**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNOLOGI INFORMASI**

**INSTITUT TEKNLOGI SEPULUH NOPEMBER**

**USULAN TUGAS AKHIR**

# IDENTITAS PENGUSUL

**NAMA : KUKUH INDRAYANA**

**NRP : 5108100122**

**DOSEN WALI : ARYA YUDHI WIJAYA, S.Kom,M.kom**

# JUDUL TUGAS AKHIR

“Pembangunan Aplikasi Penghilangan Duplikasi Data Seleksi Nasional Masuk Perguruan Tinggi Negeri (SNMPTN) Menggunakan Algoritma Pencocokan String Berbasis Kerangka Kerja Vaadin”

# LATAR BELAKANG

Seleksi Nasional Masuk Perguruan Tinggi Negeri adalah salah satu bentuk ujian yang diselenggarakan oleh pemerintah. Dilaksanakan serentak dan terpadu di seluruh Indonesia untuk menyeleksi siswa-siswi Sekolah Menengah Atas (SMA) yang akan melanjutkan pendidikannya ke perguruan tinggi negeri di Indonesia. SNMPTN terdiri dari dua macam ujian yaitu ujian jalur undangan dan ujian jalur tertulis.

Pada ujian jalur tertulis, peserta mengisikan data pribadi serta jawabannya pada Lembar Jawaban Komputer (LJK). LJK ini kemudian di-*scan* dan datanya dimasukkan ke dalam basis data. Saat diperiksa secara manual, ternyata banyak didapati duplikasi data pada basis data. Duplikasi data disebabkan oleh kesalahan pengisian data nama atau nomor peserta oleh peserta pada saat hari ujian. Peserta mengisikan data nama yang berbeda dengan yang diisikan pada data pendaftaran. Sistem tentu akan menganggap dua data ini sebagai data yang berbeda. Sehingga terjadi duplikasi data pada sebuah nomor peserta. Ada juga peserta yang salah mengisikan nomor peserta sehingga satu nomor peserta diklaim oleh dua peserta.Jika ada duplikasi data, sistem akan menganggap data tersebut tidak sah. Akibatnya, peserta yang mampu menjawab soal dengan baik pun bisa dinyatakan tidak lulus ujian hanya karena kesalahan pengisian data. Agar hal ini tidak terjadi, maka duplikasi data harus dihilangkan.

Saat ini, untuk pemeriksaan duplikasi data, ditunjuk orang tertentu untuk memeriksa data secara manual. Pemeriksa dapat memutuskan apakah duplikasi data tersebut dianggap benar-benar tidak sah atau hanya kesalahan pengisian data. Pemeriksaan secara manual tentunya membutuhkan waktu yang lama, dan tidak menjamin terbebas dari *human error*. Dalam tugas akhir ini akan dibuat sistem yang mengimplementasikan algoritma pencocokan string untuk menghilangkan masalah duplikasi data tersebut. Sistem ini akan menggantikan pemeriksaan manual yang tidak efisien dan hasilnya kurang maksimal.

# RUMUSAN MASALAH

Detail permasalahan yang diangkat dalam tugas akhir ini adalah sebagai berikut:

* 1. Bagaimana membangun aplikasi penghilangan duplikasi data berbasis web menggunakan kerangka kerja Vaadin.
  2. Bagaimana mengolah data sampel yang memiliki duplikasi (data SNMPTN).
  3. Bagaimana mengimpelementasikan algoritma pencocokan string untuk penghilangan duplikasi data.

# BATASAN MASALAH

Masalah yang dibahas pada tugas akhir ini dibatasi lingkupnya pada:

1. Data sampel yang diolah adalah data SNMPTN.
2. Aplikasi dibangun dengan menggunakan kerangka kerja Vaadin.
3. Algoritma pencocokan string yang digunakan adalah algoritma *Soundex*, *Levenshtein Distance*, dan *Longest Common Subsequence*.

# TUJUAN PEMBUATAN TUGAS AKHIR

Tujuan dari penyusunan tugas akhir ini adalah:

1. Merancang bangunaplikasi penghilangan duplikasi data berbasis web menggunakan kerangka kerja vaadin.
2. Mengimplementasikan algoritma pencocokan string untukpenghilangan duplikasi data SNMPTN.

# MANFAAT TUGAS AKHIR

Manfaat dari tugas akhir ini adalah untuk menghilangkan duplikasi data SNMPTN.

# TINJAUAN PUSTAKA

## Pencocokan String

*String* adalah susunan dari karakter-karakter dan biasanya direpresentasikan sebagai struktur data *array*. *String* dapat berupa kata, frase, atau kalimat. Pencocokan *string* diartikan sebagai sebuah permasalahan untuk menemukan pola susunan karakter *string*di dalam *string* lain atau bagian dari isi teks[1].

## Fonetik

Fonetik adalah ilmu yang menyelidiki bunyi bahasa tanpa melihat fungsi bunyi itu sebagai pembeda makna dalam suatu bahasa. Pencocokan *string*fonetik adalah suatu teknik pencocokan string yang membandingkan suatu *string* dengan *string* yang lain berdasarkan kode fonetis masing-masing[2]. Sebuah *string* yang berbeda namun mempunyai cara pengucapan yang sama, akan memiliki kode fonetis yang sama. Contohnya, “akhmad” dengan “achmad”.

## Algoritma Soundex dan Normalisasi String

Algoritma *Soundex* pertama kali dipatenkan oleh Margaret O’Dell dan Robert C. Russell pada tahun 1918[3]. Algoritma *Soundex* mengambil masukan berupa sebuah kata atau nama. Algoritma ini menghasilkan sebuah *string* yang mengidentifikasi apakah sepasang kata tersebut mirip secara fonetis. String ini disebut dengan kode fonetis.

Metode yang digunakan algoritma *Soundex* berdasarkan pada klasifikasi fonetis dari suara cara berbicara manusia (*human speech sound*) yaitu labial, dental, alveolar, palato-alveolar, palatal, velar, dan glottal[3]. Klasifikasi ini berdasarkan pada dimana manusia meletakkan bibir dan lidahnya ketika melafalkan bunyi tertentu. Penjelasan klasifikasi ini adalah sebagai berikut[1] :

1. Labial adalah bunyi bibir.
2. Dental adalah bunyi diartikulasi oleh ujung lidah dengan gigi atas.
3. Alveolar adalah bunyi diartikulasi oleh ujung lidah dengan punggung gigi.
4. Palato-alveolar adalah bunyi yang memiliki artikulasi alveolar diikuti dengan naiknya lidah sampai pada langit-langit mulut secara simultan.
5. Palatal adalah bunyi diartikulasi oleh bagian depan lidah dengan langit-langit keras.
6. Velar adalah bunyi diartikulasi oleh bagian belakang lidah dengan langit-langit lunak.
7. Glottal adalah bunyi diartikulasi oleh glottis.

Aturan pemberian kode fonetis per huruf pada algoritma*Soundex* adalah sebagai berikut[4]:

**Tabel 1. Aturan pemberian kode fonetis pada algoritma *Soundex***

|  |  |  |
| --- | --- | --- |
| Huruf | Kode | Klasifikasi fonetis |
| A,E,H,I,O,U,W,Y | 0 | Diperlakukan sebagai bunyi vokal |
| B,F,P,V | 1 | Labial dan labio-dental |
| C,G,J,K,Q,S,X,Z | 2 | Glottal |
| D,T | 3 | Dental-mute |
| L | 4 | Palatal fricative |
| M,N | 5 | Labio-nasal dan dental |
| R | 6 | Dental fricative |

Langkah-langkah algoritma *Soundex* dalam menghasilkan kode fonetis dari sebuah *string* masukan adalah sebagai berikut[3]:

1. Ubah semua huruf menjadi huruf kapital, dan hilangkan tanda baca (misal tanda ‘).
2. Pertahankan huruf pertama pada kata tersebut.
3. Ubah huruf lainnya menjadi kode fonetis berdasarkan tabel.
4. Hapus semua pasangan dari kode fonetis yang berurutan.
5. Hapus semua kode fonetis 0.
6. Tulis empat posisi pertama yang mengikuti pola : <uppercase letter><digit><digit><digit>. Ini adalah kode fonetis sebagai keluaran.Jika kode fonetis tidak sampai empat karakter, maka ditambahkan digit 0 sampai menjadi empat karakter.

Misalkan ada sebuah string masukan “Baharuddin”. Langkah-langkah pemberian kode fonetisnya adalah sebagai berikut. Ubah semua huruf menjadi huruf kapital (“BAHARUDDIN”). Pertahankan huruf pertama, lalu ubah huruf lainnya menjadi kode fonetis berdasarkan tabel (B000603305). Hapus semua pasangan dari kode fonetis yang berurutan (B060305). Hapus semua kode fonetis 0 (B635). Empat karakter pertama adalah kode fonetis sebagai keluaran (B635).

Algoritma *Soundex* dibuat berdasarkan pengucapan dalam bahasa Inggris. Untuk mendukung pencocokan string berdasarkan bahasa Indonesia, sekaligus melakukan optimasi terhadap algoritma tersebut, diperlukan suatu proses normalisasi.

Salah satu cara untuk proses normalisasi adalah dengan menggunakan aturan translasi q-gram[5]. Aturan q-gram dibuat berdasarkan fakta bahwa banyak ditemukan nama dalam bahasa Indonesia yang menggunakan cara penulisan dan pengucapan asing, seperti Belanda, Arab, Cina, maupun Inggris. Aturan translasi q-gram dijelaskan dalam tabel berikut

**Tabel 2. Aturan Translasi q-gram**

|  |  |
| --- | --- |
| q-gram | |
| Awal | Translasi |
| KH,CH | HH |
| DJ | JJ |
| TJ | CC |
| CQ,CK | KK |
| PH | FF |
| DZ | ZZ |
| SJ | SY |
| SY | SS |
| BH,DH,GH,JH,SH,TH,ZH | BB,DD,GG,JJ,SS,TT,ZZ |
| V | F |
| KS | XX |
| OE | UU |
| IE | II |
| Y | I |

## Sebelum dilakukan normalisasi, string “Budjana” memiliki kode fonetis B325, dan string “Bujana” memiliki kode fonetis B250. Padahal, kedua nama tersebut sama secara fonetis. Setelah dilakukan normalisasi, “Budjana” akan menjadi “Bujjana”. “Bujjana” memiliki kode fonetis yang sama dengan “Bujana”, yakni B250 karena pasangan karakter yang muncul berurutan (“jj”) hanya akan digantikan oleh satu kode fonetis. Dengan demikian, setelah dilakukan normalisasi, string nama yang mempunyai kesamaan pengucapan akan mendapatkan kode fonetis yang sama.

## Algoritma Levenshtein Distance

Algoritma *Levenshtein Distance* dinamakan berdasarkan penemunya, yaitu Dr. Vladimir Levenshtein. Dr. Vladimir Levenshtein dikenal sebagai bapak dari teori koding (*father of coding theory*) di Rusia[6]. *Levenshtein Distance* saat ini adalah dasar dari banyak aplikasi komputer di bidang pemeriksaan ejaan.

*Levenshtein Distance* adalah jumlah minimal operasi yang dibutuhkan untuk mengubah suatu *string* ke *string* yang lain. Dengan algoritma ini, dapat diketahui tingkat perbedaan dua buah *string* dalam representasi angka. Operasi yang dilakukan ada 3 macam yaitu[7]:

1. *Insertion*

*Insertion* adalah penyisipan sebuah karakter kedalam sebuah *string* tertentu. Contohnya, penyisipan karakter ‘m’ kedalam *string* “julah” pada posisi setelah karakter ‘u’ akan mengubah *string* “julah” menjadi “jumlah”.

1. *Deletion*

*Deletion* adalah penghapusan sebuah karakter dari sebuah *string* tertentu. Contohnya, penghapusan karakter ‘a’ pada posisi terakhir *string* “algoritma” akan mengubah *string* “algoritma” menjadi “algoritm”.

1. *Substitution*

*Substitution* adalah penggantian sebuah karakter dari sebuah *string* dengan karakter lain. Contohnya, penggantian karakter ‘l’ dengan karakter ‘b’ pada *string* “julah” akan mengubah *string* “julah” menjadi “jubah”.

Misalkan ada dua buah masukan *string* yaitu “kepala” dan “kapal”. Langkah untuk mengubah “kepala” menjadi “kapal” adalah sebagai berikut. Substitusi karakter ‘e’ dengan karakter ‘a’. Kemudian hapus karakter ‘a’ pada posisi terakhir. Jadi, jumlah operasi yang dilakukan adalah dua kali. Satu kali substitution dan satu kali deletion. *Levenshtein Distance* adalah jumlah dari banyaknya operasi yang dilakukan. Sehingga, *Levenshtein Distance* dari “kepala” dan “kapal” adalah 2.

*Levenshtein Distance* menggunakan konsep *dynamic programming* dalam penerapannya. Algoritma ini menggunakan sebuah matriks berukuran panjang dari *string*-*string* yang menjadi masukan. *Levenshtein Distance* yang dihasilkan adalah angka yang terletak di kotak paling bawah-kanan dari matriks, sebagai jumlah minimum operasi yang dibutuhkan untuk mengubah *string* awal menjadi *string* target. Berikut *pseudocode* algoritma *Levenshtein Distance*:

functionLevenshteinDistance(s1[1..m],s2[1..n])

levDist ← array(0..m, 0..n)

for i ←0 to m

for j ← 0 to n

if i = 0

levDist[i,j] ← j

else if j = 0

levDist[i,j] ← i

else

if(s1[i]=s2[j])levDist[i,j] ← levDist[i-1,j-1];

elselevDist[i,j] ← 1 + minimum(levDist[i,j-1],

levDist[i-1,j],

levDist[i-1,j-1]);

returnlevDist[i,j];

**Gambar 1 Algoritma *Levenshtein Distance***

Berikut ini contoh matriks *Levenshtein Distance* untuk masukan “kepala” dan “kapal”:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | K | E | P | A | L | A |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| K | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| A | 2 | 1 | 1 | 2 | 2 | 3 | 4 |
| P | 3 | 2 | 2 | 1 | 2 | 3 | 4 |
| A | 4 | 3 | 3 | 2 | 1 | 2 | 3 |
| L | 5 | 4 | 4 | 3 | 2 | 1 | 2 |

**Gambar 2 contoh matriks algortima *Levenshtein Distance***

*String* pada baris pertama adalah *string* awal. Sedangkan *string* pada kolom pertama adalah *string* target. Angka pada setiap sel matriks adalah banyaknya operasi yang dibutuhkan untuk mengubah *substring* awal menjadi *substring* target. Misalnya pada baris pertama dan kolom pertama, nilainya adalah 0, karena tidak ada operasi yang diperlukan untuk mengubah *substring* awal “k” menjadi *substring* target “k”. Pada baris kedua dan kolom kedua, nilainya adalah 1, karena ada satu operasi yang dilakukan, yaitu substitusi karakter ‘e’ dengan ‘a’.Demikian seterusnya sampai dengan baris terakhir kolom terakhir.

## Algoritma Longest Common Subsequence

## *Subsequence* adalah sebuah rangkaian yang dapat diperoleh dari rangkaian lain dengan cara menghapus beberapa elemen tertentu tanpa mengubah urutan dari elemen lain yang tersisa[8].Misalkan ada rangkaian X dan Y, rangkaian S dikatakan sebuah *common subsequence* dari X dan Y apabila S adalah *subsequence* dari keduanya. Longest Common Subsequence (LCS) adalah *common subsequence* terpanjang dari seluruh *common subsequence*. Contohnya sebagai berikut:

X = {M,A,K,A,N}

Y = {M,A,L,A,M}

Contoh *common subsequence* dari X dan Y adalah S = {M,A,A} dan T= {M,A}. Karena S merupakan *subsequence* yang lebih panjang, maka *longest common subsequence* dari X dan Y adalah S.

Algoritma untuk masalah LCS ini berdasarkan teorema berikut[9]:

Misal X = <x1,x2, .. , xm> dan Y = <y1,y2, .. , yn> adalah rangkaian dan Z = <z1,z2, .. , zk> adalah suatu LCS dari X dan Y.

1. Jika xm = yn, maka zk = xm = yn dan Zk-1 adalah suatu LCS dari Xm-1 dan Yn-1.
2. Jika xm ≠ yn maka zk ≠ xm mengimplikasikan bahwa Z adalah suatu LCS dari Xm-1 dan Y.
3. Jika xm ≠ ym maka zk ≠ yn mengimplikasikan bahwa Z adalah suatu LCS dari X dan Yn-1.

Misalkan *lcs* adalah matriks berukuran *m* x *n* dan *lcs* adalah panjang LCS dari *i* karakter pertama rangkaian X dengan *j* karakter pertama rangkaian Y maka fungsi rekursif untuk mendapatkan panjang LCS adalah sebagai berikut:

lcs[*i*,*j*] = 0 jika*i* = 0 atau*j* = 0

= lcs [*i*-1,*j*-1] + 1 jikaX[*i*] = Y[*j*]

= max ( lcs [*i*,*j*-1], lcs [*i*-1,*j*]) jikaX[*i*] ≠Y[*j*]

Misal untuk X = “SELALU” dan Y = “SALUT” matriks yang terbentuk adalah sebagai berikut:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | S | E | L | A | L | U |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| A | 0 | 1 | 1 | 1 | 2 | 2 | 2 |
| L | 0 | 1 | 1 | 2 | 2 | 3 | 3 |
| U | 0 | 1 | 1 | 2 | 2 | 3 | 4 |
| T | 0 | 1 | 1 | 2 | 2 | 3 | 4 |

**Gambar 3 contoh matriks algoritma *Longest Common Subsequence***

Nilai angka pada kolom i dan baris j pada matriks adalah panjang lcs dari i karakter pertama dari rangkaian X dan j karakter pertama dari rangkaian Y. Misalkan pada kolom ketiga dan baris ketiga, nilainya adalah 2, karena lcs dari “SEL” dan “SAL” adalah “SL” yang mempunyai panjang 2 karakter. Nilai lcs dari rangkaian X dan Y ditunjukkan oleh nilai pada baris terakhir dan kolom terakhir, yaitu 4.

Pseudocode dari algoritma Longest Common Subsequence adalah sebagai berikut:

**function** LongestCS(x[1..m], y[1..n])

lcs := **array**(0..m, 0..n)

**for** i := 0..m

**for** j := 0..n

**if** i = 0 or j = 0

lcs[i,j] := 0

**else**

**if**x[i] = y[j]

lcs[i,j] := lcs[i-1,j-1] + 1

**else**

lcs[i,j] := max (lcs[i,j-1],lcs[i-1,j])

**return**lcs[m,n]

**Gambar 4 Algoritma *Longest Common Subsequence***

# RINGKASAN ISI TUGAS AKHIR

Dalam tugas akhir ini akan dibuat sebuah Sistem yang mengimplemetasikan algoritma pencocokan string untuk penghilangan duplikasi data SNMPTN.Duplikasi data terjadi akibat perbedaan pengisian data nama pada saat pendaftaran dan ujian. Duplikasi data bisa juga disebabkan kesalahan pengisian data tanggal lahir. Namun, kejadian kesalahan pengisian data tanggal lahir sangan jarang terjadi.Jika data nama sama, namun data tanggal lahir berbeda, sistem tetap menganggap data tersebut sah. Berdasarkan pengamatan data, ada 3 macam kesalahan pengisian data nama yang menyebabkan duplikasi. Kesalahan tersebut adalah sebagai berikut:

1. Terjadi kesalahan penulisan karakter, namun data nama pada saat pendaftaran dan ujian bisa dianggap sama secara fonetis. Misalkan salah satu peserta mengisikan “Achmad” pada data pendaftaran dan “Ahmad” pada saat ujian. Ada karakter ‘c’ pada data pendaftaran, namun tidak ada pada data ujian. Namun, secara fonetis, “Achmad” dan “Ahmad” dianggap sama. Untuk kesalahan jenis ini akan diperiksa dengan algoritma soundex.
2. Terjadi kesalahan penulisan karakter, dan data nama pada saat pendaftaran dan ujian dianggap tidak sama secara fonetis setelah diperiksa dengan algoritma soundex.Kesalahan seperti ini biasanya disebabkan oleh ketidaktelitian ketika mengisi LJK. Misalkan salah satu peserta dengan nama “Bambang” mengisi data nama di LJK dengan “Bambanf”. Kesalahan seperti ini akan diperiksa dengan algoritma levenshtein distance. Toleransi kesalahan ditetapkan sebanyak tiga karakter. Artinya, dua masukan dianggap sama apabila nilai levenshtein distance dari kedua masukan tersebut kurang dari sama dengan tiga.
3. Peserta menulis data nama pendaftaran dengan menggunakan singkatan, dan menulis data ujian dengan tidak menggunakan singkatan, ataupun sebaliknya. Misalkan salah satu peserta mengisi data pendaftaran dengan “Muh Iqbal” dan mengisi data ujian dengan “Muhammad Iqbal”. Kesalahan seperti ini tidak bisa ditoleransi oleh algoritma levenshtein distance, karena jumlah karakter yang berbeda bisa sangat banyak, dan levenshtein distancenya akan lebih besar dari tiga. Untuk kasus seperti ini akan diperiksa dengan algoritma longest common subsequence. Dua nama masukan akan dibandingkan panjangnya. Jika lcs dari dua nama ini sama dengan panjang dari nama masukan yang lebih pendek, maka akan disimpulkan bahwa nama yang lebih pendek adalah penulisan singkatan dari nama yang lebih panjang. Misalkan pada contoh diatas, “Muh Iqbal” mempunyai panjang 9 (spasi diperhitungkan) dan “Muhammad Iqbal” mempunyai panjang 14. Lcs dari dua nama tersebut adalah 9 (‘M’,’u’,’h’,’ ‘,’I’,’q’,’b’,’a’,’l’) sehingga dapat disimpulkan bahwa “Muh Iqbal” adalah penulisan singkatan dari “Muhammad Iqbal”.

Ada 4 jenis data ujian pada SNMPTN, yaitu Tes Potensi Akademik, Tes Kemampuan Dasar, Tes Kemampuan IPA, dan Tes Kemampuan IPS. Peserta yang memilih jalur IPA atau IPS hanya akan mengisi 3 dari 4 jenis tes tersebut. Peserta yang memilih jalur IPC akan mengisi semua jenis tes. Data-data ini nantinya akan dimasukkan secara berurutan kedalam sistem.



**Gambar 5 Diagram Kasus Pengguna**

**Gambar 6 Arsitektur Sistem**

Diagram alir dari sistem adalah sebagai berikut:



**Gambar 6 Diagram Alir Sistem**

* Sistem menerima masukan awal berupa data pendaftaran SNMPTN.
* Sistem menerima masukan berupa data ujian Tes Potensi Akademik (TPA).
* Sistem memeriksa apakah *string* nama pada data pendaftaran dan pada data TPA sama.
* Jika sama, maka tidak terjadi duplikasi data, dan data ujian dianggap sah.
* Jika tidak, maka terjadi duplikasi data. Sistem akan mengubah kedua data nama tersebut menjadi dua buah kode fonetis menggunakan algoritma soundex.
* Jika kedua kode fonetis tersebut sama, maka disimpulkan bahwa dua nama tersebut adalah sama, dan data ujian dianggap sah.
* Jika kedua kode fonetis tidak sama, maka akan dilanjutkan pemeriksaan berikutnya, yaitu pemeriksaan dengan algoritma levenshtein distance.
* Kedua data nama tersebut akan dicari levenshtein distance-nya. Jika nilai levenshtein distance yang didapat kurang dari sama dengan tiga, maka disimpulkan bahwa dua nama tersebut adalah sama, dan data ujian dianggap sah. Jika nilai yang didapat lebih dari dua, maka akan masuk ke pemeriksaan selanjutnya.
* Kedua data nama tersebut akan dibandingkan panjangnya.Jika longest common subsequence dari dua nama ini sama dengan panjang dari data nama yang lebih pendek, maka akan disimpulkan bahwa nama yang lebih pendek adalah penulisan singkatan dari nama yang lebih panjang, dan data ujian dianggap sah.
* Jika tidak, maka akan dilakukan pemeriksaan terakhir, yaitu pencocokan data tanggal kelahiran. Jika data tanggal kelahiran sama, maka dua nama tersebut masih dianggap sama, dan data ujian dianggap sah. Jika tidak, maka data ujian dianggap tidak sah, dan peserta dianggap tidak mengikuti tes tersebut.
* Proses pengecekan ini diulang untuk Tes Kemampuan Dasar, Tes Kemampuan IPA, dan Tes Kemampuan IPS.

# METODOLOGI

## Penyusunan proposal tugas akhir

Pada tahap ini penulis menyusun proposal tugas akhir sebagai langkah awal dalam pengerjaan tugas akhir. Pada proposal ini penulis menggagas penulisan tugas akhir untuk merancang bangun sistem yang mengimplementasikan algoritma pencocokan string untuk penghilangan duplikasi data SNMPTN.

## Studi literatur

Pada tahap ini dilakukan pengumpulan informasi yang diperlukan untuk perancangan sistem. Informasi tersebut dapat diperoleh dari literatur, paper, maupun buku-buku terkait dengan perancangan maupun pembangunan perangkat lunak.

## Analisis dan desain perangkat lunak

Analisis kebutuhan dan perancangan sistem dilakukan untuk merumuskan solusi yang tepat dalam pembuatan aplikasi serta kemungkinan yang dapat dilakukan untuk mengimplementasikan rancangan tersebut. Fase desain meliputi arsitektur perangkat lunak yang digunakan, desain class-class yang terlibat dalam aplikasi, desain basis data, dan lain-lain.

## Implementasi perangkat lunak

Pada fase ini desain perangkat lunak diwujudkan ke dalam bentuk kode program. Pembangunan aplikasi dilakukan menggunakan platform Java.Pada fase ini, penulis menggunakan *NetBeans IDE* sebagai alat bantu untuk implementasi pengkodean program.

## Uji coba dan evaluasi

Pada tahap ini dilakukan ujicoba terhadap perangkat lunak yang dibuat.Tujuan uji coba perangkat lunak adalah untuk menemukan kesalahan-kesalahan (*bug*) sedini mungkin sehingga dapat diperbaiki sesegera mungkin.

## Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat. Sistematika penulisan buku tugas akhir secara garis besar antara lain:

1. Pendahuluan
   1. Latar Belakang
   2. Rumusan Masalah
   3. Batasan Tugas Akhir
   4. Tujuan
   5. Metodologi
   6. Sistematika Penulisan
2. Tinjauan Pustaka
3. Desain dan Implementasi
4. Uji Coba dan Evaluasi
5. Kesimpulan dan Saran
6. Daftar Pustaka

# JADWAL KEGIATAN

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tahapan | 2012 | | | | | | | | | | | | | | | | |
| September | | | | Oktober | | | | November | | | | Desember | | | | |
| Penyusunan Proposal |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| StudiLiteratur |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Perancangansistem |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Implementasi |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ujicobadanevaluasi |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Penyusunanbuku |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

# DAFTAR PUSTAKA

[1]**Syaroni, Mokhamad dan Munir, Rinaldi**.*Pencocokan String Berdasarkan Kemiripan Ucapan (Phonetic String Matching) Dalam Bahasa Inggris*. Departemen Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Bandung.

[2]**Purnamasari, Tryas Ayu dan Luthfi, Emha Taufiq**.*Membangun Aplikasi Pencocokan String Berdasarkan Penulisan Dan Kemiripan Pengucapan*. Jurusan Teknik Informatika, STMIK AMIKOM Yogyakarta.

[3]*Understanding Classic Soundex Algorithms*. Creativyst, Inc.http://www.creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm

[4]*Soundex Coding Rules*. http://www.genealogyintime.com/GenealogyResources/Articles/what\_is\_soundex\_and\_how\_does\_soundex\_work\_page2.html.

[5]**Karhendana, Arie, Wizanajani, Dicky, dan Yuliawan, Fajar**. *Normalisasi String untuk Optimasi Phonetic String Matching dalam Bahasa Indonesia*. Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

[6]*Vladimir I. Levenshtein biography*<http://www.ieeeghn.org/wiki/index.php/Vladimir_I._Levenshtein>

[7]**Pinzon, Yoan**. *Algorithms for Approximate String Matching*. Universidad Nacional de Colombia, 2006.

[8]**Sala, Jared**. *Longest Common Subsequence*. University of New Mexico.

[9]**Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford**. *Introduction to Algorithm.*Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1992.

**LEMBAR PENGESAHAN**

Surabaya, 27 September 2012

Menyetujui

|  |  |
| --- | --- |
| Dosenpembimbing I **DwiSunaryono, S.Kom, M.Kom**  **NIP.197205281997021001** | Dosenpembimbing II **Abdul Munif, S.Kom, M.Sc.**  **NIP.051100114** |