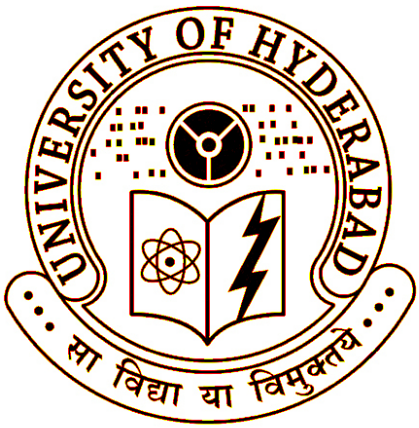
KSS Word Index Engine

V.S.S.Krishna (11MCMT35)
A.Siva Krishna Reddy (11MCMT02)
V.Subba Reddy (11MCMT03)

Adviser: Anupama Potluri

April, 10 2012

Dept. of Computer Science and Information Sciences
University of Hyderabad



Contents

Chapter 1

Software Requirements Specifications

1.1 INTRODUCTION

1.1.1 product overview

Word index engine is a software like search engine, based on skip-list.
It offers two type of searching.

- Word based search.
- Frequency based search.

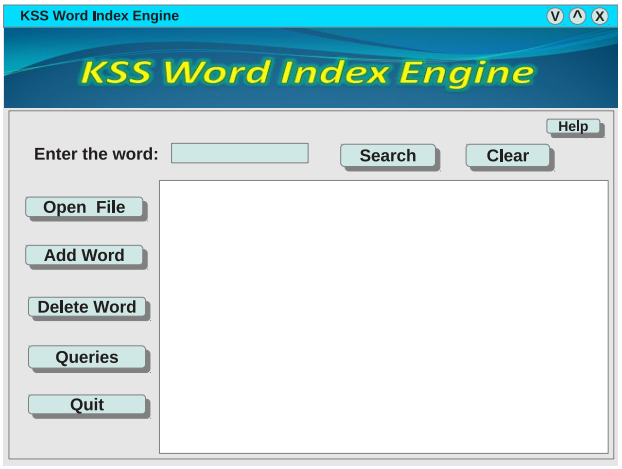
It offers users to add and delete words through a beautiful Graphical Use Interface(GUI).
The Skiplist is used to store data, which is an advanced data structure, in which searching of an element in the list can be done with a time complexity of "log n".
It takes input as a file, using which the dictionary is built up. The dictionary grows when ever new file is opened.

1.2 SPECIFIC REQUIREMENTS

1.2.1 External interface requirements

User Interface

Inputs search word from keyboard



Software Interfaces

- Language : Java
- Version : jdk 1.6
- Operating System : Linux

Hardware Interfaces

- Processor : 500MHZ or Above.
- RAM : 500 MB.
- Hard Disk : 64GB.

1.2.2 Software System Attributes

Releability

No failure.

Availability

This system is available at all time.

Performance

KSS Word Index Engine gives best performance as far as it can.

1.2.3 Database Requirements

An input text file. All words are taken into account, except some sequences which contain numbers and special symbols.

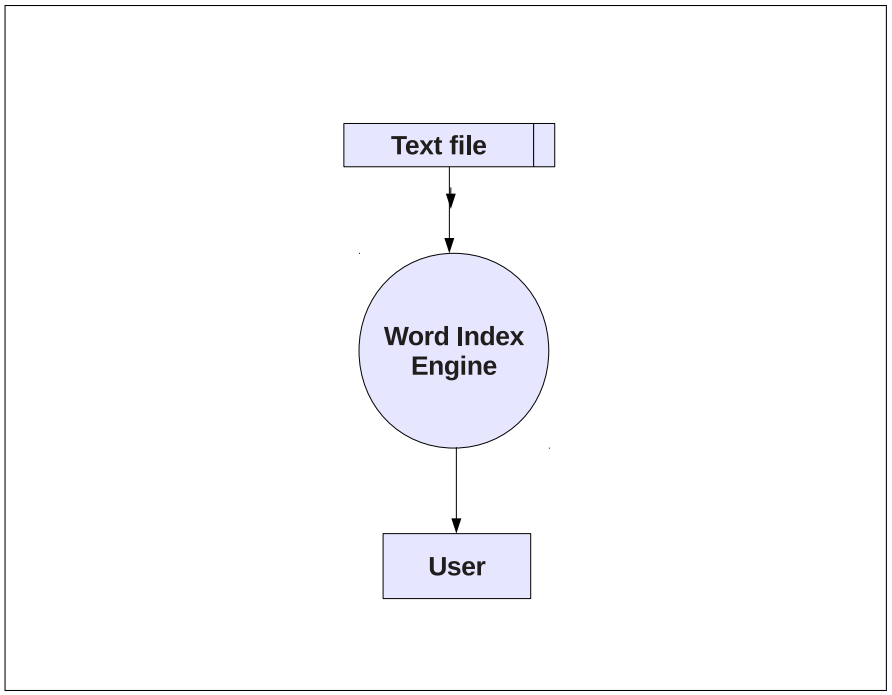
1.3 ADDITIONAL MATERIAL

A user manual is provided.

Chapter 2

High Level Design

2.1 High Level Architecture

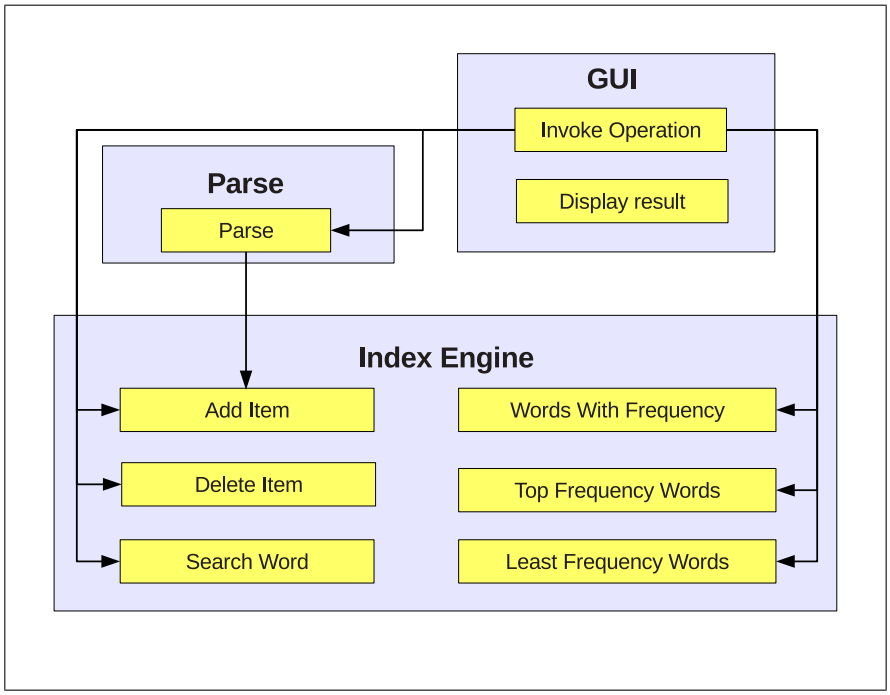


2.2 Software Interface Specifications (API)

2.2.1 Modules of the architecture

:

1. Parse.
2. Index Engine.
3. Graphical User Interface (GUI).



Parse

Functionality : This module gets a text file as input, and parses the text file.

Interface Description :

- Int parse(FileInputStream file)
  Purpose : To parse the given text file.
  Called by : GUI module.
  Calls : Index Engine.
  file : Reference for the text file.
  Returns : Status of success.

Index Engine

Functionality : This module processes all the operations required to develop and modify the Dictionary, to search the given word and to process frequency based queries.

Interface Description :

- Item addItem(Item item)
  Purpose : To start building the dictionary and to add item, which have word and frequency.
  Called by : Parse, Graphical User Interface.
  Calls : Nothing.
  item : The item to be added into the dictionary.
  Returns : Returns item if success, null on failure.

- Item deleteItem(Item item)
  Purpose : To delete item from the dictionary.
  Called by : Graphical User Interface.
  Calls : Nothing.
  item : The item to be deleted form the dictionary.
  Returns : Returns item if success, null on failure.

- Item search(String word)
  Purpose : To search words in the dictionary.
  Called by : Graphical User Interface.
  Calls : Nothing.
  word : The item which contains the word to be searched.
  Returns : Returns item if found or null if not found.

- ItemList wordsWithFrequency(int frequency).
  Purpose : To search the words which have the given frequency.
  Called by : Graphical User Interface.
  Calls : Nothing.
  frequency : The frequency of the words.
  Returns : List of the words.

- ItemList topFrequencyWords(int number)
  Purpose : To search the items those have words which are frequent in a given top range.
  Called by : Graphical User Interface.
  Calls : Nothing.
  number : The number of top frequencies of words.
  Returns : The list of items found.

- ItemList leastFrequencyWords(int number)
  Purpose : To search the words which are rare in terms of given frequency range.
  Called by : Graphical User Interface.
  Calls : Nothing.
  number : The number of least frequencies.
  Returns : The list of item found.

Graphical User Interface (GUI)

Functionality : To take input from user and display output to the user.

Interface Description:

- buttonResponser()
  Purpose : To invoke the operation for which the button has been pressed.
  Called by : Home module.
  Calls : Index Engine.

Chapter 3

Detail Design

3.1 Parse

This module takes a text file as input. The file is just opened for reading and it won't be modified. The file is parsed line by line and the interface function which adds the item into the dictionary is called. The functioning is completed by parsing the entire file.

This module represents the process of assisting the other module which builds dictionary, by providing items which are parsed from the text file. This module can be manipulated such that to handle different kinds of files.

3.1.1 Interface Data Structures

Item :

Item is a data structure which contains a word and its frequency. Item is formed from the text file which is parsed. Item is exported to the module Index Engine so that it can be placed in the dictionary. The word in an item is unchanged through out the life of the item. But frequency is changed when insertion or deletion of the corresponding word takes place.

word :

Datatype : String

Purpose : It is the heart of the dictionary, because the dictionary is created to have words.

frequency :

Datatype : Integer

Purpose : It says how frequent the associated word is. This may be the number of times the word repeats in a document, etc.

3.1.2 Internal Data Structures

None

3.1.3 Interface Functions

int parse(FileInputStream filefd);

Description : This function parses the give text file which is in a designed format.

Input Parameters : filefd (descriptor of the file opened for reading).

Output Parameters : None.

Return Values : 0 on success, –1 on failure.

Pseudo code :

Algorithm 1: Parse(filefd)

```
fp = openFile(FilePath, "r")
if fp = NIL then
    return FileNotFoundError
while line = readLine(fp) != EOF do
    word, frequency = line
    item.word = word
    item.frequency = frequency
    addItem(item)
end
```

3.1.4 Internal Functions

None.

3.2 Index Engine

The functionality of this module is to build up a dictionary by getting Items from the Parse module. It searches the word given by the GUI module whether it's exist in the Dictionary or not and also replies for the queries posed by the GUI module.

It represents an Index Engine. A data structure called skip list is being used to build up the dictionary which in turn used by the Index Engine. However, changing the data structure from skip list to another doesn't effect the system.

3.2.1 Interface Data Structure

Item : It has two components word and frequency. This module gets an item from the Parse module and the GUI module, and it is exported to the GUI module. The frequency of an item can be modified when add or delete operations performed, but the word is unchanged.

word

Datatype : String

Purpose : It is the key thing in the dictionary, because the word index engine is to deal with words.

frequency

Datatype : Integer

Purpose : It represents how frequent the corresponding word is.

3.2.2 Internal Data Structure

Skip list : This data structure is useful to build the dictionary. The dictionary contains words in lexicographic order. Skip list contains a list of Nodes. Node is a data structure which contains two sub structures as Item & Levels.

The item in turn has two components word and frequency. Levels are references that a node has to refer to the next nodes. It gives the search results with a time complexity of "O(log n)" which shows the best performance of the Index Engine.

Node : It contains two sub structures as Item & Levels. The item in turn has two components word and frequency. Levels are the references that a node has to refer to the next nodes.

Item : It has two components word and frequency.

Purpose : To pack word and its frequency into a single entity.

Levels : The number of other nodes that this node can refer to.

Datatype : Reference to a Node.

Purpose : It represents how many levels that a node should contain.

3.2.3 Interface Functions

Item addItem(Item item);

Description : It adds items into the dictionary.

Input Parameters : item.

Output Parameters : None.

Return Values : Item on success, null on failure.

Pseudo code :

Algorithm 2: addItem(item)

```
temp = head
fanin = head
fanout = null
level = N
foreach i=level decreases to 1 do
    if temp.item.word = item.word then
        temp.item.frequency+=item.frequency
        return temp.item
    while temp.levels[i] != null and temp.levels[i].item.word <
    item.word do
        temp = temp.levels[i]
        fanin.levels[i] = temp
        flag = true
    end
    if flag=false then
        fanin.levels[i] = temp
    fanout.levels[i] = temp.levels[i]
end
Iitem.level = randomLevel(1 to N)
Iitem.levels[Iitem.level]
Iitem.item = item
foreach i=Iitem.level decreases to 1 do
    Iitem.levels[i] = fanout.levels[i]
    fanin.levels[i].levels[i] = Iitem
end
return item
```

Item deleteItem(Item item);

Description : It deletes an item from the dictionary.

Input Parameters : item.

Output Parameters : None.

Return Values : Item on success, null on failure.

Pseudo code :

Algorithm 3: deleteItem(item)

```
temp = head
fanin = head
fanout = null
level = N
foreach i=level decreases to 1 do
    if temp.item.word = item.word then
        temp.item.frequency-=item.frequency
        return temp.item
    while temp.levels[i] != null and temp.levels[i].item.word <
    item.word do
        temp = temp.levels[i]
        fanin.levels[i] = temp
        flag = true
    end
    if flag=false then
        fanin.levels[i] = temp
    fanout.levels[i] = temp.levels[i]
end
if temp.item.word = item.word then
    foreach i = temp.leveldecreases to 1 do
        fanin.levels[i].levels[i] = fanout.levels[i]
        item.frequency = 0
        return item
    end
else
    return null
end
```

Item search(String word):

Description : To search a word in the dictionary.

Input Parameters : word.

Output Parameters : None.

Return Values : item on success, null on failure.

Pseudo code :

Algorithm 4: Search(word)

```
temp = head
foreach i=level decreases to 0 do
    while temp.levels[i] != null and temp.levels[i].item.word = word
    do
        temp = temp.levels[i]
    end
end
if temp != null and item.word = word then
    return temp
else
    return null
end
```

ItemList wordsWithFrequency(int frequency);

Description : To search for the words which have the given frequency in the dictionary.

Input Parameters : frequency.

Output Parameters : None.

Return Values : List of words on success, none on failure

Pseudo code :

Algorithm 5: wordsWithFrequency(frequency)

```
temp = head
while temp.levels[1] != null do
    if temp.item.frequency = frequency then
        ItemList = item
    temp = temp.levels[1]
end
return ItemList
```

ItemList topFrequencyWords(int number);

Description : To search for the words which have top frequencies in the dictionary.

Input Parameters : Number.

Output Parameters : None.

Return Values : List of words on success, none on failure

Pseudo code :

Algorithm 6: topFrequencyWords(frequency)

```
temp = head
frequencyarray[1 to frequency]
while temp != null do
    minfreq = minimumFrequency(frequencyarray)
    if minimumFrequency[minfreq] ¡ temp.item.frequency then
        minimumFrequency[minfreq] = temp.item.frequency
    temp = temp.levels[1]
    Sort minimumFrequency in decreasing order
    foreach i = 1 to size of minimumFrequency do
        temp = head
        while temp != null do
            if temp.item.frequency = minimumFrequency[i] then
                ItemList = item
            temp = temp.levels[1]
        end
    end
    return ItemList
end
int minimumFrequency(frequencyarray)
foreach i = 1 to size of frequencyarray do
    if frequencyarray[minfreq] ¿ frequencyarray[i] then
        minfreq = i
    return minfreq
end
```

ItemList leastFrequencyWords(int number);

Description : To search for the words which have least frequencies in the dictionary.

Input Parameters : Number

Output Parameters : None.

Return Values : array list of words

Pseudo code :

Algorithm 7: leastFrequencyWords(frequency)

```
temp = head
frequencyarray[1 to frequency]
while temp != null do
    maxfreq = maximumFrequency(frequencyarray)
    if maximumFrequency[minfreq] ¿ temp.item.frequency then
        maximumFrequency[minfreq] = temp.item.frequency
    temp = temp.levels[1]
    Sort maximumFrequency in increasing order
    foreach i = 1 to size of maximumFrequency do
        temp = head
        while temp != null do
            if temp.item.frequency = maximumFrequency[i] then
                ItemList = item
            temp = temp.levels[1]
        end
    end
    return ItemList
end
int maximumFrequency(frequencyarray)
foreach i = 1 to size of frequencyarray do
    if frequencyarray[maxfreq] ¡ frequencyarray[i] then
        maxfreq = i
    return maxfreq
end
```

3.2.4 Internal Functions

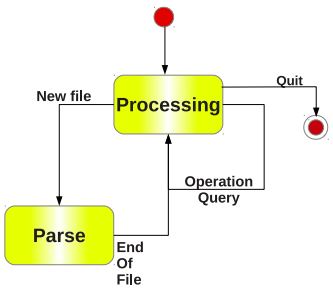
No Internal functions are exist in this function.

3.3 Graphical User Interface(GUI)

This module takes input from the user and displays output in the area specified. It offers users to search words in the dictionary, to pose queries related to frequency of the words, provides operations to modify the dictionary. It provides beautiful means to give input and to get output to the user.

This module represents the interface for the end user. This module can be manipulated such that the look of the entire GUI can be changed.

Home Window



Finite State Machine

3.3.1 Interface Data Structures

Item :

Item is a data structure which contains word, frequency. In this module the item is build up with the word (for addition or deletion) given by the user from the GUI and a pseudo frequency 1, saying that increase (for addition) or decrease (for deletion) the frequency of the word by 1. This happens when the word exists in the dictionary.

word :

Datatype : String

Purpose : To be added or deleted from the dictionary.

frequency :

Datatype : Integer

Purpose : To be incremented or decremented. Or to make a new item to be added or to make an old item to be deleted.

3.3.2 Internal Data Structures

None.

3.3.3 Interface Functions

None.

3.3.4 Internal Functions

void searchResponser(Event e);

Description : This function listens to the event generated when the "Search" button pressed.

Input Parameters : event.

Output Parameters : None.

Return Values : None.

Pseudo code :

Algorithm 8: searchResponser(Event e)

```
word = read(ae.line)
item = Search(word)
if item.word != NIL then
 | print item.word item.frequency
else
 | print word does not exist
end
```

void helpResponser(Event e);

Description : This function listens to the event generated when the "help" button pressed.

Input Parameters : event.

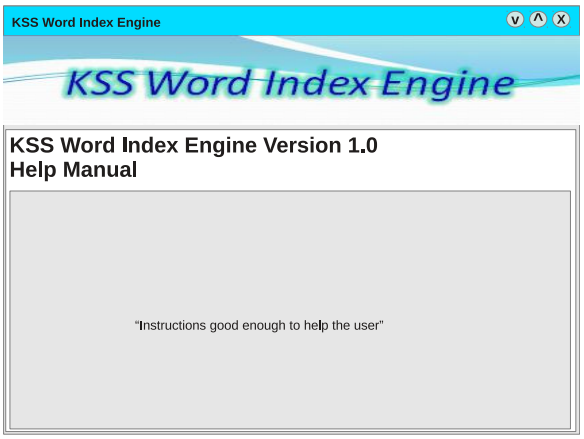
Output Parameters : None.

Return Values : None

Pseudo code :

Algorithm 9: helpResponser(Event e)

```
openWindow(HelpWindow)
print help message
```



Help Window

void clearResponser(Event e);

Description : This function listens to the event generated when the "clear button pressed.

Input Parameters : event.

Output Parameters : None.

Return Values : None.

Pseudo code :

Algorithm 10: clearResponser(Event e)

```
HomeWindow.DisplayArea = NIL
HomeWindow.TextField = NIL
```

void openFileWindowResponser(Event e);

Description : This function listens to the event generated when the "Open File" button on "Home Window" is pressed.

Input Parameters : event.

Output Parameters : None.

Return Values : None.

Pseudo code :

Algorithm 11: openAddWindowResponser(Event e)

```
openWindow(FileSelectWindow)
file = select file
parse(file)
```

void openAddWindowResponser(Event e);

Description : This function listens to the event generated when the "Add button on "Home Window" is pressed.

Input Parameters : event.

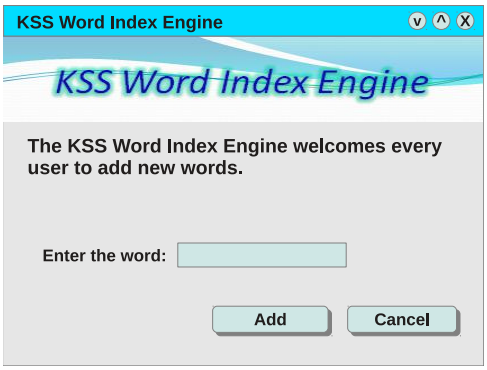
Output Parameters : None.

Return Values : None.

Pseudo code :

Algorithm 12: openAddWindowResponser(Event e)

```
openWindow(AddWindow)
```



Add Window

void addItemResponser(Event e);

Description : This function listens to the event generated when the "Add" button on "Add Window" pressed.

Input Parameters : event.

Output Parameters : None.

Return Values : None.

Pseudo code :

Algorithm 13: addItemResponser(Event e)

```
Item.word = textField.readline()
Item.frequency = 1
rval =addItem(Item)
closeWindow(current window)
print Item is added
```

void openDeleteWindowResponser(Event e);

Description : This function listens to the event generated when the "delete button on "Home Window" is pressed.

Input Parameters : event.

Output Parameters : None.

Return Values : None.

Pseudo code :

Algorithm 14: openDeleteWindowResponser(Event e)

```
openWindow(DeleteWindow)
```

Delete Window

void deleteItemResponser(Event e);

Description : This function listens to the event generated when the "Delete" button on "Delete Window" pressed.

Input Parameters : event.

Output Parameters : None.

Return Values : None.

Pseudo code :

Algorithm 15: deleteItemResponser(Event e)

```
Item.word = textField.readline()
Item.frequency = 1
deleteItem(Item)
closeWindow(current window)
print Item is deleted
```

void openQueryWindowResponser(Event e);

Description : This function listens to the event generated when the "Queries" button on "Home window" pressed.

Input Parameters : event.

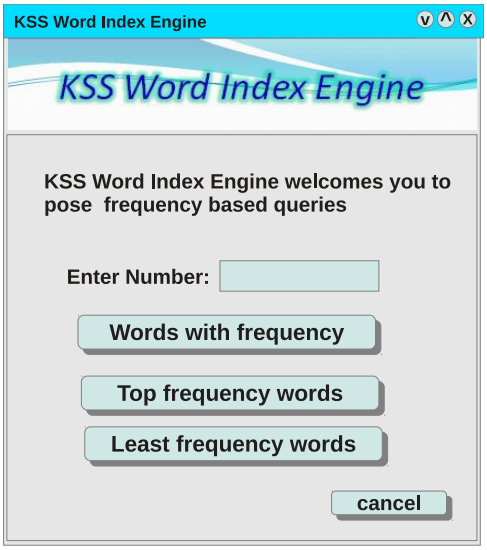
Output Parameters : None

Return Values : None.

Pseudo code :

Algorithm 16: openQueryProcessingWindowResponser(Event e)

```
openWindow(QueryProcessingWindow)
```



Query Window

void queryOneResponser(Event e);

Description : This function listens to the event generated when the "Words with frequency" button pressed.

Input Parameters : event.

Output Parameters : None.

Return Values : None.

Pseudo code :

Algorithm 17: queryoneResponser(Event e)

```
Item.frequency = textField.readline()
list = wordsWithFrequency(frequency)
if list != NIL then
 | print list
else
 | print no element exists with this frequency
end
```

void queryTwoResponser(Event e);

Description : This function listens to the event generated when the "Top frequency words" button pressed.

Input Parameters : event.

Output Parameters : None.

Return Values : None.

Pseudo code :

Algorithm 18: querytwoResponser(Event e)

```
number = textField.readline()
list = topFrequencyWords(number)
print list
```

void queryThreeResponser(Event e);

Description : This function listens to the event generated when the "Least frequency words" button pressed.

Input Parameters : event.

Output Parameters : No output parameters.

Return Values : None.

Pseudo code :

Algorithm 19: querythreeResponser(Event e)

```
number = textField.readline()
list = leastFrequencyWords(number)
print list
```

void cancelResponser(Event e);

Description : This function listens to the event generated when the "Cancel" button is pressed.

Input Parameters : event

Output Parameters : None.

Return Values : None.

Pseudo code :

Algorithm 20: cancelResponser(Event e)

```
closeWindow(current window)
```

void quitResponser(Event e);

Description : This function listens to the event generated when the "Quit" button is pressed.

Input Parameters : event

Output Parameters : None.

Return Values : None.

Pseudo code :

Algorithm 21: quitResponser(Event e)

```
closeWindow(HomeWindow)
```

Chapter 4

Testing

4.1 Unit Testing

4.1.1 Introduction

System Overview

The *KSS Word Index Engine* based on skip list data structure has the following modules

- **Parse:** Parses the input text file and makes each valid word in the text file as an item, and sends it to database, to be added.
- **Index Engine:** It is the data base of the *KSS Word Index Engine* built on skip list, provided with query to search words, other frequency based queries, operations to add and delete words.
- **Graphical User Interface:** Provides a beautiful user interface to use *KSS Word Index Engine*.

4.1.2 Test Approach

Unit testing is used to test the modules, where each module is tested individually, to be judged whether the module is giving desired results or not. This can be done by giving different input sets to each module, which contain valid and invalid inputs. Based on the results emerged, the module goes for further modifications such that it can become a module with out bugs. The white box testing approach is used to test each module.

4.1.3 Test Plan

Parse

Features to be tested

- File empty or non empty.
- Checking valid word or not.

Features not to be tested

- File name given or not (because FileChooser takes care of it).
- Return value of the *addItem* function.

Index Engine

Features to be tested

- Adding a word in to dictionary.
- Deleting a word in to dictionary.
- Searching a word in to dictionary.
- Search for the words with given frequency.
- Search for top frequency words.
- Search for least frequency words.

Features not to be tested

- Whether the parse module send perfect data or miscellaneous data.

Graphical User Interface

Features to be tested

- Checking the operation to be atomic.
- Checking the functionality of *Search* button.
- Checking the functionality of *Open File* button.
- Checking the functionality of *AddWord* button on *Home* window.

- Checking the functionality of *DeleteWord* button on *Home* window.
- Checking the functionality of *Queries* button.
- Checking the functionality of *Quit* button.
- Checking the functionality of *Help* button.
- Checking the functionality of *Clear* button.
- Checking the functionality of *Add* button on *Add* window.
- Checking the functionality of *Cancel* button on *Add* window.
- Checking the functionality of *Delete* button on *Delete* window.
- Checking the functionality of *Cancel* button on *Delete* window.
- Checking the functionality of *Words With Frequency* button.
- Checking the functionality of *Top Frequency Words* button.
- Checking the functionality of *Least Frequency Words* button.
- Checking the functionality of *Cancel* button on *Queries* window.
- Checking the functionality of tabs on *Help* window.
- Checking the functionality of hyper links on *Help* window.
- Checking input validation on *Home* window.
- Checking input validation on *Add* window.
- Checking input validation on *Delete* window.
- Checking input validation on *Queries* window.
- Checking dialog display for each invalid input.

Features not to be tested

- Closing window using *Close* button on title bar.

4.1.4 Test Cases

Parse

KWEP-1 :
Purpose : To check for the non emptyness of given file.
Input : File name.
Expected Output: A line of text.
Pass Criterion : If the first line of the file is not null.
Test Procedure : Checking the first line read is null or not.

KWEP-2 :
Purpose : To check for the emptyness of given file.
Input : File name.
Expected Output: null.
Pass Criterion : If the first line of the file is null.
Test Procedure : Checking the first line read is null or not.

KWEP-3 :
Purpose : To check whether the word is valid or not.
Input : a word.
Expected Output: An item.
Pass Criterion : If the word contains only characters or characters with ‘”’ , ‘’ ’, ‘!’, ‘?’, ‘,’, ‘.’.
Test Procedure : Check the each character of the word.

Index Engine

KWEIE-1 :
Purpose : To test the functionality of *addItem* function to add new item.
Input : An item which contains a perfect word and its frequency.
Output : Returns an item which has been added.
Pass criteria : If the item which is added is returned.
Test Procedure : A perfect new(not existing in dictionary) item is passed to the *addItem* function.

KWEIE-2 :
Purpose : To test the functionality of *addItem* function to increase

the frequency of item.
Input : An item which contain a perfect word and its frequency.
Output : Returns an item which has been added.
Pass criteria : If the item which is added is returned.
Test Procedure : An item (already existing in dictionary) is passed to the *addItem* function.

KWEIE-3 :
Purpose : To test the functionality of *addItem* function, when gets failed because of max limit of frequency.
Input : An item which contain a perfect word and its frequency.
Output : null.
Pass criteria : If the function returns null
Test Procedure : An item is passed to the *addItem* function. The word in that item should exist in the dictionary and will have max limit of frequecy.

KWEIE-4 :
Purpose : To test the functionality of *deleteItem* function to handle deletion of non existing item.
Input : An item which contain a perfect word and its frequency.
Output : null.
Pass criteria : When it returns 'null'.
Test Procedure : A perfect new item (not existing in dictionary) is passed to the *deleteItem* function.

KWEIE-5 :
Purpose : To test the functionality of *deleteItem* function to decrease frequency of item.
Input : An item which contain a perfect word and its frequency.
Output : Returns an item.
Pass criteria : If the item which got deleted is returned with updated frequency.
Test Procedure : An item (already existing in dictionary with frequency more than 1) is passed to the *deleteItem* function.

KWEIE-6 :

Purpose	: To test the functionality of *deleteItem* function to completely delete the item.
Input	: An item which contain a perfect word and its frequency.
Output	: Returns an item.
Pass criteria	: If the item which got deleted is returned with updated frequency.
Test Procedure	: An item (already existing in dictionary with frequency 1) is passed to the *deleteItem* function.

KWEIE-7 :

Purpose	: To test the functionality of *search* function.
Input	: A word.
Output	: Returns an item.
Pass criteria	: If an item with the given word and its frequency is returned.
Test Procedure	: A word (exist in dictionary) is passed to the *search* function.

KWEIE-8 :

Purpose	: To test the functionality of *search* function.
Input	: A word.
Output	: Returns an item.
Pass criteria	: If null is returned.
Test Procedure	: A word (not exist in dictionary) is passed to the *search*function.

KWEIE-9 :

Purpose	: To test the functionality of *wordsWithFrequency* function.
Input	: Frequncy of word(s).
Output	: Returns an array list of items.
Pass criteria	: If a list of items with the given frequency is returned.
Test Procedure	: A frequency is passed to the *wordsWithFrequency* function.

KWEIE-10 :
Purpose : To test the functionality of *wordsWithFrequency* function.
Input : Frequncy of word(s).
Output : null.
Pass criteria : If null is returned.
Test Procedure : A frequency (no item exists with this frequency in the dictionary) is passed to the *wordsWithFrequency* function.

KWEIE-11 :
Purpose : To test the functionality of *topFrequencyWords* function.
Input : The number of top frequencys.
Output : The list of items with words having top frequencies is returned.
Pass criteria : If expected list of items returned.
Test Procedure : A number is passed to the *topFrequencyWords* function.

KWEIE-12 :
Purpose : To test the functionality of *topFrequencyWords* function.
Input : The number of top frequencys.
Output : null
Pass criteria : If null is returned.
Test Procedure : A number is passed to the *topFrequencyWords* function (when the dictionary is empty).

KWEIE-13 :
Purpose : To test the functionality of *leastFrequencyWords* function.
Input : The number of least frequencys.
Output : The list of items with words having least frequencies is returned.
Pass criteria : If expected list of items returned.
Test Procedure : A number is passed to the *leastFrequencyWords* function.

KWEIE-14 :

Purpose	: To test the functionality of $leastFrequencyWords$ function.
Input	: The number of least frequencys.
Output	: null
Pass criteria	: If null is returned.
Test Procedure	: A number is passed to the $leastFrequencyWords$ function (when the dictionary is empty).

Graphical User Interface

KWEG-1 :

Purpose	: To check the operation is atomic or not.
Input	: Mouse click.
Expected Output	: The *Home* window does not respond to other if some operation is already being done.
Fail Criterion	: If the *Home* window responds.
Test Procedure	: Click some button which opens a new window, try to click any button on the home window.

KWEG-2 :

Purpose	: To check the functionality of *Search* button.
Input	: Mouse click.
Expected Output	: word, frequency or No results.
Fail Criterion	: No output display.
Test Procedure	: Enter some word in the text area and press *Search* button.

KWEG-3 :

Purpose	: To check the functionality of *Open File* button.
Input	: Mouse click.
Expected Output	: *File chooser* window opens.
Fail Criterion	: *File chooser* window not opens.
Test Procedure	: Click *Open File* button.

KWEG-4 :

Purpose	: To check the functionality of $AddWord$ button on *Home* window.
Input	: Mouse click.
Expected Output	: Opening of *Add* window.

Fail Criterion : If the window isn't opened.
Test Procedure : Click on *AddWord* button on *home* window.

KWEG-5 :
Purpose : To check the functionality of *DeleteWord* button on *Home* window.
Input : Mouse click.
Expected Output : Opening of *Delete* window.
Fail Criterion : If the window isn't opened.
Test Procedure : Click on *DeleteWord* button on *Home*window.

KWEG-6 :
Purpose : To check the functionality of *Queries* button.
Input : Mouse click.
Expected Output : Opening of *Queries* window.
Fail Criterion : If the window isn't opened.
Test Procedure : Click on *Quries* button.

KWEG-7 :
Purpose : To check the functionality of *Quit* button.
Input : Mouse click.
Expected Output : Opening of *Quit* window.
Fail Criterion : If *Quit* window isn't opened.
Test Procedure : Click on *Quit* button.

KWEG-8 :
Purpose : To check the functionality of *Help* button.
Input : Mouse click.
Expected Output : Opening of *Help* window.
Fail Criterion : If *Help* window isn't opened.
Test Procedure : Click on *Help* button.

KWEG-9 :
Purpose : To check the functionality of *Add* button on *Add* window.
Input : Mouse click.
Expected Output : Closing the *Add* window, displaying "word is added".
Fail Criterion : If nothing is displayed or *Add* window isn't closed.
Test Procedure : Enter some word in text area, click *Add* button on *Add* window.

KWEG-10 :

Purpose	: To check the functionality of *Cancel* button on *Add* window.
Input	: Mouse click.
Expected Output	: Closing of *Add* window.
Fail Criterion	: If window isn't closed.
Test Procedure	: Click *Cancel* button on *Add* window.

KWEG-11 :

Purpose	: To check the functionality of *Delete* button on *Delete* window.
Input	: Mouse click.
Expected Output	: Closing *Delete* window, displaying "word is deleted".
Fail Criterion	: If nothing is displayed or *Delete* window isn't closed.
Test Procedure	: Enter some word in text area, click *Delete* button on *Delete* window.

KWEG-12 :

Purpose	: To check the functionality of *Cancel* button on *Delete* window.
Input	: Mouse click.
Expected Output	: Closing of *Delete* window.
Fail Criterion	: If window isn't closed.
Test Procedure	: Click *Cancel* button on *Delete* window.

KWEG-13 :

Purpose	: To check the functionality of *Words With Frequency* button.
Input	: Mouse click.
Expected Output	: Closing *Queries* window, displaying "WordsWithFrequency".
Fail Criterion	: If nothing is displayed or *Queries* window isn't closed.
Test Procedure	: Enter some number on the text area, press *Words With Frequency* button.

KWEG-14 :

Purpose	: To check the functionality of *Top Frequency Words* button.
Input	: Mouse click.
Expected Output	: Closing *Queries* window, displaying

"TopFrequencyWords".
Fail Criterion : If nothing is displayed or *Queries* window isn't closed.
Test Procedure : Enter some number on the text area, press *Top Frequency Words* button.

KWEG-15 :
Purpose : To check the functionality of *Least Frequency Words* button.
Input : Mouse click.
Expected Output : Closing *Queries* window, displaying "LeastFrequencyWords" .
Fail Criterion : If nothing is displayed or *Queries* window isn't closed.
Test Procedure : Enter some number on the text area, press *Least Frequency Words* button.

KWEG-16 :
Purpose : To check the functionality of *Cancel* button on *Queries* window.
Input : Mouse click.
Expected Output : Closing of *Queries* window.
Fail Criterion : If window isn't closed.
Test Procedure : Click *Cancel* button on *Queries* window.

KWEG-17 :
Purpose : To check the functionality of tabs on *Help* window.
Input : Mouse click.
Expected Output : Display html page on that tab.
Fail Criterion : If the html page isn't displayed.
Test Procedure : Click on the tabs on *Help* window.

KWEG-18 :
Purpose : To check the functionality of hyper links on *Help* window.
Input : Mouse click.
Expected Output : Loading of new html page.
Fail Criterion : If new page isn't loaded.
Test Procedure : Click on hyper link.

KWEG-19 :
Purpose : To validate the input on *Home* window.

Input : Word.
Expected Output : Displaying the word.
Fail Criterion : If the word is displayed.
Test Procedure : Enter the valid word, press *Search* button.

KWEG-20 :
Purpose : To validate the input on *Home* window.
Input : Invalid Word.
Expected Output : Display error message on dialog box.
Fail Criterion : If the word isn't displayed.
Test Procedure : Enter invalid word, press *Search* button.

KWEG-21 :
Purpose : To validate the input on *Add* window.
Input : Word.
Expected Output : Displaying "word is added" on *Home* window.
Fail Criterion : If nothing is displayed.
Test Procedure : Enter valid word, press *Add* button.

KWEG-22 :
Purpose : To validate the input on *Add* window.
Input : Invalid Word.
Expected Output : Display error message on dialog box.
Fail Criterion : Displaying "word is added" on *Home* window.
Test Procedure : Enter invalid word, press *Add* button.

KWEG-23 :
Purpose : To validate the input on *Delete* window.
Input : Word.
Expected Output : Displaying "word is deleted" on *Home* window.
Fail Criterion : If nothing is displayed.
Test Procedure : Enter valid word, press *Delete* button.

KWEG-24 :
Purpose : To validate the input on *Delete* window.
Input : Invalid Word.
Expected Output : Display error message on dialog box.
Fail Criterion : Displaying "word is deleted" on *Home* window.
Test Procedure : Enter invalid word, press *Delete* button.

KWEG-25 :
Purpose : To validate the input on *Queris* window.
Input : Number.
Expected Output : Displaying appropriate button output.
Fail Criterion : If nothing is displayed.
Test Procedure : Enter valid number, press any button except *Cancel*.

KWEG-26 :
Purpose : To validate the input on *Queris* window.
Input : Number.
Expected Output : Display error message on dialog box.
Fail Criterion : Displaying appropriate button output.
Test Procedure : Enter invalid number, press any button except *Cancel*.

KWEG-27 :
Purpose : To check the *Dialog Box* opens or not.
Input : Invalid word or number.
Expected Output : Opening of *Dialog Box*.
Fail Criterion : If the *Dialog Box* isn't opened.
Test Procedure : Enter invalid data, press any functional button.

4.2 Intigration Testing

4.2.1 Introduction

System Overview

Word index engine is software like search engine based on skip-list. It offers two type of searching.

- Word based search.
- Frequency based search.

It offers users to add and delete words also with a beautiful Graphical Use Interface(GUI). The Skiplist is an advanced data structure which searches an element in the list with a time complexity of log n.

4.2.2 Test Approach

Integration testing is used to test the interfaces between the modules by combining all the modules together. Test each path which formed with recursive calls of interface functions. This can be done by different Input

sets(containing valid and invalid inputs) to each interface. Based on results the modules goes for further changes such that all interface functions become perfect. The block box testing is used to test each interface function.

4.2.3 Test Plan

Featured to be tested

- Checking of *Parse* function called from GUI
- Checking of *addItem* function from Parse
- Checking of *addItem* function from GUI
- Checking of *deleteItem* function from GUI
- Checking of *search* function from GUI
- Checking of *wordsWithFrequency* function from GUI
- Checking of *topFrequencyWords* function from GUI
- Checking of *leastFrequencyWords* function from GUI

Features to be not tested

- Internal functions in modules

4.2.4 Test Cases

KWEGP-1 :

Purpose	: To check the interface of *Parse* function.
Input	: Text file.
Output	: "Dictionary is updated" is displayed on the GUI.
Pass criteria	: If "Dictionary is updated" is displayed on the GUI.
Test Procedure	: Call function Parse with a File(which contain atleast one valid word).

KWEPIE-1 :

Purpose	: To check the interface of *addItem* function.
Input	: Item with frequency of the word 1.
Output	: Item with updated frequency.
Pass criteria	: If Item with updated frequency is returned.
Test Procedure	: Call function *addItem* with an Item.

KWEGP-2 :

Purpose	: To check the interface of Parse function.
Input	: Text file.
Output	: "Dictionary isn't updated" is displayed.
Pass criteria	: If "Dictionary isn't updated" is displayed.
Test Procedure	: Call Parse with a file (Which contains all Invalid lines or which is empty.)

KWEGIE-1 :

Purpose	: To check the interface of *addItem* function on successful addition.
Input	: A word which is formed as an item with frequency of the word being 1.
Output	: The message "The word is added successfully" , the word and its frequency are displayed.
Pass criteria	: If The message "The word is added successfully" , the word and its frequency are displayed.
Test Procedure	: Call function *addItem* with an Item.

textbfKWEGIE-2 :

Purpose	: To check the interface of *addItem* function on failure of addition.
Input	: A word which is formed as an item with frequency of the word being 1
Output	: The message "The word isn't added because of frequency limit." is displayed.
Pass criteria	: If The message "The word isn't added because of frequency limit." is displayed.
Test Procedure	: Call function *addItem* with an Item whose frequency in the dictionary is the maximum frequency.

KWEGIE-3 :

Purpose	: To check the interface of *deleteItem* function on successful deletion.
Input	: A word which is formed as an item with frequency of the word being 1.
Output	: The message "The word is deleted successfully" , the word and its frequency are displayed.
Pass criteria	: If The message "The word is deleted successfully" , the word and its frequency are displayed.
Test Procedure	: Call function *deleteItem* with an Item(which exists in the dictionary).

KWEGIE-4 :

Purpose	: To check the interface of *deleteItem* function on failure of deletion.
Input	: A word which is formed as an item with frequency of the word being 1
Output	: The message "The word doesn't exist." is displayed.
Pass criteria	: If The message "The word doesn't exist." is displayed.
Test Procedure	: Call function *deleteItem* with an Item(which doesn't exist in dictionary).

KWEGIE-5 :

Purpose	: To check the interface of *Search* function on successful search.
Input	: word
Output	: The word and its frequency displayed.
Pass criteria	: If the word and its frequency displayed.
Test Procedure	: Call the *Search* function with a word(which exists in dictionary).

KWEGIE-6 :

Purpose	: To check the interface of *Search* function on failure.
Input	: word
Output	: The message "No results" displayed.

Pass criteria : If The message "No results" displayed.
Test Procedure : Call the *Search* function with a word(which doesn't exists in dictionary).

KWEGIE-7 :
Purpose : To check the interface of $wordsWithFrequency$ function on successful search.
Input : Frequency of word(s).
Output : Words with that frequency are displayed.
Pass criteria : If Words with that frequency are displayed.
Test Procedure : Call the $wordsWithFrequency$ function with frequency (at least one word has to contain this frequency in the dictionary).

KWEGIE-8 :
Purpose : To check the interface of $wordsWithFrequency$ function on failure search.
Input : Frequency of word(s).
Output : The message "No words with the given frequency" is displayed.
Pass criteria : If The message "No words with the given frequency" is displayed.
Test Procedure : Call the $wordsWithFrequency$ function with a frequency (No word has this frequency in dictionary).

KWEGIE-9 :
Purpose : To check the interface of $topFrequencyWords$ function on successful search.
Input : Number of top frequencies.
Output : Top frequency words and their frequencies are displayed.
Pass criteria : If Top frequency words and their frequencies are displayed.
Test Procedure : Call the $topFrequencyWords$ function with a frequency (dictionary is not empty).

KWEGIE-10 :

Purpose	: To check the interface of $topFrequencyWords$ function on failure search.
Input	: Number of top frequencies.
Output	: The message "The dictionary is empty" is displayed.
Pass criteria	: If The message "The dictionary is empty" is displayed.
Test Procedure	: Call the $topFrequencyWords$ function with a frequency (dictionary is empty).

KWEGIE-11 :

Purpose	: To check the interface of $leastFrequencyWords$ function on successful search.
Input	: Number of least frequencies.
Output	: Least frequency words and their frequencies are displayed.
Pass criteria	: If Least frequency words and their frequencies are displayed.
Test Procedure	: Call the $leastFrequencyWords$ function with a frequency (dictionary is not empty).

KWEGIE-12 :

Purpose	: To check the interface of $leastFrequencyWords$ function on failure search.
Input	: Number of least frequencies.
Output	: The message "The dictionary is empty" is displayed.
Pass criteria	: If The message "The dictionary is empty" is displayed.
Test Procedure	: Call the $leastFrequencyWords$ function with a frequency (dictionary is empty).

Chapter 5

Future Enhancements

- Search can be enhanced such that the words that are nearer to the given search word are also be displayed as search results.
- Adding word from GUI is increasing frequency by only 1. This can be enhanced such that user can enter the frequency that has to be increased, through GUI.
- Deleting word from GUI is decreasing frequency by only 1. This can be enhanced such that user can enter the frequency that has to be decreased, through GUI.
- New query can be added such as listing words starting or ending with some string.
- New query can be added such as listing words having some string as sub-string.