AVL-Tree has $O(\log n)$ time for search, and seems the best.......

Can we improve it, such as constant searching time?

Hash table: a totally different data structure achieving expected constant searching time.
AVL-Tree has $O(\log n)$ time for search, and seems the best.......  
Can we improve it, such as constant searching time?
AVL-Tree has $O(\log n)$ time for search, and seems the best....... 

Can we improve it, such as constant searching time?

Google is much faster than $O(\log n)$. 
AVL-Tree has $O(\log n)$ time for search, and seems the best........

Can we improve it, such as constant searching time?

- Google is much faster than $O(\log n)$.

**Hash table:** a totally different data structure achieving expected constant searching time.
Suppose there are $n$ elements with each having a key value $x$. 
Suppose there are $n$ elements with each having a key value $x$.

Hash table $T$ contains an array storing $m$ pointers.
Hash Function

• Suppose there are \( n \) elements with each having a key value \( x \).
• Hash table \( T \) contains an array storing \( m \) pointers.
• Hash function \( h(x) \in \{0, 1, 2, \cdots, m - 1\} \).
Suppose there are $n$ elements with each having a key value $x$.

Hash table $T$ contains an array storing $m$ pointers.

Hash function $h(x) \in \{0, 1, 2, \cdots, m - 1\}$.

Two different kinds of Hashing: chaining and open address.
Some elements may have the same hash value, i.e., \( h(x) = h(y) \) but \( x \neq y \) (which is called “collision”).
Some elements may have the same hash value, i.e., $h(x)=h(y)$ but $x \neq y$ (which is called “collision”).

Thus each item in the hash table is a list for the elements having the same hash value.
Some elements may have the same hash value, i.e., $h(x)=h(y)$ but $x \neq y$ (which is called “collision”).

Thus each item in the hash table is a list for the elements having the same hash value.

**Simple Uniform Hashing Assumption:** $\forall x, k$, the probability that $h(x)=k$ is $\frac{1}{m}$. 

The expected length of a list is $\alpha = \frac{n}{m}$, called load factor.

The expected search time is $O(1+\alpha)$. 

Finding a good hash function is the key (needs very complicated proof by number theory and probabilistic analysis, and not covered by this course).

In practice, we often use $h(x)=x \mod p$, where $p$ is a prime number.
Some elements may have the same hash value, i.e., \( h(x) = h(y) \) but \( x \neq y \) (which is called “collision”).

Thus each item in the hash table is a list for the elements having the same hash value.

**Simple Uniform Hashing Assumption:** \( \forall x, k \), the probability that \( h(x) = k \) is \( \frac{1}{m} \).
- The expected length of a list is \( \alpha = \frac{n}{m} \), called load factor.
Some elements may have the same hash value, i.e., \( h(x) = h(y) \) but \( x \neq y \) (which is called “collision”).

Thus each item in the hash table is a list for the elements having the same hash value.

**Simple Uniform Hashing Assumption:** \( \forall x, k, \) the probability that \( h(x) = k \) is \( \frac{1}{m} \).
- The expected length of a list is \( \alpha = \frac{n}{m} \), called load factor.
- The expected search time is \( O(1 + \alpha) \).
Some elements may have the same hash value, i.e., $h(x)=h(y)$ but $x \neq y$ (which is called “collision”).

Thus each item in the hash table is a list for the elements having the same hash value.

**Simple Uniform Hashing Assumption:** $\forall x, k$, the probability that $h(x)=k$ is $\frac{1}{m}$.
- The expected length of a list is $\alpha = \frac{n}{m}$, called load factor.
- The expected search time is $O(1 + \alpha)$.

Finding a good hash function is the key (needs very complicated proof by number theory and probabilistic analysis, and not covered by this course).
Some elements may have the same hash value, i.e., $h(x) = h(y)$ but $x \neq y$ (which is called “collision”).

Thus each item in the hash table is a list for the elements having the same hash value.

**Simple Uniform Hashing Assumption:** $\forall x, k$, the probability that $h(x) = k$ is $\frac{1}{m}$.

- The expected length of a list is $\alpha = \frac{n}{m}$, called load factor.
- The expected search time is $O(1 + \alpha)$.

Finding a good hash function is the key (needs very complicated proof by number theory and probabilistic analysis, and not covered by this course).

- In practice, we often use $h(x) = x \mod p$, where $p$ is a prime number.
An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ...... 
Hash function: $h(x) = x \mod 7$. 

0 [ ] 
1 [ ] 
2 [ ] 
3 [ ] 
4 [ ] 
5 [ ] 
6 [ ]
Hash: Chaining

- An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ....
- Hash function: $h(x) = x \mod 7$. 

<table>
<thead>
<tr>
<th>0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
**Hash: Chaining**

- An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ......
- Hash function: \( h(x) = x \mod 7 \).
Hash: Chaining

- An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ......
- Hash function: $h(x) = x \mod 7$.
Hash: Chaining

- An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ....
- Hash function: $h(x) = x \mod 7$. 

```
0
1
2
3 10
4 11
5
6 6
```
Hash: Chaining

- An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ......
- Hash function: \( h(x) = x \mod 7 \).
**Hash: Chaining**

- An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ....
- Hash function: \( h(x) = x \mod 7 \).
Hash: Chaining

- An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ......
- Hash function: $h(x) = x \mod 7$.

![Hash Chaining Diagram]
Hash: Chaining

- An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ......
- Hash function: $h(x) = x \mod 7$. 

![Diagram showing chaining example](attachment:image.png)
An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ......

Hash function: $h(x) = x \mod 7$. 

![Diagram showing hash table entries and values]
Hash: Chaining

- An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ......
- Hash function: $h(x) = x \mod 7$. 

![Hash Chaining Diagram]
Hash: Chaining

- An example: suppose the stream of numbers are 10, 6, 11, 20, 21, 13, 25, 71, 15, 12, ....
- Hash function: $h(x) = x \mod 7$. 

![Diagram showing hash chaining with numbers mapped to different buckets]
Hash: Open Addressing

Each item in the hash table contains only one element. m must be larger than n.
Hash: Open Addressing

- Each item in the hash table contains only one element. $m$ must be larger than $n$.
- When having a collision, update the hash function and get a new hash value.
Hash: Open Addressing

- Each item in the hash table contains only one element. \( m \) must be larger than \( n \).
- When having a collision, update the hash function and get a new hash value.
- Hash function \( h(x, i) \), where \( i \) counts the number of trials from 0 to \( m-1 \). Two common ways:
  - **Linear Probing:** \( h(x, i) = h_1(x) + i \mod m \).
  - **Double Hashing:** \( h(x, i) = h_1(x) + i \times h_2(x) \mod m \), where \( h_2(x) \) and \( m \) should be relatively prime.
Each item in the hash table contains only one element. $m$ must be larger than $n$.

When having a collision, update the hash function and get a new hash value.

Hash function $h(x, i)$, where $i$ counts the number of trials from 0 to $m-1$. Two common ways:

- Linear Probing: $h(x, i) = h_1(x) + i \mod m$. 

- Double Hashing: $h(x, i) = h_1(x) + i \times h_2(x) \mod m$, where $h_2(x)$ and $m$ should be relatively prime.
Hash: Open Addressing

- Each item in the hash table contains only one element. $m$ must be larger than $n$.
- When having a collision, update the hash function and get a new hash value.
- Hash function $h(x, i)$, where $i$ counts the number of trials from 0 to $m-1$. Two common ways:
  - Linear Probing: $h(x, i) = h_1(x) + i \mod m$.
  - Double Hashing: $h(x, i) = h_1(x) + i \times h_2(x) \mod m$, where $h_2(x)$ and $m$ should be relatively prime.
Hash: Open Addressing

- Each item in the hash table contains only one element. \( m \) must be larger than \( n \).
- When having a collision, update the hash function and get a new hash value.
- Hash function \( h(x, i) \), where \( i \) counts the number of trials from 0 to \( m-1 \). Two common ways:
  - Linear Probing: \( h(x, i) = h_1(x) + i \mod m \).
  - Double Hashing: \( h(x, i) = h_1(x) + i \times h_2(x) \mod m \), where \( h_2(x) \) and \( m \) should be relatively prime.
- A good hash function could yield constant searching time (again, needs very complicated proof by number theory and probabilistic analysis, and not covered by this course).
Hash: Open Addressing

- An example: suppose the stream of numbers are 11, 32, 53, ......
- Hash function: \( h(x,i) = (x + i \times (x \mod 3)) \mod 7 \).
Hash: Open Addressing

- An example: suppose the stream of numbers are 11, 32, 53, ....
- Hash function: \( h(x,i) = (x + i \times (x \mod 3)) \mod 7 \).

\[
\begin{array}{c}
0 & \square \\
1 & \square \\
2 & \square \\
3 & \square \\
4 & 11 \\
5 & \square \\
6 & \square \\
\end{array}
\]
An example: suppose the stream of numbers are 11, 32, 53, ......  

Hash function: \( h(x,i) = (x + i \times (x \mod 3)) \mod 7 \).
Hash: Open Addressing

- An example: suppose the stream of numbers are 11, 32, 53, ......
- Hash function: \( h(x,i) = (x + i \times (x \mod 3)) \mod 7 \).
Hash: Open Addressing

- An example: suppose the stream of numbers are 11, 32, 53, ....
- Hash function: \( h(x,i) = (x + i \times (x \mod 3)) \mod 7 \).

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td></td>
<td>32</td>
</tr>
</tbody>
</table>
Hash: Open Addressing

- An example: suppose the stream of numbers are 11, 32, 53, ......
- Hash function: \( h(x,i) = (x + i \times (x \mod 3)) \mod 7 \).
An example: suppose the stream of numbers are 11, 32, 53, ...... 
Hash function: \( h(x,i) = (x + i \times (x \mod 3)) \mod 7 \).
An example: suppose the stream of numbers are 11, 32, 53, ....
Hash function: $h(x, i) = (x + i \times (x \mod 3)) \mod 7$. 

![Diagram showing hash function application]

0
1  53
2
3
4  11
5
6  32
An example: suppose the stream of numbers are 11, 32, 53, ....

Hash function: \( h(x,i) = (x + i \times (x \mod 3)) \mod 7 \).
Remember that hash table is **dynamic data structure** (similar to AVL-Tree).
Insertion And Deletion

- Remember that hash table is **dynamic data structure** (similar to AVL-Tree).
- When inserting $n > m$ elements, double the size $m$ to $2m$, and re-hash all the existing elements.
Insertion And Deletion

- Remember that hash table is **dynamic data structure** (similar to AVL-Tree).
- When inserting \( n > m \) elements, double the size \( m \) to \( 2m \), and re-hash all the existing elements.
- When deleting too many elements such that \( n \leq m/4 \), shrink \( m \) to \( m/2 \), and re-hash all the existing elements.

The amortized complexity for \( n \) insertions/deletions is \( \Theta(n) \).
Insertion And Deletion

- Remember that hash table is dynamic data structure (similar to AVL-Tree).
- When inserting $n > m$ elements, double the size $m$ to $2m$, and re-hash all the existing elements.
- When deleting too many elements such that $n \leq m/4$, shrink $m$ to $m/2$, and re-hash all the existing elements.
- The amortized complexity for n insertions/deletions is $\Theta(n)$. 

Note: for open addressing hash table, keep a "flag" when deleting an element for future search.
Insertion And Deletion

- Remember that hash table is **dynamic data structure** (similar to AVL-Tree).
- When inserting \( n > m \) elements, double the size \( m \) to \( 2m \), and re-hash all the existing elements.
- When deleting too many elements such that \( n \leq m/4 \), shrink \( m \) to \( m/2 \), and re-hash all the existing elements.
- The amortized complexity for \( n \) insertions/deletions is \( \Theta(n) \).
- **Note:** for open addressing hash table, keep a “flag” when deleting an element for future search.