CS2302 - Data Structures

Spring 2018

Lab # 5

Deadline: Friday, March 30, 11:59 p.m.

1 Heaps (30%)

Using the *draw_tree* method for binary search trees as guide, write a method to display a heap given a reference to its root. Display a sequence of images showing the execution of heapsort.

2 Hash Tables (70%)

Natural Language Processing (NLP) is the sub-field of artificial intelligence that deals with designing algorithms, programs, and systems that can understand human languages in written and spoken forms. Word embeddings are a recent advance in NLP that consists of representing words by vectors in such a way that if two words are used in similar contexts, their embeddings are also similar. See https://nlp.stanford.edu/projects/glove/ for an overview of this interesting research.

In order to work in real-time, NLP applications such as Siri and Alexa use hash tables to efficiently retrieve the embeddings given their corresponding words. In this lab you will implement a simple version of this.

The web page mentioned above contains links to files that contain word embeddings of various lengths for various vocabulary sizes. Use the file *glove.6B.50d.txt* (included in the file *glove.6B.zip*), which contains word embeddings of length 50 for a very large number of words. Each line in the file starts with the word being described, followed by 50 floating point numbers that represent the word's vector description (the embedding). The words are ordered by frequency of usage, so "the" is the first word. Some "words" do not start with an alphabetic character (for example "," and "."); feel free to ignore them.

Your task for this lab is to write a program that does the following:

1. Read the file "glove.6B.50d.txt" and store each word and its embedding in a hash table (see appendix). Choose a prime number for your initial table size and increase the size to twice the current size plus one every time the load factor reaches 2. Caution: do NOT recompute the load factor every time an item is entered to the table!
2. Read another file containing pairs of words (two words per line) and for every pair of words find and display the "similarity" of the words. To find the similarity of words w_0 and w_1, with embeddings e_0 and e_1, we use the cosine distance, which ranges from -1 to 1, given by:

$$sim(w_0, w_1) = \frac{e_0 \cdot e_1}{|e_0||e_1|}$$

   where $e_0 \cdot e_1$ is the *dot product* of e_0 and e_1 and $|e_0|$ and $|e_1|$ are the magnitudes of e_0 and e_1.
3. Analyze the distribution of words in your hash table to verify that your hash function distributes the words as evenly as possible among the lists. Compute the following items from your table:
   (a) Load factor
   (b) The percentage of the lists in the table that are empty (lower is better)
   (c) The standard deviation of the lengths of the lists (lower is better)

Since the key used for hashing is a string, you need to convert it to an integer value in a way that would ultimately result in as few collisions as possible. A simple way is to add the int values of all the characters

in the string, and then apply the mod operation. A better way is to consider strings as numbers in a base 27 alphabet; see appendix for an example.

As usual, write a report describing your work.

Appendix

Sample run:

```
Word similarities:
Similarity [barley,shrimp] = 0.5353
Similarity [barley,oat] = 0.6696
Similarity [federer,baseball] = 0.2870
Similarity [federer,tennis] = 0.7168
Similarity [harvard,stanford] = 0.8466
Similarity [harvard,utep] = 0.0684
Similarity [harvard,ant] = -0.0267
Similarity [raven,crow] = 0.6150
Similarity [raven,whale] = 0.3291
Similarity [spain,france] = 0.7909
Similarity [spain,mexico] = 0.7514
Similarity [mexico,france] = 0.5478
Similarity [mexico,guatemala] = 0.8114
Similarity [computer,platypus] = -0.1277

Table stats:
Load factor: 1.45
Percentage of empty lists: 5.71
Standard deviation of the lengths of the lists: 1.56
```

Hash table class definition:

```
public class sNode{
   public String word;
   public float[] embedding;
   public sNode next;

   public sNode(String S, float[] E, sNode N){
      word = S;
      embedding = new float[50];
      for (int i=0;i<50;i++)
         embedding[i] = E[i];
      next = N;
   }
}

public class hashTableStrings{
   private sNode [] H;

   public hashTableStrings(int n){ // Initialize all lists to null
      H = new sNode[n];
```

```
      for(int i=0;i<n;i++)
         H[i] = null;
   }

   private int h(String S){
      int h = 0;
      for(int i=0;i<S.length();i++)
         h = (h*27+S.charAt(i))%H.length;
      return h;
   }
}
```