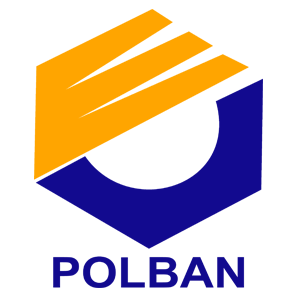
**LAPORAN TUGAS BESAR**

**PROGRAM *TEXT SIMILARITY (PLAGIARISM CHECK)***

Untuk Memenuhi Salah Satu Tugas

Mata Kuliah Struktur Data dan Algoritma

Dosen : Ani Rahmani



Disusun oleh

Annazar Darul Ismawan (171524005)

Zikri Ariachandra (171524032)

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

PROGRAM STUDI D4 TEKNIK INFORMATIKA

POLITEKNIK NEGERI BANDUNG

2018

**Program *Text* *Similarity* (*Plagiarism* *Check*)**

**Penjelasan Singkat Plagiarisme**

Plagiarisme atau sering disebut plagiat adalah penjiplakan atau pengambilan karangan, pendapat, dan sebagainya dari orang lain dan menjadikannya seolah karangan dan pendapat sendiri. Plagiat dapat dianggap sebagai tindak pidana karena mencuri hak cipta orang lain. Di dunia pendidikan, pelaku plagiarisme dapat mendapat hukuman berat seperti dikeluarkan dari sekolah/universitas. Pelaku plagiat disebut sebagai plagiator [1].

Singkat kata, plagiat adalah pencurian karangan milik orang lain. Dapat juga diartikan sebagai pengambilan karangan (pendapat dan sebagainya) orang lain yang kemudian dijadikan seolah-olah miliknya sendiri. Setiap karangan yang asli dianggap sebagai hak milik si pengarang dan tidak boleh dicetak ulang tanpa izin yang mempunyai hak atau penerbit karangan tersebut. Sesudah 2 × 24 jam berita surat kabar tersiar, maka seseorang dapat mengambil alih dengan syarat harus menyebutkan sumbernya.

**Deskripsi Program *Text* *Similarity***

*Text* *Similiarity* atau *Plagiarism* *Check* adalah program untuk memeriksa kemiripan antara dua *file* berekstensi .txt yang dipilih. Tujuan pembuatan program ini adalah untuk membantu pemeriksaan seberapa mirip sebuah suatu *file* dengan *file* lainnya.

Kemiripan suatu *file* dengan *file* lainnya ditentukan dengan menghitung seberapa banyak kata yang sama antara file yang diperiksa dengan *file* pembanding dibagi jumlah kata yang ada pada *file* yang diperiksa.

Tingkat kemiripan = jumlah kata yang sama / jumlah kata keseluruhan

Plagiarisme memiliki tiga tingkatan, yaitu :

* Persentase di bawah 30% merupakan plagiat ringan
* Persentase 30% sampai 70% merupakan plagiat sedang
* Persentase di atas 70% merupakan plagiat berat

(Tambahkan penjelasan mengenai spesifikasi program, mis. : definisi similarity pada program, file yang mampu dibaca program, text Bahasa…)

**Struktur Data dan Alur Proses**

**Struktur Data yang digunakan**

Berikut adalah struktur data yang digunakan dalam program yang akan kami buat.

Statistik.h

Statistik = record

word : String

size : integer

End

String.h

String = record

data : pointer of char

size : integer

allocated : integer

End

Statistik\_Tree.h

typedef pointer of Node Address

typedef Statistik Infotype

Node = record

info : Infotype

bf : integer

left : Address

right : Address

End

**Alur Proses Program**

Berikut adalah alur proses dari program yang akan kami buat.

**Step ke-1 : Membuat *tree* yang berisi *stopword***

1. *File* yang berisi kamus *stopword* dibaca kata per kata
2. Kata yang dibaca di-*insert*-kan ke dalam *tree*
3. Ulangi langkah ke-1 dan langkah ke-2 sampai *EOF* (*End* *Of* *File*) dari *file* tersebut.
4. Lanjut ke proses *tokenizing*

**Step ke-2 : Membaca isi *text* dari *file* per kata (*tokenizing*)**

1. File yang berisi *text* yang akan diperiksa dibaca karakter per karakter.
2. Karakter yang sedang dibaca akan diubah ke bentuk *lowercase dan* diperiksa apakah karakter tersebut merupakan spasi atau bukan.
   1. Jika iya, berarti string sudah menampung satu kata dan dilanjutkan ke proses *stopword* *removal*.
   2. Jika bukan, dilanjutkan ke langkah 3.
3. Karakter yang sedang dibaca akan diperiksa apakah karakter tersebut merupakan huruf atau bukan.
   1. Jika iya, maka karakter tersebut akan di-*insert*-kan ke dalam string.
   2. Jika bukan, maka karakter tersebut tidak akan di-*insert*-kan ke dalam string.

**Step ke-3 : Memeriksa apakah kata yang sudah dibaca termasuk *stopword* (*stopword* *removal*)**

Kata yang sudah didapat dari proses *tokenizing* akan diperiksa apakah kata tersebut terdapat pada *tree* yang menampung *stopword* atau tidak.

1. Jika iya, maka kembali ke proses *tokenizing*.
2. Jika tidak, maka dilanjutkan ke proses *stemming*.

Pemeriksaan dilakukan dengan cara melakukan *Binary* *Search* di dalam *tree* yang menampung kamus *stopword*.

**Step ke-4 : Menghilangkan imbuhan pada kata (*stemming*)**

Untuk penghilangan imbuhan dilakukan dengan menggunakan algoritma buatan pemilik akun *github* dengan nama “wooorm”.

**Step ke-5 : Meng-*insert*-kan kata yang telah melewati *preprocessing* ke dalam statistik *tree***

Kata yang sudah didapat dari proses stemming akan di-*insert*-kan ke dalam *AVL* *Tree* yang merepresentasikan statistik kemunculan huruf.

*AVL* *Tree* yang digunakan akan memiliki spesifikasi sebagai berikut :

1. Node yang memiliki nilai string lebih kecil dari *parent* akan berada di sebelah kiri dari *parent* (*Left* *Son*) dan Node yang memiliki nilai string lebih besar dari *parent* akan berada di sebelah kanan *parent* (*Right* *Son*).
2. Jika ditemukan Node yang memiliki nilai string sama dengan nilai string yang sedang di-*insert*, maka *value* count dari Node tersebut akan ditambahkan dan *insert* string dibatalkan.

**Step ke-6 : Ulangi step ke-2 sampai step ke-5 hingga semua *text* pada kedua *file* selesai diperiksa.**

Lakukan proses *tokenizing*, *stopword* *removal*, dan *stemming* serta *insert* ke dalam statistik *tree* ke semua kata yang terdapat pada kedua *file* yang akan diperiksa kemiripannya.

**Step ke-7 : Periksa kemiripan antara *file* pertama dan *file* kedua**

1. Akan dilakukan *traversal* ke semua Node pada *tree* yang merepresentasikan statistik kemunculan kata pada *file* pertama (*tree* statistik pertama) sambil dijumlahkan jumlah kemunculan huruf dari semua Node yang dikunjungi.
2. Node yang sedang dikunjungi akan diperiksa terhadap *tree* yang merepresentasikan statistik kemunculan kata pada *file* kedua (*tree* statistik kedua). Apakah ada Node yang memiliki nilai kata yang sama dengan Node yang sedang dikunjungi.
   1. Jika ada, maka ambil jumlah kemunculan yang paling kecil antara Node yang sedang dikunjungi dari *tree* statistik pertama dan Node yang sedang dikunjungi dari *tree* statistik kedua serta tambahkan ke variabel untuk menampung jumlah kata yang sama.
   2. Jika tidak, lanjut ke langkah ke 3.
3. Ulangi langkah 1 dan langkah 2 sampai semua Node pada *tree* statistik pertama selesai dikunjungi.
4. Hitung kemiripan dengan rumus :

Tingkat kemiripan = jumlah kata yang sama / jumlah kata keseluruhan

Pembahasan alur proses akan lebih baik jika dilas berdasakan tahapan proses (prepro…, …, …). Disetiap penjelasan alur proses harus tercakup input, proses, dan output (setiap tahapannya).

**Timeline dan Pembagian Tugas**

**Timeline**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Kegiatan | | Waktu Pengerjaan | | |
| Minggu I | Minggu II | Minggu III |
| 1 | Pre-Processing | Pencarian kamus stopword |  |  |  |
| 2 | Pencarian atau pembuatan algoritma tokenizing, stopword removal, stemming dan pembuatan statistik kata |  |  |  |
| 3 | Processing | Implementasi tokenizing |  |  |  |
| 4 | Implementasi stopword removal |  |  |  |
| 5 | Implementasi stemming |  |  |  |
| 6 | Implementasi pembuatan statistik kata |  |  |  |
| 7 | Implementasi penghitungan kemiripan file |  |  |  |
| 8 | Post-Processing | Uji Coba |  |  |  |
| 9 | Debugging |  |  |  |
| 10 | Laporan Akhir |  |  |  |

**Pembagian Tugas**

|  |  |  |
| --- | --- | --- |
| No | Nama | Task |
| 1 | Annazar Darul Ismawan | Pencarian atau pembuatan algoritma tokenizing, stopword removal, stemming, pembuatan statistik kata dan penghitungan kemiripan file |
| Implementasi tokenizing |
| Implementasi pembuatan statistik kata |
| Implementasi penghitungan kemiripan file |
| Debugging |
| Laporan Akhir |
| 2 | Zikri Ariachandra | Pencarian kamus stopword |
| Implementasi stopword removal |
| Implementasi stemming |
| Uji Coba |

**Daftar Pustaka**

[1] Wikipedia. Plagiarisme. <https://id.wikipedia.org/wiki/Plagiarisme> (6 Juni 2018)

[2] Wooorm. Porter Stemmer algorithm in C. <https://github.com/wooorm/stmr.c> (6 Juni 2018)