Journal of Theoretical and Applied Information Technology
15th February 2025. Vol.103. No.3
© Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

THE MOST EFFECTIVE METHOD FOR FINDING REGULAR EXPRESSIONS IN DOCUMENT DATABASES

[1]KOTESWARA RAO KODEPOGU, [2]MOLIGI SANGEETHA, [3]RAVI UYYALA, [4]CHALLAPALLI SUJANA, [5]NARESH VURUKONDA, [6]DR. SIVUDU MACHERLA,[7]CHETLA CHANDRA MOHAN

[1]Assoc. Prof, Dept. of CSE, PVP Siddhartha Institute of Technology Vijayawada, Andhra Pradesh
[2]Sr Asst. Professor, CSE DEPARTMENT CVR COLLEGE OF ENGINEERING, HYDERABAD.
[3]Assoc Professor, Department of CSE. Chaitanya Bharathi Institute of Technology, Gandipet, Hyderabad,
[4]Asst Prof, Dept. of CSE, Aditya College of Engineering and Technology, Surampalem, Andhra Pradesh,
[5]Dept. of AI, School of Technology Management and Engineering, SVKM's Narsee Monjee Institute of Management Studies (NMIMS) Deemed-to-Be-University, Hyderabad Campus, Jadcherla-509301,
[6]Asst Professor, Dept. of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, A.P, India.
[7]Asst Professor, Dept. of IT, PVP Siddhartha Institute of Technology Vijayawada, Andhra Pradesh

,
Email: drkoteswararao83@gmail.com

Abstract
Considering a list of n strings with a maximum length of k, where l is the longest string. The goal is to use as few regular expressions as possible to cover the strings (r1, r2, r3……………..,rm for m≥1) so that: a) Every text in the database meets at least one ri and b) Any string "X" of length at most K satisfying r1+r2+r3+And so on.The distance between a string "y" in the database and +rm is at most p, where "p" is a specified constant parameter. We presume that the database is in the form of B+ tree. We begin with leaf nodes and gather all of the database's longest strings. The goal of the paper is to create a process for detecting regular expressions in databases that is comparable to that for Boolean formulae (in DNF or CNF), where function values and don't care words are supplied.

Keywords: *CNF, DNF, Document, Databases, Effective Method.*

1. INTRODUCTION

A document database is a sizable collection of documents from a variety of sources, including books, research papers, news stories, e-mail messages, websites, and digital libraries. The majority of text databases contain semi-structured data, which is material that is neither entirely unstructured nor entirely structured. An example of this would be a document that has some structured fields, such titled, authors, publications, and so on, but also has entirely unstructured text components, like abstract and contents. The database is assumed to be a B+ tree. In this case, the database will contain the dictionary's data. The data is at the leaf level, and since the leaves can be connected, the data can be accessed sequentially when the leaves are reached. Additionally, they indicate that interior nodes serve as a pointer to the information stored in leaves and only include referential data. We start with the leaf nodes and gather all of the database's longest strings. The dynamic programming approach (DPA) serves as the foundation for the distance computation technique, which in this study is the edit distance.

To compensate for edit errors in the database strings, a precision parameter is employed. The most widely used generic class of formal symbolic representations for describing character strings, including words, sentences, or any other arbitrary text that is specified by a pattern, are regular expressions. Automata-realization in regular expressions are sequential circuits that decide whether to go on to the next state just by looking at the current symbol in the automaton's scanned state. Sequential circuits are implementers of Boolean functions of a fixed number for binary strings of a fixed length. These circuits can also be extended, or at least attempted to be extended, for the realization of regular expressions.

While the general class of regular expression operators include UNION, CONCATENATION, and KLEEN Star, the work's goal is to demonstrate the transformation required for an extension of the state minimization rules using the general class of Boolean operations AND, or, and not. The (normal) set UNION operation and the Boolean OR operation are identical, but the others are not. Therefore, by using an analogy with Boolean circuits, we examine a more useful class of regular expression operators (found in Unix utilities) that permit or make it easier to formulate the state minimization criteria.

There are certain unusual operators in the practical class of regular expressions that can be combined using state-based rules. The grep command in UNIX handles or permits the practical regular expressions. The grep command line utility looks for lines of text that match one or more regular expressions given a list of files or standard input to read. Depending on the options provided, it either prints the lines that match or the lines that don't match (with the –v option). For the best representation of patterns in queries, the grep commend enables the generation of regular expressions using a variety of meta characters, including +,., *,?,\{n,m}\, substring, and finite extent repeaters. The following patterns can be used with the finite extent repeaters:

. match any single character except< newline >

* match zero or more instances of the single character (or meta-character) immediately preceding it (equivalent to the regular expression matching the full set of strings)

[ABC]correspond to any character in the enclosed [a–d]correspond to any character in the range that is contained.

[\exp] correspond to any character that isn't in the expression ^abcThe regular expression needs to say "Anchor" at the start of the line. The regular expression abc$ must terminate at the line's end (Anchor); take the following character literally. This is typically used to get around special characters like "." and "*." The regular expression that comes before this must be matched at least n times and at most m times by \{n,m|} (0 through 255 are acceptable for n and m). Consider the \{and \} sets as single operators.. In this case the \ preceding the bracket does not escape it special meaning , but rather turns on a new one.

As long as it is a single word, / will match the contained regular expression. Word boundaries are defined as ending with the same character or an end-of-line character, or starting with or anything other than a letter, number, or underscore(_). Once more, \< and \> sets should be viewed as individual operations.

The contained pattern is saved in a buffer by abc\). It is possible to save up to nine patterns for every line. These latter can be referred to using the character set. Once more, the \(and \)set should be viewed as a single operator. \n where n is between 1and9. This matches the nth expression previous saved for this line. Expressions are numbered starting from the left. The\n should be thought of as a single operation .& print the previous search pattern (used in the replaced

There are a few meta-characters used only by awq and egrep.

These are

+ match one or more of the preceding expression (same as kleene star excluding the null string)

? match zero or once of the preceding expression

| match either the preceding or following expression

. match any single character except <newline>

() group the regular expressions within

Command/syntax:

grep/ egrep/fgrep[options] 'search string' file

`Search the argument (in this case probably a file) for all occurrences of the search string, and list them .

To find generalized regular expressions in UNIX files, using the grep program. When using the grep program, it is recommended to specify regular expressions like the ones above in apostrophes or single quotes. An expanded set of meta-characters can be used for searching with the egrep function. Below are some samples, the grep utility's syntax, and some of the settings that are available.

.Syntax:

grep [options] regexp [file[s]]

common options:

-I ignore case

-c report only a count of the number of lines containing matches ,

-v invert the search, displaying only lines that do not match

-n display the line number along with the line on which a match was found

-s work silently ,reporting only the final status:

0, for match(es) found

1. For no matches
2. For errors

1.1 survey of the paper

The purpose of the paper is to find R.E. in text databases. Here, we examine what a text database is. How regular expressions are handled. Regular expressions that function as Boolean expressions. B+ tree, DPA for editing distance, regular expressions, Boolean operators with regular expression, state minimization, and grep are some of the chapters in this thesis. The chapter on B+ trees covers topics such as what a B+ tree is, how it is constructed, and where data is stored. This chapter discusses the current work on editing distance using the dynamic programming method. We shall go over the definitions of regular expression and Boolean expression in the chapter on regular expression. Here, we solve for Boolean expressions by applying the methods for solving to regular expressions. The circuit minimizing techniques utilized in Boolean circuits can also be extended for regular expression realizations in the State minimizing chapter. The chapter grep will explain what grep is and what practical regular expressions are handled or permitted by the Unix grep tool. The grep command line utility looks for lines of text that match one or more regular expressions when given a list of files or standard input to read. Depending on the options you choose, it will either output the lines that match or the lines that don't match (with-v option).Finally, we will conclude with a bibliography and online references.

1.2Flow of the Paper:

Step 1: Assume that the database is a B+ tree. The dictionary's data will be the database. We begin with leaf nodes and gather all of the database's longest strings.
Step 2: We use the dynamic algorithm (DPA) as our distance method to modify the programming distance with the appropriate precision value.
Step 3: Which regular phrase is appropriate for describing text? Although they are equal, the regular expression specifications permitted by the UNIX egrep program are utilized because they are composed of more practical operators than those found in the formal definitions.
Step 4: Regular expressions are combined using the same rules as Boolean expressions.
Step 5: Reducing superfluous states using a state minimization algorithm
Step 6: Report and collect feedback if needed.

2.B$^+$ Tree: It is assumed that the database is a B+ tree. We begin with leaf nodes and gather all of the database's longest strings. The interior nodes of a B+ tree do not contain any data. Since all of the information is contained at the leaf level, the leaves can be connected to one another, enabling sequential access to the data after the leaves reach T. This implies that interior nodes serve as a guide to the data stored at leaves and solely include referential data. A dynamic index structure that adapts well to insertion and deletions is the B+tree. The tree is in balance.
Leaf pages are not allocated sequentially .they are linked together pointers(a doubly linked list)

2.1 B+ tree properties

B+ tree is a rooted tree satisfying the following properties.

- All paths from root to leaf area of the same length
- Each node that is not a root or a leaf has between [n/2] and n children.
- A leaf node has between [(n-1)/2 and n-1 values special cases:
- If the root is not a leaf, it has at least 2 children.
- If the root is a leaf (that is there are no other nodes in the tree), it can have between 0and (n-1) values

2.2 Main characteristics

Maintain a balanced tree height while inserting or removing at log N F caste.(N= #leaf pages, f=fan-out). 50% occupancy minimum (root excluded). There are d<=m<=2d entries in every node. The order of the tree is denoted by the parameter d. effectively supports range searches and equality. A B+ tree is a type of data structure that can hold a large amount of data. Large volumes of data that cannot be stored in main system memory are typically stored in B+ trees. To view this, the tree's leaf nodes are stored on secondary storage, often a disk. Computer memory only contains the interior nodes of the tree.
The only nodes in a B+ tree that genuinely store data items are the leaf nodes. The remaining nodes, referred to as index nodes or i-nodes, merely store "guide" values that enable us to navigate the tree structure from the root down to the leaf node that contains the desired data item. These leaf nodes each hold many data items because disk I/O is more slower than memory access. Actually, under most operating systems, the data structure will function best in a leaf node size that closely resembles a disk sector.
. Therefore, we still need to read that data node from the disk and search its contents when we search a B+ tree (by going from the root node down to the correct dat node). Maintaining a memory cache of previously read nodes is another technique to increase query operation speed. A structure called a B-tree, in which data items can be stored in any node on the tree, is the ancestor of the B+-tree.

The B+-tree is a little more robust and complex version of the B-tree.

2.3 Example of a B+ -tree

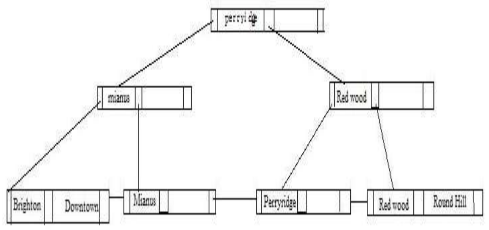


Fig.2.1 B+ tree

The B+ tree index is a balanced tree where data entries are found in the leaf nodes and the search is guided by the internal nodes (the top two levels).All that is needed to search for a record is a traversla from the root to the relevant leaf node. The height of the tree is the distance from the root to a leaf, which is often two or three. We follow the leftmost child pointer from the root (as 9<10) to look for entry 9*. The correct child pointer is then followed at level two (since 9>6). Data entries can be located consecutively after arriving at the leaf node.Because the leaf nodes are connected, range queries can be used with it.

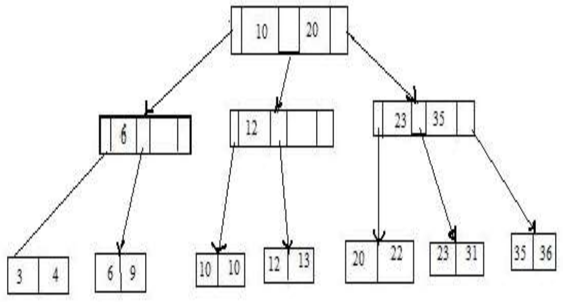


Fig.2.2 B+ tree

The records can be accessed via an index or in insertion order.

3.DPA To Edit Distance: The dynamic programming algorithm (DPA) is the distance method we use to modify distance with the associated precision parameter. Because the terms "computer" and "commuter" are so similar, changing just one letter—from p to m—will turn the former become the latter. The word "sport" can be transformed to "sort" by removing the letter "p," or vice versa, "sort" can be changed to "sport" by adding the letter "p." When two strings, s1 and s2, are edited, the edit distance is the smallest number of point mutations needed to transform s1 into s2, where a point mutation is one of

- Change a letter,
- Insert a letter or
- Delete a letter

The following recurrence relations define the edit distance, d(s1,s2), of two strings s and s2:

d(“ “) = 0 --“ = empty string

d(s,”) = d(“,s) = |s| --i.e length of s

d(s1+ch1, s2+ch2) = min (d(s1,s2) +if ch1=ch2 then 0 else 1 fi, d(s1+ch1,s2)+1, d(s1,s2+ch2)+1)

The first two rules above are obviously true , so it is only necessary consider the last character,ch1 and ch2 respectively. Somehow , ch1 and ch2 have to be explained in an edit of s1+ch1 into s2+ch2. If ch1 equals ch 2, they can be matched for no penalty i.e 0, and the overall edit distance is d(s1,s2). If ch1 differs form ch2 then ch1 could be changed into ch2, i.e . 1,giving an overall cost d(s1,s2)+1. Another possibility is to delete ch1 and edit s1 into s2+ch2, d(s1,s2+ch2)+1. the last possibility is to edit s1+ch1 into s2 and then insert ch2, d(s1+ch1,s2)+1.There are no other alternative . we take the least expensive i.e. min, of these alternatives.

Examination of the relations reveals that d(s1,s2) depends only on d(s1,s2) where s1 is shorter than s1, or s2 is shorter than s2, or both. This allows the dynamic programming technique to be used.

A two-dimensional matrix, m[0..|s10|,o …|s2|] is used to hold the edit distance values:

M[I,j] = d(s1[1…i],s2[1….j])

M[0,0] = 0

M[I,0] = I, I=1….|s2|

M[I,j] = min(m[I-1,j-1] +if s1[1]=s2[j] then o else 1 fi, m[I-1, j]+1, m[I,j-1]+1)I=1…|s1|,j=1..|s2|

M[.] can be computed row by row. Row m[I,]depends only on row m[I-1]. The time complexity of this algorithm is O(|s1|*|s2|). If s1 and s2 have a ‘similar’ length, about ’n’ say, this complexity is O(n2), much better than exponential

1.1 Algorithm

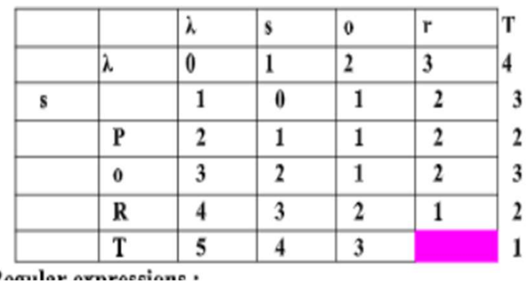
```
edit distance(A[1….m],B[1….n])
for i ⟵ 1 to m
   edit[i , 0] ⟵ i
for j ⟵ 1to n
   edit[0 ,j] ⟵ j
for i ⟵ 1to m
      for j ⟵ 1 to n
            if A[i] = B[j]
                  Edit[I,j] ⟵ min{ edit[i-1,j]+1,Edit[I,j-1]+1,Edit[i-1,j-1]}
      Else
                  Edit[I,j] ⟵ min { Edit[i-1,j]+1, Edit[I,j-1]+1,Edit[i-1,j-1]+1}
```

Return Edit[m,n]

3.1.1 Example

let us consider an example where we are editing form strings sport to sort.So we follow the dynamic programming algorithm for editing distance. i.e sport to so? Rt.Form the algorithm we get the distance as

Solution: 1. 3.1 Table for example



		λ	s	o	r	T
	λ	0	1	2	3	4
s		1	0	1	2	3
	P	2	1	1	2	2
	o	3	2	1	2	3
	R	4	3	2	1	2
	T	5	4	3	🟪	1

So the distance (sport , sort) is 1.

4.REGULAR EXPRESSIONS :

To specify a set of strings, such as the set of all valid email addresses or the set of all binary strings with an even number of 1s, use a regular expression. We cannot just list all of the members of the set because it may include an unlimited number of them.We are creating a process that is comparable to that for Boolean formulas (DNF and CNF), where function values and don't care are defined, in order to find regular expressions in databases. Regular expression: simple expressions can readily explain the language that finite automata accept. Strings of characters (words, sentences, or any random text) are described by regular expressions. It is a collection of characters that define a style. In addition to unions, concatenation, and kleene closure (or closure), we assume AND & OR operations. The general classes of regular expression operators are UNION,KLEEN STAR and CONCATENATION.A formal recursive definition of regular expressions over ∑ as follows

1. Λ and Ø are regular expressions, as are any terminal symbol (that is, an element of ∑ n).
An in ∑ is represented by a when we consider it to be a regular expression.
2. R1+R2, which is the union of two regular expressions R1 and R2, is likewise a regular expression.
3. Two regular expressions R1 and R2 concatenated as R1, R2 is likewise a regular expression.
4. A regular expression R* is an iteration (or closure) of a regular expression R.
5. (R) is also a regular expression if r is a regular expression.
6. The recursive results of applying rules 1–5 once or more times are the exact regular expressions across ∑.

4.1 Identities For Regular Expressions

These are useful for simplifying regular expressions.
Ø + R= R
ØR=R Ø= Ø
ΛR=RΛ=R
Λ*=Λ and Ø*=Λ
R+R=R
R*R*=R*
RR*=R*R
(R*)*=R*
Λ+RR*=R*= Λ +R*R
(PQ)*P=P(QP)*
(P+Q)*=(P*Q*)=(P*+Q*)*
(P+Q)R= PR+QR and R (P+Q) =RP+RQ
If R=Q+RP by Ardens theorem r=QP*.

4.2 Regular expressions notation

The language operations are union, Kleenstar, concatenation

4.2.1Concatenation operator

If x,y Є I * , then the concatenation of x and y is written as
Z=x,y
X=101 |x|=3.
Y=111 |y|=3.
Z=x,y -> 101111 |z|=6
Concatenation of any string with null string results in the original strings.
x.e = e.x = x.
x. Λ = Λ .X=X.
Note : e & Λ are null strings
Example :- x=100 e=Ø.
x.e--> 100
concatenation is associative :
x=101 y=111 z=110
x.(y.z) = (x.y).z
101111110=101111110

4.2.2 Kleene star.

If L C= I* is a language , then
L* is the set of all strings obtained by concatenating zero or more strings of L.
Concatenation of zero strings is Λ.
Concatenation of one strings is the string itself.
L+ = L*-{ Λ}.
Eg:- L = {0,1}
L*
{
Λ,0,00,000,…………………0*,1,11,111………

……………1*,01,001,0001,…………0*1……… ……..}
L={ab, f}
L* = { Λ,ab,abf,fab,ffab,ffabf…………}
Ø* → { Λ}
If l = { Λ.} then L* = { Λ}.
Let I = {a}
L=language((ab)*)
→{ Λ,a,b,ab,aab,abb,……..}
The laguage of all strings a's and b's in which the a's if any come before b's.

4.2.3 union

L = {001,10,11} ,m= { Є,001}

L U m = { Є,10,001,11,001,10+001,11+001}
If E &F are RE's then E+F is a RE's denoting the union L(E) and L(F).
L(E+F)→ l(E) U L(F).

4.3 Boolean 'OR'

1. distribute over concatenation
L=language((a+bc)c*b)
L = language(ac*b+bcc*b)
Which is the laguage of all strings beginning with a ending with b and having non or more L's in the middle and All strings beginning and ending with b and having atleast one 'L' in the middle
2.distribute when it is inside a kleene starred expression, but only incertain ways.
L = Language ((a+bc)*b) = (a+bc)(a+bc)(a+bc) Λb.
≠a*b+bc*b
≠(ab+bcb)*.
(a+b)*→ the set of V(for all) strings of a&b of any length.
L=Langyage((a+b)*){Λ, ab,abab,abaab,abbaab,bbb,……………}
If L C =I* is finite then L is regular .
If L2 and L2 are regular , so are
L3= L1 UL2
L4 = L1.L2={x1.x2/x1Є L1 ,x2ЄL2}
If L is regular, then so is L* ,where * is the Kleene star.
In this we follow same rules as followed for solving Boolean expressions for solving regular expressions using AND, OR.
Given a set.
L= { 0,1,00,11,10,01,100,010,000,110,001,011,101,111 ……….}
R1+r2→ 0+1→1(AND)
R1.r2→ 0.1 → 1 (OR)
R1*.r2→ 0*1→ 001

4.4 Basic operations for creating regular expressions

There are five basic operations for creating regular expression, and the table below illustrates them by example.

Concatenation: Concatenating a group of symbols one after the other, like aabaab, creates the most basic kind of regular expression. Only the single string aabaab is matched by this regular expression. The concatenation technique allows us to do basic spell checking. For instance, we could create the regular phrase neither, and then determine that neither is misspelled if every word in a dictionary matches. Logical OR: We can select among a number of options using the logical OR operator.. The regular phrase aa| baab, for instance, matches precisely two strings, aa and baab. Numerous spam filters, like spam Assaain, function by looking for a lengthy list of frequently used spam phrases. A regular expression like AMAZING| GUARANTEE|viagra might be formed by them. We can define many strings with a single regular expression thanks to the logical OR operator. For instance, we would want to know if the phonepad prints out any words if our number is 734-8527 (2 = abc, 3 = def, 4 = ghi, 5 = jkl, 6 = mno, 7 = prs, 8 = tuv, and 9 = wxy);

All 3^7 possible combinations are specified by the following regular expression: (p|r|s)(d|e|f)(g|h|i)(t|u|v)(j|k|l)(a|b|c)(p|r|s). It turns out that the word regular is the sole English term that matches (substitute deciphering an instant message that uses the "phone code" for this example)

.Replication: We can provide an endless number of options thanks to the replication operator. The regular expression ab*a, for instance, matches a,aba,abba,a,bbb, and so on. Keep in mind that b may not be replicated 0 times. Grouping: We can give the different operators precedence by using the grouping operator. The highest precedence operator is *, followed by |, and finally concatenation. The ab pattern must be reproduced jointly, thus we must first write (ab)*a to specify the set of strings a,aba,ababa,abababa, and so on.

Wildcard: the wild card symbol matches exactly one occurrence of any single character.

4.5 Boolean expression

A statement using Boolean operators that expresses a condition that is either true re false. An expression consisting solely of Boolean variables and values and Boolean operations, such as and, or, not, implies, etc

1. AND operator.
Conjunction: 0 AND 0 =0, 0AND 1 = 0,1 AND 1= 1.
2. OR operator:
Disjunction: 0 OR 0 = 0, 0 OR 1=1, 1 OR0=1,1 OR 1=1.
3. NOT operator:
Negation: NOT 0 = 1, NOT 1= 0 . Also know as complement.
(1) Of a Boolean , 0 if 1, or 1 if 0.
Of a set A, a set having all the member s which are in the universe, but not in A.
There are various Boolean algebra rules for solving Boolean Expressions

5.STATE MINIMIZATION:

Fewer states may mean fewer state variables .
High level synthesis may generate many redundant states.
Two states are equivalent if they are impossible to distinguish from the output of finite
state machine , i.e. for any input sequence the output s are the same.
Two conditions for two states to be equivalent:
- Outputs must be same in both states.
- Must transition to equivalent states for all input combination.

5.1 Algorithmic Approach To State Minimization

Goal -- identify and combine states that have equivalent behavior.
Equivalent states:
- Same output
- For all input combinations, state transition to same or equivalent states.

5.2 Algorithmic sketch

1. place all the states in one set
2. initially partition set based on output behavior.
3. successively partition, resulting subsets based on next states transitions.
4. Repeat (3) until no further partitioning is required
   States left in the same set are equivalent.
5.polynomial time procedure.

5.3 state minimization example1

Sequence detector for 010,110.

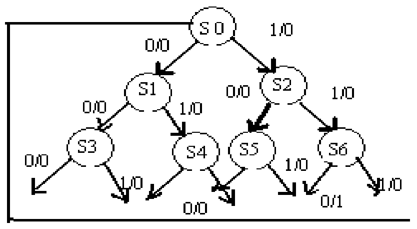


Fig.5.1 State Minimization Example1

Method for successive partition
(S0 S1 S2 S3 S4 S5 S6) S1 is equivalent to S2
(S0 S1 S2 3 S4 S5)(s4 S5) S3 is equivalent to S5
(S0S1S2)(S3S4)(S4S6) S4 is equivalent to S6
(S0)(S3 S5)(S1 S2)(S4 S6)
Minimized FSM:
State minimized sequence detector for 010 or 110
Minimized state chart:

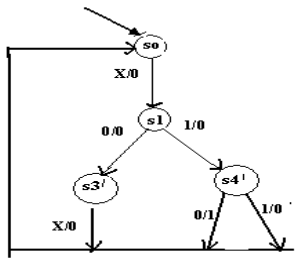


Fig.5.2 State Minimization Example2

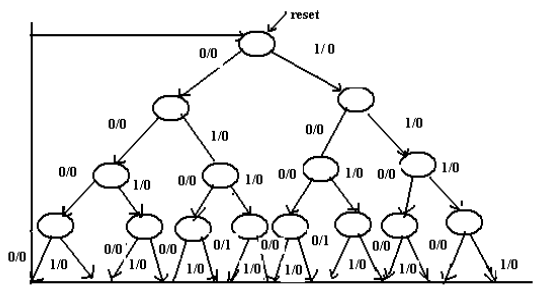


Fig.5.3 State Minimization Example 3
State Transition Table

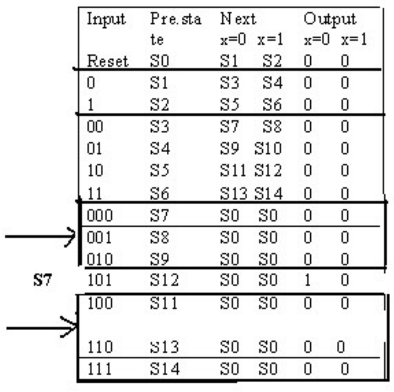
Grouping states with same next state and same output:

Table 5.1 State Transition

S10

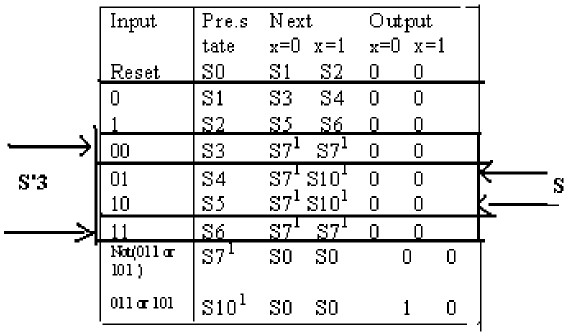
Input	Pre.state	Next x=0	Next x=1	Output x=0	Output x=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S7	S8	0	0
01	S4	S9	S10	0	0
10	S5	S11	S12	0	0
11	S6	S13	S14	0	0
000	S7	S0	S0	0	0
001	S8	S0	S0	0	0
010	S9	S0	S0	0	0
011	S10	S0	S0	1	0
100	S11	S0	S0	0	0
101	S12	S0	S0	1	0
110	S13	S0	S0	0	0
111	S14	S0	S0	0	0

Table 5.2 State Transition



Input	Pre.state	Next x=0	Next x=1	Output x=0	Output x=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S7	S8	0	0
01	S4	S9	S10	0	0
10	S5	S11	S12	0	0
11	S6	S13	S14	0	0
000	S7	S0	S0	0	0
001	S8	S0	S0	0	0
010	S9	S0	S0	0	0
101	S12	S0	S0	1	0
100	S11	S0	S0	0	0
110	S13	S0	S0	0	0
111	S14	S0	S0	0	0

5.3 State Transition Table



Input	Pre.state	Next x=0	Next x=1	Output x=0	Output x=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	$S7^1$	$S7^1$	0	0
01	S4	$S7^1$	$S10^1$	0	0
10	S5	$S7^1$	$S10^1$	0	0
11	S6	$S7^1$	$S7^1$	0	0
Not(011 or 101)	$S7^1$	S0	S0	0	0
011 or 101	$S10^1$	S0	S0	1	0

Fig.5.4 Final Reduced State Diagram

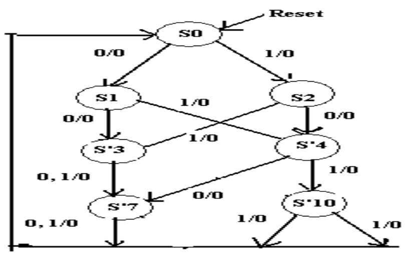


Table 5.4 Reduced State Transition

Input	Pre.state	Next x=0	Next x=1	Output x=0	Output x=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00 or 11	S'3	S'7	S'7	0	0
01 or 10	S'4	S'7	S'10	0	0
Not(011 or 101)	S'7	S0	S0	0	0
011 or 101	S'10	S0	S0	1	0

15 state (min 4FF) got reduced to 7states (min 3FF).

6.GREP COMMAND:

Grep is a command for text processing. There are several different text processing commands available. By combining regular characters and special characters, commonly referred to as meta-characters, with the following rules, several text processing systems, including grep, egrep, sed, awk, and vi, allow you to search on patterns rather than preset expressions. You can use these regular expressions to match patterns in text material that has been replicated.

6.1examples

Consider the following file:

```
{unix prompt 5} cat num.list
1               15      fifteen
2               14      fourteen
3               13      thirteen
4               12      twelve
5               11      eleven
6               10      ten
7               9       nine
8               8       eight
9               7       seven
10              6       six
11              5       five
12              4       four
13              3       three
14              2       two
15              1       one
```

Here are some grep examples using this file. In the first we_ll search for the number 15:

```
{unix  prompt 6} grep '15' num.list
     1           15        fifteen
     15          1         one
```

Now we_ll use the "-c" option to count the number of lines matching the search criterion:

```
{unix prompt 7}grep –c '15' num.list  2
```

Here we_ll be a little more general in out search, selecting for all lines containing the character 1 followed by either of 1,2 or 5:

```
   {unix prompt 8} grep '1[125]' num.list
        1          15            fifteen
         4         12             twelve
        5          11             eleven
       11          5        five
       12          4              four
       15          1              one
```

Now we_ll search for all lines that begin with a space:

```
{unix prompt 5} cat num.list
1               15      fifteen
```

```
2                 14     fourteen
3                 13     thirteen
4                 12     twelve
5                 11     eleven
6                 10     ten
7                  9     nine
8                  8     eight
9                  7     seven
```

Or all lines that don_t begin with a space:

```
 { unix prompt 10} grep '^[^]' num.list
10                 6     six
11                 5     five
12                 4     four
13                 3     three
14                 2     two
15                 1     one
```

The latter could also be done by using the –v option with the original search string, e.g:

```
{UNIX prompt 11} grep –v  '^' num.list
10                 6     six
11                 5     five
12                 4     four
13                 3     three
14                 2     two
15                 1     one
```

This example will search for any instances of t followed be zero or more occurrences of eg:

```
{unix prompt 13}grep 'te*' num.list
1                 15     fifteen
2                 14     fourteen
3                 13     thirteen
4                 12     twelve
6                 10     ten
8                  8     eight
13                 3     three
14                 2     two
```

This example will search for any instances of t followed by one or more occurrences of e:

```
{unix prompt 14} grep 'tee*' num.list
 1            15     fifteen
 2            14     fourteen
 3            13     thirteen
 6            10     ten
```

We can also take out input from a program, rather than a file. Here we report on any lines output but the who program that begin with the letter I.

```
   { unix prompt 15} who| grep '^|'
 Icondron ttypo  Dec 1 02: 41 (Icondron-pc.acs.)
```

7.CONCLUSION:

We assume that the database is a B+ tree in order to identify regular expressions in the text database. The data will be dictionary-style. Starting with leaf nodes, we gather the database's longest strings. The dynamic programming algorithm (DPA) is the distance method we use to modify distance with the associated precision parameter. We employ regular expressions, which are useful for describing text. The three general classes of regular expression operators are KLEENE STAR, UNION, and CONCATENATION. To find regular expressions with the associated meta characters, we compare with the UNIX grep program.

. We apply the same rules that we use to combine regular expressions with Boolean expressions. Specifically, the grep command only offers restricted repetition and does not permit Kleene star operation. These limitations aid in the development of a unified automaton creation technique for regular and Boolean expressions. State minimization reduces superfluous state while maintaining the same level of external input and output needs.

REFERENCES:

[1] ZVI KOHAVI Switching and Finite Automata theory. Tata McGraw-Hill Edition. Second edition 2004.

[2] M. Morris Mano . Digital Design, PHI Publications, second Edition April 2001

[3] John E.Hopcroff, Jeffery D.Ullaman , introduction to Automata Theory, languages computation, narosa publishing house 2001

[4] John Martin , introduction to languages and the theory of computation. Tata McGraw-Hill EDITION, third Edition. 2000

[5] Harry R.Lewies , Christos H.Papadamitriou. Elements of the theory of computation. Second EDITION, Third Edition. 2000. Raghurama Krishna , Johannes gehrke ,Database management systems. Tata McGraw-HILL EDITION, Third Edition.2003

[5] Abraham silberschatz, Herry F .Korth, S.Sudarshan. Database Systems. Tata Mc Graw-Hill edition. Third EDITION 1997

[6] Zhang, C., Tang, X., & Peng, Y. (2024). Enhancing regular expression processing through field-programmable gate array-based multi-character non-deterministic finite automata. *Electronics*, 13(9), 1635. https://doi.org/10.3390/electronics13091635

[7] Peterfreund, L., Freydenberger, D. D., Kimelfeld, B., & Kröll, M. (2019). Complexity bounds for relational algebra over document spanners. *arXiv preprint arXiv:1901.04182*. https://arxiv.org/abs/1901.04182

[8] Tsang, D., & Chawla, S. (2011). An index for regular expression queries: Design and implementation. *arXiv preprint arXiv:1108.1228*. https://arxiv.org/abs/1108.1228

[9] Li, X., Liu, M., Wu, X., & Zhu, S. (2015). Design issues of JPQ: A pattern-based query language for document databases. *arXiv preprint arXiv:1504.03770*. https://arxiv.org/abs/1504.03770

[10] Freydenberger, D. D., & Holldack, M. (2019). Document spanners: From expressive power to decision problems. *ACM Transactions on Database Systems (TODS)*, 44(2), 1-43.

[11] Peterfreund, L., & Kimelfeld, B. (2020). Complexity bounds for relational algebra over regex formulas. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (pp. 235-248).

[12] Freydenberger, D. D., & Peterfreund, L. (2020). A logic for document spanners. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (pp. 115-129).

[13]Doleschal, J., & Neven, F. (2020). A framework for learning document spanners. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (pp. 103-114).

[14] Freydenberger, D. D., & Holldack, M. (2020). A survey on document spanners. *Logic Journal of the IGPL*, 28(5), 877-899.

[15] Peterfreund, L., & Kimelfeld, B. (2021). On the complexity of enumeration and reliability for conjunctive queries over document spanners. *ACM Transactions on Database Systems (TODS)*, 46(1), 1-38.

[16]Doleschal, J., & Neven, F. (2021). Learning document spanners with variable dependencies. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (pp. 1-14).

[17]Freydenberger, D. D., & Peterfreund, L. (2021). A logic for document spanners with string equality tests. *ACM Transactions on Database Systems (TODS)*, 46(2), 1-38.

[18] Doleschal, J., & Neven, F. (2022). Learning document spanners with numeric data. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (pp. 1-14).

[19]Freydenberger, D. D., & Peterfreund, L. (2022). A logic for document spanners with arithmetic expressions. *ACM Transactions on Database Systems (TODS)*, 47(1), 1-38.

[20]Doleschal, J., & Neven, F. (2023). Learning document spanners with complex data types. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (pp. 1-14).