# Regex - trudniejszy niż myślisz!

Mikołaj Pietrek

## 1 Zamiast wstępu...

Ten krótki poradnik powstał w mocno ograniczonym czasie, proszę nie oczekiwać teorii wyrażeń regularnych. Nie zaprzeczam, jest to oczywiście bardzo, bardzo proste, ale mimo wszystko pomijamy. Podstawą do stworzenia PDFa był więc materiał wymagany na kolokwium. Zamiast traktować regex jako opis języka (czym jest w rzeczywistości), przeanalizujemy tylko praktyczne zastosowania. Skoro nie zniechęcił Cię anty-clickbaitowy tytuł, zaczynamy.

## 2 Po co regex w informatyce?

Prawdopodobnie każdy, kto opanował obsługę edytora tekstu, wie jak działają symbole wieloznaczne (np. *). Dzięki nim możemy napisać `L*e*` i jednym wyrażeniem znaleźć `Lemiesz`, `Lee` oraz `Lenin`. Łatwo można jednak zauważyć ogromne ograniczenia. Przykładowo w zależności od ustawienia case-sensitive wyraz `lewarek` może zostać znaleziony lub nie. Nie wyeliminujemy też w prosty sposób nazwy `Lenino`. Z pomocą przychodzi regex.

## 3 Podstawy wyrażeń regularnych

Najważniejszą cechą regexa w porównaniu z trywialnymi symbolami wieloznacznymi jest to, że osobno określamy rodzaj znaku, a osobno ilość wystąpień. W większości implementacji prostego wyszukiwania gwiazdka (`*`) oznacza dowolną ilość wystąpień dowolnego znaku, a znak zapytania (`?`) to najwyżej jeden dowolny znak (również spacja). Natomiast w regexie najpierw podajemy, jakich znaków szukamy, a dopiero potem ilość wystąpień. Dowolny znak to symbol kropki (`.`), więc wyrażenie „dowolna ilość dowolnych znaków" przybierze postać `.*` w regexie. Analogicznie, wyrażenie `.?` oznacza od zera do jednego wystąpienia dowolnego znaku. Rodzaj znaku (lewa strona) musi być podany zawsze, ilość (prawa strona) może być domyślnie przyjmowana jako 1.

## 4 Jakie znaki?

Zajmijmy się lewą stroną, czyli zakresem wyszukiwanych znaków.

### 4.1 Klasy zbiorów

Najważniejsze są klasy, definiujemy je za pomocą nawiasów kwadratowych. Przykładowo `[ab]*` to dowolna ilość wystąpień znaków `a` oraz `b` w dowolnej kolejności. Analogicznie definiujemy zakresy, wyrażenie `[a-z]*` znajdzie dowolną ilość wystąpień znaków od `a` do `z`, czyli dowolne słowo złożone z małych liter alfabetu łacińskiego.

### 4.2 Sumy zbiorów

Można łączyć całe zakresy, `[a-zA-Z]*` to dowolne słowo z małych i dużych liter w dowolnej kolejności. Muszą natomiast być one w porządku rosnącym, np. `a-b` przed `A-B`. Brak przecinka jest celowy! Oczywiście, znaki mogą się powtarzać, słowo `aBabb` spełnia podane wyrażenie.

### 4.3 Negacja

Jeśli pierwszym znakiem w klasie jest kareta (∧), nastąpi negacja całej klasy. `[∧ab]*` to dowolna ilość wystąpień dowolnych znaków oprócz `a` oraz `b`.

### 4.4 Część wspólna

Poprzednio zostało zaznaczone, że przy sumowaniu zakresów nie używamy przecinka. Możemy jednak, jeśli jest taka potrzeba, wykorzystać podwójny ampersand (`&&`) do stworzenia iloczynu zbiorów zamiast sumy. Tutaj przykład sztuczny, ale przynajmniej prosty. Klasa `[0-6&&4-9]` wyznacza dowolną cyfrę od 4 do 6.

### 4.5 Różnica

Poprzez część wspólną możemy zdefiniować różnicę. Z prostych praw logicznych wynika, że różnicę $A \setminus B$ można równoważnie zapisać jako $A \cup B'$. Przepiszmy to na regex. Klasa `[0-9&&[∧37]]` zawiera dowolną cyfrę z wyjątkiem 3 i 7. Gdybyśmy wykorzystywali wyłącznie sumę, wynikiem byłoby `[0-24-68-9]`, czyli aż trzy zakresy. Dodatkowo czytelność takiego sposobu pozostawiam bez komentarza.

### 4.6 Klasy predefiniowane

Regex pozwala na uproszczenia zapisu. Przykładowo zamiast długiego wyrażenia `[a-zA-Z0-9]` można wykorzystać `\w`, czyli klasę predefiniowaną dla wszystkich cyfr i liter alfabetu łacińskiego.

## 5 Ile wystąpień?

Podobnie jak przy zakresie znaków używałem najprostszej możliwej ilości wystąpień (dowolnej), tak teraz dla uproszczenia opisu wystąpień, to znak zawsze będzie dowolny.

### 5.1 Proste kwantyfikatory

Znamy już dowolną ilość wystąpień (`*`) oraz 0 lub 1 raz (`?`), a trzeci najważniejszy operator jest równie prosty. Plus (`+`) oznacza co najmniej jedno wystąpienie. Dokładną ilość podajemy w nawiasach klamrowych: `{12}`

### 5.2 Zakresy

Możemy też definiować zakresy, będą to dwie wartości oddzielone przecinkiem w nawiasach klamrowych. Wyrażenie `.{21,36}` to po prostu od 21 do 36 wystąpień dowolnego znaku. Przykładowo: dla `''uwielbiam wzorce projektowe''` zadziała, natomiast `''nienawidzę regexa''` nie zostanie dopasowane. Jeśli nie podamy drugiej wartości, w składni regexa wyrażenie `{a,}` oznacza co najmniej $a$ razy.

### 5.3 Tryby kwantyfikatorów

Domyślny jest tryb zachłanny, który dopasuje pierwszy możliwie największy fragment tekstu. Odwrotnie działa tryb wstrzemięźliwy, który szuka pierwszego możliwie najmniejszego pasującego tekstu. Aktywujemy go poprzez dopisanie po ilości wystąpień znaku `?`. Natomiast kwantyfikator zaborczy sprawdza zgodność z całym tekstem (wszystko albo nic), aktywujemy go przez dopisanie `+`.

# 6 Wyjście

Znak wyjścia brzmi nieciekawie[1], jednak to prosta sprawa. W poprzednich akapitach używaliśmy nawiasów kwadratowych do definiowania klas, a klamrowymi określaliśmy zakres. Oczywiste jest więc, że skoro wykorzystujemy je jako symbole specjalne, nie możemy ich użyć w wyszukiwaniu. A gdybyśmy chcieli? Wtedy należy poprzedzić znakiem wyjścia, ukośnikiem (\).

# 7 Kolejność wykonywania działań

Od pierwszych: znaki wyjścia, nawiasy okrągłe, kwantyfikatory, konkatenacja, alternatywna.

# 8 Grupowanie

Dowolny fragment wyrażenia możemy objąć nawiasami okrągłymi, umożliwi to odwołania do tej części.

## 8.1 Odwołania wsteczne

Mając co najmniej jedną grupę w nawiasie, możemy się do niej odwołać poprzez znak wyjścia (\) i odpowiedni numer grupy (1-9). Załóżmy, że chcemy znaleźć wszystkie słowa postaci $aXbX$, gdzie X jest dowolnym znakiem. Możemy więc pierwszy X potraktować jako grupę, a w miejsce drugiego X podstawić odwołanie wsteczne. Przepisując na składnię regex: `a(.)b\1`

## 8.2 Kotwice

Kotwice są specjalnymi operatorami, które pozwalają nam określić gdzie konkretnie w tekście regex ma szukać odpowiednich wyrażeń. Dla nas najważniejsza są znak dolara ($), czyli koniec linii oraz kareta (∧), początek linii. Uwaga, nie ma to nic wspólnego z karetą w klasach, gdzie definiowała ona negację. Przykładowo `∧ab` sprawdzi wystąpienie `ab` na początku linii. Analogicznie `ab$` będzie szukać `ab` na końcu linii.

# 9 Łączenie

Na zakończenie weźmy kilka małych wyrażeń i połączmy je w jedno, stanowczo zbyt skomplikowane:
`(M[a-zA-Z]*)␣[0-9&&[∧37]]+␣\1A{2,4}`
Teraz tłumaczenie na polski. Co ważne, spacja w regexie jest traktowana jak każdy inny znak - wstawiamy ją tylko wtedy, gdy faktycznie tam ma się znaleźć! Natomiast w innych przypadkach składamy bez żadnych łączników, co zresztą zaraz się okaże. Nasze wyrażenie składa się z trzech ciągów oddzielonych spacjami. Pierwsze z nich to dowolne słowo złożone z małych i dużych liter, które zaczyna się na literę M. Druga będzie liczba dowolnej długości, która składa się ze wszystkich cyfr oprócz 3 oraz 7. Natomiast trzecie słowo ma być połączeniem pierwszego znalezionego słowa z dwiema, trzema lub czterema literami A.

Przykładowy ciąg zgodny z wzorcem: `Matma␣111␣MatmaAA`
Przykładowy ciąg niezgodny z wzorcem: `I tak nie zdasz`

# 10 Zestawienie najważniejszych operatorów

A teraz to, na co wszyscy czekali :–) Zezwalam na kopiowanie w celu wiadomym.

[1] Faraon triggered

## Klasy znaków

| | |
|---|---|
| . | dowolny znak |
| [ab] | a lub b |
| [∧ab] | dowolny oprócz a,b |
| [a-t] | od a do t |
| [a-zA-Z] | suma zbiorów |
| [1-9&&3-5] | część wspólna zbiorów |
| [1-9&&[∧47]] | różnica zbiorów |

## Kwantyfikatory

| | |
|---|---|
| ? | 0 lub 1 |
| * | 0 lub więcej |
| + | 1 lub więcej |
| {n} | dokładnie n |
| {n,} | co najmniej n |
| {n,m} | od n do m |

## Tryby kwantyfikatorów

| | |
|---|---|
| | domyślny zachłanny (max) |
| ? | wstrzemięźliwy (min) |
| + | zaborczy (wszystko) |

## Kotwice

| | |
|---|---|
| ∧ | początek linii |
| $ | koniec linii |
| \b | na granicy słowa |
| \B | nie na granicy |
| \A | początek wejścia |
| \G | koniec poprz. dopas. |
| \Z | koniec wejścia |

## Klasy predefiniowane

| | |
|---|---|
| \d | [0-9] |
| \D | [∧\d] |
| \s | [\t\n\f\r] |
| \S | [∧\s] |
| \w | [a-zA-Z0-9] |
| \W | [∧\w] |
| \p{ASCII} | znak ascii |
| \p{Lower} | małe Unicode |
| \p{Upper} | duże Unicode |
| \p{Punct} | interpunkcyjne |

## Dodatkowe operatory

| | |
|---|---|
| \ | znak wyjścia |
| \| | logiczna alternatywa |
| ( ) | grupowanie |
| \n | odwołanie n-tej grupy |

## Flagi

| | |
|---|---|
| (?i) | off case-sensitive |
| (?-i) | on case-sensitive |
| (?i)(?u) | jak wyżej, plus unicode |