ide pemecahan permasalahan SPOJ

[www.spoj.com/problems/CTSTRING](http://www.spoj.com/problems/CTSTRING)

1. **Parsing regex string kedalam bentuk NFA (dilakukan seperti evaluasi operasi aritmatika** **dengan menggunakan stack)**

karena dari 3 operator ada operator yang tidak memiliki karakter (concatenate), sehingga concatenate dalam kode yang ditulis disimbolkan dengan ‘+’

**Contoh** regex ((ab)|a) akan dirubah menjadi ((a+b)|a)

Pseudocode:

string regex

stack op

stack nfa\_op

for i=0 to length(regex)

if(regex[i]==’(’) continue loop

else if(regex[i]==’a’ or regex[i]==’b’)

tmp = create 2 state with regex[i] as transition character

push tmp to nfa\_op;

else if(regex[i]==’)’)

operator = op.top()

op.pop()

if(operator == ‘|’)

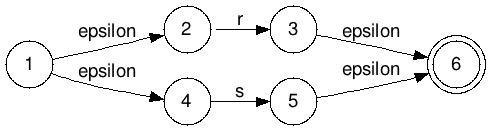
s = nfa\_op.top()

nfa\_op.pop()

r = nfa\_op.top()

nfa\_op.pop()

create this model



else if(operator == ‘+’)

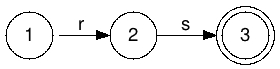
s = nfa\_op.top()

nfa\_op.pop()

r = nfa\_op.top()

nfa\_op.pop()

create this model

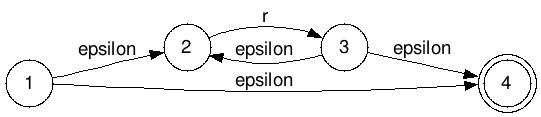


else if(operator == ‘\*’)

r = nfa\_op.top()

nfa\_op.pop()

create this model



else

push regex[i] to op

endif

endfor

NFA sudah terbentuk dengan menggunakan parsing seperti diatas.

**Contoh:** untuk regex ((ab)|a) akan terbentuk NFA dengan tahapan sebagai Berikut

Ubah ((ab)|a) menjadi ((a+b)|a)

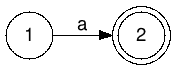
* ‘(‘ -> skip
* ‘(‘ -> skip
* ‘a’ -> push ‘a’ ke stack states
* ‘+’ -> push ‘+’ ke stack ops
* ‘b’ -> push ‘b’ ke stack states
* ‘)’ -> pop stack ops teratas ke op

Cek op yang ternyata adalah ‘+’, maka bentuk NFA concatenate

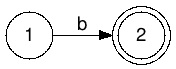
Karena concatenate membutuhkan 2 operand maka:

pop stack states teratas ke tmp2, pop stack states teratas ke tmp1 kemudian

tmp1 adalah



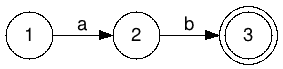
tmp2 adalah



-> tmp = concatenate (tmp1,tmp2)

-> push tmp to stack states

* Hasil NFA Sementara



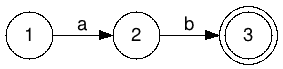
* ‘|’ push ‘|’ ke stack ops
* ‘a’ push ‘a’ ke stack states
* ‘)’ -> pop stack ops teratas ke op

Cek op yang ternyata adalah ‘|’, bentuk NFA union

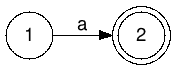
Karena concatenate membutuhkan 2 operand maka:

pop stack states teratas ke tmp2, pop stack states teratas ke tmp1 kemudian

tmp1 adalah



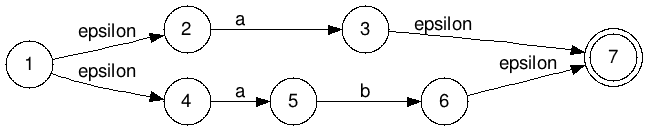
tmp2 adalah



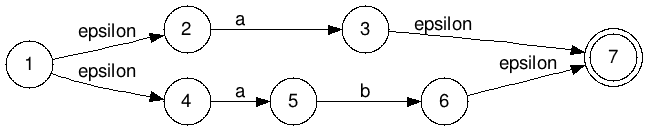
-> tmp = union (tmp1,tmp2)

-> push tmp to stack states

* Hasil NFA Sementara



* Seluruh string regex sudah selesai diiterasi, maka pop stack state teratas yang merupakan hasil NFA.
* NFA terakhir



1. **Bentuk DFA dari NFA yang sudah selesai dibuat**

Ada dua fungsi utama yang digunakan untuk membentuk DFA dari NFA yang sudah ada

* Fungsi *move*(array nfa\_node, char karakter)

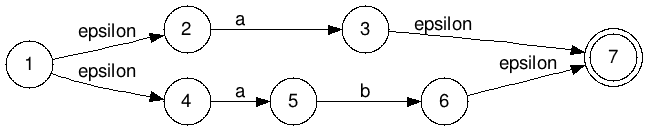
Fungsi ini memiliki return value array node yang merupakan kumpulan node yang dapat dikunjungi dari kumpulan nfa\_node dengan input karakter tertentu.

* Fungsi *eps\_move*(array nfa\_node)  
  Fungsi ini memiliki return value array node yang dapat dikunjungi dari kumpulan nfa\_node dengan perpindahan tanpa input (epsilon)

Tahap tahap pembentukan DFA dari NFA yang sudah ada

1. Buat node awal untuk DFA dengan melakukan *eps\_move* node awal dari NFA
   * node awal dari DFA merupakan array node dari NFA yang dapat dikunjungi dengan transisi epsilon dari node awal dari NFA
2. Untuk setiap node baru yang terbentuk lakukan step berikut untuk semua input karakter
   1. lakukan *move* dengan parameter node DFA dan input karakter tertentu untuk mendapatkan array node **tmp**
   2. Lakukan *eps\_move* pada array node **tmp** untuk mendapatkan node baru untuk DFA (node baru ini merupakan satu atau lebih node dari NFA yang dikombinasikan menjadi satu node DFA) apabila **tmp** merupakan array kosong maka tidak ada node DFA baru yang dihasilkan
3. Apabila masih ada node baru yang terbentuk pada step 2, lakukan kembali step 2 sampai tidak ada node baru yang terbentuk. Apabila tidak ada lanjut ke step 4.
4. DFA selesai dibentuk, dan node akhir (accepting node) merupakan node DFA yang memuat node akhir dari NFA.

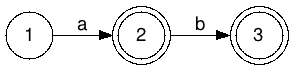
**Contoh** untuk NFA berikut



1. eps\_move({1}) menghasilkan {1,2,4}
2. * move({1,2,4}, ‘a’) menghasilkan {3,5}
   * eps\_move({3,5}) menghasilkan {3,5,7}
   * {1,2,4} -> {3,5,7} dengan input ‘a’
   * move({1,2,4}, ‘b’) menghasilkan {}
   * eps\_move({}) tidak menghasilkan apa-apa
3. * move({3,5,7}, ‘a’) menghasilkan {}
   * eps\_move({}) tidak menghasilkan apa-apa
   * move({3,5,7}, ‘b’) menghasilkan{6}
   * eps\_move({6}) menghasilkan {6,7}
   * {3,5,7} -> {6,7} dengan input ‘b’
4. * move({6,7}, ‘a’) menghasilkan {}
   * eps\_move({}) tidak menghasilkan apa-apa
   * move({6,7}, ‘b’) menghasilkan {}
   * eps\_move({}) tidak menghasilkan apa-apa
5. Tidak ada node baru terbentuk maka konstruksi DFA selesai
6. Hasil DFA adalah

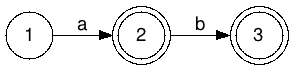


atau dapat disederhanakan menjadi



1. **Hitung string dengan panjang N yang bisa diterima oleh DFA yang dihasilkan**

Buat ajacency matrix dari DFA



Menghasilkan

Dimana merupakan banyak jalan dari i ke j dengan panjang 1, untuk mencari banyak cara dari i ke j, atau bisa digeneralisasi adalah banyak jalan dari i ke j dengan panjang N.

Jadi total banyak cara sebuah string dengan panjang N diterima oleh suatu regex adalah jumlah banyak jalan dari i ke j dimana I adalah node awal dan j adalah node akhir

Misal pada contoh diatas,

Untuk N == 1

regex tersebut akan menerima satu kemungkinan string yaitu {‘a’}

Karena

Untuk N==2

Regex tersebut akan menerima satu kemungkinan yaitu {‘ab’}

Karena

DAFTAR PUSTAKA

Sumber belajar dari blog ashar fuadi dan dari codeproject.com