Решение домашнего задания
от компании Яндекс
Визуализация веб-портала

Толмачев Александр

Формулировка задания

Поиск по большому объему текстовой информации часто сложнее, чем по массиву частично визуализированных данных. Для облегчения восприятия интернет-портала мы предлагаем вам решить задачу визуализации структуры портала.

Ваша программа должна уметь рисовать карту сайта по заданному URL. В качестве тестового URL предлагаем вам рассмотреть портал проекта Яндекс.Авто (http://auto.yandex.ru/).

Обоснование выбора задания

Это задание показалось мне с одной стороны достаточно интересным, с другой – не самым сложным и не самым простым из предложенных.

Общие предположения

В контексте данного задания под картой сайта я понимаю древовидную структуру, которая отражает логическую структуру интернет-поратала. Ее внешнее представление может быть различным – это могут быть различного рода изображения с нарисованным деревом карты сайта или просто вложенные списки, представляющие иерархию древовидной структуры и так далее. Для того, чтобы разделить логическое и внешнее представление карты сайта, я решил представлять результат в виде HTML файла, содержащего информацию о древовидной структуре карты. Это также имеет следующие преимущества:

- Позволяет отображать карту сайта различными способами, используя CSS и javascript.
- HTML документ всегда можно отредактировать, это позволяет при необходимости корректировать полученную карту.
- В таком виде карту сайта удобно отобразить в качестве одной из страниц портала.
- Если требуется нарисовать дерево карты сайта в прямом смысле, то удобнее написать программу, которая анализирует структуру HTML документа, который уже содержит информацию о карте, чем выкачивать страницы сайта и анализировать их (в этом случае полученное изображение нельзя будет изменить и для создания нового изображения придется заново выкачивать страницы сайта, что гораздо дольше, чем анализ HTML документа).

То есть выбранное представление результата работы программы является выгодным с точки зрения универсальности и дальнейшего использования карты сайта. Я постарался продемонстрировать это, в качестве примера написав небольшой CSS-файл и javascript-файл для отображения карты в виде интерактивного дерева, подобного тому, что отображается в файловых системах операционных систем.

Также я решил предоставить пользователю возможность задать ограничение на глубину дерева карты сайта, для того, чтобы он смог получить карту с необходимым ему уровнем детализации. Это также позволит ограничивать время создания карты сайта – если сайт очень большой, то это время может быть слишком большим.

Основные идеи решения

Анализ логической структуры сайта

Для построения карты сайта требуется проанализировать логическую структуру сайта.

Портал *может* содержать XML-файлы или текстовые файлы sitemap, которые содержат ссылки, на те страницы портала, которые должны быть проиндексированы поисковыми роботами. Но, в отличие от файла robots.txt для файлов sitemap нет четких правил по поводу названия и расположения, к тому же портал может и не иметь подобных фалов. Кроме того, файлы sitemap фактически содержат просто наборы ссылок, и извлечь информацию о логической структуре сайта из этих файлов очень сложно.

Поэтому для анализа структуры сайта требуется написать веб-робота (в другой терминологии – веб-паука или кроулера), который “перемещается” сайту и анализирует его логическую структуру. “Перемещение” робота по сайту (кроулинг) должно происходить стандартным способом: робот получает начальную ссылку (URL главной страницы сайта), загружает ее, анализирует ее содержимое, извлекает ссылки, которые ведут на другие страницы сайта, загружает их (если необходимо), анализирует и так далее. В процессе перемещения по сайту робот должен добавлять новые элементы в дерево карты сайта.

Кроулинг веб-сайта

При кроулинге веб-сайта важно учитывать политику вежливости (politeness policy), которая предполагает отправку запросов на сервер портала с некоторыми задержками, чтобы не перегружать сервер запросами, а также учет инструкций файла robots.txt. Кроме того, многие сервера, в частности, сервер портала Яндекс.Авто, имеют защиту от большого числа частых запросов и ограничивают в этом случае доступ к порталу.

В процессе разработки приложения я как раз столкнулся с упомянутой выше проблемой – ограничением доступа после закачивания определенного числа страниц. Для портала Яндекс.Авто это число было около 220. Этого количества достаточно, чтобы построить карту с уровнем глубины, равным 1 (на главной странице сайта присутствуют ссылки на страницы 145 марок). Но для уровня детализации, равного 2, этого уже недостаточно. Причем выставление достаточно больших задержек между запросами не помогало. Сначала я подумал, что решить эту проблему мне поможет какой-нибудь специализированный фреймворк для кроулинга веб-сайтов и изучил фреймворк Scrapy (тоже для языка Python). Но после этого я понял, что проще и удобнее будет написать веб-робота самостоятельно, чем приспосабливать этот серьезный фреймворк под нужды своего проекта, и лишь подсмотрел некоторые параметры кроулинга, которые использует Scrapy, в частности задание случайного интервала между задержками в некоторых пределах. Это помогло решить проблему. Оказалось достаточным задавать задержку в среднем равную 15 секундам и использовать для величины задержки случайное число из диапазона [0.5 * 15, 1.5 * 15].

Анализ содержимого страниц сайта

При анализе содержимого страницы веб-сайта и извлечении новых ссылок важно извлекать только те ссылки, которые действительно характеризуют логические разделы сайта, на которые можно попасть с анализируемой страницы, и группировать эти ссылки по смыслу. К примеру, ссылка в параграфе текста (внутри HTML тегов `<p></p>`) вида “об этом можно прочитать здесь” под данное определение не попадает.

Так как в задании предложено рассмотреть портал проекта Яндекс.Авто (http://auto.yandex.ru/), то и предположения, связанные с извлечением семантически нагруженных групп ссылок я делал, исходя из структуры страниц данного портала.

Проанализировав HTML код страниц сайта Яндекс.Авто, я заметил, что ссылки, которые представляют интерес с точки зрения логической структуры сайта, сгруппированы в списки (обрамлены тегами `<ul></ul>` или `<ol></ol>`). Однако, это характерно не только для данного отдельно взятого портала: многие сайты оформляют меню сайта в HTML-коде как список и выделяют остальные блоки ссылок, которые соответствуют логическим разделам сайта, в списки тоже. Поэтому сделанные предположения верны не только для сайта Яндекс.Авто.

Используемые технологии

Код программы написан на языке Python версии 2.7. Для получения интернет-ресурсов по URL, логирования, и разбора файла robots.txt используются соответствующие встроенные библиотеки языка Python.

Также используются сторонние open-sourse библиотеки:

Для разбора (парсинга) HTML страниц используется сторонняя библиотека `lxml` (http://lxml.de/). Выбор этой библиотеки объясняется ее быстродействием посравнению с другими DOM-парсерами (например, по сравнению с BeautifulSoup) и поддержкой Xpath.

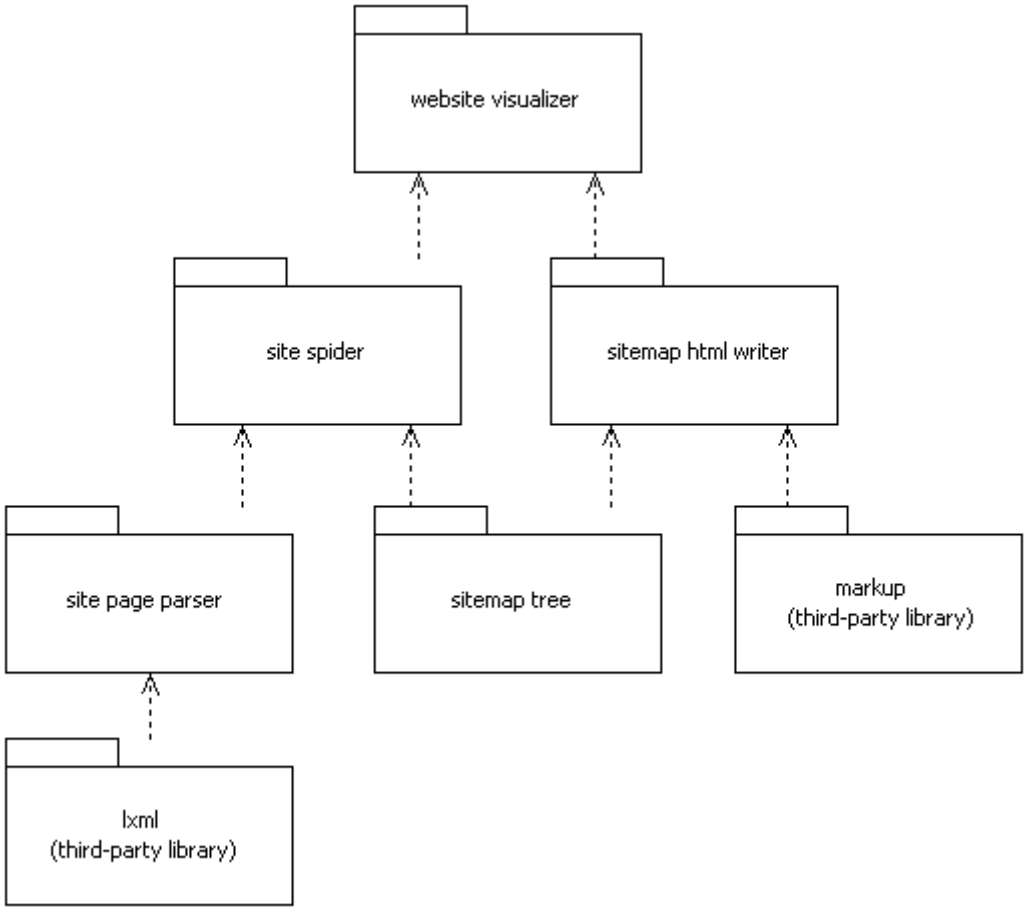
Для создания HTML-файла с картой сайта используется сторонняя легковесная библиотека `markup.py` (http://markup.sourceforge.net/). Она предоставляет простой и удобный интерфейс для генерации HTML-кода. Ее выбор обусловлен тем, что для решения задач моего приложения нет необходимости привлекать серьезные многофункциональные библиотеки для создания HTML.

Архитектура приложения

Приложение состоит из пяти модулей:

- sitemap_tree.py – содержит классы элементов дерева карты сайта.
- site_page_parser.py – содержит класс парсера страниц веб-сайта.
- site_spider.py – содержит класс веб-робота.
- sitemap_html_writer.py – содержит класс генератора HTML-файла с результатом работы программы.
- website_visualizer.py – точка входа приложения.

Иерархию модульной структуры приложения можно изобразить при помощи UML-диаграммы зависимостей пакетов:

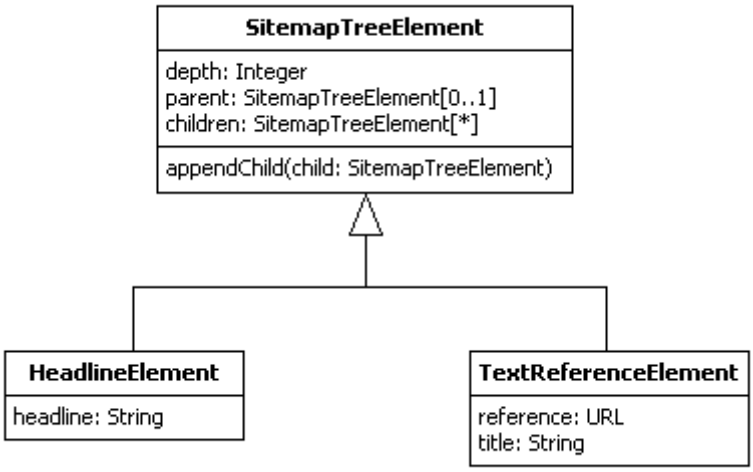


Теперь остановимся подробнее на каждом из модулей, чтобы детальнее описать архитектуру приложения.

sitemap tree

В этом модуле содержится определение классов для элементов дерева карты сайта.

UML-диаграмма этих классов:



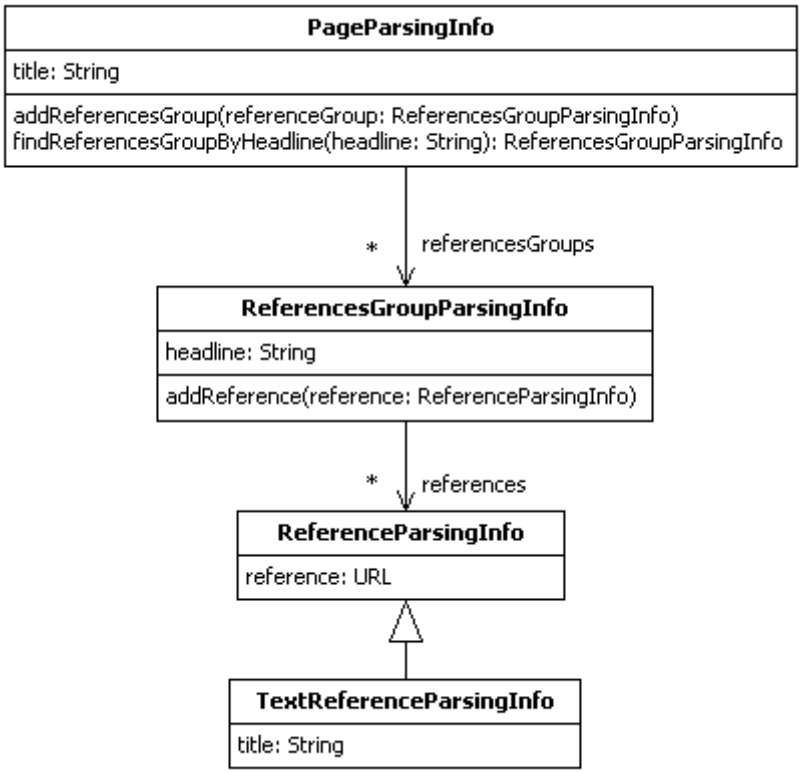
На данном этапе разработки приложения в дереве карты могут содержаться два типа элементов: текстовые ссылки (ссылки с текстовым заголовком), которые соответствуют страницам (разделам) веб-сайта; ссылки могут группироваться по смыслу и группа ссылок может иметь заголовок, поэтому элементом карты сайта также может быть заголовок этой группы. В дальнейшем можно расширять приложение и добавлять новые виды элементов в карту сайта – к примеру, на странице могут встретиться ссылки, которые являются картинками. Сейчас же мы будем оперировать только текстовыми ссылками. Если ссылка не текстовая, то будем считать ее заголовком заголовок страницы, на которую она ведет.

Любой элемент дерева карты сайта имеет атрибут `depth`, значение которого соответствует уровню (глубине) этого элемента в иерархии структуры сайта. Будем считать глубину страницы веб-сайта равной числу кликов, которое нужно сделать, чтобы попасть на эту страницу с главной страницы сайта. Глубину ссылки, ведущей на эту страницу, и глубину заголовков групп новых ссылок, расположенных на этой странице, положим равными глубине этой страницы. Таким образом, корень дерева карты сайта имеет глубину, равную 0. Также любой элемент содержит ссылку на отца и список ссылок на дочерние элементы дерева.

site page parser

Этот модуль содержит определение класса парсера страниц веб-сайта, определение вспомогательных классов для хранения информации о странице в целом, группах ссылок и самих ссылках, которая была получена в процессе разбора страницы, а также класс исключения, генерируемого парсером при ошибке разбора. Использует стороннюю библиотеку `lxml` для разбора HTML кода.

UML-диаграмма вспомогательных классов для хранения информации, извлеченной в процессе разбоа:

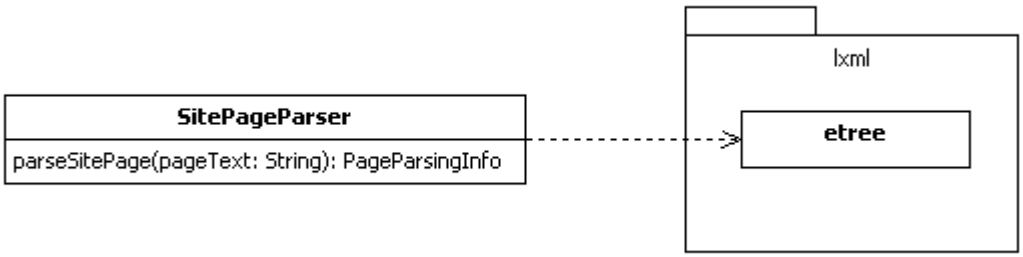


Класс `ReferenceParsingInfo` – базовый класс для хранения информации об извлеченных ссылках. Содержит единственный атрибут, который соответствует URL'у данной ссылки. Текстовая ссылка также должна содержать заголовок ссылки.

Ссылки на страницы могут объединяться в группы, которые могут иметь или не иметь заголовок. Класс `ReferencesGroupParsingInfo` позволяет сохранить заголовок группы ссылок, если он есть, и информацию о самих ссылках группы.

Класс `PageParsingInfo` позволяет сохранить информацию о странице, полученную в процессе разбора, которая нас интересует, а именно заголовок страницы и информацию о группах ссылок.

Класс `SitePageParser` имеет единственный предназначенный для внешнего использования метод, который осуществляет разбор страницы и возвращает объект класса `PageParsingInfo`, содержащий информацию о разобранной странице.



Для разбора страницы используется класс etree библиотеки lxml.

Алгоритм извлечения групп ссылок со страницы работает следующим образом, с учетом предположений о структуре страницы, описанных выше:

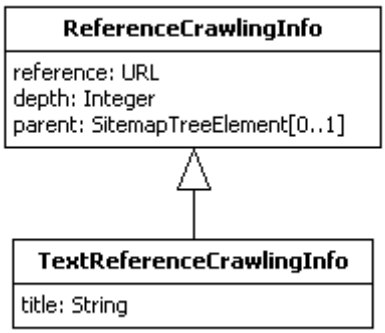
При помощи Xapth я нахожу все списки страницы, заключенные в теги `<ul></ul>` или `<ol></ol>`. Если найденный список является группой ссылок, то я пробую найти заголовок этой группы (его может и не быть), поднимаясь вверх по документу следующим образом: просматривая сначала sibling-элементы, предшествующие найденному списку, затем отца этих элементов и предшествующие sibling-элементы этого отца и так далее. Далее я рассматриваю следующий список и так далее. Важно, что соседние списки ссылок могут

образовывать одну смысловую группу (например, на страницах портала Яндекс.Авто так и есть), поэтому это нужно учитывать. В случае, если у группы нет заголовка, то это не страшно. В случае, если заголовок есть, то он будет одинаков для таких списков и соответствующие группы ссылок можно объединить.

site spider

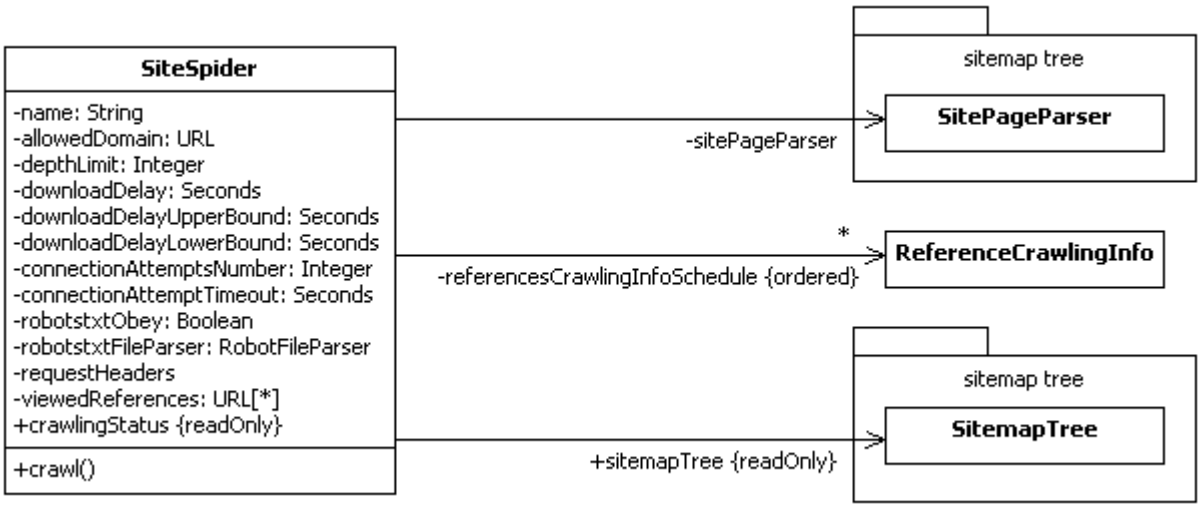
Содержит определение класса веб-робота, а также определения вспомогательных классов для хранения информации о ссылках сайта, связанной с кроулингом, и определение классов исключений, генерируемых роботом при ошибках.

UML-диаграмма вспомогательных классов:



В процессе кроулинга нам необходимо знать следующую информацию о ссылке: URL ссылки, ее глубину для задания глубины соответствующего элемента карты сайта (и определения, нужно ли рассматривать данную ссылку, если пользователем задано ограничение на глубину дерева), а также необходимо держать ссылку на тот элемент, который уже добавлен в карту сайта и который должен быть отцом нового элемента, соответствующего данной ссылке. Для текстовой ссылки также необходимо знать ее заголовок. В дальнейшем можно расширять приложение, поддерживая не только текстовые ссылки.

UML-диаграмма самого класса веб-робота:



Класс `SiteSpider` содержит ряд атрибутов, не предназначенных для внешнего использования, которые задают параметры кроулинга:

`name` – имя робота, которое используется для формирования User-Agent заголовка запросов, посылаемых на сервер.

`allowedDomain` – URL, который соответствует адресу главной страницы портала и ограничивает перемещения робота по Интернету.

`depthLimit` – ограничение на глубину кроулинга. Если ограничение не задано, то этот атрибут равен 0.

`downloadDelay` – средняя величина задержки в секундах между посылаемыми запросами на сервер. Как описано выше, задается равным 15 секундам.

`downloadDelayUpperBound` – верхняя граница диапазона задержек. Как описано выше, задается равной `0.5 * downloadDelay` (заимствовано из Scrapy).

`downloadDelayLowerBound` – нижняя граница диапазона задержек. Как описано выше, задается равной `1.5 * downloadDelay` (заимствовано из Scrapy).

`connectionAttemptsNumber` – число попыток установить соединение с сервером.

`connectionAttemptTimeout` – величина задержки в секундах между попытками установить соединение.

`robotstxtObey` – параметр, определяющий, следует ли веб-роботу учитывать инструкции файла robots.txt. Хоть я и оставил в приложении возможность не учитывать инструкции этого файла, но эта возможность не регулируется пользователем, и файл robots.txt учитывается всегда (если он существует), чтобы следовать политике вежливости.

`robotstxtFileParser` – экземпляр класса `RobotFileParser`, предоставляемого встроенной библиотекой `robotparser` языка Python.

`requestHeaders` – заголовки запросов, отправляемых серверу.

`viewedReferences` – множество ссылок, уже добавленных в карту сайта.

`sitePageParser` – экземпляр класса `SitePageParser`, описанного выше.

`referencesCrawlingSchedule` – стек для хранения информации о тех ссылках, которые были извлечены из страниц в процессе кроулинга, но еще не добавлены на карту сайта.

Также присутствуют атрибуты, предназначенные только для чтения:

`crawlingStatus` – текущий статус процесса кроулинга. Содержит информацию о том, успешно ли проходит кроулинг или произошла ошибка.

`sitemapTree` – дерево карты сайта (ссылка на его корень), которое создается роботом в процессе кроулинга.

Метод `crawl()` выполняет сам процесс кроулинга. Помимо этого метода класс имеет много вспомогательных методов, не предназначенных для внешнего использования. Я не стал включать их в диаграмму класса и не буду их описывать, их описание можно прочитать в комментариях к исходному коду программы.

Процесс кроулинга и построения дерева карты сайта происходит следующим образом:

В начале робот посылает пробный запрос по заданному URL, соответствующему главной странице сайта. Если происходят какие-либо ошибки, то процесс кроулинга прерывается и карта сайта не может быть построена. Далее робот пытается загрузить файл robots.txt и если он найден, то разбирает его. В противном случае кроулинг будет продолжаться без учета инструкций этого файла (параметр `robotstxtObey` устанавливается равным `False`). Далее информация о ссылке на главную страницу сайта помещается в стек `referencesCrawlingSchedule` и дальнейший процесс кроулинга происходит по следующему алгоритму:

Информация об очередной ссылке извлекается из стека. Если задано ограничение на глубину дерева карты сайта (глубину кроулинга), и глубина извлеченной ссылки превосходит это ограничение, то робот переходит к следующей итерации (извлекает из стека информацию о следующей ссылке). Так как я поддерживаю на данный момент только текстовые ссылки, то ссылка может не иметь заголовка (к примеру, если извлеченная ссылка была картинкой). В этом случае в качестве заголовка ссылки я использую заголовок страницы, на которую ведет ссылка. Если пользователем задано ограничение на глубину кроулинга и ссылка такова, что у нее есть заголовок и ее глубина равна заданному пределу глубины, то нет смысла загружать страницу, на которую ведет эта ссылка – даже если страница содержит новые ссылки, они не будут добавлены на карту. В остальных случаях

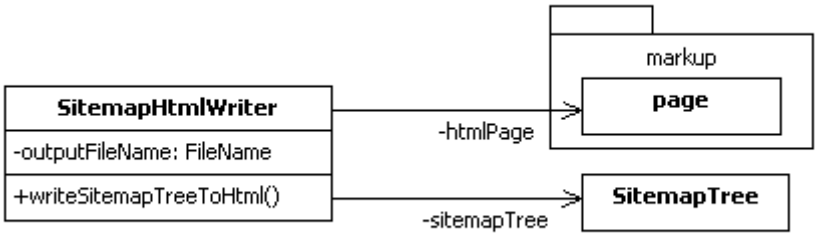
робот загружает страницу, если это разрешено файлом robots.txt. В случае невозможности установить соединение с сервером (например, произошел разрыв Интернет-связи) процесс кроулинга прерывается и карта сайта не может быть построена. В случае других ошибок (например, неправильно прописан URL в коде страницы) робот переходит к следующей итерации. Если страница была загружена, то робот делегирует ее разбор классу `SitePageParser`, получая информацию о заголовке страницы и извлеченных группах ссылок. Если текущая ссылка, извлеченная из стека, не имела заголовка, то он устанавливается как заголовок разобранной страницы. Если же разобранная страница заголовка не имеет (отсутствуют теги `<title></title>`), то ссылка не может быть добавлена на карту сайта. Если при разборе страницы экземпляром класса `SitePageParser` возникла ошибка, то робот переходит к следующей итерации. Если же все проходит успешно, то робот добавляет новый элемент на карту сайта, соответствующий текущей ссылке и помещает текущую ссылку в множество уже просмотренных ссылок.

Далее, в случае если задано ограничение на глубину кроулинга, и если только что обработанная ссылка имела глубину равную установленному пределу глубини, то робот переходит к следующей итерации, игнорируя извлеченные группы новых ссылок. В противном случае робот рассматривает группы извлеченных ссылок. Ссылки в каждой группе сначала нормализуются – относительные ссылки делаются абсолютными и при необходимости к ним добавляется “/” на конце. После этого они фильтруются – если ссылка ведет не на страницу сайта, или она соответствует ресурсу, который не является страницей, или она уже просмотрена, или она уже добавлена в стек для дальнейшей обработки, то ссылка игнорируется. Если в данной группе новых ссылок после фильтрации остались ссылки, робот анализирует, имеет ли эта группа заголовок. Если заголовок есть, то робот добавляет соответствующий заголовочный элемент на карту сайта. Затем он формирует информацию о каждой ссылке, которая потребуется ему для дальнейшей обработки, и помещает эту информацию в стек. Далее итерации повторяются, пока стек не опустеет.

Заметим, что при такой реализации текстовая ссылка (у которой есть заголовок), посещение которой запрещено файлом robots.txt, будет добавлена на карту сайта, но страница, соответствующая ей, загружаться не будет.

sitemap html writer

Содержит определение класса генератора HTML файла с результатом работы программы, а также класс исключения, генерируемого в случае ошибок. Использует сторонний модуль `markup`.



Для генерации HTML кода используется класс `page` библиотеки `markup`.

Карта сайта представляется в выходном HTML-файле в виде вложенных списков HTML–разметки (с тегами `<ul></ul>`). Структура элемента дерева карты сайта в HTML-документе имеет следующий вид:

```
<ul class="NodeContainer">
    <li class="SitemapNode">
            <div class="Expand"></div>
            <div class="NodeContent">
                    // Элемент карты – текстовая ссылка или заголовок группы ссылок
            </div>
            <ul class="NodeContainer">
                    // Дочерние элементы
            </ul>
    </li>
</ul>
```

К указанным классам добавляются дополнительные классы в зависимости от конкретного элемента карты. К классу SitemapNode добавляется класс Root, если элемент является корневым елементом, и/или один из классов ExpandClosed, ExpandOpen или ExpandLeaf, а также класс LastChild, если элемент является последним в списке дочерних элементов. К классу NodeContent добавляется класс TextReference_level_%d или Headline_level_%d в зависимости от того, каким именно элементом карты сайта является данный элемент. %d – число, соответствующее глубине элемента.

Классы Expand, ExpandClosed, ExpandOpen или ExpandLeaf я использую для отображения карты в виде интерактивного CSS дерева, упомянутого выше. Однако такой набор классов (или даже часть этого набора), на мой взгляд, вполне достаточен для последующего универсального отображения карты.

Также в HTML-документ добавляется источник для CSS-файла “sitemap.css” и источник для javascript-файла, которые можно использовать для отображения HTML-страницы с картой.

website_visualizer

Точка входа приложения. Управляет процессом создания карты сайта и осуществляет взаимодействие с пользователем.

Сборка и запуск приложения

Написанное мною приложение является консольным, оно запускается из командной строки.

Так как Python является интерпретируемым языком, то для запуска приложения достаточно иметь интерпретатор языка Python (я использовал Python 2.7). В этом случае специальной сборки приложение не требует. Однако, так как приложение использует стороннюю библиотеку `lxml`, то ее необходимо установить. Ее можно загрузить с официального сайта http://lxml.de/. Библиотека `markup` установки не требует, она распространяется в виде файла исходного кода и этот файл я включил в состав репозитория. Однако, при необходимости скачать эту библиотеку или прочитать о ней это можно сделать на странице проекта этой библиотеки на sourcefourge: http://markup.sourceforge.net/.

При необходимости можно создать исполняемый файл из исходного кода, специфичный для операционной системы при помощи соответствующих инструментов, к примеру, при помощи pyinstaller (http://www.pyinstaller.org/).

Запуск приложения осуществляется следующим образом:

```
website_visualizer.py <site address> <output file name> [<depth limit>]
```

Параметры запуска:

`<site address>` - URL сайта, карту которого требуется построить.

`<output file name>` - имя выходного HTML-файла.

`<depth limit>` – параметр, ограничивающий глубину карты сайта. Это необязательный параметр, но его рекомендуется задавать, особенно для больших порталов

для того, чтобы ограничить уровень детализации и избежать чрезмерно большого времени работы программы. Глубина главной страницы считается равной нулю. Нулевое значение данного параметра означает отсутствие ограничения глубины карты.

Результаты

Написанное приложение в силу сделанных предположений дает, на мой взгляд, хороший результат для порталов, для которых действуют сделанные предположения, в частности для предложенного для рассмотрения в качестве примера портала проекта Яндекс.Авто.

Я прилагаю к исходному коду программы в репозитории примеры сгенерированных моим приложением карт для сайтов Яндекс.Авто (http://auto.yandex.ru/) и CSCenter (http://compscicenter.ru/).

На моем компьютере построение карты для Яндекс.Авто с ограничением глубины, равным 1, занимает около 1 минуты (происходит фактически загрузка одной страницы и ее анализ, не считая тестового запроса и загрузки файла robots.txt). Для ограничения глубины, равного 1, время работы приложения составило уже около 2 часов. Карта сайта с таким ограничением уже дает более менее исчерпывающее представление о структуре портала Яндекс.Авто. Построение карты сайта CSCenter заняло также около минуты.

Особые замечания

Я прошу Вас принять во внимание то, что это приложение является моим первым знакомством с языком Python. До начала выполнения задания я не знал об этом языке ровно ничего, кроме названия. Также я в первый раз использую диаграммы UML.

Идеи по поводу дальнейшего развития проекта

Улучшение анализа страницы с целью выявления групп ссылок, характеризующих логическую структуру сайта.

Сделанное мною предположение о группировке семантически нагруженных ссылок верно для сайта портала Яндекс.Авто и ряда других сайтов. Но отнюдь не универсально. Требуется анализ большего количества сайтов для построения более универсального алгоритма.

Добавление поддержки новых элементов для карты сайта

Как я уже упоминал, ссылки, к примеру, могут быть не только текстовыми, они могут быть картинками. Помимо этого могут существовать другие элементы, которые позволят представить логическую структуру сайта более наглядно. Поэтому требуется добавлять поддержку новых элементов карты сайта.