**1. Uvod**

Regex predstavlja sablon koji opisuje odredjeni tekst.

Sablon koji se definise sa regex-om obradjuje se od strane regex engina.

Regex engine je softver koji pokusava sa pronadje podudaranje izmedju definisanig sablona i nekog teksta na koji se taj sablon primenjuje. Sam rtegex engine je obicno deo nekog veceg alata, grafickog ili konzolnog, i korisnig ga poziva indirektno od preko tog alata.

Obzirom na postojanje vise razlicitih tipova regex engina nacin na koji oni funkcionisu nije u potpunosti isti kada su u pitanju neki detalji, stoga, jedan regex sablon se moze tumaciti na vise nacina.

**2. Tipovi karaktera**

**2.1. Obicni karakteri (Literal characters)**

Najbazicniniji regex se sastoji od jednog ili vise obicnih karaktera, primer: a, abc, r2d2, C3P0 idt. Prilikom provere, regex engine, ce vratiti liniju teksta nakon sto se pronadje prvo podudaranje sa regex sablonom u toj liniji. Ako je potrebno vrattit liniju teksta tek nakon drugug podudaranja to se mora posebno naglasiti, ovo vazi za vecinu posebnih ulova za pretragu. Po defoltu, regex je case sensitive, "abc" i "ABc" se tumaci razlicito. Case insensitive pretraga se mora posebno naglasiti.

**2.2. Specijalni karakteri**

Specijalni karakteri, ili meta karakteri, dodatno definisu ponasanje regex sablona, postoji 12 specijalnih karaktera: **\, ^, $, ., |, ?, \*, +, (, ), [, {**

**Napomena**: Karakteri poput navodnika, jednosturkih ili dvostrukih i donje crte, nisu specijalni.

Specijalni karakteri ce uglavnom proizvesti gresku ako se koriste sami.

Ako je potrebano koristit bilo koji od specijalnih karaktera u regexu oni se moraju izbeci sa kosom crtom, \, primer:

Regex 10\^2 ce se podudariti sa 10^2 u tekstu.

Karakter poput { se uglavnom tretira kao obican karakter sem ako se ne koriti u formatu npr. karakter{broj,broj}, primer: a{2,5}.

Ostale, obicne, karaktere ne treba izbegavati sa znakom \ jer obican karkter poput "d" sa kosom crtom "\d" ima specijalno znacanje.

Iako regex ima svoja prvila za izbegavanje specilanih karaktera. programski jezik u kojem se regex definise ima i svoja dodatna pravila za definisanje string tipa vrednosti

koja ce tek nakog pravilog kompajliranja biti prosladjena regex enginu.

Primer: Ovo je pravilno definisan regex izraz: 2\+\(3\^3\)= koji trazi sablon 2+(3^3)=

Medjutim u jaziku poput Jave on se mora definisati kao:

String regex = "2\\+\\(3\\^3\\)=";

**2.3. Neispisivi karaketri**

Svaki tekst sadrzi karaktere koji su njegov deo ali koji se ne vide u samom tekstu, npr. karakter/i koji signaliziraju da se prece u novu red i sl.

Neispisivi karakteri:

**\r** - vracanje na pocetak (carrige return).

**\n** - novi red (linija).

**\t** - tab.

**\e** - esc (escape).

**Vazno**: Definisanje prekida linije:

Windows: \r\n

Unix: \n

**3. Ponasanje regex engina**

Iako razliciti regex engini mogu imati znacajne razlike u interpteaciji karaktera i sl. postoje samo dva bazicna regex engina po pitanu osnovnog nacina funkcionisanja:

* **tekst driven engini**
* **regex driven engini** - vecina najpopularnijih regex engina su ovog tipa.

Sam regex sablom moze imati rezultate razlicitog kvaliteta/tacnosti prilikom pretrage iako svi oni zadovoljavaju usam regex izraz. Kakava god da se podudaranja mogu naci u liniji teksta regex engine se uvek zaustavlja na prvom levom pronadjenom podudaranju. Kada nadje prvo podudaranje s leva engine staje sa daljom pretragom, ovo se opisuje kao "regex engine se ZELJAN da prikaze rezultat sto pre".

**4. Skup karaktera, []**

Opsti oblik: [karakter\_1,karakter\_2,karakter\_3.....karakter\_N]

**4.1. Osnovno**

Skup karaktera definise skup karaktera koji se mogu uzeti u obzir priliko podudaranja.

Ako je definisan skup [a,g] to znaci da ce se naci podudaranje na bilo kojoj poziciji u tekstu na kojoj se nalaze karakteri a ILI g.

Unutar skupa sa znakom - moze se definisati opseg karaktera umesto pojedinacno navodjenje svakog, opsti oblik:

pocetak\_opsega-kraj\_opsega

Primer:

0-9 - svi jednocifreni brojevi.

a-z - sva nala slova.

A-Z - sva velika slova.

Primer: Sledeci regex pronalazi podudaranje sa bilo kojim alfanumerickim znakom ILI velikim ili malim slovo (case insensitive):

[0-9a-zA-Z]

Opseg se moze kombinovati sa pojedinacnim karakterima.

Primer: Sledeci regex pronalazi podudaranje sa bilo kojim brojem ILI velikim ili malim slovom A ILI velikim slovom G:

[0-9aAG]

Primer: Sledeci regex pronalazi podudaranje sa bilo kojim velikim slovom, donjom crtom i brojem na kraju:

[A-Z]\_[0-9] -> G\_3, B\_2 .....

**Negacija skupa karaktera**

Definisanjem znaka ^ na samom pocetku skupa karaktera definise da ce podudaranje biti pronadjeno samo za karaktere koji nisu definisani u skupu.

Ako je definisam izraz: \$a\_[^1-5] on se timaci kao svaki tekst koji pocinje sa "$a\_" i koji se zavrsava sa brojem koji nije od 1 do 5 ali jeste od 6 do 9, ukljucujuci nulu.

Specijalni karakeri koji su dozvoljeni u skupovima u vecini regex engina su:

] - otvaranjem skupa sa [, znak ] postaje njegov deo.

\ - izbegavanje drugih specijalnih karaktera.

**Napomena**: Definisanje kose crte kao obicnog karaktera zahteva izbegavanje sa dodatnom kosom crtom.

^ - negacija skupa.

- - definisanje opsega u skupu, ako se koristi bez ogranicenje opsega, primer: [a-] ili [-a] onda znak - ima znacenje obicnog karaketra.

**4.2. Pod skup (Subtraction)**

Podrzano od strane: XML-a, XPath, .NET (ver 2.0 i naredne), JGsoft

Opsti oblik: [skup\_karaktera-[pod\_skup\_karaktera-[pod\_skup\_karaktera-[...]]]]

Omogucava pronalazenje karaktera koji su definisani u skupu ali nisu definisani u pod skupu. Pod skup predstavlja dodatnu filtraciju na osnovni skup karaktera. Pod-skup uvek mora da je definisan na kraju skupa na koji se odnosi i vazi za ceo taj skup.

Primer: Sledeci regex izraz pronalazi sva slova koja nisu samoglasnici.

[a-z-[aeiou]]

Pod skupovi podrzavaju sve opcije kao i standardni skupovi ukljucujuci i definisanje opsega, stim sto se pri interpretaciji definisanog izraza ide od zadnjeg pod-skupa ka osnovnom skupu. Ako je definisan izraz [0-9[0-6[0-3]]] prvo se tumaci pod skup 0-3 od skupa 0-6, sto znaci da se dobija skup od 4-6, zatim se tumaci pod skuo 4-6 od skupa 0-9 sto znaci da je rezultujuci skup brojvi od 0-3 i od 7-9.

Izraz [0-37-9] je ekvivalentan proslom izrazu.

Primer: Pronaci sve jednocifrene brojeve ili mala slova ali ne i velika slova i broj 8:

[0-9a-z-[A-Z8]]

Tumcanje pod-skupa u regex engin-u

U engin-ima koji ne podrzavaju pod-skupove skup izraz poput [a-z-[aeiou]] bi bio tumacen kao sva mala slova ili donja crta ili [ ili neki od samoglasnika (ovo je visak). Ovo znaci da zadnja ] ima znacenje obicnog karaktera.

**4.3. Ukrstanje skupova (Intersection)**

Podrzan od strane: Java i Ruby

Opsti oblik: [skup\_karaktera&&[skup\_karaktera&&[skup\_karaktera[...]]]]

Omogucava pronalazenje karaktera koji se nalaze u oba skupa. Ovaj tip izraza ako se korsiti sa znakom negacije (^) dozvoljava definisanje logicke funkcionalnosti pod-skupa u Javi i sl. obzirom da to nije podrzano direktno od strane regex engina.

Sledeci izraz su ekvivalentni:

[skup\_karaktera&&[^skup\_karaktera]] i [skup\_karaktera-[pod\_skup\_karaktera]]

u regex engine-ima Jave i Ruby-ja ako se ne koristi znak negacije dozvoljeno je definisati skupove bez dodatnih zagrada, ovo ne vazi za grep alat, opsti oblik:

[skup\_karaktera&&skup\_karaktera&&skup\_karaktera$$...]

Primer: Pronaci sve brojeve u rasponu od 4-7:

[0-7&&[4-9]]

Primer: Pronaci sva slova koja nisu samoglasnici:

[a-z&&[^aeiou]]

Napomena: Izbegavati postavljanje znakova && na sam pocetak ili kraj skupa, ponasanje ovih izraza je dosta nepredvidivo i cesto puno bagova za razlicite regex engine.

**4.4. Skraceni skupovi**

Obzirom da se neki skupovi karaktera cesto koriste pri definisanju izraza kreirani su posebni karakeri za brzo predstavljanje ovih skupova.

**Napomena**: Postoje znacajne razlike u detaljima za razlicite regex engine.

\d - predstavlja bilo koji jednocifren broj, [0-9].

\D - sve sem jednocifrenog broja, [^\d].

\w - karaket za reci (word caracter), [a-zA-Z0-9\_].

\W - [^\w].

\s - prazan prostor, [ \t\r\n\f].

\S - [^\s].

\b - predstavlja celu rec, primer \bneka\_rec\b

\B - [^\b]

Skraceni skupovi se mogu koristiti izvan i unutar definisanih skupova.

[\s\d] - prazan prostor ili broj [0-9].

\s\d - prazan prostor pracen brojem [0-9].

**5. Tacka, .**

Tacka kao meta karakter moze da predstavlja skoro bilo koji drugi pojedinacni karakter. Ovo je jedan od najcesce koriscenih meta karakera ali i karakter koji je cesto izvor raznih gresaka u izrazima.

Jedine karaktere koje tacka po defoltu ne moze da zameni su karakteri za prekid linije teksta. Mode rada u kojem tacka moze da menja karakter za prekid linije teksta se zove "single-line mode". Single-line mode ne treba pomesati sa Multi-line modom koji nema veze sa tackom.

Izraz a.b se podudara sa svakim tekstom koji na poziciji izmedju a i b ima alfanumericki znak, a1b, adb, alb, a0b.....

**Napomena**: Tacka se u skupu ponasa kao obican karakter.

Primer: Defnisanje izraza koji predstavlja datum u formatu dd?mm?GG stim sto na mestu znaka ? moze stajati neki od standardnih karaktera za razgranicenje:

\d\d.\d\d.\d\d - podudara se sa 12345678, u ovo slucaju tacka moze da bude bilo sta.

\d\d[-/.]\d\d[-/.]\d\d - podudara se sa 12.45.78 ili 33-45-98 i sl.

[0-3]\d[-/.][01]\d[-/.]\d\d - podudara se sa datuemom tako da broj za dane moze biti u opsegu od 00-39, mesec od 00-19 i godina od 00-99

**6. Fiksiranje pretrage za poziciju (Anchors), ^, $**

**6.1. Upotreba znakova ^ i $**

Anchors sluze za fiksiranje pretrage na odredjenu poziciju, ova pozicija moze biti pre, polse ili izmedju karaktera.

Fiksiranje se postize sa znakom ^ ili $.

Znak ^ odgovara poziciji pre prvog karaktera u stringu. Primena ^a na string abc nalazi podudaranje na prvom karakteru a ali primena ^b na istom karakteru ne nalazi podudaranje jer je karakter b na narednoj poziciji.

Znak $ odgovara poziciji nakon zadnjeg karaktera u stringu. Primena ^c na string abc nalazi podudaranje na zadnjem karakteru c.

Anchors su jako korisna kada je potrebno proveriti da je unos podataka od strane korisnika u odgovarajucem formatu. Ako se zna da unos svakog karaktera (slova i broja), koji kreira neki veci string ili broj, mora biti istog tipa ili deo istog skupa kao i predhodni ali broj pojedinacnih unosa moze biti proizvoiljan i soga je nepozeljno koristiti izraze koji predpostavljaju fiksnu maksimalnu duzinu nekog stringa ili broj cifara u broju, celom ili realnom.

Primer. Proveriti da je unet ceo broj (int):

^\d+$

Ovaj izraz prvo proverava da je prvi broj cifra zatim znak + definise da se mora pojaviti cifra barem jednom ili i da se zatim ceo unos zavrsvaa sa cifrom. Ovaj izraz "brani" program od unosa koji u sebi sadrze space.

Primer. Proveriti da je unet realan broj (double, float):

^[\d.]$

Razlika naspram proslog izraza je ta sto je sad dozvoljen unos tacke, medjutim nije specificirano da se tacka moze pojaviti samo jednom, ovo je mana ovog resenja.

Primer. Pretraga pojave pocetnog ili zadnjeg (na kraju stringa) praznog prostora.

^\s+ i \s+$

Multi-line mode

Problem pri koriscenju Anchors-a nastaje kada se proverava jedan string koji je sastavljen od pojedniacnih linija odvojenih sa \n znakom iako sam string formira jednu liniju. Pri defolnom radu znak ^ predstavlja poziciju na pocetku stringa a znak $ na njegovom kraju a ne na kraju prve linije. Da bi se ovakvi stringovi obradjivali liniju po liniju mora se aktivirati Multi-line mode.

**Napomena**: Multi-line mode nema veze sa Single-line modom koji se tice rada sa **.** znakom, tacnije, omogucavanje znaku **.** da zameni karaktera za kraj linije poput \n i \r.

Multi-line se mora posebno aktivirati. Metodi nekog programskog jezika koji podrzavaju regex izraze cesto imaju dodatni parametar gde se moze proslediti konstanta koja aktivira Multi-line mode rada.

Pravilan nacin za tumacenje izraza poput ^a$ je taj da karaktera mora da se nalazi na pocetku stringa i pre bilo kog znaka za novu liniju, drugim recima ovaj izraz ce se podudariti sa stringom a\n ili u sklopu veceg stringa abcd\nga\na\n sa poslednjim a u stringu, preposlednji karakter.

Prvo a u stringu abcd\nga\na\n nije praceno karaketrom za kraj linije vec karakterom b dok drugo a nije na pocetku linije.

**6.2. Upotreba \b (word boundary)**

\b karaker takodje predstavlja pozicioni karakter ali on sluzi za definisanje pretrage za celim recima.

Ako se definise izraz \bJava\b on ce naci podudaranje u tesktu, npr "Programski jezik Java i C++."

Obratiti paznju da izraz rec predstavlja skup karaktera koji predstavljeju karakere za reci i dobijaju se preko \w. Vazno je zapamtiti da svi karateri nisu i karakteri za reci.

**7. Alternacija (OR pretraga), |**

Opsti oblik: regex\_izraz\_1|regex\_izraz\_2|regex\_izraz\_3.......

Alternacije predstavlja logicku fukciju OR pretrage sa razlicitim regex izrazima. Za definisanje se koristi znak | sto je dosta slicnao kako i u programiranju. Alternacija ce prikazati tekst ako se nadje podudaranje sa bilo kojim od definisanih regex izraza, drugom recima dovoljno je jedan da bude tacan.

Primer. Izraz Java|PHP pronalazi podudaranje u tekstu jer je drigi regex izraz tacan.

U razvoju web aplikacija se cesto koristi PHP programski jezik.

ali i u izrazu

Za rad sa MySQl bazom se cesto koriti PhpMyAdmin alat.

Primer. Da bi se pretraga ogranicila na cele reci mora se koristiti \b (word boundary), primer:

\b(Java|Php)\b

Zagrdae u ovom slucaju predstavljaju nesto poput ogranicenje opsega, bez njih, izraz bi se tumacio kao pocetak reci pracen sa JAVA, ILI, Php pracen sa krajem reci. Sa zagradama izraz se tumaci kao cela rec, JAVA ili Php.

Bitno je uzeti u obzir karakteritiku regex engina da je on "ZELJAN" da vrati rezultat sto pre. Ako imamo izraz:

get|getName|set|setName

kojim zelimo da pretrazimo kod u potrazi za nekim metodom. Pri prvom nailasku na tekst poput "getName", prvi regex izraz "get" ce javiti rezulata jer je pronasao podudaranje iako bi bilo idealno da se to uradi sa izrazom getName.

Jedan nacin da se to resi je da se uzme u obzir zelja engina da sto pre nadje rezultat i da se izmeni redosled izraza u:

getName|get|setName|set

Na ovan nacin opcije getName i setName ce se pokusati pre get i set. Postoji i opcija da se ovaj izraz optimizuje sa znakom ?, primer:

get(Name)?|set(Name)?

Znak ? daje opciju da se "Name" i moze a i ne mora pojaviti sto je isto kako i getName|get ili setName|set, narocito kada se uzme u obzir da je znak ? "POHLEPAN", odnodno, prvo ce pokusati sa getName pa tek onda sa get ili prvo sa setName pa tek onda sa set.

Pod uslovom da je to tacno, najboli nacin da se ovo resi je da se definise da su pojmovi get, getName, set i setName cele reci:

\b(get|getName|set|setName)\b

Razlika izmenju teksualnih i regex engina

Alternacija je mesto velike razlike izmedju teksualnih i regex engina. U izrazu get|getName|set|setName, regex engine ce vratiti rezulatat nakon prvog pojavljivanja pojma get ili set. Obzirom da text driven engine analizira ceo string pre donosenja odloke o vracanju rezultata, on zna, da ako u tesku postoji pojam setName, i izrazi set i setName ispunjavaju uslov.

Text driven engine se primoran da bira izmedju ova dva pojma ali po defoltu je podesen da uvek vrati duze podudaranje kao tacnije i stoga ce podudaranje biti na pojmu setName a ne na set.

**8. Opciona pretraga (?,{})**

**8.1 Znak ?**

Opsti oblik: karakter? ili (skup\_karaktera)?

Znak ? cini karakter ili pod-skup ili grupu karaktera pre njega opcionim po pitanju pretrage. Znak ? se naziva i kolicinska direktiva, u vom slucaju kolicina je 0 ili 1.

Bitna stavka znaka ? je to sto je on "POHLEPAN" u nacinu na koji funkcionise jer ce se prvo pokusati sa pronalazenjem karaktera koji su definisani pre njega a zatim bez tih karaktera u stringu.

Ako je definisan izraz Jan(uar)? 23 i pretrazuje se string "Danas je Januar 23, 2018 god." prvo ce se pronaci podudaranje sa pojmom "Januar 23" a ne sa "Jan 23".

"Pohlepnost" znaka ? se moze iskljuciti i tada on postje "LENJ", ovo se postize sa dodatnim nzakom ?, primer:

Jan(uar)??

u teksu: "Danas je Januar 23, 2018 god."

Nalazi podudaranje sa pojmom: Jan 23

Dodavanje znaka ? je nacin za prebacivanje svih "POHLEPNIH" i opcionih metoda pretrage u "LENJI" rezim rada, ovo vazi i za upotrebu znakova \* i +.

**8.2. Znakovi {}**

Opcina pretraga se moze postici i sa zakrivljenim zagradama, { i }.

Opsti oblik: karakter{minimum, maksimum} ili (izraz){minimum, maksimum}

* minimum - ceo pozitivni broj i nula, minimalni broj pojavljivanja karaktera da bi se pronaslo podudaranje, 0 znaci da karakter i ne mora da se pojavi u stringu.
* maksimum - ceo pozitivni broj, opciono, maksimalni broj pojavljivanja karaktera da bi se pronaslo podudaranje.

Da bi se simulirala funkcija znaka ? potrebno je definisati karater/skup{0,1}, ovime se definise da karakter moze sa pojavi samo jednom ili ga i ne mora biti.

Primeri:

a{0,} - a se moze pojaviti 0 ili vise puta: bcdefg, aaaaaa, a

a{1,} - a se moze pojaviti 1 ili vise puta: a, aaabcd, Polaaaaaako

Kao i znak ?, znakovi {} se mogu primeniti na jedan karakter ili na citav skup karaktera.

**9. Ponavljanje (\*,+)**

Kao i znak ?, zakovi \*,+ definisu da se nesto moze pojaviti vise puta, samo jednom, ili nijednom sto sve spada pod logiku opcione pretrage.

**\*** - definise da se neki karakter ili skup moze pojaviti nula ili vise puta, opsti oblik:

karakter\* ili (skup\_karaktera)\*

**+** - definise da se neki karakter ili skup mora pojaviti barem jedanput ili vise puta, opsti oblik:

karakter+ ili (skup\_karaktera)+

Izraz karakter\* je isto kao i karakter{0,}.

Izraz karakter+ je isto kao i karakter{1,}.

**Napomena**: Oprezno koristiti mogucnosti znakova za ponavljanje jer mogu usled visestrukih provera da znacajno opterete CPU.

Primer: U HTML kodu ponaci pojavljivanje prvog taga koji pocinje sa B ili ne mora da ima nista nakon toga:

<B[a-zA-Z0-9]\*>

ili, pojavljivanje prvog taga koji pocinje sa B ali mora da ima nesto nakog toga:

<B[a-zA-Z0-9]+>

Ono sto je bitno zapamtiti je to da su ovi znaci u vecini slucajeva po pitanju funkcionisanja i "ZELJNI" da sto pre vrate rezultat i "POHLEPNI", jer ce prvo pokusati sa opcijom da karaketera ima pa tek onda da ih nema, u slucaju \* ili da se pojavljuje samo jedanput, u slucaju +.

**Funkcionisanje engina**

Ako definisemo izraz <.+> u nastojanju da nadjemo prvo pojavljivanje nekog HTML taga u teksu:

HTML tag <h2>naslov</h2> predstavlja naslov sa tekstualnim sadrzajem.

Rezultata ovog izraza je podudaranje sa pojom **<h2>naslov</h2>** a ne samo **<p>**.

Nakon sto engine naidje na prvo pojavljivanje karaktera < u tekstu i uspesno uporedi to sa prvim delom regex izraza (<), prelazi na sledece znakove u izrazu. Sledeci znak je . pracena sa +, sto predstravlja bilo koji karakter, iz ovog razloga engine ce naci podudaranje sa svim karakerima do kraja recenice. Engine nece stati ne prvoj pojavi karaktera > jer podudaranje sa .+ jos nije gotovo, ne zaboraviti da je + "POHLEPAN".

Tek nakon sto engine dodje do kraja stringa, napustice pretragu sa .+ jer zna da i dalje ima deo izraza koji nije proveren i krenuce unazad, "BACKTRACKING", sa sledecim delom izraza, >, u nastojanju da nadje poklapanje.

Bitno je primetiti da je engine svestan da usled postojanja dela u izrazu .\* one je mozda presao preko karaktera koji je mogo biti pronadjen sa narednim delom izraza (>). Stoga, engine sada ide u nazad pokusavajuci da nadje podudaranje sa >, obzirom, da je engine "zeljan", naci ce podudaranje sa prvom pojavom znaka >.

**"LENJI" rezim rada (samo kod regex driven engina)**

Prebacivanje u "LENJI" rezim rada znaci da ce znak + prvo proveravati da se neki karaker pojavio barem jedanput i ako jeste to ce zadovoljiti pretragu i preci ce se na sledeci deo izraza.

Primena izraza: <.+?> na string:

"HTML tag <h2>naslov</h2> predstavlja naslov sa tekstualnim sadrzajem."

Pronalazi se podudaranje sa pojmom **<h2>**, sto se i ocekuje.

Prvi karakter sa kojim se nalazi podudaranje iz izraza je <, zatim se prelazi na .+?. Ovaj deo izraza proverava da li postoji barem jedan karaker (sto manje) nakon <, karakter koji je nadjen je h ovime se pretraga prekida i prelazi se na sledeci deo izraza, >. Deo izraza koji trazi porudaranje sa > nailazi na karaker 1 u tekstu i on prekida pretragu usled neuspaha da se nadje podudaranje. Engine sada ide u nazad kroz izraz pokusavajuci na ispuni uslov da se moze naci vise od jednog karaketra bilo kog tipa pre pojave karaktera > u stringu.

Ne zaboraviti, engine zeli da nadje podudaranje u okviru zadatog izraza. Primenom .\*? na karaker 1 pronalazi se podudaranje i ponovo se prelazi na pretragu sa >. Sa > se ponovo proverava sledeci karaketr u stringu i ovaj put se pronalazi poklapanje. Jasno je da kada bi se radilo o puno vecem pojmu, broj konstantog premestanja bi bio velik i trosilo bi se vreme kao i pri pretrazi unazad.

**Text VS Regex driven engines i Backtracking**

Uvodjenje "LENJIH" kvantitativnih direktiva resava problem "POHLEPE" istih ali ne resava problem da engine ide unazad da bi pronasao odgovarajuci pojam. Backtracking engina prilikom kompleksnih pretraga u sklopu petlje moze znacajno da optereti CPU. Text driven engini nemaju problem Backtracking-a ali zato ne podrzavaju "LENJI" rezim rada.

**POHLEPA i negacija nasuprot LENJEM rezimu rada**

Da bi se resio problem backtracking-a, zajedno sa znakom + u defoltnom stanju ("Pohlepnost") moze da se koristi negacija. Sledeci izraz resava problem backtracking-a a ne zahteva "LEJNI" rezim rada:

<[^>]+>

Do god je HTML tag u nekom teksu validan ovaj izraz ce pronaci njegovo prvo pojavljivanje. Kao i ranije prvo se uporedjuje karakter < u izrazu i sa njim se nalazi podudaranje. Sledeci deo izraza, [^>]+, pronalazi sve sto NIJE karakter >, sto znaci da ce sa uporedjivanjem stati kada se dodje do karaktera >. Zadni deo izraza, >, pronalazi podudaranje sa zavrsnim delom HTML taga. Sa ovim izrazom nema potrebe da se koristi "LENJI" mode a ne postoji ni backtracking.

U kombinacij sa tackom zakovi \* i + ce naci podudaranje sa bilo kojim karakterom, primeri:

Izraz: [Bb].\*a ce naci podudaranje sa svim dole navedenim stringovima:

Beograd - Beogra

Beogradska - Beogra

Vozio sam bicikli kroz Beograd - bicikli kroz Beogra

**Napomena**: Problem "pohlepe" i backtracking-a je ovde i dalje prisutan.

Znakvi ., \* ili + i | se mogu koristit za postizanje AND tipa pretrage, odnosno provera da se dva pojma nalaza u nekom tekstu.

Opsti oblik: izraz\_1[^\(izraz\_kontr\_1\)]\*izraz\_2|izraz\_2[^\(izraz\_kontr\_2\)]\*izraz\_1

**Vazno**:

1. Nije dovoljno da se definise izraz izraz\_1.\*izraz\_2 jer ce se time iskljucivo pokriti opcije da izraz\_1 ide pre izraz\_2. Zato se mora uvesti alternacija sa znakom | kao bi se pokrila i druga opcije, da izraz\_2 ide pre izraz\_1. Ovime se dobija prava AND pretraga.
2. izraz\_kontr\_1 je neophodan jer bez njega postojao bi jako velik problem nepotrebnog backtracking-a. Ovaj izraz moze biti jednak izrazu izraz\_2 ali radi
3. optimizacije i ako je izraz izraz\_2 dosta velik, za izraz izraz\_kontr\_1 se moze defnisati neki straceni jedinstaveni pojam koji je se sigurno nece pojaviti izmedju izraza izraz\_1 i izraz\_2. izraz\_kontr\_2, kao i izraz izraz\_kontr\_1 ali suprotno.

**10. Grupisanje, ()**

Opsti oblik: (izraz)

Smestanjem regex izraza unutar zagrada moze se grupisati taj izraz kako bi se nad njime izvrsaila neka druga operacija.

Primer: Pronaci pojavu imena Jovan, Marko ili Milos uneko mtekstu: (Jovan)|(Marko)|(Milos)

Primer: Izraz set(Value)? odgovara stringu set i setValue, stim sto ce znak ? usled "pohlepe" prvo proveriti setValue.