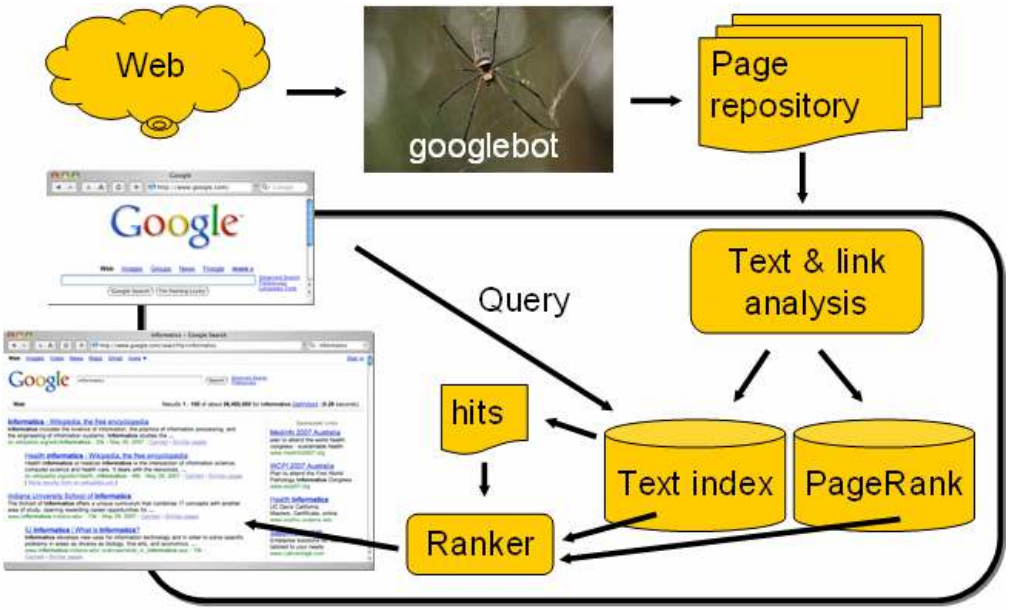
1. **Plan laboratorium X**
   - 1.1. Podstawowe zastosowania crawlerów
   - 1.2. Przebieg procesu crawlowania
   - 1.3. Charakterystyka crawlera
   - 1.4. Moduły crawlera
   - 1.5. Omówienie zadania do realizacji w zespołach
2. **Crawlowanie**

Skąd wyszukiwarka wie, że wszystkie te strony zawierają dany term?



Wiele nazw: crawler, spider, robot (bot), web agent, wanderer,worm, itd.

Gromadzenie stron z sieci w celu ich zindeksowania.

Dwa główne wymagania: szybkość i efektywność (zgromadź tak wiele użytecznych (useful) stron i linków między nimi, jak to tylko możliwe)

Zastosowanie:

- Wsparcie działania **wyszukiwarek** (Google, Yahoo, MSN/Windows Live, Ask, itd.).
- Wyszukiwarki **specjalizowane** dla określonego segmentu zawartości online, np. wiadomości, zakupy, przepisy, recenzje, itd.
- **Inteligencja biznesowa** (business intelligence): śledzenie działań rywali, partnerów.
- Monitorowanie interesujących stron.
- Zło :) zbieranie e-maili dla późniejszego wysłania **spamu**, phishingu.

Proces crawlowania (sekwencyjnego):

a) Crawler rozpoczyna od **zbioru początkowego** (seed set) adresów URL do pobrania

Inicjalizacja listy Q

b) Crawler pobiera i parsuje strony odpowiadające adresom; potem wyciąga z nich **tekst oraz linki**

Dopóki Q nie jest puste lub nie upłynął limit czasu:

Pobierz URL, L, z początku Q

Jeśli L nie jest stroną HTML (.gif, .jpeg, .ps, .pdf, .ppt…), kontynuuj

Jeśli L już odwiedzono, kontynuuj

Załaduj stronę P znajdującą się pod adresem L

Jeśli nie można pobrać P (np. 404, polityka względem robotów), kontynuuj.

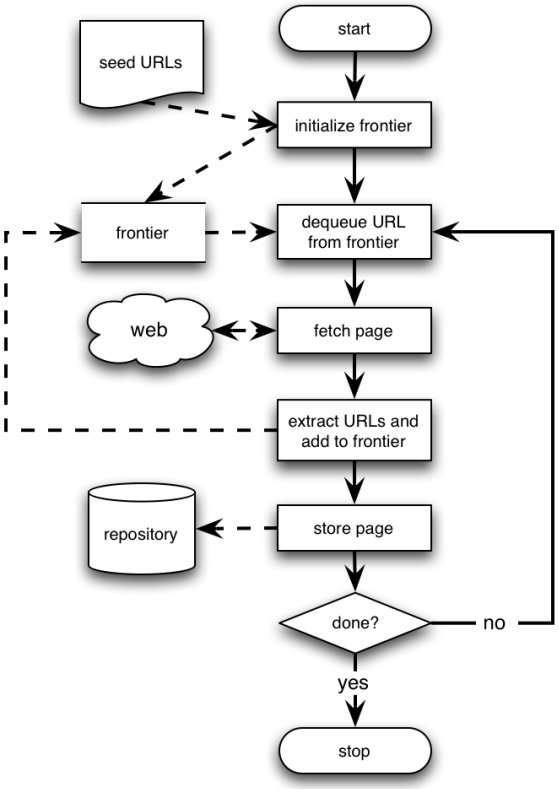
Parsuj P, by otrzymać listę nowych linków N

c) Dla nowych (lub zmienionych) stron, tekst jest wrzucany na wejście programu ***indeksującego***; linki (URL) są dodawane na ***listę adresów URL do odwiedzenia*** (URL frontier, crawling agenda)

Zindeksuj P (np. dodaj do ineksu odwrotnego)

Dodaj N na koniec listy Q

d) (ciągłe crawlowanie) Już pobrane strony są doklejone to *URL frontier* w celu odwiedzenia ich w późniejszym terminie



Najprostszy crawler w Perlu:

```
my @frontier = read_seeds($file);
while (@frontier && $tot < $max) {
    my $next_link = shift @frontier;
    my $page = fetch($next_link);
    add_to_index($page);
    my @links = extract_links($page, $next_link);
    push @frontier, process(@links);
}
```

Crawler musi:

- ***rozpoznawać strony, które odwiedził*** (sieć to graf, a nie drzewo);
- ***efektywnie indeksować*** odwiedzone strony; najczęściej indeksowanie stron z wykorzystaniem URL jako klucza;
- ***pobrać*** stronę w celu jej zindeksowanie pod względem zawartości;
- musi ***przestrzegać ograniczeń*** zdefiniowanych przez właścicieli stron (robot exclusion).

Cechy dobrego crawlera:

- ***jakość*** (quality) – wykrycie najbardziej użytecznych stron w celu zindeksowania ich w pierwszej kolejności;
- ***uprzejmość*** (politeness) – odpowiednio dostosowana polityka częstości wizyt crawlera;
- ***odporność*** (robustness) – umiejętność poradzenia sobie ze spider-traps (cykle, strony dynamiczne; nie wywalać się, jeśli pobranie strony się nie powiodło; mechanizm timeout);
- ***efektywność*** (efficiency) – mądrość w użytkowaniu procesora, pamięci (np. tak mało uśpionych procesów jak to tylko możliwe);
- ***świeżość*** (freshness) – ciągłość crawlowania sieci (częstotliwość odwiedzania strony powinna być bliska jej częstotliwości modyfikowania);
- ***rozszerzalność*** (extensibility) – wspieranie nowych formatów danych (np. bazujących na XML), nowych protokołów (np. ftp), itd.;
- ***rozproszenie*** (distribution) – jeśli to możliwe i crawlowane ma być sporo stron, to proces powinien być rozproszony na wiele maszyn;
- ***skalowalność*** (scalability) – proces crawlowania powinien być łatwo rozszerzalny w przypadku dodania nowych maszyn i zasobów.

Architektura crawlera – niezbędne moduły

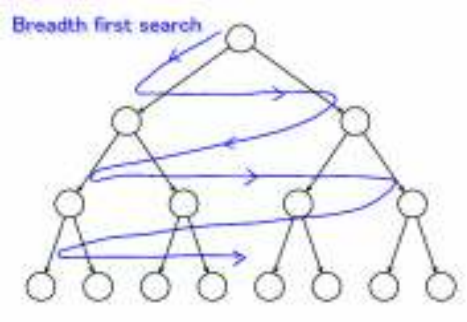
- **URL frontier** – zarządzanie adresami URL stron do pobrania.
- **DNS resolution** – określanie hosta (serwera), z którego należy pobrać stronę.
- **Moduł pobierania** – pobranie strony w celu jej dalszego przetwarzania.
- **Moduł parsowania** – identyfikacja tekstu i linków.
- **Eliminacja duplikatów** – wykrywanie URL (i ewentualnie zawartości), które zostały przetworzone niedawno.

URL frontier

Zadanie: zarządzanie kolejnością odwiedzania URL.

Strony o wysokiej jakości, które są często odświeżane powinny być częściej crawlowane (mieć wyższy priorytet)

Crawler może wykorzystywać różne sposoby crawlowania **breadth-first search** (standardowa metoda; FIFO (dodajemy na koniec listy); jeśli zaczniemy od dobrych stron, to inne dobre strony powinny być blisko) albo **depth-first search** (LIFO (dodajemy na początek listy) daje depth-first search; „lost in cyberspace")



Topic-Directed Spidering

- Preferuj strony dotyczące określonego tematu.
- Dany jest opis danego tematu lub próbki stron, które są przedmiotem zainteresowania
- Posortuj strony w kolejce wg podobieństwa stron, które do nich linkowały i/lub tekstów z linków do opisu tematu.

Link-Directed Spidering

- Dokonaj analizy linków wchodzących i wychodzących dla każdej strony..
- Posortuj kolejkę stron wg liczby linków wchodzących (strony popularne, authorities) lub liczbę linków wychodzących (strony podsumowujące, drogowskazy w sieci; hubs)

Crawler może być ograniczony do działania na ***określonym hoście lub katalogu*** (usuwanie linków z kolejki stron do odwiedzenia).

Sieć jest dynamiczna: wiele nowych stron, uaktualnienia, inne strony znikają, itd.

Crawler okresowo musi sprawdzać uaktualniania i usunięcia: header info (META tag dot. ostatniej aktualizacji)

Określ, jak często dana strona jest odświeżana i dostosuj do tego częstość crawlowania

DNS resolution

Domain Name System.

Tłumaczenie nazwy serwera i domeny na adres IP.

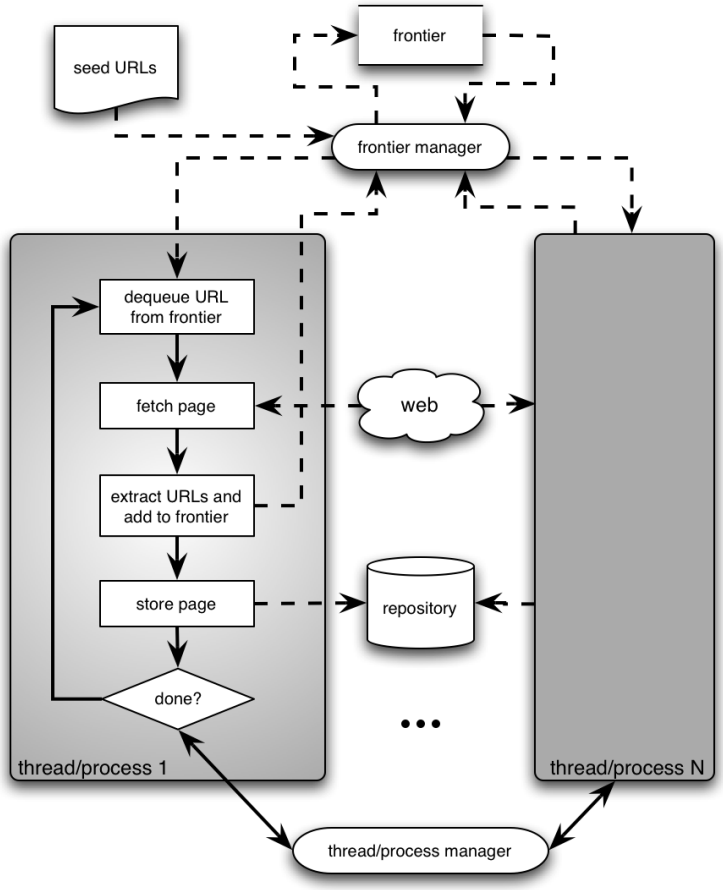
Działa jak „książka telefoniczna”.

Konieczność komunikacji z serwerem DNS (które być może będzie musiał skontaktować się z innymi serwerami DNS)

„Wąskie gardło” crawlowania.

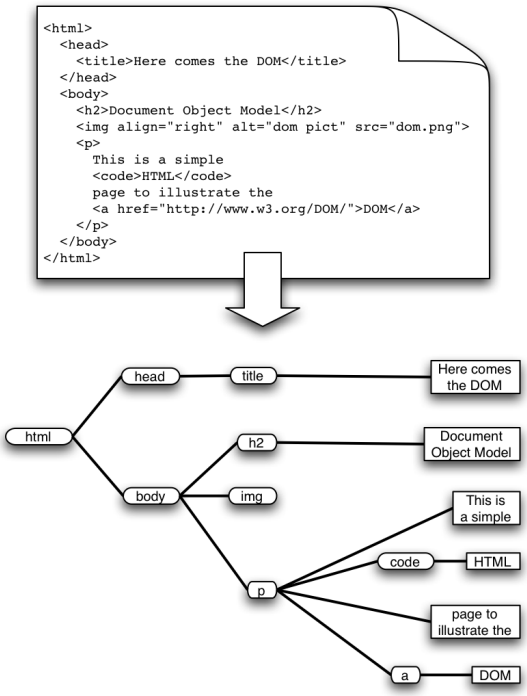
Współbieżność:

- Działanie crawlera jest narażone na wiele opóźnień:
  - Przetłumaczenie host name z URL na adres IP z wykorzystaniem DNS
  - Połączenie z serwerem i wysłanie żądania
  - Otrzymanie żądanej strony w odpowiedzi
  - Nowe żądania są przetwarzane dopiero, gdy aktualne żądania jest obsłużone
- Rozwiązanie: Nałożenie opóźnień na siebie przez pobieranie wielu stron równocześnie
- Wiele wątków/procesów
- Każdy wątek/process działa jak crawler sekwencyjny, z zastrzeżeniem, że wszystkie dzielą struktury danych: URL frontier i repozytorium danych
- Dostęp do dzielonych struktur danych musi być synchronizowany (blokady dla współbieżnych zapisów)
- Łatwo osiągnąć dzięki temu przyspieszenie rzędu 5-10 razy



Parsowanie

- HTML ma strukturę drzewa DOM (Document Object Model)
- Wiele stron jest niepoprawnych pod względem składni stron
- Na szczęście, istnieją narzędzia, które pozwalają z tym walczyć, np. ***tidy.sourceforge.net***



Link Extraction

URL Syntax

<scheme>://<authority><path>?<query>#<fragment>

- *authority* przekłada się na <host>:<port-number>
- *query przekłada się na* <variable>=<value>&<variable>=<value>…
- *fragment* (*reference*, *ref*) *wskaźnik w ramach dokumentu służący do przejścia do tekstu oznaczonego tagiem* <A NAME="<fragment>">

Znalezienie i wydobycie wszystkich linków na stronie

<a href="http://www.cs.utexas.edu/users/mooney/ir-course">

<frame src="site-index.html">

Uzupełnienie adresów względnych z wykorzystaniem aktualnego adresu URL (base URL z nagłówka http lub meta tagu http)

<a href="lab10"> to http://www.cs.put.poznan.pl/mkadzinski/ezi/lab10

<a href="../wd/kolo.html"> to http://www.cs.put.poznan.pl/mkadzinski/wd/kolo.html

URL canocalization

Wszystkie poniższe adresy:

- http://www.cnn.com/TECH
- http://WWW.CNN.COM/TECH/
- http://www.cnn.com:80/TECH/
- http://www.cnn.com/bogus/../TECH/

odpowiadają: http://www.cnn.com/TECH/ i trzeba je sprawdzić do tej formy, by uniknąć duplikacji

Niektóre transformacje są trywialne:

- ✓ http://cs.put.poznan.pl
- ✓ http://cs.put.poznan.pl/

- ✓ http://cs.put.poznan.pl/index.html#fragment
- ✓ http://cs.put.poznan.pl/index.html

- ✓ http://cs.put.poznan.pl/dir1/./../dir2/
- ✓ http://cs.put.poznan.pl/dir2/

- ✓ http://cs.put.poznan.pl/%7Efil/
- ✓ http://cs.put.poznan.pl/~fil/

- ✓ http://CS.PUT.POZNAN.PL/fil/
- ✓ http://cs.put.poznan.pl/fil/

Inne transformacje wymagają heurystycznych założeń co do intencji autora lub konfiguracji serwera:
Usunięcie domyślnej nazwy:

- ✓ http://cs.put.poznan.pl/fil/index.html
- ✓ http://cs.put.poznan.pl/fil/

Rozsądne w ogólnym wypadku, ale czasem domyślną stroną może być np. default.asp

Dodanie / przy założeniu o wystąpieniu katalogu:

- ✓ http://cs.put.poznan.pl/fil
- ✓ http://cs.put.poznan.pl/fil/

Skąd pewność, że nie ma tam pliku o nazwie “fil”?

Spider traps:

Zwrócić uwagę na wszystko, co może prowadzić do zbyt wielu żądań do jednego serwera.
Nieskończona liczba stron generowanych automatycznie przez skrypty CGI.
„Dziwne” ścieżki.
Obrona: sprawdzać długość URL (próg np. 128 znaków), uważać na strony z dużą liczbą linków; eliminować URL z nietekstowymi typami danych; zabronić crawlowania dynamicznych stron (jeśli można je wykryć)

Pobieranie strony

Podaj swoją tożsamość:

- Wykorzystaj nagłówek HTTP ‘User-Agent’ do identyfikacji crawlera, podania strony z jego opisem i kontaktu do twórcy (e-mail)
- W żadnym wypadku nie wolno fałszować swojej tożsamości (np. kopiować tekst podawany przez przeglądarki jako ‘User-Agent’)

Podstawowe zasady:

- Przestrzegania zasady uprzejmości (brak powtórnych pobrań stron z danego hosta).
- Tylko jedno otwarte połączenie z danym hostem w określonym momencie.
- Poczekaj kilka sekund pomiędzy kolejnymi żądania do hosta.

Najprostszy sposób: przechowuj każdą stronę w osobnym pliku;
generuje dużą liczbę plików; dla większych zastosowań nieefektywne.

Lepszy sposób (dla większych zastosowań): złóż wiele stron w jeden plik i użyj znacznika XML do ich oddzielenia (konieczne mapowanie URL na {filename, page_id})
Baza danych: RDBMS; Berkeley DB
Koniecznie zawsze sprawdź, czy żądanie http zakończyło się sukcesem; w przypadku błędu trzeba zidentyfikować problem (kod HTTP) i odpowiednio go zaadresować.

Robots Exclusion

Serwisy i strony określają, czy crawlery/roboty mogą/nie mogą indeksować dany obszar.

Dwie możliwości:

Robots Exclusion Protocol: specyfikacja katalogów wyłączonych z crawlowania dla danych robotów (user-agent) poprzez umieszczenie pliku „robots.txt" w korzeniu hosta; zobacz np. cnn.com/robots.txt, ebay.com/robots.txt.

Przykłady: Zakaz działania wszystkich robotów w całym serwisie

User-agent: *

Disallow: /

Zakaz crawlowania określonych folderów:

User-agent: *

Disallow: /tmp/

Disallow: /cgi-bin/

Zakaz crawlowania dla określonych robotów:

User-agent: GoogleBot

Disallow: /

Pozwolenie na crawlowanie określonemu robotowi:

User-agent: GoogleBot

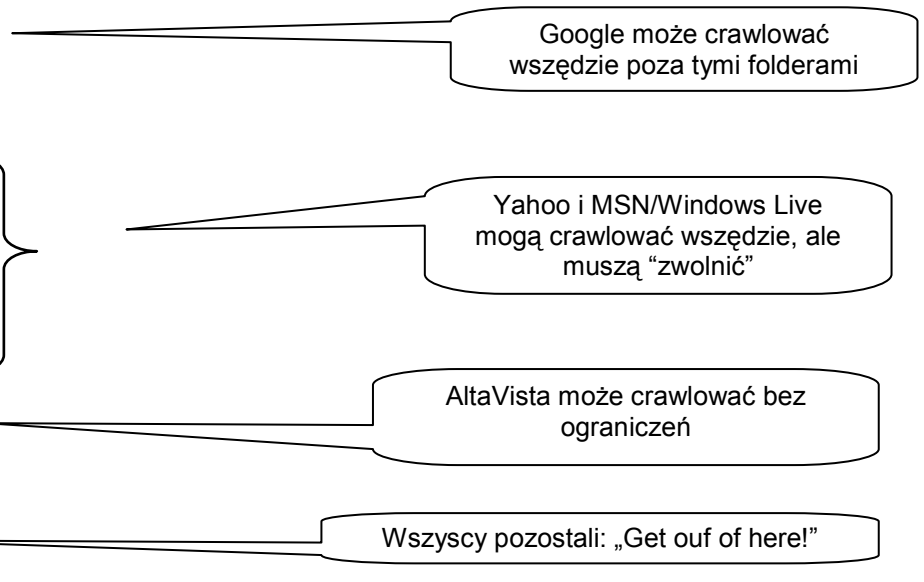
Disallow:

User-agent: *

Disallow: /

Podstawowe zasady: puste linie, by oddzielić User-agents; jeden katalog na linię Disallow; nie można wykorzystywać wyrażeń regularnych.

```
# Robots.txt for http://www.springer.com (fragment)

User-agent: Googlebot
Disallow: /chl/*
Disallow: /uk/*
Disallow: /italy/*
Disallow: /france/*

User-agent: slurp
Disallow:
Crawl-delay: 2

User-agent: MSNBot
Disallow:
Crawl-delay: 2

User-agent: scooter
Disallow:

# all others
User-agent: *
Disallow: /
```



Robots META Tag

Nowszy i mniej rozpowszechniony sposób niż „robots.txt”

Uwzględnij META tag w sekcji HEAD danej strony:

<meta name=”robots” content=”none”>

przy czym content to para wartości dotyczących dwóch aspektów:

index | noindex: pozwól/zabroń indeksować stronę;

follow | nofollow: pozwól/zabroń na poruszanie się po linkach na stronie.

Wartości specjalne: all=index,follow; none=noindex,nofollow

Przykłady: <meta name=”robots” content=”noindex,follow”>, <meta name=”robots” content=”none”>

Przykłady crawlerów:

- Reference C implementation of HTTP, HTML parsing, etc
  - w3c-libwww package from World-Wide Web Consortium: www.w3c.org/Library/
- LWP (Perl)
  - http://www.oreilly.com/catalog/perllwp/
  - http://search.cpan.org/~gaas/libwww-perl-5.804/
- Crawlery open source
  - **Nutch: http://www.nutch.org/ (Jakarta Lucene: jakarta.apache.org/lucene/)**
  - Heretrix: http://crawler.archive.org/
  - WIRE: http://www.cwr.cl/projects/WIRE/
  - Terrier: http://ir.dcs.gla.ac.uk/terrier/
- Open source topical crawlers
  - http://informatics.indiana.edu/fil/IS/JavaCrawlers/

3. Zadanie do realizacji w grupach

[9] Zadanie składa się z dwóch części, przy czym do realizacji drugiej konieczne jest wykonanie pierwszej. Dla uproszczenia dostępu do stron w sieci wykorzystany zostanie istniejący pakiet, który powstał dla potrzeb podobnego jak nasz kursu Information Retrieval and Web Search na University of Texas pod kierownictwem Raya Mooney i Teda Wilda. Dokumentacja kodu znajduje się pod adresem:

http://www.cs.utexas.edu/~mooney/ir-course/doc/index.html

Na oryginalny kod z UoT naniesiono kilka drobnych poprawek, które miały na celu wyeliminowanie błędów kompilacji dla nowszych wersji Java. Stąd lab10.jar, który jest do pobrania z katalogu lab10 różni się nieznacznie od ir.jar, który jest udostępniony na stronach UoT.

Najważniejsza klasa, do której będziesz się odwoływał/a to ***Spider*** z pakietu ir.webutils. Użycie metody main() z odpowiednimi opcjami pozwala na rozpoczęcie procesu crawlowania od podanego URL, crawlowanie z wykorzystaniem metody breadth-first search oraz zachowanie stron/dokumentów w podanym folderze w celu ich późniejszego zindeksowania i przeszukiwania. Wywołanie funkcji ***main()*** z tej klasy przyjmuje kilka parametrów:

-safe - oznacza, że przestrzegane będą zapisy plików typu robot exclusion;

-u <start-URL> - określa stronę, od której rozpocznie się crawlowanie;

-d <save-directory> - określa folder na dysku lokalnym, w którym zachowywane będą strony (**uwaga: folder należy utworzyć zanim rozpoczniesz crawlowanie**); strony po zachowaniu w katalogu mają zmieniony nazwy na Pindeks.html (np. P001.html);

-c <page-count-limit> - określa maksymalną liczbę stron, które zostan pobrane.

Aby dokonać crawlowania kolekcji **stron, na której sprawdzane będzie rozwiązanie poniższych zadań**, należy użyć opcji:

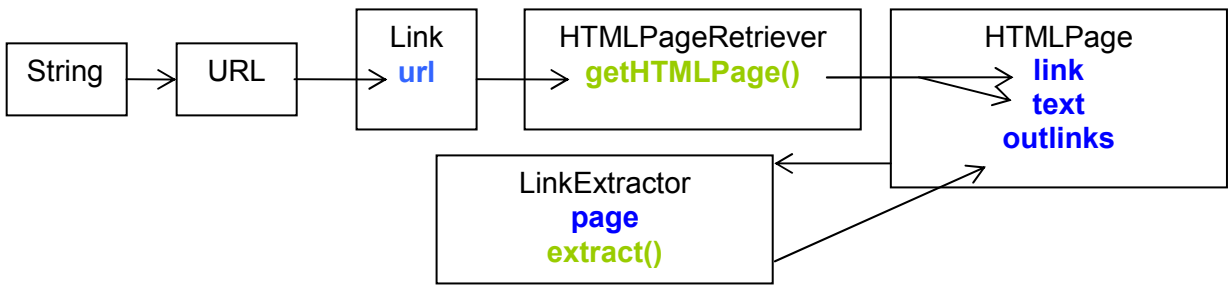
-u http://www.cs.put.poznan.pl/mkadzinski/ezi/dzienne/lab10/crawler/anaconda/anaconda01.html

-d repository-folder -c 100

Wykonanie tego polecenia powinno skutkować pobraniem 64 stron (jeśli zaczniesz crawlowanie od dokumentu anaconda01.html, to gwarantowany jest dostęp do wszystkich dokument) i zapisaniem ich w katalogu repository-folder, który uprzednio należy utworzyć w wybranym przez Ciebie miejscu.

Stworzono też specjalizacje klasy ***Spider***: ***SiteSpider*** ogranicza crawlowanie do podanego hosta, a ***DirectorySpider*** do określonego folderu.

Główne klasy wykorzystywane przez ***Spider*** przedstawiono na poniższym schemacie:



Zadanie:

Ściągnij lab10.jar, który zawiera pakiet information retrieval.

Możesz go rozpakować, żeby zobaczyć kod źródłowy poszczególnych klas.

Dla rozwiązania poniższych zadań nie powinieneś zmieniać kodu źródłowego, który został dostarczony (jeśli cokolwiek zmienisz – co nie jest wskazane – to musisz przesłać wszystkie klasy, które zmieniłeś).

a) [6] Utwórz ***specjalizację klasy Spider*** o nazwie ***PageRankSpider***, która oblicza PageRank crawlowanych stron na podstawie struktury połączeń pomiędzy nimi. Zaimplementuj (override) tylko te metody, których musisz użyć do rozwiązania zadania. W czasie crawlowania PageRankSpider powinien utworzyć graf sieci na bazie linków (wchodzących i wychodzących) między stronami. Do stworzenia grafu i manipulowania jego strukturą wykorzystaj klasy ir.webutils.***Graph*** oraz ir.webutils.***Node***. Graph jest tworzony na podstawie danych odczytanych z pliku w formacie:

w1 wierzchołek1_wskazywany_przez_w1 wierzchołek2_wskazywany_przez_w1…

w2 wierzchołek1_wskazywany_przez_w2 wierzchołek2_wskazywany_przez_w2 …

…

wi lista_wierzchołków_wskazywanych_przez_wi

Do grafu dodaj jako wierzchołki tylko te strony, które zachowano na dysku. Następnie uruchom algorytm PageRank dla tak utworzonego obiektu klasy Graph. Dla obliczenia miary PageRank utwórz klasę ***PageRankFile.java***, która ostateczne wyniki działania algorytmu zapisze w pliku tekstowym pageRanks, który powinien znajdować się w tym samym katalogu, co strony pobrane w procesie crawlowania **(pamiętaj o wcześniejszym utworzenie tego katalogu)**. Format pliku powinien być następujący:

nazwa_strony1.html wartość_PageRank1

nazwa_strony2.html wartość_PageRank2

Cały algorytm PageRank jest już zaimplementowany w klasie ir.webutils.***PageRank***, tyle że tam wyniki są wypisywane tylko na konsolę. Dla swoich obliczeń wykorzystaj parametry **alpha=0.15** oraz **iteration=50**. Log działania ***PageRankSpider*** powinien zawierać:

standardowe logi, które wypisuje Spider

strukturę wierzchołkow w formacie:

strona_1 -> [lista stron wskazywanych przez stronę_1], np. P1 -> [P2, P3]

strona_2 -> [lista stron wskazywanych przez stronę_2]

…

miary PageRank dla wszystkich zachowanych na dysku stron

PR(strona_1) PR_strony_1

PR(strona_2) PR_strony_2

…

Działanie poprawności ***PageRankSpider*** programu możesz sprawdzić dla kolekcji testowej w katalogu http://www.cs.put.poznan.pl/mkadzinski/ezi/dzienne/lab10/test. W treści plików w tym folderze są wypisane wartości PageRank obliczone z użyciem zaimplementowanego w pakiecie algorytmu (nie przejmuj się drobnymi różnicami w sposobie obliczenia PageRank w stosunku do algorytmu omówionego w ramach naszego przedmiotu).

b) [3] **Koniecznie umieść plik stopwords.txt w katalogu głównym projektu** (możesz go umieścić gdzie indziej pod warunkiem, że zrobisz sobie nowego jara, bo ścieżka do pliku stowords.txt jest niestety podana w kodzie). Utwórz ***specjalizację klasy InvertedIndex*** o nazwie ***PageRankInvertedIndex***, aby w nowy sposób

obliczyć adekwatność stron do zapytania. ***PageRankInvertedIndex*** powinien nie tylko indeksować dokumenty i umożliwiać zadawanie zapytań jak ***InvertedIndex***, ale również czytać wartości ***PageRank*** dokumentów, tzn. w chwili obliczenia miary podobieństwa dla danego zapytania, powinien dodać do klasycznie obliczonej miary dla danej strony wartość jej PageRanku przemnożoną przez parametr weight (podawany jako parametr z linii komend „-weight value"; czyli ostatecznie new_score = score + weight * PageRank). Wywołanie klasy InvertedIndex może przyjmować trzy parametry, po czym podawany jest folder, gdzie znajduje się kolekcja stron, która ma być indeksowana. Dla uproszczenia przyjmijmy, że lista parametrów ***PageRankInvertedIndex*** ma wyglądać w następujący sposób „-weight value repository-folder", a więc bez parametrów oryginalnie rozważanych w InvertedIndex.

Jako rozwiązanie prześlij:

- klasy ***PageRankSpider.java, PageRankFile.java*** oraz ***PageRankInvertedIndex.java***
- ***dwa trace'y*** z działaniu programu: jeden dla crawlowania stron; drugi dla interakcji programu z użytkownikiem z zadaniem trzech zapytań (np. „anaconda", „programming", „python") przez użytkownika i wyświetleniem wyników opartych na uwzględnieniu miary PageRank (np. dla weight=2).

Termin przesłania rozwiązania: 28 listopada