CS2100

Hashing & Comparisons
Recap

* HW due next Wed
* Next Week: graphs
* Review session: last day of class

* Final worksheet (not graded)

* Exam: Wednesday, May 9 at 8am

* Today: in from noon-2 or 3-4 pm

- HW4 is done (check b-b or get)
Hashing
Problem: Data Storage

Example

<table>
<thead>
<tr>
<th>Locker#</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Dan</td>
</tr>
<tr>
<td>355</td>
<td>Kevin</td>
</tr>
<tr>
<td>101</td>
<td>Nitish</td>
</tr>
<tr>
<td>201</td>
<td>David</td>
</tr>
<tr>
<td>56</td>
<td>Erin</td>
</tr>
</tbody>
</table>

Goal: Given a locker #, want to be able to retrieve the name quickly.

Let \( n = \# \) of people
Let \( m = \# \) of lockers

\( m \) is much bigger
Could store using:

1. Vector/array:
   - Size is: $O(m)$  
   - Lookup: \(\geq O(1)\)
   - Insert/Remove: \(O(1)\)
   - \(X\)

2. List:
   - Size: \(O(n)\)
   - Lookup: \(O(n)\)
   - Insert/Remove: \(X\)

3. BST:
   - \(O(\log n)\)
Other examples:

- Course # & schedule info
- URL and html page
- Flight # & arrival info
- Color and BMP
- Directors & movies

Python Course

Takeaway:

- Not always clear how to get to vector indices!
- Unwilling to tolerate list penalties

$m^m n$ : Goal: $O(n)$ space $O(1)$ lookup
Dictionary
A data structure which supports:
- insert (key, data)
- find (key) → return data
- remove (key)

Note: An array is a kind of dictionary!
key: index or position
data: whatever is stored in it
Hashing

Assume $m \gg n$, so array takes too much space.

Goal: $O(n)$ space, fast lookup/insert/remove $O(1)$

A hash function $h$ maps each key to an integer in range $[0..N-1]$

Goal: $N$ is bigger than $n$, but much smaller than $m$.

Then: Given $(k,e)$, store it in $A[h(k)]$ (in an array).
Good hash functions:

- are fast: $O(1)$
- avoid collisions (a deal w/ them if they happen)

Key space $m \Rightarrow n$

$h(k) \Rightarrow O(1)$

$\{ (k_1, e_1), (k_2, e_2), \ldots \}$

$h(k_1) = h(k_2) = 3$
So, how to do this?

1. Make the key a # (binary data)
2. Compress # to $[0, \ldots, N-1]$
3. Handle collisions

1 + 2: often combined,

taught some of it last week

We'll recap a bit...
First idea
For something like ASCII, can break into pieces & treat as bits:

69 + \text{114} \times 10^5 \equiv 110 = \text{32-bit #}

Then what?
Problem: this can backfire w/ words:

\[ h(\text{temp01}) = h(\text{temp10}) = \text{pm0te1} \]

Want to avoid collisions.

So...
Polynomial Hash Codes

Split date to 32-bit pieces.

\[ X = (x_0, \ldots, x_{k-1}) \]

Pick \( a \neq 1 \). upper 32-bit pieces (\( k \) of them)

Let \( p(x) = x_0 a^{k-1} + x_1 a^{k-2} + \ldots + x_{k-2} a + x_{k-1} \)

Ex: Erin (or 69, 105, 114, 110)
and \( a = 37 \):

\[ p(x) = 69 \cdot 37^3 + 105 \cdot 37^2 + 114 \cdot 37 + 110 \]

\[ p("Erin") \neq p("Eirn") \]

Why?
- relatively fast
- avoids collisions (ones that result from permuting)
Next: Compress:

\[ h(k) \mod N \]

Idea: Take \( h(k) \mod N \) % in C++

Recall: 3 mod 10 = 3
80 mod 10 = 0
14 mod 10 = 4

(remainder)
Example: \[ h(k) = k \mod \frac{11}{N} \]

A:

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
\]

Insert: for key \( h(12) = \)

\((12, E) : 12 \mod 11 = 1\)

\((21, R) : h(21) = 10\)

\((37, I) : h(37) = 4\)

\((16, N) : h(16) = 5\)

\((26, C) : h(26) = 4 \times\)

\((5, +) \)

Comment: Works best if \( x \) is prime.

Why? Go take number theory or crypto.
Another way: M.A.D. (multiply, add, divide)
Instead of \( h(k) \mod N \),
do \( h(k) = |ak+b| \mod N \)
where \( a \) and \( b \) are:
- relatively prime
- less than \( N \)

Why?
- go take \( N \) or crypto
Example: \( h(k) = 3k + 5 \mod 11 \)

A: \[ 0 1 2 3 4 5 6 7 8 9 10 \]

Insert:
\[
(12, E) \quad h(12) = 3 \cdot 12 + 5 \mod 11 = 8
\]
\[
(21, R)
\]
\[
(37, I)
\]
\[
(16, N)
\]
\[
(26, C)
\]
\[
(5, H)
\]

Why bother?

**Much** better in practice
Step 3: Handle Collisions
(Hint: What data structures can store more than 1 thing?)

Ex: Simple Chaining:

```
      A
     / |
   0  1 |
   2  3  |
   4  5  6
   7  8  |
   9  10 |
```

Run times:
- hash: $O(1)$
- collision time: $O(n)$ or $O(\log n)$ worst case

Use a list for linear search.
Another idea:

Linear probing: If we hash to a "full" spot, just walk down to open one.

Issues:
- remove?

- Need to mark as removed, but can't actually free up the space.

Run time:
- Worst: $O(n)$
- Average: $O(1)$ "expected"
Example: \( h(k) = k \mod 11 \)

\[
A = \begin{bmatrix}
E & I & C & H & R \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10
\end{bmatrix}
\]

Insert:

\[
(12, E) : 12 \mod 11 = 1
\]

\[
(21, R) \quad h(21) = 10
\]

\[
(37, I) \quad h(37) = 4
\]

\[
(16, N) \quad h(16) = 5
\]

\[
(26, C) \quad h(26) = 4 \leftarrow
\]

\[
(5, H)
\]

Remove \((N)\) & can’t! mark it as “dirty”
Quadratic probing:

linear probing checks
\[ A[h(k)+j \mod N] \]
for \( j = 1, 2, 3, \ldots \)

Quadratic:
check \[ A[h(k)+j^2 \mod N] \]
where \( j = 1, 2, \ldots \)

Why?
- Avoids these "primary clusters"
- Still fast
Example: $h(k) = k \mod 11$

$A: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$

Insert:

$(12, E)$
$(21, R)$
$(37, I)$
$(16, N)$
$(26, C)$
$(5, H)$
Load Factors

Whatever method you use, usually starts to do badly if $n$ gets close to $N$:

$$\frac{n}{N} < 0.5$$

Rehashing:

When more than half full, most implementations double the array size (still: not too bad — think vectors + our amortized analysis.)
Next time:

- Look at how all these do in practice
- Intro to graphs