Scrawl

Scrawl ist ein Website-Analystetool zur Generierung konfigurierbarer Sitemaps. Die Funktionalität wird über ein Command-Line Interface zur Verfügung gestellt.

Von *David Hettler* und *Nico Hein*

Benutzung

Die Syntax des Command-Line Interfaces lautet:

`scrawl <url> [<url>, ...] [-<option>, ...]`

- `<url>`: Eine Webseite, die gecrawlt werden soll. Es können mehrere URLs angegeben werden, die nacheinander mit den getroffenen Einstellungen gecrawlt werden.
- `-<option>`: Optionen für den Crawlprozess. Für eine Auflistung der verfügbaren Optionen kann der Befehl `scrawl -help` aufgerufen werden.

Aufsetzen des Projekts

Das Build-Skript `build.sbt` ist derzeit nur in der Lage die Projektabhängigkeiten herunterzuladen und das Programm auszuführen. Es ist jedoch nicht in der Lage eine ausführbare .jar-Datei auszugeben, wie zunächst angedacht war. Das Programm muss daher aus IntelliJ heraus gestartet werden und die Command-Line-Parameter über dessen Oberfläche eingestellt werden. Ist das Scala Plug-In installiert, erstellt IntelliJ beim Öffnen des Projektordners ein vollständig generiertes SBT-Projekt, das ohne weiteres Zutun lauffähig sein sollte.

Die API Docs können entweder

- mit `sbt doc` generiert werden und liegen anschließend im `doc/`-Verzeichnis
- oder sind auf *dieser Seite* zu finden

Projektstruktur

- `src/main/`: enthält das ausführbare Hauptprogramm.
- `src/cli/`: enthält das Kommandozeilensystem.
- `src/crawling/`: enthält das Crawlsystem.
- `src/graph/`: enthält die abstrakte Graphen-Datenstruktur für Sitemaps.
- `src/webgraph/`: enthält die konkrete Graphen-Datenstruktur.
- `src/test/`: enthält einige Tests für den Graphen.
- `src/analyze/`: enthält einen Websiteanalysealgorithmus.
- `resources/`: enthält die akka-Konfigurationsdatei.

Komponenten

Crawler

Der Crawler ist mit akka umgesetzt. Er besteht aus zwei Subsystemen: einem **Collectorsystem** und einem **Crawlersystem**.

- Im Crawlersystem arbeiten die Crawler-Worker (Aktoren), die jeweils eine Webseite herunterladen und analysieren. Ist der Vorgang beendet, sendet der Crawler eine Antwort mit der gecrawlten Seite an den Collector. Ein Crawler-Worker crawlt alle auf einer Webseite vorkommenden internen Links, indem er für jeden Link einen Worker erstellt und den Job an diese delegiert. Dies ist ein rekursiver Vorgang.
- Der Collector ist ein Aktor und dient als zentrale Anlaufstelle für die Crawler-Worker. Hat ein Worker eine Webseite gecrawlt, sendet er das Ergebnis an den Collector. Der Collector fügt den gecrawlten Link daraufhin in die Datenstruktur ein.

Während des Crawl-Vorgangs wird sichergestellt, dass jede Seite nur ein Mal heruntergeladen und analysiert wird. Stößt der Crawler auf eine besuchte Seite, wird der Link auf die Seite dennoch in die Datenstruktur eingefügt, auch wenn dieser nicht gecrawlt wird.

Datenstruktur

Zum Abspeichern der gecrawlten Websites wird ein Graph genutzt. Der abstrakte **Graph** (implementiert mit Traits und Generics) besteht aus einem Set von **Edges** und einem Set von **Nodes**. Jede Node und jede Edge hat ein **Label** (Dependency Injection) und jedes Label besteht aus einer Liste von Key-Value-Paaren (**LabelEntries**). Der Graph ist gerichtet und ungewichtet. In der Implementierung zeigt sich dies insbesondere dadurch, dass jede Node auf ihre ausgehenden Edges verweist und jede Edge auf ihre Start- und Endnode. (Der Graph kann durch Labels an Edges in einen gewichteten Graphen transformiert werden. So sind mehrere Gewichte möglich.) Der Graph stellt eine veränderbare Datenstruktur dar.

Implementierte Algorithmen im Graphen

Der abstrakte Graph lässt sich traversieren mit der:

1. Tiefensuche (`depthFirstTraversal`)
2. Breitensuche (`breadthFirstTraversal`)
3. Beschränken Breitensuche (`constraintBreadthFirstTraversal`): Bei dieser können Nodes wie auch Edges, die bestimmte Bedingungen nicht erfüllen, zum Abbrechen des Pfades führen. (Ein Beispiel: gehen wir davon aus, dass jede Node ein Label mit der eigenen Distanz zu der Node mit

der begonnen wird hat. Wird dann die Breitensuche mit einer maximal erlaubten Distanz beschränkt, so entspricht das Ergebnis dem einer Range Query.)

Auch ist der Dijkstra Algorithmus implementiert. Dieser setzt für jede Node zwei LabelEntries. Eines das einen Verweis auf die Parent Node enthält (um kürzeste Pfade zurückverfolgen zu können) und eines welches die Distanz zum Startknoten speichert. Der Basisalgorithmus auf einem ungewichteten Graphen nimmt für jede Edge die Länge 1 an. Es kann jedoch auch eine Gewichtsfunktion angegeben werden, die jede Edge auf eine Länge abbildet *(Genutzt für die Generierung der Seitenstruktur).*

Über die Funktionen `analyzeNodes` und `analyzeEdges` können alle Nodes bzw. Edges mir einem LabelEntry versehen werden. Der Inhalt des LabelEntries ergibt sich aus der übergebenen Funktion.

Konkretisierung des abstrakten Graphen

Die tatsächliche Implementierung des Graphen für den Crawler ist der **Webgraph**. Der Webgraph hat insbesondere eine Root. Allein durch die Art und Weise des Crawlens und den Fakt das nur **Weblinks** (konkretisierte Edges) hinzugefügt werden, ergibt sich, dass der Graph auch verbunden (connected) ist. Neben den Weblinks, die spezifizieren, dass sie ein **Weblabel** als Label nutzen, gibt es auch die **Webpage** - die Konkretisierung der Node. Diese führt die weiteren Variablen "url" (de facto Primärschlüssel für eine Webpage), "content" (speichert den Inhalt der Website) und "crawled" gibt an, ob die Seite bereits gecrawled wurde oder nicht. Auch die Webpage nutzt das Weblabel als Label.

Export des Graphen zu XML

Der Webgraph kann seinen Inhalt via XML beschreiben. Hierfür gibt es zwei Möglichkeiten.

1. Wird die einfache **XML** Version des Graphen angefordert, so werden alle Webpages mit all ihren Labels und ausgehenden Links sowie alle Weblinks mit all ihren Labels ausgegeben. (Für die Definition des Graphen würden die Nodes reichen. Sind jedoch die Weblinks auch mit Labels versehen, würde diese Information fehlen. Aus diesem Grund die Ausgabe.)
2. Wird die **Seitenstruktur** (`sitestructure`) des Graphen angefordert, so wird ein XML-Output generiert, der die Seitenstruktur abbildet. Wie die Seitenstruktur berechnet wird, wird im folgenden Abschnitt erklärt

Berechnung der Seitenstruktur

Eine Seitenstruktur entspricht der Hierarchie einer gecrawlten Website. Um die hierarchische Struktur der Website zu finden, wird der Dijkstra-Algorithmus mit Gewichtsfunktion auf dem Graphen ausgeführt. Dabei wird die Länge einer

Kannte mit einem Längenmaß - hier Distanzmaß auf Basis einer Ähnlichkeitsfunktion (siehe Paper) berechnet. *Dieses ist in* ***`analyze/AnalyzeURL`*** implementiert. **(Die Distanzfunktion nutzt einen heuristischen Parameter, der über den Parameter `-similarity` angepasst werden kann)** Wurde Dijkstra ausgeführt, so hat jede Webpage jene andere Webpage als Parent gesetzt, die die Ähnlichste URL zu sich selbst hat.

Wird nun das Sitestructure-XML generiert, so ergibt sich die hierarchische Struktur wie folgt: Einer Seite A sind jene Seiten untergeordnet, die A als Parent haben.

Command-Line Interface

Das Command-Line Interface ist auf Erweiterbarkeit ausgelegt. Neue Befehle, sowie deren Verhalten, können in dem `Argument` Companion-Objekt definiert werden. Soll beispielsweise ein neuer Analysealgorithmus implementiert werden, muss hierfür ein neues Argument Objekt erstellt werden, das, wenn es vom Command-Line-Interpreter ausgewertet wird, ein Funktionsobjekt in die Crawler-Einstellungen einfügt. Dieses Funktionsobjekt definiert den Analysealgorithmus, der automatisch auf jede gecrawlte Seite ausgeführt wird. Ein Analysealgorithmus könnte beispielsweise eine Liste aller Bilder aus dem HTML der gecrawlten Seite extrahieren. Die Liste der extrahierten Bilder würde daraufhin in der Sitemap unter dem jeweiligen Seiteneintrag auftauchen.