Trabalho Prático 1 - Algoritmos 2

Manipulação de sequências

Lucas Jacone da Silva - 2019006922

1 - Implementação da Trie

O trabalho proposto foi implementado em Python, utilizando uma classe chamada TrieNode para representar a árvore Trie tradicional, que é usada para auxiliar na compressão do arquivo de entrada. A classe TrieNode possui três atributos principais:

⬇ O atributo "filhos" é um dicionário que guarda os filhos do nó atual. Cada filho é representado por um caractere como chave e um nó correspondente como valor.

⬇ O atributo "valor" armazena o valor da string formada a partir da raiz até o nó atual. Esse valor é
utilizado no dicionário de prefixos, que é responsável pela codificação do arquivo.

⬇ O atributo "simbolo" representa o caractere guardado no nó atual da árvore.

Além disso, a classe TrieNode também possui um método chamado "criaFilho", que é responsável por adicionar um novo nó ao dicionário de filhos do nó pai.

Para calcular o tempo de execução do programa, foi utilizado o módulo "time" do Python. Esse módulo permite medir o tempo decorrido desde o início até o final da execução do programa, tanto na compressão quanto na descompressão.

O módulo "sys" foi utilizado para capturar e manipular os argumentos passados na linha de comando durante a compilação e interpretação do programa. Isso inclui a verificação do primeiro argumento, que determina se o programa deve executar a compressão ou a descompressão do arquivo.

Também foram tratados os casos em que é fornecido um arquivo de saída opcional, garantindo que os arquivos corretos sejam gerados com as extensões apropriadas.

O código em si é essencialmente governado por dois condicionais principais, dependendo do primeiro argumento passado na linha de comando:

Se o argumento for "-c", o programa entra no caso de compressão.

Se for "-x", o programa executa a descompressão do arquivo recebido.

Antes de entrar em um dos casos, o código abre dois arquivos: um para leitura (que é o arquivo recebido como o segundo argumento) e outro para escrita do resultado do

programa. Após todas as operações necessárias, os arquivos são fechados e o tempo de execução do programa é exibido.

2 - Etapa de Compressão

1 - Inicializamos a árvore Trie com o nó raiz, cujo valor é 0 e símbolo é uma string vazia. A variável "valor" é atribuída como 1, já que o valor 0 já foi usado no nó raiz.
valor

2 - Entramos em um loop que continua até que não haja mais caracteres para ler no arquivo original. Esse loop garante a leitura de todos os caracteres do arquivo. A cada iteração, incrementamos a variável "valor" em 1 unidade, pois uma nova inserção de nó na Trie é realizada.

3 - Dentro desse loop, lemos um caractere e o atribuímos a "char". Em seguida, começamos o processamento do nó raiz, com o padrão casado sendo uma string vazia até o momento.

4 - Procuramos no nó sendo processado se "char" é um dos seus filhos, verificando o dicionário de filhos do nó. Enquanto "char" for um filho do nó em questão, avançamos na árvore em direção a esse nó, concatenando "char" ao padrão casado e lendo mais um caractere do arquivo (atribuindo-o novamente a "char").

Esse processo de busca é repetido até que a busca retorne "None" (ou seja, até que o nó sendo processado não contenha o caractere lido como filho).

5 - Quando a condição acima ocorre, significa que encontramos o local de inserção do novo símbolo na Trie.

Nesse caso, escrevemos no arquivo de compressão o valor do nó que era prefixo do padrão lido até então, juntamente com o último caractere lido. Em seguida, criamos um novo nó na Trie contendo esse caractere, com o valor apropriado, e o inserimos como filho do nó que estávamos processando anteriormente.

6 - Por fim, calculamos o número mínimo de bits necessários (numBits) para armazenar todos os inteiros gerados no campo "valor". Isso é utilizado para determinar a taxa de compressão.

Essencialmente, é como se tivéssemos implementado um tipo de dado "int" que usa apenas a quantidade de bits necessária para expressar o número desejado. Por exemplo, se o valor vai até 9, apenas 4 bits são necessários. Isso não significa que o trabalho foi implementado para usar apenas 4 bits no arquivo comprimido, mas indica que em uma implementação otimizada, cada inteiro escrito no arquivo ocuparia apenas a quantidade necessária de espaço, melhorando a taxa de compressão.

7 - Seguindo o padrão do exemplo visto na Wikipedia, o espaço ocupado pelo arquivo comprimido é calculado multiplicando a quantidade de inteiros (valor) pelo espaço que eles ocupam (numBits), e somando o resultado com a quantidade de caracteres (desconsiderando os caracteres usados para espaçamento de símbolos) no arquivo comprimido multiplicado pelo espaço ocupado por cada caractere (valor * 8 bits por caractere). Com essa fórmula, podemos calcular o tamanho do arquivo comprimido.

Em seguida, comparamos esse tamanho com o tamanho original do arquivo (8 vezes o comprimento do arquivo original) e apresentamos a taxa de compressão, que é a divisão entre o tamanho do arquivo original e o tamanho do arquivo comprimido.
Esses passos são seguidos durante a compressão do arquivo, visando a construção da árvore Trie, a identificação de padrões repetidos e a escrita do arquivo comprimido com base nos nós da Trie encontrados.

3 - Etapa de Descompressão

Durante o processo de descompressão, seguimos os seguintes passos:

1 - Inicialmente, lemos todo o arquivo de entrada (arquivo codificado) e geramos uma lista de tokens no formato valor_char. Também inicializamos um dicionário de descompressão com a instância {0: string vazia} e definimos a variável "decompCount" como 1 para gerar as próximas chaves a serem adicionadas ao dicionário.

2 - Em seguida, entramos em um loop que lê um elemento da lista de tokens a cada iteração e realiza operações com ele. Esse loop continua até que o elemento lido na lista seja vazio, e a variável "decompCount" é incrementada em 1 unidade a cada iteração.

3 - Dentro do loop, separamos o token recebido em duas partes, "index" e "char", e construímos o novo texto que será adicionado ao dicionário concatenando a entrada do dicionário indexada pelo "index" e o símbolo "char". Em seguida, inserimos esse texto no dicionário, tendo "decompCount" como chave, e escrevemos o mesmo texto no arquivo de saída (arquivo decodificado).

Esses passos são seguidos durante a descompressão do arquivo, permitindo reconstruir o texto original com base nas informações contidas nos tokens e no dicionário de descompressão.

4 - Testes e Resultados

Durante a execução do programa, os seguintes dados são apresentados para cada arquivo:

- Tamanho original: representa o tamanho do arquivo antes da compressão.
- Tamanho após descompressão: é o tamanho estimado do arquivo após a descompressão, calculado de acordo com o método descrito na seção 2 do relatório.
- Tempo de compressão: indica o tempo necessário para realizar a compressão do arquivo.
- Tempo de descompressão: representa o tempo requerido para realizar a descompressão do arquivo.
- Taxa de compressão: é calculada com base no tamanho do arquivo exibido na saída padrão.

O tamanho do arquivo é exibido na saída padrão, mas pode variar um pouco em relação aos valores declarados previamente. Essa variação ocorre devido ao cálculo realizado, que consiste em multiplicar o tamanho do arquivo lido por 8 (bits por caractere do arquivo). A taxa de compressão é calculada levando em consideração o tamanho do arquivo exibido na saída padrão.

Nome do Arquivo	bolo.txt	musica.txt	artigo.txt	dante.txt	dante2.txt	poesias.txt	igual.txt	aleatorio.txt	lorem.txt	sample2.txt
Tamanho original (em bytes)	5432	14248	224392	268992	1075632	526304	12582912	1225624	14907824	17341896
Tempo de compressão (em segundos)	0.002983	0.003999	0.532387	0.024076	0.109504	0.043672	0.532878	0.075054	1.295726	1.848631
Tamanho após compressão (em bytes)	4964	10368	146433	183876	659939	332354	33706	428881	2073675	4321746
Tempo de descompressão (em segundos)	0.001002	0.000991	0.020038	0.006996	0.025339	0.012214	0.006505	0.014111	0.059867	0.148991
Taxa de compressão	1.094279	1.374228	1.006006	1.462899	1.629896	1.583565	373.313713	2.857725	7.189084	4.012706

5 - Conclusões

Podemos notar que, em diversos casos, como utilizamos os caracteres “ ‘ “ e “ \\ ” e também os tokens em sua forma inteira e não binária, o arquivo comprimido acaba ficando maior do que o arquivo pré-compressão, o que prejudica também os resultados observados na taxa

de compressão. O programa obtém resultados aceitáveis em ambos os tempos de compressão e descompressão.