BAB II

LANDASAN TEORI

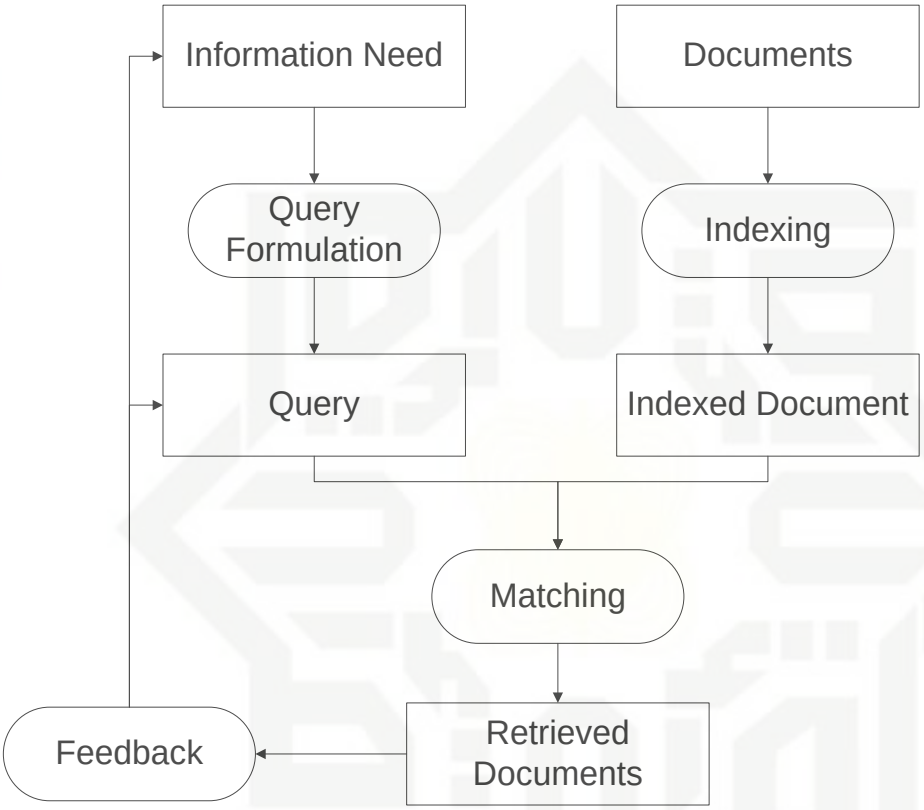
2.1 Sistem Temu Kembali Informasi (*Information Retrieval)*

Sistem Temu Kembali Informasi atau *Information Retrieval* (IR) adalah menemukan materi (biasanya dokumen) dari sekumpulan data yang tidak terstruktur (biasanya teks) untuk memenuhi kebutuhan informasi dari koleksi yang besar (Manning et al., 2009). Sistem temu kembali informasi merupakan suatu sistem yang menemukan (*retrieve)* informasi yang sesuai dengan kebutuhan *user* dari kumpulan informasi secara otomaris. Prinsip kerja sistem temu kembali informasi yaitu jika ada sebuah kumpulan dokumen dan seorang *user* yang memformulasikan sebuah pertanyaan (*request* atau *query)* maka jawaban dari pertanyaan tersebut adalah sekumpulan dokumen yang relevan dan membuang dokumen yang tidak relevan (Gerald, 1988).

Proses ini melibatkan berbagai tahapan dimulai dengan mewakili data dan diakhiri dengan mengembalikan informasi yang relevan kepada pengguna. Di antara tahap itu meliputi *filtering, searching, matching* dan perankingan. Tujuan utama dari sistem temu kembali informasi adalah untuk menemukan informasi yang relevan atau dokumen yang memenuhi kebutuhan informasi pengguna dari koleksi besar dokumen. Penggunaan *information retrieval* dapat diringkas sebagai berikut: memerlukan informasi dalam konteks aplikasi, pengguna memasukkan *query* dengan harapan dapat mengambil satu sumber yang relevan. Dalam mencapai tujuan ini, *information retrieval* biasanya menerapkan 3 proses yaitu (Roshdi & Roohparvar, 2015):

1. Pada proses *indexing* dokumen direpresentasikan dalam bentuk konten yang dirangkum
2. Pada proses *filtering* semua *stopwords* dan kata umum dihapuskan
3. *Searching* atau pencarian merupakan proses inti dari sistem temu kembali informasi banyak teknik untuk *retrieving* atau mengambil kembali dokumen yang sesuai dengan kebutuhan *user.*

Ada tiga dasar proses yang harus sistem temu kembali informasi yaitu representasi dari konten dokumen, representasi kebutuhan informasi pengguna dan perbandingan dari dua representasi tersebut. Proses tersebut dijelaskan pada Gambar 2.1 (Djoerd Hiemstra, 2009):



Gambar 2. 1 Proses Sistem Temu Kembali Informasi (Djoerd Hiemstra, 2009)

Pada Gambar 2.1, kotak persegi menunjukan data dan kotak bulat menunjukkan proses. Representasi dokumen biasanya disebut *indexing proses.* Prosesnya berlangsung secara *offline*, yaitu pengguna akhir dari sistem pencarian informasi tidak terlibat secara langsung. Proses pengindeksan menghasilkan representasi dokumen. Proses representasi dari informasi yang dibutuhkan pengguna disebut sebagai proses perumusan *query*. Representasi yang dihasilkan adalah *query*. Membandingkan kedua representasi tersebut dikenal sebagai proses pencocokan. Proses ini akan menghasilkan daftar peringkat dokumen hasil pencarian. (Djoerd Hiemstra, 2009)

2.1.1 Model Sistem Temu Kembali Informasi *(Information Retrieval)*

Model *information retrieval* menentukan rincian representasi dokumen, representasi *query* dan fungsi pengambilan. Model sistem temu kembali informasi diklasifikan sebagai berikut (Saini, 2016):

1. *Boolean* Model

   Model *Boolean* didasarkan pada aljabar *Boolean*. Algoritma *Boolean* memainkan peran penting dalam sistem *information retrieval* dan mempengaruhi ilmu komputer. Sebuah dokumen terkait dengan satu set kata kunci. *Query* juga ekspresi dari kata kunci yang dipisahkan dengan AND, OR atau NOT. Pada Model *Bolean* pengguna merumuskan kombinasi *term* dan *Boolean* operator untuk mendapatkan dokumen relevan sesuai denngan informasi yang dibutuhkan.

2. *Vector Space* Model

   Model Ruang Vektor atau *Vector Space Model* merupakan model ajabar yang merepresentasikan dokumen sebagai pengenal vektor seperti *indexing* istilah/*term.* Model ini dibangun untuk mengambil informasi dan *index* atau relevan dokumen. Model ini pertama kalai digunakan di SMART IR System.

3. *Probabilistic* Model

   Model probabilistic sebagai pengambilan dokumen sebagai kesimpulan probabilistik. Model ini didasarkan pada prinsip peringkat probabilitas dimana *information retrieval* seharusnya menentukan peringkat dokumen sesuai dengan probabilitas relevansi dengan *query*

2.2 Komponen Sistem Temu Kembali Informasi (*Information Retrieval)*

Information Retrieval dibuat dari sejumlah komponen perangkat lunak yang berkaitan dengan fungsi utama sistem, yaitu (Bates, 2012):

1. Menerima masukan dalam bentuk dokumen, mengekstrak informasi dari dokumen tersebut dan menyimpan informasi tersebut dalam bentuk yang dapat diakses dengan cepat agar sesuai dengan pencarian pengguna

2. Menerima *query* pengguna dan mengubah menjadi bentuk yang dapat dibandingkan dengan informasi tersimpan tentang dokumen. Kedua proses ini sering disebut sebagai *indexing* dan *retrieval.*

Bagian ini akan menjelaskan proses umum *indexing*, *retrieval* dan indeks untuk mencocokan *query* dan dokumen.

2.2.1 *Teks Processing*

Proses *indexing* dalam sistem *information retrieval* berkaitan dengan pemberian representasi dokumen sesuai dengan peraturan dan proses yang ditetapkan untuk sistem tertentu. Salah satu bentuk yang paling sederhana adalah mengekstrak semua kata yang ada di setiap dokumen dan menyimpannya dalam indeks. Umumnya proses ini dikenal sebagai *binary indexing* yaitu sebuah kata tertentu yang terkait dengan dokumen. Normalisasi dokumen dan pembuatan index melibatkan *text processing.* Sistem dapat menggunakan berbagai proses ini (Bates, 2012). Ada beberapa tahapan umum yang digunakan pada proses *text processing* yaitu (Stefano Ceri, 2012):

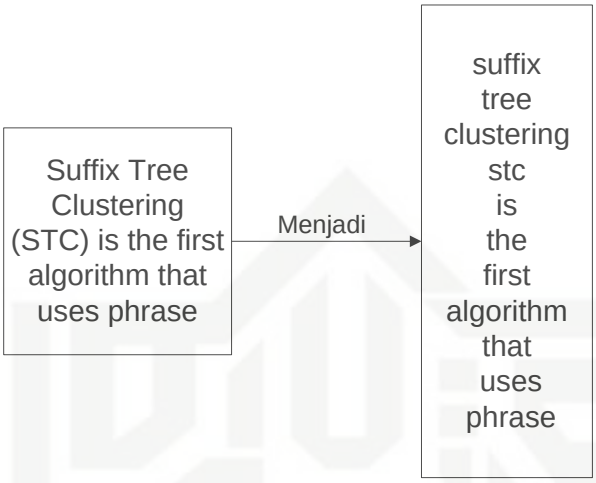
a. *Document Parsing.*

   Dokumen tersedia dalam berbagai Bahasa, rangkaian karakter dan format, seringkali dokumen yang sama berisi beberapa format dan Bahasa. *Document Parsing* berkaitan dengan pengenalan dan pemecahan struktur dokumen menjadi komponen individu. Pada tahap ini unit dokumen dibuat contoh email dengan lampiran akan dibagi menjadi satu representasi dokumen email dan sebanyak dokumen lampiran.

b. *Lexical Analysis.*

   Lexical analysis yaitu proses *tokenize* sebuah dokumen menjadi kata. Menurut Croft et al. (2015) *Tokenization* merupakan proses pembentukan kata-kata dari urutan karakter dalam sebuah dokumen. Dalam teks bahasa Inggris, ini tampaknya sederhana. Pada banyak sistem awal, sebuah "kata" didefinisikan sebagai urutan karakter alfanumerik dengan panjang 3 atau lebih, diakhiri oleh spasi atau

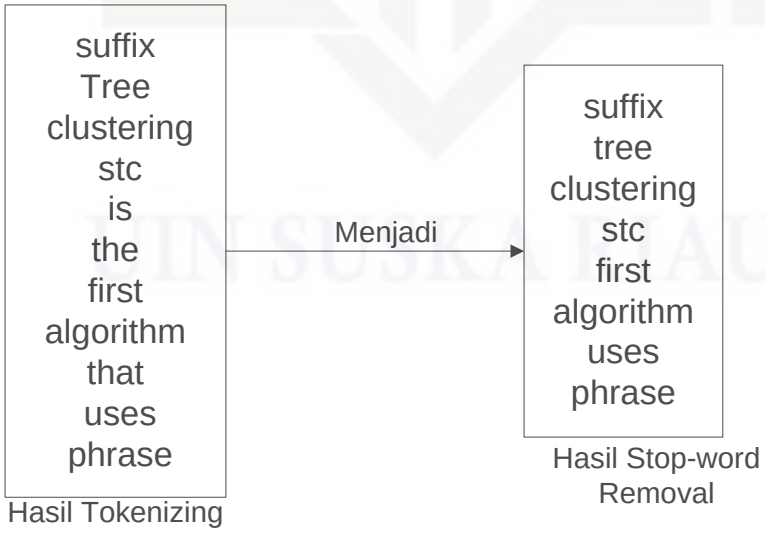
karakter khusus lainnya. Semua huruf besar juga dikonversi menjadi huruf kecil.



Gambar 2. 2 Proses *Lexical Analysis (Tokenization)*

c. *Stop-word Removal*

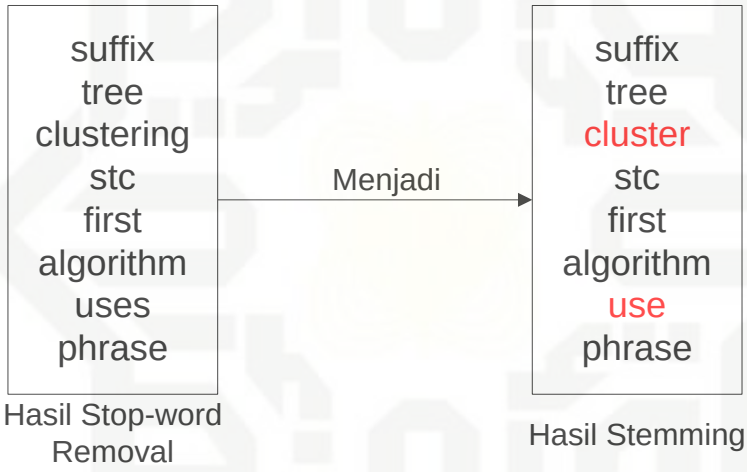
Stop-word Removal yaitu penghapusan kata yang berfrekuensi tinggi. Menurut Bates et al. (2012) Sistem *information retrieval* memiliki daftar kata-kata yang tidak berguna pada pencarian (contoh "an", "the", "a"). beberapa sistem mempunyai daftar *stop-word* yang berbeda untuk setiap index. Proses ini dapat meningkatkan efisiensi *retrieval* informasi. (Croft et al, 2015)



Gambar 2. 3 Proses *Stop-word Removal*

d. *Stemming*

Stemming yaitu proses yang bertujuan untuk menghilangkan akhiran kata untuk menormalkan kata. *Stemming* adalah proses untuk meminimalkan ketidakcocokan antara *query* dan dokumen karena penggunaan bentuk kata yang berbeda (Peters, Braschler, & Clough, 2012). Komponen *stemming* akan menghilangkan akhiran dan awalan yang sama dari kata-kata sehingga mendekati bentuk dasar kata. *Stemming* dapat mengurangi ukuran *indexing* sebanyak 40-50% (Vairaprakash Gurusamy, 2014)



Gambar 2. 4 Proses *Stemming*

e. *Weighting*

Kata-kata dalam sebuah teks memiliki kekuatn deskriptif yang berbeda. Oleh karena itu *term* indeks dapat diberi bobot yang berbeda untuk mrnghitung signifikannya dalam dokumen ataua koleksi dokumen. Bobot semacam itu bisa berupa biner misalnya 0 jika *term* tersebut tidak ada pada koleksi dokumen dan 1 jika ada . Pada penelitian ini penulis menggunakan pembobotan *TFi-idf (Term Frequency-Inverse Document Frequency)*

2.2.2 Index dan *Query* Matching

Hasil *text processing* pada dokumen yang masuk disimpan dalam indeks sistem *information retrieval*. Indeks ini menyediakan mekanisme pencarian cepat untuk setiap token yang diekstraksi dengan pengolahan teks, bersama dengan informasi lainnya termasuk beberapa pengenal untuk dokumen yang berasal dari token, dan biasanya informasi lainnya. Struktur file spesifik yang digunakan untuk indeks bervariasi dari satu sistem ke sistem lainnya, namun yang paling umum adalah beberapa bentuk "*inverted file*". *Inverted file* menyediakan pemetaan antara *term* dan lokasi *occurrence*-nya pada koleksi teks. (Bates, 2012)

2.3 *Porter Stemmer*

Porter stemmer dikembangkan oleh Martin Porter pada tahun 1980 di University of Cambridge. *Porter stemmer adalah* algoritma akhiran penghapusan. Hal ini diketahui sangat sederhana dan ringkas. Algoritma ini didasarkan pada gagasan bahwa sebagian besar sufiks Bahasa Inggris terdiri dari yang sederhana lainnya. Dengan demikian, *stemmer* adalah algoritma linier yang menerapkan aturan morfologi secara berurutan sehingga memungkinkan pelepasan sufiks secara bertahap. Secara khusus, algoritma ini memiliki lima langkah. Setiap langkah mendefinisikan seperangkat aturan. Untuk membendung sebuah kata, aturan itu diuji secara berurutan. Jika salah satu dari aturan ini sesuai dengan kata yang ada, maka syarat yang dilampirkan pada peraturan tersebut diuji. Begitu aturan diterima, akhiran akan dihapus dan langkah selanjutnya dijalankan. Jika aturan tersebut tidak diterima, maka peraturan berikutnya dalam langkah yang sama diuji, sampai salah satu peraturan dari langkah tersebut diterima atau tidak ada lagi peraturan dalam langkah tersebut dan oleh karena itu kontrol lolos ke langkah berikutnya. Proses ini berlanjut untuk semua enam langkah; Pada langkah terakhir *stem* yang dihasilkan dikembalikan (Nagamalai, 2009). Aturan untuk menghapus sebuah akhiran (*suffix)* diberikan dalam bentuk: (kondisi) S1→ S2. Adapun langkah-langkah *porter stemmer* adalah sebagai berikut (Porter, 1980).

1. Tahap 1

Pada tahap ini dirancang untuk menangani *past participles* dan *plural*. Ini adalah langkah yang paling kompleks dan dipisahkan menjadi tiga bagian:

a. Tahap 1a

Tabel 2.1 Tahap 1a

RULES		Contoh	
SUFFIX 1	*SUFFIX 2*	Kata	Hasil *Stemming*
SSES	SS	Caresess	Caress
IES	I	Ponies	Poni
		Ties	Ti
SS	SS	Caress	Caress
S		Cats	Cat

b. Tahap 1b

Tabel 2.2 Tahap 1b

RULES		Contoh	
SUFFIX 1	*SUFFIX 2*	Kata	Hasil *Stemming*
(m>0) EED	EE	Feed	Feed
		Agreed	Agree
(*v*) ED		Plastered	Plaster
		Bled	Bled
(*v*)ING		Motoring	Motor
		Sing	Sing

Jika aturan kedua dan ketiga pada langkah 1b makan aturan dibawah ini juga akan berhasill

Tabel 2.3 lanjutan Tahap 1b

RULES		Contoh	
SUFFIX 1	*SUFFIX 2*	Kata	Hasil *Stemming*
AT	ATE	Conflat(ed)	Conflate
BL	BLE	Troubl(ed)	Trouble
IZ	IZE	Siz(ed)	Size
(*d dan tidak (*L atau *S atau *Z))	*Single Letter*	Hopp(ing)	Hop
		Tann(ed)	Tan
		Fall(ing)	Fall
		Hiss(ing)	Hiss
		Fizz(ed)	Fizz
(m=1 dan *o)	E	Fail(ing)	Fail
		Fill(ing)	File

c. Tahap 1c

Tabel 2.4 Tahap 1c

RULES		Contoh	
SUFFIX 1	*SUFFIX 2*	Kata	Hasil *Stemming*
(*v*) Y	I	Happy	Happi
		Sky	Sky

2. Tahap 2

Tahap ini berhubungan dengan pencocokan pola pada beberapa sufiks yang umum. Pada tahap menghilangkan akhiran *derivational (d-suffixes)* dan mengikuti beberapa peraturan seperti berikut:

Tabel 2.5 Tahap 2

RULES		Contoh	
SUFFIX 1	*SUFFIX 2*	Kata	Hasil *Stemming*
(m>0) ATIONAL	EE	Relational	Relate
(m>0) TIONAL	TION	Conditional	Condition
		Rational	Relational
(m>0) ENCI	ENCE	Valenci	Valence
(m>0) ANCI	ANCE	Hesitanci	Hestitance
(m>0) IZER	IZE	Digitizer	Digitalize
(m>0) ABLI	ABLE	Comfortabli	Comfortable
(m>0) ALLI	AL	Radically	Radical
(m>0) ENTLI	ENT	Differently	Different
(m>0) ELI	E	Vilely	Vile
(m>0) OUSLI	OUS	Analogously	Analogous
(m>0) IZATION	IZE	Vietnamization	Vietnamize
(m>0) ATION	ATE	Predication	Predicate
(m>0) ATOR	ATE	Operator	Operate
(m>0) ALISM	AL	Feudalism	Feudal
(m>0) IVENESS	IVE	Decisiveness	Decisive
(m>0) FULNESS	FUL	Hopefulness	Hopeful
(m>0) OUSNESS	OUS	Callousness	Callous
(m>0) ALITI	AL	Formaliti	Formal
(m>0) IVITI	IVE	Sensitiviti	Sensitive
(m>0) BILITI	BLE	Sensibiliti	Sensible

3. Tahap 3

Langkah 3 membahas akhir kata khusus. Ini juga menghilangkan akhiran *derivational (d-suffixes).*

Tabel 2.6 Tahap 3

RULES		Contoh	
SUFFIX 1	*SUFFIX 2*	Kata	Hasil *Stemming*
(m>0) ICATE	IC	Triplicate	Triplic
(m>0) ATIVE		Formative	Form
(m>0) ALIZE	AL	Formalize	Formal
(m>0) ICITI	IC	Electricity	Electric
(m>0) ICAL	IC	Electrical	Electric
(m>0) FUL		Hopeful	Hope
(m>0) NESS		Goodness	Good

4. Tahap 4

Tahap 4 pada *porter stemmer* yaitu:

Tabel 2.7 Tahap 4

RULES		Contoh	
SUFFIX 1	*SUFFIX 2*	Kata	Hasil *Stemming*
(m>1) AL		Revival	Reviv
(m>1) ANCE		Allowance	Allow
(m>1) ENCE		Inference	Infer
(m>1) ER		Airline	Airlin
(m>1) IC		Gyroscopic	Gyroscop
(m>1) ABLE		Adjustable	Adjust
(m>1) IBLE		Defensible	Defens
(m>1) ANT		Irritant	Irrit
(m>1) EMENT		Replacement	Replac
(m>1) MENT		Adjustment	Adjust
(m>1) ENT		Dependent	Depend
(m>1 dan (*S atau *T)) ION		Adoption	Adopt
(m>1) OU		Homologou	Homolog
(m>1) ISM		Comunism	Commun
(m>1) ATE		Activate	Activ
(m>1) ITI		Angulariti	Angular
(m>1) OUS		Homologous	Homolog
(m>1) IVE		Effective	Effect
(m>1) IZE		Bowdlerize	Bowdler

5. Tahap 5

Tahap 5 terdiri dari 2 langkah yaitu

Tabel 2.8 langkah 5a

RULES		Contoh	
SUFFIX 1	*SUFFIX 2*	Kata	Hasil *Stemming*
(m>1) E		Probate	Probat
		Rate	Rate
(m=1 dan tidak *o) E		Cease	Ceas

Tabel 2.9 langkah 5b

RULES		Contoh	
SUFFIX 1	*SUFFIX 2*	Kata	Hasil *Stemming*
(m>1 dan *d dan *L)	*Single Letter*	Controll	Control
		Roll	Roll

2.4 *Vector Space Model* (Model Ruang Vector)

Menurut Budiharto (2016), *Vector Space Model* (VSM) merupakan model aljabar untuk representasi dokumen teks seperti pada *term index*, digunakan pada *ranked retrieval*, termasuk *document clustering and classification. Vector Space Model* mempertimbangkan setiap vektor dokumen menjadi vektor dalam ruangan dimensi (satu dimensi untuk setiap *term* dalam koleksi). *Vector Space* sistem *information retrieval* mendasarkan peringkat pada seberapa dekat vektor dokumen dan vektor *query* pada ruangan. Hal ini dapat dilihat sebagai perhitungan dari ukuran similaritas berdasarkan dari *term* yang digunakan pada *query* dan pada dokumen di koleksi. (Bates, 2012)

Pada *Vector Space Model* dokumen dan *query* diasumsikan sebagai *t* ruang dimensi vektor, dengan *t* sebagai indeks *term.* Sebuah dokumen D_i direpresentasikan sebagai sebuah vektor dari indeks *term* (Croft, Metzler, & Strohman, 2015)*:*

$$D_i = (d_{1,}\, d_{2,} \dots\dots\, d_{it}) \quad \text{.....(2.1)}$$

Dengan d_{ij} sebagai bobot dari ke-*j term.* Sebuah koleksi dokumen berisi *n* dokumen dapat direpresentasikan sebagai matriks bobot *term,* dengan setiap baris mewakili

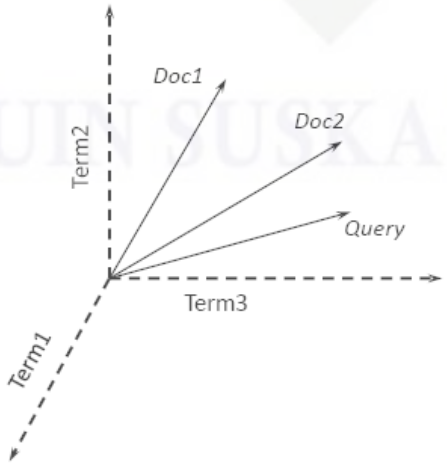
sebuah dokumen dan setiap kolom menggambarkan bobot yang ditugaskan pada sebiah istilah dokumen tertentu

$$\begin{bmatrix} & Term_1 & Term_2 & \dots & Term_t \\ Doc_1 & d_{11} & d_{12} & \dots & d_{1t} \\ Do_2 & d_{21} & d_{22} & \cdots & d_{2t} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ Doc_n & d_{n1} & d_{n2} & \dots & d_{nt} \end{bmatrix}$$

Stop-word tidak di indeks dan setiap kata sudah *stemming. Query* direpresentasikan sama halnya dengan dokumen. *Query Q* direpresentasikan oleh sebuah vektor bobot:

$$Q_i = (q_{1,} q_{2,} \dots \dots q_t) \quad \dots\dots\dots\dots\dots\dots(2.2)$$

Dengan q_j adakah bobot dari ke-*j term* pada *query.* Dengan adanya representasi ini, dokumen dapat digolongkan dengan menghitung jarak antara titik-titik yang mewakili dokumen dan *query*. Lebih umum lagi, ukuran similaritas digunakan, sehingga dokumen dengan nilai tertinggi paling mirip dengan *query*. Sejumlah pengukuran similaritas telah diusulkan dan diuji untuk tujuan ini. Yang paling berhasil adalah pengukuran similaritas korelasi kosinus. Korelasi kosinus mengukur kosinus sudut antara *query* dan vektor dokumen. Jika vektor dinormalisasi sehingga semua dokumen dan *query* diwakili oleh vektor dengan panjang yang sama, kosinus sudut antara dua vektor identik adalah 1 (sudutnya nol), dan untuk dua vektor yang tidak memiliki nilai nol *term*, kosinus akan menjadi 0. Ukuran kosinus didefinisikan sebagai:



Gambar 2. 5 Representasi vektor dokumen dan *query* (Croft et al, 2015)

Perhitungan *Vector Space Model* menggunakan *cosine similarity* antara dokumen dan *query* sebagai berikut

$$\textit{Cosine}\ \mathbf{Di} = \frac{\boldsymbol{q} \cdot \boldsymbol{d_j}}{|\boldsymbol{q}| * |\boldsymbol{d_j}|} \quad \text{.....(2.3)}$$

Keterangan:

$\boldsymbol{q}$: bobot *query*

$\boldsymbol{d_j}$: bobot dokumen

$|\boldsymbol{q}|$: akar jumlah kuadrat *q*

$|\boldsymbol{d_j}|$: akar jumlah kuadrat dokumen

Meskipun tidak ada definisi eksplisit relevansi pada *vector space model*, ada asumsi implisit bahwa relevansi terkait dengan kesamaan vektor *query* dan dokumen. Dengan kata lain, dokumen "lebih dekat" dengan *query* lebih mungkin relevan.

2.5 *Term Frequency-Inverse Document Frequency (TF-IDF)*

Indeks bobot *term* mencerminkan kepentingan relatif kata-kata dalam dokumen, dan digunakan dalam menghitung skor untuk perangkingan. Bentuk spesifik dari bobot ditentukan oleh model pengambilan. Komponen pembobotan menghitung bobot dengan menggunakan statistik dokumen dan menyimpannya dalam tabel pencarian. Bobot dapat dihitung sebagai bagian dari proses *query*. Salah satu jenis yang paling umum digunakan pada model pencarian dikenal dengan pembobotan *TF-IDF (term frequency-inverse document frequency).* Ada banyak variasi dari bobot ini, namun semuanya didasarkan pada kombinasi frekuensi atau hitungan dari kemunculan indeks *term* dalam sebuah dokumen (*term frequency*) dan frekuensi kemunculan indeks *term* terhadap keseluruhan koleksi dokumen (*inverse document frequency*). Bobot idf disebut frekuensi dokumen invers karena memberi bobot tinggi pada istilah yang terdapat dalam dokumen yang sangat sedikit. (W. Bruce Croft, Donald Metzler, 2015)

Term Frequency (TF) adalah metode pembobotan yang menentukan bobot dokumen berdasarkan pada kemunculan *term*. Lebih sering sebuah *term* muncul, bobot dokumen lebih tinggi untuk *term* tersebut. Hasil pembobotan ini selanjutnya

akan digunakan dengan fungsi perbandingan untuk menentukan dokumen yang relevan. *Inverse Document Frequency* (IDF) meningkatkan nilai bobot dokumen berdasarkan rumus: "lebih banyak dokumen mengandung sebuah *term*, lebih kecil bobot *term*-nya, karena tidak dapat digunakan untuk membedakan relavan antara dokumen ". Rumus IDF adalah sebagai berikut (Cios, Pedrycz, Swiniarski, & Kurgan, 2007):

$$idf = \log \frac{N}{df_t} \quad \text{.....(2.4)}$$

Keterangan:

N : total koleksi dokumen

d : frekuensi dokumen

Penggabungkan definisi *term frequency* dan *inverse document frequency*, menghasilkan pembobotan untuk setiap *term* dalam setiap dokumen. Skema pembobotan *TF-IDF* yaitu (Cios et al., 2007)

$$Wf_{(t,d)} = Tf_{(t,d)} x\ idf_{(t)} \quad \text{.....(2.5)}$$

Keterangan:

idf : hasil perhitungan *idf*

w : bobot dokumen ke-d terhadap kata ke-t

tf : banyaknya kata yang dicari pada sebuah dokumen

TF-IDF menetapkan untuk memberi bobot pada dokumen d yaitu (Premalatha & Srinivasan, 2014)

a. Tertinggi bila t terjadi berkali-kali dalam sejumlah kecil dokumen
b. Lebih rendah jika *term* tersebut terjadi lebih sedikit sebuah dokumen
c. Terendah bila *term* itu terjadi hampir semua dokumen

2.6 *Clustering*

Clustering adalah model *data mining* yang diadopsi secara luas, sehingga data dibagi menjadi beberapa kelompok yang disebut *cluster* (Ilic, Rancic, & Spalevic, 2016). *Cluster* adalah daftar data yang memiliki karakteristik yang familier. Analisis *cluster* dapat dilakukan dengan mencari kesamaan antara data

sesuai karakteristik yang terdapat pada data dan pengelompokan data yang sesuai kedalam *cluster* (Amandeep Kaur Mann, 2013)

Clustering telah digunakan pada *information retrieval* untuk memperluas hasil pencarian relevan. *Document Clustering* memberikan informasi yang lebih relevan pada hasil pencarian (Gong, Ke, Zhang, & Broussard, 2013). Berdasarkan hipotesis yaitu "dokumen pada *cluster* yang sama relevan terhadap kebutuhan informasi. Hipotesis tersebut menyatakan bahwa jika ada dokumen dari sebuah *cluster* yang relevan terhadap permintaan pencarian maka dokumen yang lain pada *cluster* yang sama juga relevan. Hal ini karena *clustering* mengumpulkan dokumen yang berbagi banyak *terms* (Ilic et al., 2016).

*Cluster*ing hasil pencarian telah dipelajari di bidang *Information Retrieval* Tujuan pengelompokan hasil pencarian adalah memberi pengguna gagasan tentang apa hasilnya. Ide ini berbentuk *cluster*. *Clustering* dalam konteks hasil pencarian berarti mengatur halaman hasil *query* ke dalam kelompok berdasarkan kemiripannya satu sama lain. (Sadaf & Alam, 2012)

2.7 *Suffix Tree*

Suffix tree adalah struktur data yang berisi semua *suffix* dari *string* yang diberikan, sehingga bisa menjalankan banyak operasi *string* penting dengan lebih efisien. *String* bisa berupa *string* karakter atau *string* kata. *Suffix tree* untuk *string* S didefinisikan sebagai pohon sedemikian rupa sehingga: jalur dari *root* ke *leaves* memiliki hubungan satu lawan satu dengan akhiran S, semua tepi diberi label dengan *string* tidak kosong, semua *node internal* (kecuali *root)* memiliki setidaknya dua anak (Deng, 2012).

Struktur data *suffix tree* adalah inti dari algoritma *suffix tree clustering*, yang mengambil dokumen sebagai string tunggal, dengan menggunakan koleksi string membangun akhiran pohon. Algoritma *suffix tree clustering* adalah algoritma *cluster*ing waktu linier dan menggunakan struktur data *suffix tree* untuk menemukan frasa yang terdapat dalam beberapa dokumen dan dibagi oleh mereka dan menggunakan informasi ini untuk membangun kelas dasar(Wang, Liu, Dong, & Zheng, 2015). Algoritma terbaik untuk *suffix tree* adalah algoritma Ukkonen. Algoritma Ukkonen membangun *suffix tree* dengan representasi paling tersusun dan

teknis. (Ilic et al., 2016). Oren Zamir adalah orang pertama yang mengusulkan Algoritma *Suffix Tree Clustering* untuk pemgelompokan hasil pencarian pada *search engine* dengan mengidentifikasi *shared phrased* dari dokumen yang berbeda (Wang et al., 2015)

2.7.1 Algoritma Ukkonen

Algoritma Ukkonen membangun *suffix tree* untuk setiap awalan S[i..i] dari S (panjang m). Membangun *suffix tree* pertama kali menggunakan T_1 menggunakan karakter pertama, T_2 menggunakan karakter kedua dan T_3 menggunakan karakter ketiga ,…, T_m menggunakan m karakter. (Gusfield, 1997)

Berikut algoritma Ukkonen untuk membuat *suffix tree* . (Gusfield, 1997)

Algoritma Ukkonen

Buatlah *tree* T_1
Untuk i dari 1 sampai m-1 lakukan
Mulai {*phase i+1}*
    Untuk j dari 1 sampai i+1 lakukan
        Mulai {*extension* j}
        Temukan akhir dari jalur *root* berlabel S [j…i] pada *tree* saat ini
        Perluas jalur dengan menambahkan karakter jika belum ada
    End;
End;

Extension suffix adalah tentang menambahkan karakter berikutnya ke *suffix tree* yang dibangun. Pada perpanjangan j dari *phase* i+1, algoritma menemukan akhir S [j...i] dan kemudian meluas untuk S [j…i] untuk memastikan akhiran S [j…i+i] ada di pohon.

Ada tiga *rule extension* (Gusfield, 1997)*:*

Rule 1 : untuk *phase* i+1 jika S[j…i] berakhir pada karakter akhir *leaf* maka karakter S [i+1] ditambahkan pada bagian akhir

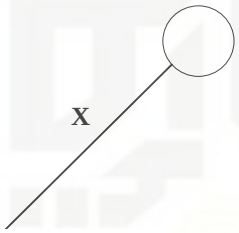
Rule 2 : untuk *phase* i+1 jika S[j…i] berakhir pada *leaf* maka tambahkan S[i+1] pada akhir karakter *leaf* dan tambahkan *leaf* baru dengan label S[i+1]. Dan jika S[j…i] berakhir pada tengah *leaf* maka tambahkan sebuah *node* pada akhir akhir posisi. Dan tambahakan sebuah *leaf* dengan label S [i+1] dan label j pada *node*

Rule 3 : jika S [j..i] berakhir di tempat di tengah tepi dan karakter berikutnya adalah S [i + 1] maka tidak melakukan apapun

Berikut ini adalah contoh pembuatan *suffix tree* untuk String xabxac menggunakan algoritma Ukkonen.

1. Buat *root.* Posisi j dan i pada karakter x maka buatlah *leaf* baru dengan karakter x.

   x a b x c
   j,i



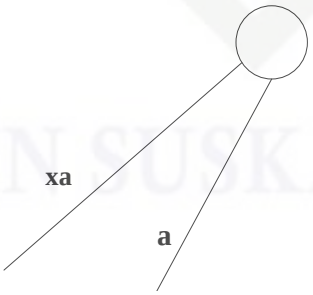
Gambar 2. 6 Langkah 1 *Suffix Tree*

2. Karakter kedua T_2 dari *string* yaitu a. Posisi j pada karakter x dan i pada karakter a, pada *leaf* ada karakter x tapi tidak ada karakter a maka tambahkan karakter a (*rule 1).*

   x a b x c
   j i

   Posisi j dan i ada pada karakter a, cek apakah ada *leaf* dengan label karakter a, jika tidak tambahkan *leaf* baru dengan label karakter a. (*rule 2).*

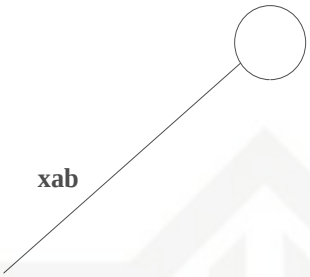
   x a b x c
   j,i



Gambar 2. 7 Hasil *Suffix Tree* Karakter Kedua

3. Karakter ketiga T_3 dari *string* yaitu b. Posisi j ada pada karakter x dan i pada karaketer b , pada *leaf* ada karakter xa tapi tidak ada karakter b maka tambahkan karakter b (*rule 1).*

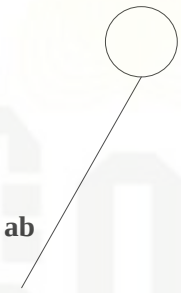
x a b x c
j i



Gambar 2. 8 Langkah 3 *Suffix Tree*

Lalu posisi j pada karakter a dan i pada karakter b. Cek apakah ada jalur dari j ke i, pada *leaf* ada karakter a tapi tidak ada karakter b maka tambahkan karakter b (*rule 1).*

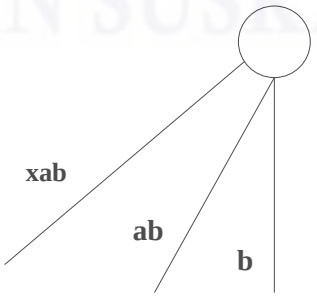
x a b x c
j i



Gambar 2. 9 Langkah 3 *Suffix Tree*

Posisi j dan i ada pada karakter b, cek apakah ada *leaf* dengan label karakter b, jika tidak tambahkan *leaf* baru dengan label karakter b. (*rule 2).*

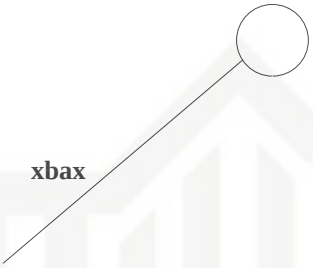
x a b x c
j,i



Gambar 2. 10 Hasil *Suffix Tree* Karakter Ketiga

4. Karakter keempat T_4 dari *string* yaitu x. Posisi j ada pada karakter x dan i pada karaketer x, pada *leaf* ada karakter xab tapi tidak ada karakter x maka tambahkan karakter x (*rule 1).*

x a b x c
j i



Gambar 2. 11 Langkah 4 *Suffix Tree*

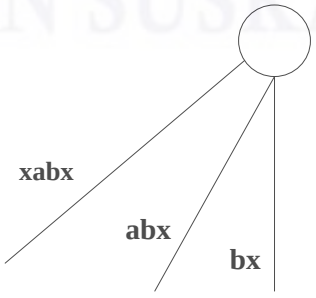
Posisi j ada pada karakter a dan i pada karaketer x , pada *leaf* ada karakter ab tapi tidak ada karakter x maka tambahkan karakter x (*rule 1).*

x a b x c
j i

Gambar 2. 12 Langkah 4 *Suffix Tree*

Posisi j ada pada karakter b dan i pada karaketer x , pada *leaf* ada karakter b tapi tidak ada karakter x maka tambahkan karakter x (*rule 1).*

x a b x c
j i

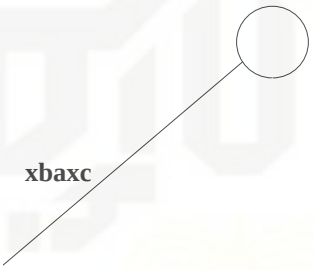


Gambar 2. 13 Hasil *Suffix Tree* Karakter Keempat

Posisi j dan i pada karakter x, Cek apakah ada *leaf* dengan label karakteer x, jika sudah maka tidak lakukan apapun. *(rule 3)*

5. Karakter kelima T_5 dari *string* yaitu c. Posisi j ada pada karakter x dan i pada karaketer c, pada *leaf* ada karakter xabx tapi tidak ada karakter c maka tambahkan karakter c (*rule 1).*

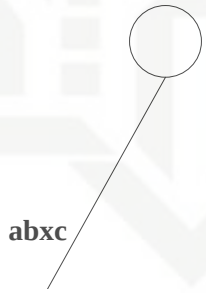
x	a	b	x	c
j				i



Gambar 2. 14 Langkah 5 *Suffix Tree*

Posisi j ada pada karakter a dan i pada karaketer c, pada *leaf* ada karakter abx tapi tidak ada karakter c maka tambahkan karakter c (*rule 1).*

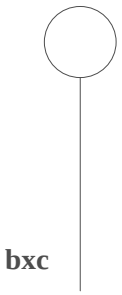
x	a	b	x	c
		j		i



Gambar 2. 15 Langkah 5 *Suffix Tree*

Posisi j ada pada karakter b dan i pada karaketer c, pada *leaf* ada karakter bx tapi tidak ada karakter c maka tambahkan karakter c (*rule 1).*

x	a	b	x	c
		j		i



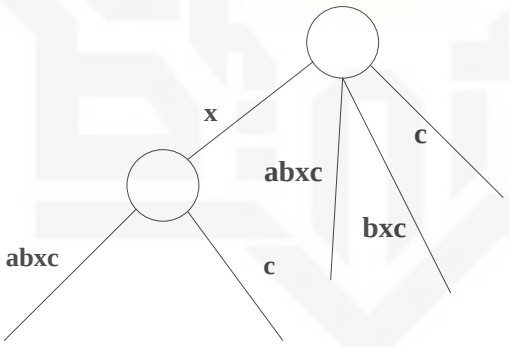
Gambar 2. 16 Langkah 5 *Suffix Tree*

Posisi j ada pada karakter x dan i pada karaketer c, cek *leaf* dengan label x, setelah karakter x tidak ada karakter c maka buatlah node setelah karakter x. *(rule 2)*

x	a	b	x	c
			j	i

Posisi j dan i pada karaketer c, cek apakah ada *leaf* dengan label karakter c, jika tidak tambahkan *leaf* baru dengan label karakter b. *(rule 2)*

x	a	b	x	c
				j,i



Gambar 2. 17 Hasil Akhir *Suffix Tree String* xabxc

Dokumen diperlakukan sebagai *string* kata-kata, bukan karakter, sehingga *suffix* mengandung satu atau lebih keseluruhan kata pada pembentukan *suffix tree* dokumen (Deng, 2012)

2.8 *Suffix Tree Clustering* (STC)

Suffix Tree Clustering (STC) merupakan algoritma pertama yang menggunakan frasa atau kata, jadi prosesnya lebih *simple* dari algoritma lain (Zamir, 1999). *Suffix Tree Clustering* memiliki dua langkah utama. Dalam langkah

pertama, pencarian *shared phrase* untuk semua dokumen yang dikoleksi. S*hared phrase* disebut sebagai *phrase cluster* atau *base cluster,* yang ditemukan dengan menggunakan suatu struktur data yang dinamakan *suffix tree.* Dalam langkah kedua dikombinasikan *base cluster-base cluster* ke dalam suatu *cluster*. Penggabungan antar dua *base cluster* didasarkan pada jumlah dokumen yang melakukan *overlap* diantara kedua *base cluster* tersebut (Zamir, 1999).

Beberapa karakteristik yang membuat *Suffix tree Clustering* cocok digunakan untuk pengelompokan dokumen. Pertama adalah men*generate cluster-cluster* untuk pengelompokan dokumen berdasarkan *phrase. Phrase* juga bermanfaat untuk membangun uraian dan keakuratan deskripsi dari *cluster-cluster*. Kedua, tidak tergantung pada model data. Hal itu mengasumsikan hanya dokumen-dokumen dengan topik yang sama yang akan memiliki *shared phrase.* Ketiga, STC memperbolehkan adanya *overlaping cluster*. Hal itu sangat penting untuk menghindari pembatasan bahwa setiap dokumen hanya memiliki satu *cluster* saja, karena sering kita jumpai satu dokumen mempunyai lebih dari satu topik dan dengan begitu terdapat kemiripan yang lebih dari satu kelompok dokumen. Keempat, STC menggunakan definisi *cluster* yang sederhana. Semua dokumen yang berisi salah satu *phrase cluster* akan menjadi anggota dari *cluster* tersebut. *Suffix tree clustering* 3 langkah logis yaitu: (Zamir, 1999)

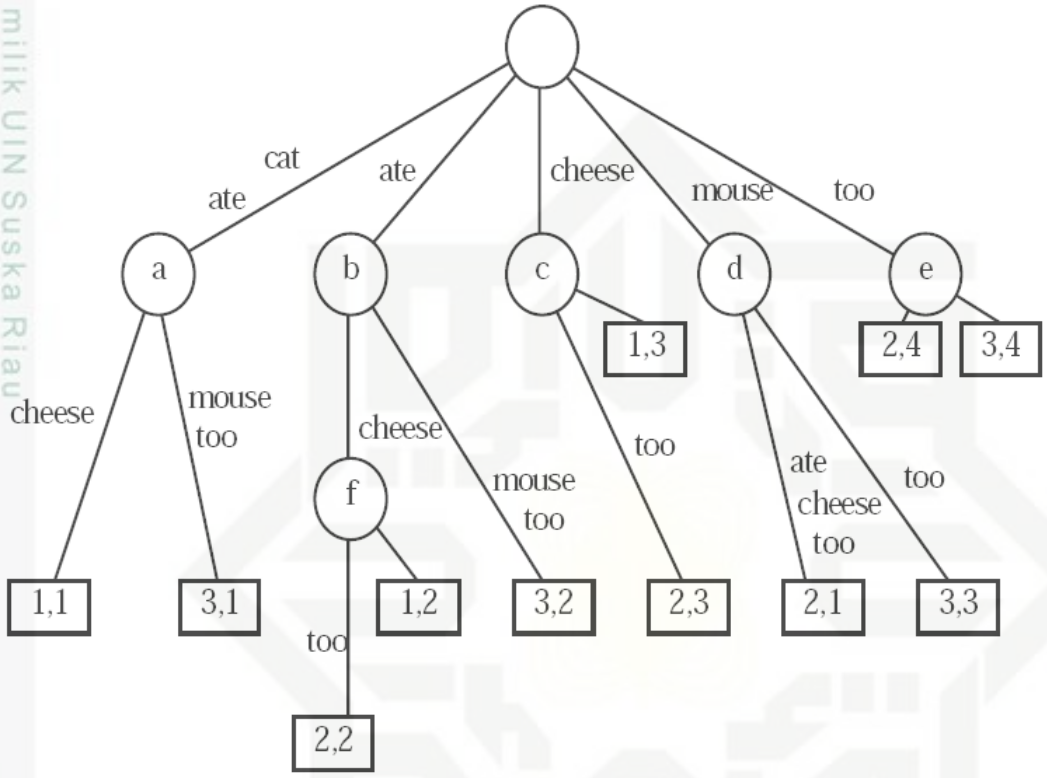
2.8.1 *Document Cleaning*

Document Cleaning adalah tahap awal dalam algoritma *Suffix tree Clustering*. Pada tahap ini akan dilakukan proses *stopword removal* dan penghapusan tanda baca.

2.8.2 *Base Cluster Identification*

Tahap identifikasi *base cluster* merupakan tahap terpenting dalam algoritma *suffix tree clustering*, karena pada tahap ini akan menghasilkan *cluster-cluster* dasar. Pembentukkan *base cluster* dilakukan dengan cara menemukan share *phrase* antar dokumen. Untuk menemukan share *phrase* digunakan struktur data *suffix tree*. Dengan menggunakan struktur data ini, maka setiap dokumen akan direpresentasikan menjadi suatu kalimat. Untuk menemukan *base cluster* dapat

dilakukan dengan cara membuat suatu *invert index* dari *phrase* untuk semua dokumen. Contoh pembentukkan *suffix tree* untuk kalimat *cat ate cheese, mouse ate cheese too,* dan *cat ate mouse too* ditunjukkan pada Gambar 2.8



Gambar 2. 18 Pembentukan *Suffix Tree* (Zamir, 1999)

Pada Gambar tersebut, lingkaran merepresentasikan *node.* Angka pada persegi merepresentasikan grup dokumen dan sebuah *pharse* yang umum bagi semuanya. Label dari *node* mewakili *phrase* umum. Kumpulan dokumen yang menandai *suffix node* yang merupakan keturunan *node* membentuk kelompok dokumen. Oleh karena itu, setiap *node* mewakili sebuah *base cluster*. Selanjutnya, semua *base cluster* yang mungkin (berisi 2 atau lebih dokumen) muncul sebagai simpul di *suffix tree.*

Setiap *base cluster* diberi skor s(B) yang merupakan fungsi dari jumlah dokumen yang dikandungnya, dan kata-kata yang membentuk frasanya. Penghitungan score merupakan suatu fungsi dari jumlah dokumen yang masuk anggota *base cluster* dan jumlah kata yang menyusun *phrase* dari *base cluster*. Fungsi diberikan pada :

$$s(B) = |B|.f(|P|)\text{.....(2.6)}$$

Disini |B| menunjukkan jumlah dokumen dalam *base cluster* dan | P | adalah jumlah kata dalam sebuah frase. Kata yang mempunyai jumlah kemunculan 3 (atau kurang) dan 40% dari jumlah dokumen mempunyai *score 0*. Dalam tabel 2.11 dapat dilihat enam *node* dari contoh yang ditunjukkan pada gambar 2.18 dan *base cluster* yang sesuai.

Tabel 2.10 *Base Cluster* yang terbentuk

Base Cluster	Frasa	Dokumen
A	cat ate	1,3
B	ate	1,2,3
C	cheese	1,2
D	mouse	2,3
E	too	2,3
F	ate cheese	1,2

2.8.3 *Basic Cluster Combination*

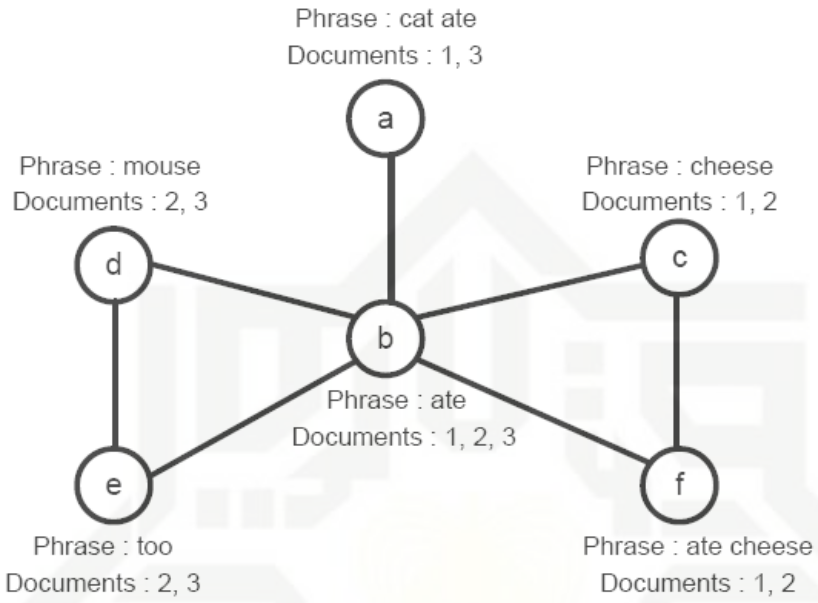
Pada langkah ini, dokumen mungkin dibagikan lebih dari satu frase. Tahap santar *base cluster*, kita harus menghitung dulu nilai *similiarity* antar *base cluster* yang didasarkan pada jumlah dokumen yang *overlap*. Adanya *overlaping* dokumen ini didasarkan karena dokumen memiliki lebih dari satu topik sehingga dokumen dapat memiliki lebih dari satu *phrase* yang dishare Ukuran nilai *similiarity* menggunakan nilai biner. Rumus untuk menghitung nilai *similiarity* antar *base cluster* ditunjukkan pada persamaan (2.7) dan (2.8)

$$|Bm \cap Bn|/|Bm| > 0.5\text{.....(2.7)}$$

$$|Bm \cap Bn|/|Bn| > 0.5\text{.....(2.8)}$$

Pada persamaan di atas, menunjukkan penggunaan nilai *threshold* 0,5 karena nilai tersebut merupakan nilai tengah antara 0 sampai 1. Jika persamaan (2.7) dan (2.8) bernilai benar maka *similiarity* akan bernilai 1 sehingga antara kedua *base cluster* tersebut akan terhubung. Jika salah satu dari persamaan (2.7) dan (2.8) bernilai benar atau keduanya bernilai salah maka *similiarity* akan bernilai 0 sehingga antara kedua *base cluster* tersebut tidak terhubung. Keterhubungan antar

base cluster dapat dilihat dalam bentuk *base cluster graph* yang ditunjukkan pada Gambar 2.19

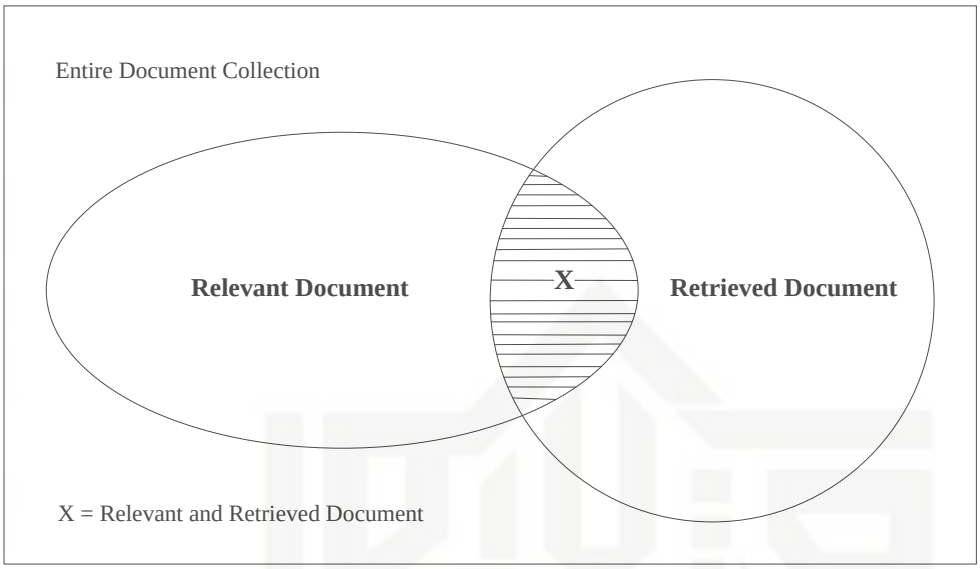


Gambar 2. 19 *Base Cluster Graph* (Zamir, 1999)

2.9 Akurasi Sistem Temu Kembali Informasi *(Information Retrieval)*

Dua ukuran efektifitas yang paling umum yaitu *recall* dan *precision*, diperkenalkan dalam studi Cranfield untuk meringkas dan membandingkan hasil pencarian. Secara intuitif, *recall* mengukur seberapa baik mesin pencari melakukan untuk menemukan semua dokumen yang relevan untuk sebuah *query*, dan *precision* mengukur seberapa baik kinerjanya dalam menolak dokumen yang tidak relevan. Definisi ukuran ini mengasumsikan bahwa, untuk kueri tertentu, ada sekumpulan dokumen yang di *retrieved* dan kumpulan yang tidak *retrieved* (sisa dokumen) (Croft et al., 2015).

Untuk mengetahui kualitas dari sistem temu kembali informasi dilakukan proses perhitungan akurasi. Sistem temu kembali informasi mengembalikan satu set dokumen sebagai jawaban atas *query user*. Ada dua dokumen yang dapat ditemukan di *database* yaitu *relevant document* dan *retrieved document*. *Relevant document* adalah dokumen relevan dengan *query user*. *Retrieved document* adalah dokumen yang diterima pengguna (Cios et al., 2007).



Gambar 2. 20 Relasi antara *Relevan* dan *Retrieved Document* (Cios et al., 2007)

Ada dua ukuran dasar untuk menilai kualitas pengambilan teks yaitu (Gerald, 1988):

a. *Precision*

Precision adalah dokumen *retrieved* yang relevan dengan *query*. Rumus *Precison* yaitu:

$$Precision = \frac{Number\ Of\ Relevant\ Items\ Retrieved}{Total\ Number\ Of\ Items\ Retrieved} \text{.....(2.9)}$$

b. *Recall*

Recall adalah persentase dokumen yang relevan dengan *query* pengguna yang terambil oleh sistem

$$Recall = \frac{Number\ Of\ Relevant\ Items\ Retrieved}{Total\ Number\ Of\ \ Relevant\ items\ in\ the\ collection} \text{...(2.10)}$$

2.10 *Cluster Label Quality*

Cluster Label Quality adalah perbandingan dari jumlah *cluster* yang terbentuk dengan jumlah *cluster* yang dianggrap relevan oleh *user*. Metode evaluasi ini bergantung pada pendapat pengguna nyata tentang kualitas *clustering.* Untuk setiap kelompok yang dibuat, berdasarkan labelnya, pengguna diminta untuk memutuskan apakah *cluster* itu relevan atau tidak. *Cluster* yang relevan

kemungkinan besar akan memiliki label yang ringkas dan bermakna, sementara yang tidak relevan akan memberikan penjelasan yang ambigu (misalnya terlalu panjang) atau tidak masuk akal (misalnya terdiri dari satu kata). (OSIŃSKI, 2003). Berikut adalah parameter perhitungan *Cluster Label Quality:*

$$q = \frac{u}{g} \text{.....(2.11)}$$

Keterangan:

q = *Cluster Label Quality*

u = jumlah *cluster* yang dinyatakan relevan

g = jumlah semua *cluster* yang terbentuk

Nilai q dimulai dari 0.0 (tidak ada *cluster* yang relevan) sampai 1 (semua *cluster* relevan) (OSIŃSKI, 2003). Nilai q adalah derajat *usefulness* atas *cluster* yang terbentuk. Misalnya seorang *user* yang ingin mencari informasi tentang produk elektronik *"Apple"* akan menganggap *cluster* yang berhubungan dengan buah *"Apple"* tidak bermanfaat. (Dyah Mustikasari, Teguh Bharata Adji, 2015)

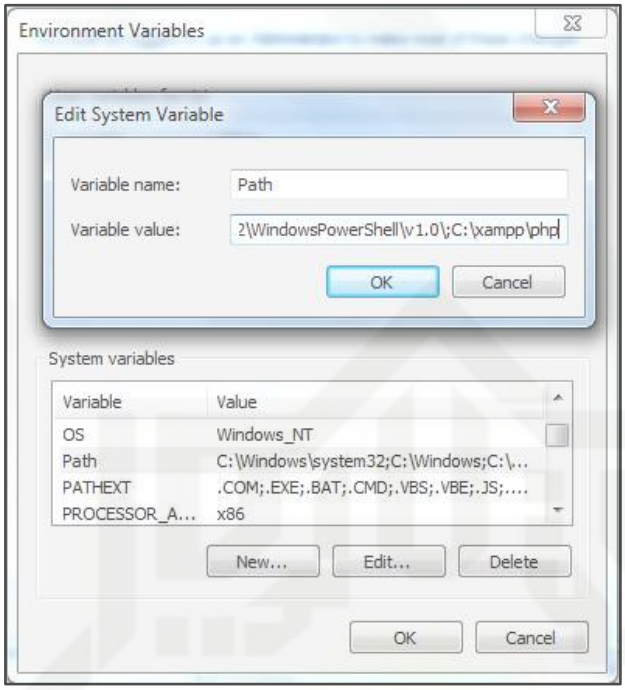
2.11 *Black Box*

Pengujian *Black Box* disebut juga sebagai pengujian fungsional yaitu teknik pengujian berdasarkan informasi dan spesifikasi. Pengujian *Black Box* hanya fokus pada *output* yang dihasilkan sebagai respons terhadap *input* dan kondisi eksekusi yang dipilih (Srinivas Nidhra, 2012).

Pada Penelitian ini pengujian *Blackbox* menggunakan *Codeception. Codeception* merupakan salah satu pengujian aplikasi *Web Php* yang fleksibel dan mudah digunakan.

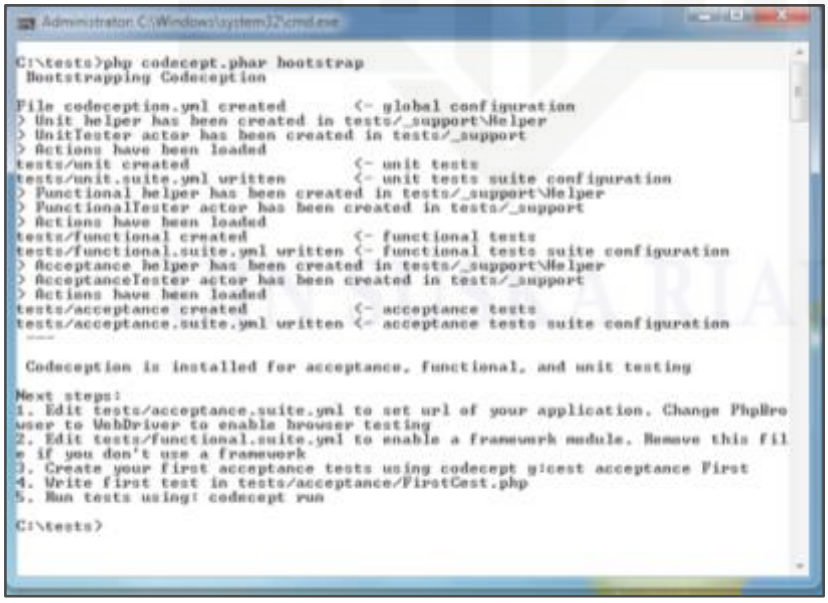
Berikut adalah langkah-langkah konfigurasi *Codeception* sebelum melakukan pengujian.

1. Buka menu *Edit the system environment variable* pilih *Advanced* lalu pilih *Environment Variable.*
2. Pilih *Variable Path* tambahkan pada *Variable Value:* C:\xampp\php



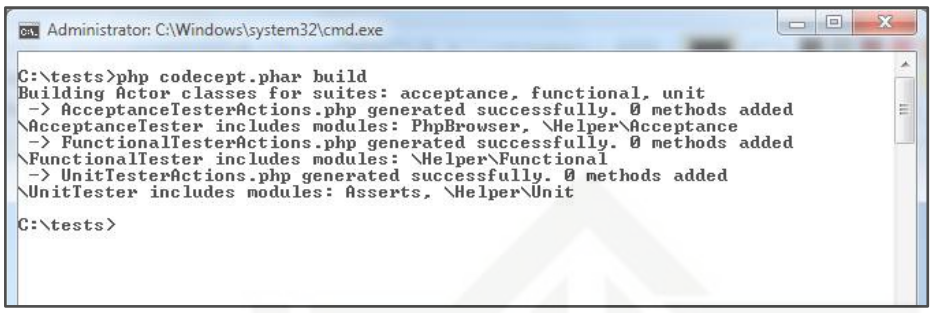
Gambar 2. 21 Langkah 2 *Edit System Variable*

3. Download *Installer Codeception,* lalu pindahkan pada suatu *folder* yang nanti dijadikan sebagai *folder* penyimpanan *file* pengujian.
4. Buka *Command Prompt* dan arahkan pada *folder instalasi.*
5. Ketikkan perintah "php codecept.phar bootstrap", jika maka hasil *output* akan seperti ini:



Gambar 2. 22 Hasil Langkah 5

6. Ketikkan perintah “php codecept.phar build”. Perintah ini untuk membuat *folder tests* pada *folder* instalasi. Jika berhasil akan seperti ini:



Gambar 2. 23 Hasil Langkah 6

Folder tests tersebut terdapat beberapa *file* dan *folder* lainnya yaitu:

a. “_output” yaitu *folder* penyimpanan *report* pengujian
b. “_data” yaitu *folder* penyimpanan SQL untuk pengujian
c. “_support” yaitu *folder* penyimpanan metode *custom* untuk pengujian
d. *Acceptance, Functional* dan *Unit* yaitu folder penyimpanan file pengujian *Acceptance, Functional* dan *Unit.*
e. *File* Konfigurasi untuk tes *Acceptance, Functional* dan *Unit* dengan format .yml

7. Buka *file* acceptance.suite.yml, pada url, ketikkan url sistem yang akan diuji

```
1  # Codeception Test Suite Configuration
2  #
3  # Suite for acceptance tests.
4  # Perform tests in browser using the WebDriver or PhpBrowser.
5  # If you need both WebDriver and PHPBrowser tests - create a separate suite.
6
7  actor: AcceptanceTester
8  modules:
9      enabled:
10         - PhpBrowser:
11             url: http://localhost/cicifix
12         - \Helper\Acceptance
```

Gambar 2. 24 Konfigurasi File *acceptance.suite.yml*

8. Untuk melakukan *Accceptance Test* buatlah *file* pengujian dengan perintah “php codecept.phar generate:cept acceptance namafile” pada *Command Prompt.* Maka *file* namafile.php akan ada pada folder *Acceptance.*
9. Pada *file* tersebut akan dibuat syntax pengujian. Berikut contoh syntax untuk pengujian *Login.*

```php
<?php
$I = new AcceptanceTester($scenario);
$I->amOnPage('/index.php');
$I->see('Sign In');
$I->wantTo('Proses Login');
$I->amOnPage('/login.php');
$I->fillField('username','cici');
$I->fillField('password','cici');
$I->click('Login');
$I->amOnPage('/beranda.php');
$I->see('HOME');
```

Gambar 2. 25 *Acceptance Login*

10. Untuk menjalankan *file* pengujian tersebut ketikkan perintah "codecept.phar run acceptance namafile.php –steps". Maka hasilnya seperti ini. Jika ada keterangan *Passed*, berarti scenario berhasil

```
C:\tests>php codecept.phar run acceptance loginCept --steps
Codeception PHP Testing Framework v2.4.1
Powered by PHPUnit 6.5.7 by Sebastian Bergmann and contributors.

←[1mAcceptance Tests (1) ←[22m-------------------------------------
---------
←[35;1mloginCept:←[39;22m Proses Login
Signature: ←[32mloginCept←[39m
Test: ←[32mtests\acceptance\loginCept.php←[39m
←[33mScenario --←[39m
←[1m I ←[22mam on page "/index.php"
←[1m I ←[22msee "Sign In"
←[1m I ←[22mam on page "/login.php"
←[1m I ←[22mfill field "username","cici"
←[1m I ←[22mfill field "password","cici"
←[1m I ←[22mclick "Login"
←[1m I ←[22mam on page "/beranda.php"
←[1m I ←[22msee "HOME"
←[32;1m PASSED ←[39;22m

-------------------------------------------------------------------------------


Time: 811 ms, Memory: 10.00MB

←[30;42mOK (1 test, 2 assertions)←[0m

C:\tests>_
```

Gambar 2. 26 Hasil *Codeception Login*

2.12 Penelitian Terkait

Tabel 2.14 berikut merupakan penelitian yang terkait yang pernah dilakukan sebelumnya tentang sistem temu kembali informasi dengan *clustering*.

Tabel 2.11: Penelitian Terkait

NO	Peneliti	Judul	Metode	Hasil
1	Andreas Handojo,Adi Wibowo, Jemmy Lay Santo (2011)	*Clustering Search Engine At Petra Christian University Library Using Suffix Tree Clustering*	*Suffix Tree Clustering*	*Cluster* yang dihasilkan dalam sistem cukup baik, dalam arti mengklasifikasikan dokumen yang berkorelasi satu sama lain atau memiliki klasifikasi yang sama. Waktu proses akan meningkat relative sama dengan jumlah dokumen. Proses penggabungan *cluster* dasar ini membutuhkan waktu terlama diantara proses lainnya
2	Hasitha Indika Arumawadu, R.M Kapila Tharanga Rathnayaka, S.K. Illangarathne (2015)	*K-Means Clustering For Segment Web Search Result*	*K-Means*	Penelitian ini menggunakan metode K-means untuk *clustering* hasil pencarian. Untuk hasil pencarian *cluster*ing K-means lebih baik tapi peningkatan jumlah hasil pencarian berarti membutuhkan banyak kekuatan untuk mengelompokkan hasil. Pengguna mengharapkan hasil yang lebih baik serta kecepatan yang lebih baik. Mereka tidak akan puas jika hasilnya membutuhkan waktu lebih lama untuk ditampilkan.
3	Zaenal Fanani (2015)	Pencarian Dokumen Berbasis *Web* Pada *Drive* Lokal dan *Offline Web* Menggunakan Metode *Suffix Tree Clustering*	*Suffix Tree Clustering*	Penerapan metode *Suffix Tree Clustering* dapat mengefisienkan proses pencarian dokumen secara efektif dan optimal, dengan metode ini memungkinkan pencarian kata bisa dilakukan lebih cepat

NO	Peneliti	Judul	Metode	Hasil
4	Dengwei Wang, Libo Lui, Jin Dong, Jiao Zheng (2015)	*Search Result Clustering Algorithm Based on the Suffix Tree*	*Suffix Tree Clustering*	Metode menghitung nilai dan kesamaan *cluster* dasar dibuat lebih baik dan label pengelompokan juga lebih mudah dibaca, lebih deskriptif dan lebih khas. Dokumen ini juga memastikan karakteristik dokumen multi topik
5	Issam Sahmoudi, Abdelmonaime Lachkar (2013)	*Clustering Web Search Result For Effective Arabic Language Browsing*	*Suffix Tree Clustering*	Algoritma *Suffix Tree Clustering* telah berhasil dan banyak digunakan dengan versi bahasa Inggris, Eropa, dan Cina dan Bahasa lainnya. Namun pada penelitian ini, algoritma STC belum pernah diterapkan untuk Hasil Klaster Penelusuran Web Arab *Cluster*ing. Dalam tulisan ini, kami telah menggambarkan bahwa algoritma STC tidak dapat diterapkan secara langsung untuk Bahasa Arab.
6	Ferry Sanjaya (2017)	Pemanfaatan Sistem Temu Kembali Informasi dalam Pencarian Dokumen Menggunakan *Vector Space Model*	*Vector Space Model*	Hasil pencarian dokumen pada sistem dapat mengembalikan hampir semua yang relevan dengan *recall* 1
7	Irmawati (2017)	Sistem Temu Kembali Informasi Pada Dokumen Dengan Metode *Vector Space Model.*	*Vector Space Model*	Kinerja Sistem Temu Kembali yang dikembangkan sudah cukup baik karena rata-rata nilai *recall* hampir 100% dan rata-rata nilai *precision* sekitar 73.6% yang didapatkan, sehingga 73.6% dokumen *online* yang berhasil ditemu-kembalikan relevan dengan *keyword* yang diberikan