PROGRAMOWANIE OBIEKTOWE
ZADANIE LABORATORYJNE NR 2 “LUCYNA”

Wstęp

Celem zadania jest stworzenie aplikacji do wyszukiwania tekstu w dokumentach. Współczesne systemy operacyjne posiadają podobną, wbudowaną funkcjonalność, jednak bywa, że mechanizm ten zachowuje się w sposób nieoczywisty dla przeciętnego użytkownika, np. dlatego, że wyszukiwanie w *zawartości* pliku jest domyślnie wyłączone. Narzędzia opracowane w ramach tego zadania laboratoryjnego mają być zorientowane na indeksowanie zawartości dokumentów w popularnych formatach (m.in. PDF, nie tylko plikach czysto tekstowych). By jednak nie była to kolejna odmiana narzędzia `grep` aplikacja powinna “rozumieć” wewnętrzną strukturę tych dokumentów (na szczęście sam implementujący aplikację szczegółów tej struktury znać nie musi; zob. sekcja Ekstrakcja tekstu z dokumentów poniżej). Ponadto, wyszukiwanie tekstu musi być szybkie, również w przypadku, gdy zbiór przeszukiwanych dokumentów jest duży i składa się z kilkuset, a nawet kilku tysięcy plików - każdorazowe skanowanie zawartości wszystkich dokumentów w poszukiwaniu zadanego w zapytaniu wzorca będzie tutaj niewystarczające. Stąd potrzeba utworzenia *indeksu*, który znacząco przyspieszy proces wyszukiwania.

Rozwiązanie powinno składać się z dwóch komplementarnych narzędzi, zaimplementowanych jako osobne programy. Zadaniem pierwszego z tych narzędzi - zwanego dalej krótko *indekserem* - jest wyodrębnienie tekstu z plików znajdujących się w drzewie katalogów (zob. sekcja Ekstrakcja tekstu z dokumentów), budowa (zob. sekcja Indeksowanie dokumentów) i aktualizacja (zob. sekcja Aktualizacja indeksu) indeksu. Drugi program - *wyszukiwarka* - ma umożliwiać użytkownikowi zadawanie zapytań i prezentować ich wyniki (zob. sekcja Wyszukiwarka).

Rozwiązanie powinno być zaimplementowane w języku Java, w wersji 8 (wtedy powinno kompilować się za pomocą kompilatora `/usr/bin/javac` na maszynie `students`) lub w wersji 11 (powinno kompilować się za pomocą kompilatora `/opt/jdk-11.0.2/bin/javac`). Wskazane jest skorzystanie w rozwiązaniu z narzędzia automatyzującego budowanie projektu, np. Apache Maven. Indekser i wyszukiwarka powinny być zaimplementowane w jednym projekcie.

Prosimy wysyłać rozwiązania przez platformę Moodle, najlepiej w postaci kompletnych projektów, zawierających konfigurację narzędzi automatyzujących budowanie projektu (plik `pom.xml` w przypadku narzędzia Maven) i/lub pliki konfiguracyjne projektu utworzone przez środowiska deweloperskie (Intellij, Eclipse, itd.).

Tutaj dostępny jest przykładowy projekt zawierający konfigurację dla narzędzia Maven ze wszystkimi bibliotekami wymienionymi w niniejszej specyfikacji.

Termin nadsyłania rozwiązań upływa 15 czerwca.

Ekstrakcja tekstu z dokumentów

Rozwiązanie powinno zapewniać wyszukiwanie w dokumentach następujących typów:

- dokumenty czysto tekstowe (TXT),
- dokumenty w formacie Portable Document Format (PDF),
- dokumenty w formacie Rich Text Format (RTF),
- dokumenty w formacie Open Document Format (ODF; w szczególności pliki `.odt`),
- dokumenty w formacie OpenXML (OOXML; w szczególności pliki `.docx`),

przy czym o typie dokumentu decyduje nie rozszerzenie pliku, w którym jest zapisany, a jego zawartość.

Fundacja Apache udostępnia zestaw kilku bibliotek dla maszyny wirtualnej Javy służących do parsowania dokumentów w wymienionych formatach. Wspólną fasadę dla tej grupy narzędzi stanowi biblioteka Apache Tika - dzięki niej możliwe jest dokonanie ekstrakcji tekstu z dokumentu do obiektu klasy `String`. Zalecane jest użycie tej biblioteki w implementowanym rozwiązaniu po uprzednim zapoznaniu się z krótkim samouczkiem. Zdecydowanie *niewskazane* natomiast jest podejmowanie próby stworzenia własnego parsera dla tych (niekiedy skomplikowanych) formatów dokumentów.

Można założyć, że parsowane dokumenty napisane są w języku polskim albo angielskim (będzie to istotne przy indeksowaniu dokumentów i obsłudze zapytań). Biblioteka Tika posiada również zestaw detektorów do wykrywania języka, w jakim napisany jest dokument (w rozwiązaniu można skorzystać np. z klasy OptimaizeLangDetector).

Ponadto można przyjąć, że biblioteka Tika wykona prawidłową konwersję (o ile będzie ona potrzebna) kodowania znaków używanego wewnętrznie w dokumencie do UTF-16 (czyli kodowania używanego przez maszynę wirtualną Javy do przechowywania napisów; innymi słowy: produktem parsowania dokumentu jest prawidłowy napis/kolekcja napisów w języku Java). Dopuszczamy sytuację, w której wynik parsowania będzie niepełny lub wręcz niepoprawny (plik z dokumentem może być uszkodzony, niezgodny ze specyfikacją lub, z powodu błędu w implementacji parsera, nie jest poprawnie obsługiwany). Należy wtedy, o ile biblioteka Tika zgłosiła błąd w trakcie przetwarzania dokumentu, pominąć dany plik (w konsekwencji nie będzie on później uwzględniany przy indeksowaniu i wyszukiwaniu). Natomiast, jeśli parsowanie zakończyło się sukcesem, można przyjąć, że wynik jest prawidłowy i przekazać go do dalszego indeksowania. Błędy zgłaszane przez bibliotekę Tika nie powinny powodować przerwania działania indeksera (por. sekcja Obsługa błędów).

W przypadku napotkania pliku, który nie reprezentuje dokumentu w żadnym z wymienionych wyżej formatów, można przyjąć dowolne rozsądne rozwiązanie (takim będzie poleganie na detektorach typów dokumentów, którymi dysponuje biblioteka Tika - jeżeli parsowanie pliku nie powiodło się, to jest on po prostu pomijany).

Za ekstrakcję tekstu z dokumentów powinien odpowiadać indekser.

Indeksowanie dokumentów

Apache Lucene to biblioteka dla maszyny wirtualnej Javy do wyszukiwania pełnotekstowego. Na rynku dostępnych jest kilka kompleksowych platform zbudowanych na bazie Lucene (np. Apache Solr; Solr i Lucene należą do jednego stosu technologicznego i rozwijane są w ramach wspólnego projektu). Lucene oferuje wysoką wydajność wyszukiwania, dzięki czemu sprawdza się nawet w dużej skali, przy przetwarzaniu wielkich zbiorów danych w czasie rzeczywistym. Implementację rozwiązania niniejszego zadania sugerujemy oprzeć właśnie o tę bibliotekę i dlatego poniżej, ilekroć będzie mowa o indeksie, będzie to oznaczać indeks Lucene.

Lucene przechowuje indeks, czyli ustrukturalizowaną informację o dokumentach tekstowych, w postaci plików zapisanych na dysku (alternatywnie, indeks może być przechowywany w całości w pamięci operacyjnej, ale w tym zadaniu takie rozwiązanie nie powinno być rozważane). To, jak wygląda wewnętrzna struktura indeksu nie jest z reguły istotne dla użytkownika biblioteki. Lucene

udostępnia interfejs programowy do zarządzania indeksem - dodawania i odczytywania informacji, które w nim się znajdują. Służą do tego klasy `IndexWriter` oraz `IndexReader`.

Jednym z centralnych pojęć używanych przez bibliotekę Lucene jest *dokument*, reprezentowany przez klasę Document. Elementami indeksu są właśnie dokumenty, które z kolei stanowią kolekcję *pól* (klasa Field i jej podklasy, w tym klasa TextField). Sugerujemy przeczytanie krótkiego samouczka dotyczącego pracy z biblioteką Lucene.

Głównym zadaniem indeksera jest wykonanie przejścia rekurencyjnego drzewa katalogów, ekstrakcja tekstu zawartego w odnalezionych plikach (o ile jest to plik jednego z obsługiwanych przez parser typów), stworzenie dokumentu Lucene odpowiadającego każdemu plikowi i zapis dokumentów do indeksu.

Przejście rekurencyjne po drzewie katalogów może być zrealizowane za pomocą metody `Files.walkFileTree()` (ew. `Files.walk()`) - bibliotecznej implementacji wzorca projektowego Odwiedzający dla drzewa katalogów. W procesie tym należy pominąć napotkane dowiązania symboliczne (ang. symbolic links) w systemie plików (tzn. z metody `Files.walkFileTree()` można skorzystać bez podawania opcji `FOLLOW_LINKS`). Indeksowaniu powinny podlegać jedynie zwykłe pliki (ang. regular file; przydatna tutaj może być biblioteczna metoda `Files.isRegularFile()`).

Dokumenty w indeksie powinny być zbudowane w ten sposób, by zapewniać możliwość wyszukiwania tekstu w następujących trybach (por. sekcja Wyszukiwarka):

- wyszukiwanie pojedynczych słów kluczowych - termów (ang. term query),
- wyszukiwanie fraz (ang. phrase query),
- wyszukiwanie rozmyte (ang. fuzzy query).

Należy przy tym pamiętać, że przy wykonywaniu zapytań uwzględniane są jedynie dane zapisane do indeksu, natomiast oryginalne pliki, na podstawie których indeks został wygenerowany, nie biorą udziału w tym procesie (dlatego tekst wyodrębniony z plików powinien być zduplikowany i zapisany w dokumencie Lucene jako część indeksu).

Wymagamy, by indeksowana była tekstowa zawartość pliku (wyodrębniona za pomocą narzędzia Apache Tika) oraz jego nazwa (bez ścieżki). Tika umożliwia również pobranie dodatkowych metadanych pliku dla niektórych typów plików (np. nazwisko autora dokumentu), ale te metadane mogą nie być uwzględniane przy indeksowaniu i wyszukiwaniu.

Przy budowie indeksu istotny jest wybór *analizatora* leksykalnego - odpowiada on za podział tekstu na najmniejsze jednostki leksykalne (tokeny), może przeprowadzić proces stemmingu, czyli usunięcia końcówki fleksyjnej wyrazu - dzięki temu różne formy fleksyjne tego samego wyrazu można utożsamić (np. wyrazy "analiza" i "analizą"). Lucene udostępnia analizatory dla języka polskiego, języka angielskiego oraz analizator generyczny. Wybór używanego analizatora powinien nastąpić na po wykryciu języka w jakim napisany jest dany dokument (przydatna tutaj będzie klasa AnalyzerWrapper; warto zapoznać się z przykładem użycia).

Indeks powinien być zapisany na dysku, w podkatalogu o nazwie `.index` w katalogu domowym użytkownika (ścieżkę do katalogu domowego można uzyskać za pomocą wywołania `System.getProperty("user.home")`)

Należy zwrócić uwagę na to, by pliki zawierające indeks zostały poprawnie zamknięte po zakończeniu zapisu (`IndexWriter.close()` lub niejawnie za pomocą konstrukcji try-with-resources).

Przykładowy kod wykonujący indeksowanie zawartości drzewa katalogów można znaleźć w repozytorium projektu Lucene oraz w dostarczonym do niniejszego zadania projekcie.

Aktualizacja indeksu

Indekser powinien zaktualizować informacje zapisane w indeksie, w przypadku gdy indeksowany plik uległ zmianie. W tym celu należy skorzystać z mechanizmu notyfikacji o zmianach w systemie plików[1] dostarczanego przez standardową biblioteką klas Javy. Umożliwia on monitorowanie wskazanego katalogu (lub drzewa katalogów) i dostarcza informacje o zmianach, które w nim zaszły.

Wyróżniamy następujące kategorie zmian w systemie plików, istotne z punktu widzenia tego zadania:

- utworzenie pliku w monitorowanym drzewie katalogów - należy wtedy wykonać parsowanie pliku i dodać nowy dokument do indeksu,
- usunięcie pliku z monitorowanego drzewa katalogów - należy usunąć odpowiadający dokument z indeksu,

[1] Wprawdzie w podanym artykule znajduje się zastrzeżenie, że mechanizm ten nie powinien być używany do indeksowania zasobów (powodem jest brak wsparcia dla notyfikacji o zmianach w niektórych systemach plików - wtedy implementacja mechanizmu w Javie stosuje aktywne odpytywanie systemu plików, co może być niewydajne). Jednak należy przyjąć, że w tym zadaniu z tego mechanizmu można skorzystać w celach edukacyjnych, a rozwiązanie testowane będzie na systemach oferujących natywną obsługę notyfikacji o zmianach.

- modyfikacja istniejącego pliku w monitorowanym drzewie katalogów - należy dokonać aktualizacji dokumentu w indeksie (po ponownym wykonaniu parsowania pliku),
- przeniesienie pliku w obrębie monitorowanego drzewa katalogów - należy potraktować tę zmianę, jak 2 niezależne operacje złożone z usunięcia i dodania nowego pliku,
- utworzenie nowego katalogu w monitorowanym drzewie katalogów - należy wykonać (rekurencyjne) indeksowanie dokumentów w dodanym katalogu,
- usunięcie katalogu z monitorowanego drzewa katalogów - należy usunąć wszystkie dokumenty z indeksu odpowiadające plikom z usuniętego poddrzewa katalogów,
- przeniesienie katalogu w obrębie monitorowanego drzewa katalogów - analogicznie, jak w przypadku przenoszenia pliku.

Dynamika zmian w monitorowanych katalogach nie będzie na tyle duża, by powodować przepełnienie kolejki zdarzeń (ewentualne przepełnienie byłoby sygnalizowane przez zdarzenie typu OVERFLOW - można przyjąć założenie, że to zdarzenie nie wystąpi).

Program demonstrujący użycie mechanizmu notyfikacji o zmianach w systemie plików znajduje się w przykładowym projekcie dostarczonym wraz z niniejszym zadaniem.

Indekser - tryby pracy

Domyślnym trybem pracy indeksera jest monitorowanie katalogów, których zawartość została dodana do indeksu, i aktualizacja tego indeksu (zgodnie z regułami podanymi w poprzedniej sekcji). Informacja o tym, które katalogi podlegają indeksowaniu, może być również zapisana wewnątrz indeksu (np. w formie osobnych dokumentów ze specjalnym polem).

Indekser rozpoczyna pracę w domyślnym trybie, jeżeli przy jego uruchamianiu nie podane zostały żadne argumenty w linii poleceń. Powinien wtedy działać w nieskończonej pętli, w oczekiwaniu na zdarzenia powiadamiające o zmianach w systemie plików. Naciśnięcie kombinacji klawiszy Ctrl+C powinno powodować zakończenie pracy indeksera.

Indekser może być również uruchomiony z dodatkowymi argumentami podanymi w linii poleceń. Ich znaczenie podane jest w poniższej tabeli:

Argument	Znaczenie
`--purge`	powoduje usunięcie wszystkich dokumentów z indeksu i kończy pracę indeksera

`--add <dir>`	powoduje dodanie katalogu, którego ścieżka jest podana w argumencie `<dir>` do zbioru indeksowanych (i monitorowanych) katalogów oraz przeprowadza proces indeksowania plików znajdujących się w tym poddrzewie katalogów; ścieżka podana jako argument `<dir>` może być bezwzględna lub względna (wobec bieżącego katalogu); należy założyć, że poddrzewa katalogów dodawane do indeksu będą parami rozłączne (tj. nigdy nie nastąpi próba dodania katalogu, który znajduje się w poddrzewie już podlegającym indeksowaniu); po wykonaniu operacji dodania katalogu indekser kończy pracę
`--rm <dir>`	powoduje usunięcie katalogu podanego w argumencie `<dir>` ze zbioru indeksowanych katalogów oraz usunięcie wszystkich dokumentów w indeksie odpowiadających plikom znajdującym się tym poddrzewie katalogów; następnie kończy pracę indeksera
`--reindex`	powoduje usunięcie wszystkich dokumentów odpowiadających plikom z indeksu i wykonanie ponownego indeksowania zawartości wszystkich katalogów dodanych wcześniej do indeksu (za pomocą opcji `--add`); następnie kończy pracę indeksera
`--list`	powoduje wypisanie listy katalogów znajdujących się w indeksie (dodanych za pomocą wywołania z podaną opcją `--add`); kanoniczne ścieżki do katalogów powinny być wypisane w osobnych liniach na standardowe wyjście programu (w dowolnej kolejności); po wypisaniu listy katalogów indekser kończy pracę

Można założyć, że indekser nigdy nie zostanie uruchomiony z więcej niż jednym z powyższych argumentów.

Wyszukiwarka

Rolą programu-wyszukiwarki jest zapewnienie tekstowego interfejsu (CLI) służącego do zadawania zapytań do istniejącego indeksu. Wyszukiwarka powinna umożliwiać pracę interaktywną i działać w nieskończonej pętli złożonej z 3 kroków: wczytywania polecenia od użytkownika, wykonywania polecenia i drukowania wyników (jest to klasyczny interpreter typu REPL; Read-Eval-Print-Loop). Pętla powinna być przerwana po naciśnięciu przez użytkownika kombinacji klawiszy Ctrl+C. Wyszukiwarka powinna wyświetlać znak zgłoszenia >, gdy jest gotowa do przyjęcia kolejnego polecenia do wykonania.

Do obsługi zapytań przydatna będzie klasa `IndexSearcher` z biblioteki Lucene. Jak zostało już wspomniane (w sekcji Indeksowanie dokumentów), wyszukiwarka powinna umożliwiać zadawanie zapytań o pojedynczy term, frazę, czyli ciąg termów, oraz zapytań rozmytych (w których dopasowanie

do wzorca następuje również w przypadku, gdy termy są "podobne" - miarą podobieństwa jest tzw. odległość edycyjna, czyli odległość Levenshteina). Tym trzem kategoriom zapytań odpowiadają kolejno klasy `TermQuery`, `PhraseQuery` oraz `FuzzyQuery` z biblioteki Lucene. Przykłady użycia modułu zapytań Lucene można znaleźć m.in. w repozytorium projektu lub tutaj, jak również w dostarczonym wraz z niniejszym zadaniem projekcie.

Przy dopasowywaniu tekstu do wzorca w zapytaniach wielkość liter nie powinna mieć znaczenia.

Polecenia do wyszukiwarki podawane są przez standardowe wejście. Wysłanie polecenia do wykonania następuje po naciśnięciu klawisza Enter. Polecenia dzielimy na 2 kategorie: polecenia sterujące, które rozpoczynają się od znaku %, oraz zapytania. Wyróżniamy następujące polecenia sterujące zachowaniem wyszukiwarki:

Polecenie	Znaczenie
`%lang en/pl`	powoduje ustawienie wybranego języka (polskiego albo angielskiego) używanego przy zapytaniach - może mieć on wpływ na to, w jakich polach dokumentu ma być przeprowadzane wyszukiwanie (tzn. w polach zawierających termy w j. polskim albo w j. angielskim); domyślnie wybrany jest język angielski
`%details on/off`	powoduje włączenie lub wyłączenie wyświetlania kontekstu wystąpienia wyszukiwanego termu lub frazy w tekście; jeżeli wyświetlanie kontekstu jest wyłączone to w wyniki wyszukiwania powinny zawierać jedynie liczbę i listę ścieżek do plików, w których fraza została odnaleziona; w przeciwnym razie - dla każdego pliku z tej listy powinny być dodatkowo podane fragmenty indeksowanego tekstu zawierające wyszukiwany term/frazę; domyślnie wyświetlanie kontekstu jest wyłączone
`%limit <n>`	powoduje ograniczenie liczby wyników wyszukiwania do co najwyżej `n` dokumentów (plików), gdzie `n` jest nieujemną liczbą całkowitą; jeżeli `n = 0` to limit powinien być ustawiony na wartość `Integer.MAX_INT`; domyślna wartość to `n = 0`
`%color on/off`	powoduje włączenie lub wyłączenie wyróżniania ("podświetlania") znalezionych termów/fraz w wyświetlanym kontekście tekstu; ustawienie to ma widoczny efekt tylko, w przypadku, gdy włączone zostało wyświetlanie kontekstu wystąpień za pomocą polecenia `%details`
`%term`	powoduje wybranie trybu wyszukiwania pojedynczego termu - kolejne zapytania wprowadzane przez użytkownika będą interpretowane jako polecenie wyszukiwania przy użyciu klasy `TermQuery`; ten tryb

	wyszukiwania jest wybrany domyślnie
`%phrase`	powoduje wybranie trybu wyszukiwania frazy, czyli ciągu termów - przy kolejnych wyszukiwaniach powinien zostać użyty obiekt klasy `PhraseQuery` skonstruowany na podstawie ciągu wyrazów oddzielonych spacjami podanych przez użytkownika
`%fuzzy`	powoduje wybranie trybu wyszukiwania rozmytego - w kolejnych wyszukiwaniach powinien zostać użyty obiekt klasy `FuzzyQuery`

Wykonanie polecenia sterującego nie powinno powodować wypisania komunikatów (poza znakiem gotowości na przyjęcie kolejnego polecenia).

Efektem wykonania zapytania powinno być wypisane wyniku w następującym formacie:

```
File count: <n>
<path_1>
<path_2>
...
<path_n>
```

gdzie n jest liczbą dokumentów, w których znaleziony został podany term/fraza, a `<path_1>`, `<path_2>`, `<path_n>` to lista ścieżek do odpowiadających tym dokumentom plików (ścieżki powinny być zapisane w kanonicznej postaci w kolejnych liniach wyniku). O kolejności wyników powinna decydować pozycja w rankingu BM25 (BestMatch 25) - jest to domyślna funkcja rankingu używana przez Lucene.

W przypadku, gdy włączone jest wyświetlanie kontekstu, lista rezultatów powinna być rozszerzona o fragmenty dokumentu, w których występuje wyszukiwany term/fraza. Powinna ona mieć następującą postać:

```
File count: <n>
<path_1>:
<context_1>
<path_2>:
<context_2>
...
<path_n>:
```

```
<context_n>
```

Do skonstruowania napisu przedstawiającego kontekst wystąpienia można skorzystać z wbudowanego w bibliotekę Lucene mechanizmu oznaczania wystąpień: klasy `Highlighter` (przykład użycia), klasy `FastVectorHighlighter`, bądź klasy `UnifiedHighlighter` (przykład użycia). W zależności od tego, który moduł oznaczania zostanie użyty w rozwiązaniu, zawartość kontekstu może się istotnie różnić.

W przypadku, gdy uaktywnione zostało wyróżnianie znalezionych fraz za pomocą koloru (polecenie sterujące `%color on`) wystąpienie termu/frazy powinno być otoczone sekwencjami kontrolnymi ANSI tak, by oznaczone były kolorem czerwonym (oznacza to, że przed wyróżniany term należy wstawić napis **"`\033[31m`"** - zob. również artykuł na temat sekwencji ANSI). W tym celu należy zastąpić formatter (np. `SimpleHTMLFormatter`) używany przez Lucene przez własną implementację. By nie posługiwać się wprost surowymi sekwencjami ANSI oraz po to, by zapewnić lepszą przenośność tego rozwiązania (np. współpracę z klasyczną konsolą w systemie Windows) można skorzystać z biblioteki Jansi lub Jline.

Dla tego dokumentu PDF wprowadzonego do indeksu, interakcja użytkownika z wyszukiwarką mogłaby wyglądać następująco (do wyświetlania kontekstu wystąpienia użyta została klasa `UnifiedHighlighter`):

```
> %lang pl
> %details on
> %color on
> %term
> blabaliza
Files count: 1
/home/testuser/Documents/rblarba-pl.pdf:
Wykorzystanie teorii Fifaka daje wreszcie możliwość
efektywnego wykonania blabalizy numerycznej. ... Podejście wprost
Najprostszym sposobem wykonania blabalizy jest siłowe przeszukanie całej przestrzeni roz-
wiązań. ... W literaturze można znaleźć kilka prób opracowania heurystyk dla problemu blabalizy
(por. ... Korzystając z heurystyk daje się z pewnym trudem dokonać blabalizy w prze-
strzeni o np. 500 fetorach bazowych. ... Zasadnicza różnica w stosunku do innych metod blabalizy po-
lega na tym, że przedstawienie
> %fuzzy
> rower
Files count: 1
/home/testuser/Documents/rblarba-pl.pdf:
Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki
Robert Blarbarucki
Nr albumu: 1337
Implementacja blabalizatora
różnicowego z wykorzystaniem
teorii fetorów ?-?
Praca magisterska
na kierunku INFORMATYKA
Praca wykonana pod kierunkiem
dr. hab. ... tłum. z chińskiego Robert Blarbarucki
>
```

Wymaganie dodatkowe (funkcjonalność do zaimplementowania dla chętnych)

Przydatną funkcją, w którą można wyposażyć wyszukiwarkę jest autouzupełnianie zapytań. Naciśnięcie klawisza tabulacji podczas wprowadzania tekstu zapytania mogłoby powodować wyświetlenie listy podpowiedzi - termów pasujących do wprowadzonego przez użytkownika prefiksu. Lucene posiada odpowiedni moduł do obsługi scenariuszy tego typu. Zawiera on m.in. klasę `AnalyzingInfixSuggester` (ew. `FuzzySugester`), w której lista sugestii będzie tworzona (metoda `Lookup.build(Dictionary)`) na podstawie słownika (klasa `LuceneDictionary`) zawierającego termy z indeksu.

Wspomniana już wcześniej biblioteka Jline udostępnia zestaw klas do tworzenia interpreterów CLI, łącznie z obsługą wczytywania poleceń oraz autouzupełnianiem. Dodanie tej funkcjonalności do rozwiązania sprowadza się do zaimplementowania jednej metody wymaganej przez interfejs `Completer`. Przykładowy projekt zawiera klasę demonstrującą użycie tej biblioteki.

Obsługa błędów

W implementacji należy zwrócić uwagę na stabilność rozwiązania. Zaimplementowane programy powinny zapewniać obsługę typowych błędów, w tym:

- błędu spowodowanego brakiem uprawnień do odczytu dla indeksowanego pliku,
- błędów występujących przy parsowaniu plików (np. z powodu uszkodzenia pliku),
- błędów związanych z dostępem do plików tworzących indeks (brak uprawnień do zapisu, założona blokada na indeks, w przypadku, gdy jednocześnie działa kilka procesów indeksera rywalizujących o dostęp do indeksu - por. dokumentacja klasy `IndexWriter`).

Niedopuszczalne jest, by program zakończył się z powodu nieobsłużonego wyjątku typu kontrolowanego. W przypadku wystąpienia błędu programy powinny wypisywać stosowny, krótki komunikat na standardowe wyjście diagnostyczne (`System.err`; można również zapisać pełną informację o wyjątku do pliku z logiem, ale jest to opcjonalne) i, jeśli to możliwe, kontynuować działanie (np. w przypadku braku uprawnień do odczytu jednego z plików w indeksowanym katalogu, proces indeksera może pominąć ten konkretny plik i wykonać próbę indeksowania pozostałych plików). Natomiast jeżeli błąd, który wystąpił jest krytyczny i uniemożliwia dalszą pracę (brak dostępu do indeksu) to proces powinien zakończyć się z niezerowym kodem błędu.

Projekt przykładowy

Projekt dostarczony wraz z niniejszym zadaniem zawiera kilka programów ilustrujących możliwości bibliotek wymienionych w zadaniu. W projekcie znajduje się plik konfiguracyjny dla narzędzia Maven ze wszystkimi potrzebnymi zależnościami - plik ten można zapożyczyć do własnego rozwiązania tego zadania. Projekt Maven może zostać zaimportowany do graficznego środowiska deweloperskiego

(instrukcje importu dla popularnych IDE można znaleźć tutaj - dla środowiska Intellij, oraz tutaj - dla Eclipse). Istnieje również możliwość zbudowania i uruchomienia projektu z poziomu konsoli (np. na maszynie `students`) - w tym celu w głównym katalogu projektu (zawierającym plik `pom.xml`) należy uruchomić polecenie

```
$ mvn compile exec:java -Dexec.mainClass=pl.edu.mimuw.kd209238.example.JLineExample
```

Spowoduje to pobranie wszystkich zależności projektu (ten krok może potrwać dłuższą chwilę, ale musi być wykonany tylko raz), przeprowadzi proces kompilacji plików źródłowych w projekcie i na koniec uruchomi program `JlineExample` z pakietu `pl.edu.mimuw.kd209238.example`. Analogicznie postępujemy w przypadku uruchamiania pozostałych programów w projekcie.

W projekcie znajduje się niewielki zestaw testów jednostkowych, które można uruchomić za pomocą polecenia

```
$ mvn compile test
```

Uwagi

Indekser oraz wyszukiwarka będą dysponować co najmniej 256 MB pamięci przeznaczonej na stertę programu. Można przyjąć, że każdy indeksowany plik oddzielnie zmieści się w dostępnej pamięci, jednak przetwarzanych dokumentów będzie na tyle dużo, że zawartość ich wszystkich *nie* może być jednocześnie przechowana w pamięci programu.

Zagadnienia współbieżności nie są omawiane w ramach przedmiotu Programowanie Obiektowe i znajdują się poza zakresem niniejszego zadania. Wprawdzie specyfikacja wymaga, by procesy indeksera i wyszukiwarki mogły działać równolegle, co wiąże się ze współbieżnym dostępem do indeksu, jednak taki scenariusz został przewidziany przez twórców Lucene i biblioteka zapewnia prawidłową obsługę tego przypadku przez klasy `IndexReader` oraz `IndexWriter`. Dla uproszczenia można przyjąć dodatkowe założenie, że indeks nie będzie ulegał zmianie przez cały czas wykonywania zapytania (czyli wykonanie zapytania i aktualizacja indeksu nigdy nie wydarzą się w tym samym momencie). Zarówno wyszukiwarka jak i indekser mogą być programami jednowątkowymi.

W projekcie można, za zgodą prowadzącego grupę laboratoryjną, korzystać również z bibliotek i narzędzi innych niż te wskazane powyżej, z wyjątkiem jednak takich, które posiadają już

zaimplementowany duży podzbiór wymaganych funkcjonalności (jak Apache Solr lub Elasticsearch) i w trywialny sposób rozwiązywałyby to zadanie.

Decyzja o tym, jak ma wyglądać podział na konkretne klasy, powiązania między nimi, jakie sygnatury będą mieć metody, jak będzie wyglądał podział na pakiety należy w całości do autora rozwiązania. Wszystkie te elementy będą podlegać ocenie.

Wszelkie nieścisłości w niniejszej specyfikacji będą rozstrzygane na korzyść osoby implementującej rozwiązanie.