Open Addressing  Linear Probing

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31

$m = 10 \sim 10$ slots in table

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>23</td>
<td>7</td>
</tr>
<tr>
<td>17</td>
<td>19</td>
</tr>
</tbody>
</table>
```

no room for 31, need a larger hash table

$h'(k) = k \mod 10$

$h(k, i) = (i + k \mod 10) \mod 10 = (i + h'(k)) \mod 10$

Don't want the load factor $\lambda = \frac{n}{m}$

to be close to 1

Typical solution is rehashing.

If the load factor become high,
create a larger hash table
and insert all the clues into it.
(if simple uniform hashing assumption is true
or if key map evenly to slots,
then
and we don't let load factor become too high
then operations (search, insert, delete) are $\Theta(1)$.\)
$m = 1000$

Section is full, has 100 slots, empty slot.

Under simple uniform hashing,

Prob. next key maps here is $10/1000 = 1/10$.

Prob. next key maps here is $(100 + 1)/1000$. 


Linear Probing

Given an ordinary hash function $h' : U \rightarrow \{0, 1, \ldots, m - 1\}$, which we refer to as an auxiliary hash function, the method of linear probing uses the hash function

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

for $i = 0, 1, \ldots, m - 1$. Given key $k$, we first probe $T[h'(k)]$, i.e., the slot given by the auxiliary hash function. We next probe slot $T[h'(k) + 1]$, and so on up to slot $T[m - 1]$. Then we wrap around to slots $T[0], T[1], \ldots$ until we finally probe slot $T[h'(k) - 1]$. Because the initial probe determines the entire probe sequence, there are only $m$ distinct probe sequences.

Linear probing is easy to implement, but it suffers from a problem known as primary clustering. Long runs of occupied slots build up, increasing the average search time. Clusters arise because an empty slot preceded by $i$ full slots gets filled next with probability $(i + 1)/m$. Long runs of occupied slots tend to get longer, and the average search time increases.

Assuming uniform simple uniform hashing

operations are $\Theta \left( \frac{1}{(1-x)^2} \right)$

<table>
<thead>
<tr>
<th>$x$</th>
<th>0</th>
<th>0.2</th>
<th>0.4</th>
<th>0.6</th>
<th>0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{(1-x)^2}$</td>
<td>1</td>
<td>0.64</td>
<td>0.36</td>
<td>0.16</td>
<td>0.04</td>
</tr>
</tbody>
</table>

keeps load factor around 0.5
Suppose $h'(k) = k \mod 1000$

id # from 1000 to 1100

\[ \{ \text{full ids 1000-1100} \} \]

id # from 2000 to 2100

id #
- 2000 to 2100 all far away from $h(i, 0)$
**Open Addressing Hash-Insert**

Dynamic Set 1
Dynamic Set 2
Direct Addressing 1
Direct Addressing 2
Hashing 1
Hashing 2
Chaining 1
Chaining 2
Chaining 3
Chaining 4
Hash Functions
Open Address 1
Open Address 2
Open Address 3
Open Address 4
Open Address 5

**HASH-INSERT** *(T, k)*

1. \( i = 0 \) // \( i = 0 \)
2. repeat // \( h(k, 0) \) thru \( h(k, i-1) \) are full
3. \( j = h(k, i) \)
4. if \( T[j] = \text{NIL} \) // \( h(k, i) \) is empty
5. \( T[j] = k \)
6. return \( j \)
7. else \( i = i + 1 \) // all slots are full
8. until \( i = m \)
9. error "hash table overflow"

---

CS 3343 Analysis of Algorithms

Hash Tables – 14
Open Addressing Hash-Search

**HASH-SEARCH**($T, k$)

1. $i = 0$
2. **repeat** $h(k, 0)$ then $h(k, i-1)$ are full
3. $j = h(k, i)$
4. **if** $T[j] == k$ **then** key is found at $h(k, i)$
5. **return** $j$
6. $i = i + 1$
7. **until** $T[j] == NIL$ or $i == m$ **then** key is not in the table
8. **return** NIL

be sure to use a special flag when deleting
Other Probing Techniques

**Quadratic probing** uses a hash function of the form

\[ h(k, i) = (h'(k) + c_1i + c_2i^2) \mod m \], \hspace{1cm} (11.5)

where \( h' \) is an auxiliary hash function, \( c_1 \) and \( c_2 \) are positive auxiliary constants.

Double hashing offers one of the best methods available for open addressing because the permutations produced have many of the characteristics of randomly chosen permutations. **Double hashing** uses a hash function of the form

\[ h(k, i) = (h_1(k) + ih_2(k)) \mod m \],

where both \( h_1 \) and \( h_2 \) are auxiliary hash functions.

\[ h_1(k) = k \mod m \]
\[ h_2(k) = k \mod (m-1) \]

\( m \) should be prime

**Example**

\[ m = 10 \]
\[ k = 3 \]
\[ h_1(k) = 1 \]
\[ h_2(k) = 4 \]

\[ \frac{1}{1-0.8} = 5 \]

Start at slot 1, then 5, 9, 3, ...