Symbol Tables

Symbol tables
Ordered array
Unordered linked list


Symbol Table ADT

Symbol table: key-value pair abstraction.

- Insert a value with specified key.
- Search for value given key.
- Delete value with given key.

Example: key = URL, value = IP address.

- Insert URL with specified IP address.
- Given URL, find corresponding IP address.

<table>
<thead>
<tr>
<th>Web Site</th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.cs.princeton.edu">www.cs.princeton.edu</a></td>
<td>128.112.136.11</td>
</tr>
<tr>
<td><a href="http://www.princeton.edu">www.princeton.edu</a></td>
<td>128.112.128.15</td>
</tr>
<tr>
<td><a href="http://www.yale.edu">www.yale.edu</a></td>
<td>130.132.143.21</td>
</tr>
<tr>
<td><a href="http://www.harvard.edu">www.harvard.edu</a></td>
<td>128.103.060.55</td>
</tr>
<tr>
<td><a href="http://www.simpsons.com">www.simpsons.com</a></td>
<td>209.052.165.60</td>
</tr>
</tbody>
</table>

key
value

Other Symbol Table Applications

Other applications.
- Online phone book: look up a name to find telephone number.
- Google: look up phrase and return most relevant web pages.
- Spell checker: look up a word to find if it’s there or present alternative.
- DNS: look up name of web site to find IP address.
- Java compiler: look up variable name to find its type and value.
- File sharer: look up song to find host machines.
- File system: look up file name to find location on hard drive.
- University registrar: look up student to find grades.
- Web traffic analyzer: look up host to find number of hits.
- Routing table: look up routing info for IP.
- Browser: highlight visited links in purple.
- Chess: detect a repetition draw.
- Bayesian spam filter: use frequencies of spam and ham words to filter email.
- Language modeling: determine frequency of next letter given prefix.
- Book index: determine pages on which each word appears.
- “Associative memory.”
- Index of any kind.

Abstract Data Types

Interface. Description of data type, basic ops.
Client. Program using ops defined in interface.
Implementation. Actual code implementing ops.
Symbol Table Client: DNS Lookup

DNS lookup client program.
- `st.put(key, value)` inserts a key-value pair into symbol table.
- `st.get(key)` searches for the given key and returns the value.

```
public static void main(String[] args) {
    SymbolTable st = new SymbolTable();
    st.put("www.cs.princeton.edu", "128.112.136.11");
    st.put("www.princeton.edu", "128.112.128.15");
    st.put("www.yale.edu", "130.132.143.21");
    st.put("www.simpsons.com", "209.052.165.60");
    System.out.println(st.get("www.cs.princeton.edu"));
    System.out.println(st.get("www.harvardsucks.com"));
    System.out.println(st.get("www.simpsons.com"));
}
```

Symbol Table Client: Remove Duplicates

Remove duplicates (e.g., from commercial mailing list).
- Read in a key.
- If key is not in symbol table, print out key and insert.

```
public class DeDup {
    public static void main(String[] args) {
        SymbolTable st = new SymbolTable();
        while (!StdIn.isEmpty() {
            String key = StdIn.readString();
            if (st.get(key) == null) {
                System.out.println(key);
                st.put(key, "");
            }
        }
    }
}
```

## Object

**Class** `Object`.
- All objects "inherit" from the special class `Object`.
- All objects have certain pre-defined methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Default</th>
<th>Typical Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>toString</td>
<td>convert to string</td>
<td>memory address</td>
<td>&quot;hello &quot; + s</td>
</tr>
<tr>
<td>equals</td>
<td>are two objects equal?</td>
<td>are two memory addresses equal?</td>
<td>if (s.equals(t))</td>
</tr>
<tr>
<td>hashCode</td>
<td>convert to integer</td>
<td>memory address</td>
<td>s.hashCode()</td>
</tr>
</tbody>
</table>

**Consequences.**
- Can have a symbol table of any object, e.g., `String` or `Student`.
- Programmer may need to override default methods.

Symbol Table: Sorted Array Implementation

Maintain array of keys and values.
- Store in sorted order by key.
- `keys[i]` = i\(^{th}\) largest key.
- `values[i]` = value associated with i\(^{th}\) largest key.

```
public class ST {
    private Object[] values = new Object[14];
    private String[] keys = new Object[14];
    private int N = 0;

    public void put(String key, Object value) {
        keys[N] = key;
        values[N] = value;
        N++;
    }
}
```
Symbol Table Search: Sorted Array Implementation

Binary search.
- Examine the middle key.
- If it matches, return the value.
- Otherwise, search either the left or right half.

```java
public Object get(String key) {
    int left = 0;
    int right = N-1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (equal(key, keys[mid])) return values[mid]; // found
        if (less (key, keys[mid])) right = mid - 1; // left half
        else left = mid + 1; // right half
    }
    return null; // not found
}
```

Symbol Table Insert: Sorted Array Implementation

Insert.
- Need to maintain entries in ascending order.
- Find insertion point and move larger keys to the right.

Sorted Array Implementation Analysis

Claim. Worst-case number comparisons to binary search in a sorted array of size N is $O(\log N)$?
- Divide list in half each time.

```
% java DeDup < toSpamList.txt
wayne@cs.princeton.edu
dgabal@cs.princeton.edu
pcalamia@cs.princeton.edu
mgsw@cs.princeton.edu

% java Dedup < mobydick.txt
moby
dick
herman
melville
call
me
in
hme
some
car
some
years
ago
...
```

Advantages: not much code, fast search.

Disadvantage: insert is hopelessly slow for large inputs.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$\log N$</td>
<td>N</td>
<td>N</td>
<td>$\log N$</td>
<td>$N / 2$</td>
<td>$N / 2$</td>
</tr>
</tbody>
</table>
Symbol Table: Linked List Implementation

Maintain a linked list of key-value pairs.
- Insert new key-value pair at beginning of list.
- Key = String, value = Object.
- Use exhaustive search to search for a key.

Linked List Implementation: Performance

Advantages: not much code, fast insertion.

Disadvantage: search is hopelessly slow for large inputs.

```
% java DeDup < toSpamList.txt
wayne@cs.princeton.edu
rs@cs.princeton.edu
dgabai@cs.princeton.edu
pcalamia@cs.princeton.edu
mgsw@cs.princeton.edu
```

```
% java Dedup < mobydick.txt
moby
dick
herman
melville
call
me
inhmzel
some
years
ago
...
```

210,028 words
16,834 distinct

Linked List Implementation: Analysis

Insertion. Constant time.

Search. Need to look at every entry if not found.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>log N</td>
<td>N</td>
<td>N</td>
<td>log N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Unsorted list</td>
<td>N</td>
<td>1</td>
<td>1</td>
<td>N / 2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Can we achieve log N performance for all ops?