Kusztelak Paulina
Okołowicz Bernadeta
Bioinformatyka, II rok
Paradygmaty Programowania

Wyrażenia regularne i transformacje przy użyciu reguł

Koncepcję wyrażeń regularnych skonstruował amerykański matematyk Stephen Cole Kleene w latach 50. XX wieku. Do powszechnego użytku wyrażenia te weszły natomiast w 1968 roku. Wykorzystywano je w dwóch celach – dopasowywanie wzorców w edytorze tekstu i analiza leksykalna w kompilatorze.

Wraz z rozwojem wyrażenia regularne zaczęły być użytkowane przez szeroką gamę programów, a obecnie wykorzystuje się w językach programowania i zaawansowanych edytorach tekstu. Wyrażenia regularne, tzw. regex są także częścią standardowych bibliotek różnych programów, takich jak Python czy Java.

Czym zatem są wyrażenia regularne? Skorzystajmy najpierw ze standardowej definicji wyrażeń regularnych podanej na stronie Microsoft: „Wyrażenie regularne to wzorzec, który aparat wyrażeń regularnych próbuje dopasować w tekście wejściowym. Wzorzec składa się z co najmniej jednego literału znakowego, operatora lub konstrukcji”.

W uproszczeniu zatem możemy powiedzieć, że jest to wzorzec, który opisuje określony tekst do wyszukania. Warto tutaj jednak wspomnień, że z konstrukcją wyrażeń regularnych mamy do czynienia nie tylko w programach, ale także w życiu codziennym, m.in na podobniej zasadzie tworzone są różnorakie numery identyfikacyjne, takie jak rejestracja samochodu, numer PESEL, czy też NIP lub REGON firmy.

Jak zatem wygląda taka składnia wyrażenia regularnego? Wszystko zależy od obranego przez nas celu. Czego dokładnie chcemy wyszukać? Co chcemy stworzyć? Odpowiadając na to pytanie posłużymy się skonstruowanym przez nas przykładem, opisując każdy składnik wyrażenia.

Jego celem jest stworzenia danej domenowej w postaci loginu, który będzie składał się będzie z: dwóch ostatnich cyfr z kolumny klucz, 3 pierwszych liter z kolumny imię oraz 3 pierwszych liter z kolumny nazwisko. Dana powinna zatem

wyglądać: BerOko23, PauKus43. Zdecydowaliśmy się na tworzenie loginów z 3 liter nazwisk i imion, ponieważ większość jest stworzona z większej ilości liter.

Plik, na którego podstawie zostało oparte zadanie ma postać:

„Klucz	Nazwisko	Imię
1020	Gębka	Agnieszka
1050	Skwarek	Gustaw
1062	Górska	Kinga
1142	Wolny	Michał
1156	Piasecki	Piotr
1193	Czerw	Milena
1206	Szewczyk	Damian”

W kolumnie “Klucz” znajdują się unikalne kody identyfikacyjne danych osób. Powinny składać się z minimum 2 cyfr, bez maksymalnego ograniczenia.
W Kolumnie “Imię” znajduje się imię lub imiona danej osoby. Do powstania loginu pobierać będziemy wyłącznie litery z pierwszego, podanego imienia.
W kolumnie “Nazwisko” będzie znajdować się nazwisko lub nazwiska danej osoby. Do powstania loginu pobierać będziemy wyłącznie litery z pierwszego, podanego nazwiska. Przypadek nazwisk niepolskich np. De Franze zostanie omówiony w dalszej części sprawozdania.
Kolumny oddzielone są od siebie tabulatorem.

Tworzenie loginu zostanie omówione w postaci poniższego schematu:

1. Pobranie dwóch ostatnich cyfr z kolumny klucz.

Jak wspomniano powyżej, klucz jest numerem w całości unikalnym, będący pierwszą kolumną pliku (dlatego zostanie wykorzystany znak “^”). W przykładowym pliku składa się on z 4 cyfr, jednak wychodzimy z założenia, że ich zakres może się zawierać w przedziale od 2 do nieskończoności (w tym celu zostanie wykorzystany zapis w postaci [^\t] - nie tabulator - umożliwiający zaznaczenie dowolnej ilość znaków, w naszym przypadku cyfr poprzedzających dwie ostatnie cyfry klucza). Do stworzenia loginu potrzebne nam są dwie ostatnie cyfry (w związku z tym wykorzystamy wyrażenie [0-9] oznaczający zakres liczb od 0 do 9, oraz wyrażenie {2} oznaczający dwukrotne powtórzenie poprzedzającego wyrażenia), które grupujemy przy pomocy znaków „()”. Wykorzystanie dwóch ostatnich cyfr umożliwia powstanie unikatowych loginów, ponieważ mogą zdarzyć się sytuacje, w których same 3 litery imienia i nazwiska nie będą wystarczające. Login „AnnKow” mógłby dotyczyć osób o danych: „Anna Kowalska” lub „Anna Kowal”.

Zatem schemat wyrażenia powinien mieć postać:

^[^\t]*([0-9]{2})

2. Pobranie 3 pierwszych liter pierwszego imienia z kolumny “Imię”.

W przypadku osób, które posiadają dwa imiona, do powstania loginu będziemy wykorzystywać tylko pierwsze podane. Imię rozpoczyna się od wielkiej litery (dlatego posługujemy się wyrażeniem [A-Z], który oznacza zakres wielkich liter od A do Z), następne dwie małe litery pobierane są według tego schematu ([a-z]{2}). Trzy pierwsze litery będą pobierane w postaci jednej grupy, dlatego wykorzystany będzie znak “()”. Pozostałe litery nie będą wykorzystywane do powstania loginu - dlatego po stworzeniu grupy liter pojawia się “.” oznaczająca dowolny znak. “*” oznacza powtórzenie poprzedzającego znaku 0 lub więcej razy.

Całe wyrażenie będzie wyglądać w następujący sposób:

([A-Z][a-z]{2}).*

3. Pobranie 3 pierwszych liter z kolumny „Nazwisko”.

Schemat pobrania 3 pierwszych liter nazwiska jest dokładnie taki, jak w przypadku imienia. Jeśli w kolumnie będzie wpisane nazwisko dwuczłonowe będziemy wykorzystywać tylko litery z pierwszej części. Nie chcemy, aby były sytuacje, w których niektóre osoby posiadają dłuższy login od innych. Mogłoby to wprowadzić zamieszanie lub zakłopotanie, dodatkowo przechowywanie większych danych zajmuje więcej miejsca w pamięci.

Sumując powyższy schemat, końcowa postać wyrażenia regularnego będzie miała postać:

^[^\t]*([0-9]{2})\t([A-Z][a-z]{2}).*?\t([A-Z][a-z]{2}).*$

Plik rozpoczyna się od kolumny „klucz” dlatego znajduje się on na samym początku. Pomiędzy kolumnami (punktami schematu, który przedstawiliśmy) znajdują się tabulatury „\t”. Wykorzystałyśmy również znaki specjalne - „*” oznacza powtórzenie wyrażenia poprzedzającego 0 lub więcej razy, w połączeniu z „.” oznacza dowolny znak nieskończenie wiele razy - może to być bardzo zachłanne i zgubne, ponieważ każdym znakiem jest również tabulator, który wykorzystujemy do rozdzielenia kolumn, czyli danych które chcemy wykorzystać. Dodając „?” po „*” ograniczamy zachłanność tego znaku specjalnego, co oznacza, że zakończy na następnym innym rodzaju znaku, a w naszym przypadku będzie to tabulator. Po wydobyciu wszystkich interesujących nas części danych, nie ma dla nas znaczenia co znajduje się w dalszej części wierszy, dlatego nie ograniczamy „*” żadnym innym znakiem.

Powyższe wyrażenia zazwyczaj działa, ale pojawiają się pewne kwestie warte dodatkowych przemyśleń.

1. Występowanie polskich znaków w imieninach i nazwiskach

Niestety nasz kod (w takiej postaci) nie obejmuje imion i nazwisk, w których występują polskie znaki. Pierwszym pomysłem, który wykorzystałyśmy, aby rozwiązać problem jest zwiększenie zakresu liter, czyli zmiana [a-z] na [a-ź]. Jest to rozwiązanie, które działa, ale nie jest ono najlepsze, ponieważ pomiędzy „z" a „ź" znajdują się również litery różnych alfabetów oraz mogą wystąpić znaki specjalne.

Drugim rozwiązaniem, które zawęża zakres powstały w poprzednim pomyśle jest dopisanie polskich znaków do podstawowego zakresu. Wtedy wyrażenie miałoby postać: [a-ząćęłńóśżź]. Będzie ono poprawne funkcyjnie, ale pisanie takiego wyrażenia może powodować błędy z powodu ludzkiego, czy nie zapomnimy o którymś polskim znaku. Dodatkowo taki kod wyglada bardzo skomplikowanie, w szczególności przy powtórzeniach tego wyrażenia.

W kolejnym pomyśle do problemu podchodzimy w trochę inny sposób. Jeśli przeszkadzają nam polskie znaki to można po prostu je zmienić na litery alfabetu łacińskiego. Można tutaj korzystać z funkcji transliteracji w „sed". Wtedy tworzymy komendę:

sed „y/ąćęłńóśżź/acelnoszz/g"

Można pomyśleć, że problem został rozwiązany i po części jest to prawda, jednak zmieniamy plik wejściowy, a nie zawsze jest to dobre rozwiązanie. Posiadając plik przejściowy, w którym dokonujemy tych zmian, nie niszcząc danych znajdujących się w wejściowym pliku, byłoby to najlepsze rozwiązanie.

Ostatnim pomysłem, który mamy jest porzucenie pierwotnej idei, czyli nie wyszukiwanie liter za pomocą wyrażenia [a-z], a w sposób, który wykorzystałyśmy przy cyfrach. Wtedy grupę trzech pierwszych liter stworzymy jako wyrażenie:

([^\t]{3}).*?

Przy wykorzystaniu tego sposobu nie musimy się martwić o to czy litery wykorzystane do wpisania imienia lub nazwiska, będą znajdować się w zakresie od alfabetu łacińskiego. Jest to również bardziej uniwersalne, ponieważ w naszych rozmyślaniach bierzemy pod uwagę tylko polskie imiona i nazwiska, a nie mamy sposobu na np. nazwiska niemieckie.

2. Nazwiska niepolskie, które złożone są z dwóch lub więcej członów (oddzielonych spacjami):

W takim przypadku bierzemy pod uwagę sytuację, w której nazwisko składa się z dwóch lub więcej członów, oddzielonych spacjami, np. De Frans, Dessa La

Luna. W celu rozwiązania następującego problemu zdecydowałyśmy stworzyć wyrażanie składające się z części odpowiadającej za dopasowanie pierwszej części nazwiska. Zaczyna się od dużej litery (od A do Z lub od Ą do Ź) i dopasowuje 3 pierwsze litery nazwiska lub dwie pierwsze litery, spacje i pierwszą literę występującą w drugim członie nazwiska. To oznacza, że będzie dopasowywać "De" lub "La", a następny człon dopasowuje kolejną literę występującą w następnym członie nazwiska. Zdecydowałyśmy się tutaj na wypisanie wszystkich liter z alfabetu polskiego, aby nie doszło do sytuacji, w której dana litera nie znajduje się w zakresie (przy skorzystaniu z zakresu [ą-ż] litera „ó" nie jest wyszukiwana).

([A-ZĄĆĘŁŃÓŚŻŹ][a-ząćęłńóśżź]{2}|[A-ZĄĆĘŁŃÓŚŻŹ][a-ząćęłńóśżź] [a-ząćęłńóśżź]).*?

Aby usunąć spacje z loginu postanowiłyśmy wykorzystać sed'a i za pomocą poniższego polecenia zamieniamy (wykorzystując funkcje „s") nasze niepożądane znaki (spacje) na "nie spację" (usuwa ją), tak aby uzyskać poprawny login. Polecenie ma postać:

sed -e „s/ //g"

Po rozmyślaniach zmieniłyśmy postać naszego wyrażenia na:

^[^\t]*([0-9]{2})\t([A-ZĄĆĘŁŃÓŚŻŹ][a-ząćęłńóśżź]{2}|[A-ZĄĆĘŁŃÓŚŻŹ][a-ząćęłńóśżź] [a-ząćęłńóśżź]).*?\t([^\t]{3}).*$

Powyższe przemyślenia dotyczą tylko jednej części procesu powstania loginu, tej trudniejszej, dotyczącej wybrania wszystkich wierszy, w których znajdują się dane osobowe. Teraz, aby powstał login wykorzystujemy w wybranym programie funkcje „sed". Wykorzystujemy regułę skryptu „s", która polega na zastępowaniu danych. Działamy na naszym ostatecznym wyrażeniu, a następnie tworzymy login. Tworzymy go z grup, które tworzyłyśmy w poprzednich częściach. Pierwsza grupą są dwie ostatnie cyfry klucza, grupa drugą trzy pierwsze litery imienia, a grupa trzecia trzy pierwsze litery nazwiska. Teraz wystarczy ułożyć nasze grupy w odpowiedniej kolejności. Na końcu dodajemy jeszcze „g" co oznacza global, czyli aby nasze polecenie zadziałało na całym pliku, a nie tylko w pierwszym wierszu. Polecenie będzie wyglądać następująco:

sed -e „s/^[^\t]*([0-9]{2})\t([A-ZĄ-Ż][a-zą-ż]{2}|[A-Z][a-z] [a-zą-ż]).*?

\t([^\t]{3}).*$/\3\2\1/g"

Kończąc omawianie przykładu chciałybyśmy zwrócić uwagę, że ze względu na ogromne możliwości użycia wyrażeń regularnych, może on być rozwiązany na wiele różnych sposobów, które będą równie poprawne. Wszystko zależy od naszych danych wejściowych, co dokładnie chcemy zrobić oraz jaki wynik uzyskać.

Reasumując, wyrażenia regularne wprowadzają użytkowników na zupełnie nowy poziom programowania, tworzenia schematów i wyszukiwania treści. Stosowanie ich możne znacznie ułatwić pracę w wielu obszarach wymagających pracy z danymi. Jako przyszłe programistki nie mamy wątpliwości, że znajomość wyrażeń regularnych wielokrotnie pomoże w przyszłej pracy.