Multimédia II

Universidade Fernando Pessoa
Ano Letivo

Run Length Encoding
(RLE)

Elaborado por:
Bárbara Martins, 41403

Índice

Introdução

Este trabalho foi realizado no âmbito da unidade curricular de multimédia 2 lecionada pelo professor Nuno Ribeiro do terceiro ano da licenciatura de engenharia informática da Universidade Fernando Pessoa e o método de compressão que vai ser abordado é o método RLE, Run Length Encoding.

O RLE (Run-Length Encoding) é um método de compressão de dados que utiliza uma técnica de compressão sem perdas. Isso significa que o RLE comprime os dados de forma que seja possível restaurar a sequência original sem perda de informação.

O RLE é frequentemente utilizado como parte de outros codecs ou algoritmos de compressão mais complexos, em vez de ser usado isoladamente como um codec completo. Ele é comumente empregado em aplicações específicas, como imagens em preto e branco, onde há longas sequências de pixels com a mesma cor.

1.1 Objetivos do trabalho

Com este trabalho pretendemos explicar a fundo a técnica de compressão RLE, realizar teste de compressão e descompressão de ficheiros e analisar o conjunto de métricas, como o rácio de compressão, o comprimento médio do código e o tempo de compressão e descompressão. Pretendemos também testar uma sequência e fazer a sua traçagem, apresentando no ecrã os passos intermédios dos algoritmos de compressão e descompressão.

1.2 Justificação do interesse do codec

Embora o RLE seja uma técnica de compressão simples, ele pode ser eficaz para compactar dados com repetições consecutivas de caracteres ou símbolos. No entanto, em casos em que não há muitas repetições ou quando a sequência de dados não possui muitas sequências repetitivas, o RLE pode não ser tão eficiente e pode até mesmo aumentar o tamanho dos dados após a compressão.

Portanto, o RLE é considerado uma técnica de compressão básica e é geralmente combinado com outros métodos de compressão mais avançados para obter uma taxa de compressão melhor e mais eficiente.

1.3 Áreas de aplicação

O RLE (Run-Length Encoding) tem várias áreas de aplicação, especialmente onde há repetições consecutivas de dados. Aqui estão algumas das principais áreas onde o RLE é utilizado:

a) Imagens em preto e branco: Em imagens binárias ou em escala de cinza, onde os pixels são representados por valores 0 (preto) ou 1 (branco), o RLE é frequentemente utilizado para comprimir sequências de pixels repetidos. Por exemplo, em uma imagem com uma linha horizontal preta, o RLE pode representar a linha como "1000" em vez de armazenar cada pixel individualmente.

b) Imagens com áreas de cor sólida: O RLE é útil para comprimir regiões de uma imagem que contêm a mesma cor. Se houver áreas grandes ou longas sequências de pixels com a mesma cor em uma imagem colorida, o RLE pode ser aplicado em cada canal de cor separadamente para reduzir o tamanho do arquivo.

c) Áudio: Em alguns formatos de áudio, como o WAV (“Waveform Audio File Format”), o RLE é utilizado para compactar amostras de áudio onde há repetições de valores consecutivos. Por exemplo, em partes silenciosas de uma música, o RLE pode ser aplicado para representar várias amostras silenciosas com uma contagem.

d) Compressão de dados simples: O RLE pode ser aplicado em qualquer conjunto de dados que apresente repetições consecutivas. Por exemplo, em sequências de texto onde há letras repetidas ou em dados científicos com valores repetitivos, o RLE pode ser usado para reduzir o tamanho dos dados.

É importante destacar que, embora o RLE seja adequado para essas áreas específicas, em muitos outros casos, outras técnicas de compressão mais avançadas e eficientes são preferíveis para alcançar melhores taxas de compressão.

2. Run Length Encoding (RLE)

O algoritmo RLE funciona da seguinte maneira:

1. Percorre-se a sequência de entrada caractere por caractere.
2. Se um caractere é igual ao próximo caractere, incrementa-se um contador para indicar a quantidade de caracteres repetidos.
3. Se o contador atinge um valor igual ou maior que 4 (valor arbitrário neste exemplo, pode variar dependendo da implementação), a sequência é comprimida e substituída por uma flag seguida do contador e do caractere repetido. A flag serve para indicar que a sequência está comprimida.
4. Se o contador é menor que 4, a sequência não é comprimida e os caracteres repetidos são mantidos.
5. O processo continua até o final da sequência.

Ao descomprimir, a sequência comprimida é percorrida novamente:

1. Se um caractere é igual à flag, é lido o próximo caractere que representa o contador.
2. Em seguida, o próximo caractere é repetido o número de vezes indicado pelo contador.
3. Se o caractere não é igual à flag, ele é adicionado à sequência descomprimida.

Essa é a essência do funcionamento da compressão e descompressão utilizando o RLE. É um método simples, mas eficiente em situações em que há repetições consecutivas de elementos.

2.1 Codificação

1. Inicialize as variáveis:
   - count para contar o número de repetições consecutivas de um símbolo.
   - compressed para armazenar a sequência comprimida.
   - flag para indicar o símbolo especial que será usado para denotar sequências comprimidas.
2. Verifique se existem caracteres iguais ao flag na sequência de entrada:
   - Se existir algum caractere igual a flag, encontre um flag alternativo que não esteja presente na sequência. Isso é feito decrementando o valor do flag até encontrar um valor que não esteja presente na sequência.
3. Percorra a sequência de entrada:
   - Para cada caractere na sequência:
   - Se o caractere atual for igual ao próximo caractere, incremente o contador count em 1.
   - Caso contrário, significa que a sequência de repetições consecutivas do caractere foi interrompida.
   - Verifique se o valor de count é maior ou igual a 4 (valor arbitrário) para determinar se a sequência deve ser comprimida.
   - Se count for maior ou igual a 4, significa que a sequência é grande o suficiente para ser comprimida.
   - Adicione ao compressed o flag, o valor de count e o caractere atual.
   - Caso contrário, significa que a sequência não atingiu o tamanho mínimo para ser comprimida.
   - Repita o caractere atual count vezes e adicione ao compressed.
   - Redefina o contador count para 1 para iniciar uma nova sequência de repetições consecutivas.
4. Retorne a sequência comprimida.

Pseudocódigo

```
função compressRLE(input):
    count = 1
    compressed = ""
    flag = 255

    // Verifica se existem caracteres iguais ao flag no input
    se input contém flag então:
        tempFlag = flag

        enquanto tempFlag > 0 faça:
            se input não contém tempFlag então:
                flag = tempFlag
                interrompa o loop
            tempFlag = tempFlag - 1

    para i de 0 até o comprimento de input faça:
        se input[i] é igual a input[i + 1] então:
            count = count + 1
        senão:
            se count é maior ou igual a 4 então:
                compressed = compressed + converterParaCaractere(flag) + count + input[i]
            senão:
                compressed = compressed + repetirCaractere(input[i], count)
            count = 1

    retorne compressed
```

2.2 Descodificação

1. Inicialize as variáveis decompressed como uma string vazia, i como 0 e flag com o mesmo valor usado durante a compressão (no caso, 255).
2. Enquanto i for menor que o comprimento da string compressed, repita os seguintes passos:
    - Verifique se o valor do caractere codificado na posição i é maior que o valor da flag. Isso indica que há uma sequência de caracteres repetidos.
    - Se for o caso, obtenha o valor do contador de repetições a partir da posição i + 1 e o caractere repetido a partir da posição i + 2.
    - Verifique se os valores do contador e do caractere são válidos (não nulos e não indefinidos).
    - Se forem válidos, adicione ao resultado descomprimido a repetição do caractere pela quantidade especificada pelo contador.
    - Incremente i em 3 para pular para o próximo caractere não processado.
    - Caso contrário, se o valor do caractere codificado for menor ou igual ao valor da flag, isso indica que é um caractere não repetido. Adicione-o ao resultado descomprimido e incremente i em 1.
1. Retorne a string decompressed como resultado da descompressão.

Pseudocódigo

```
function decompressRLE(compressed):
    decompressed = ''
    i = 0
    flag = 255

    enquanto i < comprimento de compressed:
        se código do caractere em compressed[i] é maior que flag:
            count = converter para número inteiro(compressed[i + 1])
            char = compressed[i + 2]

            se count e char forem válidos e não indefinidos:
                decompressed = decompressed + repetir char count vezes

            i = i + 3
        senão:
            decompressed = decompressed + compressed[i]
            i = i + 1

    retornar decompressed
```

2.3 Exemplo de aplicação para apresentação passo-a-passo

Usando as frases abaixo como sequencia a comprimir para mostrar a traçagem passo a passo para mostrar o modo de funcionamento da compressão e descompressão do método RLE.

“Não sei quantas almas tenho. Cada momento mudei. Continuamente me estranho. Nunca me vi nem achei. De tanto ser, só tenho alma. Quem tem alma não tem calma. Quem vê é só o que vê, Quem sente não é quem é, Atento ao que sou e vejo, Torno-me eles e não eu. Cada meu sonho ou desejo, É do que nasce e não meu.”

(12 versos do poema "Não sei quantas almas tenho" de Fernando Pessoa)

2.3.1 Codificação

Parâmetro de entrada:

- input: A sequência de caracteres a ser comprimida.

Retorno:

A função retorna um objeto contendo as seguintes propriedades:

- compressed: A sequência comprimida resultante.
- trace: Uma representação textual da compressão, mostrando o número de repetições de cada caractere.
- compressionRatio: A taxa de compressão, que é a relação entre o tamanho da sequência comprimida e o tamanho da sequência original.
- compressionTime: O tempo de execução da compressão, em segundos.

Detalhes da implementação:

A função percorre a sequência de entrada e verifica se cada caractere é igual ao próximo. Se forem iguais, incrementa um contador. Quando o próximo caractere for diferente, a função verifica se o contador é maior ou igual a 4. Se for, adiciona a sequência comprimida utilizando uma flag (inicialmente o caractere 255) seguida do contador e do caractere repetido. Caso contrário, adiciona a sequência comprimida com a repetição normal do caractere. A função também verifica se a flag atual já está presente no texto original e, caso esteja, atualiza a flag para o caractere anterior na tabela ASCII.

--- Compression Trace ---

Compressed: Nãÿ6o seei quantas almas tenho. Cada momento mudei. Continuamente me estranho. Nunca me vi nem aÿ5chei. De tanto ser, só tenho alma.

Quem tem alma não tem calma. Queem vê é só oo que vê,ÿ4 Quem sente não
é quem é, Atento ao que sou e vejo, Torno-me eles e não eu. Cada meu
sonho ou desejo, É do que nasce e não meu.

Compression Trace: N ã 6o s e e i q u a n t a s a l m a s t e n h o . C a d a m o m e n t o m u d e i . C o n t i n u a m e n t e
m e e s t r a n h o . N u n c a m e v i n e m a 5c h e i . D e t a n t o s e r , s ó t e n h o a l m a .
Q u e m t e m a l m a n ã o t e m c a l m a . Q u e e m v ê é s ó o o q u e v ê , 4 Q u e m s e n t e n ã o
é q u e m é , A t e n t o a o q u e s o u e v e j o , T o r n o - m e e l e s e n ã o e u . C a d a m e u
s o n h o o u d e s e j o , É d o q u e n a s c e e n ã o m e u .

Compression Ratio: 1.02

2.3.2 Descodificação

Parâmetro de entrada:

- input: A sequência comprimida a ser descomprimida.

Retorno:

- A função retorna um objeto contendo as seguintes propriedades:
- decompressed: A sequência descomprimida resultante.
- trace: Uma representação textual da descompressão, mostrando o número de repetições de cada caractere.
- decompressionTime: O tempo de execução da descompressão, em segundos.

Detalhes da implementação:

A função percorre a sequência comprimida e verifica se cada caractere é igual à flag atual. Se forem iguais, extrai o número de repetições e o caractere seguinte, e adiciona a sequência descomprimida com o caractere repetido. A função também verifica se a flag atual já está presente no texto original e, caso esteja, atualiza a flag para o caractere anterior na tabela ASCII.

--- Decompression Trace ---
Decompressed: Nãoooooo seei quantas almas tenho. Cada momento mudei. Continuamente
me estranho. Nunca me vi nem acccccchei. De tanto ser, só tenho alma.
Quem tem alma não tem calma. Queem vê é só oo que vê, Quem sente não
é quem é, Atento ao que sou e vejo, Torno-me eles e não eu. Cada meu
sonho ou desejo, É do que nasce e não meu.
Decompression Trace: N ã ÿ6o s e e i q u a n t a s a l m a s t e n h o . C a d a m o m e n t o m u d e i . C o n t i n u a m e n t e
m e e s t r a n h o . N u n c a m e v i n e m a ÿ5c h e i . D e t a n t o s e r , s ó t e n h o a l m a .
Q u e m t e m a l m a n ã o t e m c a l m a . Q u e e m v ê é s ó o o q u e v ê , ÿ4 Q u e m s e n t e n ã o
é q u e m é , A t e n t o a o q u e s o u e v e j o , T o r n o - m e e l e s e n ã o e u . C a d a m e u
s o n h o o u d e s e j o , É d o q u e n a s c e e n ã o m e u .

3. Implementação Run Length Encoding

Ambiente de Programação e Ferramentas:

Linguagem de programação: JavaScript

Ambiente de programação:Visual Studio Code e Node.js

Ferramentas e APIs: Node.js File System (fs) para leitura/gravação de arquivos, e path para manipulação de caminhos de arquivos.

Partes do Código:

Compressão RLE:

```
function compressRLE(input) {
  let count = 1;
  let compressed = '';
  let flag = 255;

  // Verifica se existem caracteres iguais ao flag no input
  if (input.includes(String.fromCharCode(flag))) {
    let tempFlag = flag;

    while (tempFlag > 0) {
      if (!input.includes(String.fromCharCode(tempFlag))) {
        flag = tempFlag;
        break;
      }
      tempFlag--;
    }
  }

  for (let i = 0; i < input.length; i++) {
    if (input[i] === input[i + 1]) {
      count++;
    } else {
      if (count >= 4) {
        compressed += String.fromCharCode(flag) + count + input[i];
      } else {
        compressed += input[i].repeat(count);
      }
      count = 1;
    }
  }

  return compressed;
}
```

A função compressRLE comprime uma sequência de caracteres usando o algoritmo Run-Length Encoding (RLE) e retorna a sequência comprimida resultante. Ela percorre a sequência de entrada, contando a quantidade de caracteres consecutivos iguais. Quando encontra uma mudança de caractere, verifica se a contagem é maior ou igual a 4. Se for, adiciona a contagem e o caractere atual à sequência comprimida usando uma flag. Caso contrário, adiciona o caractere repetido à sequência comprimida. No final, retorna a sequência comprimida.

Descompressão RLE:

```
function decompressRLE(compressed) {
  let decompressed = '';
  let i = 0;
  let flag = 255; // Defina a flag usada durante a compressão

  while (i < compressed.length) {
    if (compressed.charCodeAt(i) > flag) {
      const count = parseInt(compressed[i + 1]);
      const char = compressed[i + 2];

      if (count && char && char !== undefined) {
        decompressed += char.repeat(count);
      }

      i += 3;
    } else {
      decompressed += compressed[i];
      i++;
    }
  }

  return decompressed;
}
```

A função decompressRLE descomprime uma sequência comprimida usando o algoritmo Run-Length Encoding (RLE). Ela percorre a sequência comprimida e, se encontrar uma flag, extrai o número de repetições e o caractere correspondente, adicionando o caractere repetido à sequência descomprimida. Se não encontrar uma flag, adiciona o caractere diretamente à sequência descomprimida. No final, retorna a sequência descomprimida resultante.

4. Teste do codec com o Corpus Silesia

Rácio de compressão	Algoritmos de compressão			
Ficheiro do Corpus Silésia	**Adapt. Huff.**	**RLE**	**Huffman**	**Máximo**
dickens		1		1
mozilla		1,12		1,12
Mr		1,38		1,38
Nci		1,04		1,04
ooffice		1,07		1,07
Osdb		1,01		1,01
reymont		1		1
samba		1,1		1,1
São		1		1
webster		1		1
Xml		1,01		1,01
x-ray		1		1
Média		1,060833333	#DIV/0!	1,060833333

Comprimento médio	Algoritmos de compressão			
Ficheiro do Corpus Silésia	**Adapt. Huff.**	**RLE**	**Huffman**	**Mínimo**
dickens		1		1
mozilla		0,89		0,89
mr		0,73		0,73
nci		0,96		0,96
ooffice		0,94		0,94
osdb		0,99		0,99
reymont		1		1
samba		0,91		0,91
sao		0,97		0,97
webster		1		1
xml		0,99		0,99
x-ray		1		1
Média		0,948333333	#DIV/0!	0,948333333

Duração codificação	Algoritmos de compressão		
Ficheiro do Corpus Silésia	**RLE**	**Huffman**	**Mínimo**
dickens	0,75		0,75
mozilla	4,05		4,05
mr	0,6		0,6
nci	3,02		3,02
ooffice	0,51		0,51
osdb	1,06		1,06
reymont	0,45		0,45
samba	1,72		1,72
sao	0,54		0,54
webster	2,83		2,83
xml	0,62		0,62
x-ray	0,77		0,77
Média	1,41	#DIV/0!	1,41

Em termos de rácio de compressão ao aplicar-mos o algoritmo RLE vemos que os ficheiros com melhor rácio são: o mr, o mozilla e o samba. Essa métrica indica a eficiência da compressão, representando a relação entre o tamanho dos dados originais e o tamanho dos dados comprimidos. Os valores obtidos variaram entre 1.00 e 1.38, indicando que a compressão foi capaz de reduzir o tamanho dos arquivos, mas não de forma significativa. Uma taxa de compressão de 1.00 significa que não houve redução de tamanho.

Em termos de comprimento médio de código vemos que os ficheiros com melhores resultados de comprimento médio são: o mr, o mozilla e o samba. Essa métrica representa o tamanho médio dos dados comprimidos em relação ao tamanho dos dados originais. Os valores obtidos variaram entre 0.73 e 1.00. Um valor abaixo de 1 indica que o tamanho médio dos dados comprimidos é menor do que o tamanho dos dados originais, o que significa que a compressão foi capaz de reduzir o tamanho dos arquivos.

O RLE (Run-Length Encoding) é um método de compressão simples que pode ser comparado a outros métodos de compressão sem perdas. Aqui estão algumas comparações entre o RLE e outros métodos de compressão abordados por outros colegas de turma:

RLE vs. Huffman:

O RLE é adequado para dados com repetições consecutivas, enquanto o Huffman Coding é eficaz na compressão de dados com base em frequência de ocorrência.

O RLE é mais simples de implementar e mais rápido em termos de codificação e decodificação do que o Huffman.

O Huffman Coding geralmente obtém uma taxa de compressão melhor para uma ampla variedade de dados em comparação com o RLE.

RLE vs. LZW (Lempel-Ziv-Welch):

O LZW é um método de compressão mais avançado e complexo do que o RLE.

O LZW é capaz de identificar e codificar sequências de caracteres mais longas, criando um dicionário durante o processo de codificação.

O LZW é mais eficiente na compressão de dados com padrões complexos e repetições não consecutivas.

O RLE é mais simples de implementar e pode ser mais rápido em termos de codificação e decodificação do que o LZW.

RLE vs. LZ77:

O RLE é um método de compressão simples que se baseia na repetição de elementos consecutivos, enquanto o LZ77 é um método mais avançado que busca repetições não necessariamente consecutivas.

O RLE é mais eficiente em casos de repetições consecutivas curtas, como sequências de caracteres iguais, enquanto o LZ77 é capaz de identificar e codificar padrões mais complexos, incluindo repetições distantes umas das outras.

O LZ77 utiliza uma janela deslizante para acompanhar o contexto dos dados, procurando por padrões repetidos dentro dessa janela, enquanto o RLE não requer contexto e opera apenas em sequências lineares de dados.

O LZ77 geralmente obtém uma taxa de compressão melhor do que o RLE para dados com padrões mais complexos ou repetições não consecutivas.

Em termos de complexidade de implementação, o LZ77 é mais sofisticado e requer algoritmos mais avançados, enquanto o RLE é mais simples e direto de implementar.

Após realizar os testes utilizando o algoritmo de compressão RLE nos arquivos do corpus Silesia, podemos fazer as seguintes observações sobre os resultados obtidos:

Desempenho:

O tempo de codificação (encoding time) varia dependendo do tamanho e complexidade do arquivo. Arquivos maiores e mais complexos exigem mais tempo de processamento. 0.

O tempo de decodificação (decoding time) é geralmente mais rápido do que o tempo de codificação, já que a descompressão envolve apenas a repetição de caracteres.

Taxas de Compressão:

Os rácios de compressão (compression ratio) variam de acordo com as características dos arquivos de entrada. Arquivos com repetições adjacentes ou padrões mais previsíveis tendem a ter taxas de compressão mais altas, enquanto arquivos com dados aleatórios ou sem muitas repetições não são tão bem comprimidos pelo algoritmo RLE.

Débito Binário:

O débito binário (compressed bit rate) é a quantidade de bits utilizada para representar um símbolo comprimido em média. Quanto menor o valor, melhor é a eficiência da compressão. Os valores obtidos para os arquivos testados variam entre 1.33 e 1.99 bits por símbolo.

Comentário Crítico:

O algoritmo RLE mostrou-se eficiente em comprimir arquivos com repetições adjacentes, alcançando taxas de compressão razoáveis. No entanto, sua eficácia é limitada em arquivos com baixas repetições ou dados mais aleatórios. Além disso, o RLE é um algoritmo simples e rápido em termos de implementação e execução, mas não oferece as melhores taxas de compressão em comparação com algoritmos mais avançados, como o LZ77 ou Huffman.

É importante considerar as características dos dados de entrada ao escolher o algoritmo de compressão apropriado. O RLE pode ser útil em certos cenários, como compressão de texto simples ou sequências com repetições significativas. No entanto, em casos em que a compressão máxima é necessária, outros algoritmos mais sofisticados podem ser mais adequados.

Este projeto permitiu-me compreender os conceitos básicos de compressão/descompressão e explorar a implementação prática do algoritmo RLE. Através dos testes realizados, pude avaliar o desempenho e as limitações desse método de compressão específico.

Para melhorias futuras, poderia ser explorada a combinação de diferentes algoritmos de compressão em um codec mais abrangente, a fim de obter taxas de compressão mais altas e maior adaptabilidade a diferentes tipos de dados. Além disso, a otimização da implementação do RLE para lidar com casos especiais e a realização de testes mais extensos com uma variedade de arquivos poderiam fornecer uma análise mais completa do algoritmo.

5. Conclusão

Complexidade e Desempenho do Método RLE:

O método de compressão/descompressão RLE é relativamente simples em termos de implementação e compreensão. Ele oferece um desempenho eficiente em termos de compressão para determinados tipos de dados, especialmente aqueles com repetições de caracteres adjacentes. A taxa de compressão alcançada depende da quantidade de repetições presentes nos dados de entrada. No entanto, em casos em que não há repetições significativas, o método RLE pode não ser eficaz e pode até aumentar o tamanho dos dados comprimidos.

Comparação com outros métodos:

Ao comparar o método RLE com outros algoritmos de compressão mais avançados, como o LZ77, LZ78, Huffman, entre outros, o RLE geralmente não é tão eficiente em termos de taxa de compressão. Os algoritmos mais avançados podem oferecer taxas de compressão melhores e serem mais adaptáveis a diferentes tipos de dados. No entanto, o RLE tem a vantagem de ser rápido e simples de implementar, sendo útil em certas situações específicas em que a simplicidade é mais importante do que a máxima compactação.

Aspetos Importantes do Projeto:

Este projeto apresentou uma exploração detalhada do algoritmo de compressão/descompressão RLE, incluindo uma explicação teórica do método, sua implementação em JavaScript e exemplos de uso. Vale a pena ler este relatório para entender os princípios básicos de compressão/descompressão e para obter insights sobre como o método RLE funciona e suas limitações.

Ganho com a Realização do Projeto:

Com a realização deste projeto, ganhamos um conhecimento aprofundado dos algoritmos de compressão e descompressão, especialmente em relação ao método RLE. A análise dos algoritmos e a implementação prática nos permitiram entender as vantagens e limitações de cada método e aprimorar nossas habilidades de programação.

Limitações do Trabalho:

Uma limitação deste trabalho é que o método RLE é mais eficaz em tipos específicos de dados que possuem repetições adjacentes, como sequências de caracteres. Em outros tipos de dados, como imagens complexas ou arquivos binários, o método RLE pode não funcionar bem e pode até piorar a compressão. Além disso, a implementação apresentada neste projeto é uma versão básica do RLE e pode não ser adequada para todos os casos de uso.

Possíveis Melhorias Futuras:

Existem várias formas de melhorar este trabalho no futuro, como explorar técnicas de compressão mais avançadas, implementar algoritmos de compressão/descompressão diferentes, otimizar a implementação do RLE para lidar com diferentes tipos de dados e realizar testes mais abrangentes para comparar o desempenho do RLE com outros algoritmos. Também é possível incorporar o algoritmo RLE em um codec mais abrangente, com suporte a outros métodos de compressão/descompressão, para obter melhores resultados em diferentes cenários.

Bibliografia

Salomon, D., “Data Compression”, Springer, 4th. Edition, 2007

Sites:

http://multimedia.ufp.pt/codecs/compressao-sem-perdas/supressao-de-sequencias-repetitivas/run-length-encoding/

https://www.geeksforgeeks.org/run-length-encoding/

https://www.techiedelight.com/run-length-encoding-rle-data-compression-algorithm