The Efficient way to Identify the Regular Expression in Text Databases

K. Koteswara Rao[1], Srinivasan. Nagaraj[2], Dr GSVP Raju[3]

[1]*Asst Professor, Dept of CSE GMRIT, RAJAM-532127. India*
[2]*Asst. Professor, Dept of CSE ,GMRIT, RAJAM-532127.India*
[3]*Associate Professor, CS&ST Dept., Andhra University, Vizag-530003*

koteswara2003@yahoo.co.in, sri.mtech04@gmail.com ards2003@rediffmail.com

Abstract

Given a list of n strings of length at most k, where l is length of the largest string. The object is to cover the strings by a minimal number of regular expressions r1, r2, r3……,r_m for m≥1,such that

a) Every string in the database satisfies at least one r_i and

b) Any string 'X' of length at most K satisfying $r_1+r_2+r_3+\ldots\ldots+r_m$ is at a distance at most p from a string 'y' in the database ,where 'p' is a prescribed constant parameter.We assume that the database is in the form of B+ tree. We start with leaf nodes and collection all the strings of maximum length in the database For identifying regular expressions in database the thesis aims at developing a procedure similar to that for Boolean formulas (in DNF or CNF), where the function values and don't care term's are specified.

Keywords*: B^+ Tree, Dynamic programming Algorithm, Regular expressions, State minimization*

1. Introduction

Text databases (document database) which consist of large collection of document from various sources, such as news articles, research papers, books, digitals libraries, e-mail message, and web pages. Data stores in most text database are semi structured data in that they are neither completely neither unstructured nor completely structured data in that they are neither completely unstructured nor completely structured. For example, a document may contain a few structured fields such as titled, authors, publications and so on, but contain some largely unstructured text components such as title, publications and so on, but contain some largely unstructured text components such as abstract and contents. We assume the database in the form a B+ tree. Here the database will as the data in the data in the dictionary. The data is contained at the level of the leaves, the leaves can be linked together allowing sequential access to the data once the leaves are reached. They also means that interior nodes contain only referential data, acting as a guide to the information kept leaves. We begin with leaf nodes and collection all the strings of maximum length in the database. The algorithm for calculation of the distance, which in the present work is the edit distance, is based on the dynamic programming method (DPA). A precision parameter is used for compensating for edit errors in the database strings. Regular expressions are the most common general class of formal symbolic representations used to describe strings of characters, such as words or phrases or any arbitrary text that specified by a pattern Regular expression have automata-realization as sequential circuits that make the decision of

transition in to the next state based only on the current symbol being scanned state of the automaton. For binary strings of fixed length, the sequential circuits are implementers of Boolean functions of fixed number of Boolean circuits can be also extended – or at least attempted to extend – for regular expression realization, as well. The objective of the work is to show the transformation needed for an extension of the state minimization rules used the general class of Boolean operation AND, or and not whereas the general class of regular expression operators are UNION, CONCATENATION and KLEEN Star. While the Boolean OR operation and the (regular) set UNION operation are the same, there is no commonness among the remaining. We therefore study a more practical class of regular expression operators (used in Unix utilities)that allow or facilitate a formulation of the sate minimization rules by drawing analogy with the Boolean circuits. The practical class of regular expressions includes certain special operators that are amenable for state based rules for combining terms. The practical regular expressions are those allowed or handled by the grep command in UNIX. Given a list of files or standard input to read, the grep command line utility searches for lines of text that match one or many regular expression and outputs the matching lines or the lines that do not match (with –v option) depending on the specified options. The grep commend allows construction of regular expression with various meta characters such as +, . , *,?,\{n,m}\, substring, and finite extent repeaters for most effective description of patterns in the queries . The patterns allowed using the finite extent repeaters are as follows:

. 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] match any character in the enclosed

[a-d] match any character in the enclosed range

[^exp] match any character not in the following expression

^abc the regular expression must state at the beginning of the line(Anchor).

abc$ the regular expression must end at the end of the line (Anchor) treat the next character literally. This is normally used to escape ht meaning of special characters such as “.” and “*”.

\{n,m|} match the regular expression preceding this a minimum number of n times and a maximum of m times (0 through 255 are allowed for n and m). The \{and \} sets should be thought of as single operators. In this case the \ preceding the bracket does not escape it special meaning , but rather turns on a new one.

/<abc\> will match the enclosed regular expression as long as it is separate word. Word boundaries are defined as beginning with a <newline> or anything except a letter, digit or underscore(_) or ending with the same or a end-of-line character. Again \<and \> sets should be thought of as single operations.

abc\) saves the enclosed pattern in a buffer. Up to nine patterns can be saved for each for each line. You can reference these latter with the \n character set. Again the \(and \)set should be thought of as single operators.

\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.

The grep utility is used to search for generalized regular expressions occurring in UNIX file. Regular expressions such as those shown above, are best specified in apostrophes(or single quotes) when specified in the grep utility. The egrep utility provides searching capability using an extended set of meta-characters. The syntax of the grep utility, some of the available options, and a few examples are show below.

Syntax:

grep[options]regexp[file[s]]

common options:

-I	ignore case
-c	report only a count of the number of lines containing matches , not the matches themselves
-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 paper aim is to identifying R.E in text data bases.. Where we study of what is text database. How to deal with regular expressions. Working of the regular expression as Boolean expressions. This thesis has various chapters such as B+ tree, DPA for editing distance, Regular expressions, Boolean operators with regular expression, state minimization and grep .In the chapter B+ tree deal with the information regarding what is B+ tree, construction of B+ tree, where the data is stored in B+ tree In this chapter Dynamic

programming method is for editing distance which is the present work for editing distance .In the chapter Regular expression we will discuss the information regarding what is regular expression, what is Boolean expression. Here we apply the rules for solving to regular expressions as we solve for Boolean expressions. In the chapter State Minimization is the circuit minimization methods used in the Boolean circuits can also be extended for regular expression realizations. In the chapter grep will have the information what is grep , the practical regular expressions are those allowed or handled by the grep command in Unix. Give a list of files or standard input to read, the grep command line utility searches for lines of text that match one or many regular expressions, and outputs the matching lines or the lines that do not match (with-v option) depending on the specified options.At last we will be having conclusion which includes bibliography and web reference.

1.2 Flow of the Paper

Step1: Assume the database is in the form of a B+ tree. Database will be as the data in the dictionary. We start with leaf nodes and collection all t he strings of maximum length in the database.

Step 2: The distance method we follow is dynamic algorithm (DPA) to edit programming distance with the related precision parameter

Step 3: What regular expression can be used for text description? The regular expression specification allowed in the egrep command in UNIX are used, as these are built up of more realistic operators than those used in the formal definitions (though both are equivalent)

Step 4: Rules for combining Boolean Expressions are applied for combining Regular Expressions.

Step 5: State minimization algorithm for reducing redundant states

Step 6:Report and collect feedback if needed.

2. B^+ Tree

We assume the database is in the form of a B+ tree . we start with leaf nodes and collection all the strings of maximum length in the database. In a B+ tree no data resides in the interior nodes of the tree. Since all the data contained at the level of the leaves, the leaves can be linked together allowing sequential access to the data once the leaves can be linked together allowing sequential access to the data once the leaves are reached T this also means that interior nodes contain only referential data , acting as a guide to the information kept at leaves. B+tree is a dynamic index structure that adjusts gracefully to inserts and deletes. It is a balanced tree. 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

Insert/delete at log N F caste, keep tree height-balanced.(f=fan-out, N=#leaf pages). Minimum 50% occupancy(except for root). Each node contains d<=m<=2d entries. The parameter d is called the order of the tree. Supports equality and range-searches efficiently. A B+ -tree is a data structure to store vast amount of information. Typically B+ -trees are used to store amounts of data that will not fit in main system memory. To see this, secondary storage (usually disk) is used to store the leaf nodes of the tree. Only the internal nodes of the tree are stored in computer memory. In a B+-tree the leaf nodes are the only ones that actually store data items. All other nodes are called index nodes or i-nodes and simply store "guide" values ,which allow us to traverse the tree structure from the root down and arrive at the leaf node containing the data item we seek. Because disk I/O is very slow in comparison to memory access these leaf nodes store more than one data item each . In fact, the data structure will perform best within the size of the leaf nodes closely approximates the size of a disk sector under most operating systems . thus , when we search a B+ -tree (by traversing from the root node down to the proper dat node) we still must read that data node from the disk and search its contents. Another way to improve the speed of a query operation is to keep a memory cache of recently read nodes. The ancestor of the B+-tree is a structure known as a B –trees un which data items can be stored in any node on the tree. A more complicated and slightly more robust variant of the B-tree is called as B+-tree.

2.3 Example of a B+ -tree

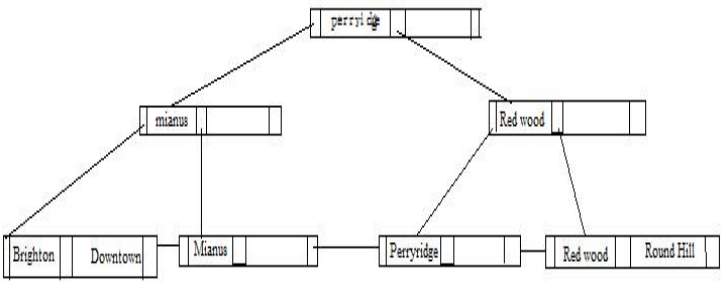

Figure 1. B+ tree Index

B+ tree index is a balanced tree in which the internal nodes (the top two levels) direct the search and the leaf nodes contain data entries. Searching for a record requires just a traversal from the root to the appropriate leaf node. The length of the path from the root to a leaf is called height of the tree (usually 2 or 3). To search for entry 9*, we follow the left most child pointer from the root (as 9<10). Then at level two we follow the right child pointer (as 9>6). Once at the leaf node, data entries can be found sequentially. Leaf nodes are inter-connected which makes it suitable for range queries.

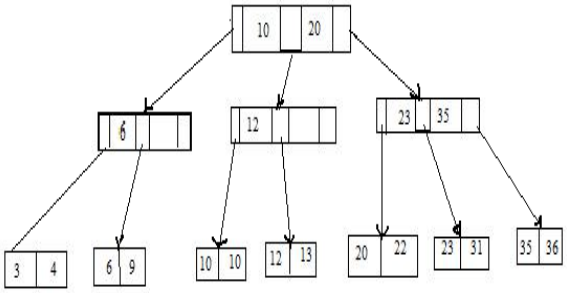

Figure 2. The records can be accessed via an index or in insertion order.

3. Dynamic Programming Algorthim (Dpa) To Edit Distance

The distance method we follow is dynamic programming algorithm (DPA) to edit distance with the related precision parameter. The words 'computer' and 'commuter' are very similar , and a change of just one letter, p>m will change the first word into the second. The word 'sport can be changed into 'sort' by the deletion of the 'p' , or equivalently, 'sort' can be changed into 'sport' by the insertion of 'p'. The edit distance of two strings, s1 ands2, is defined as the minimum number of point mutations required to change s1into s2 where a point mutation is one of

1. Change a letter,
2. Insert a letter or
3. Delete a letter

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

3.1 Algorithm

```
editdistance(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]
```

3.1Example

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 1.

Table 1. Solution

	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	2	1

So the distance (sport , sort) is 1.

4. Regular Expressions

A regular expression is notation for specifying a set of strings e.g. the set of all valid email addresses or the set of all binary strings with an even number of 1's. since the set might contain infinitely many members, we can't simply enumerate them .For identifying regular expressions in database we are developing a procedure similar to that for Boolean formula's (DNF and CNF) where function values and don't care are specified. Regular expression: language accepted by finite automate are easily described by simple expressions. Regular expressions describes strings of characters (words or phrases or any arbitrary text). It is a set

of characters that specify a pattern. We assume AND & OR operations besides unions, concatenation, kleene closure(or closure).

The general classes of regular expression operators are UNION,KLEEN STAR and CONCATENATION

A formal recursive definition of regular expressions over $\sum$ as follows

1.Any terminal symbol(i.e. an element of $\sum$ n), Λ and Ø are regular expressions. When we view an in $\sum$ as a regular expression, we denote it by a.

2.The union of two regular expresions R1 and R2 written as R1+R2 is also a regular expression

3.The concatenation of two regular expressions R1and R2, written as R1,R2 is also a regular expression.

4.The Iteration (or closure) of a regular expression R, written as R* is also a regular expression.

5.If r is regular expression expression , then (R) is also a regular expression.

6.The regular expression over $\sum$ are precisely those obtained recursively by the application of the rules 1-5 once or several times.

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+Qrand R(P+Q) =RP+RQ

If R=Q+RP by Ardens theorem r=QP*.

4.2 Regular Expressions Notation

Languages {a} ⟺ a/
a U b⟺ {a,b)//set union.
a*⟺ {a}* //Kleene star
a+⟺ {a}+ //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.

Table 2

Operation	Regular expression	Yes	No
Concatenation	Aabaab	Aabaab	Every other string
Logical OR(alternation)	aa|baab	Aa baab	Every other string
Replication(Kleene closure)	ab*a	Aa aba abba	E ab ababa
Grouping	a(a|b)aab	Aaaab Abaab	Every other string
Wildcard	a….a	Abba abaa	Aa Aaaaa

Concatenation: the simplest type of regular expression is formed by concatenating a bunch of symbols together, one after the other like aabaab. This regular expression matches only the single string aabaab. We can perform simple spell checking by using the concatenation operation. For example, we could form the regular expression neither and then for each word in a dictionary would match, and we would conclude that neither is misspelled. Logical OR: the logical OR operator enables us to choose form one of several possibilities. For example, the regular expression aa| baab matches exactly two strings aa and baab. Many spam filters(e.g. spam Assaain) work by searching for a long list of common spamming terms. They might form a regular expression likeAMAZING| GUARANTEE|viagra. The logical OR operator enables us to specify many strings with a single regular expression. For example, if our phone number is 734-8527, we might like to know whether it spells out any word on the phonepad (2 = abc, 3 = def, 4 = ghi, 5 = jkl, 6 = mno, 7=prs, 8= tuv, 9 = wxy);

The following regular expression specifies all of the 3^7 possible the combinations (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 only english word that matches is the word regular (replace this example with decoding an IM message that use the "phone code")Replication : the replication operator enables us to specify infinitely may possibilities. For example, the regular expression ab*a matches a,aba,abba,a,bbb, an so forth. Note that 0 replications of b are permitted. Grouping : the grouping operator enables us to specify precedence to the various operators. The * operator has the highest precedence, then |,then concatenation. If we want to specify the set of strings a,aba,ababa,abababa, and so forth, we nust write (ab)*a to indicate that the ab pattern must be replicated together.

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.

(2) 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

Table 3. Sequence detector for 010,110.

Input	Present	Next state X=0	X=1	Output X=0	X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0

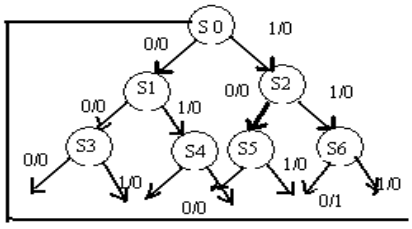

Figure 1. Method for Successive Partition

(**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:

Table 4. State minimized sequence detector for 010 or 110

Input	Present	Next X=0 X=1	Output X=0 X=1
Reset	S0	$S1^1$ $S1^1$	0 0
0+1	$S1^1$	$S3^1$ $S4^1$	0 0
X0	$S3^1$	S0 S0	0 0
X1	$S4^1$	S0 S0	1 0

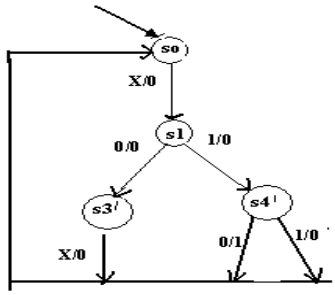

Figure 2. Minimized state chart

5.4 state Minimization Example 2

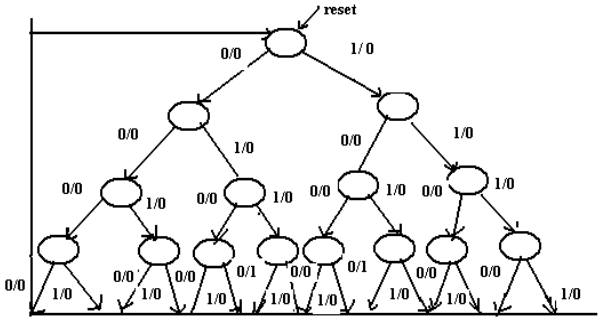

Figure 3. State transition table

	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	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
S10	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

Figure 4. Grouping states with same next state and same output

	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
S7	101	S12	S0	S0	1	0
	100	S11	S0	S0	0	0
	110	S13	S0	S0	0	0
	111	S14	S0	S0	0	0

Figure 5. Iterate the row matching algorithm

	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	
S'3	01	S4	$S7^1$	$S10^1$	0	0	S'4
	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	

Figure 6. Iterate one last time

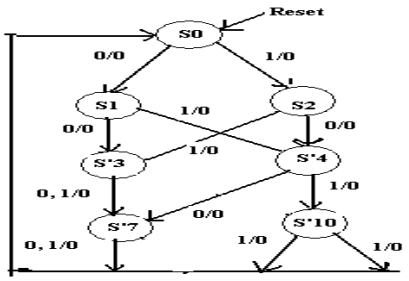

Figure 7. Final reduced state

Input	Pre.s tate	Next x=0	x=1	Output x=0	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

Figure 8. 15 state (min 4FF) got reduced to 7states (min 3FF).

6. GREP COMMAND

Grep is a text processing command. There are various text processing commands are present. Some text processing programs, such as **grep, egrep, sed, awk and vi,** let you search on patterns instead of fixed expression by combining normal characters and special characters, also known as **meta-characters,** with the rules below. With these regular expressions you can do pattern matching on text data.

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 e:

```
{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 bu 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

For identifying regular expressions in text database, we assume database in the form of a B+ tree. Data will be as of dictionary. We begin with leaf nodes and collect the strings of maximum length in the database. The distance method we follow is dynamic programming algorithm (DPA) to edit distance with the related precision parameter. We make use of regular expressions that can be used for text description. General class of regular expression operators are UNION, CONCATENATION and KLEENE STAR. We make a comparison with grep command in UNIX for identifying of regular expressions with the related meta characters. We make use rules for combining Boolean Expressions are applied for combining regular Expressions. In particular , the grep command does not allow Kleene star operation but provides only bounded repetition. These restrictions help formulate a unified automata construction method for Boolean expressions and regular expressions. State Minimization method for reducing redundant state, keeping the external input and output requirements unchanged.

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] G.S.S Bhishma Rao, discrete structures and Grap theory, step Publications, Second Edition Sept 2002

Authors

Mr K Koteswara rao received his B.Tech in CSE from Nagarjuna University in 2004,M.Tech in CSE in 2006 from JNTU ,presently pursuing PhD at JNTU Kakinada. since 2008 working as Sr Asst Professor in CSE Department at GMR Institute Of Technology, His research interest includes software Engineering with soft computing, Project Management, object Oriented systems development, data engineering

Mr Srinivasan Nagaraj received his B.Tech in CSE from Nagarjuna University, ,M.Tech in both CSE and IT in 2004and 2010 from Nagarjuna University ,presently pursuing PhD at JNTU Kakinada. since 2006 working as Sr Asst Professor in CSE Department at GMR Institute Of Technology, His research interest includes network Security with soft computing, Project Management, object Oriented systems development, data engineering