Dictionary ADT

Stores a collection of records of the form key, data

Provides the following operations:
  get(key), put(key,data), remove(key)
Dictionary ADT

Stores a collection of records of the form key, data

Provides the following operations:

- get(key)
- put(key, data)
- remove(key)

A dictionary can be implemented with different data structures:

- get(key)
- put(key, data)
- remove(key)
Dictionary ADT

Stores a collection of records of the form key, data

Provides the following operations:
- `get(key)`, `put(key, data)`, `remove(key)`

A dictionary can be implemented with different data structures:

- **Linked list**
  - `get(key)` \(O(n)\)
  - `put(key, data)` \(O(n)\)
  - `remove(key)` \(O(n)\)
Dictionary ADT

Stores a collection of records of the form key, data

Provides the following operations:
- get(key)
- put(key, data)
- remove(key)

A dictionary can be implemented with different data structures:

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>get(key)</th>
<th>put(key, data)</th>
<th>remove(key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linked list</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Sorted array</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
The diagram illustrates a hash table with the following key points:

- The hash table is represented as a list with entries labeled as $K_k, D_k$ for $k = 1, 2, ..., n$.
- There is a hash function $h(key)$ that maps keys to the hash table entries.
- The hash function $h(key)$ is claimed to have $O(1)$ average running time for operations like get, put, and remove.
- The diagram shows that we can achieve $O(1)$ average running time by using a hash function that distributes keys uniformly across the hash table.

Mathematical notation:

- $M$: Number of entries in the hash table.
- $N$: Size of the universe from which keys are drawn.
- $M \geq N$: The condition for the table to be able to handle $N$ keys without collisions.

The diagram also includes a note: "we can achieve $O(1)$ average running time."
Set of all possible UWO ID's:
$$d_0 d_1 d_2 \ldots d_{10}$$

Since each $d_i$ can take 10 different values, this set has $10^9$ elements, which is much bigger than the size of the table, so the hash function will produce collisions.

**Inertia:**
*tins*: *produce collisions that must give a position for EVERY ID*
Design issues

- Determine size of the table
- Select hash function
Design issues

- Determine size of the table
- Select hash function
  - low time complexity
  - causes few collisions
Design issues

- Determine size of the table
- Select hash function
  - low time complexity
  - causes few collisions
    must spread keys uniformly
    across entire table
Design issues

• Determine size of the table
• Select hash function
  - low time complexity
  - causes few collisions
  must spread keys uniformly across entire table

• How to deal with collisions
Hash function

Keys can be of any type: integer, character, string, array, ...

Hash code

\[ g(key) \rightarrow \text{integer} \]

We want a large integer to ensure all entries of the hash table are used

Compression map

\[ f(integer) \rightarrow \text{position of hash table} \]

Hash function

\[ h(key) = f(g(key)) \]
Hash Code

Converting a string into an integer: First a character can be converted into an integer through casting:

\[(\text{int}) \text{char} \rightarrow \text{integer}\]

Then, apply some function to the integers corresponding to the characters of the string to produce a single integer. For example:

\[g("c_{k-1} c_{k-2} \ldots c_1 c_0") = \sum_{i=0}^{k-1} (\text{int})c_i\]

This is not a good hash code because it produces small integer values.
Polynomial Hash Code

\[ S = \text{"} C_{k-1} \ C_{k-2} \ \ldots \ C_1 \ C_0 \text{"} \]

(int)\(C_{k-1}\) (int)\(C_{k-2}\) (int)\(C_1\) (int)\(C_0\)
Polynomial Hash Code

\[ S = "C_{k-1} \ C_{k-2} \ldots\ C_1\ C_0" \]

\[ (\text{int})C_{k-1} \ (\text{int})C_{k-2} \ldots\ (\text{int})C_1\ (\text{int})C_0 \]

Use these numbers as the coefficients of a polynomial:

\[ g(S) = (\text{int})C_{k-1} \ x^{k-1} + (\text{int})C_{k-2} \ x^{k-2} + \ldots + (\text{int})C_1 \ x + (\text{int})C_0 \]

\[ x = 33, 37, 39, 41 \]  good choices for \( x \)
Hash Function

\[ g \left( C_{k-1}C_{k-2}\ldots C_1C_0 \right) = \left( \int \text{int} \right) C_{k-1} X^{k-1} + \left( \int \text{int} \right) C_{k-2} X^{k-2} + \ldots + \left( \int \text{int} \right) C_1 X^1 + \left( \int \text{int} \right) C_0 \]
Hash Function

\[ h(\text{"}C_{k-1}C_{k-2}\ldots C_1 C_0\text{"}) = (\text{\texttt{int}} C_{k-1} X^{k-1} + (\text{\texttt{int}} C_{k-2} X^{k-2} + \ldots + (\text{\texttt{int}} C_1 X^1 + (\text{\texttt{int}} C_0 \mod M) \mod M) \]
Hash Function

\[ h(\text{"Ck-1Ck-2\ldots C1 C0"}) = (\text{int} C_{k-1} x^{k-1}) + (\text{int} C_{k-2} x^{k-2}) + \ldots + (\text{int} C_1 x^1) + (\text{int} C_0) \mod M \]

= \left( (\ldots (((\text{int} C_{k-1}) x + (\text{int} C_{k-2}) x + \ldots + (\text{int} C_1) x + (\text{int} C_0) \mod M) \right)

Algorithm hashFunction(S)
In: String S = "Ck-1Ck-2\ldots C1 C0"
Out: position for S in hash table
\[ h(k) = k \mod 7 \]
$$h(k) = k \mod 7$$

<p>| $T$ | $14, d_1$ | $12, d_2$ | $13, d_3$ | $21, d_4$ | $19, d_5$ | $2, d_6$ |</p>
<table>
<thead>
<tr>
<th>( h(k) = k \mod 11 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0 )</td>
</tr>
<tr>
<td>( 1 )</td>
</tr>
<tr>
<td>( 2 )</td>
</tr>
<tr>
<td>( 3 )</td>
</tr>
<tr>
<td>( 4 )</td>
</tr>
<tr>
<td>( 5 )</td>
</tr>
<tr>
<td>( 6 )</td>
</tr>
<tr>
<td>( 7 )</td>
</tr>
<tr>
<td>( 8 )</td>
</tr>
<tr>
<td>( 9 )</td>
</tr>
<tr>
<td>( 10 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( h(k) = k \mod 11 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0 )</td>
</tr>
<tr>
<td>( 1 )</td>
</tr>
<tr>
<td>( 2 )</td>
</tr>
<tr>
<td>( 3 )</td>
</tr>
<tr>
<td>( 4 )</td>
</tr>
<tr>
<td>( 5 )</td>
</tr>
<tr>
<td>( 6 )</td>
</tr>
<tr>
<td>( 7 )</td>
</tr>
<tr>
<td>( 8 )</td>
</tr>
<tr>
<td>( 9 )</td>
</tr>
<tr>
<td>( 10 )</td>
</tr>
</tbody>
</table>