LABORATORY 11 : Hashing

OBJECTIVES

- to understand how hashing works
- to implement hash table with open addressing and probing
- to see effect of hash function on data distribution

LABORATORY 11: In-lab, Post-lab

Study http://interactivepython.org/courselib/static/pythonds/SortSearch/Hashing.html and Goodrich's chapter 10.

❖ Spell checker

A spell checker can be implemented using hash table. Table contains all the valid words. If a word is found in the table, it is assumed to be spelled correctly. Otherwise, it is assumed to be spelled incorrectly.

You are ask to build a spell checker capable of identifying correctly spelled words, (optional) suggesting words from a hash table of commonly misspelled words, and making as many other enhancements as you'd like.

- Valid words are stored in a dictionary file.
- Spell checker validate spelling against those valid words.
- Spell checker takes input (a word) from keyboard and response whether the word is correctly spelled.

  Sample dialog is shown below.

```
Enter your word : bird
"bird" is correctly spelled
Enter your word : wird
"wird" is not in the dictionary
```

- (optional) provide suggestion list for a misspelled word

```
Enter your word : wird
"wird" is not in the dictionary.
Possible corrections are : bird gird ward word
wild wind wire wiry
```

❖ Dictionary files

Dictionary files are text files containing valid words in lower case, separated by whitespace. There are two file provided :-

- small.txt (341 words)
- full.txt (34,829 words)

Below is how dictionary file may look like

```
a about above absolutely acceptable add adjacent after
algorithm all along also an analyses and any anyone are
argument as assignment assume at available be been below
bird body but by c can cannot capitalization ...
```

❖ <u>Hash Table</u>

- Spell checker reads all the words in the dictionary file into its hash table with the following hash function.

```
def hash(str):
    h = 0
    for ch in str:
        h *= 37
        h += ord(ch)
    return h
```

- Table size is a prime number (rounded to the next nearest prime). Initial table size is approximately 20% of total number of words in the dictionary. Table size is doubled (approximately) when load factor is greater than or equal to 0.5.

- Two collision handling approaches:-
  - separate chaining (each chain could be implemented with a LinkedList or a BST. Totally your choice.)
  - linear probing

❖ <u>Statistics</u>

- Collect the following data when adding valid words into hash table
  1. Number of times the table is expanded
  2. Load factor
  3. Number of times collision happens
  4. Length of collision chain which is
     - number of longest consecutive collisions (when linear probing is used), or
     - length of the longest list (when separate chaining is used)

  Note that these values must be reset after each table expansion.

- When all valid words are added into the table, show the statistics as follows.

```
Collision resolution : [linear probing or chaining]
Total words : x
n expansions, load factor f, n collisions, longest chain n
```

Submission: In-lab and post-lab : November 24 at 2:30pm.