**Báo Cáo Nghiên Cứu Tốt Nghiệp**

1. **Crawl dữ liệu. Khó khăn và giải quyết.**
   1. **Định nghĩa.**

Crawl tổng quát là quá trình lấy thông tin từ các nguồn thông tin là các trang web khác nhau về hệ thống cục bộ một cách tự động. Chương trình crawler sẽ tự động load về nhiều trang web khác nhau theo một quy luật xác định trước.

Hệ thống crawl của nhóm dữ liệu hiện nay chủ yếu thu thập thông tin địa điểm, tập trung vào các lĩnh vực như ẩm thực, khách sạn, ngân hàng …

Sau khi có được dữ liệu ban đầu từ trang web gọi là dữ liệu thô, ta tiếp tục tiến hành trích rút lấy dữ liệu quan trọng như tên địa điểm, số điện thoại, địa chỉ, nhận xét … một cách có cấu trúc. Các dữ liệu này sẽ được lưu trữ trong CSDL hoặc file xml làm đầu vào cho các bước xử lý tiếp theo.

* 1. **Crawl với ngôn ngữ Ruby.**

**“**Ruby được tạo ra bởi [Yukihiro "Matz" Matsumoto](http://vi.wikipedia.org/w/index.php?title=Yukihiro_Matsumoto&action=edit&redlink=1) từ [24 tháng 2](http://vi.wikipedia.org/wiki/24_tháng_2), [1993](http://vi.wikipedia.org/wiki/1993) và đưa ra bản chính thức vào năm [1995](http://vi.wikipedia.org/wiki/1995).” - Wikipedia.

Ruby là ngôn ngữ lập trình kịch bản tương tự như PHP với một số khác biệt về cú pháp như

* Câu lệnh giống với lời thoại (Tiếng Anh).
* Không có dấu chẩm phẩy
* Hướng đối tượng chặt chẽ. Mỗi bít dữ liệu đều là một đối tượng.

Ruby có cộng đồng người dùng lớn, có khả năng mở rộng thông qua các Gems có thể tim thấy dễ dàng trên các kênh phân phối.

Một số Gems của Ruby hỗ trợ mạnh mẽ cho crawl dữ liệu như load trang web, điều khiển trình duyệt, duyệt Html, các gems này sẽ được mô tả cụ thể ở phần sau.

Bên cạnh đó, câu lệnh đơn giản của ruby cũng khiến việc lập trình trở nên đơn giản hơn. Tuy nhiên khi mới bắt đầu sẽ gặp một số kho khăn do Ruby khác biệt rất lớn với các ngôn ngữ khác như C hay Java.

* + 1. **Giới thiệu Ruby Gems.**

Gems là module cho phép Ruby mở rộng thêm các chức năng của mình. Khi cài đặt, ruby đã có sẵn một số gems cơ bản, tuy nhiên người dùng có thể tự mình phát triển những gems riêng phục vụ nhu cầu cụ thể. Trong thực tế, cộng đồng người dùng đã phát triển được một hệ thống gems rất đồ sộ. Các gems này có thể viết bằng ruby hoặc bằng ngôn ngữ khác như C hay Java. Do đó nó đảm bảo được về mặt hiệu năng.

Trong quá trình xây dựng hệ thống Crawl, nhóm đã sử dụng nhiều gems khác nhau, nhưng ở đây em xin trình bày về ba gems chính, đóng vai trò quan trọng.

* **Nokogiri.**

Đây là bộ gems cung cấp tính năng duyệt file XML, HTML, SAX một cách nhanh chóng, cung cấp các phương thức tìm kiếm phần tử qua Xpath, CSS3 selector. Nokogiri giúp việc thao tác với các phần tử html trong trang một cách đơn giản và thuận tiện hơn.

**Khai báo :** *require 'nokogiri'.*

**Sử dụng**:

*doc = Nokogiri::HTML(string).*

*s*tring là xâu Html cần đọc. Sau đó nokogiri sẽ tạo ra một document object cho phép thao tác dược với dữ liệu.

*VD:*

*doc.css("a[class='hot\_name']")*

Dòng lệnh này sử dụng truy vấn CSS selector, trả về tất cả thẻ <a> với class là *hot\_name.*

* **Rest-open-uri.**

Giúp mở trang web với giao thức http một cách dễ dàng.

**Khai báo:**

*require 'rest-open-uri'*

**Sử dụng:**

*link = URI.parse(URI.encode(link))*

*open(link)*

Trong đó *link* ban đầu chứa đường dẫn đến trang web cần mở. Lệnh *open* sẽ trả về toàn bộ má html của trang web.

* **Selenium-webdriver.**

Selenium là công cụ dùng để điều khiển trình duyệt hoạt động theo các tác vụ định trước. Selenium hỗ trợ nhiều ngông ngữ khác nhau như C, Java, Python và trong đó có Ruby.

Ban đầu, selenium ra đời với mục đích cho phép lập trình viên có thể kiểm tra một cách tự động xem trang web của mình có hoạt động như mong đợi hay không bằng cách tự động mở trang web bằng trình duyệt. Tuy nhiên trong quá trình crawl dữ liệu với các trang web có nhiều thánh phần thay đổi phức tạp, khó nắm bắt Selenium hoàn toàn có thể sử dụng để lấy được các dữ liệu này.

Do phải mở trình duyệt nên crawl bằng selenium sẽ chậm hơn so với sử dụng Rest-open-uri.

**Khai báo:**

*require 'selenium-webdriver'*

***Sử dụng:***

*wait = Selenium::WebDriver::Wait.new(:timeout => 15)*

*profile = Selenium::WebDriver::Firefox::Profile.from\_name "default"*

*driver = Selenium::WebDriver.for(:firefox, :profile => profile)*

Từ đây một trình duyệt sẽ được mở lên và ta có thể điều khiển thông qua đối tượng driver.

Đi đến một link.

*driver.navigate.to* [*http://google.com*](http://google.com/)

Tìm một phần tử bằng Xpath:

*element = driver.find\_element(:xpath, "//div[@class='footer small']")*

* 1. **Khó khăn trong crawl dữ liệu và giải quyết.**

Lý thuyết crawler khá đơn giản tuy nhiên khi cài đặt vào thực tế lại nảy sinh không ít khó khăn. Đơn cử như khi truy vấn đến một trang web với tần suất quá lớn sẽ làm trang web đó hoạt động khó khăn, do đó cần có cơ chế quản lý truy vấn. Giữa hai lần truy vấn phải cách nhau một thời gian nhất định. Bên cạch đó việc lặp đi lặp lại một thao tác nhiều lần với một máy chủ web cũng khiến máy chủ web từ chối truy cập vào hệ thống.

Trên thực tế khi cài đặt, xây dựng web crawler cũng gặp rất nhiều khó khăn, sau đây sẽ trình bày về một số vấn đề và cách giải quyết.

* + 1. **Crawl với các trang web nội dung tĩnh.**

Đây là các trang web có nội dung tương đối ổn định, cấu trúc đơn giản rõ ràng. Không chứa nhiều các thành phần Ajax.

Đối với những trang web loại này, ta có thể đơn giản sử dụng Nokogiri để load trang web về, sau đó sử dụng Xpath hoặc Css3 selector để trích rút những thông tin cần thiết.

Khi crawl cần chú ý đến tệp tin robots.txt đặt trong thư mục root của trang web. Tệp tin này chứa thông tin dành cho các crawler, qui định khu vực nào của trang web được phép truy cập. Nếu ta truy cập không theo quy định này, có thể xảy ra trường hợp máy chủ web chặn Nokogiri truy cập tiếp. Khi đó quá trình crawl sẽ phải dừng lại.

Ngoài ra như đã đề cập ở trên. Phải có một cơ chế giới hạn truy cập đến máy chủ web. Tránh tình trạng máy chú web này bị truy cập dồn dập.

Một số trang web loại này như amthuc365, dendau ...

* + 1. **Crawl với các trang web có cung cấp web services.**

Đây là các trang web mà ta đã nắm được các services, sử dụng services với các tham số truyền vào thích hợp giúp ta lấy được thông tin một cách nhanh chóng mà không cần phải load toàn bộ trang web.

Services trả về có thể có cấu trúc XML, HTML, hay string tùy theo từng trang web. Tùy từng services cụ thể mà ta có thể sử dụng các phương pháp khác nhau như load trang web bằng Rest-open-uri hay Nokogiri.

Một số trang web loại này như foody.vn ...

* + 1. **Crawl với các trang web phức tạp, sử dụng dàn trang bằng Javascript.**

Đây là nhóm những trang web khó crawl nhất do cấu trúc trang web phức tạp, phần lớn do sử dụng nhiều xử lý phía clients như sử dụng javascript, Ajax. Nếu ta đơn thuần load những trang web này bằng nokogiri hay Rest-open-uri thì sẽ không thể thu được nội dung cần thiết, do những nội dung này được trình bày bằng cách sử dụng script phía clients để trình bày. Ruby không có cơ chế giúp chạy các script này trên servers.

Đối với những trang web này ta phải sử dụng Selenium để mở các trang này bằng trình duyệt, cho phép clients side script chạy trên trình duyệt sau đó mới trích rút được thông tin.

Phương pháp này là phương pháp phức tạp và có thời gian crawl lâu nhất. Tuy nhiên có thể dùng để crawl bất kì trang web nào, kể cả các trang web có hạn chế truy cập bởi Nokogiri.

Một số trang web loại này như agoda.vn, chudu24 ...

1. **Xử lý dữ liệu Ontology.**
   1. **Xử lý địa chỉ.**
   2. **Xử lý số điện thoại.**
   3. **Xử lý type và Category.**
2. **Nghiên cứu tiếp theo.**