6	+	S_7	)	)	+	S_8	∇

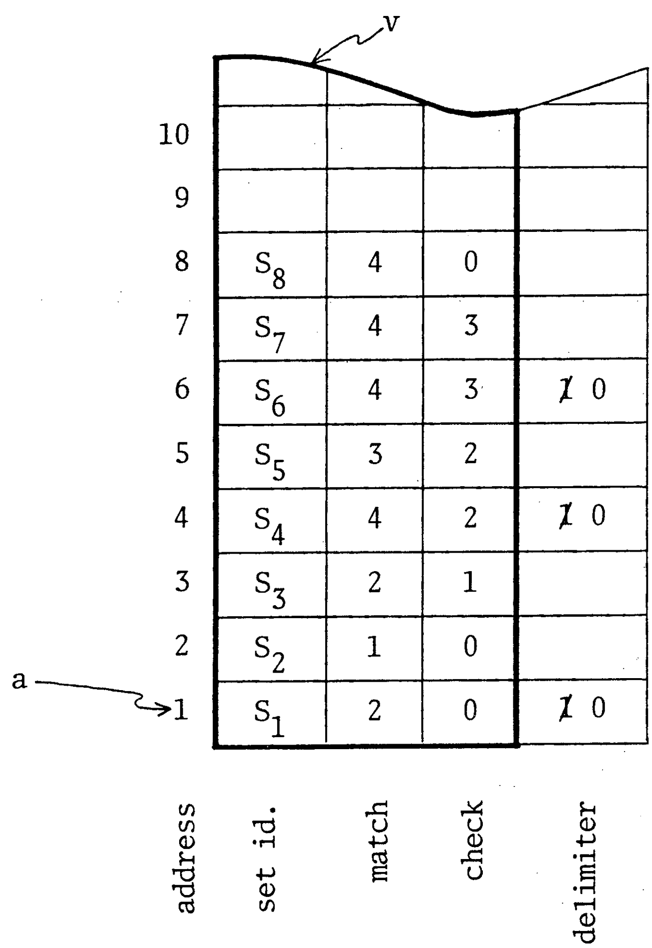

Fig.B An example of preprocessing (Phase 1).

The simulation program to estimate the efficiency.

```
C EXECUTING SET FUNCTIONS BY USING HASHING TECHNIQUES
C FEBRUARY 1976     BY   S. NISHIHARA
      COMMON ITAB(3,2000),IOVF(3,1000),IS1(1800),IS2(900),IS3(360)
      DIMENSION ICOUNT(21)
C PHASE 0 ************************************************************
C INPUT PARAMETERS, CARDINAL NUMBER OF EACH SET
C AND INCREMENT SIZES.
      READ(5,1000) N1,N2,N3,N12,N23,N31,N123
      READ(5,1000) INC1,INC2,INC3
 1000 FORMAT(7I5)
C PHASE 1 ************************************************************
C GENERATE RANDOM NUMBERS USED AS KEYS.
      IPOSSN=0
      IY=1471541918
      KURI=1
  602 CONTINUE
      DO 100 IW=1,N1
      CALL RANDOM2(YFL,IY)
      IS1(IW)=IY
  100 CONTINUE
      I1=N12+N123
      DO 101 IW=1,I1
      IS2(IW)=IS1(IW)
  101 CONTINUE
      I2=I1+1
      DO 102 IW=I2,N2
      CALL RANDOM2(YFL,IY)
      IS2(IW)=IY
  102 CONTINUE
      DO 103 IW=1,N123
      IS3(IW)=IS2(IW)
  103 CONTINUE
      DO 104 IW=1,N23
      I3=N123+IW
      I4=I1+IW
      IS3(I3)=IS2(I4)
  104 CONTINUE
      I5=N123+N23
      DO 105 IW=1,N31
      I6=I5+IW
      I7=I1+IW
      IS3(I6)=IS1(I7)
  105 CONTINUE
      I8=I5+N31+1
      DO 106 IW=I8,N3
      CALL RANDOM2(YFL,IY)
      IS3(IW)=IY
  106 CONTINUE
CC
C PHASE 2 ************************************************************
C CALCULATE THE STARTING ADDRESS OF EACH SET IS1, IS2 AND IS3.
      CALL RANDOM2(YFL,IY)
      IP1=IY-(IY/N1)*N1
      CALL RANDOM2(YFL,IY)
      IP2=IY-(IY/N2)*N2
```

```
  550 CALL RANDOM2(YFL,IY)
      IP3=IY-(IY/N3)*N3
      WRITE(6,1010) IY
 1010 FORMAT(1H ,25HCURRENT RANDOM NUMBER****,I15)
C PHASE 3 ***********************************************************
C STORE ALL ELEMENTS IN SET IS1, AND COUNT
C THE COLLISIONS FOR X**2 TEST.
CHAINING METHOD
      CALL CLEAR(IPOVF)
      IPROB=0
      DO 200 I=1,N1
      KP=IP1+1
      KEY=IS1(KP)
C STORE THE KEY
      I1=KEY/3
      IAD=I1-(I1/2000)*2000+1
C** PROBING **  ACCESS THE FIRST KEY
      IPROB=IPROB+1
      IF(ITAB(3,IAD) .NE. 0) GO TO 201
C** PROBING **  STORE
      IPROB=IPROB+1
      ITAB(1,IAD)=1
      ITAB(3,IAD)=KEY
      GO TO 202
C
  201 IF(ITAB(2,IAD) .NE. 0) GO TO 203
C** PROBING **  POINTER SET
      IPROB=IPROB+1
      ITAB(2,IAD)=IPOVF
C** PROBING **  STORE
      IPROB=IPROB+1
      IOVF(1,IPOVF)=1
      IOVF(3,IPOVF)=KEY
      GO TO 204
C
  203 I2=ITAB(2,IAD)
C** PROBING **  ACCESS NEXT KEY
  206 IPROB=IPROB+1
      IF(IOVF(2,I2) .EQ. 0) GO TO 205
      I2=IOVF(2,I2)
      GO TO 206
C
C** PROBING **  POINTER SET
  205 IPROB=IPROB+1
      IOVF(2,I2)=IPOVF
C** PROBING **  STORE
      IPROB=IPROB+1
      IOVF(1,IPOVF)=1
      IOVF(3,IPOVF)=KEY
C
  204 IPOVF=IPOVF+1
      IF(IPOVF .GT. 1000) STOP 9999
C
  202 IP1=IP1+INC1
      IF(IP1 .GE. N1) IP1=IP1-N1
```

```
  200 CONTINUE
      WRITE(6,1001) IPROB
 1001 FORMAT(1H ,//35H**NUMBER OF PROBES TO STORE SET S1=,I8)
C
C PHASE 7 ***********************************************************
      DO 500 I=1,21
      ICOUNT(I)=0
  500 CONTINUE
C
      DO 501 I=1,2000
      LEN=1
      IF(ITAB(3,I) .EQ. 0) GO TO 502
      LEN=LEN+1
      IF(ITAB(2,I) .EQ. 0) GO TO 502
      LEN=LEN+1
      J=ITAB(2,I)
  503 IF(IOVF(2,J) .EQ. 0) GO TO 502
      LEN=LEN+1
      J=IOVF(2,J)
      GO TO 503
C
  502 IF(LEN .GT. 21) LEN=21
      ICOUNT(LEN)=ICOUNT(LEN)+1
  501 CONTINUE
C
      XX=N1
      XX=XX/2000.0
      WRITE(6,1500) XX,N1
 1500 FORMAT(1H ,12H   X**2-TEST,5X,12HLOAD FACTOR=,
     1F6.3,5X,3HN1=,I6)
      WRITE(6,1501) ICOUNT(1)
      WRITE(6,1502) (ICOUNT(I),I=2,11)
      WRITE(6,1502) (ICOUNT(I),I=12,21)
 1501 FORMAT(1H ,5X,I7,7H(BLANK))
 1502 FORMAT(1H ,5X,10I7)
C*******************************************************************
C PHASE 4 ***********************************************************
  600 CONTINUE
      DO 300 I=1,N2
      KP=IP2+1
      KEY=IS2(KP)
C SEARCH THE KEY
      I1=KEY/3
      IAD=I1-(I1/2000)*2000+1
C** PROBING **  ACCESS THE FIRST KEY
      IPROB=IPROB+1
      IF(ITAB(3,IAD) .EQ. KEY) GO TO 301
      IF(ITAB(2,IAD) .EQ. 0) GO TO 302
      IAD=ITAB(2,IAD)
C** PROBING **  ACCESS NEXT KEY
  304 IPROB=IPROB+1
      IF(IOVF(3,IAD) .EQ. KEY) GO TO 303
      IF(IOVF(2,IAD) .EQ. 0) GO TO 302
      IAD=IOVF(2,IAD)
      GO TO 304
```

```
C
C** PROBING **  SET FLAG 2
  303 IPROB=IPROB+1
      IOVF(1,IAD)=2
      GO TO 302
C
C** PROBING **  SET FLAG 2
  301 IPROB=IPROB+1
      ITAB(1,IAD)=2
C
  302 IP2=IP2+INC2
      IF(IP2 .GE. N2) IP2=IP2-N2
  300 CONTINUE
C SET IS2 PROCESSING  COMPLETED
C
C PHASE 5 ************************************************************
      KOSU=0
      DO 400 I=1,N3
      KP=IP3+1
      KEY=IS3(KP)
C SEARCH THE KEY
      I1=KEY/3
      IAD=I1-(I1/2000)*2000+1
C** PROBING **  ACCESS THE FIRST KEY
      IPROB=IPROB+1
      IF(ITAB(3,IAD) .EQ. KEY) GO TO 401
      IF(ITAB(2,IAD) .EQ. 0) GO TO 402
      IAD=ITAB(2,IAD)
C** PROBING **  ACCESS NEXT KEY
  404 IPROB=IPROB+1
      IF(IOVF(3,IAD) .EQ. KEY) GO TO 403
      IF(IOVF(2,IAD) .EQ. 0) GO TO 402
      IAD=IOVF(2,IAD)
      GO TO 404
C
  403 IF(IOVF(1,IAD) .NE. 2) GO TO 402
C** PROBING **  UPDATE FLAG TO 3
      IPROB=IPROB+1
      IOVF(1,IAD)=3
      KOSU=KOSU+1
      GO TO 402
C
  401 IF(ITAB(1,IAD) .NE. 2) GO TO 402
C** PROBING **  UPDATE FLAG TO 3
      IPROB=IPROB+1
      ITAB(1,IAD)=3
      KOSU=KOSU+1
C
  402 IP3=IP3+INC3
      IF(IP3 .GE. N3) IP3=IP3-N3
  400 CONTINUE
C SET IS3 PROCESSING  COMPLETED
C PHASE 6 ************************************************************
      WRITE(6,1002) IPROB,KOSU
 1002 FORMAT(1H ,//24H TOTAL NUMBER OF PROBES=,
```

```
     1I8,13H      *RESULT=,I8)
 601 WRITE(6,1003) KURI
1003 FORMAT(1H ,10H**********,I2,10H**********,//)
     IF(IPOSSN .EQ. 0) KURI=KURI+1
     IF(KURI .LT. 31) GO TO 602
     STOP
     END
```

```
      SUBROUTINE CLEAR(IPOVF)
      COMMON ITAB(3,2000),IOVF(3,1000)
      DO 10 I=1,2000
      ITAB(1,I)=0
      ITAB(2,I)=0
      ITAB(3,I)=0
   10 CONTINUE
      DO 11 I=1,1000
      IOVF(1,I)=0
      IOVF(2,I)=0
      IOVF(3,I)=0
   11 CONTINUE
      IPOVF=0
      RETURN
      END
```

INSTETUTE OF INFORMATION SCIENCES AND ELECTRONICS
UNIVERSITY OF TSUKUBA
SAKURA-MURA, NIIHARI-GUN, IBARAKI 300-31 JAPAN

REPORT DOCUMENTATION PAGE	REPORT NUMBER ISE-TR-77-7

TITLE

PERFORMING SET OPERATIONS BY USING HASHING TECHNIQUES

AUTHOR(s)

Seiichi Nishihara (Institute of Information Sciences and Electronics, University of Tsukuba)

Hiroshi Hagiwara (Department of Information Science, Kyoto University)

REPORT DATE	NUMBER OF PAGES
September 12, 1977	24
MAIN CATEGORY	CR CATEGORIES
Data Management	4.33, 4.34, 3.73

KEY WORDS

set processing, hashing, scatter storage, database, data manipulation, information retrieval

ABSTRACT

Performing set operations is one of the basic techniques in the fields of information retrieval, data structure and data base management.

In this paper, it is shown that hashing techniques can effectively be applied to performing set operations, where each set is a set of keys. Each entry of a hash table contains a key field, a pointer field and a match level field. The last field is used to indicate how well the key satisfies the set formula under consideration.

Some algorithms to process set formulas containing no complementary set are given and the efficiency is proved by some experiments.

SUPPLEMENTARY NOTES