Aplikasi Algoritma Program Dinamis untuk Mencari Perbedaan File pada Git

Timothy Pratama / 13512032
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512032@std.stei.itb.ac.id

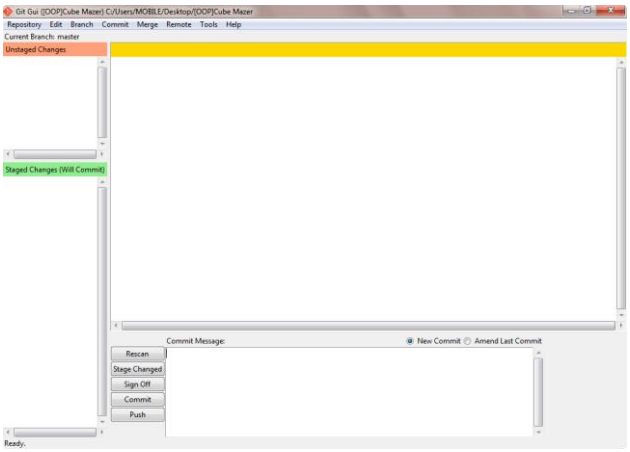
Makalah ini menjelaskan mengenai penggunaan algoritma Program Dinamis pada Git untuk menemukan perbedaan yang ada pada sebuah file setelah dan sebelum di-edit. Teknik yang digunakan adalah dengan cara mencari Longest Common Subsequence dari kedua file tersebut. Subsequence dapat ditemukan dengan cara menghapus beberapa bagian text dari file pertama dan beberapa bagian text dari file kedua sehingga terdapat sebuah sequence yang sama. Sequence yang dicari harus memiliki panjang semaksimal mungkin. Setiap karakter yang dihapus dari file yang lama menunjukkan bahwa file pada file yang baru bagian tersebut tidak ada, biasanya diberi warna merah dan tanda -. Sedangkan untuk file baru, bagian yang dihapus berarti bagian itu baru ditambahkan dan biasanya diberi warna hijau dan tanda +.

Dynamic Programming, Longest Common Subsequence, Version Control System, Git

I. INTRODUCTION

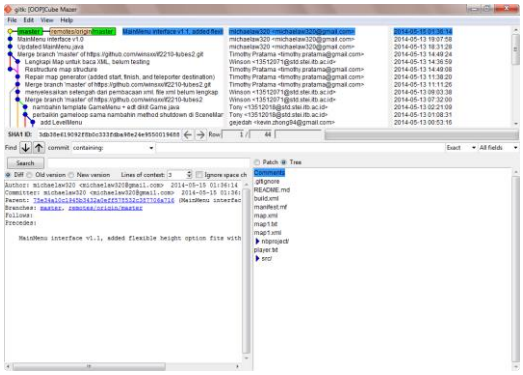
[1]*Version Control System* adalah sebuah sistem yang menyimpan semua perubahan – perubahan yang dilakukan pada sebuah file atau sekumpulan file. Pada setiap perubahan akan diperlihatkan siapa yang mengubah file tersebut dan bagian mana yang diubah. Apabila ternyata setelah bagian file tersebut diubah muncul *bug* / *error*, maka kita dapat melakukan *revert* ke versi sebelumnya dimana file tersebut tidak ada error / bug. Jika file-file tersebut *corrupt*, file tersebut dapat di-*restore* dengan cepat.

[2]Git adalah sebuah *distributed version control system.* Git pertama kali dikembangkan dan diciptakan oleh Linux Torvalds untuk pengembangan kernel Linux pada tahun 2005. Tujuan utama dari Git ini adalah untuk melakukan *versioning control* terhadap sekumpulan file tertentu. Pada git, *versioning control* dapat dilakukan secara *local*. Setiap file dan folder yang ada di dalam sebuah folder akan dibuat *versioning*-nya, sehingga setiap ada perubahan dapat terlihat kapan, oleh siapa, dan bagian mana dari file / folder tersebut yang diubah. Pada dasarnya, git tidak pernah menghapus data dari databasenya ketika sebuah file / folder kita hapus, hanya pada bagian historynya saja dihapus. Ini dapat terlihat dari ukuran folder yang diberi *version control* tidak pernah mengecil, tetapi semakin bertambah besar.



Gambar 1. Tampilan Menu GIT

Pada git, kita dapat melihat file-file apa saja yang terdapat dalam folder kita saat ini, kemudian kita juga dapat melihat *history* dari perubahan-perubahan yang terjadi pada file-file tersebut, *statistic database*. Pada git terdapat sebuah *branch* utama, yaitu *master branch*. Pengguna git dapat membuat branch baru, sehingga dapat melakukan eksperimen terhadap isi dari file file, seperti menambahkan fitur baru dan sebagainya. Pengguna juga dapat menyimpan git ke *remote repository* dengan menggunakan *push*, dan mengambil data dari *remote repository* dengan menggunakan *pull*.



Gambar 2. Tampilan history dari branch

Sedangkan tampilan untuk melakukan *push* dan *pull* adalah seperti ini.

Pada git, setiap kali pengguna akan melakukan *commit*,

maka akan ditampilkan perubahan-perubahan yang telah terjadi pada file. Jika ada bagian yang baru ditambahkan, maka pada bagian samping akan ditandai dengan tanda + dan diberi warna hijau, sedangkan jika bagian tersebut dihilangkan, akan diberi tanda – dan diwarnai merah. Berikut adalah contoh dari perubahan yang terjadi pada sebuah file.

```
@@ -24,24 +24,26 @@ public class LevelMenu extends Scene

    @Override
    public void Initialize()
    {
        nama_klmpk="Cube Mazer";
-       judul="Level Menu";
-       nama_lvl="Indonesian Maze";
-       level=1;
-       height=55;
-       width=170;
+       judul="Level Menu";[redacted]
+       level=0;
+       height=56;
+       width=168;[redacted]
        spacebetween=width/25;
        space=((width)-(spacebetween*10)-10-4)/2;
    }

    @Override
    public void Update()
    {
        if (level<5) level+=1;
-       if (level==2){
+        if (level==1){
+           nama_lvl="Indonesian Maze";
+       }[redacted]
```

Gambar 3. Perubahan pada sebuah file

Untuk menemukan perubahan-perubahan yang terjadi pada sebuah file, salah satu algoritma yang dapat digunakan adalah dengan menggunakan strategi Program Dinamis. Program Dinamis ini akan mencari *Longest Common Subsequence* yang terdapat pada sebuah file lama dan file yang telah dimodifikasi. Caranya adalah dengan cara menghilangkan beberapa bagian dari file lama dan beberapa bagian dari file baru sehingga setelah bagian-bagian itu dihilangkan, terdapat sebuah *sequence* yang sama di kedua file itu dengan panjang semaksimal mungkin.

II. DASAR TEORI

A. Program Dinamis

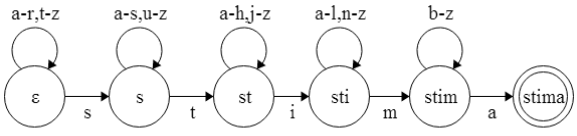
[3]Program dinamis adalah metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan langkah atau tahapan sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan. Salah satu prinsip yang penting dalam program dinamis ini adalah Prinsip Optimalitas. [3]Prinsip ini berbunyi bahwa jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal. Artinya, untuk menentukan solusi pada tahap k, maka dapat dianggap bahwa solusi yang diambil dari tahap pertama sampai tahap k-1 adalah optimal, sehingga tidak perlu menghitung ulang dari tahap pertama lagi untuk menentukan solusi optimal berikutnya. Oleh karena itu, hanya rangkaian keputusan yang optimal yang akan dihasilkan oleh program dinamis ini. Biasanya program dinamis ini digunakan untuk memecahkan masalah optimasi.

Penyelesaian dengan menggunakan program dinamis dapat dilakukan dengan dua cara. Pertama, dengan cara maju (*forward* atau *up-down*). Kedua, dengan cara mundur(*backward* atau *bottom-up*). Kedua cara ini akan menghasilkan hasil yang sama. Dalam menentukan keputusan yang harus diambil, program dinamis akan membuat tabel-tabel yang berisi berbagai kemungkinan yang diambil beserta dengan cost nya. Dari *table* tersebut, program dinamis akan menentukan solusi optimal untuk setiap data pada table itu, kemudian membuat table baru berdasarkan hasil dari table sebelumnya. Sehingga setelah *Cost* optimalnya ditentukan, maka program dinamis akan membangun solusi dari table yang paling terakhir ke table paling awal. Dengan demikian, maka program dinamis dapat menghasilkan solusi yang optimal (bisa lebih dari satu). Program dinamis ini menggunakan rekursif dalam menyelesaikan permasalahannya.

B. *Longest Common* Sub*sequence* (LCS)

[4]*Longest Common* Sub*sequence* adalah salah satu permasalahan yang dapat diselesaikan dengan menggunakan Program Dinamis. Sebuah *subsequence* adalah sebuah *sequence* of character yang merupakan bagian dari sebuah *string*. Letak sub*sequence* ini tidak harus bersebelahan seperti pada *substring*, tetapi yang penting adalah urutan kemunculan *subsequence* ini harus sama dengan kemunculan pada stringnya. Sebagai contoh, misalkan terdapat sebuah string berisi strategi_algoritma. Stima dapat dikatakan sebagai sebuah *subsequence* dari *string* strategi_algoritma ini, karena jika diperhatikan, **st**rategi_algorit**ma** mengandung *subsequence* STIMA (bagian yang dicetak tebal). Contoh lain yaitu terdapat sebuah *string* strategi_algoritma. Stmik bukan *subsequence* dari strategi_algoritma, karena pada *string* strategi_algoritma tidak ada huruf k.

[4]Salah satu cara yang dapat digunakan untuk memeriksa apakah sebuah *sequence* merupakan *subsequence* dari sebuah string atau bukan adalah dengan menggunakan *finite automata*. Misalkan untuk persoalan di atas, untuk memeriksa apakah stima merupakan *subsequence* dari strategi_algoritma dapat digunakan *finite automata* di bawah ini.



Gambar 4. *Finite State Automata* untuk mencari *subsequence*

[4]Longest Common Subsequence adalah sebuah permasalahan dimana kita harus menemukan subsequence

terpanjang dari dua buah string. Longest Common Subsequence ini dapat digunakan dalam beberapa bidang, seperti biologi *molecular*, perbandingan dua buah file, dan *screen redisplay*.

III. Analisis Algoritma Program Dinamis untuk Mencari Perbedaan File pada Git

[5]Untuk mencari Longest Common Subsequence dari dua buah string yang berbeda, maka akan diselesaikan secara rekursif dengan menggunakan algoritma Program Dinamis.

Misalkan terdapat dua buah string, yaitu a dan b. LCS(i,j) berarti Longest Common Subsequence dari index ke 0 sampai ke i untuk string a dan Longest Common Subsequence dari index ke 0 sampai ke j untuk string b. Definisi dari fungsi rekursifnya adalah sebagai berikut.

$$LCS(i,j) = \begin{cases} 0, & i = 0 \; atau \; j = 0 \\ LCS(i-1, j-1), & a[i] = b[j] \\ \max(LCS(i-1,j), (LCS(i,j-1)), & a[i] \neq b[j] \end{cases}$$

LCS(i,j) akan menghasilkan string kosong pada saat i = 0 atau j = 0 karena subsequence dari sebuah string dengan string kosong adalah tidak ada. Sedangkan pada saat string a[i] sama dengan string b[j], maka fungsi LCS ini akan mencari Longest Common Subsequence berikutnya dari indeks ke (i-1) dan (j-1). Sedangkan jika berbeda, maka fungsi ini akan mencari LCS terpanjang dari a[i-1], b[j] dan a[i], b[j-1].

Untuk menyimpan hasil perhitungan dari fungsi LCS ini digunakan sebuah array dua dimensi berukuran i x j. Array ini akan menyimpan hasil dari LCS(i,j). Dari array ini baru kemudian ditentukan solusinya, yaitu sebuah Longest Common Subsequence dari kedua string tersebut.

Pseudocode untuk fungsi LCS(i,j) yang bersifat rekursif ini adalah sebagai berikut.

```
function LCS(i : integer, j : integer) -> string
{
    {kamus}
    char1 : character
    char2 : character

    {a dan b adalah 2 buah string yang berbeda}
    char1 = a[i]
    char2 = b[j]

    if(i==0 || j==0)
    {
        return string kosong
    }
    else if (char1 == char2)
    {
        return char1 dikonkat dengan LCS(i-1,j-1)
    }
    else
    {
        return max(LCS(i-1,j),LCS(i,j-1))
    }
}
```

Gambar 5. Pseudocode fungsi LCS rekursif

Fungsi max pada pseudocode di atas akan mengembalikan sebuah string yang memiliki panjang terbesar di antara kedua string yang dibandingkan. Fungsi LCS rekursif ini akan digunakan untuk mengisi table LCS.

Adapun fungsi LCS secara keseluruhannya adalah sebagai berikut.

```
function FindLCS() -> array of string
{
    {Kamus}
    i : integer
    j : integer

    {a dan b adalah 2 buah string yang berbeda}
    LCS : array of string[a.length()][b.length()
          {digunakan untuk menampung solusi}

    for(i=0; i<=a.length(); i++)
    {
        for(j=0; j<=b.length(); b++)
        {
            LCS[i][j] = LCS(i,j);
        }
    }
    return semua string dengan panjang maksimum pada tabel
}
```

Gambar 6. Pseudocode Fungsi LCS

Misalkan kita memiliki sebuah repository local yang diurus oleh git. Pada saat pertama kali kita melakukan commit, maka setiap file akan dimasukkan ke dalam database untuk dicatat track historynya, yaitu jika terjadi perubahan isi file, penghapusan file, atau penambahan file baru. Misalkan kita mempunya sebuah file berisi string "BasisData". Kemudian, file tersebut kita ubah isinya menjadi "Database". Pada saat akan melakukan commit, maka git akan memeriksa bahwa file dengan nama tersebut ternyata sudah ada, dan git akan langsung membandingkan isi kedua file. Jika diterapkan algoritma LCS tadi, maka langkah-langkah yang dilakukan untuk menemukan solusi adalah sebagai berikut.

	0	D	A	T	A	B	A	S	E
0									
B									
A									
S									
I									
S									
D									
A									
T									
A									

Pertama, program akan membuat sebuah table yang berukuran panjang string "BasisData" dikalikan dengan panjang string "Database", yaitu sebanyak 10 * 9 = 90.

	0	D	A	T	A	B	A	S	E
0	0	0	0	0	0	0	0	0	0
B									
A									
S									
I									

S									
D									
A									
T									
A									

Program Dinamis akan mulai mengisi table dari baris 0 kolom 0 sampai baris 10 kolom 9 secara per baris pada setiap kali iterasinya. Pada iterasi pertama, semua baris 0 akan berisi 0, yaitu string kosong, sesuai dengan definisi bahwa LCS(i,0) atau LCS(0,j) bernilai 0.

	0	D	A	T	A	B	A	S	E
0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	B	0	0	0
A									
S									
I									
S									
D									
A									
T									
A									

Kemudian, tabel akan masuk ke iterasi ke 2 dan mulai mengisi baris ke 2. Kolom pertama tabel juga akan bernilai 0, karena *Longest Common Subsequence* dari sebuah string dengan string kosong adalah 0(string kosong). Untuk setiap huruf yang tidak sama, maka program akan mengisi bagian sel tersebut dengan *subsequence* terpanjang dari isi tabel yang ada di bagian atas dan kirinya. Jika sama, seperti pada huruf B, maka program akan mengisi sel itu dengan huruf tersebut di konkat dengan huruf yang ada di kiri atasnya. Prosedur ini akan dilakukan secara terus-menerus hingga pada akhirnya semua tabel sudah terisi. Saat semua tabel terisi, maka program dapat melakukan konstruksi solusi dari data yang ada pada tabel tersebut.

Iterasi ke 3:

	0	D	A	T	A	B	A	S	E
0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	B	0	0	0
A	0	0	A	A	A	A/B	BA	BA	BA
S									
I									
S									
D									
A									
T									
A									

Iterasi ke 4:

	0	D	A	T	A	B	A	S	E
0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	B	0	0	0
A	0	0	A	A	A	A/B	BA	A	A
S	0	0	A	A	A	A/B	BA	BAS	BAS

I									
S									
D									
A									
T									
A									

Iterasi ke 5:

	0	D	A	T	A	B	A	S	E
0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	B	0	0	0
A	0	0	A	A	A	A/B	BA	A	A
S	0	0	A	A	A	A/B	BA	BAS	BAS
I	0	0	A	A	A	A/B	BA	BAS	BAS
S									
D									
A									
T									
A									

Iterasi ke-6

	0	D	A	T	A	B	A	S	E
0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	B	0	0	0
A	0	0	A	A	A	A/B	BA	A	A
S	0	0	A	A	A	A/B	BA	BAS	BAS
I	0	0	A	A	A	A/B	BA	BAS	BAS
S	0	0	A	A	A	A/B	BA	BAS	BAS
D									
A									
T									
A									

Iterasi ke 7

	0	D	A	T	A	B	A	S	E
0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	B	0	0	0
A	0	0	A	A	A	A/B	BA	A	A
S	0	0	A	A	A	A/B	BA	BAS	BAS
I	0	0	A	A	A	A/B	BA	BAS	BAS
S	0	0	A	A	A	A/B	BA	BAS	BAS
D	0	D	A/D	A/D	A/D	A/D/B	BA	BAS	BAS
A									
T									
A									

Iterasi ke 8

	0	D	A	T	A	B	A	S	E
0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	B	0	0	0
A	0	0	A	A	A	A/B	BA	A	A

S	0	0	A	A	A	A/B	BA	BAS	BAS
I	0	0	A	A	A	A/B	BA	BAS	BAS
S	0	0	A	A	A	A/B	BA	BAS	BAS
D	0	D	A/D	A/D	A/D	A/D/B	BA	BAS	BAS
A	0	D	DA	DA	AA/DA	AA/DA	AA/DA/BA	BAS	BAS
T									
A									

Iterasi ke 9

	0	D	A	T	A	B	A	S	E
0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	B	0	0	0
A	0	0	A	A	A	A/B	BA	A	A
S	0	0	A	A	A	A/B	BA	BAS	BAS
I	0	0	A	A	A	A/B	BA	BAS	BAS
S	0	0	A	A	A	A/B	BA	BAS	BAS
D	0	D	A/D	A/D	A/D	A/D/B	BA	BAS	BAS
A	0	D	DA	DA	AA/DA	AA/DA	AA/DA/BA	BAS	BAS
T	0	D	DA	DAT	DAT	DAT	DAT	BAS/DAT	BAS/DAT
A									

Iterasi ke 10

	0	D	A	T	A	B	A	S	E
0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	B	0	0	0
A	0	0	A	A	A	A/B	BA	A	A
S	0	0	A	A	A	A/B	BA	BAS	BAS
I	0	0	A	A	A	A/B	BA	BAS	BAS
S	0	0	A	A	A	A/B	BA	BAS	BAS
D	0	D	A/D	A/D	A/D	A/D/B	BA	BAS	BAS
A	0	D	DA	DA	AA/DA	AA/DA	AA/DA/BA	BAS	BAS
T	0	D	DA	DAT	DAT	DAT	DAT	BAS/DAT	BAS/DAT
A	0	D	DA	DAT	DATA	DATA	DATA	DATA	**DATA**

Setelah iterasi ke 10, tabel tadi pun menjadi lengkap. Salah satu hal yang penting adalah bahwa pada setiap iterasi, program dinamis akan menggunakan hasil perhitungan dari iterasi sebelumnya untuk menghasilkan sebuah keputusan yang optimal, kompleksitas waktu untuk program dinamis pun tergolong cepat, yaitu O(mn) dengan m adalah panjang string a dan n adalah panjang string b. Dari tabel di atas dapat dilihat bahwa Longest Common Subsequence dari BasisData dengan Database adalah Data. Dengan diketahuinya informasi ini, maka Git dapat menentukan bahwa dari string pertama, yaitu BasisData, huruf yang dihapus adalah Basis. Sehingga git akan memberikan tanda pada string pertama bahwa Basis dihapus dari string pertama. Kemudian, pada string kedua yaitu database, huruf yang ditambahkan adala base. Sehingga git juga akan memberi tanda bahwa dari string ke 2, ditambah dengan huruf base. Dengan demikian, maka git mampu untuk mengetahui perbedaan apa yang ada pada dua buah file yang berbeda beserta dengan dimana letak perbedaannya itu. Dari sini dapat terlihat bahwa Algoritma Program Dinamis ini dapat digunakan untuk menentukan *Longest Common Subsequence* secara optimal.

Contoh kasus lain yaitu misalkan kita mempunyai string StrategiAlgoritma dengan SistemOperasi. Untuk menentukan *Longest Common Subsequence* di atas, diselesaikan dengan menggunakan cara yang sama. Tabel LCS nya pun seperti di bawah ini.

	S	**I**	**S**	**T**	**E**	**M**	**O**	**P**	**E**	**R**	**A**	**S**	**I**
S	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	T	T	T	T	T	T	T	T	T	T
R	0	0	0	T	T	T	T	T	T	TR	TR	TR	TR
A	0	0	0	T	T	T	T	T	T	TR	TRA	TRA	TRA
T	0	0	0	T	T	T	T	T	T	TR	TRA	TRA	TRA
E	0	0	0	T	TE	TE	TE	TE	TE	TE/TR	TRA	TRA	TRA
G	0	0	0	T	TE	TE	TE	TE	TE	TE/TR	TRA	TRA	TRA
I	0	I	I	I/T	TE	TE	TE	TE	TE	TE/TR	TRA	TRA	TRAI
A	0	I	I	I/T	TE	TE	TE	TE	TE	TE/TR	TEA/TRA	TEA/TRA	TRAI
L	0	I	I	I/T	TE	TE	TE	TE	TE	TE/TR	TEA/TRA	TEA/TRA	TRAI
G	0	I	I	I/T	TE	TE	TE	TE	TE	TE/TR	TEA/TRA	TEA/TRA	TRAI
O	0	I	I	I/T	TE	TE	TEO	TEO	TEO	TEO	TEA/TRA/TEO	TEA/TRA/TEO	TRAI
R	0	I	I	I/T	TE	TE	TEO	TEO	TEO	TEOR	TEOR	TEOR	TEOR/TRAI
I	0	I	I	I/T	TE	TE	TEO	TEO	TEO	TEOR	TEOR	TEOR	TEORI
T	0	I	I	I	IT/TE	IT/TE	TEO	TEO	TEO	TEOR	TEOR	TEOR	TEORI
M	0	I	I	I	IT/TE	ITM/TEM	ITM/TEM/TEO	ITM/TEM/TEO	ITM/TEM/TEO	TEOR	TEOR	TEOR	TEORI
A	0	I	I	I	IT/TE	ITM/TEM	ITM/TEM/TEO	ITM/TEM/TEO	ITM/TEM/TEO	TEOR	TEORA	TEORA	TEORA/TEORI

Dari String StrategiAlgoritma dengan SistemOperasi,

LCS nya adalah TEORA atau TEORI. Walaupun data yang harus dihitung cukup besar, dengan menggunakan Program Dinamis ini maka hasil dapat diperoleh dalam waktu yang relative cepat, sedangkan jika menggunakan *brute force* maka waktu yang dibutuhkan untuk mendapatkan hasil akan jauh lebih lama.

Untuk membedakan dua buah file yang berbeda, git dapat menggunakan algoritma program dnamis ini karena dapat menghasilkan solusi dalam waktu yang cepat dan jumlah ruang yang relative sedikit jika dibandingkan dengan brute force.

IV. KESIMPULAN

Algoritma Program Dinamis dapat memberikan solusi dari permasalahan Longest Common Subsequence dengan kompleksitas waktu yang rendah dan membutuhkan ruang yang relative kecil bila dibandingkan denga brute force, sehingga dapat digunakan pada git untuk mencari perbedaan yang ada pada dua buah file. Untuk menemukan perbedaan itu, pertama git akan memeriksa apakah file yang akan dicommit tersebut sudah pernah di track atau belum. Jika belum, maka git akan melakukan track terhadap file itu. Jika sudah, maka git akan memeriksa isi dari file tersebut dan memeriksa apakah terdapat perbedaan atau tidak. Caranya adalah dengan mencari longest common subsequence dari kedua file yang dibandingkan. Dari file pertama, jika ada bagian yang bukan common subsequence, berarti bagian itu dihilangkan dari file pertama. Sedangkan untuk file kedua, jika ada bagian yang tidak ada di common subsequence, berarti bagian itu adalah bagian yang baru ditambahkan. Bagian yang dihilangkan ditandai dengan warna merah dan diberi tanda – , sedangkan bagian yang baru ditambah akan diberi warna hijau dan diberi tanda +.

REFERENCES

[1] http://git-scm.com/book/en/Getting-Started-About-Version-Control (diakses pada tanggal 15 Mei 2014 pukul 08:12 WIB)

[2] http://git-scm.com/book/en/Getting-Started-A-Short-History-of-Git (diakses pada tanggal 15 Mei 2014 pukul 10:05 WIB)

[3] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algoritma. 2009. Penerbit Informatika : Bandung

[4] http://www.ics.uci.edu/~eppstein/161/960229.html (diakses pada tanggal 17 Mei 2014 pukul 15:12 WIB)

[5] http://www.youtube.com/watch?v=wJ-rP9hJXO0 (diakses pada tanggal 17 Mei 2014 pukul 18:00 WIB) video kuliah oleh S. Saurabh, B.Tech dari IIT dan MS, USA.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Mei 2014

Timothy Pratama / 13512032