BAB II

LANDASAN TEORI

2.1 Sistem Biometrik

Menurut Wikipedia, biometrik berasal dari bahasa Yunani yaitu *bios* = hidup dan *metron* = ukuran, suatu ukuran pengenalan mahluk hidup yang berbasis pada tubuhnya yang unik. Dalam Teknologi Informasi, biometrik lebih sering dipakai sebagai alat otentikasi dengan cara menganalisis karakteristik tubuh manusia yang digunakan, misalnya sidik jari, retina mata, bentuk wajah, cetakan tangan, suara dan lain-lain.

Untuk penggunaan sebagai otentikasi, biometrik harus terlebih dahulu dimasukkan ke dalam data base sebuah sistem. Sidik jari biometrik seseorang hanya akan berfungsi bila sidik jari orang tersebut telah terlebih dahulu dimasukkan ke dalam database sistem, sehingga sistem dapat mengenalinya. Teknologi yang digunakan untuk masing-masing jenis biometrik tentunya berlainan, beberapa telah dapat dilakukan dan dapat ditemui di pasaran namun beberapa masih dalam tahap penelitian.

Biometrik adalah identitas manusia yang unik, tetapi biometrik bukanlah sesuatu yang dapat dengan mudah dirahasiakan. Kita sulit menyembunyikan biometrik yang kita punyai. Biometrik juga tidak dapat memperbaiki kesalahan yang telah terjadi, sekali biometrik kita dicuri, tidak ada cara untuk mengamankannya kembali. Tidak mungkin manusia mengubah sidik jarinya, karena sidik jarinya itu telah dicuri orang lain dan digunakan untuk melakukan kriminalitas, misalnya

Sebuah sistem biometrik pada hakikatnya merupakan sebuah sistem pengenalan pola yang melakukan identifikasi personal dengan menentukan keotentikan dari karakteristik fisiologis dan perilaku tertentu yang dimiliki seseorang.

Secara logika sistem ini dibagi menjadi dua modul: modul pendaftaran dan modul identifikasi. Modul pendaftaran berfungsi untuk mengambil data dari individu dan menyimpannya ke dalam sistem. Pada saat pendaftaran, karakteristik biometrik dipindai terlebih dahulu oleh sebuah pemindai biometrik untuk menghasilkan sebuah representasi digital yang belum diolah. Untuk dapat digunakan dalam proses pencocokan, representasi digital tersebut diproses lebih lanjut untuk mendapatkan representasi yang cukup untuk mewakilinya yang disebut sebagai *template. Template* ini kemudian disimpan dalam pusat database dalam sistem biometrik atau pada kartu magnetik atau s*mart card.*

Modul identifikasi berfungsi untuk mengidentifikasi individu pada titik akses. Pada saat tahap pengoperasian, pemindai biometrik menangkap karakteristik yang akan diidentifikasi dan diubah menjadi format digital, kemudian oleh ekstraktor fitur diproses menjadi representasi yang sama dengan *template*-nya dan kemudian dicocokkan untuk mendapatkan suatu identitas.

2.1.1 Pengenalan Wajah

Model Wajah manusia 3D telah secara luas diaplikasikan seperti wajah, pengkodean/ kompresi video, pengenalan ekspresi wajah, penjejakan kepala, pengenalan tindakan manusia dan pengenlan wajah. Pemodelan wajah manusia menyediakan solusi untuk mengidentifikasi dengan berbagai iluminasi, pose dan ekspresi. Metode berbasis

pada model wajah umum (model segitiga mesh) dan pengukuran wajah individu berisi informasi bentuk dan tekstur. Kecenderungan dalam pengenalan wajah adalah menggunakan model wajah 3d. Sebagai penyajian wajah manusia *object centered,* model wajah 3D digunakan untuk memenuhi bermacam – macam citra wajah. Variasi yang meliputi variasi *extra-subject* (penampilan individu) dan variasi indra-subject (gerakan pose kepala 3D, ekspresi wajah, pencahayaan, dan keriput) masih menjadi tantangan utama.

Peneliti grafik komputer tertarik untuk memperagakan wajah / kepala manusia untuk wajah animasi. DECARLO, dkk (1998) menggunakan pengukuran yang anthropometrik untuk menghasilkan model wajah umum. Pendekatan ini mulai dengan B-Spline yang dibentuk manual dan kemudian diterapkan pada perwajahan dan optimasi batasan ke perwajahan ini. Pendekatan yang kedua, pengukuran wajah secara langsung diperoleh dari digitizers 3D atau cahaya dari sensor, dikenal dengan model wajah WATERS (1996). Model yang dapat dibentu dari BLANTZ (1999) dibentuk dari kombinasi linear bentuk eigendan kombinasi linear dari tekstur eigen berdasarkan sensor laser dari 200 subyek. Pendekatan ketiga, model direkonstruksi dari foto, hanya memrlukan perangkatmasukan pasif dan murah (kamera video). Sebagai contoh, CHEN dan MEDIONI (1998) membangun model wajah dari sepasang citra stereo.

Berbagai teknik yang dapat diterapkan pada pola pengenalan wajah yaitu dengan menggunakan beberapa teknik dan algoritma yang berbeda. Algoritma dan teknik yang pernah dipakai antara lain: algoritma quickprop, active learning, supervised learning, Jaringan Syaraf Tiruan (JST), ekstrasi subcitra dan lain sebagainya.

2.1.2 Analisis DNA

Analisis DNA adalah ciri pengenalan yang sangat bagus-saat ini telah digunakan untuk melacak pelaku kejahatan. Pertimbangan etis dan perlindungan data menghalangi penerapan analisis DNA di bidang lainnya.

DNA merupakan suatu unit informasi kehidupan terkecil yang dimiliki oleh semua makhluk hidup dan diturunkan secara turun temurun. Semakin dekat kekerabatan seseorang maka semakin mirip DNA yang dimilikinya. DNA yang dimiliki tiap makhluk hidup ini adalah unik, dengan kata lain DNA seseorang tidak mungkin sama dengan orang lain. Karena keunikan itu maka pencocokan DNA menjadi salah satu senjata ampuh untuk mengidentifikasi seseorang meskipun sampel yang akan diidentifikasi hanya berupa sehelai rambut atau secuil kulit karena DNA seseorang tersebar di semua bagian tubuh dan memiliki pola yang sama. Karena hal tersebut, maka pencocokan DNA ini sering dipakai untuk identifikasi jasad seseorang yang telah hancur dan tidak bisa dikenali hanya dengan melihatnya.

Rangkaian DNA, yang mengandung informasi kehidupan unik untuk setiap makhluk hidup terdiri dari kumpulan gugus karbon dimana setiap gugus karbon dapat dianalogikan sebagai karakter, sehingga DNA itu sendiri dapat dianalogikan sebagai rangkaian karakter atau String. DNA terdiri dari empat jenis gugus karbon yaitu Adenin (A), Sitosin (S), Timin (T), dan Guanin (G). Dengan demikian maka DNA sama dengan string yang merupakan kombinasi dari 4 jenis karakter yaitu A, S, T, dan G.

2.1.3 Pola Pengenalan Mata

Sebagai suatu fitur yang sangat penting pada tubuh manusia, mata memainkan peran penting di dalam ekstrasksi fitur wajah dan analisis ekspresi perwajahan. Secara nyata dapat kita katakan bahwa mata selain merupakan fitur yang sangat penting juga dipertimbangkan sebagai fitur yang relatif stabiluntuk diangkat di dlam kajian ekstraksi citra wajah dibanding fitur – fitur wajah yang lain. Inilah alasan yang kuat mengapa mata menjadi suatu fitur yang diambil sebagai salah satu tema yang dapat dijadikan bagian pada sistem biometrik.

Eye – detection (Pola Pengenalan Mata) dibagi menjadi pendeteksian posisi mata dan pendeteksian kontur mata. Pendeteksian kontur mata sangat berperan di dalam pengolahan video seperti pada penggunaan aplikasi video conferencing dan video vision. Pada umumnya algoritma untuk pendeteksian kontur mata menggunakan template yang dapat dibuah – ubah disesuaikan dengan keinginan user.

2.1.4 Pengenalan Sidik Jari

Sidik jari adalah gurat-gurat yang terdapat di kulit ujung jari. Fungsinya adalah untuk memberi gaya gesek lebih besar agar jari dapat memegang benda-benda lebih erat. Sidik jari manusia digunakan untuk keperluan dentifikasi karena tidak ada dua manusia yang memiliki sidik jari persis sama. Hal ini mulai dilakukan pada akhir abad ke-19.

Sidik jari manusia merupakan bukti materi yang amat penting. Tak ada sidik jari yang identik di dunia ini sekalipun di antara dua saudara kembar. Dalam dunia sains

pernah dikemukakan, jika ada 5 juta orang di bumi, kemungkinan munculnya dua sidik jari manusia yang sama baru akan terjadi lagi 300 tahun kemudian.

Mengingat betapa akuratnya mengidentifikasikan seseorang lewat sidik jari, diciptakanlah sebuah alat pendeteksi sidik jari dengan sistem elektronik. Alat ini pertama kali digunakan Federal Bureau Investigation (atau populer dengan sebutan FBI) di Amerika Serikat sekitar tahun 60-an. Sidik jari ini biasanya tertinggal di tempat kejadian perkara sebuah peristiwa kriminal. FBI kemudian menggunakannya untuk mengetahui jati diri korban atau bahkan tersangkanya. Hanya dengan memasukkan sidik jari seseorang melalui melalui teknologi komputer, pihak berwenang pun langsung mendapatkan data seputar nama, tanggal lahir dan sejarah kriminalnya.

Sistem identifikasi sidik jari ini yang masuk melalui dunia sains kini telah bergeser keberadaannya. Tak hanya kepentingan dunia pengetahuan atau aparat keamanan saja yang terpenuhi dengan penggunaan alat ini, setiap perusahaan komersial pun merasakan manfaatnya. Yang paling jelas, bukti kehadiran karyawan (absensi) bisa didapat lewat alat ini. Alat ini pun amat populer di antara nasabah perbankan. Dunia otomotif juga tak mau ketinggalan dalam memanfaatkan keunggulan alat ini. Mereka bisa menggunakan sidik jari untuk system kunci. Tujuannya tidak lain tidak bukan untuk keamanan terhadap data yang telah tersimpan. Ini salah satu bukti pula bahwa pemakaian alat pendeteksi sidik jari manusia menjadi lebih berkembang dengan pemakaian yang variatif. Dengan kata lain, kegunaan mesin otomatis pendeteksi sidik jari semakin luas penggunaannya. Tidak hanya untuk mengetahui latar belakang tersangka kriminal—seperti kegunaannya pada awalnya, tetapi juga untuk kemudahan lainnya dalam sebuah perusahaan atau institusi.

Setiap sidik jari memiliki pola unik tertentu, namun dalam pengelompokannya sidik jari di bagi dalam tiga kelompok besar katagori yaitu :

1. Arch

   Pada *pattern* ini kerutan pada sidik jari muncul dari ujung, kemudian mulai naik di tengah, dan berakhir di ujung yang lain.



Gambar 2.1 Sidik Jari Tipe Arch

2. Loop

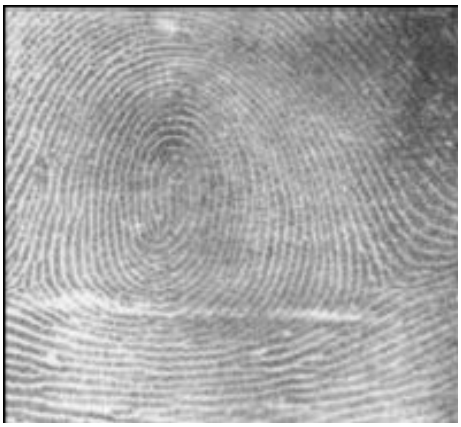
   Pada *pattern* ini kerutan muncul dari satu sisi jari, kemudian membentuk sebuah kurva , dan menuju keluar dari sisi yang sama ketika kerutan itu muncul.



Gambar 2.2 Sidik Jari Tipe Loop

3. Whorl

   Pada *pattern* ini, kerutan berbentuk sirkuler yang mengelilingi sebuah titik pusat dari jari.



Gambar 2.3 Sidik Jari Tipe Whorl

2.2 Komputer Grafik

Menurut Hill (1990 : 2) menyatakan bahwa grafika komputer merupakan sekumpulan alat yang digunakan untuk membuat gambar (*to create pictures*) dan berinteraksi dengan gambar dengan cara – cara seperti yang biasa digunakan (*and to interact with them in natural ways*)

2.2.1 Pengolahan Citra

Pengolahan citra merupakan teknik untuk memodifikasi atau menginterpretasi gambar yang ada, sperti foto dan rangkaian gambar film. Dua macam prinsip pengolahan citra adalah

1. Meningkatkan kualitas gambar
2. Memberikan persepsi dari informasi visual, seperti pada robotic

Untuk melakukan pengolahan citra, pertama – tama membuat digitasi dari foto menjadi file image. Selanjutnya metode digital dapat digunakan untuk memodifikasi gambar sehingga mendapatkan kualitas yang bagus.

Beberapa faktor yang terdapat dalam citra atau *image*, adalah :

1**.** Intensitas cahaya

Intensitas dapat ditranslasikan menjadi suatu sinyal elektris, dan secara paling sederhana menggunakan *photosensitive cells* atau *photosensitive sesistive devices.* Salah satu dari alat – alat photosensitive dapat digunakan untukmembuat kamera primitif yang menghasilkan sederetan sinyal yang menghasilkan tingkatan – tingkatan intensitas cahaya untuk masing – masing spot pada gambar.

2. Warna

Penangkapan warna pada suatu citra meliputi penangkapan tiga citra secara simultan. Dengan sistem RGB, sebagai suatu standarisasi, intensitas masing – masing warna baik red, green, maupun blue harus diukur pada masing – masing spot. Dengan kamera yang beroperasi secara linear yang menjelajahi keseluruhan *visible spectrum,* kumpulan – kumpilanwarba yang sederhana dapat digunakan untuk mengambil tiga citra yang masing – masing, satu untuk spektra red, green, dan blue.

2.2.2 Bitmap

Sering disebutkan bahwa bitmapped graphics berlawanan dengan vektor graphics. Hal ini disebabkan oleh cara menyimpan informasi pada perangkat lunak aplikasi. Botmapped program, juga disebut paint program menyimpan image seperti apa yang ditampilkan pada layar, merupakan array dari titik kecil dengan bentuk segirmpat. Pada

saat gambar diedit dengan paint program, pengontrolan warna dari tiap pixel dapat dilakukan langsung. Jenis penyimpanan image dan manipulasi juga disebut raster graphics dimana raster adalah sekumpulan garis yang sihasilkan oleh cathode ray tube pada monitor.

2.2.3 Gray Scalling

Gray Scalling adalah proses knversi suatu warna – warna pixel pada citra menjadi pixel yang hanya mengandung unsur warna putih – hitam citra dimana hanya bisa dilakukan dengan manipulasi oleh komputer.

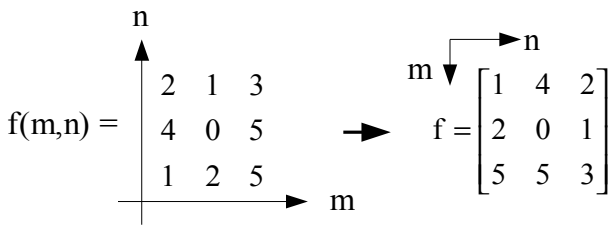
Untuk mengubah suatu citra kontinu ke dalam suatu representasi numerik dilakukan dengan proses digitalisasi oleh suatu *digitizer*, misalnya *scanner*, sehingga citra ini dapat diproses oleh sebuah komputer.

Digitalisasi sebuah citra dilakukan baik terhadap ruang (koordinat (x,y)), maupun terhadap skala keabuannya (f(x,y)). Proses digitalisasi koordinat (x,y) dikenal sebagai "pencuplikan citra" *(image sampling)*, sedangkan proses digitalisasi skala keabuan f(x,y) disebut sebagai "kuantisasi derajat keabuan" *(grey-level quantization)*.

Sebuah citra kontinu f(x,y) akan didekati oleh cuplikan-cuplikan yang seragam jaraknya dalam bentuk matriks MxN, M adalah baris dan N adalah kolom. Nilai elemen-elemen matriks menyatakan derajat keabuan citra, sedangkan posisi elemen tersebut (dalam baris dan kolom) menyatakan koordinat titik-titik (x,y) dari citra. Bentuk matriks di bawah ini dikenal sebagai suatu citra digital.

$$f(x,y)=\begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ . & . & \dots & . \\ . & . & \dots & . \\ . & . & \dots & . \\ f(M-1,0) & . & \dots & f(M-1,N-1) \end{bmatrix}$$

Matriks di atas dapat disajikan dalam bentuk 2 dimensi dalam sistem koordinat Cartesius dengan memutar posisi matriks di atas sejauh 90° searah jarum jam.



Sedangkan derajat keabuan [0,L] dibagi ke dalam G selang dengan panjang selang yang sama, yaitu: $\mathbf{G = 2^m}$ dimana m adalah kedalaman *bit* dan m bilangan bulat positif, bila hal ini diterapkan pada penyimpanan maka sebuah citra digital membutuhkan sejumlah b *bit*, dengan :

$$\mathbf{b = M \times N \times m}$$

Ketika gray scalling meningkatkan melebihi monochrome, hal itu memerlukan kapasitas memory yang lebih besar, karena setiap pixel direpresentasikan dari 4 bits menjadi 8 bits. Pada resolusi 300 dpi anda akan memerlukan lebih dari 8 Megabytes untuk merepresentasikan sebuah halaman dengan ukuran 8,5 X 11 inch menggunakan 256 shades abu – abu.

Banyak perangkat optik yang sudah mendukung proses gray scalling, mulai dari penggunaan 16 sampai 256 shades yang berbeda. Bagaimanapun juga gray scalling akan sangat verguna jika anda mempunyai alat output yang telah mendukung untuk menampilkan senua shades. Banyak monitor yang telah mendukuing untuk tampilan gray scall, namun untuk sebuah gambar tidak baik untuk didedikasikan sebagai gray scalling.

2.3 Algoritma Pencocokan String

String-matching atau pencocokan-kata adalah subjek yang penting dalam kaitannya dengan *text-processing*. Algoritma pencocokan-kata adalah komponen dasar yang dipakai dalam implementasi dari *software* yang berjalan dibawah sistem operasi. Bahkan, algoritma membuat metoda pemrograman menjadi penting sebagai paradigma bidang ilmu komputer lain. Akhirnya, algoritma ini juga memegang peranan penting dalam ilmu komputer teoritik dengan menyediakan masalah yang menantang.

Walaupun data disimpan dalam cara yang berbeda, teks menjaga bentuk utama untuk menukar informasi. Dapat kita lihat secara jelas dalam literatur atau pembicaraan dimana data dibentuk dari kamus. Data tersebut juga diterapkan sama pada ilmu komputer dimana data yang besar disimpan dalam bentuk file linear. Ada juga contoh lain, contohnya dalam biologi molekuler karena molekul biologi dapat kita anggap sebagai barisan nukleotida atau asam amino. Terlebih lagi, jumlah dari data yang tersedia

di bidang itu meningkat duakali lipat tiap delapanbelas bulan. Inilah mengapa alasan algoritma harus efisien walaupun kecepatan komputer saat ini meningkat terus-menerus.

2.3.1 Algoritma Rabin - Karp

Algorima *Rabin-Karp* menggunakan metoda hash dalam mencari suatu kata. Algoritma ini dibuat oleh Michael O. Rabin dan Richard M. Karp. Teori ini jarang digunakan untuk mencari kata tunggal, namun teori ini cukup penting dan sangat efektif bila digunakan untuk mencari lebih dari satu kata. Kasus terbaik dari algoritma ini bisa mencapai O(n), dimana *n* adalah panjang teks. Sayangnya kasus terburuk dari algoritma ini adalah O(*nm*), dimana *m* adalah panjang kata yang menjadi sebab mengapa jarang digunakan. Keuntungan tersendiri dari algoritma ini adalah dapat mencari *k* kata dalam waktu rata-rata O(*n*), yang tidak bergantung pada ukuran *k*.

Algoritma Rabin-Karp ini tidak melakukan pergeseran yang rumit untuk menyelesaikan masalah, algoritma ini mempercepat pengecekan kata pada suatu teks dengan menggunakan fungsi *hash.* Fungsi *hash* adalah fungsi yang mengkonversikan suatu kata menjadi nilai yang disebut nilai *hash* (*hashvalue*). Contohnya seperti hash(“hello”)=5. Rabin-Karp menggunakan fakta bahwa jika suatu kata adalah sama maka nilai *hash*-nya juga sama. Jadi kita dapat melihat bahwa yang perlu kita lakukan adalah mencari nilai *hash* dari kata yang kita punya kemudian mencari kata dalam teks yang mempunyai nilai *hash* yang sama.

2.3.2 Algoritma Boyer – Moore

Algoritma pencarian kata Boyer-Moore termasuk algoritma yang cukup efisien. Dikembangkan oleh Bob Boyer dan J Strother Moore pada tahun 1977. Algoritma ini akan bertambah cepat jika kata yang dicari panjang. Keefisienan yang algoritma ini punya berasal dari fakta bahwa, setiap pencocokan yang gagal antara teks dan kata yang dicari, algoritma ini menggunakan informasi yang didapat dari proses awal untuk melewati karakter-karakter yang tidak cocok.

Algoritma *Boyer-Moore* didasarkan pada dua *heuristic*: looking-glass heuristic dan *character-jump heuristic*.

Looking-glass heuristic melakukan perbandingkan suatu karakter akhir pada kata *w* dengan suatu karakter pada teks *s*. Jika karakter tersebut sama maka jendela karakter akan berjalan mundur pada kedua string dan mengecek kembali kedua karakter. Ilustrasinya seperti tabel berikut, nomor merupakan langkah dari awal pencocokan

Character-jump heuristic melakukan suatu aksi ketika perbandingan antara dua karakter yang berbeda. Ada dua aksi yang tergantung pada teks *s* dan kata *w* yang dimiliki; jika *p* yaitu karakter pada *s* yang sedang diproses yang tidak cocok, maka ada dua kemungkinan aksi.

Pertama jika *p* terdapat dalam kata *w*, geser kata *w* agar karakter *p* pada *w* (pilih yang paling akhir jika ada lebih dari satu) sejajar dengan karakter *p* pada *s*. Di bawah ini ilustrasi dengan *s*=”a pattern” dan *w*=”rithm”.

2.3.3 Algoritma Knuth – Morris – Pratt

Algoritma *Knuth-Morris-Pratt* (KMP) mencari kehadiran sebuah kata *w* dalam teks *s* dengan melakukan observasi awal (*preprocessing*) yaitu ketika muncul ketidaksamaan kata ini mempunyai informasi mengenai kapan kesamaan selanjutnya

bermula, dengan cara mengece ulang kata sebelumnya. Algoritma ini dibuat oleh Knuth dan Pratt dan sendiri oleh J. H. Morris pada tahun 1977, namun ketiganya mem-*publish*-nya bersamaan.

Algoritma Knuth-Morris-Pratt (KMP) merupakan algoritma yang digunakan untuk melakukan proses pencocokan *string*. Algoritma ini merupakan jenis *Exact String matching Algorithm* yang merupakan pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama Contoh : kata *algoritmik* akan menunjukkan kecocokan hanya dengan kata *algoritmik*. Pada algoritma Knuth-Morris-Pratt (KMP), kita simpan informasi yang digunakan untuk melakukan pergeseran lebih jauh, tidak hanya satu karakter seperti algoritma *Brute Force*. Algoritma ini melakukan pencocokan dari kiri ke kanan.

Terdapat beberapa definisi pada algoritma Knuth-Morris-Pratt (KMP) :

1. Misalkan A adalah alfabet dan $x = x1x2\ldots xk$ adalah *string* yang panjangnya k yang dibentuk dari karakter karakter di dalam alfabet A.

a) Awalan (*prefix*) dari x adalah upa-*string* (*substring*) u dengan $u = x1x2\ldots xk - 1$, $k \in \{1, 2, \ldots, k-1\}$ dengan kata lain, x diawali dengan u.

b) Akhiran (*suffix*) dari x adalah upa-*string* (*substring*) u dengan $u = xk - b\ xk - b + 1\ \ldots xk$, $k \in \{1, 2, \ldots, k- 1\}$ dengan kata lain, x di akhiri dengan v

c) Pinggiran (*border*) dari x adalah upa-*string* r sedemikian sehingga $r = x1x2\ldots xk - 1$ dan $u = xk - b,\ xk - b + 1\ \ldots xk$, $k \in \{1, 2, \ldots, k-1\}$, dengan kata lain, pinggiran dari x adalah upa-*string* yang keduanya awalan dan juga akhiran sebenarnya dari x.

2. Fungsi Pinggiran b(j) didefinisikan sebagai ukuran awalan terpanjang dari P yang merupakan akhiran dari P[1..j].

2.3.3.1 Contoh Algoritma KMP

Variabel *m* menunjukan indeks karakter pada *s* dan variabel *i* menunjukan karakter pada *w*. Berikut ini adalah gambaran pada awal eksekusi:

```
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W: ABCDABD
i: 0123456
```

Algoritma ini dimulai dengan membandingkan karakter pada kata *w* secara paralel pada karakter pada *s*, bergeser ke indeks selanjutnya (*m*=*m*+1) jika sama. Namun ketika mencapai langkah ke-4 (*m*=3), kita mendapat *s*[*m*] adalah spasi dan yang menyatakan karakter tidak sama dengan *w*[*m*]=’D’. Kita tidak akan kembali dari *s*[0], kita perhatikan bahwa tidak ada ‘A’ pada posisi 0 sampai 3 di *w* kecuali pada saat 0; karena pada saat *preprocessing* kita sudah mengecek semua posisi karakter, kita tau bahwa tidak akan ada peluang untuk menemukan awal kata *w* (‘A’) yang cocok dari awal. Oleh karena itu kita langsung menuju ke karakter selanjutnya (logis: mengapa kita harus capai-capai mundur kalau tidak ada?), dan mulai mencocokan dengan kata *w* kembali sehingga *m*=4 dan *i*=0.

```
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W: ABCDABD
i: 0123456
```

Pada langkah selanjutnya kita akan mendapatkan sebuah kata yang hampir lengkap yaitu “ABCDAB” saat *w*[6] (ketika *s*[10]). Karena belum menemukan kata yang tepat kita akan mencari lagi namun tidak dari awal. Ternyata pada sebelumnya kita

menemukan bagian awal kata *w*. karena terdapat suatu awal yang sesuai yaitu “AB” oleh karena itu kita cukup me-*reset m*=8 dan *i*=2 dan melanjutkan pencocokan.

```
m: 012345678901234567890 12
S: ABC ABCDAB ABCDABCDABDE
W: ABCDABD
i: 0123456
```

Pengecekan selanjutnya akan langsung gagal karena spasi tidak sama dengan ‘C’, namun, karena tidak ada awal yang sama pada *substring* teks (‘A’), maka seperti pada eksekusi pertama, kita mulai pecarian pada karakter selanjutnya dari *s*: *m*=11 dan reset *i*=0.

```
m: 012345678901234567890 12
S: ABC ABCDAB ABCDABCDABDE
W: ABCDABD
i: 0123456
```

KMP menemukan kata yang hampir cocok “ABCDAB” namun karakter selanjutnya ‘C’, tidak sama. Dengan alasan sebelumnya kita buat *m*=15, dan *i*=2 karena ada kata “AB” pada akhir pencarian.

```
m: 012345678901234567890 12
S: ABC ABCDAB ABCDABCDABDE
W: ABCDABD
i: 0123456
```

Pada akhirnya kita dapat menemukan kata yang sepadan pada karakter awal *s*[15].

Contoh di atas menjelaskan seluruh kemungkinan dari algoritma KMP. Untuk saat ini, kita anggap terdapat suatu tabel T atau disebut *partial table* yang mempengaruhi algoritma sudah terbuat,. Isi dari *T* dibuat sedemikian rupa sehingga jika kita memulai pencocokan dari *s*[*m*] yang gagal ketika membandingkan pada karakter ke *s*[*m* + *i*] pada *w*[*i*], maka pencocokan selanjutnya akan dimulai pada indeks *m* + *i* - *T*[*i*] pada *s* (*T*[*i*] adalah jumlah langkah mundur yang harus dilakukan), yang mempunyai dua implikasi. Pertama, *T*[0] = -1 yang berarti bahwa jika pada saat *w*[0] tidak cocok, maka kita tidak

bisa kembali ke karakter sebelumnya sehingga kita harus maju ke karakter selanjutnya; dan kedua walaupun kemungkinan karakter sama selanjutnya dimulai pada $m + i - T[i]$ seperti contoh di atas, kita tidak perlu mengecek karakter T[i] setelahnya, jadi kita melanjutkan pencarian mulai dari $w[T[i]]$.

```
algorithm kmp_search:
input:
an array of characters, S (the text to be searched)
an array of characters, W (the word sought)
output:
an integer (the zero-based position in S at which W is found)
define variables:
an integer, m ← 0 (the beginning of the current match in S)
an integer, i ← 0 (the position of the current character in W)
an array of integers, T (the table, computed elsewhere)
while m + i is less than the length of S, do
: if W[i] = S[m + i], let i ← i + 1
if i equals the length of W, return m
otherwise, let m ← m + i - T[i], and if i > 0, let i ← T[i]
(if we reach here, we have searched all of S unsuccessfully)
return the length of S
```

2.3.3.2 Efisiensi algoritma pencarian KMP

Dengan memperhitungkan kehadiran tabel T, porsi pencarian dengan algoritma Knuth-Morris-Pratt mempunyai kompleksitas O(k), di mana k adalah panjang dari S. Kita anggap semua ekseskusi yang dilakukan berada dalam kalang while, kita akan menghitung jumlah iterasi yang terjadi dalam loop ini. Untuk melakukannya kita harus memperhatikan nilai dalam tabel T. T dibentuk sedemikian rupa sehingga ketika pencocokan dimulai pada $s[m]$ gagal ketika mencocokan $s[m + i]$ pada $w[i]$, pencocokan selanjutnya harus dimulai pada karakter $s[m + (i - T[i])]$. Karena indeks tidak boleh lebih tinggi daripada m, sehingga $T[i] < i$.

Melihat kejadian tadi, kita akan melihat bahwa loop melakukan eksekusi paling banyak $2k$ kali. Untuk setiap iterasi, loop ini mengeksekusi salah satu dari dua cabang dalam loop:

Cabang pertama menambah *i* dengan satu dan tidak mengubah *m*, sehingga indeks *m* + *i* yang merupakan karakter yang sedang diproses bertambah. Cabang kedua menambahkan *i* – *T*[*i*] pada *m*, dan seperti yang kita lihat nilainya selalu positif. Lalu lokasi *m* pada awal kemungkinan pencocokan bertambah.

Jika diperhatikan sekarang loop akan berhenti jika *m* + *i* = *k*, maka setiap cabang dari loop dapat mencapai maksimum eksekusi *k* kali, karena bertambahnya *m* atau *m* + *i*, dan $m \leq m + i$; jika *m* = *k*, maka tentu saja $m + i \geq k$, sehingga karena bertambah satu-satu maka kita pasti mempunyai *m* + *i* = *k* pada suatu titik sebelumnya, dan oleh karena itu kedua cara dapat dikerjakan.

Loop yang dieksekusi paling banyak 2*k* kali menunjukan bahwa kompleksitas waktu pencarian algoritma ini adalah O(k).

2..3.3.3 *Partial table* (*failure function*)

Tujuan ari tabel ini adalah untuk mencegah algoritma melakukan pencocokan pada suatu karakter pada *s* lebih dari satu kali. Hal yang memungkinkan ini terjadi adalah kita tahu dimana pencocokan selanjutnya akan berlangsung, setelah kita mengecek beberapa bagian pada teks utama terhadap bagian dari kata (karakter per karakter). Dengan kata lain kita harus mengecek ulang kata itu sendiri dan membuat list kemungkinan posisi (pada *preprocessing*) yang melewati karakter-karakter yang tidak mungkin sama, namun tentu saja tidak melewati karakter yang sama.

Hal ini lebih baik dari pada kembali ke posisi *w*[0] yang sudah pasti gagal; sepanjang inilah kita harus mundur kebelakang (*backtrack*) untuk mencocokan *substring* berikutnya. Nilai tersebut disimpan dalam `T[i]` yang merupakan panjang kemungkinan

inisialisasi segmen terpanjang dari *w* yang juga merupakan segmen dari *substring* yang berakhir pada *w*[*i* - 1]. Kita menggunakan konvensi dimana *string* kosong mempunyai panjang 0. Karena *mismatch* pada awal pola adalah kasus khusus (tidak ada kemungkinan mundur), kita set *T*[0] = -1, seperti yang didiskusikan diatas.

2.2.4. Deskripsi dan *pseudocode* pembuatan *partial table*

Contoh diatas mengilustrasikan teknik umum untuk membuat tabel. Prinsipnya adalah dari pencarian keseluruhan: sebagian besar pekerjaan sudah dikerjakan saat menuju ke karakter saat ini, jadi hanya sedikit perlakuan yang perlu dilakukan. Dibutuhkan sedikit inisialisasi di awal agar berjalan sesuai rencana.

```
/* algorithm kmp_table */
Void kmp_table(int T[]) {
/*kamus*/
i=2; /*posisi saat ini*/
j=0; /*indeks berbasis nol di w*/
/*algoritma*/
/*nilai awal diset sesuai algo*/
t[0] = -1;
t[1] = 0;
while (i<strlen(w))
{
if (w[i-1]==w[j])
/*kasus pertama: untaian kata berlanjut*/
{
t[i] = j + 1;
i++;
j++;
}
else if (j > 0)

/*kasus kedua: tidak berlanjut namun dapat kembali lagi*/
{
j = t[j];
}
else /*kasus ketiga: benar-benar tidak ada kandidat. I.S. j = 0*/
{
t[i] = 0;
i++;
}
}
}
```

2.3.3.5. Contoh pembuatan *partial table*

Mari kita lihat contoh di atas kembali dari w = "ABCDABD". Kita set dahulu *T*[0] = -1. Untuk mencari *T*[1], kita harus menemukan posisi awal yang merupakan ”A“ yang juga merupakan awal kata *w*. Namun ternyata bukan “A”, maka kita set *T*[1]=0. sama dengan selanjutnya *T*[2]=0.

Berikut langkah-langkah selanjutnya berdasarkan algoritma:

Inisialisasi awal

i	0	1	2	3	4	5	6
W[i]	A	b	c	d	a	b	d
T[i]	**-1**	**0**	-	-	-	-	-

i=2, j=0, kasus tiga

i	0	1	2	3	4	5	6
W[i]	a	b	c	d	a	b	d
T[i]	-1	0	**0**	-	-	-	-

i=3, j=0, kasus tiga

i	0	1	2	3	4	5	6
W[i]	a	b	c	d	a	b	d
T[i]	-1	0	0	**0**	-	-	-

i=4, j=0, kasus tiga

i	0	1	2	3	4	5	6
W[i]	a	b	c	d	a	b	d
T[i]	-1	0	0	0	**0**	-	-

i=5, j=0, kasus satu

i	0	1	2	3	4	5	6
W[i]	a	b	c	d	a	b	d
T[i]	-1	0	0	0	0	**1**	-

i=6, j=1, kasus satu

i	0	1	2	3	4	5	6
W[i]	a	b	c	d	a	b	d
T[i]	-1	0	0	0	0	1	**2**

2.3.3.6. Efisiensi pembuatan *partial table*

Kompleksitas algoritma tabel adalah O(*n*), dimana *n* adalah panjang kata *w*. Kecuali beberapa inisialisasi, semua pekerjaan kita diselesaikan dalam kalang loop while, kita dapat melihat loop ini dieksekusi dalam waktu O(*n*), yang dapat diselesaikan meneksaminasi jumlah *i* dan *i* - *j*. dalam kasus pertama, *i* - *j* tetap, karena keduanya naik secara bersamaan. Di kasus kedua, *j* di-*assign* oleh *t*[*j*]—nilainya selalu kurang dari *j*—

yang menyebabkan $i - j$ naik. Pada kasus ketiga i dinaikkan dan j tidak, jadi baik i dan $i - j$ naik. Karena $i \geq i - j$, ini berarti pada saat itu baik i atau batas bawah i naik; maka karena algoritma berhenti ketika $i = n$, algoritma ini harus berhenti setelah paling maksimum $2n$ iterasi loop, karena $i - j$ mulai dari 1. Maka kompleksitas algoritma ini O(n).

2..3.3.7. Mengenai efisiensi algoritma

Melihat dari waktu kompleksitas kedua subalgoritma dari *partial table* dan proses pencocokan yaitu O(k) dan O(n), maka rata-rata waktu kompleksitasnya adalah O($n + k$). Dapat dilihat dari contoh diatas, algoritma ini memberikan keuntungan yang sangat besar pada saat melewati karakter-karakter dalam selang waktu tertentu. Semakin sedikit eksekusi untuk mundur, maka algoritma ini semakin cepat, yang dapat kita lihat dari kehadiran nol pada tabel T. Kata seperti “ABCDEFG“ bekerja sangat baik dengan algoritma ini karena tidak ada pengulangan dalam katanya, sehingga tabel menjadi nol semua dengan awalnya -1. Namun bila kita melihat kasus algoritma dengan kata “AAAAAAA“, kata ini sangat buruk, karena tabelnya akan seperti:

W[i]	a	a	a	a	a	a	a
T[i]	-1	0	1	2	3	4	5

Bentuk seperti ini adalah pola yang paling buruk untuk *partial table* T, dan dapat dilihat keburukannya jika menggunakan kata seperti S = "AAAAAABAAAAAABAAAAAAA". Algoritma ini akan mengecek seluruh ‘A’ terhadap karakter ‘B’ hingga berhenti, mencapai duakali dari jumlah karkter S. Walaupun pekerjaan untuk pembuatan tabel ini cepat untuk kata ini (karena tidak kembali ke belakang), pekerjaan ini dijalankan hanya sekali untuk w yang diberikan, namun akan

melakukan eksekusi pencarian berkali-kali. Jika pada setiap waktu, *w* dicari dalam kata S, efisiensi rata-rata akan menjadi sangat buruk. Contoh tersebut cocok untuk algoritma pencarian Boyer-Moore.

Algoritma KMP ini menjanjikan karena mempunyai waktu pencarian yang linier, baik dalam kasus terbaik atau terburuk, sedangkan Boyer-Moore memiliki kasus terbaik yang baik, juga kasus terburuk yang buruk.