CS 61B Software Engineering, Re: Hashing, Re: Tries
Spring 2026 Exam-Level Discussion 12: April 13, 2026

1 Summary

Hash Tables are used as backing data structures for `Set`s and `Map`s, and, in the best case, allow for constant-time `add` and `contains` operations. In memory, we construct them as an array with a "separate chain" at each index, such as a linked list. Often, we refer to these array indices as **buckets**. N is used to denote the number of items in the hash table, and M is used to denote the number of buckets.

- **Add**: Take the object's hash code, apply modulus by M. Search the corresponding bucket. If the bucket does not contain the object, add it to the end of the separate chain. Runs in $\Theta(1)$ assuming **good spread** (see below).
- **Contains**: Take the object's hash code, apply modulus by M. Search the corresponding bucket. If the bucket contains the object, return `True`. Else, return `False`. Runs in $\Theta(1)$ assuming **good spread** (see below).
- **Resizing**: We define our hash table's **load factor** as the ratio N/M. We also define some load factor that will trigger a resize, and a **scaling factor** by which the array's size will be multiplied. After each `add` operation, we check N/M, and if our target load factor is reached, we initiate a resize. During a resize, we rehash every item in the hash table, which may change its corresponding bucket.
- **Good Spread vs Bad Spread:** To reduce the runtime of `add` and `contains` we want items to be evenly distributed within our hash table. `hashCode` implementations that result in a relatively even distribution are said to have "good spread". `hashCode` implementations that result in uneven distribution are said to have "bad spread".

Tries are trees specialized for language tasks. Each node represents a character, and "marked" nodes denote the ends of words.

- **longestPrefixOf(String s):** Follow the trie until the letters no longer match, keeping track of the most recent "marked" node. Runs in $O(L)$, where L is the length of the longest key. Alternately, can bound as $O(|s|)$ where is the length of `s`.
- **keysWithPrefix(String s):** Follow the trie until the end of the prefix, and return all words indicated by marked nodes below that node. Runs in $O(L + Z)$, where L is the length of the longest key and Z is the number of keys returned.

2 Hashing

from Spring 2021's midterm 2

(a) Throughout this problem, assume we're using a hash table to back a set. Suppose that each bucket of the hash table is stored as a left leaning red black tree, and we are inserting items that implement the Comparable interface. Which of the following statements are true about such a hash table? Write either **"true"** or **"false"** below each statement.

(a) The runtime of contains is $O(N)$

True

(b) The runtime of contains is $O(\log N)$

True

(c) The runtime of contains is $O(1)$

False

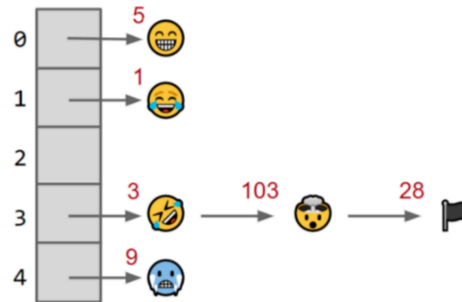
(d) One advantage of using an LLRB for the buckets is that it makes it possible to efficiently iterate over all of the keys in the set in ascending order.

False

(e) Assuming items are nicely spread out in the hash table, we expect that an LLRB bucket would yield significantly better performance for contains and add than if we used an `ArrayList` for each bucket.

False

(b) Suppose now that we build a Set using a hashtable, where we represent each bucket with a linked list. Suppose we've added the Picture objects below, with their hash codes listed in the top left.



(a) We add a new picture 👽 with hash code −6. In which bucket will 👽 end up? Assume that the hash table does not resize.

4. $-6 \bmod 5$ evaluates to 4.

(b) Suppose we resize our hashtable by doubling its size. Circle the items that will end up in a different bucket. Assume we're starting from the original hash table without 👽.

Items with hash code 5, 9, and 28 should be the only ones circled.

(c) Suppose that we perform the following actions on the original hashtable with 5 buckets illustrated in the image above...

```
Picture x = 🥶;
hashTable.add(x);
x.turnPink();
System.out.println(hashTable.contains(x)); // line 4
```

Assume that the turnPink method changes some of x so that it looks like 😳. This change to the object may result in its hashcode changing.

For which of the following hashcodes will line 4 of the above code print out `true`? Note that a `Picture` object's equals method returns `true` only if their pixel values are exactly identical.

- `x.hashCode` is −1
- `x.hashCode` is 0
- `x.hashCode` is 4
- `x.hashCode` is 14
- None of the above

Line 4 will print out `true` for hashCodes of −1, 4, and 14. All of these are congruent to $4 (\bmod 5)$.

3 Wordle

From Spring 2023's final exam

Write a class to store a Wordle dictionary. Your solution must allow for the following functions:

`insert(String word)`: Inserts a new word of length W into the dictionary and takes $O(W)$ time. All word arguments are unique, non-null, and consists of only CAPITAL letters or the empty string.

`contains(String word)`: Returns true if the dictionary contains the word and takes $O(W)$ time. All word arguments are unique, non-null, and consists of only CAPITAL letters or the empty string.

`wordList()`: Returns a list of all words in the dictionary, in alphabetical order. This must take $O(WN)$ time, where N is the number of words in the dictionary.

`getRandomWord(Random random)`: Returns a random word from the dictionary. This must take $O(W)$ time, so calling wordList will exceed your runtime. All words must be equally likely to be returned by your function regardless of the words in the dictionary.

You may assume all string operations in the reference sheet and string concatenation run in O(1) time.

```
class Wordle { // we will use a modified trie data structure!
  private boolean valid;
  private Wordle[] children;
  private int size;

  public Wordle() {
    valid = false;
    children = new Wordle[26];
    size = 0;
  }
  private int charToInt(char c) { return c - 'A'; } // Converts A to 0, B to 1,... Z to 25
  private char intToChar(int i) { return (char) ('A' + i); } // Converts 0 to A, 1 to
B,... 25 to Z

  public void insert(String word) {
    size += 1;
    if (word.length() == 0) {

      this.valid = true;

      return;
    }

    int index = charToInt(word.charAt(0));

    if (this.children[index] == null) {

      this.children[index] = new Wordle();

    }

    this.children[index].insert(word.substring(1));

  }
```

```
public boolean contains(String word) {
  return ((___1___) && (___2___)) || ((___3___) && (___4___) && (___5___))
}
```

Put the letter that corresponds to the code to run **contains**. An answer choice may be used more than once.

(a) `this.valid`

(b) `this == null`

(c) `word.length == 0`

(d) `this.contains(word.substring(1))`

(e) `children[charToInt(word.charAt(0))] == null`

(f) `children[charToInt(word.charAt(0))].contains(word.substring(1))`

(g) `!this.valid`

(h) `this != null`

(i) `word.length != 0`

(j) `!this.contains(word.substring(1))`

(k) `children[charToInt(word.charAt(0))] != null`

1: *a*	2: *c*	3: *i*	4: *k*	5: *f*

Options a and c can be in either order. For blanks 3 - 5, the correct ordering is i, k, f only.

```
  public List<String> wordList() {
    List<String> words = new ArrayList<>();

    if (this.valid) { words.add("") }

    for (int i = 0; i < 26; i += 1) {
      if (children[i] != null) {

        for (String s: children[i].wordList()) {

          words.add(intToChar(i) + s);
        }
      }
    }
    return words;
  }

  public String getRandomWord(Random random) {
    int randNum = random.nextInt(this.size);

    if (<randNum == size - 1|>{32} && valid) { return ""; }

      int letter = -1;
      while (randNum >= 0) {
        letter += 1;

        if (children[letter] != null) {

        randNum -= children[letter].size;

      }
    }

    return intToChar(letter) + children[letter].getRandomWord(random);

  }
}
```

4 Software Engineering

Note: This question is experimental. Let us know what you think.

In John Ousterhout's guest lecture, he talked about how the best classes are *deep*: they hide a lot of complexity, and the interface to users (i.e., what the users need to know in order to use the class) is comparatively small and simple. Let's reflect on how deep one of the classes that you wrote for project 4CD is. Open up your code on your laptop (really!), and pick one class that you implemented as an example.

What's the name of the class? ____________________

(a) What's one piece of information or complexity that is completely hidden within the class, so users don't need to know about it. Possible examples include "the relationships between different words and how they're stored" or "how we keep track of the word frequency for each year."

Whatever you wrote above is an example of hiding complexity within a class, which is great! That class is helping its users, by shielding them from complexity that the user doesn't need to know about.

(b) Next, identify something in that class that's not obvious from method signatures, but is something that a user needs to know. For example, do you have a set of somewhat obscure methods that must all be called in order to achieve some task?

At a minimum, this information should be described in a comment, to warn users about it. Is there any way to restructure your code to keep this from leaking out (the best solution!) or to make it obvious from the method signatures?

(c) As you were thinking about this and looking back at your code, you got a look at the variable and method names that you used. List the worst two variable and method names that you used in your code. Explain one to your group. Can anyone in your group come up with a better name?