CSE 373

APRIL 24TH – HASHING
EXAM FRIDAY

• Practice exam after class today
• Topics:
  • Stacks and Queues
  • BigO Notation and runtime Analysis
  • Heaps
  • Trees (BST and AVL)
  • Traversals
  • Design Tradeoffs
EXAM FRIDAY

• Format
  • No note sheet
  • One section of short answer
  • 4-5 Technical Questions
  • 1 Design Decision Question
  • Less than 10 minutes per problem
EXAM FRIDAY

• No Java material on the exam
• Looking for theoretical understanding
  • Explanations are important (where indicated)
• If you get stuck on a problem, move on
• Any questions?
TODAY’S LECTURE

• Hashing
  • Basic Concept
  • Hash functions
  • Collision Resolution
  • Runtimes
• Introduction

  • Suppose there is a set of data \( M \)
  • Any data we might want to store is a member of this set. For example, \( M \) might be the set of all strings
  • There is a set of data that we actually care about storing \( D \), where \( D \ll M \)
  • For an English Dictionary, \( D \) might be the set of English words
HASHING

• What is our ideal data structure?
  • The data structure should use $O(D)$ memory
    • No extra memory is allocated
  • The operation should run in $O(1)$ time
    • Accesses should be as fast as possible
HASHING

• What are some difficulties with this?
  • Need to know the size of $D$ in advance or lose memory to pointer overhead
  • Hard to go from $M \rightarrow D$ in $O(1)$ time
HASHING

• Memory: The Hash Table
  • Consider an array of size $c \times D$
  • Each index in the array corresponds to some element in $M$ that we want to store.
  • The data in $D$ does not need any particular ordering.
THE HASH TABLE

• How can we do this?
THE HASH TABLE

• How can we do this?
  • Unsorted Array
THE HASH TABLE

• How can we do this?
  • Unsorted Array
THE HASH TABLE

• How can we do this?
  • Unsorted Array

M

Apple
Pear
Orange
THE HASH TABLE

• How can we do this?
  • Unsorted Array
THE HASH TABLE

• How can we do this?
  • Unsorted Array
THE HASH TABLE

• What is the problem here?
  • Takes O(D) time to find the word in the list
  • Same problem with sorted arrays!
THE HASH TABLE

• What is another solution?
  • Random mapping
THE HASH TABLE

• What’s the problem here?
  • Can’t retrieve the random variable, O(D) search!
THE HASH TABLE

• What about a pseudo-random mapping?
  • This is “the hash function”
The Hash Function maps the large space $M$ to our target space $D$.

We want our hash function to do the following:

- Be repeatable: $H(x) = H(x)$ every run
- Be equally distributed: For all $y,z$ in $D$, $P(H(y)) = P(H(z))$
- Run in constant time: $H(x) = O(1)$
Let’s consider an example. We want to save 10 numbers from all possible Java ints

What is a simple hash function?

\[ h(x) = \text{key}\%10 \]
HASH EXAMPLE

• Let’s insert(519) table
  • Where does it go?
  • \( 519 \% 10 = \)

\[
h(x) = \text{key} \% 10
\]
Let’s insert(519) table

- Where does it go?
- $519 \% 10 = 9$
HASH EXAMPLE

- Insert(204)

\[ h(x) = \text{key} \% 10 \]

204
519

0
1
2
3
4
5
6
7
8
9: 519
HASH EXAMPLE

• Insert(204)
HASH EXAMPLE

• insert(1001)
HASH EXAMPLE

• insert(1001)
HASH EXAMPLE

• Is there a problem here?

\[ h(x) = \text{key}\%10 \]
Is there a problem here?

- insert(3744)

\[ h(x) = \text{key} \% 10 \]
HASH EXAMPLE

- Is there a problem here?
- insert(3744)

\[ h(x) = \text{key}\%10 \]
HASH EXAMPLE

• Is there a problem here?
  • insert(3744)
  • This is called a collision!

\[
h(x) = \text{key}\%10
\]

<table>
<thead>
<tr>
<th>0</th>
<th>1: 1001</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>214</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>214</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>519</td>
</tr>
</tbody>
</table>
HASH EXAMPLE

• How to rectify collisions?
  • Think of a strategy for a few minutes

• Possible solutions:
  • Store in the next available space
  • Store both in the same space
  • Try a different hash
  • Resize the array
LINEAR PROBING

• Consider the simplest solution
  • Find the next available spot in the array
  • This solution is called linear probing

\[
h(x) = \text{key}\%10
\]
LINEAR PROBING

• What are the problems with this?
  • How do we search for 3744?
    • Need to go to 4, and then cycle through all of the entries until we find the element or find a blank space
  • What if we need to add something that ends in 5?
    • It also ends up in this problem area
    • This is called clustering
CLUSTERING

• What are the negative effects of clustering?

  • If the cluster becomes too large, two things happen:
    • The chances of colliding with the cluster increase
    • The time it takes to find something in the cluster increases. This isn’t O(1) time!
How can we solve this problem?

- Resize the array
  - Give the elements more space to avoid clusters. *How long does this take? O(n)! all of the elements need to be rehashed.*
- Store multiple items in one location
  - This is called **chaining**
  - We'll discuss it after the midterm
HASH TABLES

• Take-aways for the midterm
  • Hashtables should provide O(1) dictionary operations
  • Collisions make this problem difficult to achieve
  • Hashtables rely on a array and a hash function
  • The array should be relative to the size of the data you want to keep
  • The hash function should run in constant time and should distribute among the indices in the target array
  • Linear probing is a solution for collisions, but only works when there is lots of free space
  • Resizing is very costly
NEXT CLASS

- Hash Tables
  - Examples, examples, examples
  - No new theory
- Exam review