CSCI 4041: Algorithms and Data Structures (Day & Evening)
Programming Project #1
March 21, 2016

Last revision March 21, 2016

0. Introduction.

A Bloom Filter is an efficient data structure that uses hashing to test if an object is a member of a large set. Bloom Filters are named for their inventor, Burton H. Bloom, who published a paper describing them in 1970.

Bloom Filters are probabilistic data structures. What does that mean? Suppose that we use a Bloom Filter to represent a set of objects. We ask if an object is in the set. If the Bloom Filter answers "no," then the object is definitely not in the set. If it answers "yes," then the object is in the set only with some (known) probability.

Because Bloom Filters are probabilistic, we can't use them for applications that require reliable answers. However, we can use them in spelling checkers. To do that, we represent a large set of correctly spelled words as a Bloom Filter. If the Bloom Filter says that a word is not in the set, then the word is spelled incorrectly. If it says that the word is in the set, then the word may or may not be spelled correctly—it may make a mistake. However, we can design the Bloom Filter so that the probability of mistakes is small.

All we need now is to decide what words should be in the set. Basic English is a proposed international language invented in 1930 by Charles K. Ogden. It uses 850 words from ordinary English, and a simplified English grammar. Ogden intended Basic English to be easy for non-English speakers to learn and use, but there has been little interest in it since the 1940's. However, if it had become popular, then today we would need programs that find spelling errors in Basic English text. For this project, you must write a Java class that uses a Bloom Filter to detect misspelled Basic English words.

1. Theory.

A Bloom Filter uses an array b of M bits, where M is large. All the bits in b are initially 0, representing an empty set. We also have a large number of words, represented as strings.

Suppose we want to add a word w to the set represented by the Bloom Filter. To do that, we compute the values of j distinct hash functions $h_1(w)$, $h_2(w)$ ..., $h_j(w)$. Each hash function returns an index into the array b. The indexes returned by the hash functions need not all be distinct. We set all the indexed elements of b to 1.

$b[h_1(w)] = 1$

$b[h_2(w)] = 1$

⋮

$b[h_j(w)] = 1$

Now suppose all the words have been added to the set represented by the Bloom filter, and we want to test if a word w is in that set. To do that, we test if the array elements $b[h_1(w)]$, $b[h_2(w)]$ ..., $b[h_j(w)]$ are all 1. If they are, then w might be in the set—but we don't know for sure,

because there may have been collisions. However, if at least one is 0, then we know for sure that w is not in the set.

How likely is it that the Bloom Filter will make a mistake? Suppose that the Bloom Filter represents a set of N words. Then p is the probability the Bloom Filter will wrongly report that a word is in the set, even though it is not.

p = (1 – e– jN/M) j

(Here e ≈ 2.718281828, the base of natural logarithms.) Using this equation, if we know how accurate we want our Bloom Filter to be, then we can determine the length of its bit array M, and how many hash functions j it needs.

Now let's consider the bit array b. The obvious way to implement b in Java is to use an array of boolean's. However, we don't know how efficiently Java represents such an array. Does it represent each element as a single bit? A byte? A word? We don't know, and Java won't tell us. Java's representation is important because M might be large. For efficiency, we must implement b as an array of Java int's, and use the bits in the int's as b's elements.

Here's how to do that. Let a be a Java array of int's. Each int has 32 bits. We use the bits of a[0] to represent the elements of b at indexes 0 through 31, and the bits of a[1] to represent the elements of b at indexes 32 through 63, etc. This is an efficient representation because it uses only one bit per element. We initially set all the int's of a to 0, so all the bits of b are 0 as well.

Suppose we want to get the value of b[n]. We first get the int element a[⌊n / 32⌋]. Then we make another int m, with a 1-bit at position n mod 32, and 0-bits elsewhere. We logically AND these two int's together. If the result is 0, then b[n] is 0, otherwise b[n] is 1.

Now suppose we want to set the value of b[n] to 1. We compute a[⌊n / 32⌋] and m as before. We logically OR the two int's together, and store the result back into a[⌊n / 32⌋]. The Bloom Filter never sets bits to 0, so this is all we need.

2. Implementation.

You must write at least two Java classes. The first class, called BitArray, implements an array of bits. It must have at least the following methods, and they must work as described here and in the previous section.

public BitArray(int M)

(5 points.) Constructor. If M < 0, then throw an IllegalArgumentException. Otherwise, make a new int array that can hold M bits, to be used by the other methods of this class. All the int's in the array must be initialized to 0. Note that the array's length is not necessarily M, but is computed from M.

public boolean get(int n)

(5 points.) If n < 0 or n ≥ M, then throw an IndexOutOfRangeException. Otherwise get the bit at index n, using the array that was made by the constructor. If that bit is 0, then return false, otherwise return true.

public void set(int n)

(5 points.) If $n < 0$ or $n \geq M$, then throw an IndexOutOfRangeException. Otherwise set the bit at index n to 1, using the array that was made by the constructor.

The second class, called BloomFilter, uses an instance of BitArray and several hash functions to probabilistically represent a large set of strings. It must have at least the following methods, and they must work as described here and in the previous section.

public BloomFilter(int M)

(5 points.) Constructor. If $M < 0$, then throw an IllegalArgumentException. Otherwise make a new instance of BitArray, whose size is M, to be used by the other methods of this class.

private int h1(String w)
private int h2(String w)
⋮
private int hj(String w)

(10 points.) Hash functions. Each method h1, h2 ..., hj must use the word w to compute an int greater than or equal to 0 and less than M. These int's are indexes into the BitArray made by the constructor. Each method must also work in O(w.length()) time.

public void add(String w)

(5 points.) Add the word w to the set, using the BitArray made by the constructor, and the hash methods h1, h2 ..., hj.

public boolean isIn(String w)

(5 points.) Test if the word w is in the set, using the BitArray made by the constructor, and the hash methods h1, h2 ..., hj. Return true if w may be in the set. Return false if w is definitely not in the set.

public double accuracy()

(5 points.) Return the probability that the method isIn will report a word is in the set, when it is not.

You must also write a driver class with a main method (10 points). The main method must show that the BloomFilter class works correctly. All printing must be done by main: no other methods are allowed to print anything. Here's what main must do.

Make a new instance of BloomFilter.

The file basic.txt (available on Moodle) contains the 850 words of Basic English, one per line. Read these words from the file and add them to the instance of BloomFilter, using its add method. The file LineReader.java (also available on Moodle) contains Java source code for an iterator that may make this easy.

Read the 850 Basic English words again, and test if each word is in the instance of BloomFilter, using its isIn method. Print all words for which isIn returns false. For full credit, all Basic English words should be recognized by isIn, so no words should be printed. You will lose 1 point for each Basic English word that your Bloom Filter reports as misspelled, when it is not. You will not lose more than 5 points for such words.

Print the value returned by the accuracy method, so we know how accurate the instance of BloomFilter is for the 850 words of Basic English.

Here are answers to some questions you may have about this project. Other questions and answers may be added later.

Q.

Do I have to write in Java?

A.

Yes. We need a somewhat low level language in order to do some things that this assignment requires. Java is the only such language that the TA's will grade. That's an important consideration in a class this size.

Q.

How accurate (p) do I need to make my Bloom Filter?

A.

There's no specific requirement. The requirements are that your Bloom Filter must identify all 850 Basic English words, and that its accuracy method must report how accurate it is.

Q.

How do I write the accuracy method?

A.

Use the formula for p.

Q.

How big (M) do I need to make the bit array?

A.

The formula for p determines that. Also consider some things about hash table sizes that were discussed in the textbook and the lectures.

Q.

How many hash methods (j) do I need?

A.

The formula for p determines that.

Q.

How can I write all those hash methods?

A.

The Java class String has a hash method called hashCode, so you have one already. You can write others yourself, or you can look for them on the Internet, or both. Try Googling "Java hash functions" or some similar phrase. Some Internet hash functions are badly designed, despite what their authors claim. It's your responsibility to use good ones, again according to criteria discussed in the lectures.

Q.

How many points is each hash method worth?

A.

All the hash methods together are worth a total of 10 points. If each hash method were worth some number of points on its own, then you could inflate your score by writing many of them. Also, you can copy hash methods from various sources, so they shouldn't be worth many points.

Q.

You said nothing about resolving collisions by chaining, with buckets and linked lists. Where do the linked lists go?

A.

Bloom Filters are hash tables, but they're different from the hash tables discussed in the lectures. They don't resolve collisions by chaining, so they don't use buckets, and they don't use linked lists.

Q.

Do I need to consider prefixes and suffixes of words? A real spelling checker would do that. For example, the word argument is detected by the Bloom Filter, but the word arguments is not.

A.

Your Bloom Filter must only identify the 850 Basic English words. It need not handle prefixes and suffixes. Doing that is a hard problem, much harder than the one you have here.

Q.

I know a way to cheat. I'll just write 850 hash functions, each of which identifies a different Basic English word, and indexes its own bit in the Bloom Filter. Can I do that?

A.

(Laughter.) You will receive no points for this project if you do that, or anything else like that. The number of hash functions (j) must be small compared to the size (M) of the bit array, and the number of words (N).

Note that some questions have deliberately not been answered. Parts of this project are left unspecified to make it allegedly more interesting.

3. Deliverables.

This assignment is worth 55 points. It will be due in about three weeks, on April 8, 2016. You must turn in Java source code for your classes BitArray, BloomFilter, and for your driver class. For convenience, these must be all in one file. Also turn in any output produced by your driver class. Put your output in a comment at the bottom of the file. Note that you will recieve no points for this assignment if the output you turn in is different from the output that is actually produced.