6 REGULÁRNE VÝRAZY

V tejto kapitole opisujeme použitie regulárnych výrazov na rôzne úlohy v oblasti vyhľadávania a extrakcie informácií. Regulárne výrazy sú silným nástrojom na spracovanie textov, ako aj iných textových dát (log súbory, výstupy programov a podobne) vo všeobecnosti. Regulárne výrazy, skrátene regex, alebo poslovenčene regexy, sú vhodným nástrojom, ak ho naozaj ovládame. Zle napísané regexy vrátia primnoho nerelevantných, alebo primálo relevantných výsledkov a môžu tiež dosť silne zahltiť výpočtové prostriedky (pamäť a procesor). S regexami je teda dobré narábať opatrne a dôkladne ich testovať na kolekcii, na ktorej budú použité. V tejto kapitole budeme regexy testovať na našej malej kolekcii 6 dokumentov (tab. 1).

Regexy sú dnes už podporované v takmer každom programovacom jazyku. V tejto kapitole budeme písať regulárne výrazy pomocou príkazov v jazyku Java, sú však použiteľné aj všeobecne. Regexy môžno použiť na úlohy predspracovania textov, tokenizácie, segmentácie, extrakcie liniek, definovanie pravidiel sťahovania, extrakciu informácií a ďalšie. V nasledujúcej kapitole prechádzame krok za krokom cez jednotlivé vlastnosti regulárnych výrazov na konkrétnych príkladoch, ktoré sú nakoniec zhrnuté v časti 6.1.1 (tab. 10).

6.1 Úvod do regulárnych výrazov

V projekte irLessons po spustení *vi.analyze.CrawlAndSegmentize* dostaneme jednoducho segmentované texty z HTML dokumentov. Segmentovanie prebieha pomocou regulárnych výrazov, k čomu sa vrátime neskôr. Pre dokument 1, však dostaneme tento text:

URL: http://irlesons.sourceforge.net/data/1.html
Title: Vyhľadávanie informácií

Vyhľadávanie informácií
Informácie o predmete:
- Výučba predmetu na: FIIT STU
- Zabezpečuje: Michal Laclavík , Martin Šeleng
- Pracovisko: UISAV miestnosť 312, 306

Na tomto texte si vysvetlíme základy regulárnych výrazov. Keď chceme napríklad vyhľadať názov dokumentu a URL môžeme hľadať pomocou tohto regexu:

```
Title|URL
```

Tento výraz by nám pri použití príkazu *egrep* v operačnom systéme Unix vrátil celý riadok, ktorý obsahuje slovo *Title* alebo *URL*, avšak v programoch regulárny výraz

vracia iba presne to, čo ním špecifikujeme, teda slovo *Title* alebo *URL* keď naň natrafí v texte. Znak „|" je operátor *alebo* pre regulárne výrazy. Ak teda chceme doplniť regex o vrátenie celého riadku, ktorý obsahuje slovo *Title* alebo *URL*, musíme regex doplniť takto:

```
Title|URL.*
```

Výsledok vidíme na obrázku nižšie. Použili sme ďalšie dva špeciálne znaky „." bodku a „*" hviezdičku. Bodka znamená akýkoľvek znak okrem znaku nového riadka. Hviezdička znamená opakovanie, ktoré vysvetlíme neskôr. Regex vrátil URL s celým riadkom, alebo slovo *Title*. Je to preto, že operátor „|" sa vzťahuje na celý výraz, kým nenarazí na koniec regexu alebo zátvorku.

```
URL: http://irlesons.sourceforge.net/data/1.html
Title: Vyhľadávanie informácií
```

Ak teda chceme správne napísať regulárny výraz musíme použiť zátvorky „()":

```
(Title|URL).*
```

Výsledkom je detekcia celého riadku tak ako sme zamýšľali.

```
URL: http://irlesons.sourceforge.net/data/1.html
Title: Vyhľadávanie informácií
```

Na obrázku však môžeme vidieť, že chceme vytiahnuť obsah URL alebo názvu dokumentu. Nezaujíma nás teda dvojbodka a medzera. Môžeme teda použiť nasledovné rozšírenie regexu a definovanie zátvorky, ktorá obsahuje hľadaný text:

```
(Title|URL): ?(.*)
```

Vidíme, že až teraz máme vyznačený text, ktorý nás zaujíma.

```
URL: http://irlesons.sourceforge.net/data/1.html
Title: Vyhľadávanie informácií
```

V definovanom regulárnom výraze máme niekoľko nových prvkov. Jedným z nich je nová zátvorka, ktorá je zobrazená aj novým farebným odtieňom na obrázku vyššie. Obsah zátvoriek, teda nájdený text, vieme vytiahnuť z jednotlivých regulárnych výrazov. Obsah zátvoriek je vrátený ako premenné očíslované v poradí od 1 zľava. Premenná číslo 1 v našom prípade obsahuje text *Title* alebo *URL*, premenná číslo 2 obsahuje samotnú URL alebo názov dokumentu. Okrem toho premenná s číslom 0 obsahuje celý text, ktorý vrátil regulárny výraz. V rôznych jazykoch sa rôzne pristupuje k týmto premenným, neskôr ukážeme príklad v jazyku Java. Vhodne riešené spracovanie zátvoriek je v jazyku C#, kde možno jednotlivé časti regexu definované zátvorkami priamo v regexe pomenovať a podľa týchto názvov potom k nim pristupovať.

Okrem zátvoriek sme do regulárneho výrazu pridali špeciálny znak „?", ktorý nasleduje za znakom medzery. Rovnaký text by bol vrátený aj bez použitia tohto znaku, avšak jeho použitie za medzerou znamená, že znak medzery sa môže ale nemusí nachádzať za dvojbodkou. Všimnime si tiež že „?" sa vzťahuje iba na jeden znak presne, ako aj „*", teda opačne ako alebo „|", ktoré sa vzťahuje na celý výraz alebo po najbližšiu zátvorku, v ktorej sa nachádza. Ak teda chceme otáznik alebo hviezdičku použiť na sekvenciu znakov alebo regexov, musíme ich dať do zátvorky a za zátvorkou uviesť otáznik.

V ďalšom príklade budeme chcieť vytiahnuť text, ktorý je za hocakým slovom končiacim dvojbodkou. Na definovanie tohto slova použijeme definovanie množiny, ktorá sa definuje pomocou hranatých zátvoriek „[]". Okrem toho sa dajú použiť aj iné množiny, ktoré spomenieme neskôr. Množinou je aj špeciálny znak bodka „.", ktorý definuje množinu ľubovoľných znakov okrem konca riadku. V nasledujúcom regulárnom výraze sme ako množinu definovali písmená s rozsahom od *a* po *z*.

Môžeme pridať aj veľké písmená *A* až *Z*, pričom množina sa definuje vymenovaním prvkov alebo určením rozsahov. V množine sú povolené všetky znaky okrem pomlčky, ktorá určuje rozsah, konca hranatej zátvorky „]“, znak spätného lomítka „\“ a strieška “^”, o ktorých ešte budeme hovoriť. Avšak aj pomlčka môže byť vymenovaná ako znak na začiatku množiny, hneď po úvodnej hranatej zátvorke.

```
[a-zA-Z]: ?(.*)
```

Použitím tohto výrazu sme dostali výsledok ako vidíme na obrázku nižšie:

```
URL: http://irlesons.sourceforge.net/data/1.html
Title: Vyhľadávanie informácií
---------------------------
Vyhľadávanie informácií
Informácie o predmete:
- Výučba predmetu na: FIIT STU
- Zabezpečuje: Michal Laclavík , Martin Šeleng
- Pracovisko: UISAV  miestnosť 312, 306
```

Použitý regex má viacero nedostatkov. Prvým je pokrytie iba jedného znaku zo slova pred dvojbodkou. Druhým problémom je zhoda regulárneho výrazu aj na texte *Informácie o predmete:* za ktorým nenasleduje text. To je spôsobené tým, že použitím hviezdičky za bodkou sme povolili rozsah (opakovanie) ľubovoľného znaku v rozsahu 0 a viac. Ak chceme vrátiť aspoň nejaký text za dvojbodkou, musíme použiť špeciálny znak „+“, ktorý znamená opakovanie 1 a viac znakov. Tento operátor opakovania použijeme aj za množinou aby sme mali pokryté celé slovo pred dvojbodkou a nielen jeden znak:

```
[a-zA-Z]+: ?(.+)
```

Výsledok je o niečo lepší ale stále nedostačujúci.

```
URL: http://irlesons.sourceforge.net/data/1.html
Title: Vyhľadávanie informácií
---------------------------
Vyhľadávanie informácií
Informácie o predmete:
- Výučba predmetu na: FIIT STU
- Zabezpečuje: Michal Laclavík , Martin Šeleng
- Pracovisko: UISAV  miestnosť 312, 306
```

Vidíme že definícia množiny neobsahuje diakritické znaky. V tomto prípade použijeme namiesto pôvodnej množiny množinu všetkých písmen, vrátane diakritických, ktorá je pre regexy v jazyku Java definovaná takto *\p{L}*. Takisto by sme mohli chcieť definovať aby nám vrátilo celý text pred dvojbodkou. V tomto prípade to môžeme zabezpečiť tak, že použijeme začiatočné veľké písmeno, po ktorom nasleduje text. Výsledný regulárny výraz bude:

```
[A-Z][ \p{L}]+: ?(.+)
```

Týmto výrazom sme dostali požadovaný výsledok aj keď pri použití na väčšom texte by pravdepodobne nie vždy fungoval správne. Všimnime si že sme definovali dve množiny, jednu na nájdenie veľkého písmena (nenasleduje + pretože chceme práve jeden znak) a ďalšiu množinu obsahujúcu písmená, vrátane diakritických znakov a medzeru. Znovu by mohol nastať problém, ak by veľké začiatočné písmeno bolo diakritické. Namiesto *[A-Z]* potom možno použiť množinu *\p{Lu}*.

```
URL: http://irlesons.sourceforge.net/data/1.html
Title: Vyhľadávanie informácií
--------------------------
Vyhľadávanie informácií
Informácie o predmete:
- Výučba predmetu na: FIIT STU
- Zabezpečuje: Michal Laclavík , Martin Šeleng
- Pracovisko: UISAV miestnosť 312, 306
```

Podobný výsledok vieme dostať aj definovaním množiny vymenovaním prvkov, ktoré množina nesmie obsahovať. Negatívna množina sa definuje úvodným znakom strieška "^" v množine. V tomto príklade sme teda definovali množinu, ktorá nesmie obsahovať pomlčku (musí byť uvedená na začiatku), dvojbodku a znak nového riadka ktorý sa definuje pomocou spätného lomítka „\":

```
[^-:\n]+: ?(.+)
```

Negatívna množina definuje nie text, ktorý tam nemá byť ale znaky, ktoré sa v nájdenom texte nemajú nachádzať. V prípade vyššie uvedenej negatívnej množiny to znamená, že bude hľadať jeden a viac znakov (definované pomocou plus) idúcich za sebou, ktoré sú iné ako vymenované znaky.

Ďalšie vlastnosti regulárnych výrazov ukážeme na jednoduchom príklade extrakcie skratiek. Ak chceme z textu extrahovať skratky ako *URL*, *FIIT STU* alebo *UISAV*, môžeme definovať regex, ktorý nájde slová písané veľkými písmenami:`[A-Z]+`
Výsledok takéhoto výrazu je však aj nájdenie veľkých písmen hocikde v texte:

```
URL: http://irlesons.sourceforge.net/data/1.html
Title: Vyhľadávanie informácií
--------------------------
Vyhľadávanie informácií
Informácie o predmete:
- Výučba predmetu na: FIIT STU
- Zabezpečuje: Michal Laclavík , Martin Šeleng
- Pracovisko: UISAV miestnosť 312, 306
```

Aby sme našli iba výskyty celých slov, je potrebné definovať koniec a začiatok slova. Toto môžeme urobiť napríklad množinou, ktorá vymenuje znaky, ktoré oddeľujú slová, teda napríklad takáto množina *[\n,;.]+*. Alebo môžeme použiť množinu bielych znakov *\s*. Takto použitý výraz *\s[A-Z]+\s* by však vrátil iba skratku *FIIT* a *UISAV* a nevrátil by *STU*, pretože neobsahuje koniec riadku. Ďalší problém nastáva, ak sa slovo nachádza na začiatku alebo konci dokumentu (niekedy aj na začiatku a konci riadku v závislosti od použitého nastavenia). Problém nastáva preto, že každá definovaná množina musí nájsť nejaký znak. Na začiatku a konci vyhodnocovaného textového reťazca ale žiadny znak nie je. Je tam iba pozícia. Regulárne výrazy používajú niekoľko špeciálnych znakov na definovanie pozícií. Tu si zadefinujeme iba začiatok a koniec riadka/textu, existujú však ďalšie prístupy na zistenie pozície, ako napríklad tzv. lookahead a lookbehind[49]. Znak strieška „^" definuje začiatok textového reťazca (alebo riadku podľa nastavenia) a znak dolár „$"definuje koniec textového reťazca (alebo riadku podľa nastavenia). Keď chceme napríklad nájsť obsah všetkých riadkov, ktoré obsahujú dvojbodku môžeme definovať nasledovný regex *^.*:.*$*

[49] http://www.regular-expressions.info/lookaround.html

Ďalším znakom, ktorý definuje pozíciu, je hranica slova (word boundary). Táto sa napríklad pre egrep definuje pomocou znakov \< a \>, ale v regexoch pre jazyk Java ako \b. Nasledovný regex nám teda vráti všetky skratky:

```
\b\p{Lu}+\b
```

Namiesto množiny [A-Z] sme použili množinu veľkých písmen obsahujúcu aj diakritické znaky \p{Lu}.

Vyhľadávanie informácií
Informácie o predmete:
- Výučba predmetu na: FIIT STU
- Zabezpečuje: Michal Laclavík , Martin Šeleng
- Pracovisko: UISAV miestnosť 312, 306

V tejto kapitole sme ukázali základné vlastnosti regulárnych výrazov. V nasledujúcej podkapitole sumarizujeme regulárne výrazy a špeciálne znaky na ich zápis.

6.1.1 Zhrnutie vlastností regulárnych výrazov

Ako prehľad uvádzame nasledovnú tabuľku, ktorá vychádza z tabuľky z knihy Regular Expressions (Friedl, 2006).

Tab. 10. Zhrnutie vlastností regulárnych výrazov

Výrazy ktoré reprezentujú hľadanie jedného znaku		
.	Bodka	Zodpovedá ľubovoľnému znaku
[ ]	Množina	Zodpovedá uvedenému znaku z množiny
[^]	Negatívna množina	Zodpovedá znaku ktorý nie je uvedený
\znak	Znak s lomítkom	Použijeme na hľadanie špeciálnych znakov pre regexy ako aj definíciu ďalších množín alebo pozícií.
Výrazy ktoré sa pridávajú na počítanie znakov, kvantifikátory		
?	Otáznik	Nula alebo jeden znak
*	Hviezdička	ľubovoľný počet znakov (aj nula)
+	Plus	minimálne jeden znak alebo viac
{*min*,*max*}	Rozsah	rozsah od *min* po *max*
Výrazy ktoré reprezentujú hľadanie pozície		
^	Strieška	Začiatok riadku alebo textu (podľa nastavenia)
$	Dolár	Koniec riadku alebo textu (podľa nastavenia)
\b	Hranica slova	Hranica slova v jazyku Java. V egrep sa používa \<, \>
Ďalšie		
|	Alebo	Zodpovedá nájdeniu ktoréhokoľvek výrazu oddeleného pomocou |
( )	Zátvorky	Ohraničuje rozsah alebo slúži na určenie skupiny pre kvantifikátory a definíciu skupiny pre referencie
\1, \2, ...	referencie	Zodpovedá textu v prvej, druhej a ďalších zátvorkách

6.2 Regulárne výrazy v úlohách vyhľadávania a extrakcie informácií

V tejto kapitole uvádzame použitie regulárnych výrazov v rôznych oblastiach IR. Príklady sú v programovacom jazyku Java.

6.2.1 Sťahovanie dokumentov

V projekte irLessons je implementovaný jednoduchý sťahovač *vi.crawl.SimpleCrawler*, na ktorom si ukážeme použitie regulárnych výrazov pri sťahovaní dokumentov.

Definovanie obmedzení na URL

Pri definovaní obmedzenia sťahovania testujeme URL či vyhovuje danému regulárnemu výrazu, napríklad týmto testom:

```
String urlRegexFilter = ".*(1|3|5).*";
if (url.matches(urlRegexFilter))
      crawlURL(url);
```

Java *String* obsahuje metódu *matches()*, ktorej vstupom môže byť regulárny výraz. V tomto prípade definujeme, že URL musí obsahovať 1, 3, alebo 5, pričom je treba definovať, že URL môže obsahovať ľubovoľné znaky pred a za výskytom týchto čísel aby metóda matches vrátila zhodu na textovom reťazci.

Extrakcia liniek

Príklad extrakcie liniek z HTML pomocou regexov môžeme vidieť v projekte irLessons v triede *vi.Link*. Nižšie vidíme časť zdrojového kódu, ktorý môže zároveň slúžiť ako všeobecný príklad použitia regulárnych výrazov v jazyku Java.

```
Pattern p =
  Pattern.compile("(?i)<a[^>]*\\s+href=\"?([^>\"]+)\"?[^>]*>(.+?)</a>");
Matcher m = p.matcher( html );
while( m.find( ) ) {
  String url = m.group(1).trim();
  String anchorText = m.group(2).trim();
  ....
}
```

Najskôr do premennej triedy *Pattern* skompilujeme regulárny výraz. Následne vytvoríme premennú triedy *Matecher* nad textom, v ktorom chceme vyhľadávať. Potom v cykle prechádzame cez všetky výskyty, ktoré regulárny výraz našiel a spracujeme jednotlivé časti, zátvorky, ktoré regulárny výraz našiel. V uvedenom príklade zátvorka číslo 1 obsahuje linku a zátvorka 2 text linky, pričom zátvorka začínajúca otáznikom má špeciálnu funkciu, a teda sa nezapočítava do skupín zátvoriek. Začiatok výrazu *(?i)* hovorí, že sa majú hľadať veľké aj malé písmená (case insensitive), teda aby slová *a* alebo *href* mohli byť napísané ľubovoľne. Za „<a“ môže nasledovať hocičo okrem ukončenia značky <a>. Musí tiež nasledovať aspoň jeden znak bielej medzery „\s”. Tento je uvedený s dvoma lomítkami, pretože ide o textový reťazec v jazyku Java, a teda každé použitie lomítka musíme opakovať dvakrát. Potom nasleduje text *href*= a negatívna množina *[^>\"]* vyjadrujúca linku, ktorá môže obsahovať všetko okrem ukončenia značky <a> a úvodzoviek. Úvodzovky musíme písať s lomítkom, keďže ide o reťazec v Jave. Linka môže a nemusí byť ohraničená

úvodzovkami, čo je vyjadrené pomocou *"?*. Znovu môžu nasledovať rôzne vlastnosti HTML značky <a> vyjadrené množinou *[^>]**, kde je značka ukončená a nasleduje text linky (anchor text), ktorý nemusí byť len textom ale môže to byť ľubovoľné HTML, avšak nemala by tam byť ďalšia značka <a>. Toto je vyjadrené takzvaným lenivým (lazzy) regulárnym výrazom „.+?“, ktorý hovorí že sa môže opakovať ľubovoľný znak, avšak pri prvom úspešnom nájdení výrazu, ktorý sa zhoduje s tým čo má nasledovať za použitým lenivým operátorom (v tomto prípade ukončenie značky), sa opakovanie zastaví. Klasické použitie .+ by sa zastavilo na konci dokumentu a začalo spätne hľadať posledný výskyt značky .

6.2.2 Predspracovanie textu

Tokenizácia

Tokenizáciu pomocou regexov môžeme vysvetliť na dokumente číslo 5:

> **CIT je spoločné pracovisko Ústavu informatiky Slovenskej akadémie vied(4) v Bratislave a Technickej univerzity v Košiciach, zriadené k 1.2.2005. Pracovisko sídli na ulici Boženy Nemcovej 3 v Košiciach.**
>
> **CIT FEI, TU v Košiciach, Letná 9, 042 00 Košice, Tel./fax: +421 55 602 4128**

Text môžeme tokenizovať na slová pomocou regulárnych výrazov. Najjednoduchšie čo nás napadne je tokenizácia pomocou medzier alebo znakov bielej medzery (whitespace characters), čo môžeme napísať takto:

```
String[] words = body.split("\\s+");
```

Výsledkom je pole slov:

```
[CIT] [je] [spoločné] [pracovisko] [Ústavu] [informatiky] [Slovenskej]
[akadémie] [vied] [v] [Bratislave] [a] [Technickej] [univerzity] [v]
[Košiciach,] [zriadené] [k] [1.2.2005.] [Pracovisko] [sídli] [na] [ulici]
[Boženy] [Nemcovej] [3] [v] [Košiciach.] [CIT] [FEI,] [TU] [v] [Košiciach,]
[Letná] [9,] [042] [00] [Košice,] [Tel./fax:] [+421] [55] [602] [4128]
```

Tak ako sme si už pri tokenizácii napísali skôr, pravdepodobne nechceme slovo Košice aj s čiarkou na konci, alebo dátum s bodkou na konci vety. Môžeme použiť niečo zložitejšie zadefinovaním napríklad negatívnej množiny, teda aby sa slová vyselektovali podľa akýchkoľvek znakov iných ako čísla a písmená:

```
String[] words = body.split("[^0-9\\p{L}]+");
```

Výsledkom je pole slov, kde sa nám napríklad rozbil dátum na viacero slov.

```
[CIT] [je] [spoločné] [pracovisko] [Ústavu] [informatiky] [Slovenskej]
[akadémie] [vied] [v] [Bratislave] [a] [Technickej] [univerzity] [v]
[Košiciach] [zriadené] [k] [1] [2] [2005] [Pracovisko] [sídli] [na] [ulici]
[Boženy] [Nemcovej] [3] [v] [Košiciach] [CIT] [FEI] [TU] [v] [Košiciach]
[Letná] [9] [042] [00] [Košice] [Tel] [fax] [421] [55] [602] [4128]
```

Konverzia HTML na text a segmentácia textu

V projekte irLessons trieda *vi.analysis.SegmentationRepository* implementuje veľmi jednoduchú segmentáciu textu, ktorá funguje na obmedzené HTML značky. Trieda String v jazyku Java obsahuje metódu *replaceAll()*, kde vstupom je regulárny výraz. Tak napríklad môžem jednoducho predspracovať text pomocou týchto regexov:

```
body = body.replaceAll("<br/?>|</li>|<ul>|<p>", "\n");
body = body.replaceAll("<li>", "- ");
body = body.replaceAll("</?[^>]+>", " ");
```

Prvý riadok prepíše html značky ako koniec riadka, koniec odrážky alebo odsek na znak nového riadka. Druhý riadok prepíše značku odrážky na pomlčku a posledný riadok prepíše všetky ďalšie HTML značky na medzeru. Výsledkom je text kde možno jednoducho identifikovať odsek, odrážku a podobne. Výsledkom je potom konverzia HTML do čistého textu s čiastočným zachovaním štruktúry, tak ako vidíme aj v príklade na začiatku kapitoly 6.1.

Ďalšími konverziami môže byť prepis tabuľky na jednoduchšiu štruktúru, teda napríklad nahradenie značky <td> pomocou tabulátora a podobne, čo možno využiť pri extrakcii informácií pomocou vzorov.

6.2.3 Identifikácia entít

V tejto kapitole si uvedieme príklady regulárnych výrazov na extrakciu nasledovných typov entít z textu:

- telefónne čísla,
- PSČ, názvy sídel (miest a dedín) a adresy,
- firmy a organizácie,
- osoby,
- dátumy

Telefónne čísla

V príkladoch dokumentov v kapitole 1.2.2 máme viacero telefónnych čísel. Môžeme ich extrahovať týmto regulárnym výrazom:

```
(?i)\b(phone|fax|tel): ?([-+ ()/0-9]+)\b
```

V regexe definujeme, že za *tel*, *fax* alebo *phone* nasleduje sekvencia čísel, ktoré môžu obsahovať zátvorku, medzeru, plus, lomítko alebo pomlčku. Na začiatku výrazu definujeme že je case insensitive. Tiež obmedzujeme výraz z oboch strán hranicami slov. Výraz je pomerne jednoduchý a nie vždy by fungoval správne. Dovoľuje, aby plus bolo hocikde a nielen na začiatku telefónneho čísla a podobne. Výsledok výrazu na dokumente číslo 4 je:

Sekretariát: Tel./Fax: 02 5477 1004

Vidíme, že ak chceme vytiahnuť samotné telefónne číslo, musíme sa odvolať na druhú zátvorku z regexu. Prvá zátvorka obsahujúca prepínač na ignorovanie rozdielu veľkých a malých písmen sa nepočíta. Tiež si môžeme všimnúť, že v množine pre definíciu povolených znakov telefónneho čísla sme použili aj špeciálne znaky, ako zátvorky a plus, bez použitia spätného lomítka. Ako už bolo spomenuté, v množine sú ako špeciálne interpretované iba tri znaky. Sú to pomlčka (iba ak nie je na začiatku), strieška a koniec množiny (hranatá zátvorka).

Lokalita

V tejto časti si ukážeme jednoduché príklady extrakcie lokality reprezentovanej adresou, PSČ alebo iným miestom, ktoré reprezentuje lokalitu.

Najjednoduchšou je extrakcia lokalít pomocou predložiek *na*, *v*, alebo *do*, za ktorými nasleduje slovo začínajúce veľkým písmenom:

```
\b(na|v|do)\s(\p{Lu}\p{L}+)\b
```

Takýto výraz nám vyextrahuje napríklad tieto lokality:

CIT - Centrum pre informačné technológie
CIT je spoločné pracovisko Ústavu informatiky Slovenskej akadémie
vied v Bratislave a Technickej univerzity v Košiciach, zriadené k 1.2.2005.
Pracovisko sídli na ulici Boženy Nemcovej 3 v Košiciach.

CIT FEI, TU v Košiciach, Letná 9, 042 00 Košice, Tel./fax: +421 55 602 4128

Použitie takýchto výrazov môže reprezentovať lokalitu ale nie vždy. Ak napríklad pošlem email s textom „Text je v PDF súbore“. Dostanem extrakciou lokalitu PDF čo je nezmysel. Jednoduché regexy teda môžu mať vysokú úplnosť (recall) ale malú presnosť (precision).

Podobným spôsobom môžeme vytiahnuť aj mestá a dediny pomocou PSČ. Teda že za piatimi číslami nasleduje slovo (slová) začínajúce veľkým písmenom:

```
\b([0-9]{3} ?[0-9]{2}) (\p{Lu}[^0-9,\n.]+)\b
```

Vidíme že sme definovali výraz tak, že má obsahovať tri za sebou idúce čísla, môže obsahovať medzeru, potom nasledujú dve čísla a za nimi sekvencia znakov začínajúca veľkým písmenom a neobsahujúca čiarku, bodku, nový riadok alebo čísla aby sme pokryli viac slovné názvy miest a obcí. Samozrejme daný výraz nemusí vždy fungovať správne, čo je treba odladiť na väčšej kolekcii dokumentov.

CIT FEI, TU v Košiciach, Letná 9, 042 00 Košice, Tel./fax: +421 55 602 4128

Z prvej zátvorky vieme zistiť PSČ a z druhej mesto. Podobným spôsobom vieme výraz rozšíriť na celú adresu aj s ulicou, kde táto predchádza PSČ na novom riadku alebo je oddelená od PSČ čiarkou. Extrakcia ulice a popisného čísla nie je však jednoduchá vzhľadom na to, že popisné číslo môže obsahovať čísla, lomítka, písmená a podobne.

Firmy a organizácie

Mená firiem môžeme extrahovať pomocou regulárnych výrazov kde za potenciálnym menom firmy nasleduje skratka ako s.r.o., alebo a.s., ktorá definuje typ firmy. Meno firmy zvyčajne obsahuje písmená a začína veľkým písmenom, avšak na príkladoch dokumentov vidíme, že môže obsahovať aj čísla, prípadne iné znaky a môže začínať aj malým písmenom alebo iným znakom. Môžeme teda definovať meno firmy ako negatívnu množinu:

```
\b([^ ,\n.]+)[, ]+(spol\. s r\. ?o\.|s\. ?r\. ?o\.|a\. ?s\.)
```

Výsledkom bude správne rozpoznanie jednoslovných firiem:

Generované redakčným systémom Buxus spoločnosti ui42, spol. s r.o.

Je však problém ako definovať viacslovnú firmu. Napríklad nasledovný regex by nám vytiahol celý riadok:

```
\b(\p{Lu}[^,\n.]+)[, ]+(spol\. s r\. ?o\.|s\. ?r\. ?o\.|a\. ?s\.)
```

Generované redakčným systémom Buxus spoločnosti ui42, spol. s r.o.

Keby sme obmedzili počet slov napríklad na tri, stále by sme dostali nesprávny názov firmy. Extrakcia firiem pomocou regexov je teda možná, ale silne závisí od aplikácie. Niekedy je vhodnejšie použiť na extrakciu zoznam mien firiem, ktorý sa dá získať z dostupných zdrojov a kombinovať to s regulárnym výrazom, aby sme nenašli v texte mená firiem na miestach ktoré reprezentujú niečo iné.

Mená ďalších organizácií vieme zistiť pomocou identifikácie slov ako fakulta, ústav, univerzita, centrum a ďalších, ktoré sú súčasťou názvu organizácie. V slovenčine je však celkovo problém s pomenovanými entitami, pretože iba prvé slovo názvu začína veľkým písmenom, a teda neviem, kde názov končí.

```
\b((Fakult.|Ústav.?|Centrum) [^,\n.]+)\b
```

Tento jednoduchý výraz predpokladá, že kľúčové slovo reprezentujúce organizáciu je prvé a za ním po koniec riadku, čiarku alebo bodku nasleduje názov organizácie.

CIT - Centrum pre informačné technológie
CIT je spoločné pracovisko Ústavu informatiky Slovenskej akadémie vied v Bratislave a Technickej univerzity v Košiciach, zriadené k 1.2.2005.
Pracovisko sídli na ulici Boženy Nemcovej 3 v Košiciach.

V dokumente číslo 4 nám správne vytiahlo organizáciu CIT a aj Ústav informatiky, ktorý je ale nesprávne ukončený. Keby sme napríklad chceli koniec názvu obmedziť vymenovaním spojok ako napríklad „a“, nenašlo by nám FIIT STU:

Fakulta informatiky a informačných technológií STU v Bratislave
Ilkovičova 3
842 16 Bratislava 4

Osoby

Osoby môžeme extrahovať pomocou vzoru kde idú po sebe dve slová začínajúce veľkým písmenom:

```
\p{Lu}\p{L}+ \p{Lu}\p{L}+
```

Samozrejme v takomto prípade dostaneme aj nesprávne identifikácie osôb, kde sme rozpoznali aj *FIIT STU* ako osobu.

Vyhľadávanie informácií
Informácie o predmete:
- Výučba predmetu na: FIIT STU
- Zabezpečuje: Michal Laclavík , Martin Šeleng
- Pracovisko: UISAV miestnosť 312, 306

Regulárny výraz možno rozšíriť o detekciu titulov, alebo kombinovať so zoznamom krstných mien.

```
(Ing|Mgr|RNDr|MUDr)\.? (\p{Lu}\p{L}+ \p{Lu}\p{L}+)
```

Riaditeľ: Doc. Ing. Ladislav Hluchý, PhD.
Adresa: Dúbravská cesta 9, 845 07 Bratislava, SR
Sekretariát: Tel./Fax: 02 5477 1004
e-mail: sekr.ui@savba.sk

V nasledujúcej kapitole 7 diskutujeme o ďalších technikách extrakcie informácií, ktoré využívajú aj extrakciu pomocou vzorov, napríklad regulárnych výrazov, ktoré sme opísali v tejto kapitole.

6.3 Príklady

- Na aké úlohy sa dajú v oblasti vyhľadávania informácií (information retrieval) použiť regulárne výrazy (regex)?
- Napíšte regex na vyhľadanie objektov v uvedených dokumentoch tak, aby boli aj všeobecnejšie použiteľné: sídla (mestá a dediny), firmy, lokality, PSČ, dátumy, roky, telefónne čísla, osoby.