

Simulações e conclusões

Ao longo deste trabalho foram estudados e desenvolvidos alguns algoritmos de compressão e descompressão de dados. De um modo geral, eles foram desenvolvidos para serem executados com arquivos de texto puro, tanto individualmente (processo de compressão através de apenas um algoritmo) quanto de forma combinada (ao menos dois algoritmos diferentes em sequência). Ao final do estudo e desenvolvimento de cada algoritmo, eles eram submetidos a um teste padrão de compressão sobre o arquivo de texto puro Os Lusíadas, com exceção do algoritmo JPEG que foi submetido a outro tipo de dado (uma vez que o JPEG comprime imagens). Neste tópico, todavia, simularemos novamente esses algoritmos de compressão e descompressão, porém com arquivos de texto puro de conteúdos bem diversos, complementando os resultados já obtidos.

Arquivos a serem usados

Nesta rodada de simulações, três novos arquivos de texto puro serão utilizados. O primeiro será denominado como “arquivo faculdade”, o segundo como “arquivo aleatório” e, o terceiro, como “arquivo léxico”. O arquivo faculdade será um arquivo de exercícios para uma disciplina cursada na Universidade Federal do ABC. O arquivo aleatório consistirá de caracteres gerados de forma aleatória e o arquivo léxico conterá palavras diferentes com

prefixos ou sufixos iguais.

Configuração dos arquivos de texto

É requisito para esta rodada de testes que todos os arquivos sejam arquivos de texto puro, que a codificação de seus caracteres seja a *ISO 8859-1* (a fim de garantir que todos os caracteres sejam codificados por apenas um *byte*) e que o caractere fim de linha seja codificado pelo “modo Linux”, para garantir, também, o uso de apenas um *byte* de dado.

Arquivo Faculdade

O arquivo faculdade é um arquivo de texto em português que foi utilizado em uma das disciplinas oferecidas pelo Bacharelado Interdisciplinar em Ciência e Tecnologia da Universidade Federal do ABC. Ele será usado como um representante de textos semânticos escritos em português.

Segue, abaixo, o conteúdo original do arquivo faculdade:

```

DD1

02. Resumo de 3 linhas com objetivos e novidades encontradas
  Caminho inverso:
  Objetivo: evidenciar como as matas tropicais nebulares são importantes na manutenção das
  nascentes dos rios de uma região.
  Novidades: O mais importante resultado do estudo foi a constatação de que certos tipos de
  vegetação podem, ao contrário do que se acreditou por décadas, absorver a umidade do ar pelas
  folhas e escoar essa água para a terra, liberando-a pelas raízes.

  Segredos do azul do mar:
  Objetivo: mostrar que ainda há um longo processo de descoberta de seres marinhos no mundo todo e
  que essa fauna e flora marinha possuem potenciais biotecnológicos para o desenvolvimento de
  soluções para os seres humanos.
  Novidades: algumas moléculas extraídas de organismos marinhos tem sido testadas como agentes
  fármacos na prevenção do desenvolvimento tumoral.

  Extinção de animais....

  Objetivo: mostrar como são importantes determinados animais frugívoros na manutenção da vegetação
  de grande porte (pois estocam mais CO2) de um ecossistema e, conseqüentemente, na redução das
  emissões de CO2.
  Novidades: Os pesquisadores notaram que a defaunação de frugívoros de grandes sementes resultou
  em diminuição da abundância de árvores de grande porte.

  03. A pesquisa de base proporciona, ainda que sem uma aplicação prática direta, compreensão sobre
  fenômenos biológicos, físicos, químicos, sociais, etc, que nos cercam. O estudo desses fenômenos,
  como consequência, nos permite tomar decisões políticas e econômicas mais assertivas, tanto do
  ponto de vista moral quanto de interesse pessoal e comunitário. Por exemplo, o caso dos animais
  frugívoros é um estudo importante, sem aplicação direta pela pesquisa em si, mas que gera um
  conhecimento extremamente útil para que se tome decisões acerca do desenvolvimento sustentável em
  regiões onde esses animais habitam.
  O estudo científico não deve ser feito pensando somente em aplicações a curto prazo. Se assim
  fosse, haveria apenas pesquisas cujos resultados gerassem interesse por parte de quem a financia.
  A compreensão do mundo é cumulativa e, muitas vezes, para se chegar a resultados realmente úteis,
  do ponto de vista da sua aplicabilidade, precisa-se acumular conhecimentos prévios que não
  necessariamente signifiquem solução de algum problema. Portanto, sim, a FAPESP como outras
  agências de fomento precisam investir em ciência de base, para que esta fortaleça o
  desenvolvimento mais horizontal de futuras pesquisas.

```

Figura 1 – Arquivo Faculdade

Arquivo aleatório

Para gerar o arquivo de caracteres aleatórios será usado um dispositivo abstrato gerador de números pseudo-aleatórios, que é nativo do sistema operacional *Linux*, chamado *urandom*, apesar de haver também o gerador *random*. Embora sejam intuitivamente semelhantes, há uma diferença crucial entre os números aleatórios que eles produzem.

De acordo com Gutterman; Pinkas (2006, p.2), a diferença entre os dois geradores está na segurança dos *bits* aleatórios gerados. Na mesma medida que o *random* gera *bits* aleatórios mais seguros que o *urandom*, ele também

demora muito mais tempo para realizar essa tarefa, podendo, ainda, bloquear o fornecimento desses dados aleatórios enquanto ele não for capaz de produzi-los. Em contrapartida, a ferramenta *urandom* gera números aleatórios menos seguros, mais rapidamente e sem bloquear o fornecimento desses *bits*, sendo mais comumente utilizada.

Com os *bits* aleatórios gerados, é preciso, para satisfazer as necessidades dos testes a serem realizados, filtrar o tipo de caractere de interesse (pois qualquer caractere da tabela ASCII pode ser gerado), definir a quantidade necessária de caracteres e fornecer um local para gravar o arquivo aleatório gerado. Para isso, a linha de comando a seguir será digitada no terminal e utilizada para realizar essa operação:

```
$ tr -dc a-zA-Z0-9 < /dev/urandom | head -c 500 >
arquivo_random.txt
```

Para entendê-la, podemos dividi-la em 4 partes:

- `< /dev/urandom`

O comando acima irá gerar dados pseudoaleatórios, por meio do dispositivo *urandom*, e, em seguida, realizará alguns filtros sucessivos.

- `tr -dc a-zA-Z0-9`

A linha acima filtra o tipo de caractere fornecido pelo gerador *urandom*.

O comando *tr*, neste caso deletará todas ocorrências de caracteres que

forem especificados nas opções do comando. A opção `-dc` sinaliza ao comando `tr` que o conjunto de caracteres a serem deletados (d) é o conjunto complementar (c) do conjunto especificado “a-zA-Z0-9”, isto é, `tr -dc` deletará todo caractere que for diferente de a a z, A a Z e 0 a 9 como, por exemplo, os caracteres de pontuação.

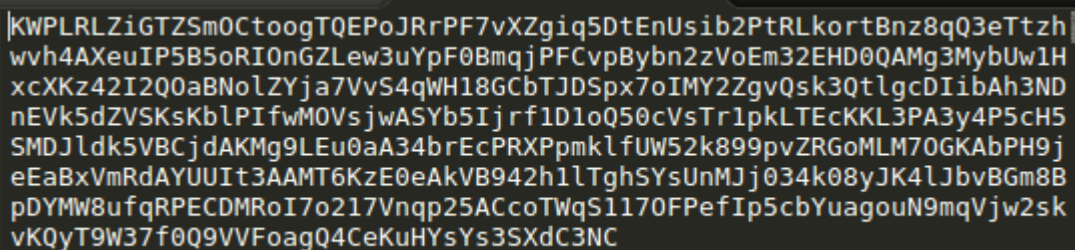
- `| head -c 500`

Uma vez que os caracteres aleatórios foram filtrados por seus tipos, restando apenas caracteres alfanuméricos, o comando `head -c 500` imprimirá apenas os 500 primeiros caracteres alfanuméricos gerados pelo dispositivo *urandom*.

- `> arquivo_random.txt`

Por fim, a linha acima redirecionará os 500 caracteres alfanuméricos gerados aleatoriamente para um arquivo de texto de saída chamado “arquivo_random.txt”.

Segue, abaixo, o conteúdo original do arquivo aleatório:



```
kWPLRLZiGTZSm0CtoogTQEPoJRrPF7vXZgiq5DtEnUsib2PtRLkortBnz8qQ3eTtzH
wvh4AXeuIP5B5oRI0nGZLew3uYpF0BmqjPFCvpBybn2zVoEm32EHD0QAMg3MybUw1H
xcXKz42I200aBNolZYja7VvS4qWH18GcbTJDSPx7oIMY2ZgvQsk3QtlgcDIibAh3ND
nEVk5dZVSKsKblPIfwM0VsJwASyb5Ijrf1D1oQ50cVsTr1pkLTEcKKL3PA3y4P5cH5
SMDJldk5VBCjdAKMg9LEu0aA34brEcPRXPpmklfUW52k899pvZRGomLM70GKAbPH9j
eEaBxVmRdAYUUIt3AAMT6KzE0eAkVB942h1lTghSYsUnMJj034k08yJK4lJbvBGm8B
pDYMw8ufqRPECDMRoI7o217Vnqp25ACcoTWqS1170FPefIp5cbYuagouN9mqVjw2sk
vKQyT9W37f0Q9VVfoagQ4CeKuHYsYs3SXdc3NC
```

Figura 2 – Arquivo Aleatório

Arquivo léxico

O arquivo léxico será um arquivo de texto puro contendo palavras selecionadas com um propósito específico de repetirem padrões de prefixos e sufixos. O intuito desse arquivo é trabalhar com a repetição de padrões sem que, necessariamente, ocorram repetições sequenciais de caracteres. Por exemplo, o arquivo conterà palavras como “amizade”, “brasilidade”, “interdisciplinaridade”, “abismo”, “civilismo”, “bacteriotropismo”, “populismo”, “atualismo”, “atualmente”, “popularmente”, “civilidade”, etc.

Segue, abaixo, o conteúdo original do arquivo léxico:

```
parati parede paraense parapente paracromatina parabolicamente
particular parlamentar civilizar civismo civicamente
civilizacional civilidade civilizabilidade atual atuarial
atualização atualidade atualismo medida mediano medidor medieval
mediação medicinal mediterrâneo mediocridade medievalização nação
ração doação ilação fração armação fixação rotação adulação
educação delineação duplicação hemossedimentação
substancialização abismo batismo egoísmo lulismo lobismo
fluidismo nepotismo polemismo biblicismo coleguismo obsoletismo
geomagnetismo inflacionismo construtivismo normossexualismo idade
cidade beldade maldade lealdade piedade laicidade liberdade
saciedade escamosidade precariedade proatividade capacitividade
cilindricidade coercitividade interdisciplinariedade
micromanipulabilidade
```

Figura 3 – Arquivo Léxico

Dados dos arquivos

Arquivo faculdade

- Tamanho original = 2458 *bytes*

Arquivo aleatório

- Tamanho original = 500 *bytes*

Arquivo léxico

- Tamanho original = 1129 *bytes*

Simulações a serem realizadas

Para cada arquivo gerado as simulações a seguir serão executadas anotando-se o tamanho original, o tamanho compactado, a taxa de compressão, a velocidade de compressão e a similitude entre o arquivo descompactado e o arquivo original, por meio do comando de comparação de arquivos via terminal de comandos.

Os algoritmos a serem testados serão o *RLE*, *LZ77*, *LZ78*, Huffman e, por fim, BWT e Huffman (simulando parcialmente o Bzip2). O LZMA não entrará na lista de testes porque, dos algoritmos que o compõem, o único estudado e implementado neste trabalho foi o *LZ77*, que já está na lista de testes.

Resultados

Os resultados obtidos estão tabulados abaixo.

Arquivo Léxico - tamanho original: 1129 bytes						
Algoritmos	Tamanho Compactado (bytes)	Taxa de Compressão	Tempos de execução (segundos)			
			real	user	sys	user + sys
RLE	2252	1,99	0m0.029	0m0.024	0m0.004	0m0.028
LZ77	2601	2,30	0m0.056	0m0.052	0m0.000	0m0.052
LZ78	2230	1,98	0m0.031	0m0.028	0m0.000	0m0.028
Huffman	4695	4,16	0m0.036	0m0.032	0m0.000	0m0.032
BWT e Huff	5437	4,82	0m0.061	0m0.048	0m0.008	0m0.056

Tabela 1 – Dados da compressão do arquivo léxico

Arquivo Faculdade - tamanho original: 2458 bytes						
Algoritmos	Tamanho Compactado (bytes)	Taxa de Compressão	Tempos de execução			
			real	user	sys	user + sys
RLE	4860	1,98	0m0.042	0m0.036	0m0.004	0m0.040
LZ77	7572	3,08	0m0.086	0m0.080	0m0.004	0m0.084
LZ78	5307	2,16	0m0.065	0m0.052	0m0.012	0m0.064
Huffman	11489	4,67	0m0.048	0m0.044	0m0.000	0m0.044
BWT e Huff	12822	5,22	0m0.089	0m0.084	0m0.000	0m0.084

Tabela 2 - Dados da compressão do arquivo faculdade

Arquivo Aleatório - tamanho original: 500 bytes						
Algoritmos	Tamanho Compactado (bytes)	Taxa de Compressão	Tempos de execução			
			real	user	sys	user + sys
RLE	986	1,97	0m0.019	0m0.020	0m0.000	0m0.020
LZ77	2218	4,44	0m0.032	0m0.032	0m0.000	0m0.032
LZ78	1603	3,21	0m0.040	0m0.032	0m0.004	0m0.036
Huffman	3510	7,02	0m0.037	0m0.032	0m0.004	0m0.036
BWT e Huff	3378	6,76	0m0.066	0m0.056	0m0.004	0m0.060

Tabela 3 - Dados da compressão do arquivo aleatório

O formato do conteúdo dos arquivos compactados gerados por cada algoritmo de compressão pode ser revisto no respectivo tópico de estudo do algoritmo de compressão.

As comparações com o comando de comparação de arquivos, via terminal, estão abaixo, feitas entre o arquivos originais e os arquivos descompactados:

Descompressão RLE

```
miguel@ $ cmp arquivo_lexico.txt arquivo_lexico.unrle
miguel@ $ cmp arquivo_faculdade.txt arquivo_faculdade.unrle
miguel@ $ cmp arquivo_random.txt arquivo_random.unrle
miguel@ $ |
```

Figura 4 – Comparações entre originais e descompressão RLE

Descompressão LZ77

```
miguel@ $ cmp arquivo_lexico.txt arquivo_lexico.u77
miguel@ $ cmp arquivo_faculdade.txt arquivo_faculdade.u77
miguel@ $ cmp arquivo_random.txt arquivo_random.u77
miguel@ $ |
```

Figura 5 - Comparações entre originais e descompressão LZ77

Descompressão LZ78

```
miguel@ $ cmp arquivo_lexico.txt arquivo_lexico.u78
miguel@ $ cmp arquivo_faculdade.txt arquivo_faculdade.u78
miguel@ $ cmp arquivo_random.txt arquivo_random.u78
miguel@ $ |
```

Figura 6 - Comparações entre originais e descompressão LZ78

Descompressão Huffman

```
miguel@ $ cmp arquivo_lexico.txt arquivo_lexico.uhuf
miguel@ $ cmp arquivo_faculdade.txt arquivo_faculdade.uhuf
miguel@ $ cmp arquivo_random.txt arquivo_random.uhuf
miguel@ $ |
```

Figura 7 - Comparações entre originais e descompressão Huffman

Descompressão Bzip2 (BWT + Huffman)

```
miguel@ $ cmp arquivo_lexico.txt arquivo_lexico.uhuf.unbwt
miguel@ $ cmp arquivo_faculdade.txt arquivo_faculdade.uhuf.unbwt
miguel@ $ cmp arquivo_random.txt arquivo_random.uhuf.unbwt
miguel@ $ |
```

Figura 8 - Comparações entre originais e descompressão Bzip2

Conclusões

Em todos os casos houve taxas de compressão maiores que 1, isto é, os arquivos compactados eram maiores que os originais. Isso se deve pelo fato de os algoritmos de compressão implementados neste trabalho terem sido desenvolvidos para comprimir bytes de dados, isto é, caracteres, ao invés de dados binários. Quando se trabalha a compressão de caracteres, a redundância de dados é menor, uma vez que a unidade de dado pode variar entre 256 possibilidades para caracteres de 8 *bits*. Por outro lado, se descer ao nível binário, onde todas as informações são armazenadas pela combinação de zeros e uns, a redundância será maior, dado que haverá muitas sequências repetidas de uns e zeros ou regiões com maiores concentrações de um determinado padrão de *bits*, favorecendo a compressão.

Por exemplo, na conclusão específica do tópico sobre Codificação *Huffman*, que gera arquivo de texto com conteúdo binário (caracteres zeros e uns), vimos que era possível estimar o real poder de compressão da técnica contando o total de caracteres binários do arquivo compactado e usando esse valor como o tamanho real da compactação. Naquele caso, o tamanho real estimado do arquivo compactado era muito menor que o tamanho original, evidenciando, assim, que ao se trabalhar no nível binário, as taxas de compressão podem ser mais expressivas.

Os tempos de execução, muito próximos, devem-se ao alto poder de processamento da máquina a que foram submetidos os testes e ao tamanho original reduzido dos arquivos. Resultados de tempos podem ser mais diversos se os algoritmos forem aplicados a arquivos realmente muito maiores que os testados.

Assim, torna-se impreciso comparar os resultados uns com os outros.

Este trabalho acabou por ter um caráter mais didático sobre cada método utilizado. O enfoque sobre o estudo dessas técnicas foi direcionado para os métodos de compressão em si, os fundamentos técnico-científicos que os embasam, o desenvolvimento de algoritmos com viés ilustrativo e a elaboração de uma coletânea de textos sobre as técnicas de compressão estudadas.

Conclusões gerais sobre o trabalho

Sobre a linguagem de programação utilizada, destaca-se como ponto positivo a simplicidade de uso e aprendizado da linguagem e sua documentação oficial rica. Comparada a linguagens mais tradicionais para esse tipo de processamento, como o C++ e o Java, o Python se mostrou demasiadamente intuitivo. Por outro lado, a sintaxe, em um primeiro momento, causa estranheza no desenvolvedor que está habituado às linguagens mais

antigas. Em síntese, o Python é uma ótima ferramenta de desenvolvimento de software para, principalmente, pessoas que não possuem um conhecimento profundo em programação. Para programadores, o seu uso passa a ser meramente mais uma questão pessoal de escolha.

Por fim, como aluno, este trabalho me enriqueceu de maneira inestimada. Foram muitos os desafios para concretizá-lo como, por exemplo, conciliar uma grade curricular quadrimestral densa, proporcionada pela UFABC, com o desenvolvimento dos estudos bibliográficos que embasaram este trabalho, o desenvolvimento prático dos algoritmos e a sua escrita. E a leitura de livros e artigos originais em inglês também foi um grande desafio transposto.

Referências

PINKAS. B; GUTTERMAN.Z. *Analysis of the Linux random number generator*.

Jerusalém, 2006. Disponível em <<http://www.pinkas.net/PAPERS/gpr06.pdf>>.

Acessado em 5/07/2017.