BAB II
TINJAUAN PUSTAKA

II.1. Landasan Teori

Landasan teori merupakan penjelasan berbagai konsep dasar dan teori-teori yang berkaitan dalam pembangunan *Implementasi Teknologi LBS (Location Based Service) Dalam Membangun Aplikasi Pencarian Rute Terdekat Menggunakan Algoritma Dijkstra*. Beberapa teori yang terkait dengan pembangunan aplikasi ini adalah perangkat lunak, Teknologi *Location Based Service* (LBS), Algoritma Dijkstra, *Twitter Application Programming Interface,* dan UML.

II.2. Algoritma

Algoritma adalah urutan logis langkah-langkah penyelesaian masalah yang disusun secara sistematis [7]. Algoritma dapat diimplementasikan dalam pembuatan program komputer. Kesalahan dalam merancang algoritma untuk menyelesaikan suatu problema dapat menyebabkan program gagal dalam implementasinya. Konsep dari suatu algoritma sering di ilustrasikan dengan mengambil contoh sebuah resep, walaupun banyak algoritma yang jauh lebih kompleks. Algoritma sering memiliki beberapa langkah perulangan (iterasi) atau memerlukan pengambilan keputusan seperti logika *(logic)* atau perbandingan *(comparison)* sampai pekerjaan diselesaikan. Menerapkan suatu algoritma secara benar belum tentu dapat menyelesaikan problema. Hal ini dikarenakan adanya kemungkinan algoritma tersebut rusak atau cacat, atau penerapannya tidak cocok (tidak tepat) untuk menyelesaikan problema. Sebagai contoh, sebuah algoritma hipotesis untuk membuat sebuah salad kentang akan gagal jika tidak terdapat kentang. Suatu pekerjaan dapat diselesaikan dengan menggunkan algoritma yang berbeda dengan kumpulan instruksi *(set of instructions)* yang berbeda dengan perbedaan waktu akses, efisiensi tempat, usaha dan sebagainya. Sebagai contoh, dierikan dua buah resep yang berbeda untuk membuat salad kentang, resep pertama mengupas kulit kentas terlebih dahulu sebelum memasak kentang tersebut, sementara resep akan mengulangi kedua langkah tersebut dan akan dihentikan pada

saat salad kentang siap untuk dimakan. Algoritma adalah hal yang mendasar untuk komputer dalam memproses informasi, karena sebuah program komputer adalah sebuah algoritma yang memberitahukan kepada komputer langkah-langkah spesifik yang akan dijalankan (dalam urutan spesifik) untuk melakukan pekerjaan tertentu, misalnya menghitung gaji karyawan untuk mencetak rapor murid. Oleh karena itu, algoritma dapat dianggap sebagai beberapa operasi sekuensial (terurut) yang dapat dijalankan oleh sebuah system lengkap Turing.

II.3. *Shortest Path*

Persoalan mencari lintasan terpendek di dalam graf merupakan salah satu persoalan optimasi. Graf yang digunakan dalam pencarian lintasan terpendek adalah graf berbobot (*weight graf*), yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot. Bobot pada sisi graf dapat menyatakan jarak antar kota, waktu pengiriman pesan, ongkos pembangunan, dan sebagainya. [7]

Seorang pengendara sepeda motor ingin menemukan rute terpendek yang dapat dilalui dari Chicago ke Boston. Diberikan sebuah peta jalan dari amerika Serikat dimana setiap daerah yang dapat dilalui dengan jalur darat dengan jarak tertentu ditandai / dihubungkan, bagaimana cara menentukan rute terpendek ini? Salah satu cara yang dapat dilakukan adalah dengan menjabarkan semua rute yang mungkin dari Chicago ke boston, menghitung jarak dari setiap rute tersebut dan mengambil rute dengan jarak terpendek dari kumpulan rute terpendek tersebut. Cara ini sangat tidak efisien dan akan memakan waktu yang sangat lama dalam mencari solusi rute terpendek tersebut karena bias terdapat banyak sekali rute yang dapat dilalui dari Chicago ke boston. Cara lainnya yang lebih efisien adalah dengan menerapkan algoritma *shortest path* pada graf. Dalam sebuah problema *shortest path*, diberikan sebuah graf berbobot dan berarah G = (V, E), dengan fungsi bobot w : E → R memetakan sisi (*edge*) dengan nilai bobot dari *path* p = {v0, v1,…, vq} adalah penjumlahan bobot dari setiap sisinya seperti ditunjukkan oleh rumusan berikut:

$$w(p) = \sum_{i-1}^{k} vi - 1, v1$$

Kemudian, didefenisikan berbobot *shortest path* dari u ke v dengan rumusan berikut:

$$\&(u,v) = \begin{cases} \min\{w(p): u \to v : jika\ terdapat\ path\ u\ ke\ v \\ \infty \qquad\qquad\qquad\quad : jika\ tidak\ terdapat\ path \end{cases}$$

Shotest path dari *node* u ke *node* vdapat didefenisikan sebagai sembarang path p denganbobot w(p) = &(u, v).

Dalam contoh pencarian rute dari Chicago ke boston, peta jalan dapat dimodelkan sebagai sebuah graf dimana *node* memiliki daerah sisi mewakili jalan antar daerah yang dapat dilalui dan bobot sisi mewakili jarak jalan. Sasarannya adalah menemukan *shortest path* dari Chicago ke boston. [8]

Ada beberapa macam persoalan lintasan terpendek, antara lain: [7]

a. Lintasan terpendek antara dua buah *node* tertentu.
b. Lintasan terpendek antara semua pasangan *node*.
c. Lintasan terpendek dari *node* tertentu ke semua *node* yang lain.
d. Lintasan terpendek antara dua buah *node* yang melalui beberapa *node* tertentu.

Algoritma *shortest path* ini dapat diterapkan untuk mencari solusi dari beberapa variasi dari *shortest path* seperti:

a. *Single-destination shortest path*

Problem ini ditunjukkan untuk mencari *shortest path* dari sebuah *node* tujuan t yang telah ditentukan dari setiap *node* v. problema ini dapat diselesaikan dengan membalikkanarah dari setiap sisi pada graf, sehingga penyelesaian problema ini sama persis dengan problema single-source shortest path.

b. *Single-pairs shortest path*

Problema ini ditunjukkan untuk mencari *shortest path* dari u ke v untuk *node* u dan v yang ditentukan. Problema ini disebut juga degan problema *singlesource shortest path.*

c. *All-pairs shortest path*

Problem ini ditunjukkan untuk mencari shortest path dari u ke v untuk setiap pasangan *node* u dan v. problema ini dapat diselesaikan dengan menerapkan

algoritma *single-source shortest path* sekali dari setiap *node*, tetapi ada algoritma spesifik lain yang dapat menyelesaikan problema ini denganlebih cepat. [8]

II.3.1. *Single Source Shortest path*

Problema *single-source shortest path* ini dapat dideskripsikan sebagai berikut: Diberikan pasangan *node* u dan v dimana *node* sebagai *node* awal dan *node* v sebagai *node* tujuan. Sasarannya adalah mencari *shortest path* dari *node* u ke *node* v. solusi yang ingin dicari mencakup bobot dari *shortest path* dan sisi-sisi yang terdapat dalam jalur terpendek *(shortest path)* tersebut. Dalam menyelesaikan problema *shortest path* ini, sering juga ditemui bahwa bobot dari sisi graf bernilai *negatif*. Bobot negatif ini sring ditemukan pada penerapan algoritma shortest path untuk mencaari solusi biaya minimum pada ilmu eonomi. Jika graf G = (V,E) tidak memiliki siklus berbobot negatf yang dapat dilalui dari *node* awal s, maka untuk semua v € V, bobot shortest path δ(s, v) dapat ditentukan, sekalipun graf tersebut memiliki bobot *negatif*. Jika terdapat siklus berbobot negatif yang dapa dilalui dari *node* awal s, maka *shortest path*-nya tidak dapat ditentukan. Jika terdapat siklus berbobot *negatif* pada beberapa jalur (path) dari s ke v, maka *shortest* path δ(s, v) = -∞. [8]

Beberapa algoritma yang dapat diterapkan untuk mencari solusi dari problema *shortest* path adalah:

a. Algoritma Dijkstra
b. Algorima Bellman-Ford
c. Algoritma Johnson
d. Algoritma Floydwarshall

II.3.2.Algoritma Dijkstra

Algoritma ini menerapkan konsep algoritma greedy dalam mencari solusi dari problema *shortest path*. Prinsip *greedy* pada algoritma Dijkstra menyatakan bahwa pada setiap langkah kita memilih sisi yang berbobot minimum dan memasukkannya ke dalam himpunan solusi [7]. Algoritma Dijkstra menyelesaikan problema *single source shortest path* pada sebuah graf berbobot dan beaeah G = (V, E) untuk kasus

dimana semua bobot sisi adalah positif aaupun mengambil asumsi bahwa w(u, v) ≥ 0 untuk semua sisi (u, v) € E. Algoritma Dijkstra ini memiliki kompleksitas waktu sebesar $O(V^2)$.

Misalkan sebuah graf berbobot dengan *n* buah *node* dinyatakan dengan matriks ketetanggan $M = [m_{ij}]$, yang dalam hal ini,

m_{ij} = bobot sisi (*i*,*j*) (pada graf tak-berarah $m_{ij} = m_{ij}$)

$m_{ij} = 0$

$m_{ij} = \infty$, jika tidak ada sisi dari *node i* ke *node j*

Selain matriks *M*, kita juga menggunakan tabel $S = [s_i]$ yang dalam hal ini,

s_i = 1, jika *node i* termasuk ke dalam lintasan terpendek

$s_i = 0$, jika *node i* tidak termasuk ke dalam lintasan terpendek

dan tabel $D = [d_i]$ yang dalam hal ini,

$[d_i]$ = panjang lintasan dari *node* awal *a* ke *node i*

Algoritma Dijkstra dinyatakan dalam notasi *pseudo-code* sebagai berikut: [7]

```
procedure Dijkstra (input m: matriks, a : node awal)
{ Mencari lintasan terpendek dari node awal a ke semua node
lainnya
Masukan: matriks ketetanggan (m) dari graf berbobot G dan node
awal a
Keluaran: lintasan terpendek dari a ke semua node lainnya
}
Deklarasi
s1, s2, ..., sn : integer {tabel integer}
d1, d2, ..., dn : integer {tabel integer}
i, j, k : integer
Algoritma
{Langkah 0 (inisialisasi)}
 for i ← 1 to n do
          si ← 0
    di ← mai
endfor
      {Langkah 1:}
```

```
        s_a ← 1    (karena node a adalah node asal lintasan
                   terpendek, jadi node a  sudah pasti terpilih
                   dalam lintasan terpendek)
        d_a ← ∞    (tidak ada lintasan terpendek dari node a ke
                   a)
     {Langkah 2, 3, ..., n − 1 :}
      for k ← 2 to n -1 do
          j  ← node dengan s_j = 0 dan d_j minimal
         s_j ← 1  {node j sudah terpilih ke dalam lintasan
terpemdek}
             { perbarui tabel d}
             for semua node i dengan s_j = 0 do
                   if d_j + m_ji < then
                        d_i ← d_j + m_ji
                   endif
              endfor
         endfor
```

II.4. Teori Graf

II.4.1.Definisi Graf

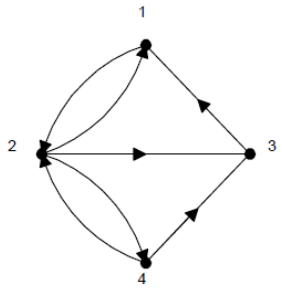
Graf adalah kumpulan *node* (*nodes*) yang dihubungkan satu sama lain melalui sisi/busur (*edges*). Secara matematis graf didefinisikan sebagai berikut: Graf *G* didefinisikan sebagai himpunan (*V, E*), ditulis dengan notasi *G =(V, E),* yang dalam hal ini *V* adalah himpunan tidak kosong dari *node-node* (*vertices* atau *node*) dan *E* adalah himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang *node*. [7]

Dimana dalam definisi dia atas menyatakan bahwa ***V* tidak boleh kosong** sedangkan ***E* boleh kosong**. Dengan demikian dimungkinkan tidak mempunyai sisi satu buah pun, tetapi *node*nya harus ada, minimal satu. Graf yang hanya mempunyai satu buah *node* tanpa sebuah sisi pun dinamakan **graf *trival*** [7].

Node pada graf dapat dinomori dengan huruf, seperti *a, b, c, ..., v, w, ...*, dengan bilangan asli 1, 2, 3, atau gabungan keduanya. Sedangkan sisi yang menghubungkan *node u* dengan *node v* dinyatakan dengan pasangan (*u,v*) atau dinyatakan dengan lambang *e1, e2, ...* Dengan kata lain, jika *e* adalah sisi yang menghubungkan *node u* dengan *node v,* maka *e* dapat ditulis sebagai [7]

$$\boldsymbol{e = (u, v)}$$

Secara geometri graf digambarkan sebagai sekumpulan noktah (*node*) didalam bidang dwimatra yang dihubungkan dengan sekumpulan garis (sisi). [7]



Gambar II. 1 Contoh Graf

II.4.2.Jenis-jenis Graf

Graf dapat dikelompokkan menjadi beberapa kategori bergantung pada sudut pandang pengelompokannya. Pengelompokan graf dapat dipandang berdasarkan ada tidaknya sisi ganda atau sisi kalang, berdasarkan jumlah *node* atau berdasarkan jumlah *node* atau berdasarkan orientasi arah pada sisi. Berdasarkan ada tidaknya gelang atau sisi ganda pada suatu graf, maka secara umu graf dapat digolongkan menjadi dua jenis : [7]

1. Graf sederhana *(simple graph)*

Graf yang tidak mengandung gelang maupun sisi ganda dinamakan graf sederhana. *Node* menyatakan siisi menyatakan saluran telepon untuk berkomunikasi. Saluran telepon dapat beoperasi pada dua arah. Pada graf sederhana, sisi adalah pasangan tak terurut (unordered pairs). Jadi, menuliskan sisi (u,v) sama saja dengan (v,u). atau dapat juga mendefenisikan graf sederhana G = (V, E) terdiri dari himpunan tidak kosong *node-node* dan E adalah himpuan pasangan tak terurut yang berbeda disebut sisi.

2. Graf tak-sederhana *(Unsimple-graph)*

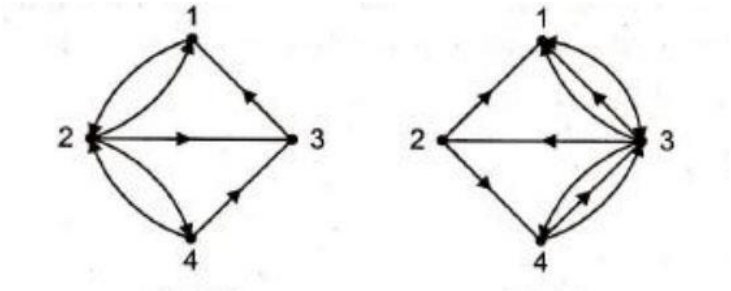
Graf yang mengandung sisi ganda atau gelang dinamakan graf tak sederhana (*unsimple graph*) . Ada dua macam graf tak sederhana, yaitu graf ganda (*multi graph*) dan graf semu (*pseudo graph*). Graf ganda adalah graf yang mengandung sisi ganda. Sisi ganda yang menghubungkan sepasang *node* bias lebih dari dua buah. Sisi ganda dapat diasosiasikan sebagai

pasangan tak terurut yang sama. Atau dapat juga didefenisikan graf gandan G = {V,E} terdiri dari himpunan tidak kosong *node-node* dan E adalah himpunan ganda (*multiset*) yang mengandung sisi ganda. Setiap graf sederhana juga adalah graf ganda, tetapi tidak setiap graf ganda merupakan graf sedrhana.

Graf semu adalah graf yang mengandung gelang *(loop)*. Graf semu lebih umum daripada graf ganda, karena sisi pada graf semu dapat terhubung ke dirinya sendiri. Jumlah *node* pada graf dapat disebut sebagai kardinalitas graf dinyatakan dengan n = |V| dan jumlah sisi dinyatakan dengan m = |E|. Berdasarkan orientasi arah pada sisi graf, maka secara umum graf dapat dibedakan atas 2 jenis: [7]

1. Graf tak-berarah *(undirected graph)*
2. Graf yang sisinya tidak mempunyai arah disebut graf tak berarah. Pada graf tak berarah, urutan pasangan *node* yang dihubungkan oleh sisi tidak diperhatikan. Jadi, (u, v) adalah sisi yang sama.
3. Graf berarah (*directed graf*)

   Graf yang setiap sisinya diberikan orientasi arah disebut sebagai graf berarah. Sisi berarah dapat juga disebut sebagai busur (arc). Pada graf berarah (u,v) dan (v,u) menyatakan dua buah busur yang berbeda, dengan katalain (u,v) ≠ (v,u). Untuk busur (u,v), *node* u dinamakan *node* asal (*initial vertex*) dan *node* v dinamakan *node* terminal *(terminal vertex)*. [7]



Gambar II. 2 (a) Graf Berarah (b) Graf Ganda Berarah

(a) G1 (b)G2

Gambar II. 2 (a) adalah contoh graf berarah. Pada graf G1 dapat dibayangkan sebagasi saluran telepon yang tidak dapat beroperasi dua arah. Saluran hanya beroperasi pada arah yang ditunjjukkan oleh anak panah. Jadi, sebagai contoh saluran telepon (1,2) tidak sama dengan salurantelepon (2,1), graf berarah sering

dipakai untuk menggambarkan aliran proses, peta lalulintas suatu kota (jalan searah atau dua arah) dan sebagainya. Pada graf berarah, gelang diperbolehkan, tatapi sisi ganda tidak. Defenisi graf dapat diperluas sehingga mencakup graf ganda berarah (directed multigraf). Pada graf ganda berarah, gelang dan sisi ganda diperbolehkan ada. Contoh graf ganda berarah dapat dilihat pada Gambar II. 2.

II.5. Ketentuan Kriteria Penetapan Klasifikasi Fungsi Jalan

Ciri-ciri umum fungsi jalan merupakan arahan yang perlu dipenuhi. Ada 3 klasifikasi dari kriteria klasifikasi fungsi jalan antara lain adalah sebagai berikut: [9]

1. Jalan arteri primer

   Jalan arteri primer adalah jalan yang menghubungkan secara terus menerus pusat kegiatan nasional atau antara pusat kegiatan nasional dengan pusat kegiatan wilayah.

   Syarat klasifikasi fungsi jalan lokal primer :

   a. Kriteria jalan lokal primer terdiri atas:
      1) Jalan lokal primer didesain berdasarkan kecepatan rencana paling rendah 60 (dua puluh) km/jam;
      2) Lebar badan jalan lokal primer paling rendah 11 meter;
      3) Besarnya lalu lintas harian rata-rata pada umumnya lebih besar dari fungsi jalan yang lain.

2. Jalan kolektor primer

   Jalan kolektor primer adalah jalan yang secara efisien menghubungkan antar pusat kegiatan wilayah atau menghubungkan kegiatan antar pusat kegiatan wilayah dengan pusat kegiatan lokal.

   Syarat klasifikasi fungsi jalan kolektor primer :

   a. Kriteria jalan lokal primer terdiri atas:
      1) Jalan lokal primer didesain berdasarkan kecepatan rencana paling rendah 40 (dua puluh) km/jam;
      2) Lebar badan jalan kolektor primer paling rendah 9 meter;

3) Besarnya lalu lintas harian rata-rata pada umumnya lebih rendah jalan arteri primer.

3. Jalan lokal primer

Jalan lokal primer adalah jalan yang secara efisien menghubungkan pusat kegiatan nasional dengan persil atau pusat kegiatan wilayah dengan persil atau pusat kegiatan lokal dengan pusat kegiatan lokal, pusat kegiatan lokal dengan pusat kegiatan di bawahnya, pusat kegiatan lokal dengan persil, atau pusat kegiatan dibawahnya sampai persil.

Syarat klasifikasi fungsi jalan lokal primer :

a. Kriteria jalan lokal primer terdiri atas:
   1) Jalan lokal primer didesain berdasarkan kecepatan rencana paling rendah 20 (dua puluh) km/jam;
   2) Lebar badan jalan lokal primer paling rendah 6,5 meter;
   3) Besarnya lalu lintas harian rata-rata pada umumnya paling rendah pada sistem primer.

II.6. *Twitter*

Twitter adalah sebuah situs *web* yang dimiliki dan dioperasikan oleh *Twitter Inc.*, yang menawarkan jaringan sosial berupa *mikroblog* sehingga memungkinkan penggunanya untuk mengirim dan membaca pesan *Tweets* (*Twitter*, 2013).

Mikroblog adalah salah satu jenis alat komunikasi *online* dimana pengguna dapat memperbarui status tentang mereka yang sedang memikirkan dan melakukan sesuatu, apa pendapat mereka tentang suatu objek atau fenomena tertentu.

Tweets adalah teks tulisan hingga 140 karakter yang ditampilkan pada halaman *profil* pengguna. *Tweets* bisa dilihat secara publik, namun pengirim dapat membatasi pengiriman pesan ke daftar teman-teman mereka saja. Pengguna dapat melihat *Tweets* pengguna lain yang dikenal dengan sebutan pengikut (*follower*).

Tidak seperti Facebook, LinkedIn, dan MySpace, *Twitter* merupakan sebuah jejaring sosial yang dapat digambarkan sebagai sebuah graf berarah, yang berarti bahwa pengguna dapat mengikuti pengguna lain, namun pengguna kedua tidak

diperlukan untuk mengikutinya kembali. Kebanyakan akun berstatus publik dan dapat diikuti tanpa memerlukan persetujuan pemilik..

Semua pengguna dapat mengirim dan menerima *Tweets* melalui situs *Twitter*, aplikasi *eksternal* yang kompatibel (telepon seluler), atau dengan pesan singkat (SMS) yang tersedia di negara-negara tertentu (*Twitter*, 2013). Pengguna dapat menulis pesan berdasarkan topik dengan menggunakan tanda # (*hashtag*). Sedangkan untuk menyebutkan atau membalas pesan dari pengguna lain bisa menggunakan tanda @.

Pesan pada awalnya diatur hanya mempunyai batasan sampai 140 karakter disesuaikan dengan kompatibilitas dengan pesan SMS, memperkenalkan singkatan notasi dan slang yang biasa digunakan dalam pesan SMS. Batas karakter 140 juga meningkatkan penggunaan layanan memperpendek URL seperti bit.ly, goo.gl, dan tr.im, dan jasa hosting konten, seperti Twitpic, Tweephoto, memozu.com dan NotePub untuk mengakomodasi multimedia isi dan teks yang lebih panjang daripada 140 karakter (*Twitter*, 2013). *Twitter* menggunakan bit.ly untuk memperpendek otomatis semua URL yang dikirim-tampil. Fitur yang terdapat dalam *Twitter*, antara lain:

1. Laman Utama (*Home*)

Pada halaman utama kita bisa melihat *Tweets* yang dikirimkan oleh orang-orang yang menjadi teman kita atau yang kita ikuti (*following*).

2. Profil (*Profile)*

Pada halaman ini yang akan dilihat oleh seluruh orang mengenai profil atau data diri serta *Tweets* yang sudah pernah kita buat.

3. *Followers*

Pengikut adalah pengguna lain yang ingin menjadikan kita sebagai teman. Bila pengguna lain menjadi pengikut akun seseorang, maka *Tweets* seseorang yang ia ikuti tersebut akan masuk ke dalam halaman utama.

4. *Following*

Kebalikan dari pengikut, *following* adalah akun seseorang yang mengikuti akun pengguna lain agar *Tweets* yang dikirim oleh orang yang diikuti tersebut masuk ke dalam halaman utama.

5. *Mentions*

Biasanya konten ini merupakan balasan dari percakapan agar sesama pengguna bisa langsung menandai orang yang akan diajak bicara.

6. *Favorite*

Tweets ditandai sebagai favorit agar tidak hilang oleh halaman sebelumnya.

7. Pesan Langsung *(Direct Message)*

Fungsi pesan langsung lebih bisa disebut SMS karena pengiriman pesan langsung di antara pengguna.

8. *Hashtag*

Hashtag "#" yang ditulis di depan topik tertentu agar pengguna lain bisa mencari topik yang sejenis yang ditulis oleh orang lain juga.

9. *List*

Pengguna *Twitter* dapat mengelompokkan ikutan mereka ke dalam satu grup sehingga memudahkan untuk dapat melihat secara keseluruhan para nama pengguna *(username)* yang mereka ikuti *(follow).*

10. Topik Terkini *(Trending Topic)*

Topik yang sedang banyak dibicarakan banyak pengguna dalam suatu waktu yang bersamaan.

II.7. *Twitter* API

Twitter telah menjadi *media social network* yang sangat populer dan efektif yang sudah hampir menyamai popularitas facebook. *Twitter* menyediakan API *(Application Programming Interface)* yang sangat baik, sehingga memudahkan setiap orang untuk mengambil data dari *twitter*. Pengumpulan data dari *twitter* dapat digunakan untuk berbagai kebutuhan seperti mengetahui popularitas kandidat pilkada atau pemilu, mendapat informasi mengenai popularitas suatu produk atau untuk yang sederhana dapat digunakan untuk melihat semua *mention*, *retwit* atas suatu account *twitter* tertentu.

API *(Application Programming Interface)* adalah sekumpulan perintah, fungsi, komponen, dan protokol yang disediakan oleh sistem operasi ataupun

bahasa pemrograman tertentu yang dapat digunakan oleh programmer saat membangun perangkat lunak.

Dalam API terdapat fungsi-fungsi atau perintah-perintah untuk menggantikan bahasa yang digunakan dalamsystem calls dengan bahasa yang lebih terstruktur dan mudah dimengerti oleh programmer.

Keuntungan menggunakan API:

1. Probabilitas

   API dapat digunakan untuk bahasa pemrograman ataupun untuk sistem operasi mana saja asalkan paket-paket API sudah terpasang.

2. Lebih Mudah Dimengerti

   API menggunakan bahasa yang lebih terstruktur dan mudah dimengerti daripada bahasa system call. Hal ini sangat penting dalam hal editing dan pengembangan.

3. Mudah Dikembangkan

   Dengan adanya API, memudahkan programmer untuk mengembangkan suatu sistem.

II.8. *Text Mining*

Text mining merupakan variasi dari data mining yang berusaha menemukan pola yang menarik dari sekumpulan data tekstual yang berjumlah besar. Langkah-langkah yang dilakukan text mining adalah sebagai berikut :

1. *Text processing*

   Tindakan yang dilakukan pada tahap ini adalah *toLowerCase*, yaitu mengubah semua karakter huruf menjadi huruf kecil, dan *Tokenizing* yaitu proses penguraian deskripsi yang semula berupa kalimat- kalimat menjadi kata-kata dan menghilangkan delimeter-delimeter seperti tanda titik(.), koma(,), spasi, dan karakter angka yang ada pada kata tersebut.

2. *Feature Selection*

   Pada tahap ini tindakan yang dilakukan adalah menghilangkan *stopword* (*stop removal*) dan *stemming* terhadap kata yang berimbuhan.

a. *Stopword* adalah kosakata yang bukan merupakan ciri (kata unik) dari suatu dokumen. Misalnya “di”, “oleh”, “pada”, “sebuah”, “karena” dan lain sebagainya.

b. *Stemming* adalah proses pemetaan dan penguraian berbagai bentuk *(variants)* dari suatu kata menjadi bentuk kata dasarnya (stem). Tujuan dari proses *stemming* adalah menghilangkan imbuhan-imbuhan baik itu berupa prefiks, sufiks, maupun konfiks yang ada pada setiap kata.

II.9. Text *Preprocessing*

Pada *text mining* data yang di gunakan berasal dari dokumen atau teks yang tidak terstruktur. Oleh karena itu, dibutuhkan suatu proses yang dapat mengubah bentuk data yang sebelumnya tidak terstruktur menjadi data yang terstruktur. Proses ini bertujuan agar data yang akan digunakan nantinya bersih dari noise atau ciri-ciri yang tidak berpengaruh pada klasifikasi filtering seperti link, “@”, “RT”, *stopword*. Proses preprosesing juga mempunyai tujuan agar data yang digunakan memiliki dimensi yang lebih kecil dan lebih terstruktur, sehingga dapat diolah lebih lanjut. Tahap *preprocessing* yang di gunakan dapat dilihat pada Gambar II. 3.

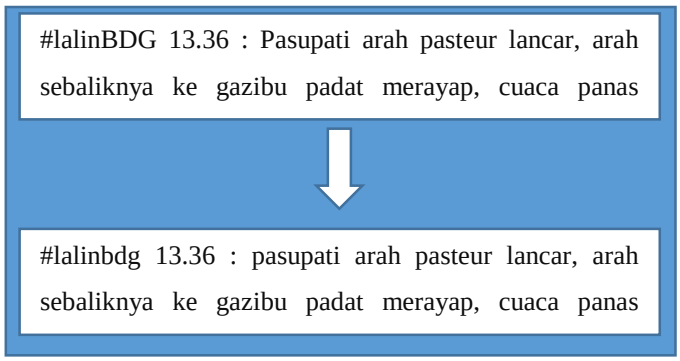


Gambar II. 3 Proses Text *Preprocessing*

Text Preprocessing pada umumnya terdiri dari tahapan Tokenisasi, *Case folding*, *Stopwords Removing*. Pada penelitian ini, tahapan *Preprocessing* adalah *case folding, cleansing, stopword removal, tokenizer* dan *stemming*. Keseluruhan tahapan memiliki fungsi dan perannya masing-masing. Untuk mendapatkan dataset yang berdimensi lebih kecil dari data sebelumnya, terstruktur, serta bersih dari noise, maka kesemua tahap harus berkesinambungan.

II.9.1.*Case folding*

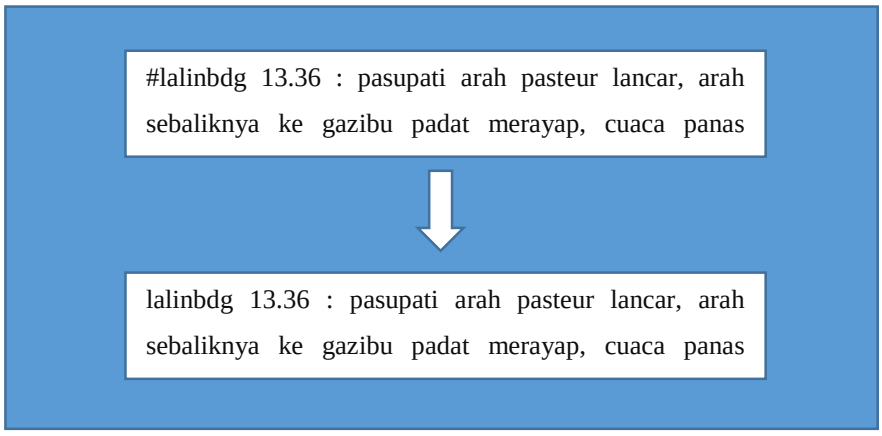
Case folding merupakan proses *text preprocessing* yang dilakukan untuk menyeragamkan karakter pada data (dokumentasi/teks). Pada proses ini, semua huruf besar (*uppercase*) dijadikan huruf kecil (*lowercase*). Bila digambarkan, proses *case folding* dapat dilihat pada Gambar II. 4.



Gambar II. 4 Gambaran proses *case folding*

II.9.2.*Cleansing*

Ada terdapat beberapa komponen yang khas pada data yang diambil dari *tweet* yaitu, *username*, *Uniform Resource Locator* (URL), "RT" (tanda *retweet*), dan *hashtag*. Karena *username*, URL, dan "RT" tidak memiliki pengaruh apapun terhadap nilai *filtering*, maka ketiga komponen di atas akan dibuang [10]. Gambaran dari proses *cleansing* dapat dilihat pada Gambar II. 5.

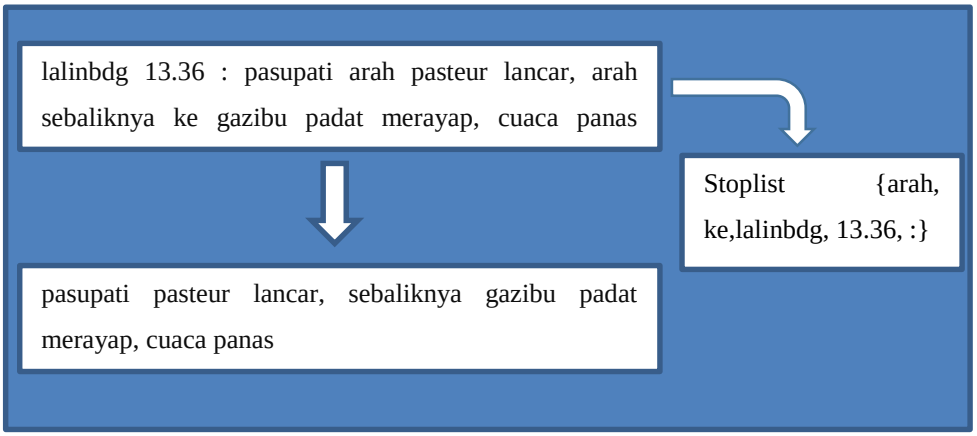


Gambar II. 5 Gambaran proses *Cleansing*

II.9.3.*Stopword* Removal

Stopword Removal merupakan tahap selanjutnya pada proses *text preprocessing*. Tahapan ini bertujuan untuk menghilangkan kata atau *term* yang dianggap tidak dapat memberikan pengaruh dalam menentukan suatu kategori tertentu dalam suatu dokumen. Proses ini dilakukan karena *term* tersebut sering muncul hampir disetiap dokumen sehingga dianggap tidak dapat menjadi pembeda yang baik dalam membedakan kategori yang satu dengan kategori yang lain. [11]

Sebelum dilakukan proses *Stopword Removal*, terlebih dahulu dibuat kata-kata yang termasuk ke dalam *stopwords* atau *stoplist*. Kata-kata yang termasuk *stoplist* tersebut biasanya berupa kata ganti orang, kata penghubung, pronominal penunjuk, dan lain sebagainya. Gambaran dari proses hapus *stopwords* dapat dilihat pada Gambar II. 6.

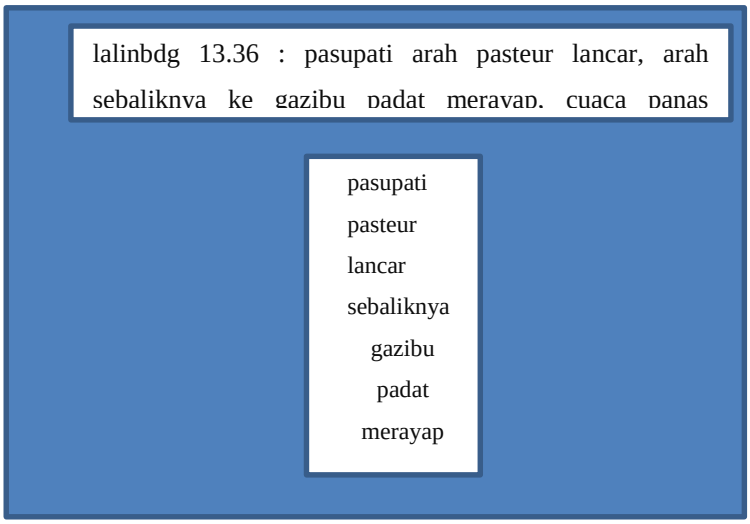


Gambar II. 6 Gambaran proses *stopword removal*

Sebuah file akan didefinisikan sebagai sebuah *string*, kemudian sistem akan mengambil satu persatu *term* yang terdapat pada *stoplist*. Jika *string* terdapat *substring stoplist,* maka *substring* tersebut akan diganti dengan karakter *blank*. Proses *stopword removal* ini, besarnya ukuran atau dimensi data yang tereduksi bergantung pada banyaknya *stopwords* yang digunakan sebagai *stoplist* dan banyaknya *term* yang mengandung *stopwords.*

II.9.4.*Tokenizer*

Tokenizer merupakan proses memisahkan kata atau *word* [10]. Pada penelitian ini *tokenizer* yang dilakukan adalah memisahkan setiap kata yang dipisahkan oleh spasi. Bila digambarkan, proses tokenisasi dapat dilihat pada Gambar II. 7.



Gambar II. 7 Gambaran *Tokenizer*

II.9.5.*Stemming*

Stemming digunakan untuk mencari kata dasar dari bentuk berimbuhan. Algoritma *steamming* untuk bahasa yang satu berbeda dengan algoritma *stemming* untuk bahasa lainnya. Bahasa Inggris memiliki morfologi yang berbeda dengan Bahasa Indonesia sehingga algoritma *steamming* yang digunakan pun berbeda [12]. Proses *stemming* pada teks berbahasa Indoensia lebih rumit karena terdapat variasi imbuhan yang harus dibuang untuk mendapatkan *root word* dari sebuah kata. Algoritma *stemming* yang digunakan pada penelitian ini adalah algoritma Nazief dan Andriani

II.10.*Machine Learning*

Machine Learning merupakan cabang ilmu *artificial intelligence* yang memiliki konsep bahwa komputer sebagai mesin memiliki kemampuan untuk melakukan adaptasi terhadap lingkungan yang baru dan mampu mendeteksi pola dari fakta yang ada. Definisi pembelajaran yang dilakukan mesin adalah ketika

mesin dari pengalaman E terhadap tugas T dan mengukur peningkatan kinerja P, jika kinerja tugas T diukur oleh kinerja P, meningkatkan pengalaman E [13].

II.11. *Naïve bayes Clas*sifier

Naïve bayes merupakan salah satu metode *machine learning* yang menggunakan perhitungan probabilitas. Konsep dasar yang digunakan oleh *Naïve bayes* adalah Teorema Bayes, yaitu melakukan klasifikasi dengan melakukan perhitungan nilai probabilitas $p(C = c_i \mid D = d_j)$, yaitu probabilitas kategori c_i jika diketahui dokumen d_j. Klasifikasi dilakukan untuk mementukan kategori $c \in C$ dari suatu dokumen $d \in D$ dimana $C = \{c_1, c_2, c_3, \ldots, c_i\}$ dan $D = \{d_1, d_2, d_3, \ldots, d_j\}$. Penentuan dari kategori sebuah dokumen dilakukan dengan mencari nilai maksimum dari $p(C = c_i \mid D = d_j)$ pada P=$\{ p(C = c_i \mid D = d_j) \mid c = C$ dan $d = D\}$. Nilai probabilitas $p(C = c_i \mid D = d_j)$ dapat dihitung dengan persamaan:

$$p(C = c_i|D = d_j) = \frac{P\left(C = c_i \cap D = d_j\right)}{P\left(D = d_j\right)} = \frac{P\left(D = d_j|C = c_i\right) \times p\left(C = c_i\right)}{P\left(D = d_j\right)} \quad \textbf{(II-1)}$$

Dengan $p(D = d_j \mid C = c_i)$ merupakan nilai probabilitas dari kemunculan dokumen d_j jika diketahui dokemen tersebut berkategori c_i, $p(C = c_i)$ adalah nilai probabilitas kemunculan kategori c_i, dan $p(D = d_j)$ adalah nilai probabilitas kemunculan dokumen d_j.

Naïve bayes menganggap sebuah dokumen sebagai kumpulan dari kata-kata yang menyusun dokumen tersebut, dan tidak memperhatikan urutan kemunculan kata pada dokumen. Sehingga perhitungan probabilitas $p(D = d_j \mid C = c_i)$ dapat dianggap sebagai hasil perkalian dari probabilitas kemunculan kata-kata pada dokumen d_j. Perhitungan probabilitas $p(C = c_i \mid D = d_j)$ dapat dituliskan sebagai berikut:

$$p(C = c_i|D = d_j) = \frac{\prod_k p\left(w_k \mid C = c_i\right) \times p\left(C = c_i\right)}{p(w_1, w_2, w_3, \ldots, w_n)} \quad \textbf{(II-2)}$$

Dengan $\prod_k p\left(w_k \mid C = c_i\right)$ adalah hasil perkalian dari probabilitas kemunculan semua kata pada dokumen dj.

Proses klasifikasi dilakukan dengan membuat model probabilistik dari dokumen *training*, yaitu dengan menghitung nilai $p(w_k \mid c)$. Untuk w_{kj} diskrit dengan $w_{kj} \varepsilon V = \{v_1, v_2, v_3, \ldots, v_m\}$ maka $p(w_k \mid c)$ dicari untuk seluruh kemungkinan nilai w_{kj} dan didapatkan dengan melakukan perhitungan:

$$p\,(c) = \frac{D_b\,(c)}{|D|} \qquad \textbf{(II-3)}$$

Dimana :

$D_b\,(c)$ adalah jumlah dokumen yang memiliki kategori c_i.

$|D|$ adalah jumlah seluruh *training* dokumen.

Dan

$$p\big(w_k = w_{kj} \big| c\,\big) = \frac{D_b\,(w_k = w_{kj}\,.c)}{D_b(c)} \qquad \textbf{(II-4)}$$

Dimana :

$D_b\,\big(\,w_k = w_{kj},\ c\big)$ adalah nilai kemunculan kata w_{kj} pada kategori c_i.

$D_b\,(c)$ adalah jumlah keseluruhn kata pada kategori c_i.

Persamaan $Db(w_k = w_{kj}\,,c)$ sering kali dikombinasikan dengan *Laplacian Smoothing* untuk mencegah persamaan mendapatkan nilai 0, yang dapat menggangu hasil klasifikasi secara keseluruhan. Sehingga persamaan $Db(w_k = w_{kj}\,,c)$ dituliskan sebagai:

$$p\,\big(w_k = w_{kj} \big|\, c\,\big) = \frac{D_b\,\big(\,w_k = w_{kj},\ c\big) + 1}{D_b\,(c) + |V|} \qquad \textbf{(II-5)}$$

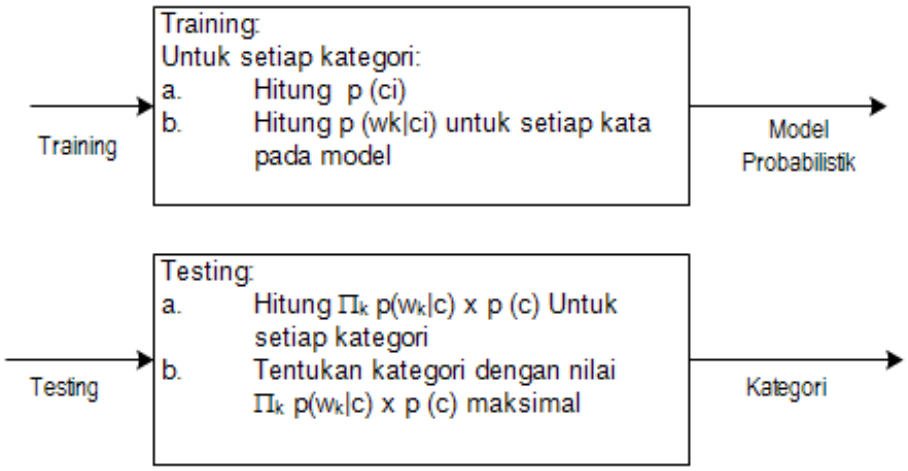
dengan $|V|$ merupakan jumlah kemungkinan nilai dari w_{kj}.

Pemberian kategori dari sebuah dokumen dilakukan dengan memilih nilai c yang memiliki nilai $p(C = c_i \mid D = d_j)$ maksimum, dan dinyatakan dengan:

$$c^* = arg\,\max_{c\,\in C} \prod_k p\,(\,w_k|c\,)\ \times p\,(c) \qquad \textbf{(II-6)}$$

Kategori c^* merupakan kategori yang memiliki nilai $p(C = c_i \mid D = d_j)$ maksimum. Nilai $p(D = d_j)$ tidak mempengaruhi perbandingan karena untuk setiap

k ategori nilainya akan sama. Berikut ini gambaran proses klasifikasi dengan algoritma *Naïve Bayes*.



Gambar II. 8 Proses Klasifikasi Algoritma *Naïve Bayes*

II.12. K-Fold *Cross Validation*

K-fold cross validation merupakan salah satu metode yang digunakan untuk mengetahui rata-rata keberhasilan dari suatu sistem dengan cara melakukan perulangan dengan mengacak atribut masukan sehingga sistem tersebut teruji untuk beberapa atribut *input* yang acak. *K-fold cross validation* diawali dengan membagi *data* sejumlah *n-fold* yang diinginkan. Dalam proses *cross validation data* akan dibagi dalam *n* buah partisi dengan ukuran yang sama D_1, D_2, D_3 .. D_n selanjutnya proses uji dan latih dilakukan sebanyak n kali. Dalam iterasi ke-I partisi D_i akan menjadi data uji dan sisanya akan menjadi data latih. Untuk penggunaan jumlah *fold* terbaik untuk uji validitas, dianjurkan menggunakan 10-*fold cross validation* dalam model [14].

Untuk setiap percobaan K, menggunakan K-1 lipatan untuk pelatihan dan sisanya satu untuk pengujian, keuntungan dari *K-fold cross validation* adalah bahwa semua contoh di dataset yang akhirnya digunakan untuk kedua pelatihan dan pengujian dataset yang akhirnya digunakan untuk kedua pelatihan dan pengujian. Cara kerja *K-fold cross validation* adalah sebagai berikut:

1. Total *instance* dibagi menjadi N bagian.
2. *Fold* ke-1 adalah ketika bagian ke-1 menjadi data uji (*testing data*) dan sisanya menjadi data latih (*training data*). Selanjutnya, hitung akurasi

berdasarkan porsi *data* tersebut. Perhitungan akurasi tersebut dengan menggunakan persamaan sebagai berikut [15]:

$$\boldsymbol{Akurasi} = \frac{\sum \boldsymbol{data\ uji\ benar\ klasifikasi}}{\sum \boldsymbol{total\ data\ uji}} \times \mathbf{100}\% \qquad \textbf{(II-7)}$$

3. *Fold* ke-2 adalah ketika bagian ke-2 menjadi data uji (*testing* data) dan sisanya menjadi data latih (*training* data). Selanjutnya, hitung akurasi berdasarkan porsi data tersebut.
4. Demikian seterusnya hingga mencapai *fold* ke-K. Hitung rata-rata akurasi dari K buah akurasi di atas. Rata-rata akurasi ini menjadi akurasi final.

II.13. Android

Android adalah sistem operasi berbasis Linux yang dirancang untuk perangkat seluler layar sentuh seperti telepon pintar dan komputer *tablet*. Android awalnya dikembangkan oleh Android, Inc., dengan dukungan finansial dari Google, yang kemudian membelinya pada tahun 2005. Sistem operasi ini dirilis secara resmi pada tahun 2007, bersamaan dengan didirikannya Open Handset Alliance, konsorsium dari perusahaan-perusahaan perangkat keras, perangkat lunak, dan telekomunikasi yang bertujuan untuk memajukan standar terbuka perangkat seluler. Ponsel Android pertama mulai dijual pada bulan Oktober 2008. [16]

II.13.1. *Android Life Cycle*

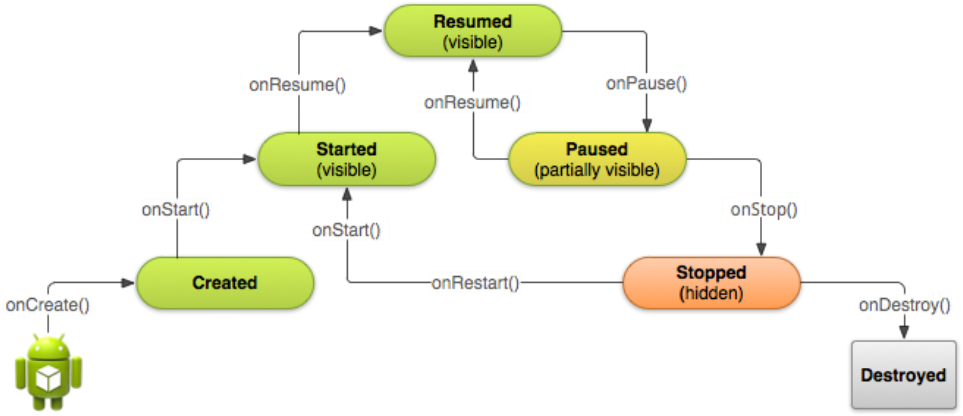
Aplikasi android terdiri dari beberapa fungsi dasar seperti mengedit catatan, memutar file musik, membunyikan alarm, atau membuka kontak telepon. Fungsi-fungsi tersebut dapat diklasifikasikan ke dalam empat komponen android yang berbeda seperti ditunjukkan pada , klasifikasi tersebut berdasarkan kelas-kelas dasar java yang digunakan. [17]

Tabel II. 1 Komponen Aplikasi Android

Functionality	*Java Base Class*	*Examples*
Focused thing a user can do	Activity	Edit a note, play a game
Background process	Service	Play music, update weather icon
Receive messages	BroadcastReceiver	Trigger alarm upon event
Store and retrieve data	ContentProvider	Open a phone contact

Setiap aplikasi pasti menggunakan minimal satu dari komponen tersebut, akan tetapi terdapat beberapa komponen yang mengharuskan mencantumkan specified permission sebelum digunakan seperti komponen *Service*, *BroadcastReceiver. ContentProvider.* [17]

Android memiliki paradigma pemrograman lain tidak seperti paradigma pemrograman biasa di mana aplikasi yang dijalankan pada *fungsi main(),* sistem android menjalankan kode dalam *method Activity* dengan menerapkan metode *callback* tertentu yang sesuai dengan tahap tertentu dari siklus hidup. Setiap aplikasi yang berjalan dalam sistem operasi android memiliki siklus hidup yang berbeda dengan aplikasi desktop atau web, Hal ini dikarenakan aplikasi *mobile* memiliki tingkat interupsi proses yang lumayan tinggi seperti ketika h*andling* panggilan masuk aplikasi diharuskan menghentikan proses sementara, Penerapan siklus hidup juga berguna untuk memastikan aplikasi tidak menghabiskan sumber daya baterai pengguna. [16]



Gambar II. 9 Siklus Hidup Android

Terdapat beberapa state dalam siklus hidup android yang terjadi seperti diilustrasikan pada Gambar II. 9 akan tetapi hanya beberapa dari state tersebut yang menjadi statis diantaranya: [16]

1. *Resumed*

Resumed terjadi ketika aplikasi berjalan setelah state paused . State ini akan menjalankan perintah program yang ditulis pada method *onResume().*

2. *Paused*

Dalam keadaan ini aktivitas yang terjadi dihentikan secara sementara tetapi masih terlihat oleh pengguna karena terdapat proses yang memiliki prioritas lebih tinggi seperti panggilan telepon. Aplikasi tidak dapat menjalankan perintah apapun ataupun menampilkan apapun dalam state ini.

3. *Stopped*

Dalam keadaan ini, aplikasi benar-benar tidak ditampilkan dan tidak terlihat oleh pengguna tetapi masih meninggalkan *service* di *background*.

State lain seperti *Created* dan *Started* bersifat sementara dan sistem dengan cepat menjalankan state berikutnya dengan memanggil metode *life cycle callback* berikutnya. Artinya, setelah sistem *OnCreate()* dipanggil, dengan cepat sistem akan memanggil *method OnStart(),* krmudian diikuti oleh *onResume().* [17]

II.13.2. Fitur

Android memiliki beberapa fitur utama yang sering digunakan dalam proses pembangunan aplikasi diantaranya adalah: [16]

1. *Multi-proses dan App Widgets*

Sistem operasi android tidak melarang prosesor menjalankan lebih dari satu aplikasi dalam satu waktu. Sistem operasi android dapat mengatur aplikasi dan thread yang berjalan secara *multitasking.* Keuntungan yang didapat adalah ketika aplikasi berjalan dan berinteraksi dengan pengguna di layer depan sistem operasi, proses dari aplikasi lain dapat berjalan untuk melakukan pembaruan informasi. Sebagai contoh misalnya ketika pengguna memainkan *game*, proses lain dapat berjalan di belakang aplikasi seperti memeriksa harga saham dan memunculkan peringatan. [16]

App Widgets adalah mini aplikasi yang dapat *embedded* dalam aplikasi seperti home screen. App widgets dapat menjalankan proses *request* seperti musik streaming atau mendeteksi suhu ruangan secara *background.*. [16]

Multi-proses dapat memberikan manfaat berupa *user experience* yang lebih banyak, namun penggunaan fitur tersebut dapat menghabiskan banyak energi baterai jika penggunaan tidak benar. [17]

2. *Touch Gestures dan Multi-touch*

Touchscreen adalah *user interface* intuitif yang digunakan banyak *smartphone* di dunia. Dengan fitur ini interaksi dapat dibuat lebih mudah karena cukup dengan menggunakan jari tangan. *Multi-touch* adalah kemampuan yang dapat melakukan tracking lebih dari satu tangan dalam satu waktu, Fitur ini sering digunakan untuk interaksi memperbesar atau memutar objek. Selain itu pengembang dapat membuat interaksi baru dengan memanfaatkan fitur tersebut. [17]

3. *Hard dan Soft Keyboard*

Salah satu fitur pada perangkat smartphone adalah tombol fisik dan non fisik, tombol fisik digunakan untuk navigasi pendukung dalam pengoperasian android.. Pengembang aplikasi tidak perlu secara manual untuk mengintegrasikan tombol tersebut dalam aplikasi. Tombol non fisik adalah tombol yang dibuat oleh sistem operasi seperti *keyboard virtual*, dan tombol navigasi aplikasi. [17]

II.14. *Global Positionong System*

Global Positioning System (GPS) merupakan sistem navigasi yang menggunakan satelit dalam penggunaannya. GPS terdiri dari tiga bagian yaitu sistem kontrol, satelit dan pengguna. Sistem kontrol adalah bagian yang mengontrol pergerakan satelit – satelit yang ada dan saling berinteraksi satu sama lain kemudian pengguna adalah alat navigasi yang digunakan seperti perangkat *mobile* yang kini sudah memiliki fitur GPS di dalamnya. GPS biasanya digunakan untuk menunjukan suatu lokasi yang berada di permukaan bumi dengan tingkat akurasi yang cukup baik yaitu kurang dari 10 meter selama tidak ada benda padat yang dapat menghambat sinyal untuk mendapatkan lokasi pengguna [18].

II.14.3. *Location Based Services*

Layanan lokasi dapat didefinisikan sebagai perangkat lokasi atau posisi dengan informasi lain sehingga dapat memberikan layanan yang mengintegrasikan nilai tambah pengguna *mobile* [19]. *Location Based Services (LBS)* atau layanan berbasis lokasi adalah sebuah layanan informasi yang dapat diakses dengan

perangkat bergerak melalui jaringan dan mampu menampilkan posisi secara geografis keberadaan perangkat tersebut. *Location Based Services (LBS)* dapat berfungsi sebagai layanan untuk mengidentifikasi lokasi dari seseorang atau objek tertentu.

Hal yang paling penting dari *Location Based Services (LBS)* dapat bekerja sesuai yang diinginkan oleh pengembang aplikasi android. Androidpun memungkinkan pengembang menentukan metode pencarian lokasi yang dibutuhkan dan juga dapat mengatur kebutuhan daya, biaya dan akurasi berdasarkan spesifik yang akan untuk aplikasi yang dikembangkan.



Gambar II. 10 Alur *Location Base Service*

II.14.4. Unsur Utama *Location Based Services* (LBS)

Location Based Service (LBS) atau layanan berbasis lokasi adalah istilah umum yang digunakan untuk menggambarkan teknologi yang digunakan untuk menemukan lokasi perangkat yang kita gunakan. Dua unsur utama dari *Location Based Services (LBS)* adalah: [20]

a. *Location Manager (API Maps):* Menyediakan pernagkat bagi sumber atau *source* untuk LBS, *Application Programming Interface* (API) *Maps* menyediakan fasilitas untuk menampilkan atau memanipulasi peta. Paket ini berada pada “com.goole.android.maps;”.

b. *Location Providers (API Location)*: Menyediakan teknologi pencarian lokasi yang digunakan oleh perangkat. PAI Location berhubungan dengan dara GPS *(Global Positioning System)* dan data lokasi *real-time*. *API Location* berada pada paket Android yaitu dalam paket "android.location". Lokasi, perpindahan, serta kedekatan dengan lokai tertentu dapat ditentukan melalui *Location Maneger.*

II.14.5. Komponen Utama *Location Based Services* (LBS)

Terdapat lima komponen pendukung utama dalam teknologi layanan berbasis lokasi, antara lain: [21]

1. Piranti *Mobile,* adalah satu komponen penting dalam LBS. Piranti ini berfungsi sebagai alat bantu bagi pengguna untuk meminta informasi. Hasil dari informasi yang diminta dapat berupa teks, suar, gambar dan lain sebgainya. Piranti *mobile* yang dapat digunakan bisa berupa PDA, *smartphone, laptop.* Selain itu, piranti *mobile* dapat juga berfungsi sebagai alat navigasi di kendaraan seperti halnya navigasi berbasis GPS.
2. Jaringan Komunikasi, komponen ini berfungsi sebagai jalur penghubung ang dapat mengirikan data-data yang dikirim oleh pengguna dari piranti *mobile*-nya untuk kemudian dikrimkan ke penyedia layanan dan kemudian hasil permintaan tersebut dikirimkan kembali oleh penyedia layanan kepada pengguna.
3. Komponen *Positioning* (Penujuk Posisi/Lokasi), setiap layanan yang di berikan oleh penyedia layanan biasanya akan berdasarkan pada posisi pengguna yang meminta layanan tersebut. Oleh karena itu diperlukan komponen yang ebrfungsi sebagai pengolah/pemroses yang akan menentukan posisi pengguna layanan saat itu. Posisi pengguna tersebut bisa didapatkan melalui jaringan komunikasi *mobile* atau juga menggunakan *Global Positioning System* (GPS).
4. Penyedia layanan dan aplikasi, merupakankomponen LBS yang memberikan berbagai macam layanan yang bsia digunakan oleh pengguna. Sebagai contoh ketika pengguna meminta layanan agar bisa tahu posisinya saat itu, maka

aplikasi dan penyedia layanan langsung memproses permintaan tersebut, mulai dari menghitung dan menetukan posisi pengguna, menentukan rute jalan, mencari *Yellow Pages* sesuai dengan permintaan, dan masih banyak lagi yang lainnya.

5. Penyedia dat adan konten, penyedia layanan tidak selalu menyimpan seluruh data dan informasi yang diolahnya. Karena bisa jadi berbagai macam data dan informasi yang diolah tersebut berasal dari pengembang/pihak ketiga yang memang memiliki otoritas untuk menyimpannya. Sebagai contoh basis data geografis dan lokasi bisa saja berasal dari badan-badan milik pemerintah atau juga data-data perusahaan/bisnis/industri bisa saja berasal dari *Yellow Page*, maupun perusahaan penyedia data lainnya.

II.15. JSON

JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (generate) oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemrograman JavaScript, Standar ECMA-262 Edisi ke-3 – Desember 1999. JSON merupakan format teks yang tidak bergantung pada bahasa pemrograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh programmer keluarga C termasuk C, C++, C#, Java, JavaScript, Perl, Python dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran data. JSON terbuat dari dua struktur:

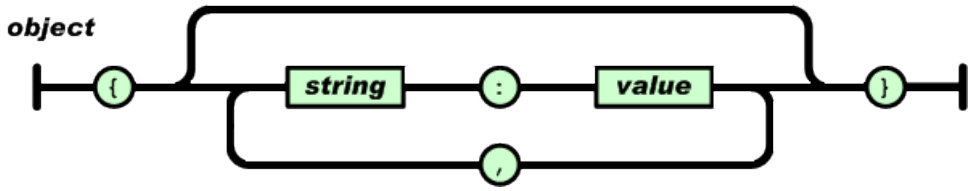
1. Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel hash (*hash table*), daftar berkunci (*keyed list*), atau *associative array* [22].
2. Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*) [22].

Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemrograman modern mendukung struktur data ini dalam

bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemrograman yang juga berdasarkan pada struktur data ini. JSON menggunakan bentuk sebagai berikut:

1. Objek

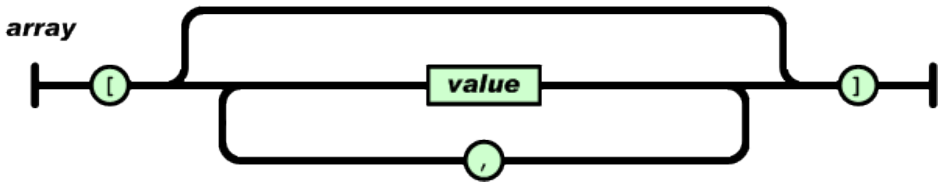
Objek adalah sepasang nama / nilai yang tidak terurutkan. Objek dimulai dengan { (kurung kurawal buka) dan diakhiri dengan } (kurung kurawal tutup). Setiap nama diikuti dengan : (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh , (koma). Objek biasanya digunakan untuk menyimpan data tunggal dalam bentuk JSON [22].



Gambar II. 11 Objek JSON

2. Larik

Larik adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung kotak buka) dan diakhiri dengan] (kurung kotak tutup). Setiap nilai dipisahkan oleh , (koma). Larik dalam JSON dapat digunakan sebagai value dari JSON object hal ini dapat berguna jika JSON menyimpan data bertingkat [22].



Gambar II. 12 *Array* JSON

Bentuk data JSON objek dan larik dapat saling dikombinasikan untuk mendukung struktur data yang lebih kompleks. JSON mendukung beberapa tipe data untuk menjadi value seperti Angka, *String*, Boolean dan nilai *NULL* [22].

II.16. Google Maps API

Google Maps API adalah sebuah layanan yang memungkinkan Google Maps agar dapat diintegrasikan ke dalam aplikasi yang memang membutuhkan fitur dari

Google Maps ini. Google Maps API adalah sebuah library Javascript. Dengan menggunakan Google Maps API ini dapat memudahkan pengembang dalam membuat aplikasi yang membutuhkan teknologi peta di dalamnya dan tidak perlu memikirkan cara membuatnya secara manual karena dengan mengunakan Google Maps API ini sudah semua yang dibutuhkan terkait teknologi peta digital sudah cukup lengkap dan dapat dimanfaatkan untuk aplikasi yang membutuhkannya [18].

Saat ini Google Maps API telah mengalami perkembangan, hingga saat ini Google Maps API telah mencapai versi 3. Namun, Google Maps API versi 3 ini masih hanya bisa digunakan oleh pemrograman berbasis *web* dengan menggunakan Javascript. Untuk penggunaan Google Maps API pada perangkat *mobile* seperti misalnya android Google Maps API yang bisa digunakan saat ini telah mencapai versi 2. Perkembangan pada Google Maps Versi 2 ini adalah dengan memiliki fitur–fitur yang lengkap dari versi sebelumnya salah satunya adalah dapat menampilkan peta dengan sudut yang di inginkan dan berupa 3D *view*. Selain itu pada Google Maps API terdapat beberapa pilihan model peta yang ditampilkan seperti *Roadmap, Satellite, Terrain dan Hybrid.*

II.17. *Structured Analysis and Design*

Structured analysis and design adalah pendekatan sistem analisis dan desain sistem informasi secara hierarki. SADM diproduksi untuk CCTA, kantor pemerintah Inggris yang berkaitan dengan penggunaan teknologi di pemerintahan dari tahun 1980 dan seterusnya [23].

II.18. *Object Oriented Analysis Desaign*

Konsep OOAD mencakup analisis dan desain sebuah sistem dengan pendekatan objek, yaitu analisis berorientasi objek (OOA) dan desain berorientasi objek (OOD). OOA adalah metode analisis yang memeriksa *requirement* (syarat/keperluan) yang harus dipenuhi sebuah sistem dari sudut pandang kelas-kelas dan objek-objek yang ditemui dalam ruang lingkup sistem. Sedangkan OOD adalah metode untuk mengarahkan arsitektur *software* yang didasarkan pada manipulasi objek-objek sistem atau subsistem. [24]

II.18.6. UML *(Unifed Modeling Language)*

Unified Modeling Language (UML) adalah termasuk ke dalam rumpun jenis pemodelan notasi grafis yang didukung oleh meta-model tunggal, Pemodelan ini berguna untuk membantu dalam menjelaskan dan merancang perangkat lunak yang dibangun dengan object-oriented (OO). UML merupakan standar terbuka yang dikelola oleh Open Management Group (OMG) yang berada dibawah naungan perusahaan-perusahaan konsorsium terbuka. UML merupakan suatu bahasa pemodelan yang terdiri banyak model diantaranya adalah [24] :

1. *Use case diagram*

Use case diagram merupakan suatu pemodelan yang menitik beratkan pada fungsionalitas utama dalam suatu konteks sistem. *Use case diagram* juga menekankan tingkah laku fungsional utama dalam sistem berinteraksi dengan objek diluar sistem tersebut. Selain itu, *use case diagram* juga telah menitik beratkan jenis hubungan diantara fungsi utama. Adapun komponen-komponen *dalam use case diagram* antaranya: [24]

a. Aktor

Aktor merupakan suatu entitas yang berkaitan dengan sistem tapi bukan dari bagian dalam sistem itu sendiri. Aktor berada diluar sistem namun berkaitan erat dengan fungsionalitas didalamnya. Aktor dapat memiliki hubungan secara langsung terhadap fungsi utama baik terhadap salah satu atau semua fungsionalitas utama. Aktor juga dapat dibagi terhadap berbagai jenis atau tingkatan dengan cara digeneralisasi atau dispesifikasi tergantung kebutuhan sistemnya. Aktor biasanya dapat berupa pengguna atau *database* yang secara pandang berada dalam suatu ruang lingkup sistem tersebut [24].

b. *Use Case*

Use case merupakan gambaran umum dari fungsi atau proses utama yang menggambarkan tentang salah satu perilaku sistem. Perilaku sistem ini terdefinisi dari proses bisnis sistem yang akan dimodelkan. Tidak semua proses bisnis digambarkan secara fungsional pada use case, tetapi yang digambarkan hanya fungsionalitas utama yang berkaitan dengan sistem. *Use case* menitik beratkan

bagaimana suatu sistem dapat berinteraksi baik antar sistem maupun diluar sistem [24].

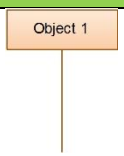
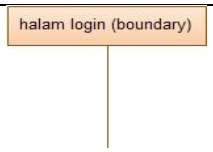
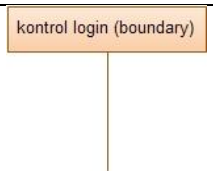
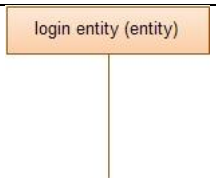
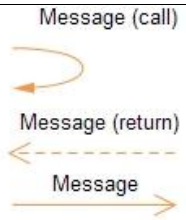
2. *Class* Diagram

Class diagram merupakan diagram yang selalu ada di permodelan sistem berorientasi objek. *Class* diagram menunjukkan hubungan antar *class* dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan. Kelas pada kelas diagram terdiri dari 3 bagian utama yaitu nama kelas, isi properti dari kelas beserta metode yang ada pada kelas tersebut. Kelas juga memiliki jenis-jenis hubungan seperti asosiatif, dependensi, agregasi, komposisi, spesifikasi dan generalisasi. Hubungan ini digunakan untuk menggambarkan bagaimana hubungan dan interaksi yang terjadi antar kelas. Masing-masing komponen penyusun kelas memiliki hak akses seperti *public, private,* dan *protected* [24].

3. *Sequence* Diagram

Sequence diagram menjelaskan secara detil urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari *use case*: interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi.

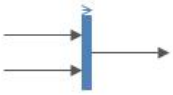
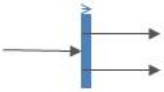
Tabel II. 2 Elemen-elemen *Sequence* Diagram

No	Nama	Penjelasan	Gambar
1	Objek Lifeline	Menggambarkan batasan objek	
2	Boundary	Berhubungan dengan proses *input output* interface	
3	Controller	Berhubungan dengan proses	
4	Entity	Berhubungan dengan inpu-*output* data	
5	Message Arrow	Menggambarkan alir proses, perintah atau pengiriman data	
6	Aktivasi	Menggambarkan aktivitas objek	
7	Aktor	Menggambarkan aktor suatu objek	

A. Diagram Aktivasi (Activity Diagram)

Diagram aktifitas adalah diagram *flowchart* yang diperluas untuk menunjukkan aliran kendali satu aktivitas ke aktivitas yang lain. Diagram aktifitas digunakan untuk memodelkan aspek dinamis sistem. Diagram aktivitas berupa operasi-operasi dan aktivitas-aktivitas di *use case*.

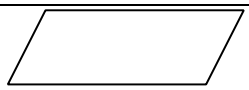
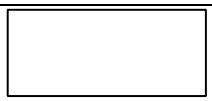
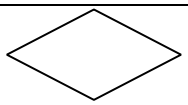
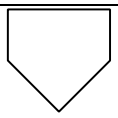
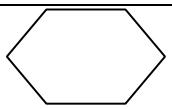
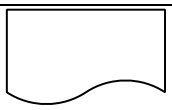
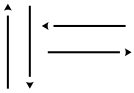
Tabel II. 3 Elemen Activity Diagram

Simbol	Keterangan
	Start point
	End point
	Aktivities
	Join (penggabungan)
	Fork(percabangan)
Swimline	Sebuah cara mengelompokkan aktivitas berdasarkan aktor (mengelompokkan aktivitas dalam sebuah urutan yang sama)

II.18.1.1. *Flowchart*

Flowchart adalah gambaran urutan logika dari suatu prosedur pemecahan masalah, sehingga *flowchart* merupakan langkah-langkah penyelesaian masalah yang dituliskan dalam simbol-simbol tertentu [25]. Fungsi dari simbol-simbol *flowchart* adalah sebagai berikut [25]:

Tabel II. 4 Fungsi simbol-simbol *flowchart*

Simbol	Nama	Keterangan
	Terminal	Menyatakan permulaan atau akhir program
	*Input* /*output*	Menyatakan proses *input*/*output* tanpa tergantung jenis peralatannya
	Process	Menyatakan suatu proses atau tindakan yang dilakukan oleh komputer
	Decission	Menunjukkan suatu kondisi tertentu yang akan menghasilkan dua jawaban ya atau tidak
	Connector	Menyatakan proses sambungan dari proses ke proses lainnya dalam halaman yang sama
	Offline connector	Menyatakan proses sambungan dari proses ke proses lainnya dalam halaman yang berbeda
	Predefined process	Menyatakan menyediakan tempat penyimpanan suatu pengolahan untuk memberi harga awal
	Punched card	Menyatakan *input* berasal dari kartu atau *output* ditulis ke kartu
	Punch tape	
	Document	Mencetak keluaran dalam bentuk dokumen (melalui printer)
	Flow	Menyatakan jalannya arus suatu proses

II.19. Pengujian Fungsionalitas Sistem

Pengujian perangkat lunak adalah sangat diperlukan dalam suatu system informasi, dimana dengan melakukan suatu pengujian akan ditemukan kesalahan atau error yang muncul dari system perangkat lunak tersebut. Dengan demikian tentunya seorang programmer akan bisa mengetahui dan apa yang harus dikerjakan selanjutnya. Pentingnya pengujian perangkat lunak dan implikasinya yang mengacu pada kualitas perangkat lunak tidak dapat terlalu ditekan karena melibatkan sederetan aktivitas produksi di mana peluang terjadinya kesalahan manusia sangat besar dan arena ketidakmampuan manusia untuk melakukan dan berkomunikasi dengan sempurna maka pengembangan perangkat lunak diiringi dengan aktivitas jaminan kualitas. [26]

Pengujian perangkat lunak mencakup :

1. Strategi : Mengintegrasikan metode perancangan kasus uji dalam sekumpulan langkah yang direncanakan.
2. Metode pengujian, mencakup perancangan kasus uji dengan menggunakan metode *White Box* atau *Black Box*

Metode *Black Box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi *input* yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. *Black Box* dapat menemukan kesalahan dalam kategori berikut :

1. Fungsi-fungsi yang tidak benar atau hilang.
2. Kesalahan *interface.*
3. Kesalahan dalam strutur data atau akses basisdata *eksternal*.
4. Inisialisasi dan kesalahan terminasi.
5. Validitas fungsional
6. kesensitifan sistem terhadap nilai *input* tertentu.
7. Batasan dari suatu data

Tipe dari Black Box *Testing* :

1. *Equivalence class partitioning*

a. Bagi domain *Input* ke dalam beberapa kelas yang nantinya akan dijadikan sebagai kasus uji.
b. Kelas yang telah terbentuk disajikan sebagai kondisi *input* dalam kasus uji.
c. Kelas tersebut merupakan himpunan nilai-nilai yang valid dan tidak valid.
d. Kondisi *input* bisa merupakan suatu range, harga khusus, suatu himpunan, atau suatu boolean.
e. Bila kondisi *input* berupa suatu range, maka *input* kasus ujinya satu valid dan dua yang invalid.
f. Bila kondisi *input* berupa suatu harga khusus, maka *input* kasus ujinya satu valid dan dua yang invalid.
g. Bila kondisi *input* berupa suatu anggota himpunan, maka *input* kasus ujinya satu valid dan dua yang invalid.
h. Bila kondisi *input* berupa suatu anggota Boolean , maka *input* kasus ujinya satu valid dan dua yang invali

2. *Sample testing*
   a. Melibatkan sejumlah nilai yang dipilih dari data masukan kelas ekivalensi.
   b. Integrasikan nilai tersebut ke dalam kasus uji.
   c. Nilai yang dipilih dapat berupa konstanta atau variabel Limit *Testing* .
   d. Kasus uji yang memproses nilai batas (atau titik singular)
   e. Nilai batas di*node*kan dari kelas ekivalensi dengan mengambil nilai yang sama atau mendekati nilai yang membatasi kelas akivalensi tersebut.
   f. Limit test also juga melibatkan data keluaran dari ekivalensi kelas.
   g. Pada kasus segi tiga, misalnya limit *testing* mencoba untuk mendeteksi apakah a+b >= c dan bukan a + b > c.
   h. Bila kondisi *input* menentukan suatu range, maka kasus ujinya harus mencakup pengujian nilai batas dari range dan nilai invalid yang dekat dengan nilai batas. Misal bila rangenya antara [-1.0, +1.0], maka *input* untuk kasus ujinya adalah -1.0, 1.0, - 1.001,1.001.

i. Bila kondisi *input*nya berupa harga khusus kasus ujinya harus mencakup nilai minimum dan maksimum. Misal suatu file dapat terdiri dari 1 to 255 record, maka kasus ujinya harus mencakup untuk nilai 0, 1, 255 and 256, atau uji saat keadaan record kosong dan record penuh.

3. *Limit testing*
4. *Robustness testing*

   Data dipilih dari luar range yang didefinisikan. Tujuan pengujian ini adalah untuk membuktikan tidak adanya kejadian yang katastropik yang dihasilkan akibat adanya keabnormalan.

5. *Behavior testing*

   Suatu pengujian yang hasilnya hanya dapat dievaluasi per sub program, tidak bisa dilakukan per modul.

6. *Requirement testing*
   a. Menyusun kasus uji untuk tiap kebutuhan yang berkorelasi dengan modul / CSU.
   b. Tiap kasus uji harus dapat dirunut dengan kebutuhan perangkat lunaknya melalui matriks keterunutuan

Metode *White Box* merupakan metode pengujian dengan menggunakan struktur kontrol program untuk untuk memperoleh kasus uji. Dengan menggunakan white box akan didapatkan kasus uji yang :

1. Menjamin seluruh jalur independen di dalam modul yang dieksekusi sekurang-kurangnya sekali
2. Menguji semua keputusan logikal
3. Menguji seluruh *Loop* yang sesuai dengan batasannya
4. Menguji seluruh struktur data internal yang menjamin validitas
5. Basis *Path* adalah teknik uji coba *white box* (Tom Mc Cabe).
6. Basis *Path* : untuk mendapatkan kompleksitas lojik dari suatu prosedur dan menggunakan ukuran ini sebagai petunjuk untuk mendefinisikan himpunan jalur yang akan diuji.
7. Basis *Path* menggunakan notasi graf untuk menggambarkan aliran kontrolnya.