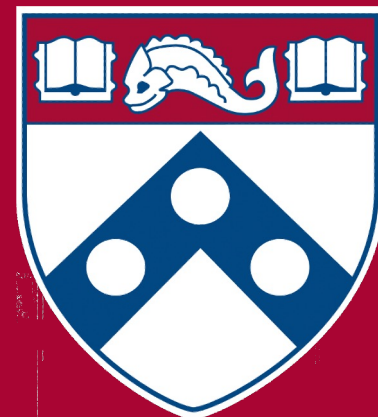


Indexing

CIT5940



Introduction

- You'll often work with a dataset that doesn't entirely fit into your program's memory
- **Indexing**: the process of associating a *search key* with the location (on disk) of a corresponding data record
 - Think of the index in a textbook: given a topic name, it tells you where to go find more information about that topic.
- Remember: program memory is fast but expensive, whereas disk space is slow and cheap.
 - Do as little seeking on disk as possible!

Index

- The index **does not store the record**
- The index **stores a *reference to the record***
- A collection of records can be supported by **multiple indices**
 - separate index for each key field in the record

Records

<Harry, Smith, sharry,
34893394, Lecturer>

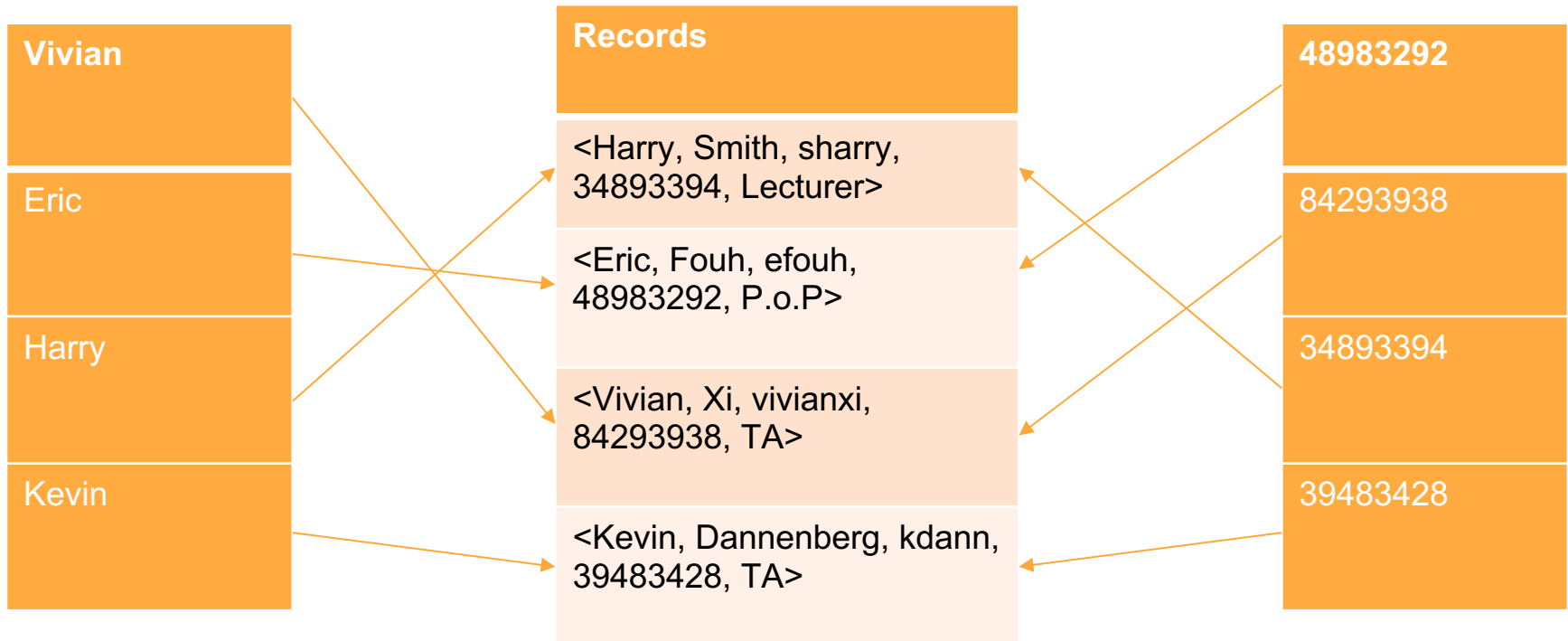
<Eric, Fouh, efouh,
48983292, P.o.P>

<Vivian, Xi, vivianxi,
84293938, TA>

<Kevin, Dannenberg, kdann,
39483428, TA>

Index

- The index **does not store the record**
- The index **stores a *reference* to the record**
- A collection of records can be supported by **multiple indices**

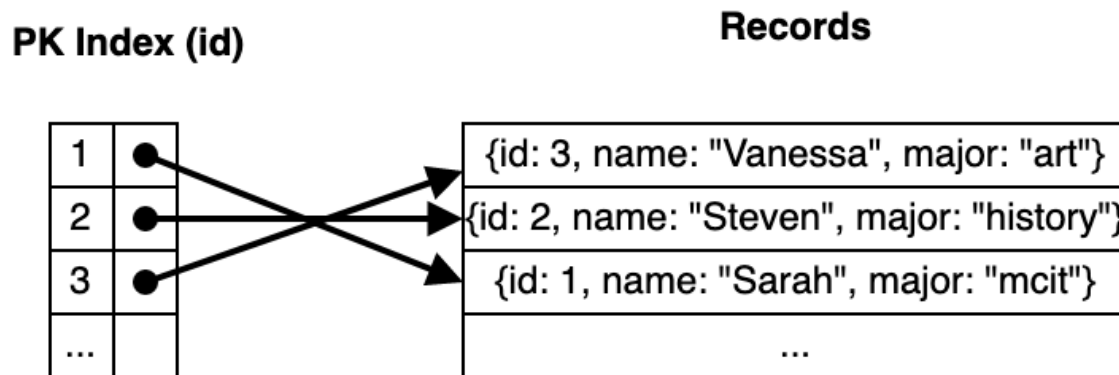


Primary Key

- Each record of a database normally has a unique identifier (filed/attribute)
- **Primary key:** an attribute that uniquely identifies a record
 - ID number, Penn ID, Social Security Number, etc.

Primary Key Index

- Associates each *primary key* value with a pointer to the actual record on disk



Primary Key: Caveats

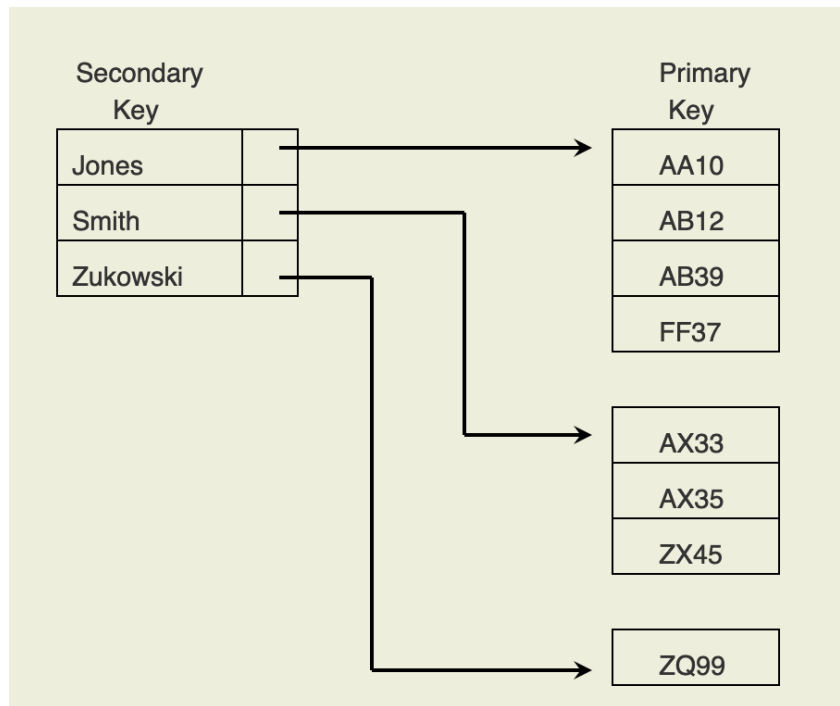
- Primary key often not known by the user of the database
- Primary key often not useful when searching for a record.
- Database searches often performed using attributes other than the primary key (name, age, major, salary, etc.)

Secondary Key

- **Secondary key:** a key field in a record where a particular key value might be **uplicated** in multiple records
 - such as salary, name, major, etc.
- Secondary key is more likely to be used by a user as a search key than is the record's primary key
 - Can't be used to uniquely identify a record, though

Secondary Key

- **Secondary key index:** associates a secondary key value with the primary key of each record having that secondary key value



Database indexing

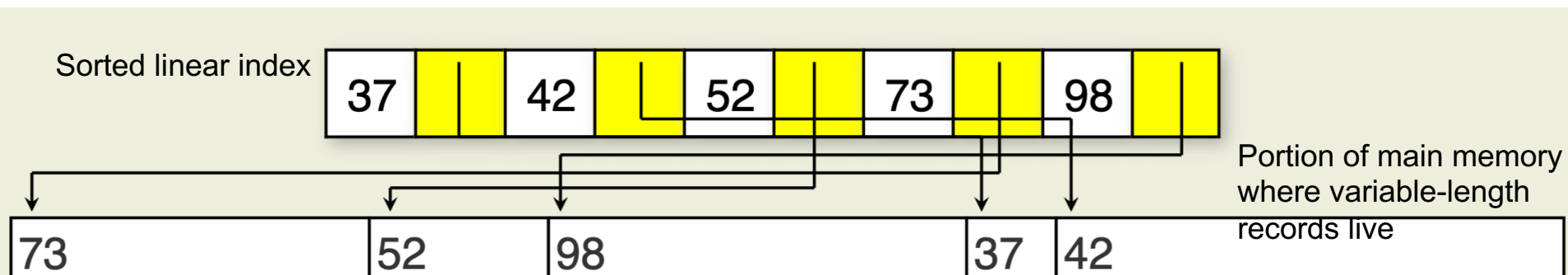
- Linear indexing
- Hash-based indexing
- Tree-based indexing

Index File

- **Index file:** a file whose records consist of key-value pairs where the pointers are referencing the complete records stored in another file

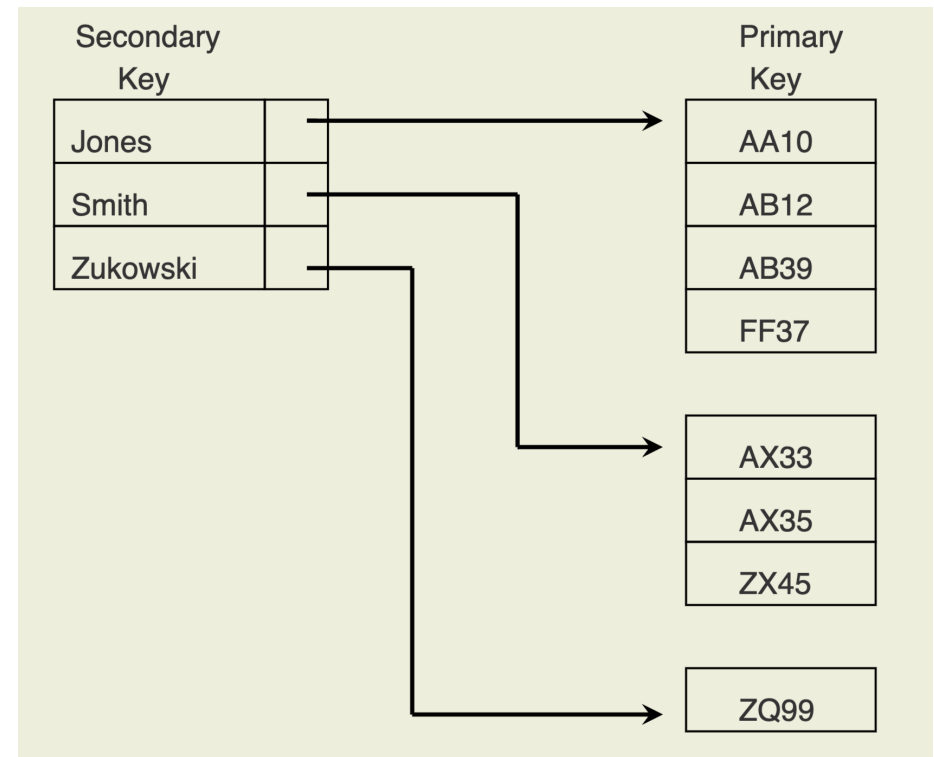
Linear indexing

- **Linear index:** *index file* organized as a sequence of *key-value pairs* where the *keys* are in sorted order and the pointers either
 - **Point to the position of the complete record on disk (pictured)**
 - Point to the position of the *primary key* in the primary key index
- Linear index amenable to binary search (efficient search)



Linear indexing

- **Linear index:** *index file* organized as a sequence of *key-value pairs* where the *keys* are in sorted order and the pointers either
 - Point to the position of the complete record on disk
 - **Point to the position of the primary key in the primary key index (pictured)**
- The secondary key index is called the **inverted list**



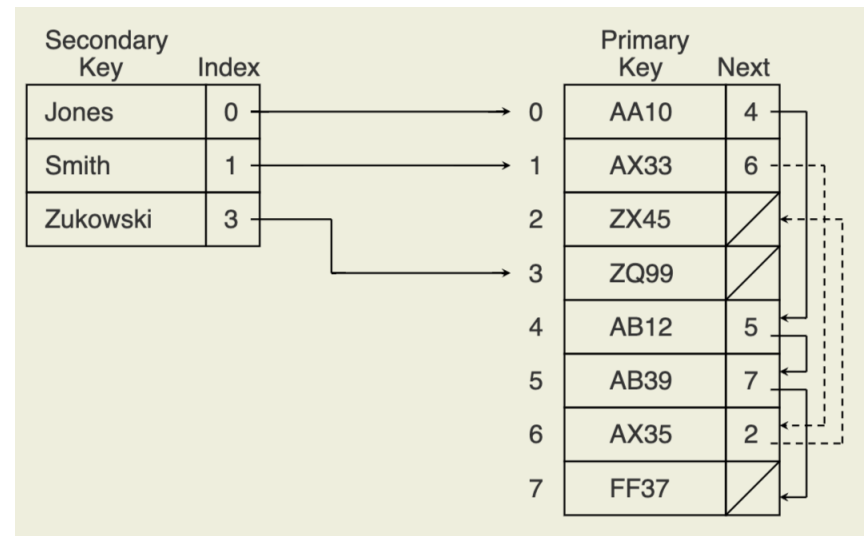
Linear indexing

- **Linear index:** *index file* organized as a sequence of *key-value pairs* where the *keys* are in sorted order and the pointers either

- Point to the position of the complete record on disk
- **Point to the position of the primary key in the primary key index (pictured)**

- The secondary key index is called the **inverted list**

A better implementation: keep primary keys in an array for better space efficiency.



Second-level index

- Linear Index as implemented so far is good when:
 - Keys are much smaller than records
 - The dataset is not too large
 - i.e. when the primary keys can all be kept in memory
- What if all primary keys can't be kept in memory?
 - For large databases, linear index array/LL cannot fit in memory
 - Leads to expensive search because of several disk accesses

Second-level index

- Solution:
 - **Second-level index** stored in main memory (array)
 - NOT NECESSARILY RELATED TO SECONDARY KEYS

2. Each cell here represents one memory block

1	2003	5894	10528
---	------	------	-------

1. Here, the value in cell *i* gives the minimum value in block *i*.

1	2001	2003	5688	5894	9942	10528	10984
---	------	------	------	------	------	-------	-------

3. Zoomed in view of block 1 from above, which is sorted for easy retrieval of elements in that range.

2003	2260	2592	2820	3000	3920	4160	4880	5550	5688
0	1	2	3	4	5	6	7	8	9

Second-level index

- Solution:
 - **Second-level index** stored in main memory (array)
 - NOT NECESSARILY RELATED TO SECONDARY KEYS
 - Index file stored across several blocks (on disk)
 - Second-level index stores the first key value in the corresponding disk block of the index file
 - Search requires 2 disk accesses: (1) load the block of the index file containing the key, (2) retrieve the record

Linear indexing: Drawback

- Insertion and deletion are expensive
 - All secondary indices must be updated: the entire contents of the array might be shifted
- Secondary key indexes contain duplicates: space expensive

Interlude: HW6

Goal of HW6

- Build a news aggregator that allows a user to view articles based on the terms contained inside of them
- Tasks:
 - Reasoning about ethics and social impact
 - Using an RSS feed to crawl webpages (!!!)
 - Calculating TF-IDF for a corpus of documents
 - Creating an inverted index for each term-based search
 - Generating a term list and incorporating autocomplete

JSoup, RSS, & HTML

- We're connecting to the internet and parsing documents hosted remotely.
- https://www.seas.upenn.edu/~cit5940/sample_rss_feed.xml

```
<rss version="2.0">
  <title>Hw6 Sample RSS Feed</title>
  <description>Sample RSS feed for CIT594 news aggregator</description>
  <link>http://localhost:8090/page1.html</link>
  <link>http://localhost:8090/page2.html</link>
  <link>http://localhost:8090/page3.html</link>
  <link>http://localhost:8090/page4.html</link>
  <link>http://localhost:8090/page5.html</link>
</rss>
```

- RSS, pictured above, is a markup language that allows you to specify a series of data sources.

JSoup, RSS, & HTML

- But wait: how do you connect to the internet?
 - Download JSoup and add it to your Eclipse/IntelliJ project (instructions included in writeup)
 - Then, to manipulate each **link** in an RSS feed:

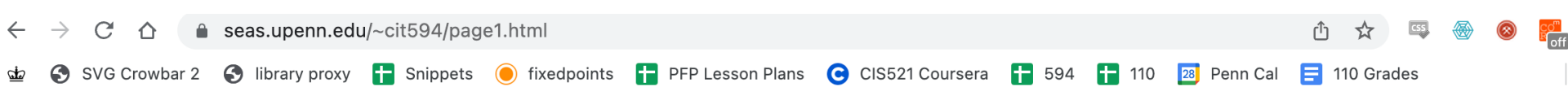
○ ○ ○

```
Document doc = Jsoup.connect(doc_url).get();
Elements linkElements = doc.getElementsByTag("link");
for (Element link : links) {
    String linkText = link.text();
    doSomething(linkText);
}
```

Each of these is a link to another page!

JSoup, RSS, & HTML

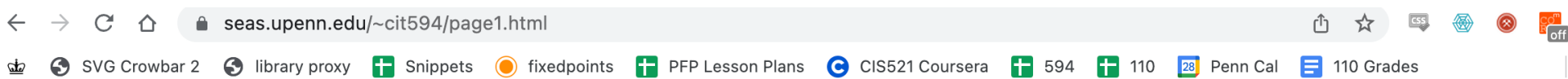
- But wait: how do you connect to the internet?
 - Download JSoup and add it to your project (instructions included in writeup)
 - You can manipulate each **link** in an RSS feed.
 - For each link, navigate to that page to find the list of terms contained in that page
 - Use JSoup the same way to manipulate HTML as the RSS document.



data structures: linear data structures Lists: arraylist, linkedlist, stacks, queues

JSoup, RSS, & HTML

- But wait: how do you connect to the internet?
 - Download JSoup and add it to your Eclipse project (instructions included in writeup)
 - You can manipulate each **link** in an RSS feed.
 - For each link, navigate to that page to find the list of terms contained in that page
 - Use JSoup the same way to manipulate HTML as the RSS document.



data structures: linear data structures Lists: arraylist, linkedlist, stacks, queues

seas.upenn.edu/~cit5940/page1.html



<data, structures, linear, data,
structures, lists, arraylist, linkedlist,
stacks, queues>

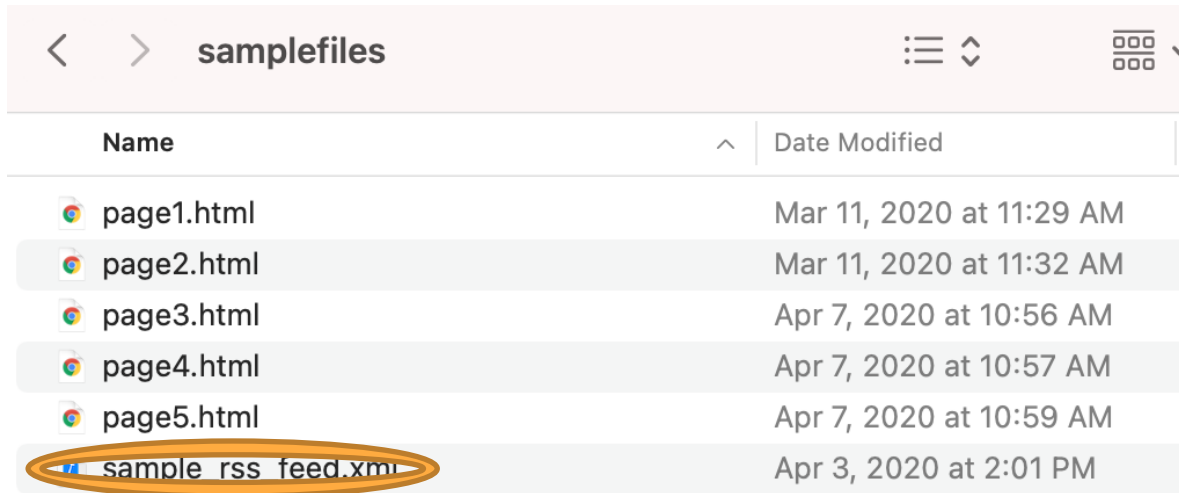
Testing and the internet

- You should write your own test cases as always, but how to host your own RSS feed for access?



<https://www.python.org/downloads/>

Writing your own testing files



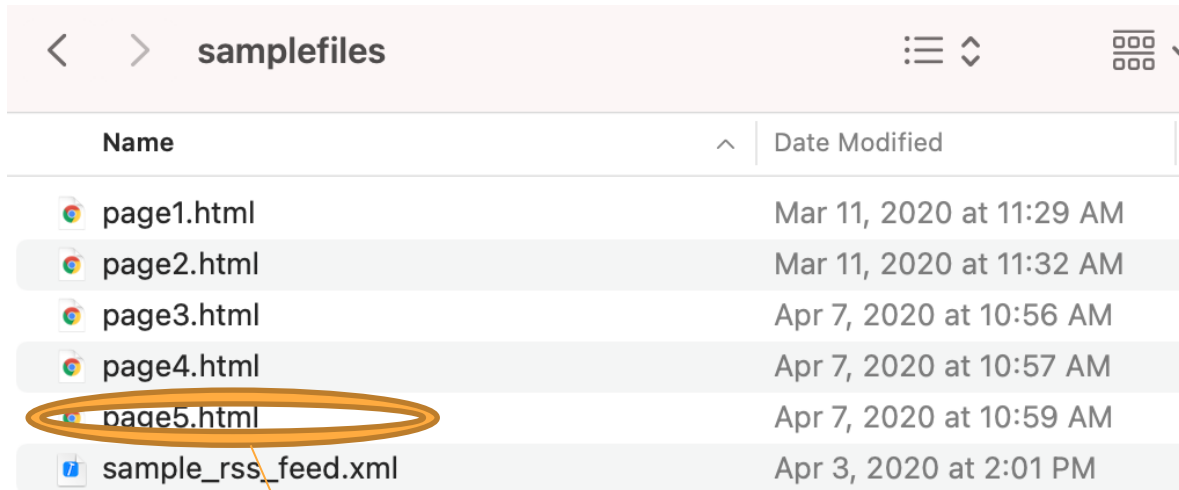
Name	Date Modified
page1.html	Mar 11, 2020 at 11:29 AM
page2.html	Mar 11, 2020 at 11:32 AM
page3.html	Apr 7, 2020 at 10:56 AM
page4.html	Apr 7, 2020 at 10:57 AM
page5.html	Apr 7, 2020 at 10:59 AM
sample_rss_feed.xml	Apr 3, 2020 at 2:01 PM

```
<rss version="2.0">  
  <title>Hw6 Sample RSS Feed</title>  
  <description>Sample RSS feed for CIT594 news aggregator</description>  
  <link>http://localhost:8090/page1.html</link>  
  <link>http://localhost:8090/page2.html</link>  
  <link>http://localhost:8090/page3.html</link>  
  <link>http://localhost:8090/page4.html</link>  
  <link>http://localhost:8090/page5.html</link>
```

```
</rss>
```

Keep the base URL for your pages <http://localhost:8090> and write whatever you want in the other .html files in your directory!

Writing your own testing files

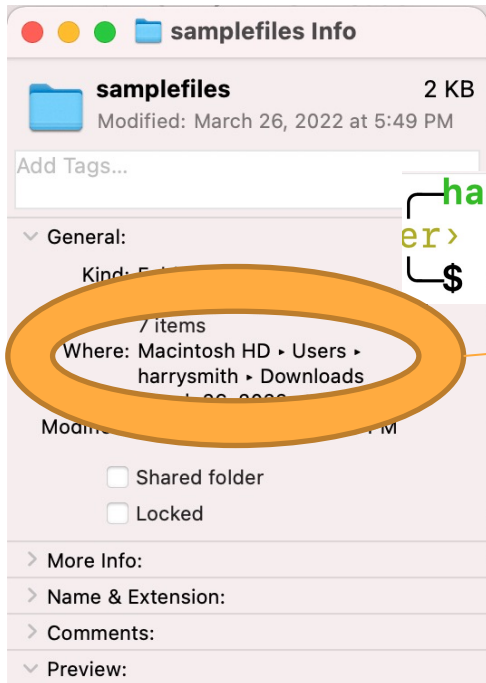


Name	Date Modified
page1.html	Mar 11, 2020 at 11:29 AM
page2.html	Mar 11, 2020 at 11:32 AM
page3.html	Apr 7, 2020 at 10:56 AM
page4.html	Apr 7, 2020 at 10:57 AM
page5.html	Apr 7, 2020 at 10:59 AM
sample_rss_feed.xml	Apr 3, 2020 at 2:01 PM

```
<!DOCTYPE html>
<html>
  <head>
    <title>hw6 feed5</title>
  </head>
  <body>
    Here's a silly little html file
  </body>
</html>
```

Put whatever text you want in the bodies of your custom pages for testing. This is page5.html

Using a terminal, navigate to the directory containing your sample files.



```
harrismith@Harrys-MBP ~/Documents/22sp/cis110/22sp <ruby-2.7.2>  
er>  
$ cd /Users/harrismith/Downloads/samplefiles
```

```
harrismith@Harrys-MBP ~/Downloads/samplefiles <ruby-2.7.2>  
$ ls  
page1.html      page3.html      page5.html  
page2.html      page4.html      sample_rss_feed.xml
```



Start a web server on port 8090

```
harrysmith@Harrys-MBP ~/Downloads/samplefiles <ruby-2.7.2>
```

```
$ python -m http.server 8090
```

← → ↻ 🏠 ⓘ localhost:8090/sample_rss_feed.xml

👑 🔄 SVG Crowbar 2 🔄 library proxy 🟢 Snippets 🟡 fixedpoints 🟢 PFP Lesson Plans

This XML file does not appear to have any style information associated with it. The document

```
▼ <rss version="2.0">
  <title>Hw6 Sample RSS Feed</title>
  <description>Sample RSS feed for CIT594 news aggregator</description>
  <link>http://localhost:8090/page1.html</link>
  <link>http://localhost:8090/page2.html</link>
  <link>http://localhost:8090/page3.html</link>
  <link>http://localhost:8090/page4.html</link>
  <link>http://localhost:8090/page5.html</link>
</rss>
```

TF-IDF: term frequency-inverse document frequency

- Term frequency: how often does a term appear in a particular document?
- Document frequency: how many documents does a particular term appear in?
- - $TF(t) = (\text{Number of times term } t \text{ appears in a document}) / (\text{Total number of terms in the document})$
- $IDF(t) = \log_e(\text{Total number of documents} / \text{Number of documents with term } t \text{ in it})$
- $TF-IDF(t) = TF * IDF$

TF-IDF: term frequency-inverse document frequency

- Term frequency: how often does a term appear in a particular document?
- Document frequency: how many documents does a particular term appear in?
- - $TF(t) = (\text{Number of times term } t \text{ appears in a document}) / (\text{Total number of terms in the document})$
- - $IDF(t) = \log_e(\text{Total number of documents} / \text{Number of documents with term } t \text{ in it})$
- - $TF-IDF(t) = TF * IDF$
- Check our understanding:
 - Term frequency of “the” in a typical English document? Document frequency in a typical English corpus?
 - In which dataset would an email where “cat” appears a ton have a higher TF-IDF score?
 - a dataset of emails between pet enthusiasts, or
 - A dataset of emails about rare fruits

This Assignment's Indices:

- The full records are, effectively, <Term, Document, TF-IDF>
- The first index you build looks up full records with the Document as the key
 - Useful for looking up the Terms that appear in a particular Document.
- The second index is **inverted**, mapping a Term to the Document it appears in
 - Useful for looking up in which Document a term had the highest (or lowest) TF-IDF

This Assignment's Indices:

- Are all maps!
- Quick trick for iteration over maps:
 - **Entry** objects are defined as Key, Value pairs for a particular map

```
Map<Integer, String> map = ...  
for (Map.Entry<Integer, String> entry : map.entrySet()) {  
    int k = entry.getKey();  
    String v = entry.getValue()  
}
```

Indexing Example: Facebook's Haystack

- Photo Storage infrastructure
- Disks are organized in volumes of fixed size
- Haystack index store consists of 2 files:
 - Haystack store (database)
 - Index file (used to rebuild the in-memory index)
- In-memory index used
- Each Haystack store manages multiple volumes
- Each haystack store has one in-memory index file
- Append-Only database

Facebook's Haystack

- Haystack Store: contains “needles”
- Each photo has a key and an alternate key

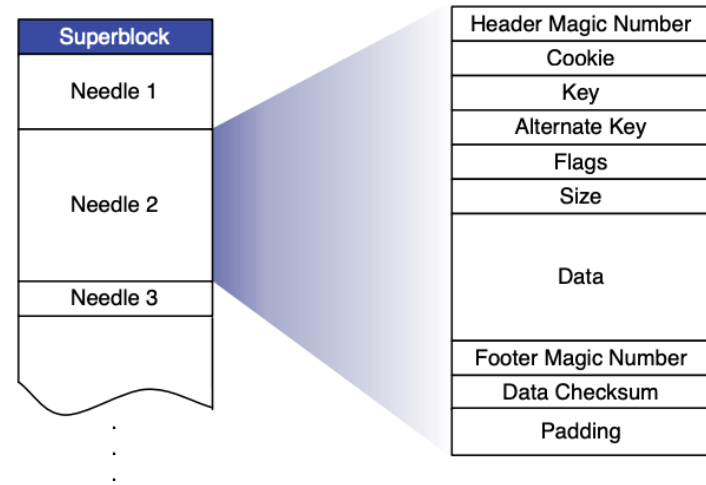


Figure 5: Layout of Haystack Store file

Field	Explanation
Header	Magic number used for recovery
Cookie	Random number to mitigate brute force lookups
Key	64-bit photo id
Alternate key	32-bit supplemental id
Flags	Signifies deleted status
Size	Data size
Data	The actual photo data
Footer	Magic number for recovery
Data Checksum	Used to check integrity
Padding	Total needle size is aligned to 8 bytes

Facebook's Haystack

- Needle:
 - Represents a photo stored in the Haystack
 - Uniquely identified by its <Offset, Key, Alternate Key, Cookie> tuple
 - Multiple needles can have the same key
 - Offset: the needle offset in the haystack store
 - Offset is stored in index (file and in-memory)

Facebook's Haystack

- In-memory index
 - Data structure that maps pairs of (key, alternate key) to the corresponding needle's flags, size, and offset
 - Key is the photo id
 - Alternate key is the photo's type. Each photo is scaled to four types/sizes
 - Google Sparsehash used (closed hashing + quadratic probing)

Facebook's Haystack

- In-memory index

hash (key, alternate key) →

location of needle in index

Alternate key: large, medium, small, thumbnail



Delete status

Needle 1 index record	Key	64-bit object key
	Alternate Key	32-bit object alternate key
	Flags	Currently unused
	Offset	Needle offset in the haystack store
	Size	Needle data size
Needle 2 index record		
Needle 3 index record		

Facebook's Haystack: Photo Read

- Exact match query
- Each request contains the photo's: logical volume id, key, alternate key, and cookie
- **Hash function** is used to find the photo in the in-memory index

Facebook's Haystack: Photo Read

- If photo is deleted (flag sets to delete) stop
- Else find the needle in the volume based on the offset, reads the entire needle, performs integrity checks and returns the image
- **One disk access for each request**

Facebook's Haystack: Photo Write

- Each request contains the logical volume id, key, alternate key, cookie, and data (photo)
- A new needle is created and appended (added at the end) to the Haystack
- A mapping for the new needle is added to the in-memory index

Facebook's Haystack: Photo Write

- Special case: Photo modification (e.g. after rotation)
- A Needle cannot be overwritten (append-only)
- A new needle is created with the same key and alternate key as the original needle
- The in-memory index is updated: offset is updated to match the new needle

Facebook's Haystack: Photo Delete

- Flag is set to “delete” in both in-memory index and Haystack store
- Requests to get deleted photos first check the in-memory flag and return errors if that flag is enabled