BAB II

LANDASAN TEORI

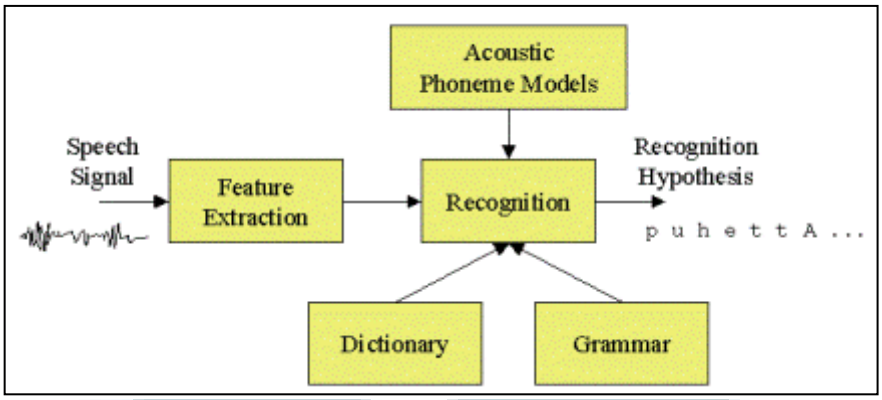
2.1. Aplikasi Pengenal Suara

Aplikasi pengenal suara, atau yang biasa disebut *speech recognizer* bekerja dengan mendigitalisasi dahulu kata ucapan tersebut dan cocokan sinyal digital tersebut dengan pola tertentu yang tersimpan dalam suatu perangkat. Kemudian, kata yang diucapkan tersebut diubah bentuknya menjadi sinyal digital dengan mengubah gelombang suara menjadi sekumpulan angka yang sesuai dengan kode tertentu untuk mengidentifikasi kata-kata tersebut.

Hasil identifikasi kata tersebut lalu ditampilkan dalam bentuk tulisan atau dibaca oleh perangkat lain sebagai perintah untuk suatu pekerjaan, seperti penekanan tombol pada telepon genggam yang dilakukan otomatis dengan komando suara (Andi Sianturi,2014).

Aplikasi jenis ini bekerja dengan memeeriksa terlebih dahulu sampel suara yang didapatkan dari lingkungan sekitar, kemudian harmoni, frekuensi dan nada dari sampel itu dianalisa dahulu sesuai dengan sampel suara yang tersimpan dalam sebuah *server*.

Contoh kegunaan dari aplikasi pengenal suara adalah perintah suara pada perangkat tertentu, dimana perangkat tersebut akan melakukan sebuah aksi tertentu jika pengguna mengucapkan kata-kata tertentu. Selain itu, penggunaan alat pengenal suara juga terdapat pada pesawat terbang untuk mendapatkan keterangan mengenai keadaan lalu-lintas udara dan berkomunikasi dengan pilot atau pihak lain.



Gambar 2.1. Diagram konteks cara kerja *speech recognizer* (sumber: www.lingsoft.fi)

Aplikasi pengenal suara ini memiliki empat tahapan dalam prosesnya (Benesty dkk.,2008), yaitu:

1. Tahap penerimaan masukan

   Masukan yang diterima ini berupa kata-kata yang diucapkan lewat perangkat *input* seperti mikrofon

2. Tahap ekstraksi

   Tahap ini berupa tahap penyimpanan masukan yang berupa suara sekaligus pembuatan basis data sebagai pola.

3. Tahap pembandingan

   Tahap ini merupakan tahap pencocokan data baru dengan data suara (pencocokan tata bahasa) pada pola. Tahap ini dimulai dengan proses konversi sinyal suara digital hasil dari proses ekstraksi ke dalam bentuk spektrum suara yang akan dianalisa dengan membandingkan pola suara sampel masukan dengan pola suara pada basis data.

Langkah berikutnya ialah proses kalkulasi yang dibagi menjadi beberapa bagian :

1. Transformasi gelombang diskrit menjadi data yang terurut
2. Menghitung frekuensi pada tiap elemen data yang terurut
3. Selanjutnya tiap elemen dari data yang terurut tersebut dikonversi ke dalam bentuk bilangan biner. Data biner tersebut akan dibandingkan dengan pola data suara yang tersimpan dan kemudian diterjemahkan sebagai keluaran.
4. Tahap validasi identitas pengguna

   Alat pengenal suara yang sudah memiliki sistem verifikasi akan mengidentifikasi orang yang berbicara sebelum menerjemahkan suara tersebut menjadi perintah.

Untuk mengimplementasikan aplikasi pengenal suara, maka *hardware* yang diperlukan adalah:

- *Sound card*

  Merupakan perangkat yang ditambahkan dalam suatu komputer yang berfungsi sebagai perangkat *input* dan *output* suara untuk mengubah sinyal elektrik menjadi analog maupun digital.
- *Microphone*

  Perangkat *input* suara untuk mengubah suara menjadi sinyal elektrik
- Komputer atau komputer *server*

  Dalam proses suara digital menerjemahkan gelombang suara menjadi suatu simbol, biasanya menjadi nomor biner yang dapat diproses lagi,

dan dicocokkan dengan *database* yang berisi berkas suara agar dapat dikenali.

2.2. Aplikasi Pengenal Lagu

Aplikasi pengenal lagu merupakan produk yang berkembang dari adanya alat pengenal suara, namun perbedaannya adalah aplikasi tersebut berfungsi untuk mengenali lagu ataupun suara jenis lainnya untuk kemudian didapatkan informasi tentang judul dan detail lainnya dari lagu tersebut untuk diteruskan kepada pengguna.

Maka itu tentu saja berbeda dalam hal *output*-nya. Jika pada alat pengenal suara *output*-nya berupa kata yang diolah dari suara pengucap, maka alat pengenal lagu ini mengeluarkan *output* berupa informasi dari suatu rekaman yang dikirimkan.

Contohnya adalah Shazam dan SoundHound. Aplikasi tersebut mempunyai *database* lagu dalam jumlah yang sangat besar. Gambaran cara kerja aplikasi ini adalah (Avery Wang,Tanpa tahun) :

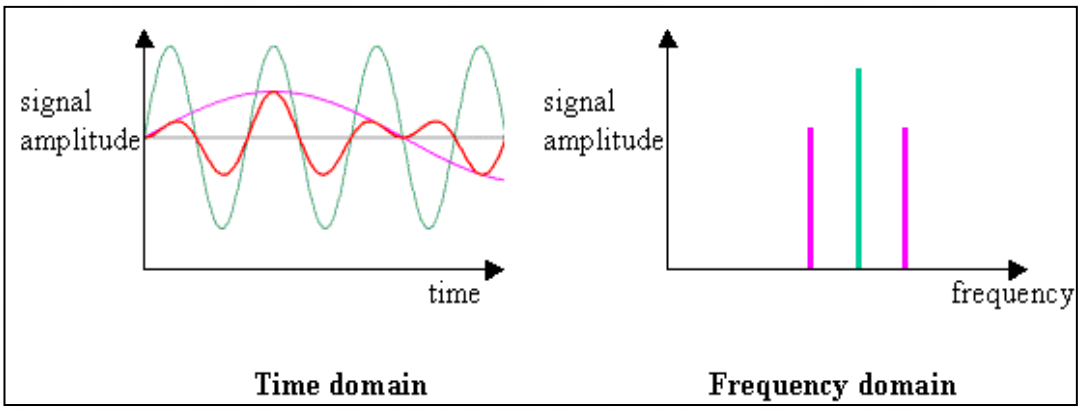
1. Pemakai aplikasi meng-*capture* sebagian nada dari suatu lagu ke aplikasi
2. Aplikasi akan meng-*extract* data dari suatu lagu ke server
3. Data dari suatu lagu dicocokkan dengan *database* lagu yang ada, kemudian di*extract* lagi untuk mengetahui lagu yang dikirim via aplikasi cocok dengan algoritma lagu yang sesuai.

4. Kemudian dalam *database* jika telah diketahui suatu kecocokan lagu maka akan mengirim balik ke pengguna aplikasi, informasi berupa judul lagu, artis, dan gambar sampul album lagu tersebut (jika ada).

2.3. Sinyal dalam Domain Waktu dan Domain Frekuensi

Domain waktu (*time domain*) adalah analisis dari fungsi matematis berupa data *environmental* dalam satuan waktu. Dalam domain waktu, sinyal atau fungsi diketahui sebagai bilangan riil. Sebuah grafik domain waktu menunjukkan perubahan sinyal dengan waktu tertentu (Lee, Y. W,dkk.,1950).

Sedangkan domain frekuensi merujuk pada analisis sinyal berdasarkan frekuensi. Jadi, grafik domainnya menunjukkan bagaimana sinyal tersebut berubah nilainya dalam rentang frekuensi. Representasi domain frekuensi bisa termasuk informasi tentang pergeseran fase yang harus diterapkan pada setiap sinusoid untuk bisa menggabungkan kembali komponen frekuensi tersebut untuk menyajikan domain waktu yang sebenarnya (Broughton, S.A. dan Bryan, K., 2008).



Gambar 2.2. Contoh Representasi domain waktu dan frekuensi (sumber : http://www.ni.com. Tanggal akses 9 Januari 2015)

Sebuah fungsi atau sinyal yang diberikan dapat dikonversi antara domain waktu dan frekuensi dengan sepasang operator matematika yang disebut transformasi. Contohnya adalah transformasi Fourier, yang mengubah fungsi waktu ke dalam sejumlah gelombang sinus frekuensi yang berbeda, yang masing-masing merupakan komponen frekuensi. Spektrum komponen frekuensi adalah representasi sinyal domain frekuensi. Invers transformasi Fourier mengubah fungsi domain frekuensi kembali ke fungsi waktu.

Dalam pengolahan suara, transformasi Fourier banyak digunakan untuk mengubah domain waktu pada suara menjadi domain frekuensi. Analisa-analisa dalam domain frekuensi banyak digunakan seperti *filtering*. Dengan menggunakan transformasi Fourier, sinyal atau suara dapat dilihat sebagai suatu objek dalam domain frekuensi (Saeed V. Vaseghi,2008).

2.4. Pengolahan Sinyal Digital

Pengolahan sinyal digital (*Digital Signal Processing* atau DSP) adalah perubahan atau analisa informasi yang dinyatakan sebagai urutan angka diskrit. Sinyal-sinyal tersebut berasal dari data seperti getaran *seismic*, gambar visual, maupun gelombang suara. Tujuan utama pengolahan sinyal digital adalah untuk mendapatkan atau mengekstrak informasi yang dibawa oleh sinyal seperti meningkatkan kualitas gambar, mengenal dan membentuk *speech*, kompresi data untuk penyimpanan dan lain-lain (Bores,1998).

Keuntungan dari pengolahan sinyal digital adalah (Bores,1998):

1. Multifungsi (*Versatility*)

   Sistem digital dapat diprogram kembali untuk aplikasi lainnya, seperti aplikasi yang bisa menggunakan *programmable DSP* dan dapat dipasang ke perangkat yang berbeda.

2. Bisa Diulang (*Repeatability*)

   Sistem digital dapat digandakan dengan mudah dan tidak bergantung pada komponen lainnya.

3. Kesederhanaan (*Simplicity*)

   Dalam hal tertentu, sinyal digital lebih mudah diproses daripada sinyal analog.

Para teknisi telah meneliti penerapan teknik pengolahan sinyal digital sejak dekade 1940-an, dimana para ilmuwan dari Bell Telephone Laboratories mendiskusikan kemungkinan pemakaian sirkuit digital untuk mengimplementasikan fungsi *filter*. Kemudian pada tahun 1950an, Profesor Linville dari MIT mendiskusikan pengolahan digital pada sebuah seminar.

Teori pengolahan sinyal digital berkembang pesat pada era 1960an saat Kalman mengembangkan *filter* yang praktis untuk menunjukkan optimasi dari suatu *filter*/kontrol. Pada tahun 1965, Cooley dan Tukey membuat sebuah *paper* yang menjelaskan metode pengolahan sinyal bernama *Fast Fourier Transform* (FFT) yang merupakan cara yang efisien untuk menerapkan metode *Discrete Fourier Transform* (DFT) (Dag Stranneby,2004).

Pada jaman sekarang ini, banyak bermuculan berbagai produk yang memanfaatkan keuntungan dari pengolahan sinyal digital ini, terutama dalam produk yang mampu menjamin ketepatan hasil dan biasa digunakan dalam

produksi berskala besar. Contohnya adalah sistem telekomunikasi, identifikasi penyakit, pengiriman sinyal radar pesawat dan sistem kendali pesawat.

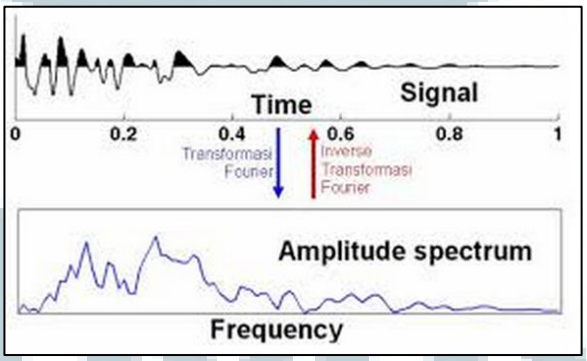
2.5. Discrete Fourier Transform (DFT)

Discrete Fourier Transform (DFT) digunakan dalam pemrosesan sinyal digital untuk mengubah data dari domain waktu menjadi domain frekuensi. DFT didefinisikan dengan rumus sebagai berikut (Martin King,2008):

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k) e^{kw_0 n}$$ Rumus 2.1

Dimana:

- N = banyaknya sampel
- x(n) = sinyal diskrit
- X(k) = koefisien DFT untuk sinyal diskrit x(n)
- k = 0,1,2,3,....,N-1
- w_0= frekuensi digital (sinusoid) ($w_0 = 2\pi/N$)



Gambar 2.3. Diagram Konversi Domain Waktu dan Frekuensi oleh DFT (Margrave.G. et al.,Tanpa Tahun)

Input dari DFT adalah sederet bilangan kompleks, sehingga memungkinkan informasi hasil prosesnya dapat disimpan di komputer (Movellan, 2009).

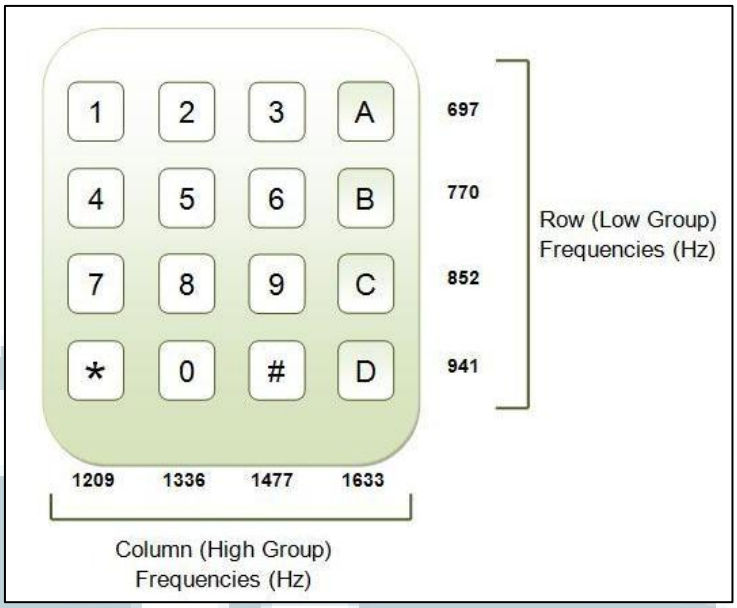
DFT memungkinkan untuk menganalisis, memanipulasi, dan mensintesis sinyal dengan cara yang tidak mungkin dilakukan dalam pemrosesan sinyal analog (Richard Lyons, 2010). Meskipun sekarang digunakan dalam hampir setiap bidang teknik. Aplikasi yang menggunakan DFT terus berkembang sebagai utilitas yang menjadikan DFT lebih mudah untuk dimengerti.

2.6. Algoritma Goertzel

Algoritma Goertzel adalah teknik *Digital Signal Processing* (DSP) yang mengevaluasi istilah *Discrete Fourier Transform* (DFT) secara efisien. Algoritma ini pertama kali dijelaskan oleh Gerald Goertzel pada tahun 1958 lewat *paper*-nya yang berjudul "*An Algorithm for the Evaluation of Finite Trigonometric Series"* (Gerald Goertzel,1958).

Algoritma ini biasanya digunakan pada aplikasi DTMF (*Dual Tone Multi Frequency*) yang dihasilkan oleh tombol *keypad* telepon yang ditekan. Namun DTMF saat ini juga diterapkan pada surat elektronik dan sistem *telephone banking*, dimana para pengguna bisa memilih pilihan dari menu dengan mengirimkan sinyal DTMF dari telepon mereka (Naim Dalmoun,2004).

Seperti DFT, Algoritma Goertzel menganalisa salah satu komponen frekuensi dipilih dari sinyal diskrit. Namun tidak seperti perhitungan DFT langsung, Algoritma Goertzel memberlakukan koefisien bernilai real tunggal pada setiap iterasi, menggunakan aritmatika bernilai *real* untuk urutan masukan bernilai *real*



Gambar 2.4. *Keypad* Telepon dengan DTMF (Nikhil Agnihotri,2012)

Untuk mencakup spektrum penuh, Algoritma Goertzel memiliki kompleksitas yang lebih tinggi dari algoritma Fast Fourier Transform (FFT); tapi lebih efisien secara numerik untuk menghitung sejumlah kecil komponen frekuensi yang dipilih. Jadi, algoritma ini juga memiliki kemiripan dengan FFT, namun berbeda dengan FFT yang menghitung semua nilai DFT pada semua *indices*, Goertzel hanya menghitung DFT pada sekumpulan *indices* tertentu saja seperti hanya pada *range* frekuensi tertentu (Brian L. Evans, tanpa tahun).

Struktur sederhana dari Algoritma Goertzel membuatnya sangat cocok untuk prosesor kecil dan aplikasi yang tertanam, namun tidak sebatas itu saja, melainkan ada keuntungan lainnya yang dimiliki oleh Algoritma Goertzel jika dibandingkan dengan FFT, yaitu (Richard Lyons, 2012):

1. Blok sampel waktu N tidak harus bilangan *integer* pangkat 2 seperti FFT

2. Tidak menyimpan blok *input* data yang diperlukan sebelum *processing* bisa dimulai (seperti pada FFT). *Processing* bisa dimulai dengan sampel *input time* pertama.
3. Jika diimplementasikan M kali untuk memeriksa M nada berbeda, Goertzel lebih efisien daripada FFT ketika jumlah M lebih kecil daripada $\log_2 N$
4. Perbandingan komputasional yang dibutuhkan untuk mendeteksi satu nada tunggal terbilang lebih kecil daripada FFT (Goertzel = N+2 berbanding dengan FFT = $2N\log_2 N$)

2.6.1. Rumus Dasar

Sebelum Algoritma Goertzel aktual bisa dilakukan, maka perlu kita lakukan beberapa penghitungan seperti (Kevin Banks, 2002):

1. Menentukan *sampling rate*.

   Aturan Nyquist menyatakan bahwa *sampling rate* yang ditentukan paling sedikit 2 kali frekuensi tertinggi yang akan dimasukkan. Karena jika mendeteksi beberapa frekuensi, maka mungkin akan menghasilkan hasil yang lebih baik ketika frekuensi *sampling* lebih besar.

2. Memilih ukuran blok N

   Ukuran blok Goertzel (N) sama seperti jumlah poin dalam FFT yang ekuivalen. Ukuran ini mengontrol resolusi frekuensi. Contohnya, jika *sampling rate*-nya adalah 8 kHz dan N berjumlah 100 sampel, maka lebar *bin* adalah 80Hz.

Jika jumlah N makin besar, maka waktu untuk mendeteksi nadanya semakin lama. Contohnya, pada sampling 8 kHz, akan membutuhkan waktu 100ms untuk setiap 800 sampel yang diakumulasi.

Faktor lainnya dalam mempengaruhi pemilihan N adalah hubungan antara *sampling rate* dan frekuensi targetnya. Maka frekuensi target yang diinginkan berupa bilangan bulat dari sample_rate/N. Tetapi N tidak harus berupa pangkat 2 seperti FFT.

3. Hitung *cosine, sine,* dan koefisien

Setelah *sampling rate* dan ukuran *block* ditentukan, ada beberapa rumus untuk menghitung konstanta (k) yang dibutuhkan:

$$k = (0.5 + \frac{N*target\_freq}{sample\_rate}) \quad \text{...Rumus 2.2}$$

$$w = \left(2 * \frac{\pi}{N}\right) * k \quad \text{...Rumus 2.3}$$

$$cosine = \cos w \quad \text{...Rumus 2.4}$$

$$sine = \sin w \quad \text{...Rumus 2.5}$$

$$coeff = 2 * cosine \quad \text{...Rumus 2.6}$$

Kemudian tentukan variabel Q_0, Q_1 dan Q_2 untuk proses per-sampel.

- Q_2 = nilai dari Q_0 2 kali yang sebelumnya
- Q_1 = nilai dari Q_0 sebelumnya
- Q_0 = nilai yang sekarang

Q_1 dan Q_2 harus diinisialisasikan nol pada setiap awal dari *block* sampel. Untuk setiap sampel, jalankan rumus ini:

$$Q_0 = coeff * Q_1 - Q_2 + sample \quad \text{...Rumus 2.7}$$

$$Q_2 = Q_1 \quad \text{...Rumus 2.8}$$

$$Q_1 = Q_0 \quad \text{...Rumus 2.9}$$

Kemudian Q_2 dan Q_1 direset ulang menjadi 0, dan mulai ke *block* berikutnya. Dengan demikian, beberapa rumus di atas dapat digabungkan menjadi rumus dalam 2 tahap berikut ini:

- Tahap 1 = menghitung urutan penengah s(n) yang merupakan nilai Q_0 dalam penjelasan sebelumnya

$$s(n) = 2 * \cos\left(\frac{2 * \pi * target_{freq}}{sample_{rate}}\right) * Q_1 - Q_2 + sample \quad \text{...Rumus 2.10}$$

- Tahap 2 = mendapatkan nilai *output* y(n)

$$y(n) = s(n) - \exp\left(\frac{2 * \pi * target_{freq}}{sample_{rate}}\right) * Q1 \quad \text{...Rumus 2. 11}$$

Dengan definisinya adalah:

- target_freq = nilai frekuensi dasar yang telah ditetapkan
- sample_rate = nilai *sample rate* yang sudah ditetapkan
- π = sebuah konstanta bernilai 3,14
- sample = data audio atau domain waktu yang sedang diproses

2.7. Teori Musik dan Nada

Teori musik merupakan cabang ilmu yang menjelaskan unsur-unsur musik. Cabang ilmu ini mencakup pengembangan dan penerapan metode untuk menganalisis maupun menggubah musik, dan keterkaitan antara notasi musik dan pembawaan musik. Hal inilah yang dibahas dalam teori musik (Boretz,1995):

2.7.1. Suara

Dalam musik, gelombang suara biasanya dibahas tidak dalam panjang gelombangnya maupun periodenya, melainkan dalam frekuensinya. Aspek-aspek dasar suara dalam musik biasanya dijelaskan dalam tala (Inggris: *pitch*, yaitu tinggi nada), durasi (berapa lama suara ada), intensitas, dan *timbre* (warna bunyi).

2.7.2. Nada

Nada adalah bunyi yang beraturan, yaitu memiliki frekuensi tunggal tertentu. Dalam teori musik, setiap nada memiliki tinggi nada atau tala tertentu menurut frekuensinya ataupun menurut jarak relatif tinggi nada tersebut terhadap tinggi nada patokan. Nada dasar suatu karya musik menentukan frekuensi tiap nada dalam karya tersebut. Nada dapat diatur dalam tangga nada yang berbeda.

Perbedaan tala antara dua nada disebut sebagai interval. Nada dalam teori musik diatonis barat diidentifikasikan menjadi 12 nada yang masing-masing diberi nama yaitu nada C, D, E, F, G, A dan B. Serta nada-nada kromatis, yaitu Cis/Des, Dis/Es, Fis/Ges, Gis/As, dan Ais/Bes. Istilah "nada" sering dipertukarkan penggunaannya dengan "not", walaupun kedua istilah tersebut berbeda arti.

Secara internasional, frekuensi nada dasar A4 (merupakan nada A sedang, yaitu nada A pada oktaf keempat piano) adalah 440 Hz, maka itu, cara menghitung frekuensi nada lainnya adalah menggunakan perbandingan berikut.

Tabel 2.1. Tabel Deretan Nada dan Nilai Perbandingan Frekuensinya

C	D	E	F	G	A	B	C’
24	27	30	32	36	40	45	48

Dari tabel tersebut, kita dapat menyimpulkan bahwa perbandingan frekuensi nada-nadanya adalah sebagai berikut:

1. fC : fE = 24 : 30
2. fC : fG = 24 : 36
3. dan seterusnya

Maka contoh soal penghitungan frekuensi nada patokannya adalah dengan menggunakan perkalian silang berikut ini.

- Nada C:

  fC : fA = 24 : 40

  fC : 440 = 24 : 40

  40fC = 24 x 440

  fC = 10560/40

  Jadi, fC = 264 Hz.

Dan dengan menggunakan cara yang sama, maka frekuensi nada patokan lainnya adalah:

- D = 297 Hz
- E = 330 Hz
- F = 352 Hz
- G = 396 Hz
- B = 495 Hz
- C' = 528 Hz

Dan berdasarkan nada patokan tersebut, maka nada lainnya yang berjarak lebih dari 1 oktaf dihitung menggunakan rumus:

$$f_n = f_0 . 2^{n/12}$$ Rumus 2.12

Dengan :

f_n = frekuensi nada yang dicari

f_0 = frekuensi nada patokan

n = jarak nada yang dicari dengan nada patokan (dalam satuan jumlah tuts piano)

Contoh soal dengan nada patokan A4 adalah:

- Nada D4 = $440.2^{-7/12}$ = 293.66 Hz
- Nada C5 = $440.2^{3/12}$ = 523.25 Hz

Apabila dihitung satu per satu dengan rumus di atas, akan didapatkan nilai frekuensi sebagai berikut:

Tabel 2.2. Hasil Penghitungan Frekuensi

Frekuensi dalam Hertz (tanda titik menunjukkan desimal)								
Octave / *Note*	1	2	3	4	5	6	7	8
C	32.703	65.406	130.81	261.63	523.25	1046.5	2093.0	4186.0
C#/Db	34.648	69.296	138.59	277.18	554.37	1108.7	2217.5	4434.9
D	36.708	73.416	146.83	293.66	587.33	1174.7	2349.3	4698.6
D#/Eb	38.891	77.782	155.56	311.13	622.25	1244.5	2489.0	4978.0
E	41.203	82.407	164.81	329.63	659.26	1318.5	2637.0	5274.0
F	43.654	87.307	174.61	349.23	698.46	1396.9	2793.8	5587.7
F#/Gb	46.249	92.499	185.00	369.99	739.99	1480.0	2960.0	5919.9
G	48.999	97.999	196.00	392.00	783.99	1568.0	3136.0	6271.9
Ab/G#	51.913	103.83	207.65	415.30	830.61	1661.2	3322.4	6644.9
A	55.000	110.00	220.00	440.00	880.00	1760.0	3520.0	7040.0

Lanjutan Tabel 2.2. Hasil Penghitungan Frekuensi

Frekuensi dalam Hertz (tanda titik menunjukkan desimal)								
Octave / *Note*	1	2	3	4	5	6	7	8
Bb/A#	58.270	116.54	233.08	466.16	932.33	1864.7	3729.3	7458.6
B	61.735	123.47	246.94	493.88	987.77	1975.5	3951.1	7902.1

Sumber = Garage Lab (http://garagelab.com/)

2.7.3. Ritme

Ritme adalah pengaturan bunyi dalam waktu. Birama merupakan pembagian kelompok ketukan dalam waktu. Tanda birama menunjukkan jumlah ketukan dalam birama dan not mana yang dihitung dan dianggap sebagai satu ketukan. Nada-nada tertentu dapat diaksentuasi dengan pemberian tekanan (dan pembedaan durasi).

2.7.4. Notasi

Notasi musik merupakan penggambaran tertulis atas musik. Dalam notasi balok, tinggi nada digambarkan secara vertikal sedangkan waktu (ritme) digambarkan secara *horizontal*. Kedua unsur tersebut membentuk paranada, di samping petunjuk-petunjuk nada dasar, tempo, dinamika, dan sebagainya.

2.7.5. Melodi

Melodi adalah serangkaian nada dalam waktu. Rangkaian tersebut dapat dibunyikan sendirian, yaitu tanpa iringan, atau dapat merupakan bagian dari

rangkaian akord dalam waktu (biasanya merupakan rangkaian nada tertinggi dalam akord-akord tersebut).

2.7.6. Harmoni

Harmoni secara umum dapat dikatakan sebagai kejadian dua atau lebih nada dengan tinggi berbeda dibunyikan bersamaan, walaupun harmoni juga dapat terjadi bila nada-nada tersebut dibunyikan berurutan (seperti dalam arpeggio). Harmoni yang terdiri dari tiga atau lebih nada yang dibunyikan bersamaan biasanya disebut akord.

2.8. Hashing

Hashing adalah transformasi aritmatik sebuah *string* dari karakter menjadi nilai yang merepresentasikan *string* aslinya. *Hashing* digunakan sebagai metode untuk menyimpan data dalam sebuah *array* agar penyimpanan data, pencarian data, penambahan data dan penghapusan data dapat dilakukan dengan cepat. Ide dasarnya adalah menghitung posisi *record* yang dicari dalam *array*, bukan membandingkan *record* dengan isi pada *array* (Puthut Prabancono, 2008).

Selain digunakan pada penyimpanan data, fungsi *hash* juga digunakan dalam bidang keamanan, seperti pada algoritma enkripsi sidik jari digital (*fingerprint*) untuk mengautentifikasi pengirim dan penerima pesan. Sidik jari digital diperoleh dengan fungsi *hash*, kemudian nilai *hash* dan tanda pesan yang asli dikirim kepada penerima pesan. Dengan menggunakan fungsi *hash* yang sama dengan pengirim pesan, penerima pesan mentransformasikan pesan yang diterima.

Nilai *hash* yang diperoleh oleh penerima pesan kemudian dibandingkan dengan nilai *hash* yang dikirim pengirim pesan (Puthut Prabancono, 2008).

Beberapa istilah penting dalam *hashing* adalah:

1. *Hash Table*, yaitu *array* tempat menyimpan data untuk *hashing*. Ukuran *hash table* lebih besar daripada jumlah data yang akan ditampung.
2. *Hash Function*, yaitu fungsi atau rumus untuk mengubah nilai kunci menjadi posisi indeks pada *hash table*. *Hash function* diperlukan karena nilai kunci tidak selalu berupa bilangan urut yang langsung dapat dijadikan *hash table*. Syarat *hash function* yang baik adalah sederhana dan terdistribusi merata. Contoh teknik *hash function* yang biasa digunakan adalah:
    - *Digit Selection*

      Pilih beberapa digit dari nilai kunci. Contohnya kata kunci (NIM) terdiri dari 8 digit berupa d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8. Pilih digit d_6 d_7 d_8 agar memperoleh nilai antara 0 dan 999.
    - *Folding*

      Digit pembentuk kunci dibagi jadi beberapa kelompok. Nilai kelompok tersebut dijumlahkan. Misalnya kata kunci yang terdiri atas 8 digit dibagi menjadi 3 bagian yang sama panjang kecuali bagian yang terakhir: 3 digit, 3 digit, 2 digit. Setelah itu, jumlahkan semua bagian untuk mendapatkan *hash address*. Dengan demikian, nilai maksimumnya adalah 999 + 999 + 99 = 2097.

- *Modular arithmetic*

  Nilai kunci dibagi (modulus) bilangan tertentu. “Bilangan tertentu” ini adalah ukuran *hash table* yang umumnya adalah bilangan prima. Contohnya adalah jika ukuran *hash table* adalah 23, maka data akan diletakkan pada posisi ke - “kode % 23”.

- *Multiplication*

  Yaitu nilai kunci dikalikan dengan nilai kunci, kemudian beberapa digit pun dipilih. Digit yang dipilih biasanya digit tengah. Untuk kunci yang terdiri dari 5 digit, menghitung *hash table*-nya adalah 99999 x 99999 = 9999800001. Kemudian, jika dipilih digit 4,5 dan 6, maka *hash function*-nya akan menghasilkan angka dari 0 sampai 980. Contoh:

  - Nilai kunci = 12345, maka dilakukanlah penghitungan *hash function* dengan cara mengalikan 12345 dengan bilangan itu sendiri. Jadi, 12345 x 12345 = 152399025. Dari hasil tersebut, data akan disimpan pada *hash table* dengan indeks 399

3. *Collision*, yaitu keadaan dimana beberapa nilai fungsi yang berbeda memiliki nilai *hash* yang sama.

2.9. MD5 Hash

MD5 (*Message-Digest Algorithm* 5) merupakan fungsi *hash* yang digunakan secara luas dalam kriptografi dan menghasilkan nilai hash 128 bit. MD5 biasanya digunakan dalam aplikasi kriptografik, seperti pengamanan kode

rahasia. Fungsi ini dibuat oleh Ron Rivest pada tahun 1991 untuk menggantikan fungsi *hash* MD4 (Mark Ciampa, 2009).

Dalam fungsi ini, pesan sebanyak b bit menjadi input. Nilai b ini bisa berupa bilangan bulat non negatif atau nol. Bit dari pesan ini dapat dituliskan sebagai:

m_0 m_1 ..... m_{b-1}

Kemudian ada 4 langkah yang dilakukan untuk menghitung *message digest* dari pesan tersebut (R. Rivest, 1992) :

1. Tambahkan (*append*) *padding bits* dari pesan.

   Pesan di-*pad* sampai panjang pesan tersebut kongruen dengan 448 modulo 512. Maka, pesan ditambah panjangnya sampai terdiri dari 64 bit sesuai dengan hasil dari modulo tersebut. *Padding* terus dilakukan meskipun panjang pesan sudah kongruen dengan 448 modulo 512.

   Padding dilakukan dengan menambah nilai bit "1" pada pesan, kemudian nilai bit "0" ditambahkan sehingga panjang bit dari pesan yang di-*pad* tersebut kongruen dengan 448 modulo 512.

2. Tambahkan panjang (*length*) dari pesan.

   Representasi 64-bit dari b (panjang pesan sebelum ditambahkan *padding bits*) ditambahkan pada hasil dari langkah sebelumnya. Kemudian hanya dipakai *low-order* (digit yang mengkontribusikan jumlah terkecil dari nilai numeral atau posisi paling kanan dari suatu kata) dari 64 bit dari b yang digunakan.

   Hasil ini memiliki panjang yang tepat 512 bit. Secara ekuivalen, pesan ini memiliki panjang yang merupakan kelipatan 16. Pesan yang

merupakan hasil dinotasikan sebagai M[0 ... N-1], dimana N merupakan kelipatan 16.

3. Inisialisasi MD *buffer*

Dalam proses ini, ada *buffer* 4 karakter (A, B, C, D) yang dipakai untuk menghitung *message digest*. Setiap A, B, C dan D adalah *register* 32 bit. Register tersebut diinisialisasi dalam nilai hexadesimal mulai dari *low-order byte*. Contoh:

Word A : 01 23 45 67

Word B : 89 ab cd ef

Word C : fe dc ba 98

Word D : 76 54 32 10

4. Proses pesan dalam blok 16 kata

Mula-mula definisikan 4 fungsi yang digunakan sebagai *input* dari 3 kata 32-bit dan menghasilkan *output* berupa kata berukuran 32-bit. Langkah ini menggunakan tabel 64-elemen T[1 ... 64] yang dibuat dari fungsi sine. T[i] adalah elemen ke-i dari tabel, yang sama dengan bagian integer dari 4294967296 kali abs(sin(i)), dimana i dalam satuan radian.

Output dari pesan ini diberikan dalam bentuk A, B, C, D. Artinya, A merupakan byte *low-order*, dan diakhiri dengan byte *high-order* dari D.

2.10. Algoritma Pencarian String

Algoritma pencarian *string* adalah algoritma yang digunakan untuk melakukan pencarian semua string pendek yang disebut *pattern* pada *string* yang lebih panjang yang disebut teks (Lecroq dkk., 2001).

Contoh algoritma pencarian *string* yang paling sering digunakan adalah Algoritma *Brute Force*, yaitu algoritma yang ditulis tanpa memikirkan kerja atau performa sistem. Cara kerjanya adalah sebagai berikut (Reno Rasyad,2013):

1. Algoritma mulai mencocokkan *pattern* pada awal teks
2. Dari kiri ke kanan, algoritma akan mencocokkan karakter di *pattern* dengan karakter pada teks yang bersesuaian sampai salah satu kondisi dibawah ini terpenuhi:

1. Karakter di *pattern* dan di teks yang dibandingkan tidak cocok
2. Semua karakter di *pattern* cocok. Kemudian algoritma akan memberitahukan posisi ini.

3. Setelah salah satu kondisi diatas terpenuhi, pattern akan digeser ke kanan dan mengulangi langkah 2, serta berhenti saat mencapai ujung teks.

```
procedure BruteForceSearch(
        input m, n : integer,
        input P : array[0..n-1] of char,
        input T : array[0..m-1] of char,
        output ketemu : array[0..m-1] of
boolean
       )

Deklarasi:
       i, j: integer

Algoritma:
        for (i:=0 to  m-n) do
               j:=0
               while (j < n and T[i+j] = P[j])
do
                        j:=j+1
               endwhile
               if(j >= n) then
                         ketemu[i]:=true;
```

Gambar 2.5. *Pseudocode* Algoritma *Brute Force*