

Extracting relational data from Wikipedia

Chris Perivolaropoulos

Tuesday 9q May 2016

Contents

I	Περίληψη	2
II	Εισαγωγή	4
1	START	5
III	Wikipediabase	7
2	Λειτουργικότητα	9
3	Getting started	15
4	Αρχιτεκτονική	17
5	Το μοντέλο provider/acquirer	28
6	Testing	29
7	Συνώνυμα	32
8	Databases and data sources	36
9	Date parser	40
10	Παραρτήματα	43
IV	WikipediaMirror	49
11	Mediawiki stack overview	51

12	Setting up	55
13	Mediawiki extensions	57
14	Φορτώνοντας τα mediawiki dumps	59
15	Εργαλεία	60
16	Αυτοματισμός	64
17	Επιδόσεις	69
18	Παραρτήματα	71
V	References	88

Part I

Περίληψη

MiT InfoLab's START (SynTactic Analysis using Reversible Transformations) είναι το πρώτο διαδικτυακό σύστημα παγκοσμίως που βασίζεται σε ερώτηση και απάντηση. Για την πρόσβαση σε περισσότερες πηγές δεδομένων χρησιμοποιεί το Omnibase, μια "εικονική βάση δεδομένων" που παρέχει πρόσβαση σε πολλαπλές πηγές στο διαδίκτυο. Αναπτύξαμε Wikipedia-Base για την παροχή παρόμοιου τρόπου δεδομένων όπως το Omnibase για το START. Με στόχο να αποκτηθεί για να αποκτηθεί πρόσβαση σε μη δομημένες και ημιδομημένες πληροφορίες στη Wikipedia. Στο πλαίσιο αυτού του στόχου επίσης δημιουργήσαμε το wikipedia-mirror, ένα πρόγραμμα που δημιουργεί κλώνους της Wikipedia που μπορεί να τρέχουν τοπικά, με αποτέλεσμα να παρέχει έλεγχο και απεριόριστη πρόσβαση στο σύνολο των δεδομένων της wikipedia χωρίς να εξαρτάται ή να καταχράται το wikipedia.org.

MiT InfoLab's START (SynTactic Analysis using Reversible Transformations) is the world's first web-based question answering system. For accessing most data sources it takes advantage of Omnibase, the "*virtual database*" providing uniform access to multiple sources on the web. We developed WikipediaBase to provide an Omnibase-like way for START to access unstructured and semi-structured information in Wikipedia. As part of this goal we also created wikipedia-mirror, a program to create Wikipedia clones that can be run locally, to provide control and unrestricted access to the wikipedia data set without depending on or abusing wikipedia.org.

Part II

Εισαγωγή

Chapter 1

START

The START Natural Language System είναι ένα σύστημα λογισμικού που έχει σχεδιαστεί για να απαντά σε ερωτήσεις που τίθενται σε αυτό σε φυσική γλώσσα. Το START σαρώνει τις εισερχόμενες ερωτήσεις, τις ταυτίζει με τα ερωτήματα που δημιουργούνται από τα parse trees, τις αναλύει σύμφωνα με την γνωστική του βάση και τελικά παρουσιάζει τα κατάλληλα τμήματα πληροφοριών για το χρήστη. Με τον τρόπο αυτό, το START παρέχει στους ανεκπαιδευτους χρήστες γρήγορη πρόσβαση σε γνώση που σε πολλές περιπτώσεις μπορεί να είναι χρονοβόρα ακόμα και για τον ειδικό να την αποκτήσει

Το START (SynTactic Analysis using Reversible Transformations) δημιουργήθηκε από τον Dr Boris Katz στο Artificial Intelligence Laboratory του MIT. Επί του παρόντος, το σύστημα υφίσταται περαιτέρω ανάπτυξη από το InfoLab Group, με επικεφαλής τον Dr Boris Katz. Το START για πρώτη φορά συνδέθηκε με το World Wide Web το Δεκέμβριο του 199, και με τις διάφορες μορφές του έχει μέχρι σήμερα απαντήσει σε εκατομμύρια ερωτήσεις από χρήστες σε όλο τον κόσμο.

Μια βασική τεχνική που ονομάζεται "natural language annotation" βοηθά το START να συνδέσει την αναζητούμενη πληροφορία με τις πηγές πληροφόρησης. Αυτή η τεχνική χρησιμοποιεί προτάσεις και φράσεις της φυσικής γλώσσας ως περιγραφές περιεχομένων που σχετίζονται με τα τμήματα πληροφοριών σε διάφορες υποδιαίρεσεις. Ένα τμήμα των πληροφοριών ανακτάται όταν ο σχολιασμός του ταιριάζει με την εισαγόμενη ερώτηση. Αυτή η μέθοδος επιτρέπει στο START να χειριστεί μεγάλη ποικιλία μέσων, συμπεριλαμβανομένων των κειμένων, διαγραμμάτων, εικόνων, βίντεο και ήχων, συνόλων δεδομένων, ιστοσελίδων και άλλων.

Η επεξεργασία της φυσικής γλώσσας του START αποτελείται από δύο ενότητες που μοιράζονται την ίδια γραμματική. Η μονάδα κατανόησης

αναλύει το αγγλικό κείμενο και παράγει μια γνωστική βάση που κωδικοποιεί πληροφορίες που βρίσκονται στο κείμενο. Λαμβάνοντας υπόψη ένα κατάλληλο τμήμα της βάσης δεδομένων, η μονάδα παραγωγής γλώσσας παράγει αγγλικές προτάσεις. Αυτές οι ενότητες σε συνδυασμό με την τεχνική της φυσικής γλώσσας σχολιασμού, δίνουν την δυνατότητα της παραγωγής φυσικής γλώσσας σε επίπεδο προτάσεων με χρήση σε υπηρεσίες πρόσβασης πληροφοριών διαφόρων πολυμέσων.

Omnibase είναι μια "εικονική" βάση δεδομένων που παρέχει μια ενιαία διεπαφή για πολλαπλές πηγές γνώσης στο Web, ικανή να εκτελεί τα δομημένα ερωτήματα που παράγονται από το START. Το Omnibase αναπτύχθηκε για πρώτη φορά το 2002, περίπου το ίδιο χρονικό διάστημα που η wikipedia έκανε την πρώτη της εμφάνιση (2001).

Η διαδικτυακή εγκυκλοπαίδεια Wikipedia είναι μια τεράστια, συνεχώς εξελισσόμενο δίκτυο από πλούσιες αλληλένδετες πληροφορίες σε μορφή κειμένου. Στην αυξανόμενη κοινότητα των νέων ερευνητών και προγραμματιστών είναι μια συνεχώς εξελισσόμενη πηγή χειροκίνητα οριζόμενων εννοιών και σημασιολογικών σχέσεων. Αποτελεί έναν απαράμιλλο και σε μεγάλο βαθμό ανεκμετάλλευτο πόρο για την επεξεργασία φυσικής γλώσσας προγραμματισμού, διαχείριση της γνώσης, εξόρυξη δεδομένων, και διάφορους άλλους τομείς έρευνας. Είναι το προϊόν της συνεργασίας εκατομμυρίων ανθρώπων. Η Wikipedia βασίζεται στο σύστημα wiki, μια κατηγορία ιστοσελίδων που επιτρέπουν την συνεργατική τροποποίηση του περιεχομένου.

Λόγω της πολυπλοκότητας και την ιδιαίτερα αδόμητη φύση της wikipedia αντί του omnibase backend START χρησιμοποιήσαμε μια ξεχωριστή υπηρεσία, το WikipediaBase, που αποτελεί και το αντικείμενο της παρούσας διατριβής. Επίσης, για να αποφευχθεί ο κορεσμός του wikiedpedia.org χρησιμοποιήσαμε το wikipedia - mirror δημιουργώντας έναν κλώνο της wikipedia για την Wikipedi-aBase χρησιμοποιώ.

Part III

Wikipediabase

Η WikipediaBase είναι ένα backend για το START υπεύθυνη για την παροχή πρόσβασης σε πληροφορίες που σχετίζονται με την wikipedia. Μιμείται την διεπαφή API που προέρχεται από το Omnibase. Η Wikipediabase έχει ξαναγραφεί δυο φορές. Η αρχική έκδοση ήταν γραμμένη σε Java. Στη συνέχεια ξαναγραφτηκε σε Ruby διατηρώντας την αρχική αρχιτεκτονική και το σχεδιασμό και τώρα ξαναγράφεται σε python με νέο σχεδιασμό και αρχιτεκτονική.

Υπάρχουν δύο βασικοί λόγοι για αυτό: Η Python διδάσκεται ως προπτυχιακό και μεταπτυχιακό μάθημα στο MIT , και ως εκ τούτου , μια βάση κώδικα σε Python θα κάνει την έναρξη των νέων φοιτητών του MIT ομαλότερη. Το πιο σημαντικό όμως είναι ότι ενώ ο αρχικός σχεδιασμός του προηγούμενου WikipediaBase θα έπρεπε να ήταν επαρκής μεγάλωσε σε τέτοιο σημείο όπου ο κώδικας ήταν ad-hoc και δύσκολα να κατανοηθεί, λόγος για την επέκταση

Η εφαρμογή python αρχικά γράφτηκε από τον Χρήστο Περιβολαρόπουλο σε στενή συνεργασία με την Dr Sue Felshin και τελικά παραδόθηκε στους Sue Felshin , Alvaro Morales και τον Michael Silver. Αργότερα και άλλοι φοιτητές έχουν ενταχθεί στο έργο.

Chapter 2

Λειτουργικότητα

Στη WikipediaBase , το καθέ (υποστηριζόμενο) Wikipedia infobox ορίζεται ως class, και κάθε μεταβλητή στο infobox ορίζεται ως ένα χαρακτηριστικό της κάθε class. Όλα τα αντικείμενα της WikipediaBase ανήκουν κληρονομικά στην υπερκλάση wikibase-term, η οποία υποστηρίζει τα χαρακτηριστικά IMAGE-DATA, SHORT-ARTICLE, URL, COORDINATES, PROPER, και NUMBER.

Οι εντολές της WikipediaBase και οι τιμές επιστροφής τους χρησιμοποιούν κωδικοποίηση σε s-expressions. Η WikipediaBase παρέχει τις ακόλουθες λειτουργίες:

1. get

Δεδομένης μιας class, ενός ονόματος αντικειμένου, και ενός τυποποιημένου χαρακτηριστικού, δηλαδή ενός χαρακτηριστικού με typecode. Έγκυρα typecodes χαρακτηριστικών είναι :code (για ένα attribute όνομα όπως στο infobox wiki markup) και :rendered (για ένα attribute όνομα στο rendered form από το infobox).

(a) Types

Τα Scripts πρέπει να επιστρέφουν μια λίστα από τυποποιημένες τιμές, δηλαδή ένα ζευγάρι τιμής - typecode. Έγκυρα typecodes είναι:

i. :HTML

Μια συμβολοσειρά προσαρμοσμένη για rendering σαν paragraph level HTML. Η συμβολοσειρά πρέπει να είναι escaped για lisp, εννοώντας quoted, και με double quotes και backslashes escaped με backslashes. Η συμβολοσειρά δεν απαιτείται να περιέχει HTML κώδικες. Για παράδειγμα:

```

(get "wikipedia-sea" "Black_Sea" (:code "
  AREA"))
=> ((:html "436,402_km2_(168,500_sq_mi)"))

(get "wikipedia-president" "Bill_Clinton" (:
  code "SUCCESSOR"))
=> ((:html "George_W._Bush"))

(get "wikipedia-president" "Bill_Clinton" (:
  rendered "Succeeded_by"))
=> ((:html "George_W._Bush"))

```

ii. :YYYYMMDD

Οι αναλυμένες ημερομηνίες αντιπροσωπεύονται σαν αριθμοί, χρησιμοποιώντας τον τύπο YYYYMMDD με αρνητικούς αριθμούς αντιπροσωπεύονται οι πχ ημερομηνίες.
(Οι μη αναλυμένες ημερομηνίες αντιπροσωπεύονται σαν HTML strings χρησιμοποιώντας το :HTML typecode.)

```

(get "wikipedia-sea" "Black_Sea" (:code "
  AREA"))
=> ((:html "436,402_km2_(168,500_sq_mi)"))

(get "wikipedia-president" "Bill_Clinton" (:
  code "SUCCESSOR"))
=> ((:html "George_W._Bush"))

(get "wikipedia-president" "Bill_Clinton" (:
  rendered "Succeeded_by"))
=> ((:html "George_W._Bush"))

```

iii. :CALCULATED

Το Typecode για χαρακτηριστικά υπολογισμένα από την με βάση χαρακτηριστικά του άρθρου, πχ., GENDER and NUMBER. Βλέπε παρακάτω στο Special Attributes για την ολοκληρωμένη λίστα των υπολογισμένων attributes.

iv. :CODE

Ξεπερασμένο συνώνυμο του :HTML.

v. :STRING

Ξεπερασμένο συνώνυμο του :HTML.

vi. Special Attributes

Μερικά χαρακτηριστικά είναι ειδικά επειδή υπολογίζονται από WikipediaBase αντί να είναι τραβηγμένα από infoboxes ή δεν παρέχονται άμεσα. Αυτά τα χαρακτηριστικά θα πρέπει να είναι ειδικά για wikibase-term, wikibase-person, και wikipedia-paragraphs.

A. SHORT-ARTICLE, wikibase-term

Η πρώτη παράγραφος του άρθρου, ή αν η πρώτη παράγραφος είναι μικρότερη από 350 χαρακτήρες, τότε το μέρος της πρώτης παραγράφου έτσι ώστε το άθροισμα των χαρακτήρων να είναι τουλάχιστον 350.

ii. URL, wikibase-term

Επιστρέφει το URL του άρθρου ως ((:url URL))

B. IMAGE-DATA, wikibase-term

Επιστρέφει μια λίστα από URLs εικόνων στο περιεχόμενο του άρθρου (αποκλείει εικόνες που είναι στη σελίδα αλλά εκτός του περιεχομένου του άρθρου). Εάν δεν υπάρχουν εικόνες θα πρέπει να επιστρέφει μια κενή λίστα.

Η "καλύτερη" εικόνα πρέπει να είναι η πρώτη της λίστας, εάν υπάρχει εικόνα στην κορυφή του infobox, αυτή θεωρείται η καλύτερη εικόνα, διαφορετικά είναι η πρώτη εικόνα που εμφανίζεται οπουδήποτε στο άρθρο. Εάν δεν υπάρχει caption, η τιμή του caption παραλείπεται

π.χ., προτιμότερο ((0 "Harimau\ _Harimau\ _cover .jpg"))
από ((0 "Harimau\ _Harimau\ _cover .jpg" " ")).

vii. COORDINATES, wikibase-term

Χαρακτηριστικά που δίνονται στο άρθρο υπολογιζόμενα από το γεωγραφικό πλάτος και το μήκος ή, εφόσον κανένα δεν μπορεί να βρεθεί, το infobox . Η τιμή είναι μια λίστα του πλάτους και μήκους, πχ. ((:coordinates latitude longitude))

Black Sea

From Wikipedia, the free encyclopedia
(Redirected from [Black sea](#))


Coordinates:  44°N 35°E

Figure 2.1: An example of coordinates in the header

viii. BIRTH-DATE, wikibase-person

Λαμβάνονται από το infobox ή αν δεν βρεθεί, λαμβάνονται από το άρθρο,

ή αν δεν βρεθεί, από τις πληροφορίες της κατηγορίας του άρθρου.

Βασίζεται πάντα στην πρώτη ημερομηνία γέννησης που εντοπίστηκε και ταιριάζει σε μια από τις διάφορες υποστηριζόμενες μορφές.

Αν αυτό το χαρακτηριστικό έχει μια τιμή,

τότε το αντικείμενο θεωρείται ότι είναι ένα πρόσωπο με αξία στην ιδιότητα ΦΥΛΟ (βλέπε παρακάτω) .

Η τιμή μπορεί να είναι μια a parsed or unparsed date. Parsed dates αντιπροσωπεύονται ως αριθμούς , χρησιμοποιώντας τη μορφή YYYYMMDD χρησιμοποιώντας αρνητικούς αριθμούς για τις ημερομηνίες Π.Χ.

ix. DEATH-DATE, wikibase-person

Λαμβάνονται με παρόμοιο τρόπο όμως το BIRTH-DATE. Επιστρέφει την ίδια τιμή όπως

BIRTH-DATE, εκτός αν το πρόσωπο ζει, τότε βγάζει λάθος τιμή με διευκρίνιση
"Currently alive".

x. GENDER, wikibase-person

Υπολογίζεται από το περιεχόμενο της σελίδας βασιζόμενο στα heuristics όπως ο αριθμός των ανδρικών ή των θηλυκών αντωνυμιών που χρησιμοποιούνται στο κείμενο

xi. NUMBER, wikibase-term

Το αν η περιγραφόμενη έννοια είναι ενικός ή πληθυντικός. Υπολογίζεται από το περιεχόμενο του κειμένου με βάση τα χαρακτηριστικά όπως ο αριθμός των φορών που ο τίτλος της σελίδας εμφανίζεται στον πληθυντικό. Έχει αξία για όλα τα αντικείμενα.

Επιστρέφει #t είναι πληθυντικός, #f αν είναι ενικός.

xii. PROPER, wikibase-term

Το αν η περιγραφόμενη έννοια είναι κύριο όνομα. Υπολογίζεται από το περιεχόμενο του κειμένου με βάση τα χαρακτηριστικά όπως ο αριθμός των φορών που ο τίτλος της σελίδας εμφανίζεται με κεφαλαία γράμματα όταν δεν είναι στην αρχή της σελίδας. Έχει τιμή για όλα τα αντικείμενα.

Επιστρέφει #t αν είναι κύριο όνομα, #f αν δεν είναι.

(b) get-classes

Δεδομένου του ονόματος ενός αντικειμένου , επιστρέφει μια λίστα με όλες τις classes όπου ανήκει το αντικείμενο, με τις classes να αντιπροσωπεύονται ως lisp-readable strings. Παραδοσιακά τα ονόματα

των τάξεων δίνονται με μικρά γράμματα χωρίς όμως αυτό να είναι απολύτως απαραίτητο.

```
(get-classes "Cardinal_(bird)")  
=> ("wikibase-term" "wikipedia-paragraphs" "  
    wikipedia-taxobox")
```

```
(get-classes "Hillary_Rodham_Clinton")  
=> ("wikibase-term"  
    "wikipedia-paragraphs"  
    "wikibase-person"  
    "wikipedia-officeholder"  
    "wikipedia-person")
```

(c) `get-attributes`

Δεδομένου του ονόματος μιας τάξης, επιστρέφει έναν κατάλογο με όλα τα χαρακτηριστικά της τάξης (δηλαδή όλες οι μεταβλητές που τα υλοποιεί infobox), ως lisp-readable strings. Τα ονόματα των χαρακτηριστικών δίνονται σε κεφαλαία γράμματα, αλλά αυτό δεν αποτελεί απόλυτη απαίτηση.

```
(get-attributes "wikipedia-officeholder" "Barack  
    _Obama")  
=> ((:CODE "TERM_END3" :VALUE :YYYYMMDD) ...)
```

(d) `Sort-symbols`

Βάζοντας σε σειρά σύμβολα παίρνει κάθε σύνολο συμβόλων και τα βάζει σε σειρά δημιουργώντας υποσύνολα κατα μήκος του σχετικού άρθρου.

```
(sort-symbols "Obama_(surname)" "Barack_Obama")  
=> (("Barack_Obama") ("Obama_(surname)"))
```

(e) `sort-symbols-named`

παίρνει ένα συνώνυμο και ένα σύνολο συμβόλων και τα βάζει σε σειρά δημιουργώντας υποσύνολα. Εάν το symbol name είναι το ίδιο με το συνώνυμο, το ίδιο και το υποσύνολό του μπαίνουν στην αρχή.

```
(sort-symbols-named
```

```

"cake"
"Cake_(TV_series)"
"Cake_(firework)"
"Cake_(film)"
"Cake_(drug)"
"Cake"
"Cake_(band)"
"Cake_(advertisement)"
"The_Cake")
=> (("Cake")
("Cake_(band)")
("Cake_(advertisement)")
("Cake_(TV_series)")
("The_Cake")
("Cake_(film)")
("Cake_(firework)")
("Cake_(drug)"))

```


Chapter 3

Getting started

Η συνολική WikipediaBase βρίσκεται σε ένα git repository στο infolab's github organization page.

```
git clone git@github.com:infolab-csail/WikipediaBase
```

Το WikipediaBase εξαρτάται από πολλά άλλα πακέτα python. Ευτυχώς, η python είναι shipped όχι μόνο με ένα σπουδαίο package manager αλλά επίσης με ένα μηχανισμό που ονομάζεται `virtualenv` το οποίο απομονώνει την εγκατάσταση των εξαρτήσεων από το υπόλοιπο σύστημα, έτσι αποφεύγονται προβλήματα όπως ασυμβατότητα εκδόσεων.

ή namespace collisions. Ο τρόπος που αυτό δουλεύει αποτελεσματικά είναι με το `global`

python installation να είναι το μισό copied και το μισό symlinked σε ένα τοπικό directory και τα dependencies να είναι εγκαταστημένα μόνο σε ένα τοπικό sandbox.

Για να δημιουργηθεί και να ενεργοποιηθεί ένα python `virtualenv`:

```
$ virtualenv --no-site-packages py
$ . py/bin/activate
$ which python
/the/local/directory/py/bin/python
```

Τώρα που ασφαλώς τα έχουμε εγκαταστήσει όλα θέλουμε χωρίς να σπάσουμε κάποιο `global installation`

```
pip install -r requirements.txt
```

Θα χρειασθούμε μερικά επιπλέον εργαλεία για να δουλέψει η Wikipedi-aBase που θα πρέπει να εγκατασταθούν system wide:

- Postgresql
- Redis

Η εγκατάσταση αυτών των πακέτων διαφέρει ανάλογα με το λειτουργικό σύστημα ή τον package manager. Και οι δύο είναι βάσεις δεδομένων. Ο σκοπός τους είναι η προσωρινή αποθήκευση επαναλαμβανόμενη υπολογισμών και για την αποθήκευση ahead-of-time υπολογισμού , όπως το όνομα infobox σήμανσης για καθίσταται χάρτες όνομα και συνώνυμα.

Chapter 4

Αρχιτεκτονική

1. Infobox

Τα Infoboxes είναι πίνακες που χρησιμοποιούνται συνήθως στη wikipedia για να παρέχουν μια επισκόπηση των πληροφοριών σε ένα άρθρο με ένα ημι δομημένο τρόπο . Infoboxes είναι η κύρια πηγή πληροφοριών για τη WikipediaBase


```

| birth_place      = [[Greifswald]], [[Germany]]
| death_date      = {{Death date and age
|1945|8|4|1909|11|24}}
| death_place     = [[Prague]], [[Czechoslovakia]]
| nationality     = [[Germany|German]]
| fields          = [[Mathematics]]
| workplaces      =
| alma_mater      = [[University of Gottingen]]
| doctoral_advisor = [[Paul Bernays]]
| doctoral_students =
| known_for      =
| awards          =
}}
```

Θα παράξει το εξής:

Οι τύποι του Infobox είναι οργανωμένοι με μια αρκετά ευρεία ιεραρχία. Για παράδειγμα `Template:Infobox Austrian district` είναι μια ειδική περίπτωση ενός `Template:Infobox settlement` και το καθένα είναι rendered διαφορετικά. Για το συγκεκριμένο σκοπό, και για να κάνουμε mirror το markup ορίζουμε τα infoboxes, ένα infobox I με χαρακτηριστικά a_i και τιμές v_i είναι ένα σύνολο από ζεύγη a_i, v_i μαζί με ένα τύπο infobox t . Κάθε χαρακτηριστικό a_i και τιμή v_i έχουν 2 μορφές:

- rendered μορφή, a_i^r και v_i^r αντίστοιχα, η rendered HTML αναπαράσταση
 - Η markup αναπαράσταση, a_i^m και v_i^m που είναι η mediawiki markup συμβολοσειρά

Ένα άρθρο μπορεί να έχει περισσότερα από ένα infoboxes, για παράδειγμα, το άρθρο για τον Bill Clinton έχει δύο infobox: ένα για `Officeholder` και ένα για `Infobox President`. Η class `Infobox` είναι ο βασικός τύπος δεδομένων για την πρόσβαση σε πληροφορίες από το infobox ενός άρθρου. Η `Infobox`, όπως και η `Article`, είναι αυτή που θα χρησιμοποιήσει κάποιος όταν χρησιμοποιεί τη `wikipediabase` ως βιβλιοθήκη Python. Οι μέθοδοι που παρέχονται από την `Infobox` δίνουν πρόσβαση στις εξής πληροφορίες:

Τύποι επειδή έχουμε ανακτήσει Infobox βασισμένοι σε ένα όνομα συμβόλου (π.χ. όνομα της σελίδας), ένα μοναδικό Infobox μπορεί στην πραγματικότητα να είναι μια διεπαφή για πολλαπλά infoboxes. Υπάρχει μια ξεχωριστή μέθοδος, που βασίζεται σε αυτό, για την ανάκτηση τύπων σε μορφή κατάλληλη για το START.

Τιμές αρακτηριστικών δεδομένης είτε a_i^r είτε a_i^m .

Ονόματα χαρακτηριστικών που παρέχονται με τη χρήση του MetaInfobox (βλέπε παρακάτω)

Εξαγωγή των πληροφοριών σε python types συγκεκριμένα

- dict για $a_i^r \rightarrow v_i^r$ or $a_i^m \rightarrow v_i^m$
- Το συνολικό infobox rendered, ή σε ένα markup μορφή.

Τα Infoboxes οργανώνονται σε μια ευρεία ιεραρχία το οποίο στον κώδικα του WikipediaBase αναφέρεται ως infobox tree. Το infobox tree ανακτάται από τη λίστα της σελίδας wikipedia List of infoboxes και χρησιμοποιείται για να συνταχθεί η οντολογία των όρων wikipedia.

2. MetaInfobox

Το MetaInfobox υλοποιείται ως μια υποκλάση του Infobox που προσδίδει πληροφορία σχετικά με το infobox, εστιάζοντας στη αντιστοιχία της rendered μορφής των χαρακτηριστικών με την markup μορφή. Έτσι δεδομένου ενός infobox τύπου I έχει δυνατά χαρακτηριστικά a_1, \dots, a_n . Κάθε χαρακτηριστικό έχει δύο αναπαραστάσεις:

- τη markup αναπαράσταση που χρησιμοποιείται στο infobox template.
- την HTML rendered αναπαράσταση, που είναι το κείμενο που φαίνεται στην αριστερή μεριά του πίνακα του infobox στη σελίδα.

Παραδείγματος χάριν στο officeholder infobox υπάρχει ένα χαρακτηριστικό με markup αναπαράσταση predecessor και μία rendered αναπαράσταση Preceded by.

Για να το πετύχει αυτό το MetaInfobox χρησιμοποιεί την σελίδα τεκμηρίωσης του template για να βρει το markup representation όλων των αποδεκτών χαρακτηριστικών ενός τύπου infobox. Στη συνέχεια δημιουργεί ένα infobox όπου κάθε χαρακτηριστικό έχει τιμή τη markup αναπαράσταση του χαρακτηριστικού αυτού, τυλιγμένη με τη συμβολοσειρά `!!!`. (Για παράδειγμα το χαρακτηριστικό με markup όνομα predecessor θα έχει τιμή `!!!predecessor!!!`). Στη συνέχεια κάνει render το infobox που δημιούργησε και ψάχνει για `!!!predecessor!!!` στις rendered τιμές. Θεωρούμε ότι οι τα αντίστοιχα rendered ονόματα αντιστοιχούν στα markup χαρακτηριστικά. Σημειώστε πως η αντιστοιχία των rendered χαρακτηριστικών με τα markup χαρακτηριστικά δεν είναι αμφοσήμαντη,

δηλαδή κάθε markup χαρακτηριστικό μπορεί να αντιστοιχεί σε μηδέν ή περισσότερα rendered χαρακτηριστικά και το αντίστροφο.

Για παράδειγμα για ένα infobox τύπου Foo με αποδεκτά χαρακτηριστικά *A*, *B*, *C* και *D* το MetaInfobox θα δημιουργούσε markup:

```
{{Infobox Foo
| A = !!!A!!!
| B = !!!B!!!
| C = !!!C!!!
| D = !!!D!!!
}}
```

Και η rendered μορφή θα ήταν, ανάλογα με την υλοποίηση του Foo infobox.

Attribute	Value
A	!!!A!!! !!!B!!! !!!C!!!
B	!!!A!!! !!!B!!! !!!C!!!
C	!!!A!!! !!!B!!! !!!C!!!
D	!!!D!!!

Έτσι η αντιστοιχία γίνεται σχετικά εμφανής.

3. Article

Η class *Article* είναι υπεύθυνη για την πρόσβαση σε κάθε πόρο σχετικό με το άρθρο γενικότερα. Αυτό περιλαμβάνει τις παραγράφους, επικεφαλίδες, τον πηγαίο markup κωδικά και τις κατηγορίες MediaWiki.

4. Fetcher

Η κλάση *Fetcher* είναι μια αφαίρεση από την επικοινωνία της WikipediaBase με τον έξω κόσμο. Είναι ένα μονήρες αντικείμενο που υλοποιεί μια συγκεκριμένη διεπαφή.

Τα υλοποιημένα *Fetchers* σε κληρονομική ιεραρχία που φαίνεται από την παρακάτω λίστα.

BaseFetcher είναι η υπερκλάση όλων των fetchers. Θα επιστρέψει το ίδιο σύμβολο αντί να προσπαθήσει να το επιλύσει με οποιονδήποτε τρόπο. Κάνουμε override αυτή τη λειτουργία στις κληρονόμους κλάσεις για να υλοποιήσουμε τη λογική της διεπαφής με τον έξω κόσμο

Fetcher Υλοποιεί τη βασική λειτουργία. Αναζητά πληροφορίες απο το wikipedia.org. Είναι δυνατόν να κατευθύνουμε ένα fetcher αυτό προς ένα mirror αλλά η εκτέλεση σε wikipedia-mirror είναι από άποψη πόρων εκτέλεσης απαγορευτική.

CachingFetcher κληρονομεί fetcher και διατηρεί τη λειτουργικότητα , μόνο που χρησιμοποιεί Redis για την προσωρινή αποθήκευση των fetched συμβόλων . Είναι η προεπιλεγμένη fetcher class.

StaticFetcher είναι μια κλάση που υλοποιεί το interface BaseFetcher αλλά αντί να φτάσει σε κάποια πηγή δεδομένων για τα δεδομένα η τιμές επιστροφής είναι στατικά ορισμένες. Χρησιμοποιείται κυρίως από το MetaInfobox για να χρησιμοποιεί τη λειτουργία του Infobox να μεταφέρει αυθαίρετες πληροφορίες.

Από προεπιλογή, το markup προέρχεται μια βάση δεδομένων. Αν η παράμετρος `force_live` έχει οριστεί σε `True` τότε το markup θα ληφθεί από live wikipedia.org. Όταν οι δοκιμές τρέχουν στο TravisCI, θέλουμε πάντα να χρησιμοποιούνται ζωντανά δεδομένα. Ελέγχουμε αν το Travis εκτελεί δοκιμές κοιτάζοντας τη μεταβλητή `WIKIPEDIABASE_FORCE_LIVE` μεταβλητή περιβάλλοντος.

5. Renderer

Renderers είναι μονήρεις classes, χρήσιμες για την απόδοση MediaWiki markup σε HTML . Αρχικά το wikipedia sandbox χρησιμοποιήθηκε από τη wiki database για την απόδοση σελίδων, επειδή είναι ελαφρώς ταχύτερο από την API, αλλά το wikipedia-mirror ήταν πολύ αργό και το wikipedia.org το θεωρούσε κατάχρηση της υπηρεσίας και μπλόκαρε το IP μας μετά από μερικά τεστ. Γι' αυτό το λόγο τελικά μεταπηδήσαμε στο API με Redis caching, το οποίο λειτουργούν αρκετά καλά, μας που τα Renderer αντικείμενα καταλήγουν να χρησιμοποιούνται μόνο απο το MetaInfobox, το οποίο έχει ένα αρκετά περιορισμένο πεδίο εφαρμογής , και έτσι το cache να χάνει σπάνια.

Μια ενδιαφέρουσα πληροφορία για την κατηγορία Renderer ήταν ότι αυτός ήταν ο λόγος που ένα ζευγάρι CSAIL IPs να αποκλειστεί προσωρινά από την επεξεργασία της wikipedia. Ενώ η wikipedia.org έχει μια πολύ επιεική πολιτική όταν πρόκειται για την αποκλεισμό των ανθρώπων που έχουν κανει spamming τους servers, επαναλαμβανόμενες δοκιμές της κατηγορίας Renderer με στόχευση wikipedia sandbox προκάλεσε το ip του δοκιμαστικού μηχανήματος να αποκλεισθεί προσωρινά με το σκεπτικό ότι "η δραστηριότητα του δεν προάγει την βελτίωση της wikipedia".

Εμείς επανατοποθετήσαμε το `Renderer` να χρησιμοποιηθεί το `wikipedia API` και ποτέ δεν είχαμε ξανά πρόβλημα με την ρύθμιση της `wikipedia`.

6. Pipeline

Κατά την επίλυση ενός ερωτήματος η `WikipediaBase` ενεργοποιεί ένα pipeline ενοτήτων για να διαπιστωθεί ποιος είναι ο καλύτερος τρόπος απάντησης.

(a) Frontend

Η `WikipediaBase` μπορεί να χρησιμοποιηθεί ως βιβλιοθήκη αλλά ο πρωταρχικός της λειτουργία είναι ως backend στο `START`. Η επικοινωνία μεταξύ `START` και `WikipediaBase` γίνεται πάνω από ένα plaintext telnet σύνδεσής στην πόρτα 8023 χρησιμοποιώντας sexpressions. Το frontend χειρίζεται το δίκτυο σύνδεσης με το `START`, μεταφράζει τις προσλαμβανόμενες ερωτήσεις σε κλήσεις της `Knowledgebase` και στη συνέχεια μεταφράζει την αντίδραση της `Knowledgebase` σε κατάλληλα διαμορφωμένες εκφράσεις και τις επιστρέφει πίσω στο telnet connection.

(b) Knowledgebase

Η `knowledgebase` είναι το σημείο εισαγωγής στο rest της `wikipediabase`.

Χρησιμοποιεί μοτίβο `Provider/Acquirer` να παρέχει διαφανή διεπαφή της frontend με αυθαίρετες μεθόδους. Οι μέθοδοι αυτοί είναι υπεύθυνοι για την επιλογή του αν θέλουμε να καταλήξουμε σε classifiers, resolvers οποιοδήποτε άλλο μηχανισμό για να δοθεί απάντηση στο ερώτημα. Διαθέσιμοι classifiers και resolvers γίνονται προσβάσιμοι αυτόματα στη `knowledgebase` χρησιμοποιώντας τη βασική τους κλάση.

(c) Classifiers

Κάθε Classifier είναι μονήρης κλάση και υλοποιεί μια ευρετική για να συνάξει μια λίστα από κατηγορίες ενός αντικειμένου. Ένα αντικείμενο μπορεί να επιστρέφει μηδέν ή περισσότερες κατηγορίες. Συνήθως, ένας Classifier θα συμπεράνει μόνο αν ένα αντικείμενο πράγματι αναστέλλει μια συγκεκριμένη κατηγορία ή όχι, αλλά αυτό δεν είναι απαραίτητο.

i. Term

Ο `TermClassifier` απλά αναθέτει την κατηγορία `wikipedia-term`. Η `Wikipediabase` διαπραγματεύεται μόνο μόνο με πληροφορίες

σχετικές με τη wikipedia. Συνεπώς όλες οι έννοιες που συναντώνται είναι σε αυτήν την κατηγορία.

ii. Infobox

Το InfoboxClassifier αναθέτει σε ένα όρο την κατηγορία infobox. Για παράδειγμα η σελίδα Bill Clinton περιέχει το infobox:

```
{{Infobox president
|name           = Bill Clinton
|image          = 44 Bill Clinton 3x4.jpg{{!}}border
|...]}
}}
```

Και γι αυτό λαμβάνει την κατηγορία wikipedia-president.

iii. Person

Το PersonClassifier αναθέτει την κατηγορία wikibase-person χρησιμοποιώντας κάποια χαρακτηριστικά με την σειρά που περιγράφονται:

A. Category regexes

Χρησιμοποιεί τις ακόλουθες συνήθεις εκφράσεις (regular expressions) για να ταυτίσει τις κατηγορίες ενός άρθρου.

- .* person
- ^\d+ deaths.*
- ^\d+ births.*
- .* actors
- .* deities
- .* gods
- .* goddesses
- .* musicians
- .* players
- .* singers

B. Category regex excludes

Αποκλείει τις ακόλουθες regexes.

- \sbased on\s
- \sabout\s
- lists of\s
- animal\s

C. Category matches

Γνωρίζουμε ότι ένα άρθρο αναφέρεται σε ένα πρόσωπο εάν η σελίδα ανήκει σε μια από τις ακόλουθες mediawikia κατηγορίες:

- american actors
- american television actor stubs
- american television actors
- architects
- british mps
- character actors
- computer scientist
- dead people rumoured to be living
- deities
- disappeared people
- fictional characters
- film actors
- living people
- musician stubs
- singer stubs
- star stubs
- united kingdom writer stubs
- united states singer stubs
- writer stubs
- year of birth missing
- year of death missing

Για ένα παράδειγμα δείτε το παράρτημα.

Όπως είναι φανερό η λίστα με τις κατηγορίες είναι αθυαίρετη και όχι πλήρης. Πολλαπλές μέθοδοι μπορούν να χρησιμοποιηθούν για να διορθωθεί αυτό.

Μερικές από αυτές είναι:

- Μέθοδοι με Supervised machine learning όπως SVM χρησιμοποιώντας άλλες μεθόδους να ορίσουν ένα πρόσωπο και να δημιουργήσουν εκπαιδευτικές ομάδες.
- Εμπλουτίζοντας την υπάρχουσα λίστα κατηγοριών χρησιμοποιώντας στατιστικά από κατηγορίες άρθρων που έχουμε βρει με άλλους τρόπους ότι αναφέρονται σε πρόσωπα.

7. Resolvers

Οι Resolvers είναι επίσης μονήρεις κλάσεις αλλά ο σκοπός τους είναι να βρουν την τιμή του αναζητούμενου χαρακτηριστικού. Όλοι οι resolvers κληρονομούν από την κλάση BaseResolver και πρέπει να υλοποιούν τις ακόλουθες μεθόδους:

- `resolve(class, symbol, attribute)` που δίνει την τιμή ενός χαρακτηριστικού δεδομένου του συμβόλου και της κλάσης.
- `attributes(class, symbol)`: που δίνει μια λίστα από τα χαρακτηριστικά που μπορεί να επιλύσει ο συγκεκριμένος resolver για το συγκεκριμένο άρθρο δεδομένης της class του.

Οι υλοποιημένοι resolvers είναι οι ακόλουθοι:

Error ο ελάχιστης προτεραιότητας resolver. Επιλύεται πάντα σε σφάλμα.

Infobox Επιλύει χαρακτηριστικά που αναφέρονται σε κάποιο πεδίο του infobox

Person επιλύει τα ακόλουθα ειδικά χαρακτηριστικά των άρθρων που αναφέρονται σε πρόσωπα

- `birth-date`
- `death-date`
- `gender`

Sections το περιεχόμενο των κεφαλαίων σε ένα άρθρο.

Term επιλύει ένα συγκεκριμένο σύνολο χαρακτηριστικών,

- `coordinates` Οι συντεταγμένες μιας γεωγραφικής περιοχής
- `image` Την εικόνα μέσα στο infobox.
- `number` Αληθής τιμή αν το σύμβολο είναι στον πληθυντικό (πχ *The Beatles*)
- `proper` Αληθής αν αναφέρεται σε κύριο όνομα.
- `short-article` Περίληψη του άρθρου, τυπικά η πρώτη παράγραφος.
- `url` Η διεύθυνση του άρθρου.
- `word-count` Το μέγεθος του άρθρου σε λέξεις.

8. Lisp types

Ο τύπος Lisp είναι περιτυλίγματα (wrappers) για python αντικείμενα ή τιμές που παρουσιάζονται σε μορφή s-expression που το START μπορεί να κατανοήσει. Έχουν δημιουργηθεί είτε από το ανεπεξέργαστο ερώτημα

και έχουν ξετυλιγεί (unwrapped) ώστε να είναι χρήσιμα στον αγωγό (pipeline), ή από την απάντηση που δίνει η WikipediaBase και στη συνέχεια κωδικοποιούνται σε ένα string και αποστέλλονται μέσω telnet στο START.

Chapter 5

Το μοντέλο provider/acquirer

Η WikipediaBase προσπαθεί να είναι modular και με δυνατότητα επέκτασης. Για να επιτευχθεί αυτό, Συχνά είναι χρήσιμο να συμπλέκει πολλαπλές πηγές του ίδιου τύπου του πόρου δεδομένων. Αυτό είναι ιδιαίτερα χρήσιμο κατά την πρόσβαση ευρετικών μεθόδων όπως των classifiers που είδαμε παραπάνω. Για την προώθηση του modularity και για να αποφευχθεί ισχυρή αλληλεξάρτηση των υποσυστημάτων δημιουργήθηκε το μοντέλο provider/acquirer.

Ο Provider είναι ένα αντικείμενο μέσω του οποίου μπορούμε να διαχειριστούμε πηγές που είναι αποθηκευμένες ως ζεύγη κλειδιού - τιμής. Η κλάση Provider προσφέρει decorators για να κάνει αυτή της διάταξη εύκολη για τον προγραμματιστή. Ένας Acquirer έχει διαφανή (transparent) πρόσβαση στους πόρους πολλαπλών Providers σαν να ήταν ένα ενιαίο σύνολο κλειδιών. Αυτό το πρότυπο κυρίως χρησιμοποιείται για την KnowledgeBase ώστε να παρέχει το Frontend με τρόπο πρόσβασης στις πηγές.

Chapter 6

Testing

Η καλή λειτουργία της WikipediaBase εξασφαλίζεται από μια ολοκληρωμένη σειρά δοκιμών των unit tests, functional tests και regression tests. Τα Unit tests ελέγχουν μια μικρή ομάδα του functionality, το οποίο έχει συντεθεί για την δημιουργία του όλου συστήματος. Για το unit testing χρησιμοποιούμε την default βιβλιοθήκη python για testing. Κάθε τεστ είναι μια κλάση που κληρονομεί από την κλάση TestCase και υλοποιεί το interface της που περιγράφεται παρακάτω.

Τα Functional tests είναι γραμμένα από πριν, κατά τη διάρκεια ή λίγο μετά της δημιουργίας του συστήματος και διεκδικούν τη σωστή συνολική λειτουργία του συστήματος. Τα Regression tests είναι πολύ παρόμοια με τα functional tests. Αποδεικνύουν ότι όταν βρεθεί ένα σφάλμα(bug)το διορθώνουν και επιβεβαιώνουν ότι δεν θα εμφανισθεί ξανά αργότερα. Τα Functional και τα regression tests είναι τοποθετημένα στα tests/examples.py

Σχεδόν όλα τα τεστ ξεκινούν με τον ακόλουθο κώδικα:

```
from __future__ import unicode_literals

try:
    import unittest2 as unittest
except ImportError:
    import unittest

from wikipediabase import fetcher
```

Το παραπάνω είναι ειδικό για το fetcher module. Όπως είναι προφανές χρησιμοποιούμε το unittest module από την βιβλιοθήκη python. Το test το ίδιο έχει το ακόλουθο format:

```

class TestFetcher(unittest.TestCase):

    def setUp(self):
        self.fetcher = fetcher.get_fetcher()

    def test_html(self):
        html = self.fetcher.html_source("Led_Zeppelin")
        self.assertIn("Jimmy_Page", html)

```

Η `setUp` μέθοδος `runs` πριν από κάθε τεστ του `TestCase`. Τα τεστ του `testcase` αντιπροσωπεύονται από μεθόδους της `class` το οπών το όνομα αρχίζει με `test_`. Στην συγκεκριμένη περίπτωση παίρνουμε την σελίδα της wikipedia για το *Led Zeppelin* ότι το όνομα *Jimmy Page* αναφέρεται τουλάχιστον μια φορά. Αυτό φανερά δεν συνάδει ότι το `fetcher` δεν φέρνει για παράδειγμα την σελίδα για το *Yardbirds, Page's first band*. Για αυτό το λόγο γράφουμε μερικά από αυτού του είδους τεστ.

Στην περίπτωση του `fetcher`, για να ακολουθήσουμε το παράδειγμα, το συνολικό τεστ υπάρχει στο παράρτημα.

Εφαρμόσαμε το εργαλείο `nose` να βρούμε και να τρέξουμε τα τεστ. Για να το κάνουμε αυτό το προσθέσαμε σαν προαπαιτούμενο στο `setup.py`.

```

from setuptools import setup

setup(
    tests_require=[
        'nose>=1.0',
        ...
    ],
    ...
    test_suite='nose.collector',
    ...
)

```

Στη συνέχεια να τρέξουμε τα τεστ:

```
$ python setup.py test
```

Η `Nose` θα βρει όλα τα αρχεία τα οποία είναι στα `tests/` και έχουν τ πρόθεμα `test_`, για παράδειγμα `test_fetcher.py`. Μέσα σ αυτά τα αρχεία η `nose` θα αναζητήσει subclass της `TestCase` και των οποίων το όνομα αρχίζει με `Test`, για παράδειγμα `TestFetcher`. Στη συνέχεια τρέχει όλες τις μεθόδους

από τις collected classes που έχουν το προθεμ test_. Είναι επίσης δυνατό να τρέξει συγκεκριμένα τεστ τεστ.

```
$ python setup.py test --help
Common commands: (see '--help-commands' for more)

    setup.py build          will build the package underneath
                           'build/'
    setup.py install       will install the package
```

Global options:

```
--verbose (-v)  run verbosely (default)
--quiet (-q)    run quietly (turns verbosity off)
--dry-run (-n)  don't actually do anything
--help (-h)     show detailed help message
--no-user-cfg   ignore pydistutils.cfg in your home
                directory
```

Options for 'test' command:

```
--test-module (-m)  Run 'test_suite' in specified
                    module
--test-suite (-s)    Test suite to run (e.g. '
                    some_module.test_suite')
--test-runner (-r)   Test runner to use
```

```
usage: setup.py [global_opts] cmd1 [cmd1_opts] [cmd2 [
    cmd2_opts] ...]
    or: setup.py --help [cmd1 cmd2 ...]
    or: setup.py --help-commands
    or: setup.py cmd --help
```

Δείτε το παράρτημα για επιτυχημένη εκτέλεση των τεστ.

Chapter 7

Συνώνυμα

Πριν μιλήσουμε για τα συνώνυμα είναι σημαντικό να ορίσουμε τα σύμβολα στο πεδίο του omnibase universe:

Σύμβολα είναι ταυτοποιητές των "αντικειμένων" "objects" στις πηγές των πληροφοριών (ο όρος "σύμβολο" ("symbol") είναι ατυχής γιατί έχει διάφορες έννοιες στην επιστήμη των υπολογιστών. Δυστυχώς έχει μείνει για ιστορικούς λόγους.)

Δεδομένου ότι η γλώσσα τείνει να έχουν πολλαπλές λέξεις που αναφέρονται στο ίδιο πράγμα, Είναι επιτακτική η ανάγκη να καθορισθούν ονόματα για τα σύμβολα. Συνώνυμα είναι τα ονόματα τα οποία οι χρήστες μπορούν να χρησιμοποιήσουν για να αναφερθούν στα σύμβολα.

(Ο όρος συνώνυμα "synonym" είναι ατυχής γιατί είναι one-way mapping - "gloss" θα ήταν καλύτερος όρος αλλά έμεινε ο όρος συνώνυμα για ιστορικούς λόγους)

Ο ορισμός συνωνύμων είναι δουλειά του backend. Για το λόγο αυτό αναλαμβάνει η WikipediaBase να ορίσει τα απαιτούμενα συνώνυμα.

1. Καλά και κακά συνώνυμα

Υπάρχουν κανόνες για το ποιο είναι καλό ή κακό συνώνυμο

- Δεν πρέπει να ξεκινούν με άρθρα ("the", "a", "an")
- Δεν πρέπει να ξεκινούν με "File:" or "TimedText:".
- Δεν πρέπει να περιέχουν HTML anchors. Πχ "Alexander^{Pushkin}#Legacy"
- Δεν πρέπει να ξεκινούν με τα ακόλουθα:
 - "List of "
 - "Lists of "

- "Wikipedia: "
- "Category: "
- ":Category: "
- "User: "
- "Image: "
- "Media: "
- "Arbitration in location"
- "Communications in location"
- "Constitutional history of location"
- "Economy of location"
- "Demographics of location"
- "Foreign relations of location"
- "Geography of location"
- "History of location"
- "Military of location"
- "Politics of location"
- "Transport in location"
- "Outline of topic"
- Δεν πρέπει να ταιριάζει `\d\d\d\d in location` ή `location in \d\d\d\d`
- Δεν πρέπει να είναι ονόματα των disambiguation pages. Για να το κάνουμε αυτό έτσι ώστε να συμπεριλαμβάνει όλες τις σχετικές σελίδες, συμπεριλαμβανομένων των τυπογραφικών λαθών, αυτό σημαίνει σύμβολα που ταιριάζουν με `\([Dd]isambig[\^])*\`
- Συνώνυμα που ότι τόσο α) θα μπορούσαν να εκληφθούν ότι ξεκινούν με άρθρα και β) μπορεί να υποτάσσουν κάτι χρήσιμο . Αυτό σημαίνει ότι για παράδειγμα « A. House» (συνώνυμο του «Abraham House») είναι ελλιπών προδιαγραφών διότι ενδέχεται να παραπλανήσει START στην περίπτωση των ερωτήσεων όπως «Πόσο κοστίζει ένα σπίτι στη Silicon Valley;» . Αφετέρου "a priori" μπορεί να διατηρηθεί επειδή δεν υπάρχουν λογικές ερωτήματα όπου "α" είναι ένα άρθρο πριν "priori" .

2. Παραγωγή συνωνύμων

Για να συμβιβάσουμε αυτούς τους περιορισμούς δύο μέθοδοι χρησιμοποιούνται qualification και modification των υποψήφιων συνωνύμων. Πρώτα προσπαθούμε τη modification και αν αυτό αποτύχει επιχειρούμε να κάνουμε disqualify. Οι κανόνες για modification έχουν ως εξής:

- Να διαγράψουμε τα άρθρα από την αρχή ενός συνωνύμου:
 - "A "
 - "An "
 - "The "
 - "(The) "
 - The
 - κτλ
- Να δημιουργούμε και τα δύο versions, με και χωρίς παρενθέσεις. Πχ, δεδομένου του συμβόλου "Raven (journal)" δημιουργούμε:
 - "Raven (journal)"
 - "Raven"
- Χρησιμοποιούμε τη συμβολοσειρά πριν και μετά toslash, αλλά όχι το αρχικό symbol, πχ. δεδομένου του συμβόλου "Russian language/Russian alphabet" δημιουργούμε
 - "Russian language"
 - "Russian alphabet"
- Ανάστροφη των ανεστραμμένων συμβόλων με κόμματα. Πχ δεδομένου "Congo, Democratic Republic Of The", αναστρέφουμε για να πάρουμε "Democratic Republic Of The Congo"
- Ός συνήθως , απορρίπτουμε leading articles εάν είναι αναγκαίο. Π.χ. δοθέντος συμβόλου "Golden ratio, the" το αντικαθιστούμε με "the Golden ratio", στη συνέχεια διαγράφουμε τα άρα για πάρουμε: "Golden ratio" το ίδιο συμβάνει για τα a, an, κτλ.

Με αυτό τον τρόπο κάναμε generate ένα αρχικό πακέτο συνωνύμων από το ίδιο το όνομα του αντικειμένου. Επιπλέον μπορούμε να κάνουμε generate ένα πακέτο από από τα wikipedia redirects στο άρθρο. Η Wikipedia παρέχει ένα SQL dump για όλα τα redirects. Για να φορτώσουμε τον πίνακα στην βάση δεδομένων όπου έχουμε φορτώσει τα δεδομένα της wikipedia, πρέπει να φορτώσουμε τον πίνακα των redirects:

```
wget https://dumps.wikimedia.org/enwiki/latest/
enwiki-latest-redirect.sql.gz \
-0 redirect.sql.gz && gzcata redirect.sql.gz |
mysql
```

Και στη συνέχεια το SQL db για να βρούμε όλα τα συνώνυμα του (καλά και κακά) Bill Clinton μπορούμε να:

```
select page_title, rd_title from redirect join page
on rd_from = page_id and (rd_title = "
Bill_Clinton" or page_title = "Bill_Clinton");
```

Για το πλήρες output δείτε στο παράρτημα.

Chapter 8

Databases and data sources

1. HTML and MediaWiki API

Η αρχική προσέγγιση για να πάρουμε τα δεδομένα είναι να ανασύρουμε τις φυσιολογικές HTML εκδόσεις των άρθρων της wikipedia και χρησιμοποιώντας edit pages να ανασύρουμε το mediawiki markup. Ανεξαιρέτως χρησιμοποιήσαμε το αρχικό wikipedia.org site για λόγους performance (Βλέπε κεφάλαιο wikipedia-mirror runtime performance).

Το Mediawiki παρέχει a RESTful API για όλη την απαιτούμενη λειτουργία (functionality). Η βαική αρχή είναι ότι κάποιος μπορεί να στείλει αιτήματα με μεθόδους POST ή GET και να λαμβάνει απάντηση με την μορφή XML ή JSON. Η προτιμητέα απάντηση για την WikipediaBase ήταν να στέλνονται GET HTTP αιτήματα και να λαμβάνουν JSON δεδομένα. Το GET επιλέχθηκε ειδικά προτάθηκε στην mediawiki API page γιατί caching συμβαίνει στο HTTP επίπεδο. Σύμφωνα με τις οδηγίες του HTTP τα POST αιτήματα δεν μπορούν να cached. Για το λόγο αυτό όταν διαβάζει κάποιος δεδομένα από web service API, θα πρέπει να χρησιμοποιεί GET αιτήματα και όχι POST.

Επίσης πρέπει να σημειωθεί ότι ένα αίτημα δεν μπορεί να εκτελεσθεί από cache εκτός αν το URL είναι ακριβώς το ίδιο. Εάν ζητήσεις ένα αίτημα για `api.php?titles=Foo|Bar|Hello`, και αποθηκεύσει το αποτέλεσμα, μετά `api.php?titles=Hello|Bar|Hello|Foo` δεν θα βρει την απάντηση στην cache παρ'όλο που είναι το ίδιο αίτημα!

Η αναπαράσταση JSON επιλέχθηκε άπλα επειδή η βιβλιοθήκη json της python πολύ πιο εύκολη στη χρήση από την lxml, τη βιβλιοθήκη που χρησιμοποιούμε για XML/HTML parsing.

2. Caching

Η Wikipediabase χρησιμοποιεί κυρίως έναν απομακρυσμένο χώρο αποθήκευσης δεδομένων και εφαρμόζει το mediawiki interface (δηλαδή το mediawiki). Προσπαθεί να αντιμετωπίσει ζητήματα επιδόσεων που προκύπτουν με την προσωρινή αποθήκευση των σελίδων σε μια τοπική key-value βάση δεδομένων. Το interface με τη βάση δεδομένων αφαιρείται με τη χρήση ενός python dictionary-style interface, το οποίο εφαρμόζεται στο `persistentkn.py`. Εφαρμογές των backends παρουσιάζονται παρακάτω, αλλά είναι ασήμαντο να παρέχεται κάθε backend που ο καθένας συναντά. Ένα άλλο χαρακτηριστικό που το interface στην βάση δεδομένων πρέπει να μπορεί να χρησιμοποιήσει είναι η κωδικοποίηση των αποθηκευμένων αντικειμένων. Επειδή όλη η αποθηκευμένη πληροφορία είναι κείμενο, η βάση δεδομένων πρέπει να είναι ικανή να ανασύρει ακριβώς το κείμενο που έχει αποθηκευθεί λαμβάνοντας υπόψη την κωδικοποίηση. Λόγω των περιορισμών του DBM's τα κλειδιά (keys) πρέπει να είναι μόνο κωδικοποιημένα ASCII. Η βασική κλάση για αλληλεπίδραση με την βάση δεδομένων, το `EncodedDict`, εφαρμόζει τις μεθόδους `_encode_key` και `_decode_key` για να παραχωρήσει ένα εύκολο hook εφαρμογών με σκοπό να διαχειρισθεί πιθανές καταστάσεις.

(a) DBM

Διάφορες υλοποιήσεις dbm παρέχονται από την σταθερή βιβλιοθήκη της python. Όμως καμιά από τις standard βιβλιοθήκες της python δεν είναι μέρος της σταθερής βιβλιοθήκης της python. Μερικές εφαρμογές DBM που είναι διαθέσιμες μέσω της σταθερής βιβλιοθήκης της python είναι:

- AnyDBM
- GNU DBM
- Berkeley DBM

Είναι σημαντικό να αναφέρουμε ότι η ομαλή λειτουργία αυτών των βιβλιοθηκών εξαρτάται σε σημαντικό βαθμό από την βασική πλατφόρμα όπως το λειτουργικό.

Όπως αναφέρθηκε παραπάνω οι interface classes του DBM μεταφράζουν από και προς ASCII. Ο ακριβής μηχανισμός που γίνεται αυτό είναι:

(b) SQLite

Η SQLite επίσης χρησιμοποιείται ως caching backend βάση δεδομένων. Δυστυχώς η αποτελεσματικότητά του στο δικό μας σκοπό ήταν απογοητευτική. Χρησιμοποιήσαμε ένα πολύ λεπτό wrapper, `sqldict`,

για να πάρουμε ένα key-value interface to SQLite – μια relational βάση δεδομένων. Ο σχετικός WikipediaBase κώδικας είναι πολύ σύντομος:

```
from sqllitedict import SqliteDict

class SqlitePersistentDict(EncodedDict):
    def __init__(self, filename, configuration
    =configuration):
        if not filename.endswith('.sqlite'):
            filename += '.sqlite'

        db = SqliteDict(filename)
        super(SqlitePersistentDict, self).
            __init__(db)

    def sync(self):
        self.db.close()
        super(SqlitePersistentDict, self).sync()
```

Παρακάτω είναι δυο benchmark functions που θα διαβάσουν και θα γράψουν 1000000 φορές στην βάση.

```
def benchmark_write(dic, times=100000):
    for i in xrange(times):
        dic['o' + str(i)] = str(i) * 1000

    def benchmark_read(dic, times=100000):
        for i in xrange(times):
            dic['o' + str(i)]
```

Και παρακάτω φαίνεται πως συγκρίνονται τα διάφορα backends χρησιμοποιώντας αυτές τις δυο συναρτήσεις.

```
>>> import timeit
>>> sqlkv = SqlitePersistentDict('/tmp/bench1.
    sqlite')
>>> timeit.timeit(lambda : benchmark_write(sqlkv
    ), number=100)
10.847157955169678
>>> timeit.timeit(lambda : benchmark_read(sqlkv)
    , number=100)
18.88098978996277
```



```
>>> dbmkv = DbmPersistentDict('/tmp/bench.dbm')
>>> timeit.timeit(lambda : benchmark_write(dbmkv
    ), number=100)
0.18030309677124023
>>> timeit.timeit(lambda : benchmark_read(dbmkv)
    , number=100)
0.14914202690124512
```

Η DBM βάση δεδομένων είναι σχεδόν 10 φορές ταχύτερη από sqlite. Η διαφορά στην εκτέλεση οφείλεται στις διαφορετικές committing policies που έχουν μεταξύ τους. Μπορεί να είναι δυνατόν να ρυθμιστεί το SQLite ώστε να είναι τόσο γρήγορο όσο η DBM αλλά όχι με κάποιον εύκολο τρόπο.

(c) Άλλα backends

Άλλα backends λαμβάνονται υπόψη, κυρίως το Redis το οποίο εφαρμόστηκε αμέσως μετά την παράδοση της εργασίας από τον Alvaro Morales. Ο λόγος που αρχικά δεν το χρησιμοποιήσαμε ήταν γιατί έχει μοντελοποιηθεί ως ένας server-client και προσθέτει περιπλοκότητα σε ένα τμήμα του συστήματος το οποίο πρέπει να είναι όσο το δυνατόν πιο απλό. Ένας άλλος λόγος του αρχικού προβληματισμού μας ήταν σχετικά με το ότι το redis είναι ανεξάρτητο project δηλαδή δεν είναι μέρος της python. Θεωρήσαμε πως ήταν καλύτερα να αποφευχθούν επιπλέον εξαρτήσεις ειδικά όταν είναι η cool database της ημέρας.

Chapter 9

Date parser

Η κατανόηση ημερομηνιών υπάρχει σε ένα ξεχωριστό πακέτο που ονομάζεται `overlay-parse`.

1. Parsing με overlays

Η έννοια του `overlay` εμπνεύστηκε από τα `emacs overlays`. Είναι αντικείμενα που εξειδικεύουν την συμπεριφορά ενός υποσυνόλου του κειμένου με το να του δίνουν ιδιότητες για παράδειγμα το κάνουν `clickable` ή `highlighted`.

Ένα `overlay` επί ενός μέρους ενός κείμενου t στο πλαίσιο μας είναι:

- Ένα ζευγάρι που ορίζει την έκταση του υπο-κείμενου
- ένα σύνολο από ετικέτες (`tag set`) που ορίζουν τα εννοιολογικά σύνολα στα οποία εμπίπτει το συγκεκριμένο υποκείμενο.
- Αυθαίρετες πληροφορίες (τύπου `A`) που το συγκεκριμένο υποκείμενο εκφράζει.

Πιο αυστηρά:

$$\begin{aligned} o_i &\in \text{TextRanget} \times \text{Set}(\text{Tag}) \times A & \text{numbers} \\ \text{Text} &\rightarrow \{o_1, o_2, \dots, o_n\} \end{aligned}$$

Για παράδειγμα, από το παρακάτω κείμενο

The weather today, $\overbrace{\text{Tuesday}}^{o_1} \overbrace{21^{st}}^{o_2}$ of $\overbrace{\text{November}}^{o_3} \overbrace{2016}^{o_4}$, was sunny.

Μπορούμε να εξάγουμε overlays $\{o_1, \dots, o_4\}$ έτσι ώστε

```
o1 = ( r("Tuesday"), {DayOfWeek, FullName}, 2)
o2 = ( r("21st"), {DayOfMonth, Numeric}, 21)
o3 = ( r("November"), {Month, FullName}, 11)
o4 = ( r("2016"), {Year, 4digit}, 2016)
```

Παρατηρείστε ότι όλα τα overlays του παραδείγματος έχουν $A = \mathbb{N}$, όπως κωδικοποιούμε την ημέρα της εβδομάδας, τη μέρα του μήνα, το μήνα του έτους ως φυσικούς αριθμούς. Κωδικοποιούμε πιο ακριβή πληροφορία (πχ αυτή η μέρα είναι διαφορετική από την μήνα από την φύση της) στο σύνολο των ετικετών (tag sets).

Μόλις έχουμε ένα σύνολο από overlays μπορούμε να ορίσουμε μια overlay sequences ως overlays τα οποία έχουν συνεχόμενο εύρος, Αυτά και τα δικά τους tag sets ταυτίζονται με ειδικά μοτίβα . Για παράδειγμα μπορούμε να ψάξουμε για σειρές από overlays που ταιριάζουν με το pattern

$p = \text{DayOfMonth}, \text{Separator}(/), (\text{Month} \wedge \text{Number}), \text{Separator}(/), \text{Year}$

ταιριάζει patterns όπως 22/07/1991, όπου *Separator(/)* ταιριάζει μόνο με τον χαρακτήρα "/"

2. Το παράδειγμα των ημερομηνιών

Η βασική εφαρμογή που θα χρησιμοποιήσουμε ως παράδειγμα για τη λειτουργία των overlays είναι η κατανόηση ημερομηνιών. Το `dates` submodule έχει 2 βασικά entry points:

- `just_dates` που ψάχνει για ημερομηνίες σε ένα κείμενο.
- `just_ranges` που ψάχνει για εύρη ημερομηνιών σε ένα κείμενο.

Παρακάτω παρουσιάζονται κάποια παραδείγματα. Σημειώστε πως 0 σημαίνει unspecified

```
>>> from overlay_parse.dates import just_dates,
      just_ranges, just_props
>>> just_dates("Timestamp: 22071991: She said she
was \
```

```

coming on april the 18th, it's 26 apr 2014
and hope is leaving me.")
... [(22, 7, 1991), (18, 4, 0), (26, 4, 2014)]
>>> dates = just_dates("200 AD 300 b.c.")
>>> just_dates("200 AD 300 b.c.")
[(0, 0, 200), (0, 0, -300)]
>>> just_ranges(u"I will be there from 2008 to 2009"
)
[((0, 0, 2008), (0, 0, 2009)))]
>>> just_ranges("I will stay from July the 20th
until today")
[((20, 7, 0), (29, 4, 2016)))]
>>> just_dates('{Birth date and age|1969|7|10|df=y
}')
[(10, 7, 1969)]
>>> just_ranges(u'German:\u02c8v\u0254lf\u0261a\
u014b\u02c8de\u02d0\u028as\u02c8mo\u02d0tsa
\u0281t], English see fn.; [1] 27 January 1756\
xa0\u20135 December 1791')
[((27, 1, 1756), (5, 12, 1791)))]

```

Chapter 10

Παραρτήματα

1. Παράδειγμα python unit test

```
class TestFetcher(unittest.TestCase):
    def setUp(self):
        self.fetcher = fetcher.get_fetcher()

    def test_html(self):
        html = self.fetcher.html_source("Led_
            Zeppelin")
        self.assertIn("Jimmy_Page", html)

    def test_markup_source(self):
        src = self.fetcher.markup_source("Led_
            Zeppelin")
        self.assertIn("{Infobox_musical_artist",
            src)

    def test_unicode_html(self):
        html = self.fetcher.html_source(u"Rhône")
        self.assertIn("France", html)

    def test_unicode_source(self):
        src = self.fetcher.markup_source("Rhône")
        self.assertIn("Geobox|River", src)

    def test_silent_redirect(self):
        # redirects are only supported when
        # force_live is set to True
        src = self.fetcher.markup_source("Obama",
```

```
        force_live=True)
self.assertFalse(re.match(fetcher.
    REDIRECT_REGEX, src))
```

2. Παράδειγμα εκτέλεσης ενός python test

```
$ python setup.py test -s tests.test_lispify
running test
running egg_info
writing requirements to wikipediabase.egg-info/
    requires.txt
writing wikipediabase.egg-info/PKG-INFO
writing top-level names to wikipediabase.egg-info/
    top_level.txt
writing dependency_links to wikipediabase.egg-info/
    dependency_links.txt
writing entry points to wikipediabase.egg-info/
    entry_points.txt
reading manifest file 'wikipediabase.egg-info/
    SOURCES.txt'
reading manifest template 'MANIFEST.in'
writing manifest file 'wikipediabase.egg-info/
    SOURCES.txt'
running build_ext
test_bool (tests.test_lispify.TestLispify) ... ok
test_bool_with_typecode (tests.test_lispify.
    TestLispify) ... ok
test_date_multiple_voting (tests.test_lispify.
    TestLispify) ... ok
test_date_simple (tests.test_lispify.TestLispify)
    ... ok
test_date_with_range (tests.test_lispify.TestLispify
    ) ... ok
test_dict (tests.test_lispify.TestLispify) ... ok
test_dict_with_escaped_string (tests.test_lispify.
    TestLispify) ... ok
test_dict_with_list (tests.test_lispify.TestLispify)
    ... ok
test_double_nested_list (tests.test_lispify.
    TestLispify) ... ok
test_error (tests.test_lispify.TestLispify) ... ok
test_error_from_exception (tests.test_lispify.
    TestLispify) ... ok
test_keyword (tests.test_lispify.TestLispify) ... ok
```

```

test_keyword_with_typecode (tests.test_lispify.
    TestLispify) ... ok
test_list (tests.test_lispify.TestLispify) ... ok
test_list_of_dict (tests.test_lispify.TestLispify)
    ... ok
test_list_of_dict_with_typecode (tests.test_lispify.
    TestLispify) ... ok
test_list_with_typecode (tests.test_lispify.
    TestLispify) ... ok
test_nested_list (tests.test_lispify.TestLispify)
    ... ok
test_none (tests.test_lispify.TestLispify) ... ok
test_none_with_typecode (tests.test_lispify.
    TestLispify) ... ok
test_number (tests.test_lispify.TestLispify) ... ok
test_number_with_typecode (tests.test_lispify.
    TestLispify) ... ok
test_string (tests.test_lispify.TestLispify) ... ok
test_string_escaped (tests.test_lispify.TestLispify)
    ... ok
test_string_not_keyword (tests.test_lispify.
    TestLispify) ... ok
test_string_with_typecode (tests.test_lispify.
    TestLispify) ... ok
test_unicode_string (tests.test_lispify.TestLispify)
    ... ok

```

```

Ran 27 tests in 0.047s

```

```

OK

```

3. Βρίσκοντας συνώνυμα με MySQL

```

mysql> select page_title, rd_title from \
redirect join page on
rd_from = page_id and
(rd_title = "Bill_Clinton" or page_title = "
    Bill_Clinton");
(rd_title = "Bill_Clinton" or page_title = "
    Bill_Clinton");

```

```

+--

```

```

| page_title                                | rd_title
|
+--
-----+-----+

| BillClinton                                | Bill_Clinton
|
| William_Jefferson_Clinton                  | Bill_Clinton
|
[... see below for a formatted verison of the data
...]
| William_Jefferson_Clinton                  | Bill_Clinton
|
+--
-----+-----+

```

46 rows in set (11.77 sec)

page_title	rd_title
BillClinton	Bill_Clinton
William_Jefferson_Clinton	Bill_Clinton
President_Clinton	Bill_Clinton
William_Jefferson_Blythe_IV	Bill_Clinton
Bill_Blythe_IV	Bill_Clinton
Clinton_Gore_Administration	Bill_Clinton
Buddy_(Clinton's_dog)	Bill_Clinton
Bill_clinton	Bill_Clinton
William_Jefferson_Blythe_III	Bill_Clinton
President_Bill_Clinton	Bill_Clinton
Bull_Clinton	Bill_Clinton
Clinton,_Bill	Bill_Clinton
William_clinton	Bill_Clinton
42nd_President_of_the_United_States	Bill_Clinton
Bill_Jefferson_Clinton	Bill_Clinton
William_J._Clinton	Bill_Clinton
Billl_Clinton	Bill_Clinton
Bill_Clinton\	Bill_Clinton
Bill_Clinton's_Post_Presidency	Bill_Clinton
Bill_Clinton's_Post-Presidency	Bill_Clinton

Continued on next page

Continued from previous page

page_title	rd_title
Klin-ton	Bill_Clinton
Bill_J._Clinton	Bill_Clinton
William_Jefferson_"Bill"_Clinton	Bill_Clinton
William_Blythe_III	Bill_Clinton
William_J._Blythe	Bill_Clinton
William_J._Blythe_III	Bill_Clinton
Bil_Clinton	Bill_Clinton
WilliamJeffersonClinton	Bill_Clinton
William_J_Clinton	Bill_Clinton
Bill_Clinton's_sex_scandals	Bill_Clinton
Billy_Clinton	Bill_Clinton
Willam_Jefferson_Blythe_III	Bill_Clinton
William_"Bill"_Clinton	Bill_Clinton
Billlll_Clinton	Bill_Clinton
Bill_Klinton	Bill_Clinton
William_Clinton	Bill_Clinton
Willy_Clinton	Bill_Clinton
William_Jefferson_(Bill)_Clinton	Bill_Clinton
Bubba_Clinton	Bill_Clinton
MTV_president	Bill_Clinton
MTV_President	Bill_Clinton
The_MTV_President	Bill_Clinton
Howard_G._Paster	Bill_Clinton
Clintonesque	Bill_Clinton
William_Clinton	Bill_Clinton
William_Jefferson_Clinton	Bill_Clinton

4. Παράδειγμα κατηγοριών άρθρων

Το άρθρο που αναφέρεται στον Leonardo DiCaprio εντάσσεται στις επόμενες κατηγορίες (με **bold** είναι η κατηγορία που χρησιμοποιεί το Wikipedia-Base για να αποφασίσει πως το άρθρο αναφέρεται σε άνθρωπο).

- Leonardo DiCaprio
- 1974 births
- **Living people**
- 20th-century American male actors

- 21st-century American male actors
- American environmentalists
- American film producers
- American male child actors
- American male film actors
- American male soap opera actors
- American male television actors
- American people of German descent
- American people of Italian descent
- American people of Russian descent
- American philanthropists
- Best Actor AACTA Award winners
- Best Actor Academy Award winners
- Best Drama Actor Golden Globe (film) winners
- Best Musical or Comedy Actor Golden Globe (film) winners
- California Democrats
- Film producers from California
- Formula E team owners
- Male actors from Hollywood, California
- Male actors from Palm Springs, California
- Male actors of Italian descent
- People from Echo Park, Los Angeles
- Silver Bear for Best Actor winners

Οι κατηγορίες αυτές μοιάζουν ως εξής στο άρθρο

Categories: [Leonardo DiCaprio](#) | [1974 births](#) | [Living people](#) | [20th-century American male actors](#) | [21st-century American male actors](#) | [American environmentalists](#) | [American film producers](#) | [American male child actors](#) | [American male film actors](#) | [American male soap opera actors](#) | [American male television actors](#) | [American people of German descent](#) | [American people of Italian descent](#) | [American people of Russian descent](#) | [American philanthropists](#) | [Best Actor AACTA Award winners](#) | [Best Actor AACTA International Award winners](#) | [Best Actor Academy Award winners](#) | [Best Actor BAFTA Award winners](#) | [Best Drama Actor Golden Globe \(film\) winners](#) | [Best Musical or Comedy Actor Golden Globe \(film\) winners](#) | [California Democrats](#) | [Film producers from California](#) | [Formula E team owners](#) | [Male actors from Hollywood, Los Angeles](#) | [Male actors from Palm Springs, California](#) | [Male actors of Italian descent](#) | [Outstanding Performance by a Male Actor in a Leading Role Screen Actors Guild Award winners](#) | [People from Echo Park, Los Angeles](#) | [Silver Bear for Best Actor winners](#)

Figure 10.1: The rendered list of categories for Leonardo DiCaprio

Part IV

WikipediaMirror

Wikipedia mirror είναι ένα σύστημα με στόχο να αυτοματοποιήσει τη δημιουργία ενός τοπικού κλώνου της wikipedia περιέχοντας μόνο τα άρθρα — δεν περιέχει τους χρήστες , συζήτηση και ιστορικό επεξεργασιών. Η αυτοματοποιημένη διαδικασία περιλαμβάνει τη ρύθμιση ενός διακομιστή, μια βάση δεδομένων και γέμισμα αυτής της βάσης δεδομένων με τα άρθρα της wikipedia. Ο σκοπός για αυτό είναι να παρέχει την δυνατότητα πρόσβασης του συνόλου των δεδομένων της Wikipedia, ανεξάρτητα από το wikipedia.org .

Chapter 11

Mediawiki stack overview

Το wikipedia-mirror βασίζεται στο MediaWiki stack που παρέχεται από το Bitnami, μια υπηρεσία που χτίζει το σύνολο του διακομιστή εντός των ορίων ενός direcotry. Αυτό είναι χρήσιμο γιατί αποφεύγεται η επιβάρυνση της χρήσης container ή VM τεχνολογίας και μας δίνει τη δυνατότητα να έχουμε άμεση πρόσβαση στο σύστημα αρχείων του stack, ενώ εξακολουθούσαμε να έχουν το σύστημα κατασκευής Bitnami να κάνει την κοπιώδη εργασία της ενορχήστρωσης των διαφόρων τμημάτων και επίσης διαχωρίζεται ο διακομιστής από το υπόλοιπο συστήματος.

Το stack αποτελείται από

- Έναν http server, στην περίπτωση μας τον apache
- Ένα web application runtime, στην περίπτωση μας PHP
- Μια βάση δεδομένων, στην περίπτωση μας η MySQL
- Το ίδιο το web application, δηλαδή mediawiki

Όλα τα παραπάνω παρέχονται από το bitnami mediawiki stack. Το Xampp παλιότερα ήταν αποδεκτό η καλύτερη επιλογή αλλά είναι unmaintained, έτσι αποφασίσαμε να χρησιμοποιήσουμε το bitnami το οποίο δουλεύει αρκετά καλά.

Όταν το stack ρυθμιστεί κατάλληλα, το wikipedia dump xml κατεβαίνει και μετατρέπεται σε sql dump με mwddumper. Θα μπορούσε να riped άμεσα στο MySQL αλλά η εξαγωγή παίρνει χρόνο τα πράγματα μπορεί να χειροτερέψουν κατά το dumping.

1. Στοιχεία του stack.

Παρουσιάζεται κάθε στοιχείο του stack με περισσότερες λεπτομέρειες παρακάτω.

(a) Apache Σύμφωνα με τη wikipedia:

The Apache HTTP Server, colloquially called Apache, is the world's most used web server software. Originally based on the NCSA HTTPd server, development of Apache began in early 1995 after work on the NCSA code stalled. Apache played a key role in the initial growth of the World Wide Web, quickly overtaking NCSA HTTPd as the dominant HTTP server, and has remained most popular since April 1996. In 2009, it became the first web server software to serve more than 100 million websites.

Apache is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation. Most commonly used on a Unix-like system (usually Linux), the software is available for a wide variety of operating systems besides Unix, including eComStation, Microsoft Windows, NetWare, OpenVMS, OS/2, and TPF. Released under the Apache License, Apache is free and open-source software.

Είναι δίκαιο να πούμε ότι apache είναι ένα από τους πιο δημοφιλείς διακομιστές web στο διαδίκτυο. Η ίδια η wikipedia.org φαίνεται να χρησιμοποιεί ένα πιο σύνθετο stack που περιλαμβάνει varnish, ένα HTTP επιταχυντή, και nginx, μια εναλλακτική λύση, επίσης αρκετά δημοφιλή διακομιστή HTTP. Καταλήξαμε σε αυτό το συμπέρασμα από την επιθεώρηση των headers που επιστρέφονται από τη wikipedia.org . Στην περίπτωση `http://www.wikipedia.org` ανακατευθυνόμαστε προς το secure domain (προσοχή στη γραμμή Server:):

```
$ curl -s -D - http://www.wikipedia.org -o /dev/null
HTTP/1.1 301 TLS Redirect
Server: Varnish
[...]
```

Και αν ζητήσουμε κατ ευθείαν για το `https://www.wikipedia.org`

```
$ curl -s -D - https://www.wikipedia.org -o /dev
/null
HTTP/1.1 200 OK
Server: nginx/1.9.4
[...]
```

Ωστόσο, είναι πέρα από το πεδίο της συγκεκριμένης εργασίας να αναπαράγουμε με ακρίβεια υποδομή της Wikipedia . Έχουμε επικεντρωθεί στην λειτουργικότητα. Σ αυτό λόγω της δημοτικότητας, και της εν δυνάμει ταχύτητας των ρυθμών της αυτόματης εγκατάστασης το Bitnami MediaWiki stack χρησιμοποιήθηκε ως διακομιστή μας

(b) PHP

Η MediaWiki , η οποία συζητείται αργότερα, είναι γραμμένη εξ ολοκλήρου σε PHP, μια δημοφιλής πλευρά του server , με δυναμική δακτυλογράφηση, προσανατολισμένη στα αντικείμενα , γλώσσα scripting. Το PHP είναι απαραίτητο και είναι εγκατεστημένο με το Bitnami mediawiki stack. Το PHP είναι δημοφιλής ανάμεσα στους προγραμματιστές του web και αυτό οφείλεται εν μέρει στην υποστήριξη που έχει από πολλούς σχετικές βιβλιοθήκες με βάσεις δεδομένων (συμπεριλαμβανομένων PostgreSQL, MySQL Microsoft SQL Server και SQLite) και είναι ουσιαστικά ένα template δημιουργίας προτύπων γλώσσας HTML.

(c) MySQL

Mediawiki μπορεί να χρησιμοποιήσει πληθώρα SQL database back-ends:

- **MSSQL:** Μια SQL βάση από τη Microsoft
- **MySQL:** Χρησιμοποιώντας τη standard PHP library για MySQL.
- **MySQLi:** Μια επέκταση του MySQL backend
- **Oracle:** Μια αποκλειστικής εκμεταλλεύσεως SQL database από την Oracle.
- **SQLite:** Μια SQL database που συνήθως χρησιμοποιείται ως βιβλιοθήκη αντί για client-server scheme όπως γίνεται με τις άλλες επιλογές της λίστας

Η Wikipedia παρέχει πολλαπλά dump files για τους SQL πίνακες δευτερογενούς σημασίας στο MySQL format (eg. page redirects, categories etc) και προτείνει mwdumper που μετατρέπει XML dumps των άρθρων της wikipedia σε MySQL. Αυτό και το ότι παρέχεται με το αυτοματοποιημένο stack του bitnami, κάνει το MySQL την προφανή επιλογή για το wikipedia-mirror stack.

(d) Mediawiki

To Mediawiki είναι η καρδιά της wikipedia. Το MediaWiki είναι ένα free και open-source wiki application. Δημιουργήθηκε από το Wikimedia Foundation και τρέχει πολλά δημοφιλή site όπως Wikipedia, Wikitionary και Wikimedia Commons.

Το λογισμικό έχει περισσότερα από 800 ρυθμίσεις και περισσότερα από 2.000 επεκτάσεις διαθέσιμες για τη διευκόλυνση διάφορα χαρακτηριστικά για να προστεθεί ή να αλλάξει. Στη Wikipedia και μόνο, πάνω από 1000 αυτοματοποιημένη και έχουν ήμι - αυτοματοποιημένα bots και άλλα εργαλεία έχουν αναπτυχθεί για να βοηθήσουν στο moderation. Τα περισσότερα από αυτά δεν έχουν σημασία για τους δικούς μας σκοπούς. Οι χρήσιμες για μας επεκτάσεις είναι οι Scriunto και parserfunctions, και οι μόνες χρήσιμες ρυθμίσεις σχετίζονται με το όνομα της τοποθεσίας, το όνομα της βάσης δεδομένων κλπ και ως επί το πλείστον τις διαχειρίζεται το Bit-nami

Chapter 12

Setting up

Στη συνέχεια είναι βήμα προς βήμα οδηγίες για να στήσει κάποιος το wikipedia mirror. Πρώτα κατεβάζουμε τον κωδικά χρησιμοποιώντας το git:

```
$ git clone https://github.com/fakedrake/wikipedia-mirror
$ cd wikipedia-mirror
```

Σ' αυτό το σημείο θεωρητικά κάποιος μπορεί να τρέξει `make sql-load-dumps` τα οποία θα φροντίσουν να στηθεί οτιδήποτε χρειάζεται να φορτωθεί βάση δεδομένων σε μορφή dumps σε μια λειτουργική SQL βάση δεδομένων. Φυσικά για να γίνει αυτό πρώτα θα εκτελεσθούν μερικά βήματα.

- Download the wikipedia database dumps in XML format.
- Transform them into a format that MySQL understands.
- Set up the bitnami stack that includes a local install of MySQL
- Load the MySQL dumps into MySQL

Όλα αυτά τα βήματα κωδικοποιούνται ως τμήμα μιας εξαρτώμενης ιεραρχίας κωδικοποιούμενα σε makefile targets και στη θεωρία αυτό πραγματοποιείται αυτόματα και αποτελεσματικά οδηγείται σε αποτελεσματική wikipedia mirror. Όμως αυτή λειτουργία είναι μεγάλη και ευθραυστή και συνιστάται κάθε βήμα να γίνεται εξατομικευμένα και χειροκίνητα.

Πρώτα, κατεβάζουμε και εγκαθιστάμε το bitnami. Η ακόλουθη εντολή θα κατεβάσει έναν executable από το bitnami website και θα κάνει μια τοπική εγκατάσταση του bitnami stack όπως συζητήθηκε παραπάνω:

```
$ make bmw-install
```

Το επόμενο βήμα είναι να βεβαιωθούμε ότι το maven, η java είναι ένα is a software project management and comprehension είναι εγκαταστημένα, απαιτείται να εγκατασταθεί και να στηθεί το mwdumper (βλέπε παρακάτω). Μπορεί να γίνει αυτό αν βεβαιωθούμε ότι τα παρακάτω έχουν επιτευχθεί:

```
$ mvn --version
```

Σημείωση: if running on Ubuntu 14.04, ίσως χρειασθεί να εγκαταστήσουμε το Maven (για Java) χρησιμοποιώντας `sudo apt-get install maven`.

Τώρα όλα είναι έτοιμα για το αυτόματο download Wikipedia's XML dumps και στη συνέχεια τα μετατρέπει σε SQL χρησιμοποιώντας mwdumper. Πρώτα το mwdumper θα πρέπει να κατέβει και να χτισθεί. Μετά τα συμπιεσμένα XML dumps θα πρέπει να κατέβουν από την wikipedia. Θα γίνουν uncompressed και τελικά θα μετατραπούν σ MySQL dumps χρησιμοποιώντας mwdumper. Αυτή είναι πολύ χρονοβόρα διαδικασία και χρειάζεται 6-11 ώρες σε ένα τυπικό μηχάνημα:

```
$ make sql-dump-parts
```

Όταν αυτο γίνει επιτυχώς μπορούμε να φορτώσουμε τα SQL dumps στη βάση δεδομένων MySQL

```
$ make sql-load-parts
```

Και τελικά

```
$ make mw-extensions
```

Chapter 13

Mediawiki extensions

Για τη MediaWiki για να ενεργήσει όπως η wikipedia απαιτούνται μια σειρά από extensions. Η διαδικασία εγκατάστασης των εν λόγω extensions δεν είναι αυτοματοποιημένη ή streamline. Για να γίνει αυτόματη διαχείριση αυτής της πολυπλοκότητας ένας μηχανισμός παρέχεται την εγκατάσταση των extensions. Για υποστηρίξουμε επιπλέον για την wiki database πρέπει να προσθέσουμε τον ακόλουθο κώδικα στο `Makefile.mwextensions` (τροποποιημένο αναλόγως)

```
MW_EXTENSIONS += newextension
mw-newextension-url = url/to/new/extension/package.tar.
gz
mw-newextension-php = NewExtensionFile.php
mw-newextension-config = '$$phpConfigVariable = "value"
;'
```

Η wikipedia-mirror θα φροντίσει ώστε το extension να είναι ήδη εγκαταστημένο και εάν δεν είναι θα τοποθετήσει τα σωστά αρχεία στο σωστό μέρος και θα διορθώσει τους κατάλληλους configuration files. Τα entry points για την διαχείριση των extensions είναι (με την προϋπόθεσή ότι το όνομά του εγγραφομένων extensions είναι newextension):

```
make mw-print-registered-extensions # Output a list of
the registered extensions
make mw-newextension-enable          # Install and/or
enable the extension
make mw-newextension-reinstall       # Reinstall an
extension
make mw-newextension-disable         # Disable the
extension
```

```
make mw-newextension-clean          # Remove the  
extension
```

Όλα τα registered extensions θα εγκατασταθούν και θα ενεργοποιηθούν
όταν το wikipedia-mirror έχει χτισθεί.

Chapter 14

Φορτώνοντας τα mediawiki dumps

Η Wikipedia παρέχει μηνιαία dumps όλων των βάσεων δεδομένων της. Το μεγαλύτερο μέρος των dumps είναι σε μορφή XML και πρέπει να κωδικοποιούνται σε MySQL να φορτωθούν στη βάση δεδομένων wikipedia-mirror. Υπάρχουν περισσότεροι από ένας τρόποι να το κάνουμε αυτό.

Το Mediawiki πακετάρεται με ένα βοηθητικό πρόγραμμα για την εισαγωγή του XML dump. Ωστόσο, η χρήση του για την εισαγωγή ενός πλήρους wikipedia-mirror αποθαρρύνεται λόγω των περιορισμών επιδόσεων. Αντ' αυτού προτείνονται εργαλεία όπως `mwddumper` που μετατρέπουν τα XML dumps σε MySQL ερωτήματα που γεμίζουν τη βάση δεδομένων.

Το `mwddumper` είναι γραμμένο σε Java και αποστέλλονται χωριστά από MediaWiki και μπορεί να μετατρέψει τα δεδομένα μεταξύ των ακόλουθων μορφών:

- XML
- MySQL dump
- SQLite dump
- CSV

Για τους σκοπούς μας έχει ενδιαφέρον μόνο ο μετασχηματισμός από XML σε MySQL.

Chapter 15

Εργαλεία

Ένας αριθμός εργαλείων αναπτύχθηκε για να βοηθήσουν τη διαδικασία του χειρισμού και της παρακολούθησης της διαδικασία του φορτώματος των dumps στη βάση. Παρουσιάζονται με λεπτομέρεια παρακάτω. Εφ' όσον ο πηγαίος κώδικάς τους είναι συνοπτικός παρατίθεται ολόκληρος στο παράρτημα

1. utf8thread.c

Το `utf8thread.c` είναι ένα άλλο χαμηλού επιπέδου πρόγραμμα το οποίο γεμίζει με κενά όλα τα `invalid utf-8 characters` από το αρχείο. Χρησιμοποιούμε `pthreads` για να επιταχύνουμε τα πράγματα.

2. webmonitor.py

Το `webmonitor.py` είναι ένα python script το οποίο sets up μια σελίδα web page που δείχνει live δεδομένα σε μορφή ιστογράμματος για την πρόοδο του πληθυσμού της βάσης δεδομένων. `webmonitor.py` σερβίρει στατικές html σελίδες μετά της στέλνει δεδομένα μέσω websocket. `Webmonitor` μπορεί να δείχνει οποιοδήποτε stream από τα ζευγάρια `<epoch date>` `<float value>` που λαμβάνει στο input. Σαν παράδειγμα:

```
$ pip install tornado
```

Πρώτα πρέπει να εγκαταστήσουμε τα dependencies του script. Το οποίο μπορεί να είναι `tornado`, `anasynchronous web framework supporting websockets`. Δίνουμε οδηγίες `tornado`, `tornado` θα υπηρετεί τις ακόλουθες σελίδες:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "
  http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/
      html; charset=utf-8">
    <title>DrNinjaBatmans Websockets</title>

    <script type="text/javascript" src="http://code.
      jquery.com/jquery-1.10.1.js"></script>
    <script type="text/javascript" src="http://code.
      highcharts.com/highcharts.js"></script>

    <script>
      var chart; // global
      var url = location.hostname + ':' + (parseInt(
        location.port));
      var ws = new WebSocket('ws://' + url + '/
        websocket');
      ws.onmessage = function(msg) {
        add_point(msg.data);
      };

      // ws.onclose = function() { alert('Connection
        closed. '); };

      var add_point = function(point) {
        var series = chart.series[0],
        shift = series.data.length > %d;

        chart.series[0].addPoint(eval(point), true,
          shift);
      };

      $(document).ready(function() {
        chart = new Highcharts.Chart(JSON.parse('%s
          '));
      });
    </script>

  </head>
  <body>
    <div id="container" style="width: 800px;
      height: 400px; margin: 0 auto"></div>
  </body>

```

```
</html>
```

Με την έννοια αυτή η σελίδα αναμένεται να διαβάξει ένα stream of values από ένα websocket στο ws://localhost:8888/hostname – αν και είναι αρκετά έξυπνο να αλλάξουμε το localhost:8888 εάν υπηρετεί αυτό μια άλλη τοποθεσία – και να κάνουμε plot αυτά σε πραγματικό χρόνο χρησιμοποιώντας highcharts.js. Ο πιθανός αναγνώστης ίσως παρατηρήσει ότι τα παραπάνω δεν είναι ακριβώς HTML αλλά περισσότερο ένα python formatted string. Αυτό συμβαίνει για 2 λόγους. Πρώτον γιατί το First script handles το configuration (βλέπε chart = new Highcharts.Chart(JSON.parse('%s' Δεύτερον, το πλάτος του graph υπολογίζεται σε page loadtime και το plot χρειάζεται να μετατιπισθεί για να δείξει μόνο τα πιο τελευταία σημεία.

```
$ for i in {1..100}; do echo $i; sleep 1; done | \
    awk -oL "{print_\$1/100}" | \
    python webmonitor.py
```

Αυτό θα παράγει σε διάστημα 1 δευτερολέπτου, αριθμούς από το 1 ως το 100. Μετά τα normalizes χρησιμοποιώντας awk και τα τροφοδοτεί σε webmonitor. Αφού αυτή η εντολή εκτελεσθεί μπορούμε να ανοίξουμε τον browser και να κάνουμε navigate στο localhost:8888.

Χρησιμοποιούμε αυτό για να ρυθμίσουμε από απόσταση το ολικό μέγεθος των δεδομένων που το mysql καταναλώνει.

3. xml-parse.sh

Άλλα αφαιρώντας συγκεκριμένα άρθρα διορθώνουμε το πρόβλημα του xerces. Αν τα άρθρα είναι απομονωμένα το επίσης error εξαφανίζεται. Το xml-parse.sh διαγράφει τα ζητούμενα άρθρα από το αρχείο xml.

```
xml-parse.sh <original-xml-file> <
    title_of_article_to_remove> [inplace]
```

αν το τελευταίο όρισμα είναι το inplace, τότε το page_remover.c θα χρησιμοποιηθεί για να καλύψει το άρθρο με κενά. Αυτή η διαδικασία είναι πολύ πιο γρήγορη. Διαφορετικά η σελίδα άπλα διαγράφεται και το αποτέλεσμα εμφανίζεται στο stdout. Μόλις το xml-parse.sh τελειώσει επιτυχώς μπορεί κάνεις να τρέξει:


```
java -jar tools/mwdumper.jar RESULTING_XML --format=  
sql:1.5 > SQL_DUMP
```

4. sql-clear.sh

Το `sql-clear.sh` είναι ένα μικρό bash script που "κουτσουρεύει" όλους τους πίνακες από την βάση δεδομένων. Με τον όρο "κουτσουρεύει" εννοούμε ότι αφήνει τα table schemata ανεπηρέαστα και διαγράφει όλα τα internal δεδομένα.

5. page_remove.c

Όπως προηγουμένως συζητήθηκε η xerces βιβλιοθήκη την οποία χρησιμοποιεί το `mwdumper` απέτυχε, φαινομενικά τυχαία να επεξεργαστεί κάποιες σελίδες. Για να διευθετηθεί αυτό το πρόβλημα αφαιρέσαμε τις σελίδες πλήρως και ξαναπροσπαθήσαμε. Επειδή αυτή η εργασία είναι εύκολη άλλα αργά γράψαμε ένα χαμηλού επιπέδου πρόγραμμα στη C για να το επιλύσουμε, το `page_remove.c`. Το `page-remover` δέχεται ως input the path του XML wikipedia dump, το offset του άρθρου που θέλουμε να καλύψουμε και το μέγεθος του άρθρου. Μετά χρησιμοποιεί το `mmap` system call για να αποκτήσει ψευδο-random-access στα δεδομένα μέσα στο αρχείο και γεμίζει το άρθρο με whitespace characters. Το `page_remove.c` δεν είναι threaded μας που το bottleneck είναι στο HDD IO speed και ο παραλληλισμός δεν θα βοηθούσε.

Chapter 16

Αυτοματισμός

Δημιουργώντας μια wikipedia mirror ίσως φαίνεται μια απλή διαδικασία αλλά συμπεριλαμβάνει πολλές αγκαθωτές λεπτομέρειες και επαναλαμβανόμενα tasks. Πολλαπλές μέθοδοι αυτοματισμού εφαρμόστηκαν για να ολοκληρώσουν μια μεγάλη ποικιλία tasks που συμπεριλαμβάνονται στην εκτέλεση.

1. Makefiles

Το πιο σημαντικό μέρος του αυτοματισμού της wikipedia-mirror είναι το make build system. Make είναι ένα build system όπου κάποιος μπορεί να δηλώσει τα απαιτούμενα αρχεία (targets), dependencies για αυτά, και ένα σύνολο από shell commands που θα χτίσουν αυτά τα targets. Κάθε target είναι ουσιαστικά μια finite state machine με δύο καταστάσεις:

- Ένα αρχείο που υπάρχει και είναι επικυρωποιημένο με τα dependencies και
- Ένα αρχείο που είτε δεν υπάρχει ή η modification date είναι παλαιότερη από αυτό ή τουλάχιστον ενός από τα dependencies.

Και μια σειρά από shell εντολές για την μεταφορά από την πρώτη στη δεύτερη κατάσταση.

Για παράδειγμα, σώζουμε το ακόλουθο ως Makefile σε ένα project που περιέχει τα αρχεία `foo.c`, `foo.h`, `bar.c` και `bar.h`:

αυτό σημαίνει ότι για να χτίσουμε το εκτελέσιμο `foobar` χρειαζόμαστε `foo.o` και `bar.o`. Και για να χτίσουμε `foo.o` και `bar.o` χρειαζόμαστε `foo.c` και `foo.h`, και `bar.c` και `bar.h` αντίστοιχα.

Επίσης παρέχουμε εντολές για να χτισθεί το `foo.o`, `bar.o` και `foobar`, οι οποίες είναι

- `gcc foo.c -c -o foo.o`
- `gcc bar.c -c -o bar.o`
- και `gcc foo.o bar.o -o foobar`

αντίστοιχα. παρατηρούμε ότι δεν υπάρχουν κανόνες για τα `.c` και `.h` αρχεία. Αυτό συμβαίνει γιατί το `make` πρέπει να αποτυγχάνει εάν δεν είναι παρόντα. Έτσι εάν τρέχουμε το `make foobar`, το `make` θα ελέγχει για την ύπαρξη του `foobar` και την ημερομηνία της τροποποίησης. Εάν το `foobar` λείπει ή η ημερομηνία τροποποίησης είναι προηγούμενη από τις εξαρτήσεις του (δηλαδή `foo.o` και `bar.o`) αυτό θα ξαναχτιστεί. Εάν κάποια από εξαρτήσεις απουσιάζει η ίδια λογική ισχύει και για αυτή. Με αυτό τον τρόπο εάν χτίσουμε μια φορά το `foobar`, και μετά τροποποιήσουμε το `bar.c` και ξανατρέξουμε `make foobar`, το `make` θα θεωρήσει αναδρομικά ότι:

- το `bar.o` είναι out of date όσον αφορά την εξάρτηση `bar.c`
- Όταν `bar.o` έχει πλέον μια πιο πρόσφατη ημερομηνία μετατροπής από το `foobar` και για αυτό το τελευταίο είναι out of date όσον αφορά την dependency του `bar.o`, έτσι χρειάζεται να ξαναχτιστεί.

με αυτόν τον τρόπο το `make` πετυχαίνει μια σχεδόν βέλτιστη στρατηγική για την επίτευξη κάθε φορά του ελάχιστου ποσοστού των απαιτούμενων στόχων. Τώρα που ξεκαθαρίσαμε την βασική λογική των `make` ας κάνουμε πιο σαφή μερικά από τα βασικά χαρακτηριστικά τους που κάνουν τη ζωή μας πιο εύκολη.

(a) Phony targets

Μερικές εργασίες δεν είναι αρχεία και χρειάζονται να τρέχουν κάθε φορά που το `make` τις συμπεριλαμβάνει στο dependency tree. Γι αυτά έχουμε ένα ειδικό keyword `.PHONY:`. Παρακάτω είναι ένα παράδειγμα.

```
.PHONY:
clean:
    rm -rf *
```

Αυτό λέει στο `make` ότι κανένα αρχείο ονομαζόμενο `clean` δε θα δημιουργηθεί τρέχοντας `rm -rf *`, και επίσης ακόμα και εάν υπάρχει ένα up-to-date ονομαζόμενο αρχείο ονομαζόμενο `clean`, αυτό το target θα τρέχει ανεξάρτητα.

Αξίζει να σημειώσουμε ότι οι phony εξαρτήσεις πάντα θα θεωρούνται out of date.

Για παράδειγμα:

```
.PHONY:
say-hello:
    echo "hello"

test.txt: say-hello
    touch test.txt
```

Όταν το `touch test.txt` θα τρέχει κάθε φορά που τρέχουμε `make test.txt` απλώς γιατί το `make` δεν μπορεί να γνωρίζει με βεβαιότητα ότι το phony target `say-hello` δεν άλλαξε τίποτε σημαντικό για το `test.txt`. Για αυτό το λόγο τα phony targets χρησιμοποιούνται για user facing tasks.

(b) Variables

Τα makefiles μπορούν να έχουν μεταβλητές ορισμένες με ποικίλους τρόπους. Μερικές περιπτώσεις που έχουν γίνει για να χρησιμοποιηθούν στην `wikipedia-mirror` παρουσιάζονται παρακάτω.

i. Simple variables

Μερικές φορές δεν είναι επιθυμητό για τις μεταβλητές να είναι expanded επ αόριστον:

```
kurma = the world $(support1)
animal1 = four elephants
animal2 = tortoise
support1 = supported by $(animal1) $(
    support2)
support2 := supported by a $(animal2) $(
    support2)

all:
    echo $(kurma)
```

Εδώ προσπαθούμε να δημιουργήσουμε ένα άπειρο μήνυμα.

```
$ make --just-print
Makefile:5: *** Recursive variable `support2'
' references itself (eventually). Stop.
```

το σύστημα μεταβλητών δηλαδή είναι κατά κάποιον τρόπο total, με άλλα λόγια η εύρεση της τιμής μεταβλητών μπορεί να είναι αναδρομική αλλά πρέπει να τερματίζει. Μπορούμε να αποφύγουμε αυτόν τον περιορισμό ορίζοντας μεταβλητές με :=:

```
make --just-print
echo the world supported by four elephants
      supported by a tortoise
```

ii. Automatic variables

Το Makefile επίσης ορίζει μερικές contextual μεταβλητές οι οποίες είναι ορισμένες. Οι πιο σημαντικές automatic variables που ορίζει το gnu make είναι οι ακόλουθες

- `$@`: Το όνομα του αρχείου του target. Εάν το target είναι ένα archive member, τότε `$@` είναι το όνομα του archive αρχείου. Στο pattern rule που έχει πολλαπλά targets, `$@` είναι το όνομα του οποιουδήποτε target που κάνει το rule's recipe να τρέχει.
- `%`: Το όνομα του target member, όταν το target είναι ένα archive member. Για παράδειγμα, εάν το target είναι `foo.a(bar.o)` τότε `%` είναι `bar.o` και `$@` είναι `foo.a`. `%` είναι άδειο όταν το target δεν είναι ένα archive member.
- `$<`: Το όνομα του πρώτου prerequisite. Εάν το target πήρε το recipe του από έναν implicit rule, αυτό θα είναι το πρώτο prerequisite που προστέθηκε από το implicit rule.
- `$?:` Τα ονόματα από όλες τις εξαρτήσεις που είναι νεότερα από το target, με κενά μεταξύ τους. Για τα prerequisites που είναι archive members, μόνο named member χρησιμοποιούνται (βλέπε Archives).
- `$^`: Τα ονόματα όλων των prerequisites, με κενά μεταξύ τους. Για τα prerequisites τα οποία είναι archive members, μόνο των named member χρησιμοποιείται. ένα target έχει μόνο ένα prerequisite σε κάθε άλλο αρχείο από το οποίο εξαρτάται, αναξαρτήτως από το πόσες φορές κάθε αρχείο είναι καταχωρημένο ως ένα no matter how many times each file prerequisite. Έτσι εάν τοποθετήσουμε στη λίστα ένα prerequisite για περισσότερο από μια φορά για ένα target, η value του `$^` περιέχει μόνο ένα αντίγραφο του ονόματος.

iii. Συναρτήσεις

Οι συναρτήσεις είναι παρόμοιες με μεταβλητές ως προς το ότι και αυτές γίνονται expand σε συμβολοσειρές. Η μόνη διαφορά είναι ότι επιδέχονται παραμέτρους.

```
greet = "Hello_␣$1_␣(from_␣$2)"
john-greets = $(call greet,$1,John)

.PHONY:
all:
    @echo $(call john-greets,Chris)
```

Εδώ η έξοδος είναι

```
$ make
Hello Chris (from John)
```

Chapter 17

Επιδόσεις

1. Compile time

2. Runtime

Το Compile time περιλαμβάνει το χρόνο που χρειάζεται για:

- Κατέβασμα όλων των στοιχείων του wikipedia server
- The bitnami stack
 - mwdumper
 - mediawiki-extensions
 - Εγκατάσταση και χτίσιμο αυτών των στοιχείων (~1 min)
 - Κατέβασμα των wikipedia dumps
 - Προεπεξεργασία των dumps (~10 mins)
 - Populating τη mysql βάση δεδομένων(~10 days)

Τα Builds έγιναν στο d Infolab's Ashmore. Τα system's specs είναι σχετικά ψηλά σε γενικές γραμμές αλλά το bottleneck ήταν το disk IO έτσι λιγότερο από 1% από τις υπόλοιπες διαθέσιμες πηγές χρησιμοποιήθηκαν κατά την διάρκεια του MySQL database population. Συγκεκριμένα τα χαρακτηριστικά του ashmore είναι:

- **CPU:** Xeon E5-1607 3GHz 4-Core 64 bit
- **Main memory:** 64G
- **HDD:** (spinning disk) 500GB + 2Tb

Εφ' όσον το βασικό bottleneck είναι η δημιουργία βάσης δεδομένων — δηλαδή οι επιδόσεις της MySQL — δόθηκε μεγάλη προσοχή και

πειραματισμός στη σωστή ρύθμιση της βάσης, άλλα η επιτάχυνση ήταν εν τέλει ελάχιστη και έτσι τα περισσότερα απ' όσα δοκιμάστηκαν δεν περιλήφθηκαν στα Makefiles.

Η backend engine που χρησιμοποιήσαμε για τη MySQL είναι η InnoDB. Μερικές από τις μεθόδους βελτιστοποίησης που επιχειρήθηκαν παρουσιάζονται παρακάτω.

- Ρύθμιση του `innodb_buffer_pool_size`. Ενώ η διαθέσιμη μνήμη του ashmore είναι αρκετά μεγάλη, αυξάνοντας το buffer pool μέχρι και κάποια GB δεν είχε σοβαρό αντίκτυπο στην επίδοση.
- Αλλάζοντας το `innodb_flush_method={{innodb_flush_method}}` `=O_DSYNC` για να αποφυγουμε κλήσεις στην `fsync`. Εν ολίγοις το πρόβλημα με την `fsync` είναι ότι ψάχνει σειριακά τις mapped σελίδες ενός αρχείου για dirty pages με αποτέλεσμα να γίνεται αργό για μεγάλα αρχεία.
- Ρυθμίζοντας το `innodb_io_capacity`. Εν τέλει η τιμή της μεταβλητής ήταν υψηλότερη από το bandwidth του σκληρού δίσκου

Η μόνη βελτιστοποίηση που είχε αισθητό αποτέλεσμα ήταν η αλλαγή του MySQL dump ώστε να θέτει

```
SET AUTOCOMMIT = 0; SET FOREIGN_KEY_CHECKS=0;
```

Αυτό επέτρεψε στο InnoDB να κάνει περισσότερη δουλειά στην κύρια μνήμη πριν επικοινωνήσει με το δίσκο και επίσης μείωσε τη συνολική δουλειά εμπιστευόμενος ότι οι τιμές των κελιών που αναφέρονταν σε άλλους πίνακες όντως έδειχναν κάπου.

Chapter 18

Παραρτήματα

1. Πηγαίοι κωδικές

(a) `page_remover.c`

```
/*
 * Copyright 2014 Chris Perivolaropoulos <
 *   cperivol@csail.mit.edu>
 *
 * This program is free software: you can
 *   redistribute it and/or
 *   modify it under the terms of the GNU General
 *   Public License as
 *   published by the Free Software Foundation,
 *   either version 3 of the
 *   License, or (at your option) any later
 *   version.
 *
 * This program is distributed in the hope that
 *   it will be useful, but
 *   WITHOUT ANY WARRANTY; without even the
 *   implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR
 *   PURPOSE.
 *
 * See the GNU General Public License for more
 *   details. You should
 *   have received a copy of the GNU General
 *   Public License along with
 *   this program.
 *
```

```

* If not, see <http://www.gnu.org/licenses/>.
*
* This should fill a range in a file with
  spaces. This is an in-place
* operation so it should be pretty fast.
*
* Usage: page_remover PATH OFFSET LENGHT
*/

#include <assert.h>
#include <fcntl.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <semaphore.h>
#include <unistd.h>
#include <unistd.h>

#define USAGE_INFO "page_remover PATH OFFSET \
LENGTH"
#define PRINT(ctx, args...) do { sem_wait(&ctx->
  stdio_mutex); \
                                printf(args);
                                \
                                fflush(stdout);
                                \
                                sem_post(&ctx->
                                  stdio_mutex);
                                \
  } while(0)

typedef struct context {
  int fd;
  size_t size;
  off_t off;
  sem_t stdio_mutex;
  void* data;
} context_t;

context_t* context_init(char* fname, off_t off,

```

```

size_t len)
{
    context_t * ctx = (context_t*)malloc(sizeof(
        context_t));
    off_t pa_off = off & ~(sysconf(_SC_PAGE_SIZE
        ) - 1);

    sem_init(&ctx->stdio_mutex, 0 /* Shared.
        Usually ignored */, 1);

    PRINT(ctx, "Opening %s at %lu (len: %lu)\n",
        fname, off, len);

    ctx->off = off-pa_off;
    ctx->fd = open(fname, O_RDWR, 0x0666);
    if (ctx->fd == -1) {
        perror("open");
        return NULL;
    }

    ctx->size = len;
    ctx->data = mmap(0, len+ctx->off, PROT_READ
        | PROT_WRITE,
        MAP_SHARED, ctx->fd, pa_off);
    if (ctx->data == MAP_FAILED) {
        perror ("mmap");
        return NULL;
    }

    return ctx;
}

void context_destroy(context_t* ctx)
{
    if (close (ctx->fd) == -1)
        perror ("close");

    if (munmap ((void*)ctx->data, ctx->size) ==
        -1)
        perror ("munmap");

    sem_destroy(&ctx->stdio_mutex);
    free(ctx);
}

```

```

int main(int argc, char *argv[])
{
    if (argc != 4)
        fprintf(stderr, USAGE_INFO);

    context_t *ctx = context_init(argv[1], atoi(
        argv[2]), atoi(argv[3]));

    /* You MIGHT want to thread this but I dont
       think it will make
       * much more difference than memset. */
    memset(ctx->data + ctx->off, '\0', ctx->size)
        ;

    context_destroy(ctx);
    return 0;
}

```

(b) utf8thread.c

```

#include <assert.h>
#include <fcntl.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <semaphore.h>
#include <unistd.h>
#include <unistd.h>

sem_t stdio_mutex;

#define PRINT(args...) do {sem_wait(&stdio_mutex
    ); \
    printf(args); \
    fflush(stdout); \
    sem_post(&stdio_mutex); \
} while(0)

```

```

/* #define DEBUG(args...)          PRINT(
    args) */
#define DEBUG(...)

#define DEFAULT_CHAR '□'
#define WORKERS 8
#define MESSAGE_DENSITY 1000000000

typedef unsigned long long u64;

#define UTF_LC(1) ((0xff >> (8 - (1))) << (8 - (
    1)))
#define UTF_CHECK(1, c) (((UTF_LC(1) & (c)) ==
    UTF_LC(1)) && (0 == ((c) & (1 << (7-(1))))))

#define UTF_LEN(x) (UTF_CHECK(6, x) ? 6 :
    \
        UTF_CHECK(5, x) ? 5 : \
        UTF_CHECK(4, x) ? 4 : \
        UTF_CHECK(3, x) ? 3 : \
        UTF_CHECK(2, x) ? 2 : -1)

struct crange {
    u64 start, end;
};

/* Get return the next character after the last
   correct one. */
inline u64 valid_utf8(u64 c)
{
    char i;
    /* Ascii */
    if ((* (char *)c & 0x80) == 0)
        return c+1;

    /* */
    for (i = UTF_LEN(* (char *)c)-1; i>0; i--) {
        c++;
        if (!UTF_CHECK(1, * (char *)c)) {
            return (u64)NULL;
        }
    }
}

```

```

        return i<0 ? 0 : c+1;
    }

void* fix_range(void* _r)
{
    struct crange* r = _r;
    u64 tmp, id = r->start;
    long long unsigned count = 0;

    while ((u64)r->start < (u64)r->end) {
        if (count++ % MESSAGE_DENSITY == 0)
            printf ("[worker: 0x%016llx] Done with
                    0x%lluK.\n", id, count % 1024);

        if (!(tmp = valid_utf8(r->start))){
            PRINT("Invalid char 0x%x (next: 0x%x)\n",
                *(char*)r->start, *(char*)(r->
                    start+1));
            *((char*)r->start) = DEFAULT_CHAR;
            (r->start)++;
        } else {
            r->start = tmp;
        }
    }

    PRINT ("[worker: 0x%016llx] OUT\n", id);
    return NULL;
}

void run(u64 p, u64 sz)
{
    int n, i;
    u64 wsize;
    pthread_t workers[WORKERS];
    struct crange rngs[WORKERS];

    wsize = sz/WORKERS + 1;
    printf("Base address: 0x%016llx, step size: 0x%016llx\n", p, wsize);

    for (i=0; i<WORKERS; i++){
        rngs[i].start = p + wsize*i;
        rngs[i].end = p + wsize*i + wsize;
    }
}

```

```

        PRINT("Spawning_worker_␣d_␣on_␣range_␣[0x%016
            llx,␣0x%016llx),␣%llu_␣bytes...", i,
            rngs[i].start, rngs[i].end, wsize);
        if ((n = pthread_create(workers+i, NULL,
            fix_range, (void*)(rngs+i)))) {
            PRINT("FAIL\n");
            perror("worker");
            return;
        }
        PRINT("OK\n");
    }

    PRINT ("Wrapping_up...\n");
    for (i=0; i<WORKERS; i++) {
        PRINT ("Joining_worker_␣d...", i);
        pthread_join(workers[i], NULL);
        PRINT ("OK\n");
        PRINT("Worker_␣d_␣went_␣through_␣llu_␣bytes.\n
            n",
            i, (u64)rngs[i].end - (u64)rngs[i].
                start);
    }
}

int main(int argc, char *argv[])
{
    int fd;
    long long int sz, p;
    struct stat buf;

    sem_init(&stdio_mutex, 0 /* Shared. Usually
        ignored */ , 1);

    fd = open(argv[1], O_RDWR, 0x0666);
    if (fd == -1) {
        perror("open");
        return 1;
    }

    fstat(fd, &buf);
    sz = buf.st_size;
    printf("File_␣size:␣0x%016llx\n", sz);

```

```

p = (u64)mmap (0, buf.st_size, PROT_READ |
    PROT_WRITE , MAP_SHARED, fd, 0);
if (p == -1) {
    perror ("mmap");
    return 1;
}

run(p, buf.st_size);

if (close (fd) == -1) {
    perror ("close");
    return 1;
}

if (munmap ((void*)p, buf.st_size) == -1) {
    perror ("munmap");
    return 1;
}

sem_destroy(&stdio_mutex);

return 0;
}

```

(c) sql-clear.sh

```

#!/bin/bash
MUSER="$1"
MPASS="$2"
MDB="$3"
MYSQL=$4

# Detect paths
AWK=$(which awk)
GREP=$(which grep)

if [ $# -ne 4 ]
then
    echo "Usage: $0 {MySQL-User-Name} {MySQL-
        User-Password} {MySQL-Database-Name} {
        MySQL_executable_to_use}"
    echo "Drops all tables from a MySQL"
    exit 1

```



```

fi

TABLES=$(($MYSQL -u $MUSER -p$MPASS $MDB -e 'show
        tables' | $AWK '{ print $1}' | $GREP -v '^
        Tables' )

for t in $TABLES
do
    echo "Clearing $t table from $MDB database
    ..."
    $MYSQL -u $MUSER -p$MPASS $MDB -e "
        truncate $table $t"
done

```

(d) webmonitor.py

```

"""
Just feed pairs of

<epoch date> <float value>

or even just

<float value>

One way to do that would be

$ <cmd> <stdbuf -oL awk '{print \ $1/$$max}' |
python webmonitor.py

and I will plot them on port 8888. This will
also pipe the input right
out to the output. Strange input will be ignored
and piped this way,
but this needs to be done by awk as well in the
above example.
"""

import sys
import json
import time

from threading import Thread
from collections import deque

```

```

import tornado.websocket as websocket
import tornado.ioloop
import tornado.web

HTML = """
<!DOCTYPE_HTML_PUBLIC "-//W3C//DTD_HTML_4.01//EN
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="
    text/html; charset=utf-8">
<title>DrNinjaBatmans Websockets</title>

<script type="text/javascript" src="http://
    code.jquery.com/jquery-1.10.1.js"></script>
<script type="text/javascript" src="http://
    code.highcharts.com/highcharts.js"></script>

<script>
var chart; // global
var url = location.hostname + ':' + (parseInt(
    location.port));
var ws = new WebSocket('ws://' + url + '/
    websocket');
ws.onmessage = function(msg) {
    add_point(msg.data);
};

// ws.onclose = function() { alert('Connection
    closed. '); };

var add_point = function(point) {
    var series = chart.series[0],
        shift = series.data.length > 100;
    chart.series[0].addPoint(eval(point), true,
        shift);
};

$(document).ready(function() {
    chart = new Highcharts.Chart(JSON.parse('%s
    '));
});
</script>

```

```

</head><body><div id="container" style="width:
    800px; height: 400px; margin: 0 auto"></div
></body></html>
"""

config = {
    'visible_points': 10,
    'py_chart_opts': { 'chart': { 'renderTo': '
        container',
                                '
                                defaultSeriesType
                                ': 'spline
                                '},
    'title': { 'text': '
        DrNinjaBatmans_data'
    },
    'xAxis': { 'type': '
        datetime',
                '
                tickPixelInterval
                ': '150'},
    'yAxis': { 'minPadding':
        0.2,
                'maxPadding':
                0.2,
                'title': { '
                    text': '
                    Value',
                    '
                    margin
                    ':
                    :
                    80}

                },
    'series': [{ 'name': '
        Data',
                  'data':
                  []}]
}

def date_float(s):
    try:

```

```

        date, val = s.split()
    except ValueError:
        val = s.strip()
        date = time.time()

    return int(date), float(val)

def send_stdin(fn=date_float):
    for raw in sys.stdin:
        sys.stdout.write(raw)

        # Ignore strange input.
        try:
            jsn = json.dumps(fn(raw))

            buf.append(jsn)

            for w in websockets:
                try:
                    w.write_message(jsn)
                except websocket.
                    WebSocketClosedError:
                        pass

            except:
                pass

        for ws in websockets:
            ws.close()

class StdinSocket(websocket.WebSocketHandler):
    def open(self):
        for i in buf:
            self.write_message(i)

        websockets.append(self)

    def close(self):
        websockets.remove(self)

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write(HTML % (int(config['

```

```

        visible_points']],
                                json.dumps(config['
                                    py_chart_opts']))
                                )

if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/", MainHandler),
        (r'/websocket', StdinSocket),
    ])
    buf = deque(maxlen=int(config['
        visible_points']))
    websockets = []

    config['args'] = []
    for a in sys.argv[1:]:
        if '=' in a:
            k, v = a.split('=', 1)
            config[k] = v
        else:
            config['args'].append(a)

    Thread(target=send_stdin).start()
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()

```

(e) xml-parse.sh

```

#!/bin/bash
#
# Simply removing specific articles fixes the
# xerces error with
# UTF8. If the articles are alone the error goes
# away
# aswell. Extremely weird but that's life.
# Fortunately the article is
# just a stub about some toad (Cranopsis
# bocourti)
#
# xml-parse.sh ORIGINAL_XML
# TITLE_OF_ARTICLE_TO_REMOVE [inplace]
#

```

```

# if `inplace` is there the c program will be
# used to cover the article
# with spaces. This is much faster. Should be
# anyway. Otherwise the
# page is just ommited and the result is dumped
# in stdout. Helping
# messages are dumped in stderr After this you
# can run:
#
# java -jar tools/mwdumper.jar RESULTING_XML --
# format=sql:1.5 > SQL_DUMP

set -e
set -o pipefail

if [[ $# -lt 2 ]]; then
    echo "xml-parse.sh ORIGINAL_XML
        TITLE_OF_ARTICLE_TO_REMOVE [inplace]"
    exit 0
fi

function my_dd {
    coreutils_version=$(dd --version | head -1 |
        cut -d\ -f3 | colrm 2 2 )
    if [[ $coreutils_version -ge 822 ]]; then
        eval "dd iflag=count_bytes iflag=direct
            oflag=seek_bytes ibs=1M $@"
    else
        echo "Your coreutils may be a bit old (
            $coreutils_version). 822 is the one
            cool kids use." >&2
        eval "dd $@ ibs=1"
    fi
}

ORIGINAL_XML=$1

# Dump a part of the file in sdout using dd.
# Usage:
# file_range <filename> <first_byte> <start/end/
# length>
#
# Length can be negative
function file_range {

```

```

file=$1
start=$2
len=$3

case $len in
    "end") my_dd if=$file skip=$start || exit
           1; return 0;;
    "start") my_dd if=$file count=$start ||
             exit 1; return 0;;
    "") echo "len was empty (file: $file,
             start: $start, len: $len). Correct
             format <filename> <byte start> <length>
             | 'start' | 'end' >" 1>&2; exit 1;;
    *) ;;
esac

if [[ $len -gt 0 ]]; then
    # Dump to stdout
    my_dd if=$file skip=$start count=$len ||
        exit 1
else
    skip=$(( $start + ( $len ) ))
    len=$(( - ( $len ) ))

    if [[ $skip -lt 0 ]]; then
        skip=0
        len=$start
    fi

    # Dump to stdout
    my_dd if=$file skip=$skip count=$len ||
        exit 1
fi
}

function backwards {
    tac -b | rev
}

function byte_offset {
    grep -b -o -m 1 -F "$1" | cut -d : -f1
}

# Throw everything but the page in stdout

```

```

#
# neg_xml_page "Barack Obama"
function neg_xml_page {
    term="$1</title>"
    title_offset=$(cat $ORIGINAL_XML |
        byte_offset "$term")
    echo -e "\n\tMethod:_$2(blank_is_ok)" 1>&2
    echo -e "\tsearch_term:_$term" 1>&2
    echo -e "\tfile:_$ORIGINAL_XML" 1>&2
    echo -e "\ttitle_offset:_$title_offset" 1>&2

    # Fail the term is invalid
    if [ -z "$title_offset" ]; then
        echo "Found_'$title_offset'_Grep-ing_(cat_
            _$ORIGINAL_XML_|_grep_-b_-m_1_-F_\
            $term\"_|_cut_-d:_-f1)" 1>&2
        exit 1
    fi

    to_page_start=$((($(file_range $ORIGINAL_XML
        $title_offset -1000 | backwards |
        byte_offset "$(echo_'<page>'_|_rev)")+7)
    )
    echo -e "\tto_page_start_(relative):_
        $to_page_start" 1>&2

    file_range $ORIGINAL_XML $title_offset end |
        byte_offset "</page>" >&2
    echo $((($(file_range $ORIGINAL_XML
        $title_offset end | byte_offset "</page>"
        )+7)) >&2
    to_page_end=$((($(file_range $ORIGINAL_XML
        $title_offset end | byte_offset "</page>"
        )+7)) # len('</page>') == 7
    echo -e "\tto_page_end_(relative):_
        $to_page_end" 1>&2

    page_start=$((($title_offset - $to_page_start
        +1 ))
    echo -e "\tpage_start:_$page_start" 1>&2

    page_end=$((($title_offset + $to_page_end))
    echo -e "\tpage_end:_$page_end" 1>&2

    echo -e "\tbytes_to_copy:_$(($(du_-b_
</pre>
</div>
<div data-bbox="484 868 508 883" data-label="Page-Footer">
<p>87</p>
</div>
```



```

$ORIGINAL_XML | cut -f1) - $page_start +
$page_end))" 1>&2

echo "Going to copy $page_start bytes" 1>&2
file_range $ORIGINAL_XML $page_start start
echo "Finished the first half up to
$page_start, $(du -b $ORIGINAL_XML |
cut -f1) - $page_end) to go" 1>&2
file_range $ORIGINAL_XML $page_end end
echo "Finished the whole thing." 1>&2
}

# Put stdin betwinn mediawiki tags and into
stdout
function mediawiki_xml {
    (head -1 $ORIGINAL_XML; sed -n "/<siteinfo
    >/,/</siteinfo>/p;/</siteinfo>/q"
    $ORIGINAL_XML ; cat - ; tail -1
    $ORIGINAL_XML )
}

# 1: XML File
# 2: Article
# 3: Method (leave blank)
# Assert that the file is there and is not empty
fsize=$(du -b $ORIGINAL_XML | cut -f1)
if [[ 0 -eq $fsize ]]; then
    echo "ERROR: empty xml file $ORIGINAL_XML"
    1>&2
    exit 1
fi

echo "Will remove article '$2' from file $1 (
size: $fsize)" 1>&2
if ! neg_xml_page "$2" "$3"; then
    ret=$?
    echo "XML parsing script failed" 1>&2
    exit $ret;
fi

```

Part V

References