BAB III
DESAIN PERANGKAT LUNAK

Pada bab ini penulis menjelaskan tentang Desain metode yang digunakan untuk menyelesaikan permasalahan pada Tugas Akhir.

3.1 Desain Sistem Secara Umum

Sistem yang dibuat akan menerima masukan berupa sebuah bilangan N pada awal sistem bekerja kemudian N masukan berikutnya berisikan *regular expression* RE dan bilangan L. Keluaran yang diharapkan dari sistem adalah bilangan T yang merupakan banyaknya *string* dengan panjang L yang dapat diterima oleh RE.

Pada setiap kasus uji, sistem akan melakukan *preprocess* pada RE untuk memberikan penanda operator *concatenate* dan kemudian sistem akan memulai untuk membaca RE. Saat pembacaan RE sistem akan membentuk NFA berdasar alfabet dan operator yang dibaca.

main()

```
1.  input N
2.  for i=1 to N
3.   initializeVariables()
4.   input RE, L
5.   RE := Preprocess(RE)
6.   NFA := ConvertREtoNFA(RE)
7.   DFA := ConvertNFAtoDFA(NFA)
8.   adjM := Create adjacency matrix from DFA
9.   MatrixPower(adjM, L)
10.  sum cell of the matrix which have finish
     state of NFA
11.  Output the sum
```

Gambar 3.1.1 *Pseudocode* fungsi main

Usai NFA terbentuk akan dilakukan konversi dari NFA ke DFA. DFA merupakan sebuah *graph* berarah yang dapat

dimodelkan menjadi *adjcacency matrix* untuk melakukan menghitung T, akan dilakukan perpangkatan *matrix* sehingga hasil T merupakan jumlah bilangan pada *matrix* yang ada pada kolom *state* selesai pada sebuah NFA. *Pseudocode* fungsi utama program dapat dilihat pada **Gambar 3.1.1**.

3.2 Desain Algoritma

Sistem terdiri dari beberapa fungsi utama yaitu Preprocess, ConvertREtoNFA, ConvertNFAtoDFA, dan MatrixPower. Pada subbab ini akan dijelaskan tentang masing-masing fungsi utama dan *pseudocode* dari fungsi-fungsi tersebut.

3.2.1 Desain fungsi Preprocess

Fungsi Preprocess digunakan untuk memberikan operator *concatenate* pada *string regular expression*. Fungsi Preprocess dibutuhkan karena pada format masukan, operator *concatenate* tidak memiliki simbol sehingga dapat menimbulkan ambiguitas. *Pseudocode* fungsi Preprocess dapat dilihat pada **Gambar 3.2.1**.

```
/*
* RE   unpreprocessed regular expression
*
* return value
* retRE      preprocessed regular expression
*/
retRE := Preprocess(RE)
```

```
1.  for i=0 to length(RE)
2.    if (RE[i] is alphabet or '(') and (RE[i-1]
    is alphabet or ')')
3.      insert operator "." to RE[i+1]
4,  return RE
```

Gambar 3.2.1 *Pseudocode* fungsi preprocess

Fungsi Preprocess dilakukan dengan menambah operator "." pada indeks di mana terdapat kemungkinan kasus seperti berikut

- "aa", "ab", "ba", "bb".
- ")a", ")b".
- "a(", "b(".
- dan ")(".

Kasus-kasus tersebut diimplementasikan dalam fungsi preprocess pada baris ke 2.

3.2.2 Desain fungsi ConvertREtoNFA

Fungsi ConvertREtoNFA digunakan untuk melakukan konversi *regular expression* ke NFA. Fungsi ini mengimplementasikan model *Automaton* pada metode Thompson. *Pseudocode* fungsi ini dapat dilihat pada **Gambar 3.2.2**. Fungsi ini berjalan dengan membaca *regular expression* per karakter (baris 1) dan akan mengevaluasi sebuah *regular expression* untuk setiap pasang karakter "(" dan ")" (baris 6). Untuk melakukan evaluasi tersebut dibutuhkan dua buah *stack*, yakni *stack* yang menyimpan *automaton* NFA sementara, dan *stack* operator yang menyimpan operator yang harus dievaluasi saat karakter ")" ditemui. Untuk setiap alfabet yang ditemui dalam pembacaan *regular expression* akan dibuat *automaton* seperti pada **Gambar 2.4.2** (baris 4). Saat karakter ")" ditemui dalam pembacaan *regular expression*, akan dilakukan evaluasi terhadap operator yang digunakan dalam operasi selanjutnya dengan mengambil operator teratas pada *stack* operator (baris 7 dan 8).

Automaton baru akan dibuat setiap operator dievaluasi, apabila operator *concatenate* ditemui, maka akan diambil dua *automaton* sementara dari *stack states* yang kemudian digunakan untuk melakukan operasi penggabungan sehingga dihasilkan *automaton* baru untuk disimpan ke *stack states* (baris 9 s.d. 11). Proses yang sama juga berlaku untuk operator *union* dengan melakukan operasi *union* untuk menghasilkan

automaton baru (baris 12 s.d. 14). Pada saat melakukan operasi *klenee star* hanya ada satu *automaton* yang diambil dari *stack states* yang kemudian dimodifikasi sehingga membentuk *automaton klenee star* (baris 15 s.d. 17).

```
* RE   preprocessed regular expression
* NFA  finite state machine model that has
*      epsilon transition
NFA := ConvertREtoNFA(RE)
```

```
1.  for i=1 to length(RE)
2.    if RE[i] is '(' continue loop
3.    else if RE[i] is alphabet
4.      create new NFA automaton with alphabet
    as transition symbol
5.      push newly created NFA automaton to
    states
6.    else if RE[i] is ')'
7.     if stack operator is not empty
8.      pop 1 from stack operator to op
9.     if op is concatenate operator
10.      pop 2 automaton from stack states to st1
    and st2 then create new automaton by st1
    concat st2
11.      push newly created automaton to states
12.    else if op is union operator
13.      pop 2 automaton from stack states to st1
    and st2 then create new automaton by st1
    union st2
14.      push newly created automaton to states
15.    else if op is klenee star operator
16.      pop 1 automaton from stack to st1 and
    create new automaton by modifying st1 to
    macth klenee star Thompson model
17.      push newly created automaton to states
18.   else if RE[i] is operator
19.    push operator to operator stack
20  return NFA = pop last automaton from states
    stack after for loop finished
```

Gambar 3.2.2 *Pseudocode* fungsi ConvertREtoNFA

Apabila karakter yang ditemui dalam *regular expression* adalah operator yakni karakter "|", ".", dan "*",

maka karakter tersebut akan disimpan ke *stack* operator (baris 18 dan 19). Setelah seluruh karakter *regular expression* selesai dibaca dan dievaluasi akan tersisa sebuah *automaton* yang ada pada *stack states*. *Automaton* ini adalah *automaton* NFA hasil konversi dari RE (baris 20).

3.2.3 Desain fungsi ConvertNFAtoDFA

```
* NFA  finite state machine model that has
*      epsilon transition
* return value
* DFA  finite state machine model that has no
*      epsilon transition
DFA = ConvertNFAtoDFA(NFA)
1.   DFA := create empty state machine
2.   NFAinitialState = NFA.getInitialstate()
3.   St := EpsilonClosure(NFAinitialState)
4.   DFASt := CreateDFAState(St, &DFA)
5.   push DFASt to stack tmp
6.   while tmp is not empty
7.    pop tmp and assign to dv
8.    for i=0 to sizeof(alphabet)
9.     St := MoveClosure(dv,alphabet[i])
10.    St := EpsilonClosure(St)
11.    if St is empty set
12.     continue loop
13.    else
14.     DFASt := CreateDFAState(St, &DFA)
15.     Assign alphabet[i] as transition symbol
      from dv to DFASt
16.     If DFAst not already created before
17.      push DFAst to stack tmp
18.  return DFA
```

Gambar 3.2.3 *Pseudocode* fungsi ConvertNFAtoDFA

Fungsi ConvertNFAtoDFA merupakan fungsi yang digunakan untuk melakukan konversi NFA menjadi DFA. Fungsi ini memiliki beberapa fungsi pendukung diantaranya adalah fungsi EpsilonClosure, MoveClosure, dan CreateDFAState. *Pseudocode* fungsi ConvertNFAtoDFA dapat dilihat pada **Gambar 3.2.3**. Fungsi ini dimulai dengan membuat

state awal DFA di mana *state* tersebut merupakan *state* awal NFA dan kumpulan *state* NFA yang dapat dicapai dari *state* awal NFA melalui transisi epsilon (baris 3 dan 4). Setelah *state* awal DFA terbentuk tahapan yang dilakukan selanjutnya adalah melakukan metode *Subset construction* yang telah dijelaskan pada subbab **2.6** halaman 11, pada *pseudocode* **Gambar 3.2.3** hal tersebut dilakukan pada baris 6 s.d 17.

Fungsi EpsilonClosure merupakan salah satu fungsi pendukung dalam proses pembuatan DFA. *Pseudocode* fungsi ini dapat dilihat pada **Gambar 3.2.4**. Fungsi ini memiliki sebuah parameter, yakni kumpulan *state* NFA yang saat ini sedang aktif. Fungsi tersebut diawali dengan membuat daftar pengecekan *state* NFA yang saat ini sedang aktif (baris 1 s.d. 3). Kemudian untuk setiap *state* NFA yang sedang aktif tersebut akan dilakukan pengecekan terhadap *state* aktif selanjutnya yang dapat dikunjungi dengan transisi epsilon (baris 4 s.d 6). Apabila dari *state* yang baru dikunjungi setelah dilakukan transisi epsilon masih memiliki *state* lain yang dapat dikunjungi dengan transisi epsilon maka *state* baru tersebut akan dimasukkan ke daftar pengecekan dan daftar *state* yang dapat dikunjungi dengan transisi epsilon untuk kemudian dilakukan pengecekan (baris 8 s.d 10). Pengecekan tersebut dilakukan selama daftar pengecekan masih belum seluruhnya dilakukan pengecekan (baris 4). Apabila seluruh daftar telah selesai dilakukan pengecekan maka daftar *state* yang dapat dikunjungi dengan transisi epsilon tersebut dikembalikan ke pemanggil fungsi EpsilonClosure.

Fungsi pendukung lain dalam konversi NFA ke DFA adalah fungsi MoveClosure. *Pseudocode* fungsi ini dapat dilihat pada **Gambar 3.2.5**. Fungsi ini menerima kumpulan *state* aktif sebagai parameter dan sebuah alfabet. Untuk setiap *state* aktif akan dilakukan pengecekan apakah ada *state* lain yang dapat dikunjungi dari state yang sedang aktif dengan masukan alfabet dari parameter fungsi (baris 1 dan 2).

```
* St          set of states to transite from
* retSt       set of states that can be reached
*             from St with epsilon transition
retSt = EpsilonClosure(St)
```

```
1.  for i=0 to sizeof(St)
2.   add st to retSt
3.   push st to stack tmp
4.  while stack tmp is not empty
5.   pop tmp and assign to t
6.   r:=next state that can be reached from t
     with epsilon transition
7.   for i=0 to r
8.    if t.next[i] not exist in retSt
9.     add t.next[i] to retSt
10.    push t.next[i] to stack tmp
11. return retSt
```

Gambar 3.2.4 *Pseudocode* fungsi EpsilonClosure

```
* St          set of states to transite from
* alphabet    symbol input to state so the
*             trasition can happen from a state
*             to another set of states
* retSt       set of states that can be reached
*             from St with input alphabet
retSt = MoveClosure(St, alphabet)
```

```
1.  for i=0 to sizeof(St)
2.   if St.next(alphabet) is not empty
3.    add St.next(alphabet) to retSt
4.  return retSt
```

Gambar 3.2.5 *Pseudocode* fungsi MoveClosure

Apabila ada *state* yang dapat dikunjungi dengan masukan alphabet tersebut, maka *state* tersebut dimasukan ke daftar state yang dapat dikunjungi (baris 3). Setelah seluruh *state* aktif selesai dilakukan pengecekan maka daftar *state* yang dapat dikunjungi dikembalikan ke pemanggil fungsi MoveClosure (baris 4).

Fungsi pendukung terakhir dalam konversi NFA ke DFA adalah CreateDFAState. *Pseudocode* fungsi ini dapat dilihat pada **Gambar 3.2.6**. Fungsi ini digunakan untuk membuat dan menambahkan *state* yang baru terbentuk dalam proses pembuatan DFA ke dalam *state machine* DFA. Dalam fungsi

ini juga dilakukan pengecekan apakah state DFA yang akan ditambahkan sudah pernah dibuat sebelumnya atau belum (baris 1). Apabila sudah pernah dibuat, maka fungsi ini hanya akan mengembalikan *state* yang sudah ada (baris 2). Apabila belum, maka fungsi ini akan membuat state DFA kemudian menambahkannya sebelum dikembalikan ke pemanggil fungsi CreateDFAState (baris 4).

```
/*
* St          set of states to join as one DFA
*             state
* DFA         DFA state machine
*
*return value
* DFASt       DFA state
*/
DFASt = CreateDFAState(St, &DFA)
```

```
1.  if St already created and exists in DFA
2.   return St
3.  else add St to DFA
4.  return St
```

Gambar 3.2.6 *Pseudocode* fungsi CreateDFAState

3.2.4 Desain fungsi MatrixPower

```
* AdjM        adjacency matrix
* L
power
* retAdjM     Lth power Matrix
retAdjM = MatrixPOwer(AdjM, L)
```

```
1.  Create identity matrix I that has same
    dimension as adjM
3.  while L>0
4.   if L is odd
5.    I := AdjM x I
6.   AdjM := AdjM x AdjM
7.   L := L/2
8.  return I
```

Gambar 3.2.7 *Pseudocode* fungsi MatrixPower

Fungsi MatrixPower memiliki dua parameter yakni matriks AdjM dan bilangan L. Fungsi ini digunakan untuk memangkatkan matriks AdjM ke pangkat L. *pseudocode* fungsi ini dapat dilihat pada **Gambar 3.2.7**. Fungsi ini menggunakan teknik *Exponentiation by Squaring* seperti yang dijelaskan pada subbab **2.7** halaman 13.

3.3 Desain Struktur Data

Sistem yang dibuat membutuhkan beberapa struktur data untuk menyimpan beberapa data penting. Struktur data pertama yang dibutuhkan adalah struktur data yang dapat menyimpan sekumpulan data tanpa duplikasi dan dapat dilakukan pengecekan secara cepat untuk kesamaan isi himpunan. Struktur data ini dibutuhkan untuk memberikan label yang berbeda untuk tiap *state* DFA. Di mana *state* DFA merupakan himpunan satu atau lebih *state* NFA. Struktur data yang tepat digunakan untuk kasus tersebut adalah struktur data *Set*. Hal ini dikarenakan struktur data *Set* dapat melakukan penyimpanan data dan dapat dipastikan tidak ada duplikasi pada data yang disimpan.

Struktur data kedua yang dibutuhkan adalah struktur data yang dapat digunakan untuk mengakses data dengan parameter tertentu yang diberikan. Struktur data ini digunakan dalam penyimpanan *automaton* DFA untuk memperoleh *state* tertentu dengan parameter himpunan satu atau lebih *state* NFA yang kemudian disimpan dalam struktur data *Set*. Struktur data yang tepat untuk permasalahan seperti ini adalah *Dictionary/Map* di mana suatu data dapat diakses berdasar *key* yang diberikan. Untuk keperluan di atas maka *Dictionary* akan menggunakan *Set state* NFA sebagai *key* dan *state* DFA sebagai data yang disimpan.

Struktur data ketiga yang dibutuhkan adalah struktur data yang dapat digunakan untuk mensimulasikan proses *Last In First Out*(LIFO). Struktur data ini digunakan untuk memproses *regular expression* yang berbentuk operasi infix.

Selain itu struktur data ini diperlukan untuk melakukan proses *traversal* yakni mengunjungi *state* dalam konversi NFA menjadi DFA. Struktur data yang tepat untuk permasalahan tersebut adalah *stack* di mana data juga dilakukan penyimpanan dengan kaidah LIFO.

Struktur data terakhir yang dibutuhkan adalah struktur data yang dapat menyimpan sepasang data sekaligus. Struktur data ini dibutuhkan untuk menyimpan dan memonitor pasangan *state* awal NFA dan *state* akhir NFA saat proses konversi *regular expression* ke NFA. Struktur data ini dapat dipenuhi oleh struktur data *Pair*.

3.4 Desain Pembangkit Kasus untuk Uji Kinerja.

Untuk melakukan uji kinerja diperlukan sebuah program pembantu yang dapat melakukan pembangkitan kasus uji sebagai masukan program Tugas Akhir. *Pseudocode* program pembangkit kasus uji dapat dilihat pada **Gambar 3.4.2**, dan **Gambar 3.4.1**.

Fungsi expand pada **Gambar 3.4.2** digunakan untuk menambah sebuah operasi regular expression baru dengan menyisipkan sebuah operasi ke *index* yang acak. Fungsi akan mengembalikan *string* hasil penambahan operasi ke fungsi utama pada **Gambar 3.4.1**. pada fungsi utama, akan dipanggil fungsi Expand hingga *string* mencapai batas karakter tertentu.

```
main()
1.  Initialize random seed()
2.  print(number of test case)
3.  for i=0 to number of test case
4.   initialize str = "a";
5.   while length of str < 100
6.    index = rand_int() between [1…length(str)]
7.    str = Expand(str)
8.   print(str+" "+rand_int() between 1…10^9)
```

Gambar 3.4.1 *Pseudocode* fungsi utama pembangkit kasus uji

```
* index       integer that represent position in
*             strIN
* strIN       string before expansion.
* str         Regex string after expansion
str = Expand(index, str)
```

```
1.  if strIN[index] is "a" or "b" change index
    to index of not "a" or "b" character
2.  str = strIN
3.  I = rand_int() with range [1…3]
4.  J = rand_int() with range [1…2]
5.  if I = 1
6.   if J = 1
7.    insert "(ab)" to str[index+1]
8.   else
9.    insert "(ba)" to str[index+1]
10. else if I = 2
11.  if J = 1
12.   insert "(a|b)" to str[index+1]
13.  else
14.   insert "(b|a)" to str[index+1]
15. else
16.  if J = 1
17.   insert (a*) to str[index+1]
18.  else
19.   insert (b*) to str[index+1]
20  return str
```

Gambar 3.4.2 *Pseudocode* fungsi Expand

Usai *string* mencapai batas karakter tertentu, kemudian akan dicetak hasil Expand terakhir yang merupakan *RE* dan sebuah bilangan *L* random pada rentang $[1…10^9]$.