Frederick H. Ruecking, Jr.
Head, Data Processing Division
Fondren Library
Rice University
Houston, Texas

MARC AT RICE UNIVERSITY

Much of the experimentation associated with the MARC Pilot Project at Rice University has been continued into the MARC Distribution Service. Consequently, this review of activity incorporates results from studies using both formats and data. That these studies have been oriented toward retrieval problems is a product of circumstance.

At the time the pilot project was started, the library was not particularly interested in the production of catalog cards or book catalogs. The primary interest was, and continues to be, the development of aids to the regional exchange of resources. One of the early dreams in library automation was user access to a bibliographic file via an on-line, remote terminal.

The magnitude of a regional base, the required latitude in controlling search input, and the variations of supporting logic suggested that our interests could be served best by studying the characteristics of user-based input and examining alternatives for retrieval. Execution of these studies generated a broader perspective and involved the library in special projects distantly related to intended purposes.

This paper reflects this broadening of perspective. The initial studies were involved with search codes, i.e., compression algorithms designed to yield proper citations from faulty search input. A second series centered upon access methodology, while the third and final series (to be reported) concerned the organization of a library system.

Segments of the studies reported were funded by the National Science Foundation, the Texas State Library, the Regional Information and Communications Exchange, and Rice University.

SEARCH CODE STUDIES

A search code may be defined as an abbreviation which serves to identify a specific record. As used in this paper a search code is defined as the product of an algorithm which utilizes bibliographic

data to produce a limited character-string which can be used to uniquely identify a given item.

Since the algorithm must operate on data base and user-generated input, it must provide for the reduction of user error by offsetting mistakes in spelling and producing more opportunities for matching than generally employed in the current cataloging system.

The code strings described incorporate significant title words and author surnames. Corporate and conference entry codes are appended to the title codes. The complete series of possible codes is produced to yield maximum matching capability. Thus, if a work has an author plus two added entries and a traced title and added title, the total number of codes generated will be six.

Four algorithms were examined, each of which has certain advantages and disadvantages for an operational system. These four algorithms have been called the consonant-string code, the three-to-one code, the phonemic code, and the bucket code.

CONSONANT-STRING CODE

Briefly, concerning this code, the compression function is based upon the knowledge that English-language words frequently have initial and terminal consonant strings which tend to identify the word itself (assuming the deletion of prefixes and suffixes).

The test performed at Rice University on a large sample indicated acceptable accuracy without excessive false drop. The primary difficulty was the English-language bias of the algorithm suggesting the possible need for language-oriented algorithms for code generation. The complexity of the generating algorithm, the possible need for multiple algorithms, and the amount of processor time indicated suggested that additional codes should be examined. One of the high-accuracy codes found was a relatively straightforward reduction code based upon known character frequency.

THREE-TO-ONE CODE

There are two variations of this system, each of which appears usable. The first variant requires the use of the first character word as the leading character of the code string. The remaining characters of the word are divided into sequential blocks of three characters each. Using established frequency tables for the language, the least frequent character of each group of three is added to the code string. This simplistic algorithm sharply reduces the total length of the generated code, as compared with the consonant-string code which is always four characters long, unless the word is more than ten characters long.

The second variation utilizes the same frequency tables and uses the least frequent characters of a three-character group. However, this variation does not extract the initial character unless it happens

to be the least frequent. This variation produces a shorter code string than the first variation, and tends to distribute generated codes more evenly through the consonants in the alphabet.

THE PHONEMIC CODE

The phonemic code uses the basic method of the reduction code just detailed. However, the actual characters may or may not be used depending upon its definable phonemic value. For example, the "c" and "h" of the character string "sch" are not separable in English. The character pair is most nearly the "k," consequently, for coding purposes a word such as "school" would become "skul" when the vowel form "oo" is altered.

The algorithm for this construction is much more complex than the three-to-one reduction code, and necessitates more processor time. It does, however, provide some benefit in overcoming errors in spelling and in spreading out generated codes because of the higher number of phonemic symbols available.

Table 1 contains fourteen words and the codes produced by the variations of the systems described. The phonemic form is not exact, the purpose is rather to illustrate the types of character strings generated.

At this point we are unable to suggest which may be more advantageous in operational use. The three-to-one offers the most distinctive codes, but the phonemic codes provide greater dispersion of codes since there are more possible phonemic symbols available. A code which is not based on sequential word order is the bucket code.

THE BUCKET CODE

The principle of the bucket code is the utilization of the title and author stream as a character string without intervening blanks or punctuation. Data are extracted from the title and author with non-significant words discarded, a count is made of the number of characters in the string and blanks are used if the total is less than 128.

An exponential algorithm is used to pick the characters for the code, using "2" as the base value. Therefore characters 1, 2, 4, 8, 16, 32, 64, 128 are extracted to form the code word (eight characters). Carter and Bonk's *Building Library Collections* contains thirty-two characters and produces the code "BULGCR--."

The bucket code is not expected to produce completely unique codes for every item. It is expected to distribute the codes sufficiently which will tend to limit the frequency with which these codes are duplicated.

Our experience in code specification and testing suggests that there may not be a universal in search code formation. The utility of

Table 1. Word Code Comparison

Word	Consonant String	Three to One		Phonemic Form	Variations	
		Variation 1	Variation 2		1	2
CHRIST	CHRI	CHS	HS	KRAIST	KRS	KS
CHRISTENDOM	CHRI	CHSDM	HSDM	KRISTENDOM	KRNM	KSDM
CHRISTENING	CHRI	CHSNG	HSNG	KRISTNIŊ	KRNŊ	KNŊ
CHRISTIAN	CHRI	CHSN	HSN	KRISTYUN	KRYN	KYŊ
CHRISTIANITY	CHRT	CHSNY	HSNY	KRISTYANITI	KRYNI	KYNI
COLLECT	COCT	CLC	LLT	KOLEKT	KLK	KK
COLLECTS	COCT	CLCS	LLS	KOLEKTS	KLK	KKS
COLLECTING	COCT	CLCG	LLNG	KOLEKTIŊ	KLKŊ	KKŊ
COLLECTION	COCT	CLCN	LLON	KOLEKŠUN	KLKN	KKN
CONNECT	COCT	CNC	NNT	KONEKT	KNK	KK
CONNECTION	COCT	CNCN	NNON	KONEKŠUN	KNKN	KKN
CORRECT	COCT	CRC	RRT	KOREKT	KRK	KK
CORRECTING	COCT	CRCG	RRNG	KOREKTIŊ	KRKŊ	KKŊ

any code to be derived from user input may be dependent upon factors which are not under complete control of the library. Such factors as user sophistication, careless keyboarding, the ratio of English to non-English material in the collection, the subjective bias of the collection (if present), and collection size can materially affect the value of any one, or all, of the search codes which can be devised.

ACCESS METHODOLOGY

The most interesting problem and the most difficult to resolve is the formation of a file system which encompasses random needs. Interfiling, referencing, and deleting information from a large file necessitate an access technique which is the foundation for file organization. Three methods have been examined experimentally at Rice University, all of which have particular strengths and weaknesses. The important fact to be recognized is the use of the text itself to provide directional assistance to the record location.

THE HASHED FILE

Hashing is intended to evenly distribute textual data throughout a file which has specific, definable limits. A convenient method of securing such a distribution is the use of a prime divisor. For example, given a file which may contain numeric data ranging from one to 1,000 and will contain 100 different records at one time, the best prime to use is 101; therefore, the file will be defined as containing 101 records.

The actual numeric value of the text is divided by 101. The remainder resulting from that division will be between zero and 100. The actual file location of the record is obtained by adding one to that remainder.

There are certain characteristics which can be used to determine whether or not a hashed file is appropriate: 1) the file must be fairly stable without a great growth rate, 2) the lowest and highest text values must be known, 3) the size of the file must be definable, 4) the frequency of access must be high, and 5) the non-use of allocated storage space must be acceptable and economically reasonable.

The utility of a text-based file of this type is potentially enormous. At Rice University we hope to utilize this basic idea in our on-line circulation system. There are three basic on-line files—current loans, holds, and borrower. There are several off-line files in the form of printed lists and indexes, the most important being the alphabetic borrower index.

The current loan file has a maximum of about 40,000 records, the hold file 8,000, and the borrower file about 8,000. Text limits for the current loan and hold files are AA through ZZ and 000000000 to 999999999 for the borrower file. The current loan and hold file

records are quite short, requiring some sixty-four characters for each one; therefore, to secure more suitable use of storage space these records will be "blocked," 120 records to the block. We will, therefore, provide 337 record blocks since 337 is the nearest prime of 333. Similarly, we will use sixty-seven for the hold file.

Call numbers of individual transactions will be translated into numbers and divided by the appropriate prime. The remainder plus one will be the block in which the record will be found. An internal core scan will be used to ascertain the appropriate position of the record in the block.

The borrower file has longer records and 200 "blocks" will be needed giving an actual size of 211 blocks based on the prime. The social security number segment of the borrower number will be used to compute the location of borrower records.

Since all files have comparable record locations, providing file interplay is quite straightforward. For example, the following procedure stream on a return transaction would be standard.

1. compute record location
2. get record
3. if not overdue go to 9
4. if paid go to 9
5. compute fine
6. compute borrower location
7. get borrower record
8. prepare bill
9. if not held go to 20
10. compute hold location
11. get hold record
12. compute borrower location
13. get borrower record
14. prepare notice
15. delete hold record
16. subtract 1 from hold indicator
17. alter borrower number to hold status in loan record
18. notify attendant to hold and refile loan record
19. go to 21
20. delete loan record
21. exit

There are several additions which would be useful and several alternatives, but the routine series described does provide some idea of the automatic nature of the file interplay. The time frame for the twenty-one steps is machine dependent—but will almost certainly be I-O (input-output) bound. A procedural rearrangement could provide some increase in through-put speed by overlapping the I-O statements.

It is possible to provide considerable variability in hashed file

techniques by using a monitor which examines the use of allocated space on a periodic basis. Such a monitor could scan the file counting unused portions, then through a regenerating system reallocate the space and use a different prime to reduce or enlarge the needed space.

THE STRUCTURAL TREE

The alphabetic nature of bibliographic data suggests that a structural tree would be useful in manipulating bibliographic data. The structural tree is a system which operates in alphabetic sequence using a high-low-equal test at decision nodes. The start-point node can be computable or addressable from the text supplied, but the record containing the matching text can be located only by stepping through successive decision nodes.

Of primary importance in establishing a structured tree is establishing the initial sixty-four decision nodes. An error or miscalculation in the initialization will produce a distorted structure at a later date and one which will be grossly inefficient.

Assume a file of 4,000 names: to construct the initial decision nodes it is necessary to determine the distinctive character-strings of the names at the sixty-fourth interval (i.e., the sixty-fourth name, the one hundred and twenty-eighth, the one hundred and ninety-second, etc.). Once these points are known, the file can be initialized.

Each file record in addition to its text will also contain three directional addresses: 1) the location of alphabetic text of lesser value, 2) the location of text of greater value, and 3) the location of the last decision node. To interfile a record, the appropriate decision node is selected and its text is compared with the new text. If not equal the high-low decision selects the appropriate path into the tree and selects the next appropriate file record for comparison. If no directional location is found, a new node is created and the record is added to the file at the end. The appropriate information concerning its location is inserted in the prior node and the control counter which provides continuous locations for the system is augmented.

The tree file has several advantages—one of which is the conservation of storage space. Records can be blocked as fixed or variable length, and packed to maximum hardware specifications. Retrieval speed is I-O bound and lapsed time is a function of the number of nodes to be stepped through to reach a final decision. Since the process is effectively binary, each decision node eliminates one half of the remaining file records if the tree is perfectly balanced. Each segment of the MARC record can be separated as a discrete file or the entire record can be treated as a single file. Exploded records require linkage to the remaining segments of other files.

The primary disadvantage of the tree is the maintenance problem. Periodically the entire file must be rebalanced and the starting nodes

reevaluated to provide the necessary balance. Reloading a structured tree file of a million bibliographic records represents a difficult and time-consuming task even for a computer.

THE MULTIPLE RING

The multiple-ring idea has some similarities to the structured tree, particularly to the alphabetic appearance of an unsequenced file. However, the location reference of the file record has different functions. Location A refers to the location of the record which alphabetically precedes the current one. Location B refers to the location of the record which follows the current one. Location C is a lateral reference to the next ring.

Multiple-ring filing has the same complexity as the structured tree. A binary search is made of the file until a position is found for the record. The location references of the preceding and trailing records are modified to fit the inclusion of the new record, and the locations of those records are added to the current record which is added to the end of the file.

The file maintenance problem is eased from that of the tree because the file is based upon its contents and not what the contents may eventually be. Periodically the ring files must be regenerated to maintain binary search speed. The reloading of such files is much less complex.

One of the primary advantages of the ring is the conservation of space. Another is the retrieval speed, although it is also I-O bound.

The three techniques of arranging data for access have distinct utility in a library situation. Our experience in testing these different systems suggests that a completely on-line library system will have several methods of data arrangement and several file structures. The problem may not be in how to organize bibliographic data in files but how to provide adequate interplay among files of dissimilar order.

The programming required for these three systems is not difficult, but the ring and tree methods are extremely involved. Some of the basic tenets of each system were written in Autocoder for a 1401, others in COBOL for a Burroughs 5500. Consideration of the three systems brought about a study of a specific system for the Fondren Library which one of our wags called the "tree-ring" study.

THE INDEXED REGISTER SYSTEM

The register system incorporates components of the three systems described and adds some additional variations. The registry system is different primarily because all files are components of the

registry. The register file record contains all of the location data for all bibliographic elements exploded into other files plus some special data concerning each entry. The remaining files are indexes and authority files containing references to the registry records incorporating the text.

As mentioned, each file is an index and authority file, and in addition is a holding file. The main entry file is a good example of the multipurpose basis for the file. There are three record types—index, content, and holding. The index record includes a reference to the location of the authoritative content record in addition to the actual text of the index. The content record contains the complete MARC text which becomes the authoritative form for that entry. The holding record contains a reference to the location of the formal text plus multiple locations of registry records which utilize the authoritative text.

It should be noted that the main entry file will actually contain added entries as well as main entries. Each reference to the registry or from the registry will contain a flagging character to indicate the use of the authoritative text in a record.

The component files of this system are perhaps unusual because the actual text occurs one time and is not repeated. If the main entry, authoritative text must be altered to add a death date or a given name, in most instances the connection can be made without modifying any of the other records in the file.

Preliminary evaluation of the major data files of the library shows no balance between the three file arrangements in the registry system.

Hashed files clearly dominate the group followed by the rings. Tree structures are used where they can be advantageous.

In retrieving data in this system, the primary problem is to reach the register as quickly as possible. Once the register record is identified all of the component elements of the text can be drawn together for output—as a MARC II record. This exploded data system does have its drawbacks; it is expensive. Nevertheless, it does provide direct access interplay between files threading data laterally through the register and vertically in all files.

The retrieval technique in terms of multiparametered, multilogical sets of data can be organized for greatest efficiency for the computer and for simplest procedures for the user. There does not appear to be any way of avoiding a special language for using library files. We are now exploring these problems and soon hope to have a working on-line cataloging system.

A user could type the command "seek" and then specify the logic text type and text to be sought, for example, "Seek not title collected or selected author hemingway." The user is requesting all materials

Table 2. File Arrangements in the Registry System

File Name	Hashed	Tree	Ring
Current Loans	x		
Reserve Inventory	x		
Borrower File	x		
Holds	x		
Main Entry			x
Title		x	
Edition	x		
Imprint	Included in Borrower File		
Collection	x		
Series Notes			x
Notes		x	
Subjects			x
Added Entries			x
Added Title		x	
Added Lines			x
Call Number	x		
Card Number	x		
Order File	x		
Invoice File	x		
Process Control			x
R.I.C.E. Accounts			x
Fund File	x		
Gift & Exchange			x
Binding File			x
Statistics		x	
Register	Sequential only		
Search Code		x	

by Hemingway which do not have the words "collected" or "selected" in the title.

Ordinarily the not logic should be tested first; however, in this instance the author entry will be more specific more rapidly and consequently the Hemingway string of data records will be used to access the register which in turn provides the location data for the title file. Each title extraction tests the text for the undesired words, and generates an output record for items fitting the requirements. Free form input with defining labels or words appears to be the best method of providing user-system interchange at Rice University.

The concluding commentary that follows is generalized. One does not compress five years of experimentation into a few pages without losing most of the detail. The use of MARC data at Rice University has been unique because of the particular circumstances which have existed and continue to exist. The absence of disastrous crises in technical processing permitted a level of experimentation with MARC data others could ill-afford. Some of the efforts in constructing concordances, providing selective dissemination of information from *Chemical Titles* and building research files for authors such as Thomas Mann provided an opportunity to compare techniques and problems caused by format differences.

The Fondren Library may be the only library to have used three different computing systems and four different languages on MARC data, as indicated below.

1. IBM 7040	MAP and COBOL
2. IBM 1401	Autocoder
3. Burroughs B-5500	ALGOL and COBOL

Our experience suggests that the existing format is easily handled by any of the current languages and probably by any hardware configuration on the market. Naturally, each system and each language present problems in programming for MARC applications. We have had our share, too, but have been able to resolve them in one way or another.

There are two points that I believe are of paramount importance to library automation. First, there does not appear to be a "best" way to do anything; everything seems to be relative. What works beautifully at one library may not do so well at another. People in library automation are under constant pressure to implement this and study that. The one thing most desperately needed and rarely obtained is adequate time to think about the situation and to use ingenuity and daring which produce the great innovations.

An aspect of the present situation which may prove more than a little disconcerting is micrographic storage. For five years we have encouraged, beseeched, and nourished the project which has grown into MARC. Now on the horizon looms a technological advance for which no one is ready—micrographic storage. The on-line storage cost for one billion bytes is not a small sum for today's libraries. The development of computer output on microfilm, microfilm and microfiche retrieval systems, some of which are computer driven, is a clear warning not to move too far toward the on-line storage concepts I have described.

The micrographic industry (into which the computer manufacturers are moving) is at the beginning of a period of successive important advances. These advances must be carefully watched during the development of any computer-based library system. MARC is here and is usable in many aspects of library processes. Use it. Be brave—but do not be foolhardy.