IMPLEMENTASI PENGGUNAAN SISTEM APLIKASI WEB PDF PARSER UNTUK MENAMPILKAN INFORMASI ISI DOKUMEN

Yulianto[1)], Fifit Alfiah[2)], Andy Nova Wijaya[3)], Muh. Rizal Ramadhan[4)], Leo Kumoro Sakti[5)], Mubtasir[6)], Abdul Mukti[7)]

[1), 2),3),4),5),6),7)] Teknik Informatika Perguruan Tinggi Raharja
Jl Jendral Sudirman No 40 Modern Cikokol, Tangerang 15117
Email : yulianto@raharja.info[1)], fifitalfiah@raharja.info[2)], andy.nova@raharja.info[3)], muh.rizal@raharja.info[4)], leo@raharja.info[5)], mubtasir@raharja.info[6)], abdul.mukti@raharja,info[7)]

Abstrak

Berdasarkan ilmu komputer Kercerdasan Buatan, Terdapat Pengolahan bahasa komunikasi untuk komputer dan manusia yaitu Natural Processing Languge (NLP). NLP adalah ilmu dalam kecerdasan buatan yang mempelajari pengolahan tata bahasa, supaya manusia dapat memahami apa yang ditampilkan oleh sebuah sistem komputer. Dalam menampilkan informasi NLP memiliki teknik khusus dimana sebuah sistem komputer mampu menghasilkan informasi atau output isi atau content dari sebuah file dokumen adalah parsing. Parsing adalah suatu cara memecah-mecah suatu rangkaian masukan (misalnya dari berkas atau keyboard) yang akan menghasilkan suatu pohon uraian (parse tree) yang akan digunakan pada tahap kompilasi berikutnya yaitu analisis semantic.

Parsing dalam penelitian ini adalah menggunakan jenis file .pdf dengan begitu penelitian ini akan menghasilkan informasi atau output dari sebuah file dokumen yang di parse atau di uraikan dengan aplikasi web yang kami buat sendiri. Untuk penggunaannya penelitian ini jg membahas beberapa metode parsing yang terkait dalam sistem aplikasi pdf parse. Implementasi penggunan aplikasi web pdf parse ini menghasilkan atau menampilkan isi dokumen parse dari sistem aplikasi yang dibuat, karna dengan adanya pdf parse sangat membantu untuk mengetahui segala info mengenai informasi dan isi dokumen yang dibutuhkan oleh para user .

Kata kunci:*Analisis , Parse/parsing, dokumen, pdf.*

1. Pendahuluan

Komunikasi adalah salah satu hal paling penting yang dibutuhkan manusia sebagai makhluk sosial, dalam komputer terdapat komunikasi atau interaksi antara komputer dengan manusia di sebut dengan bahasa *Natural Language Processing* (NLP) atau pengolahan bahasa alami dalam bidang *Artificial intellegent* yang memperlajari komunikasi antara manusia dengan komputer melalui bahasa alami.

Dalam ilmu komputer juga dikenal suatu ilmu yang disebut dengan automata . Teori automata dapat digunakan sebagai salah satu solusi dalam masalah pengolahan bahasa natural, yaitu dengan menggunakan pohon penurunan untuk menghasilkan *parsing* dari suatu kalimat. Banyak penelitian Bahasa natural dan bergabai macam buku *linguistik* yang menggunakan pohon parsing. Ratnapharki[1], Utami[1], dan beberapa penelitian bahasa natural lainnya kerap kali menggunakan pohon parsing sebagai pemodelan struktur kalimat bahasa natural. Sedangkan sebelum dapat membentuk pohon penurunan, *Shift-Reduce Parser* digunakan oleh Ratnapharki[1], Nivre[1] dan Sagae K, Livre A[1] sebagai pengolah leksikal input suatu kelompok kata.

Natural Language Processing (NLP) merupakan salah satu cabang ilmu AI yang berfokus pada pengolahan bahasa natural. Bahasa natural adalah bahasa yang secara umum digunakan oleh manusia dalam berkomunikasi satu sama lain. Bahasa yang diterima oleh komputer butuh untuk diproses dan dipahami terlebih dahulu supaya maksud dari user bisa dipahami dengan baik oleh komputer.

Parsing adalah suatu cara memecah-mecah suatu rangkaian masukan (misalnya dari berkas atau *keyboard*) yang akan menghasilkan suatu pohon uraian (*parse tree*) yang akan digunakan pada tahap kompilasi berikutnya yaitu analisis semantik.

Didalam komputasi, *parser* adalah salah satu komponen dalam sebuah interpreter atau kompiler yang bertugas memeriksa sintaks secara benar serta membangun struktur data yang tersirat dalam token masukan.

Contoh teknologi *parser* didalam kehidupan sehari-hari yang sering kita temui yaitu google terjemah, dimana kosakata yang kita inputkan dianalisis menjadi tata bahasa yang lebih baik. Namun dalam makalah kali ini, penulis tidak akan membahas mengenai contoh *parser* tersebut, melainkan mengenai PDF *parser*.

Karena PDF *parser* adalah sebuah program yang bertujuan untuk mengurai dokumen PDF, menganalisis dokumen PDF, menangani dokumen PDF yang rusak, serta memeriksa struktur dokumen PDF dan kontennya. Jika dalam bentuk umum *parser* akan menguraikan pola tata bahasa menjadi lebih baik dan menguraikan struktur serta kalimat yang *error, p*ada PDF parser ini kita akan menguraikan file dokumen dalam bentuk ekstensi .pdf

Metode *parsing*

Analisis sintaksis (atau proses *parsing*) berguna untuk memeriksa urutan kemunculan token. Pada proses ini, hal yang perlu diperhatikan adalah [2]:

a. Kebutuhan waktu eksekusi
b. Penanganan kesalahan
c. Penanganan kode

Rumusan masalah yang menjadi inti dari penelitian ini yaitu apakah metode PDF *parser* dapat menguraikan struktur file PDF menjadi dokumen yang baik atau tidak serta aplikasi pdf *parser* yang mana yang menghasilkan output paling lengkap dan baik dalam mengurai isi file dokumen pdf yang sesuai dengan kebutuhan user.

2. Pembahasan

Algoritma *Parsing*

Parsing dapat terjadi karena kombinasi dari semua pohon (*tree*) dalam lingkup hutan (*forest*) sampai setiap pohon (*tree*) dari akar (*root*) S terproduksi, atautidak ada lagi operasi yang mungkin. Dalam formalisasi TFG, pohon (*tree*) dikombinasikan melalui dua operasi dasar, yaitu: subtitusi dan furkasi. Subtitusi adalah penggantian node yang belum terdefinisikan (*pre-defined*), berfungsi sebagai pemegang tempat dengan pohon (*tree*) yang kompatibel pada kategori yang sama (cth. *Tree* N) .

Sebagai contoh, furkasi dari pohon (*tree*) pengganti tipe N menggantikan hampir seluruh bagian kanan dari N daun (*leave*) pada pohon (*tree*) target. Dapat kita lihat, secara umum metode furkasi meliputi pengubah (*modifier*) seperti adjective (N* *root*), yang dapat ditambahkan informasi semantic kedalam pohon yang dimodifikasi selama proses furkasi. Algoritma *parsing* pada contoh kali ini merupakan algoritma *parsing* buttom-up yang disederhanakan.

Metoda *Parsing*

Metode-metode *parsing* yang dibahas berikut khusus digunakan dalam NLP. Sebelumnya perlu diketahui arti dari istilah constituent , yaitu unsur-unsur pembentuk kalimat yang dapat berdiri sendiri, contohnya *noun phrase*, *verb phrasedan* sebagainya; dan istilah *parser* yaitu programyang melakukan proses *parsing*.

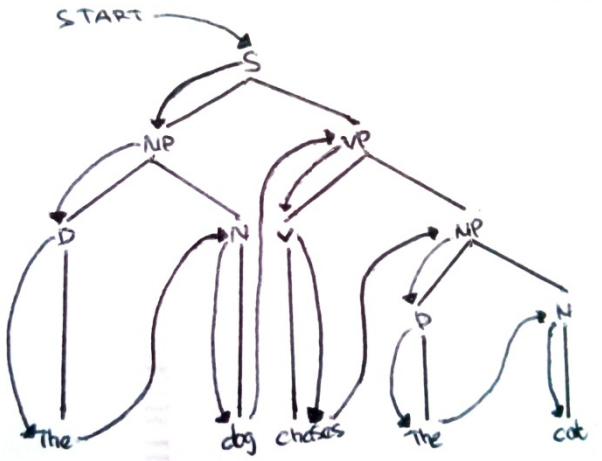
1. Top-down Parsing

Top-down parser bekerja dengan cara menguraikan sebuah kalimat mulai dari constituent yang terbesar yaitu sampai menjadi *constituent* yang terkecil. Hal ini dilakukan terus-menerus sampai semua komponen yang dihasilkan ialah *constituent* terkecil dalam kalimat, yaitu kata.

Sebagai contoh, dengan menggunakan contoh *grammar* di atas, dapat dilakukan proses *top-down parsing* untuk kalimat "*The dog chased the cat*"yang ditunjukkan pada gambar 2.1. Dari gambar ini terlihat bahwa *top-down parser* menelusuri setiap node pada *parse tree* secara *pre-order.*

Beberapa metode *parsing* yang bekerja secara *topdown* ialah:

- *Top-down parser* biasa
- *Recursive-descent parser*
- *Transition-network parser*
- *Chart parser*



Gambar 1. Cara Kerja*Top-down Parser*[3]

Top-down parser dapat diimplementasikan dengan berbagai bahasa pemrograman, namun akan lebih baik jika digunakan *declarative language* seperti Prolog atau LISP. Hal ini disebabkan oleh karena pada dasarnya proses parsing ialah proses *searching* yang dilakukan secara rekursif dan *backtracking*, dimana proses ini sudah tersedia secara otomatis dalam bahasa Prolog.

Dengan demikian *parser* yang ditulis dalam Prolog atau bahasa deklaratif lainnya akan menjadi jauh lebih sederhana daripada parser yang dibuat dalam bahasa prosedural biasanya seperti Pascal, C dan sebagainya.

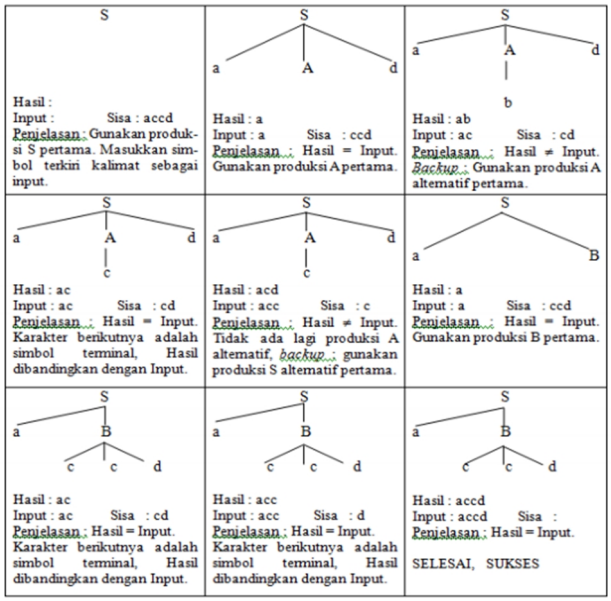
Ada 2 kelas metoda *parsing top-down*, yaitu kelas metoda dengan *backup* dan kelas metoda tanpa *backup*. Contoh metoda kelas dengan *backup* adalah metoda Brute-Force, sedangkan contoh metoda kelas tanpa *backup* adalah metoda *recursive descent*.

a. Metoda Brute-Force

Kelas metoda dengan *backup*, termasuk metoda Brute-Force, adalah kelas metoda *parsing* yang menggunakan produksi alternatif, jika ada, ketika hasil penggunaan sebuah produksi tidak sesuai dengan simbol input. Penggunaan produksi sesuai dengan nomor urut produksi [2].

Contoh :

Diberikan *grammar* G = {S →aAd|aB, A →b|c, B →ccd|ddc}.Gunakan metoda BruteForce untuk melakukan analisis sintaks terhadap kalimat x = accd.



Gambar 2. Metoda Brute-Force [4]

Metoda Brute-Force tidak dapat menggunakan *grammar* rekursi kiri, yaitu *grammar* yang mengandung produksi rekursi kiri *(left recursion)* : A →A∝. Produksi rekursi kiri akan menyebabkan *parsing* mengalami *looping* tak hingga.

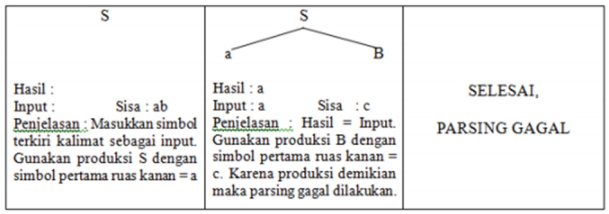
b. Metoda *Recursive-Descent* [2]

- Kelas metoda tanpa *backup*, termasuk metoda *recursive descent*, adalah kelas metoda parsing yang tidak menggunakan produksi alternatif ketika hasil akibat penggunaan sebuah produksi tidak sesuai dengan simbol input. Jika produksi A mempunyai dua buah ruas kanan atau lebih maka produksi yang dipilih untuk digunakan adalah produksi dengan simbol pertama ruas kanannya sama dengan input yang sedang dibaca. Jika tidak ada produksi yang demikian maka dikatakan bahwa parsing tidak dapat dilakukan.

- Ketentuan produksi yang digunakan metoda *recursive descent* adalah : Jika terdapat dua atau lebih produksi dengan ruas kiri yang sama maka karakter pertama dari semua ruas kanan produksi tersebut tidak boleh sama. Ketentuan ini tidak melarang adanya produksi yang bersifat rekursi kiri.

Contoh :

Diketahui *grammar* G = {S → aB A, A → a, B → b d}. Gunakan metoda recursive descent untuk melakukan analisis sintaks terhadap kalimat x= ac.

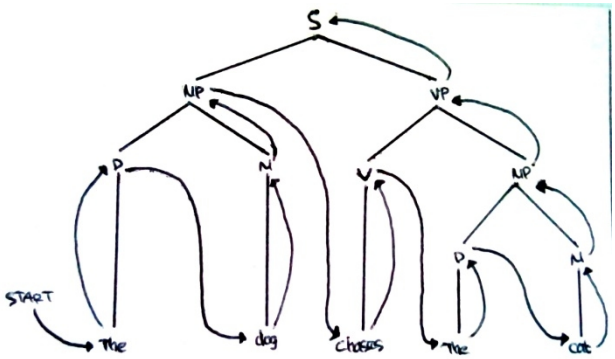


Gambar 3. Metoda *Recursive-Descent* [4]

2. *Bottom-Up Parser*

Bottom-up parser bekerja dengan cara mengambil satu demi satu kata dari kalimat yang diberikan, untuk dirangkaikan menjadi constituent yang lebih besar. Hal ini dilakukan terus-menerus sampai constituent yang terbentuk ialah *sentence* atau kalimat.

Dengan demikian metode *bottom-up* bekerja dengan cara yang terbalik dari *top-down*. Cara kerja *bottom-up parser* ditunjukkan pada gambar 2.4. [3]



Gambar 4. Cara Kerja *Bottom-up Parser* [3]

Metode *parsing* yang bekerja secara bottom-up antara lain ialah *bottom-up parser* biasa dan *shift-reduce parser*. Program 2 menunjukkan contoh implementasi *bottom-up parser* biasa dalam bahasa Turbo Prolog.

Perhatikan bahwa parser ini tidak membedakan antara rule (*grammar*) dan *word (lexicon)* sehingga cara kerjanya sangat sederhana namun sangat "bodoh" karena akan terus mengulang-ulang kesalahan yang sama.

Kesederhanaan metode ini terletak pada predikat untuk *parsing*, yaitu *parse* yang hanya memiliki sebuah argumen. Argumen ini berisi kalimat yang akan diparse dalam bentuk list dari symbol. Kata-kata dari input kalimat akan dirangkaikan sambil mencari aturan yang

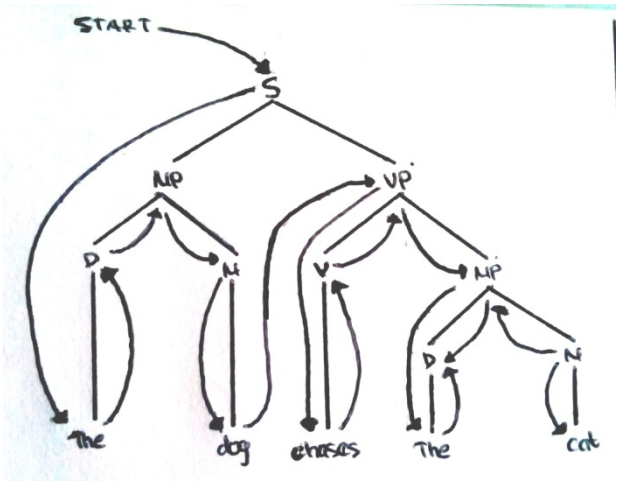
lebih luas, sampai tinggal sebuah simbol saja dalam list, yaitu s.

3. Gabungan *Top-Down* dan *Bottom-Up Parsing*

   Baik *top-down parsing* mapun *bottom-up parsing* memiliki kekurangan dan kelebihannya masing-masing. Metode *top-down* mampu menangani *grammar* dengan *empty production* (misalnya d → 0) namun tidak dapat menangani *grammar* dengan *left recursion* (misalnya np → np conj np). Sedangkan metode *bottom-up* dapat menangani *left recursion* namun tidak dapat menangani *empty production*.[3]

   Dengan demikian metode *parsing* yang terbaik ialah metode yang dapat menggabungkan *top-down* dan *bottom-up parsing*. Ada beberapa metode yang dikembangkan yang menggabungkan kedua metode ini, di antaranya ialah *left-corner parsing* serta Earley's *parsing*. [3] Cara kerja *left-corner parsing* ialah dengan mula-mula menerima sebuah kata, menentukan jenis constituent apa yang dimulai dengan jenis kata tersebut, kemudian melakukan proses *parsing* terhadap sisa dari constituent tersebut secara *top-down*.

   Dengan demikian proses *parsing* dimulai secara *bottom-up* dan diakhiri secara *top-down*. Dan untuk alur kerjanya ditunjukkan pada gambar 2.5. [3]



Gambar 5. Cara Kerja *Left-Corner Parser* [3]

Proses Pembuatan Aplikasi PDF Parser

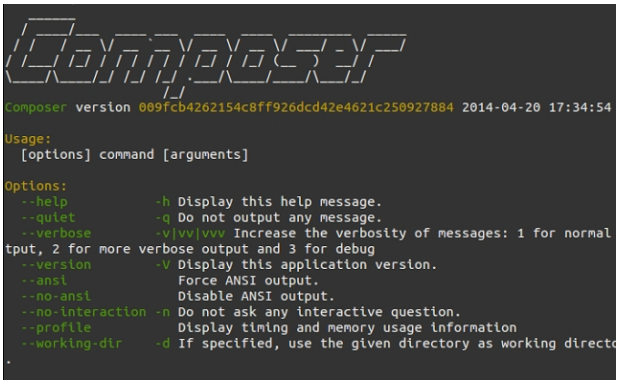
Aplikasi PDF Parse yang kami buat adalah dalam bentuk aplikasi web, dengan menggunakan bahasa pemrograman php dan database mysql sebagi media pemnyimpanan file yang telah diupload serta menggunakan composer sebagai package untuk menjalankan *parser* itu sendiri.

Composer adalah *depedency manager* untuk php. Artinya :

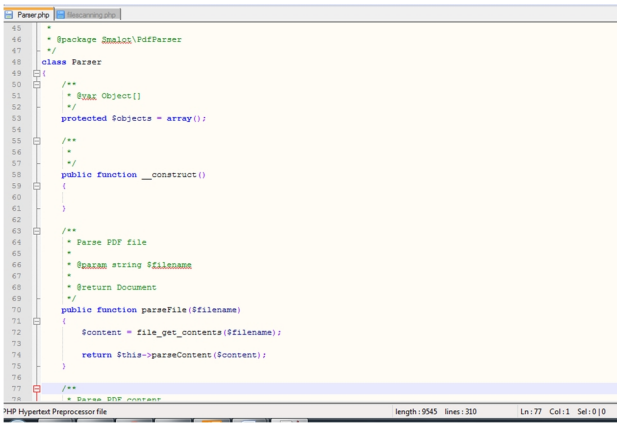
1. Composer bisa meng*install package* yang dibutuhkan
2. Composer bisa meng*update package* yang memiliki *release* terbaru
3. Composer bisa menghapus *package* yang sudah tidak diperlukan

Selain sebagai *dependensi manager*, Composer memiliki dua peran penting lain [5]:

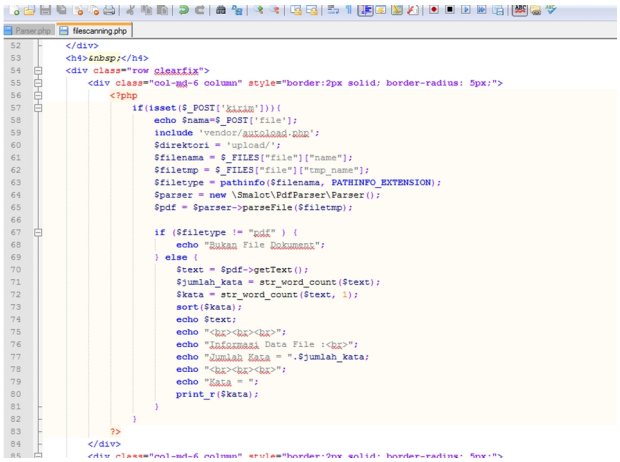
1. Memungkinkan kita mereproduksi environment aplikasi yang sama di semua mesin yang kita pakai.
2. Memungkinkan kita melakukan *automatisasi* dalam *lifecycle development*.



Gambar 6. Tampilan Composer Setelah Ter-*install*



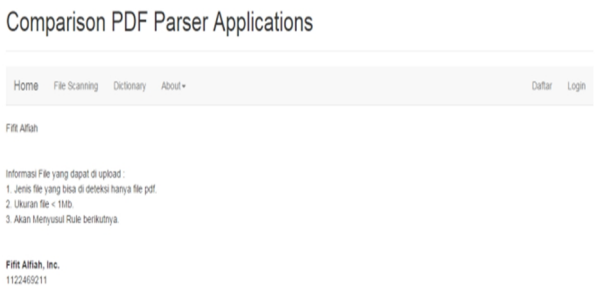
Gambar 7. *Source Code* Fungsi *Parser* yang digunakan dalam aplikasi pdf *parser*



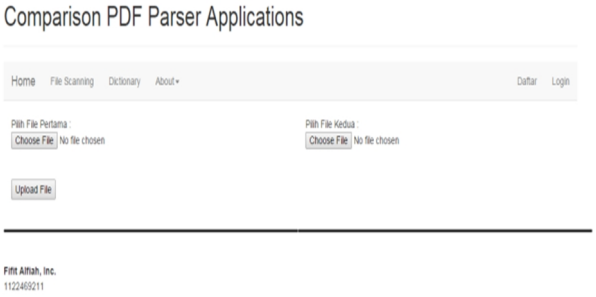
Gambar 8. *Source Code* File *Scanning* yang menghasilkan *output* dari file pdf yang di *submit*

Implemetasi Pengunaan Aplikasi PDF Parser

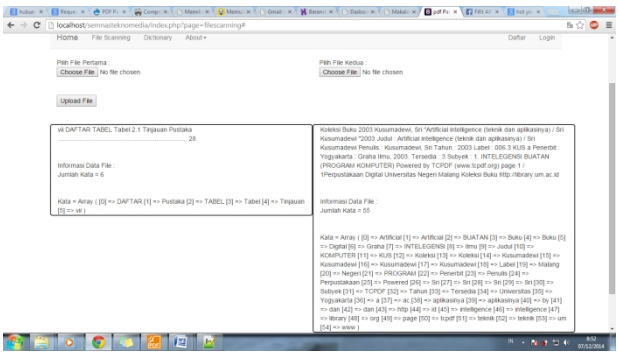
Aplikasi PDF *Parser* ini akan menghasilkan atau menguraikan isi dari file dokumen yang di lakukan scanning.



Gambar 9. Tampilan awal dari aplikasi pdf *parser*



Gambar 10. Tampilan *input* dan *submit* file pdf pada aplikasi pdf *parser*



Gambar 11. Tampilan hasil atau *output* dari file pdf yang *scanning parse*

Hasil atau output dari aplikasi pdf parser ini sebatas menampilkan keseluruhan isi dari file dokumen dan untuk perhitungan jumlah kata benar-benar hanya kata yang di hitung, karena untuk spasi dan tandabaca lainnya sudah dihilangkan agar tidak terhitung. Serta untuk tapilannya juga kami menampilakan urutan kata sesuai dengan abjad (*descending*) menggunakan array, sehingga mudah untuk menghitung jumlah kata yang sama.

3. Kesimpulan

Dari isi penelitian yang sudah di bahas bisa kita ambil kesimpulannya, sebagai berikut :

1. Metode *Parsing* yang terbaik adalah metode gabungan *to-dwon parser* dan *top-up parser* atau *left-corner parser* karena mengambil setiap kelebihan dari metode *to-dwon parser* dan *top-up parser*.
2. Aplikasi pdf parser sederhana yang kami buat mampu menguraikan atau *parsing* isi dari dokumen dengan baik dan menampilkan jumlah kata dan mengurutkan isi kalimat secara berurutan menggunkan *array* dari isi kalimat dokumen pdf yang di *scanning*.

Daftar Pustaka

[1] Parasta, Julianisya Tri, "Penguraian Kata Pada Kalimat Dalam Bahasa Komering Rasuan Berdasarkan Kaidah Bahasa Indonesia Menggunakan Teori Automata" Universitas Sriwijaya, 2014.
[2] Utdirartatmo, Firrar, "Teknik Kompilasi", edisi kedua, Penerbit Graha Ilmu, Yogyakarta, 2005.
[3] James, Suciadi, "Studi Analisis Metode-Metode Parsing Dan Interpretasi Semantik Pada Natural Language Processing" Universitas Kristen Petra, 2013.
[4] Bindu.M.S, Sumam Mary Idicula, "A Hybrid Model For Phrase Chunking Employing Artificial Immunity System And Rule Based Methods" International Journal of Artificial Intelligence & Applications (IJAIA), Vol.2, No.4, October, 2011.
[5] Jurnalis Web, di akses tanggal 5 desember 2014. http://www.jurnalweb.com/content/tutorial-composer-php/

Biodata Penulis

Yulianto, S.Kom, memperoleh gelar Sarjana Komputer (S.Kom), Jurusan Sistem Informasi Perguruan Tinggi Raharja, lulus tahun 2014. Saat ini menjadi Staff Dosen di Perguruan Tinggi Raharja.

Fifit Alfiah, menempuh pendidikan Strata 1 , Jurusan Teknik Informatika Perguruan Tinggi Raharja, insyaallah akan lulus awal tahun 2015. Saat ini sedang menempuh Skripsi/Tugas Akhir.

Andy Nova Wijaya, menempuh pendidikan Strata 1, Jurusan Teknik Informatika Perguruan Tinggi Raharja. Saat ini masih menempuh pendidikan semester 7 dan sedang melaksanakan kuliah kerja praktek (KKP) .

Muh. Rizal Ramadhan, menempuh pendidikan Strata 1, Jurusan Teknik Informatika Perguruan Tinggi Raharja. Saat ini masih menempuh pendidikan semester 7 dan sedang melaksanakan kuliah kerja praktek (KKP) .

Leo Kumoro Sakti, menempuh pendidikan Strata 1, Jurusan Teknik Informatika Perguruan Tinggi Raharja. Saat ini masih menempuh pendidikan semester 7 dan sedang melaksanakan kuliah kerja praktek (KKP) .

Mubtasir, menempuh pendidikan Strata 1, Jurusan Teknik Informatika Perguruan Tinggi Raharja. Saat ini masih menempuh pendidikan semester 7 dan sedang melaksanakan kuliah kerja praktek (KKP) .

Abdul Mukti, menempuh pendidikan Strata 1, Jurusan Teknik Informatika Perguruan Tinggi Raharja. Saat ini masih menempuh pendidikan semester 7 dan sedang melaksanakan kuliah kerja praktek (KKP) .

ISSN : 2302-3805