CS-21 LECTURE #6

Program 3 now due Friday!

HASH TABLES WILL BE ON THE EXAM!

MIDTERM INFO - In class, written, closed book, 90 min - 2hr time expectancy, primarily writing functions, 7 questions, 5 will be writing functions, 2 will be ..., quicksort, priority queues, heaps, WILL BE ON EXAM, insertion sort, ~~count~~ sort, merge sort will NOT BE ON EXAM! Radix bucket hybrid Count will be on exam, & hash tables, & linked list review, stack, queue.

- How to study: Memorize code! or internalize it!

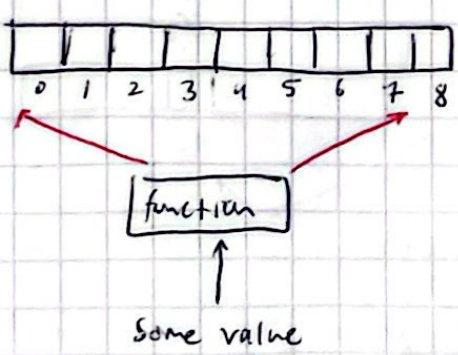
Dynamic Set

- tracks membership that could change over time
  - Insert, delete, find functions... ← A Dictionary! (ADT)

what datastructure can we build that has the above functions?

- A HASH TABLE → Expected time:
  - Built on an array!

insert, remove, search: O(1)

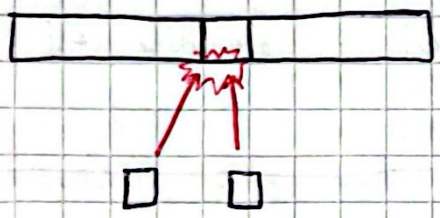


Basic Idea: Hashing Function

* Given value, it returns an index, & we index into a container

2 complications:

1. Writing a good hashing function.
2. Collisions: when two different items are trying to go to the same spot in the hash table!

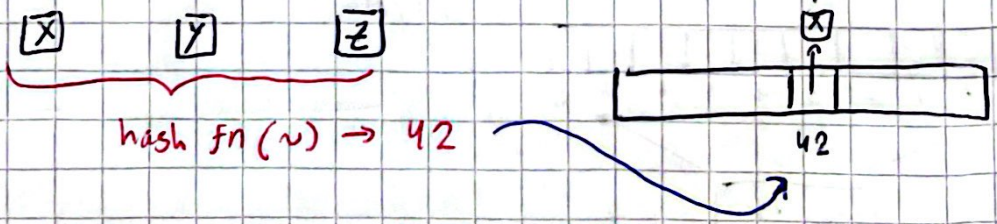


How to resolve?

- Chaining → >1 entry per sort
- Open addressing → >1 possible location

(simplest?) techique for Collision resolution → **Chaining**

- each 'spot' in the array is a list, NOT a single element.



- USE a vector in C++ for the list!
- MAKE AN ARRAY OF VECTORS!

A [hash (somedata)]. insert (some data)

Get index where the data belongs

Insert the data at the index specific to it.

Expected time:

$\Theta(1+\alpha)$ → α = load factor = $\frac{m}{n}$ = $\frac{\text{data size}}{\text{array size}}$

Rule for search & delete: Delete first found of the value you want to remove

Hashing Functions

Def: simple uniformed Hashing! = All spots in the table are equally likely. Simple concept, NOT simple to implement.

3 techniques: BAD, GOOD, BETTER

1. BAD! Division method: ← Not using size of table.

$h(k) = k \bmod m$

Performance method is critical on the value of m.

- Usually a power of 2 is a bad idea.
- Prime # not near a power of 2

(2) GOOD! ← we will use this on assignment!

Multiplication Method

Floor

$$h(k) = \lfloor m(kC - \lfloor kC \rfloor) \rfloor$$

where $0 < C < 1$ ← C is a constant

Choice of C depends on data

$\cong$ 0.618034 $\qquad \frac{\sqrt{5}-1}{2} = \frac{1}{\Phi}$

↑ Use this # for C (inverse of golden ratio)

---

hash (k) = (int) (m * (k*C - (int) (k*C)))

[0.00 - 1.00]

k = key value of data inserted into hash table.

k*C - (int)(k*C)

```
  xxxxxx.yyyyyy
- xxxxxx.000000
--------------
  00000.yyyyyy
```

└ value between 0 & 1 ——→

→ Val from 0-1 gives a %. & is multiplied to M ~~log~~ to get index inside of array/hash table.

L SO MUCH CLEANER & EFFICIENT

NOT DEPENDENT ON M like division method

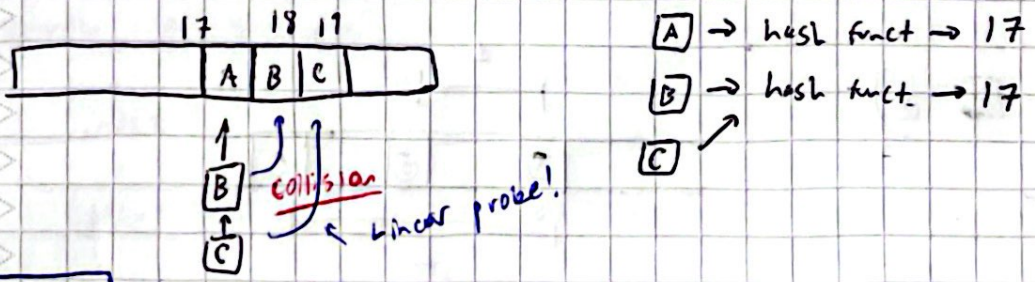
(3) GREAT! ← Not using

select C at runtime

Method #2 for collision Resolution

← Not using this on assignment #5

By open addressing on collision, pick another spot



Method 1: Linear Probe

$$h'(k, i) = (h(x) + i) \% m$$

k: value, i: attempt

Easy to code but suffers from primary clustering. Its like a traffic jam!

Method 2: Quadratic Probe

$$h'(k, i) = \left(h(k) + C_1 * i + C_2 * (i * 1)\right) \% m$$

C_1 & $C_2 \neq 0$. good choice of C_1, C_2, & m is difficult

Problem: Secondary clustering. Like traffic on backroads rather than freeway.

method 3

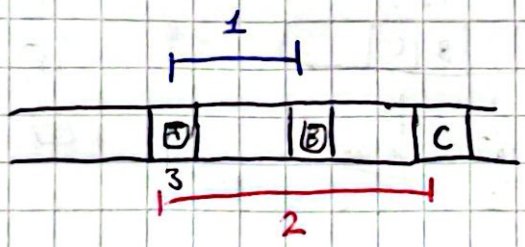
Double Hashing → Avoids primary & secondary clustering

$$h'(x, i) = \left(\overbrace{h(x)}^{1} + i \cdot \overbrace{h_2(x)}^{2}\right) \% m$$

original Hash funct

second hash funct w/ diff choice of C.

A B C



puts your values far away from others!

In your Program:

hash function (int) ← key value

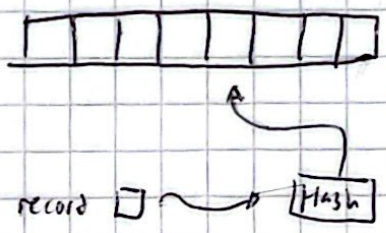
(record) ← gets key value from record

record

<int, string>

insert (record)

A [hash (record)]. push_back (record)



Search, Delete (record)

index = hash (record)

loop through entries in A [index]

time complexity matters!

Performance matters!

↓

fast array delete ?