IMPLEMENTASI ALGORITMA *WEIGHTED TREE SIMILARITY* PADA APLIKASI TERJEMAHAN HADITS *MUTTAFAQUN ALAIHAL LU'LU' WAL MARJAN*

Muhammad Fahri[1], Hendra Bayu Suseno[2], Anif Hanifah Setyaningrum [3]

Prodi Teknik Informatika, Fakultas Sains dan Teknologi
UIN Syarif Hidayatullah Jakarta
assegafahri253@gmail.com, hendra.bayu@uinjkt.ac.id, anif.hanifa@uinjkt.ac.id

ABSTRAK

Berdasarkan wawancara yang dilakukan penulis dengan *mubaligh* sebagai narasumber, *Al Lu'lu' Wal Marjan* adalah kitab yang menjadi referensi utama di kalangan *mubaligh* dan umat muslim karena tingkat keshahihannya yang tinggi. Walaupun kitab *Al Lu'lu' Wal Marjan* menjadi refensi utama, kuesioner yang penulis sebar menunjukkan bahwa masih banyak umat muslim yang belum mengenal hadits–hadits di dalamnya. Pada umumnya penceramah selama ini menggunakan kitab sebagai dasar ceramahnya. Hal ini dirasa mempersulit penceramah karena kitab membutuhkan ruang, berat dan tidak praktis dalam melakukan pencarian. pemakalah bermaksud mendigitalisasikan kitab *Al Lu'lu' Wal Marjan* kedalam aplikasi untuk membantu *mubaligh* menyampaikan *dalil* dan Juga mengenalkan hadit–hadits *muttafaqun alaih* kepada umat muslim pada umumnya. jumlah hadits di dalam kitab *Al Lu'lu' Wal Marjan* mencapai 1906 hadits, maka dibutuhkan metode pencarian kata dengan hasil pencarian yang cepat serta dapat menyajikan urutan kemiripan untuk memberikan pilihan. Algoritma *Weighted Tree Similarity* yang mengelompokan indikator pencarian dengan bobot, algoritma Boyer Moore yang mencocokan *keyword* dengan teks Serta rumus consine yang menghitung nilai kesamaan antara *tree* berbobot dari *keyword* dengan *tree database*, mampu menghasilkan urutan kemiripan dengan total bobot yang diurutkan secara *decending* dari data yang mendekati pencarian sampai yang jauh dari data pencarian.

Kata Kunci: *Hadits, Muttafaqun Alaih, algoritma Weighted Tree Similarity, algoritma Boyer Moore, Consine*

ABSTRACT

Based on interviews conducted by the author with a preacher as a resource, Al Lu'lu 'Wal Marjan is a book that became the main reference in the Muslim and Muslim preachers because of high level of highness. Even the book of Al Lu'lu 'Wal Marjan became the main reference, the questionnaire writers spread, indicating that there are still many Muslims who have not know hadith - hadith didi. In general, preachers have been using the book as the basis of his speech. This makes it difficult for speakers because the book needs space, weight and is not practical in searching. the devil's devotees digitized the book of Al Lu'lu 'Wal Marjan to the application to help preachers submit their theorem and also introduce the hadits of muttafaqun alaihong Muslim in general. the number of hadiths in the book of Al Lu'lu 'Wal Marjan reached 1906 hadith, then used to find the choice. Algorithm Moving Tree Similarity that categorizes the search indicator by weight, the Boyer Moore algorithm matching the keyword with text And the consine formula that calculates the price between the weighted tree of the keyword with the tree database, is able to generate a sequence of similarities with the accurately ranked total weight of the remote data from search data.

Keywords*: Hadith, Muttafaqun Alaih, Weighted Tree Similarity Algorithm, Boyer Moore algorithm, Consine.*

http://dx.doi.org/10.15408/jti.v11i2.7776

I. PENDAHULUAN

Hadits yang disepakati oleh imam Bukhari dan imam Muslim disebut *muttafaqun alaih*. Syaikh Muhammad Fuad Abdul Baqi pada tahun 1903 mengumpulkan 1906 hadits *muttafaqun alaih* kedalam kitab yang diberi judul *Al Lu'lu' Wal Marjan*. Berdasarkan wawancara yang dilakukan penulis dengan mubaligh sebagai narasumber, *Al Lu'lu' Wal Marjan* adalah kitab yang menjadi referensi utama di kalangan umat muslim karena tingkat keshahihannya yang tinggi. Walaupun kitab *Al Lu'lu' Wal Marjan* menjadi refensi utama, kusioner yang penulis sebar menunjukkan bahwa masih banyak umat muslim yang belum mengenal hadits–hadits di dalamnya. Di antaranya dari 20 reponden didapat dengan metode *simple random sampling*, 40% reponden belum pernah membaca hadits Jibril yang di dalam terdapat iman Islam dan ihsan. 75% reponden belum pernah membaca hadits keutamaan sahabat utama seperti Abu Bakar r.a. dan Umar ibnu Khattab r.a. dan 20% reponden belum pernah membaca hadits tentang sunnah-sunnah kehidupan sehari-hari Nabi SAW seperti cara makan, memakai sendal dan lain-lain.

Pada penelitian ini, algoritma Weighted Tree Similarity dibentuk berdasarkan struktur kitab *Al Lu'lu' Wal Marjan.* Algoritma *Weighted Tree Similarity* digunakan untuk mencari nilai keserupaan antara *keyword* pencarian dari *user* dengan ketersediaan *database* hadits di kitab *Al Lu'lu' Wal Marjan.* Pembobotan diinput oleh *user* untuk menekankan prioritas pencarian pada salah satu *leaf node*. Hasil pencarian menyajikan urutan kemiripan berdasarkan bobot secara *decending* (besar ke kecil) untuk mempermudah user memilih hadits yang dimaksud dari daftar pencarian. Kontribusi dari penelitian ini adalah untuk mempermudah *mubaligh* dan umat muslim dalam membaca dan melakukan pencarian hadist.

II. TINJAUAN PUSTAKA

2.1 *Muttafaqun Alaih Al Lu'lu' Wal Marjan*

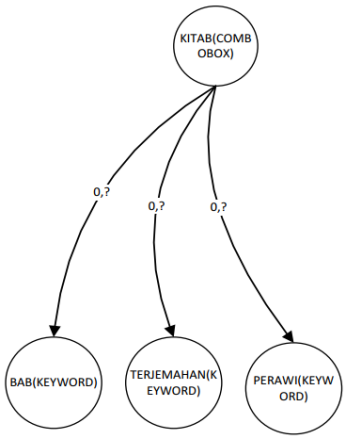
Muhammad bin Ismail Al Bukhari lahir tahun 194H dan Muslim bin Al Hajjaj bin Muslim Al-Qusyairi An Nasaisaburi lahir tahun 204 H merupakan dua orang periwayatan hadits. Hadits yang disepakati oleh dua periwayat hadits tersebut disebut bukhari muslim atau *muttafaqun alaih*. Kitab *Al Lu'lu' Wal Marjan* berisi kumpulan hadits *muttafaqun alaih*. Kitab ini terdiri dari 54 kitab dan 1906 hadits dalam urutan penyusunannya, nama kitab, bab serta penomorannya berdasarkan kitab Shahih Muslim. Dalam hal redaksi matan dan hadits, penyusun kitab ini mengambil dari kitab Shahih Bukhari. Kitab *Al Lu'lu' Wal Marjan* disusun oleh Muhammad Fuad Abdul Baqi bin Shalih bin Muhammad. Beliau lahir di Mesir di desa Balqilyubiyah 1299 H dan tumbuh besar di Kairo.

Pada tahun 2013 PT Fathan Prima Media menerbitkan kitab *Al Lu'lu' Wal Marjan* dalam terjemahan Indonesia dengan judul Hadits Shahih Bukhari Muslim. Cetakan keenam diterbitkan pada tahun 2016 berdasarkan terjemahan dari Abu Firly Taqiy.

a. *Weighted Tree Similarity*

Tree adalah salah satu struktur data untuk representasi pengetahuan dengan membentuk hierarki struktur pohon dengan sejumlah *node* yang saling berhubungan. Bentuk dasar *tree* ditunjukan gambar di bawah ini. Sebuah *node* dapat mengandung nilai, kondisi, atau data. Setiap *node* tidak harus memiliki *child node*.

Node yang tidak memiliki anak disebut *leaf node*. Sebuah *node* hanya memiliki *parent node*. Simpul yang paling atas dalam *tree* adalah *root*. *Root* tidak memiliki *parent*. Internal *root* atau *inner node* adalah sebuah *node* yang memiliki *child*. *External node* atau *outer node* adalah sebuah *node* yang tidak memiliki *child node* atau *external node* adalah *leaf node*. Tinggi sebuah *tree* adalah panjang maksimal dari *root* ke *leaf node*. Kedalaman sebuah *node* adalah jarak *root* ke *node* tersebut.



Gambar 1. Algoritma *Weighted Tree Similarity*

Weighted Tree Similarity adalah algoritma yang digunakan untuk mengukur kemirian dua buah *tree*. *Weighted Tree Similarity* pada awalnya adalah agen pada *e-business* untuk mencocokkan atau mendapatkan keinginan *buyer* dan persediaan *seller* yang paling mirip melalui *weighted tree* sebagai bentuk representasi.

Dalam *marketplace*, *buyer* dan *seller* mengiklankan kebutuhan dan penawaran produk/jasa mereka. Untuk mendapatkan kecocokan antara keduanya, perhitungan kemiripan antara penawaran dan permintaan harus menghasilkan daftar kemiripan terurut untuk *buyer* dan *seller*.

Weighted tree memiliki konsep *node* belabel, *arc* berlabel, dan *arc* berbobot untuk merepresentasikan relasi *parent-child* dari suatu atribut. *Arc* berlabel menunjukan atribut dan bobot *arc* menunjukan tingkat kepentingan dari *arc*. Ada banyak metode dalam pencocokan *leaf node* diantaranya *String Maching* Boyer Moore dan *Consine Similarity*. Berikutnya setelah di dapat bobot dari setiap vektor, bobot diurutkan secara *decending.*

b. Pencarian Kata

Robert Stephen Boyer (Bob Boyer), dan J. Strother Moore pada tahun 1977 mempublikasikan algoritma Boyer Moore yaitu salah satu algoritma yang digunakan untuk pencarian string. Boyer Moore dianggap sebagai algoritma yang paling efisien pada aplikasi umum. Berbeda dengan algoritma pencarian *string* lain yang telah ada sebelumnya, Boyer Moore mulai mencocokkan karakter dari sebelah kanan *pattern*.

Ide utama dari algoritma ini adalah dengan melakukan pencocokan dari paling kanan *string* yang dicari. Dengan menggunakan algoritma ini, secara rata-rata proses pencarian akan lebih cepat dibandingkan dengan proses pencarian lainnya. Ide dibalik algoritma ini adalah bahwa dengan memulai pencocokan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat (Mufthy, 2011).

Ide dari algoritma adalah dengan menggunakan pengetahuan tentang pencarian teks untuk meningkatkan kecepatan pencarian secara signifikan. Algortima Boyer Moore menggunakan suatu langkah sebelum proses untuk membuat *Occurance Heuristic* dan *Shifting Function* yang digunakan untuk melakukan *Bad Character Heuristic* dan *Good Suffix Heuristic*. (Prabhakar Gupta, et al. 2010)

Boyer Moore menggunakan dua heuristik untuk memutuskan seberapa jauh melompat: *Bad Character Heuristic* sering disebut *Occurrence Heuristic*, dan heuristik yang *Good Character* disebut *Match Heuristic*. Informasi untuk heuristik setiap dipertahankan dalam sebuah *array* yang dibangun pada awal operasi yang cocok. (Jon Orwant, et al, 1999).

Secara sistematis, langkah-langkah yang dilakukan algoritma *Boyer Moore* pada saat mencocokkan string adalah:

1. Algoritma Boyer-Moore mulai mencocokkan *pattern* pada awal teks.
2. Dari kanan ke kiri, algoritma ini akan mencocokkan karakter per karakter *pattern* dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
   a. Karakter di *pattern* dan di teks yang dibandingkan tidak cocok (mismatch).
   b. Semua karakter di *pattern* cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma kemudian menggeser *pattern* dengan memaksimalkan nilai penggeseran *good-suffix* dan penggeseran *bad-character*, lalu mengulangi langkah 2 sampai *pattern* berada di ujung teks.

c. Perhitungan Kemiripan

Consine Similarity adalah ukuran kesamaan yang lebih umum digunakan dalam *information retrieval* dan merupakan ukuran sudut antara vektor dokumen Da(titik(ax,bx)) dan Db (titik(ay,by)). Tiap vektor tersebut merepresentasikan setiap kata dalam setiap dokumen teks yang dibandingkan antara dua buah *weighted tree.*

$$Cosine(q,d) = \frac{\sum_{k=1}^{m} w_{qk} \times w_{dk}}{\sqrt{\sum_{k=1}^{m}(w_{qk})^2} \cdot \sqrt{\sum_{k=1}^{t}(w_{dk})^2}}$$

Ketika dua dokumen identik, nilainya adalah 1 ketika dua dokumen tidak identik sama sekali, nilainya adalah 0.

III. METODOLOGI

Dalam penyebaran kuisioner penulis menggunakan google form dibatasi 20 responden secara acak. Pengambilan sampel responden dalam penyebaran kuesioner, penulis menggunakan teknik *Simple Random Sampling*. Dari kuisioner didapat beberapa kesimpulan di antaranya:

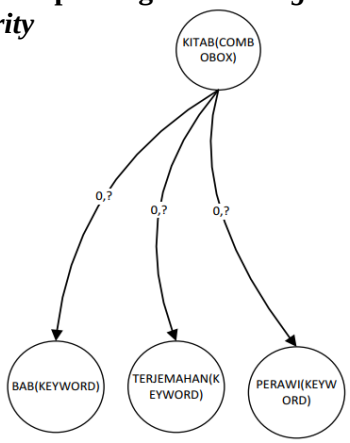
1. 100% responden pengguna OS Android.
2. 90% responden berkeinginan menginstall hadits *muttafaq 'alaih.*
3. 40% reponden belum pernah membaca hadits tentang rukun agama.
4. 20,8% responden belum pernah membaca hadits-hadits yang berkaitan dengan sunnah sehari-hari seperti sunnah memakai sandal, sunnah tata cara makan dan sunnah lain-lainnya.
5. 75% reponden tidak pernah membaca hadits tentang keutamaan sahabat.

Berdasarkan wawancara yang dilakukan penulis dengan mubaligh sebagai narasumber, *al-Lu'lu' wa al-Marjan* adalah kitab yang menjadi referensi utama dikalangan mubaligh dan umat muslim karena tingkat keshahihannya yang tinggi. Walaupun kitab *al-Lu'lu' wa al-Marjan* menjadi refensi utama, kusioner yang penulis sebar, menunjukkan bahwa masih banyak umat muslim yang belum mengenal hadits–hadits di dalamnya. Pada umumnya penceramah selama ini menggunakan kitab sebagai dasar ceramahnya. Hal ini dirasa mempersulit penceramah karena kitab membutuhkan ruang, berat dan tidak praktis dalam melakukan pencarian.

Dari uraian di atas, peneliti bermaksud mendigitalisasikan kitab *al-Lu'lu' wa al-Marjan* ke dalam aplikasi android untuk membantu *mubaligh* menyampaikan dalil. Juga mengenalkan hadit – hadits *muttafaq alaih* kepada umat muslim pada umumnya. jumlah hadits didalam kitab *al-Lu'lu' wa al-Marjan* mencapai 1906 hadits maka dibutuhkan metode pencarian kata dengan hasil pencarian cepat serta dapat menyajikan urutan kemiripan mulai dari yang paling mirip/mendekati data pencarian hingga yang paling jauh dari data pencarian

IV. HASIL DAN PEMBAHASAN

4.1. Penerapan Algoritma *Weighted Tree Similarity*



Gambar 2. Algoritma *Weighted Tree Similarity*

Algoritma *Weighted Tree Similarity* adalah sebuah algoritma untuk mencari nilai keserupaan(*similarity*) dengan menggunakan bentuk *tree* yang berbobot. Pada gambar 2 terlihat bahwa, bentuk *tree* dari kitab *al-Lu'lu' wa al-Marjan* memiliki *node* berlabel diantaranya *root node* kitab, *leaf node* bab, perawi dan terjemahan. bobot diinput oleh *user* sebagai penekanan prioritas pada salah satu *leaf node*. Proses *Weighted Tree Similarity* pada kitab *alLu'lu' wa al-Marjan* dimulai dari *user* mengklik menu cari. Selanjutnya *user* akan disajikan 1 *combobox* atau lebih di kenal di android sebagai *spinner* bersikan judul-judul kitab dan 3 *textbox* di antaranya berisikan bab, perawi, terjemahan. *User* memilih kitab dan menginput *keyword* ke dalam 3 *textbox*, dilanjutkan dengan mengklik button cari. Jika semua hadits-hadits yang didalam kitab yang dipilih tidak ada kandungan *keyword* dari 3 *textbox* yang diinput maka sistem akan mengakhiri proses dengan menampilkan bobot nol dari hadits-hadits yang terdapat dari kitab yang dipilih. Jika tidak, maka sistem akan melakukan proses pencarian dengan algoritma Boyer Moore di masing-masing *leaf node* dimulai dari *leaf node* bab, jika terdapat *keyword leaf node* bab di salah satu hadits maka sistem akan menyisipkan bobot. Begitu juga dengan *leaf node* perawi dan terjemahan. Berikutnya sistem akan melakukan proses perhitungan total bobot dengan *Consine Similarity*. Terakhir user akan disajikan hadits dengan bobot yang diurutkan secara *decending*(besar ke kecil) untuk memudahkan *user* memilih hadits yang dinginkan.

Berikut simulasi dari proses algoritma *Weighted Tree Similarity*. Misalkan *user* memilih semua kitab di *spinner*. Berikutnya *keyword* untuk bab yakni wudhu dengan bobot 0.32, *keyword* untuk perawi yakni Ali dengan bobot 0.13 dan *keyword* untuk terjemahan yakni membasuh dengan bobot0.45. Selanjutnya sistem akan melakukan proses pencarian kata dengan algoritma Boyer Moore dan perhitungan bobot dengan rumus *Consine Similarity*. Berikut adalah penjelasan rinci tentang proses pencarian kata dan perhitungan bobot:

4.1.1. Proses Pencarian Kata dengan Algoritma Boyer Moore

Algoritma Boyer Moore memulai pencocokan karakter dari kanan. Dimisalkan ada sebuah usaha pencocokan yang terjadi pada text[i..i+n-1], anggap ketidakcocokan pertama terjadi di antara text[i + j] dan pattern[j], dengan 0 < j < n. Berarti, teks[i + j + 1..i + n - 1] = pattern[j + 1..n - 1] dan a =teks[i + j] tidak sama dengan b = pattern[j]. Secara sistematis langkah-langkah yang dilakukan sebagai berikut:

1. Langkah pertama mencari nilai table(penggeseran karakter) dari *Occurance Heuristic/* MakeCharTable dan *Match Heuristic*/MakeOffset table. Berikut sampel kodingan dari MakeCharTable dan MakeOffsetTable.

```
private int[] makeCharTable(char[] pattern)
{
   final int ALPHABET_SIZE = 256;
   int[] table = new int[ALPHABET_SIZE];
   for (int i = 0; i < table.length; ++i)
      table[i] = pattern.length;
   for (int i = 0; i < pattern.length - 1; ++i)
      table[pattern[i]] = pattern.length - 1 - i; return table;
}
```

Diperoleh rumus table OH/MakeCharTable

BAB								
INDEX	W	U	D	H	*			
TABLE	5	4	3	2	5			
PERAWI								
INDEX	A	L	*					
TABLE	2	1	3					
TERJEMAHAN								
INDEX	M	E	M	B	A	S	U	*
TABLE	5	6	5	4	3	2	1	8

```
private static int[] makeOffsetTable(char[] pattern)
{
   int[] table = new int[pattern.length];
   int lastPrefixPosition = pattern.length;
   for (int i = pattern.length - 1; i >= 0; --i)
   {
      if (isPrefix(pattern, i + 1))
         lastPrefixPosition = i + 1;
      table[pattern.length - 1 - i] = lastPrefixPosition - i + pattern.length - 1;
   }
   for (int i = 0; i < pattern.length - 1; ++i)
   {
      int slen = suffixLength(pattern, i);
      table[slen] = pattern.length - 1 - i + slen;
   }
   return table;
}
```

Diperoleh rumus table MH/MakeOffsetTable

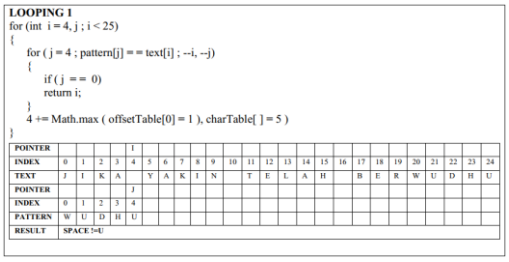
BAB								
INDEX	0	1	2	3	4			
TABLE	1	4	7	8	9			
PERAWI								
INDEX	0	1	2					
TABLE	1	4	5					
TERJEMAHAN								
INDEX	0	1	2	3	4	5	6	7
TABLE	1	9	10	11	12	13	14	15

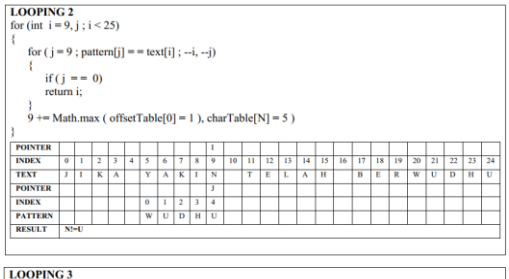
3. Langkah kedua pencocokan *pattern* dengan *text* menggunakan rumus *table* (penggeseran karakter)

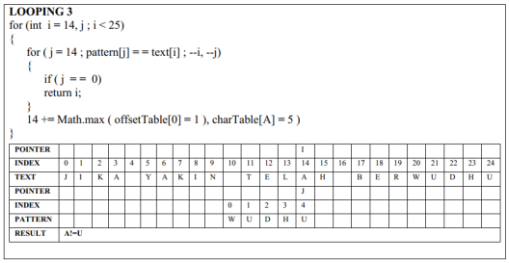
```
for (int i = pattern.length - 1, j; i < text.length;)
{
   for (j = pattern.length - 1; pattern[j] == text[i]; --i, --j)
      if (j == 0)
         return i;

   i += Math.max(offsetTable[pattern.length - 1 - j], charTable[text[i]]);
}
```

Berikut pola penggeseran *text* dan *pattern* di dalam for untuk pencarian kata dalam bab:







LOOPING 4

```
for (int  i = 19, j ; i < 25)
{
    for ( j = 19 ; pattern[j] = = text[i] ; --i, --j)
    {
        if ( j  = = 0)
        return i;
    }
    19 += Math.max ( offsetTable[0] = 1 ), charTable[R] = 5 )
}
```

POINTER																	I							
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
TEXT	J	I	K	A		Y	A	K	I	N		T	E	L	A	H		B	E	R	W	U	D	H	U
POINTER																				J					
INDEX																0	1	2	3	4					
PATTERN																W	U	D	H	U					
RESULT	J!=U																								

LOOPING 5

```
for (int  i = 24, j ; i < 25)
{
    for ( j = 24 ; pattern[j] = = text[i] ; --i, --j)
    {
        if ( j  = = 0)
        return i;
    }

}
```

POINTER																								I
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
TEXT	J	I	K	A		Y	A	K	I	N		T	E	L	A	H		B	E	R	W	U	D	H	U
POINTER																									J
INDEX																					0	1	2	3	4
PATTERN																					W	U	D	H	U
RESULT	U=U, H=H, D=D, U=U, W=W return i																								

Berikut pola penggeseran *text* dan *pattern* di dalam *for* untuk pencarian kata dalam perawi:

LOOPING 1

```
for (int  i = 2, j ; i < 12)
{
    for ( j = 2 ; pattern[j] = = text[i] ; --i, --j)
    {
        if ( j  = = 0)
        return i;
    }
    2 += Math.max ( offsetTable[0] = 1 ), charTable[Y] = 3 )
}
```

POINTER			I																					
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
TEXT	S	A	Y	Y	I	D	I	N	A		A	L	I												
POINTER			J																						
INDEX	0	1	2																						
PATTERN	A	L	I																						
RESULT	Y!=I																								

LOOPING 2

```
for (int  i = 5, j ; i < 12)
{
    for ( j = 5 ; pattern[j] = = text[i] ; --i, --j)
    {
        if ( j  = = 0)
        return i;
    }
    5 += Math.max ( offsetTable[0] = 1 ), charTable[D ] = 3 )
}
```

POINTER						I																		
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
TEXT	S	A	Y	Y	I	D	I	N	A		A	L	I												
POINTER						J																			
INDEX					0	1	2																		
PATTERN					A	L	I																		
RESULT	D!=I																								

LOOPING 3

```
for (int  i = 8, j ; i < 12)
{
    for ( j = 8 ; pattern[j] = = text[i] ; --i, --j)
    {
        if ( j  = = 0)
        return i;
    }
    8 += Math.max ( offsetTable[0] = 1 ), charTable[A] = 2 )
}
```

POINTER								I																
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
TEXT	S	A	Y	Y	I	D	I	N	A		A	L	I												
POINTER								J																	
INDEX							0	1	2																
PATTERN							A	L	I																
RESULT	A!=I																								

LOOPING 4

```
for (int  i = 10, j ; i < 12)
{
    for ( j = 10 ; pattern[j] = = text[i] ; --i, --j)
    {
        if ( j  = = 0)
        return i;
    }
    10 += Math.max ( offsetTable[0] = 1 ), charTable[A] = 2 )
}
```

POINTER										I														
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
TEXT	S	A	Y	Y	I	D	I	N	A		A	L	I												
POINTER									J																
INDEX								0	1	2															
PATTERN								A	L	I															
RESULT	A!=I																								

LOOPING 5

```
for (int  i = 12, j ; i < 12)
{
    for ( j = 12 ; pattern[j] = = text[i] ; --i, --j)
    {
        if ( j  = = 0)
        return i;
    }

}
```

POINTER											I													
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
TEXT	S	A	Y	Y	I	D	I	N	A		A	L	I												
POINTER													J												
INDEX											0	1	2												
PATTERN											A	L	I												
RESULT	I=I, L=L, A=A return i																								

Berikut pola penggeseran *text* dan *pattern* di dalam for untuk pencarian kata dalam terjemahan:

LOOPING 1

```
for (int  i = 7, j ; i < 12)
{
    for ( j = 7 ; pattern[j] = = text[i] ; --i, --j)
    {
        if ( j  = = 0)
        return i;
    }
    7 += Math.max ( offsetTable[0] = 1 ), charTable[K] = 8 )
}
```

POINTER							I																	
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
TEXT	M	E	N	U	A	N	G	K	A	N		L	A	L	U		M	E	M	B	A	S	U	H	
POINTER							J																		
INDEX	0	1	2	3	4	5	6	7																	
PATTERN	M	E	M	B	A	S	U	H																	
RESULT	K!=H																								

LOOPING 2

```
for (int  i = 15, j ; i < 12)
{
    for ( j = 15 ; pattern[j] = = text[i] ; --i, --j)
    {
        if ( j  = = 0)
        return i;
    }
    7 += Math.max ( offsetTable[0] = 1 ), charTable[ ] = 8 )
}
```

POINTER															I									
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
TEXT	M	E	N	U	A	N	G	K	A	N		L	A	L	U		M	E	M	B	A	S	U	H	
POINTER															J										
INDEX								0	1	2	3	4	5	6	7										
PATTERN								M	E	M	B	A	S	U	H										
RESULT	SPACE !=H																								

LOOPING 3

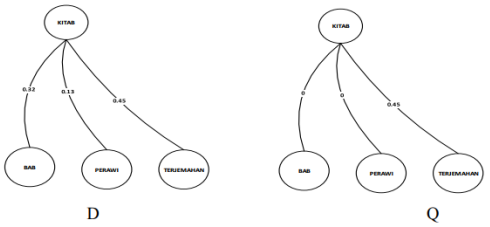
```
for (int  i = 23, j ; i < 12)
{
    for ( j = 23 ; pattern[j] = = text[i] ; --i, --j)
    {
        if ( j  = = 0)
        return i;
    }

}
```

POINTER																								I
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
TEXT	M	E	N	U	A	N	G	K	A	N		L	A	L	U		M	E	M	B	A	S	U	H	
POINTER																								J	
INDEX																0	1	2	3	4	5	6	7		
PATTERN																M	E	M	B	A	S	U	H		
RESULT	H=H, U=U, S=S, A=A, B=B, M=M, E=E, M=M return i																								

4.1.2. Proses Perhitungan Dengan Consine

Setelah pada tahap sebelumnya proses pencocokan dengan algoritma boyer moore antara *keyword* dengan database.Vektor d mewakili *keyword* dan vektor q mewakili *database*. Dimensi yang tidak ada di salah satu vektor diadakan dengan bobot 0. Berikut adalah contoh akumuasi antara vektor d dengan vektor q. Bobot dengan nilai 0.45 untuk terjemahan, 0.13 untuk bab dan 0.32 untuk perawi. Di bawah ini adalah bentuk vektor *tree* dari *keyword* dan vektor *tree* dari database



Gambar 3. Proses perhitungan dengan consine

Berikut adalah hasil perhitungan dua vektor *tree* di atas

$$\text{Consine}(q,d)=\frac{\sum_{k=1}^{m} wqk * wdk}{\sqrt{\sum_{k=1}^{m}(wqk)^2}.\sqrt{\sum_{k=1}^{m}(wdk)^2}}$$

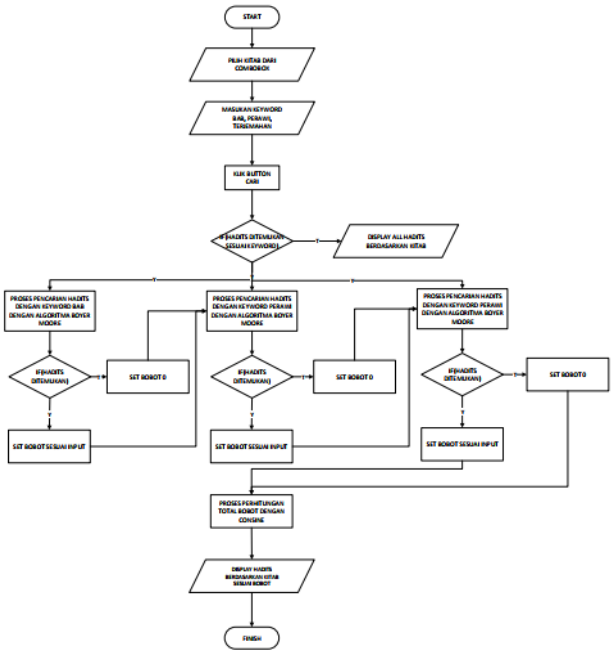
$$\text{Consine}(q,d)=\frac{(0.32*0.32)+(0.13*0.13)+(0.45*0.45)}{\sqrt{0.32^2+0.13^2+0.45^2}.\sqrt{0.32^2+0.13^2+0.45^2}}$$

$$\text{Consine}(q,d)=0.79$$

Berikut adalah rumus consine yang diterjemahan dalam bentuk *coding*

```
public double hasilConsine(){
   double consine = ((WD1 * WQ1) + (WD2 * WQ2) + (WD3 * WQ3)) /
          ((Math.sqrt(Math.pow(WD1, 2) + Math.pow(WD2, 2) + Math.pow(WD3, 2))) *
(Math.sqrt(Math.pow(WQ1, 2) + Math.pow(WQ2, 2) + Math.pow(WQ3, 2))));
   consine1 = Math.round(consine*100.0)/100.0; return consine1;
```

4.1.3. Flowchart *Weighted Tree Similarity*

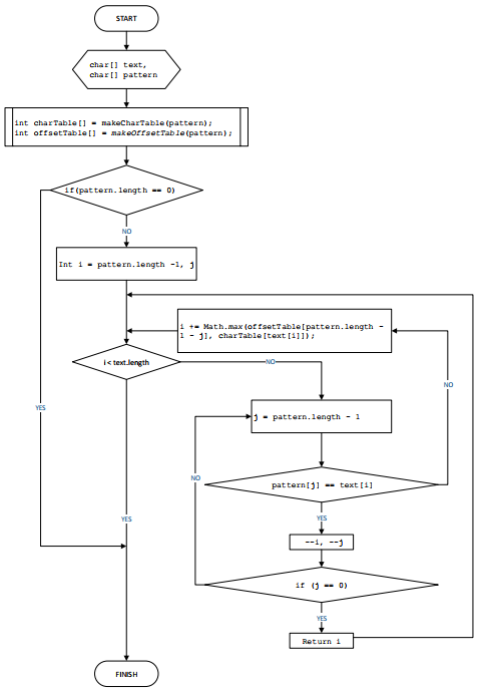


Gambar 4. Flowchart *weighted tree similarity*

Berikut adalah gambaran *flowchart* secara rinci dari proses Boyer Moore dan consine yang terdapat pada algoritma *Weighted Tree Similarity*:

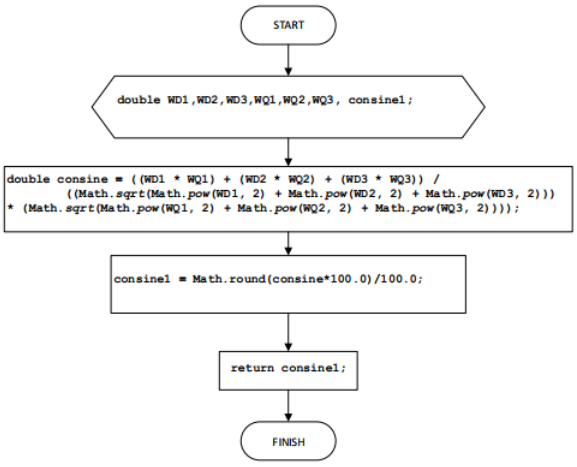
1. *Flowchart* pencarian kata dengan algoritma Booyer Moore.
2. *Flowchart* perhitungan total bobot dengan rumus Consine

4.1.4. *Flowchart* Pencarian Kata dengan Boyer Moore



Gambar 5. *Flowchart Boyer Moore*

4.1.5. *Flowchart* Perhitungan Dengan Consine



Gambar 6. *Flowchart consine*

4.2. Perancangan model UML

4.2.1. *Use Case Diagram*

Pada tahap pertama dengan merancang diagram *Use Case*. Pada diagram ini dapat terlihat interaksi antara sistem dan pengguna. Berikut ini adalah spesifikasi *Use Case*:

1. Identifikasi Aktor

Pengidentifikasian terhadap aktor (pengguna) agar sistem dapat digunakan sesuai kebutuhan. Berikut ini identifikasi aktor pada aplikasi:

Tabel 1. Identifikasi aktor

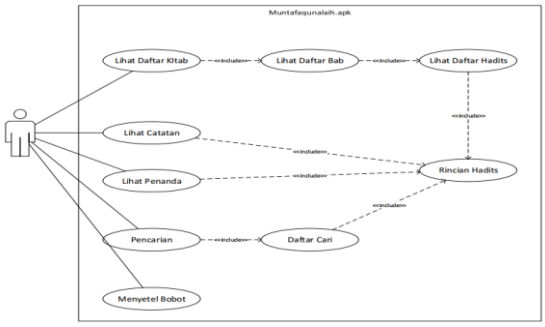
No.	Aktor	Deskripsi
1	User	Orang yang membaca atau mencari Hadits *muttafaqun alaih* dengan bantuan aplikasi ini

2. Identifikasi *Use Case*

Tabel 2. Identifikasi use case

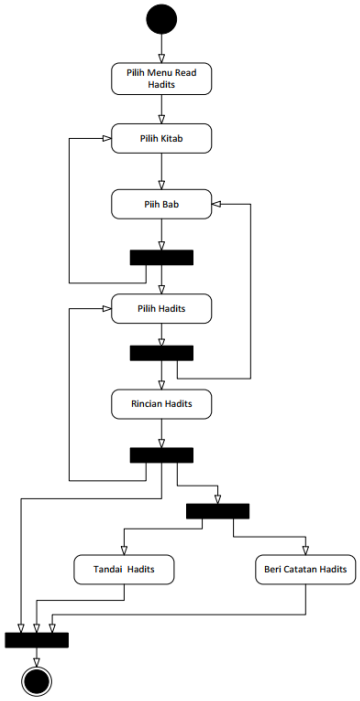
No.	Use Case	Deskripsi
1	Lihat Daftar Kitab	Menampilkan daftar kitab dari database
2	Lihat Daftar Bab	Menampilkan daftar bab dari database berdasarkan kitab yang dipilih
3	Lihat Daftar Hadits	Menampilkan daftar Hadits dari database berdasarkan bab yang dipilih
4	Lihat Catatan	Menampilkan daftar Hadits yang diberi catatan
5	Lihat Penanda	Menampilkan daftar Hadits yang ditandai
6	Pencarian	Menggambarkan proses pencarian Hadits yang dilakukan oleh user dengan cara menginputkan sebuah kata kunci ke dalam kolom pencarian
7	Daftar Pencarian	Menampilkan daftar hadits dari hasil pencarian berdasarkan bobot secara decending
8	Setting Bobot	Mengatur niai bobot bab, terjemahan dan perawi
9	Rincian Hadits	Menampilkan isi konten hadits dari database berdasarkan Hadits yang diilih melalui daftar pencarian, lihat penanda, lihat catatan dan lihat daftar Hadits

3. *Use Case Diagram*



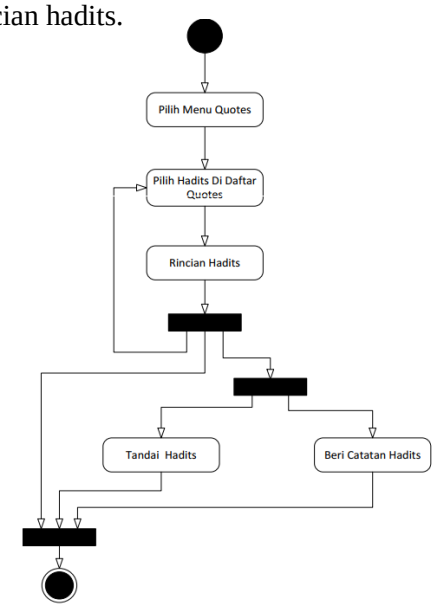
Gambar 7. *Use case diagram*

4.2.2. *Activity Diagram*



Gambar 8. *Activity diagram read hadits*

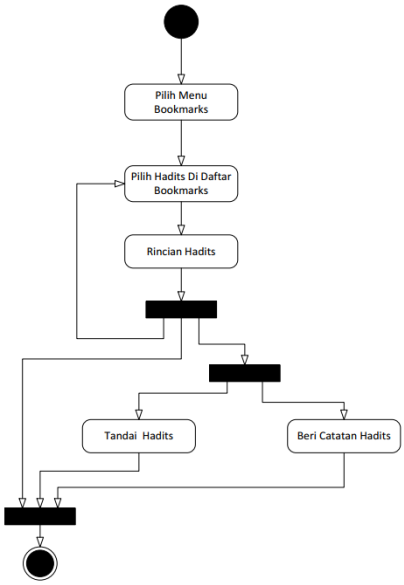
Pada Gambar 8 terlihat bahwa pemilihan rincian hadits tergantung dari pemilihan hadits, pemilihan hadits tergantung dari pemilihan bab dan pemilihan bab tergantung dari kitab yang dipilih. Selain proses *read* hadits dari kitab sampai rincian hadits, terdapat juga proses *bookmark* dan *quotes* di rincian hadits.



Gambar 9. *Activity diagram quotes*

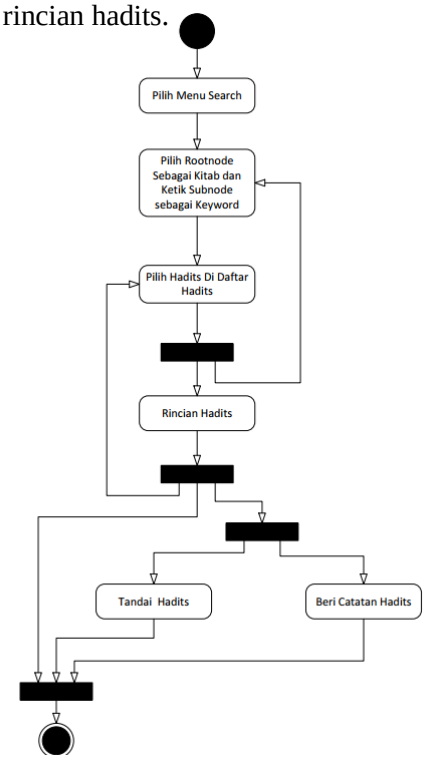
Pada Gambar 9 terlihat bahwa *user* dapat melihat daftar hadits yang

diberi catatan sebelumnya. *ListView* catatan yang diklik *user* merujuk pada rincian hadits.



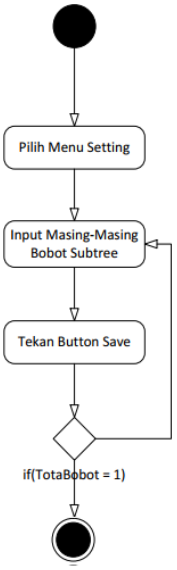
Gambar 10. *Activity diagram bookmark*

Pada Gambar 10 terlihat bahwa *user* dapat melihat daftar hadits yang diberi tanda sebelumnya. *ListView bookmark* yang diklik *user* merujuk pada rincian hadits.



Gambar 11. *Activity diagram search hadits*

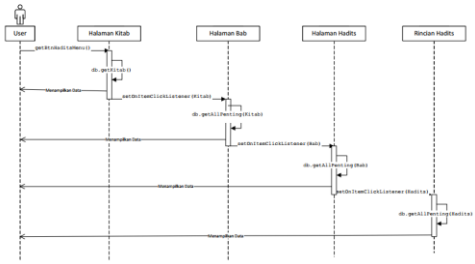
Pada Gambar 11 terlihat bahwa *user* memulai aktivitas pencarian dengan memilih *root node* yakni kitab dari combo box. Berikutnya memberikan *keyword* di masing-masing subtree yakni terjemahan, perawi dan bab. Setelah proses input *keyword* dan *press button search*, user akan melihat daftar hadits dengan bobot yang mewakili *keyword* tersusun secara decending. *ListView* daftar hadits merujuk pada rincian hadits.



Gambar 12. *Activity diagram setting*

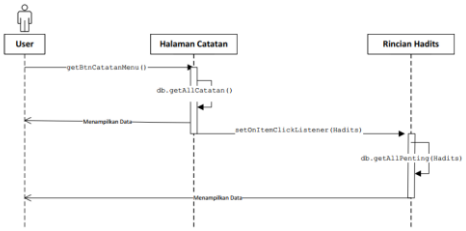
Pada gambar 12 terlihat bahwa pembobotan hanya terjadi di *subtree* dan total dari pembobotan harus bernilai 1.0. Hal ini mengikuti rule dari algoritma *Weighted Tree Similarity* dimana *rood node* tidak berbobot dan angka 1 mewakili maksimal keindentikan hasil pencarian.

4.2.3. *Sequence Diagram*



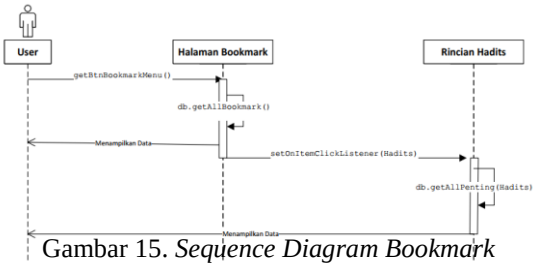
Gambar 13. *Sequence diagram read hadits*

Pada Gambar 13 terlihat bahwa *user* memulai interaksi dengan mengakses halaman kitab. Selanjutnya sistem menampilkan nama-nama kitab yang terdapat dari database dengan method db.getKitab() yang diolah. Berikutnya *user* memilih kitab di *ListView* yang merujuk pada bab berdasarkan kitab yg dipilih. Selanjutnya *user* memilih bab di *ListView* yang merujuk pada hadits berdasarkan bab yg dipilih. Terakhir sistem menampilkan rincian hadits dari rujukan hadits yang dipilih



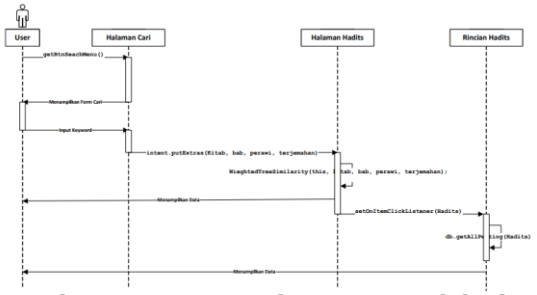
Gambar 14. *Sequence diagram quotes*

Pada Gambar 14 terlihat bahwa *user* memulai interaksi dengan mengakses halaman catatan. Selanjutnya sistem menampilkan *ListView* hadits yang terdapat catatan sebelumnya. *ListView* hadits yang diklik merujuk pada halaman rincian hadits.



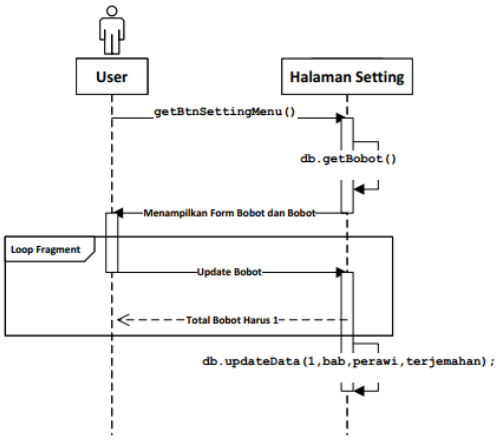
Gambar 15. *Sequence Diagram Bookmark*

Pada Gambar 15 terlihat bahwa *user* memulai interaksi dengan mengakses halaman *bookmark*. Selanjutnya sistem menampilkan *ListView* hadits yang terdapat *bookmark* sebelumnya. *ListView* hadits yang diklik merujuk pada halaman rincian hadits.



Gambar 16. *Sequence diagram search hadits*

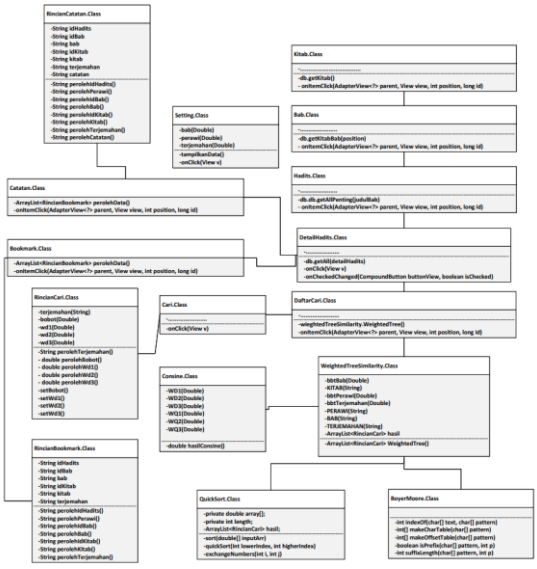
Pada gambar 16 terlihat bahwa user memulai interaksi dengan mengakses halaman cari. Selanjutnya *user* menginput *keyword* baik untuk *root node* maupun *subtree*. Halaman Cari mengirim *keyword* dengan method intent.putExtras (Kitab, bab, perawi, terjemahan) ke halaman hadits atau daftar cari. Dan di halaman inilah *keyword* yang tadi dikirim diproses dengan memanggil metode algoritma induk WeightedTreeSimilarity(this, kitab, bab, perawi, terjemahan). Metode ini menghasilkan *ListView* hadits dengan bobot berdasarkan *keyword* yang diinput. *ListView* hadits yang diklik merujuk pada halaman rincian hadits.



Gambar 17. *Sequence diagram setting hadits*

Pada gambar 17 terlihat bahwa *user* memulai interaksi dengan mengakses halaman *Setting*. Selanjutnya *user* menginput bobot untuk masing-masing subtree. Proses pembobotan terdapat validasi dimana akumulasi total bobot harus 1.0. Hal ini mengikuti *rule* dari algoritma *Weighted Tree Similarity* dimana *root node* tidak berbobot dan angka 1 mewakili maksimal keindentikan hasil pencarian.

4.2.4. *Class Diagram*



Gambar 18. *Class diagram*

4.3. Pengujian

Pengujian metode pencarian *Weighted Tree Similarity* Pada Aplikasi Hadits *Muttafaq Alaih Al Lu'lu' Wal Marjan* dilakukan dengan tiga narasumber, Android dan *Operating System* berbeda.

Tabel 3.Pengujian Sistem

No	Nama	Smartphone	Operating System	Keyword	Waktu Pencarian
1	Eka Pramudita	Lenovo	5.1 (Lolipop)	Iman, Dosa, Ali, Berdusta	+1s144ms
2	Hidaya Sari	Samsung-J510FN	6.0.1()	Iman, Rukun, Islam, Umar, Zakat	+893ms
3	Yogi Arimaska	Sony Ericson E5333	6.0	Bersuci, Wudhu, Abdullah, Membasuh	+544ms

1. Hasil Pencarian Pertama

Berikut adalah rincinan perhitungan consine pertiap hadits berdasarkan *keyword*:

Tabel 4. Perhitungan Consine Pencarian pertama

Hadits	Wqk			Wdk			$\sum_{k=1}^{m} wqk * wdk$	$\sqrt{\sum_{k=1}^{m}(wqk)^2}.\sqrt{\sum_{k=1}^{m}(wdk)^2}$	Con(q,d)
	Q1	Q2	Q3	D1	D2	D3			
1	0.32	0.13	0.45	0.32	0.13	0.45	0.3218	0.3218	1.0
2	0.32	0.13	0.45	0.32	0	0.45	0.3049	0.3132360451799888	0.97
3	0.32	0.13	0.45	0.32	0	0.45	0.3049	0.3132360451799888	0.97
4	0.32	0.13	0.45	0.32	0	0.45	0.3049	0.3132360451799888	0.97
5	0.32	0.13	0.45	0	0	0.45	0.2025	0.25527338286629103	0.79
6	0.32	0.13	0.45	0	0	0.45	0.2025	0.25527338286629103	0.79
7	0.32	0.13	0.45	0	0	0.45	0.2025	0.25527338286629103	0.79
8	0.32	0.13	0.45	0.32	0	0	0.1024	0.18152773892714028	0.56
9	0.32	0.13	0.45	0.32	0	0	0.1024	0.18152773892714028	0.56
10	0.32	0.13	0.45	0.32	0	0	0.1024	0.18152773892714028	0.56
11	0.32	0.13	0.45	0.32	0	0	0.1024	0.18152773892714028	0.56
12	0.32	0.13	0.45	0.32	0	0	0.1024	0.18152773892714028	0.56
13	0.32	0.13	0.45	0.32	0	0	0.1024	0.18152773892714028	0.56
14	0.32	0.13	0.45	0.32	0	0	0.1024	0.18152773892714028	0.56
15	0.32	0.13	0.45	0.32	0	0	0.1024	0.18152773892714028	0.56
16	0.32	0.13	0.45	0.32	0	0	0.1024	0.18152773892714028	0.56
Estimasi Waktu Pencarian								**+1144ms**	

2. Hasil Pencarian Kedua

Berikut adalah rincinan perhitungan consine pertiap hadits berdasarkan *keyword*:

Tabel 5. Perhitungan Consine Pencarian kedua

Hadits	Wqk			Wdk			$\sum_{k=1}^{m} wqk * wdk$	$\sqrt{\sum_{k=1}^{m}(wqk)^2}.\sqrt{\sum_{k=1}^{m}(wdk)^2}$	Con(q,d)
	Q1	Q2	Q3	D1	D2	D3			
1	0.32	0.13	0.45	0.32	0.13	0.45	0.3218	0.3218	1.0
2	0.32	0.13	0.45	0.32	0	0.45	0.3049	0.31323604517998	0.97
3	0.32	0.13	0.45	0	0.13	0.45	0.2194	0.26571209983739	0.83
4	0.32	0.13	0.45	0	0	0.45	0.2025	0.25527338286629	0.79
5	0.32	0.13	0.45	0	0	0.45	0.2025	0.25527338286629	0.79
6	0.32	0.13	0.45	0	0	0.45	0.2025	0.25527338286629	0.79
7	0.32	0.13	0.45	0	0	0.45	0.2025	0.25527338286629	0.79
8	0.32	0.13	0.45	0	0	0.45	0.2025	0.25527338286629	0.79
9	0.32	0.13	0.45	0	0.13	0	0.0169	0.07374564393915	0.23
10	0.32	0.13	0.45	0	0.13	0	0.0169	0.07374564393915	0.23
11	0.32	0.13	0.45	0	0.13	0	0.0169	0.07374564393915	0.23
12	0.32	0.13	0.45	0	0.13	0	0.0169	0.07374564393915	0.23
13	0.32	0.13	0.45	0	0.13	0	0.0169	0.07374564393915	0.23
14	0.32	0.13	0.45	0	0.13	0	0.0169	0.07374564393915	0.23
Estimasi Waktu Pencarian								**+893ms**	

3. Hasil Pencarian Ketiga

Berikut adalah rincinan perhitungan consine pertiap hadits berdasarkan *keyword*:

Tabel 6. Perhitungan Consine Pencarian ketiga

Hadits	Wqk			Wdk			$\sum_{k=1}^{m} wqk * wdk$	$\sqrt{\sum_{k=1}^{m}(wqk)^2} \cdot \sqrt{\sum_{k=1}^{m}(wdk)^2}$	Con(q,d)
	Q1	Q2	Q3	D1	D2	D3			
1	0.32	0.13	0.45	0.32	0.13	0.45	0.3218	0.3218	1.0
2	0.32	0.13	0.45	0.32	0	0.45	0.3049	0.3132360451799888	0.97
3	0.32	0.13	0.45	0	0	0.45	0.2025	0.25527338286629103	0.79
4	0.32	0.13	0.45	0.32	0	0	0.1024	0.18152773892714028	0.56
5	0.32	0.13	0.45	0	0.13	0	0.0169	0.07374564393915074	0.23
6	0.32	0.13	0.45	0	0.13	0	0.0169	0.07374564393915074	0.23
7	0.32	0.13	0.45	0	0.13	0	0.0169	0.07374564393915074	0.23
8	0.32	0.13	0.45	0	0.13	0	0.0169	0.07374564393915074	0.23
Estimasi Waktu Pencarian								+544ms	

V. PENUTUP

5.1. Kesimpulan

1. Algoritma *Weighted Tree Similarity* adalah sebuah algoritma untuk mencari nilai keserupaan(*similarity*) dengan menggunakan bentuk *tree* yang berbobot. terlihat bahwa, bentuk *tree* dari kitab *al-Lu'lu' wa al-Marjan* memiliki *node* berlabel diantaranya *root node* kitab, *leaf node* bab, perawi dan terjemahan.
2. Algoritma Boyer Moore memulai pencocokan karakter dari kanan. Dimisalkan ada sebuah usaha pencocokan yang terjadi pada text[i..i+n-1], anggap ketidak cocokan pertama terjadi di antara text[i + j] dan pattern[j], dengan 0 < j < n. Berarti, teks[i + j + 1..i + n - 1] = pattern[j + 1..n - 1] dan a =teks[i + j] tidak sama dengan b = pattern[j].
3. Dari hasil pengujian yang dilakukan dengan tiga responde, android dan *keyword* yang berbeda, algoritma *Weighted Tree Similarity* mampu menampilkan urutan kemiripan data dari yang mendekati data pencarian hingga yang paling jauh dari data pencarian.

5.2. Saran

Dalam beberapa kasus algoritma atau aplikasi memerlukan beberapa jenis interaksi manusia. Dalam aplikasi ini dibutuhkan inputan berupa pilihan kitab dan memasukan *keyword* untuk bab perawi dan terjemahan. Hal ini bertujuan agar sistem menampilkan hadits yang dimaksud. Sehingga user tidak harus membuka satu per satu hadits pada daftar pencarian untuk menemukan hadits yang diinginkan. jika segmentasinya untuk user pada umumnya, hal ini dirasa sulit untuk sebagian user yang tidak mengetahui susunan kitab *Al Lu'lu' wal Marjan*. Akan lebih mudah jika terdapat opsi menonaktifkan pencarian kitab, bab dan perawi. Sehingga *user* cukup hanya menginput terjemahannya saja.

DAFTAR PUSTAKA

[1] Anggeriana, Herwin, 2011, *Cloud Computing*, Jurnal Teknik Informatika, Vol 1 September 2011

[2] Sarno, R., Anistyasari, Y., & Fitri, R. 2012. *Semantic Search.* Yogyakarta:C.V Andi Offset.

[3] Kendall, K. E., & Kendall, J. E. 2011. *SYSTEM ANALYSIS AND DESIGN.* New Jersey: Pearson Education, Inc

[4] Fu'ad Bin Abdul Baqi, Muhammad. *Mutiara Hadits Shahih Bukhari Muslim.* [online] (https://ibnumajjah.com/2013/03/04/al-lulu-wal-marjan diakses tanggal 16 September 2017 pukul 10:54 WIB)

[5] Rahutomo, F. 2009. Tesis. *Penerapan Algoritma Weighted Tree Similarity Untuk Pencarian Semantik Wikipedia*. Institut Teknologi Sepuluh November.

[6] Yuthika, Dian. 2014. *Implementasi Algoritma Porter Stemmer dan AlgoritmaBoyer Moore Pada Pencarian Ensiklopedia Etika Islam Berdasarkan AlQur'an dan Hadits*. Jakarta: Universitas Islam Negeri Jakarta Syarif Hidayatullah.