# *Music-Finder* Menggunakan Algoritma KMP *Extension*

Ismail Sunni - 13508064
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*if18064@students.itb.ac.id*, *ismailsunni@yahoo.co.id*

***Perkembangan dunia teknologi sekarang ini sangat pesat. Banyak sekali aplikasi yang telah diciptakan untuk mempermudah manusia. Salah satu contoh yang paling bagus adalah keberadaan Google. Dengan kekuatan pencariannya, banyak masalah bisa diselesaikan dengan lebih cepat.. Perkembangan juga terjadi pada dunia hiburan, industri musik misalnya. Jutaan atau bahkan miliaran lagu telah tercipta. Tidak ada manusia yang mampu menghapal semua lagu tersebut, bahkan hanya untuk mengetahui judul lagunya.***

***Music-finder merupakan aplikasi yang bisa dijadikan solusi untuk permasalahan ini. Music-finder bisa dikatakan sebuah "Google" dengan inputan berupa file audio yang bisa berasal dari suara speaker di suatu toko buku atau bahkan gumaman dari mulut kita. Hasil yang dikeluarkan oleh music-finder adalah berupa informasi mengenai musik yang kita jadikan inputan.***

***Penggunaan algoritma pencocokan string sangat berperan besar di sini. Algoritma yang dipilih di music-finder adalah KMP-Extension. Suatu algoritma yang memiliki kelebihan dalam pemrosesan data yang besar, seperti data base musik.***

***Terakhir, aplikasi music-finder berpotensi untuk memiliki suatu nilai ekonomi yang cukup tinggi. Karena bisa dijadikan sebagai tempat menjual musik dan mempromosikan lagu itu sendiri.***

***Kata Kunci : Music-Finder, KMP Extension, Audio Encoding, Pencarian, Musik.***

## I. PENDAHULUAN

Pada jaman sekarang, musik merupakan suatu hal yang sudah lumrah. Sudah banyak grups musik, penyanyi, maupun musisi yang menelurkan banyak karyanya.

Musik tidak hanya dapat didengar melalui radio atau *walkman* seperti jaman dahulu. Sekarang, hampir semua orang mempunya *music-player* mereka sendiri. Bahkan, hampir di setiap area umum, misal mall, toko, rumah makan pasti ada musik yang diperdengarkan oleh pemiliknya untuk membuat pengunjung betah dan merasa nyaman di tempat tersebut.

Kadang kala, kita tanpa sengaja mendengar lagu yang menurut kita sangat bagus Namun, karena "hanya" mendengar tanpa sengaja, kita tidak tahu lagu tersebut lagu siapa dan judul lagunya apa. Mungkin, jika kita bisa mendengar lirik lagu tersebut, kita bisa mencarinya di internet. Namun, bagaimana jika lagu yang kita dengar tersebut tidak ada liriknya ? Atau mungkin, kita tidak sempat mendengar si penyanyi, hanya sampai intro atau sudah di akhir lagu ? Seandainya kita sempat bertanya dan kebetulan ada orang yang tahu, mungkin bisa selesai permasalahan. Bagaimana jika tidak ? Ingin menyanyikan ulang dan meminta orang lain mengenali itu lagu siapa? Yang jelas, tidak semua orang bisa menyanyikan ulang sebuah lagu dan tidak semua orang tahu semua lagu.

Pencarian dengan menggunakan *Google* atau sejenisnya tidak lah efektif. Selain *Google* tidak bisa menerima inputan audio tentunya.

Selanjutnya, muncullah apa yang bisa disebut, ***MUSIC-FINDER***. *Music-finder* ini mungkin bisa dianalogikan *Google* untuk lagu. Dengan memberikan inputan dalam bentuk audio, *music-finder* akan mencarikan lagu yang mungkin kita maksud.

Pada makalah ini, akan dijelaskan secara sederhana bagaiamana melakukan pencarian suatu lagu hanya berbekall pada potongan suara yang terdengar dengan menggunakan algoritma pencocokan string, Knuth-Morris-Prat.

Kesulitan yang dihadapi adalah, bagaimana memanajemen inputan. Seperti yang kita ketahui, inputan yang diterima pasti mengandung banyak *noise*. Di sini, ada *trade off* . Jika terlalu menganggap inputan benar, maka dikhawatirkan tidak ada file di data kita yang sesuai. Namun, jika tidak terlalu percaya pada inputan, kita bisa menemukan banyak sekali hasil. Sehingga kurang membantu.

## II. ALGORITMA KNUTH-MORRIS PRAT *EXTENSION*

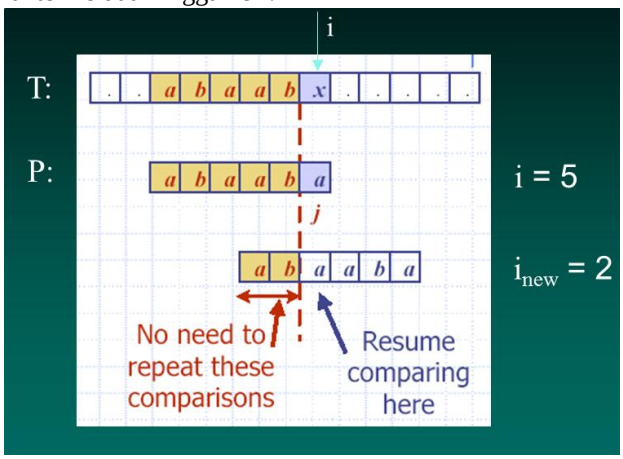
### *A. Algoritma KMP*

Salah satu algoritma yang terkenal dalam pencocokan string adalah Algoritma Knuth-Morris Prat atau selanjutnya kita sebut sebagai **KMP**. Pencocokan string sendiri adalah suatu permasalahan untuk menentukan keberadaan suatu string yang diberikan dengan panjang *m* karakter, yang disebut *pattern* dalam suatu string yang lebih panjang, misal *n* karakter, yang disebut *teks*

KMP sendiri merupakan salah satu teknik pencocokan string dengan menukarkan ruang atau *space* dan waktu atau *time*. Hal ini biasa disebut ***Space and Time Tradeoffs***. Pada intinya, algoritma KMP ini akan melakukan *preprocessing* terhadap suatu hal, dan menyimpannya. Selanjutnya, hasil yang disimpan ini akan dipergunakan untuk mempercepat penyelesaiann.

KMP bisa juga dikatakan sebagai hasil modifikasi dari algoritma *brute-force* untuk pencocokan string. Namun

terdapat perbedaan, jika *brute-force* menemukan ketidakcocokan di suatu karakter, maka algoritma *brute-force* akan melakukan pencocokan dengan terlebih dulu menggeser *pattern* sebanyak 1 karakter. Untuk KMP, pergeseran serupa dilakukan dengan lebih cerdik, yakn idengan menggeser sebesar suatu nilai *k*, dimana *k* merupakan panjang maksimum *prefix* dari *pattern* hingga karakter ke-*i* dengan *suffix* dari *pattern* mulai dari karakter ke dua hingga ke-*i*.




**Gambar 1 Pergeseran KMP**

Nah, disinilah *Space and Time Tradeoff* ini dilakukan. Yakni dengan menyimpan nilai dari *k* untuk setiap *i*. Jadi, sebelum melakukan pencocokan seperti halnya algoritma *brute-force*, KMP akan melakukan suatu mekanisme untuk mendapatkan hasil yang diinginkan. Mekanisme ini dilakukan dengan menjalankan *failure function* dengan inputan

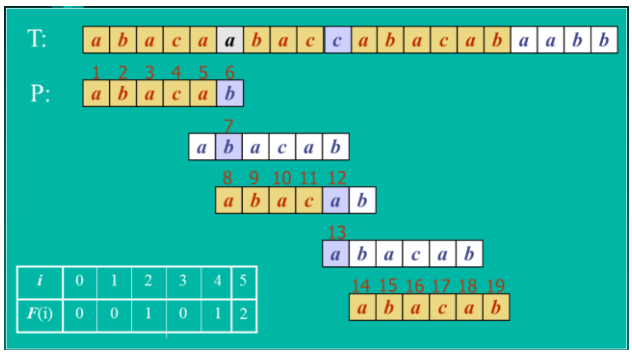
Untuk lebih jelas mengenai *failure function* ini, perhatikan contoh berikut:

*Pattern* : "abaaba"
i : 0, 1, 2, 3, 4, 5
f(i) : *prefix* terpanjang yang sama dengan *suffix* untuk *pattern*[1,i]

sehingga diperoleh nilai untuk masing-masing i adalah sebagai berikut :

| *i* | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| *f(i)* | 0 | 1 | 1 | 2 | 1 |
| *prefix* | - | a | a | ab | a |

Selanjutnya, setelah menyimpan hasil yang diperoleh dari *failure function*, KMP akan melakukan pencocokan mulai dari karakter pertama dari 1 hingga karakter ke-(*n-m*).
Jika ditemukan suatu karakter yang tidak sesuai, maka akan dicek posisi karakter pada *pattern* yang tidak sesuai, misal saja *i*, kemudian mengecek nilai *failure function* untuk posisi *i*, yakni *f(i)*. Kemudian, KMP akan melakukan penggeseran sejauh *n – f(i)*. Langkah ini dilakukan seterusnya hingga *pattern* ditemukan atau sampai pencocokan mencapai ujung *teks*. Perhatikan ilustrasi berikut ini untuk lebih jelasnya :

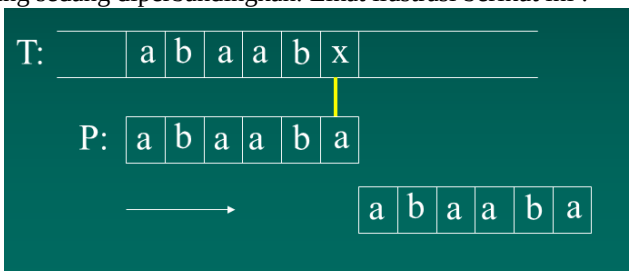



**Gambar 2 Ilustrasi KMP**

Pada saat langkah ke enam, terjadi ketidakcocokan, sehingga, algoritma KMP akan mengecek, pada posisi ini, ini, nilai *f(i)* adalah 2, sehingga, pergeseran dilakukan sebesar *n-2*. Dan begitu seterusnya hingga *pattern* ditemukan.

## B. *Algoritma KMP Extension*

Seperti namanya, algoritma *KMP Extension* merupakan perbaikan atau perluasan dari algoritma KMP. Perbaikan yang terjadi adalah dengan melakukan penampungan karakter yang ada di *pattern* tersebut ke dalam suatu memori. Selanjutnya, saat terjadi ketidakcocokan pada suatu tahap pencocokan karakter, akan dicek terlebih dahulu apakah karakter yang pada posisi tersebut ada di *pattern*. Jika ada, maka akan dilakukan pergeseran seperti biasa. Jika tidak ada, maka akan dilakukan pergeseran sebesar *i*, dimana *i* adalah posisi karakter pada *pattern* yang sedang diperbandingkan. Lihat ilustrasi berikut ini :




**Gambar 3 KMP *Extension***

Karena tidak ada karakter 'x' di dalam *pattern*, maka pergeseran dilakukan sesuai dengan posisi karakter yang sedang dibandingkan, dalam hal ini 'a'. Hal ini akan meningkatkan kualitas algoritma ini, karena tidak perlu melakukan perbandingan dengan karakter yang sudah pasti tidak ada di *pattern*.

## C. *Kelebihan KMP Extension*

Ada beberapa kelebihan algoritma KMP ini. Salah satunya adalah tidak diperlukannya langkah mundur atau pengecekan ke karakter sebelumnya. Hal ini sangat menguntungkan ketika melakukan pemrosesan data yang besar yang dibaca dari file eksternal atau file yang berada di arus suatu jaringan.

Kelebihan lain adalah, ketika tidak banyak alfabet yang digunakan. Misal dalam file biner yang hanya berisi 0 dan 1. Sehingga kemungkinan terjadi ketidakcocokan kecil.

## III. *AUDIO ENCODING*

Suara yang biasa kita dengar selama ini merupakan sebuah gelombang. Gelombang suara ini adalah gelombang akustik satu dimensi atau *one-dimensional acoustic wave*. Ketika gelombang akustik ini masuk ke dalam telinga kita, gendang telinga akan bergetar dan akan menggetarkan tulang-tulang kecil di bagian telinga yang lebih dalam untuk bergetar. Selanjutnya, akan mengirimkan suatu pulsa yang diteruskan ke otak kita. Pulsa inilah yang selama ini kita sebut sebagai suara oleh pendengar.

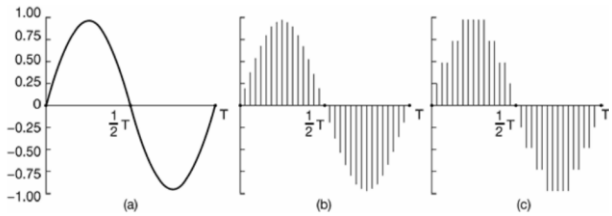
Dengan cara yang sama, gelombang akustik akan "menabrak" *microphone* sebagai alat untuk menerima gelombang suara. *Microphone* ini selanjutnya akan menggenerate suatu sinyal listrik yang merepresentasikan amplitudo suara sebagai fungsi dari waktu.

Jangkauan frekuensi suara yang mampu didengar oleh manusia berkisar antara 20 Hz hingga 20.000Hz. Namun, beberapa binatang, anjing atau kelelawar mampu mendengar bungi dengan frekuensi di luar jangkauan manusia. Karena telinga kita mendengar secara logaritmis, maka perbandingan 2 buah suara dengan amplitudo A dan B biasa dilambangkan dengan **dB** atau **desibels,** sesuai dengan formula sebagai berikut :

$$dB = 20\ log_{10} A/B$$

Jika kita mendefinisikan ambang bawah pendengaran kita untuk frekuensi 1kHz dengan tekanan udara sekitar 0.0003 dyne/cm$^2$ sebagai 0dB, maka percakapan yang biasa kita dengar memiliki nilai sebesar 50dB dan ambang batas pendengaran normal adalah sekitar 120dB.

Gelombang bunyi dapat dikonversi ke dalam bentuk digital dengan menggunakan **ADC** atau **Analog Digital Converter**. Sebuah ADC akan menerima input dalam bentuk tegangan listrik dan akan menhasilkan suatu bilangan biner atau *binary number* sebagai output.




**Gambar 4 Tahap Audio *Encoding***

Secara umum, ada tiga langkah untuk menggenerate sebuah gelombang analog ke dalam bentuk digital.

1. Menerima gelombang
   Misalkan saja gelombang yang diterima seperti pada gambar (4.a)
2. Melakukan *sampling*
   Pada tahap ini, akan dilakukan suatu digitalisasi gelombang analog. Hal ini dilakukan dengan cara mengambil suatu selang waktu, misal $\Delta t$. Selanjutnya, setiap $\Delta t$, akan dilakukan *sampling* atau pengambilan nilai. Yang selanjutnya akan merubah nilai analog menjadi digital.
3. Melakukan *quantization*
   *Quantization* adalah metode untuk meotong-motong dan mengelompokan nilai, dalam contoh ini dari rentang -1 hingga 1, ke dalam beberapa jenis nilai. Pada gambar (4.c), kuantisasi dilakukan setiap 0.25, dan akan memperoleh 8 buah nilai berbeda. Oleh karenanya, dibutuhkan 3 bits memori. Kita juga bisa melakukan kuantisasi dengan lebih detail, misal dengan menggunakan 8 bit yang akan menghasilkan nilai berbeda sebanyak $2^8 = 256$ buah atau 16 bit untuk memperoleh $2^{16} = 65536$ nilai berbeda. Semakin besar bit yang digunakan, maka telinga kita makin sulit mende*teks*i perbedaan analog dan digital. Namun, hal ini juga makin besar pula memory yang digunakan.

Kebanyakan file lagu atau sejenisnya, mengambil sampel sebanyak 44.100 sampel per detik dengan 16 bit untuk melakukan kuantisasi. Sehingga, memerlukan *bandwidth* sebesar $44100 \times 16 = 705600 bit/sec$ atau sebesat 705.5 Kbps untuk mono dan 1.411Mbps untuk stereo.

Karena cukup besar, maka biasanya file musik seperti ini disimpan dalam bentuk kompresinya. Kompresi yang banyak digunakan adalah MPEG layer 3 atau MP3 dengan kompresi hingga 10 kali. Bentuk file MP3 inilah yang biasanya digunakan untuk menyimpan file-file musik kesayangan kita.

## IV. *MUSIC-FINDER*

***Music-finder*** adalah sebuah aplikasi yang dapat mengenali suatu lagu tertentu hanya dengan menerima inputan sebuah inputan potongan lagunya. Ada juga yang menyebutnya *music-recognition*, *track-recognition*, atau lainnya. Namun, pada intinya sama. Mencari informasi lengkap suatu musik dengan hanya menerima inputan berupa potongan musik tersebut.

Cara kerja *music-finder* cukup sederhana. Yakni dengan melakukan searching di data base yang dimiliki. Data base ini berisi kumpulan file musik yang bisa mencapai jutaan file. Berikut langkah-langkah yang dilakukan *music-finder* mulai dari menerima inputan hingga mengeluarkan hasil pencarian.

1. Menerima inputan
   Langkah pertama yang dilakukan *music-finder* adalah menerima inputan dari pengguna. Inputan ini bisa berupa rekaman suara, nyanyian langsung sesorang, atau bahkan hanya dengan menggumamkan nada lagu yang diketahui. Semakin panjang inputan, maka hasil yang diperoleh akan semakin sesuai.
2. Melakukan *encode* inputan
   Inputan yang telah diterima kemudian dilakukan *encoding*. Metode yang dilakukan untuk *encode* sama seperti yang telah disebutkan di pasal 2. Namun, yang patut dicermati adalah mengenai penentuan berapa besar bit yang akan diambil. Makin besar bit yang diambil, tentunya makin presisi. Namun, dengan tidak mengabaikan *noise* yang ada ketika input diterima, penggunaan bit yang lebih rendah, mungkin 8 bit, bisa diterima.

Penentuan banyak sampling juga berpengaruh terhadap hasil. Namun, lebih baik menggunakan sampling standar musik audio, yakni 44.100 sampling per detik.

3. Mengubah ke bentuk biner
   Hasil yang telah diperoleh diubah ke dalam bentuk biner untuk memudahkan pencarian. Namun, ada kalanya tidak harus dalam bentuk biner. Mungkin bisa juga dalam bentuk octadesimal atau bahkan hexadesimal. Hal ini dimaksudkan untuk memperpendek panjang file. Sehingga diharapkan akan mempercepat pencarian.
4. Melakukan *searching* di data base
   Setelah diperoleh inputan dalam bentuk biner atau bentuk lainnya, langkah selanjutnya adalah melakukan pencarian. Pencarian ini menggunakan teknik pencocokan string. Pemabahasan lebih lanjut akan disampaikan pada bagian selanjutnya. Data base yang dipergunakan bisa disimpan di server. Sehingga, aplikasi *music-finder* memerlukan koneksi internet. Karen tidak mungkin menyimpan jutaan file lagu di perangkat elektronik kita, HP atau MP3 player misalnya.
5. Melakukan pengurutan hasil
   Ketika melakukan pencarian, hasil yang didapatkan tidaklah selalu pasti. Bisa saja hanya separuh dari potongan inputan yang ditemukan di dalam data base. Menanggapi hal ini, maka untuk tiap file, saat pencarian selesai dilakukan, diberi nilai, sesuai dengan hasil yang ditemukan. Dengan nilai inilah, maka *music-finder* dapat memberikan hasil yang terurut kepada pengguna. Hasil yang ditampilkan dalam langkah ini bisa nama artis, judul lagu, pencipta lagu, dan lain sebagainya. Bisa juga memunculkan pilihan untuk "membeli" lagu tersebut. Namun yang pasti, sebaiknya menampilan potongan lagu yang "dicurigai" sama dengan inputan. Hal ini tentu akan mempermudah pengguna dalam menentukan apakah lagu tersebut sesuai atau tidak.

## V. KMP *EXTENSION* PADA *MUSIC-FINDER*

Dari lima langkah yang diperlukan oleh suatu aplikasi *music-finder* untuk menemukan potongan suatu lagu, langkah ke empat, yakni melakukan searching di data base merupakan langkah yang vital. Untuk lebih lengkapnya, berikut mekanisme pencarian di *music-finder*.

1. Inputan yang telah diubah menjadi bentuk biner, bisa dipandang sebagai sebuah *pattern*.
2. File yang ada di dalam data base. Data base ini, seperti yang telah disebutkan disimpan di suatu lokasi. Sehingga memerlukan koneksi internet.
3. File di dalam data base tersebut, bisa dipandang sebagai suatu *teks.*
4. Langkah terakhir adalah dengan menjalankan algoritma KMP *Extension* untuk melakukan pencarian terhadap suatu file.

Pencarian tidak serta merta harus menghasilkan suatu kondisi di mana satu *pattern* tersebut ada di suatu lagu. Mengingat *noise* yang diterima cukup besar, maka tidak ada salahnya untuk memeberikan ambang toleransi atas kecocokan karakter. Toleransi ini bisa diimplementasikan dengan beberapa metode.

1. Penggunaan bit yang lebih rendah.
   Metode pertama adalah dengan penggunaan bit yang tidak terlalu besar saat melakukan *encoding* inputan. Jika biasanya file audio menggunakan *encoding* 16 bit, aplikasi ini bisa menggunakan 8 bit. Penggunaan *encoding* 8 bit ini tidak hany berlaku di file inputan. File yang ada di dalam data base pun sebaiknya menggunakan *encoding* yang sama.
2. Penggunaan nilai toleransi
   Metode ke dua adalah dengan mendefinisikan ulang "cocok". Pada umumnya, "cocok " bernilai benar ketika "a = b". Namun, kita bisa mendefinisikan ulang "cocok" ini. Misalkan, "cocok" bernilai benar ketika
   $$|a| \leq |b - \Delta x|$$
   Dimana, $\Delta x$ merupakan suatu nilai toleransi. Jika perlu, nilai toleransi ini bisa kita pilih, misalkan dengan menggunakan masukan dari pengguna. Pengguna bisa memilih besar nilai toleransi ini berdasarkan kondisi saat inputan dimasukan. Untuk memudahkan pengguna, nilai toleransi ini bisa diambil dalam bentuk persentase terhadap nilai a. Dimana nilai toleransi 100% sangat tegas terhadap pencocokan.
3. Pendekatan jumlah.
   Cara lain yang bisa dipergunakan adalah dengan mendekati penjumlahan nilai suatu potongan yang diduga cocok. Misalkan, kita hanya melakukan pencocokan terhadap separuh *pattern*. Selanjutnya, untuk melakukan pengecekan apakah setelah kita menemukan posisi kesamaan, kita akan menghitung nilai dari potongan tersebut, mulai dari ditemukan potongan yang sesuai hingga sepajang *pattern*. Jika nilai yang dihasilkan tidak jauh berbeda dengan hasil penjumlahan yang dimiliki oleh *pattern*, maka ada kemungkinan potongan tersebut merupakan hasil yang diharapkan. Dengan metode ini, kita bisa langsung membuat pengurutan sesuai dengan hasil penjumlahan yang diperoleh.

Penggunaan algoritma KMP Exstension ini sudah tepat. Karena, secara umum, KMP bagus dalam kondisi karakter yang tidak terlalu bervariasi. Misalkan string yang hanya berisi 0 dan 1. Selain itu, kelebihan KMP *Extension* dalam menangani file yang mengalir, bisa mempermudah kita untuk menangani file yang besar tanpa harus menyimpan posisi karakter. Karena, file yang ada di data base pasti sangat besar. Jika 1 file berukuran 5 MegaByte, maka 2.000.000 file akan mempunyai ukuran 10.000.000

MegaByte, atau 10 TeraByte.

Namun, tidak ada yang sempurna di dunia ini. Hal yang paling penting dalam pencarian lagu di sini adalah bagaimana menyikapi inputan. Apakah kita akan menganggap inputan benar, dan beresiko tidak menemukan file, atau menganggap file terlalu banyak noise, sehingga beresiko menghasilkan file yang terlalu banyak.

## VI. IMPLEMENTASI *MUSIC-FINDER*

*Music-finder* cocok untuk dibuat dalam bentuk aplikasi perangkat mobile. Misalkan saja dalam aplikasi untuk smart-phone, PDA, atau bahkan HP kualitas middle-end. Hal ini disebabkan, kebutuhan akan hasil yang cepat dan instan.

Implementasi di perangkat mobile cukup dengan membuat *audio-encoder*. Selanjutnya, melakukan pemilihan nilai toleransi. Hasil *encoding* selanjutnya akan dikirim ke server, dan di server lah akan dilakukan pencarian file musik.

Akhirnya, setelah file musik yang dimaksud ditemukan, server akan mengirim hasilnya ke perangkat mobile tersebut. Dan menampilkan hasil sesuai dengan urutan. Hasil yang ditampilkan bisa berupa atribut musik, misal judul lagu, nama artis, pencipta, dan sebagainya. Namun yang pasti, harus ada *preview* potongan lagu yang sama. Dan terakhir, bisa ditambahkan tombol untuk melakukan pembelian lagu tersebut.

Melihat rancangan di atas, perangkat mobile yang diinstal *music-finder* harus lah mendukung teknologi jaringan, minimal GPRS, dan perangkat untuk melakukan penerimaan input audio.

Untuk server, cukup membutuhkan spesifikasi server standar dan *storage* yang cukup untuk menyimpan data musik kita.

Terakhir, kerja sama dengan pihak label musik untuk penjualan lagu mungkin bisa dijadikan proyeksi ke depan sebagai sumber penghasilan selain dari pemasukan dari biaya *download* aplikasi *music-finder* itu sendiri.

## VII. KESIMPULAN

Algoritma KMP-*Extension* bisa diaplikasikan untuk membuat suatu perangkat lunak, yakni *music-finder*. Keunggulan yang dimiliki KMP-*Extension* sangat mendukung untuk aplikasi *music-finder*. Dengan melakukan sedikit manipulasi inputan dan menggunakan salah satu dari jenis *trade-off*, aplikasi *music-finder* sudah bisa diimplementasikan. Namun, kebutuhan penyimpanan data yang cukup besar untuk file musik membutuhkan dana yang cukup besar pula.

## DAFTAR PUSTAKA


Tanenbaum, Andrew, "Modern Operating System", 2nd Edition, Prentice Hall, 2001, halaman 459-461.

Levityn, Anany, "Introduction to The Design and Analysis of Algorithms", Pearson Education Inc., 2003. Halaman 245-251

Munir, Rinaldi, Strategi Algoritma, Program Studi Informatika, Institut Teknologi Bandung, 2007.

http://toncar.cz/Tutorials/VoIP/VoIP_Basics_Converting_Voice_to_Digital.html, tanggal akses 6 Desember 2010

http://www.nowpublishers.com/product.aspx?product=INR&doi=1500000002, tanggal akses 6 Desember 2010

http://www.midomi.com/, tanggal akses 6 Desember 2010


## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Desember 2010

Ismail Sunni
13508064