

Chapter 20 notes

Hash table want to make searching constant average time

You can know if something is there or not instantly

Better then binary search tree but there is not gonna be as much

All we can do with hash table is:

Add stuff

Delete stuff

Add stuff

All in constant time

We lose ordering so we cant print. Sorting is gone

Basic ideas:

Items have a name

When there is more finding should be harder? But no not true

We aren't going to use integer we are going to use string but array

Example:

Spot["hello"];

Hash function: is taking the key and you get back the item position in array

If s is a string, we can convert to a large integer x by using the method suggested previously and then apply the mod operator to get a suitable index

Example:

If you have alphabet a-z and have 10 spot to find where z is you do $26\%10$ so spot 6

Collision: means that 2 things go to the same spot

On the back end of a hash is an array

Task:

When you are giving a key where do we find that in an array

$$\begin{array}{|c|c|c|c|} \hline & 0 & 1 & o \\ \hline + & 1 & 1 & 0 \\ \hline \end{array}$$

=

$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline \end{array}$$

HashSet:

A set contains no duplicates

Hashset uses a has to implement the set

A TreeSet gives sorted order

HashMap:

Allows duplicates

Does not keep keys in sorted order.

Every single object that .hashCode()

Every single object has a built in way to get its own hash value

On the test

Linear Probing:

Size is 10 and number we compute we have to map in down to 10 so anything we just do number%10 to find it

So if the spot is full then it will overflow. So $89\%10$ is spot 9 $49\%9$ is also supposed to be in spot 9 but it's full so it goes ++ and moves to next spot which is spot 1 and see if it's empty it is so it goes to spot 1.

Problem:

What if there are no free spots?

Big Oh:

assume hash table is large. It is constant. But when things start grouping up

As the load factor increases the faster we get away from constant time.

The hash function tells you where to start.

Quadratic Probing:

Start at h then add 1^2 then we add 4 then 9

Table sizes that are prime number are great for quadratic probing

To insert $H + 1^2$ then $H + 2^2$ then $H + 3^2$ and so on

Separate Chaining Hashing:

hash into an array of linked lists

Nice way to go, it avoids

-primary clustering

-secondary clustering

Uses of Hashing

Compilers use them to keep track of variable names

In games hash are useful to see if you've seen this position or not

Dictionary use them

Chapter 20 programming

Quadratic probing

But change it to linear probing
Take while loop and find a new spot that is empty
For programming figure 20.16

20.20

Check to see if they are cluster or are in the 100 thousands