Abstract Data Type: Set of strings []\(\text{say} T\)

Ops: (*Initialize*)
- Insert(s)  // s a string
- IsMember(s)  // true/false,

Do example, human alg., requesting
Write on slips.
Store. First unordered. Then ordered....
Linked-list? BST...? Could get into those...?

Binary Search Tree: \(O(\log n)\) time ops if \(n\) elts.

(kind of complicated, Can we exploit \(O(1)\)-time array lookup?)

If you know all elts \(s\) are 3-digit #s...

alloc a Boolean array of len. 1000!

But what if they're 10-digit nums? 10B-size array ?

280-char tweets ?

Idea: **Hashing:** invent "hash" fn: \(h: \text{all strings} \Rightarrow \{0,1,\ldots,n-1\}\)

- Initialize: \(T[0\ldots n-1] := \text{false}\)
- Insert(s): \(T[h(s)] := \text{true}\)
- IsMember(s): return \(T[h(s)]\)

Issues!
Issue 1: Where/how to get \( h \)?

Issue 2: Collisions: \#strings = \( \infty \). Even if len. bounded, probably \#strings \( \gg n \).

So in PHP multiple strings \( s_1, s_2 \) have \( h(s_1) = h(s_2) \) ⇒ erroneous lookups, potentially

Solution 2: "Chaining" or other possibs...

\( T[.] \) stores linked lists, not just true/false.

\( T[i:] \) := null

Init \( \rightarrow \)

Insert \( s \) := insert "s" at end of list \( \rightarrow T[h(s)] \)

IsMember \( \rightarrow \)

Look thru \( (\text{list of } T[h(s)] \text{ for } s) \)

Bonus: can store any data/val \( v \) together with "key" \( s \), get a "dictionary" like

\( \text{MyDict}[s] = v \).

Space (mem.) usage: \( O(n \leq \text{len. of } T[.] \text{, you choose} \)

\( + \text{size of item, but say } O(1) \)

Worst-case time

for insert/lookup(s): \( O(\text{length of } T[h(s)]) \)

\( \# \text{items inserted} \)

\( \text{BST has } O(\log n) \)

→ could set \( n = 1 \), \( h(s) = 0 \) \( \forall s \rightarrow \) just a linked list!
Idea: Suppose \( h(s) \) was somehow a "random-looking" \# from 0...n-1 (but actually deterministic)

\[ \rightarrow \text{e.g.} \quad \text{\text{ascii int. represent of } s \mod n} \quad ??? \]

Hope: after inserting \( S_1, ..., S_n \),
the \#s \( h(S_1), ..., h(S_n) \) are "random-ish",
so "average" len. of a list is \( \frac{m}{n} \).

\[ \Rightarrow \text{"average-case" insert/lookup } = O(1 + \frac{m}{n}) \quad \text{curr #elts} \]

\[ \Rightarrow \text{If you estimate set will have } \approx m \quad \text{elts,} \]
\[ \text{can choose } n = m, \quad \text{have space } O(m) \]
\[ \text{"avg-case time" } O(1) \]

\[ \text{This is the key idea. But many q's to answer...} \]

- \( h(s) = \text{int}(s) \mod n \) is bad... // strings ending w/ same few chars map to same thing...
- how to make a "random-ish" \( h \)?
- what is "avg-case" formally? any worst-case guarantees?
- what if you can't est. \( m \) in advance - or,
  what if you know \( m \) in advance, just want to do lookups?
- methods other than chaining to handle collisions?
Random-ish hash fun? Let's think on this first?

1. Dream scenario: Some oracle does...
   for s in all strings (!!?)
   \[ h(s) := \text{RandInt}(0 \ldots n-1) \]
   And then you can "magically" use \( h \) in \( O(1) \) time.
   
2. Reality attempt, python:
   \[ h = \text{ord}(s[0]) \ll 7 \]
   for char in 
   \[ h = \text{E}_{-mul}(1000 \times 03, h) \lor \text{ord(char)} \]
   \[ h = (h \lor \text{len(s)}) \mod n. \]

   // this is completely deterministic, but "seems sorta random".
   // But knowing python does this, an evil person could keep inserting \( s \)s that hash to same #... !
   "Hash flooding" or "HashDoS"

3. Theorists:

   Today: "SUHA": Simple Uniform Hashing Assumption: do \( \Theta \), pretend it's \( \Theta \)
   "Good for analyzing hash tables in ideal cases."

   Later: Principled ways to provably achieve things achievable with SUHA.
   "Balls and Bins"
   [a very common scenario's]
"Average load / length": \[ \lambda = \frac{m}{n} \]

Say you now do an \underline{unsuccessful lookup}\(s\):
- \(s\) is not in table, \(h(s) \sim \text{RandInt}(0 \ldots n-1)\)
  - already

\[ E[\text{length of } T(h(s))] = \text{avg \{ bin sizes \} } = \lambda \quad \rightarrow \quad \text{Time } O(\lambda) \]

Say \underline{successful lookup} \(s\) is in table, say \(h(s) = c\)
- All \(m-1\) other items act like \text{rand balls}, so

\[ E[\text{length of } T(h(s))] = E[\# \text{ balls in bin } c \text{ after } m-1 \text{ throws}] = \frac{m-1}{n} \leq \frac{m}{n} = \lambda \quad \rightarrow \]

- Space = \(O(n)\), \(\lambda \leq \frac{n^2}{2} \) \(\leq \frac{n}{2} \)
- Time = \(O(1 + \lambda) = O(\lambda)\) on average

**M = n**: popular choice
- Space: \(O(n)\), \(O(1)\) on average

\(\lambda = 1\) but ... \(\exists\) very unlikely all balls in \(\text{diff bins}!!\)

- Probability first 2 balls go into \(T[c]\)

\[ \Pr [\text{first 2 balls go into } T[c]] = \frac{1}{n^2} \]

- Probability exactly \(k\) balls go into \(T[c]\)

\[ \Pr [\text{exactly } \lambda \text{ balls go into } T[c]] = \frac{\binom{n}{\lambda} \cdot \left(\frac{1}{n}\right)^\lambda \cdot \left(1 - \frac{1}{n}\right)^{n-\lambda}}{2^\lambda} \]

\[ \text{if } n = 1000 \]
Pr[exactly 10 go into T(0)] ≤ \frac{1}{10^6}

Pr[\geq 10 go into T(0)] = Pr[\text{\# 10 go in}^3 \cup \text{\# 11 go in}^3 \\
u \leq 12 \text{ go in}^3 u \ldots]

≤ Pr[\text{\# 10 go in}^3] + Pr[\text{\# 11 go in}^3] + \ldots "Union Bound"

\leq \frac{1}{10!} + \frac{1}{11!} + \frac{1}{12!} + \frac{1}{13!} + \ldots

= \frac{1}{10!} \left( 1 + \frac{1}{11} + \frac{1}{11 \cdot 12} + \frac{1}{11 \cdot 12 \cdot 13} + \ldots \right)

≤ \frac{1}{10!} \left( 1 + \frac{1}{11} + \frac{1}{11^2} + \frac{1}{11^3} + \ldots \right)

= \frac{1}{10!} \left( \frac{1}{1 - \frac{1}{11}} \right) = \frac{1}{10!} \cdot \frac{11}{10}

Pr[\geq k \text{ go into T(0)}] \leq \frac{1}{k!} \cdot \frac{k}{k} \leq \frac{2}{k!}

Pr[\geq k \text{ in T(0)}] \cup \geq k \text{ in T(1)} \cup \ldots \cup \geq k \text{ in T(n-1)}

≤ Pr[\geq k \text{ in T(0)}] + \ldots + Pr[\geq k \text{ in T(n-1)}]

≤ \frac{2}{k!} + \frac{2}{k!} + \ldots + \frac{2}{k!}

\leq \frac{4}{2^k \cdot n} \rightarrow \frac{1}{2^k \cdot n} \rightarrow \text{if } a < 2^k \cdot n \leq k

How much is k!?

100! = 100 \cdot 99 \cdot 98 \cdot 97 \cdot \ldots \cdot 2 \cdot 1

\leq 100 \cdot 100 \cdot 100 \cdot 100 \cdot \ldots \cdot 100

\approx 100^{50} \approx 100^{50}

So k! ≥ \frac{k\sqrt{k}}{2} \cdot \log(k!) ≥ \frac{k}{2} \log(\frac{k}{2}) = \Omega(k \log k).

2^{2n} \leq 2^{\Omega(k \log k)}

\Rightarrow k = O\left( \log \left( \frac{n}{\log \log n} \right) \right).

Q: how strongly do we expect? 1/million

max load ≤ light a lot except w.p.