KOMPRESI PADA BARCODE PDF-417

Joanna Helga

Sekolah Tinggi Manajemen Informatika dan Komputer LIKMI
Jl. Ir. H. Juanda 96 Bandung 40132

Abstract

Dewasa ini *barcode* banyak digunakan dalam berbagai keperluan hidup. *Barcode* 2-Dimensi PDF-417 menawarkan jenis *barcode* baru yang dapat menampung hingga 1000 *byte* data. Artikel ini membahas teknik kompresi yang digunakan pada barcode PDF-417, sehingga dapat menampung hingga 1100 *byte* data, 1800 karakter ASCII, atau 2700 *digit* angka.

Key Word : Barcode, kompresi, dynamic programming

1. Pendahuluan

Sebuah data dapat direpresentasikan dalam berbagai cara, salah satunya adalah dengan menggunakan gambar. Cara representasi data seperti ini kadangkala lebih mudah dibaca dan lebih padat. Jenis representasi data lewat gambar yang sering kita jumpai sehari-hari diantaranya *barcode*. Barang-barang yang dijual di supermaket umumnya diberi sebuah label *barcode* untuk menyimpan kode barang. Selanjutnya kode tersebut akan dibaca dengan bantuan *scanner* dan dipetakan ke sebuah *record* data pada *database.*

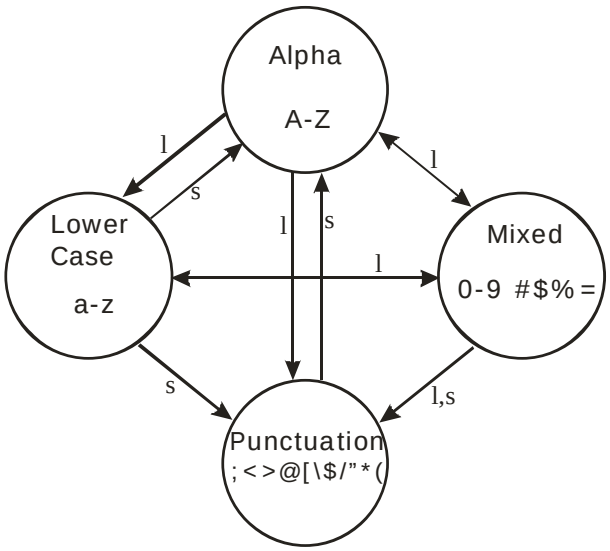
Gambar 1-1 Label barcode PDF-417

Pada perkembangannya, telah ditemukan jenis *barcode* 2 dimensi yang dapat menampung data lebih banyak. Dengan kemampuannya menampung sekitar 1000 *byte*, *barcode* jenis ini tidak hanya dapat menampung kode barang, tapi dapat menampung seluruh *record* data, seperti nama barang, harga jual, dan sebagainya. *Barcode* 2 dimensi ini telah distandarisasi pada tahun 1994 dengan nama PDF-417.

Pada implementasinya, digunakan sebuah teknik kompresi. Dengan teknik ini, sebuah label *barcode* PDF-417 dapat menampung sekitar 1100 *byte* data, atau 1800 karakter ASCII, atau 2700 *digit* angka.

2. Teknik Kompresi PDF-417

Cara kompresi data pada PDF-417 adalah dengan membagi karakter ASCII kedalam beberapa himpunan yang berbeda. Dengan cara ini, setiap karakter dapat direpresentasikan hanya dengan 5 *bit* saja. Ada 2 macam operasi yang dapat dipakai untuk membantu berpindah dari satu himpunan ke himpunan lainnya. Masing-masing operasi ini membutuhkan 5 *bit* tambahan. Operasi-operasi tersebut adalah *shift* dan *latch*. Operasi *shift* berarti pindah untuk 1 simbol saja, lalu kembali ke himpunan sebelumnya, sedangkan operasi *latch* berarti pindah permanen. Contoh bagan karakter pada teknik kompresi teks untuk PDF-417 adalah seperti pada gambar 2-1.



Gambar 2-1 Bagan himpunan karakter pada kompresi PDF-417 Teks

Pada penerapannya, sejumlah data dapat direpresentasikan menggunakan teknik kompresi PDF-417 dengan banyak cara. Contohnya, data ”abcdefABCghijkl” dapat dimampatkan dengan menggunakan himpunan ASCII huruf besar dan huruf kecil.

1. Cara pertama adalah dengan mulai pada himpunan huruf kecil, menuliskan ”abcdef”. Lalu berpindah ke himpunan huruf besar dengan *latch*, menuliskan ”ABC”. Kemudian latch lagi ke huruf kecil, menuliskan ”ghijkl”.

2. Cara kedua adalah dengan mulai pada himpunan huruf kecil, menuliskan ”abcdef”. Lalu *shift* ke huruf besar, menuliskan ”A”, *shift* lagi untuk menuliskan ”B”, *shift* lagi untuk menuliskan ”C”. Kemudian menuliskan ”ghijkl”.

Kedua cara ini benar dan dapat dipakai, tapi cara kedua lebih boros dalam penggunaan *bit*. Pada contoh diatas, cara pertama menggunakan 2 buah operasi perpindahan, sedangkan cara kedua menggunakan 3 buah operasi perpindahan, sehingga cara kedua menggunakan 5 *bit* lebih banyak dari cara pertama.

3. Representasi Data yang Paling Optimal

Representasi data yang paling optimal adalah representasi yang membutuhkan jumlah *bit* paling sedikit. Bagaimana mencari representasi yang paling optimal ? Cara yang paling sederhana adalah dengan membuat semua kemungkinan representasi, menghitung harga masing-masing, lalu mencari yang paling kecil. Namun cara ini tidak mungkin dilakukan, karena representasi suatu data akan sangat banyak jumlahnya.

Cara yang paling cepat adalah dengan menggunakan *Algoritma Greedy*. Dengan cara ini kita hanya perlu melihat beberapa huruf di kedepan, lalu menentukan langkah yang terbaik untuk diambil dengan mempertimbangkan huruf-huruf didepannya tersebut. Cara ini memiliki kompleksitas $O(n)$, dimana n adalah jumlah data. Namun cara ini tidak menghasilkan representasi paling optimal.

Cara lain yang sedikit lebih lama, tapi dapat menghasilkan representasi paling optimal adalah Dynamic Programming. Algoritma Dynamic Programming merupakan algoritma yang mencatat hasil dari iterasi-iterasi sebelumnya, untuk kemudian digunakan pada iterasi selanjutnya. Algoritma ini memiliki kompleksitas $O(n \times m)$, dimana n adalah jumlah data, dan m adalah jumlah himpunan kecil.

4. Implementasi Algoritma Dynamic Programming

Misalkan kita memiliki sebuah data “abcABC” dan himpunan yang digunakan adalah huruf besar dan huruf kecil. Salah satu representasi yang mungkin adalah :

1. mulai dari himpunan huruf kecil,
   tulis ’a’ (harga = 1, total = 1)

2. tulis ’b’ (harga = 1, total = 2)
3. tulis ’c’ (harga = 1, total = 3)
4. latch ke himpunan huruf besar (harga = 1, total = 4),
   tulis ‘A’ (harga = 1, total = 5)
5. tulis ‘B’ (harga = 1, total = 6)
6. tulis ’C’ (harga = 1, total = 7)

Total harga pada karakter ke-*i* amat dipengaruhi harga pada karakter ke-*i-1* dan himpunan saat *i-1*. Misalnya pada saat i = 2, karakter yang akan ditulis adalah ’b’ yang merupakan huruf kecil. Kita tahu bahwa total harga sebelumnya adalah 1 dan himpunan yang sedang digunakan adalah huruf kecil, maka kita dapat langsung menuliskan karakter ’b’ tersebut tanpa harus menggunakan operator *shift/latch*. Sebaliknya ketika i = 4, karakter yang akan dituliskan adalah ’A’ yang merupakan huruf besar. Himpunan yang sedang digunakan adalah huruf kecil, karena itu kita perlu menambahkan operator *shift/latch* untuk berpindah ke himpunan yang sesuai.

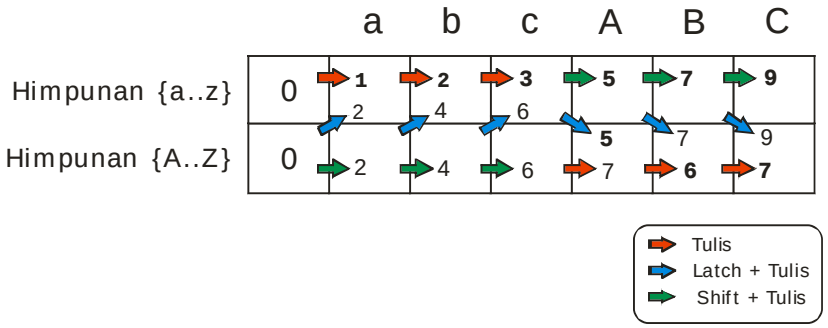
Jika kita mengetahui representasi yang paling optimal untuk setiap himpunan pada saat *i-1*, maka dengan mudah kita dapat mencari representasi optimal pada saat *i*. Harga paling kecil untuk himpunan tertentu pada saat karakter ke-*i* dapat didefinisikan sebagai minimum dari :

- harga saat *i-1* pada himpunan yang sama + 1 (tulis)
- harga saat *i-1* pada himpunan yang berbeda + 1 (*shift/latch*) + 1 (tulis)

Pada saat i = 1, harga untuk himpunan yang sesuai adalah 1 (tulis) dan harga untuk himpunan yang tidak sesuai adalah 2 (tulis + *shift/latch*).

Pada implementasinya, digunakan sebuah tabel untuk menyimpan harga-harga optimal ini. contoh tabel harga untuk data ”abcABC” ditampilkan pada gambar 4-1. Baris menandakan jenis himpunan yang sedang digunakan, sedangkan kolom menunjukkan karakter yang akan ditulis. Tanda panah berwarna merah menunjukkan langkah tulis, yaitu langkah yang dapat dilakukan jika karakter yang akan ditulis sesuai dengan himpunan saat itu. Tanda panah berwarna biru merupakan langkah *latch* + tulis, yaitu langkah yang dapat dilakukan untuk berpindah himpunan. Tanda panah berwarna hijau merupakan langkah *shift* + tulis, yaitu langkah yang dapat

dilakukan jika karakter yang akan ditulis tidak sesuai dengan himpunan saat ini tapi tidak ingin pindah permanen. Angka bercetak tebal merupakan harga optimal untuk kotak yang bersangkutan. Angka ini akan digunakan untuk mencari nilai kotak selanjutnya. Pada contoh ini, harga optimal untuk teks ”abcABC” adalah 7.



Gambar 4-1 Tabel harga optimal

5. Referensi

- Tysso., **PDF417 Barcode Symbology Introduction**, In http://www.barcodemanufacturer.com/barcode_scanner/ home.html, 2005.
- Steven S. Skiena. **The Algorithm Design Manual,** Springer-Verlag, New York, 1998.