**Aim:**

**Theory:**

**Finite automata**

A ***finite automaton*** *M* is a 5-tuple (*Q, q*0, *A*, http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/sum14.gif, http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/delta12.gif), where

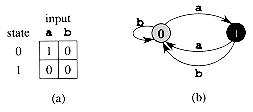
http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/dot12.gif*Q* is a finite set of **states**,

http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/dot12.gif*q*0 http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/memof12.gif*Q* is the **start state**,

http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/dot12.gif*A* http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/rgtubar.gif*Q* is a distinguished set of **accepting states**,

http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/dot12.gifhttp://staff.ustc.edu.cn/~csli/graduate/algorithms/images/sum14.gifis a finite **input alphabet**,

http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/dot12.gifhttp://staff.ustc.edu.cn/~csli/graduate/algorithms/images/delta12.gifis a function from *Q* X http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/sum14.gifinto *Q*, called the **transition function** of *M*.



**Figure 34.5 A simple two-state finite automaton with state set Q = {0, 1}, start state q0 = 0, and input alphabet http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/sum14.gif= {a, b}. (a) A tabular representation of the transition function http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/delta12.gif. (b) An equivalent state-transition diagram. State 1 is the only accepting state (shown blackened). Directed edges represent transitions. For example, the edge from state 1 to state 0 labeled b indicates http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/delta12.gif(1, b) = 0. This automaton accepts those strings that end in an odd number of a's. More precisely, a string x is accepted if and only if x = yz, where y = http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/epsilon.gifor y ends with a b, and z = ak, where k is odd. For example, the sequence of states this automaton enters for input abaaa (including the start state) is http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/lftwdchv.gif0, 1, 0, 1, 0, 1http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/wdrtchv.gif, and so it accepts this input. For input abbaa, the sequence of states is http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/lftwdchv.gif0, 1, 0, 0, 1, 0http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/wdrtchv.gif, and so it rejects this input.**

The finite automaton begins in state *q*0 and reads the characters of its input string one at a time. If the automaton is in state *q* and reads input character *a*, it moves ("makes a transition") from state *q* to state http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/delta12.gif(*q, a*). Whenever its current state *q* is a member of *A*, the machine *M* is said to have ***accepted*** the string read so far. An input that is not accepted is said to be ***rejected***. Figure 34.5 illustrates these definitions with a simple two-state automaton.

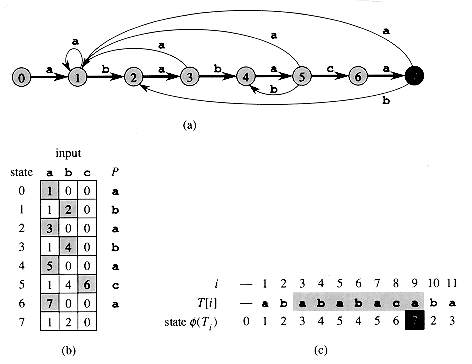
A finite automaton *M* induces a function ø, called the ***final-state function***, from http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/sum14.gif\* to *Q* such that ø(*w*) is the state *M* ends up in after scanning the string *w*. Thus, *M* accepts a string *w* if and only if ø(*w*) http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/memof12.gif*A*. The function ø is defined by the recursive relation

ø(http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/epsilon.gif*) =* q*http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/omicr.gif* ,

ø(*wa*) = http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/delta12.gif*(ø(*w*),* a*) for* w *http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/memof12.gif http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/sum14.gif\*,* a *http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/memof12.gif http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/sum14.gif.*

**String-matching automata**

There is a string-matching automaton for every pattern *P*; this automaton must be constructed from the pattern in a preprocessing step before it can be used to search the text string. Figure 34.6 illustrates this construction for the pattern *P* = ababaca. From now on, we shall assume that *P* is a given fixed pattern string; for brevity, we shall not indicate the dependence upon *P* in our notation.



**Figure 34.6 (a) A state-transition diagram for the string-matching automaton that accepts all strings ending in the string ababaca. State 0 is the start state, and state 7 (shown blackened) is the only accepting state. A directed edge from state i to state j labeled a represents http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/delta12.gif(i,a) = j. The right-going edges forming the "spine" of the automaton, shown heavy in the figure, correspond to successful matches between pattern and input characters. The left-going edges correspond to failing matches. Some edges corresponding to failing matches are not shown; by convention, if a state i has no outgoing edge labeled a for some a http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/memof12.gifhttp://staff.ustc.edu.cn/~csli/graduate/algorithms/images/sum14.gif, then http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/delta12.gif(i,a) = 0. (b) The corresponding transition function http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/delta12.gif, and the pattern string P = ababaca. The entries corresponding to successful matches between pattern and input characters are shown shaded. (c) The operation of the automaton on the text T = abababacaba. Under each text character T[i] is given the state ø(Ti) the automaton is in after processing the prefix Ti. One occurrence of the pattern is found, ending in position 9.**

**Program:**

#include<stdio.h>

#include<string.h>

#define MAX 100

void makesufix(int len,char input[],char sufix[])

{

int size=strlen(input);size--;

while(len!=-1)

sufix[len--]=input[size--];

// printf("sufix is %s\n\n",sufix );

}

void makeprefix(int len,char input[],char prefix[])

{

int i=0;

for(i=0;i<=len;i++)

prefix[i]=input[i];

//printf("prefix is %s\n\n",prefix );

}

int getstate(char newstring[],char substring[])

{

printf("String is : %s\n",substring);

int i,len=strlen(substring);

len--;

for(i=len;i>=0;i--)

{

// printf("calling sufix function\n");

char sufix[i+2];sufix[i+1]='\0';

makesufix(i,newstring,sufix);

// printf("calling prefix function\n");

char prefix[i+2];prefix[i+1]='\0';

makeprefix(i,substring,prefix);

if(!strcmp(prefix,sufix))

return strlen(prefix);

}

return 0;

}

void addletter(char newstring[],int type,int i)

{

switch(type)

{

case 0:newstring[i]='a';break;

case 1:newstring[i]='b';break;

case 2:newstring[i]='c';break;

}

}

void constructTable(int table[][3], char pattern[])

{

int height=strlen(pattern);

int i=0,j=0;

for(i=0;i<height;i++)

{

char substring[i+2];//one more for end character

int c=0;

while (c <= i) {

substring[c] = pattern[c];

c++;

}

substring[c] = '\0';

printf("Substring is %s\n",substring);

for(j=0;j<3;j++)

{

char newstring[i+2];

strcpy(newstring,substring);

addletter(newstring,j,i);//overwriting ith position with column head;

printf("newstring is %s\n",newstring);

table[i][j]=getstate(newstring,substring);

printf("table at (%d,%d) is%d\n",i,j,table[i][j]);

}

}

for(j=0;j<3;j++)

{

char newstring[strlen(pattern)+2];

strcpy(newstring,pattern);

addletter(newstring,j,height);

newstring[height+1]='\0';

printf("newstring is %s\n",newstring);

table[height][j]=getstate(newstring,pattern);

printf("table at (%d,%d) is%d\n",height,j,table[i][j]);

}

printf("printing table\n");

printf("a\tb\tc\n");

for(i=0;i<=height;i++)

{

for(j=0;j<3;j++)

printf("%d\t",table[i][j]);

printf("\n");

}

}

void main()

{

char pattern[10];

char inputString[50];

printf("Enter pattern made of 'a','b','c' only\n");

//fgets(pattern,MAX,stdin);

gets(pattern);

printf("pattern lenght is %d\n",strlen(pattern));

printf("Enter String \n");

//fgets(inputString,MAX,stdin);

gets(inputString);

int len=strlen(pattern);

len++;

int table[len][3];

constructTable(table,pattern);len--;

//now pattern matching.

int size=strlen(inputString);

int nowstate=0;int occurence=0;

int i;

for(i=0;i<size;i++)

{

char ch=inputString[i];

switch(ch)

{

case 'a': nowstate=table[nowstate][0];break;

case 'b': nowstate=table[nowstate][1];break;

case 'c': nowstate=table[nowstate][2];break;

default: nowstate=0;

}

if(nowstate==len)

{

occurence++;

printf("pattern found at index %d \n",(i-len+1));

}

}

printf("pattern \"%s\" occured %d in \"%s\"\n",pattern,occurence,inputString);

}

**Output:**

Enter pattern made of 'a','b','c' only

aaba

pattern lenght is 4

Enter String

aabaacaadaabaaba

printing table

a b c

1 0 0

2 0 0

2 3 0

4 0 0

2 0 0

pattern found at index 0

pattern found at index 9

pattern found at index 12

pattern "aaba" occured 3 in "aabaacaadaabaaba"

**Conclusion:**

Time complexity of string matching with finite automata is O(n) which is much better than naive string matching( having time complexity of O(mn)).