Hash Tables

$T[1 \ldots m]$ (m slots)

Hash function $h$:

- $h : \text{keys} \rightarrow \{1, \ldots, m\}$
- $h$ uniformly random
  (indep for different keys)
- $h$ is easy to compute
  (assume $O(1)$ time)

$k_1 \neq k_2$ and $h(k_1) = h(k_2)$

called collision.

Insert item with key $k$ into $T[h(k)]$.

Dealing with collisions:

1. open addressing:
   at most 1 item per slot. If new item collides with
   an existing item, find another slot
   for new item.

2. chaining — $T[i]$
   is a pointer to a linked list of
   items whose keys hash to $i$
   (i.e., $h(\text{key}) = i$).
   (any # of items in the same
   slot.)
**Load factor.**

Hash table with \( m \) slots and \( n \) items.

*Load factor is*

\[
\alpha = \frac{n}{m}
\]

(\# items per slot)

**Insertion time:**

key \( k \).

Insert:

1. Compute \( i = h(k) \).
2. Insert item into the \( T[i] \) list (at the front)

**Time = \( O(1) \)**

**Search time:**

Search for item with key \( k \):

1. Compute \( i := h(k) \).
2. Search list \( T[i] \) linearly for the item, comparing keys (all keys distinct — no duplicate key values)

Average search time is more important than worst case.
Unsuccessful search
(no item with key
k — what we’re
looking for — is
in T.)

Time (expected)

\[ T = \frac{1}{\beta} + \left( \text{average length of a list} \right) \]

\[ = 1 + \alpha \]

\[ = (1+\alpha) \]

Successful search

Expected access time conditioned on the event that the item
already exists.

(search for some item
from hash tab uniformly
at random)

Assume distinct keys
\( k_1, k_2, \ldots, k_n \)
have been
inserted in order into
an initially empty
hash tab with \( m \) slots.

For \( 1 \leq i, j \leq n \) let

\( X_{ij} \) be the event that

\( h(k_i) = h(k_j) \)
Pr \[ X_{ij} \] = \frac{1}{m} \text{ for } i \neq j.

(assume)

Let \( I[X_{ij}] \) be the indicator random var for \( X_{ij} \).

\[ E(I[X_{ij}]) = Pr[X_{ij}] = \frac{1}{m} \text{ for } i \neq j. \]

\[ E(\text{successful search time}) = E\left[ \frac{1}{n} \sum_{i=1}^{n} (\text{time to search for } k_i) \right] \]

\[ = E\left[ \frac{1}{n} \sum_{i=1}^{n} \left( 1 + \sum_{j=i+1}^{n} I[X_{ij}] \right) \right] \]

1 if \( k_j \) collides with \( k_i \), and

0 otherwise

\[ = \frac{1}{n} \sum_{i=1}^{n} \left( 1 + \sum_{j=i+1}^{n} \frac{1}{m} \right) \]

\[ = 1 + \frac{1}{n} \sum_{i=1}^{n-1} \frac{n-i}{m} \]
\[
= 1 + \frac{1}{n} \sum_{i=1}^{n} \frac{n-i}{m}
\]
\[
= 1 + \frac{1}{mn} \sum_{i=1}^{n} (n-i)
\]
\[
= 1 + \frac{1}{mn} \sum_{i=0}^{n-1} i
\]
\[
= 1 + \frac{n(n-1)}{2mn}
\]
\[
= 1 + \frac{n-1}{2m}
\]
\[
= 1 + \frac{n}{2m} - \frac{1}{2m}
\]
\[
= 1 + \frac{\alpha}{2} - \frac{\alpha}{2n}
\]

vs \(1 + \alpha\)

---

Binary Search Trees (BSTs)
Red-Black Trees (RB trees)

RB tree is a BST with height \(O(\log n)\)

Plain BST

min height is \(\lceil \log n \rceil\)
max height is \(n-1\)

insert, search, removal take \(\Theta(h)\), where \(h\) is the height of the tree.
AVL trees — later
RB trees — first

I don’t allow duplicate keys.

Defining RB trees:
In a red-black tree, each node has a color attribute with 2 possible values red & black.

Any null pointer in a BST will instead point to a “phantom” leaf node in a RB tree:
dummy node with no data except its color which is black.

```
    20
   /  \
  18   30
    /    /  \
   15   25   35
```

all leaves are dummy black nodes. Implementation!
A BST satisfying these conditions is a **RB tree**.

**Thm:** Any RB tree with $n$ nodes has height $O(\log n)$.  


t, b, n

All leaves are black, contain no data.
Internal nodes are the normal, data-bearing nodes.

Additional rules:
1. The root is black.
2. If a node is red, then both children are black.
3. From any node $p$, the number of black nodes on any path from $p$ to a leaf is the same, independent of the path.
Def: Let T be a RB tree, p node in T.
The black height of p (bh(p)) is the number of black nodes along some (any) path from p to a leaf, (including the leaf but excluding p).

Lemma: Let T be a RB tree with n nodes and height h. Then

\[ h \leq 2 \log(n+1). \]

Proof: Prove by induction on the height of a node

\[ x : \text{subtree rooted at } x \text{ has size } \geq 2^{bh(x)} - 1 \]

x has height 0:
x is a leaf, bh(x) = 0
Size of subtree is 1.
1 \geq 2^0 - 1 = 0 \checkmark
Assume height of x is >0, so x has two children y & z.
\(y \& z\) have
\(bh\) at least
\(bh(x) - 1\).

height of \(y \& z\) is
\(<\) height of \(x\),
can assume
subtrees of \(y + z\)
have at least
\[
2 \left( \frac{bh(x) - 1}{2} \right) - 1
\]

many nodes
size of subtree of \(x\)
\[
= 1 + (\text{size of subt. } y)
+ (\text{size of subt. } z)
\geq 1 + (2^{\frac{bh(x) - 1}{2}} - 1)
+ (2^{\frac{bh(x) - 1}{2}} - 1)
= 2^{bh(x)} - 1
\]