COMP 250

Lecture 25

maps

Nov. 7, 2016
A map is a set of pairs \( \{ (x, f(x)) \} \).
The set of \( \{ f(x) \} \) is called the “range”.
Each \( x \) in domain maps to exactly one \( f(x) \) in codomain, but it can happen that \( f(x_1) = f(x_2) \) for different \( x_1, x_2 \), i.e. many-to-one.
Familiar examples

Calculus 1 and 2 ("functions"):

\[ f : \text{real numbers} \rightarrow \text{real numbers} \]

Asymptotic complexity in CS:

\[ t : \text{input size} \rightarrow \text{number of steps in an algorithm} \]
Object.hashCode() map in Java

objects in a Java program (runtime) \rightarrow \{0, 1, 2, ..., 2^{24} - 1\}

object’s base address in JVM memory (24 bits)
Object.hashcode() map in Java

By default, “obj1 == obj2” means “obj1.hashcode() == obj2.hashcode()”
String.hashCode() in Java

For each String, define an integer.
Example hash code for Strings
(not used in Java)

\[ h(s) \equiv \sum_{i=0}^{s.length-1} s[i] \]

e.g.

\[ h("eat") = h("ate") = h("tea") \]

ASCII and UNICODE values of ‘a’, ‘e’, ‘t’ are 97, 101, 116.
String.hashCode() in Java

\[ s.\text{hashCode}() \equiv \sum_{i=0}^{s.\text{length}-1} s[i] \cdot x^{s.\text{length} - 1 - i} \]

where \( x = 31 \) and using \text{int} arithmetic.

e.g. \( s = \text{“eat”} \) then \( s.\text{hashCode}() = 101 \cdot 31^2 + 97 \cdot 31 + 116 \)

\( s = \text{“ate”} \) then \( s.\text{hashCode}() = 97 \cdot 31^2 + 116 \cdot 31 + 101 \)
String.hashCode() in Java

\[ \text{s.hashCode()} \equiv \sum_{i=0}^{s.length-1} s[i] \times (31)^{s.length - 1 - i} \]

If \( \text{s1.hashCode()} == \text{s2.hashCode()} \) then ... ?

If \( \text{s1.hashCode()} != \text{s2.hashCode()} \) then ... ?
ASIDE: Use Horner’s rule for efficient polynomial evaluation

\[ s[0] \times x^3 + s[1] \times x^2 + s[2] \times x + s[3] \]

There is no need to compute each \( x^i \) separately.
ASIDE: Use Horner’s rule for efficient polynomial evaluation


\[
\begin{align*}
h &= 0 \\
&\text{for } (i = 0; \ i < s.\ length; \ i++) \\
&\quad h = h*31 + s[i]
\end{align*}
\]

For a degree \( n \) polynomial, Horner’s rule uses \( O(n) \) multiplications, not \( O(n^2) \).
A map is a set of (key, value) pairs. For each key, there is at most one value.
<table>
<thead>
<tr>
<th>Map</th>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address book</td>
<td>Name</td>
<td>Address, email..</td>
</tr>
<tr>
<td>Caller ID</td>
<td>Phone #</td>
<td>Name</td>
</tr>
<tr>
<td>Student file</td>
<td>ID or Name</td>
<td>Student record</td>
</tr>
<tr>
<td>Index at back of</td>
<td>keyword</td>
<td>List of book pages</td>
</tr>
<tr>
<td>book</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Map Entries

Each (key, value) pair is called an entry.

In this example, there are four entries.
In COMP 250 this semester, the above mapping has over 400 entries.

Most McGill students are not taking 250 this semester.

Student ID also happens to be part of the student record.
Map ADT

• put(key, value)  // add

• get(key)  // why not get(key, value) ?

• remove(key)

• ...

16
Data Structures for Maps

How to organize a set of (key, value) pairs, i.e. entries?
Array list

0 1 2 3 4
null null

Red and blue circles are connected to the null values.
Singly linked list
Doubly linked list

next  prev  element
Assumptions about keys

Can two keys have the same value? Yes.

Can one key have two values? No.
Special case #1: what if keys are comparable?

Next few slides consider this case.
Array list (sorted by key)

What is the O() bound for get(key), put(key, value), remove(key)?
What is the $O()$ bound for `get(key)`, `put(key, value)`, `remove(key)`?
What is $O()$ bound for put() and removeMin()?

What is $O()$ bound for get(key) and remove(key)?
Special case #2: what if keys were unique positive integers in small range?

Then, we could use an array of type $V$ (value) and have $O(1)$ access.

This would not work well for 9 digit student IDs.
Next lecture: hash maps

Somehow we want to map the keys to a small range of positive integers.

How to make this work?