**BAB IV**

**EKSTRAKSI INFORMASI LEXICAL DATABASE FILES**

Lexical Database Files merupakan sekumpulan text files yang menyimpan informasi mengenai gloss, sinonim, dan relasi yang dimiliki oleh kata-kata pada suatu bahasa. Lexical Database Files ini dikembangkan oleh Princeton University sebagai bagian dari PrincetonWordNet (PWN), yaitu sebuah WordNet berbahasa Inggris. Dalam perkembangannya Princeton University menyediakan sebuah library untuk mengekstrak informasi yang terdapat dalam Lexical Database Files agar PWN dapat dimanfaatkan oleh para programmer yang memiliki kepentingan terhadap PWN. Sayangnya library tersebut hanya tersedia untuk bahasa pemrograman Java dan .NET. Oleh karena itu pada Tugas Akhir ini dikembangkan sebuah library untuk bahasa pemrograman Objective-C yang berperan untuk melakukan ekstraksi informasi pada Lexical Database Files. Library yang dikembangkan pada Tugas Akhir ini berupa sebuah class bernama WordNet.

Class WordNet berfungsi sebagai ekstraktor Lexical Database serta penghubung antara Lexical Database dengan WordNet Browser. Lexical Database Files akan mengalami pre-processing dimana informasi-informasi pada Lexical Database Files tersebut akan disimpan ke dalam sekumpulan dictionary. Ketika WordNet Browser melakukan pencarian kata maka class WordNet akan mencari kata tersebut pada dictionary yang dimilikinya dan mengembalikan informasi yang dimiliki oleh kata tersebut. Agar pengguna class WordNet dapat melakukan pencarian tersebut maka terdapat beberapa prosedur yang disediakan oleh class WordNet. Terdapat lima buah prosedur bersifat public yang dapat digunakan untuk melakukan ekstraksi informasi milik sebuah kata, selain kelima prosedur tersebut juga terdapat prosedur private yang membantu dalam melakukan ekstraksi informasi. Tabel 4.1 Menunjukkan daftar prosedur yang dimiliki oleh libary WordNet.

**Tabel 4.1**

**Daftar Prosedur Class WordNet**

|  |  |  |
| --- | --- | --- |
| Nama Prosedur | Sifat | Fungsi |
| getWordData | Public | Mendapatkan synset offset sebuah kata. |
| getSynsetData | Public | Mendapatkan informasi yang dimiliki sebuah synset offset. |
| getSynsetLemma | Public | Mendapatkan sinonim pada index tertentu. |
| getLexName | Public | Mendapatkan lexname dari lexFileNum tertentu. |
| getRelationName | Public | Mendapatkan jenis relasi dari relation pointer tertentu. |
| readFromFile | Private | Memindahkan isi Lexical Database ke dalam dictionary yang bersangkutan. |
| getWordClassData | Private | Melakukan Ekstraksi berdasarkan String Data yang diterima. |

Setiap prosedur tersebut memiliki fungsi yang spesifik dan setiap prosedur tersebut akan dijelaskan pada subbab yang berhubungan dengan proses yang dilakukan. Walaupun begitu semua prosedur tersebut memiliki ketergantungan satu sama lain. Misalnya prosedur getSynsetLemma memerlukan synset offset yang didapatkan dengan menggunakan prosedur getWordData. Hasil ekstraksi akan disimpan ke dalam sebuah struktur penyimpanan yang khusus, struktur-struktur yang digunakan akan dijelaskan bersama dengan proses yang dilakukan. Proses-proses yang dilakukan untuk melakukan ekstraksi informasi akan dijelaskan secara detail pada bab ini.

* 1. **Pre-Processing Lexical Database Files**

Sebelum ekstraksi informasi yang dimiliki oleh suatu kata dapat dilakukan oleh class WordNet, terlebih dahulu harus dilakukan pre-processing terhadap Lexical Database Files. Tahap pre-processing ini mencakup pengambilan data pada kelompok index files, data files, dan lexname file dan menyimpannya ke dalam dictionary yang dimiliki oleh class WordNet.



**Gambar 4.1**

**Pre-Processing Lexical Database Files**

Pada gambar 4.1 ditunjukkan bahwa tahap pre-processing Lexical Database Files dilakukan dua buah tahap proses. Tahap pertama adalah pengambilan string dari index pertama hingga ditemukannya spasi pertama, aturan ini berlaku unuk kelompok index file dan data file. Untuk file lexnames pemotongan string dilakukan berdasarkan tab atau “\t” karena pada file lexnames pemisah antar fieldnya bukanlah spasi melainkan tab. Tahap yang kedua adalah menyimpan string hasil ekstraksi tersebut ke dalam dictionary, string yang didapatkan dari proses pemotongan berdasarkan spasi atau tab akan menjadi key dari dictionary sedangkan keseluruhan string tersebut akan menjadi value bagi dictionary. Output dari proses ekstraksi ini berupa dictionary yang berisi data hasil ekstraksi Lexical Database Files. Gambar 4.2 menunjukkan ilustrasi penyimpanan data pada file index.noun ke dalam noun dictionary.

aba n 1 1 @ 1 0 05462574

Index.noun

Key : aba

Value : aba n 1 1 @ 1 0 05462574

nounDictionary

Pengambilan Lemma

**Gambar 4.2**

**Ilustrasi Penyimpanan pada Dictionary**

Pre-processing hanya bertujuan memindahkan isi dari Lexical Database Files ke dalam dictionary dengan lemma / synset offset / lexical id sebagai key dari dictionary dengan tujuan mempercepat proses pencarian pada Lexical Database Files. Apabila ekstraksi informasi pada setiap kata dilakukan pada tahap ini maka akan pre-processing akan memakan waktu yang lama. Tahap Pre-processing ini ditangani oleh prosedur readFromFile milik class WordNet. Prosedur tersebut bersifat private dan hanya ditujukan sebagai inisialisasi class WordNet yang dijalankan ketika instrance dari class WordNet dibuat.

**Segmen Program 4.1 Prosedur ReadFromFile**

1. #include <stdio.h>
2. #include <string.h>
3. -(void) readFromFile :(NSString \*)path

forDict : (NSMutableDictionary \*) dict{

1. if ([[NSFileManager defaultManager] fileExistsAtPath:path])
2. {
3. const char \*target = [path StringUsingEncoding:NSASCIIStringEncoding];
4. char str[10000];
5. FILE \*fp = fopen(target, "r");
6. int lineNumber=0;
7. while(fgets(str, 10000, fp))
8. {
9. lineNumber++;
10. char keys[] = "lexnames";
11. char \*j;
12. j = strstr (target,keys);
13. if (j == nil)
14. {
15. if (lineNumber>29)
16. {
17. char keys[] = " ";
18. int i;
19. i = strcspn (str,keys);
20. char str2[i];
21. strncpy(str2,str,i);
22. str2[i] = '\0';
23. NSString \*key = [NSString stringWithUTF8String:str2];
24. NSString \*value = [NSString stringWithUTF8String:str];
25. [dict setObject:value forKey:key];
26. }
27. }else {
28. char keys[] = "\t";
29. int i;
30. i = strcspn (str,keys);
31. char str2[i];
32. strncpy(str2,str,i);
33. str2[i] = '\0';
34. NSString \*key = [NSString stringWithUTF8String:str2];
35. NSString \*value = [NSString stringWithUTF8String:str];
36. [dict setObject:value forKey:key];
37. }
38. }
39. fclose(fp);
40. }
41. }

Segmen Program 4.1 menunjukan bahwa prosedur readFromFile ini menggunakan bahasa pemrograman Objective-C dan C++. Bahasa pemrograman C++ digunakan untuk mempercepat proses pembacaan dan pemotongan text file pada Lexical Database Files. Pada baris 1 dan baris 2 ditunjukkan include yang digunakan oleh C++ kedua include tersebut mengandung fungsi-fungsi string dan file. Pada baris 4 dilakukan pemeriksaan apakah file benar-benar ada pada path yang dikirimkan sebagai parameter. Baris 6-8 merupakan insialisasi yang diperlukan untuk melakukan pembacaan file pada C++. Pada baris 9 dilakukan insialisasi variabel lineNumber, variabel ini bertujuan untuk mengetahui apakah license agreement dari file sudah terlewati. Pembacaan file dilakukan pada baris 10 dimana satu baris data akan diambil atau higga data mencapai 10000 byte, hasil pembacaan data akan disimpan pada variabel str. Pada baris 12-16 akan dilakukan pemeriksaan apakah file yang dibaca merupakan file lexnames, jika benar maka program akan melanjutkan ke baris 31 dimana dilakukan pemisahan data berdasarkan tab atau “\t” dan memasukkannya ke dalam dictionary.

Jika ternyata file tersebut bukanlah file lexnames maka program akan melanjukan ke baris 18 dimana akan dilakukan pemeriksaan apakah license agreement sudah terlewati. Pada baris 20-25 dan 31-36 program akan melakukan pemotongan string, dimana string pertama hingga ditemukannya spasi atau tab pertama kali akan dianggap sebagai key dan keseluruhan string akan disimpan sebagai value pada dictionary. Hal tersebut berlaku baik untuk index file maupun data file. Konversi dari tipe data char \* milik C++ menjadi tipe data NSString milik Objective-C dilakukan pada baris 26-27 dan 37-38. Seteleh konversi selesai dilakukan maka NSString akan dimasukkan ke dalam NSMutableDictionary yang juga didapatkan melalui parameter, proses ini terjadi pada baris 28 dan 38. Prosedur ini diakhiri dengan menutup koneksi dengan file yang dilakukan pada baris 42.

* 1. **Ekstraksi Informasi Sebuah Kata**

Setelah Lexical Database Files mengalami pre-processing, kini class WordNet telah siap melakukan ekstraksi informasi pada Lexical Database Files. Proses ekstraksi informasi ini mengalami dua tahap proses, tahap pertama adalah pencarian kata pada index dictionary dan tahap kedua ekstraksi informasi pada data dictionary. Tahap pertama akan menghasilkan kumpulan synset yang dimiliki oleh kata dan pada tahap kedua akan dihasilkan informasi mengenai gloss, sinonim, dan relasi yang dimiliki oleh setiap synset tersebut. Kedua tahap proses ekstraksi informasi ditunjukkan berupa dua buah lingkaran pada gambar 4.3.



**Gambar 4.3**

**Proses Ekstraksi Informasi**

Pada gambar 4.3 ditunjukkan bahwa terdapat empat buah kelompok dictionary yang berperan dalam ekstraksi informasi kata yaitu index dictionary, lex dictionary, data dictionary, dan relation dictionary. Keempat kelompok dictionary tersebut merupakan variabel-variabel yang dimiliki oleh class WordNet, pada tabel 4.2 ditunjukkan penjelasan mengenai masing-masing kelompok dictionary.

**Tabel 4.2**

**Dictionary milik Class WordNet**

|  |  |  |
| --- | --- | --- |
| Nama dictionary | Kelompok Dictionary | Fungsi |
| noun | Index Dictionary | Menampung data index.noun |
| verb | Index Dictionary | Menampung data index.verb |
| adj | Index Dictionary | Menampung data index.adj |
| adv | Index Dictionary | Menampung data index.adv |
| nounData | Data Dictionary | Menampung data data.noun |
| verbData | Data Dictionary | Menampung data data.verb |
| adjData | Data Dictionary | Menampung data data.adj |
| advData | Data Dictionary | Menampung data data.adj |
| lex | Lex Dictionary | Menampung data lexnames |
| relation | Relation Dictionary | Mencatat relation pointer |

Kelompok index dictionary dan kelompok data dictionary tediri dari empat buah dictionary, dimana setiap dictionary mewakili satu buah class kata sedangkan kelompok lex dictionary dan relation dictionary hanya terdiri dari satu buah dictionary. Kelompok index dictionary akan menampung data yang didapat dari kelompok index dictionary sedangkan kelompok data dictionary akan menampung data yang didapat dari kelompok data dictionary dan lex dictionary akan menampung data yang didapat dari lexnames file. Ketiga kelompok dictionary tersebut mendapatkan nilai yang berasal dari Lexical Database Files Berbeda dengan kelompok relation dictionary yang nilainya tidak berasal dari Lexical Database Files. Seperti yang telah dijelaskan pada bab II bahwa setiap jenis relasi akan memiliki sebuah simbol yang mewakili jenis relasi tersebut. Karena relasi ini berlaku untuk setiap bahasa maka relation dictionary memiliki nilai yang statis dan pengisian dari dictionary ini dilakukan secara otomatis oleh class WordNet. Pengisian nilai relation dictionary ditunjukkan oleh segmen program 4.2.

**Segmen Program 4.2 Pengisian Relation Dictionary**

1. [relation setObject:@"Antonim" forKey:@"!"];
2. [relation setObject:@"Hipernim" forKey:@"@"];
3. [relation setObject:@"Insctance Hipernim" forKey:@"@i"];
4. [relation setObject:@"Hiponim" forKey:@"~"];
5. [relation setObject:@"Instance Hiponim" forKey:@"~i"];
6. [relation setObject:@"Member Holonim" forKey:@"#m"];
7. [relation setObject:@"Substance Holonim" forKey:@"#s"];
8. [relation setObject:@"Part Holonim" forKey:@"#p"];
9. [relation setObject:@"Member Meronim" forKey:@"%m"];
10. [relation setObject:@"Substance Meronim" forKey:@"%s"];
11. [relation setObject:@"Part Meronim" forKey:@"%p"];
12. [relation setObject:@"Atribut" forKey:@"="];
13. [relation setObject:@"Derivationally Related Form"

forKey:@"+"];

1. [relation setObject:@"Domain of synset - Topic"

forKey:@";c"];

1. [relation setObject:@"Member of this domain - Topic"

forKey:@"-c"];

1. [relation setObject:@"Domain of synset - Region"

forKey:@";r"];

1. [relation setObject:@"Member of this domain - Region"

forKey:@"-r"];

1. [relation setObject:@"Domain of synset - Usage"

forKey:@";u"];

1. [relation setObject:@"Member of this domain - Usage"

forKey:@"-u"];

1. [relation setObject:@"Entailment" forKey:@"\*"];
2. [relation setObject:@"Cause" forKey:@">"];
3. [relation setObject:@"Also See" forKey:@"^"];
4. [relation setObject:@"Verb Group" forKey:@"$"];
5. [relation setObject:@"Similar To" forKey:@"&"];
6. [relation setObject:@"Participle of Verb" forKey:@"<" ];
7. [relation setObject:@"Pertainim (Perains to Noun)"

forKey:@"\\"];

Segmen program 4.2 menunjukkan proses pengisian relation dictionary. Key pada dictionary ini berupa simbol yang mewakili suatu relasi, sedangkan valuenya adalah relasi yang diwakilkan oleh simbol tersebut. Baris 1 menunjukkan proses menyimpan simbol “!” sebagai key pada dictionary dan relasi antonim pada valuenya, baris 2-26 menunjukkan proses penyimpanan berbagai relasi yang lain. Pada Tugas Akhir ini relasi yang disediakan adalah antonim, hipernim, hiponim, holonim dan meronim, tetapi semua jenis relasi akan disimpan pada dictionary. Hal ini dilakukan untuk mempersiapkan apabila relasi tersebut tersedia pada Lexical Database Files pada bahasa lain.

* + 1. **Mencari Kata dan Mengekstrak Synset Offset pada Index Dictionary**

Tahap pertama dari pengekstrakan informasi milik kata adalah melakukan ekstraksi synset yang dimiliki oleh kata tersebut. Ekstraksi ini dilakukan pada kumpulan index dictionary yang dimiliki oleh class WordNet. Kata yang akan dicari akan mengalami pemeriksaan pada setiap class kata, perlu diingat bahwa sebuah kata bisa berada pada lebih dari satu class kata. Apabila kata tersebut ditemukan pada sebuah class kata maka synset offset yang dimiliki oleh kata tersebut akan disimpan. Selain synset offset juga terdapat informasi lain yang dapat diekstrak dari index dictionary, oleh karena itu class WordNet menggunakan struktur penyimpanan berupa class bernama Word untuk menampung hasil ekstraksi pada index dictionary.

**Tabel 4.3**

**Struktur Cass Word**

|  |  |  |
| --- | --- | --- |
| Nama Variabel | Tipe Variabel | Fungsi |
| lemma | NSString | Menyimpan lemma kata |
| noun | WordClass | Menyimpan hasil ekstraksi noun dictionary |
| verb | WordClass | Menyimpan hasil ekstraksi verb dictionary |
| adj | WordClass | Menyimpan hasil ekstraksi adj dictionary |
| adv | WordClass | Menyimpan hasil ekstraksi adv dictionary |

Pada tabel 4.3 ditunjukkan bahwa terdapat 2 tipe variabel pada class Word yaitu NSString dan WordClass. Variabel dengan tipe NSString digunakan untuk menyimpan lemma dari kata sedangkan keempat variabel dengan tipe WordClass digunakan untuk meyimpan informasi dari setiap synset yang dimiliki oleh lemma yang terdapat pada setiap class kata. Setiap variabel pada class Word juga dideklarasikan sebagai property dengan parameter nonatomic dan retain. Parameter nonatomic berarti property tersebut tidak akan mengatasi masalah berupa *race-condition* apabila dilakukan proses multi-threading. Karena WordNet Browser yang dibuat pada Tugas Akhir ini tidak menggunakan multi-threading maka parameter yang digunakan adalah nonatomic. Sedangkan property retain menandakan bahwa property tersebut akan terus tersimpan didalam memory. Dengan adanya deklarasi sebagai property maka variabel-variabel yang dimiliki oleh class Word secara otomatis akan memiliki accessor dan mutator.

Pada struktur class Word ditunjukkan bahwa class Word Menggunakan sebuah class lain sebagai bagiannya, class tersebut adalah class WordClass. Peranan dari class WordClass adalah untuk menampung hasil ekstraksi synset pada setiap class kata, karena itu terdapat empat buah variabel dengan tipe WordClass pada class Word dimana setiap variabel melambangkan satu buah class kata.

**Tabel 4.4**

**Struktur Class WordClass**

|  |  |  |
| --- | --- | --- |
| Nama Variabel | Tipe Variabel | Fungsi |
| wordType | NSString | Menyimpan class kata |
| tagSenseCounter | NSString | Menyimpan jumlah penggunaan lemma |
| synsets | NSMutableArray | Menyimpan synset offset yang dimiliki |
| relations | NSMutableArray | Menyimpan pointer relation yang dimiliki |

Pada tabel 4.4 ditunjukkan bahwa variabel synsets dan relations memiliki tipe NSMutableArray hal ini dilakukan untuk mengatasi kondisi dimana suatu kata memiliki lebih dari satu buah synset dan suatu kata memiliki lebih dari satu buah relasi. Tetapi pada tahap ini masih belum diketahui synset manakah yang memiiliki relasi tersebut. Jumlah sense yang dimiliki oleh kata tidak dicatat karena jumlah sense dari suatu kata pasti bernilai sama dengan jumlah synset yang dimiliki.Setiap variabel yang terdapat pada class WordClass juga dideklarasikan sebagai property dengan parameter nonatomic dan retain.

Proses ekstraksi yang terjadi meliputi pencarian pada keempat index history, pencarian kata akan dilakukan pada setiap index dictionary dan hasilnya akan disimpan dengan struktur class Word. Gambar 4.4 menunjukkan ilustrasi proses ekstraksi yang terjadi.



**Gambar 4.4**

**Ilustrasi Pencarian Kata pada Index Dictionary**

Pada gambar 4.4 ditunjukkan contoh pencarian terhadap kata mobil. Kata tersebut akan dicari pada keempat index dictionary. Hasil dari pencarian tersebut akan diproses dan disimpan dengan menggunakan struktur class WordClass, tetapi apabila tidak ditemukan maka hasil pencarian akan berupa nil. Hasil pencarian yang tidak bernilai nill akan disimpan ke dalam variabel dengan struktur class Word.

Algoritma 4.1 menunjukkan algoritma yang digunakan untuk melakukan ekstraksi synset yang dimiliki oleh suatu kata. Algoritma tersebut sesuai dengan proses yang ditunjukkan oleh gambar 4.4.

**Algoritma 4.1 Ekstraksi Informasi Pada Index Dictionary**

[Digunakan sebagai algoritma dasar untuk ekstraksi synset offset]

1. [Lakukan iterasi untuk mengambil semua synset offset pada semua index dictionary]
   1. FOR EACH dictionary in indexDictionary
      1. IF SearchKata in dictionary
         1. StringData = GET\_DATA(SearchKata)
2. [String Data yang didapatkan akan mengalami proses ekstraksi]
   1. tempSplit = SPLIT\_WITH\_DELIMITER(StringData,” “)
   2. Class = GET\_INDEX(tempSplit,1)
   3. SynsetNumber = GET\_INDEX(tempSplit,2)
   4. RelationNumber = GET\_INDEX(tempSplit,3)
   5. TagSense = GET\_INDEX(tempSplit,RelationNumber+5)
3. [Lakukan ekstraksi synset offset]
   1. FOR (i=0;i<SynsetNumber;i++)
      1. Synset[i] = GET\_INDEX(tempSplit,i+RelationNumber+6)
4. [Lakukan ekstraksi relasi]
   1. FOR (i=0;i<RelationNumber;i++)
      1. Relation[i] = GET\_INDEX(tempSplit,i+4)

Alogitma 4.1 tersebut diwujudkan ke dalam dua buah prosedur. Langkah satu dari algoritma 4.1 ditangani oleh prosedur getWordData sedangkan langkah 2-4 ditangani oleh prosedur getWordClassData. Prosedur getWordClassData adalah sebuah prosedur private yang berperan mengekstrak informasi dari sebuah string yang berasal dari index dictionary. Prosedur ini digunakan oleh prosedur getWordData untuk mengekstrak string yang didapatkan dari suatu index dictionary. Ekstraksi yang dihasilkan berupa synset offset, relasi, class synset dan tag sense counter synset. Sedangkan prosedur getWordData adalah sebuah prosedur bersifat public yang dapat diakses oleh pengguna class WordNet untuk melakukan ekstraksi kata.

**Segmen Program 4.3 Prosedur getWordClassData**

1. -(id) getWordClassData:(NSString \*)stringData
2. {
3. id wordDataType=[[WordClass alloc]init];
4. NSArray \*tempSplit = [stringData componentsSeparatedByString:@" "];
5. int numOfSynsetsID = [[tempSplit objectAtIndex:2] intValue];
6. int numOfRelations = [[tempSplit objectAtIndex:3] intValue];
7. NSMutableArray \*arrayOfSynsets = [[NSMutableArray alloc] init];
8. for(int i=0;i<numOfSynsetsID;i++)
9. {
10. [arrayOfSynsets addObject:[tempSplit objectAtIndex:i+numOfRelations+6]];
11. }
12. NSMutableArray \*arrayOfRelations = [[NSMutableArray alloc] init];
13. for(int i=0;i<numOfRelations;i++)
14. {
15. [arrayOfRelations addObject:[tempSplit objectAtIndex:i+4]];
16. }
17. [wordDataType setWordType:[tempSplit objectAtIndex:1]];
18. [wordDataType setSynsets:arrayOfSynsets];
19. [wordDataType setRelations:arrayOfRelations];
20. [wordDataType setTagSenseCounter:[tempSplit objectAtIndex:numOfRelations+5] ];
21. return [wordDataType autorelease];
22. }

Pada baris 3 segmen program 4.3 disiapkan sebuah variable dengan tipe data WordClass untuk menampung informasi hasil dari ekstraksi string yang menjadi parameter prosedur. Pada baris 4-6 dilakukan pemotongan string berdasarkan spasi dan dilakukan ekstraksi untuk jumlah synset dan jumlah relasi yang dimiliki kata tersebut. Kemudian ekstraksi synset yang dimiliki akan dilakukan berdasarkan jumlah synset yang telah didapatkan, proses ini ditunjukkan pada baris 8-11. Setelah synset didapatkan maka relasi juga akan diekstrak, proses ini ditunjukkan pada baris 13-16. Pada baris 18-21 dilakukan penyimpanan informasi hasil ekstraksi ke dalam variabel penampung.

Prosedur getWordData adalah sebuah prosedur bersifat public yang berperan melakukan ekstraksi pada index dictionary, dengan bantuan prosedur getWordClassData. Prosedur getWordData akan menerima inputan berupa kata dan mencari kata tersebut pada semua index dictionary, apabila sebuah kata terdapat pada suatu index dictionary maka string yang didapatkan akan dikirim kepada prosedur getWordClassData. Hasil ekstraksi akan disimpan pada variabel dengan tipe data class Word. Segmen program 4.4 akan menunjukkan prosedur getWordData.

**Segmen Program 4.4 Prosedur getWordData**

1. -(id)getWordData:(NSString \*)word
2. {
3. id data = [[Word alloc]init];
4. [data setLemma:word];
5. NSString \*stringData=[[NSString alloc]init];
6. if (stringData =

[noun objectForKey:[[data lemma]lowercaseString]])

1. [data setNoun:[self getWordClassData:stringData]];
2. if (stringData =

[verb objectForKey:[[data lemma]lowercaseString]])

1. [data setVerb:[self getWordClassData:stringData]];
2. if (stringData =

[adj objectForKey:[[data lemma]lowercaseString]])

1. [data setAdj:[self getWordClassData:stringData]];
2. if (stringData =

[adv objectForKey:[[data lemma]lowercaseString]])

1. [data setAdv:[self getWordClassData:stringData]];
2. return data;
3. }

Input dari prosedur ini adalah sebuah NSString yang merupakan kata yang ingin dicari, dan mencarinya pada semua class kata. Pada baris 1 segmen program 4.4 disiapkan sebuah variabel dengan tipe Word untuk menampung data hasil ekstraksi. Pada baris 4 kata yang diterima sebagai parameter akan disimpan sebagai lemma kata. Pemeriksaan tersebut akan dilakukan pada setiap index dictionary, apa bila index dictionary tertentu tidak mengembalikan nilai nil untuk key kata yang diinputkan maka string yang diterima akan dikirimkan kepada prosedur getWordClassData dan hasil ekstraksi akan disimpan pada variabel penampung. Proses ini ditunjukkan pada baris 6-13.

Bagi para pengguna class WordNet dapat menggunakan prosedur getWordData untuk melakukan ekstraksi pada index dictionary. Prosedur getWordClass data merupakan sebuah prosedur private yang tidak dapat oleh pengguna class WordNet.

* + 1. **Mencari Synset Offset dan Mengekstrak Informasi pada Data, Relation dan Lexname Dictionary**

Tahap kedua dari proses ekstraksi informasi adalah mengekstrak informasi yang dimiliki oleh synset tersebut pada data dictionary, relation dictionary dan lex dictionary. Setiap synset yang dimiliki oleh kata akan dicari pada data dictionary yang sesuai dengan synset tersebut. Pada subbab 4.2.1 dijelaskan bahwa ekstraksi synset offset juga akan menghasilkan class kata tempat synset tersebut berada. Pencarian pada data dictionary akan menghasilkan gloss, sinonim, lexFileNum dan pointer relation yang dimiliki oleh synset. Pointer relation yang didapatkan akan menjadi key dicari pada relation dictionary untuk mengetahui jenis relasi yang disimbolkan. Sedangkan lexFileNum akan menjadi key yang dicari pada lex dictionary untuk mengetahui lexname dari synset tersebut. Untuk menampung seluruh informasi yang dimiliki oleh synset maka class WordNet menggunakan struktur class bernama Synset. Tabel 4.5 menunjukkan struktur data yang dimiliki oleh class Synset.

**Tabel 4.5**

**Struktur Class Synset**

|  |  |  |
| --- | --- | --- |
| Nama Variabel | Tipe Variabel | Fungsi |
| lexFileNum | NSString | Menyimpan lexFileNum |
| ssTypes | NSString | Menyimpan class kata synset |
| synsetWords | SynsetWord | Menyimpan sinonim synset |
| synsetRelations | NSMutableArray | Menyimpan relasi synset |

Untuk menyimpan informasi mengenai sinonim yang dimiliki oleh synset maka class Synset menggunakan bantuan dari class SynsetWord. Hal ini dilakukan karena dalam proses ekstraksi sinonim terdapat beberapa informasi lain yang bisa didapatkan. Sedangkan untuk menyimpan informasi mengenai relasi yang dimiliki oleh synset digunakan bantuan dari class SynsetRelation, variabel-variabel dengan tipe SynsetRelation akan disimpan pada array dengan nama synsetRelations milik class Synset. Setiap variabel pada class Synset dideklarasikan sebagai property dengan tipe nonatomic dan retain, hal ini juga terjadi pada class SynsetWord dan SynsetRelation.

**Tabel 4.6**

**Struktur Class SynsetWord**

|  |  |  |
| --- | --- | --- |
| Nama Variabel | Tipe Variabel | Fungsi |
| gloss | NSString | Menyimpan gloss synset |
| example | NSString | Menyimpan example |
| synsetLemma | NSMutableArray | Menyimpan lemma sinonim synset |
| lexId | NSMutableArray | Menyimpan lexId milik lemma sinonim |

Pada tabel 4.6 ditunjukkan bahwa selain menyimpan informasi mengenai sinonim class SynsetWord juga menyimpan informasi mengenai gloss dan example yang dimiliki oleh synset. Class SynsetWord juga mampu mengatasi kondisi dimana apabila synset memiliki lebih dari satu buah sinonim dengan memanfaatkan array untuk menyimpan lemma yang didapatkan.

**Tabel 4.7**

**Struktur Class SynsetRelation**

|  |  |  |
| --- | --- | --- |
| Nama Variabel | Tipe Variabel | Fungsi |
| synsetRelationType | NSString | Menyimpan tipe relasi |
| synsetRelationOffset | NSString | Menyimpan synset offset relasi |
| synsetTargetClass | NSString | Menyimpan class kata relasi |
| synsetLemmaOriginalIndex | NSString | Menyimpan id lemma asal |
| synsetLemmaTargetIndex | NSString | Menyimpan id lemma tujuan |

Tabel 4.7 menunjukkan variabel synsetLemmaOriginalIndex dan synsetLemmaTargetIndex akan menyimpan index dari lemma asal dan index lemma dari synset pemilik relasi yang saling berhubungan. Apabila relasi bersifat sematik maka kedua variabel akan bernilai 00.



**Gambar 4.5**

**Ilustrasi Ekstraksi Informasi**

Proses ekstrasi informasi yang dimiliki oleh synset dimulai dengan melakukan ekstraksi pada data dictionary. Pada gambar 4.5 ditunjukkan proses ekstraksi dari synset 05269933. Synset tersebut akan dicari ke dalam nounData dictionary, hasil dari ekstraksi ini berupa sinonim, gloss, lexFileNum dan relasi yang dimiliki. Sinonim, gloss dan example akan disimpan dengan menggunakan struktur class SynsetWords. Ekstraksi sinonim, gloss dan example ini tidak melibatkan lex dan relation dictionary, berbeda dengan ekstraksi lexFileNum dan ekstraksi relasi. Pada ekstraksi relasi akan didapatkan relation pointer yang dimiliki oleh synset, untuk mengetahui jenis dari relasi tersebut maka perlu dilakukan pencarian pada relation dictionary, hasil pencarian itulah yang kemudian akan disimpan ke dalam variabel synsetRelationType milik class SynsetRelation. Pada ekstraksi relasi terdapat juga ekstraksi index pemilik relasi apabila nlai dari index tidak 00 maka akan terjadi pencarian ke dalam synsetlemma untuk mendapatkan lemma pemilik index. Demikian juga pada ekstraksi lexFileNum, ekstraksi ini hanya mendapatkan kode angka dari lexnames.

Setiap proses yang ditunjukkan pada gambar 4.4 ditangani oleh sebuah prosedur yang disediakan oleh class WordNet. Untuk melakukan ekstraksi pada data dictionary digunakan prosedur getSynsetData, untuk melakukan ekstraksi pada relation dictionary digunakan prosedur getRelationName dan untuk melakukan ekstraksi pada lex dictionary digunakan prosedur getLexName. Penerapan gambar 4.4 ditunjukkan oleh algoritma 4.2.

Algoritma 4.2 menunjukkan algoritma yang digunakan untuk melakukan ekstraksi pada data dictionary. Algoritma ini ditangani oleh sebuah prosedur bernama getSynsetData. Prosedur getSynsetData memerlukan dua buah parameter dengan tipe string. Parameter pertama merupakan synset offset yang ingin diekstrak sedangkan parameter kedua merupakan class kata tempat synset tersebut berada. Prosedur getSynsetData terbagi menjadi tiga tahap, tahap pertama adalah ekstraksi lexFileNum dan sinonim. Tahap kedua adalah ekstraksi gloss dan example, tahap ketiga adalah ekstraksi relasi. Setiap tahap tersebut akan dibahas secara detail disertai dengan contoh data yang dihasilkan.

**Algoritma 4.2 Ekstraksi Informasi Pada Data Dictionary**

[Digunakan sebagai algoritma dasar untuk ekstraksi informasi yang dimiliki oleh synset offset]

1. [Periksa semua dictionary dan ambil data milik synset offset]
   1. FOR EACH dictionary in indexDictionary
      1. IF SynsetOffset in dictionary
         1. StringData = GET\_DATA(SearchKata)
2. [String Data yang didapatkan akan mengalami proses ekstraksi]
   1. dataSplit = SPLIT\_WITH\_DELIMITER(StringData,”|“)
   2. tempSplit = SPLIT\_WITH\_DELIMITER(StringData,” “)
   3. LexFileNum = GET\_INDEX(tempSplit,1)
   4. Type = GET\_INDEX(tempSplit,2)
   5. SynonymNum = GET\_INDEX(tempSplit,3)
   6. RelationIndex= SynonymNum\*2+4
   7. RelationNum = GET\_INDEX(tempSplit,RelationIndex)
3. [Lakukan ekstraksi lemma synonym]
   1. Index=0
   2. FOR (i=0;i<SynonymNum;i++)
      1. Lemmas[i] = GET\_INDEX(tempSplit,index+4)
      2. ID[i] = GET\_INDEX(tempSplit,i+5)
      3. Index+=2
4. [Lakukan ekstraksi gloss dan example]
   1. glossSplit = SPLIT\_WITH\_DELIMITER(stringData,”;”)
   2. FOR (i=0;i<glossSplit Size;i++)
      1. IF (glossSplit[i] contain “\”)
         1. Gloss+=glossSplit[i]
      2. ELSE
         1. Example+=glossSplit[i]
5. [Lakukan ekstraksi relasi synonym]
   1. Index=0
   2. FOR (i=0;i<RelationNum;i++)
      1. tempIndex = index+RelationIndex
      2. RelationPointer = GET\_INDEX(tempSplit, tempIndex +1)
      3. RelationName = GET\_REL\_NAME(RelationPointer)
      4. RelationOffset = GET\_INDEX(tempSplit, tempIndex +2)
      5. OffsetClass = GET\_INDEX(tempSplit tempIndex +3)
      6. RelationTemp = GET\_INDEX(tempSplit tempIndex +4)
      7. RelationOri = SUBSTRING\_FROM\_TO(RelationTemp,0,1)
      8. RelationTar = SUBSTRING\_FROM\_TO(RelationTemp,2,3)
      9. Index+=4

Segmen program 4.5 menunjukkan tahap pertama dari prosedur getSynsetData yaitu ekstraksi sinonim dan lexFileNum. Pada tahap ini digunakan class WordData untuk menampung informasi hasil ekstraksi, selain itu juga disiapkan class Synsets sebagai penampung keseluruhan informasi.

**Segmen Program 4.5 Ekstraksi Sinonim & LexFileNum**

1. id data = [[Synsets alloc]init];
2. NSString \*stringData;
3. if ([Dict isEqualToString:@"noun"])
4. stringData = [nounData objectForKey:synsetOffset];
5. if ([Dict isEqualToString:@"verb"])
6. stringData = [verbData objectForKey:synsetOffset];
7. if ([Dict isEqualToString:@"adj"])
8. stringData = [adjData objectForKey:synsetOffset];
9. if ([Dict isEqualToString:@"adv"])
10. stringData = [advData objectForKey:synsetOffset];
11. NSArray \*synsetGlossExampleSplit =

[stringData componentsSeparatedByString:@"|"];

1. NSArray \*dataSplit = [[synsetGlossExampleSplit objectAtIndex:0]componentsSeparatedByString:@" "];
2. [data setLexFileNum:[dataSplit objectAtIndex:1]];
3. [data setSsTypes:[dataSplit objectAtIndex:2]];
4. id wordsData = [[SynsetWords alloc]init];
5. NSMutableArray \*lemmas = [[NSMutableArray alloc]init];
6. NSMutableArray \*lexIds = [[NSMutableArray alloc]init];
7. int x=0;
8. NSString \*hexStr = [dataSplit objectAtIndex:3];
9. NSScanner \*scanner = [NSScanner scannerWithString:hexStr];
10. unsigned int outVal ;
11. [scanner scanHexInt:&outVal];
12. for(int j=0;j<outVal;j++)
13. {
14. [lemmas addObject:[dataSplit objectAtIndex:x+4]];
15. [lexIds addObject:[dataSplit objectAtIndex:x+5]];
16. x+=2;
17. }
18. [wordsData setLexId:lexIds];
19. [wordsData setSynsetLemma:lemmas];

Pada baris 3-10 dari segmen program 4.5 dilakukan pemeriksaan dari class kata manakah synset offset berasal lalu dilakukan pencarian pada data dictionary yang sesuai dan hasil pencarian disimpan pada variabel stringData. Seteleh data didapatkan maka langkah pertama adalah memisahkan bagian yang mengandung gloss dan example dengan bagian yang mengandung sinonim dan relasi kata. Kedua bagian tersebut dipisahkan oleh sebuah simbol “|” proses ini ditunjukkan pada baris 11. Contoh dari data adalah : *00001740 03 n 01 sepelarian 0 000 | sejauh berlari tentang jarak* . Kumpulan string yang berada seteleh simbol “|” merupakan gloss yang dimiliki oleh kata.

Setelah pemisahan bagian dilakukan maka pertama-tama pengekstrakan sinonim akan dilakukan. Pada baris 12 bagian yang mengandung sinonim dan relasi akan dipisah berdasarkan karakter spasi lalu disimpan ke dalam variabel dataSplit. Seperti yang telah dijelaskan pada bab 2 bahwa setiap field pada data file (dalam Tugas Akhir ini telah dipindahkan ke dalam data dictionary) bahwa setiap field akan dipisahkan berdasarkan karakter spasi. Hasil ekstraksi yang bisa didapatkan adalah kode lexnames dari synset dan class kata dari synset, proses ini ditunjukkan pada baris 13 dan 14. Proses selanjutnya adalah melakukan ekstraksi sinonim yang dimiliki oleh synset beserta id dari tiap lemma sinonim tersebut, id pada kata tersebut melambangkan jumlah kata. Untuk melakukan ekstraksi sinonim pertama-tama perlu diketahui jumlah dari sinonim yang dimiliki, pada baris 29-22 dilakukan ekstraksi jumlah sinonim yang dimiliki (jumlah diwakilkan 2 digit bilangan hexadecimal) serta konversi dari bilangan hexadecimal ke bilangan integer. Konversi ini memanfaatkan variabel dengan tipe NSScanner.

Pada baris 20 scanner akan menerima inputan berupa sebuah string yang merupakan bilangan hexadesimal. Pada baris 22 scanner melakukan konversi dari bilangan hexadesimal dan menyimpannya ke dalam variabel outVal yang bertipe integer. Setelah jumlah dari sinonim telah didapatkan maka langkah selanjutnya adalah mengambil setiap lemma beserta id yang dimiliki lalu menyimpannya ke dalam array. Proses ini ditunjukkan pada baris 23-28, pengambilan lemma sinonim dan id akan berselang 2 field karena posisi lemma dan id bersebelahan misalnya *0c anak\_adam 2 bani\_adam 0*, 0c adalah jumlah sinonim sedangkan anak\_adam dan bani\_adam adalah lemma, 2 dan 0 merupakan id dari masing-masing lemma. Pada saat tahap ini selesai dilakukan maka data yang sudah didapatkan adalah lexFileNum, sinonim, dan ssType. Ilustrasi dari kondisi informasi ditunjukkan oleh gambar 4.6 dimana ditampilkan informasi berupa lemma sinonim, id, lexFileNum dan ssTypes yang disimpan oleh class Synset dan SynsetWord. Ditunjukkan pada gambar 4.6 bahwa informasi berupa gloss dan example masih belum didapatkan.



**Gambar 4.6**

**Hasil ekstraksi tahap pertama**

Untuk melengkapi data dari synsetWords maka tahap kedua dari prosedur akan melakukan ekstraksi gloss dan example yang dimiliki oleh synset. Setelah ekstraksi selesai dilakukan maka informasi pada synsetWords telah lengkap dan langkah akhir yang harus dilakukan adalah menyimpan synsetWords ke dalam class Synset. Tahap ketiga dari ekstraksi pada data dictionary baru akan dilaksanakan setelah gloss, example, sinonim telah didapatkan dan disimpan pada class Synset.

Proses ekstraksi yang ditunjukkan oleh segmen program 4.6 adalah ekstraksi gloss dan example apabila tersedia. Pada baris 3 ditunjukkan bahwa bagian yang mengandung gloss dan example akan dipisah berdasarkan simbol “;”. Tetapi sebuah synset dapat memiliki lebih dari satu example dan setiap example tersebut juga dipisahkan oleh simbol “;”. Kemungkinan kedua adalah gloss tersebut terbagi menjadi beberapa bagian dan setiap bagiannya dipisahkan oleh simbol “;” juga. Untuk membedakan apakah suatu field merupakan bagian dari gloss atau bagian dari example maka dilakukan sebuah pemeriksaan, sebuah example selalu diawali dan diakhiri oleh tanda petik (“) oleh karena itu apabila ditemukan tanda petik pada suatu field maka field tersebut merupakan bagian dari example tetapi jika tidak ditemukan maka field tersebut merupakan bagian dari gloss.

**Segmen Program 4.6 Ekstraksi Gloss & Example**

1. NSString \*gloss=[[NSString alloc]init];
2. NSString \*example=[[NSString alloc]init];
3. NSArray \*glossSplit = [[synsetGlossExampleSplit objectAtIndex:1]componentsSeparatedByString:@";"];
4. for(int i=0;i<[glossSplit count];i++)
5. {
6. if ([[glossSplit objectAtIndex:i]rangeOfString:@"\""].location == NSNotFound)
7. {
8. gloss=[gloss stringByAppendingFormat:@"%@ ",[glossSplit objectAtIndex:i]];
9. }
10. else {
11. example=

[example stringByAppendingFormat:@"%@ ",[glossSplit objectAtIndex:i]];

1. }
2. }
3. [wordsData setGloss:gloss];
4. [wordsData setExample:example];
5. [data setSynsetWords:wordsData];

Proses ekstraksi gloss dan example ini ditunjukkan pada baris 3-13 segmen program 4.6. Pada baris 6 dilakukan pemeriksaan apakab tanda petik berada pada field yang sedang diperiksa, pemeriksaan ini memanfaatkan prosedur rangeOfString milik class NSString. Karena sebuah string pada Objective-C harus diawali simbol @” dan diakhiri simbol “ maka untuk melakukan pencarian terhadap tanda petik harus diawali dengan sebuah garis miring (@”\””). Hasil ekstraksi gloss dan example ditampung pada sebuah variabel string dan variabel string tersebut akan disimpan oleh variabel wordData, wordData tersebut pada akirnya akan ditampung oleh variable synsetWords pada class Synset. Setelah tahap ini selesai dilakukan maka sinonim, gloss dan example dari synset telah berhasil didapatkan, ilustrasi kondisi ekstraksi ditunjukkan oleh gambar 4.7. Pada saat tahap kedua selesai dilakukan, informasi yang belum diekstrak hanyalah relasi yang dimiliki oleh synset. Ekstraksi ini akan melibatkan pencarian pada relation dictionary. Hasil dari ekstraksi tahap ketiga adalah array yang mengandung variabel dengan tipe class synsetRelation. Variabel-variabel tersebut akan mengandung informasi mengenai relation pointer, synset pemilik relasi dan juga index lemma pemilik relasi.



**Gambar 4.7**

**Hasil ekstraksi tahap kedua**

Detail dari ekstraksi relation dictionary ditunjukkan oleh segmen program 4.8. Relation Pointer akan menjadi parameter yang digunakan untuk melakukan pencarian pada relation dictionary. Jenis relasi yang didapat dari hasil pencarian itulah yang akan disimpan ke dalam variabel synsetRelation. Segmen program 4.7 menunjukkan tahap ketiga ekstraksi.

Proses ekstraksi yang ditunjukkan oleh segmen program 4.7 adalah proses ekstraksi relasi yang dimilki oleh synset, proses ini akan memanfaatkan class SynsetRelation untuk menampung hasil ekstraksi. Untuk melakukan ekstraksi relasi maka terlebih dahulu harus diketahui jumlah relasi yang dimiliki oleh synset, jumlah relasi dilambangkan dengan tiga digit bilangan integer. Pada baris 3 ditunjukkan sebuah variabel *y* dengan nilai (outVal\*2+4) . Variabel outVal dikalikan dengan 2 karena outVal merupakan jumlah sinonim yang dimiliki sedangkan setiap sinonim terdiri dari lemma dan LexId. Sedangkan ditambahkan dengan empat karena index array yang mengandung sinonim dan relasi dimulai dari index keempat dari array dataSplit. Variabel y ini merupakan index field yang mengandung jumlah relasi yang dimiliki, jika tidak memiliki relasi apapun maka field ini akan bernilai 0. Pada baris 7-9 dilakukan ekstraksi tipe relasi yang dimilki, synset offset pemilik relasi tersebut dan class kata dari synset tersebut. Sedangkan pada baris 10-14 dilakukan ekstraksi index asal dan target lemma. Apabila field ini bernilai 0000 berarti relasi bersifat semantik dan semua lemma dari synset asal dan tujuan memiliki relasi ini. Tetapi bila bernilai selain 0000 berarti relasi bersifat lexical dimana relasi ini hanya dimiliki oleh salah satu lemma dalam synset asal dengan salah satu lemma pada target synset. Lemma pertama pada synset memiliki index satu. Ekstraksi relasi mengakhiri proses ekstraksi pada data dictionary, kondisi informasi yang telah diekstrak ditunjukkan oleh gambar 4.8.

**Segmen Program 4.7 Ekstraksi Relasi Synset**

1. NSMutableArray \*relations = [[NSMutableArray alloc]init];
2. x=0;
3. int y=(outVal\*2+4);
4. for(int j=0;j<[[dataSplit objectAtIndex:y]intValue];j++)
5. {
6. SynsetRelations \*synsetRelation = [[SynsetRelations alloc]init];
7. [synsetRelation setSynsetRelationType:[self getRelationName:[dataSplit objectAtIndex:x+y+1]]];
8. [synsetRelation setSynsetRelationOffset:[dataSplit objectAtIndex:x+y+2]];
9. [synsetRelation setSynsetTargetClass:[dataSplit objectAtIndex:x+y+3]];
10. NSString \*ori, \*tar;
11. ori=[[dataSplit objectAtIndex:x+y+4] substringToIndex:2];
12. tar = [[dataSplit objectAtIndex:x+y+4] substringWithRange:NSMakeRange(2,2)];
13. [synsetRelation setSynsetLemmaOriginalIndex:ori];
14. [synsetRelation setSynsetLemmaTargetIndex:tar];
15. [relations addObject:synsetRelation];
16. x+=4;
17. }
18. [data setSynsetRelations:relations];
19. return data;

Pada gambar 4.8 ditunjukkan salah satu relasi milik synset yang merupakan hasil ekstraks tahap ketiga. Array yang menampung relasi-relasi hasil ekstraksi akan disimpan ke dalam variabel synsetRelations milik class Synset. Pada tahap ini semua relasi milik synset telah berhasil diekstrak. Untuk lexname milik synset akan disimpan dalam bentuk lexFileNum, apabila user memerlukan lexname synset maka user dapat memanfaatkan prosedur getSynsetName milik class WordNet.



**Gambar 4.8**

**Hasil ekstraksi tahap ketiga**

Pada baris 7 segmen program 4.7 ditunjukkan bahwa prosedur getSynsetData memanfaatkan prosedur getRelationName untuk mengetahui jenis relasi yang dimiliki. Prosedur tersebut memanfaatkan relation pointer yang didapatkan untuk melakukan pencarian pada relation dictionary. Segmen program 4.8 menunjukkan detail dari prosedur getRelationName.

**Segmen Program 4.8 Prosedur getRelationName**

1. -(id) getRelationName:(NSString \*) relID
2. {
3. NSString \*relName = [[NSString alloc]init];
4. relName = [relation objectForKey:relID];
5. return relName;
6. }

Prosedur getRelationName merupakan sebuah prosedur bersifat public yang berperan untuk mendapatkan jenis relasi berdasarkan relation pointer yang diinputkan. Input dari prosedur ini berupa string yang merupakan relation pointer dan output dari prosedur ini berupa sebuah string yang merupakan jenis relasi dari relation pointer yang diinputkan. Baris 4 dari segmen program 4.7 menunjukkan proses mendapatkan value dari dictionary dimana key yang digunakan adalah input yang diterima. Pada baris 5 value tersebut akan diberikan pada user.

Untuk melakukan ekstraksi lexname maka class WordNet menyediakan prosedur getLexName. Prosedur getLexName merupakan sebuah prosedur bersifat public dan berperan untuk mendapatkan lexnames dari suatu lexId tertentu. Input dari prosedur ini berupa string yang merupakan lexId dan output dari prosedur ini berupa sebuah string yang merupakan lexname dari lexId yang diinputkan. Segmen program 4.9 menunjukkan detail dari prosedur getLexName.

**Segmen Program 4.9 Prosedur getLexName**

1. -(id) getLexName:(NSString \*) lexId
2. {
3. NSString \*temp = [[NSString alloc]init];
4. temp=[lex objectForKey:lexId];
5. NSArray \*lexSplit = [temp componentsSeparatedByString:@"\t"];
6. return [lexSplit objectAtIndex:1];
7. }

Prosedur getLexName akan mengakses lex dictionary dan mengambil value yang dimiliki oleh dengan nilai lexId, proses ini ditunjukkan pada baris 4 segmen program 4.9. Setelah mendapatkan value tersebut maka dilakukan pemotongan string berdasarkan tab atau “\t” dab disimpan pada sebuah array lexSplit. Prosedur akan mengembalikan value yang berada pada index 1 dari array. Index pertama dari array akan mengandung lexId dari lexname tersebut, lexId ini bernilai sama dengan lexId yang diinputkan. Format dari file lexnames dijelaskan secara detail pada bab II. Proses pemotongan string dan pengembalian data ditunjukkan oleh baris 5 dan 6.

* + 1. **Mendapatkan Lemma Sinonim pada Index Tertentu**

Apabila suatu synset memiliki hubungan lexical maka perlu diketahui lemma sinonim yang memiliki relasi tersebut. Dalam suatu relasi semantik tidak semua lemma pada sinonim mengandung hubungan tersebut hanya suatu lemma tertentu, hal ini juga berlaku pada synset tujuan dari relasi. Pada gambar 4.9 ditunjukkan relasi antonim antara synset 03258365 dengan synset 00353392. Synset 03258365 memiliki dua buah sinonim tetapi karena relasi semantik hanya dimiliki oleh salah satu lemma maka perlu diketahui lemma yang dimaksud.



**Gambar 4.9**

**Relasi Leksikal**

Untuk mengatasi hal ini maka class WordNet menyediakan sebuah prosedur bernama getSynsetLemma untuk mendapatkan lemma yang memiliki relasi tersebut, index lemma didapatkan pada saat ekstraksi relasi dan tersimpan pada class SynsetRelation. Prosedur getSynsetLemma merupakan sebuah prosedur bersifat public dan berperan untuk mengambil sebuah lemma pada index tertentu yang dimiliki oleh sebuah synset. Prosedur ini digunakan ketika user ingin mengetahui kata yang memiliki suatu relasi tertentu dengan kata yang sedang dicari. Apabila field source/target dari suatu relasi bernilai 0000 maka prosedur ini akan mengambil lemma sinonim pertama dari synset tetapi bila field tersebut memiliki index tertentu maka prosedur akan mengambil lemma pada posisi yang dimaksud. Segmen program 4.10 menunjukkan detail dari prosedur getSynsetLemma.

Input dari prosedur ini berupa synset offset, class kata tempat synset berada dan index lemma yang diinginkan. Output yang dihasilkan adalah sebuah string yang merupakan lemma sinonim yang berada pada yang diinginkan pada suatu synset. Prosedur getSynsetLemma memiliki kemiripan proses dengan prosedur getSynsetData hanya saja proses yang dijalankan tidak sebanyak proseudr getSynsetData.

**Segmen Program 4.10 Prosedur getSynsetLemma**

1. -(id) getSynsetLemma:(NSString \*) synsetOffset

forDict:(NSString \*)Dict forIndex:(NSString \*)index;

1. {
2. id data = [[NSString alloc]init];
3. NSString \*stringData;
4. if ([Dict isEqualToString:@"noun"])
5. stringData = [nounData objectForKey:synsetOffset];
6. if ([Dict isEqualToString:@"verb"])
7. stringData = [verbData objectForKey:synsetOffset];
8. if ([Dict isEqualToString:@"adj"])
9. stringData = [adjData objectForKey:synsetOffset];
10. if ([Dict isEqualToString:@"adv"])
11. stringData = [advData objectForKey:synsetOffset];
12. NSArray \*synsetGlossExampleSplit = [stringData componentsSeparatedByString:@"|"];
13. NSArray \*dataSplit =[[synsetGlossExampleSplit objectAtIndex:0]componentsSeparatedByString:@" "];
14. int x=0;
15. NSString \*hexStr = [dataSplit objectAtIndex:3];
16. NSScanner \*scanner = [NSScanner scannerWithString:hexStr];
17. unsigned int outVal ;
18. [scanner scanHexInt:&outVal];
19. if([index isEqualToString:@"00"])
20. index=@"01";
21. for(int j=0;j<outVal;j++)
22. {
23. if (j+1 == [index intValue])
24. data=[[dataSplit objectAtIndex:x+4]retain];
25. x+=2;
26. }
27. return data;
28. }

Baris 4-19 segmen program 4.10 juga diilakukan pada prosedur getSynsetData. Perbedaan dari kedua prosedur dimulai dari baris 20, proses-proses tersebut hanya dilakukan oleh prosedur getSynsetLemma. Baris 20 merupakan proses pemeriksaan apakah index yang diberikan bernilai 00, jika benar maka prosedur akan menetapkan bahwa index yang akan diambil adalah index yang pertama. Pada baris 22 dilakukan iterasi sejumlah sinonim yang dimiliki oleh synset, pada baris 24 dilakukan pengecekan apakah iterasi berada pada index yang diinginkan jika benar maka lemma sinonim pada posisi tersebut akan diambil.

* 1. **Mendapatkan Daftar Kata pada Lexical Database Files**

Daftar kata yang terdapat pada Lexical Database Files dapat didapatkan dengan memanfaatkan kelompok dictionary. Daftar kata yang didapatkan tersebut akan menjadi daftar kata yang ditampilkan oleh browser pada halaman search serta untuk menjalankan fitur auto complete. Karena proses pengurutan daftar kata memakan waktu yang cukup lama maka hasil daftar kata yang didapatkan akan disimpan ke dalam sebuah plist. Segmen program 4.15 menunjukkan proses yang terjadi.

**Segmen Program 4.11 Proses Mendapatkan Daftar kata**

1. NSFileManager \*fileManager = [NSFileManager defaultManager];
2. if(![fileManager fileExistsAtPath:listPlistFilePath])
3. {
4. listOfAvailableWords = [[NSMutableArray alloc]init];
5. NSMutableDictionary \*tempList = [[[NSMutableDictionary alloc]init]retain];
6. for(NSString \*str in noun)
7. [tempList setObject:str forKey:str];
8. for(NSString \*str in verb)
9. [tempList setObject:str forKey:str];
10. for(NSString \*str in adj)
11. [tempList setObject:str forKey:str];
12. for(NSString \*str in adv)
13. [tempList setObject:str forKey:str];
14. for(NSString \*str in tempList)
15. {
16. str=[str stringByReplacingOccurrencesOfString:@"\_" withString:@" "];
17. [listOfAvailableWords addObject:str];
18. }
19. NSArray \*sorted = [listOfAvailableWords sortedArrayUsingSelector:@selector(compare:)];
20. listOfAvailableWords = [sorted mutableCopy];
21. [tempList release];
22. [listOfAvailableWords writeToFile:listPlistFilePath atomically:YES];
23. }
24. else {
25. listOfAvailableWords = [[NSMutableArray alloc]initWithContentsOfFile:listPlistFilePath];
26. }

Pada baris 1 dan 2 segmen program 4.11 ditunjukkan bahwa pada proses mendapatkan daftar kata melakukan pemeriksaan terhadap sebuah file plist bernama listPlist. File dengan ekstensi plist adalah sebuah text file yang digunakan oleh Objective-C, struktu penyimpanan file ini berbentuk sebuah xml. File listPlist berperan menampung daftar kata yang menjadi hasil proses ini. Apabila file tersebut tidak ditemukan pada folder document milik device berarti proses mendapatkan daftar kata belum pernah terjadi, tetapi apabila file tersebut ditemukan maka class WordNet akan melakukan pembacaan daftar kata dari file plist tersebut. Proses pembacaan daftar kata dari file plist ditunjukkan oleh baris 25. Tetapi bila file plist tidak ditemukan proses akan berlanjut pada baris 4. Pada baris 5 disiapkan sebuah dictionary yang akan menapung semua kata yang terdapat pada keempat index dictionary.

Proses pembacaan keempat dictionary ditunjukkan pada baris 6-13, setiap index dictionary akan mengalami iterasi dan key pada setiap index dictionary akan menjadi key dan value pada dictionary penampung daftar kata. Pada tahap ini digunakan dictionary untuk menampung daftar kata agar kata-kata yang terdapat pada lebih dari satu index dictionary dapat dieliminasi secara otomatis. Setelah daftar kata yang berada pada keempat index dictionary didapatkan maka langkah selanjutnya adalah melakukan penggantian string underscore (\_) menjadi string spasi ( ), hal ini dilakukan karena daftar kata ini akan disajikan oleh WordNet Browser pada user sebagai bagian dari fitur auto complete. String yang mengalami proses penggantian tersebut akan disimpan ke dalam sebuah array, karena key dan value pada dictionary penampung memiliki nilai yang sama maka hanya key yang akan diproses dan disimpan ke dalam array. Proses tersebut ditunjukkan pada baris 14-18, dimana pada baris 16 dilakukan proses penggantian string dan pada baris 17 dilakukan penyimpanan ke dalam sebuah array.

Kata yang tersimpan pada array masih belum terurut sesuai dengan abjad karena itu dilakukan proses pengurutan daftar kata. Proses pengurutan ini memanfaatkan sebuah prosedur milik Objective-C dimana kata dalam array akan diurutkan sesuai abjad. Prosedur tersebut ditunjukkan oleh baris 19. Langkah terakhir yang perlu dilakukan adalah menyimpan daftar kata tersebut ke dalam sebuah plist. Dengan menyimpan daftar kata ke dalam plist maka ketika class WordNet melakukan inisialisasi lagi proses mendapatkan daftar kata tidak perlu dilakukan cukup dengan melakukan pembacaan plist, hal ini akan mempercepat waktu inisialisasi dari class WordNet.