CS29003 ALGORITHMS LABORATORY
Assignment 5
Last Date of Submission: 16 – August – 2018

General Instruction

1. Please do not use any global variable unless you are explicitly instructed so.
2. Please use proper indentation in your code.
3. Please name your file as `<roll_no>_<assignment_no>`. For example, if your roll number is 14CS10001 and you are submitting assignment 3, then name your file as `14CS10001_3.c or 14CS10001_3.cpp` as applicable.
4. Please write your name and roll number at the beginning of your program.

One of the very common problems in web search is to find out if two words are synonyms of each other. For instance, while you are searching for 'car', a document containing the word 'automobile' may be relevant too. One particular way in which this similarity is captured is by representing words as vectors in R^d such that similar words are represented by similar vectors. The similarity between words can now be captured using distance measures such as euclidean distance.

Suppose you are given a vocabulary of n words. To find the most similar word to a given word w, you will need to compute its distance with each of the n words, and find the minimum. This corresponds to the nearest neighbor of w in R^d, and the complexity is $O(n)$. In this assignment, we will use a method to reduce this to expected $O(1)$ time. The idea is as follows:

Let's take the case with $d = 2$, and each word is a real vector in R^2 with both values between $(0, 1)$ (in unit square). So each word can be represented as a tuple $(word, x, y)$ consisting of the string word and the 2-d vector (x, y). To achieve expected $O(1)$ time for search, we use a 2-d hashtable of size $m \times m$. You should choose $m = \lceil \sqrt{(n)} \rceil$ to ensure that $m^2 \geqslant n$. Each word tuple can now be mapped to the corresponding cell where it actually lies using a simple hash function: $< word, x, y > \rightarrow [\lfloor m{*}x \rfloor, \lfloor m{*}y \rfloor]$. For example, if $m = 2$, using this map, a word tuple $(word1, 0.125, 0.625)$ falls in the cell $[0, 1]$ while a word tuple $(word2, 0.782, 0.698)$ falls in cell $[1, 1]$. At the time of search, given a word tuple, you will locate the square in the hashtable that this word would be part of, retrieve the words stored at that location and find the distance from each of these to get the nearest neighbor. Note, however, that if the queried word is also present in the hashtable, you should not return the same word but the nearest neighbor instead. The search for the nearest neighbor is not entirely trivial though, as we will see below. However, inserting elements in the hashtable is quite easy, and we will be using chaining to avoid collisions.

Part 1. Read the values from a file and insert in the hashtable

In this part, you will read n, the number of words to be hashed as well as the corresponding word tuples from an input file 'input.txt' as provided to you. Please note that you will need to use file handling for this. After reading n, you should use a dynamic memory allocation for the hashtable of size $m \times m$, where $m = \lceil \sqrt{(n)} \rceil$. You can use functions $sqrt()$ and $ceil()$ from the library $< math.h >$.

Next, you want to insert these n word tuples inside the hashtable.

Hashtable: To resolve collisions, you will use the concept of chaining. For this 2-d hashtable, you will use a 2-d array of linked lists to store the actual word tuples. Hash function will map a word tuple to the cell of the hashtable, where this word lies, i.e., $(word, x, y) \rightarrow [\lfloor m * x \rfloor, \lfloor m * y \rfloor]$

You can use the following structure to represent the hashtable.

```
typedef struct _wordR {
  char word[100];
  double x, y;
} wordR;

typedef struct _node{
  wordR w;
  struct _node *next;
} node, *nodePointer;

typedef nodePointer **hashTable;
```

Then, write the following function to insert the n words inside the hashtable one by one.

```
void insertH(hashTable H, int m, wordR w);
```

Now, you should print your hashtable in a file 'output.txt' as per the format given to you. For each cell of the hashtable, you should print the words in the cell. Write the following function. Passing the pointer to the output file helps you print directly in the file.

```
void printH(hashTable H, int m, FILE *fp);
```

Part 2. Search for the nearest neighbor of a given word

In this part, you will read word tuples from the user and find out the nearest neighbor from the hashtable. Note that only the 2-d vector (x, y) is of importance for this task. You can use the following steps:

Step 1: locate the cell in the hashtable: Use the hash function to locate the cell (i, j) to which this tuple belongs.

Step 2: Initial search through expanding squares centered at (i,j): The nearest neighbor may not lie in cell (i, j). In that case, you will have to locate it in the nearby squares. Thus for $k = 0, 1, 2, \ldots$, you can search for a closest word in squares (s, t) with $|s - i| \leqslant k$ and $|t - j| \leqslant k$. You will stop only when you find a word, different from the input word. Also compute the euclidean distance *dist* of this word from the input word.

Step 3: Find the nearest neighbor through circle centered at (i, j): The closest word found above need not be the closest to (x, y). The trick is to use the minimum distance $\mathtt{dist}$ obtained by the above method and consider all squares that lie in the circle centered at (x, y) with $\mathtt{dist}$ as the radius. Write the function $findNN$ that prints the correct result as per the sample output.

```
void findNN (hashTable H, int m, wordR w);
```

Note that this function will not return anything but print as per the sample output.

Main Function

main() function does the following:

1. Reads n from 'input.txt'. Initializes hashtable of size $m \times m$ with $m = \lceil\sqrt(n)\rceil$ as described in Part 1 above.
2. Reads n word tuples from 'input.txt' and calls $insertH()$ to insert these elements into the hashtable one by one.
3. Prints the hashtable using function $printH()$ in the file 'output.txt'.
4. Reads the number of words to be searched from the user.
5. For each of the words to be searched, it reads the tuple from the user and prints the nearest neighbor using $findNN()$.

Sample Output

```
Number of words to be inserted in the hashtable are 1000
Size of the hashtable is: 32 x 32
Finished inserting the elements in the hashtable
Finished printing the elements in the hashtable
Enter the number of words to search: 5

Enter the word tuple 1:beautfully 0.221297335693 0.678641178277
The correct result is  (frattini,0.22191,0.67090)

Enter the word tuple 2:cove 0.188206745435 0.58869043627
The correct result is  (declaimer,0.18743,0.59558)

Enter the word tuple 3:preserved 0.113018755642 0.182384388501
The correct result is  (beelong,0.12360,0.18190)

Enter the word tuple 4:scared 0.197037033514 0.326766176658
The correct result is  (georgrtte,0.22035,0.32867)

Enter the word tuple 5:vengeance 0.110597931214 0.658077052937
The correct result is  (director,0.12071,0.65253)
```