ITU COMPUTER ENGINEERING DEPARTMENT

BLG 223E DATA STRUCTURES

HOMEWORK -2

Due Date: 1th of November, 2018 23:59 pm.

"If you really look closely, most overnight successes took a long time."

-- Steve Jobs

In this assignment, you are required to build up an extension of the linked list data structure.

A diagram of the structure is given below.

The general way of generating a sequence of text is to train a model to predict the next word/character given all previous words/characters. Such model is called a **Language Model**. Language models also allows you to calculate the possibility of predicting the next word/character given all previous words/characters.

An example calculation of the character probability is:

$$P(a|d) = \frac{C(d \cap a)}{C(d)}$$

The $\boldsymbol{P(a|d)}$ is the probability the character "a" comes right after the character "d".

The $\boldsymbol{C(d \cap a)}$ is the total count of concurrence "da" characters seen in given sentences sequence.

The $\boldsymbol{C(d)}$ is the total count of "d"'s seen in given sentences sequence.

For this homework, you will be given a set of sentences (**input.txt**) and ask to perform a basic character based language model by using linked list structure.

Workflow:

1. You will read the **input.txt** and split each unique character, cases of letters will be neglected (case insensitive) meaning both "A" and "a" will be counted as same. You will also see special characters (in input.txt file) such as dots (.), commas (,) or whitespaces (<SP>), these special characters will be added to your vocabulary list as well.
2. Then add each unique character to the **vocab_list** structure by **alphabetical order**. You should also add special characters to your **vocab_list**. The order of special characters are not important just they should be placed to the end of the alphabetical characters (after "z").
3. Then, trace your input again but this time add occurrences of each unique character by each unique next character. This structure will contain occurrences of each character pairs together and it will be stored by order of appearance. Meaning for the character "a", you will add each unique characters which comes next to "a" and keep their occurrence counts in your structure.

Explanation of the workflow by an example sentence:

“Dilara bugün okula gitti. ”

The given sentence is split by spaces, and for this sentence the **vocab_list** contains 15 nodes (one is for “.”(dot) and one is for the space character (<SP>) seen in between and at the end of sentences).

“D i l a r a <SP> b u g ü n <SP> o k u l a <SP> g i t t i . <SP>”

While traversing all sentences fill the **vocab_list** with alphabetical order. When you finish adding all the vocabulary, for the next time traverse the **input.txt** to add the occurrences. For example, for the character “a” in the input sentence “a-r” (occurrence is 1), “a-<SP>”(occurrence is 2) union co-occurrences have been found, and they are added to the occurrence structure by order of appearance. For the characters “i”, “l”, “r”, “.” and “<SP>” (white space) diagram is filled as an example, the rest characters are left blank on purpose.

The probability character sequence of the “la” will be calculated as:

$$P(a|l) = \frac{C(l \cap a)}{C(l)} = \frac{2}{2} = 1$$

Or

The probability character sequence of the “ar” will be calculated as:

$$P(r|a) = \frac{C(a \cap r)}{C(a)} = \frac{1}{3} = 0.33$$

Or

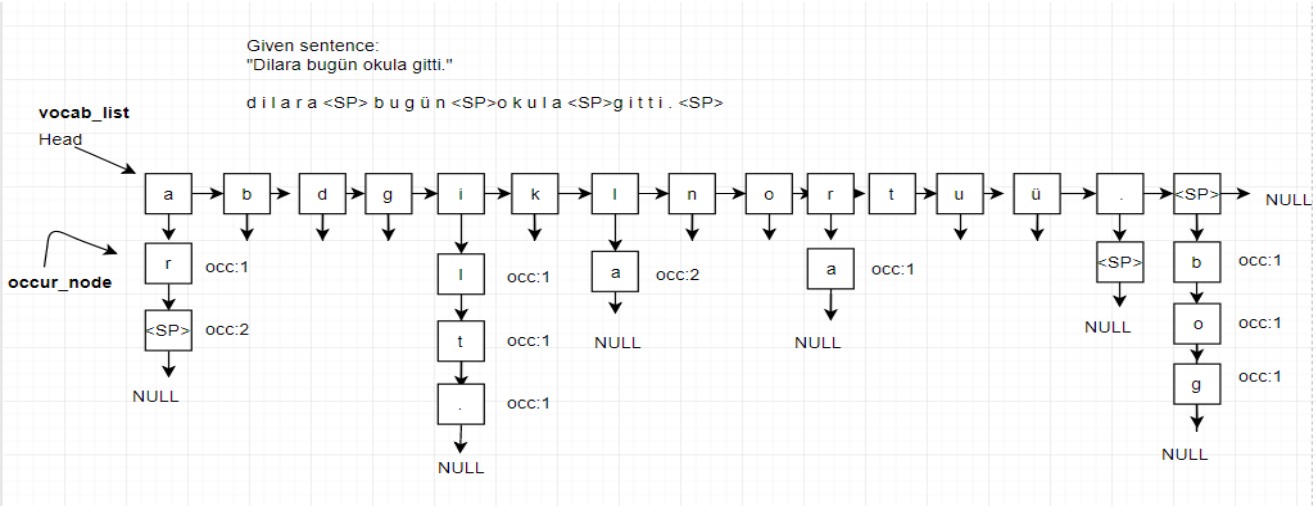
The probability character sequence of the “il” will be calculated as:

$$P(l|i) = \frac{C(i \cap l)}{C(i)} = \frac{1}{3} = 0.33$$

Hint1: calculation of C(x) (the total count of “x”’s seen in given sentences sequence) could be calculated as adding all the union co-occurrences of x with other characters. For example for C(a);

$$C(a) = C(a \cap r) + C(a \cap < SP >)$$

Hint2: $P(r|a) \neq P(a|r)$ probabilities of “ar” and “ra” are not same.



Structure:

```
struct vocab_node {

char character;
vocab_node *next;
occur_node *list;
}

struct occur_node {

char character;
occur_node *next;
int occurence;
};

struct vocab_list
{

vocab_node *head;
      void create();
      void print();
      void add_char(char );
      void add_occurence(char , char );
      int get_occurence(char );
      int get_union_occurence (char , char );
};

struct language_model {

vocab_list *vocabularylist;
      void readData (const char *);
      double calculateProbability (char, char);

};
```

Implementation:

Implement the following methods with appropriate arguments and return types for your structure:

a. **create():** Creates vocab_list.
b. **readData():** reads given input txt, calls **create**(), **add_char** and **add_occurences** functions and fills entire language model.
c. **print():** prints entire language model in the form:
character:
<co-occurencecharacter, occurencecount>
<co-occurencecharacter, occurencecount>

Example:
a:
<r, 1>
<<SP>,2>
b:
<u,1>
d:
<i,1>

d. **add_char():** gets one character from txt file add all the unique characters in alphabetical order to vocab_list. Meaning it first checks the character if it exists in vocab_list, if not it adds it to vocab_list. If it already exists then continues with the next character.
e. **add_occurence():** gets two union co-occurrence characters from txt file traces the first character in vocab_list, finds it and searches second character in its occur_node, if not found add this second character in its occur_node and initializes occurrence as 1, if found then only increases occurrence by one.
f. **get_occurence():** returns the total occurrence count of given character. Use hint to calculate this method.
g. **get_union_occurence():** returns the total union co-occurrence count of given two characters.
h. **calculateProbability():** returns calculated probability of given two characters.

Submission

1. Make sure you write your name and number in all of the files of your project, in the following format:

```
/* @Author

 * Student Name: <student_name>
 * Student ID : <student_id>
 * Date: <date> */
```

2. Use comments wherever necessary in your code to explain what you did.
3. **Your program should compile and run on Linux environment using g++ (version 4.8.5 or later). You can test your program on ITU's Linux Server using SSH protocol.**
   To compile the code, you can use the following command:
   g++ lm.cpp –o lm
   The program should produce the probability of co-occurrence two characters if the program is executed with three command line arguments which are <name-of-file> <first-char> <second-char>. However, the program should print the whole language model only if <name-of-file> given as single argument.

./lm input.txt a r (output will be probability of co-occurrence “ar” First character =a, second character =r)

./lm input.txt (output will be printed entire language model in given form)

4. After you make sure that everything is compiled smoothly, archive all files into a zip file. Submit this file through www.ninova.itu.edu.tr. Ninova enables you to change your submission before the submission deadline.

Do not miss submission deadline. **Do not** leave your submission until the last minute. The submission system tends to become less responsive due to high network traffic.

HOMEWORKS SENT VIA E-MAIL WILL NOT BE GRADED.

Academic dishonesty including but not limited to cheating, plagiarism and collaboration is unacceptable and subject to disciplinary actions. Your homeworks will be checked with a plagiarism checker system, any student found guilty will receive 0 as his/her grade for the homework and subject to disciplinary actions.

If you have any question about the homework, contact the teaching assistant **Dilara TORUNOĞLU SELAMET** via e-mail (**torunoglud@itu.edu.tr**) or in **4307**.