

Questions

- Both hashing and binary search trees allow you to search through keys by taking advantage of the key ordering? What are the advantages/disadvantages of each.
- Suppose \mathbf{s} is a string. Define its hash code:

$$h(\mathbf{s}) = \sum_{i=0}^{n-1} \mathbf{s}[i] p^i$$

where $s[i]$ is the 16-bit unicode value of the character at position i in the string, n is the length of s , and p is some positive integer.

Give an upper bound on the number of bits needed for the hash code as a function of p and n (the length of the string).

- Suppose you have approximately 1000 images that you would like to store. Rather than labelling them using a string (filename) and indexing them based on filename, you would like to label them using small images called “thumbnails”. Let’s say each thumbnail image is 64×64 pixels, and each pixel is one of 256 colors.

Suggest a suitable hash function, namely one that avoids collisions and that doesn’t take too much space.

- Canadian postal codes are of the form $L_1D_1L_2D_2L_3D_3$ where L is always a letter (A-Z) and D is always a digit (0-9). Suppose you have your own company and you wish to index your customer addresses using the postal code as the key. Let the digits A-Z be coded with numbers 1 to 26, for example, $code(B) = 2$. The codes of the digits are just the digits themselves.

(a) Define a hash function:

$$h(L_1D_1L_2D_2L_3D_3) = \left(\sum_{i=1}^3 code(L_i) + \sum_{i=1}^3 D_i \right) \bmod 10.$$

Give an example of a postal code that begins with H3A and that collides with H3A2A7.

- (b) Give an example of a hash function that would *never* result in a collision. How large would the hash table need to be?

Answers

- With a BST, we would search for a key by following a path from the root towards the leaves. For each node in the BST we encounter, we do a key comparison ($<$, $=$, $>$). If the BST is balanced (best case), then on average it will take us $O(\log n)$ steps to find a key, or determine that there is no matching key, where n is the number of keys in the BST. So, for example, if $n = 2000$ and the tree is balanced, then it will take us on average roughly 10 comparisons to find an item (i.e. $2000 \approx 2^{11}$, and about that half the nodes in a complete binary tree are leaves.) If the BST is not balanced¹, then it will take longer to find the key. This is faster than using a linked list, but still slow.

With hashing, the idea is that there is a function $h(key)$ which provides a hash value which is a number from 0 to $m - 1$ where $m \approx n$. Given a key, k , we compute $h(k)$ using some formula or algorithm, and then we go directly to the entry $h(k)$ in the hash table and search (typically very short!) list for key k . If the hash function is easy to compute, then you can find the key more quickly than with a BST.

Another way to think about the difference is to note that binary search trees do a sequence of comparisons, i.e. they ask for one of three values ($<$, $=$, $>$), which is less than 2 bits of information. By contrast, a hash function gives you $\log m$ bits of information, namely it specifies one of 2^m values. You still need to scan the entries in the bucket at index $h(k)$. But if the hash function does a good job, then most of the buckets will have very few elements.

- The largest hash code is $2^{16}(p^0 + p^1 + \dots + p^{n-1}) = 2^{16}(\frac{p^n - 1}{p - 1}) < 2^{16}p^n$. Taking the log gives the number of bits of the hashCode: $16 + n \log p$.

For example, if $p = 31$ (as in Java's String class's hashCode method), the hashCode would be at most $16 + 5n$ bits.

```

                *****          s[3] x 31^0
            *****          s[2] x 31^1
        *****          s[1] x 31^2
+   *****          s[0] x 31^3
    -----
*****          hashCode( s )
    
```

- Let's make a hash table with $m = 2000$ entries so that we are sure there will be plenty of buckets with no elements (and hence collisions are relatively rare).

For the hash function, we need to map the 64×64 pixel colors to a number larger than $m = 2000$. To do so, we could define a hash code to be the sum of the intensity values at the pixels. Assume the average intensity value is 128, we would get a number on average of $64 * 64 * 128$ which is much bigger than m . Then, to get the hash value, we could take the *sum* of the color values and compute "*sum mod m*".

¹keep in mind that it takes extra work to keep it balanced – wait for COMP 251

4. (a) $h(\text{H3A2A7}) = (8 + 3 + 1 + 2 + 1 + 7) \bmod 10 = 2$

So, you need to come up with a postal code $D_2L_3D_3$ such that $D_2 + \text{code}(L_3) + D_3 \bmod 10$ is the same as $2 + 1 + 7 \bmod 10$. The latter is 0. So, for example, if you take "3A6", then $D_2 + \text{code}(L_3) + D_3 \bmod 10$ is 0, and $h(\text{H3A3A6}) = (8 + 3 + 1 + 3 + 1 + 6) \bmod 10 = 2$

(b) One simple solution is to use base $b = 26$ and define:

$$h(L_1D_1L_2D_2L_3D_3) \equiv \text{code}(L_1) + D_1b + \text{code}(L_2)b^2 + D_2b^3 + \text{code}(L_3)b^4 + D_3b^5$$

In these cases, the postal code $Z9Z9Z9$ would give the largest hash value, and you can plug in the numbers and letters to get the largest value. That is how big the hash table would need to be for this hash code.

Notice that there are $10^3 * 26^3$ possible postal codes. You could come up with a hash function that maps each postal code to one of the numbers from 0 to $10^3 * 26^3 - 1$. It is not so difficult to do this.