PENERAPAN ALGORITMA X-DIFF DALAM MENDETEKSI PERUBAHAN PADA DOKUMEN XML

Oleh : Adriansyah (13505070)

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : ryan_if05@students.itb.ac.id

ABSTRAK

XML (eXtensible Markup Language) merupakan bahasa yang sekarang ini telah menjadi standar universal dalam pertukaran informasi dan data. Banyak sekali sistem dan aplikasi pada web yang memanfaatkan XML sebagai format standar dalam mengirim dan menerima data berupa dokumen atau yang lainnya. Oleh karena itu, tidak mengherankan jika XML sekarang menjadi primadona dan mampu menggeser pendahulunya, yaitu HTML (HyperText Markup Language).

Dikarenakan pemanfaatannya yang sudah meluas dan juga didukung dengan fakta bahwa saat ini alur informasi bergerak sedemikian cepatnya, maka kemampuan untuk mengubah format dan isi pada XML secara efektif dan efisien telah menjadi sebuah keharusan. Oleh karena itu, dibutuhkan algoritma yang mangkus untuk memenuhi kebutuhan ini. Salah satunya ialah algoritma X-Diff. Algoritma ini bertujuan untuk mendeteksi berbagai perubahan yang telah terjadi pada dua atau lebih dokumen XML, baik perubahan struktur maupun isinya.

Kata kunci: XML, HTML, X-Diff, web.

1. PENDAHULUAN

XML adalah bahasa yang telah menjadi standar dalam transportasi data di internet. Tidak seperti pendahulunya, yaitu HTML, bahasa ini memiliki keunggulan dalam desain struktur. Pemrogram dapat mendefinisikan sendiri tag dan strukturnya sesuai dengan keinginan dan kebutuhan mereka masing-masing.

XML menyimpan data tanpa menspesifikasi secara khusus bagaimana data tersebut akan ditampilkan. Hal ini merupakan keuntungan tersendiri karena kita hanya terfokus pada proses strukturisasi data saja tanpa memperdulikan tampilan *interface* dari data tersebut. Hal ini sangat berbeda dengan HTML di mana selain menyimpan data, kita juga harus menambahkan tag untuk pengaturan *interface*, seperti <p> (paragraf), , <b>, dan sebagainya. Keuntungan lain XML ialah bahasa ini *platform-independent* sehingga dapat digunakan di berbagai jenis aplikasi yang tidak kompatibel sehingga memudahkan pengguna dalam pertukaran data.

Struktutr komponen dalam dokumen XML terbagi menjadi 3 jenis, yaitu :

a. Simpul Elemen, yaitu simpul bukan daun dengan sebuah label nama. Simpul ini dapat terbagi lagi menjadi beberapa simpul sub-elemen.
b. Simpul Teks, yaitu simpul daun dengan sebuah label nama.
c. Simpul Atribut, yaitu simpul daun dengan 2 buah label, yaitu label nama dan nilai.

Selain itu, dalam XML dikenal istilah DOM (Document Object Model). DOM ialah bentuk representasi model dari XML. Bahasa ini juga dapat direpresentasikan dalam bentuk pohon (*tree model*), yang terbagi menjadi dua jenis, yaitu *ordered trees* dan *unordered trees*. Dua pembagian tersebut didasarkan pada urut tidaknya simpul-simpul pada pohon.

Salah satu alasan mengapa dibutuhkan algoritma yang efisien untuk mendeteksi perubahan pada struktur XML ialah karena arus informasi yang begitu cepat. Salah satu contoh ilustrasi dari masalah ini ialah ketika seseorang ingin memesan buku melalui sebuah situs *online*. Pada saat kunjungan pertama, katakanlah dia baru melakukan survei dan mengumpulkan informasi tentang buku yang ingin dibelinya. Ketika ingin memesan, ternyata dia lupa nomor rekeningnya dan terpaksa dia kembali ke rumah untuk melihat nomor rekeningnya. Satu jam kemudian, sewaktu dia ingin memesan buku yang ingin dibelinya, ternyata buku tersebut telah habis terjual.

Dari contoh ilustrasi di atas jelas sekali menggambarkan pesatnya arus informasi. Oleh karena itu, diperlukan sebuah algoritma khusus yang dapat menangani dengan cepat berbagai perubahan pada struktur internal data. Salah satu algoritma yang dinilai dapat menjawab masalah ini ialah algoritma X-Diff.

2. ALGORITMA X-DIFF

Algoritma X-Diff merupakan salah satu algoritma yang khusus menangani XML. Algoritma lainnya yang juga mempunyai fungsi yang sama ialah algoritma XyDiff. Algoritma X-Diff melakukan pendekatan dengan metode *top-down*, berbeda dengan algoritma XyDiff yang *bottom-up*.

Ada 3 operasi dasar pada pohon DOM yang akan kita gunakan dalam algoritma X-Diff ini, yaitu :

a. **Insert (x (nama,nilai), y)** : Menambahkan sebuah simpul x dengan label nama "*nama*" dan label nilai "*nilai*" sebagai simpul anak dan daun dari simpul y.
b. **Delete (x)** : Menghapus simpul daun x.
c. **Update (x, nilai_baru)** : Mengubah nilai dari simpul daun x dengan "*nilai_baru*", x dapat berupa simpul teks atau simpul atribut. Operasi *update* hanya mengubah nilainya bukan namanya.

Sebagai tambahan, 3 operasi dasar di atas hanya berlaku pada simpul daun. Oleh karena itu, ditambahkan dua operasi komposisi sebagai berikut :

a. **Insert (Tx, y)** : Menambah sebuah subpohon Tx.
b. **Delete (Tx)** : Menghapus subpohon Tx.

Sekuensial dari beberapa operasi dasar di atas dikenal dengan istilah edit script. Edit script bertujuan untuk mengkonversi dari satu pohon ke pohon lainnya. Contoh edit script ialah : E (T1 → T2) = Delete (5,a), Insert (5, (B,x), 3), Update (6,w). Edit script ini akan berguna pada saat penerapan algoritma X-Diff ini nantinya.

Selain itu, dikenal juga istilah *signature* pada X-Diff. *Signature* sebuah simpul ialah konkatenansi semua nama ancestor dari simpul tersebut, contohnya adalah Bacaan.Buku.Penulis.$Teks$.

Algoritma X-Diff melakukan beberapa tahap penyelesaian umum dalam melakukan pendeteksian perubahan pada dokumen XML, yaitu :

a. *Parsing* dan *Hashing*
b. *Checking* dan *Filtering*
c. *Matching*
d. *Generating Minimum-Cost EditScript*

Di bawah ini merupakan *pseudo-code* dari algoritma X-Diff.

```
Input : (DOC1, DOC2)

{Tahap Parsing dan Hashing}
Parsing DOC1 ke T1 dan hash T1;
Parsing DOC2 ke T2 dan hash T2;

{Tahap Checking dan Filtering}
If ( XHash (Akar (T1)) = XHash (Akar(T2)) ) Then
    {DOC1 dan DOC2 berarti ekivalen, berhenti }
Else
    If ( XHash (x1) = XHash (x2) ) Then
        Hapus subpohon dari Akar ke x1;
        Hapus subpohon dari Akar ke x2;

{Tahap Matching, mencari Mmin(T1,T2) dari (T1→T2) }
Do Matching;

{Tahap Generating minimum-cost edit script dari Mmin(T1,T2)}
Do Edit Script;
```

2.1. *Parsing* dan *Hashing*

Pada tahap ini, X-Diff akan mem-*parsing* dua dokumen XML (DOC1 dan DOC2) ke dalam pohon T1 dan T2 (pohon DOM). Selama proses *parsing*, X-Diff akan menggunakan sebuah fungsi yang menghitung nilai XHash pada setiap simpul. Fungsi tersebut bernama hash(z) dengan z merupakan parameter yang bertipe *string*. Adapun persamaan umunya ialah sebagai berikut :

XHash (x) = hash (tipe (x) . nama (x) . nilai (x))

Untuk simpul elemen x (simpul bukan daun) yang mempunyai *list* simpul anak x_1, x_2, …, x_n, XHash(x) diperoleh dari beberapa tahapan berikut :

a. Mengurutkan nilai XHash sehingga XHash x_1) ≤ XHash (x_2) ≤ … ≤ XHash (x_n).
b. Konkatenasi tipe (x) dan nama (x) dengan nilai XHash yang sudah terurut tadi. Kemudian hitung nilai XHash (x) dengan rumus :

XHash(x) = hash (tipe(x) . nama(x) . XHash (x_1) . XHash(x_2) . … . XHash (x_n))

2.2. *Checking* dan *Filtering*

Pada tahap ini, X-Diff akan membandingkan nilai XHash antara Akar (T1) dan Akar (T2). Jika nilai XHash mereka sama, maka T1 dan T2 ekivalen atau isomorfik. Sedangkan jika tidak sama, X-Diff menggunakan nilai XHash untuk mengurangi ukuran dari dua pohon tersebut

dengan mengurangi subpohon pada level kedua yang ekivalen.

2.3. *Matching*

Pada tahap ini, X-Diff akan mencari M_{min} (T1,T2), yaitu *minimum-cost matching* di antara dua pohon T1 dan T2. Untuk menemukan *minimum-cost matching*, pertama-tama X-Diff akan memfilter subpohon yang ekivalen di antara dua akar pohon tersebut dengan cara membandingkan nilai XHash dari simpul anak level kedua. Yang kedua, X-Diff akan menghitung jarak di antara T1 dan T2 yang akhirnya akan menghasilkan *minimum-cost matching* Mmin (T1, T2). Algoritma lengkapnya dapat dilihat pada *pseudo-code* di bawah ini.

```
Input : pohon T1 and T2.
Output : minimum-cost matching Mmin(T1, T2).
Inisialisasi :
N1 = {semua simpul daun di T1}
N2 = {semua simpul daun di T2}
Tabel Jarak DT = {}.

{ Langkah 1: Kurangi matching space}
Filter subpohon di level selanjutnya yang mempunyai
nilai XHash sama

{ Langkah 2: Hitung jarak (distance editing) untuk
(T1 → T2)}
Do {
    For (setiap simpul x di N1)
        For (setiap simpul y di N2)
            If Signature(x) = Signature(y) Then
                Hitung Dist(x, y);
                Simpan matching (x, y) dengan Dist(x,
                y) di tabel DT.
    N1 = {simpul orangtua dari simpul sebelumnya
         di N1};
    N2 = { simpul orangtua dari simpul sebelumnya
         di N2}.
} While (N1dan N2 tidak kosong)

{Langkah 3: Tandai yang matching di pohon T1 dan
T2}
 Mmin(T1, T2) = {}
If Signature(Akar(T1)) ≠ Signature(Akar (T2)) Then
    Return;  {Mmin(T1, T2) = {}}
Else
    Add (Akar(T1), Akar(T2)) ke Mmin(T1, T2).
    For (setiap simpul bukan daun mapping (x, y) ∈
    Mmin(T1,T2))
        Ambil simpul anak yang matching dan
        simpan di DT
        Tambahkan simpul anak yang matching ke
        Mmin(T1, T2).
```

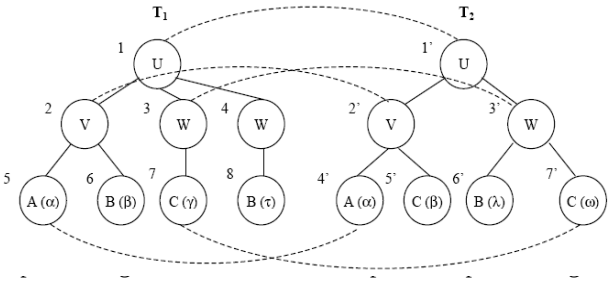
2.4. *Generating Minimum-Cost Edit Script*

Pada tahap ini, X-Diff akan membangkitkan *minimum-cost* editscript E untuk (T1 → T2) berdasarkan M_{min} (T1,T2) yang sudah didapatkan di tahap *Matching*. Proses pembangkitan (*generate*) ini bekerja secara rekursif dari akar ke daun. Algoritma lengkapnya dapat dilihat pada *pseudo-code* di bawah ini.

```
Input: Pohon T1 and T2, minimum-cost matching
Mmin(T1, T2),
Tabel Jarak DT.
Output:  edit script E.
Inisialisasi:
E = Null;
x = Akar(T1), y = Akar(T2).
If (x, y) ∉ Mmin(T1, T2)
    Return "Delete(T1), Insert(T2)".  {Delete dan
    Insert subpohon}
Else if Dist(T1, T2) = 0
    Return "";
Else
    For setiap pasang simpul (xi , yj) ∈ Mmin(T1, T2),
    xi ialah simpul anak dari x,
    yj ialah simpul anak dari y.
    If xi dan yj  adalah simpul daun
        If Dist(xi, yj) = 0
            Return "";
        Else
            Add Update(xi , Value(yj)) to E;  {Update
            simpul daun}
    Else
        Add Edit Script(Txi, Tyj) to E; { Panggil
        subpohon matching}
        Return E;

    For setiap simpul xi yang tidak ada di Mmin(T1,
    T2)
        Add "Delete(Txi)" to E;
    For setiap simpul yj yang tidak ada di Mmin(T1,
    T2)
        Add "Insert(Tyj )" ke E;
    Return E.
```

3. CONTOH PENERAPAN ALGORITMA X-DIFF



Gambar 1. Pohon T1 dan T2 hasil Parsing oleh algoritma X-Diff

Pada gambar di atas, kita mendapatkan *minimum-cost matching*, yaitu M = { (1,1'), (2,2'), (3,3'), (5,4'), (7,7') }. Garis putus-putus menunjukkan bahwa mereka tidak mempunyai nilai yang sama. (6,5') tidak masuk ke dalam himpunan M karena memiliki *signature* yang berbeda. (8,6') juga tidak masuk karena meskipun mereka memiliki *signature* yang sama, namun *parent* mereka (4,6') tidak berada di M.

Kemudian pada tahap selanjutnya kita dapat membangkitkan edit script untuk (T1 → T2). Caranya sangat sederhana, hapus simpul di T1 yang tidak berada di himpunan M, sisipkan simpul di T2 yang tidak berada di himpunan M, dan ubah simpul yang berada di M tapi memiliki nilai yang berbeda. Hasilnya ialah :

E = Delete(6), Delete(8), Delete(4), Insert(5,'(C, β), 2), Insert(6'(B, λ), 3), Update(7, ω)

4. KESIMPULAN

Algoritma X-Diff terbilang cukup mangkus dalam mendeteksi perubahan yang terjadi pada struktur dokumen XML. Algoritma ini bekerja dengan cara membandingkan setiap simpul pada pohon DOM untuk mendeteksi apakah ada perubahan pada dua atau lebih dokumen XML yang sedang dibandingkan.

Algoritma ini dapat menjadi salah satu solusi dalam memfasilitasi kebutuhan akan informasi yang sekarang ini bergerak sedemikian cepatnya.

5. REFERENSI

[1] WestLake, Internet Training, "Introduction to XML Tutorial", Hal. 3-7, 2002.

[2] Yuan Wang, "X-Diff: An Effective Change Detection Algorithm for XML Documents", University of Wisconsin.

[3] Alex Lopez-Ortiz, "diffX: An Algorithm to Detect Changes in Multi-version XML Documents", University of Waterloo.

[4] http://cs.wisc.edu