REDUCED COMBINED INDEXES
FOR EFFICIENT MULTIPLE ATTRIBUTE RETRIEVAL

Ben Shneiderman

Computer Science Department
Indiana University
Bloomington, Indiana 47401

TECHNICAL REPORT NO. 43

REDUCED COMBINED INDEXES
FOR EFFICIENT MULTIPLE ATTRIBUTE RETRIEVAL

BEN SHNEIDERMAN

REVISED: JUNE, 1976

Reduced Combined Indexes For Efficient Multiple Attribute Retrieval

Ben Shneiderman

Computer Science Department
Indiana University
Bloomington, Indiana 47401

Abstract

Combined indexes were proposed by Lum [4] as an alternative to the traditional approach of intersecting multiple single attribute indexes. The combined index approach is appealing for queries requiring conjunctions of attribute values since it eliminates the need for time consuming intersections. The large penalty of wasted auxiliary storage space in the combined index approach can be minimized by adopting the reduced combined index technique proposed in this paper. A detailed description, with a number of variations and suggestions for implementation economies, is presented.

Keywords

combined indexes, reduced combined indexes, inverted files, indexing, multiple attribute retrievals, file organization, secondary index files, information retrieval, data management, file organization, query.

CR Categories

3.73, 3.74, 3.79.

A common file organization is a collection of records with unique primary keys, such as a part number or employee number, and multiple secondary keys which are attributes of the item. Classic indexed sequential organizations enable direct access to a record with a specified primary key, but access to all records with a specified secondary key, requires more complex indexing strategies. The popular method of constructing an index for each secondary key field [1,2,3] is simple to implement but inefficient when conjunctive queries on multiple attributes are required.

We limit the discussion to conjunctive queries and refer to the number of components as the degree of the query. The following examples show one, two and three degree queries:

Select student-records where (age equals 18)

Select student-records hwere (age equals 18) and (home-state equals Indiana)

Select student-records where (age equals 18) and (home-state equals Indiana) and (class-standing equals sophomore)

If indexes exist for age, home-state and class-standing, then the first query can be easily answered by finding all the references to 18 year old students in the age index. The second query requires searching both the age and home-state indexes and an intersection of the references from the two indexes. The intersection may be time consuming, especially in this example, because a large number of references may be obtained from each index. The third query requires three searches and a three way intersection. When the intersection is complete, the relevant records may be obtained from the student-records file by direct access retrievals of the referenced records.

To remedy the high cost of performing an intersection each time a query is made, Lum [4] proposed the idea of combined indexes or compound indexing. If we relabel the age, home-state and class standing attributes, A,B, and C, respectively, then all the possible conjunctive query forms could be represented by:

A	A and B	A and B and C
B	B and A	A and C and B
C	A and C	B and A and C
	C and A	B and C and A
	B and C	C and A and B
	C and B	C and B and A

Lum recognized that the ordering of query components was not critical and that it would be possible to answer all of the 15 above queries <u>with no intersections</u> if only three specially designed combined indexes were implemented. These combined indexes would use the key values combined (or concatenated) in the following three patterns: ABC, BCA and CAB. With this set of three three-degree (the degree of an index is the number of combined attributes) combined indexes, A queries could be answered directly from the first index, B queries from the second, C queries from the third, AB (equivalent to BA) queries from the first, BC (or CB) from the second, CA (or AC) from the third and all third degree queries from any one of the three indexes. Figures 1-3 show the three combined indexes for an example file of twenty records where three attributes are keyed.

Lum showed that with four keyed attributes, six four-degree combined indexes were necessary to answer all one through four degree queries: they were the ABCD, BCDA, BDAC, CADB, CDAB and DABC indexes. In general, for n keyed attributes the number of combined

indexes necessary to respond to all first through n degree queries by searching only one index is given by:

$$CI(n) = \binom{n}{\left\lfloor \frac{n+1}{2} \right\rfloor}$$

Sample values for this function are:

n	2	3	4	5	6
CI(n)	2	3	6	10	20

Lum's method required each of the CI(n) indexes to have an entry for each of the N records in the primary file. This led to extremely large indexes of N entries each of which were costly to search and required huge amounts of storage. These burdens and the extra updating costs when insertions or deletions were made, limited the applicability of Lum's method to situations where substantial amounts of storage could be sacrificed to reduce response time for queries and where updates were infrequent.

Fortunately, there is a method by which we can significantly reduce the size of the combined indexes, although the number of indexes remains unchanged. This method, reduced combined indexes (RC indexes), eliminates the redundancies in the indexes and reduces the number of entries in all but one of the indexes. This improvement shortens the response time for queries and lessens the storage overhead, but does not affect the update problem. Reduced combined indexes require less storage space and are less costly to search than Lum's combined indexes. In the case of three keyed attributes it is necessary to have only one three-degree index and two two-degree indexes: these are the ABC, the BC and the CA combined indexes.

Figures 4 and 5 show the BC and CA indexes which replace Figures 2 and 3. Although the reduction is modest in this case, the advantage with additional keyed attributes or longer files is more substantial. With four keyed attributes we need only one four-degree index, ABCD, three three-degree indexes, BCD, BDA, CAD and two two-degree indexes, CD, DA to answer all queries, without any intersections.

As an example of the savings consider a primary file of 10,000 records with four secondary attributes. If we assume a uniform distribution of attribute values within the records so that there are 10 values for each attribute, then there are 10,000 possible combinations each appearing once to make up the 10,000 records. Lum's combined index method requires six indexes, each of length 10,000 with 10,000 record pointer buckets each containing one record pointer. (Of course, if this optimum situation existed a convenient computational method could be used to locate the records without any indexes at all. Realistic situations have a sparse covering of the product space of attributes.) The reduced combined index method would require only one index with 10,000 entries (the ABCD index), three indexes with 1000 entries (where each record pointer bucket contains 10 pointers; the BCD, BDA and CAD indexes) and two indexes with 100 entries (where each record pointer bucket contains 100 pointers; the CD and DA indexes). The reduced method requires only 13,200 entries compared to 60,000 for Lum's method.

Although the number of entries is reduced, the number of reference pointers to records in the primary file is the same as in Lum's method. If we assume that a pointer is four bytes and a

key value is also four bytes (a conservative figure) then the four, three and two degree combined index keys will be 16, 12 and 8 bytes, respectively. This simple implementation, which does not assume any compression techniques, follows Lum's presentation rigidly. Under these assumptions, Lum's combined index approach would require (6 indexes) *(10,000 entries/index)* (20 bytes/entry) for a total of 1,200,000 bytes. The reduced combined index approach would require:

```
     (1 index) * (10,000 entries/index) * (16 bytes/entry)  }
   + (3 indexes) * (1000 entries/index) * (12 bytes/entry)  } the keys
   + (2 indexes) * (100 entries/index) * (8 bytes/entry)    }
   + (6 indexes) * (10,000 pointers/index) * (4 bytes/pointer) } the pointers
```

which yields a total of 437,600 bytes. A realistic implementation evaluation would have to consider bucketing techniques and disk accesses, but these figures do suggest that substantial reductions in storage space are possible.

The reduction in the number of entries, which produce the storage economies, also shortens the response time since all but one of the indexes have fewer entries. An order of magnitude reduction in the number of entries will probably lead to about one less disk access for the search process. A 100 entry index would require only one disk access while a 10,000 entry index would require three, assuming root nodes are not kept in the high speed storage. Figure 6 shows reduced combined indexes for two to six keyed attributes and Figure 7 shows the required number of RC indexes of each degree.

In general, for n keyed attributes the number of indexes, a_k, of precisely length k necessary for answering all queries

by searching a single index is

$$a_k = 1 \qquad k = n$$

$$a_k = b_k - b_{k+1} \qquad n < k \leq \lfloor (n+1)/2 \rfloor$$

where b_k is the number of indexes of length k or greater and is

$$b_k = \binom{n}{k} \qquad n \leq k \leq \lfloor (n+1)/2 \rfloor$$

The intriguing algorithm for finding a set of CI(n) combined indexes of degree n is given by Knuth (Exercise 1, page 567, reference 5). The necessity and sufficiency of CI(n) combined indexes is given in Lum's original paper [4]. Lum's proof is directly applicable to the reduced combined index scheme and Knuth's algorithm can be modified to generate the reduced combined index sets. For the sake of completeness, Knuth's algorithm and the modification is given in the Appendix.

Modified Combined Indexes

The reduced combined index method does produce the minimum number of indexes of minimum length to perform all queries by looking in a single index without doing intersections. An alternative idea, modified combined indexes (MC indexes), does not have the minimum number of indexes, but often the index to be searched to answer a query has fewer entires and can be searched faster. For n keyed attributes 2^{n-1} indexes are required. Figure 8 shows the indexes required for the modified combined index scheme for n equals 2 to 6 . The indexes required are generated by writing down a list of all of the $2^n - 1$ subsets of the integers $1,2,\ldots,n$ such that the values in each subset are in ascending order. Eliminate all list elements which are a prefix of any other list element, leaving precisely 2^{n-1} list elements. The letters A,B,C... were used to replace the integers 1,2,3... in Figure 8. The number of modified combined indexes of degree k is $\binom{n}{k}$ where n is the number of keyed attributes.

The shorter the string of letters in the index descriptor, the fewer entries required for the index. A crude estimator for the query response time is the average number of letters in the index descriptors. This is true because the numbered letters suggests the depth of the tree in the natural implementation. Not surprisingly, the modified combined index method has shorter average lengths for index descriptors than the reduced combined index method. This suggests that where response time is more critical than storage utilization and update rates are low, the modified combined index method may be preferable. A careful analysis of query and attribute value distributions is required to make the optimal choice.

The "natural" implementation of combined indexes

The high level of redundancy in combined indexes, particularly the leftmost attributes, suggests a tree structured implementation. In such an implementation, a three-degree index would produce a three-level tree and an n degree index would produce an n-level tree. Figures 9-11 show the natural implementation of Figures 1, 4 and 5. This factoring out of redundant data not only reduces the volume of information to be stored but simplifies the search process. For example, in an ABC index (see Figure 9) a three degree query could be responded to by a scan of the root node to match the A attribute in the request. Next the appropriate block at the second level of the tree would be scanned to match the B attribute of the request and the process would be repeated for the C attribute at the third level. The result of this search would be the list of record pointers which have records satisfying the A , B and C attributes of the request.

Additional efficiency can be obtained for queries specifying fewer than the complete set of attributes by having pointers from each level leading directly to the record pointer buckets. For example, if there were pointers from the root node (A attribute values) in Figure 9 to the 1st, 5th, and 11th record pointer buckets, then simple A attribute queries could be answered without going through the B and C levels of the tree. In general there could be pointers from each of the first (n - 1) levels of the tree to the record pointer buckets. We assume that there is some way of sequentially traversing the record pointer buckets. We have chosen to call such trees with pointers from each level to the bottom level, shrubs.

An additional implementation efficiency can be obtained by reducing the redundancy in the first few levels of several shrubs. For $n = 5$ (see Figure 6), three of the shrubs would have the same B attribute roots, three shrubs would have the same C attribute roots and two shrubs would have the same D attribute root. These redundant attribute value lists could be eliminated if we created only one list and simply had multiple pointers leading to the second level of the proper shrub. For $n = 6$ (or greater), we find that the first two (or more) levels of the shrub are identical and the redundancy can be eliminated by careful grafting of shrubs. We call the complete collection of these elaborate structures, shrubbery.

Practical Considerations

A large number of variations geared to special situations can be envisioned. Lum [4] and Knuth [5] both acknowledge that if an attribute has only two possible values, then special provisions can be made to substantially reduce the indexing burden.

If attributes can be isolated into groups, based on the knowledge that conjunctive queries are not required then substantial reductions can be made. For example, if a student data base has four health-related attributes and four financial aid-related attributes which are never components of a single query then only CI(4) + CI(4) indexes need be created as opposed to CI(8) indexes, that is, only 12 indexes instead of 72 indexes.

Since the generation of the complete set of combined indexes may be considered extravagent, implementers may be tempted to build a limited set of combined indexes for precisely the combinations

of attributes that appear frequently in queries [6]. This reasonable approach is highly recommended in situations where attributes are paired in a limited number of ways. For example, if only A, AB, C, D and CD queries are anticipated then only AB, CD and D indexes need to be built to avoid multiple index searches. Of course, B queries, any two degree queries involving (A or B) and (C or D), three degree queries and four degree queries will require multiple index searches and/or multiple searches within an index.

One implementation difficulty is determining which index should be searched to respond to a particular query. At least two solutions seem workable. Keep the CI(n) descriptor strings available and scan the list attempting to match the attributes in the query with the prefix of each string. More than one index descriptor string will work if there are less than or equal to $\lfloor (n+1)/2 \rfloor$ attributes in the query. The shortest index descriptor string should be used since it will reference the shortest index. The second method is to construct a list of each of the subsets (where the elements of each subset are arranged in ascending order) of keyed attributes in length order and lexicographic order. This list can be quickly binary searched to produce a reference to the appropriate index. Figure 10 shows an example list for this method for $n = 4$.

The updating burden is high for combined indexes since every index must be altered if there is a record insertion or deletion [7]. Reduced combined indexes improve this situation only in the case where an attribute in a record is modified. Then only the indexes which contain the modified attribute need be altered, thereby reducing the cost of modification. Still, reduced combined

indexes would be inappropriate for volatile files. Detailed performance estimates are a function of not only the implementation technique, but the query and attribute value distribution.

Conclusions

Although reduced and modified combined indexes can be compared to combined indexes, it is extremely difficult to make an accurate comparison against the traditional method of intersecting record pointers from multiple one degree indexes (inverted index lists). Reduced combined indexes seem most appropriate for situations in which there is a high retrieval to update ratio, a large number of records with similar attribute values, frequent high degree queries, and minimizing response time is more critical than efficient use of auxiliary storage. Multiple first degree indexes are preferable for situations in which there is a low retrieval to update ratio, a small number of records with similar attribute values, most queries are low degree queries and minimizing auxiliary storage space is more critical than reducing response time. Hardware considerations and implementation techniques further cloud the comparison. Although simulation studies might aid a database administrator in choosing between reduced combined indexes and multiple first degree indexes, only empirical studies can be conclusive.

There are further optimization problems once reduced or modified combined indexes are selected for an implementation. The assignment of attributes to the letters A,B,C,... can effect performance since the reduced combined index sizes will vary depending on the choice. A thorough knowledge of the attribute value frequencies and the query distributions are necessary to properly optimize performance.

Finally, it must always be remembered that implementation details have a profound effect on performance: a poor implementation of a good idea can lead to unacceptable performance.

Although there are unresolved optimization issues, reduced combined indexes can be competitive with multiple one degree indexes in some applications.

Acknowledgements

I would like to thank Andrew Lenard of the Indiana University Mathematics Department for taking an interest in this problem. He provided important insights to an alternate proof of the necessity and sufficiency of reduced combined indexes by use of the elegant "Marriage Lemma." The referees' comments substantially improved the organization and clarity of this paper.

References

1. Bleier, R.E., Treating hierarchical data structures in the SDC Time-Shared Data Management System, Proc. ACM National Conference, 1967, pp. 41-49.

2. Wagner, R.E., Indexing design considerations, IBM Systems Journal, Vol. 12, No. 4, 1973.

3. Vose, M.R. and Richardson, J.S., An approach to inverted index maintenance, Computer Bulletin, Vol. 16, No. 5, May 1972.

4. Lum, V.Y., Multi-attribute retrieval with combined indices, Comm. ACM 13, 11 (November, 1970).

5. Knuth, D., The Art of Computer Programming, Volume III, Searching and Sorting, Addison-Wesley Publishing Company, Reading, Massachusetts (1973).

6. Stonebraker, M., The choice of partial inversion and combined indices, International Journal of Computer and Information Sciences, Vol. 3, No. 2, 1974.

7. Mullin, J.K., Retrieval-update speed tradeoffs using combined indices, Comm. ACM 14, 12 (December, 1971).

Appendix: Generation algorithm for reduced combined indexes

First generate the $CI(n)$ combined indexes then reduce the degree of the indexes, where possible.

To generate the $CI(n)$ indexes of degree n we consider a path of n steps in a plane from the origin $(0,0)$. Each step of the path is from (x,y) to $(x+1,y+1)$ or to $(x+1,y-1)$ with the constraint that the path never goes below the x-axis. There are exactly $CI(n)$ such paths. Following each path separately, begin with three empty lists R, S and T. For $i = 1,2,\ldots,n$ if the ith step goes up, put the number i into S; if the step goes down, put i into R and move the currently largest element of S into T. If R, S and T are each arranged in ascending order and then concatenated, the sequence of numbers will be a representation of one of the combined indexes of degree n. Repeat the process for all of the $CI(n)$ paths to produce $CI(n)$ permutations of the integers $1,2,\ldots n$. If n is 4 and $1,2,3,4$ are made to correspond to A,B,C,D then the list of six combined indexes of degree 4 given earlier is produced. The paths and lists for the case of $n = 4$ are shown in Figure 13.

To reduce the degree of the indexes we follow an acceptance process.

1) The prefix consisting of the first $\lfloor (n+1)/2 \rfloor$ digits from the left are accepted in each of the $CI(n)$ combined index strings.

2) Generate a list of the $\binom{n}{\lfloor \frac{n+1}{2} \rfloor + 1}$ subsets of the integers $1,2,\ldots,n$ which are of length $\lfloor (n+1)/2 \rfloor + 1$.

3) Each element of the list of subsets generated in step (2) is matched to one of the combined index string prefixes such that a single digit extension to the right would be a permutation of the subset. The extensions of the combined index string prefixes are accepted.

4) Repeat steps (2) and (3) with increasingly larger subsets of the integers, such that only the strings which were extended in the previous iteration are eligible for matching in the current iteration. The final step will be the creation of the set of integers $1,2,\ldots,n$ and the extension and acceptance of a single string.

This process for $n = 5$ is shown in Figure 14, but the letters A-E have been substituted for the digits 1-5. In iteration 1, all of the subsets of length 1 , 2 and 3 are contained in a left-most prefix. In iteration 2, five strings have been extended to accommodate the five subsets of length 4 and in iteration 3, a single string has been extended to accommodate the single subset of length five.

Fortunately, this tedious process is necessary only to generate the reduced combined index descriptor strings which can be stored in a table for use when required.

Neither the generation process nor the acceptance process produce unique solutions. Generating all the possible sets of reduced combined indexes and selecting the optimum set based on distributions of the attribute values and the query profiles is beyond the scope of this paper.

A	B	C	record pointer list
18	IN	1FRSH	r_1
18	IN	2SOPH	r_2, r_3, r_4
18	MD	1FRSH	r_5
18	NJ	3JUNR	r_6, r_7
19	IN	1FRSH	r_8, r_9
19	IN	2SOPH	r_{10}
19	IN	3JUNR	r_{11}
19	MD	1FRSH	r_{12}
19	NY	4SENR	r_{13}
19	NY	5GRAD	r_{14}
20	MD	1FRSH	r_{15}, r_{16}
20	MD	2SOPH	r_{17}
20	MD	3JUNR	r_{18}
20	NJ	1FRSH	r_{19}
20	NJ	3JUNR	r_{20}

Figure 1.

Combined index ABC

B	C	A	record pointer list
IN	1FRSH	18	r_1
IN	1FRSH	19	r_8, r_9
IN	2SOPH	18	r_2, r_3, r_4
IN	2SOPH	19	r_{10}
IN	3JUNR	19	r_{11}
MD	1FRSH	18	r_5
MD	1FRSH	19	r_{12}
MD	1FRSH	20	r_{15}, r_{16}
MD	2SOPH	20	r_{17}
MD	3JUNR	20	r_{18}
NJ	1FRSH	20	r_{19}
NJ	3JUNR	18	r_6, r_7
NJ	3JUNR	20	r_{20}
NY	4SENR	19	r_{13}
NY	5GRAD	19	r_{14}

Figure 2. Combined index BCA

C	A	B	record pointer list
1FRSH	18	IN	r_1
1FRSH	18	MD	r_5
1FRSH	19	IN	r_8, r_9
1FRSH	19	MD	r_{12}
1FRSH	20	MD	r_{15}, r_{16}
1FRSH	20	NJ	r_{19}
2SOPH	18	IN	r_2, r_3, r_4
2SOPH	19	IN	r_{10}
2SOPH	20	MD	r_{17}
3JUNR	18	NJ	r_6, r_7
3JUNR	19	IN	r_{11}
3JUNR	20	MD	r_{18}
3JUNR	20	NJ	r_{20}
4SENR	19	NY	r_{13}
5GRAD	19	NY	r_{14}

Figure 3. Combined index CAB

B	C	record pointer list
IN	1FRSH	r_1, r_8, r_9
IN	2SOPH	r_2, r_3, r_4, r_{10}
IN	3JUNR	r_{11}
MD	1FRSH	r_5, r_{12}, r_{15}, r_{16}
MD	2SOPH	r_{17}
MD	3JUNR	r_{18}
NJ	1FRSH	r_{19}
NJ	3JUNR	r_6, r_7, r_{20}
NY	4SENR	r_{13}
NY	5GRAD	r_{14}

Figure 4. Reduced combined index BC, reduced from Fig. 2

C	A	record pointer list
1FRSH	18	r_1, r_5
1FRSH	19	r_8, r_9, r_{12}
1FRSH	20	r_{15}, r_{16}, r_{19}
2SOPH	18	r_2, r_3, r_4
2SOPH	19	r_{10}
2SOPH	20	r_{17}
3JUNR	18	r_6, r_7
3JUNR	19	r_{11}
3JUNR	20	r_{18}, r_{20}
4SENR	19	r_{13}
5GRAD	19	r_{14}

Figure 5. Reduced combined index CA, reduced from Fig. 3

n = 2

A B
B A

n = 3

A B C
B C A
C A B

n = 4

A B C D
B C D A
B D A C
C A D B
C D A B
D A B C

n = 5

A B C D E
E A B C D
B C D E A
B E C A D
B D E A C
C A D E B
C E A B D
C D E A B
D A B E C
D E A B C

n = 6

A B C D E F
F A B C D E
E A B C F D
E F A B C D
B C D E F A
B F C D A E
B E C F A D
B E F A C D
B D E F A C
B D F A C E
C A D E F B
C F A D B E
C E A F B D
C E F A B D
C D E F A B
C D F A B E
D A B E F C
D F A B C E
D E A F B C
D E F A B C

Figure 6.

Reduction of combined indexes for 2 to 6 keyed attributes

number of indexes of length	number of keyed attributes				
	2	3	4	5	6
1	1				
2	1	2	2		
3		1	3	5	5
4			1	4	9
5				1	5
6					1
TOTAL	2	3	6	10	20

Figure 7.

For n-attribute records, the table shows the number of indexes of each length required for the reduced combined index scheme.

n = 2	n = 3	n = 4	n = 5	n = 6
AB	ABC	ABCD	ABCDE	ABCDEF
B	AC	ABD	ABCE	ABCDE
	BC	ACD	ABDE	ABCEF
	C	AD	ABE	ABCF
		BCD	ACDE	ABDEF
		BD	ACE	ABDF
		CD	ADE	ABEF
		D	AE	ABF
			BCDE	ACDEF
			BCE	ACDF
			BDE	ACEF
			BE	ACF
			CDE	ADEF
			CE	ADF
			DE	AEF
			E	AF
				BCDEF
				BCEF
				BCEF
				BCF
				BDEF
				BDF
				BEF
				BF
				CDEF
				CDF
				CEF
				CF
				DEF
				DF
				EF
				F

Figure 8.

Modified Combined Indexes for n equal two to six.

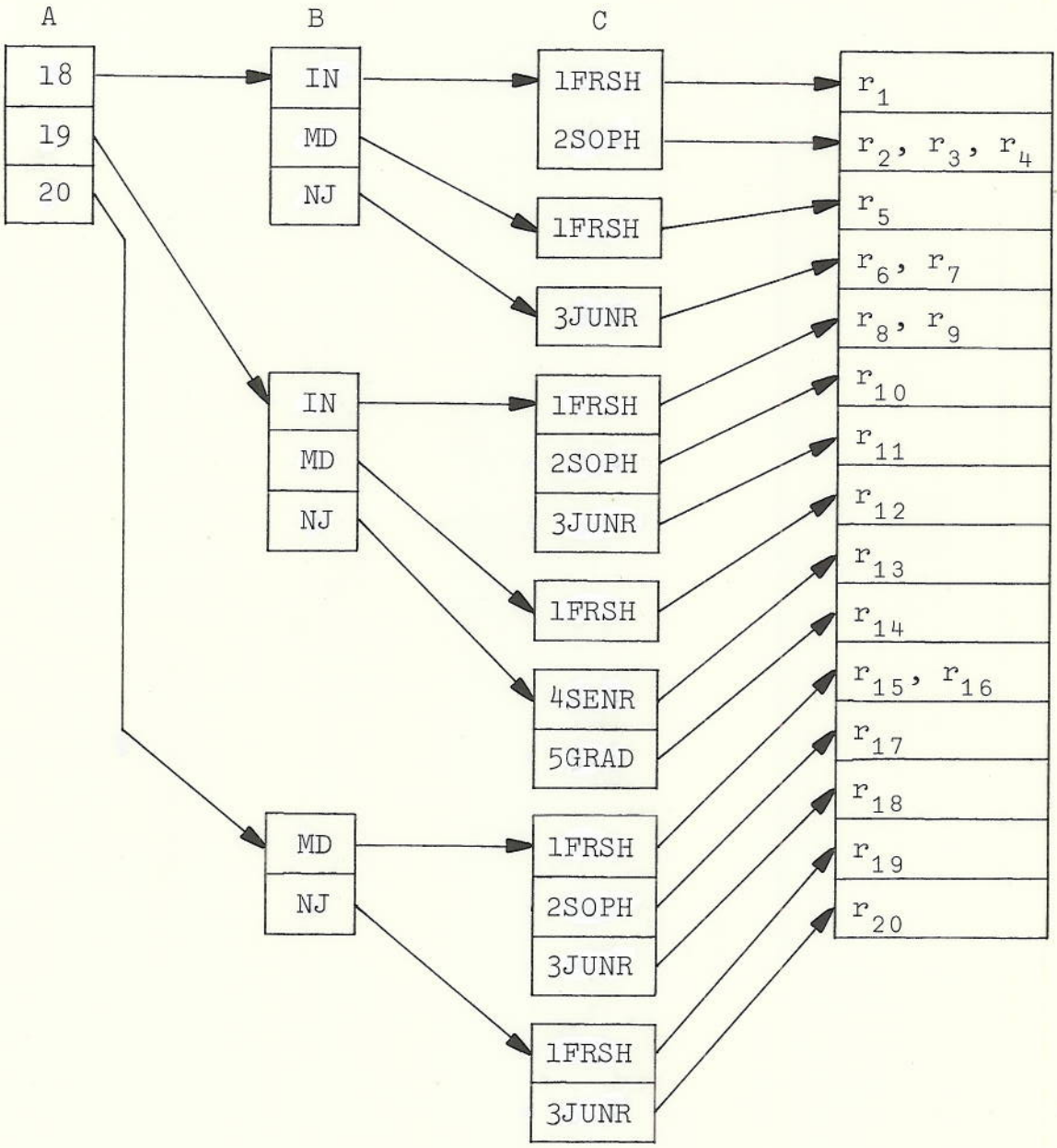
A
B
C
18
19
20
IN
MD
NJ
1FRSH
2SOPH
3JUNR
4SENR
5GRAD
r_1
r_2, r_3, r_4
r_5
r_6, r_7
r_8, r_9
r_{10}
r_{11}
r_{12}
r_{13}
r_{14}
r_{15}, r_{16}
r_{17}
r_{18}
r_{19}
r_{20}

Figure 9 .

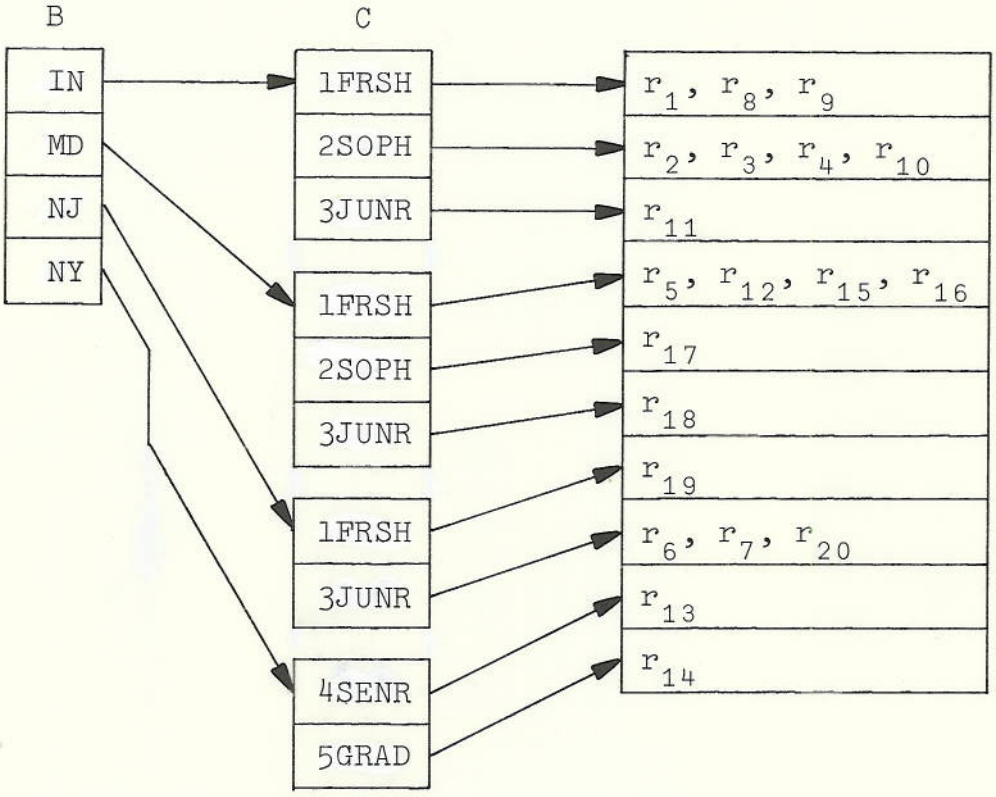
B
C
IN
MD
NJ
NY
1FRSH
2SOPH
3JUNR
1FRSH
2SOPH
3JUNR
1FRSH
3JUNR
4SENR
5GRAD
r_1, r_8, r_9
r_2, r_3, r_4, r_{10}
r_{11}
$r_5, r_{12}, r_{15}, r_{16}$
r_{17}
r_{18}
r_{19}
r_6, r_7, r_{20}
r_{13}
r_{14}

Figure 10.

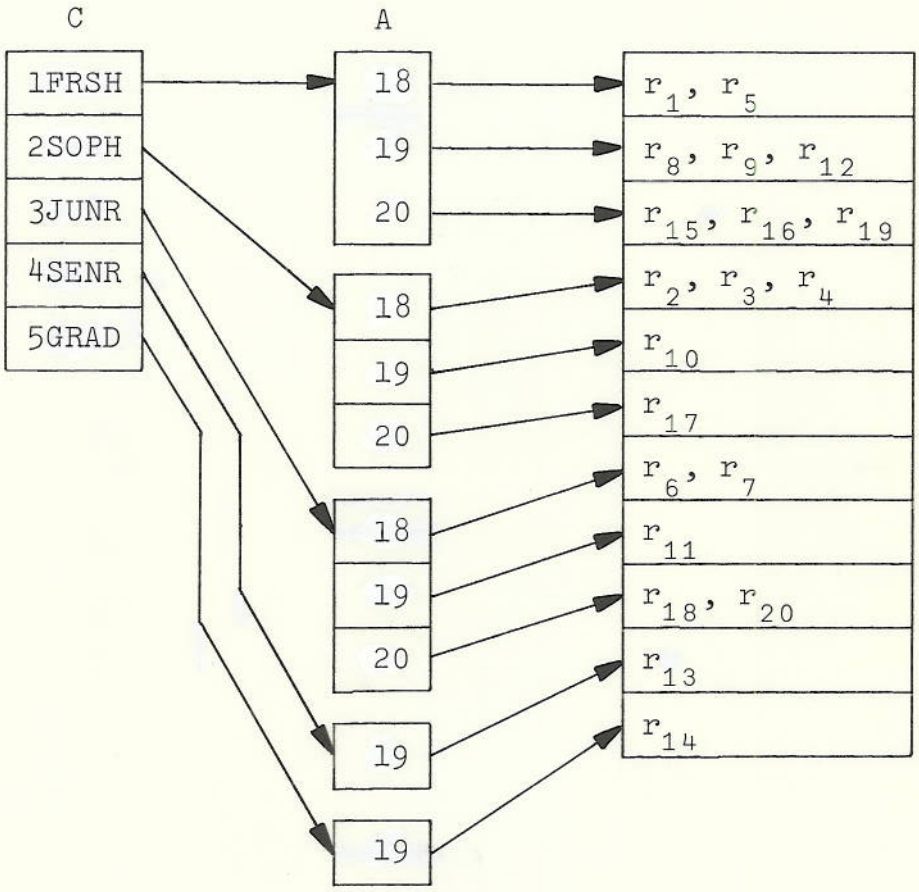
C
A
1FRSH
2SOPH
3JUNR
4SENR
5GRAD
18
19
20
18
19
20
18
19
20
19
19
r_1, r_5
r_8, r_9, r_{12}
r_{15}, r_{16}, r_{19}
r_2, r_3, r_4
r_{10}
r_{17}
r_6, r_7
r_{11}
r_{18}, r_{20}
r_{13}
r_{14}

Figure 11.

query	index
A	ABCD
B	BCD
C	CD
D	DA

query	index
AB	ABCD
AC	CAD
AD	DA
BC	BCD
BD	BDA
CD	CD

query	index
ABC	ABCD
ABD	BDA
ACD	CAD
BCD	BCD
ABCD	ABCD

Figure 12.

List of queries showing which index to search for $n = 4$.

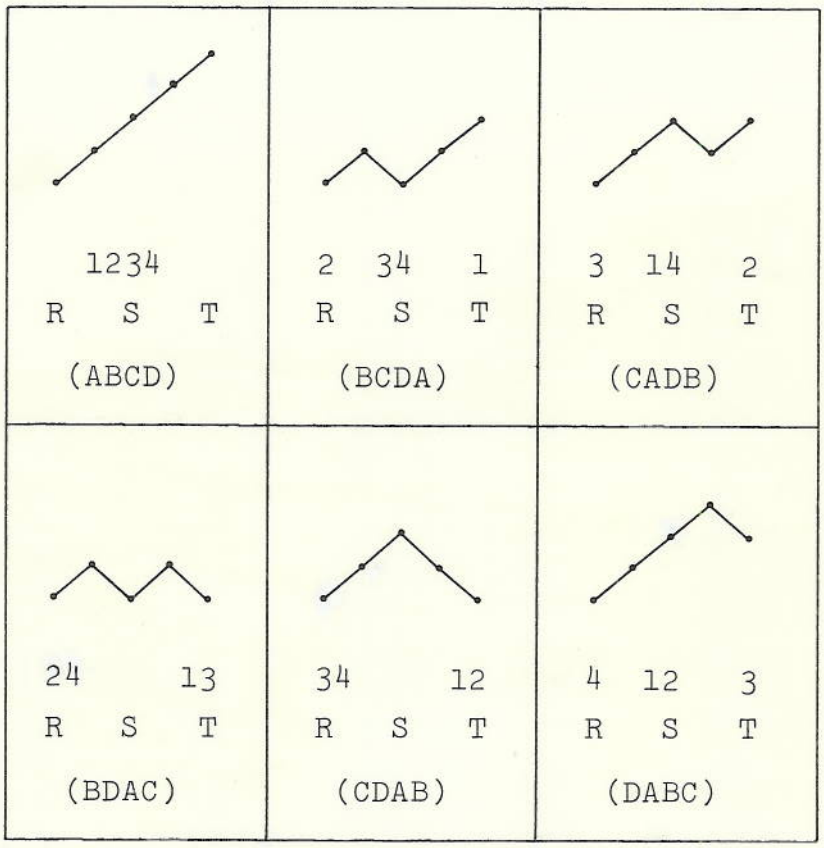

Figure 13.

Paths to generate CI(n) combined indexes of degree n=4.

Iteration 1	Iteration 2	Iteration 3
ABC DE	ABCD E	ABCDE
EAB CD	EABC D	EABC D
BCD EA	BCDE A	BCDE A
BEC AD	BEC AD	BEC AD
BDE AC	BDEA C	BDEA C
CAD EB	CAD EB	CAD EB
CEA BD	CEA BD	CEA BD
CDE AB	CDEA B	CDEA B
DAB EC	DAB EC	DAB EC
DEA BC	DEA BC	DEA BC

Figure 14.

The three iterations of the extension process for $n = 5$ to generate reduced combined indexes.