

Hash Tables

Concepts

Goals

- Hash Functions
- Dealing with Collisions

Searching...Better than $O(\log n)$?

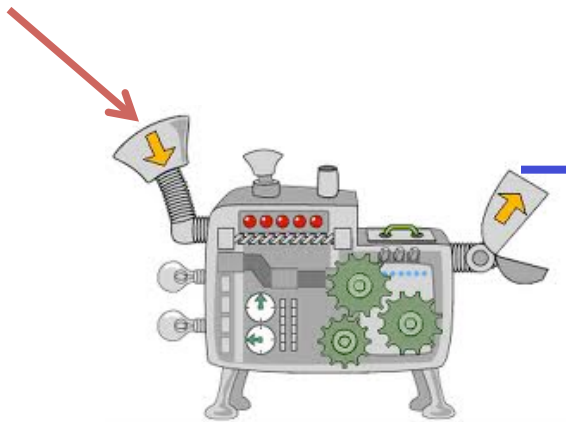
- Skip lists and AVL trees reduce the time to perform operations (add, contains, remove) from $O(n)$ to $O(\log n)$
- Can we do better? Can we find a structure that will provide $O(1)$ operations?
- Yes. No. Well, maybe. . .

Hash Tables

- Hash tables are similar to arrays except...
 - Elements can be indexed by values other than integers **Huh???**
 - Multiple values may share an index **What???**

Hashing with a Hash Function

Key
ie. string,
url,etc.



**Hash
function**

**integer
index**

Hash Table

0	Key y
1	Key w
2	Key z
3	
4	Key x

*Hash to index for storage AND
retrieval!*

Hashing to a Table Index

- Computing a hash table index is a two-step process:
 1. Transform the value (or key) to an integer (using the hash function)
 2. Map that integer to a valid hash table index (using the mod operator)
- Example App: spell checker
 - Compute an integer from the word
 - Map the integer to an index in a table (i.e., a vector, array, etc.)

Hash Function Goals

- FAST (constant time)
- Produce UNIFORMLY distributed indices
- REPEATABLE (ie. same key always results in same index)

Step 1: Transforming a key to an integer

- Mapping: Map (a part of) the key into an integer
 - Example: a letter to its position in the alphabet
- Folding: key partitioned into parts which are then combined using efficient operations (such as add, multiply, shift, XOR, etc.)
 - Example: summing the values of each character in a string

Key	Mapped chars (char in alpha)	Folded (+)	
eat	5 + 1 + 20	26	

Step 1: Transforming a key to an integer

- Shifting: can account for position of characters

Shifted by position in the word (right to left): 0th letter shifted left 0, first letter shifted left 1, etc.

Key	Mapped chars (char in alpha)	Folded (+)	Shifted and Folded
eat	$5 + 1 + 20$	26	$20 + 2 + 20 = 42$
ate	$1 + 20 + 5$	26	$4 + 40 + 5 = 49$
tea	$20 + 5 + 1$	26	$80 + 10 + 1 = 91$

Step 1: Transform key to an integer

- Mapping: Map (a part of) the key into an integer
 - Example: a letter to its position in the alphabet
- Folding: key partitioned into parts which are then combined using efficient operations (such as add, multiply, shift, XOR, etc.)
 - Example: summing the values of each character in a string
- Shifting: get rid of high- or low-order bits that are not random
 - Example: if keys are always even, shift off the low order bit
- Casts: converting a numeric type into an integer
 - Example: casting a character to an int to get its ASCII value
 - ie.

```
char myChar = 'b';  
int idx = (int) myChar;
```

Typical Hash Functions

- Character: the char value cast to an int → it's ASCII value
- Date: a value associated with the current time
- Double: a value generated by its bitwise representation
- Integer: the int value itself
- String: a folded sum of the character values
- URL: the hash on the host name
- Use the provided hash function!!! (ie. Java classes inherit a hashCode function ...which you can override if desired)

Step 2: Mapping to a Valid Index

- Use modulus operator (%) with table size:
 - Example: `idx = hash(val) % size;`
- Use only positive arithmetic or take absolute value
- To get a good distribution of indices, prime numbers make the best table sizes:
 - Example: if you have 1000 elements, a table size of 997 or 1009 is preferable

Hash Tables: Collisions

- A **collision** occurs when two values hash to the **same** index
- We'll discuss how to deal with **collisions** in the next lecture!
- Minimally Perfect Hash Function:
 - No collisions
 - Table size = # of elements
- Perfect Hash Function:
 - No collisions
 - Table size equal or slightly larger than the number of elements

Minimally Perfect Hash Function

Position of 3rd letter (starting at left, index 0) , mod 6

Alfred $f = 5 \% 6 = 5$

Alessia $e = 4 \% 6 = 4$

Amina $i = 8 \% 6 = 2$

Amy $y = 24 \% 6 = 0$

Andy $d = 3 \% 6 = 3$

Anne $n = 13 \% 6 = 1$

0	Amy
1	Anne
2	Amina
3	Andy
4	Alessia
5	Alfred

Hashing: Why do it??

- Assuming
 - Hash function can be computed in constant time
 - computed indices are equally distributed over the table
- Allows for $O(1)$ time bag/map operations!

Application Example

- spell checker
 - Know all your words before hand
 - Need FAST lookups so you can highlight on the fly
 - Compute an integer index from the string