Universidade Federal de Pernambuco
Centro de Informática

IF767 - Processamento de Cadeias de Caracteres 2017.2
Prof. Paulo Fonseca

Projeto 2 — versão 1 (13/11/2017)

- 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.

A ferramenta deve chamar-se **`ipmt`** (*Indexed Pattern Matching Tool*).

Equipes

O projeto deve ser feito em equipes de 2 integrantes. Cada integrante é suposto participar e conhecer em detalhes todas as atividades envolvidas (implementação, documentação e testes).

Data de entrega

O trabalho deve ser entregue por e-mail até **10 de Dezembro de 2017** (veja a Seção *Deliverables*).

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.

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

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`

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.

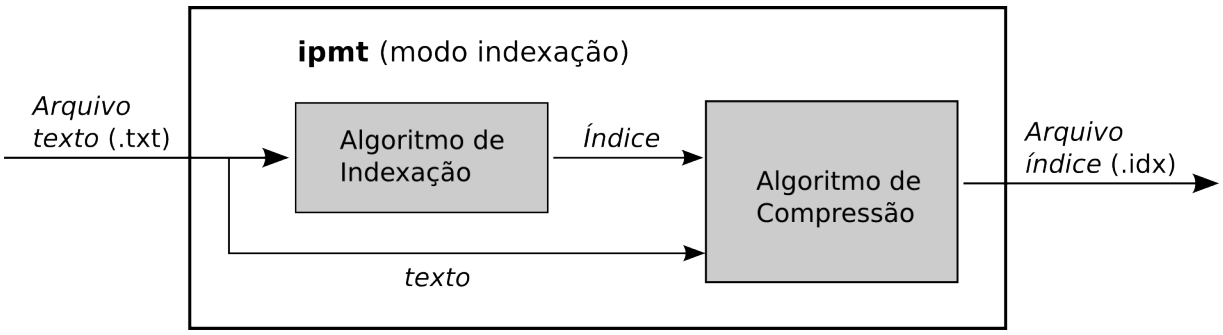

Figura 1: Modo de indexação

A ferramenta pode, opcionalmente, 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) *não precisa* conhecer as opções usadas na sua construção. *Nota:* O texto original pode ser omitido do arquivo `.idx` se for possível reconstruir o texto a partir do índice, o que resultará em arquivos menores.

Modo de busca

No modo de busca, o objetivo é procurar ocorrências exatas de padrões num texto em tempo linear (na soma dos tamanhos 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*.

O formato de saída do modo de busca deve ser similar ao do grep, ou seja, devem ser impressas as linhas do texto contendo os padrões procurados para a saída padrão.

Importante O modo de busca deve incluir *obrigatoriamente* uma opção

`-c, --count:` Imprime apenas uma linha com o número total de ocorrências de todos os padrões procurados.

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

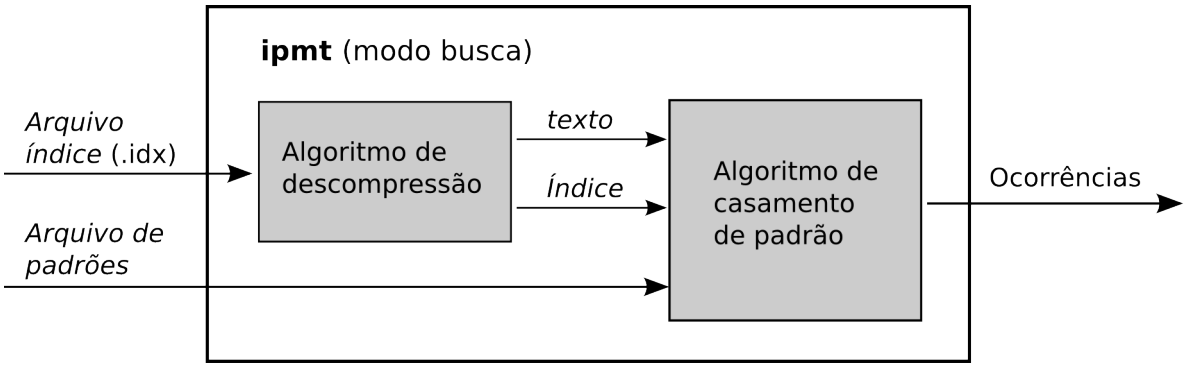

Figura 2: Modo de busca

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.

Podem ser utilizadas APIs externas apenas para o *frontend* da ferramenta. Entretanto, o *backend* da ferramenta deve consistir *apenas de algoritmos vistos em aula* e (re-)implementados diretamente pelos alunos, com possíveis extensões e/ou otimizações devidamente reportadas (cf. Seção Relatório, abaixo). *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 mínimo consiste essencialmente na implementação dos quatro algoritmos representados pelas caixas sombreadas 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. Podem, opcionalmente, ser implementadas diferentes alternativas para os algoritmos de indexação, compressão e busca. Nesse caso, além da escolha automática, devem ser incluidas opções para forçar a adoção de uma ou outra alternativa (e.g. **`--compression=LZ77`**, **`--indextype=suffixtree`**).

Testes/Experimentos

Devem ser realizados experimentos para aferir o desempenho prático da ferramenta em termos de tempo/espaço. Para isso deve ser compilado um conjunto de dados de teste composto da textos de diferentes fontes e origem. Como ponto de partida (chegada?) podem ser utilizados os corpora disponíveis em

1. Pizza&Chili (`http://pizzachili.dcc.uchile.cl/texts.htmli`)
2. SMART (`http://www.dmi.unict.it/~faro/smart/download.php`)

Os resultados dos experimentos para diversas configurações texto/padrão devem ser organizados em tabelas e gráficos. Para além dos simples dados brutos, deve-se tentar caracterizar um padrão de desempenho dos algoritmos em função dos parâmetros e características das entradas que nos permitam, eventualmente, prever o comportamento em cenários não testados diretamente. Ferramentas padrão como o `grep` e o `gzip`, bem como outros algoritmos e ferramentas disponíveis através da literatura e de software de terceiros podem/devem ser utilizados como benchmark para comparação.

Deliverables

Deve ser entregue um arquivo comprimido em formato `.tgz` ou `.zip`. Para facilitar a identificação nomeie o arquivo no formato

login-versão.tgz

onde ***login*** corresponde ao primeiro username em ordem lexicográfica da equipe e ***versão*** corresponde a um número sequencial (1,2,3,...) indicativo da versão submetida[2]. Esse arquivo comprimido deve consistir de um diretório com o seguinte conteúdo *mínimo*.

```
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] É comum que sejam submetidas mais de uma versão, devido a correções de última hora. Nesse caso, apenas a última versão é considerada para avaliação

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 relatório dividido em três principais seções:

1. Identificação
   - Identificação da equipe
   - Breve descrição da contribuição de cada membro da equipe ao trabalho
2. Implementação
   - Descrição do funcionamento da ferramenta, incuindo:
     - Algoritmos implementados
     - Situações nas quais cada algoritmo é empregado
   - Detalhes de implementação relevantes, com impacto significativo para o desempenho da ferramenta, incluindo:
     - Estruturas de dados
     - Estratégia de leitura das entradas
     - Heurísticas para combinação do algoritmos
     - Valores padrão dos parâmetros (e.g. tamanho da janela)
     - etc.
   - Bugs conhecidos e limitações de desempenho notáveis. Se o trabalho não foi integralmente concluído, o que faltou deve ser explicitamente reportado aqui.
3. Testes e Resultados
   - Descrição dos dados e ferramentas de comparação utilizados

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

- Descrição do ambiente de testes
- Descrição dos experimentos realizados
- Dados e resultados obtidos (tabelas, gráficos, ...)
- Discussão dos resultados
- Conclusões

Importante Dados experimentais brutos muito detalhados e volumosos não devem ser enviados. Se necessário, deve-se se incluir referências no relatório para onde esses dados podem ser obtidos. No relatório, deve-se buscar expor dados compilados que favoreçam a visualização e interpretação. Atentem para os objetivos indicados na Seção *Testes/Experimentos* acima. Incluam testes com dados variados e de volume significativo. Resultados obtidos com testes isolados e *ad hoc* carecem de robustez estatística.

Esse relatório deve estar contido no subdiretório `doc/`, num arquivo `.pdf` (*Não* use MSWord ou qualquer formato proprietário).

Data sets Os dados utilizados nos testes **NÃO** devem ser submetidos junto com o trabalho em nenhuma hipótese. A inclusão de arquivos de dados será penalizada. Caso seja considerado necessário, deve-se torná-los disponíveis online e indicar o endereço na seção da descrição dos testes do relatório.

Avaliação

A avaliação será feita com base nos seguintes critérios:

1. Implementação (peso 6). Inclui a correção, eficiência e qualidade do código-fonte levando-se em conta a quantidade e dificuldade intrínseca dos algoritmos implementados, bem como a facilidade de instalação e utilização da ferramenta.
2. Relatório (peso 4). Inclui a reprodutiblidade dos experimentos, a abrangência dos dados, a organização e apresentação dos resultados, a correção e profundidade das análises e a exposição das conclusões.

Arguição

A avaliação será feita mediante análise do material submetido e de uma arguição a ser agendada, posteriormente, 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 alguma bonificação.

■ ■ ■