Primeiro Trabalho de CT-200: Um Tokenizador para Textos por Expressões Regulares

Carlos Henrique Q. Forster

23 de março de 2009

1 Introdução

Muita informação se encontra perdida na forma de arquivos de texto semi-estruturados. Sem uma estruturação adequada é bastante difícil realizar algumas buscas, especialmente se não houver padrões para representar dados complexos como datas. Datas consistem em três campos: dia, mês e ano. A mesma data pode ser representada de diferentes formas:

- 23-Jun-1912
- June, 23rd 1912
- 23/06/12

Felizmente, neste caso, essas várias formas de representação seguem padrões próprios que podem ser especificados através de expressões regulares. Seria interessante ter uma ferramenta que marcasse em arquivos de texto as porções que correspondem a determinados padrões.

2 Tarefa

Sua tarefa consiste em construir um aplicativo em C ou C++ para localizar as ocorrências de cadeias em arquivos de texto que casem com uma expressão regular específica fornecida pelo usuário. O programa deve gerar como saída um novo texto, cópia do original, mas com o início e o fim das cadeias marcados com rótulos (tags). O rótulo é da forma "<X>cadeia</X>" onde o texto "X" é fornecido pelo usuário.

O programa deve ser chamado da linha de comando e utilizar a seguinte sintaxe:

```
programa nome_da_tag "expressao_regular"
```

O programa deve então ler a entrada padrão (stdin) e produzir o resultado na saída padrão (stdout).

```
programa nome_da_tag "expressao_regular" <entrada >saida
```

Dessa forma, o programa se torna mais útil permitindo a conexão da entrada de uma chamada com a saída de outra através do mecanismo de piping existente nos sistemas operacionais.

Suponha que o programa seja entregue como o arquivo “tokenize.c”. O programa será compilado através da linha de comando:

```
gcc -o tokenize tokenize.c
```

O seguinte comando seria utilizado para anotar as datas de um texto:

```
./tokenize data "(9+_)9/99/(99+_)99+(9+_)9-Zzz-
  (99+_)99+(Z+z)*, (9+_)(1st+2nd+3rd+9th) (99+_
  )99" <news.txt >anews.txt
```

Para um arquivo de entrada news.txt com o conteúdo:

```
An important date is June, 23rd 1912
also known as 23/06/12. A wrong
representation is 23/23/06/12.
```

A saída no arquivo anews.txt deve ter como o conteúdo:

```
An important date is <data>June, 23rd 1912</data>
also known as <data>23/06/12</data>. A wrong
representation is <data>23/23/06<data>/12.
```

Veja no exemplo que uma vez identificado o fim de uma cadeia como aceita pela expressão regular, deve-se iniciar a busca da próxima cadeia a partir do fim desta. Não se deve localizar cadeias em sobreposição! Perceba também no exemplo a ordem de precedência dos operadores: Kleene star, concatenação e união.

Especificamente, seu programa deve:

1. Interpretar a expressão regular e construir um autômato correspondente.
2. Simular o autômato tendo o texto como entrada.
3. Produzir na saída texto anotado com rótulos no início e fim das sub-cadeias relacionadas às expressões regulares.

Melhorias **opcionais**, mas que serão bem vistas:

1. Ler o texto de entrada uma única vez sem armazená-lo por inteiro.
2. Gerar um autômato com mínimo número de estados.
3. Aceitar qualquer nível de aninhamento de parênteses.

3 Especificação da Expressão Regular

As expressões regulares devem utilizar os operadores e os conjuntos descritos a seguir:

(a)	agrupar sub-expressões
a+b	operador de união
ab	operador de concatenação
a$*$	a estrela de Kleene
z	qualquer letra minúscula
Z	qualquer letra maiúscula
9	qualquer dígito de 0 a 9
_	a cadeia vazia
espaço	um caractere espaço em branco, tab, fim-de-linha
outros dígitos	apenas o dígito indicado
outras letras	apenas a letra indicada
outros símbolos	apenas o símbolo indicado
'x'	exatamente o caractere x (mesmo se for zZ9()+$*$ ou _)
'xyz'	exatamente a cadeia xyz
''	o caractere '

4 Requisitos

Os seguintes requisitos são mandatórios para a tarefa:

1. O aplicativo deve poder ser compilado com o gcc ou o g++ e não deve fazer uso de bibliotecas adicionais à libc. A entrada e saída devem ser feitas com stdio (ou o equivalente em C++).
2. O aplicativo pode ser composto de vários arquivos fonte, porém deve ser fornecida a linha de comando para compilação do programa (não utilizar make).
3. Tudo o que for utilizado de terceiros deve acompanhar a devida citação e deve-se deixar bem claro o que consiste contribuição do grupo de trabalho.
4. O programa deve estar adequadamente documentado com comentários.
5. A entrega deverá ser feita pelo sistema Tidia.

5 Relatório

Seu relatório deve ser um texto completo (introdução, conclusão etc). No seu relatório deve constar a descrição da tarefa com suas próprias palavras, a solução proposta e implementada, a avaliação dos resultados e suas experiências pessoais com o trabalho. Sempre fazer referência a trabalhos de terceiros que foram utilizados no seu relatório e até mesmo a informações obtidas de outras pessoas.

6 Avaliação

Será passado um conjunto de testes com 5 instâncias e haverá um outro conjunto de testes do mesmo tamanho, só que secreto. É desejável que o aluno teste o programa com conjuntos de teste desenvolvidos por ele próprio também.