Universidade Federal de Pernambuco
Centro de Informática

IF767 - Processamento de Cadeias de Caracteres 2014.2
Prof. Paulo Fonseca

Projeto 2 — versão 1 (25/01/2015)

- Este documento contém as regras e diretrizes para o segundo projeto. Leia com atenção todo o conteúdo do documento e tente ater-se às orientações o mais fielmente possível.
- As regras abaixo podem ser modificadas a qualquer tempo pelo professor no melhor interesse acadêmico e didático. As modificações serão comunicadas em tempo útil através do grupo de discussão da disciplina.
- Eventuais omissões serão tratadas de maneira discricionária pelo professor, levando-se em conta o bom senso, a praxe acadêmica e os interesses didáticos.

Objetivo

Neste projeto deve ser desenvolvida uma ferramenta para indexação, armazenagem e busca de padrões num arquivo texto. O objetivo é de consolidar o conhecimento dos algoritmos vistos no curso através da implementação de um software com correção, documentação e escalabilidade em nível de produção.

No que se segue, chamaremos a ferramenta de **`ipmt`** (*Indexed Pattern Matching Tool*).

Funcionamento básico

A ferramenta deve ter uma interface em linha de comando (*command line interface—CLI*) seguindo as diretrizes GNU/POSIX [1].

A ferramenta deve suportar dois modos:

1. Modo de indexação
2. Modo de busca.

Modo de indexação

No modo de indexação, o objetivo é produzir um índice completo a partir de um texto de entrada que poderá ser usando posteriormente para casamento *offline* exato de padrões. Este modo deve ser acionado através do comando

`$ ipmt index [opções]` ***`textfile`***

que fará com que seja produzido um índice a partir do arquivo texto *`textfile`*. Este índice deverá ser armazenado num arquivo com mesmo nome base do arquivo texto acrescido da terminação `.idx`.

Exemplo

```
$ ipmt index moby-dick.txt
```

[1] `https://www.gnu.org/prep/standards/html_node/Command_002dLine-Interfaces.html`

deve produzir um arquivo

`moby-dick.idx`

O arquivo de índice de saída deve ser gerado em formato *comprimido*, de forma a reduzir o espaço necessário para armazenagem. Assim, temos o esquema da Figura 1.

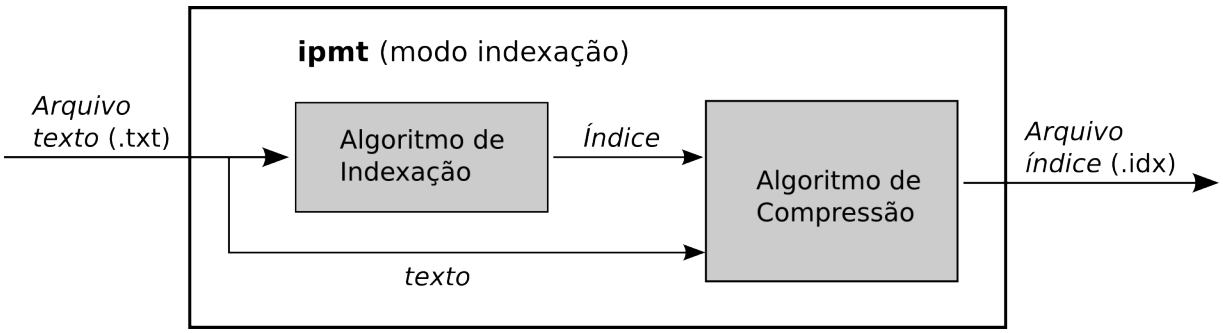

Figura 1: Modo de indexação

A ferramenta pode (não-obrigatoriamente) incluir opções para parametrização da estrutura de índice e/ou do algoritmo de compressão. Esses parâmetros devem eventualmente ser incorporados ao arquivo índice de forma que ele seja auto-contido, ou seja, o utilizador deste arquivo (vide seção a seguir) nâo precisa conhecer as opções usadas na sua construção.

Modo de busca

No modo de busca, o objetivo é procurar ocorrências exatas de padrões num texto em tempo linear (no tamanho dos padrões) com auxílio de um índice completo previamente computado. Este modo deve ser acionado a partir do comando

`$ ipmt search` ***`pattern indexfile`***

que fará com que o padrão `pattern` seja procurado no índice do arquivo `indexfile`. A ferramenta também poderá receber um conjunto de padrões a serem procurados num arquivo, sendo um padrão por linha, o que deve ser feito através da opção

`-p, --pattern` ***`patternfile`*****`:`** Realiza a busca de todos os padrões contidos no arquivo *patternfile*.

Repare que o arquivo de índice está comprimido, sendo necessário descodificá-lo antes de utilizar o índice. Assim, temos o esquema da Figura 2.

Implementação

A ferramenta deve ser implementada preferencialmente em C/C++. O objetivo é torná-la a mais eficiente possível. A ferramenta deve ser baseda na plataforma GNU/Linux. Deve-se tentar minimizar as depedências externas para torná-la facilmente portável entre plataformas.

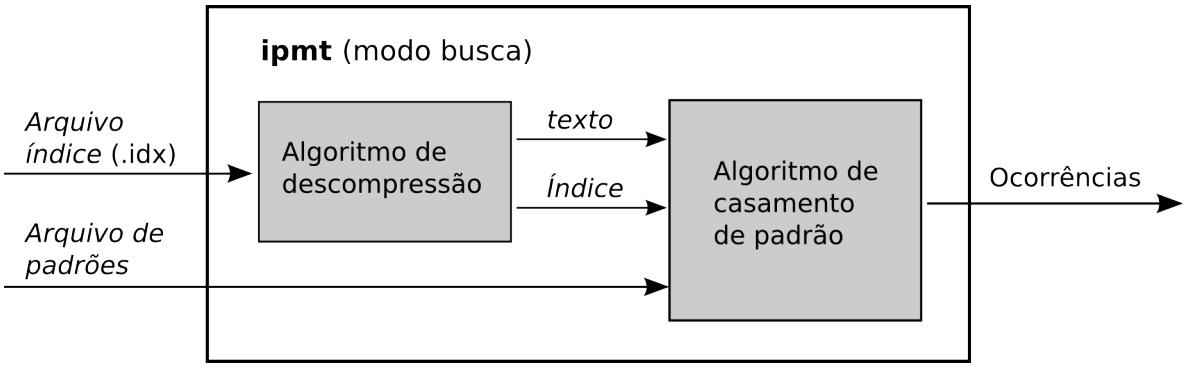

Figura 2: Modo de busca

Podem ser utilizadas APIs externas apenas para o *frontend* da ferramenta. Por exemplo, a GNU C Library (glibc)[2] contém funções para o parsing das opções de linha de comando (getopt), e para os nomes de arquivos com wildcards (fnmatch). Entretanto, o *backend* da ferramenta deve consistir *apenas de algoritmos vistos em aula* e (re-)implementados diretamente pelos alunos. *A detecção de cópia de partes substanciais do código desses algoritmos implicará na atribuição da nota 0.0 (zero) ao trabalho como um todo, independente de outras partes*.

O projeto completo consiste essencialmente na implementação dos quatro algoritmos representados pelas caixas sobreadas nas figuras 1 e 2. A estrutura de índice implementada deve ser uma árvore de sufixos ou array de sufixos. O algoritmo de compressão deve ser baseado no LZ77 ou LZ78.

Deliverables

Deve ser entregue uma tarball com nome no fomato *toolname.tgz*, onde *toolname* representa o nome da ferramenta (e.g. `pmt.tgz`). Este arquivo deve conter um diretório com o seguinte conteúdo *mínimo* (trocando-se, obviamente, `pmt` pelo nome escolhido).

z

```
pmt/
 |
 +-- doc/
 +-- src/
 +-- README.txt
```

O arquivo `README.txt` deve conter uma identificação da ferramenta, dos autores, e as instruções para compilação (vide seção abaixo). O conteúdo de cada diretório será especificado a seguir.

Código-fonte

Deve ser entegue o código fonte da ferramenta juntamente com um Makefile ou script para compilação no subdiretório `src/`. As instruções para o processo de compilação da ferramenta devem ser dadas no arquivo `README.txt`. Idealmente a compilação deveria consistir apenas na execução de um simples `make`.

[2] `http://www.gnu.org/software/libc/`

O código deve ser o mais *limpo*[3] possível. Entretanto, os objetivos principais são 1) correção e 2) eficiência. Portanto, deve-se evitar o uso exagerado de modelagem por objetos, padrões de projetos, etc. que tornem o programa mais lento. Um programa bem estruturado, com nomes expressivos para funções e variáveis, e com uma separação clara entre interface e motor de busca, deve ser suficiente.

Após a compilação, o arquivo executável deve estar num diretório `bin`, criado dentro do diretório original, isto é, teremos

```
pmt/
 |
 +-- bin/   <=== executável aqui
 +-- doc/
(...)
```

Documentação

Conforme as diretrizes adotadas para a CLI, uma ajuda com as instruções para a utilização básica da ferramenta deve ser obtida através da execução da ferramenta com a opção

`-h, --help`

Além disso, deverá ser entregue um breve relatório ($\leq$ 4 págs) contendo:

- Nome da ferramenta
- Identificação da equipe
- Descrição do funcionamento da ferramenta, incluindo a aescrição das opções e valores-padrão associados
- Detalhes de implementação relevantes, com impacto significativo para o desempenho da ferramenta, incluindo:
  - Indicação dos algoritmos e estruturas de dados básicos implementados
  - Descrição em alto nível das heurísticas, otimizações e adaptações
  - etc.
- Bugs conhecidos e limitações de desempenho notáveis. Se o trabalho não foi integralmente concluído, o que faltou deve ser explicitamente reportado aqui.

Esse relatório deve estar contido no subdiretório `doc/`, num arquivo chamado *toolname.ext*, onde *toolname* corresponde ao nome da ferramenta, e *ext* à extensão do formato do arquivo, que pode ser um `.pdf` ou um simples `.txt` (*Não* use MSWord ou qualquer formato proprietário).

[3] *RC Martin*. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008.

Avaliação

A avaliação será feita com base nos seguintes critérios:

1. Implementação (peso 3). Inclui a correção e qualidade do código-fonte levando-se em conta a quantidade e dificuldade intrínseca dos algoritmos implementados.
2. Qualidade da documentação (peso 3). Inclui o relatório, o `README.txt` e a ajuda do programa.
3. Eficiência prática do programa (peso 2). As ferramentas serão submetidas a testes automáticos nos quais serão aferidos
   (a) Os tempos de execução em diferentes entradas: serão utilizadas entradas grandes (porém não gigantes).
   (b) A taxa de compressão dos arquivos.

   Os resultados dos experimentos serão medidos em vários cenários. Para cada caso, serão então calculadas a média μ e o desvio padrão σ dos valores obtidos. As ferramentas com tempo $t \leq \mu + \sigma$ receberão a pontuação integral nesse teste. A partir desse limiar, a pontuação sofrerá uma penalização de $(1 + e^{-4(\delta - 1.5)})^{-1} \cdot 100\%$, onde $\delta := (t - \mu)/\sigma$.
4. Arguição (peso 2). Será agendada, posteriormente, uma breve arguição de cerca de 15min com cada equipe. Cada integrante deve ter participado de todas as atividades e, portanto, deve conhecer integralmente ser capaz de responder questões sobre qualquer aspecto do projeto.

Extras

Além desse conjunto mínimo de requisitos, cada equipe está livre para implementar recursos extras. Esses recursos devem ser assinalados no relatório e poderão receber, eventualmente, alguma bonificação.

Equipes

O projeto deve ser feito em duplas. Cada integrante deve participar de todas as atividades envolvidas (implementação, documentação e testes).

IMPORTANTE: De preferência, devem ser mantidas as mesmas equipes do Projeto 1.

Data de entrega

O tarball deve ser enviado por e-mail até **22 de Fevereiro de 2015**, impreterivelmente. As arguições ocorrerão de 23 a 27 de Fevereiro.

■ ■ ■