CMPSC 250

Analysis of algorithms

Dr. Aravind Mohan

Allegheny College
Search Problem

Given a set of data
e.g., int [] array = 10, 2, 7, 9, 7, 4;
and a particular value,
e.g., int value = 7;

Find the first index of the value in the data
e.g., return index = 2
Search Problem

- **Input**: A set of data (for example, an array of Strings)
  A single data element to search for (e.g., “Yates”)

- **Output**: Position of the data element in the data set, or -1 if the element does not appear in the data set
Symbol Tables (Dictionary)

Key-value pair abstraction

- Insert a value with specified key
- Given a key, search for the corresponding value
- **Assumption:** Keys are unique - no key can be associated with more than one value
## Symbol Table Applications

<table>
<thead>
<tr>
<th>application</th>
<th>purpose of search</th>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary</td>
<td>find definition</td>
<td>word</td>
<td>definition</td>
</tr>
<tr>
<td>book index</td>
<td>find relevant pages</td>
<td>term</td>
<td>list of page numbers</td>
</tr>
<tr>
<td>file share</td>
<td>find song to download</td>
<td>name of song</td>
<td>computer ID</td>
</tr>
<tr>
<td>financial account</td>
<td>process transactions</td>
<td>account number</td>
<td>transaction details</td>
</tr>
<tr>
<td>web search</td>
<td>find relevant web pages</td>
<td>keyword</td>
<td>list of page names</td>
</tr>
<tr>
<td>compiler</td>
<td>find properties of variables</td>
<td>variable name</td>
<td>type and value</td>
</tr>
<tr>
<td>routing table</td>
<td>route Internet packets</td>
<td>destination</td>
<td>best route</td>
</tr>
<tr>
<td>DNS</td>
<td>find IP address given URL</td>
<td>URL</td>
<td>IP address</td>
</tr>
<tr>
<td>reverse DNS</td>
<td>find URL given IP address</td>
<td>IP address</td>
<td>URL</td>
</tr>
<tr>
<td>genomics</td>
<td>find markers</td>
<td>DNA string</td>
<td>known positions</td>
</tr>
<tr>
<td>file system</td>
<td>find file on disk</td>
<td>filename</td>
<td>location on disk</td>
</tr>
</tbody>
</table>
Basic Symbol Table API

```java
public class ST<Key, Value>

    ST()                   // create a symbol table
    void put(Key key, Value val)  // put key-value pair into the table
                                  // (remove key from table if value is null)
    Value get(Key key)         // value paired with key
                                  // (null if key is absent)
    void delete(Key key)       // remove key (and its value) from table
    boolean contains(Key key)  // is there a value paired with key?
    boolean isEmpty()         // is the table empty?
    int size()                // number of key-value pairs in the table
    Iterable<Key> keys()      // all the keys in the table

```
Keys and Values

- **Value type:** any generic type
- **Key type:** several natural assumptions
  - Assume keys are Comparable, use compareTo()
  - Assume keys are any generic type, use equals() to test equality
  - Assume keys are any generic type, use equals() to test equality; use hashCode() to scramble key
Frequency counter
Read a sequence of strings from standard input and print out one that occurs with highest frequency.

```bash
% more tinyTale.txt
it was the best of times
it was the worst of times
it was the age of wisdom
it was the age of foolishness
it was the epoch of belief
it was the epoch of incredulity
it was the season of light
it was the season of darkness
it was the spring of hope
it was the winter of despair

% java FrequencyCounter 1 < tinyTale.txt
it 10

% java FrequencyCounter 8 < tale.txt
business 122

% java FrequencyCounter 10 < leipzig1M.txt
government 24763
```
import java.util.*;

public class FrequencyCounter {
    public static void main(String[] args) {
        int minlen = Integer.parseInt(args[0]);
        ST<String, Integer> st = new ST<String, Integer>();
        while (!StdIn.isEmpty()) {
            String word = StdIn.readString();
            if (word.length() < minlen) continue;
            if (!st.contains(word)) st.put(word, 1);
            else st.put(word, st.get(word) + 1);
        }
        String max = "";
        st.put(max, 0);
        for (String word : st.keys()) {
            if (st.get(word) > st.get(max)) {
                max = word;
                StdOut.println(max + " " + st.get(max));
            }
        }
    }
}
Sequential Search Symbol Table

- **Data structure:** Maintain an (unordered) linked list (ArrayList) of \((key, value)\) pairs
- **Search:** Do a sequential traversal (scan) of all keys until find a match
- **Insert:** Search through all keys until find a match; if no match is found, add to the front of the list
## Running Time

<table>
<thead>
<tr>
<th>ST implementation</th>
<th>worst-case cost (after N inserts)</th>
<th>average case (after N random inserts)</th>
<th>ordered iteration?</th>
<th>key interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>search</td>
<td>N</td>
<td>N / 2</td>
<td>no</td>
<td>equals()</td>
</tr>
<tr>
<td>insert</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*sequential search (unordered list)*
Binary Search

- **Data structure:** Maintain an ordered array of \((key, value)\) pairs
- **Rank Helper Function:** How many keys \(< k\)?
Binary Search Example

Trace of binary search for rank in an ordered array

<table>
<thead>
<tr>
<th>keys[]</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>successful search for P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lo</td>
<td>hi</td>
<td>m</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>A</td>
<td>C</td>
<td>E</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>7</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>unsuccessful search for Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lo</td>
<td>hi</td>
<td>m</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>A</td>
<td>C</td>
<td>E</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>7</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>6</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>P</td>
<td>R</td>
</tr>
</tbody>
</table>

entries in black are a[lo..hi]
entry in red is a[m]
loop exits with keys[m] = P: return 6
loop exits with lo > hi: return 7
Binary Search Implementation

```java
public Value get(Key key)
{
    if (isEmpty()) return null;
    int i = rank(key);
    if (i < N && keys[i].compareTo(key) == 0) return vals[i];
    else return null;
}

private int rank(Key key)
{
    int lo = 0, hi = N-1;
    while (lo <= hi)
    {
        int mid = lo + (hi - lo) / 2;
        int cmp = key.compareTo(keys[mid]);
        if (cmp < 0) hi = mid - 1;
        else if (cmp > 0) lo = mid + 1;
        else if (cmp == 0) return mid;
    }
    return lo;
}
```
Binary Search Implementation Problem

To insert, need to shift all greater keys over

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>R</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
</tr>
<tr>
<td>X</td>
<td>7</td>
</tr>
<tr>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>M</td>
<td>9</td>
</tr>
<tr>
<td>P</td>
<td>10</td>
</tr>
<tr>
<td>L</td>
<td>11</td>
</tr>
<tr>
<td>E</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>keys[]</th>
<th>vals[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 N</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>
### Binary Search Implementation

<table>
<thead>
<tr>
<th>ST implementation</th>
<th>worst-case cost (after N inserts)</th>
<th>average case (after N random inserts)</th>
<th>ordered iteration?</th>
<th>key interface</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>search</td>
<td>insert</td>
<td>search hit</td>
<td>insert</td>
</tr>
<tr>
<td>sequential search</td>
<td>N</td>
<td>N</td>
<td>N / 2</td>
<td>N</td>
</tr>
<tr>
<td>(unordered list)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>binary search</td>
<td>log N</td>
<td>N</td>
<td>log N</td>
<td>N / 2</td>
</tr>
<tr>
<td>(ordered array)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Problem

- Cricket is a large cell phone company. They want to provide called ID capability: given a phone number, return the caller’s name.
- Phone numbers range from 0 to $r = 10^{11} - 1$
- There are $n$ phone numbers, $n << r$
- Want to do this as efficiently as possible.
The Problem

- Unordered sequence - sequential search
- Ordered sequence - binary search
- Sub optimal way - using a huge array
- Other ways?
Textbook Reading

- Chapter 03, Section 3.1, 361 - 395