Hashing

- Table \((hd)\) with \(D\) (or TableSize) entries or \(b\) (buckets).
- Hash function \(f(x)\) maps keys to \(\{0, 1, 2, \ldots, D - 1\}\), i.e., the universe is partitioned into \(D\) regions by the hash function.
- In the ideal situation the objects to be represented are mapped (via the hash function) to different positions in the hash table.
- Therefore, initialization takes \(O(D)\), and insert, delete and membership can be done in \(O(1)\) time (assuming the ideal situation).
- Most common hash function \(f(k) = k \% D\). \(f(k)\) gives the home bucket.
Linear Open Addressing Hashing

- Example: \( D = 11 \).
- \( f(80) \rightarrow 3, f(40) \rightarrow 7, f(65) \rightarrow 10 \).

\[ \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
& 80 & & & & & & & & 65 \\
\end{array} \]

- If we try to insert 58 which maps to 3 also, there is a collision.
- An overflow occurs when there is no more space for the element.
- Where do we store it? Next available space (circular) [linear open addressing].
- In this case is inserted in position 4.

\[ \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
& 80 & 58 & & & & & 40 & & 65 \\
\end{array} \]
• Insert 24 (maps to 2) does not cause a collision.

0 1 2 3 4 5 6 7 8 9 10
 24 80 58 40 65

• Insert 35 (maps to 2) causes a collision. So it ends up in position 5.

0 1 2 3 4 5 6 7 8 9 10
 24 80 58 35 40 65

• Insert 98 (maps to 10) causes a collision and ends up in position 0.

0 1 2 3 4 5 6 7 8 9 10
 98 24 80 58 35 40 65
Search for $x$

- Begin at the home bucket till
  - You find the element ($x$ is in the table), or
  - An empty spot ($x$ is not in the table), or
  - Back at the home bucket ($x$ is not in the table)
Deletion

- Just erase the element will not work! (Like delete 80)
- Use a NeverUsed bit (and modify search and insert)

Performance (No Proofs for this part Discussed)

- Number of buckets is \( b = D \).
- \( \alpha = n/b \) is the load factor.
- Avg. Num. of buckets examined during unsuccessful searches \( U_n \sim \frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2}\right) \)
- Avg. Num. of buckets examined during successful searches \( S_n \sim \frac{1}{2} \left(1 + \frac{1}{1-\alpha}\right) \)

\( \alpha = 0.5 \) the \( U_n \) is 2.5 and \( S_n \) is 1.5.

\( \alpha = 0.9 \) the \( U_n \) is 50.5 and \( S_n \) is 5.5.

\( D \) should be a prime or have no prime factors less than 20.
Random Proving: Defn. & Analysis

- Overflow: Next bucket is found at random.
- Actually pseudo-random in order to be able to reproduce results.

Theorem 10.1 [Sa, Origin: Probability Theory]: Let $p$ be the probability that certain event occurs. The expected number of independent trials needed for that event to occur is $1/p$.

Coin flips (for H or T): 2, and

Die throw (for number in 1 - 6): 6.

- $\alpha = n/b$ is the load factor.
- Probability of an occupied bucket is $\alpha$.
- Probability that a bucket is empty is $1 - \alpha$.
- Unsuccessful search: Looks for an empty bucket. Using independent trials the expected number of buckets examined is:

$$U_n \approx \frac{1}{1-\alpha}$$
Random Proving: $S_n$

- Eqn for $S_n$ is derived from $U_n$.
- Elements in table are 1, 2, ..., $n$ (in the order inserted).
- When element $i$ is inserted an unsuccessful search is performed and the element is inserted.
- From above, the buckets searched were $\frac{1}{1 - \frac{i-1}{b}}$.
- Assuming the each element in the table is searched with equal probability, we know that ...

(Next Slide)
Random Proving: $S_n$ cont'

\[ S_n \approx \frac{1}{n} \sum_{i=1}^{n} \frac{1}{1 - \frac{i-1}{b}} \]

\[ = \frac{1}{n} \sum_{i=0}^{n-1} \frac{1}{1 - \frac{i}{b}} \]

\[ \approx \frac{1}{n} \int_{i=0}^{n-1} \frac{1}{1 - \frac{i}{b}} \, di \]

\[ \approx \frac{1}{n} \int_{i=0}^{n} \frac{1}{1 - \frac{i}{b}} \, di \]

\[ = -\frac{b}{n} \ln\left(1 - \frac{i}{b}\right) \bigg|_{0}^{n} \]

\[ = -\frac{1}{\alpha} \ln(1 - \alpha) \]
Linear v.s. Random Proving

- When $\alpha = 0.9$ $U_n = 50.5$ with linear proving, but only 10 when using random proving.

- The important thing is run-time rather than number of buckets searched. It take more time to generate a random number than to search a few buckets.

- The cache effect also comes to play in random proving as the places being searched may cause “caching and paging faults” (section 4.5 [Sa]).
Hashing with Chaining

- Bucket has a linked list of the keys that mapped to that bucket (inc. order)

0 -> 11 -> 33 -> 55 -> 66
1
2
3 -> 36 -> 69
4
5 -> 16 -> 49 -> 82

- plus infinity object at the end of the list simplifies the code (Actually only one object).

0 -> 11 -> 33 -> 55 -> 66 -> BIG
1 -> BIG
2 -> BIG
3 -> 36 -> 69 -> BIG
4 -> BIG
5 -> 16 -> 49 -> 82 -> BIG
Performance (No Proofs Discussed (See 10.5.4 in [Sa]))

- $\alpha = n/D$ is the load factor.
- Avg. Num. of nodes examined during successful searches $S_n \sim 1 + \frac{\alpha}{2}$
- Avg. Num. of nodes examined during unsuccessful searches $U_n \leq \alpha$, $\alpha < 1$

$U_n \approx \frac{\alpha(\alpha+3)}{2(\alpha+1)}$, $\alpha \geq 1$
Text Compression: LZW

- Compress string aaabbbbbbaabaaba, with \( \Sigma = \{a, b\} \)
- “a” is assigned code 0 and “b” is assigned code 1.
- Mapping is stored in table

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

- Beginning with the above dictionary, find the longest prefix, \( p \), of the un-encoded part of the input file that is in the dictionary and output its code.
- If there is a next character \( c \), in the input file then \( pc \) is assigned the next code and inserted in the dictionary.
Example

0 1
a b

a aabbbbbbaabaaba
we output 0 and add aa with code 2
0 1 2
a b aa

aa bbbbbbaabaaba
we output 2 and add aab with code 3
0 1 2 3
a b aa aab

b bbbbbbaabaaba
we output 1 and add bb with code 4
0 1 2 3 4
a b aa aab bb
bb   bbbaabaaba
we output 4 and add bbb with code 5
0 1 2 3 4 5
a  b  aa aab  bb bbb

bbb   aabaaba
we output 5 and add bbba with code 6
0 1 2 3 4 5 6
a  b  aa aab  bb bbb bbba

aab   aaba
we output 3 and add aaba with code 7
0 1 2 3 4 5 6 7
a  b  aa aab  bb bbb bbba aaba

aaba
we output 7.
0 1 2 3 4 5 6 7
a  b  aa aab  bb bbb bbba aaba
Actual Dictionary

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>0a</td>
<td>2b</td>
<td>1b</td>
<td>4b</td>
<td>5a</td>
<td>3a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>aa</td>
<td>aab</td>
<td>bb</td>
<td>bbb</td>
<td>bbba</td>
<td>aaba</td>
</tr>
</tbody>
</table>

- Output is 0214537

Representation of Code Table

- Codes are 4096
- Access via code number plus symbol
- Use hash table with chaining ($D = 4099$)
- Can use tries too (reqs. more space).
- Code table is not transmitted to decompressor, because it can be reconstructed from the output of the compressor.
Decompression

0  214537
0  1
a  b
The 0 outputs a
The code 2 is 0*

2  14537
The 2 implies that the fc (first character) is a
So code 2 is 0a and added to the table
0  1  2
a  b  0a
a  b  aa <-- extra line
The 2 outputs aa
The code 3 is 2*
The 1 implies that the fc is b
So code 3 is 2b and added to the table

0 1 2 3
a b 0a 2b
a b aa aab <-- extra line
The 1 outputs b
The code 4 is 1*

The 4 implies that the fc is b
So code 4 is 1b and added to the table

0 1 2 3 4
a b 0a 2b 1b
a b aa aab bb <-- extra line
The 4 outputs bb
The code 5 is 4*
5  37
The 5 implies that the fc is b
So code 5 is 4b and added to the table
0  1  2  3  4  5
a  b  0a  2b  1b  4b
a  b  aa  aab  bb  bbb  <-- extra line
The 5 outputs bbb
The code 6 is 5*

3  7
The 3 implies that the fc is a
So code 6 is 5a and added to the table
0  1  2  3  4  5  6
a  b  0a  2b  1b  4b  5a
a  b  aa  aab  bb  bbb  bbb a <-- extra line
The 3 outputs aab
The code 7 is 3*
7

The 7 implies that the fc is a

So code 7 is 3a and added to the table

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>0a</td>
<td>2b</td>
<td>1b</td>
<td>4b</td>
<td>5a</td>
<td>3a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>aa</td>
<td>aab</td>
<td>bb</td>
<td>bbb</td>
<td>bbba</td>
<td>aaba</td>
</tr>
</tbody>
</table>

The 7 outputs aaba

The code 8 is 7*

- Output is aaabbbbbbaabaaba

Representation of Code Table

- Codes are 4096
- Access via code number
- Use 1D array of size 4096
Universal Hashing

- If the hash function is fixed in advance, then one can choose \( n \) keys so that all keys hash into the same place. So worst case may occur.

- Universal Hashing: Choose hash function randomly (independently of the hash keys being stored). Good performance on average.

- Randomized algorithms behave differently in each execution, even when the input is the same.

- Probability of a bad hash function is low.
• $\mathcal{H}$: Finite collection of hash functions that map a given universe $U$ of keys into the range $\{0, 1, \ldots, m - 1\}$.

• $\mathcal{H}$ is said to be *universal* if for each pair of distinct keys $x, y \in U$, the number of hash functions $h \in \mathcal{H}$ for which $h(x) = h(y)$ is precisely $|\mathcal{H}|/m$. In other words, with a hash function randomly chosen from $\mathcal{H}$, the chance of a collision between $x$ and $y$ when $x \neq y$ is exactly $1/m$, which is exactly the chance of a collision if $h(x)$ and $h(y)$ are randomly chosen from the set $\{0, 1, \ldots, m - 1\}$. 
Expected Number of Collisions

• Theorem: If \( h \) is chosen from a universal collection of hash functions to map \( n \) keys to a table with \( m \) entries \( (n \leq m) \), the expected number of collisions involving a key \( x \) is less than one.

• Proof: Given \( y \) and \( z \), let \( C_{yz} \) be a random variable equal to 1 if \( h(y) = h(z) \) and 0 otherwise.

• By definition (of universal hashing)
  \( E[C_{yz}] = 1/m. \)

• Given \( x \), let \( C_x \) be the total number of collisions involving key \( x \) in hash table \( T \) of size \( m \) with \( n \) keys.

• \( E[C_x] = \sum_{y \in T, y \neq x} E[C_{xy}] = (n - 1)/m \)

• Since \( n \leq m \), we know \( E[C_x] < 1. \)
Universal class of hash Functions

- $m$ is prime
- Decompose key $x$ into $r + 1$ bytes
  \( x = [x_0, x_1, \ldots, x_r] \) such that the maximum value of a byte is less than $m$.
- Let \( a = [a_0, a_1, \ldots, a_r] \) denote a sequence of $r + 1$ elements chosen randomly from the set \( \{0, 1, \ldots, m - 1\} \).
- The hash function $h_a \in \mathcal{H}$ is defined as
  \[ h_a(x) = \sum_{i=0}^{r} a_i x_i \mod m. \]
- $\mathcal{H} = \cup_a \{h_a\}$. Which has $m^{r+1}$ members.
• $\mathcal{H}$ just defined is Universal.

• Let $x$ and $y$ be two distinct keys.

• Assume that $x_0 \neq y_0$. (Similar argument can be made in other cases).

• For fixed values of $a_1, a_2, \ldots, a_r$, there is exactly one value of $a_0$ that satisfies $h(x) = h(y)$ since $a_0$ is the solution of $a_0(x_0 - y_0) = -\sum_{i=1}^{r} a_i(x_i - y_i)(\text{mod } m)$. Because $m$ is a prime.

• Therefore each pair of keys $x$ and $y$ collide for exactly $m^r$ values of $a$.

• Since there are $m^{r+1}$ possible sequences $a$, the probability of collision is exactly $m^r / m^{r+1} = 1/m$.

• Therefore $\mathcal{H}$ is universal.
Suppose that $x_0 > y_0$ (other case is similar)

Let $m = 11$ and $x_0 - y_0$ is 5

\[
\begin{array}{cccccccccccc}
a_0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
a_0 \times (x_0 - y_0) & 0 & 5 & 10 & 15 & 20 & 25 & 30 & 35 & 40 & 45 & 50 \\
\text{mod 11} & 0 & 5 & 10 & 4 & 9 & 3 & 8 & 2 & 7 & 1 & 6 \\
\end{array}
\]

So probability of a conflict (previous theorem) is $1/m = 1/11$. 

$m$ is a prime
However if $m = 10$ and $x_0-y_0$ is 5

<table>
<thead>
<tr>
<th>$a_0$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0 \cdot (x_0-y_0)$</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
</tr>
<tr>
<td>mod 10</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

So probability of a conflict (previous theorem) is NOT $1/m = 1/10$. It is $1/2$.

That is why $m$ is selected as a prime.