Εξορύττοντας σχεσιακά δεδομένα από την Wikipedia

του
Χρήστος Περιβολαρόπουλος

Μια διπλωματική διατριβη.

Επιβλέποντες καθηγητές:
Κυριάκος Σγάρμπας, Boris Katz

University of Patras

Δευτέρα 23 Μαΐου 2016

Περίληψη

Το START (SynTactic Analysis using Reversible Transformations) είναι ένα πακέτο λογισμικού γραμμένο σε γλώσσα Common Lisp το οποίο αντλεί πληροφορίες από διαδικτυακές πηγές και τις χρησιμοποιεί για απαντήσει σε αυθαίρετες ερωτήσεις που δέχεται. Αναπτύχθηκε στο εργαστήριο Infolab του MIT. Για τον εμπλουτισμό των πληροφοριών που χρησιμοποιεί το START χρησιμοποιείται το πακέτο λογισμικού Omnibase μέσω του οποίου επιτυγχάνεται πρόσβαση του START σε πολλαπλές πήγες στο διαδίκτυο. Στην παρούσα εργασία αναπτύσσουμε μια επέκταση του Omnibase που επιτρέπει την πρόσβαση του START στην wikipedia και στις πληροφορίες που αυτή περιέχει. Η επέκταση αυτή του Omnibase ονομάζεται wikipediabase. Για την ευκολότερη πρόσβαση του START στη Wikipedia δημιουργήσαμε ένα πρόγραμμα (wikipedia-mirror) που δημιουργεί κλώνους της Wikipedia που αποθηκεύονται τοπικά ανεξάρτητα από το διαδίκτυο. Έτσι επιτυγχάνεται ταχύτερη και πιο αξιόπιστη πρόσβαση του START στο σύνολο των δεδομένων της wikipedia.

Abstract

START (SynTactic Analysis using Reversible Transformations) is a piece of sofware written in common lisp that retrieves information from internet resources and uses them to answer to arbitrary natural language questions. It was developed in InfoLab of MiT. For the enrichment of the retrieved information it uses the Omnibase software through which START gets access to multiple sources on the internet. In the present thesis we present an extension to Omnibase that allows START to get access to wikipedia and the information that it contains. This extension to Omnibase is called WikipediaBase. For easier access to wikipedia we also developed a separate program (wikipedia-mirror) that creates clones of wikipedia running locally and independently to the internet. This way faster and more reliable access to wikipedia is accomplished.

“Elegance is not a dispensable luxury but a quality that decides between success and failure.”

- Edsger W. Dijkstra

Ευχαριστίες

Πρώτα απ ολα θα ηθελα να ευχαριστήσω τους επιβλέποντες μου κ. Κυριάκο Σγάρμπα και Dr Boris Katz. Χωρίς την ανεκτίμητη βοηθεια τους η εργασία αυτή δε θα γινόταν πραγματικότητα.

Αδράττω αυτή την ευκαιρία για να εκφράσω τις ευχαριστίες μου σε όλα τα μέλη της Ακαδημαϊκής κοινότητας του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών και του MiT InfoLab για την υποστήριξή τους κατά όλη τη διάρκεια των σπουδών μου.

Τέλος ευχαριστώ τους γονείς μου και όλους όσους με στήριξαν σε αυτή μου την προσπάθεια.

Πρόλογος

Το τεχνικό μερος αυτης της εργασίας (development) εκπονήθηκε κατα το έτος 2014-2015 στο MiT InfoLab CSAIL υποτ την πείβλεψη των Dr Κυριάκου Σγάρμπα του Πανεπιστημίου Πατρών και dr Boris Katz του MiT CSAIL. Η συγγραφή του κειμένου πραγματοποιήθηκε στις αρχές του έτου 2016. Το WikipediaBase ενσωματώθηκε στο δικτυο του CSAIL προκειμένου να χρησιμοποιηθεί απο το START στα μέσα του έτους 2015.

Περιεχόμενα

Μέρος Ι

Εισαγωγή

Η παρούσα εργασία έχει σαν στόχο να βελτιώσει τη δυναντότητα απάντησης ερωτήσεων σε φυσική γλώσσα χρησιμοποιώντας γνώση αντλημένη από τη Wikipedia. Το καταφέρνει αυτό δημιουργόνας ένα software σύστημα που επιτρέπτει στο σύστημα START του MiT InfoLab να αντλήσει πληροφορίες από τη wikipedia. Οι μέχρι τώρα δυνατότητες άντλησης πληροφορίας από τη wikipedia ήταν περιορισμένη του απαρχαιομένου προκατόχου.

Το START Natural Language System[2] είναι ένα σύστημα λογισμικού που έχει σχεδιαστεί για να απαντά σε ερωτήσεις που τίθενται σε αυτό σε φυσική γλώσσα. Το START σαρώνει την εισερχόμενη ερώτηση, αναλύει τη δομή της και προσπαθεί να εντοπίσει παρόμοιες δομές στη γνωστική του βάση δεδομένων με στόχο να βρει απάντηση στην ερώτηση που του έχει τεθεί. Με τον τρόπο αυτό, το START παρέχει σε μη εξειδηκευμένους χρήστες γρήγορη πρόσβαση σε γνώση, η ευρεση της οποίας σε πολλές περιπτώσεις μπορεί να είναι χρονοβόρα ακόμα και για τον ειδικό.

Το START 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 επικεφαλή τον Dr Katz. Το START για πρώτη φορά συνδέθηκε με το World Wide Web το Δεκέμβριο του 1992, και με τις διάφορες μορφές του έχει μέχρι σήμερα απαντήσει σε εκατομμύρια ερωτήσεις χρηστών από όλο τον κόσμο.

Το START μπορεί να χειριστεί μεγάλη ποικιλία μέσων, συμπεριλαμβανομένων κειμένων, διαγραμμάτων, εικόνων, βίντεο και ήχων, ιστοσελίδων και άλλων χρησιμοποιώντας μια βασική τεχνική που ονομάζεται ”natural language annotation”. Κατά την τεχνική αυτή παίρνουμε ως είσοδο μια αντιστοιχία ανάμεσα σε αυθαίρετα δεδομένα και φράσεις ή προτάσεις που τα περιγράφουν και χρησιμοποιούμε τις φράσεις ή προτάσεις αυτές για να απαντηθούν οι ερωτήσεις του χρήστη με τα αντίστοιχα δεδομένα.

Η επεξεργασία της φυσικής γλώσσας του START αποτελείται από δύο μονάδες που αναγνωρίζουν την ίδια σύνταξη. Αυτές οι δυο μονάδες είναι: Η μονάδα κατανόησης και η μονάδα παραγωγής γλώσσας. Η μονάδα κατανόησης αναλύει το αγγλικό κείμενο και παράγει μια γνωστική βάση που κωδικοποιεί τις πληροφορίες του κειμένου. Η μονάδα παραγωγής γλώσσας παράγει αγγλικές προτάσεις λαμβάνοντας υπόψη μόνο ένα κατάλληλο τμήμα της γνωστικής βάσης. Αυτές οι μονάδες σε συνδυασμό με την τεχνική της ”natural language annotation”, δίνουν την δυνατότητα της παραγωγής απαντήσεων σε φυσική γλώσσα.

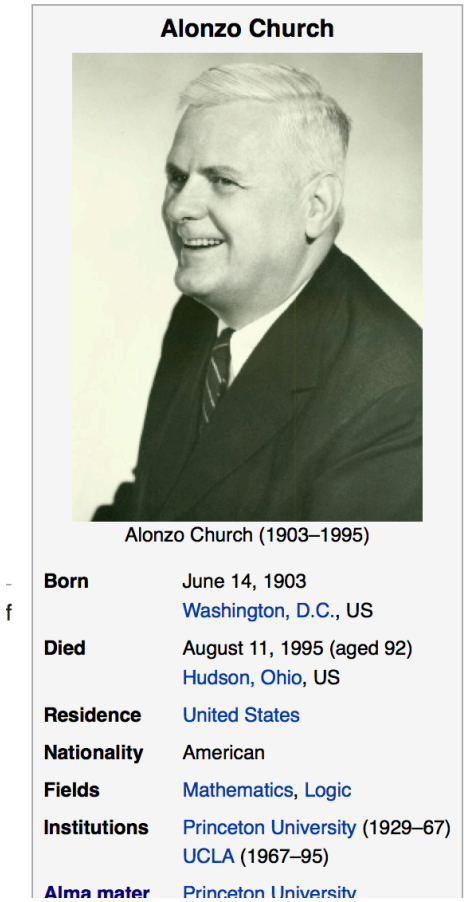
Το Omnibase[3] είναι μια ”εικονική” βάση δεδομένων που το START χρησιμοποιεί για να έχει ομοιογενή πρόσβαση σε ετερογενείς πήγες γνώσης. Για παράδειγμα δίνει ομογενή πρόσβαση στο imdb[**?**] και στο CIA world factbook[**?**]. Το Omnibase αναπτύχθηκε για πρώτη φορά το 2002, περίπου το ίδιο χρονικό διάστημα που η wikipedia έκανε την πρώτη της εμφάνιση (2001).

Η Wikipedia[**?**] είναι μία διαδικτυακή εγκυκλοπαίδεια που χρησιμοποιείται σε 250 γλώσσες. Το όνομά της έχει 2 συνθετικά την λέξη Wiki που είναι ένα είδος ομαδικού ιστότοπου και τη λέξη pedia που σημαίνει εγκυκλοπαίδεια. Στις 15 Ιανουαρίου 2001 ξεκίνησε η συγγραφή της σαν συμπλήρωμα στη γραμμένη από ειδήμονες Nupedia με ιδρυτή τον Τζίμη Γουέιλς. Η αγγλική wkipedia περιλαμβάνει πάνω από 4.000.000 άρ-

θρα. Η wikipedia έγινε σταδιακά δημοφιλής και διάφορα άλλα συναφή προγράμματα αναπτύχθηκαν όπως το Wiktionary, τα Wikibooks, τα Wikinews και τα Wikivoyage. Στη wikipedia εκφράζονται διάφορες απόψεις αλλά γίνεται προσπάθεια να τηρηθεί μια ουδέτερη στάση των συντακτών απέναντι στα γεγονότα. Το κύρος της έχει πολλές φορές αμφισβητηθεί και έχει επικριθεί για την αξιοπιστία της λόγο της ανοιχτής φύσης της. Παρόλα αυτά έχει χρησιμοποιηθεί επανειλημμένως από την ακαδημαϊκή κοινότητα. Οι συντάκτες της wikipedia είναι εθελοντές χρήστες. Ο καθένας που χρησιμοποιεί το ιντερνετ μπορεί να επεξεργαστεί ή να γράψει άρθρα στην wikipedia. Περιέχει περισσότερα από 30.000.000 άρθρα. Οι χρήστες της υπερβαίνουν τα 685.000.000 ετησίως. Άνθρωποι όλων των ηλικιών, κοινωνικού ή πολιτισμικού περιβάλλοντος μπορεί να συμμετέχουν στη συγγραφή χωρίς να έχουν συγκεκριμένα προσόντα. Τα παλαιότερα άρθρα της είναι πιο πλήρη σε αντίθεση με τα νεότερα που υπόκεινται σε συνεχείς αλλαγές. Επειδή ο καθένας μπορεί να προσθέσει πληροφορίες, οι λανθασμένες ή ανακριβείς πληροφορίες αφαιρούνται ή διορθώνονται από επόμενους χρήστες. Διαφέρει από μια έντυπη πηγή στο ότι ανανεώνεται συνεχώς ακολουθώντας την επικαιρότητα μέσα σε ελάχιστο χρονικό διάστημα που μερικές φορές φτάνει τα δευτερόλεπτα.

Λόγω της πολυπλοκότητας και της ιδιαίτερα αδόμητης φύσης της wikipedia, αντί του omnibase αναπτύξαμε μια ξεχωριστή υπηρεσία, το WikipediaBase, για να την καταστήσουμε προσβάσιμη από το START. Το wikipediaBase αποτελεί και το αντικείμενο της παρούσας διατριβής. Επιπλέον, για να αποφευχθεί ο κορεσμός του wikiedpedia.org, δημιουργήσαμε το wikipedia-mirror, που δημιουργεί κλώνους της wikipedia.

Σε πολλά άρθρα της Wikipedia συναντάμε μια δομή, το infobox, που χρησιμοποιείται συχνά από το WikipediaBase. Τα Infoboxes είναι πίνακες που χρησιμοποιούνται συνήθως στη wikipedia για να παρέχουν την περίληψη ενός άρθρου με ήμι δομημένο τρόπο. Τα Infoboxes είναι η κύρια πηγή πληροφοριών για τη WikipediaBase



Σχήμα 1: Ένα παράδειγμα ενός infobox

Σε ορους mediawiki markup, ένα infobox είναι ένα typed template που αποδίδεται σε html. Για παράδειγμα:

```
{{Infobox scientist
| name              = Gerhard Gentzen
| image             = Gerhard Gentzen.jpg
| image_size        =
| alt               =
| caption           = Gerhard Gentzen in Prague, 1945.
| birth_date        = {{Birth date|1909|11|24}}
| 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, αναφερόμενοι και ως κλάσεις, είναι οργανωμένοι με μια αρκετά ευρεία ιεραρχία[10]. Για παράδειγμα `Template:Infobox Austrian district` είναι μια ειδική περίπτωση ενός `Template:Infobox settlement` και το καθένα μετατρέπεται από mediawiki markup σε HTML (rendering) με διαφορετικό τρόπο.

Ένα άρθρο μπορεί να έχει περισσότερα από ένα infoboxes, για παράδειγμα, το άρθρο για τον Bill Clinton έχει δύο infobox: ένα για `Infobox Officeholder` και ένα για `Infobox President`.

Η γλώσσα που χρησιμοποιήθηκε για τη WikipediaBase είναι η Python. Η python είναι μια γλώσσα προγραμματισμού που χαρακτηρίζεται ως υψηλού επιπέδου. Δημιουργήθηκε το 1990 από τον Ολανδό Guido van Rossum και πήρε το όνομά της από την ομάδα άγγλων κωμικών Monti Python. Τα ιδιαίτερα χαρακτηριστικά της είναι ότι έχει πολλές βιβλιοθήκες, είναι γρήγορη στην εκμάθηση, είναι εύκολη στη χρήση κια λόγο του εύχρηστου συντακτικού της επιτρέπει στον χρήστη να γράψει τις ίδιες έννοιες σε λιγότερες γραμμές κώδικα από ότι σε άλλες γλώσσες όπως η C++ ή. Java.Μπορεί να εγκατασταθεί σε ευρεία γκάμα συστημάτων. Η Python Softoware Foundation διαχειρίζεται την python που αναπτύσσεται σαν ανοιχτό λογισμικό.

Μέρος II

Wikipediabase

Η WikipediaBase είναι μια πηγή πληροφοριών χρησιμοποιούμενη από το START. Είναι υπεύθυνη για την παροχή πρόσβασης σε πληροφορίες που σχετίζονται με την wikipedia. Μιμείται τον τρόπο επικοινωνίας του το Omnibase. Μέτα την αρχική της έκδοση η WikipediaBase έχει ξαναγραφεί δυο φορές. Η αρχική έκδοση ήταν γραμμένη σε Java. Στη συνέχεια ξαναγράφτηκε σε Ruby διατηρώντας την αρχική αρχιτεκτονική και το σχεδιασμό, και η παρούσα έκδοση ως αντικείμενο της παρούσας διατριβής είναι γραμμένη σε σε python με νέο σχεδιασμό και αρχιτεκτονική.

Υπάρχουν δύο βασικοί λόγοι για την τελευταία επανεγγραφή: η Python επελέγη διότι διδάσκεται ως προπτυχιακό μάθημα στο MIT, και ως εκ τούτου, ένα πρόγραμμα σε Python θα κάνει την εισαγωγή νέων φοιτητών του MIT στην ομάδα εργασίας ομαλότερη. Ο δεύτερος και πιο σημαντικός όμως λόγος για την επανεγγραφή είναι ότι ενώ ο αρχικός σχεδιασμός του προηγούμενου WikipediaBase ήταν στην αρχή επαρκής, στη συνέχεια η WikipediaBase μεγάλωσε σε τέτοιο σημείο, όπου ο κώδικας περιείχε πολλές ειδικές περιπτώσεις και ήταν δύσκολο να κατανοηθεί.

Το WikipediaBase σε python αρχικά γράφτηκε από τον Χρήστο Περιβολαρόπουλο σε στενή συνεργασία με την Dr Sue Felshin και τελικά παραδόθηκε στους Sue Felshin, Alvaro Morales και ton Michael Silver. Αργότερα και άλλοι φοιτητές εντάχθηκαν στο έργο.

Κεφάλαιο 1

Λειτουργικότητα

Στην επικοινωνία με το START υπεισέρχονται κάποιες βασικές έννοιες:

- `symbol` είναι μια έννοια στην οποία μπορούμε να αναφερθούμε.
- `attribute` είναι ένα χαρακτηριστικά ενός `symbol`.
- `class` είναι σύνολα που περιέχουν διάφορες έννοιες. Κάθε έννοια μπορεί να εντάσσεται σε παραπάνω από ένα class. Όλα τα symbols που εντάσσονται σε ένα class υποστηρίζουν ένα συγκεκριμένο σύνολο από attributes.

Υπάρχει ένα προς ένα αντιστοιχία αυτών των εννοιών με τις βασικές έννοιες του infobox. Δηλαδή το άρθρο στο οποίο αναφέρεται ένα infobox είναι ένα `symbol`, το `class` αυτού του `symbol` είναι η κλάση του infobox και τα χαρακτηριστικά που εκφράζει το infobox είναι τα `attributes`.

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 s-expressions. s-exrpressions είναι εκφράσεις που έχουν τη μορφή `(a b c (d e) f g)`.

Η WikipediaBase παρέχει τις ακόλουθες λειτουργίες:

1. Η εντολή `get`

   Δεδομένης μιας class, ενός symbol, και ενός τυποποιημένου attribute, δηλαδή ενός χαρακτηριστικού με typecode, η εντολή `get` επιστρέφει την τιμή του attribute. 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).

(α΄) Typecodes

Οι get εντολές πρέπει να επιστρέφουν μια λίστα από τυποποιημένες τιμές, δηλαδή ενα ζευγάρι τιμής - typecode. Έγκυρα typecodes είναι:

i. `:HTML`
Μια συμβολοσειρά προσαρμοσμένη για μετατροπή σε 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 Π.Χ. ημερομηνίες.
(Οι μη αναλυμένες ημερομηνίες αντιπροσωπεύονται σαν HTML strings χρησιμοποιώντας το `:HTML` typecode.)

```
(get "wikibase-person" "Barack␣Obama" (:ID "BIRTH-DATE
   "))
=> ((:yyyymmdd 19610804))

(get "wikibase-person" "Julius␣Caesar" (:ID "
   BIRTH-DATE"))
=> ((:YYYYMMDD -1000713))
```

iii. `:CALCULATED`
Το Typecode για χαρακτηριστικά υπολογισμένα 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 τη WikipediaBase αντί να προέρχονται από infoboxes. Αυτά τα χαρακτηριστικά θα πρέπει να είναι ειδικά για τις classes `wikibase-term`, `wikibase-person`, και `wikipedia-paragraphs`.

Α΄. `SHORT-ARTICLE`, για την class `wikibase-term`
Η πρώτη παράγραφος του άρθρου. Αν η πρώτη παράγραφος είναι μικρότερη από 350 χαρακτήρες, τότε η επιστρεφόμενη τιμή είναι το πρώτο μέρος του κειμένου έτσι ώστε το άθροισμα των χαρακτήρων είναι τουλάχιστον 350.

Β΄. `URL`, για την class `wikibase-term`
Το URL του άρθρου ως `((:url URL))`

Γ΄. `IMAGE-DATA`, για την class `wikibase-term`
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 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. Εαν δεν υπάρχει caption, η τιμή του caption παραλείπεται π.χ., προτιμότερο `((0 "Harimau\_Harimau\_cover.jpg"))` από `((0 "Harimau\_Harimau\_cover.jpg" ""))`.

Δ΄. `COORDINATES`, για την class `wikibase-term`
Το γεωγραφικό πλάτος και μήκος. Εντοπίζονται είτε στο πάνω δεξιό άκρο του άρθρου, είτε στο infobox. Η τιμή είναι μια λίστα του πλάτους και μήκους, πχ. `((:coordinates latitude longitude))`

Σχήμα 1.1: An example of coordinates in the header

Ε΄. `BIRTH-DATE`, για την class `wikibase-person`
Η ημερομηνία γέννησης. Λαμβάνεται από το infobox, το άρθρο, ή τις πληροφορίες κατηγορίας του άρθρου.
Η τιμή μπορεί να είναι μια a parsed or unparsed date. Οι parsed dates αντιπροσωπεύονται ως αριθμοί, χρησιμοποιώντας τη μορφή YYYYMMDD.

Ϛ΄. `DEATH-DATE`, για την class `wikibase-person`
Η ημερομηνία θανάτου. Λαμβάνεται με παρόμοιο τρόπο όπως το `BIRTH-DATE`. Επιστρέφει τον ίδιο τύπο τιμής όπως BIRTH-DATE, εκτός αν το πρόσωπο ζει, τότε βγάζει διευκρίνηση `(error /"Currently alive"/)`.

Ζ΄. `GENDER`, για την class `wikibase-person`
Το φύλο του προσώπου στο οποίο αναφέρεται το άρθρο. Λαμβάνεται από το περιεχόμενο της σελίδας βασιζόμενο ευρετικές μεθόδους όπως ο αριθμός των ανδρικών ή των θηλυκών αντωνυμιών που χρησιμοποιούνται στο κείμενο.

Η΄. `NUMBER`, για την class `wikibase-term`
Το αν η περιγραφόμενη έννοια αναφέρεται σε ενικό ή πληθυντικό

αριθμό. Λαμβάνεται από το περιεχόμενο του κειμένου με βάση χαρακτηριστικά όπως το πόσες φορές ο τίτλος της σελίδας εμφανίζεται στον πληθυντικό ή στον ενικό αριθμό. Έχει αξία για όλα τα αντικείμενα.
Επιστρέφει `#t` αν είναι πληθυντικός, `#f` αν είναι ενικός.

Θ΄. `PROPER`, για την class `wikibase-term`
Το αν η περιγραφόμενη έννοια είναι κύριο όνομα. Λαμβάνεται από το περιεχόμενο του κειμένου με βάση τα χαρακτηριστικά όπως το πόσες φορές 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.

2. Η εντολή `get-classes`

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

3. Η εντολή `get-attributes`

   Δεδομένου του ονόματος μιας class, επιστρέφει μια λίστα με όλα τα χαρακτηριστικά της class, ως lisp-readable strings. Τα ονόματα των χαρακτηριστικών δίνονται με κεφαλαία γράμματα, αλλά αυτό δεν αποτελεί απόλυτη απαίτηση.

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

4. Η εντολή `sort-symbols`

   Βάζει σε σειρά τα δεδομένα σύμβολα με βάση το μέγεθος του αντίστοιχου άρθρου, ομαδοποιώντας σύμβολα με ίσο μέγεθος άρθρου.

```
(sort-symbols  "Obama␣(surname)" "Barack␣Obama")
=> (("Barack␣Obama") ("Obama␣(surname)"))
```

5. Η εντολή `sort-symbols-named`

Παίρνει ένα σύμβολο α και ένα σύνολο συμβόλων β_i. Βάζει τα β_i σε σειρά έτσι ώστε εάν κάποιο σύμβολο είναι το ίδιο με το α, 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)"))
```

Κεφάλαιο 2

Getting started

Η συνολική WikipediaBase βρίσκεται σε ένα git repository στο infolab's github orginization page{{{ref(infolab$_{github}$}}}.

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

Το `WikipediaBase` εξαρτάται από πολλά άλλα πακέτα python για τη λειτουργία του. Ευτυχώς, η python είναι πακεταρισμένη όχι μόνο με ένα σημαντικό package manager (το pip) 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 δουλεύει το `virtualenv` είναι αντιγράφοντας ένα μέρος από το global python installation και κάνοντας symlink το υπόλοιπο σε ένα τοπικό φάκελο και εγκαθιστώντας τα dependencies στο τοπικό sandbox.

Ένα python virtualenv δημιουργείται και ενεργοποιείται ως εξής:

```
$ 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 global installation

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

Θα χρειαστούμε μερικά επιπλέον εργαλεία για να δουλέψει η WikipediaBase που θα πρέπει να εγκατασταθούν system wide:

- Postresql
- 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 πρώτον, η προσωρινή αποθήκευση συχνά επαναλαμβανόμενων υπολογισμών (caching), και δεύτερον η αποθήκευση ahead-of-time υπολογισμών, όπως το START.

Κεφάλαιο 3

Αρχιτεκτονική

Παρακάτω παρουσιάζονται τα μέρη του συστήματος WikipediaBase και ο τρόπος που αλληλεπιδρούν.

1. Infobox

   Για το σκοπό της παρούσας εργασίας θεωρούμε 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

   Η python 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:

   Κλάσεις επειδή έχουμε δημιουργήσει python αντικείμενα `Infobox` βασισμένοι σε ένα όνομα συμβόλου (π.χ. όνομα της σελίδας) το οποίο στο άρθρο του μπορεί να έχει παραπάνω από ένα wikipedia infoboxes διαφορετικών κλάσεων, ένα python αντικείμενο `Infobox` μπορεί στην πραγματικότητα να είναι μια διεπαφή για πολλαπλά wikipedia infoboxes. Για την ανάκτηση μιας symbol class σε μορφή κατάλληλη για το START, υπάρχει μια διαφορετική μέθοδος.

   Τιμές χαρακτηριστικών δηλαδή είτε v_i^r είτε v_i^m δεδομένου 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 $a_i^r \rightarrow v_i^r$ ή $a_i^m \rightarrow v_i^m$

- Το συνολικό infobox rendered, ή σε ένα markup μορφή.

Τα Infoboxes οργανώνονται σε μια ευρεία ιεραρχία η οποία στον κώδικα του WikiepdiaBase αναφέρεται ως infobox tree. Το infobox tree ανακτάται από σελίδα wikipedia List of infoboxes και χρησιμοποιείται για να συνταχθεί η οντολογία των όρων wikipedia δηλαδή η κατάταξή τους σε κλάσεις.

2. MetaInfobox

   Το `MetaInfobox` υλοποιείται ως μια υποκλάσση του `Infobox` και προσδίδει πληροφορία σχετικά με το infobox, εστιάζοντας στην αντιστοιχία της rendered μορφής των χαρακτηριστικών με την αντίστοιχη markup μορφή. Έτσι δεδομένου ενός infobox τύπου I έχει πιθανά χαρακτηριστικά $a_1, ..., a_n$. 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 της wikipedia.

   Παραδείγματος χάριν στα `officeholder` infoboxes υπάρχει ένα χαρακτηριστικό με markup αναπαράσταση `predecessor` που έχει rendered αναπαράσταση `Preceded by`.

   Για να το πετύχει αυτό το `MetaInfobox` βρίσκει το markup representation όλων των αποδεκτών χαρακτηριστικών μιας κλάσης infobox μέσα από την σελίδα τεκμηρίωσης του αντίστοιχου template. Στη συνέχεια δημιουργεί ένα infobox όπου κάθε χαρακτηριστικό έχει ως τιμή τη markup αναπαράσταση του χαρακτηριστικού αυτού, προσθέτοντας πριν και μετά 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 τη συμβολοσειρά `!!!predecessor!!!` στις rendered τιμές. Θεωρούμε ότι οι τα αντίστοιχα rendered ονόματα αντιστοιχούν στα markup χαρακτηριστικά. Πρέπει να σημειωθεί ότι η αντιστοιχία των rendered χαρακτηριστικών με τα markup χαρακτηριστικά δεν είναι αμφοσήμαντη, δηλαδή κάθε markup χαρακτηριστικό μπορεί να αντιστοιχεί σε μηδέν η 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 πληροφορία σχετική με το άρθρο γενικότερα. Αυτό περιλαμβάνει τις παραγράφους, επικεφαλίδες, τον πηγαίο markup κωδικα και τις κατηγορίες MediaWiki.

4. Fetcher

   Η κλάση `Fetcher` αναλαμβάνει την επικοινωνία της WikipediaBase με τις πηγές πληροφοριών. Είναι ένα μονήρες αντικείμενο που υλοποιεί μια συγκεκριμένη διεπαφή.

   Τα υλοποιημένα `Fetchers` έχουν μια κληρονομική ιεραρχία που φαίνεται από την παρακάτω λίστα.

   `BaseFetcher` είναι η υπερκλάση όλων των fetchers. Θα επιστρέψει αυτούσιο το symbol. Κάνουμε override αυτή τη λειτουργία στις κληρονόμους κλάσεις για να υλοποιήσουμε τη λογική της διεπαφής με τις πηγές πληροφοριών.

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

   `CachingFetcher` κληρονομεί από την class `Fetcher` και διατηρεί τη λειτουργικότητα, μόνο που χρησιμοποιεί μια βάση δεδομένων για την προσωρινή αποθήκευση των πληροφοριών. Είναι η προεπιλεγμένη fetcher class.

   `StaticFetcher` είναι μια κλάση που υλοποιεί το interface της `BaseFetcher` αλλά αντί να φτάσει σε κάποια πηγή πληροφοριών για τα δεδομένα δίνει τιμές επιστροφής στατικά ορισμένες. Χρησιμοποιείται κυρίως από το `MetaInfobox`.

   Από προεπιλογή, το markup προέρχεται από μια βάση δεδομένων. Αν η παράμετρος `force_live` του constructor έχει οριστεί σε `True` τότε το markup θα ληφθεί από το wikipedia.org. Όταν οι δοκιμές τρέχουν στο TravisCI[28], θέλουμε πάντα να χρησιμοποιούνται live δεδομένα. Ελέγχουμε αν ο Travis εκτελεί δοκιμές κοιτάζοντας τη μεταβλητή περιβάλλοντος `WIKIPEDIABASE_FORCE_LIVE`.

5. Renderer

   Οι `Renderers` είναι μονήρεις classes, χρήσιμες για την μετατροπή MediaWiki markup σε HTML. Αρχικά χρησιμοποιήθηκε για την μετατροπή το wikiepedia sandbox[32], επειδή είναι ελαφρώς ταχύτερο από το Wikipedia API. Μεταπηδήσαμε στο wikipedia.org API γιατί το wikipedia-mirror ήταν πολύ αργό και το wikipedia.org θεώρησε κατάχρηση της υπηρεσίας με αποτέλεσμα να μπλοκάρει το IP μας μετά από μερικά τεστ. Γι' αυτό το λόγο χρησιμοποιήθηκε τελικά το API, με Redis caching. Αυτό

λειτούργησε αρκετά καλά, επειδή τα `Renderer` αντικείμενα καταλήγουν να χρησιμοποιούνται μόνο από το `MetaInfobox`, το οποίο έχει ένα αρκετά περιορισμένο πεδίο εφαρμογής, και έτσι τα cache misses είναι σπάνια.

Μια ενδιαφέρουσα πληροφορία για την class `Renderer` ήταν ότι αυτός ήταν ο λόγος που ένα ζευγάρι CSAIL αποκλείστηκε προσωρινά από την επεξεργασία της wikipedia. Ενώ η wikipedia.org έχει μια πολύ επιεική πολιτική όταν πρόκειται για τον αποκλεισμό των χρηστών που έχουν κάνει spamming τους servers, επαναλαμβανόμενες δοκιμές της κατηγορίας `Renderer` με στόχευση το wikipedia sandbox προκάλεσε το IP του δοκιμαστικού μηχανήματος να αποκλεισθεί προσωρινά με το σκεπτικό ότι "η δραστηριότητα του δεν προάγει την βελτίωση της wikipedia". Επανατοποθετήσαμε το `Renderer` να χρησιμοποιεί 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 λειτουργιών για να διαπιστωθεί ποιος είναι ο καλύτερος τρόπος απάντησης.

   (α΄) Frontend

   Η WikipediaBase μπορεί να χρησιμοποιηθεί ως βιβλιοθήκη αλλά η πρωταρχική της λειτουργία είναι ως backend στο START. Η επικοινωνία μεταξύ START και WikipediaBase γίνεται πάνω από μια plaintext telnet σύνδεσή στην πόρτα 8023 χρησιμοποιώντας s-expressions. 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.

   (β΄) Knowledgebase

   Η knowledgebase είναι το σημείο εισαγωγής στην υπόλοιπη wikipediabase.

   Χρησιμοποιεί μοτίβο Provider/Acquirer (βλ. παρακάτω) για 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 ή οποιοδήποτε άλλο μηχανισμό για να δοθεί απάντηση στο ερώτημα που τέθηκε. Οι διαθέσιμοι classifiers και resolvers γίνονται προσβάσιμοι αυτόματα στη knowledgebase χρησιμοποιώντας τη βασική τους κλάση.

   (γ΄) Classifiers

   Κάθε Classifier είναι μονήρης κλάση και υλοποιεί μια ευρετική μέθοδο για να συντάξει μια λίστα από classes ενός `symbol`. Ένα `symbol` μπορεί να επιστρέφει μηδέν ή περισσότερες classes.

   Συνήθως, ένας Classifier θα επιλέξει μόνο αν ένα αντικείμενο πράγματι ανήκει σε μια συγκεκριμένη κατηγορία ή όχι, αλλά αυτό δεν είναι απαραίτητο.

   i. Term

   Ο `TermClassifier` απλά αναθέτει την κατηγορία `wikipedia-term`. Η Wikipediabase διαπραγματεύεται μόνο με πληροφορίες σχετικές με τη wikipedia. Συνεπώς όλες οι έννοιες που συναντώνται ανήκουν σε αυτήν την κατηγορία.

ii. Infobox
To `InfoboxClassifier` αναθέτει σε ένα symbol 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 χαρκτηριστικά με την σειρά που περιγράφονται:

- Category regex matches
- Category regex excludes
- Category matches

Περιγράφονται λεπτομερώς στο παράρτημα.

(δ΄) Resolvers

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

- `resolve(class, symbol, attribute)` που δίνει την τιμή ενός χαρακτηριστικού δεδομένου του `symbol` και της `class`.
- `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 περιεχόμενοτων 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-cout` *Το μέγεθος του άρθρου σε λέξεις.*

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

Κεφάλαιο 4

Το μοντέλο provider/acqirer

Η WikipediaBase προσπαθεί να είναι modular και με δυνατότητα επέκτασης. Για να επιτευχθεί αυτό, συχνά είναι χρήσιμο να συμπλέκει πολλαπλές πηγές του ίδιου τύπου 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 class Provider προσφέρει python decorators για να κάνει αυτή τη 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 στο Frontend ενιαίο τρόπο πρόσβασης στις πηγές.

1. Παράδειγμα

   Εκθέτουμε το μοτίβο provider/acquirer με ένα παράδειγμα ενθέτοντας μια μικρή lisp μέσα στην python, και χειριζόμενοι το state του εκτελούμενου προγράμματος με providers και acquirers.

```
from wikipediabase.provider import Provider, Acquirer,
   provide


class EvalContext(Acquirer):
    def __init__(self, closures):
        super(EvalContext, self).__init__(closures)
        self.closures = closures

    def __call__(self, _ctx, expr):
        if isinstance(expr, list):
            # Handle quotes
```

```
            if expr[0] is 'quote':
                return expr[1]

            # Call the lambda
            fn = self(_ctx, expr[0])
            return fn(self, *[self(_ctx, e) for e in expr
                [1:]])

        if isinstance(expr, basestring) and expr in self.
            resources():
            return self(_ctx, self.resources()[expr])

        return expr


class Lambda(Acquirer):
    def __init__(self, args, expr, env):
        # Get your symbols from all the available closures
            plus an
        # extra for local variables
        super(Lambda, self).__init__([env] + [Symbols()])
        self.args = args
        self.expr = expr

    def __call__(self, _ctx, *args):
        # Add another closure to the list
        arg_provider = Provider();
        for s, v in zip(self.args, args):
            arg_provider.provide(s, v)

        # Build an eval context and run it
        ctx = EvalContext([arg_provider, Provider(self.
            resources())])
        return [ctx(ctx, e) for e in self.expr][-1]

class Symbols(Provider):
    @provide('setq')
    def setq(self, ctx, symbol, val):
        self.provide(symbol, val)

class Builtins(Provider):
    @provide('lambda')
    def _lambda(self, ctx, args, *body):
        return Lambda(args, list(body), Provider(ctx.
            resources()))

    @provide('if')
    def _if(self, ctx, proposition, then, _else):
        if ctx(ctx, proposition):
            return ctx(ctx, then)
        else:
            return ctx(ctx, _else)
```

```
GLOBAL_EVAL = EvalContext([Builtins(), Symbols()])
```

Αυτή η μικρή lisp αν και πρωτόγονη υποστηρίζει:

- lambdas
- A global symbol table
- lexical scoping
- conditionals
- Quoted literals

Προφανώς δεν είναι μια χρήσιμη γλώσσα αλλά μπορεί να πετύχει μερικά ενδιαφέροντα κόλπα:

Μπορούμε να χρησιμοποιήσουμε python types:

```
>>> GLOBAL_EVAL({}, 1)
1
>>> GLOBAL_EVAL({}, True)
True
>>> GLOBAL_EVAL({}, "hello")
'hello'
>>> GLOBAL_EVAL({}, list)
<type 'list'>
```

Μπορούμε να ορίσουμε lambdas και να τις καλέσουμε. Το ακόλουθο παράδειγμα είναι ισοδύναμο με το $(\lambda a.a)1$, το οποίο πρέπει να εκτιμηθεί στην τιμή `1`:

```
>>> GLOBAL_EVAL({}, [["lambda", ['quote', ['a']], 'a'], 1])
1
```

Η μικρή μας lisp δεν είναι pure εφ όσον έχουμε mutable global symbol table. Αυτό σημαίνει πως η σειρά των διεργασιών έχει σημασία. Εφ όσον δεν έχουμε `progn` η άλλα macros συνηθισμένα σε lisp dialects ο καλύτερος τρόπος να κάνουμε διεργασίες σε σειρά είναι να τις εντάξουμε σε ένα lambda και να το εκτιμήσουμε (evaluate).

```
>>> GLOBAL_EVAL({}, [['lambda', ['quote', []], ['setq', 'b',
    2], 'b']])
2
```

Ο προσεκτικός αναγνώστης ίσως παρατηρήσει ότι η λίστα για τα lambda arguments είναι quoted. Ο λόγος γι αυτό είναι ότι δεν θέλουμε η λίστα να εκτιμηθεί.

Συνεχίζοντας την έκθεση του provider/acqirer. Σε κάθε σημείο του κώδικα το κάθε σύμβολο λαμβάνει τιμές από πολλαπλές πηγές. Με σειρά προτεραιότητας:

- The local closure

- The arguments of the lambda
- Builtin functions

Όλα τα προηγούμενα εκτίθενται περιληπτικά χρησιμοποιώντας το provider-aquirer model.

Σε κάθε σημείο ένα διαφορετικό `EvaluationContext` είναι υπεύθυνο για την εκτίμηση και κάθε `EvaluationContext` έχει πρόσβαση στα γνωστά σύμβολα του μέσω μιας array of providers τα οποία εκτίθενται περιληπτικά χρησιμοποιώντας το υπό συζήτηση μοντέλο.

Κεφάλαιο 5

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 της WikipediaBase. Για το unit testing χρησιμοποιούμε την default βιβλιοθήκη python για testing. Κάθε τεστ είναι μια class μου κληρονομεί από την class `TestCase` και υλοποιεί το interface της που περιγράφεται παρακάτω.

Τα Functional tests είναι γραμμένα από πριν, κατά τη διάρκεια ή λίγο μετά τη δημιουργία του συστήματος που τεστάρουν και επιβεβαιώνουν τη σωστή συνολική λειτουργία του συστήματος. Τα Regression tests είναι πολύ παρόμοια με τα to functional tests. Αποδεικνύουν ότι όταν βρεθεί ένα σφάλμα (bug) αυτό διορθώθηκε 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 the 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 τρέχει πριν από κάθε τεστ του `TestCase`. Τα τεστ του testcase αντιπροσωπεύονται από μεθόδους της class των οποίων το όνομα αρχίζει με `test\_`. Στην συγκεκριμένη περίπτωση παίρνουμε την σελίδα της wikipedia για το *Led Zeppelin* και τσεκάρουμε ότι το όνομα *Jimmy Page* αναφέρεται τουλάχιστον μια φορά. Αυτό φανερά δεν αποδεικνύει ότι το fetcher δεν φέρνει για παράδειγμα την σελίδα για το *Yardbirds, Page's first band*. Για αυτό το λόγο γράφουμε παραπάνω από ένα αυτού του είδους τεστ.

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

Εφαρμόσαμε το εργαλείο nosetests 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 συνέχει να τρέξουμε τα τεστ:

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

Η Nose θα βρει όλα τα αρχεία τα οποία είναι στο φάκελο tests/ και έχουν το 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 πρόθεμα `test\_`. Είναι επίσης δυνατό να τρέξει μόνο συγκεκριμένα 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.

Κεφάλαιο 6

Συνώνυμα

Πριν μιλήσουμε για τα συνώνυμα είναι σημαντικό να ορίσουμε πιο αυστηρά τα `symbols` 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 έχει πολλαπλές λέξεις που αναφέρονται στο ίδιο πράγμα, είναι επιτακτική η ανάγκη να καθορισθούν πολλά ονόματα για κάθε σύμβολο. Συνώνυμα είναι τα ονόματα τα οποία οι χρήστες μπορούν να χρησιμοποιήσουν για να αναφερθούν σε ένα συγκεκριμένο σύμβολο.

(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.

Δεν είναι όλα τα συνώνυμα που μπορούμε να δημιουργήσουμε αποδεκτά. Σε γενικές γραμμές συνώνυμα που δεν θα σκεφτόταν ένας άνθρωπος δεν είναι αποδεκτά. Λεπτομέρειες για τις ευρετικές που χρησιμοποιούμε για να αποφασίσουμε αν ένα συνώνυμο είναι αποδεκτό η όχι καθώς και ο τρόπος που παράγουμε συνώνυμα υπάρχουν στο παράρτημα.

Κεφάλαιο 7

Databases and data sources

1. HTML and MediaWiki API

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

   Το Mediawiki παρέχει ένα RESTful API για όλη την απαιτούμενη λειτουργία της wikipedia. Η βασική 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 στέλνουμε GET HTTP αιτήματα και να λαμβάνουμε JSON δεδομένα. Το GET επιλέχθηκε επειδή προτάθηκε στην mediawiki API page γιατί το caching συμβαίνει στο HTTP επίπεδο. Σύμφωνα με τις οδηγίες του HTTP τα POST αιτήματα δεν μπορούν να είναι 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 ζητήσει κάνεις ένα αίτημα για `api.php?titles=Foo|Bar` 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 αιτήμα!

   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 που 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 υλοποιείται στο `persistentkv.py`. Ένα άλλο χαρα-

κτηριστικό που το interface στην βάση δεδομένων πρέπει να υλοποιεί είναι η κωδικοποίηση των αποθηκευμένων αντικειμένων. Επειδή όλη η αποθηκευμένη πηροφορία 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 class για αλληλεπίδραση με την βάση δεδομένων, το `EncodedDict`, εφρμόζει τις μεθόδους `_encode_key` και `_decode_key`.

(α΄) DBM

Διάφορες υλοποιήσεις dbm[6] παρέχονται από την standard βιβλιοθήκη της python. Μερικές διαθέσιμες εφαρμογές DBM 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.

(β΄) SQLite

Η SQLite[25] επίσης χρησιμοποιείται ως caching backend βάση δεδομένων. Δυστυχώς η αποτελεσματικότητά του στο δικό μας σκοπό ήταν απογοητευτική. Χρησιμοποιήσαμε ένα πολύ λεπτό wrapper, το `sqlitedict`[26], για να πάρουμε ένα key-value interface στην SQLite – μια relational βάση δεδομένων. Ο σχετικός WikipediaBase κώδικας είναι πολύ σύντομος:

```
from sqlitedict 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 100000 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 βαση δεδομένων είναι σχεδόν 100 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.

(γ΄) Άλλα backends

Και άλλα backends λαμβάνονται υπόψη, κυρίως το Redis το οποίο εφαρμόσθηκε 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 du jour.

Κεφάλαιο 8

Date parser

Η κατανόηση ημερομηνιών υλοποιήθηκε σε ένα ξεχωριστό πακέτο που ονομάζεται `overlay-parse`[23].

1. Parsing με overlays

   Η έννοια του `overlay` εμπνεύστηκε από τα `emacs overlays`[7]. 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 φυσικών αριθμών 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) που το συγκεκριμένο υποκείμενο εκφράζει.

   Πιο αυστηρά:

   $$o_i \in TextRanget \times Set(Tag) \times A \qquad emacs_overlays$$
   $$Text \to \{o_1, o_2, ..., o_n\}$$

   Για παράδειγμα, από το παρακάτω κείμενο

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

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

   $$\begin{array}{lllll} o_1 = (& r(\text{"}Tuesday\text{"}), & \{\text{DayOfWeek}, \text{FullName}\}, & 2) \\ o_2 = (& r(\text{"}21^{st}\text{"}), & \{\text{DayOfMonth}, \text{Numeric}\}, & 21) \\ o_3 = (& r(\text{"}November\text{"}), & \{\text{Month}, \text{FullName}\}, & 11) \\ o_4 = (& r(\text{"}2016\text{"}), & \{\text{Year}, \text{4digit}\}, & 2016) \end{array}$$

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

Όταν έχουμε ένα σύνολο από overlays μπορούμε να ορίσουμε overlay sequences ως overlays τα οποία είναι κατά συνέχεια, Αυτά και τα δικά τους tag sets ταυτίζονται με ειδικά μοτίβα. Για παράδιγμα 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 $22/07/1991$, οπού $Separator(/)$ ταιριάζει μονό 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` sumbmodule 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␣ama\
   u02c8de\u02d0\u028as␣\u02c8mo\u02d0tsa\u0281t],␣English␣
   see␣fn.;[1]␣27␣January␣1756\xa0\u2013␣5␣December␣1791')
[((27, 1, 1756), (5, 12, 1791))]
```

Κεφάλαιο 9

Παραρτήματα

1. Παραγωγή συνωνύμων

   Για να συμβιβάσουμε αυτούς τους περιορισμούς δύο μέθοδοι χρησιμοποιούνται: qualification και modification των υποψήφιων συνωνύμων. Πρώτα προσπαθούμε τη modification μέθοδο και αν αυτό αποτύχει επιχειρούμε να εκτελέσουμε qualification. 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 και τα δύο versions, με και χωρίς παρενθέσεις. Πχ, δεδομένου του συμβόλου ”Raven (journal)” δημιουργούμε:
     - ”Raven (journal)”
     - ”Raven”
   - Να χρησιμοποιήσουμε τη συμβολοσειρά πριν και μετά το slash, αλλά όχι το αρχικό `symbol`, πχ. δεδομένου του `symbol` ”Russian language/Russian alphabet” δημιουργούμε
     - ”Russian language”
     - ”Russian alphabet”
   - Να αναστρέψουμε των ανεστραμμένων συμβόλων με κόμματα. Πχ δεδομένου ”Congo, Democratic Republic Of The”, αναστρέφουμε για να πάρουμε ”Democratic Republic Of The Congo”
   - Ώς συνήθως, να απορρίψουμε 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 δημιουργήσαμε ένα αρχικό πακέτο συνωνύμων από το ίδιο το όνομα του αντικειμένου. Επιπλέον μπορούμε να δημιουργήσουμε άλλο ένα πακέτο από τα wikipedia redirects στο άρθρο. Η Wikipedia παρέχει ένα SQL dump για όλα τα redirects. Για να φορτώσουμε τον πίνακα του SQL dump σε μια βάση δεδομένων όπου έχουμε φορτώσει ήδη τα δεδομένα της wikipedia:

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

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

```
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 formated 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

Continued on next page

Continued from previous page

page_title	rd_title
Bill_Clinton's_Post_Presidency	Bill_Clinton
Bill_Clinton's_Post-Presidency	Bill_Clinton
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
Billll_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

2. 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 disabiguation pages. Για να το κάνουμε αυτό έτσι ώστε να συμπεριλαμβάνει όλες τις σχετικές σελίδες (συμπεριλαμβανομένων των τυπογραφικών λαθών) εννοούμε σύμβολα που ταιριάζουν με `\([Dd]isambig[\^)]*\)`.

• Συνώνυμα που α) θα μπορούσαν να εκληφθούν ότι ξεκινούν με άρθρα και β) μπορεί να εκλείπουν κάτι χρήσιμο. Αυτό σημαίνει ότι για παράδειγμα ”A. House” (συνώνυμο του «Abraham House”) είναι ελλιπών προδιαγραφών διότι ενδέχεται να παραπλανήσει το 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”.

3. 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
           ))
```

4. 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
```

5. Χαρακτηριστικά για τον person classifier

   (α΄) 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`

   (β΄) Category regex excludes
   Αποκλείει τις ακόλουθες regexes.

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

   (γ΄) 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.

6. Παράδειγμα κατηγοριών άρθρων

   Το άρθρο που αναφέρεται στον Leonardo DiCaprio εντασσεται στις επόμενες κατηγορίες (με bold είναι η κατηγορία που χρησιμοποιεί το WikipediaBase 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

Σχήμα 9.1: The rendered list of categores for Leonardo DiCaprio

Μέρος III

WikipediaMirror

Wikipedia mirror είναι ένα σύστημα με στόχο να αυτοματοποιήσει τη δημιουργία ενός τοπικού κλώνου της wikipedia περιέχοντας μόνο τα άρθρα — δεν περιέχει τους χρήστε , 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 της είναι να παρέχει την δυνατότητα πρόσβασης του συνόλου των δεδομένων της Wikipedia, ανεξάρτητα από το wikipedia.org.

Κεφάλαιο 10

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 μέσω της χρήσης ενός container ή VM τεχνολογίας και μας δίνει τη δυνατότητα να έχουμε άμεση πρόσβαση στο σύστημα αρχείων του stack, ενώ εξακολουθούμε να έχουμε το σύστημα κατασκευής Bitnami να κάνει την κοπιώδη εργασία της ενορχήστρωσης των διαφόρων τμημάτων και επίσης διαχωρίζεται ο διακομιστής από το υπόλοιπο του συστήματος.

Το stack αποτελείται από

- Έναν http server, στην περίπτωση μας τον apache [4]
- Ένα web application runtime, στην περίπτωση μας PHP[24]
- 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[34] παλιότερα ήταν αποδεκτά 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 `mwdumper`[18]. Θα μπορούσε να κάνουμε pipe στο MySQL αλλά η εξαγωγή είναι χρονοβόρα και είναι πιθανό να προκύψουν προβλήματα κατά το dumping.

1. Στοιχεία του stack

   Παρουσιάζεται κάθε στοιχείο του stack με περισσότερες λεπτομέρειες παρακάτω.

   (α΄) 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 ο apache είναι ένας από τους πιο δημοφιλείς διακομιστές web στο διαδίκτυο. Η ίδια η wikipedia.org φαίνεται να χρησιμοποιεί ένα πιο σύνθετο stack που περιλαμβάνει τον `varnish`, έναν HTTP επιταχυντή, και nginx[21], μια εναλλακτική του apache, 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 την υποδομή της Wikipedia. Έχουμε αποκλειστικά επικεντρωθεί στην λειτουργικότητά της. Γι αυτό, λόγω της δημοτικότητας, και της ταχύτητας της αυτόματης εγκατάστασης του Bitnami MediaWiki stack χρησιμοποιήθηκε ως διακομιστή μας ο apache.

(β΄) PHP

Η MediaWiki , η οποία συζητείται παρακάτω, είναι γραμμένη εξ ολοκλήρου σε PHP, μια δημοφιλή server-side γλώσσα προγραμματισμού, με dynamic typing, object-oriented scripting γλώσσα. Η PHP εγκαθίσταται με το Bitnami mediawiki stack. Η PHP είναι δημοφιλής ανάμεσα στους προγραμματιστές

του web και αυτό οφείλεται εν μέρει στην υποστήριξη που έχει από πολλές σχετικές βιβλιοθήκες για βάσεις δεδομένων (συμπεριλαμβανομένων PostgreSQL, MySQL Microsoft SQL Server και SQLite) και είναι ουσιαστικά ένα template δημιουργίας προτύπων γλώσσας HTML.

(γ΄) MySQL

Mediawiki μπορεί να χρησιμοποιήσει πληθώρα SQL database backends:

- **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 δευτερεύοντος σημασίας στο MySQL format (eg. page redirects, categories etc) και προτείνει `mwdumper` που μετατρέπει τα XML dumpls των άρθρων της wikipedia σε MySQL. Το γεγονός αυτό, και το ότι παρέχεται με το αυτοματοποιημένο stack του bitnami, κάνει τη MySQL την προφανή επιλογή για το wikipedia-mirror stack.

(δ΄) Mediawiki

Το Mediawiki είναι η καρδιά της wikipedia. Είναι ένα free και open-source[8] wiki application. Δημιουργήθηκε από το Wikimedia Foundation[29] και τρέχει πολλά δημοφιλή site όπως Wikipedia, Wikitionary[33] και Wikimedia Commons[30].

Το λογισμικό έχει περισσότερες από 800 ρυθμίσεις και περισσότερες από 2.000 επεκτάσεις διαθέσιμες για τη διευκόλυνση προσθήκης ή αλλαγής διάφορων χαρακτηριστικών. Αποκλειστικά στη Wikipedia, πάνω από 1000 αυτοματοποιημένα και 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. Ως επί το πλείστον αυτές τις διαχειρίζεται το Bitnami.

Κεφάλαιο 11

Setting up

Στη συνέχεια είναι βήμα προς βήμα οδηγείες για να στήσει κάνεις το wikipedia mirror. Πρώτα κατεβάζουμε τον κωδικά χρησιμοποιοντας το git[9]:

```
$ 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 σε μορφή dumps για να φορτωθεί σε μια λειτουργική SQL βάση δεδομένων. Φυσικά για να γίνει αυτό πρώτα θα εκτελεσθούν μερικά βήματα.

- Να κατεβάσουμε τα wikipedia database dumps σε XML format.
- Να τα μετατρέψουμε σε format που καταλαβαίνει η MySQL.
- Να στήσουμε το bitnami stack που περιλαμβάνει ένα local install της MySQL
- Να φορτώσουμε τα MySQL dumps στη MySQL.

Όλα αυτά τα βήματα κωδικοποιούνται ως τμήμα μιας ιεραρχίας εξαρτήσεων σε makefile targets και στη θεωρία αυτό πραγματοποιείται αυτόματα και δημιουργεί μια wikipedia mirror. Όμως αυτή λειτουργία είναι μεγάλη και εύθραυστη και συνιστάται κάθε βήμα να γίνεται εξατομικευμένα και χειροκίνητα.

Πρώτα, κατεβάζουμε και εγκαθιστάμε το bitnami. Η ακόλουθη εντολή θα κατεβάσει ένα 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, ένα εργαλείο για software project management για java, είναι εγκαταστημένο. Απαιτείται για να εγκατασταθεί και να στηθεί το mwdumper (βλέπε παρακάτω). Μπορεί να γίνει αυτό αν βεβαιωθούμε ότι το παρακάτω δουλεύει:

```
$ mvn --version
```

Σημείωση: Αν είμαστε σε Ubuntu 14.04, μπορούμε να εγκαταστήσουμε το Maven (για Java) χρησιμοποιώντας `sudo apt-get install maven`.

Τώρα όλα είναι έτοιμα για το αυτόματο download Wikipedia's XML dumps[31] και στη συνέχεια να 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 αποσυμπιεστούν και τελικά θα μετατραπούν σε 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
```

Για να ρυθμιστούν τα mediawiki extensions.

Κεφάλαιο 12

Mediawiki extensions

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

```
MW_EXTENSIONS += newextension
mw-newextension-url = url/to/new/extnesion/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
    registed 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.

Κεφάλαιο 13

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 για να φορτωθούν στη βάση δεδομένων του 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 των επιδόσεων. Αντ αυτού προτείνονται εργαλεία όπως το `mwdumper` που μετατρέπουν τα XML dumps σε MySQL ερωτήματα τα οποία γεμίζουν τη βάση δεδομένων.

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

- XML
- MySQL dump
- SQLite dump
- CSV

Για τους σκοπούς μας έχει ενδιαφέρον μονό ο μετασχηματισμός από XML σε MySQL, ωστόσο συναντήθηκαν σημαντικές δυσκολίες σε αυτή τη διαδικασία. Λεπτομέρειες για το ποιές ήταν και πως αντιμετοπίστηκαν δείτε την περιγραφή του xerces bug στο παράρτημα.

Κεφάλαιο 14

Εργαλεία

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

1. utf8thread.c

   Το `utf8thread.c` είναι ένα 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 σερβίρει μια σελίδα που δείχνει live δεδομένα σε μορφή ιστογράμματος για την πρόοδο του γεμίσματος της βάσης δεδομένων. Το `webmonitor.py` σερβίρει στατικές html σελίδες και μετά τους στέλνει δεδομένα μέσω websocket. Το Webmonitor μπορεί να δείχνει οποιοδήποτε stream από ζευγάρια `<epoc date> <float value>` που λαμβάνει στο input. Για παράδειγμα:

   ```
   $ pip install tornado
   ```

   Πρώτα πρέπει να εγκαταστήσουμε τα dependencies του script. Το οποίο μπορεί να είναι tornado, anasynchronous web framework που υποστηρίζει websockets. Δίνουμε οδηγίες στο tornado, που θα σερβίρει την ακόλουθη σελίδα:

   ```
   <!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>
```

Η σελίδα αναμένεται να διαβάζει ένα stream τιμών από ένα websocket στο `ws://localhost:8888/` και τα κάνει plot σε πραγματικό χρόνο χρησιμοποιώντας το highcharts.js. Ο προσεκτικός αναγνώστης ίσως παρατηρήσει ότι τα παραπάνω δεν είναι ακριβώς HTML αλλά περισσότερο ένα python formatted string. Αυτό συμβαίνει για 2 λόγους. Πρώτον γιατί το python script διαχειρίζεται το configuration και δεύτερον γιατί το πλάτος του graph υπολογίζεται σε page load time και το plot χρειάζεται να μετατοπισθεί για να δείξει μόνο τα πιο πρόσφατα 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 θα τα κανονικοποιήσει χρησιμοποιώντας `awk` και τα θα τα τροφοδοτήσει στο

webmonitor. Αφού αυτή η εντολή εκτελεσθεί μπορούμε να ανοίξουμε τον browser και να κάνουμε navigate στο localhost:8888.

Χρησιμοποιούμε αυτό το script για να ελέγξουμε από απόσταση το ολικό μέγεθος των δεδομένων που το mysql λαμβάνει, μιας που αυτή είναι η πιο χρονοβόρα διαδικασία της δημιουργίας του mirror.

3. xml-parse.sh

   Άπλα αφαιρώντας συγκεκριμένα άρθρα διορθώνουμε το πρόβλημα του `xerces`. Αν τα άρθρα είναι απομονωμένα τότε επίσης το 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 κανείς 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 ”κουτσουρευει” εννοούμε ότι αφήνει τα table scheamata ανεπηρέαστα και διαγράφει όλα τα internal δεδομένα.

5. page\remover.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 αργή γράψαμε ένα χαμηλού επιπέδου πρόγραμμα στη C για να το επιλύσουμε, το `page_remover.c`. Το `page-remover` δέχεται ως input το 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 withespace characters. Το `page_remover.c` δεν είναι threaded μιας που το bottleneck είναι στο HDD IO speed και ο παραλληλισμός δεν θα βοηθούσε.

Κεφάλαιο 15

Αυτοματισμός

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

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

Επιπλέον περιλαμβάνεται μια σειρά από shell εντολές για την μεταφορά από την πρώτη στη δεύτερη κατάσταση. Το αποτέλεσμα είναι οτι ο χρήστης απαιτεί τη δημιουργία ενός αρχείου και το `make` τρέχει όσο το δυνατόν λιγοτερες διεργασίες, αποφεύγοντας να ξαναδημιουργήσει αρχεία που χρειάζεται αλλα ήδη υπάρχουν. Λεπτομέρειες για τη λειτουργία των makefiles θα βρείτε στο παράρτημα.

Χρησιμοποιούμε τα makefiles για να μπορούμε να συνεχίζουμε μια διαδικασία που σταματήσαμε ή απέτυχε οσο πιο κοντα γίνεται στο σημείο οπου εμεινε την τελευταία φορα. Επίσης η λειτουργία των makefiles κανει το make αρκετά έξυπνο ώστε να μην επαναλαμβάνει βήματα που πιθανώς κάναμε χειροκίνητα.

Κεφάλαιο 16

Επιδόσεις

Το Compile time περιλαμβάνει το χρόνο που χρειάζεται για:

- Κατέβασμα όλων των στοιχείων του wikipedia server
- Το στήσιμο του 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 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 τέλη 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 της μεθόδους βελτιστοποίησης που επιχειρήθηκαν παρουσιάζονται παρακάτω.

- Ρύθμιση του `innodb_buffer_pool_size`[11]. 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={{{ref(fsync)}}}.` □□ □□□□□□□ □□ π□□□□□□□ □□ □□□ `=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`[13]. Εν τέλη 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 εμπιστευόμενη ότι οι τιμές των κελιών που αναφέρονταν σε άλλους πίνακες όντως έδειχναν κάπου.

Κεφάλαιο 17

Παραρτήματα

1. Το bug στη βιβλιοθήκη xerces

   Πιθανότατα η μεγαλύτερη δυσκολία κατά τη δημιουργία του wikipedia-mirror ήταν η αντιμετώπιση ενός bug στο `mwdumper` — το εργαλείο για τη μετατροπή των XML dumps σε SQL dumps — το οποίο κάνει το εργαλείο να αποτυγχάνει σε τυχαία άρθρα. Εφ όσον δεν μπορέσαμε να βρούμε τον κριβή λόγο που συμβαίνει αυτό το bug, το παρακάμπτουμε διαγράφοντας τα άρθρα που προκαλούν το πρόβλημα, και εφ όσον είναι ένα μεγάλο εμπόδιο σε μια κατά τα άλλα θεωρητικά πεπατημένη διαδικασία περιγράφουμε τη διαδικασία μας λεπτομερώς.

   Ιδού λοιπόν τι ακριβώς συμβαίνει: ενώ τρέχει το `make sql-dump-parts` εγείρεται το παρακάτω exception:

```
...

376,000 pages (14,460.426/sec), 376,000 revs (14,460.426/sec)
377,000 pages (14,458.848/sec), 377,000 revs (14,458.848/sec)
Exception in thread "main" java.lang.
   ArrayIndexOutOfBoundsException: 2048
        at org.apache.xerces.impl.io.UTF8Reader.read(Unknown
           Source)
        at org.apache.xerces.impl.XMLEntityScanner.load(
           Unknown Source)
        at org.apache.xerces.impl.XMLEntityScanner.
           scanContent(Unknown Source)
        at org.apache.xerces.impl.
           XMLDocumentFragmentScannerImpl.scanContent(Unknown
            Source)
        at org.apache.xerces.impl.
           XMLDocumentFragmentScannerImpl$FragmentContentDispatcher
           .dispatch(Unknown Source)
        at org.apache.xerces.impl.
           XMLDocumentFragmentScannerImpl.scanDocument(
           Unknown Source)
        at org.apache.xerces.parsers.XML11Configuration.parse
           (Unknown Source)
```

```
        at org.apache.xerces.parsers.XML11Configuration.parse
           (Unknown Source)
        at org.apache.xerces.parsers.XMLParser.parse(Unknown
           Source)
        at org.apache.xerces.parsers.AbstractSAXParser.parse(
           Unknown Source)
        at org.apache.xerces.jaxp.SAXParserImpl$JAXPSAXParser
           .parse(Unknown Source)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java
           :392)
        at javax.xml.parsers.SAXParser.parse(SAXParser.java
           :195)
        at org.mediawiki.importer.XmlDumpReader.readDump(
           XmlDumpReader.java:88)
        at org.mediawiki.dumper.Dumper.main(Dumper.java:142)
make: *** [/scratch/cperivol/wikipedia-mirror/drafts/
   wikipedia-parts/enwiki-20131202-pages-articles20.xml-
   p011125004p013324998.sql] Error 1
```

Διερευνούμε το πρόβλημα τρέχοντας `make --just-print sql-dump-parts` με σκοπό να βρούμε επακριβώς την αλληλουχία εντολών που προκάλεσαν το πρόβλημα και ανακαλύπτουμε πως η εντολή που αποτυγχάνει είναι:

```
$ java -jar /scratch/cperivol/wikipedia-mirror/tools/mwdumper
   .jar   --format=sql:1.5 /scratch/cperivol/wikipedia-mirror
   /drafts/wikipedia-parts/enwiki-20131202-pages-articles20.
   xml-p011125004p013324998.fix.xml > /root/path/wikipedia-
   parts//enwiki-20131202-pages-articles20.xml-
   p011125004p013324998.sql
```

Ευτυχώς αυτή η εντολή δεν τρέχει για μεγάλο διάστημα έτσι μπορούμε εύκολα να πειραματιστούμε. Εδώ είναι το `time` output:

```
26.65s user 1.73s system 78% cpu 35.949 total
```

Το λάθος φαίνεται να συμβαίνει κατά την διάρκεια του διαβάσματος του XML dump συνεπώς δεν είναι ειδικό για το SQL output. Αυτό θα μπορούσε να είναι χρήσιμο για να διαπιστώσουμε ποιο άρθρο προκαλεί το λάθος, αποσύροντάς το και ελπίζοντας να λυθεί το πρόβλημα. Για να το εντοπίσουμε κατ αρχάς προσπαθήσαμε να κάνουμε export σε XML:

```
$ java -jar /scratch/cperivol/wikipedia-mirror/tools/mwdumper
   .jar   --format=xml /scratch/cperivol/wikipedia-mirror/
   drafts/wikipedia-parts/enwiki-20131202-pages-articles20.
   xml-p011125004p013324998.fix.xml > /tmp/just-a-copy.xml
```

Όπως ήταν αναμενόμενο, το ίδιο λάθος εμφανίστηκε. Στη συνέχεια κοιτάμε τα τελευταία δύο άρθρα που έγιναν export. Θέλουμε να μπορούμε να αυτοματοποιήσουμε τη διαδικασία συνεπώς το κάνουμε χρησιμοποιόντας shell commands

και όχι με το χέρι: Ξεκινάμε τυπώνοντας με αντίστροφη σειρά το xml αρχείο που δημιουργήθηκε, βρίσκοντας τις τελευταίες δύο εμφανίσεις του <title> χρησιμοποιώντας το εργαλείο grep και αναστρέφοντας ξανά τις προκύπτουσες γραμμές για να τυπώσουμε με την αρχική σειρά. Σημειώνουμε ότι το εργαλείο tac είναι μέρος των coreutils[**?**] άλλα όχι του BSD toybox[**?**]. Κατά συνέπεια θα υπάρχει σε όλες τις διανομές GNU linux προεγκατεστημένο άλλα όχι και σε mac os x και BSD. Δουλεύει όπως cat, μόνο που εμφανίζει τις γραμμές με αντίστροφη σειρά):

```
$ tac /tmp/just-a-copy.xml | grep "<title>" -m 2 | tac
      <title>The roaring 20s</title>
      <title>Cranopsis bocourti</title> # <- This is the last
          one
```

Αυτή η λειτουργία τελειώνει γρήγορα παρά το ότι το /tmp/just-a-copy.xml είναι αρκετά μεγάλο διότι το εργαλείο tac ψάχνει πρώτα το τέλος του αρχείου και διαβάζει προς τα πίσω μέχρι το grep να βρει τα 2 περιστατικά που ψάχνει για <title> και κλείνει. Στο filesystem ext3, και τα παρόμοια filesystems, κατά την διαδικασία αυτή δεν διασχίζεται ολόκληρο το αρχείο. Πράγματι, από τον κώδικα tac βλέπουμε ότι γίνεται χρήση της lseek που αναζητεί το τέλος του αρχείου χωρίς να το προσπελάσει ολόκληρο και στη συνέχεια διαβάζει αντίστροφα:

```
if (lseek (input_fd, file_pos, SEEK_SET) < 0)
    error (0, errno, _("%s: seek failed"), quotef (file));
/* Shift the pending record data right to make room for the
   new.
   The source and destination regions probably overlap.  */
memmove (G_buffer + read_size, G_buffer, saved_record_size);
past_end = G_buffer + read_size + saved_record_size;
/* For non-regexp searches, avoid unnecessary scanning. */
if (sentinel_length)
    match_start = G_buffer + read_size;
else
    match_start = past_end;

if (safe_read (input_fd, G_buffer, read_size) != read_size)
{
    error (0, errno, _("%s: read error"), quotef (file));
    return false;
}
```

Ας σώσουμε την διαδρομή από το αρχικό xml αρχείο σε μια μεταβλητή γιατί θα το χρησιμοποιήσουμε χρησιμοποιούμε πολύ. Έτσι από δω και πέρα το $ORIGINAL_XML θα έχει διαδρομή από το αρχικό xml.

```
$ export ORIGINAL_XML=/scratch/cperivol/wikipedia-mirror/
   drafts/wikipedia-parts/enwiki-20131202-pages-articles20.
   xml-p011125004p013324998.fix.xml
```

Πρώτα ας δούμε αν κάτι περίεργο συμβαίνει με το xml αρχείο

```
$ grep "<title>Cranopsis␣bocourti</title>" -A 200 -B 100
   $ORIGINAL_XML | less
```

Τίποτα περίεργο δεν βρέθηκε, έτσι δεν μπορούμε πραγματικά να διορθώσουμε το πρόβλημα επιτόπου. Θα προσπαθήσουμε να αποσύρουμε ολόκληρο το άρθρο και ελπίζουμε ότι θα δουλέψει (spoiler alert: δουλεύει). Δεν μπρούμε όμως να κάνουμε κανονικό parsing του xml format μιας που το μέσο μέγεθος του αρχείου που αντιμετωπίζουμε είναι της τάξης των δεκάδες GB. Θα ήταν καλύτερα να βρούμε έναν πιο γρήγορο και πιο low level τρόπο να ανασύρουμε το αρχείο. Θα χρησιμοποιήσουμε λοιπόν καθαρά byte και string operations.

Θα προσπαθήσουμε αρχικά να επιθεωρήσουμε τους parents του τίτλου από το προβληματικό άρθρο. Ευτυχώς το xml που δημιουργήθηκε είναι indented, έτσι μπορούμε να βρούμε τους parents που βασιζόμενοι σε αυτό. Αριθμήσαμε 6 spaces από indentation στη γραμμή που ο mwdumper απέτυχε, έτσι θα ψάξουμε προς τα πίσω από εκεί για τις γραμμές στις οποίες το indentation μειώνεται. Οι γραμμές αυτές θα αντιπροσωπεύουν τους προγόνους του άρθρου μέσα στην ιεραρχία του XML:

```
$ for i in {0..6}; do \
    echo "Level␣$i:"; \
    tac /tmp/just-a-copy.xml | grep "^␣\{$i\}<[^/]" -m 1 -n |
        tac; \
done

Level 0:
17564960:<mediawiki xmlns="http://www.mediawiki.org/xml/
   export-0.3/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xsi:schemaLocation="http://www.mediawiki.org/xml
   /export-0.3/␣http://www.mediawiki.org/xml/export-0.3.xsd"
   version="0.3" xml:lang="en">
Level 1:
Level 2:
38:  <page>
Level 3:
Level 4:
35:    <revision>
Level 5:
Level 6:
26:      <text xml:space="preserve">&lt;!-- This article was
   auto-generated by [[User:Polbot]]. --&gt;
```

Φαίνεται ότι η δομή του συγκεκριμένου xml έχει ως εξής: κάθε `page` βρίσκεται σε σε ένα domain που ονομάζεται `mediawiki`. Θα μπορούσαμε να δούμε αυτό επίσης και από την java source αλλά αν και πιο ακριβό από άποψη υπολογιστικών πόρων είναι πιο γρήγορο από το να προσπαθήσουμε να διαβάσουμε τη java του mwdumper.

Ο πιο εύκολος τρόπος να αφαιρέσουμε αυτό το άρθρο θα ήταν το εργαλείο `awk`. Αλλά είναι πολύ αργό για τους σκοπούς μας και θέλουμε να βελτιστοποιήσουμε και

να αυτοματοποιήσουμε την όλη διαδικασία. Πρώτα ας προσπαθήσουμε απλώς να συγκρίνουμε το orginal xml και το xml που δημιουργήσαμε byte προς byte μιας που αυτό είναι πολύ γρήγορη διεργασία, με την ελπίδα πως το άρθρο υπό συζήτηση θα είναι η μόνη διαφορά και έτσι θα μπορούμε πανεύκολα να βρούμε το σημείο οπού θα αρχίσουμε διαγράφουμε:

```
$ cmp /tmp/just-a-copy.xml $ORIGINAL_XML
/tmp/just-a-copy.xml /scratch/cperivol/wikipedia-mirror/
   drafts/wikipedia-parts/enwiki-20131202-pages-articles20.
   xml-p011125004p013324998.fix.xml differ: byte 2, line 1
```

Η πρώτη διαφορά είναι στο 2ο byte, συνεπώς σίγουρα δεν πρόκειται περί του άρθρου υπό συζήτηση.

```
$ head $ORIGINAL_XML
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.8/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
   schemaLocation="http://www.mediawiki.org/xml/export-0.8/␣
   http://www.mediawiki.org/xml/export-0.8.xsd" version="0.8"
    xml:lang="en">
  <siteinfo>
    <sitename>Wikipedia</sitename>
    <base>http://en.wikipedia.org/wiki/Main_Page</base>
    <generator>MediaWiki 1.23wmf4</generator>
    <case>first-letter</case>
    <namespaces>
      <namespace key="-2" case="first-letter">Media</
         namespace>
      <namespace key="-1" case="first-letter">Special</
         namespace>
      <namespace key="0" case="first-letter" />

$ head /tmp/just-a-copy.xml
<?xml version="1.0" encoding="utf-8" ?>
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.3/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
   schemaLocation="http://www.mediawiki.org/xml/export-0.3/␣
   http://www.mediawiki.org/xml/export-0.3.xsd" version="0.3"
    xml:lang="en">
  <siteinfo>
    <sitename>Wikipedia</sitename>
    <base>http://en.wikipedia.org/wiki/Main_Page</base>
    <generator>MediaWiki 1.23wmf4</generator>
    <case>first-letter</case>
    <namespaces>
      <namespace key="-2">Media</namespace>
```

Τα γνωρίσματα των xml tags είναι αρκετά διαφορετικά. Ελπίζουμε τουλάχιστον τα line numbers να συμπίπτουν ώστε αν δε μπορούμε να πάμε κατ ευθείαν στο byte που μας ενδιαφέρει για να αρχίσουμε να διαγράφουμε, τουλάχιστον να μπορούμε γρήγορα να μετρήσουμε τα newlines για να βρούμε το άρθρο. Μετρήσαμε

τον αριθμό των γραμμών στο `/tmp/just-a-copy.xml` και ελπίζουμε ότι η αντίστοιχη γραμμή στο `$ORIGINAL_XML` θα αναφέρεται στο προβληματικό άρθρο. Εάν αυτό συμβεί μπορούμε να αγνοήσουμε τις περιβάλλουσες xml πληροφορίες και να σβήσουμε το προβληματικό άρθρο με βάση αυτήν την πληροφορία. Θα χρησιμοποιήσουμε `wc` το οποίο είναι αρκετά γρήγορο.

```
$ wc -l /tmp/just-a-copy.xml
17564961 /tmp/just-a-copy.xml
```

Και η αντίστοιχη γραμμή στο `$ORIGINAL_XML` είναι:

```
$ sed "17564960q;d" $ORIGINAL_XML
[[Willie Jones (American football)|Willie Jones]],
```

Ποδόσφαιρο (football)… καμιά σχέση με τα βατράχια (frogs). Φαίνεται ότι δεν μπορούμε να αποφύγουμε κάποιο επίπεδο του parsing.

(α΄) Parsing

Θα κάνουμε τις ακόλουθες θεωρήσεις για να αποφύγουμε το σε βάθος parsing του εγγράφου:

- Το XML στο αρχικό αρχείο είναι valid.
- Κάθε XML μέσα στο άρθρο είναι HTML escaped

Κατ αρχάς δουλεύοντας με γραμμές είναι αργή διαδικασία γιατί το user space code χρειάζεται να ψάχνει newlines. Δουλεύοντας με bytes αναθέτουμε εργασία στο kernel, επιταχύνοντας την εργασία σημαντικά. Έτσι το `dd` είναι το σωστό εργαλείο για την συγκεκριμένη δουλειά. Αλλά πρώτα θα βρούμε σε ποιο byte είναι το άρθρο που μας ενδιαφέρει

```
$ grep -b "<title>Cranopsis␣bocourti</title>" -m 1
    $ORIGINAL_XML
1197420547:    <title>Cranopsis bocourti</title>
```

Αυτό ίσως πάρει κάποιο χρόνο αλλά δυστυχώς είναι η μόνη μας επιλογή. Η στρατηγική μας είναι να φτιάξουμε 2 αρχεία: το `/tmp/original_tail.xml` το οποίο να περιέχει όλα τα δεδομένα που υπάρχουν **μετά** τη σελίδα που θέλουμε να βγάλουμε και το `/tmp/original_head.xml` το οποίο περιέχει όλα τα δεδομένα **πριν** τη σελίδα που θέλουμε να βγάλουμε.

Τώρα θα χρησιμοποιήσουμε `sed` να ψάξει για `</page>` μετά το byte 1197420547 το οποίο θα είναι το σημείο x. Βάζουμε το μέρος του `$ORIGINAL_XML` μετά στο σημείο x μέσα στο αρχείο `/tmp/original_tail.xml`:

```
$ dd if=$ORIGINAL_XML skip=1197420547 ibs=1 | sed '0,/<\/
    page>/d' > /tmp/original_tail.xml
```

Θαυμάσια, αυτό δούλεψε! Το `dd` δεν αντιγράφει αντίστροφα έτσι θα χρειαστεί να κάνουμε κάτι πιο περίπλοκο για να κατασκευάσουμε `/tmp/original_head.xml`.

Ας υποθέσουμε ότι η θέση που βρήκαμε τον τίτλο της σελίδας που θέλουμε να αφαιρέσουμε είναι $\alpha = 1197420547$ και το σημείο που η σελίδα αρχίζει είναι στο σημείο β. Είναι ασφαλές να υποθέσουμε ότι $\beta > \alpha - 1000$ (μπορούσαμε να αλλάξουμε τη σταθερά 1000 εάν αυτή η υπόθεση ήταν λάθος, αλλά τελικά ήταν σωστή). Με αυτό τον τρόπο χρειάζεται μόνο να ψάξουμε στο 1Kb για τη συμβολοσειρά <page>.

Αυτό θα ήταν ισοδύναμο με το εξής: αντί να κάνουμε copy τα bytes στο εύρος $[0, \beta)$, να συνδέσουμε δυο διαστήματα $[0, \alpha - 1000] \cup (\alpha - 1000, \beta)$ δημιουργώντας ένα subshell το οποίο θα έχει πρώτο output το πρώτο εύρος και στη συνέχεια output $(\alpha - 1000, \alpha)$ σταματόντας όταν βρεί <page>, όπως φαίνεται στον παρακάτω κώδικα. Το αποτέλεσμα της διεργασίας αυτής είναι το αρχείο /tmp/original_head.xml:

```
$ (dd count=$((1197420547-1000)) ibs=1 if=$ORIGINAL_XML;
    \
   dd if=$ORIGINAL_XML count=1000 skip=$
      ((1197420547-1000)) ibs=1 \
       | tac | sed '/<page>/,$d' | tac) > /tmp/
          original_head.xml
```

τελικά ενώνουμε το /tmp/original_head.xml με το /tmp/original_tail.xml στο αρχείο που δουλεύει κανονικά μ το mwdumper.

(β΄) Η τελική λύση

Όλα τα παραπάνω χρησιμοποιήθηκαν για να συντεθεί ένα script που υπάρχει στο data/xml-parse.sh το οποίο χρησιμοποιήθηκε από το makefiles για να απομακρύνει όλα τα προβληματικά άρθρα. Εάν το mwdumper αποτύχει, ταυτοποιούμε το άρθρο που προκάλεσε το πρόβλημα και το απομακρύνουμε χρησιμοποιώντας xml-parse.sh. Στη συνέχεια ξανατρέχουμε το mwdumper. Το επαναλαμβάνουμε αυτό μέχρι το mwdumper να πετύχει. Συνολικά τα προβληματικά άρθρα είναι περίπου 10-15, και είναι διαφορετικά ανάλογα με το dump που χρησιμοποιείται.

(γ΄) Καλύπτοντας με κενά

Από την παραπάνω έκθεση των τρόπων που αντιμετωπίσουμε το θέμα του άρθρου που σπάει παραλείψαμε κάτι προφανές. Μια θεματικά διαφορετική προσέγγιση: το να καλύψουμε το άρθρο που προκαλεί το πρόβλημα με κενά. Μόλις εντοπίσουμε το εύρος στο οποίο η σελίδα βρίσκεται μπορούμε να κάνουμε mmap επακριβώς στο τμήμα του $ORIGINAL_XML και στη συνέχεια να κάνουμε memset καλύποντας το με χαρακτήρες κενών. Η το πρόγραμμα ζει στο data/page_remover.c, Παρακάτω παρουσιάζουμε την κλήση στο mmap:

```
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;
}
```

και το memset

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

Περιέργως αυτό δεν έλυσε το πρόβλημα του mwdumper το οποίο δείχνει ότι μάλλον πρόκειται για memory leak από τη μεριά του `xerces` αλλά αυτό ξεπερνά τους στόχους της παρούσας εργασίας.

(δ΄) Η εντολή sed

Στο κεφάλαιο για το xerces bug αναφέραμε την χρήση της εντολής sed και ίσως να είναι χρήσιμο να το αναπτύξουμε περεταίρω. `sed` είναι ένα unix εργαλείο πού βρίσκεται στο πακέτο GNU coreutils το οποίο σύμφωνα με το man page είναι ένας stream editor που φιλτράρει και μεταμορφώνει κείμενο. Η βασική λειτουργία είναι ότι το ”pattern space”, ή το input stream το οποίο είναι ένα unix stream — που έρχεται από το αρχείο, ένα pipe ή απλά stdin — περνά μέσα από ένα προγραμματίσιμο pipeline. Εκτυπώνεται είτε αυτούσιο το modified pattern space είτε — με τη χρήση του `-n` flag — επιλεγμένα τμήματα αυτού. Ας δούμε τη χρήση που κάναμε παραπάνω για το sed.

Αρχικά χρησιμοποιήσαμε sed για να εκτυπώσουμε μια μεμονωμένη line σε ένα αρχείο:

```
$ sed "17564960q;d"
```

Αυτό το `sed` πρόγραμμα διαχωρίζει τις εντολές με semicolon (`;`). Το `sed` απαριθμεί τις γραμμές του input stream και τρέχει καθένα από τις εντολές διαχωρισμένες με ”;” σε σειρά μέχρι μια να επιτύχει.

Οι εντολές εδώ είναι `17564960q` και `d`. Η `17564960q` θα σταματήσει το sed όταν φτάσει η παρούσα γραμμή είναι η γραμμή νούμερο 17564960. Η `d` θα απορρίπτει την παρούσα γραμμή κάθε φορά. Έτσι το sed απορρίπτει γραμμές μέχρι να συναντήσει τη γραμμή 17564960 την οποία εκτυπώνει και τελειώνει.

Χρησιμοποιούμε μια εντολή sed ως μέρος μιας σειράς εντολών shell piped όλες μαζί με στόχο να εκτυπωθούν όλες ο γραμμές ενός stream μετά από ένα συγκεκριμένο μοτίβο (στην περίπτωσή μας `</page>`).

```
$ sed '0,/<\/page>/d'
```

αυτή τη φορά είχαμε μόνο μια εντολή sed, `d`. Το `sed` απαριθμεί στις γραμμές στο stream, απορρίπτοντας γραμμές στο εύρος των γραμμών από 0 μέχρι τη γραμμή που ταυτίζεται με το `<\/page>`, ουσιαστικά τυπώνοντας μόνο γραμμές μετά το `</page>`.

Η τελική μας χρήση του `sed` είναι η αντίστροφη της προηγούμενης,

```
$ sed '/<page>/,$d'
```

Εδώ το sed απαριθμεί ξανά σε όλες τις lines toustream. Αυτή τη φορά απορρίπτοντας γραμμές ανάμεσα στην πρώτη και αυτήν που ταιριάζει το `<page>` μέχρι την τελική γραμμή, σημειωμένη με `$`.

2. Λεπτομερώς τα Makefiles

   Ας αρχίσουμε με ένα 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 dependencytou `bar.o`, έτσι χρειάζεται να ξανχτιστεί.

   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.

   (α΄) 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.

(β΄) Variables

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

i. Αναδρομικές μεταβλητές

```
OBJCETS = foo.o bar.o

show:
    echo $(OBJECTS)
```

Τρέχοντας `make show` θα εμφανίσει `foo.o bar.o` στην κονσόλα. Όλες οι μεταβλητές αντικαθιστώνται με τις τιμές τους αν βάλει κάνεις πρενθέσεις γύρω από το όνομά τους και προθέσει ένα δολάριο (`$`). Οι μεταβλητέσ των makefiles δεν έχουν τύπους, αναφορά σε μια μεταβλητή είναι ισοδύναμη με string substitution, όπως είναι και στο shell scripting.
Οι μεταβλητές που ορίζονται με ένα απλό `\=` είναι recursively expanded. Αυτό σημαίνει ότι αφού το όνομα της μεταβλητής αντικαθίσταται από την τιμή της μια αναδρομική διαδικασία συνεχίζει να κάνει expand τις τιμές που προκύπτουν με την ίδια τη μεταβλητή ακόμα στο local scope.

```
library = foo

foo-libs = -lfoo
foo-includes = -I./include/foo

bar-libs = -lbar
bar-includes = -I./include/bar

libs = $($(library)-libs)
includes = $($(library)-includes)

waz:
   gcc waz.c $(includes) $(libs)
```

τρέχοντας `make`

```
gcc waz.c $(includes) $(libs)
gcc waz.c $($(library)-includes) $($(library)-libs)
gcc waz.c $(foo-includes) $(foo-libs)
gcc waz.c -I./include/foo -lfoo
```

Παρατηρήστε πως οι αναφορές στις μεταβλητές καθαυτές δημιουργήθηκαν.
Μεταβλητές μπορούν επίσης να ορισθούν στην εντολή `make`

```
$ make --just-print library=bar
gcc waz.c -I./include/bar -lbar
```

ii. 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 πορσπαθουμε 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[27], με άλλα λογία η εύρεση της τιμής μεταβλητών μπορεί να είναι αναδρομική

άλλα 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
```

iii. Automatic variables
To 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 toutarget 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.

iv. 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)
```

3. Πηγαίοι κωδικές

(α΄) page$_{\text{remover.c}}$

```
/*
 * 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, '␣', ctx->size);

    context_destroy(ctx);
    return 0;
}
```

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


#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␣%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%016llx,␣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",
            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;
}
```

(γ´) 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
```

(δ΄) webmonitor.py

```
"""
Just feed pairs of

<epoc 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 aswell 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␣>␣%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>
"""

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 closs(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()
```

(ε΄) 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]" 1>&2
    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="<title>$1</title>"
    title_offset=$(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 $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
```

Μέρος IV

Συμπεράσματα και μελλοντικές επεκτάσεις

Η παρούσα εργασία είχε ως αποτέλεσμα τη δημιουργία του WikipediaBase που χρησιμοποιείται σήμερα αποτελεσματικά από το START. Το πρόγραμμα αυτό πέτυχε τους στόχους του δηλαδή είναι επεκτάσιμο, modular, γρήγορο και μπορεί να χρησιμοποιηθεί αποτελεσματικά από νέους φοιτητές. Η επεκτασιμότητα διασφαλίζεται κυρίως από το provider / acqurirer pattern που ευήβραμε κατά την παρούσα εργασία, καθώς και με τη βελτίωση της ποιότητας του κωδικά και της αρχιτεκτονικής γενικότερα, ειδικά εκμεταλλευόμενοι την κληρονομική ιεραρχία. Το modularity εξασφαλίζεται μέσω της θέσπισης interfaces μεταξύ των modules που είναι ανεξάρτητα από τη λειτουργία τους. Η χρήση της γλώσσας python, η εκτενής τεκμηρίωση, η ιδιοματική χρήση των τεχνολογιών, το ενδελεχές testing και η χρήση συνεργατικών εργαλείων όπως το github, έπαιξαν καταλυτικό ρολό στη διευκούνση της μελλοντικής εισαγωγής νέων φοιτητών στην εργασία.

Μελλοντικά το WikipediaBase θα πρέπει βετιωθεί ώστε να μπορεί να αντηλεί περισσότερες πληροφορίες από το κύριο σώμα του κειμένου, να κάνει cache πιο αποτέλεσματικα τις σελίδες που λαμβάνει από το διαδίκτυο, να μπορεί να απαντήσει ερωτήσεις συνδυάζοντας πληροφορίες από περισσότερα άρθρα και να βελτιώσει την κατανόηση των infoboxes.

Το wikipedia-mirror αντίθετα, αν και πέτυχε το στόχο του να δημιουργεί πιστά αντίγραφα της wikipedia, δεν ήταν αρκετά αποτελεσματικό ώστε να είναι χρήσιμο στην πράξη λόγω της ταχύτητας που ήταν μια με δυο τάξεις μεγέθους μικρότερη από αυτήν του wikipedia.org. Η βελτίωση αυτού του προβλήματος είναι δύσκολη από μια μικρή ομάδα της οποίας ο βασικός σκοπός είναι άλλος — στην παρούσα περίπτωση η έρευνα στην επεξεργασία φυσικής γλωσσάς. Αντ αυτού είναι προτιμότερη η χρήση των dumps κατ ευθείαν όπως είναι κατεβασμένα από τη wikipedia, παρ όλο που η μορφή της πληροφορίας δεν είναι ίδια, είτε η χρήση κατ ευθείαν της wikipedia.org.

Μέρος V

Βιβλιογραφία

[1]

[2] Boris Katz: Annotating the World Wide Web using Natural Language (RIAO '97)

[3] Boris Katz, Sue Felshin, Deniz Yuret, Ali Ibrahim, Jimmy Lin, Gregory Marton, Alton Jerome McFarland, and Baris Temelkuran: Omnibase: Uniform Access to Heterogeneous Data for Question Answering (NLDB 2002).

[4] Apache Software Foundation https://en.wikipedia.org/wiki/Apache_Software_Foundation

[5] Bitnami https://en.wikipedia.org/wiki/Bitnami

[6] DBM https://en.wikipedia.org/wiki/Dbm

[7] Emacs Overlays https://www.emacswiki.org/emacs/EmacsOverlays

[8] Free and open-source software https://en.wikipedia.org/wiki/Free_and_open-source_software

[9] Git (software) https://en.wikipedia.org/wiki/Git_(software)

[10] Wikipedia:List of infoboxes https://en.wikipedia.org/wiki/Wikipedia:List_of_infoboxes

[11] The InnoDB Buffer Pool https://dev.mysql.com/doc/refman/5.5/en/innodb-buffer-pool.html

[12] innodb\flush\method https://dev.mysql.com/doc/refman/5.5/en/innodb-parameters.html#sysvar_innodb_flush_method

[13] innodb\io\capacity https://dev.mysql.com/doc/refman/5.5/en/innodb-parameters.html#sysvar_innodb_io_capacity

[14] Make (software) https://en.wikipedia.org/wiki/Make_(software)

[15] Apache Maven https://en.wikipedia.org/wiki/Apache_Maven

[16] MediaWiki https://en.wikipedia.org/wiki/MediaWiki

[17] Microsoft SQL Server https://en.wikipedia.org/wiki/Microsoft_SQL_Server

[18] MWDumper https://www.mediawiki.org/wiki/Manual:MWDumper

[19] MySQL https://en.wikipedia.org/wiki/MySQL

[20] MySQLi https://en.wikipedia.org/wiki/MySQLi

[21] Nginx https://en.wikipedia.org/wiki/Nginx

[22] Oracle https://en.wikipedia.org/wiki/Oracle

[23] overlay-parse https://pypi.python.org/pypi/overlay-parse/1.1

[24] PHP https://en.wikipedia.org/wiki/PHP

[25] SQLite https://en.wikipedia.org/wiki/SQLite

[26] sqlitedict https://pypi.python.org/pypi/sqlitedict

[27] Total functional programming https://en.wikipedia.org/wiki/Total_functional_programming

[28] Travis CI https://en.wikipedia.org/wiki/Travis_CI

[29] Wikimedia movement https://en.wikipedia.org/wiki/Wikimedia_movement

[30] Wikimedia Commons https://pam.wikipedia.org/wiki/Wikimedia_Commons

[31] Wikipedia dumps https://dumps.wikimedia.org/enwiki/latest/

[32] Wikipedia Sandbox https://en.wikipedia.org/wiki/Wikipedia:Sandbox

[33] Wiktionary https://en.wikipedia.org/wiki/Wiktionary

[34] XAMPP https://en.wikipedia.org/wiki/XAMPP

[35] fsync http://man7.org/linux/man-pages/man2/fsync.2.html