Regularni izrazi

Uvod

Regularni izrazi (regex ili regexp skraćeno) su specijalni tekstualni stringovi koji opisuju šablon za pretragu. Mogu se posmatrati kao unapređena verzija *wildcard* karaktera. Na primer, poznato je da *.txt se koristi kako bi se pronašli svi tekstualni fajlovi-regex ekvivalent toga bi bio .*\.txt .

Međutim, sa regularnim izrazima moguće je uraditi mnogo više od toga. Na primer:

\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}\b

omogućava pretragu email adrese-bilo koje email adrese. Veoma sličan izraz, u kojem je prvo \b zamenjeno sa ^ a poslednje sa $ omogućava programeru da proveri da li je korisnik korektno uneo formatiranu email adresu, u samo jednoj liniji koda, bez obzira da li je u pitanju Perl, PHP, Java, Python .NET ili neki drugi programski jezik. Na primer, u C++ programskom jeziku, program koji bi proveravao da li je dati string korektna email adresa izgledao bi ovako:

```
#include <iostream>
#include <string>
#include <regex>

bool is_email_valid(const std::string& email)
{
   // define a regular expression
   const std::regex pattern
      ("\\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4}\\b");

   // try to match the string with the regular expression
   return std::regex_match(email, pattern);
}

int main()
{
    std::string email1 = "text.example@randomcom";
    std::cout << email1 << " : " << (is_email_valid(email1) ?
      "valid" : "invalid") << std::endl;
}
```

Prethodni primer pokazuje kako se regularni izraz koristi da verifikuje uneti string, a naredni pokazuje kako se korišćenjem regularnog izraza mogu prepoznati sve email adrese u datom tekstu:

```
#include <iostream>

#include <string>

#include <regex>


int main(int argc, char** argv)

{

    //input string is passed as an argument

    std::string text = argv[1];

    // regularni izraz (obratiti paznju na zagrade)

    const std::regex pattern ("(\\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4}\\b)");

    std::smatch m;
```

```
    std::cout << "Prepoznate email adrese: " << std::endl;
    while(std::regex_search(text, m, pattern)){
        std::cout << m[1].str() << std::endl;
        text = m.suffix().str();
    }
}
```

Ako se ovaj program pokrene sa argumentom koji predstavlja tekst u kojem se traže email adrese, dobija se sledeći izlaz:

```
./email_address_search "Moj email je petar_petrovic@uns.ac.rs, a email moga drugara je marko.markovic@gmail.com"
```

```
Prepoznate email adrese:
petar_petrovic@uns.ac.rs
marko.markovic@gmail.com
```

Regularni izrazi - o čemu će biti reči

Ukoliko vam prethodni primer sa email adresom nema previše smisla, nemojte se uzbuđivati oko toga. Bilo koji ne-trivijalni regex deluje zbunjujuće za nekoga ko nije familijaran sa njima, ali već nakon nekoliko primera i pojašnjenja pravila bićete u stanju da sami kreirate regularne izraze, čak i one prilično komplikovane. Pokušaćemo ne samo da objasnimo sintaksu, koja je neophodna, već takođe i detalje, kako regex engine zaista radi u pozadini. Naučićete dosta, čak i ako ste koristili regularne izraze ranije. Biće vam jasno zašto neki regularni izraz ne radi ono što ste očekivali (često se dešava i iskusnijima) i nadamo se sačuvati vam dosta vremena dok pokušavate da uradite nešto korisno sa regularnim izrazima.

Aplikacije i jezici koji podržavaju regularne izraze

Postoji dosta aplikacija koje podržavaju regularne izraze. Većina programskih jezika takođe omogućava biblioteke za rad sa njima. Kao programer, dosta vremena ćete sačuvati i truda koristeći ih, tako što ćete često u jednoj liniji koda uraditi posao koji bi inače zahtevao mnogo više.

Međutim, čak i ako niste programer, regularni izrazi vam mogu pomoći veoma često. Oni vam omogućavaju da pronađete informaciju od interesa mnogo brže i jednostavnije. Možete ih koristiti u moćnim *search and replace* alatima kako biste brzo napravili mnoštvo izmena u velikom broju fajlova istovremeno. Jednostavan primer je **gr[ae]y** koji će pronaći obe vrste izgovora iste reči.

Terminologija

Regularni izrazi su šabloni koji opisuju određeni segment teksta. Ime su dobile iz matematike i teorije na kojoj su bazirani, ali mi nećemo ulaziti u te detalje. Često ćemo koristiti termin *regex* umesto regularni izraz, kako bismo pojednostavili pisanje. Regularne izraze ćemo ispisivati boldovanim slovima, da bi bili bolje vidljivi.

U terminima regularnih izraza *poklapanje* se odnosi na deo teksta, ili sekvencu bajtova/karaktera koji odgovaraju datom regularnom izrazu.

Izraz koji smo videli ranije

\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}\b

je malo složeniji šablon. On opisuje sekvencu slova, cifara, tačaka, underscore karaktera, znakova za procenat ili minus znakova, praćenu @ znakom, nakon čega opet sledi sekvenca slova, cifara ili minus znakova ili tačaka, na kraju terminiranih sa jednom tačkom i dva do četiri slova. Ukratko, taj šablon opisuje email adresu. Sa tim regularnim izrazom, primenjenim na datom tekstu, izdvojićete sve email adrese, ili jednostavno proveriti da li je data email adresa korektno formatirana.

U tekstu ćemo koristiti termin string kao ulaz za regularni izraz.

Literali

Osnovni regularni izraz se sastoji od jednog literala, na primer **a**. On će dovesti do poklapanja kod prve pojave istog toga karaktera u datom stringu. Ako je string "Jack is a boy", poklopiće se sa prvim a odmah iza slova J. Činjenica da je a u sredini reči, ne utiče na regex engine. Ako je vama značajan taj detalj, moraćete to da posebno naglasite, kako bi bilo prepoznato koristeći mehanizme koje ćemo opisati kasnije. I drugo "a" će biti prepoznato, ali samo nakon što se kaže regex engine-u da nastavi tamo gde je prethodno stao (ako je u pitanju neki tekst editor koji podržava regularne izraze, u programskim jezicima, obično postoji posebna naredba za nastavak pretrage).

Slično **cat** će poklopiti reč "cat" u stringu "About cats and dogs". U ovom slučaju, regularni izraz se sastoji od sekvence tri literala, što praktično znači: Nađi mi c, koji je praćen sa a, nakon čega sledi t. Obratite pažnju da su regularni izrazi case-sensitive podrazumevano. Samim tim regex **cat** neće poklopiti "Cat" osim ukoliko to ne naglasite posebno.

Specijalni karakteri

Pošto želimo više od jednostavne pretrage literala u okviru teksta, neophodno je da određene karaktere rezervišemo za specijalnu upotrebu. Ukupno će biti 11 karaktera sa specijalnim značenjem u okviru ovog upustva. To su:

[, \ , ^, $, ., |, ?, *, +, (i).

Ovi specijalni karakteri se često nazivaju i metakarakteri. Ako želite da bilo koji od njih koristite u svojim regularnim izrazima, neophodno je da ispred njih stavite \ (backslash). Na primer, ako želite da pronađete string „1+1=2", ispravan regex bi bio **1\+1=2**. U suprotnom, plus znak bi imao specijalno značenje. Takav regularan izraz bi bio validan, ali ne bi doveo do poklapanja sa stringom koji je očekivan već sa delom stringa „111=2" u "123+111=234", usled specijalnog značenja znaka +.

Ukoliko zaboravite da dodate \ ispred karaktera gde to nije dozvoljeno kao na primer u izrazu **+1**, dobićete poruku o grešci.

Važno je napomenuti da većina regex engine-a tretira { kao literal, osim ukoliko se nalazi u okviru bloka za ponavljanje kao npr. {1,3}. Dakle, generalno, ne morate stavljati \ ispred njih iako možete (izuzetak ovoga je java.util.regex paket koji zahteva da se ispred vitičaste zagrade stavlja \ ).

Takođe, obratite pažnju da \ u kombinaciji sa nekim literalima može da kreira regex specijalnog značenja, npr **\d** će poklopiti cifru 0 do 9.

Karakteri koji se ne mogu prikazati (non-printable characters)

Specijalne sekvence karaktera se mogu koristiti kako bi se dodali ovakvi karakteri u regularne izraze. Koristite **\t** kako biste poklopili tab karakter (ASCII 0x09), **\r** za carriage return (0x0D) i **\n** za line feed (0x0A). Egzotični bi bili **\a** (bell, 0x07), **\e** (escape, 0x1B), **\f** (form feed, 0x0C) i **\v** (vertical tab, 0x0B).

Treba voditi računa o tome da Windows u tekstualnim fajlovima linije terminira sa \r\n, dok je u Linux-u to \n.

Svaki karakter može biti dodat u regularni izraz ukoliko se zna njegov ASCII kod, na isti način.

Kako radi regex engine

Poznavanje rada omogućava bolje i efikasnije korišćenje regularnih izraza. Osim toga, olakšaće razumevanje zašto neki regularni izraz nema efekat koji smo mi zamislili.

Regularni izraz uvek vraćaju levo poklapanje prvo (Leftmost Match First). Ovo je važno imati na umu: prvo poklapanje sa leve strane će biti vraćeno, čak i ukoliko kasnije postoji "bolje" poklapanje. Kada mu se prosledi string, engine će krenuti od njegovog početka, pokušaće sa svim mogućim permutacijama regularnog izraza na tom prvom karakteru i samo ukoliko se sve mogućnosti ispostave kao pogrešne, nastaviće sa narednim karakterom u stringu. Tada, na identičan način kao i u slučaju prvog, isprobaće sve moguće permutacije regularnog izraza sa tim drugim karakterom, u istom redosledu kao i prvi put. Kao rezultat toga, "najlevlje" podudaranje će biti vraćeno. Na primer, kada se primeni regex **cat** na string "He captured a catfish for his cat.", engine će pokušati da poklopi prvi token izraza c sa prvim karakterom u stringu H, što će biti neuspešno. Pošto u ovom slučaju nema permutacija koje bi mogle biti probane (regex je sekvenca literala), pokušaće isto da uradi sa "e" u ulaznom stringu, što će takođe biti neuspešno. Tek kod četvrtog karaktera **c** će da se poklopi sa c, nakon čega će pokušati da sledeći token poklopi sa narednim karakterom, što će takođe biti uspešno. Međutim, **t** neće moći biti poklopljen sa "p". U tom trenutku, prelazi na sledeći u ulaznom stringu: "a". Na sličan način, tek na 15. poziciji u stringusva tri tokena iz regularnog izraza će se poklopiti sa karakterima u stringu, i biće vraćeni nazad kao poklapanje, bez nastavka u pokušaju da se pronađe bolje poklapanje: prvo poklapanje je dovoljno dobro u tom smislu.

U ovom primeru, regex engine manje više radi kao regularni pretraživači teksta. Ipak, važno je imati u vidu korake koje izvodi on prilikom pokušaja da pronađe poklapanje jer će rezultati nekad biti iznenađujući, ali će uvek biti logični imajući u vidu gore opisanu proceduru.

Klase karaktera ili Skupovi karaktera

Koristeći klase (skupove) karaktera, moguće je reći regex engine-u da poklopi samo jedan od nekoliko karaktera. Da bi se kreirali, potrebno je samo smestiti sve karaktere unutar uglastih zagrada. Ako želimo da pronađemo poklapanje sa **a** ili **e** to ćemo uraditi sa [ae].

Na taj način, **gr[ae]y** će poklopiti ili „gray" ili „grey", što može biti veoma korisno ukoliko ne znamo da li je dokument koji pretražujemo pisan britanskim ili američkim dijalektom.

Klasa karaktera poklapa uvek samo jedan karakter! Regex od gore neće poklopiti "graay", "graey" ili bilo šta slično. Redosled karaktera unutar klase nije relevantan, rezultati će biti identični. Znak "**-**" unutar uglastih zagrada se može koristiti da specificira opseg karaktera: **[0-9]** će poklopiti jednu cifru između 0 i 9. Takođe, moguće je specificirati više od jednog opsega: **[0-9a-fA-F]** opisuje jednu heksadecimalnu cifru, prihvatajući i mala i velika slova. Dodatno, moguće je kombinovati opsege zajedno sa pojedinačnim karakterima: **[0-9a-fxA-FX]** će poklopiti heksadecimalnu cifru ili slovo X, pri čemu redosled navođenja karaktera ponovo nije važan.

Korisne primene klasa karaktera

Pronaći reč, čak i ako je pogrešno napisana, na primer **sep[ae]r[ae]te** ili **li[cs]en[cs]e**. Ili pronalaženje promenljive u programskom jeziku **[A-Za-z_][A-Za-z_0-9]***. Takodje, heksadecimalni broj u C stilu **0[xX][A-Fa-f0-9]+**.

Invertovane klase karaktera

Dodavanjem znaka **^** nakon otvaranja uglaste zagrade označava invertovanje klase karaktera. Rezultat će biti poklapanje sa bilo kojim karakterom koji ne pripada navedenoj klasi karaktera. Za razliku od tačke, invertovanje klase karaktera takođe poklapa i (nevidljive) karaktere za oznaku kraja reda. Važno je zapamtiti da invertovanje klase karaktera i dalje moraju da poklope karakter: **q[^u]** ne znači "q iza koga ne ide u", već znači "q praćeno karakterom koji nije u". U tom smislu, neće biti poklopljeno sa q u stringu "Iraq". Biće poklopljeno sa q i space karakterom iza njega u "Iraq is a country".

Metakarakteri unutar klasa karaktera

Jedini specijalni karakteri unutar klasa karaktera su], \, ^ i -. Ostali metakarakteri se tretiraju kao normalni karakteri kada se nađu unutar klasa karaktera i ne moraju imati \ ispred.

Ako želimo da tražimo + ili *, to ćemo raditi sa **[+*]**. Regularni izraz će raditi dobro ako dodamo \ ispred metakaraktera unutra klase karaktera, ali to samo pogoršava čitljivost koja svakako može predstavljati problem.

Da bi se dodao \ kao karakter bez specijalnog značenja unutar klase karaktera, mora biti prefiksovan još jednim \ : **[\\x]** poklapa \ ili x.

Metakarakteri], ^ i - mogu biti dodati tako što se ispred njih stavlja \, ili tako što ih postavljamo na poziciju u kojoj one više nemaju specijalno značenje: da bi se dodao ^ potrebno ga je staviti na bilo koje mesto osim prvog unutar klase karaktera. Na primer **[x^]** poklapa x ili ^. Slično zatvorena uglasta zagrada može da se stavi neposredno nakon otvorene ili nakon ^: []x] poklapa zatvorenu uglastu zagradu ili x, a **[^]x]** poklapa bilo koji karakter koji nije zatvorena uglasta zagrada ili x. Znak - može da se uključi neposredno nakon otvorene zagrada, ili neposredno pre zatvorene zagrade, kao i odmah nakon ^ za negiranje. I **[-x]** i **[x-]** poklapaju x i znak minus.

Skraćeno označavanje klasa karaktera

Pošto se određene klase karaktera koriste često, postoje posebne oznake za određene klase karaktera: **\d** je oznaka za [0- 9], dok **\w** stoji za slovo. Šta to tačno podrazumeva zavisi od regex-a koji se koristi. U svim varijantama, to mora da uključi **[A-Za-z]**. Međutim, u većini, underscore i cifre su takođe uključene. **\s** stoji kao "whitespace character", što podrazumeva space ili tab. Često su tu obuhvaćeni i carriage return i line feed kao u **[\t\r\n]**.

Skraćeno zapisivanje klasa karaktera može biti korišćeno unutar i izvan uglastih zagrada. **\s\d** poklapa "whitespace" karakter praćen cifrom, dok **[\s\d]** poklapa jedan karakter koji je ili whitespace ili cifra. Kada se primeni na "1 + 2 = 3", prvi regex će rezultovati sa „ 2" (space dva), dok će drugi rezultovati sa „1" (jedan). **[\da-fA-F]** odgovara heksadecimalnoj cifri i ekvivalentno je sa **[0-9a-fA-F]**.

Invertovano skraćeno označavanje klasa karaktera

Prethodno navedene tri skraćenice imaju takođe i invertovane simbole: **\D** je isto što i **[^\d]**, **\W** je isto što i **[^\w]**, a **\S** je ekvivalentno **[^\s]**. Treba biti pažljiv prilikom korišćenja negirane skraćene notacije unutar uglastih zagrada: **[\D\S]** nije isto što i **[^\d\s]**. Drugi zapis poklapa bilo koji karakter koji nije cifra ili whitespace: dakle poklopiće x ali ne i 8. Prvi izraz, sa druge strane, poklapa bilo koji karakter koji ili nije cifra, ili nije whitespace karakter. Pošto cifra nije whitespace, a whitespace nije cifra **[\D\S]** će poklopiti bilo koji karakter, cifru, whitespace ili nešto treće.

Ponavljanje klasa karaktera

Ako ponavljate klasu karaktera koristeći ?, * ili + operatore, ponavljaćete celu karakter klasu, ne samo karakter koji se poklopio. Na primer regularni izraz **[0-9]+** poklapa string „837" kao i „222".

Ukoliko vam treba poklapanje specifičnog karaktera, pre nego klase, moraju se koristiti reference (o kojima će biti reči kasnije), kao npr. **([0- 9])\1+** koje će poklopiti „222" ali ne i "837". Ako joj se prosledi string "833337", poklopiće „3333" u sredini stringa. Ako ni to nije poželjno ponašanje, moraju se koristiti pogled napred (lookahead) i pogled nazad (lookbehind). Ali idemo korak po korak.

Kao što je već rečeno ranije, redosled karaktera unutar klase karaktera nije bitan: **gr[ae]y** će poklopiti „grey" u stringu "Is his hair grey or gray?" jer je to prvo poklapanje sa leve strane.

Već smo videli kako radi regex engine kada pokušava da poklopi samo literale. Sada ćemo objasniti šta se dešava kada postoji više od jedne permutacije, kao u slučaju od gore. Ništa značajno se ne dešava na prvih 12 karaktera stringa. Engine ne uspeva da poklopi g na svakom koraku, i nastavlja na sledeći karakter stringa. Kada stigne do 13-tog, g se poklapa, kao i r koji sledi iza njega. Treći token [ae] je sledeći koji se pokušava poklopiti sa tekstom "e". Klasa karaktera daje dve opcije: poklopiti a ili poklopiti e. Prvo će pokušati sa a što će biti bezuspešno, ali onda nastavlja sa svim mogućim permutacijama pre nego što odluči da nije moguće konačno preklapanje. Kao rezultat toga, poklopiće e sa „e" u stringu. Poslednji regex token je y koji se uspešno poklapa sa narednim karakterom stringa. Kao rezultat, vraćen je „grey" kao rezultat poklapanja, pri čemu se ne gleda dalje. Još jednom, najlevlji pogodak je vraćen nazad, iako smo stavili a kao prvi karakter u okviru klase karaktera i činjenice da je i tekst "gray" mogao biti poklopljen u nastavku.

Tačka poklapa (skoro) sve karaktere

U regularnim izrazima, tačka je jedan od najčešće korišćenih metakaraktera. Nažalost, ona je takođe i jedan od najčešće POGREŠNO korišćenih metakaraktera.

Tačka zamenjuje jedan karakter, bez obzira na to koji je to karakter. Jedini izuzetak su newline karakteri. Dakle, tačka je kratka notacija za invertovanu klasu karaktera **[^\n]** (UNIX/Linux regex) odnosno **[^\r\n]** (Windows regex).

Ovaj izuzetak postoji pretežno iz istorijskih razloga. Prvi alati koji su koristili regularne izraze su radili liniju po liniju. Čitajući jednu po jednu iz datoteke, primenljivali su regularne izraze odvojeno na svakoj od njih. Kao rezultat, korišćenjem takvih alata string nikako nije mogao da sadrži newline karaktere, tako da . nikako nije ni mogla da se koristi za poklapanje istih.

Koristite . pažljivo

Tačka je veoma moćan regex metakarakter. Omogućava vam da budete "lenji". Stavite tačku i sve će biti poklopljeno kada testirate regex na validnim podacima. Problem sa ovim je što će regex biti poklopljen i u slučajevima kada to ne bi trebalo da se desi. Ako ste početnik sa regularnim izrazima, ovakve situacije vam možda deluju kao nešto što se retko ili nikada ne dešava, pa ćemo dati primer za to.

Primer je jednostavan: recimo da želimo da poklopimo datum u formatu dd/mm/yy, ali da ostavimo korisniku odabir separatora u zapisu datuma. Brzo rešenje bi onda bilo **\d\d.\d\d.\d\d**. Deluje dobro na prvi pogled, poklopiće uspešno datum zapisan kao „02/12/03". Međutim, problem je „02512703" koji će takođe biti prepoznat kao dobro formatiran datum u skladu sa našim regularnim izrazom. U njemu, prva tačka poklapa „5", dok druga poklapa „7". Očito ne ono što smo prvobitno zamislili. Bolje rešenje je

\d\d[- /.]\d\d[- /.]\d\d

Ovaj regularni izraz dozvoljava -, razmak, / ili tačku kao separatore datuma (obratite pažnju da tačka nije metakarakter kada se nađe unutar klase karaktera, tako da nema potrebe da je prefiksujemo sa backslash karakterom.

Ipak, ovaj regex je i dalje daleko od savršenog jer će da poklapa i datume kao što je „99/99/99”.

[0-3]\d[- /.][0-1]\d[- /.]\d\d

je poboljšanje, iako i dalje nije savršeno jer će poklopiti i „39/19/99”. Koliko savršen vaš regex treba da bude, zavisi od toga šta želite da radite sa njim.

Ako želite da validirate korisnički ulaz, mora da bude savršen. Ako parsirate podatke iz datoteke izvora kojem se veruje, jer generiše izlaze uvek na isti način, poslednje navedeno je verovatno dovoljno dobro. Kasnije ćemo navesti bolji regex koji omogućava poklapanje datuma.

Koristite invertovanu klasu karaktera umesto tačke

Ovo će biti detaljnije objašnjeno kada budemo predstavili operatore ponavljanja * i +, ali već sada je dobro skrenuti pažnju na to. Pogledajmo primer:

Recimo da hoćemo da poklopimo string koji se nalazi u dvostrukim navodnicima. Zvuči jednostavno. Možemo imati proizvoljan broj karaktera između dvostrukih navodnika, dakle **".*"** izgleda kao regex koji obavlja posao. Tačka poklapa bilo koji karakter, a * omogućava da tačka bude ponovljena proizvoljan broj puta, uključujući nijednom. Ako testiramo regex na stringu “Put a "string" between double quotes” poklopiće uspešno tekst "string". Ukoliko ga testiramo na stringu “Houston, we have a problem with "string one" and "string two". Please respond.”, izlaz će biti „"string one" and "string two"”. Definitivno ne ono što smo planirali, a razlog za to je činjenica da je * operator koji je pohlepan (greedy) i poklopiće što je više moguće karaktera.

U primeru sa datumom, poboljšali smo naš regex tako što smo tačku zamenili sa klasom karaktera. Ovde ćemo uraditi isto. Ne želimo proizvoljan broj bilo kojih karaktera između dvostrukih navodnika. Mi želimo proizvoljan broj karaktera koji nisu dvostruki navodnici ili newline karakteri unutar dvostrukih navodnika. Dakle, regex bi trebao da bude **"[^"\r\n]*"**.

Markeri za početak i kraj stringa

Do sada smo ispričali priču o literalima i klasama karaktera. U oba slučaja, stavljanje bilo kojeg od njih u regularni izraz, regex engine će probati da poklopi jedan jedini karakter. Markeri su drugi tip operatora. Oni ne vode ka poklapanju karaktera, već poklapanju pozicije pre, iza ili između karaktera. Metakarakter ^ poklapa poziciju pre prvog karaktera u stringu. Ako primenimo **^a** poklopićemo „a” na stringu “abc”, dok **^b** neće dovesti do poklapanja na “abc”, jer se b ne nalazi odmah nakon početka stringa označanog sa **^**.

Slično, **$** poklapa poziciju neposredno iza poslednjeg karaktera u stringu: **c$** poklapa „c” u “abc”, dok **a$** ne rezultuje poklapanjem.

Primeri primene

Kada se koriste za validaciju ulaznih podataka programskog jezika, markeri su veoma značajni. Ako želimo proveriti da li je unet celobrojni podatak, korektan regex za to bi bio **^\d+$**. Razlog za to je što početak stringa mora biti pre **\d+**, a završetak neposredno iza poslednje cifre, pri čemu se samo cifre mogu pojavljivati u stringu.

Ako ima potrebe za trim-ovanjem space karaktera na početku i kraju stringa, regularni izrazi nam takođe mogu biti od velike pomoći. **^\s+** poklapa whitespace karaktere, dok **\s+$** poklapa one koji se nalaze na kraju stringa.

Korišćenje ^ i $ metakaraktera za markere početka i kraja linije

Ukoliko radite sa stringom koji se sastoji od nekoliko linija, npr "first line\nsecond line" (gde \n predstavlja kraj linije), često se javlja potreba da se radi sa linijama, radije nego sa celim stringom. Stoga, svi regex engine-i omogućavaju proširenje značenja oba markera predstavljena iznad. U tom smislu, **^** može da predstavlja početak stringa (ispred karaktera f gore) ali i poziciju iza svakog karaktera za novi red (između "\n" i "s"). Slično, **$** i dalje poklapa kraj stringa (posle poslednjeg e), ali takođe i poziciju pre svakog line break karaktera (između "e" i "\n").

U većini programskih jezika i posebnih biblioteka koje se koriste za rad sa regularnim izrazima, ipak, neophodno je eksplicitno uključiti ovu proširenu funkcionalnost.

\A će uvek poklapati samo početak stringa, dok **\Z** jedino poklapa kraj stringa. Ova dva tokena nikada neće poklapati kraj reda, čak i ukoliko je uključen prethodno opisani "multiline mode".

Poklapanje nulte dužine

Pošto markeri poklapaju poziciju, ne same karaktere, regex koji se sastoji samo od markera može da rezultuje poklapanjem nulte dužine. Zavisno od situacije, ovo može da bude neželjeno ponašanje. Koristeći **^\d*$** da bi se testirala lozinka koja se sastoji samo od cifara (* umesto znaka +), izazvaće prihvatanje praznog stringa kao validnog ulaza.

Granica reči

Metakarakter **\b** je marker kao i prethodno predstavljena dva. On poklapa sa pozicijom koja se naziva granica reči (*word boundary*), nulte dužine.

Postoje četiri različite pozicije koje se karakterišu kao granice reči:

- Pre prvog karaktera u stringu, ako je prvi karakter karakter u okviru reči
- Posle poslednjeg karaktera u stringu, ako je poslednji karakter karakter iz reči
- Između karaktera iz reči i jednog koji nije, ali dolazi odmah nakon karaktera iz reči
- Između karaktera koji ne pripada reči i karaktera koji pripada reči neposredno iza njega

Jednostavnim rečnikom, **\b** omogućava pretragu celih reči korišćenjem regularnih izraza u formi **\ bword\b**. Karakter reči je karakter koji se može koristiti da se formira reč od njega, a svi takvi se mogu dobiti sa skraćenim zapisom klase karaktera **\w**. Svi koji ne spadaju u tu kategoriju se mogu poklopiti sa **\W**.

Primetiti da **\w** obično takođe poklapa i cifre. Kao rezultat, **\b4\b** može da se koristi da se poklopi cifra 4 koja nije deo zapisa drugog broja. Ovakav regex neće poklopiti "44 sheets of a4".

Invertovana granica reči

\B se naziva invertovana verzija **\b**. Ona pokloapa svaku poziciju koju \b neće.

Efektivno **\B** poklapa bilo koju poziciju između dva karaktera reči, kao i bilo koju poziciju između dva karaktera koji to nisu.

Kako to radi?

Pogledajmo primer ako se primeni regex **\bis\b** na stringu “This island is beautiful”. Engine počinje sa prvim tokenom **\b** na prvom karakteru “T”. Pošto je taj token nulte dužine, pozicija pre karaktera se očekuje. **\b** poklapa na tom mestu jer je T karakter iz reči, a karakter pre njega je prazan (void) karakter pre početka stringa. Engine onda nastavlja sa sledećim tokenom koji je literal i, koji se ne podudara sa T, tako da engine pokušava sa prvim tokenom na sledećoj poziciji. **\b** ne može da se poklopi na poziciji između karaktera T i h, ali ni između “h” i “i” takođe, kao ni “i” i “s”.

Sledeći karakter u stringu je space. \b se ovde poklapa jer space nije karakter reči, a prethodni karakter jeste. Engine nastavlja sa “i” koji se ne poklapa sa space karakterom.

Nastavljajući dalje, **\b** poklapa između space karaktera i drugog “i” u stringu. Zatim, regex engine primećuje da **i** poklapa „i” ali i **s** poklapa „s”. Na ovom mestu, engine pokušava da poklopi drugi **\b** na poziciji ispred slova “l”. Ovo će biti neuspešno jer je to pozicija između dva karaktera iz reči.

Neuspešna poklapanja se nastavljaju sve dok se ne stigne do pozicije ispred trećeg “i” u stringu, koji će biti poklopljen sa **\b**. Takođe, **i** poklapa „i” kao i **s** što poklapa “s”. Poslednji token regex-a **\b**, takođe poklapa poziciju ispred drugog space karaktera jer on nije karakter reči, a karakter pre njega jeste. Kao rezultat, engine je uspešno poklopio reč “is” u našem stringu, odbacujući dva prethodna pojavljivanja karaktera “i” i “s”. Da smo koristili regularni izraz **is**, poklopio bi već deo reči “This”.

Korišćenje vertikalne crte za alternativni izbor

Alternativni izbor omogućava poklapanje jednog regularnog izraza od više ponuđenih regularnih izraza, slično kao i sa klasama karaktera.

Ako vam treba pretraga za tekstom cat ili dog, odvojte obe opcije vertikalnom crtom kao u **cat|dog**. Ako je potrebno više opcija, samo proširite listu : **cat|dog|mouse|fish**.

Alternativni izbor ima najniži prioritet od svih regex operatora. To znači da on govori regex engine-u da pokuša da poklopi sve levo od vertikalne crte ili sve desno od vertikalne crte. Ako želite da limitirate doseg alternacije, moraćete da koristite zagrade (obične), za grupisanje. Ukoliko bismo želeli da poboljšamo prethodni regex, u ovom smislu, kako bi uzimao u obzir samo cele reči, to bi bilo **b(cat|dog)\b**. Ovo govori engine-u da pronađe granice reči, onda ili “cat” ili “dog”, i na kraju opet granicu reči.

Već smo pominjali da je regex engine takav da će stati sa pretragom čim pronađe odgovarajuće poklapanje. Kao posledica, redosled alternativnih izbora je značajan. Zamislimo da hoćemo da napravimo regex koji poklapa listu naziva funkcija u programskom jeziku: Get, GetValue, Set ili SetValue. Očigledno rešenje je **Get|GetValue|Set|SetValue**. Kako će ovo raditi na stringu “SetValue”.

Engine počinje sa prvim tokenom u regex-u «G», i prvim karakterom stringa “S”. Poklapanje je neuspešno. Međutim, regex engine je već proučio ceo izraz pre početka obrade, tako da zna da on koristi alternativni izbor, tako da kompletan regularni izraz nije još odbačen. Nastavlja sa drugom opcijom «G», koja je takođe neuspešna. Sledeći token je “S” koji uspeva, engine nastavlja dalje “e” pa “t”, svi se poklapaju sa odgovarajućim karakterima u tekstu koji se pretražuje.

Na ovom mestu, treća opcija je uspešno poklopljena i završava se pretraga. Suprotno onome što smo očekivali, regex nije poklopio ceo string. Postoji nekoliko rešenja za ovaj problem. Prvi podrazumeva zamenu redosleda u alternativnom izboru. Ako koristimo **GetValue|Get|SetValue|Set**, “SetValue” će se pokušati pre «Set», i engine će poklopiti ceo string. Takođe, mogli smo da kombinujemo četiri opcije u dve korišćenjem znaka pitanja: **Get(Value)?|Set(Value)?**. Pošto je ? pohlepan, «SetValue» će biti poklopljen pre nego «Set».

Najbolja opcija je verovatno iskazivanje činjenice da hoćemo samo da poklopimo celu reč. Dakle, **\b(Get|GetValue|Set|SetValue)\b** ili **\b(Get(Value)?|Set(Value)?)\b**.

Pošto obe opcije imaju isti završetak, dodatno možemo optimizovati sa **\b(Get|Set)(Value)?\b**.

Opciona polja

Znak ? čini prethodni token u regularnom izrazu opcioni. Na primer, **colou?r** poklapa string „colour" ali i string „color".

Moguće je napraviti nekoliko tokena opcionim, tako što ih grupišemo zajedno koristeći () i stavljanjem znaka pitanja neposredno iza grupe: **Nov(ember)?** će poklopiti „Nov" i „November".

Moguće je napisati regularni izraz koji poklapa nekoliko alternativnih stringova kombinovanjem više od jednog ?: **Feb(ruary)? 23(rd)?** poklapa stringove „February 23rd", „February 23", „Feb 23rd" i „Feb 23".

Važan koncept u regularnim izrazima: pohlepnost (*greediness*)

Dodajući ?, dodali smo prvi metakarakter koji je pohlepan u smislu pretrage teksta. On daje regularnom izrazu dve opcije: pokušaj da poklopiš deo koji obuhvata ?, ili nemoj pokušati. Engine će uvek, pri tome, pokušati da poklopi i taj fragment. Samo ukoliko će to dovesti do neuspešne primene regularnog izraza u celini, on će pokušati da ignoriše deo teksta koji je označen sa ?.

Efekat je da ako pokušate primeniti regex **Feb 23(rd)?** na stringu "Today is Feb 23rd, 2003", poklapanje će uvek biti „Feb 23rd" a ne „Feb 23". Moguće je napraviti metakarakter "lenj" (lazy) tj. isključiti pohlepnost u vezi sa pretragom teksta dodajući drugi ? iza prvog. Više o tome ćemo reći malo kasnije kada budemo diskutovali pohlepnost kod ostalih operatora ponavljanja.

Da bismo videli kako regex engine radi, pogledaćemo primer primene regularnog izraza **colou?r** u stringu "The colonel likes the color green".

Prvi token u regex-u je literal c. Prva pozicija na kojoj će ona biti poklopljena uspešno je c u reči "colonel". Engine nastavlja dalje, nalazi da **o** poklapa "o", **l** poklapa "l" i drugo **o** poklapa "o". Tada regex engine proverava da li **u** poklapa "n", ovo je neuspešno, ali znak ? govori regex engine-u da to nepoklapanje može biti prihvatljivo. Kao rezultat, engine će pokušati da preskoči na sledeći token: **r**. Ali, ovaj takođe ne uspeva da bude poklopljen sa "n". Nakon ovoga, engine može da zaključi jedino da ceo regularni izraz ne može biti primenjen na ovom mestu, i nastavlja da traži naredni literal "c".

Nakon serije neuspešnih poklapanja, **c** će se poklopiti sa „c" u reči "color", a **o**, **l** i **o** će poklopiti karaktere koji slede. Sada, engine proverava da li **u** poklapa "r", što je neuspešno. Ponovo, to nije problem, ukoliko će nastavak biti ok. Nastavlja sa **r**. On će poklopiti „r" i regex engine predstavlja da je uspešno poklopljen tekst „color" u ulaznom stringu.

Ponavljanje sa znakovima * i +

Nakon prvog predstavljenog metakaraktera za ponavljanje, red je na ostale. Metakarakter * govori engine-u da pokuša da poklopi prethodni token nijednom ili više puta, dok + zahteva poklapanje jednom ili više puta.

<[A-Za-z][A-Za-z0-9]*>

poklapa HTML tag bez atributa. Prvi karakter mora da poklapa slovo, dok drugi karakter može da bude slovo ili cifra. Zvezda nakon njega omogućava ponavljanje drugog karaktera. Pošto smo koristili zvezdu, u redu je da drugi karakter ne bude ništa, odnosno da bude prazan string. Ovakav regex će uspešno prepoznati tag „<B>". Prilikom poklapanja sa „<HTML>", prvi karakter će poklopiti

„H”. Zvezda omogućava drugi karakter da se ponovi tri puta poklapajući redom „T”, „M” i „L” u svakom koraku.

Da li se mogao koristiti regularni izraz **<[A-Za-z0-9]+>** ? Odgovor je ne. Ne, zato što bi u tom slučaju uspešno bio poklopljen i tag „<1>”, koji nije validan HTML tag. Opet, i ovaj regex će biti u redu, ako znamo da string koji pretražujemo neće sadržati ne-validne tagove.

Ograničavajuće ponavljanje

Dodatno, postoji operator za ponavaljanje koji specificira koliko puta token može biti ponovljen. Sintaksa za to je {min,max}, gde je min pozitivan celobrojni podatak koji određuje minimalan broj ponavaljanja poklapanja, dok je max pozitivan broj koji je veći ili jednak sa min i označava maksimalan broj poklapanja. Ako je zarez prisutan, ali max nedostaje, maksimalan broj ponavaljanja je neograničen. Dakle, **{0,}** je isto kao samo *****, a **{1,}** kao **+**. Izostavljanje max granične vrednosti i zareza govori regex engine-u da ponavljanje tokena mora biti tačno min puta.

Mogli bismo koristiti **\b[1-9][0-9]{3}\b** da bismo poklopili broj između 1000 i 9999. Obratite pri tome pažnju granice reči.

Oprez na pohlepnost!

Recimo da želimo da koristimo regularni izraz za poklapanje HTML tag-ova. Znamo da je ulaz biti validan HTML fajl, tako da regularni izraz ne mora da isključuje ne-validne karaktera: sve što stoji između < i > je validan HTML tag.

Većina bi odmah krenula sa **<.+>**.

Međutim, biće iznenađeni kada testiraju na stringu “This is a <EM>first</EM> test”. Iako se očekuje poklapanje sa „<EM>” , a u narednoj pretrazi sa „</EM>”, rezultat će biti „<EM>first</EM>”. Očigledno to nije željeno ponašanje. Razlog za to je taj da **+** čini regex engine pohlepnim: on želi da poklapa tokene, dogod je to moguće. Samo ako će to dovesti do neuspeha celog izraza, tada će se vratiti unazad. Kao i **+**, ***** i **{}** su takođe pohlepni.

Prvi token u regex-u je **<**. To je literal, i kao što znamo, prvo mesto gde će moći da bude poklopljen je prvo „<” u stringu. Sledeći token je tačka, koja poklapa bilo šta osim karaktera za novi red. Ona se ponavlja u + stilu, a pošto je taj metakarakter pohlepan, engine će nastaviti da poklapa tačku dokle god može. Poklopiće „E”, nastaviće dalje i poklopiti i „M”. Sledeći karakter je “>”, ali pošto tačka poklapa i to, engine nastavlja dalje. Na sličan način, poklopiće se svi preostali karakteri iz stringa. Poklapanje tačke će biti neuspešno kada se dođe do kraja stringa. Tek na ovom mestu regex engine ide na sledeći token **>**.

Do sada **<.+** je poklopio „<EM>first</EM> test” i engine je došao do kraja stringa, **>** ne može biti poklopljen ovde, i engine primećuje da je + ponovio . više puta nego što je trebalo (+ zahteva da se poklopi barem jednom). Pre nego što reportira neuspeh, engine će se vraćati unazad. Smanjivaće broj ponavljanja za jedan i pokušavati da nastavi pretragu za preostalim regex tokenom.

Kao rezultat, **.+** se prvo redukuje na „EM>first</EM> tes”. Naredni token u regexu je i dalje **>**, ali sada je naredni karakter u stringu poslednje “t”. Opet, ovo ne može biti poklopljeno i engine nastavlja da se vraća nazad: poklopljeni string postaje „<EM>first</EM> te”. Pošto **>** i dalje ne može da se poklopi, nastavlja se procedura sve dok poklopljeni string ne bude „<EM>first</EM”. Na ovom mestu **>** može biti poklopljen sa sledećim karakterom u stringu i poslednji regex token je uspešno poklopljen time. Engine vraća „<EM>first</EM>” kao preklopljen deo ulaznog stringa.

Zapamtite da je regex engine voljan da vrati poklapanje, prvo poklapanje. Kao rezultat toga, on neće nastaviti da se vraća nazad kako bi video da li postoji "bolje" poklapanje. Zbog pohlepnosti, vraćen je najduži poklopljen string sa leva.

Lenjost umesto pohlepnosti

Brzo rešenje bi bilo da se + načini "lenjim" umesto pohlepnim. Lenji kvantifikatori se nekad nazivaju i nepohlepni (*ungreedy*) ili nevoljan (*reluctant*). Ovo se postiže dodavanjem oznake **?** neposredno iza **+** regex-a.

Isto može da se radi i sa * metakarakterom, vitičastim zagradama kao i samim ?. U našem primeru, regularni izraz postaje **<.+?>**.

Sa ovakvim izrazom, opet **<** poklapa prvi „<" u stringu. Sledeći token je tačka koja treba da se ponavlja sa nevoljnim +. Ovo govori engine-u da ponavlja tačku što je manje moguće puta, pri čemu je minimum jedan. Dakle, engine poklapa tačku sa „E". Zahtev je ispunjen i on nastavlja sa tokenom **>** i karakterom u stringu "M". Ovo je neuspešno. Isto kao i pre, regex se vraća nazad, ali ovog puta vraćanje nazad podrazumeva proširenje opsega "lazy" tačke, umesto skraćivanje poklopljenog stringa. Kao rezultat, **.+?** je proširen na „EM" i engine pokušava da poklopi sledeći token sa sledećim karakterom stringa: **>** sa „>". Ovo će biti uspešno i engine raportira „<EM>" kao uspešno poklopljen string.

Alternativa za nevoljnost

U ovom slučaju, postoji bolja opcija nego da se koristi "lazy" plus znak. Regularni izraz **<[^>]+>** je bolja opcija zbog toga što se ne koristi vraćanje nazad (*backtracking*). Kada se koristi invertovana klasa karaktera, nema vraćanja, koje usporava regex engine, ukoliko se koristi validan HTML kod. Kod jedne pretrage, razlika se neće osetiti, ali hoće ako se primeni na direktorijumu sa mnogo datoteka, razlika je primetna.

() za grupisanje

Stavljanjem dela regularnog izraza unutar zagrada, taj deo izraza se grupiše, što omogućava primenu regex operatora, na primer za ponavljanje, na celu grupu. Već smo dali iznad primer ovoga.

Obratite pažnju da se samo obične zagrade mogu koristiti u ovu svrhu. Uglaste zagrade uvek definišu klasu karaktera, dok vitičaste definišu specijalni operator ponavljanja definisan gore.

Obične zagrade kreiraju reference

Osim što omogućavaju grupisanje delova regularnih izraza zajedno, zagrade takođe kreiraju reference. Reference skladište delove stringa koji je poklopljen delom regularnog izraza koji se nalazi unutar zagrada ().

To znači da će regex engine time biti usporen (osim ako se ne koriste zagrade koje ne kreiraju reference), jer ima više posla da obavi. Ukoliko ne želimo da koristimo reference, možemo ga ubrzati korišćenjem tzv *non-capturing* zagrada, sa cenom malo manje čitljivog regularnog izraza.

Regex **Set(Value)?** poklapa „Set" ili „SetValue". U prvom slučaju referenca će biti prazna, ali u drugom će sadržati „Value".

Optimizacija koju smo gore pomenuli podrazumeva korišćenje **Set(?:Value)?**. ? i : iza otvorene zagrade su specijalna sintaksa koja se koristi da kaže regularnom izrazu da ovaj par zagrada ne treba

da čuva referencu. Obratite pažnju da ? nakon otvorene zagrade nema veze sa ? nakon zatvorene. Ovaj drugi čini prethodni regex token opcionim, a ovaj operator ne može da se nađe nakon otvorene zagrade, jer otvorena zagrada, sama po sebi, nije regularan regex token. Samim tim, nema zabune u ovom slučaju, osim što zapis čini regularni izraz malo nečitljiviji. Operator : daje indikaciju da želimo da isključimo čuvanje reference.

Kako se koriste reference

Reference omogućavaju ponovno korišćenje dela teksta koji se poklopio sa regularnim izrazom i njega možemo iskoristiti unutar ostatka regularnog izraza, ili kasnije, zavisno od alata koji koristite (npr korišćenjem sed alata, može se iskoristiti referenca prilikom *search and replace* funkcionalnosti, npr \1 koji označava prvu referencu prepoznatu).

Da bismo odredili broj reference, gledamo regularni izraz sa leva na desno i brojimo otvorene zagrade. Prva zagrada kreira referencu broj 1, druga broj 2 itd. Pri tome, ne broje se *non-capturing* zagrade, naravno, što donosi još jedan benefit korišćenja ovakvih zagrada: možemo ih jednostavno dodavati u regularni izraz, bez izmene već postojećih dodeljenih brojeva u slučaju kompleksnih regularnih izraza.

Reference se mogu koristiti nakon što je došlo do poklapanja, već takođe i tokom poklapanja. Na primer, zamislimo da želimo da poklopimo par otvorenih i zatvorenih HTML tagova, kao i tekst između. Stavljanjem otvorenog taga u referencu, možemo da iskoristimo ime taga u zatvarajućem tagu. To radimo sa

<([A-Z][A-Z0-9]*)[^>]*>.*?</\1>

Ovaj regularni izraz sadrži samo jedan par zagrada koje "hvataju" string poklopljen sa **[A-Z][A-Z0-9]*** u okviru prve reference. Ova referenca se može koristiti kasnije kao **\1** (backslash pa jedan).

«**/**» označava samo karakter koji se koristi u zatvarajućem HTML tagu.

Možemo više puta koristiti istu referencu. Na primer, **([a-c])x\1x\1** će poklopiti „axaxa", „bxbxb" i „cxcxc".

Referenca ne može biti korišćena unuta sebe same: **([abc]\1)** neće raditi.

Pogledajmo kako gornji regex radi na stringu "Testing <B><I>bolditalic</I></B> text".

Prvi token u regex-u je <. Regex engine prolazi kroz string dok ne dođe do prvog "<" karaktera. Sledeći token je **[A-Z]**. Regex engine takođe uzima u obzir da je to sada već deo zagrada koje kreiraju reference. Token će se poklopiti sa „B". Engine nastavlja na sledeći token **[A-Z0-9]** i ">". Ovo će biti neuspešno. Međutim, zbog zvezde, ovo je ipak prihvatljivo. Pozicija u stringu ostaje na karakteru ">", dok pozicija u regex-u prelazi na **[^>]**.

U ovom koraku se prešlo preko zatvorene zagrade za kreiranje reference, tako da regex engine skladišti sve što je bilo "uhvaćeno" i smešta u prvu referencu. U ovom slučaju „B" je skladišten.

Nakon smeštanja reference, engine nastavlja sa sledećim pokušajem. **[^>]** ne poklapa „>". Međutim, ponovo, zbog * ovo neće biti problem. Pozicija u stringu ostaje ">", a pozicija u regex-u prelazi na **>**. Ovde dolazi do poklapanja, sledeći token je . ponovljena lenjim *. Zbog lenjosti (nevoljnosti), regex engine inicijalno preskače ovaj token, uzimajući u obzir gde treba da se vrati u slučaju da ostatak regularnog izraza bude neuspešan.

Engine je sad došao do drugog **<** u regularnom izrazu i drugog "<" u stringu. Ova dva se poklapaju, sledeći token je **/**, što ne može da se poklopi sa "I", i engine je primoran da se vraća unazad (*backtrack*) do tačke. Tačka poklapa drugi „<" u stringu. * je i dalje "lazy", tako da engine opet pamti da treba da se vraća ako bude potrebe i nastavlja na **<** i "I". Pošto nema poklapanja, odmah se vraća nazad.

Ovo vraćanje se nastavlja sve dok . ne konzumira „<I>bold italic”. Na ovom mestu, **<** poklapa treći „<” u stringu, a naredni token **/** poklapa “/”. Sledeći token u regex-u je **\1**. Engine ne zamenjuje referencu u regularnom izrazu. Svaki put kada dođe do nje, on će pročitati njenu vrednost. Da je engine morao da se vraća nazad i da je prešao preko prvog para zagrada koje kreiraju referencu, pre nego što je pozvana referenca **\1** po drugi put, nova vrednost reference bi bila iskorišćena. Ovde to nije bio slučaj, tako da se „B” zamenjuje. Ovo ne uspeva da se poklopi sa “I”, tako da engine nastavlja da se vraća nazad ponovo i . konzumira treći “<” u stringu.

Vraćanje nazad se nastavlja sve dok . ne konzumira „<I>bold italic</I>”. Na ovom mestu, **<** poklapa „<”, **/** poklapa „/”. Engine opet stiže do **\1**. Referenca i dalje drži vrednost „B”. **B** poklapa „B” i poslednji token u regex-u **>** poklapa „>”. Kompletano poklapanje je završeno i pronađen je string: „<B><I>bold italic</I></B>”.

Reference i ponavljanje

Kao što smo pomenuli ranije, regex engine neće permanentno zameniti reference u regularnim izrazima. Koristiće poslednji poklopljen tekst u okviru reference svaki put kada je korišćen. Ako se novo poklapanje desilo u međuvremenu, prethodno sačuvan sadržaj je prepisan.

Postoji jasna razlika između **([abc]+)** i **([abc])+**. Iako i jedan i drugi uspešno poklapaju „cab”, prvi regex će staviti „cab” u prvu referencu, dok će drugi regex staviti samo „b”. Ovo je zbog toga što u drugom regex-u, plus izaziva par zagrada da se ponove tri puta. Prvi put, „c” je skladište, drugi put „a” i treći put „b”. Svaki put, prethodna vrednost je prepisana, tako da na kraju ostaje „b”.

Ovo takođe znači da **([abc]+)=\1** će poklopiti „cab=cab”, a **([abc])+=\1** neće. Razlog je taj što kada engine dođe do **\1**, on drži vrednost «b» što se neće poklopiti sa “c”. Ovo je očigledno kada se gleda jednostavan primer kao ovaj, ali u kompleksnijim izrazima ovakve stvari neće biti lako primetiti. Kada se koriste reference, uvek treba proveriti da li ste zaista uhvatili ono što ste želeli.

Korisna primena ovoga je kada treba prepoznati dva puta otkucane reči u tekstu. Dok se tekst kuca, lako je napraviti greške kao što je “the the” ili slično. Korišćenjem regex-a **\b(\w+)\s+\1\b** lako možete pronaći takve situacije. Ako se koristi sed alat za search and replace, brisanje druge suvišne reči se vrši tako što se u drugo polje stavi samo \1 (obratiti pažnju da je neophodno staviti -r da bi sed radio sa referencama i da tekst.txt sadrži tekst koji treba da se koriguje):

sed -r -i -e 's/\b(\w+)\s+\1\b/\1/g' tekst.txt

Obične zagrade ne mogu da se koriste unutar klasa karaktera, barem ne kao metakarakteri. Kada se stavi obična zagrada unutar klase karaktera, ona se tretira baš kao i literal. Tako, regex **[(a)b]** poklapa „a”, „b”, „(” i „)”. Takođe, reference ne mogu biti korišćene unutar klasa karaktera. **\1** u regex-u kao što je **(a)[\1b]** će uglavnom biti interpretirano kao oktalna vrednost, dakle regex će poklopiti sa „a” praćeno sa \x01 ili “a” praćeno sa “b”.