EX1 – String Pattern Matching Algorithm

Background:

String pattern matching algorithms are very useful for several purposes, like simple search for a pattern in a text or looking for attacks with predefined signatures.

We will implement a dictionary-matching algorithm that locates elements of a finite set of strings (the "dictionary") within an input text. It matches all patterns "at once", so the complexity of the algorithm is linear in the length of the patterns plus the length of the searched text plus the number of output matches.

Since all matches are found, there can be a quadratic number of matches if every substring matches (e.g. dictionary = `a`, `aa`, `aaa`, `aaaa` and input string is `aaaa`).

The algorithm matches multiple patterns simultaneously, by first constructing a Deterministic Finite Automaton (DFA) representing the patterns set, and then, with this DFA on its disposal, processing the text in a single pas.

Specifically, the DFA construction is done in two phases. First, the algorithm builds a DFA of the patterns set: All the patterns are added from the root as chains, where each state corresponds to one symbol. When patterns share a common prefix, they also share the corresponding set of states in the DFA.

The edges of the first phase are called *forward transitions*. Thses transitions are represented by the Goto function (black arrows), g(state,symbol)→state.

Each state corresponds to a label; the label is the sequence of characters on its goto path from the root. The length of the label is the depth of the state.

The DFA has Regular states and accepting states.

The edges of the second phase are called *failure transitions*. These edges deal with situations where, given an input symbol *b* and a state *s*, there is no forward transition from *s* using *b*. In such a case, the DFA should follow the failure transition to some state *s'* and take a forward transition from there. This process is repeated until a forward transition is found.

The following DFA was constructed for patterns set {E, BE, BD, BCD, CDBCAB, BCAA}. Solid black edges are forward transitions (goto function) while red scattered edges are failure transitions.

L(s) - the *label* of a state *s*, is the concatenation of symbols along the path from the root to *s*.

d(s) - the depth of a state *s*, is the length of the label *L(s)*.

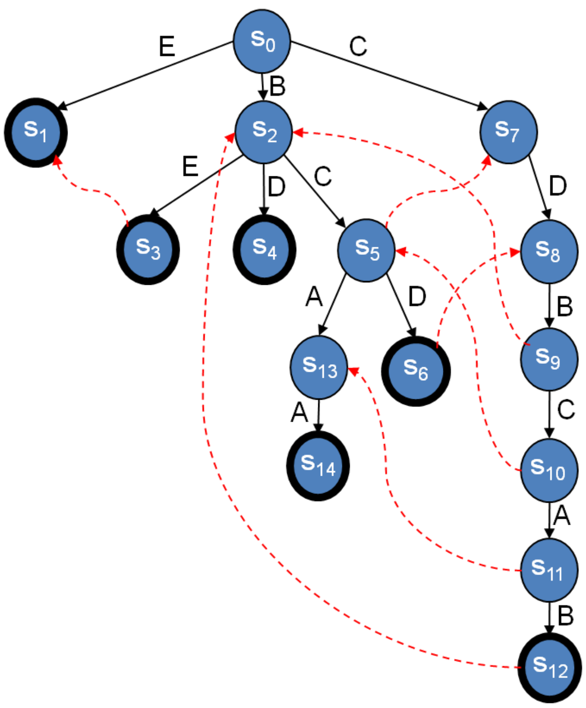
The failure transition from *s* is always to a state *s'*, whose label *L(s')* is the longest suffix of *L(s)* among all other DFA states.

The DFA is traversed starting from root. When the traversal goes through an *accepting state*, it indicates that some patterns are a suffix of the input; one of these patterns always corresponds to the label of the accepting state.

Finally, we denote by *scan(s,b)*, the AC procedure when reading input symbol *b* while in state *s*; namely, transiting to a new state *s'* after traversing failure transitions and a forward transition as necessary, and reporting matched patterns in case *s'.output != emptyset*. *scan(s,b)* returns the new state *s'* as an output.

Patterns Set:

E, BE, BD, BCD, CDBCAB, BCAA



Program Description and What You Need to Do:

The algorithm operation:

Let $K=\{y_1, y_2, \ldots, y_k\}$ be a finite set of strings which we shall call keywords and let x be an arbitrary string which we shall call the text string.

The behavior of the pattern matching machine is dictated by three functions:

1. a *goto* function g - maps a pair consisting of a state and an input symbol into a state or the message fail
2. a *failure* function *f* - maps a state into a state, and is consulted whenever the goto function reports fail
3. an *output* function *output* - associating a set of keyword (possibly empty) with every state

Constructing the FSM:

1. Determine the states and the goto function
2. Compute the failure function
3. Output function start at first step, complete at the second step

1. **Determine the states and the goto function**

Input: set of keywords $K=\{y_1, y_2, \ldots, y_k\}$

Output: the goto function *g*

```
Begine
     newstate = 0;
     For i=1 until k do enter(yi)
     For all b such that g(0,b)= fail, do g(0,b)=0
     //the root has no failure transition
End

Procedure enter(b1,b2,…bm):
Begin
     state=0;  //root
     j=1
     while g(state,bj) != fail do begin
     //there is an edge for that symbol in the FSM
          state = g(state,bj)
          j = j + 1
     end
```

```
    for p=j until m do begin
    // creating states for the rest of the symbols
        newstate = newstate+1
        g(state, b_p) = newstate
        state = newstate
    end
    output(state) = {b_1,b_2,…b_m}
end
```

Compute the failure function

Input: goto function g, and output function output.

Output: failure function f and updated output function.

```
Begin
    Queue ← empty
    //insert to the queue all the states with depth 1
    For each b such that g(0,b)=s!=0 do
    //the new state s is not the root
    Begin
        Queue.enqueue(s)
        f(s)=0
        //failure transitions of states in depth 1 is
        //the root
    End
    While queue != empty do
    Begin
        //Let r be the next state in queue
        r = Queue.dequeue()
        For each b such that g(r,b)=s!=fail do
        //there is goto transition
        Begin
            Queue.enqueue(s)
            state = f(r)
            While(g(state,b)=fail) do state=f(state)
            f(s)=g(state,b)
            output(s)=output(s) U output(f(s))
        End
    End
End
```

Searching for patterns in a given text.

```
Input: The DFA, text,
Output: positions of all matchs, Ptterns and output
states.

Begin
     List matched_list;
     State = 0;
     For j=0 to length do
       While g(state,string[j])=FAIL
          State=f(state)
       State = g(state, string[j];
       If(state.output.size > 0)
          Add matched patterns from state.output to
          matched_list.
     Return matched_list
end
```

Printing to stdout

While constructing the tree:

Each time you create new state in the FSM print:

"Allocating state i\n"

Each time you create an edge for the goto function from state i to state j in the FSM print:

"I -> a -> j\n", where 'a' is the character that generates this edge.

Each time you create an edge for the failure function from state i with depth > 1 to any state j in the FSM print:

"Setting f(i) = j\n"

Algorithm interface

You should use pattern_matcing.h and slist.h and implement the functions that are defined in pattern_matcing.c. **YOU SHOULD NOT CHANGE THESE FILES.**

Test you program

In order to test your program, create another file with a main and use the functions in pattern_matcing to construct the FSM and search for patterns in a text.

With the patterns: abc, bca, cab and acb, and the text "xyzabcabde", the output should be:

```
Allocating state 1
0 -> a -> 1
Allocating state 2
1 -> b -> 2
Allocating state 3
2 -> c -> 3
Allocating state 4
0 -> b -> 4
Allocating state 5
4 -> c -> 5
Allocating state 6
5 -> a -> 6
Allocating state 7
0 -> c -> 7
Allocating state 8
7 -> a -> 8
Allocating state 9
8 -> b -> 9
Allocating state 10
1 -> c -> 10
Allocating state 11
10 -> b -> 11
Setting f(2) = 4
Setting f(10) = 7
Setting f(5) = 7
Setting f(8) = 1
Setting f(3) = 5
Setting f(11) = 4
Setting f(6) = 8
Setting f(9) = 2
```

Error handling

Return with -1 on any fatal error.
Before returning “Clean” allocated memory as much as you can.

What you should submit
ex1.tar containing README, pattern_matching.c and slist.c.

In order to create the tar file use:
tar –cvf ex1.tar README slist.c pattern_matching.c