Supposed to be hashing…

Need to maintain a list that is changing over time

We want to be able to quickly and space efficiently

1. query

2. insert

3. delete

4. update

Scenarios:

1. a small known input domain from which we are drawing the data. We want to track age distribution of collection of people(in years)

Read all the data into the list, then sort it

If there are o(n) to sort n data points

Or you could:

Subdivide the data into decades

May reduce to o(n/10)

To search for something

Or you could:

Create and initialize a list with 120 entries

Use to count number of each age

Agelist[age] = [agelist[age] + 1

Every task can be done in constant time o(1)

\*\* startedwi

Scenario 2: \*\*\*arbitraritly large input donaia0

You want to keep track of the powerballl winners from recnt years

174 million combiminations

1. map each powerball outcome to a list index and use that indecto count the number of times that outcome occur

Drawbacks:

1. huge input domain

2. huge waste of memory

Alternatives;

1. dump all inputs unordered into a list

Query, o(n)\_ searcj emtore search entire list with n data points, we could sort data as it comes in

Query of sorted list 0(logn

Insert a data point 0(n)

Scanrio 3:

Catalog words in a book & number of occurrence

We don’t even know what all possible input could

HASH TABLE:

List that stores

In python we have a data structure known as a dictionary: stores (key, value) pairs

Key – words – “doctors”

Value --- number of occurrances

Generally called a data structure, a MAP

For eaech pair it stores the number of occurrances and complexity that we finished

HASH TABLE REQUIREMENTS:

1 we need to be able to convert an arbitrary input (immutable) to an index location

2. convert to an integer ion another function

A = 0

B =1

C = 2

Z =25

2\*26^2 + .26^1

+ 19\*26

This conversion is the HASH FUNCTION – takes input value and converts into an ineger (used to access our table)

Problem: even for strings of length 10:

Alpha, 26&9 aaaae3 , alpha 2 + 26^8 … alpha number

2. keep the hash table a manageable so

We cam artificially reduce table size using modulo division.

Seea….if we change table size = 1000

Hash function % 1000, modulo 1000, for remainder

COLLISIONS now we have them.

Handling collisons:

Abs(s) >> n, have a table of values, group of s values maps to different locations

Allows a list of values or whatever to be stored, listof elements

Chaining mapping alist of elements in the same, sinlular, locato

Rist is query seach slows down if lots of collisions and long lists

Chaining, store a growing list of all(key, value) pairs that mot to a gove

Open addressing:::::: doing it naoww!!

Each key mpas to a certain table index, if that location is empty, then you can insert anew key there, else you stp forwad through the table, until youfind an empty spot

--- search starts at mapped point, not necessarily the beginning of the lost\*\*

Considerations:

Put these 4 values in table

S1

S2

S3

S4

Table:

S1  
s3

S2

S4

Query: stop search,

1 found key (may be at first location, or some point after)

2 you found empty spot

4 you wrap around to start point1)

111111Table is full? Rehashing, make a bigger table

2222 When does efficiency decrease when lots of keys map to the same location in table

3333 what will an ideal hash be as da?

Be as random as possible