Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

Team project ©2017
Dony Pratidana S. Hum | Bima Agus Setyawan S. IIP

BAB II

TINJAUAN PUSTAKA

2.1 Kalori

Kalori adalah sebuah satuan untuk menghitung jumlah energi. Setiap makanan yang dimakan, mengandung sejumlah kalori yang dibutuhkan oleh tubuh untuk melakukan suatu aktivitas. Kalori bisa diibaratkan sebagai bahan bakar dari suatu mesin untuk bergerak atau menjalankan tugasnya. Kalori yang terkandung dalam makanan disediakan oleh karbohidrat, protein, dan lemak. Tiap gram lemak mengandung 9 kalori, sedangkan tiap gram protein dan karbohidrat masing-masing mengandung 4 kalori (Adi, 2013).

Untuk mengetahui kebutuhan kalori seseorang, dapat dilakukan perhitungan berdasarkan *Basal Metabolic Rate* (BMR). BMR adalah kebutuhan kalori minimum yang dibutuhkan seseorang hanya untuk sekedar mempertahankan hidup, dengan asumsi bahwa orang tersebut dalam keadaan istirahat total, tidak melakukan aktivitas sedikitpun. Berikut persamaan yang digunakan untuk menghitung BMR:

BMR Pria = 66 + (13.7 * Berat) + (5 * Tinggi) – (6.8 * Usia) ……..Rumus 2.1

BMR Wanita = 655 + (9.6 * Berat) + (1.8 * Tinggi) – (4.7 * Usia) …...Rumus 2.2

Dengan ketentuan :

- Berat dalam kg
- Tinggi dalam cm
- Usia dalam tahun

Rancang Bangun..., Daniel Enrico, FTI, 2014

Setelah itu, tingkat aktivitas *user* juga memiliki pengaruh yang cukup besar dalam pengelolaan energi manusia. Jumlah energi yang digunakan untuk beraktivitas akan berbeda-beda sesuai dengan intensitas dan jenis aktivitasnya (http://www.topendsports.com/nutrition/index.htm, 2014). Setiap tingkat aktivitas memiliki konstanta tersendiri dalam perhitungan kebutuhan kalori harian. Salah satu cara untuk mengukur intensitas dari aktivitas seseorang adalah dengan *Metabolic Equivalent Task* (MET). Tingkat aktivitas yang diukur berdasarkan MET dapat dibagi menjadi tiga tingkat, yaitu ringan, sedang dan berat. Berikut adalah tabel tingkat aktivitas beserta contoh aktivitasnya (Stephanie, 2012).

Tabel 2.1 Tingkat Aktivitas, Nilai MET, dan Contoh Aktivitasnya (Stephanie, 2012)

Tingkat Aktivitas	**MET/jam**	**Aktivitas**
Ringan	1,3	Berdiri
	1,5	Membaca, berbicara
	1,8	Duduk, belajar, mencatat
	2	Berjalan pelan (< 1.5 km/jam), bermain musik, menjaga toko, memasak, pekerjaan rumah tangga
Sedang	3	Menjalankan kendaraan berat (traktor, bus, truk), naik-turun tangga, mengangkut barang ringan, pekerjaan kantoran
	4	Bersepeda santai (< 2 km/jam), mengecat, pekerjaan tukang
	4,5	Berenang
	5	Berjalan cepat, bersepeda (2-4 km/jam)
Berat	6	Tenis lapangan, berkebun (mencangkul, menggali), mendaki gunung
	7	Jogging
	8	Kegiatan aerobik (Push-up, Sit-up, dll)
	10	Berlari (8-10 km/jam)
	13,5	Berlari (10-12 km/jam)

Setelah mengelompokkan aktivitas ke dalam tingkatan-tingkatan berdasarkan nilai MET-nya, masing-masing tingkatan tersebut juga memiliki nilai

konstanta tingkat aktivitas yang berbeda-beda, baik pria maupun wanita, dimana nilai tersebut akan mempengaruhi kebutuhan kalori seseorang. Konstanta tingkat aktivitas tersebut akan ditampilkan pada tabel berikut (Stephanie, 2012).

Tabel 2.2 Konstanta Tingkat Aktivitas

Aktivitas	Pria	Wanita
Ringan	1,3	1,3
Sedang	1,65	1,55
Berat	1,76	1,7

Setelah mengetahui konstanta tingkat aktivitas, maka digunakan persamaan Harris-Benedict untuk menghitung kebutuhan kalori harian sebagai berikut.

Kebutuhan Kalori = BMR x Konstanta Tingkat Aktivitas ..... Rumus 2.3

Selanjutnya, akan dilakukan pengecekan proporsi tubuh *Body Mass Index* (BMI) atau berat ideal tubuh dengan menggunakan persamaan berikut.

BMI = Berat / (Tinggi * Tinggi) ……. Rumus 2.4

Dengan ketentuan :

- Berat dalam kg
- Tinggi dalam m

Kemudian, berdasarkan hasil perhitungan BMI tersebut, maka seseorang dapat digolongkan ke dalam kategori-kategori tertentu, yang mana juga akan mempengaruhi kebutuhan kalori akhir orang tersebut. Berikut adalah tabel kategori dan pengaruhnya terhadap kebutuhan kalori seseorang.

Tabel 2.3 Kategori Proporsi Tubuh dan Pengaruhnya

BMI	Kategori	Kalori
< 18,5	Underweight	Kalori + 100
18,5 <= x <= 24,9	Normal	Kalori
> 24,9	Overweight	Kalori - 700

2.2 Sistem Operasi Android

Gambar 2.1 Logo Android
(Sumber: http://www.info-asik.com/2013/01/macam-macam-sistem-operasi.html)

Android adalah sistem operasi berbasis linux yang dirancang untuk perangkat seluler layar sentuh, seperti *smartphone* 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. Android menyediakan platform yang terbuka bagi para pengembang untuk menciptakan aplikasi mereka (Riyan, 2014).

Android merupakan generasi baru *platform mobile, platform* yang membiarkan pengembang untuk melakukan pengembangan sesuai dengan yang diharapkannya. Sistem operasi yang mendasari Android dilisensikan di bawah GNU, *General Public Licence Version* 2 (GPLv2), yang sering dikenal dengan istilah "*copyleft*" lisensi dimana setiap perbaikan pihak ketiga harus terus jatuh di bawah *terms*. Android didistribusikan di bawah Lisensi *Apache Software* (ASL/Apache2), yang memungkinkan untuk distribusi kedua dan seterusnya. Komersialisasi pengembang dapat memilih untuk meningkatkan *platform* tanpa harus memberikan perbaikan mereka ke masyarakat *open source*. Sebaliknya, pengembang dapat keuntungan dari perangkat tambahan, seperti perbaikan dan mendistribusikan ulang pekerjaan mereka di bawah lisensi apapun yang mereka inginkan. Pengembang aplikasi Android diperbolehkan untuk mendistribusikan aplikasi mereka di bawah skema lisensi apapun yang mereka inginkan (Ichwan, Muhammad & dkk, 2013).

Google mengibaratkan Android sebagai sebuah tumpukan *software*. Setiap lapisan dari tumpukan ini menghimpun beberapa program yang mendukung fungsi-fungsi spesifik dari sistem operasi (Setiyawan, 2012). Berikut ini susunan dari lapisan-lapisan tersebut jika dilihat dari lapisan dasar hingga lapisan teratas.

- Linux Kernel

  Android dibangun di atas kernel Linux 2.6. Namun secara keseluruhan Android bukanlah linux, karena dalam Android tidak terdapat paket standar

yang dimiliki oleh linux lainnya. Linux merupakan sistem operasi terbuka yang handal dalam manajemen memori dan proses. Oleh karenanya pada Android hanya terdapat beberapa *service* yang diperlukan, seperti keamanan, manajemen *memori*, manajemen proses, jaringan dan *driver*. Kernel linux menyediakan *driver* layar, kamera, keypad, WiFi, *Flash Memory*, *audio*, dan IPC (*Interprocess Communication*) untuk mengatur aplikasi dan lubang keamanan, Android *Runtime* (Subiyantoro, 2013).

- Libraries

  Android menggunakan beberapa paket pustaka yang terdapat pada C/C++ dengan standar *Berkeley Software Distribution* (BSD) hanya setengah dari yang aslinya untuk tertanam pada kernel Linux. Beberapa pustaka diantaranya (Subiyantoro, 2013):

  • *Media Library* untuk memutar dan merekam berbagai macam format *audio* dan *video*,

  • *Surface Manager* untuk mengatur hak akses *layer* dari berbagai aplikasi,

  • *Graphic Library* termasuk di dalamnya SGL dan OpenGL, untuk tampilan 2D dan 3D,

  • SQLite untuk mengatur relasi *database* yang digunakan pada aplikasi,

  • SSl dan WebKit untuk *browser* dan keamanan internet.

- Android Runtime

  Pada Android tertanam paket pustaka inti yang menyediakan sebagian besar fungsi Android. Inilah yang membedakan Android dibandingkan dengan sistem operasi lain yang juga mengimplementasikan Linux. Android *Runtime*

merupakan mesin virtual yang membuat aplikasi android menjadi lebih tangguh dengan paket pustaka yang telah ada (Subiyantoro, 2013).

- Application Framework

  Kerangka aplikasi menyediakan kelas-kelas yang dapat digunakan untuk mengembangkan aplikasi Android. Selain itu, juga menyediakan abstraksi generik untuk mengakses perangkat, serta mengatur tampilan *user interface* dan sumber daya aplikasi. Bagian terpenting dalam kerangka aplikasi Android adalah sebagai berikut (Subiyantoro, 2013).

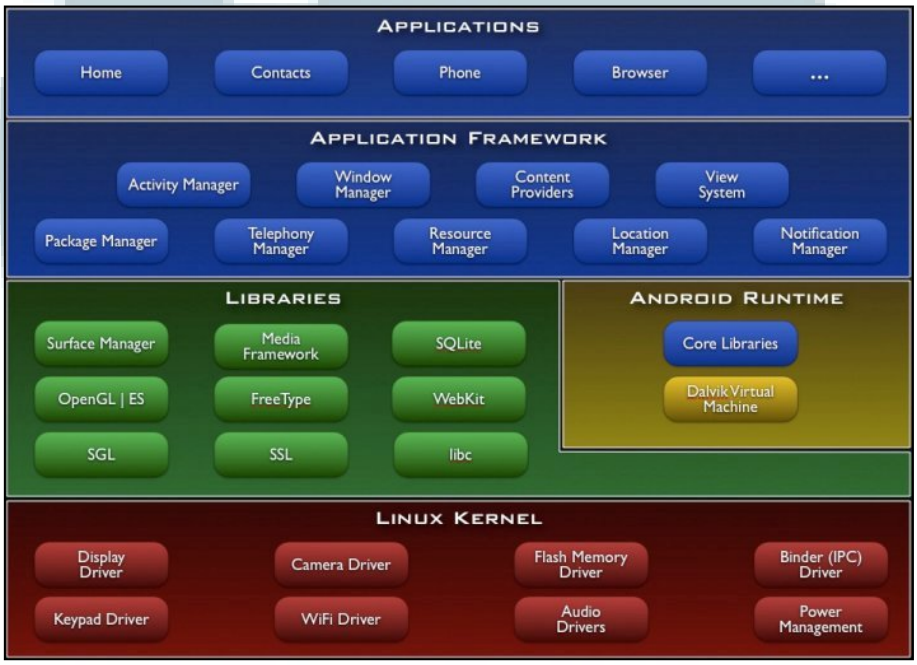
  1. *Activity Manager*, berfungsi untuk mengontrol siklus hidup aplikasi dan menjaga keadaan ”*Backstack*“ untuk navigasi penggunaan,
  2. *Content Providers*, berfungsi untuk merangkum data yang memungkinkan digunakan oleh aplikasi lainnya, seperti daftar nama,
  3. *Resuource Manager*, untuk mengatur sumber daya yang ada dalam program, serta menyediakan akses sumber daya di luar kode program, seperti karakter, grafik, dan *file layout*,
  4. *Location Manager*, berfungsi untuk memberikan informasi detail mengenai lokasi perangkat Android berada,
  5. *Notification Manager*, mencakup berbagai macam peringatan, seperti pesan masuk, janji, dan lain sebagainya yang akan ditampilkan pada status bar.

- Application Layer

  Puncak dari diagram arsitektur Android adalah lapisan aplikasi dan *widget*. Lapisan aplikasi merupakan lapisan yang paling tampak pada pengguna ketika menjalankan program. Pengguna hanya akan melihat program ketika

digunakan tanpa mengetahui proses yang terjadi di balik lapisan aplikasi. Lapisan ini berjalan dalam Android *runtime* dengan menggunakan kelas dan *service* yang tersedia pada *framework* aplikasi.

Lapisan aplikasi Android sangat berbeda dibandingkan dengan sistem operasi lainnya. Pada Android semua aplikasi, baik aplikasi inti (*native*) maupun aplikasi pihak ketiga berjalan di atas lapisan aplikasi dengan menggunakan pustaka API (*Application Programming Interface*) yang sama.



Gambar 2.2 Arsitektur Android
(Sumber: http://putrajatim.blogspot.com/2011/05/arsitektur-dan-aplikasi-android.html)

2.3 SQLite

SQLite merupakan sebuah sistem manajemen basis data relasional yang bersifat *ACID-compliant* dan memiliki ukuran pustaka kode yang relatif kecil, ditulis dalam bahasa C. SQLite merupakan proyek yang bersifat *public domain* yang dikerjakan oleh D. Richard Hipp.

Tidak seperti pada paradigma *client-server* umumnya, Inti SQLite bukanlah sebuah sistem yang mandiri yang berkomunikasi dengan sebuah program, melainkan sebagai bagian integral dari sebuah program secara keseluruhan sehingga protokol komunikasi utama yang digunakan adalah melalui pemanggilan API secara langsung melalui bahasa pemrograman. Mekanisme seperti ini tentunya membawa keuntungan karena dapat mereduksi *overhead*, *latency times*, dan secara keseluruhan lebih sederhana. Seluruh elemen basis data (definisi data, tabel, indeks, dan data) disimpan sebagai sebuah *file*. Kesederhanaan dari sisi desain tersebut bisa diraih dengan cara mengunci keseluruhan *file* basis data pada saat sebuah transaksi dimulai (Purnomo, Fredy & dkk, 2011).

Gambar 2.3 *Database* SQLite
(Sumber: http://www.mobnasesemka.com/tag/sqlite-database-server/)

2.4 Teori Voice Recognition

Voice Recognition atau pengenalan suara adalah proses otomatis mengenali kata-kata yang diucapkan kemudian mengubahnya ke dalam bentuk teks. Dengan kata lain, *voice recognition* merupakan suatu proses yang memungkinkan komputer untuk mengenali apa yang diucapkan oleh seseorang lalu mengubahnya ke bentuk tulisan. *Input* data yang diterima berupa suara atau ucapan manusia, kemudian sistem akan mengidentifikasi kata atau kalimat yang diucapkan dan menghasilkan keluaran atau output berupa teks yang sesuai dengan

apa yang diucapkan. *Input* yang berupa kata-kata tersebut diubah menjadi sinyal digital dengan cara mengubah gelombang suara menjadi sekumpulan angka lalu disesuaikan dengan kode-kode tertentu dan dicocokkan dengan suatu pola yang tersimpan dalam suatu perangkat. Untuk setiap ucapan yang berbeda akan dihasilkan pola ciri yang berbeda (Nurcahyadi, 2012).

Penganalisis sintaks akan melakukan transformasi sinyal ucapan dari domain waktu ke domain frekuensi. Pada domain frekuensi, untuk kurun waktu yang singkat, setiap sinyal dapat terlihat memiliki ciri-ciri yang unik. Namun demikian, pengucapan suatu unit bunyi ucapan atau fonem seringkali bervariasi antar orang yang berbeda, juga terpengaruh oleh berbagai faktor, seperti fonem-fonem di sekitarnya, kondisi, emosi, *noise*, dan faktor-faktor lainnya. Sistem *voice recognition* akan melakukan pengenalan untuk setiap unit bunyi pembentuk ucapan atau fonem, selanjutnya mencoba mencari kemungkinan kombinasi hasil ucapan yang paling dapat diterima. Bentuk sistem *voice recognition* yang sederhana adalah sistem yang hanya dapat mengenal sejumlah kata yang jumlahnya terbatas. Sistem ini biasanya lebih akurat dan lebih mudah dilatih, tetapi tidak dapat mengenal kata yang berada di luar kosakata yang pernah diajarkan (Nurcahyadi, 2012).

Ada dua jenis atau tipe *voice recognition* jika dilihat dari ketergantungan pembicara, antara lain :

a. *Independent Speech Recognition*, yaitu sistem pengenal ucapan tanpa terpengaruh dengan siapa yang berbicara, tetapi mempunyai keterbatasan dalam jumlah kosakata. Model ini akan mencocokkan setiap ucapan dengan kata yang dikenali dan memilih yang ”sepertinya” cocok. Untuk mendapatkan

kecocokan kata yang diucapkan maka digunakan model *statistic* yang dikenal dengan nama *Hidden Markov Model* (HMM).

b. *Dependent Speech Recognition*, yaitu sistem pengenal ucapan yang memerlukan pelatihan khusus dari pembicara, dimana hasil pelatihan dari masing-masing pembicara akan disimpan dalam sebuah profil. Profil inilah yang nantinya digunakan untuk berinteraksi dengan sistem pengenal ucapan dan sistem akan bergantung siapa yang berbicara. Sistem ini biasanya lebih mudah untuk dikembangkan, dimana contoh suara sudah dibuat sebelumnya dan disimpan dalam *database* atau basis data dan jumlah kosakatanya lebih besar dibandingkan dengan *independent speech recognition*. Proses pengenalan ucapan dengan cara membandingkan ucapan pembicara dengan contoh suara yang sudah ada (Nurcahyadi, 2012).

2.5 Algoritma Levenshtein Distance

Algoritma Levenshtein *Distance* merupakan matriks yang digunakan untuk mengukur perbedaan jarak antara dua *sequence*. Levenshtein *Distance* dibuat oleh Vladimir Levenshtein pada tahun 1965. Perhitungan *edit distance* didapatkan dari matriks yang digunakan untuk menghitung jumlah perbedaan *string* antara dua *string*. Perhitungan jarak antara dua *string* ini ditentukan dari jumlah minimum operasi perubahan untuk membuat *string* A menjadi *string* B. Ada tiga macam operasi utama yang dapat dilakukan oleh algoritma ini, yaitu (Adriyani, Ni Made Muni & dkk, 2012):

1. Operasi Pengubahan Karakter

   Operasi pengubahan karakter merupakan operasi menukar sebuah karakter dengan karakter lain contohnya penulis menuliskan *string* ‘yamg’ menjadi ‘yang’. Dalam kasus ini karakter ‘m’ diganti dengan huruf ‘n’.

2. Operasi Penambahan Karakter

   Operasi penambahan karakter berarti menambahkan karakter ke dalam suatu *string*. Contohnya *string* ‘kepad’ menjadi *string* ‘kepada’, dilakukan penambahan karakter ‘a’ di akhir *string*. Penambahan karakter tidak hanya dilakukan di akhir kata, namun bisa ditambahkan di awal maupun disisipkan di tengah *string*.

3. Operasi Penghapusan Karakter

   Operasi penghapusan karakter dilakukan untuk menghilangkan karakter dari suatu string. Contohnya *string* ‘barur’ karakter terakhir dihilangkan sehingga menjadi *string* ‘baru’. Pada operasi ini dilakukan penghapusan karakter ‘r’.

Perhitungan harga pengeditan pada setiap operasi yang dilakukan adalah sesuai dengan aturan berikut:

- $d(a,\varepsilon) = 1$ harga untuk menghapus *substring* a
- $d(\varepsilon,a) = 1$ harga untuk penyisipan *substring* a
- $d(a,b) = 1$ harga untuk substitusi *substring* a ke *substring* b
- $d(a,a) = 0$

Semakin besar angka yang dihasilkan oleh operasi Levenshtein *Distance* maka semakin besar perbedaan di antara kedua *string* tersebut (Benisius, Tanpa Tahun).

Untuk lebih jelasnya perhatikan matriks ilustrasi pencari Levenshtein *Distance* antara dua *string* berikut, yaitu 'penjara' dan 'jarak'. Jika dilihat sekilas, kedua *string* 'penjara' dan 'jarak' memiliki jarak 4. Berarti untuk mengubah *string* 'jarak' menjadi 'penjara' diperlukan 4 operasi sebagai berikut (Nafik, Muhammad Zakiya & dkk, 2014).

1. Menyisipkan karakter 'p'

   jarak -> pjarak

2. Menyisipkan karakter 'e'

   pjarak -> pejarak

3. Menyisipkan karakter 'n'

   pejarak -> penjarak

4. Menghapus karakter 'k'

   penjarak -> penjara

Dengan menggunakan representasi matriks, dapat ditunjukan dengan gambar berikut.

		p	e	n	j	a	r	a
	0	1	2	3	4	5	6	7
j	1	1	2	3	3	4	5	6
a	2	2	2	3	4	3	4	5
r	3	3	3	3	4	4	3	4
a	4	4	4	4	4	4	4	3
k	5	5	5	5	5	5	5	4

Gambar 2.4 Matriks Levenshtein *Distance* Kata Jarak dan Penjara

Algoritma ini berjalan mulai dari pojok kiri atas sebuah *array* dua dimensi yang telah diisi sejumlah karakter *string* awal dan *string* target dan diberikan nilai *cost*. Nilai *cost* pada ujung kanan bawah menjadi nilai *edit distance* yang menggambarkan jumlah perbedaan dua *string* (Benisius, Tanpa Tahun).

Dari proses pencarian nilai *distance*-nya, didapat bahwa jumlah modifikasi minimum yang dilakukan untuk mengubah *string*, jarak' menjadi *string* 'penjara' adalah sebanyak 4 operasi, yaitu: penyisipan karakter 'p', penyisipan karakter 'e', penyisipan karakter 'n', dan penghapusan karakter 'k'.

```
Pseudo code:
int LD(String s, String t) {
m = s.length
n = t.length int[m][n]
d = empty int table for i from 0 to m
d[i, 0] := i for j from 0 to n d[0, j] := j
for j from 1 to n
for i from 1 to m
if s[i] = t[j] then d[i, j] := d[i-1, j-1] else d[i, j] := minimum(d[i-1, j] + 1,d[i, j-1] + 1,d[i-1, j-1] + 1)
return d[m, n]}
```

Gambar 2.5 *Pseudocode* Algoritma Levenshtein *Distance*
(Sumber : http://heuristicswiki.wikispaces.com/Levenshtein+distance+%28Heuristic%29)