Introduction to Web Science: Midterm Exam

Dr. Nelson

Alexander Nwala

Thursday, March 26, 2015

Contents

Problem 1

How many yards did Kevin Jones rush for in the 2001 Virginia Tech - University of Virginia football game? What site/service/URL did you use to find this information? Is it a DL? If yes, why? If no, why not & what could be added to make it a DL? How did you discover the site/service/URL that provided the scores? Was that a DL? Briefly discuss the hierarchy of resources that helped you find these scores.

SOLUTION 1[1]

Definition/characteristics referenced: A Digital Library (DL) is a managed collection of information, with associated services which include search, where the information is stored in digital formats and accessible over a network.

Consider a DL which has the following characteristics and points associated with each characteristic:

1. Organized - 10 points
2. Groomed content - 10 points
3. Lots of metadata - 15 points
4. Structured changes - 10 points
5. Active preservation & curation - 45 points
6. DLs are generally more tightly controlled, and have a targeted customer set - 10 points

A non DL has the following characteristics:

1. A disorganized free-for-all
2. Very little metadata
3. Haphazard additions, deletions, modifications
4. No preservation strategy

Consider the following terms used:

1. **Data provider** - http://www.hokiesports.com/football/records/
2. **Discovery provider** - http://www.google.com

In the November 17, 2001 Virginia Tech vs. University of Virginia football game, Kevin Jones accomplished a rushing yards of 181 yards. I used Google (Discovery provider) to find this information at http://www.hokiesports.com/football/records/individual.html (Data provider).

Is the Data provider a DL? The answer to this question is neither Yes nor No. This is because while some services are not purely Digital Libraries since they may not possess some characteristics critical to be classified as DLs, they may still deserve some credit for DL characteristics which they possess. Consequently, I consider a weighted point systems as a fair means to grade services as DLs or not. The points associated with each criteria in the DL characteristics was chosen based on how important each element contributes to the definition.
For this reason, let us consider the Data provider as a DL:

1. **Organized - 10/10 points:** From the parent page it can be seen that records can be viewed in different ways: Scoring, Rushing, Individual records etc. This shows good structure.

Problem 1 continued on next page...

2. **Groomed content - 10/10 points:** The data is presented nicely in human-readable form.
3. **Lots of metadata - 0/15 points:** Not enough metadata is provided. For example information such as the count of spectators etc.
4. **Structured changes - 5/10 points:** The Data provider (at time of answering this question) did not have any mementos, so it was not possible to ascertain exactly how changes occur of over time.
5. **Active preservation & curation - 30/45 points:** Since the data provider is a record book, and has records as far back as the 1940s (http://www.hokiesports.com/football/records/punting.html), it is fair to say the data provider engages in active preservation.
6. **DLs are generally more tightly controlled, and have a targeted customer set - 10/10 points:** The data provider is responsible for managing these records and the records are available to the public.

In conclusion, based on the point system, the Data provider is 65% DL

How can the Data provider improve its status as a DL: The Data provider can improve its standing to be called a DL by addressing its weakness in the following areas:

1. **Lots of metadata:** Extensive meta data associated with relevant information regarding a game may be provided such as the location, spectators count, as well as game statistic.
2. **Structured changes:** Rules for making updates, deletions and additions should follow a definition. Also, changes such as updates, deletions and additions have to be documented. And the state saved before all such operations.
3. **Active preservation & curation:** Preservation can be implemented by saving the state of the data before any addition. The different versions of the data source may be labelled with a timestamp.

How was the data provider discovered: The data provider was discovered through Google.

Is Google a DL: Considering the same point system already visited, let us consider Google's status as a DL:

1. **Organized - 10/10 points:** The data provided and presented is organized.
2. **Groomed content - 10/10 points:** The data provided and presented is groomed and labelled.
3. **Lots of metadata - 5/15 points:** The search interface used to discover the data provider does not have a lot of metadata.
4. **Structured changes - 0/10 points:** Given that Google is not the source of the data, but merely a pointer to the data provider, it has no control over how changes occur.
5. **Active preservation & curation - 25/45 points:** Even though Google caches searches, since the goal of its search result it to provide as much fresh content as possible, active preservation is not considered significant.
6. **DLs are generally more tightly controlled, and have a targeted customer set - 0/10 points:** Google tightly controls its services. However, again, since Google is merely a pointer to the data sources, the content providers still influence Google's content.

Problem 1 continued on next page...

In conclusion, based on the point system, the Discovery provider is 50% DL

Consider the hierarchy of resources used to find these scores.

1. **Discovery:** http://www.google.com
2. **Discovery Query:** "How many yards did Kevin Jones rush for in the 2001 Virginia Tech - University of Virginia football game"

```
https://www.google.com/search?q=How+many+yards+did+Kevin+Jones
+rush+for+in+the+2001+Virginia+
Tech+-+University+of+Virginia+football
+game&oq=How+many+yards+did+Kevin+Jones+
rush+for+in+the+2001+Virginia+Tech+-+University+of+Virginia+
football+game&aqs=
chrome..69i57&sourceid=chrome&es_sm=122&ie=UTF-8
```

3. **Data provider:** http://www.hokiesports.com/football/records/individual.html
4. **Verification:**
   (a) **Discovery:** http://www.google.com
   (b) **Discovery Query:** 2001 Virginia Tech - University of Virginia football game
   (c) **Data provider:** http://en.wikipedia.org/wiki/Virginia%E2%80%93Virginia_Tech_football_rivalry

Problem 2

Install "Memento For Chrome". Install "Mink". Using a popular web site (i.e., one that we can expect to be archived), compare and contrast your experiences with both tools.

SOLUTION 2

I used http://www.arsenal.com to consider the experiences of "Memento For Chrome" and Mink. My goal was to go back in time as well as to see mementos of the selected site.

Comparison and contrast:

I consider Memento for Chrome's feature of travelling to the nearest archived date to the saved date useful especially when a site is highly archived. This could help a user do the date calculation.

However, I find Mink easier to use, more intuitive and more direct. Mink immediately exposes the count of mementos for a site.

I consider Mememto for Chrome more useful for multiple site time travel for a given date window, targetting densely archived sites.

I consider Mink more useful for single site time travel, especially for sparsely archived sites.

Therefore, considering my goal, I consider Mink as the better tool.

Problem 3

Summarize Chris Anderson's 2006 article (http://archive.wired.com/wired/archive/12.10/tail.html). Discuss a (non-Amazon) example of a live web service that encourages you to navigate the long tail.

SOLUTION 3

I listen to movie scores, which is not as popular as mainstream genres such as pop. So I use iTunes to explore the long tail: Whenever, I use iTunes, I rarely focus on the hits but rely on recommendations from users who liked the same music as me. I have discovered many songs I like based on the suggestion of others who explore the long tail just as me.

In the absence of online marketing, vendors of products such as music, movies, books and other media, were confronted with the problem of limited shelve space to hold their products. This limitation led to the rational decision to only hold popular products - hits, in order to increase the profit margin. This rationale for only holding hits was futher amplified by the fact that since the cost of production and space for any physical medium accounts into the cost. Similarly, in order to maintain profitability, it was wise to keep popular content in order to cover the cost production associated with the product (the cd or book). Consequently, in the absence of online marketing, the market for these products could be easily labelled as a hit-driven market: which is a market in which retailers only carry materials which can generate enough demand to earn their shelf space. Moreover, the demand for these products was confined to the geographic location of the store carrying the product. At the dawn of e-commerce, the limitations of space was been dramatically reduced - it almost cost nothing to hold digital media on a server. The consequence of this means the initial rationale of keeping only hits is no longer valid - misses (less popular media) are equally profitable as hits. This is because there are more misses than the popular hits. And the availability of misses to millions of consumers online means they are more profitable than hits. This is the central concept of the long tail.

The long tail emphasizes the fact that less popular media are equally or even more profitable (hence worth hosting for sale) than hits (head of demand curve) because of the large number of misses (long tail of demand curve).

Problem 4

Explain in detail how and why these are different:

http://mementoproxy.cs.odu.edu/wiki/timemap/link/http://en.wikipedia.org/wiki/DJ_Shadow
http://web.archive.org/web/timemap/link/http://en.wikipedia.org/wiki/DJ_Shadow

SOLUTION 4

`http://mementoproxy.cs.odu.edu/wiki/timemap/link/http://en.wikipedia.org/wiki/DJ_Shadow - labelled URI M`

`http://web.archive.org/web/timemap/link/http://en.wikipedia.org/wiki/DJ_Shadow - labelled URI A`

`Both URI M and URI A are timemaps - collection of mementos`

1. **Difference 1:** Dereferencing URI M, yields an XML representation (of a list of mementos) which was derived from the Wikipedia change history accessed by clicking the View history of the DJ Shadow's Wikipedia page.

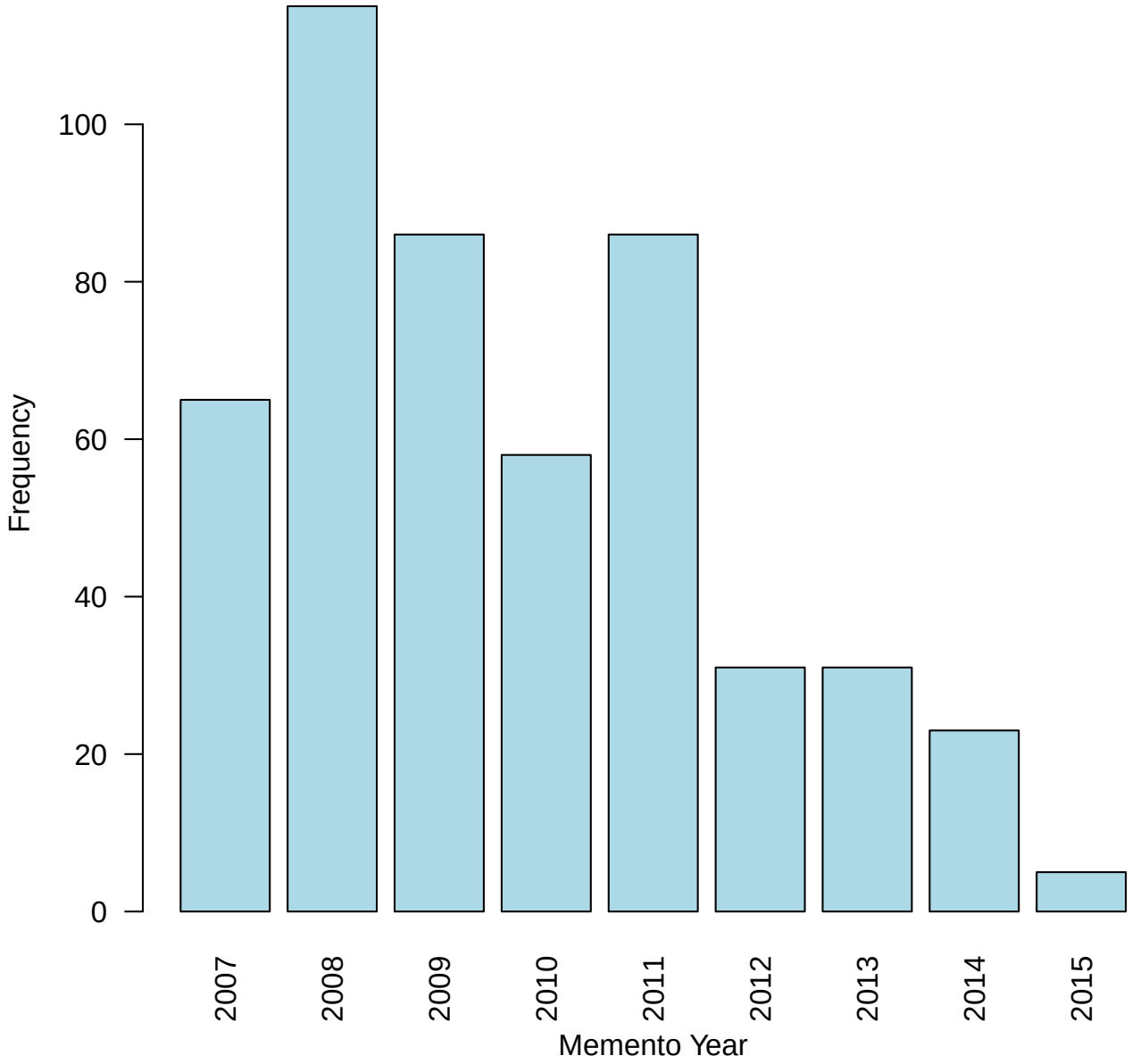
   Dereferencing URI A, yields an XML representation (of a list of mementos) which was derived from the Internet Archive's crawl of the DJ Shadow's Wikipedia page.

2. **Difference 2:** URI M has more mementos than URI A. This is no surprise because Wikipedia tracks the change history of its content more frequently than the Internet Archive. The frequency of the Internets Archive's crawl on a page could depend on the popularity of the page.

3. **Difference 3:** A quick observation may incorrectly lead to the observation that URI A has a older mementos (as far back as 2004) when compared to URI M - 2007. However, after visiting the change history page of the DJ Shadow Wikipedia page, I observed URI M was derived from the newest 500. Pagination could solve this problem. Consequently, suppose URI M implemented pagination, the following statement is valid: URI M has older mementos than URI A

4. **Difference 4:** As seen from Charts 1 and 2 (due to Listing 1.), URI M is more densely archived as compared to URI A

Chart 1: Distribution of memento year value for http://mementoproxy.cs.odu.edu/wiki/timemap/link/ http://en.wikipedia.org/wiki/DJ_Shadow



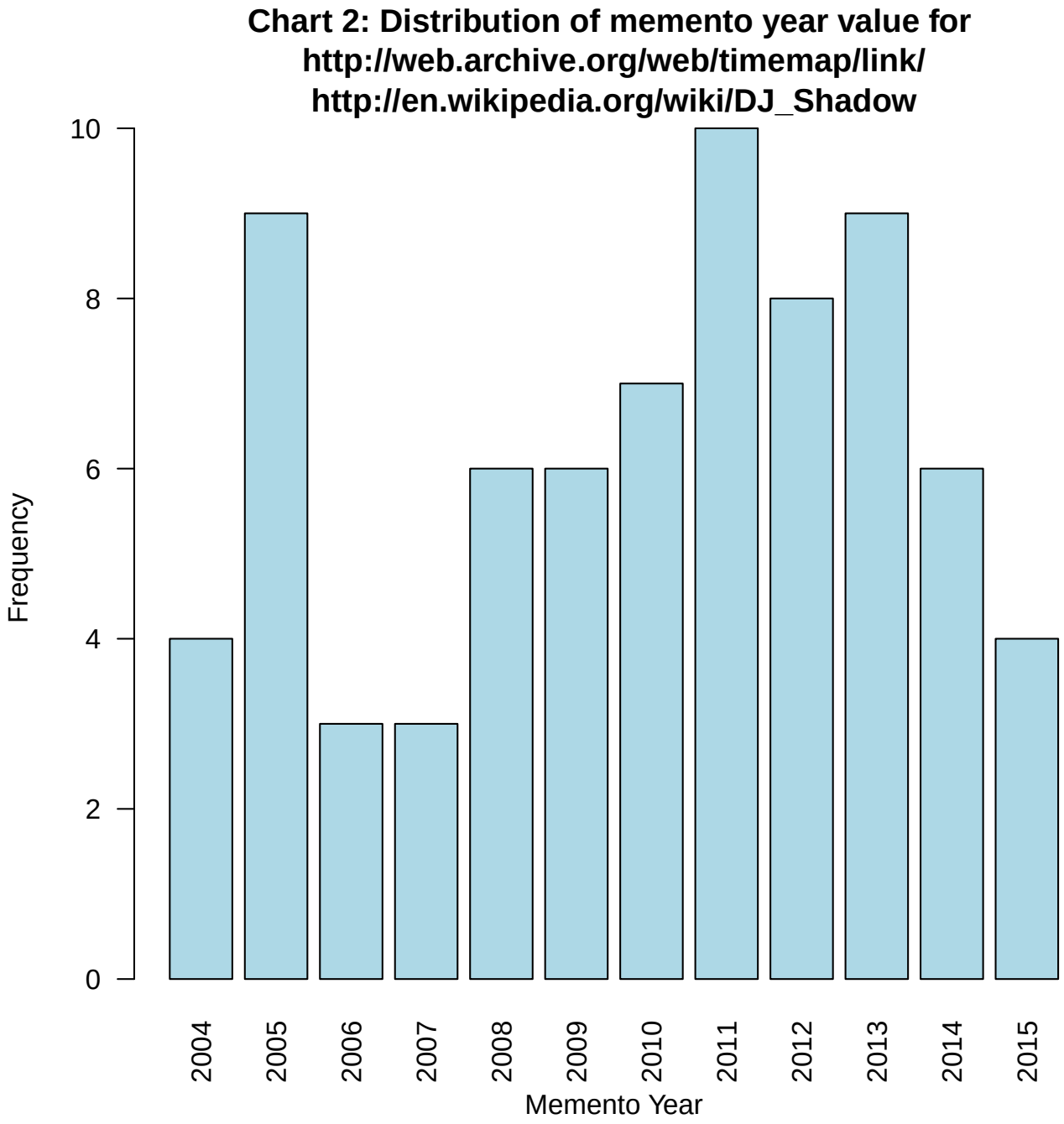


Chart 2: Distribution of memento year value for
http://web.archive.org/web/timemap/link/
http://en.wikipedia.org/wiki/DJ_Shadow
Frequency
0
2
4
6
8
10
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
Memento Year

Listing 1: Plot Memento Year Distribution

```
#!/usr/bin/env Rscript


rawYearInputWikipedia <- read.table('wikipediaTimemap.txt', header=T)
rawYearInputInternetArchive <- read.table('internetArchiveTimemap.txt', header=T)

yearBinsWikipedia <-
c(
2007,
2008,
2009,
2010,
2011,
2012,
2013,
2014,
2015
)

yearBinsInternetArchive <-
c(
2004,
2005,
2006,
2007,
2008,
2009,
2010,
2011,
2012,
2013,
2014,
2015
)



barplot( rawYearInputWikipedia$MementoYearFrequency, las=2, names=
    yearBinsWikipedia, xlab="Memento Year", ylab="Frequency", main="Chart 1:
    Distribution of memento year value for \nhttp://mementoproxy.cs.odu.edu/wiki/
    timemap/link/\nhttp://en.wikipedia.org/wiki/DJ_Shadow", col="lightblue")

barplot( rawYearInputInternetArchive$MementoYearFrequency, las=2, names=
    yearBinsInternetArchive, xlab="Memento Year", ylab="Frequency", main="Chart 2:
     Distribution of memento year value for \nhttp://web.archive.org/web/timemap/
    link/\nhttp://en.wikipedia.org/wiki/DJ_Shadow", col="lightblue")
```

Problem 5

Is the Twitter API RESTful? Why or why not? Does it follow the HATEOAS constraint? Why or why not? If the answer for either question is "no", explain in detail what Twitter could do to make it compliant with REST or HATEOAS.

SOLUTION 5[3][2][4]

Representational State Transfer (REST): is a software architecture style consisting of guidelines and best practices for creating scalable web services (Fielding, R. T.; Taylor, R. N. (2000). "Principled design of the modern Web architecture". pp. 407416. doi:10.1145/337180.337228).

Hypermedia as the Engine of Application State (HATEOAS): is a constraint of the REST application architecture.

Is the Twitter API RESTful?

Yes and No. The Twitter API is RESTful because it conforms to fundamental REST constraints as outlined. However, it does not conform to the HATEOAS constraint and maintains client credentials during sessions. For an API to be considered RESTful, it has to conform to some architectural recommendations. Consider these:

1. **Client-Cache-Stateless-Server Architecture:** Each client request contains all the information required to service the request. For example in order to make a request for a particular Twitter client's Friend list, one must include the authentication token with the URI to get the Friend list. The consequence of this is that the server does not store the client's credential.

   According to a REST constraint, a server should not store data from client requests, but the client can reuse response data, sent by the server, by storing it in a local cache. Twitter violates the foregoing contraints because in order to maintain the rate limit quota information, it is required that it maintains some data in order to check if a client has exceeded its quota for a particular request.

2. **Layered System Architecture:** Given the large number of Twitter users and the seamlessness of the service they provide, it is easy to see that even though they provide a single access point to their service, this is handled by multiple servers due to indirection for scalability.

3. **Uniform Interface Constraint - Resource identifiers, resource representations, self-descriptive messages, and HATEOAS:** Resources are identified by URIs such as: http://twitter.com/#!/jack/status/20

   Resources are represented in MIME types such as json.

   Also the twitter API provides self descriptive data to describe the meaning of responses.

Does it follow the HATEOAS constraint? Why or why not?
Only during pagination. But mostly No.

Firstly, Twitter complies to the HATEOAS constraint when the first page of large responses contains a link (transition) to the next response. However, for the most part, Twitter's json responses are terminal - they do not include all the permissive operations (through links) to transition to the next state. For example consider the excerpt below returned for a GET on https://api.twitter.com/1.1/lists/list.json?screen_name=twitterapi

```
[
  {
    "slug": "meetup-20100301",
    "name": "meetup-20100301",
    "created_at": "Sat Feb 27 21:39:24 +0000 2010",
    "uri": "/twitterapi/meetup-20100301",
    "subscriber_count": 147,
    "id_str": "8044403",
    ...
   }
]
```

It can be seen that the response returned does not contain links which point to the permissive operations at the current state.

How can Twitter comply to HATEOAS:

Twitter can comply to HATEOAS by including links its json responses. This can be done by considering the following implementation:
Most of the API consists of GET requests such as:

```
GET statuses/mentions_timeline
GET statuses/user_timeline
GET statuses/home_timeline
GET statuses/retweets_of_me
GET statuses/retweets/:id
GET statuses/show/:id
```

and POST requests such as:

```
POST statuses/destroy/:id
POST statuses/update
POST statuses/retweet/:id
POST statuses/update_with_media
```

Given that a GET/POST operation does not preclude another GET/POST operation, the response json should include a links to the other API GET/POST operations allowed. For example:

```
{
    .
    .
    .
    "APIlinks:
    [
        {
            "RequestMethod": "GET",
            "ResourceName": "statuses/user_timeline"
            "link": "https://api.twitter.com/1.1/statuses/user_timeline.json"
        },
```

```
            .
            .
            .
            {
                "RequestMethod": "POST",
                "ResourceName": "statuses/update"
                "link": "https://api.twitter.com/1.1/statuses/update.json"
            }

        ]
    }
```

Problem 6

Many people don't believe content negotiation occurs because they never see it. Document and discuss an example of content negotiation occuring on the live web (not Memento, and not a web site that you control).

SOLUTION 6

Content negotiation: URIs identify resources. Resources could have multiple representations. Therefore the client could specify what representation it seeks through the HTTP request header.

Whenever I visit www.google.com with my browser, content negotiation takes place, but the browser hides this. For example consider the header below sent by my browser for a request to google.com.

```
:authority:www.google.com
:method:GET
:path:/
:scheme:https
accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;
q=0.8
accept-encoding:gzip, deflate, sdch
accept-language:en-US,en;
q=0.8
cache-control:max-age=0
cookie:PREF=ID=8019a46955c921ae:U=417a0043cee92341:FF=0:LD=en:TM=1421020795:
LM=1423960432:GM=1:S=INtzC-nKgl2BShcU; NID=
67=QVj9Te8mOkB-zLCWwKU7wWK-vwyHjqxeMkgToeyBGchRiaP2wowGbW1a0bo-frE5nxumqqkOz
Wf28lDJEVvmAKwt75CU6N8ujTLa57h2CKq9d9OTiIBSCMGk6E8fALsSRZE03AATtYV4Wm18rLgck
PD2nBviJ2Uj
user-agent:Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_2) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/41.0.2272.104 Safari/537.36
x-client-data:CKa1yQEIlbbJAQiktskBCKm2yQEIwbbJAQjsiMoBCLSVygE=
```

As seen, my browser explicitly negotiates my language - "en-US" . In order to further illustrate content negotiation, I sent a custom header due to Listing 2. in other to switch my language to Deutsch. As seen in Figure 1. this was accomplished. This is due to the fact that the google resource has multiple representations, and I requested a different representation from my default.

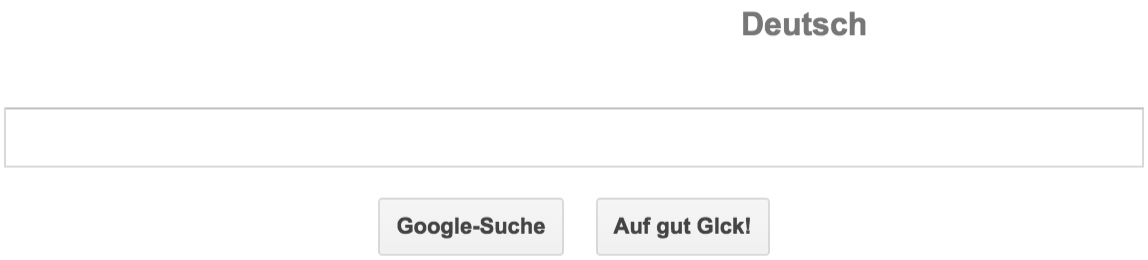


Figure 1: Deutsch Google Page

Listing 2: Content Negotiation Demo Through Custom HTTP Headers

```
import requests

r=requests.get("http://www.google.com/", headers={"Accept":"text/html,application/
    xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8", "Accept-Encoding":"gzip,
    deflate, sdch", "Accept-Language":"de;q=0.8", "Host":"www.google.com", "User-Agent"
    :"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
    /41.0.2272.89 Safari/537.36"});
#print r.headers
print r.text.encode('ascii', 'ignore')
```

Problem 7

Find an example of URI aliasing (as per the W3C web architecture) won the live web (i.e., find a "real" example). Why is URI aliasing bad? What could the web admininstrators do to "fix" the aliasing (other than "don't do it")?

SOLUTION 7

URI aliases: Multiple different URIs that identify the same resource

An example of URI alias: www.1channel.ch and www.primewire.ag point to the same resource:

```
$ curl -I www.1channel.ch
HTTP/1.1 301 Moved Permanently
...
Location: http://www.primewire.ag
Server: cloudflare-nginx
...

$ curl -I www.primewire.ag
HTTP/1.1 200 OK
...
Server: cloudflare-nginx
```

. . .

URI aliasing is bad because:

1. **Reason 1:** A major problem with URI aliasing is the fact that it diminishes the value of a network. According to Metcalfe's Principle, the value of a resource in a network is proportional to the count and value of other resources in its network neighborhood which link to it.

   The problem with URI aliasing is that if for instance in a network, a resource has three URIs, then the value of the resource is split to a triple due to the three subsets of the set of resources which point to the three different URIs for the same resource. Consequently, the value of the resource is reduced and by extension the value of the overall network is reduced. This could be of consequence when the page rank of a resource is to be calculated.

2. **Reason 2:** It violates the a good practice regarding Identification as specified by W3C: A resource is identified by a URI, and while a resource could have multiple representations, the URI must not identify the representations but the resource.

What could the web admininstrators do to "fix" the aliasing (other than "don't do it")?

URI aliasing can be avoided by administrators by implementing (server side) a permanent redirect (301) on the alias to point to the main URI. It can also be fixed with a mod_rewrite entry in the configuration of the server (Apache server).

Problem 8

Compare and contrast the five crawling algorithms in the Pant, Srinivasan, and Menczer article. If you were writing a crawler to find Va Tech box scores, which algorithm would you implement?

SOLUTION 8

Summarized Comparison: All 5 algorithms are variations of the best-first scheme: this means the frontier is implemented as a priority queue in which unvisited URLs are scored based on some determined rule.

Summarize Contrast: All 5 algorithms have different heuristics which determines how they compute the scores of unvisited URLs - this determines the priority.

Head to head comparison/constrast:

1. **Heuristic algorithms: Naive Best-First Crawler vs. SharkSearch:** In Naive Best-First, the unvisited URL's (in the frontier) score is computed by the cosine similarity of a page and a user defined query. The cosine similarity calculation is facilitated by the fact that pages are represented as weighted vectors - weights derived from word frequency. When new URLs are added the frontier, the priority is still maintained. Unlike Naive Best-First, SharkSearch scores the unvisited URLs in the frontier differently: The total score is not simply calculated by locally available information. The anchor text surrounding links and an inherited score from the ancestor pages (pages which are internal nodes to the current link) contribute to the final score. The final score is calculated by:

Problem 8 continued on next page. . .

```
score(url) = Gamma  inherited(url) +
             (1 - Gamma)  neighborhood(url)
```

The only similarity SharkSearch has with Naive Best-First search is the fact that as can be seen in the equation above, the similarity measures in inherited(url) and neighborbood(url) is based on cosine similarity. Another, important disparity between Naive Best-First and SharkSearch is the fact that unlike Naive Best-First, SharkSearch seeks to stop crawling along a path when it begins to see unimportant pages. This is due to a user determined depth value.

2. **Learning Algorithms: Focused Crawler vs. Context Focused Crawlers vs. InfoSpiders:** Even though all of these algorithms are sophisticated in that they are learning algorithms which seek to self tune during the crawl, they differ in the following ways:

   In the Focused Crawler algorithm, we classify or label pages with a user provided taxonomy (list of classes). This is kicked-off by a user input of a class such as "Facebook" and an associated list of URLs which belong to the class "Facebook." Given these input, the crawler labels pages with labels from the taxonomy. This is done by finding the highest conditional probability that a page p belongs to a taxonomy c. In order words, Focused Crawlers use a Bayesian classifier. Just like Focused Crawlers, Context Focused Crawlers use a Bayesian classifier. But the Bayesian classifier of Context Focused Crawlers take into account the estimated link distance between a crawled page and the relevant pages. This is very useful because it has the effect of consider words outside the vicinity of the current page. This is facilitated by a context multilayered graph associated with each page.

   The foregoing two learning crawling algorithms are similar to InfoSpider because InfoSpiders are topic driven, that is they search for pages which are similar to a user specified topic. However, InfoSpiders use neural networks to decide the scores of unvisited URLs. In the neural network each input unit is given a tuple consisting of a keyword and the frequency in which the keyword appears within the vicinity of the current link. This can be seen as a weight. The neural network also consists of a single output unit which produces an estimated similarity to the parent page.

Given the problem of writing a crawler to find Va Tech box scores, I would implement the Context Focused algorithm for the following reasons:

For this problem, heterogenous page content is out of the question, thus disqualifying the Naive Best-First algorithm. Implementing a Context Focused algorithm provides the flexibility of providing a topic "VA scores" and an initial list of seed URLs. This means the algorithm can find pages similar to those provided and self correct once it begins to pick unimportant pages.

Problem 9

Construct a 15 node graph that illustrates how a node with high in degree does not have high PageRank. Compute 3 iterations of PageRank for all 15 nodes. Show the PageRank values for each of the nodes at each iteration. Either show your work (if computed by hand or your own program) or cite the program you used to do your computations.

SOLUTION 9

Listing 3: Calculate PageRank

```
#calculate PageRank
```

```
import networkx as nx
dg=nx.DiGraph()
dg.add_node(11)

dg.add_edges_from([(0,3), (3,0), (2,1), (2,0), (5,0), (4,0), (4,2), (10,4), (9,4),
    (8,4), (7,4), (6,4), (5,4), (4,5), (10,0), (9,0), (8,4)])
pr = nx.pagerank(dg, alpha=0.85, max_iter=100)
print pr
```

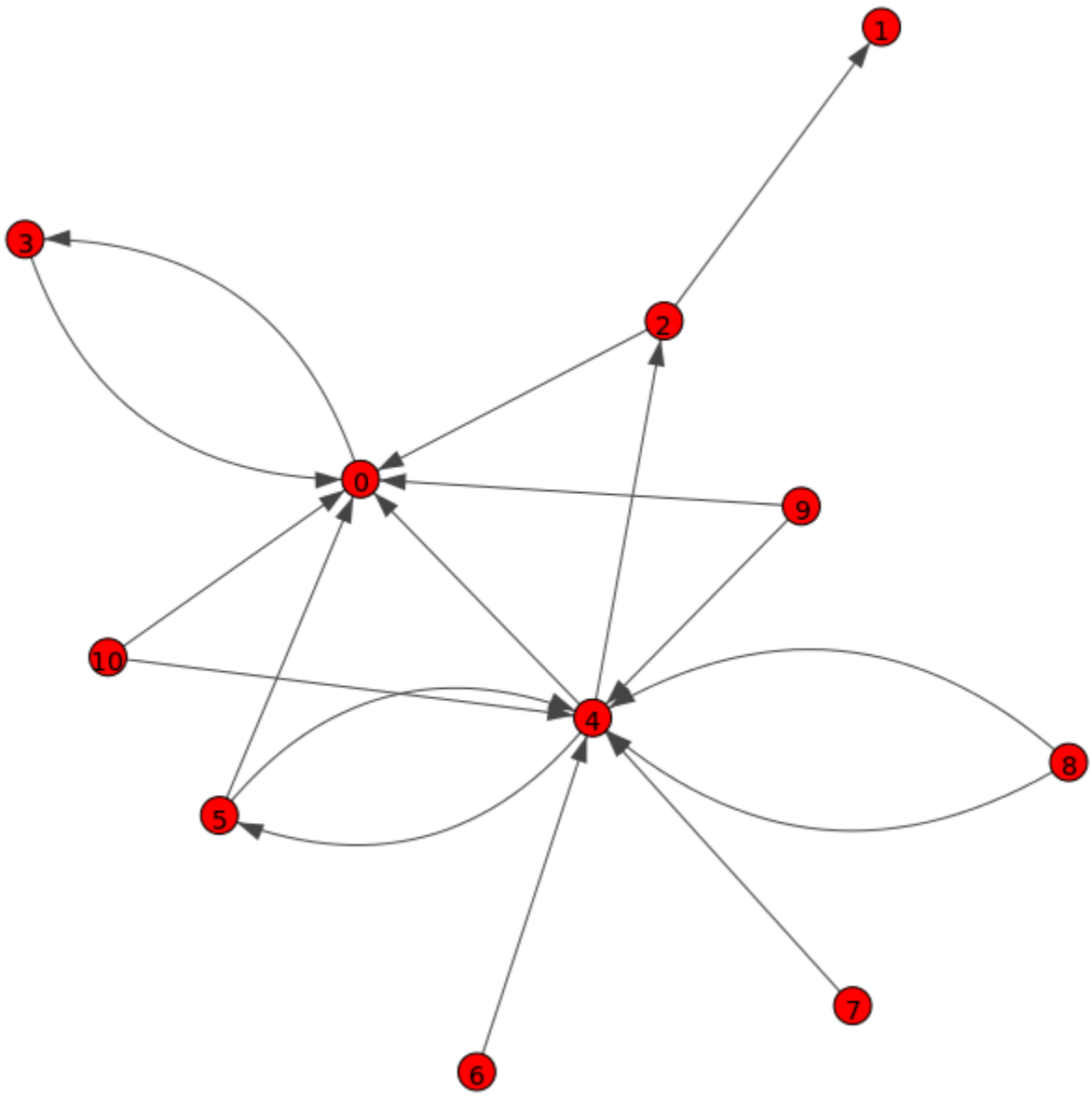


Figure 2: Arbitrary Graph

Consider the graph in Figure 2: Consider the table generated by calculating the PageRank algorithm (Listing 3.) for 3 iterations.

As seen from the Tables even though Node 4 has the highest in-degree it does not have a proportionate Page rank due to the fact that most of its in-degree arises from unimportant (0 in-degree; 6, 7, 8, 9) nodes in the

Table 1: PageRank For Iteration 1

Node	PageRank	InDegree	OutDegree
0	0.2604	6	1
1	0.0597	1	0
2	0.0479	1	2
3	0.0951	1	1
4	0.3430	7	3
5	0.0479	1	2
6	0.0243	0	1
7	0.0243	0	1
8	0.0243	0	2
9	0.0243	0	2
10	0.0243	0	2

Table 2: PageRank For Iteration 2

Node	PageRank	InDegree	OutDegree
0	0.2579	6	1
1	0.0388	1	0
2	0.1156	1	2
3	0.2398	1	1
4	0.1214	7	3
5	0.1156	1	2
6	0.0184	0	1
7	0.0184	0	1
8	0.0184	0	2
9	0.0184	0	2
10	0.0184	0	2

Table 3: PageRank For Final Iteration

Node	PageRank	InDegree	OutDegree
0	0.3687	6	1
1	0.0657	1	0
2	0.0509	1	2
3	0.2357	1	1
4	0.1284	7	3
5	0.0509	1	2
6	0.0165	0	1
7	0.0165	0	1
8	0.0165	0	2
9	0.0165	0	2
10	0.0165	0	2

Problem 9 continued on next page...

Table 4: PageRank For Iteration 3

Node	PageRank	InDegree	OutDegree
0	0.3703	6	1
1	0.0333	1	0
2	0.0408	1	2
3	0.3308	1	1
4	0.0877	7	3
5	0.0408	1	2
6	0.0159	0	1
7	0.0159	0	1
8	0.0159	0	2
9	0.0159	0	2
10	0.0159	0	2

network. This trend continues into the final iteration of the algorithm as seen in Table 4.

References

[1] An Introduction to Digital Libraries. http://www.cs.cornell.edu/wya/diglib/ms1999/Chapter1.html. Accessed: 2015-03-19.

[2] API Design: Honing in on HATEOAS. https://blog.apigee.com/detail/api_design_honing_in_on_hateoas. Accessed: 2015-03-19.

[3] REST, HATEOAS, and Follow Your Nose. http://ws-dl.blogspot.com/2013/11/2013-11-19-rest-hateoas-and-follow-your.html. Accessed: 2015-03-19.

[4] Understanding HATEOAS. https://spring.io/understanding/HATEOAS. Accessed: 2015-03-19.