Dynamic data structures
Feeling smug
   About that data structure
But can it dance
   When data chooses to prance
Can it zig
   When the data zags?
Or has it tied itself
   Up in knots

Professor: SHRIDEEP PALICKARA
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- Switches vs routers
- How does NAT know which private IP to route to?
- Difference between static and dynamically assigned IPs
- Do all devices have a public IP unless connected to a router?
- Subnet masks: Used to separate network and host
Topics covered in this lecture

- DNS wrap-up
- OSI Network Architecture
- Storage
  - Linear scan
  - Binary search
  - Dynamic data structures

DNS
An example

Root Level

Top Level
com
org
edu

Second Level
example

www
mail
ftp

DNS Hierarchy

- At the top of this hierarchical tree is the root domain

- The root domain doesn’t get a textual representation in a DNS name like www.example.com
  - But it's an essential part of the DNS hierarchy

- The root domain contains records for all the top-level domains (TLDs) like .com, .org, .edu, .net, and so forth

- There are 13 root name servers worldwide, each responsible for knowing the details of all the top-level domain servers.
Resolving a name [1/2]

- Let's say you want to look up a record in a domain that ends with .com
- A root server can point you to a TLD server that knows about domains under .com
- A top-level DNS server is responsible for knowing about all the second-level domains under its hierarchy
  - A top-level DNS server for .com could point you to the second-level DNS server for example.com

Resolving a name [2/2]

- The DNS servers for second-level domains maintain records for hosts and third-level domains that fall under second-level domains
  - This means that the DNS server(s) for example.com are responsible for maintaining the records for hosts like www.example.com and mail.example.com
- This pattern continues, allowing for nested domains
  - Once a domain is registered under a top-level domain?
    - Domain owner can create as many records as needed under their domain
When a computer needs to find the IP address for an FQDN, it sends a request to its configured DNS server.

If the server has recently looked up the requested record?
- May have a copy of that record stored in its cache
- Can immediately return the IP address to the client

If the DNS server doesn’t have the response in cache
- It may query other DNS servers as needed to get the answer
  - Starting at the root
  - Working down the hierarchy of servers to find the record in question
- Once the server has the record, it can cache it so that it can immediately respond to future queries for that record
  - Eventually the cached record is removed
    - To ensure that the server always provides reasonably recent data
OSI NETWORK ARCHITECTURE

OSI network architecture

- Model is a product of the Open Systems Interconnection (OSI) project
  - At the International Organization for Standardization (ISO)
- Partitions network functionality into 7 layers
- Physical Layer
  - Handles transmission of raw bits
  - Standardizes electrical, mechanical, and signaling interfaces
    - 0 bit should be received as 0 not 1
OSI network architecture:
Data link Layer

- Collects stream of bits into a **frame**
  - Puts special **bit pattern** at the start/end of each frame
  - Frames, not raw bits, are delivered to host

- Compute **checksum** for frame
  - Check for correctness and request retransmission

- Network adaptors & device drivers implement this

OSI network architecture

- **Network layer**
  - Handles routing among nodes in a **packet-switched** network
  - Unit of data exchanged is **packet** not frames

- Layers implemented on all network nodes?
  - Physical, data and network
OSI Architecture

Usually run only on the end host, not switches

One or more nodes within the network

How messages flowing through the OSI stack will appear on the network

Data link layer header
Network layer header
Transport layer header
Session layer header
Presentation layer header
Application layer header
Data

Data link layer trailer
OSI network architecture

- **Transport**
  - Implements process-process *channel*
  - **Messages** (not packet or frame)

- **Presentation**
  - **Format** of data exchanged between peers

- **Session**
  - **Namespace** to tie different transport-streams that are part of the application

---

**DATA STRUCTURES FOR STORAGE SYSTEMS**
Problem definition

- Given a set of \( N \) data points \( X = \{ x_1, x_2, \ldots, x_N \} \) and a target value \( x' \)
- Find a point \( x_i \in X \) such that \( x' = x_i \)
- Or indicate that no such point exists
- In our everyday lives, we would likely describe the task as “Find me this particular thing.”

Let’s start with a simpler solution: Linear scan

- Linear scan searches for a target value by testing each value in our list, one after the other, until:
  - The target is found or
  - We reach the end of our list
Linear Scan: Searching for the value 21

About Linear Scan

- **Thorough but inefficient**, especially for large lists
- Guaranteed to find the item of interest (if the item is in the data)
  - Checks *every possible* item until
    - It finds a match or confirms the item is missing
- Brute force!
Linear Scan: Why is this the case?

- We know nothing about the **structure** of the data
  - There is nothing we can do to streamline the process
- The target value could be in any bin, so we may need to check them all

What’s next?

- Let’s see how a small amount of structure in the data changes everything
Binary Search

- An algorithm for efficiently searching a sorted list
- Checks the sorted list for a target value by
  - Repeatedly dividing the list in half
  - Determining which of the two halves could contain the target value
    - And discarding the other half
Binary search

- Binary search is an algorithm to find a target value \( v \) in a sorted list.
- **Only works on sorted data**
- The algorithm can be written to work with data sorted in either increasing or decreasing order.
  - We will look at increasing order—lowest to highest.

Secret sauce?

- The key to efficient algorithms is using information or structure within the data.
- In the case of binary search, we use the fact that the array is sorted in increasing order.
Binary search: The algorithm

1) Partition the list in half and determine in which half \( v \) must reside
2) Discards the half that \( v \) is not in
   - Repeat the process with only the half that can possibly still contain \( v \)
   - Until only one value remains

More formally ...

- Consider a sorted array \( A \):
  - \( A[i] \leq A[j] \) for any pair of indexes \( i \) and \( j \) such that \( i < j \)
- While this might not seem like a lot of information
  - It's enough to allow us to rule out entire sections of the array
- Similar to how we avoid the ice cream aisle when searching for coffee
  - Once we know an item won't be in a given area, we can rule out that entire set of items in that area without individually checking them
What Binary Search needs

- Binary search tracks the current search space with two bounds:
  - the upper bound \( \text{IndexHigh} \) marks the highest index of the array that is part of the active search space, and
  - the lower bound \( \text{IndexLow} \) marks the lowest

- Invariant: if the target value is in the array?
  - \( A[\text{IndexLow}] \leq v \leq A[\text{IndexHigh}] \)

Binary search: Details

- Start each iteration by choosing the midpoint of the current search space:
  - \( \text{IndexMid} = \text{Floor}((\text{IndexHigh} + \text{IndexLow}) / 2) \)
  - Floor is a mathematical function that rounds a number down to an integer
Binary search: Details

- Compare value at the middle location, A[IndexMid], with the target value v
  - If A[IndexMid] < v, the target value must lie after the middle index
    - Allows us to chop the search space in half by making IndexLow = IndexMid + 1
  - If A[IndexMid] > v, the target value must lie before the middle index
    - Allows us to chop the search space in half by making IndexHigh = IndexMid - 1
  - Of course, if A[IndexMid] == v, we immediately conclude the search
    - We've found the target

Searching for the value 15

<table>
<thead>
<tr>
<th>L</th>
<th>M</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>30</td>
</tr>
<tr>
<td>35</td>
<td>51</td>
<td>54</td>
</tr>
</tbody>
</table>

\[
\left\lfloor \frac{0+11}{2} \right\rfloor
\]
Searching for the value 15

\[
\begin{array}{cccccccccccc}
-5 & -1 & 0 & 3 & 9 & 11 & 15 & 17 & 30 & 35 & 51 & 54 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\end{array}
\]

\[
\frac{(6+11)}{2}
\]

Searching for the value 15

\[
\begin{array}{cccccccccccc}
-5 & -1 & 0 & 3 & 9 & 11 & 15 & 17 & 30 & 35 & 51 & 54 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\end{array}
\]

\[
\frac{(6+7)}{2}
\]
Absent value: Linear scan

- In the linear scan case, we know that an element is not in the list?
  - As soon as we hit the end of the list

Absent value: Binary search

- We can assert our target item does not exist by testing the bounds
- As the search progresses?
  - The upper and lower bounds move closer and closer until there are no unexplored values between them
  - Since we are always moving one of the bounds past the midpoint index?
    - We can stop the search when IndexHigh < IndexLow
    - At that point, we can guarantee the target value is not in the list
Searching for the value 16

-5  -1  0  3  9  11  15  17  30  35  51  54

LMH

0  1  2  3  4  5  6  7  8  9  10  11

Searching for the value 16

-5  -1  0  3  9  11  15  17  30  35  51  54

LMH

0  1  2  3  4  5  6  7  8  9  10  11
Dynamic data structures

- **Alter their structure** as the data changes

- These **structural adaptations** may include
  - Growing the size of the data structure on demand
  - Creating dynamic, mutable linkings between different values, etc.

- Dynamic data structures are at the heart of almost every computer program in the world
  - Underpin some of the most exciting, interesting, and powerful algorithms in computer science
But arrays are so easy to work with ... [1/2]

- Arrays are like parking lots
- They give us a place to store information, but don’t provide much in the way of adaptation

Sure, we can sort the values in an array (or cars in our parking lot) and use that structure to make binary search efficient
- But we’re just changing the ordering of the data within the array
- The data structure itself is neither changing nor responding to changes in the data

If we later change the data in a sorted array, say by modifying the value of an element?
- We need to re-sort the array
Binary Search Trees

To dwellers in a wood, almost every species of tree has its voice as well as its feature.

Thomas Hardy, Under the Greenwood Tree

Binary Search Tree

- Binary search trees use the concepts underpinning the binary search algorithm to create a dynamic data structure
  - The key word here is dynamic

- Unlike sorted arrays, binary search trees support the efficient addition and removal of elements in addition to searches
  - Making them the perfect blend of the algorithmic efficiency of binary search and the adaptability of dynamic data structures
The contents of this slide-set are based on the following references
