

## **BAB II DASAR TEORI**

### **2.1 *Character Encoding***

*Character encoding* adalah suatu metode untuk memasangkan suatu karakter dengan sesuatu simbol alami. Simbol alami yang dimaksud dapat berupa angka, huruf atau bahkan pulsa elektrik. Tujuan proses *character encoding* adalah untuk memfasilitasi penyimpanan teks dalam komputer ataupun pengiriman teks melalui jaringan telekomunikasi. Contoh dari proses pengkodean semacam ini adalah kode morse, yang memasangkan alphabet latin dengan panjang atau pendek suatu sinyal morse [WIK06].

Contoh lainnya adalah pengkodean dengan menggunakan standar ASCII yang memasangkan alphabet latin dengan angka yang direpresentasikan dalam tujuh bit kode biner. Pengkodean dengan standar ASCII menjadi hal yang umum dilakukan pada proses pengkodean dokumen.

#### **2.1.1 *Sejarah Character Encoding***

Pada awal perkembangannya, komputer hanya dapat menggunakan karakter-karakter dari Bahasa Inggris. Pada saat itu digunakan pengkodean ASCII yang merepresentasikan seluruh karakter dengan angka. Angka yang digunakan berada dalam kisaran 32 – 127, sebagai contoh karakter “A” menjadi 65, spasi menjadi 31, dan seterusnya. Kode di bawah angka 32 disebut *unprintable character* dan digunakan sebagai karakter kontrol. Sebagai contoh kode 7 akan mengakibatkan komputer mengeluarkan bunyi *beep* [SPO03].

Dengan pengkodean semacam ini maka huruf-huruf latin tersebut dapat disimpan dalam 7 bit. Pada saat itu sebagian besar komputer yang ada merupakan komputer berbasis 8 bit. Dalam delapan bit biner dapat tersimpan maksimal 255 kode, akibatnya kode 128 – 255 tidak digunakan dalam pengkodean ASCII. Beberapa pihak menggunakan kode 128 – 255, yang tidak terpakai ini untuk kepentingannya masing-masing. Sebagai contoh *IBM-PC* menggunakan kisaran kosong ini untuk

beberapa huruf yang beraksen serta *line drawing character*. Di banyak negara, dimana Bahasa Inggris bukan merupakan bahasa utama, kisaran kosong ini digunakan untuk mendukung bahasa negara tersebut. Tentunya hal semacam ini mempersulit pertukaran dokumen antar bahasa.

Standar ANSI merupakan alternatif solusi terhadap permasalahan ini. Dalam standar ini dilakukan persetujuan untuk kode yang lebih kecil dari 128, hal ini tidak banyak berbeda dari ASCII. Untuk kode dengan kisaran 128 sampai 255, ditangani dengan cara yang berbeda-beda. Kelompok cara penanganan inilah yang kemudian dikenal dengan *code pages*. Dengan metode semacam ini dimungkinkan untuk menggunakan dua bahasa yang berbeda pada satu dokumen.

Walaupun demikian masih terdapat keterbatasan dari sistem *code pages* ini. Sistem *code pages* dapat digunakan untuk dokumen yang terdiri atas dua bahasa dengan dua jenis karakter yang berbeda, latin dan non-latin. Pengkodean untuk bahasa dengan karakter latin akan menggunakan kode dengan kisaran 32-127, sedangkan bahasa dengan karakter non-latin akan menggunakan kode dengan kisaran 128-255. Contohnya adalah dokumen yang menyimpan tulisan dalam Bahasa Inggris dan Bahasa Arab. Namun penggunaan dua bahasa yang keduanya menggunakan karakter non-latin yang berbeda (contoh: Bahasa Arab dan Bahasa Jepang) merupakan hal yang mustahil karena keduanya memerlukan *code pages* yang berbeda untuk kode dalam kisaran yang sama, yaitu 128-255.

Masalah lain yang timbul berasal dari negara-negara di Asia. Fakta bahwa banyak negara Asia yang menggunakan ribuan jenis huruf, dimana tidak mungkin untuk huruf-huruf tersebut dapat dikodekan dalam 8 bit. Alternatif penyelesaian untuk masalah ini adalah sebuah sistem yang disebut *Double Bytes Character Sets (DBCS)*. Pada sistem ini ada huruf-huruf ada yang menggunakan dua byte, dan ada pula yang menggunakan satu byte. Hal semacam ini sangat menyulitkan para pengembang aplikasi. Proses pembacaan suatu *string* menjadi lebih kompleks, karena harus membedakan apakah suatu karakter direpresentasikan menggunakan satu byte atau dua bytes. Karena kesulitan inilah sistem DBCS tidak umum digunakan oleh pengembang aplikasi.

Munculnya internet membawa komplikasi baru pada masalah ini. Pertukaran dokumen melalui media internet menjadi hal sangat umum dilakukan. Pertukaran ini sering kali melibatkan pihak yang menggunakan bahasa yang berbeda.

### **2.1.2 Character Encoding pada saat ini**

Berbagai masalah yang telah disebutkan sebelumnya, memaksa untuk melakukan pendekatan secara lebih sistematis terhadap metode pengkodean yang ada. *Unicode* adalah salah satu dari pendekatan sistematis ini. Karena sangat kompleksnya metode pengkodean yang digunakan pada *Unicode* maka topik ini akan dibahas secara mendalam di sub bab selanjutnya.

## **2.2 Unicode**

Unicode adalah suatu standar untuk melakukan *encoding* terhadap karakter universal yang digunakan untuk merepresentasikan teks pada proses komputer [UNI06]. Unicode menyediakan suatu cara yang konsisten untuk melakukan *encoding* terhadap berbagai bahasa. Tujuan dari adanya standar ini adalah untuk memudahkan para pengguna komputer ketika berhubungan dengan teks yang berasal dari berbagai bahasa. Selain itu pengguna komputer yang menggunakan berbagai simbol matematis ataupun simbol teknis lainnya, akan sangat dimudahkan dengan penggunaan standar unicode.

Ide dasar dari desain Unicode adalah kesederhanaan dan konsistensi pada standar ASCII. Tidak seperti ASCII yang hanya mampu melakukan pengkodean terhadap alphabet latin, Unicode memiliki kemampuan untuk melakukan encoding terhadap hampir seluruh bahasa tulis yang ada di dunia. Hal ini dilakukan dengan memberikan suatu nilai numerik dan nama yang unik untuk tiap karakter yang ada. Pada bagian berikut, akan dijelaskan mengenai elemen-elemen penting yang terdapat didalam standar Unicode.

### 2.2.1 Code Point

Metode encoding sebelum Unicode menggunakan asumsi dasar bahwa sebuah huruf akan diwakilkan oleh sekelompok bit. Sekelompok bit ini dapat disimpan didalam memori atau disk, sebagai contoh karakter “A” yang pada standar ASCII memiliki kode 65, akan tersimpan di memori komputer dalam bentuk biner sebagai: 0100 0001.

Unicode memiliki cara pandang yang berbeda mengenai karakter. Pada Unicode sebuah huruf akan diwakilkan oleh sebuah *code point*. *Code point* adalah sebuah konsep logik, sehigga terbebas dari masalah penulisannya didalam memori atau disk. *Code Point* diawali dengan “U+” yang berarti “Unicode” kemudian diikuti oleh angka dalam basis heksa desimal. Sebagai contoh *code point* untuk karakter “A” adalah : U+0041. Untuk setiap karakter yang telah terdaftar dalam *Unicode* akan diberikan satu *code point* [SPO03],

Didalam *Unicode* Setiap karakter yang diwakilkan oleh *code point* bersifat *platonik ideal*. Dimana suatu huruf “A” berbeda dengan “B” atau “a” tetapi sama dengan “*A*” (dalam *italic*) dan “A” (dalam ukuran huruf yang berbeda).

Berbeda dengan mitos yang menyatakan bahwa *Unicode* adalah pengkodean 16-bit, sehingga hanya memiliki maksimal 65.536 karakter, tidak ada batas dalam jumlah karakter yang mungkin disimpan dalam *Unicode*. Bahkan untuk *Unicode* versi 4.0 telah memberikan *code point* kepada sebanyak 96.447 karakter [UNI06]. Perlu ditekankan bahwa huruf dan *code point* yang mewakilinya, masih bersifat logik sehigga terlepas dari metode penulisan pada memori, disk ataupun proses penampilannya di layar.

### 2.2.2 Ecoding Forms

*Character encoding* pada *Unicode* akan mendefinisikan bagaimana suatu *code point* jika direpresentasikan dalam bentuk bit biner, atau dengan kata lain bagaimana suatu *code point* disimpan didalam memori atau *disk* [UNI06].

Dalam Unicode, saat ini terdapat tiga metode *encoding* yang mungkin dilakukan. Ketiga *encoding* ini memungkinkan representasi hingga  $2^{32}$ . Jumlah ini mencukupi untuk melakukan *encoding* terhadap seluruh karakter yang ada, termasuk didalamnya karakter-karakter dalam sejarah, dan berbagai simbol yang ada. Ketiga metode encoding ini adalah: UTF-8, UTF-16, dan UTF-32. Perlu diingat bahwa Unicode tidak membatasi *encoding* hanya pada UTF-32. Jika diperlukan, maka dapat disusun model encoding dengan menggunakan jumlah bit yang lebih banyak, misalnya UTF-64.

Sebagai contoh sebuah string “Hello” akan menjadi (dalam basis heksadesimal, di tulis per *byte*):

<i>Code Point</i>	:U+0048 U+0065 U+006C U+006C U+0066
UTF-8	:48 65 6C 6C 6F
UTF-16	:00 48 00 65 00 6C 00 6C 00 6F
UTF-32	:00 00 00 48 00 00 00 65 00 00 00 6C 00 00 00 6C 00 00 00 6F

UTF-8 sangat populer pada HTML dan protokol sejenisnya. Pada metode encoding ini seluruh karakter Unicode disimpan pada *byte* atau kelompok *byte* yang panjangnya bervariasi. Keuntungan terbesar dari metode *encoding* ini adalah seluruh karakter Unicode yang memiliki *code point* U+0000 – U+007F akan disimpan dalam satu *byte*, sehingga memiliki bentuk yang sama seperti ASCII (0-127). Hal ini membuat banyak perangkat lunak yang berbasis ASCII masih dapat menggunakan UTF-8. Untuk *code point* U+0080 (ASCII 128) dan keatas akan disimpan dalam 2, 3 atau bahkan 4 *byte* [SPO03][UNI06].

UTF-16 sangat baik digunakan pada lingkungan operasi dimana diperlukan keseimbangan antara kecepatan akses karakter dan jumlah ruang penyimpanan yang dipergunakan. Pada metode *encoding* ini hampir seluruh karakter yang sering dipergunakan dapat masuk kedalam kode 16-bit. Sementara untuk karakter yang lain dapat diakses melalui 2 buah kode 16-bit.

UTF-32 biasanya digunakan pada lingkungan operasi dimana jumlah memori bukan menjadi masalah utama. Pada metode ini setiap *code point* disimpan dalam satu buah kode 32-bit.

Ketiga *encoding* tersebut membutuhkan maksimal 4 byte untuk merepresentasikan sebuah karakter (*code point*) [UNI06].

## 2.3 *N-gram*

*Definisi dari n-gram* adalah potongan sebanyak  $n$  karakter dari suatu *string* yang lebih panjang [CAV94]. Pemotongan *string* untuk menghasilkan *n-gram* dapat dilakukan dengan pemotongan yang tidak kontinu, dan pemotongan yang kontinu. Sebagai contoh, pada *string* “abcdefg”, hasil pemotongan satu karakter pada awal dan akhir *string* tersebut akan menghasilkan *string* “ag”, yang dapat dikategorikan sebagai sebuah *n-gram* hasil dari pemotongan yang tidak kontinu..

Apabila *string* “abcdefg” tersebut dipotong 3 karakter pada awal string, maka akan menghasilkan *string* “abc”, yang dapat dikategorikan sebagai sebuah *n-gram* hasil dari pemotongan yang kontinu. Untuk selanjutnya, pada dokumen ini *n-gram* yang digunakan adalah *n-gram* yang merupakan hasil pemotongan yang kontinu.

Dengan mengubah suatu kata menjadi *n-gram* maka *n-gram* tersebut dapat dianggap suatu ruang vektor, sehingga komparasi dapat dilakukan dengan mudah. Sebagai contoh pada suatu, string yang diubah menjadi *tri-gram*, komparasi dapat dilakukan pada suatu ruang vektor dengan dimensi  $26^3$  dimana dimensi pertama adalah “aaa”, dimensi kedua adalah “aab” dan seterusnya. Walaupun setelah diubah menjadi *tri-gram* informasi mengenai string tersebut menjadi hilang, namun secara empiris jika dua string yang memiliki vektor yang serupa (memiliki nilai cosinus yang dekat) maka kedua string tersebut kemungkinan besar juga serupa [WIK06].

## 2.4 *Language Identification*

*Language identification* adalah proses untuk menentukan isi dari suatu dokumen termasuk dalam bahasa yang mana [WIK06]. Dikarenakan banyaknya jenis maupun perspektif yang ada dalam mempelajari bahasa, munculah beberapa isu yang kontroversial dalam proses indentifikasi bahasa. Salah satu isu yang utama adalah definisi bahasa itu sendiri. Karena bahasa bukan merupakan suatu benda yang memiliki ciri-ciri yang unik, maka dasar dari sebuah bahasa harus di definisikan secara operasional, dalam artian definisi suatu bahasa yang dipilih oleh seseorang akan bergantung kepada tujuan orang tersebut untuk mengidentifikasi bahasa. Beberapa orang menggunakan definisi yang murni berdasarkan linguistik, beberapa orang lain mungkin saja mengikutsertakan aspek-aspek lain seperti sosial, budaya, ataupun politik [ETH06].

Secara garis besar, ada dua pendekatan untuk melakukan identifikasi bahasa, yaitu *non-computational / non-statistical approach* dan *computational / statistical approach*. *Non-computational / non-statistical approach* merupakan metode yang mendasarkan identifikasi bahasa pada kemunculan kata-kata kunci, atau kata-kata yang sering digunakan dalam suatu bahasa. Proses identifikasi bahasa dengan metode ini melibatkan suatu daftar “*common words*” untuk bahasa tertentu. Kata-kata pada dokumen yang sedang diidentifikasi bahasanya, akan dicocokkan dengan kata-kata pada daftar “*common words*” tersebut. Beberapa kemunculan kata yang terdapat dalam daftar sudah cukup untuk menjadi landasan dalam pengambilan keputusan mengenai bahasa yang digunakan oleh dokumen. Kesederhanaan dan kecepatan dalam proses identifikasi adalah keunggulan idetntifikasi bahasa dengan menggunakan metode ini [WIK06].

Ada beberapa masalah yang timbul dari penggunaan metode ini. Pertama, membuat suatu daftar “*common words*” yang representatif tidaklah mudah [AHM04]. Jika terlalu banyak kata-kata yang masuk kedalam daftar tersebut, maka proses penentuan bahasa akan menjadi sulit, karena jumlah kata yang perlu diperhatikan menjadi semakin banyak. Sedangkan bila jumlah kata yang dimasukkan terlalu sedikit, akan mengakibatkan daftar tersebut tidak representatif,

yang pada akhirnya dapat mengakibatkan kesalahan pada proses identifikasi bahasa.

Selain itu, masalah lain yang mungkin timbul adalah fakta bahwa suatu dokumen belum tentu hanya terdiri atas satu bahasa saja. Dalam suatu dokumen, ada kemungkinan terdapatnya bagian-bagian tertentu yang memiliki bahasa lain. Hal ini mengakibatkan proses identifikasi akan menjadi sulit, dan kemungkinan salah melakukan identifikasi bahasa semakin besar.

*Computational / statistical approach*, sesuai dengan namanya, menggunakan metode statistik dalam penentuan bahasa yang dimiliki oleh dokumen. Secara garis besar, metode ini membandingkan suatu statistik tertentu pada dokumen yang bahasanya belum diketahui dengan dokumen lain yang bahasanya telah diketahui. Beberapa variasi dari metode ini adalah membuat model suatu bahasa untuk dibandingkan dengan dokumen yang belum diketahui bahasanya. Secara umum dapat dikatakan metode ini lebih modern.