CS21 WEEK 06

↳ Hash Tables

↳ Midterm: (90 min - 2 hrs) ← Expected Time

- In class, written exam
- Closed book
- Primarily Writing Functions
- Can memorize code or memorize steps to then write code
- Code in C++/Java for full cred. or code in psuedo for partial cred.
- Won't have time to write comments

→ Def on Exam:
- Quicksort
- Radix Bucket Hybrid
- Hash Tables (Only exact var multiplication method)
- Priority Queue/Heaps
- Review Stuff
  ↳ Linked Lists, Stacks, Queues

(Guestimating ~7-10 q's)

NOT ON EXAM
- Insert Sort
- Merge Sort
- Counting Sort

HASH TABLES

DYNAMIC SET
- tracks membership that can change overtime

insert
delete
Find/Search

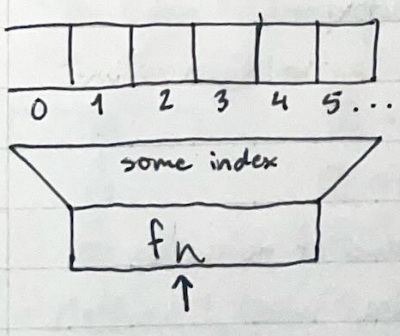
] Dictionary ADT (Abstract Data Type)

- Array PFA
- Linked List

] Insert O(1) Search Delete O(n)

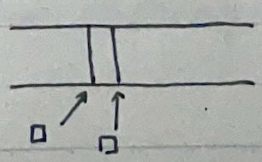
Hash Table — expected time insert, remove, search O(1) (w.c. is O(n))

built on an array



Basic Idea: Hash(ing) Function
- Given a value
- Returns an index (index into container-array)

Complications:
- Writing a good hashing function
- Collision



How resolve?
- Chaining
- Open dressing

WORST:

```
int hash(something){
    return 0;
}
```

*A perfect hashing fn

n data

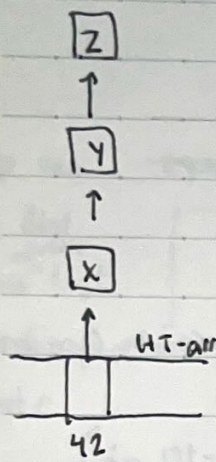
Required to support duplicates in our hash program

↑

causes collisions

(Simplest?) technique For Collision Resolution

- Chaining
  - ↳ Each 'spot' in an array is a list, not a single element.



$\underbrace{\boxed{X}\ \boxed{Y}\ \boxed{Z}}$

hashfn (~) → 42

* Good HT Lists:

Vector - C++

Array List - Java

PSUEDO

A[Hash(some_data)].insert(some_data)

Expected Time:

$\Theta(1+\alpha)$ α - load factor

⋮ $\underline{n}$ = data m: array

↑ size

$O(1)$

RULE FOR SEARCH & DELETE → First Found Only

HASHING FUNCTIONS

- Def: Simple, Uniform Hashing
  - ↳ All spots equally likely
  - ↳ simple concept, not simple implementation
- 3 Techniques: BAD, GOOD, BETTER

1) BAD, DON'T USE: Division Method

$h(k) = k \bmod m$ ← size of table

↳ Performance method is critical on value of m

↳ Usually a power of 2 is a bad idea

↳ Prime # not near a power of 2 (Eg. 701)

m = size of table

$\frac{1}{\phi} = \frac{\sqrt{5}-1}{2}$

$\phi + 1 = \phi^2$

2) GOOD ← What to implement

Multiplication Method

$h(k) = \lfloor m\,(kC - \lfloor kC \rfloor) \rfloor$ (k*c, floor)

- Where $0 < C < 1$ ← C is a const
- Optimal Choice of C depends on data $\cong 0.618034$

(inverse of golden ratio)

CODE

Hard cast into ints

$hash(k) = (int)(m * \underbrace{(k*C - (int)(k*C))}_{[0.00 - 1.00]})$

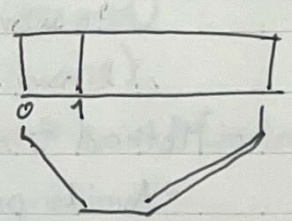
size of table

k * C

key value of data inserted in hash table

```
  xxxxx.yyyyy
- xxxxx.00000
  00000.yyyyy
```

Value between



0-99%

Ideal Const: $\frac{1}{\phi} = 0.13478$

golden ratio

3) Not Using (Protects You from Malicious attacks)

- Select C at runtime

METHOD #2 FOR COLLISION RESOLUTION

- By Open Addressing (3 variations)
  - On collision

17

B collision

A → hash func → 17

B → hash func → 17

C

= Method 1 Linear Probe

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

value, attempt

- Easy to code, suffers from ⊛ primary clustering
- Bad performance

- Method 2 Quadratic Probe

$h'(k,i) = (h(k) + c_1 * i + c_2 * (i*i)) \% m$

$c_1, c_2 \neq 0$, they are constants

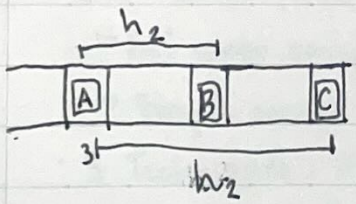
- Good choice of c_1, c_2, and m is difficult :/

Add array simulation later

- Method 3 Double Hashing
  - Avoids primary & secondary clustering

$h'(k,i) = (\underbrace{h(k)}_{\text{original hash func.}} + i \cdot \underbrace{h_2(k)}_{\text{second hash func., different c}}) \% m$

$i = \emptyset$



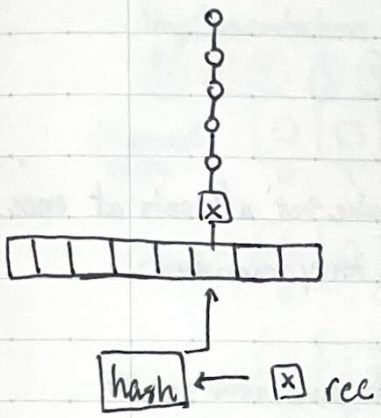
Date

Outline/Pseudo

[hash function (int)
(record) ← gets key value from record
↳ not required

record
<int, string>

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



<u>Search, Delete (record)</u>
index = hash(record)
loop through entries in A[index]

Time Complexity Matters
Performance Matters
↳ Do a fast array delete