Indexing

September 29, 2021
Plan for today

- Naming
  - Attacks on DNS
  - DNSSEC
- Inverted indices
- B+ trees
- Data interchange
- Extensible Markup Language (XML)
- XML Schema; DOM
- XPath
Example B+ Tree

- Data (inverted list pointers) is at the leaves; intermediate nodes have copies of search keys
  - Why is that a good thing?
- Search begins at root, and key comparisons direct it to a leaf
- Search for be↓, bobcat↓ ...

> Based on the search for bobcat*, we know it is not in the tree!
Some details

- How would you find all the words starting with the letters 'com'? 

- How would you delete something? 

- Do you always have to split/merge? 

- How would you choose the order of the tree?
How do we distribute a B+ Tree?

- We need to host the root at one machine and distribute the rest.

- What are the implications for scalability?
  - Consider building the index as well as searching.

![Diagram of B+ Tree distribution]
Eliminating the root

- Sometimes we don’t want a tree-structured system because the higher levels can be a central point of congestion or failure.

- Two strategies:
  - Modified tree structure (e.g., the BATON p2p tree)
  - Non-hierarchical structure (e.g., distributed hash table)
Example: B+ tree

- Insert 15, 11, 12, 32, 74
Virtues of the B+ Tree

- B+ tree and other indices are quite efficient:
  - Height-balanced; $\log_F N$ cost to search
  - High fanout (F) means depth rarely more than 3 or 4
  - Almost always better than maintaining a sorted file
  - Typically, 67% occupancy on average
Recap: B+ trees

- A very common data structure for indices
  - Used, e.g., in many file systems and many DBMS

- Berkeley DB library is a toolkit for B+ trees that you will be using later in the semester:
  - Interface: open B+ Tree; get and put items based on key
  - Handles concurrency, caching, etc.

- Very efficient
  - Height-balanced; $\log_F N$ cost to search
  - High fanout ($F$) means depth rarely more than 3 or 4
  - Typically, 67% occupancy on average
Plan for today

- B+ trees ✔
- Data interchange
- Extensible Markup Language (XML)
- XML Schema; DOM
- XPath
Kinds of content

- Keyword search and inverted indices are great for locating text documents
- But what if we want to index and/or share other kinds of content?
  - Spreadsheets
  - Maps
  - Purchase records
  - Objects
  - etc.
- Let’s talk about **structured data**!
  - Now: Representation and transport
  - Later: Indexing and retrieval
Sending data

- How do we send data **within** a program?
  - What is the implicit model?
  - How does this change when we need to make the data persistent?

- What happens when we are coupling systems?
  - How do we send data **between** programs
    - on the same machine?
    - between different programming languages?
    - on different machines?
Motivating example: Web services

- Accessible to other applications over the web
  - Examples: Google Search, Google Maps API, Facebook Graph API, eBay APIs, Amazon Web Services, ...

Map service (used by Alice)
Mashup example: Google Transit
A key challenge

- Nodes need to communicate with each other
  - E.g., using remote procedure calls
- Network messages are strings of bytes
  - No particular structure - must be defined by the application
  - Sender marshals the data and produces a string of bytes
    - Pointers must be encoded somehow
    - Specific byte order; metadata to describe the data
  - Receiver unmarshals the data again
Communication and streams

- When storing data to disk, we have a combination of sequential and random access.

- When sending data on “the wire”, data is only sequential.
  - “Stream-based communication” based on packets.

- What are the implications here?
  - Pipelining, incremental evaluation, ...
Data interchange is hard

What does Bob need to know to understand Alice's document?

- Physical data model (data encoding)
  - Code: ASCII or Unicode or ...?
  - Byte order: Little-endian? Big-endian?
  - Marshalling format: Tagged? Fixed? Which field sizes?

- Logical data model (data representation)
  - Semantic heterogeneity
  - Imprecise and ambiguous values or descriptions
  - ...

Data comes in many formats

<table>
<thead>
<tr>
<th>Data type</th>
<th>Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>ASCII, Word document, RTF, TeX, PDF, HTML, ...</td>
</tr>
<tr>
<td>Database</td>
<td>MySQL, Oracle, Access, Works, OpenOffice, ...</td>
</tr>
<tr>
<td>Image</td>
<td>JPG, GIF, BMP, PNG, RAW, TIFF, Corel, Photoshop, ...</td>
</tr>
<tr>
<td>Music</td>
<td>AIFF, MP3, AAC, RA, Ogg, MID, MOD, SWA, ...</td>
</tr>
<tr>
<td>Video</td>
<td>AVI, M4V, MPEG, Ogg, WMV, RM, DVD, MOV, ...</td>
</tr>
<tr>
<td>Scientific data</td>
<td>Probably at least as many as there are researchers</td>
</tr>
</tbody>
</table>
Example: Old ID3v1 tags in MP3

<table>
<thead>
<tr>
<th>Offs</th>
<th>Len</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>Identifier: &quot;TAG&quot;</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>Song title string</td>
</tr>
<tr>
<td>33</td>
<td>30</td>
<td>Artist string</td>
</tr>
<tr>
<td>63</td>
<td>30</td>
<td>Album string</td>
</tr>
<tr>
<td>93</td>
<td>4</td>
<td>Year string</td>
</tr>
<tr>
<td>97</td>
<td>28</td>
<td>Comment string</td>
</tr>
<tr>
<td>125</td>
<td>1</td>
<td>Zero byte separator</td>
</tr>
<tr>
<td>126</td>
<td>1</td>
<td>Track byte</td>
</tr>
<tr>
<td>127</td>
<td>1</td>
<td>Genre byte</td>
</tr>
</tbody>
</table>

"TAG"

"Members Only"

"1998"

"Sheryl Crow"

"The Globe Sessions"

Track #10

Genre not specified
Example: JPEG header

JPEG “JFIF” header:
- Start of Image (SOI) marker -- two bytes (FFD8)
- JFIF marker (FFE0)
- length -- two bytes
- identifier -- five bytes: 4A, 46, 49, 46, 00
  (the ASCII code equivalent of a zero terminated "JFIF" string)
- version -- two bytes: often 01, 02
  - the most significant byte is used for major revisions
  - the least significant byte for minor revisions
- units -- one byte: Units for the X and Y densities
  - 0 => no units, X and Y specify the pixel aspect ratio
  - 1 => X and Y are dots per inch
  - 2 => X and Y are dots per cm
- $X_{\text{density}}$ -- two bytes
- $Y_{\text{density}}$ -- two bytes
- $X_{\text{thumbnail}}$ -- one byte: 0 = no thumbnail
- $Y_{\text{thumbnail}}$ -- one byte: 0 = no thumbnail
- (RGB)n -- 3n bytes: packed (24-bit) RGB values for the thumbnail pixels,
  \[ n = X_{\text{thumbnail}} \times Y_{\text{thumbnail}} \]
Problem: Too many formats

- You need to look into a manual to find a specific file format
  - http://www.wotsit.org/

- Automating data exchange is very hard
  - $O(N^2)$ problem: Everyone needs to understand everyone else's data format
  - http://tika.apache.org/

- The web is about making data exchange easier... maybe we can do better?
  - Goal: "The mother of all file formats"
Problem: Too many formats

How standards proliferate:
(See: A/C chargers, character encodings, instant messaging, etc.)

Situation: There are 14 competing standards.

14?! Ridiculous! We need to develop one universal standard that covers everyone’s use cases. Yeah!

Soon:

Situation: There are 15 competing standards.
Desiderata for data interchange

- Ability to represent many kinds of information
  - Different data structures

- Hardware-independent encoding
  - Endian-ness, UTF vs. ASCII vs. EBCDIC

- Standard tools and interfaces

- Ability to define “shape” of expected data
  - With forwards- and backwards-compatibility!

- Two common solutions:
  - XML (which established the basic approach)
  - JSON (less formalized, different symbols, but – the same)
Plan for today

- B+ trees ✔
- Data interchange ✔
- Extensible Markup Language (XML)
  - Data model
  - Encoding data in XML
  - Namespaces
  - Well-formed and valid
- XML Schema; DOM
- XPath
Extensible Markup Language (XML)

What is it?
- A set of rules for encoding documents
- A subset of SGML

Who uses it?
- Document Object Model (DOM) -- OO representation of XML
- Simple API for XML (SAX) -- event-driven parser for XML
- Ant -- Java's 'make' tool, whose 'Makefile' uses XML
- XPath, XQuery, XSL, XSLT
- Web service standards (e.g., REST)
- Anything Ajax, e.g., jQuery
  - Often used for marshalled data in web services
- Your e-filed tax returns! (IRS MeF)
  - Generally as an exchange format for data (research, govt. data, ...)
- Your Office documents (DOCX, XSLX, ...)

© 2021 A. Haeberlen, Z. Ives, V. Liu
Example XML document

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<dblp>
  <mastersthesis mdate="2002-01-03" key="ms/Brown92">
    <author>Kurt P. Brown</author>
    <title>PRPL: A Database Workload Specification Language</title>
    <year>1992</year>
    <school>Univ. of Wisconsin-Madison</school>
  </mastersthesis>
  <article mdate="2002-01-03" key="tr/dec/SRC1997-018">
    <editor>Paul R. McJones</editor>
    <title>The 1995 SQL Reunion</title>
    <journal>Digital System Research Center Report</journal>
    <volume>SRC1997-018</volume>
    <year>1997</year>
    <ee>db/labs/dec/SRC1997-018.html</ee>
    <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
  </article>
</dblp>
```

- Structure is very similar to HTML
  - This is not an accident - both are subsets of SGML
XML data model

- To model an XML document, we need at least the following ('XML information set'):
  - Document (root)
  - Element
  - Attribute
  - Processing instruction
  - Text (content)
  - Namespace
  - Comment
  - ... and a few more obscure items
XML data model visualized

Root

?xml

dblp

mastersthesis

author title year school

2002...

ms/Brown92

PRPL...

Kurt P....

1992

Univ....

article

editor title journal volume year ee ee

title

date

key

2002...

tr/dec/...

The...

Paul R.

Digital...

SRC...

http://www.

1997

db/labs/dec

28
XML easily encodes tables

<table>
<thead>
<tr>
<th>ID</th>
<th>Course</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>553-s20</td>
<td>B+</td>
</tr>
<tr>
<td>23</td>
<td>455-f20</td>
<td>A</td>
</tr>
</tbody>
</table>

"student-course-grade" table

```xml
<student-course-grade>
  <tuple>
    <sid>1</sid>
    <course>533-s20</course>
    <grade>B</grade>
  </tuple>
  <tuple>
    <sid>23</sid>
    <course>455-f20</course>
    <grade>A</grade>
  </tuple>
</student-course-grade>
```
XML also encodes objects

- What do we do about the pointers?
  - Can be represented as IDs and indirection/references
Handling name clashes

What if a tag is used by multiple sources?
- Example: XML document with book reviews that includes HTML for display formatting

Confuses software that is parsing this document (book title or page title?)
XML namespaces

Solution: XML namespaces

- Part 1: Bind namespaces to URIs
- Part 2: Qualified names
XML is not enough on its own

- Too unconstrained for many cases!
  - How will we know when we're getting garbage?
  - How will we query data in an XML document?
  - How will we understand the data we've got?
Well-formed and valid

How will we know whether document is ok?

- Idea: Check whether begin and end tags are correctly nested, special characters (<, & ) are properly used, etc.
- If this (and a few other conditions) hold, the document is well-formed
- But is the document valid, i.e., is the structure okay?
- Need some form of specification for valid documents
Recap: XML

- **Textual data format**
  - Familiar from HTML: Tags, elements, attributes, ...
  - Can encode complex data structures: Relations, pointers, ...
  - Used as a data exchange format, for marshalling, ...
  - Namespaces to help with name clashes

- **Not enough on its own**
  - Need a way to detect whether document is valid/well-formed
  - Need a way to query data in XML documents
  - Need a way to understand/work with the data
Plan for today

- **B+ trees ✓**
- **Data interchange ✓**
- **Extensible Markup Language (XML) ✓**
  - Data model
  - Encoding data in XML
  - Namespaces
  - Well-formed and valid
- **XML Schema; DOM**
  - XML Schema
  - Document Object Model (DOM)
- **XPath**
Example XML Schema

```xml
<?xml version="1.0" encoding="utf-8"?><xs:schema elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Address">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Recipient" type="xs:string"/>
        <xs:element name="Street" type="xs:string"/>
        <xs:element name="Town" type="xs:string"/>
        <xs:element name="County" type="xs:string" minOccurs="0"/>
        <xs:element name="PostCode" type="xs:string"/>
        <xs:element name="POBox" type="xs:boolean"/>
        <xs:element name="Since" type="xs:date"/>
        <xs:element name="Country">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="FR"/>
              <xs:enumeration value="DE"/>
              <xs:enumeration value="UK"/>
              <xs:enumeration value="US"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Root of every XML Schema

Structured type

Elements can have minOccurs, maxOccurs

Actual data types
Basic constructs of Schema

- Separation of elements (and attributes) from types:
  - complexType is a structured type, which can have sequences or choices
    - Sequence: Elements in the sequence must be present, in that order
    - Choice: Only one of the elements must be present
  - element and attribute have name and type; elements may also have minOccurs and maxOccurs

- Subtyping, most commonly using

```xml
<complexContent>
  <extension base="prevType">
    ...
  </extension>
</complexContent>
```
Some more examples

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="pets">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="dog" type="xs:string"/>
      <xs:element name="cat" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Adds three elements to 'personinfo'
Designing an XML schema or DTD

- Often we are given an existing DTD or schema
  - Example: HTML DTD

- If not, we need to design one
  - What would be a good approach?
  - Idea: Orient the XML tree around the 'central' objects in the application of interest
  - We've already discussed this in the context of mapping data structures to XML; XML schema can specify such a mapping
Manipulating XML documents

- **Typical tasks:**
  - Restructure a XML document
  - Add/remove/modify elements
    - Example: Dynamically changing a document in response to inputs
  - Retrieve certain elements that satisfy some constraint
    - Examples: All books, all addresses in New Hampshire

- **How do we do this in a program?**
  - Need an interface that allows programs and scripts to dynamically access and update the content, structure, and style of documents
  - Solution: The Document Object Model (DOM)