24: Symbol Tables and Binary Search
Lecture 24: Symbol Tables and Binary Search

- Symbol Tables
- Binary search
- Elementary Implementations of Symbol Tables
- Ordered Operations

Some slides adopted from Algorithms 4th Edition or COS226
Printed symbol tables are all around us

- **Dictionary**: key = word, value = definition.
- **Encyclopedia**: key = term, value = article.
- **Phonebook**: key = name, value = phone number.
- **Math table**: key = math functions and input, value = function output.

**Unsupported operations:**

- Add a new key and associated value.
- Remove a given key and associated value.
- Change value associated with a given key.
Symbol tables

- Key-value pair abstractions.
  - Insert a value with a specific key.
  - Given a key, search for the corresponding value.
- Also known as: maps, dictionaries, associative arrays.
- Generalize arrays: keys not be integers between 0 and $n - 1$.
- Supported either with built-in or external libraries by the majority of programming languages.
Basic symbol table API

- `public class ST <Key extends Comparable<Key>, Value>`
  - `ST()`: create an empty symbol table. By convention, values are not null.
  - `void put(Key key, Value val)`: insert key-value pair.
    - Overwrites old value with new value if key already exists.
  - `Value get(Key key)`: return value associated with key.
    - Returns null if key not present.
  - `boolean contains(Key key)`: is there a value associated with key.
  - `Iterable keys()`: all the keys in the symbol table.
  - `void delete(Key key)`: delete key and associated value.
  - `boolean isEmpty()`: is the symbol table empty?
  - `int size()`: number of key-value pairs.
Lecture 24: Symbol Tables and Binary Search

- Symbol Tables
- Binary search
- Elementary Implementations of Symbol Tables
- Ordered Operations
Binary search

- **Goal**: Given a sorted array and a key, find index of the key in the array.

- Basic mechanism: Compare key against middle entry.
  - If too small, repeat in left half.
  - If too large, repeat in right half.
  - If equal, you are done.
Binary search implementation

- First binary search published in 1946.
- First bug-free one in 1962.

```java
public static int binarySearch(int[] a, int key) {
    int lo = 0, hi = a.length-1;
    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (key < a[mid])
            hi = mid - 1;
        else if (key > a[mid])
            lo = mid + 1;
        else return mid; }
    return -1;
}
```
- Uses at most $1 + \log n$ key compares to search in a sorted array of size $n$, that is it is $O(\log n)$. 
Lecture 24: Symbol Tables and Binary Search

- Symbol Tables
- Binary search
- Elementary Implementations of Symbol Tables
- Ordered Operations
Sequential search in a linked list

- **Data structure**: Maintain an unordered linked list of key-value pairs.
- **Search**: Scan through all the keys until you find a match.
- **Insert**: Scan through all the keys until you find a match. If you found it, update value, otherwise, add to front of list.
- If our cost model counts how many times we will compare keys, both search and insert are $O(n)$ both for worst and average case.
Sequential search in a linked list
Binary search in an ordered array

- **Data structure**: Maintain parallel arrays for keys and values, sorted by keys.

- **Search**: Use binary search to find key.
  - At most $O(\log n)$ compares to search a sorted array of length $n$.

- **Insert**: Use binary search to find key. If it does not exist, shift all larger keys over.
  - At most $O(n)$ time.
Binary search in an ordered array
Binary search in an ordered array

```
<table>
<thead>
<tr>
<th></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>lo</td>
<td>hi</td>
<td>mid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**successful search for P**

- entries in black are a[lo..hi]
- entry in red is a[mid]

```
<table>
<thead>
<tr>
<th></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>lo</td>
<td>hi</td>
<td>mid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**unsuccessful search for Q**

```
<table>
<thead>
<tr>
<th></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>lo</td>
<td>hi</td>
<td>mid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- loop exits with keys[mid] = P: return
- loop exits with lo > hi: return 7
Lecture 24: Symbol Tables and Binary Search

- Symbol Tables
- Binary search
- Elementary Implementations of Symbol Tables
- Ordered Operations
Examples of ordered operations in a symbol table

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>min(09:00:00)</td>
<td>Chicago</td>
</tr>
<tr>
<td>09:00:03</td>
<td>Phoenix</td>
</tr>
<tr>
<td>09:00:13</td>
<td>Houston</td>
</tr>
</tbody>
</table>

get(09:00:13) 09:00:59 Chicago
09:01:10 Houston

floor(09:05:00) 09:03:13 Chicago
09:10:11 Seattle

select(7) 09:10:25 Seattle
09:14:25 Phoenix
09:19:32 Chicago
09:19:46 Chicago

keys(09:15:00, 09:25:00) 09:21:05 Chicago
09:22:43 Seattle
09:22:54 Seattle
09:25:52 Chicago

ceiling(09:30:00) 09:35:21 Chicago
09:36:14 Seattle

max() 09:37:44 Phoenix

size(09:15:00, 09:25:00) is 5
rank(09:10:25) is 7
Ordered symbol table API

- Key `min()`: smallest key.
- Key `max()`: largest key.
- Key `floor(Key key)`: largest key less than or equal to given key.
- Key `ceiling(Key key)`: smallest key greater than or equal to given key.
- `int rank(Key key)`: number of keys less than given key.
- Key `select(int k)`: key with rank `k`.
- `Iterable keys()`: all keys in symbol table in sorted order.
- `Iterable keys(int lo, int hi)`: keys in `[lo, ..., hi]` in sorted order.
Order of growth for ordered symbol table operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Sequential search</th>
<th>Binary search</th>
</tr>
</thead>
<tbody>
<tr>
<td>search</td>
<td>$n$</td>
<td>log $n$</td>
</tr>
<tr>
<td>insert</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>min/max</td>
<td>$n$</td>
<td>1</td>
</tr>
<tr>
<td>floor/ceiling</td>
<td>$n$</td>
<td>log $n$</td>
</tr>
<tr>
<td>rank</td>
<td>$n$</td>
<td>log $n$</td>
</tr>
<tr>
<td>select</td>
<td>$n$</td>
<td>1</td>
</tr>
</tbody>
</table>
Lecture 24: Symbol Tables and Binary Search

- Symbol Tables
- Binary search
- Elementary Implementations of Symbol Tables
- Ordered Operations
Readings:

- Textbook: Chapter 3.1 (Pages 362-386)
- Website: https://algs4.cs.princeton.edu/31elementary/

Practice Problems:

- 3.1.1-3.1.6