Dynamic Programming dalam *Levenshtein Distance* untuk Mengetahui Keterbedaan Dua String

Rizka Irawan Ardiyanto (NIM: 13506012)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha no. 10 Bandung
e-mail: if16012@students.if.itb.ac.id

ABSTRAK

Dalam perkembangan ilmu pengetahuan dan terknologi, kemampuan pengolahan string pada komputer adalah salah satu kemampuan dasar yang penting. *Levenshtein distance* adalah representasi tingkat perbedaan dua buah string dalam angka yang dikembangkan algoritmanya dengan konsep dynamic programming.Dengan membagi permasalahan perbandingan dua string ini dalam tahap-tahap sejumlah panjang string yang lebih panjang, algoritma ini dapaat menemukan solusi optimal yang menunjukkan tingkat perbedaan dua string tersebut. Algoritma Levenshtein distance juga menggunakan sebuah matriks untuk menyimpan angka-angka yang merepresentasikan banyaknya langkah yang diperlukan untuk mengubah suatu string menjadi string lain, dan mengembalikan langkah minimumnya.

Kata kunci: *String, Levenshtein Distance, Insertion, Deletion, Substitution*.

1. PENDAHULUAN

Komputer, sebuah benda yang sejak pertama kalinya dibuat telah banyak membantu memudahkan berbagai macam urusan manusia. Seiring dengan perkembangan Ilmu Pengetahuan dan Teknologi, perkembangan kemampuan komputer juga semakin meningkat.

Tetapi, komputer sehebat apapun tetap memiliki berbagai macam fungsi dan kemampuan dasar yang membuatnya dapat melakukan berbagai macam hal. Salah satu dari kemampuan dasar tersebut adalah kemampuan mengolah data *string*.

Kemampuan mengolah *string* ini menjadi dasar untuk berbagai macam fitur yang dimiliki komputer zaman sekarang, mulai dari untuk membantu memperbaiki tulisan (*spell checker*) dalam berbagai macam dokumen sampai mendeteksi plagiarisme dokumen atau kode.

Selain dengan berkembangnya Teknologi, perkembangan Ilmu Pengetahuan juga menjadi faktor yang mempengaruhi kemampuan kerja komputer. Salah satunya adalah perkembangan Ilmu Pengetahuan yang berkaitan dengan komputasi dan algoritma yang sangat banyak mempengaruhi bagaimana komputer dapat bekerja dengan optimal hanya dengan sedikit sentuhan dari manusia.

Salah satu dari ilmu pengetahuan tersebut adalah Strategi Algoritmik, di mana di sini dibahas bagaimana merumuskan berbagai macam algoritma untuk dapat membuat komputer bekerja dengan sangat efisien.

Berkaitan dengan hal-hal di atas, salah satu topik bahasan Strategi Algoritmik, yaitu *dynamic programming*, telah membantu melahirkan sebuah algoritma baru dalam hal pengolahan *string* sebagaimana sebuah *string* dapat diketahui tingkat keterbedaannya dengan sebuah string lain. Tingkat Keterbedaan ini dinamakan *Levenshtein Distance* dan menjadi salah satu cikal bakal teknologi-teknologi pengolahan string tingkat tinggi yang telah semakin memudahkan banyak hal dan urusan manusia.

2. DYNAMIC PROGRAMMING

Dynamic Programming adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah (step) atau tahapan (stage) sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan.

Pada *dynamic programming,* rangkaian keputusan yang optimal dibuat dengan menggunakan Prinsip Optimalitas. Prinsip ini berbunyi: *jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal*. Prinsip Optimalitas ini berarti bahwa jika kita bekerja dari tahap k sampai tahap k+1, kita dapat menggunakan hasil optimal dari tahap k tanpa harus kembali ke tahap awal. Jika pada setiap tahap kita menghitung ongkos (cost), maka dapat dirumuskan bahwa:

(cost sampai tahap k+1) = (cost tahap k) + (cost tahap k ke tahap k+1)

Dengan prinsip optimalitas ini, dijamin bahwa pengambilan keputusan pada suatu tahap adalah keputusan yang benar untuk tahap-tahap selanjutnya.

Dalam menyelesaikan persoalan dengan program dinamis, orang dapat menggunakan dua pendekatan (*approach*) berbeda: maju (*forward* atau *up-down*) dan mundur (*backward* atau *bottom-up*). Misalkan x_1, x_2, ..., x_n menyatakan peubah (*variable*) keputusan yang harus diambil pada masing-masing tahap 1,2...,n, maka:

1. Program dinamis Maju. Program dinamis bergerak mulai dari tahap 1, terus maju ke tahap 2, 3 dan seterusnya sampai tahap n.
2. Program dinamis Mundur. Program dinamis bergerak mulai tahap n, terus mundur ke tahap n-1, n-2 dan seterusnya sampai tahap 1.

Penyelesaian secara maju atau mundur keduanya ekivalen dan menghasilkan solusi optimum yang sama. Akan tetapi, pengalaman menunjukkan bahwa penyelesaian dengan p[rogram dinamis mundur umumnya lebih efisien.

Secara umum ada empat langkah yang dilakukan dalam mengembangkan algoritma program dinamis:

1. Karakteristikkan struktur solusi optimal
2. Definisikan secara rekursif nilai solusi optimal
3. Hitung nilai solusi optimal secara maju atau mundur
4. Konstruksi solusi optimal

Perlu dicatat bahwa solusi optimal yang dihasilkan oleh program dinamis dapat lebih dari satu buah.

3. PENGOLAHAN STRING

String adalah sebuah representasi data dalam komputer yang berupa kumpulan dari karakter/ huruf yang disimpan dalam sebuah array/ tabel yang memiliki indeks. Dengan demikian, string dapat dengan mudah diakses karakter-karakternya (dengan mengakses indeks tabel) dan membandingkannya dengan *string* lain.

Inti dari pengolahan string adalah Penambahan (*Insertion*), Penghapusan (*Deletion*), Penukaran (Substitution), dan Pencocokan String (Matching).

Penambahan adalah jika suatu string ditambahkan dengan string lain atau jika ke dalam suatu string ditambahkan suatu karakter.

Penghapusan adalah jika di dalam suatu string, dihilangkan satu karakter atau lebih.

Penukaran adalah jika suatu karakter dalam suatu string diganti dengan karakter lain yang berbeda.

Pencocokan String adalah bagaimana suatu string dibandingkan dengan string lain dan dilihat kesamaan karakter-karakternya.

4. LEVENSHTEIN DISTANCE

Dalam kajian Ilmu Komputer dan Informatika, *Levenshtein Distance* adalah suatu ukuran untuk mengetahui tingkat perbedaan antara dua string yang disajikan berupa suatu angka. Angka ini menunjukkan jumlah modifikasi yang perlu dilakukan (*Insertion, Deletion, Substitution*) terhadap string awal agar dapat menjadi tepat sama dengan string target.

Sebagai contoh, *Levenshtein distance* antara string "Mahar" dan string "Rahang" adalah 3. Hal ini dikarenakan terdapat sedikitnya 3 langkah untuk memodifikasi string "Mahar" agar dapat menjadi "Rahang". Berikut Ilustrasi sederhananya:

1. Mahar -> Rahar (*Substitution,* ganti M dengan R)
2. Rahar -> Rahan (*Substitution,* ganti r dengan n)
3. Rahar -> Rahang (*Insertion,* tambahkan g pada akhir string)

Seperti yang telah dijelaskan di atas, *Levenshtein distance* dapat diartikan pula sebagai jumlah modifikasi yang perlu dilakukan pada suatu string untuk menjadi string lain. Oleh karena itu, *Levenshtein distance* sering disebut juga sebagai *edit distance*

Dalam aplikasinya, *Levenshtein distance* dipakai menjadi dasar dalam berbagai teknologi dan fitur dalam program-program komputer yang berurusan dengan string. Salah satu diantaranya adalah fitur *spell checker* pada Microsft Word. Dengan mengetahui *Levenshtein distance* antara suatu kata dengan kata-kata yang lain, dapat ditampilkan berbagai macam kata yang mungkin pada awalnya user salah mengetikkan.

5.DYNAMIC PROGRAMMING DALAM LEVENSHTEIN DISTANCE

Levenshtein distance tentunya tidak didapat dengan melakukan perhitungan manual untuk setiap masalah. Algoritmanya telah ada dan dikembangkan untuk kembali memudahkan manusia dalam hal perhitungan.

Dalam Algoritma *Levenshtein distance*, dilakukan *dynamic programming bottom-up* yang bekerja dengan melibatkan suatu matriks yang berisi angka-angka yang merupakan ongkos (cost) yang dibutuhkan untuk mengubah suatu string menjadi string lain. Algoritma ini mengembalikan suatu angka yang merupakan ongkos (cost) minimum yang diperlukan untuk mengubah string awal menjadi string target yang kita kenal bersama dengan *Levenshtein distance*. Berikut *pseudocode* algoritma *Levenshtein distance*:

```
int LevenshteinDistance(char s[1..m], char
t[1..n])
   // d is a table with m+1 rows and n+1
columns
   declare int d[0..m, 0..n]

   for i from 0 to m
       d[i, 0] := i
   for j from 0 to n
       d[0, j] := j

   for i from 1 to m
       for j from 1 to n
       {
           if s[i] = t[j] then cost := 0
                          else cost := 1
           d[i, j] := minimum(
             d[i-1, j] + 1, //deletion
             d[i, j-1] + 1, //insertion
             d[i-1, j-1] + cost //substitution
                               )
       }

   return d[m, n]
```

Seperti telah disebutkan di atas, algoritma ini mengguanakn prinsip *dynamic programming* di mana jumlah tahap yang dibutuhkan untuk menemukan solusi adalah sejumlah panjang string yang lebih panjang diantara dua string yang dibandingkan. Pada setiap tahap, pilihan yang ada adalah *Insertion*, *Deletion*, dan *Substitution* karakter.

Algoritma ini juga melibatkan penggunaan sebuah matriks berukuran n+1 x m+1 dimana n dan m adalah panjang string yang akan dibandingkan. Perhatikan gambar berikut:

		M	a	h	a	r
	0	1	2	3	4	5
R	1	1	2	3	4	5
a	2	2	1	2	3	4
h	3	3	2	1	2	3
a	4	4	3	2	1	2
n	5	5	4	3	2	2
g	6	6	5	4	3	3

Gambar 1. Matriks *Levenshtein distance* untuk string "Mahar" - "Rahang"

Pada gambar di atas, terlihat angka-angka yang membingungkan. Cara membaca matriks tersebut tidak dapat dilakukan dengan cara mengambil semua elemen matriks. Matriks tersebut selalu memberikan satu hasil yaitu yang kotak yang berada pada kanan bawah. String pada baris pertama adalah string awal dan string pada kolom pertama adalah string target.

Pada matriks di atas juga telah diberikan tanda-tanda bagaimana mengubah string "Mahar" menjadi "Rahang" dengan langkah yang minimum. Pada angka 1 yang pertama, dilakukan *Substitution* 'M' dengan 'R', oleh karena itu telah ada 1 langkah. Pada angka 1 yang kedua, karakter yang dibandingkan sama yaitu 'a' sehingga tidak ada penambahan langkah, oleh karena itu angka yang terisi tetap 1. Pada angka 1 yang ketiga dan yang keempat tetap tidak ada penambahan langkah.

Pada angka 2, terdapat sekali lagi *substitution* antara 'r' dengan 'n'. Kemudian, pada angka 3, dilakukan *insertion* karakter 'g'. Pada titik ini, matriks telah terisi semua, dan angka yang berada di kanan bawah akan diambil sebagai ongkos minimum.

Berikut satu lagi ilustrasi untuk memudahkan pengertian anda untuk membaca tabel ini.

		M	a	h	a	r
	0	1	2	3	4	5
R	0	1	2	3	4	5
a	1	2	1	2	3	4
h	2	3	2	1	2	3

Gambar 2. Matriks Levenshtein distance untuk string "Mahar" - "Rah"

Dapat dikatakan bahwa gambar di atas adalah potongan dari gambar sebelumnya. Pada gambar ini, terlihat bahwa *Levenshtein distance* antara "Mahar" dengan "Rah" adalah juga 3. Dengan ini diperlihatkan bahwa matriks ini sebenarnya hanya bergfungsi sebagai *container* untuk memudahkan perhitungan, sementara hasil yang dicari hanya pada 1 elemen terakhirnya saja.

Penjelasan gambar di atas kurang lebih sama dengan gambar sebelumnya. Hanya pada angka 2 yang ditandai, terjadi *deletion* karakter 'a', dan pada angka 3 terjadi *deletion* huruf 'r'.

6.BATAS ATAS DAN BATAS BAWAH LEVENSHTEIN DISTANCE

Satu hal yang menarik dari *Levenshtein distance* adalah terdapatnya angka-angka yang menjadi batas maksimum dan minimumnya. Hal ini dapat berguna untuk berbagai macam aplikasi yang melakukan komputasi atau perbandingan terhadap angka-angka ini. Berikut beberapa diantara batas- batas tersebut:

1. *Levenshtein distance* minimum adalah selisih panjang string yang dibandingkan.
2. *Levenshtein distance* maksimum adalah panjang string yang lebih panjang.
3. *Levenshtein distance* adalah nol, jika kedua string identik (sama).

4. Jika string yang dibandingkan adalah s dan t, *Levenshtein distance* minimum adalah jumlah karakter yang ada di s yang tidak ditemukan di t.

7.KESIMPULAN

Levenshtein distance, sebuah algoritma yang dikembangkan olehVladimir Levenshtein menggunakan konsep *dynamic programming* dalam penerapannya. Dengan algoritma ini, dapat diketahui tingkat perbedaan dua buah string dalam representasi angka.

Dalam pengguanaannya, algoritma ini menggunakan sebuah matriks berukuran panjang string-string yang dibandingkannya. Nilai yang diberikan oleh *Levenshtein distance* adalah selalu angka yang terletak pada kotak paling kanan-bawah matriks, sebagai langkah minimum untuk mengubah string awal menjadi string target.

Levenshtein distance juga dapat diperkirakan tanpa perlu diperhitungkan sebelumnya. Perkiraan ini dilakukan dengan dasar adanya batas maksimum dan minimum *Levenshtein distance* antara dua buah string yang memenuhi kasus tertentu.

Pengembangan algoritma ini telah memberikan banyak kemajuan dalam bidang teknlogi komputer. Berbagai fitur komputer seperti *spellchecker* dan detektor plagiarisme dapat dikembangkan berkat ditemukannya *Levenshtein distance* ini. Dan tentunya, pengembangan algoritmanya tidak akan ada tanpa munculnya ilmu *dynamic programming*.

REFERENSI

[1] Munir, Rinaldi. Strategi Algoritmik. Institut Teknologi Bandung : Bandung 2007.

[2] http://en.wikipedia.org/wiki/Levenshtein_distance. Waktu akses: 19 Mei 2008

[3]http://www.csse.monash.edu.au/~lloyd/tildeStrings/Alignment/92.IPL.html Waktu akses: 20 Mei 2008

[4] http://www-igm.univ-mlv.fr/~lecroq/seqcomp/node2.html Waktu akses: 19 Mei 2008

[5] http://www.merriampark.com/ld.htm. Waktu akses: 19 Mei 2008