Why should we consider another indexing technique when B-trees are so great?

• To avoid the multi-level index structure on disk.
• To get a small constant search time for equality queries to databases.
• To provide another structure that is useful for internal memory look-up tables.

Given:
1. a relatively large block of storage called the hash table
2. an attribute or key

Possible Goals:
1. Insert: store the key and its value in the table.
2. Find: find the key in the table and return its value.
3. Delete: remove the key and its value from the table.

Hash Function:
A hash function maps keys to ‘random’ addresses within the hash table.

Example:
Let the hash table be N locations long.
Suppose the keys are integers or can somehow be converted to integers.

\[ f(key) = key \mod N \]

is the most common, simple hash function.

NOTE: in hashing, the potential number of possible keys is much greater than the number of keys in use at any given time.
**DS.H.7**

**Character String Keys:**

- $h_4(key) = \text{char\_sum}(key) \mod N$
  - **Add together the byte representations of each of the characters and normalize to table size.**

- $h_5(key) = \text{extract}(P, Q, \text{char\_product}(key))$
  - **Multiply together the byte representations of each of the characters and extract Q bits starting at bit P to generate addresses in the correct range.**

- $h_6(key) = \sum_{i=0}^{\text{key size} - 1} key[i] \times 32^i \mod \text{table size}$
  - **(book’s Fig 5.5 is wrong)**

---

**DS.H.8**

**Separate Chaining**

Separate chaining is a collision strategy that uses linked lists to solve the collision problem.

- A “bin” of the hash table merely points to a linked list that hold all keys that hash to that bin.

**Example:**

- Bin 25 holds 3 different keys.

---

**DS.H.9**

**Open Addressing**

Open addressing uses one big contiguous hash table.

When there is a collision, it tries alternate locations, using the function:

$$h_i(x) = (h_0(x) + F(i)) \mod N$$

- It first tries $h_0(x)$, then $h_1(x)$, etc. until it finds the key in the table or comes to an empty cell to put it in.

- $h_0(x) = 25$
- $h_1(x) = 45$
- $h_2(x) = 35$

So how do we define $F(i)$?

---

**DS.H.10**

There are several common ways:

- **Linear Probing**
  - $F(i)$ is a linear function of $i$
  - $F(i) = i$ is most common.
  - It tries the next position for each probe.
  - $h_0(x) = h(x)$
  - $h_1(x) = h(x) + 1$
  - $h_2(x) = h(x) + 2$

- This is simple, but has the problem of primary clustering.

- Clusters develop in the table and most keys lead to some search.

**Theorem:**

If quadratic probing is used and the table size is prime, then a new element can always be inserted if the table is at least half empty. (proof by contradiction is readable)

---

**DS.H.11**

**2. Quadratic Probing**

- $F(i)$ is a quadratic function of $i$
  - $F(i) = i^2$

  - This spreads out the probes more.
  - $h_0(x) = h(x)$
  - $h_1(x) = h(x) + 1$
  - $h_2(x) = h(x) + 4$

  - This works better, but some secondary clustering effects have been reported.

---

**DS.H.12**

2. **Double hashing**

- Use a second hash function $h_2(x)$
  - $h(x) = h_1(x) + i \times h_2(x)$

  - This spreads out the probes more.

**Variant:** Use a sequence of hash functions

$$h(i) = h(i) \mod N$$

**Advantages of Open Addressing:**

- Insertion and deletion easy, but it requires pointers, which isn’t so good on disk.

---
Rehashing
/* If the table is getting too full, rebuild it with twice the space. Do this infrequently and at night. */

<table>
<thead>
<tr>
<th>Name</th>
<th>Bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Louis Smith</td>
<td>0</td>
</tr>
<tr>
<td>John Smith</td>
<td>1</td>
</tr>
<tr>
<td>Kate Green</td>
<td>2</td>
</tr>
<tr>
<td>Ray Finch</td>
<td>3</td>
</tr>
<tr>
<td>Craig Mir</td>
<td>4</td>
</tr>
<tr>
<td>John Smith</td>
<td>5</td>
</tr>
<tr>
<td>Ray Finch</td>
<td>6</td>
</tr>
</tbody>
</table>

Why is 45 in bin 1 when 45 % 5 = 0?

Complexity Analysis
Let \( N \) be the number of entries in the table at the current time.
Let \( T \) be the table size.
Let \( \lambda = NT \) be the load factor.

Chaining:
\( N \) can be larger or smaller than \( T \).
* e.g. We can have 10 lists of 3 elements each.
1. What is the longest any list can be?
2. What is the shortest any list can be?
3. What is the average length of a list?

Insertion time: \( O(1) \)

Open Addressing:
\( N \leq T, \lambda < 1 \) (full is BAD)

 Linear Probing:
What is the average number of cells probed in a successful search?
\( \lambda = \text{percentage of full cells.} \)
\( 1 - \lambda \) is the percentage of empty cells.
\( \lambda (1 - \lambda) \) is the number of cells searched before an empty one is found.
Maximum probes is \( 1 + \lambda (1 - \lambda) \).
Average probes is \( \frac{1}{2} (1 + \lambda (1 - \lambda)) \).

Extendible Hashing
Extendible hashing is a fast access method for dynamic files.
For data on disk, we don’t want to chase pointers.
Suppose that \( m \) (key, data) pairs fit in one disk block and that the hash function returns a 32-bit string.
Keep a directory that is organized according to the leading 4 bits of the hash value.
- \( D \) changes dynamically as the table grows.
- Use only enough bits to distinguish blocks.
Suppose we add key 0010. It belongs in bucket 0, which is full. So we split it into buckets 00 and 01.

\[
\begin{array}{c|c|c}
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

In extensible hashing, the index grows as needed.

**Complexity:**

\[
N: \text{entries} \\
M: \text{block size}
\]

Expected number of leaves: \((NM) \log_{2^M} \]

Expected directory size: \(O(N^{1+1/M}) / M)

The bigger \(M\) is the better.

**Hashing Applications**

- in compilers: to store and access identifiers
- in databases: for fast equality queries
- in image analysis for storing large structures

- Region Adjacency Graph Construction
  - Large number of regions with only a small percentage active at one time.

- Geometric Hashing
  - Large number of (object, transform) pairs requiring lots of quick lookups.