Universidade Federal de Pernambuco – UFPE

Centro de Informática – CIn

Processamento de Cadeias de Caracteres

Buscador ipmt

Recife – 2018.2

Processamento de Cadeias de Caracteres

Buscador ipmt

Relatório do Projeto

Escrito por:

Alexsandro Vítor Serafim de Carvalho (avsc@cin.ufpe.br).

Entrega do relatório: 16/12/2018

Sumário

Sumário **3**

Introdução **4**

Implementação **4**

Algoritmos 4

Estrutura do arquivo .idx 4

Limitações 5

Bugs conhecidos 5

Correções 5

Testes **5**

Arquivos utilizados 5

Testes 5

bin/ipmt index -l 10000 testfiles/shakespeare.txt 6

bin/ipmt search roses testfiles/shakespeare.idx 6

bin/ipmt search -c roses testfiles/shakespeare.idx 6

bin/ipmt search and testfiles/shakespeare.idx 6

bin/ipmt search -c and testfiles/shakespeare.idx 6

bin/ipmt search -l 1000 " " testfiles/shakespeare.idx 6

bin/ipmt search -c -l 1000 " " testfiles/shakespeare.idx 6

bin/ipmt search -l 5000 batata testfiles/shakespeare.idx 7

bin/ipmt search -c -l 5000 batata testfiles/shakespeare.idx 7

Conclusões gerais 7

1. Introdução

O projeto que este relatório apresenta trata-se de um indexador / buscador de textos em índice feito com o objetivo de aplicar os conhecimentos adquiridos na cadeira de Processamento de Cadeias de Caracteres.
Este relatório descreverá o funcionamento da versão corrigida do mesmo.

2. Implementação

Ao realizar a busca, o programa pode ser configurado para retornar os matches (match = local no texto onde o padrão é encontrado) encontrados ou apenas a quantidade deles, através das opções equivalentes -c ou --count. Também existe uma opção para delimitar quantas linhas serão usadas na indexação ou na busca.

Cada match é retornado da seguinte forma:
[path]:[linha]:[coluna]:Lorem ipsum **match** dolor sit amet
Onde [path] é o caminho do arquivo onde houve o match, [linha] e [coluna] definem as coordenadas no texto do primeiro caractere do match e são seguidas por uma transcrição da linha onde ocorreu o match com o mesmo destacado.

Mais detalhes sobre as opções são dados no arquivo README.md e usando as opções -h ou --help na aplicação.

2.1. Algoritmos

Os algoritmos implementados neste trabalho foram o LZ-77 para compressão e o array de sufixos para indexação, com busca através de busca binária.

2.2. Estrutura do arquivo .idx

O arquivo .idx foi estruturado da seguinte forma:

```
[ls][ll]
// P/ cada linha indexada:
[tamanho do suffix array (== tamanho da linha original)]
[sa[0]][sa[1]][sa[2]] ... [sa[sa.size()-1]]
[tamanho da linha comprimida (== encoded.size())]
[encoded[0]][encoded[1]][encoded[2]] ... [encoded[encoded.size()-1]]
```

Cada componente entre colchetes acima é composto de 4 bytes, com exceção dos *encoded*. Cada encoded[i] é uma tupla estruturada de seguinte forma:

```
[p (4 bytes)][l (4 bytes)][c (1 byte)]
```

Quebras de linhas e comentários estão presentes apenas para visualização, eles não fazem parte do arquivo.

2.3. Limitações

Os algoritmos de busca são aplicados linha por linha. Por isso, não é possível buscar por strings com quebras de linha no meio delas. Além disso, uma string composta por mais de uma palavra só pode ser encontrada se as duas palavras estiverem na mesma linha do texto.

O programa considera que os textos foram formatados em ASCII, então está limitado aos caracteres do mesmo.

2.4. Bugs conhecidos

A formatação de cores só funciona no Linux ou no Git Bash do Windows, pois o Prompt de Comando do Windows não dá suporte à formatação usada no Linux.

O programa não trata o caso de tentar executá-lo em algo que não seja um arquivo índice.

O programa quebra ao tentar rodar com texto não codificado em ASCII.

2.5. Correções

Para resolver os problemas da versão apresentada anteriormente, foram feitas as seguintes correções:

- Foi criado um makefile para realizar a compilação.
- O código foi modificado para poder ser compilado no gcc no Ubuntu.
- A opção de alfabeto foi removida do programa e agora o arquivo de índice gerado é um arquivo binário. Com essa mudança, algumas otimizações das funções de compressão se tornaram possíveis.
- Foi incluída uma opção para ler um número máximo de linhas (-l ou --lines). Assim é possível testar o programa em arquivos muito grandes.

3. Testes

Os testes a seguir foram realizados em uma máquina do Laboratório de Graduação 3 (GRAD 3), com o sistema operacional Ubuntu 18.04.1 LTS instalado.

3.1. Arquivos utilizados

- shakespeare.txt: https://raw.githubusercontent.com/paguso/if76720182/master/data/shakespeare.txt

3.2. Testes

3.2.1. bin/ipmt index -l 10000 testfiles/shakespeare.txt

Tempos de execução: 14.00s; 14.02s; 14.03s
Tamanho: 3,8 MB
Observações: A indexação do texto shakespeare.txt, que gerou o arquivo shakespeare.idx, usado nas buscas seguintes. Comparando com o tempo de execução das buscas observa-se que é uma operação muito mais lenta.

3.2.2. bin/ipmt search roses testfiles/shakespeare.idx

Tempos de execução: 0,10s; 0,10s; 0,11s
Matches: 6
Observações: Uma busca com poucos retornos em shakespeare.idx.

3.2.3. bin/ipmt search -c roses testfiles/shakespeare.idx

Tempos de execução: 0,09s; 0,10s; 0,10s
Matches: 6
Observações: A mesma busca, porém sem imprimir os matches. Mais rápido, porém com uma diferença insignificante.

3.2.4. bin/ipmt search and testfiles/shakespeare.idx

Tempos de execução: 0,11s; 0,11s; 0,11s
Matches: 1564
Observações: Uma busca com um termo mais comum ("and") para avaliar o impacto da quantidade de matches na velocidade da busca. Houve um aumento no tempo, embora pouco significativo.

3.2.5. bin/ipmt search -c and testfiles/shakespeare.idx

Tempos de execução: 0,09s; 0,09s; 0,10s
Matches: 1564
Observações: A busca anterior, sem imprimir os matches. O resultado é mais próximo da busca 3.2.3, indicando que a quantidade de matches tem maior efeito no tempo de execução quando eles são impressos.

3.2.6. bin/ipmt search -l 1000 " " testfiles/shakespeare.idx

Tempos de execução: 0,06s; 0,06s; 0,07s
Matches: 8454
Observações: Uma busca com uma quantidade muito grande de matches. Apesar de parecer mais rápida que as outras, ela só foi executada em 1000 linhas (sendo que o arquivo de índice tem 10x isso). Estimando uma velocidade constante para o resto do arquivo, essa busca levaria 0,6 segundos, um tempo bem mais longo que o das outras buscas.

3.2.7. bin/ipmt search -c -l 1000 " " testfiles/shakespeare.idx

Tempos de execução: 0,01s; 0,01s; 0,01s

Matches: 8454

Observações: A busca anterior, sem imprimir os matches. O tempo de execução ainda parece ser proporcional à quantidade de linhas (Estimando 1000 linhas, o tempo seria 0,10, próximo dos outros).

3.2.8. bin/ipmt search -l 5000 batata testfiles/shakespeare.idx

Tempos de execução: 0,05s; 0,05s; 0,05s; 0,05s; 0,06s

Matches: 0

Observações: Uma busca por uma palavra em português, que não aparece em lugar algum. Esse teste confirmou que o tempo de execução é fortemente influenciado pela quantidade de linhas na busca e que ele cresce linearmente com a quantidade de linhas.

3.2.9. bin/ipmt search -c -l 5000 batata testfiles/shakespeare.idx

Tempos de execução: 0,05s; 0,05s; 0,05s; 0,06s; 0,08s

Matches: 0

Observações: A busca anterior, sem imprimir os matches. Com exceção de um outlier, os tempos de execução ficaram bastante próximos da busca anterior. Considerando que o teste anterior não gerou nenhuma impressão a mais que esse, faz sentido que os tempos de execução sejam similares.

3.3. Conclusões gerais

A opção de não imprimir os matches tem maior efeito com uma quantidade grande dos mesmos. Isso porque para se imprimir os matches é preciso armazená-los ao longo da execução, formatá-los e imprimí-los. Tudo isso possui um custo na execução da busca.

Por outro lado, o fator que mais alterou o tempo de execução foi a quantidade de linhas na busca. Dentre as buscas de apenas contagem, a mais rápida foi a que buscou em 1000 linhas e as mais lentas buscaram em 10000 delas, com diferença proporcional a quantidade de linhas.