Information Retrieval Lab 4

1. Objectives

- Study method of storing inverted index on disk
- Write code to hash words and store on disk
- Retrieve from disk index

2. Storing Index on Disk

Review the notes on the use of disk files for storing an inverted index.

We will only try a simplified task here: Hashing words and then retrieving them. We will work with ASCII files.

3. Byte Offset in Python

Look at sdf_show_offset() in sdf_seek_data_file.py.

It shows how to find the current byte offset in a file, using f.tell().

Make sure you have file input.txt and then run sdf_show_offset().

Note that each ASCII character counts as 1: If you write ‘a’ for example, the offset increases by 1.

Note that each newline counts as 2 (on Windows): CRLF.

4. Seek to Byte Offset

Now look at sdf_write_at_offset(filename, word, offset)

It will write a string at a byte offset. Try these and look at t1.txt to see the result:

sdf_write_at_offset('t1.txt', 'z', 0)

sdf_write_at_offset('t1.txt', 'z', 3)

sdf_write_at_offset('t1.txt', 'z', 9)

Remember, CRLF (end of line) counts as two characters.

Note that the number of characters in the file remains the same. Individual bytes are being overwritten.

WARNING: The string being written to the file must always be no longer than the existing line if you want the number of lines to remain the same!

5. Experiment with Hash Algorithm

In the slides, you can find a simple hash algorithm (around Slide 150, Example Hashing Algorithm).

Example:

Word	Ascii codes	Sum	Mod 20	Mod 5
lion	[108,105,111,110]	434	14	4

For an ASCII string, you sum the ASCII codes and perform Mod 20 to map any string to one of 20 values, 0-19. You choose the value of the Mod to fix the size of the hash space - very convenient.

Write a function sdf_hash(word, mod) which returns the hash.

So sdf_hash('lion', 20) should return 14.

6. Store a Word with the Hash Algorithm

We will start with a 'blank' file which already contains *n* lines each of length line_size.

line_size does not include the CRLF at the end of each line (two more characters).

You can create such a file using sdf_create_blank_data_file(filename, rows, line_size).

Try out this function and verify that it works. The lines are made out of '%' characters so you can easily see the result.

Now, write a function sdf_store_word(word, data_file, line_size)

word is a string you wish to store
data_file is a .txt file which already contains many lines of length line_size

Suppose we have a file like this:

```
%%%%%%%%%%
%%%%%%%%%%
```

- two lines each containing 10 %s, plus CF LR
- total number of characters 2*(10+2) = 24.

Seek to position 12 and overwrite with `dog`:

```
%%%%%%%%%%
dog%%%%%%%
```

Seek to position 0 and overwrite with `hello`:

```
hello%%%%%
dog%%%%%%%
```

This is our hash data being stored on disk.

Warning: Data added must always be no longer than the line!!

So, the function sdf_store_word(word, data_file, line_size) should:

- Check length of word is shorter than line_size
- Compute the hash *h*
- Open data_file
- Seek to the start of line *h* in the file
- Write the word
- Return True if succeeds, otherwise False

If word is too long, do not write but give an error message instead and return False.

If the line you seeked to already contains data, this is a Hashing clash (see lecture notes for more information). Do not write, give an error message and return False.

7. Look Up a Word

Suppose we have a word and we wish to see if it is in the file. Write a function which will look it up:

sdf_retrieve_word(word, data_file, line_size)

It will:

- Compute the hash
- Open file
- Seek to the right place
- Retrieve the word at that place (read the line, remove the trailing '%')
- If word retrieved is the same, return True, otherwise False

Note: This is just an experiment. In a proper system, we need to store some information as well as the key. e.g. we could store the offset in the Postings File for that word.

8. Store Words and Choose Mod

words.txt contains a suitable list of 100 words.

Write a function sdf_store_all_words(word_file, data_file, line_size).
It will store all the words in word_file in data_file which has line_size.

Find the smallest value of *n* which gives no clashes for those words.

Hence create final_data.txt.

Hint 1: n will be a lot bigger than 100, so there will be many lines in final_data.txt which are all % signs.

Hint 2: Fix line_size to any value which is larger than all the words. e.g. 30.

9. Produce Sample Output

Create a transcript file output.txt where you:

Call sdf_hash on two words, 'wolf' and 'lion'.

Create a blank data file 'test_data.txt', line_length = 8, 20 rows.

Use two calls of sdf_store_word() to store wolf and lion in 'test_data.txt'.

Then print test_data.txt to show the two words in it.

Call sdf_retrieve_word() on 'wolf', 'lion' and 'dog' ('dog' is not in there, of course).

10. Files and Functions

Call your program sdf_seek_data_file.py - just add your code to the end of it.

As described above, you need to write the following functions

sdf_hash(word, mod)

sdf_store_word(word, data_file, line_size)

sdf_retrieve_word(word, data_file, line_size)

sdf_store_all_words(word_file, data_file, line_size)

You also need to create the following files:

final_data.txt

output.txt

11. Upload your program to Moodle

Upload sdf_seek_data_file.py and final_data.txt.

On Moodle, go to Week 5, i.e. 27 September - 3 October. Click Week 5 Lab 4. Upload your files there.

Please check you have:

- Comment at the start (marks for this) with your name and number
- Code for the functions added to sdf_seek_data_file.py
- Files output.txt and final_data.txt