Performing Set Operations by Using Hashing Techniques

Seiichi NISHIHARA* and Hiroshi HAGIWARA**

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 file can be accepted by unifying files. Most of these 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 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 that the operation to get each key in a set one after another without repetition is available. Let card(S) be the cardinality 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

---

This paper first appeared in Japanese in Joho－Shori (Journal of the Information Processing Society of Japan), Vol. 18, No. 1 (1977), pp. 11～18.

* Institute of Information Sciences, University of Tsukuba

** Department of Information Science, Kyoto University

a union of elementary intersections, i.e.

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

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 methods have been proposed[1], whose detailed explanation is entirely omitted here. However, we 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. 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 the chaining method is not adopted. Now our problem is to get the matched keys of a given set formula by using a hash table.

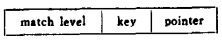


Fig. 1 Structure of an entry.

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.

Phase 1 (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 *match value* of the set. Then assign to each determinating set, a same value called the *final match value*, which is the least integer greater than any match value.

For example,

$$\underset{1}{S_1}\cdot\underset{2}{S_2}\cdot\underset{3}{S_3}\cdot\underset{7}{S_4}+\underset{4}{S_5}\cdot\underset{7}{S_6}+\underset{5}{S_7}\cdot\underset{6}{S_8}\cdot\underset{7}{S_9} \quad ,$$

where the final match value is 7.

Phase 2 (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 it 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 a result of the algorithm, the key in the entry whose ML-field is equal to the final match value, is a matched 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 $\mathrm{card}(\bigcup_{i=1}^{m} S_{i1})$.

In the situation that the table size is fixed, several ways to reduce the processing time are considered, e.g.:

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 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 $\mathrm{card}(\bigcap_{j=1}^{n(i)} S_{ij})$, $(1 \leq i \leq n)$.

In general, 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

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) but also set formula(, which influences the filtering sequence of sets), six cases(casel.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 function	case no.
card (S_A)=1000, card (S_B)=500, card (S_C)=200, card (S_A·S_B)=200, card (S_B·S_C)=100, card (S_C·S_A)=40, card (S_A·S_B·S_C)=30	S_A·S_B·S_C	case 1.1
	S_C·S_B·S_A	case 1.2
	S_C·S_A·S_B	case 1.3
card (S_A)=1800, card (S_B)=900, card (S_C)=360, card (S_A·S_B)=360, card (S_B·S_C)=180, card (S_C·S_A)=72, card (S_A·S_B·S_C)=54	S_A·S_B·S_C	case 2.1
	S_C·S_B·S_A	case 2.2
	S_C·S_A·S_B	case 2.3

Table 2 Summary of results of simulations and theoretical values.

	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

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. $card(S_A)+card(S_B)+card(S_C)$.

It is easy to estimate analytically the average number of probes needed to get the intersection of three sets. Here the calculation process is omitted. But the results of theoretical evaluation 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 Extention

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:

$$(S_1+S_2 \cdot S_3) \cdot (S_4+S_5 \cdot (S_6+S_7))+S_8 \quad ; \qquad (2)$$

where the determinating sets are S_4, S_6, S_7 and S_8, and the candidate sets are S_1, S_2 ans 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 numbers from left to right with 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 (2) are as follows:

$$\begin{matrix}(S_1+S_2\cdot S_3)\cdot(S_4+S_5\cdot(S_6+S_7))+S_8\\ 2\quad 1\quad 2\quad\ 4\quad 3\quad 4\quad 4\quad\ 4\end{matrix}\Big\} \qquad (2')$$

where the final match value is 4.

In section 2, the match value of S_{ij-1} ($1\leq i\leq m$, $2\leq j\leq n(i)$) 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, another value, called *check value*, 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 (2) are as follows:

$$\begin{matrix}&(S_1+S_2\cdot S_3)\cdot(S_4+S_5\cdot(S_6+S_7))+S_8\\ \text{match value}&2\quad 1\quad 2\quad 4\quad 3\quad 4\quad 4\quad 4\\ \text{check value}&0\quad 0\quad 1\quad 2\quad 2\quad 3\quad 3\quad 0\end{matrix}\Big\} \qquad (2'')$$

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 using a hash table is started. Let S_i mean 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 appearing 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≠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 operators 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 paper, the influence of complementary sets on the algorithms 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.

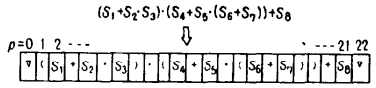
APPENDIX An Algorithm of Phase 1

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 parentheses 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 the 'operation' column are applied.

For example, the results of the processing of set formula (2) are shown in Fig.B, which coincide with (2'').



address	set id.	match	check	delimiter
10				
9				
8	S_6	4	0	
7	S_7	4	3	
6	S_5	4	3	+ 0
5	S_3	3	2	
4	S_4	4	2	+ 0
3	S_1	2	1	
2	S_2	1	0	
1	S_1	2	0	+ 0

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

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;

)	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.