COMP3009J Information Retrieval

Worksheet 3

The purpose of this week's worksheet is to explore the efficiency that using skip lists can bring, compared to using ordinary linked lists.

Download the file named "Worksheet3.py" from Moodle. This contains the following:

- A class called `SkipList` to represent a linked list with skip pointers.
    - This has the following attributes:
        - `start` – a reference to the first node in the list (of type `SkipNode`)
        - `length` – the number of elements in the list (type `int`)
    - It also has a constructor that allows you to pass in a list of elements to store in a `SkipList`.
- A class called `SkipNode` to represent a node in a skip list.
    - This has the following attributes:
        - `next` – a reference to the next node in the list (type `SkipNode`).
        - `skip` – the skip pointer. If this node does not have a skip pointer, this will be `None`. If it does, this is a reference to another node later in the list (type `SkipNode`).
        - `element` – the element stored in this node.

The file also contains:

- a function called "`intersect(…)`" that calculates the intersection of two lists *without* using the skip pointers.
- a section of code at the end of the file that creates two skip lists and measures the time it takes to merge them using the `intersect(…)` function.

Timing your code

When comparing the time efficiency of a piece of code, it is often useful to measure the time it takes to run. A typical way to do this is to run the code many times. Python's `timeit` library allows you to do this. When calling its `timeit(…)` function, a few things are needed:

- A string that contains the code you want to run.
- The `number` parameter says how many times you want the code to be run.
- Setting `globals=globals()` allows `timeit` to see variables and functions that you have defined.

Tasks

Q1. Study the code in the intersect(…) function and the timing code to make sure that you understand it. If you don't understand it, ask!

Q2. Make sure you can run the code as it is.

Q3. Create a new function called "`intersect_skip(…)`" that is based on the code in `intersect(…)` but uses the skip pointers in the list. Be sure to check that the output of your function is the same as `intersect(…)`.

Q4. Measure how long it takes `intersect_skip(…)` to merge the two lists, compared to `insersect(…)`. You should ask `timeit` to run each function a larger number of times (e.g. 1,000).

Advanced

If you finish the above exercises early, here are some more you can try.

Q5. Conduct an experiment to see what effect the use of skip lists has on an OR merge (i.e. the union of two postings lists).

Q6. In the constructor for the `SkipList`, the `skip_frequency` variable is used to control how many skip pointers are in the list (it is approximately the square root of the number of elements in the list). If you change this frequency, how does this affect the performance of the list?

Q7. Adapt the program to work with the postings lists from Worksheet 2. Are there circumstances where using the skip list makes no difference, or makes the performance worse?

Q4. Copy “q3.py” to a file named “q4.py” and modify it so that it counts the number of times each word appears in the text. For each word, print the word and the number of times it appears.

Sample output (the order of your output might be different):

```
--------------------
2048: is
128: answer
4096: this
...
--------------------
```

Hint: You will need to choose an appropriate data structure for this task.

Q5. Copy “q4.py” to a file named “q5.py” and modify it so that it prints the words (and their frequencies) in order of frequency. The most common word should be first, followed by the others in descending order.