CSCI 136: Data Structures and Advanced Programming
Lecture 28
Hash tables

Instructor: Dan Barowy Williams

Topics
Hash tables
Hash functions

Your to-dos
1. Read before Mon: Bailey, Ch. 16.3.
2. Lab 9 (solo lab), due Tuesday 5/3 by 10pm.

Announcements
Computer Science Colloquium
Friday, April 29 @ 2:35pm in Wege (TCL 123)
Brian Brubach (Wellesley)
Gerrymandering, redistricting, and the quest for fairer representative democracy
Partisan gerrymandering in the United States is an old problem. However, our most effective tools for measuring and regulating it are fairly new and still not well-understood. This talk will highlight what roles computer science can play in the evolution of electoral systems using political redistricting as the primary example. We’ll summarize recent advances in the area of measuring and quantifying gerrymandering that have led to partisan maps being struck down in state courts. Then, we’ll examine how these new tools can alter the theoretical analysis of electoral systems and even be used to draw fairer maps in practice. Finally, we’ll look to the future at what can be achieved through bigger, systemic changes. Along the way, we’ll explore how to identify new research directions and how computer science can help redefine what a right to vote means.
Hash tables

My favorite data structure

Note about lab 9:

You may use the structure5 `Hashtable` implementation.

Recall: arrays

An array is a data structure consisting of a **sequential collection of elements**, each identified by an **index**.

<table>
<thead>
<tr>
<th>A</th>
<th>13</th>
<th>2</th>
<th>451</th>
<th>42</th>
<th>9</th>
<th>6</th>
<th>–4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Performance guarantees:
1. **read** an element: \( O(1) \)
2. **write** an element: \( O(1) \)

Can we capture some of this for a more general structure?

Generalization: associative array

An associative array or is a data structure consisting of a **sequential collection of elements**, each identified by a **key**.

An associative array is a **map**.

<table>
<thead>
<tr>
<th>A</th>
<th>13</th>
<th>2</th>
<th>451</th>
<th>42</th>
<th>9</th>
<th>6</th>
<th>–4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>Adam</td>
<td>Sue</td>
<td>Ed</td>
<td>Sam</td>
<td>Fay</td>
<td>Dan</td>
<td>Ted</td>
<td></td>
</tr>
</tbody>
</table>

Performance guarantees:
1. **read** an element: \( O(1) \)?
2. **write** an element: \( O(1) \)?

**How** can we make this happen?
Need: function to map key to index

Suppose we have a function:

\[ h(k) \rightarrow z \]

where \( k \) is a key of arbitrary type and \( z \in \mathbb{Z} \),

then we could construct another function:

\[
\text{int index(K key)} \{
    \text{return abs}(h(key) \% A.length);}\
\]

Hash function

A hash function is any function that can be used to map data of arbitrary size onto data of a fixed size.

Hash table

A hash table is a data structure that implements the map abstract data type. A hash table uses a hash function to compute an index into an array of buckets, from which the desired value can be found.

Hash table

A hash table is a data structure that implements the map abstract data type. A hash table uses a hash function to compute an index into an array of buckets, from which the desired value can be found.

Hash function

A hash function is any function that can be used to map data of arbitrary size onto data of a fixed size.

Nerd rant

A.O. Scott in *The New York Times*’ review deduced from the film that Turing was “a sentient robot, an empathetic space alien, a warm-blooded salamander with crazy sex appeal.”

“[C]olleagues at the time called him intensely shy and kindly.”

“… unfailingly generous with his time and expertise …”

“… inspired loyalty and affection among those who appreciated his unusual gifts.”

See: http://blog.yalebooks.com/2015/01/07/alan-turing/
Hash function

Useful hash functions also provide the following guarantees:

**Determinism**: a given input value must always generate the same hash value.

**Uniformity**: maps the expected inputs as evenly as possible over its output range.

**Equivalence**: any two values that are considered equivalent should produce the same hash value.

---

Question

Is a function that *generates a random number* a good hash function?

**No.** Random numbers do tend to be uniform, but are not deterministic.

---

Activity

See if you can come up with a simple hash function for strings.

**Determinism**: a given input value must always generate the same hash value.

**Uniformity**: maps the expected inputs as evenly as possible over its output range.

**Equivalence**: any two values that are considered equivalent should produce the same hash value.

---

Hash codes

Good hash functions are provided for common types. You can override for your own classes.

```java
public int hashCode()
Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by `HashMap`.

The general contract of `hashCode` is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer, provided no information used in equal (or `equals`) comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.

- If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.

- It is not required that if two objects are unequal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

As much as is reasonably practical, the `hashCode` method defined by class `Object` does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java™ programming language.)

Returns:
- a hash code value for this object

See Also:
- `equals`, `hashCode`, `HashMap`
Code: let’s check the quality of hash

Recap & Next Class

Today:
- Hash tables
- Hash functions

Next class:
- Collisions
- Graphs