Relatório TP1 - Compressor de Arquivos

Algoritmos II, UFMG, 2024

Ester Sara Assis
Júlia Paes de Viterbo 2021032137

0. Funcionamento e exemplo

Vamos supor um texto simples e com redundância: “a barba barbada de barbara era muito barbuda e tinha barbeiros”. O algortimo então cria um dicionário com todos os caracteres ASCII.

Em seguida, começa a ler os caracteres do texto:

1. começa com a string vazia: `""`, lê o primeiro caractere `'a'`:
    a. string formada é `"a"`, que já está no dicionário, então o algoritmo continua.
2. lê o próximo caractere `' '` (espaço):
    a. string formada é `"a"`, que não está no dicionário;
    b. código para `"a"` é encontrado (supondo que seja 0 no dicionário);
    c. código 0 é adicionado ao arquivo comprimido;
    d. algoritmo adiciona `"a"` ao dicionário com um novo código e continua com `" "`.
3. lê o próximo caractere `'b'`:
    idem
4. lê o próximo caractere `'a'`:
    a. string formada é `"ba"`, que não está no dicionário;
    b. código para `"b"` é encontrado (supondo que seja 2);
    c. código 2 é adicionado ao arquivo comprimido;
    d. algoritmo adiciona `"ba"` ao dicionário.
5. lê o próximo caractere `'r'`:
    a. string formada é `"bar"`, que não está no dicionário;
    b. código para `"ba"` é encontrado (novo código adicionado ao dicionário);
    c. código de `"bar"` é adicionado ao arquivo comprimido;
    d. algoritmo adiciona a sequência `"bar"` ao dicionário com um novo código.

O processo continua dessa forma até o final do texto. O algoritmo identifica novas sequências como "ba", "bar", "barb", e assim por diante, e as adiciona ao dicionário. Quando uma sequência não é encontrada no dicionário, ela é codificada e a sequência é adicionada.

Ao final, as sequências de caracteres que foram encontradas repetidamente no texto são substituídas por códigos mais curtos, representando essas sequências no dicionário. O arquivo comprimido será uma sequência de códigos que representam essas sequências.

Por exemplo, o arquivo comprimido (em vez de conter a frase original "a barba barbada de barbara era muito barbuda e tinha barbeiros") pode ter algo como: [0, 1, 2, 3, 4, 2, 5, 6, ...].

Para descomprimir, o caminho é inverso: inicializa-se o dicionário com caracteres básicos, lêem-se os códigos comprimidos e recuperam-se as sequências do dicionário; reconstrói-se o texto original a partir das sequências de códigos e atualiza-se o dicionário durante a descompressão conforme novos padrões são encontrados.

Note que o algoritmo é eficiente quando o texto possui alguma redundância pela natureza de lidar com prefixos.

1. Testes

Teste 0.

Conteúdo do arquivo: “Ju”
Propositalmente usamos um arquivo sem redundância para ver o que ocorreria.

Resultados:
compression_ratio: 1.0
dictionary_size: 257
execution_time: 0.0010731220245361328

decompression_time: 0.002169370651245117
dictionary_size: 257

Teste 1.

Conteúdo do arquivo: “a barba barbada de barbara era muito barbuda e tinha barbeiros”

compression_ratio: 1.441860465116279
dictionary_size: 298
execution_time: 0.0011098384857177734

decompression_time: 0.0006761550903320312
dictionary_size: 298

2. Análises e conclusões

Para arquivos pequenos e sem padrões repetitivos, como o arquivo no Teste 0, a compressão não é eficaz. A taxa de compressão ficou igual a 1, indicando que o algoritmo não conseguiu reduzir o tamanho do arquivo. Isso é esperado, pois o LZW depende da repetição de padrões para ser eficaz. O dicionário, com apenas os caracteres básicos, não cresceu muito e o tempo de execução foi muito baixo.

Para arquivos com muitos padrões repetitivos, como o arquivo de texto maior, Teste 1, a compressão foi muito mais eficaz. A taxa de compressão de 1.44 indicou uma redução significativa no tamanho do arquivo. O dicionário cresceu consideravelmente, pois mais sequências foram adicionadas à medida que o algoritmo identificava padrões repetidos. O tempo de execução e de descompressão ainda foi baixo, mostrando a eficiência do algoritmo mesmo com um dicionário maior.

Apesar de a quantidade de testes não ter sido maior pela falta de tempo, vê-se que o LZW é eficaz quando o arquivo contém redundância e padrões repetitivos, como no segundo teste. Em arquivos simples e sem padrões, como o primeiro teste, a compressão não resulta em uma redução de tamanho. Esse comportamento é esperado e mostra que o LZW é mais útil para compressão de arquivos com dados redundantes.