BAB 2
LANDASAN TEORI

2.1 Text Mining

Text mining yang juga disebut sebagai *Text* Data *Mining* (TDM) atau *Knowledge Discovery in Text* (KDT), secara umum mengacu pada proses ekstraksi informasi dari dokumen-dokumen teks tak terstruktur (*unstructured*). Teks *mining* dapat didefinisikan sebagai penemuan informasi baru dan tidak diketahui sebelumnya [7].

Tujuan utama teks *mining* (*text mining*) adalah mendukung proses *knowledge discovery* pada koleksi dokumen yang besar. Pada prinsipnya, teks *mining* adalah bidang ilmu multidisipliner, melibatkan *information retrieval* (IR), *text analysis*, *information extraction* (IE), *clustering*, *categorization*, *visualization*, *database technology*, *natural language processing* (NLP), *machine learning*, dan data *mining*. Dapat pula dikatakan bahwa teks *mining* merupakan salah satu bentuk aplikasi kecerdasan buatan (*artifisial intelligence* / AI).

Teks *mining* mencoba memecahkan masalah *information overload* dengan menggunakan teknik-teknik dari bidang ilmu yang terkait. Teks *mining* dapat dipandang sebagai suatu perluasan dari data *mining* atau *knowledgediscovery in database* (KDD), yang mencoba untuk menemukan pola-pola menarik dari basis data berskala besar. Namun teks *mining* memiliki potensi komersil yang lebih tinggi dibandingkan dengan data *mining*, karena kebanyakan format alami dari penyimpanan informasi adalah berupa teks. Teks *mining* menggunakan informasi teks tak tar struktur dan mengujinya dalam upaya mengungkap struktur dan arti Yang "tersembunyi" dalam teks.

Perbedaan mendasar antara teks *mining* dan data *mining* terletak pada sumber data yang digunakan. Pada data *mining*, pola-pola di ekstrak dari basis data yang tar struktur, sedangkan di teks *mining*,pola-pola di ekstrak dari data tekstual (*natural language*). Secara umum, basis data didesain untuk program

dengan tujuan melakukan pemrosesan secara otomatis, sedangkan teks ditulis untuk dibaca langsung oleh manusia.

2.1.1 Ruang Lingkup *Text Mining*

Teks *mining* merupakan suatu proses yang melibatkan beberapa area teknologi. Namun secara umum proses-proses pada teks *mining* mengadopsi proses data *mining*. Bahkan beberapa teknik dalam proses teks *mining* juga menggunakan teknik-teknik data *mining*. Ada empat tahap proses pokok dalam teks *Mining* [7], yaitu pemrosesan awal terhadap teks (*text preprocessing*), transformasi teks (*text transformation*), pemilihan fitur (*feature selection*), dan penemuan pola (*pattern discovery*), di antaranya :

a. *Text Preprocessing*

Tahap ini melakukan analisis semantik (kebenaran arti) dan sintaktik (kebenaran susunan) terhadap teks. Tujuan dari pemrosesan awal adalah untuk mempersiapkan teks menjadi data yang akan mengalami pengolahan lebih lanjut. Operasi yang dapat dilakukan pada tahap ini meliputi *part-of-speech* (PoS) *tagging*, menghasilkan *parse tree* untuk tiap-tiap kalimat, dan pembersihan teks.

b. *Text Transformation*

Transformasi teks atau pembentukan atribut mengacu pada proses untuk mendapatkan representasi dokumen yang diharapkan. Pendekatan representasi dokumen yang lazim digunakan adalah *"bag of word"* dan model ruang vektor (*vector space model*). Transformasi teks sekaligus juga melakukan pengubahan kata-kata ke bentuk dasarnya dan pengurangan dimensi kata di dalam dokumen. Tindakan ini diwujudkan dengan menerapkan *stemming* dan menghapus *stopwords*.

c. *Feature Selection*

Pemilihan fitur (kata) merupakan tahap lanjut dari pengurangan dimensi pada proses transformasi teks. Walaupun tahap sebelumnya sudah melakukan penghapusan kata-kata yang tidak deskriptif (*stopwords*), namun tidak semua kata-kata di dalam dokumen memiliki arti penting. Oleh karena itu, untuk mengurangi dimensi, pemilihan hanya dilakukan terhadap kata-kata yang relevan

yang benar-benar merepresentasikan isi dari suatu dokumen. Ide dasar dari pemilihan fitur adalah menghapus kata-kata yang kemunculannya di suatu dokumen terlalu sedikit atau terlalu banyak.

d. *Pattern Discovery*

Pattern discovery merupakan tahap penting untuk menemukan pola atau pengetahuan (*knowledge*) dari keseluruhan teks. Tindakan yang lazim dilakukan pada tahap ini adalah operasi teks *mining*, dan biasanya menggunakan teknik-teknik data *mining*. Dalam penemuan pola ini, proses teks *mining* dikombinasikan dengan proses-proses data *mining*. Masukan awal dari proses teks *mining* adalah suatu data teks dan menghasilkan keluaran berupa pola sebagai hasil interpretasi atau evaluasi. Apabila hasil keluaran dari penemuan pola belum sesuai untuk aplikasi, dilanjutkan evaluasi dengan melakukan iterasi ke satu atau beberapa tahap sebelumnya. Sebaliknya, hasil interpretasi merupakan tahap akhir dari proses teks *mining* dan akan disajikan ke pengguna dalam bentuk visual.

2.1.2 Ekstraksi Dokumen

Teks yang akan dilakukan proses teks *mining*, pada umumnya memiliki beberapa karakteristik di antaranya adalah memiliki dimensi yang tinggi, terdapat *noise* pada data, dan terdapat struktur teks yang tidak baik. Cara yang digunakan dalam mempelajari suatu data teks, adalah dengan terlebih dahulu menentukan fitur-fitur yang mewakili setiap kata untuk setiap fitur yang ada pada dokumen. Sebelum menentukan fitur-fitur yang mewakili, diperlukan tahap *preprocessing* yang dilakukan secara umum dalam teks *mining* pada dokumen [7], yaitu pemecahan paragraf menjadi kalimat, *case folding*, dan *tokenizing*.

1. Pemecahan kalimat

   Pemecahan kalimat yaitu proses memecahkan dokumen yang panjang menjadi kalimat-kalimat.

2. *Case Folding*

   Case folding adalah mengubah semua huruf dalam dokumen menjadi huruf kecil hanya "**a**" sampai huruf "**z**" yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*.

3. *Tokenizing*

   Tahap *tokenizing*/*parsing* adalah tahap pemotongan *string input* berdasarkan tiap kata yang menyusunnya.

2.2 CNF

CNF merupakan salah satu bentuk normal yang sangat berguna untuk CFG. Bentuk CNF dapat dibuat dari sebuah CFG yang telah mengalami penyederhanaan enggan menghilangkan produksi *useless*, *unit*, dan *empty* (himpunan kosong yang dilambangkan dengan ε). Aturan produksi dalam bentuk normal Chomsky ruas kanannya tepat berupa sebuah terminal atau dua variabel [8].

Langkah-langkah dalam pembentukan CNF yaitu:

1. Biarkan aturan produksi yang sudah dalam bentuk CNF.
2. Lakukan penggantian aturan produksi yang ruas kanannya memuat simbol terminal dan panjang ruas kanan > 1.
3. Lakukan penggantian aturan produksi yang ruas kanannya memuat > 2 simbol variabel.
4. Penggantian dapat dilakukan berkali-kali sampai semua aturan produksi berada dalam bentuk CNF.
5. Selama dilakukan penggantian, kemungkinan akan memperoleh aturan-aturan produksi baru dan memunculkan simbol-simbol variabel baru.

2.3 Sintaksis Bahasa Indonesia

Sintaksis membicarakan penataan dan pengaturan kata-kata itu ke dalam satuan-satuan yang lebih besar, yang disebut satuan-satuan sintaksis, yakni kata, frase, klausa, kalimat dan wacana [9].

2.3.1 Fungsi Sintaksis

Fungsi sintaksis adalah semacam kotak-kotak atau tempat-tempat dalam struktur sintaksis yang ke dalamnya akan diisikan kategori-kategori tertentu. Kotak-kotak itu bernama subjek (S), Predikat (P), Objek (O), komplemen (Komp) dan keterangan (ket). Tetapi di dalam praktek berbahasa urutannya bisa tidak

sama. Secara formal fungsi S dan P harus selalu ada dalam setiap klausa karena keduanya saling berkaitan. Dalam hal ini bisa dikatakan, bahwa S adalah bagian klausa yang menandai apa yang dinyatakan oleh pembicaraan sedangkan P adalah bagian klausa yang menandai apa yang dinyatakan oleh pembicaraan mengenai S [9].

Objek (O) adalah bagian dari verba yang menjadi predikat dalam klausa i. Kehadirannya sangat ditentukan oleh ke transitifan verba tersebut. Artinya, kalau verba bersifat transitif maka objek itu akan muncul, tetapi kalau verbanya bersifat tidak transitif (intransitif) maka objek itu tidak akan ada. Objek afektif adalah objek yang bukan merupakan hasil perbuatan predikat. Sebaliknya, Objek efektif adalah objek yang merupakan hasil perbuatan predikat.

Komplemen (Komp) atau pelengkap adalah bagian dari P verbal yang menjadikan P itu menjadi lengkap. Kedudukannya beda dengan O. Hanya bedanya, kalau O keberadaannya ditentukan oleh sifat verbanya yang transitif, sedangkan komp keberadaannya bukan ditentukan oleh faktor ke transitifan, melainkan oleh faktor keharusan untuk melengkapi P. Unsur S, P, O dan Komp merupakan inti klausa, sedangkan unsur ket merupakan bagian luar inti klausa. Hal ini karena kedudukan ket di dalam klausa lebih fleksibel, artinya dapat berapa pada awal klausa maupun pada akhir klausa.

2.3.2 Kategori Sintaksis

Kategori sintaksis adalah jenis atau tipe kata atau frase yang menjadi pengisi fungsi-fungsi sintaksis. Kategori sintaksis berkenaan dengan istilah nomina (N), verba (V), adjektiva (A), adverba (Adv), numeralia (Num), preposisi (Prep), konjungsi (Konj) dan pronomina (Pron). Dalam hal ini N, V, dan A merupakan kategori utama, sedangkan yang lain merakan kategori tambahan. Pengisi fungsi sintaksis dapat berupa kata dapat pula berupa frase, sehingga di samping ada kata nomina ada pula frase nominal (FN), di samping kata verba ada pula frase verbal (FV) dan di samping ada kata adjektiva ada pula frase adjektiva (FA). Selain itu di samping ada kata berkategori adverbia ada pula frase adverbial (FAdv), di

samping kata berkategori numeralia ada pula frase numeral (FNum) dan di samping kata berkategori preposisi ada pula frase preposisional (FProp) [9].

2.3.3 Penyusunan Frase

Frase adalah satuan sintaksis yang tersusun dari dua buah kata atau lebih, yang di dalam klausa menduduki fungsi-fungsi sintaksis. Dilihat dari kedudukan kedua unsurnya, dibedakan adanya frase koordinatif dan frase *subordinatif*. Frase koordinatif yaitu yang kedudukan kedua unsurnya sederajat, sedangkan frase *subordinatif*, yaitu yang kedudukan kedua unsurnya tidak sederajat. Ada yang berkedudukan sebagai unsur atasan yang disebut sebagai inti frase dan ada yang berkedudukan sebagai bawahan, yang disebut tambahan penjelas frase [9]. Frase dibagi menjadi beberapa kelompok:

1. Frase Nomina Koordinatif
2. Frase Nomina *Subordinatif*
3. Frase Verba Koordinatif
4. Frase Verba *Subordinatif*
5. Frase Adjektiva Koordinatif
6. Frase Adjektiva *Subordinatif*
7. Frase Preposisional

1. Penyusunan Frase Nomina

Frase Nomina (FN) adalah frase yang dapat mengisi fungsi subjek atau objek di dalam klausa. Menurut strukturnya dapat dibedakan adanya frase nomina koordinatif (FNK) dan frase nomina *subordinatif* (FNS) [9].

1. Penyusunan Frase Nomina Koordinatif

   FNK dapat disusun dari:

   a. Dua buah kata berkategori nomina yang merupakan pasangan dan antonim relasional.
   b. Dua buah kata berkategori nomina yang merupakan anggota dari suatu medan makna.

2. nyusunan Frase Nomina *Subordinatif*

Frase nomina *subordinatif* dapat disusun dari nomina + nomina (N + N), nomina + verba (N + V), nomina + adjektiva (N + A), adverbia + nomina (Adv + N), nomina + adverbia (N + Adv), nomina + numeralia (N + Num), numeralia + nomina (Num + N), dan nomina + demonstrativa (N + Dem).

2. Penyusunan Frase Verba

Frase verba adalah frase yang mengisi atau menduduki fungsi predikat pada sebuah klausa. Dilihat dari kedudukan di antara kedua unsur pembentuknya dibedakan adanya frase verba koordinatif (FVK) dan frase verba *subordinatif* [9].

1. Penyusunan Frase Verba Koordinatif (FVK), frase verba koordinatif dapat disusun dari:
   a. Dua buah kata berkategori verba yang merupakan anggota dari antonim relasional dan memiliki makna gramatikal ‘menggabungkan’ sehingga di antara keduanya dapat disisipkan kata ‘dan’.
   b. Dua buah kata berkategori verba yang merupakan anggota dari satu medan makna dan memiliki makna gramatikal ‘menggabungkan’ sehingga di antara keduanya dapat disisipkan kata ‘dan’.
2. Penyusunan Frase Verba *Subordinatif* (FVS), frase verba *subordinatif* dapat disusun dari adverbia + verba (Adv + V), verba + adverbia (V + Adv), verba + nomina (V + N), verba + adjektiva (V + A).

3. Penyusunan Frase Adjektiva

Frase adjektiva adalah frase yang mengisi atau menduduki fungsi predikat dalam sebuah klausa adjektiva. Dilihat dari kedudukan kedua unsurnya dibedakan adanya frase adjektiva koordinatif (FAK) dan frase adjektiva subordinatif (FAS) [9].

1. Penyusunan Frase Adjektiva Koordinatif (FAK), frase adjektiva koordinatif (FAK) dapat disusun dari:

a. Dua buah kata berkategori adjektiva yang merupakan anggota dari antonim relasional dan memiliki makna gramatikal ‘pilihan’, sehingga di antara kedua dapat disisipkan kata ‘atau’.
b. Dua buah kata berkategori adjektiva yang merupakan anggota dari pasangan bersinonim, dan memiliki makna gramatikal ‘sangat’.
c. Dua buah kata berkategori adjektiva yang maknanya sejalan tidak bertentangan dan memiliki makna gramatikal ‘himpunan’ sehingga di antara keduanya dapat disisipkan kata ‘dan’.
d. Dua buah kata berkategori adjektiva yang maknanya tidak sejalan (bertentangan) dan memiliki makna ‘berkebalikan’ sehingga di antara kedua unsurnya harusnya disisipkan kata ‘tetapi’.

2. Penyusunan Frase Adjektiva Subordinatif (FAS), frase adjektiva subordinatif disusun dengan struktur adjekiva + nomina (A + N), adjektiva + adjektiva (A + A), adjektiva + verba (A + V), adverbia + adjektiva (Adv + A), dan adjektiva + adverbia (A + Adv). Aturannya sebagai berikut Tabel 2.1.

Tabel 2.1 Penyusunan FAS

FAS	Makna
A + N	memiliki makna gramatikal ‘seperti’ apabila unsur pertama berkategori adjektiva dan memiliki komponen makna (+ warna) dan unsur kedua berkategori nomina dan memiliki komponen makna (+ perbandingan), sehingga di antara kedua unsurnya dapat disisipkan kata ‘seperti warna’
A + A	memiliki makna gramatikal ‘jenis warna’ dapat disusun dari Unsur pertama berkategori adjektiva dan berkomponen makna (+ warna) dan unsur kedua berkategori adjektiva dan berkomponen makna (+ cahaya).
A + V	bermakna gramatikal ‘untuk’ dapat disusun apabila unsur pertama berkategori adjektiva dan memiliki komponen makna

FAS	Makna
	(+ sikap batin), sedangkan unsur kedua berkategori verba dan memiliki komponen makna (+ tindakan) atau (+ kejadian)
Adv + A	memiliki makna gramatikal ‘ingkar’ dapat disusun apabila unsur pertama berkategori adverbia yang berkomponen makna (+ ingkar) dan unsur kedua berkategori adjektiva dan berkomponen makna (+ keadaan) atau (+ sikap batin).
A + Adv	a. bermakna gramatikal ‘sangat’ atau ‘tingkat superlatif’ dapat disusun apabila unsur pertama berkategori adjektiva dan bermakna gramatikal (+ keadaan), sedangkan kedua berkategori adverbia dan berkomponen makna (+ paling) dalam bentuk kata ‘sekali’

4. FAS yang berstruktur A + Adv dan Penyusunan Frase Preposisional

Frase preposisional adalah frase yang berfungsi sebagai pengisi fungsi keterangan di dalam sebuah klausa. Frase preposisional ini buka frase koordinatif maupun subordinatif, melainkan fase eksosentrik. Jadi, di dalam frase ini tidak ada unsur inti dan unsur tambahan. Kedua unsurnya merupakan satu kesatuan yang utuh. Frase preposisional tersusun dari kata berkategori preposisi dan kata atau frase berkategori nomina [9].

2.3.4 Penyusunan Klausa

Klausa adalah satuan sintaksis yang bersifat prediktif. Artinya, di dalam satuan atau konstruksi itu terdapat sebuah predikat, bila di dalam satuan tidak terdapat predikat, maka satuan itu bukan disebut klausa. Kedudukan predikat ini sangat penting, sebab jenis dan kategori dari predikat itulah yang menentukan hadirnya fungsi subjek (S), fungsi objek (O), fungsi pelengkap, dan sebagainya [9]. Berdasarkan kategori yang mengisi fungsi P dapat dibedakan menjadi:

1. Klausa verba
2. Klausa nomina
3. Klausa adjektiva
4. Klausa preposisional
5. Klausa numeral

1. Penyusunan Klausa Verba

Secara semantik ada tiga buah jenis verba, yaitu verba tindakan, verba kejadian, dan verba keadaan [9].

1. Klausa Verba Tindakan, klausa Verba tindakan dibagi menjadi beberapa kategori:
   a. Klausa Verba Tindakan Bersasaran Tak lengkap

      Klausa tindakan bersasaran tak lengkap dapat disusun dari sebuah verba berkomponen makna (+ tindakan) dan (+ sasaran), sehingga klausanya memiliki fungsi sintaksis S, P, O. Dalam hal ini komponen makna V yang mengisi fungsi P harus sejalan dengan komponen makna yang dimiliki fungsi S dan fungsi O.

   b. Klausa Verba Tindakan Bersasaran lengkap, klausa tindakan bersasaran lengkap dapat disusun dari sebuah verba berkomponen makna (+ tindakan), (+ sasaran), dan (+ pelengkap), sehingga klausanya memiliki fungsi S, P, O, dan Pel. Dalam hal ini tentu saja komponen makna yang dimiliki P harus sejalan dengan komponen makna yang dimiliki fungsi-fungsi lain.
   c. Klausa Verba Tindakan Tak Bersasaran, klausa verba tindakan tidak bersasaran dapat disusun dari sebuah verba yang memiliki komponen makna (+ tindakan) dan (- sasaran), sehingga klausanya hanya memiliki fungsi S dan fungsi P. Dalam hal ini tentu saja komponen makna yang dimiliki P harus sejalan dengan komponen makna yang dimiliki S.
2. Klausa Verba Kejadian, klausa verba kejadian dapat disusun dari predikat verba yang memiliki komponen makna (+ kejadian). Dalam hal fungsi sintaksis yang wajib hadir adalah fungsi S dan fungsi P. Fungsi S berupa nomina yang mengalami kejadian seperti disebutkan oleh predikat.

3. Klausa Verba Keadaan, klausa Verba keadaan dapat disusun dari predikat verba yang memiliki komponen makna (+ keadaan). Dalam hal ini fungsi sintaksis yang muncul hanyalah fungsi S dan fungsi P. Fungsi S berupa nomina yang mengalami keadaan seperti yang disebutkan oleh predikat.

Klausa nomina hanya memiliki fungsi wajib S dan P. Klausa nomina ini dapat disusun dari fungsi S yang berupa kata atau frase berkategori nomina dan P yang berupa kata atau frase berkategori nomina [9]. Klausa nomina, antara lain, dapat disusun jika:

1. Nomina yang mengisi fungsi S merupakan jenis (spesifik) dari nomina pengisi fungsi P (generik).
2. Nomina yang mengisi fungsi S mempunyai nama pada nomina pengisi fungsi P.
3. Nomina pengisi fungsi P adalah profesi (jabatan, pekerjaan) bagi nomina pengisi fungsi S.
4. Nomina pengisi fungsi P adalah relasi bagi nomina pengisi fungsi S.
5. Nomina pengisi fungsi S mempunyai ciri atau sifat khas yang disebutkan oleh nomina pengisi fungsi S.

3. Penyusunan Klausa Adjektiva

Klausa adjektiva memiliki fungsi wajib S dan P. Klausa adjektiva dapat disusun dari fungsi S yang berkategori N dan fungsi P yang berkategori A [9]. Klausa adjektiva ini dapat disusun jika:

1. Fungsi P yang berkategori adjektiva memiliki komponen makna (+ keadaan fisik).
2. Fungsi P yang berkategori adjektiva memiliki komponen makna (+ sifat batin).
3. Fungsi P yang berkategori adjektiva memiliki komponen makna (+ perasaan batin).

4. Penyusunan Klausa Preposisional

Klausa preposisional adalah klausa yang fungsi P nya diisi oleh frase preposisional [9].

Klausa preposisional ini lazim digunakan dalam bahasa ragam lisan dan ragam bahasa nonformal. Dalam ragam formal fungsi P akan diisi oleh sebuah verba dan frase preposisinya menjadi fungsi keterangan.

5. Penyusunan Klausa Numeral

Klausa numeral adalah klausa yang fungsi P nya diisi oleh frase numeral [9].

2.3.5 Penyusunan Kalimat

Satuan bahasa yang menjadi inti dalam pembicaraan sintaksis adalah kalimat yang merupakan satuan di atas klausa dan di bawah wacana. Kalimat adalah satuan sintaksis yang disusun dari konstituen dasar, yang biasanya berupa klausa, dilengkapi dengan konjungsi bila diperlukan, serta disertai dengan intonasi final [9].

Intonasi final merupakan syarat penting dalam pembentukan sebuah kalimat dapat berupa intonasi deklaratif (yang dalam bahasa ragam tulis diberi tanda titik), intonasi interogatif (yang dalam bahasa ragam tulis diberi tanda tanya), intonasi imperatif (yang dalam bahasa ragam tulis diberi tanda seru), dan intonasi interjektif (yang dalam bahasa ragam tulis diberi tanda seru). Tanpa intonasi final ini sebuah klausa tidak akan menjadi sebuah kalimat.

1. kalimat Sederhana

Kalimat sederhana adalah kalimat yang dibentuk dari sebuah klausa dasar atau klausa sederhana, yaitu klausa yang fungsi-fungsi sintaksisnya hanya diisi oleh sebuah kata atau sebuah frase sederhana [9].

Semua kalimat sederhana yang dibentuk dari klausa dasar ini adalah kalimat deklaratif positif. Lalu, berdasarkan kategori klausanya dapat disusun kalimat dasar atau kalimat sederhana yang berkategori seperti berikut Tabel 2.2.

Tabel 2.2 kalimat sederhana

Kalimat sederhana	Pengertian
Kalimat verba mono transitif	kalimat yang predikatnya berupa verba yang memiliki komponen makna (+ tindakan) dan (+ sasaran).
Kalimat verba bitransitif	kalimat yang predikatnya berupa verba yang memiliki komponen makna (+ tindakan), (+ sasaran), dan (+ pelengkap).
Kalimat verba intransitif	kalimat yang predikatnya berupa verba yang memiliki komponen makna (+ tindakan) dan (- sasaran).
Kalimat nomina	kalimat yang predikatnya berkategori nomina atau dibentuk dari sebuah klausa nomina dan intonasi final.
Kalimat adjektiva	kalimat yang predikatnya berkategori adjektiva dibentuk dari sebuah klausa adjektiva dan intonasi final.
Kalimat preposisional	kalimat yang predikatnya berupa frase preposisional atau dibentuk dari sebuah klausa preposisional dan intonasi final.
Kalimat *numeral*	kalimat yang predikatnya berupa frase *numeral* dibentuk dari sebuah klausa *numeral* dan intonasi final.

2. Kalimat Luas

Dalam praktek berbahasa yang sebenarnya sering kali tidak cukup hanya dengan menggunakan kalimat dasar atau kalimat sederhana. Sebuah kalimat

biasanya terangkum informasi lebih banyak di dalamnya dan disebut dengan kalimat luas [9]. Beberapa cara dalam menyusun kalimat luas di antaranya:

1. Disusun dengan cara memberi fungsi keterangan lebih dari satu pada sebuah kalimat.
2. Disusun dengan cara memberi keterangan tambahan pada fungsi subjek, fungsi objek, atau fungsi lainnya pada kalimat tersebut.
3. Disusun dengan memberi keterangan aposisi pada fungsi subjek, objek, atau fungsi lainnya pada kalimat itu.
4. Disusun dengan cara menyisipkan sebuah klausa pada klausa lain. Klausa yang disisipkan disebut klausa sisipan, dan klausa yang tersisipi disebut klausa utama. Penyisipan dilakukan dengan menggunakan konjungsi.

2.4 POS *Tag*

POS (*Part-of-Speech*) *Tag* merupakan cara untuk kategori kelas kata, seperti kata benda, kata kerja, kata sifat, dan lain-lain. POS *Tagging* adalah aktivitas anotasi setiap kata dengan nilai *part-of-speech tag* yang sesuai [10]. Informasi nilai POS *Tag* ini merupakan hal yang mendasar bagi keperluan pengolahan bahasa manusia .

POS *Tag* yang digunakan sebagai bantuan dalam mengenali *token/tag* setiap kata diambil dari POS *Tag* Indonesia yang dibuat oleh Arawinda Dinakaramani, Fam Rashel, Andry Luthfi, dan Ruli Manurung. POS *Tag* diambil pada tanggal 19 September 2016.

POS *Tag* akan mengenali kata mana yang termasuk kata benda, kata kerja, kata sifat, kata keterangan, kata depan, kata sambung, kata ganti benda, dan kata bilangan. POS *Tag* ini menggunakan pendekatan *Rule-Based* berdasarkan aturan tata bahasa Indonesia. Pertama-tama, POS *Tag* akan melakukan *tokenizer* terhadap teks menggunakan kamus bahasa Indonesia. Selanjutnya kata-kata yang termasuk ke dalam jenis *closed-class word* diproses. Lalu setiap kata yang ambigu diproses menggunakan aturan-aturan yang sudah didefinisikan untuk menemukan

kelas kata yang tepat. Secara garis besar, fungsionalitas POS *Tagger* dapat terbagi menjadi 6 modul besar , yaitu:

1. *Multi-word Expression Tokenizer*

Merupakan modul *tokenizer* yang memperhatikan ekspresi frase yang terdiri lebih dari satu kata. Dokumen diproses oleh *tokenizer* untuk menghasilkan *token-token* untuk diberikan *part-of-speech* yang sesuai kemudian.

2. *Name Entity Recognizer*

Merupakan modul yang secara khusus menangani *token-token* entitas (*proper-noun*), seperti nama orang, nama tempat, nama organisasi, dan lain-lain.

3. *Closed-Class Word Tagging*

Merupakan modul yang berguna untuk menangani *token-token* yang termasuk ke dalam kategori *closed-class word* menggunakan sebuah kamus.

4. *Open-Class Word Tagging*

Merupakan modul yang berguna untuk menangani *token-token* yang termasuk ke dalam kategori *open-class word* dengan memanfaatkan *MorphInd*.

5. *Rule-Based Tagging*

Merupakan modul yang berguna untuk menyelesaikan *token-token* ambigu berdasarkan aturan-aturan yang telah didefinisikan sebelumnya.

6. *Resolver*

Merupakan modul yang berguna untuk memberikan *tag* kepada *token-token* yang tidak diketahui dapat dilihat pada Tabel 2.3.

Tabel 2.3 Aturan produksi

Nama Kelompok Kata	Diubah Menjadi
Noun / Kata Benda (N)	NN, NND, NNP, PRP, atau WH
Adjective / Kata Sifat (A)	JJ
Verb / Kata Kerja (V)	VB
Adverb / Kata Keterangan (Adv)	RB
Numeralia (Num)	OD dan CD
Preposisi (P)	IN
Demonstrative (Dem)	PR
Konjungsi koordinatif	CC
Artikula	DT
Verba bantuan	MD
Kata ingkar	NEG
Partikel	RP
Konjungsi subordinatif	SC
Simbol	SYM
Tanda baca	Z

2.5 Algoritma *Zhu-Takaoka*

Algoritma *Zhu-Takaoka* merupakan salah satu algoritma pencocokan *string* (*String Matching*) yang merupakan pengembangan dari algoritma *Boyer-Moore*. Algoritma ini dipublikasikan oleh Zhu Rui Feng dan Tadao Takaoka pada tahun 1986 [3]. Dalam penelitiannya, Zhu dan Takaoka menyebut algoritma pencocokan *string* ini sebagai BM" *Algorithm* (*Boyer-Moore Algorithm*) karena merupakan modifikasi dari algoritma pencocokan *string Boyer-Moore*, yaitu algoritma yang dibuat oleh Boyer R.S. dan Moore J.S. Algoritma Zhu-Takaoka dan algoritma *Boyer-Moore* memiliki ciri yang sama dalam proses pencarian *string*, yaitu terdapat tahap *Preprocessing*, *Right-to-left scan*, *Bad character rule* dan *Good-suffix rule* [3]. Sementara itu, perbedaan dari kedua algoritma tersebut terletak pada tahap penentuan *Bad character rule*. Dalam Boyer-Moore, bad character hanya terdiri dari *array* satu dimensi, sedangkan dalam *Zhu-Takaoka array* dimodifikasi menjadi dua dimensi. Berikut ini adalah penjelasan mengenai karakter algoritma *Zhu-Takaoka*.

Tahap yang akan dilakukan pertama kali adalah menentukan nilai tabel ztBc(*Zhu-Takaoka Bad Character*) dan ztGs(*Zhu-Takaoka Good Suffix*). Tabel ztBc digunakan pada saat melakukan pencocokan *pattern* dengan sumber teks dan ditemukan ketidakcocokan karakter pada saat pertama kali dibandingkan, maka nilai pergeseran selanjutnya diambil dari nilai tabel ztBc. Sedangkan untuk tabel ztGs digunakan pada saat melakukan pencocokan *pattern* dengan sumber teks dan ditemukan kecocokan pada saat pertama kali dibandingkan, maka nilai pergeseran selanjutnya diambil dari nilai tabel ztGs.

Sebagai langkah awal, pengguna akan memasukan panjang *pattern* atau kata yang dicocokkan. Setelah kata kunci atau panjang *pattern* dimasukkan, langkah selanjutnya adalah mencari nilai *Bad Character* dan *Good Suffix* dari *pattern* yang telah dimasukkan. Setelah nilai tabel *Bad Character* dan *Good Suffix* terisi semua, langkah selanjutnya adalah melakukan pencocokan, yaitu dengan cara membandingkan setiap karakter dari mulai karakter paling kanan. Apabila pada perbandingan pertama sudah tidak ada kecocokan, maka nilai perpindahan yang digunakan adalah nilai perpindahan dari table *Bad Character* [3]. Tetapi jika pada perbandingan pertama ada kecocokan, nilai perpindahan selanjutnya bisa dilihat dari tabel *Good Suffix*.

1. Preprocessing

Preprocessing adalah proses yang dilakukan sebelum proses inti, yaitu proses pencarian *pattern* dalam sebuah teks. Dalam algoritma *Zhu-Takaoka*, *preprocessing* meliputi proses pencarian nilai pergeseran karakter (*good-suffix shift*) yang nilainya ditentukan dalam *good suffix preprocessing*, dan pergeseran karakter jika ditemukan ketidakcocokan karakter (*bad-character shift*) yang nilainya ditentukan dalam *bad character preprocessing*.

2. Right-to-Left Scan Rule

Sebagaimana disebutkan sebelumnya, bahwa *Zhu-Takaoka* merupakan salah satu algoritma yang melakukan pencocokan *string* dari kanan ke kiri, maka proses pencarian algoritma ini dilakukan dengan teknik *right-to-left scan rule*. Teknik ini

dilakukan dengan membandingkan *pattern* dengan sumber teks. Perbandingan dilakukan dari karakter paling kanan pada *pattern* dengan karakter paling kanan pada sumber teks. Apabila dalam proses tersebut ditemukan kecocokan, maka perbandingan akan dilanjutkan dengan bergerak ke kiri sampai pada karakter pertama dari *pattern*. Tetapi apabila ditemukan ketidakcocokan, maka akan dilakukan pergeseran yang jumlah pergeserannya ditentukan oleh aturan *bad-character shift* atau *good-suffix shift*.

3. Bad-Charcter Shift Rule

Ketika ditemukan ketidakcocokan dalam proses perbandingan, aturan *bad-character shift* dibutuhkan untuk menghindari pengulangan perbandingan. Dalam aturan *bad-character shift*, jumlah pergeseran yang dilakukan disimpan dalam bentuk tabel *array* dua dimensi. Tabel tersebut terdiri dari beberapa kolom, yaitu kolom karakter dan kolom *shift* yang menunjukkan jumlah pergeseran.

4. Good Suffix Shift Rule

Aturan *good-suffix shift* dibuat untuk menangani kasus di mana terdapat pengulangan karakter pada *pattern*.

5. Langkah – langkah pencarian

Proses inti pencarian Algoritma *Zhu-Takaoka* yaitu dilakukan dengan teknik *Right-to-left scan rule*. Teknik ini membandingkan *pattern* yang dicari dengan sumber teks dimulai dari kanan ke kiri.

Karakteristik algoritma *Zhu-Takaoka*, di antaranya :

1. Pengembangan algoritma *Booyer-Moore.*
2. Menggunakan *array* dua dimensi.
3. Arah pencocokan dari kanan ke kiri.

2.6 Algoritma *Levenshtein*

Algoritma Levenshtein merupakan algoritma pencarian jumlah perbedaan *string* yang ditemukan oleh Vladimir Levenshtein, seorang ilmuwan Rusia, pada tahun 1965. Algoritma ini digunakan secara luas dalam berbagai bidang, misalnya mesin pencari, pengecek ejaan (*spell checking*), pengenalan pembicaraan (*speech recognition*), pengucapan dialek, analisis DNA, dan lain-lain [4].

Dua buah *string* dibandingkan untuk mengetahui nilai *edit distance*-nya. *Edit distance* adalah suatu pengukuran (metrik) yang dihasilkan melalui perhitungan jumlah perbedaan yang terdapat pada dua *string*. *Edit distance* antara dua *string* didefinisikan sebagai jumlah minimum perubahan yang diperlukan untuk mengganti suatu *string* dengan *string* lain, dengan operasi penambahan (*insert*), operasi penggantian (*substitute*), operasi penghapusan (*delete*).

Algoritma ini menggunakan matriks dua dimensi dalam membandingkan kedua *string*. Algoritma ini berjalan mulai dari pojok kiri atas hingga ke pojok kanan bawah. Seluruh baris pertama diisi dengan nilai berurut mulai dari 0 hingga n begitu juga dengan kolom pertama [4].

		T	O	P
	0	1	2	3
T	1			
O	2			
P	3			
I	4			

Pada gambar di atas kata “top” akan dilakukan perbandingan dengan kata “topi” untuk melihat nilai *edit distance*-nya. Pengisian dilakukan mulai dari blok [1,1] dengan membandingkan karakter yang ada pada blok [0,1] dan blok [1,0].

Apabila karakter yang dibandingkan sama, turunkan nilai yang ada pada blok [0,0] yaitu angka 0 [4].

		T	O	P
	0	1	2	3
T	1	0		
O	2			
P	3			
I	4			

Pengisian blok berikutnya dilakukan dengan membandingkan karakter dengan baris yang sama. Apabila karakter yang dibandingkan tidak sama, maka nilai blok [0,2] ditambahkan dengan angka 1. Sama halnya dengan blok[1,1] dan blok [0,1]. Setelah itu lakukan perbandingan nilai dari blok [0,2], [1,1], dan [0,1] untuk mencari nilai minimumnya. Selanjutnya simpan pada blok [1,2] [4].

		T	O	P
	0	1^{+1}	2^{+1}	3
T	1	0^{+1}	1	
O	2			
P	3			
I	4			

Langkah selanjutnya isi semua blok sampai akhir dengan langkah yang sama. Hasil akhir dari pengisian tabel *Levenshtein* di atas [4].

		T	O	P
	0	1	2	3
T	1	0	1	2
O	2	1	0	1
P	3	2	1	0
I	4	3	2	1

Nilai *edit distance* dapat dilihat dari baris dan kolom terakhir, yaitu angka 1. Nilai inilah yang akan mengetahui seberapa banyak perbedaan karakter antara dua *string*.

2.7 Algoritma CYK

Algoritma CYK merupakan algoritma *parsing* dan keanggotaan (*membership*) untuk tata bahasa bebas konteks. Syarat untuk menggunakan algoritma ini adalah tata bahasa harus berada dalam bentuk CNF [5]. Tujuan dari algoritma ini adalah untuk menunjukkan apakah *string* dapat diterima dari tata bahasa itu.

Proses *parsing* untai dengan algoritma CYK menggunakan struktur data sebuah *array* dua dimensi dengan jumlah baris dan kolom n x n. Nilai n ditentukan dari jumlah kata dalam sebuah kalimat. Proses *parsing* dari algoritma ini memanfaatkan hasil *parsing* sebelumnya untuk menentukan apakah proses yang sedang berlangsung dapat diterima atau tidak [5].

Sebagai contoh terdapat sebuah aturan produksi dengan variabel awal S [5].

S → EF | EC

E → AB

F → CD

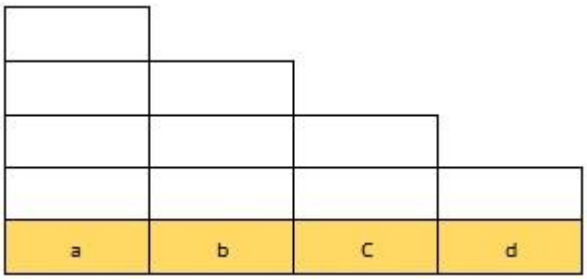
A → a

B → b

C → c

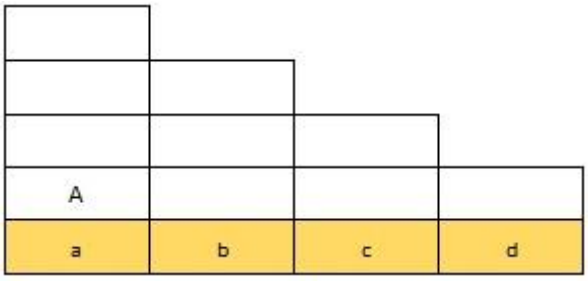
D → d

String yang akan diperiksa misalnya “abcd”. Langkah pertama yang dilakukan adalah dengan membuat tabel dua dimensi seperti Gambar 2.1 [5]:



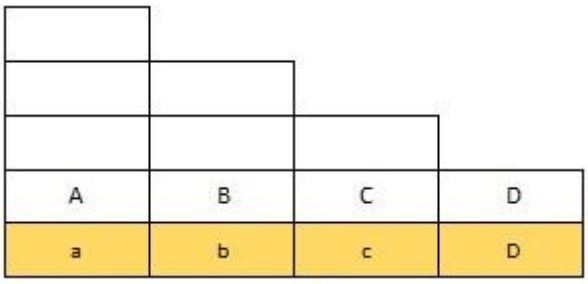
Gambar 2.1 Pembentukan tabel *parsing*

Setiap kata yang ada pada *string* ditempatkan pada masing-masing blok. Setiap blok yang kosong diisi berdasarkan variabel yang sesuai dengan aturan produksi dari CNF yang sudah dibentuk sebelumnya. Sebagai contoh pada baris pertama dan kolom pertama diisi dengan variabel yang dapat menghasilkan huruf a, yaitu variabel A. Hasilnya dapat dilihat pada gambar Gambar 2.2 [5]:



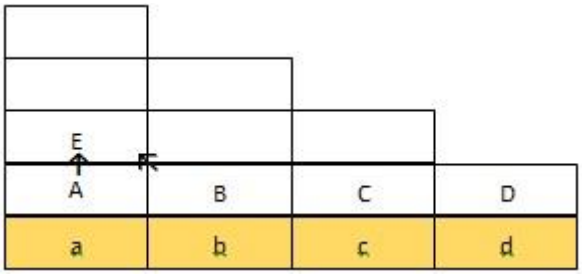
Gambar 2.2 Pengisian tabel

Blok selanjutnya diisi dengan langkah yang sama, sehingga dapat dilihat baris pertama isiannya seperti Gambar 2.3 [5]:



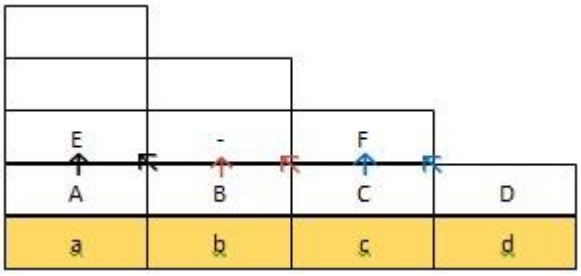
Gambar 2.3 Pengisian baris pertama

Pada baris selanjutnya cara pengisian dilakukan dengan melihat variabel pada dua blok, kemudian dilihat variabel mana yang menghasilkan kedua variabel tersebut. Sebagai contoh baris kedua kolom pertama diisi dengan melihat variabel pada blok pertama dan blok dua, yaitu variabel A dan B yang termasuk ke dalam variabel E, sehingga cara pengisiannya terlihat seperti pada Gambar 2.4 [5]:



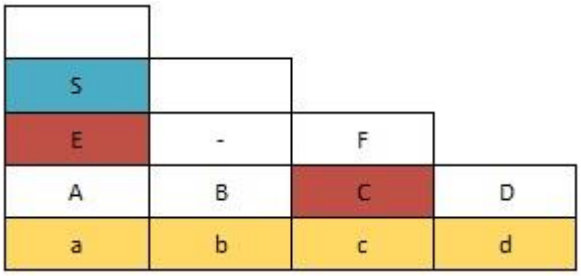
Gambar 2.4 Pengisian tabel dengan variabel B

Blok selanjutnya diisi dengan langkah yang sama sesuai dengan blok yang akan diisi. Apabila pengisian dilakukan pada blok kedua, variabel diambil dari kolom kedua dan ketiga. Untuk lebih jelasnya dapat dilihat pada Gambar 2.5 [5]:

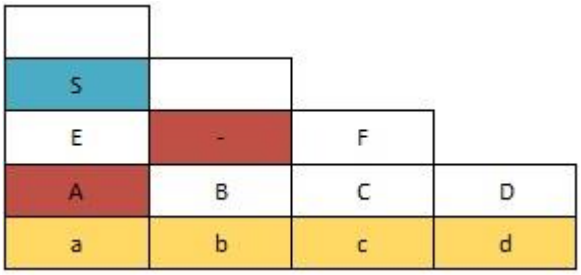


Gambar 2.5 Pengisian tabel baris kedua

Pengisian pada baris ketiga dan selanjutnya hampir sama seperti baris kedua, namun terdapat perbedaan dari cara pengecekan yang dilakukan. Pengisian baris ketiga dilakukan dari baris pertama hingga sebelum baris ketiga (n-1). Cara pengecekan dapat dilihat seperti gambar Gambar 2.6 dan Gambar 2.7 [5]:



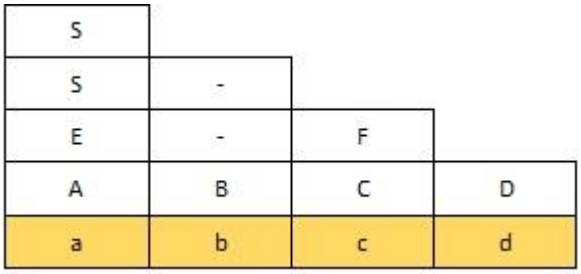
Gambar 2.6 Cara pengecekan langkah pertama



Gambar 2.7 Cara pengecekan langkah kedua

Warna biru menunjukkan blok yang akan diisi, sedangkan warna merah merupakan blok yang dicek variabelnya. Pada kedua gambar di atas dapat dilihat untuk baris ketiga cara pengecekan dilakukan secara diagonal dan vertikal. Gambar pertama pada saat pengecekan ditemukan variabel yang menghasilkan variabel E dan C. Cara pengisian untuk blok selanjutnya sama seperti gambar Gambar 2.7.

Hasil akhir dari pengecekan *string* di atas dapat dilihat pada Gambar 2.8 [5]:



Gambar 2.8 Hasil *parsing string*

Dapat dilihat dari gambar di atas *string* “a b c d” termasuk ke dalam aturan produksi yang sudah dibuat, artinya *string* tersebut dapat dikatakan sebagai kalimat dalam contoh ini. Jika baris dan kolom terakhir tidak menghasilkan tanda “-”, *string* tersebut belum dapat dikenali oleh algoritma CYK.

2.8 Pemrograman Berorientasi Objek

Metodologi berorientasi objek adalah strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya. Metodologi berorientasi objek merupakan cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metode berorientasi objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas [11]. Beberapa konsep dasar dari pemrograman berorientasi objek adalah:

2.1 Kelas (*Class*)

Kelas adalah kumpulan objek-objek dengan karakteristik yang sama. Sebuah kelas akan mempunyai sifat (*attribut*), tingkah laku (operasi/metode), hubungan (*relationship*) dan arti. Secara teknis, kelas adalah sebuah struktur tertentu dalam pembuatan perangkat lunak. Kelas merupakan bentuk struktur pada kode program yang menggunakan metodologi berorientasi objek. Kelas secara fisik adalah berkas atau *file* yang berisi kode program, di mana kode program merupakan semua hal yang terkait dengan nama kelas.

2.2 Objek (*Object*)

Objek adalah abstrak dan sesuatu yang mewakili dunia nyata seperti benda, manusia, satuan organisasi, tempat, kejadian, struktur, status, atau hal-hal lain yang bersifat abstrak. Objek merupakan sebuah entitas yang mampu menyimpan informasi dan mempunyai operasi yang dapat diterapkan atau dapat berpengaruh pada status objeknya. Secara teknis, sebuah kelas saat program dieksekusi maka akan dibuat sebuah objek. Objek dilihat dari segi teknis adalah elemen pada saat *runtime* yang akan diciptakan, dimanipulasi, dan dihancurkan saat eksekusi sehingga sebuah objek hanya ada saat sebuah program dieksekusi.

2.3 Metode (*Method*)

Metode pada sebuah kelas hampir sama dengan fungsi atau prosedur pada metodologi struktural. Sebuah kelas boleh memiliki lebih dari satu metode. Metode berfungsi untuk memanipulasi objek itu sendiri. Metode merupakan fungsi atau transformasi uang dapat dilakukan terhadap objek atau yang dilakukan objek. Metode dapat berasal dari *event*, aktivitas atau keadaan, fungsi, atau tingkah laku. Contohnya seperti *read, write, move, copy*, dan lain-lain.

2.4 Atribut

Atribut adalah variabel yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut dipunyai secara individual oleh sebuah objek, misalnya seperti berat, jenis, nama, dan lain-lain.

2.5 Antarmuka (*Interface*)

Interface sangat mirip dengan kelas, tetapi tanpa atribut kelas dan memiliki metode yang dideklarasikan tanpa isi. Deklarasi metode dari sebuah *interface* dapat diimplementasikan oleh kelas lain. Sebuah kelas dapat mengimplementasikan lebih dari satu *interface*.

2.9 UML

UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi.

UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson [11].

2.9.1 *Use Case* Diagram

Use case diagram merupakan pemodelan untuk kelakuan (*behavior*) sistem informasi yang akan dibuat. Diagram ini juga mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem yang akan dibuat. *Use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa yang berhak menggunakan fungsi-fungsi tersebut. Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami [11]. Ada dua hal utama pada *use case* yaitu:

a. Aktor

Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri. Simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.

b. Use case

Use case merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.

2.9.2 *Class* Diagram

Class diagram atau kelas diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode. Diagram kelas dibuat agar pembuat program atau *programmer* membuat kelas-kelas sesuai rancangan di dalam diagram kelas agar antara dokumentasi perancangan dan perangkat lunak dapat sinkron [11].

2.9.3 *Activity* Diagram

Activity diagram merupakan diagram menggambarkan *workflow* (alur kerja) atau aktivitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak.

2.9.4 *Sequence* Diagram

Sequence diagram merupakan diagram yang menggambarkan kelakuan perilaku objek pada *use case* dengan mendeskripsikan waktu hidup objek dan pesan (*message*) yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambar diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang di instansiasi menjadi objek [11].

2.10 C#

C# adalah bahasa pemrograman komputer, sama seperti C, C++, Java, maupun lainnya. Perbedaannya, *C#* menggunakan *library* kelas yang terdapat pada .NET *framework*. Hal ini tentu berbeda dengan C, C++, maupun Java , yang masing-masing memiliki *library* kelas sendiri-sendiri. Kelebihan *library* kelas yang terdapat di dalam .NET *framework* adalah dapat digunakan oleh bahasa-bahasa lain yang mendukung .NET, seperti Visual Basic (VB) dan Visual C++ (VC). Karena C# menggunakan *library* kelas yang terdapat pada .NET *framework*, maka kita perlu mempelajari dan melakukan eksplorasi terhadap kelas-kelas tersebut [12].