Expressões Regulares

Uma expressão regular (ER) é um método formal de se especificar um padrão de texto. É um conjunto de símbolos e caracteres que quando analisados determinam uma regra. A este conjunto chamamos padrão. Depois, verifica-se um texto maior para procurar pedaços dele que obedeçam à regra criada no padrão. Ao texto maior, chama-se texto. O objetivo é localizar (contar, substituir,...) ocorrências do padrão dentro do texto. Qualquer editor de textos faz isso, por exemplo quando você tem uma longa carta e manda o editor trocar 'Curitiba' por 'São Paulo'. Ao usar expressões regulares, tal processo ganha inteligência e abrangência. Por exemplo, com o padrão = ((su|hi)per)?mercado, você vai recuperar todas as ocorrências das palavras "mercado", "supermercado"e "hipermercado".

Histórico

As expressões regulares começaram na teoria da linguística, quando o matemático Stephen Kleene descreveu o que ele chamou de *regular sets* e sugeriu alguns dos símbolos que usamos até hoje. Em 1968, criou-se um editor de textos (*ed*) que passou a fazer buscas usando expressões regulares. Em 1986, apareceu um pacote C livre, chamado regex, que permitia incluir o manuseio de ER dentro de qualquer programa. Finalmente o WORD, a partir da versão 2000 aceita usar ER (devidamente modificadas pela Microsoft, já que ela sempre consegue piorar algo no que mete a mão) em buscas e trocas.

Metacaracteres

Para construir o padrão, usamos alguns metacaracteres que controlam a busca do padrão. Eis os principais

caracter	mnemônico	função
.	ponto	um caractere qualquer
[ ]	lista	conjunto de caracteres aceitos
[^]	lista negada	conjunto de caracteres rejeitados
?	opcional	0 ou 1 ocorrência
*	qualquer	de 0 até qualquer ocorrência
+	pelo menos 1	1 ou mais ocorrências
{n,m}	de...até	de n até m
^	início	início da linha
$	fim	final da linha
\b	borda	início ou final da palavra
\c	escape	torna literal o caractere c
|	ou	ou um ou outro
( )	grupo	delimita um grupo
\1 a \9	retrovisor	texto casado nos grupos 1 a 9

. É o maior coringa. O ponto casa com qualquer outro caractere, inclusive com o caractere ponto. Veja no exemplo:

```
n.o          não, nao, neo, noo,...
ca.a         casa, cata, capa,...
e.tendido    estendido, entendido, extendido,...
```

lista

A lista é um conjunto de caracteres escritos entre chaves. Ela descreve os caracteres que farão parte do padrão (usando OU). Veja

```
n[aã]o       não, nao - mas deixa de fora o noo
17[:.h ]20   17:20, 17h20, 17.20, 17 20
<[BIP]>      <B>, <I>, <P>
```

Note que a especificação `[0123456789]` pode e deve ser trocada por `[0-9]`. Usa-se também `[a-z]`.

Já a lista negada tem a mesma lógica, mas com o conectivo lógico ∼ na frente. Aqui a negação é escrita com ^. Então, `[^a-z]` é qualquer caractere que não seja uma letra minúscula.

Opcional

Usa o símbolo `?` que significa 0 ou uma vez o que vem antes. Então, os exemplos:

```
bate[ri]?     bater, batei, bate, mas não bateu
</?[BIP]>     <B>, </B>, <I>, </I>, <P> ou </P>
```

Tanto faz

O tanto faz é indicado pelo `*` e ele indica zero vezes, uma vez, ou um monte de vezes o que vem antes.

```
5*0          0,50,550,55555555550,...
va*t         vt, vat, vaaaaaat,...
b[hj]*k      bk, bhjhhhk, bhjhjhjjjjk, ...
```

Note que `.*` significa qualquer coisa em qualquer quantidade. Então, a pesquisa `prazo.*semana` vai procurar a linha de texto em que alguém prometeu terminar no prazo máximo de 3 semanas.

Pelo menos um

Identificado pelo símbolo `+` funciona igual ao `*`, mas tem uma diferença: o que vem antes do `+` tem que ocorrer pelo menos 1 vez. Compare:

```
5+0          50,550,55555555550,...
va+t         vat, vaaaaaat,...
b[hj]+k      bhjhhhk, bhjhjhjjjjk, ...
```

Controle {n,m}

É uma quantificação mais controlada, onde `{n,m}` significa um quantificador de no mínimo n e no máximo m. Por exemplo `3{2,4}` equivale a 33, 333, 3333 e só. Acompanhe os exemplos

```
{2,3}     de 2 a 3 vezes
{4,}      pelo menos 4 vezes
{0,3}     até 3 vezes
{4}       exatamente 4 vezes
{1}       1 vez
{0,1}     0 ou 1 (igual ao ?)
{0,}      zero ou mais (igual ao *)
{1,}      um ou mais (igual ao +)
```

Âncora início

Usa o caractere ^ (fora de `[]` – dentro é a negação da lista). Indica que o casamente deve ocorrer NO INÍCIO DA LINHA. Exemplo: `^[0-9]` significa case um número no começo da linha. Outro exemplo `^[^a-z]` significa case qualquer linha que não comece com uma letra minúscula. Mais um: `[0-9]^`: Como não tem sentido haver algo antes do início da linha (se tiver algo antes do início da linha, então não é o começo da linha...) Neste caso o ^ é exatamente o caractere circunflexo. Para terminar: `^^` acha todas as linhas que começam com ^.

Âncora fim

O símbolo é `$` e sinaliza o final de linha. Exemplo: `[0-9]$` casa com linhas que terminam por um número. Outro: `^$` significa uma linha vazia. Mais: `^.{10,50}$` casa com linhas que tenham entre 10 e 50 caracteres.

Borda de palavra

Marca os limites de uma palavra. Seu símbolo é `\b`. Palavra engloba apenas `[A-Za-z0-9_]`, caracteres fora daqui não fazem parte de palavras. Veja os exemplos

```
dia       dia, diafr, radial, melodia, meio-dia!
\bdia     dia, diafr, meio-dia!
dia\b     dia, melodia, meio-dia!
\bdia\b   dia, meio-dia!
```

Escape

Escapa-se um caractere (escrevendo uma barra invertida antes do caractere) quando o mesmo deve ser procurado literalmente. Por exemplo, suponha que você precisa procurar literalmente `viva*`. Se escrever `viva*` isso vai casar com `viv`, `viva`, `vivaaaaa`, cadê o asterisco ? Tudo errado. Um jeito aceitável seria `viva[*]`, mas fica esquisito ter uma lista de um único caractere. A melhor solução é escapar o `*`. Vale para qualquer metacaractere, inclusive a \ que neste caso deve ser escrito como `\\`.

Ou

Partes alternativas do padrão. Por exemplo `aqui|ali` casa com o padrão `aqui` ou com o padrão `ali`. Lembre que `[ab]` casa com `a` ou com `b`, mas dentro de uma lista o OU funciona apenas para 1 caractere. Se o padrão tem mais de 1 caractere, você precisa usar o `|`. Compare `[gtr]ato` equivale a `gato|tato|rato`.

Grupo

Escrito entre parênteses, serve para juntar coisas. Por exemplo: `(oi)+` casa com `oi`, `oioi`, `oioioioi` e assim por diante.

```
(\.[0-9]){3}      .0.5.4, .2.1.1, .3.3.3, ...
(www\.)?aa.com    www.aa.com, aa.com
(in|con)?certo    incerto, concerto, certo
```

Retrovisor

O retrovisor usa o caracter \. Logo a barra invertida escapeia (que palavra feia, e a propósito, inexistente) qualquer caractere EXCETO se for um número de 1 a 9. Então `\1` é o último pedaço do texto que casou. `\2` é o penúltimo, `\3` é o ante-penúltimo e assim por diante até o nono último. Veja o exemplo: `(binho)-\1` busca o casamento de `binho-binho`. Outro: `([A-Za-z]+)-?\1` casa com `quero-quero`, `bibi`, `dudu`, `bombom`... . Como lembrete final, o retrovisor referencia o texto casado e não a ER do grupo.

Metacaracter tipo barra-letra

Além daquela tabela da linguagem C (`\a` = bipe; `\n` = newline, `\b` = backspace, ...) criaram-se novos conjunto barra alguma coisa, a saber

\d	dígito
\D	não dígito
\w	palavra
\W	não palavra
\s	espaço
\S	não espaço

Tem uma sequência enorme de barra-alguma coisa. Cada aplicativo|linguagem aplica os seus. Confirme antes de usar!

Exemplos

Seja um texto
```
123456789012345678901234567890123456789012345678901234567
HDHEIJCEIGCDAACBFCEBBFGBEBJGFHEEIJGIFBEFJGFEACE
```
Com um padrão igual a `HE+IJ` Suas ocorrências serão em 3 30.

```
123456789012345678901234567890123456789012345678901234567
EHGBFJIGDDJHJGEGBCIGBFFFGCIBIEGDJFCACCCIFGBAHHJ
```
Com um padrão igual a `CIF?GB` Suas ocorrências serão em 18,39.

```
123456789012345678901234567890123456789012345678901234567
AJFEEHICJFCIAEIJDIIICBCBIFBGJGIJBCCCBAICFBGBDDF
```
Com um padrão igual a `B[C|C|G]+B` Suas ocorrências serão em 22 33 42.

☞ Para você fazer

1. Ache onde o padrão ocorre no texto:
```
123456789012345678901234567890123456789012345678901234567
HIICIFBGBAIBDCDJEJCEGFCHCCBFEIECIJBABBDCDEICIHJ
```
com o padrão =
```
B+DCD
```

2. Ache onde o padrão ocorre no texto:
```
123456789012345678901234567890123456789012345678901234567
JAJGHEBIHCFHCBHAAHJABEIHDCCJFCHJAAHFAAHAGJHJCDI
```
com o padrão =
```
AAH
```

3. Ache onde o padrão ocorre no texto:
```
123456789012345678901234567890123456789012345678901234567
IFIGAGECFCCEFHFBBDEDDDJIFJHCEHIJBCEIBAEAFHFBIBC
```
com o padrão =
```
.HFB
```

4. Ache onde o padrão ocorre no texto:
```
123456789012345678901234567890123456789012345678901234567
GJBIBBDCGEEIAIGJIHCEIFIIHGFCEJIFBAJBIAIHAIJECBJ
```
com o padrão =
```
CEJ?IF
```

1	2	3	4