Cauã Borges Faria (834437)

1. Quais as características desejáveis de um método de compressão?

Um bom método de compressão busca um equilíbrio entre diversos fatores. As características desejáveis incluem:

- **Alta Taxa de Compressão:** É o objetivo principal, quanto menor o arquivo final, melhor. Queremos remover o máximo de redundância possível.
- **Velocidade de Compressão (Codificação):** O tempo que leva para o algoritmo compactar o arquivo. Em algumas aplicações, como streaming em tempo real, isso é crucial.
- **Velocidade de Descompressão (Decodificação):** O tempo para restaurar o arquivo original. Geralmente, a descompressão precisa ser muito rápida para que o dado possa ser utilizado prontamente.
- **Baixa Demanda de Memória:** O algoritmo não deve consumir muita memória RAM durante o processo de compressão e descompressão. Isso é especialmente importante em dispositivos com recursos limitados.
- **Ausência de Perda de Dados (para compressão *lossless*):** Para dados como documentos de texto, programas ou imagens médicas, é fundamental que o arquivo descomprimido seja **idêntico** ao original. Métodos *lossless* garantem isso. Para outros tipos de dados, como imagens e áudio, pode-se aceitar alguma perda (*lossy*) em troca de maior compressão.
- **Versatilidade:** A capacidade de o método lidar bem com diferentes tipos de dados (texto, imagens, áudio, etc.).
- **Robustez:** A resiliência do método a erros nos dados compactados.

2. O que é e como podemos calcular uma taxa de compressão? Use um exemplo para ajudar a conceituar.

A **taxa de compressão** é uma métrica que indica o quanto um arquivo foi reduzido em tamanho após a aplicação de um algoritmo de compressão. Ela expressa a eficiência do método.

Podemos calculá-la de algumas formas, mas as mais comuns são:

- Razão de Compressão:

  Razaˆo de Compressaˆo=Tamanho do Arquivo ComprimidoTamanho do Arquivo Original

  Uma razão de compressão de 2:1 significa que o arquivo original tinha o dobro do tamanho do arquivo comprimido. Quanto maior esse valor, melhor a compressão.
- Taxa de Redução (ou Economia Percentual):

  Taxa de Reduc¸aˆo (%)=(1−Tamanho do Arquivo OriginalTamanho do Arquivo Comprimido)×100%

  Ou

  Taxa de Reduc¸aˆo (%)=(Tamanho do Arquivo OriginalTamanho do Arquivo Original−Tamanho do Arquivo Comprimido)×100%

  Essa métrica mostra a porcentagem do tamanho original que foi "economizada".

Exemplo para Conceituar:

Imagine que você tem um arquivo de texto com **1000 bytes** de tamanho. Após aplicar um método de compressão, o arquivo resultante tem **250 bytes**.

- Razão de Compressão:

$$\frac{250 \text{ bytes}}{1000 \text{ bytes}} = 4$$

Isso significa uma razão de compressão de 4:1. O arquivo original era 4 vezes maior que o comprimido.

- Taxa de Redução:

$$\left(1 - \frac{250}{1000}\right) \times 100\% = (1 - 0.25) \times 100\% = 0.75 \times 100\% = 75\%$$

Ou seja, houve uma redução de 75% no tamanho do arquivo.

---

3. Como funciona o método de compressão por Redução de Redundâncias? Descreva a estratégia geral, e use um exemplo para ilustrar o funcionamento.

O método de **Redução de Redundâncias** (muitas vezes conhecido como **Run-Length Encoding - RLE**) é uma técnica de compressão *lossless* que se baseia na ideia de substituir sequências repetidas de um mesmo símbolo (redundâncias) por uma representação mais compacta.

Estratégia Geral:

A estratégia é identificar **sequências consecutivas do mesmo símbolo** e, em vez de armazenar cada ocorrência do símbolo, armazenar o símbolo uma única vez seguido de um **contador** que indica quantas vezes ele se repetiu. Para distinguir um símbolo que faz parte de uma sequência de repetição de um símbolo que é um contador, geralmente usa-se um **símbolo indicador/marcador** especial.

Funcionamento:

1. **Símbolo Indicador:** É definido um símbolo especial que não aparece normalmente nos dados ou que é reservado para indicar que a sequência seguinte representa uma repetição.

2. **Identificação de Sequências:** O algoritmo percorre o arquivo de entrada, buscando por sequências de dois ou mais símbolos idênticos consecutivos.
3. **Codificação:** Quando uma sequência de `N` ocorrências de um símbolo `X` é encontrada:
    - Se `N` for pequeno (por exemplo, 1 ou 2), pode-se optar por não comprimir, pois a codificação extra (indicador + contador) poderia tornar o arquivo maior.
    - Se `N` for suficientemente grande, a sequência é substituída pela tripla: `[Símbolo_Indicador][Símbolo_Repetido][Contador_de_Repetições]`.

Exemplo para Ilustrar o Funcionamento:

Vamos usar um exemplo simples com uma sequência de letras: `AAAABBCDDDDDE`

Suponha que nosso **Símbolo Indicador** seja `#`.

- **`AAAA`**: Temos 4 'A's. Codificamos como `#A4`
- **`BB`**: Temos 2 'B's. Codificamos como `#B2`
- **`C`**: Apenas 1 'C'. Não há repetição significativa, então mantemos 'C'.
- **`DDDDDD`**: Temos 6 'D's. Codificamos como `#D6`
- **`E`**: Apenas 1 'E'. Mantemos 'E'.

Então, a sequência `AAAABBCDDDDDE` seria compactada para:

```
#A4#B2C#D6E
```

Perceba que, se o contador for muito pequeno, como no caso do 'C' ou 'E' (que aparecem apenas uma vez), a compressão pode não ser vantajosa ou até mesmo aumentar o tamanho do arquivo (se `#C1` fosse maior que `C`). Por isso, geralmente há um limite mínimo para aplicar a compressão de repetições.

---

4. Considere um alfabeto Hexadecimal de 00 a FF, considere que o símbolo FF foi reservado como indicador de redundância, e considere que o arquivo de entrada seja a seguinte sequência: “12 22 00 00 00 00 00 00 00 12 22 12 00 00 00 00 00 00 00 00”. Os espaços foram incluídos apenas como separadores; desconsiderar. Proponha uma codificação com base em Redução de Redundâncias, defina o que for necessário definir, gere o arquivo codificado.

Definições:

- **Alfabeto:** Hexadecimal (00 a FF).
- **Símbolo Indicador de Redundância:** `FF` (reservado).
- **Formato de Codificação de Redundância:** `[FF][Símbolo_Repetido][Contador_de_Repetições]`
- **Limiar de Compressão:** Para que a compressão seja eficaz, vamos definir que uma sequência só será compactada se tiver **3 ou mais** repetições do mesmo símbolo. Se tiver 1 ou 2, ela será mantida como está (pois `FF X N` é 3 bytes, o que seria maior ou igual a 3 bytes de repetição).

Arquivo de Entrada (sem espaços):

12 22 00 00 00 00 00 00 00 12 22 12 00 00 00 00 00 00 00 00

Análise e Codificação:

1. **`12`**: Não repetido. Mantém `12`.
2. **`22`**: Não repetido. Mantém `22`.
3. **`00 00 00 00 00 00 00`**: Sequência de **7** zeros (`00`). Codificamos como `FF 00 07`.
4. **`12`**: Não repetido. Mantém `12`.
5. **`22`**: Não repetido. Mantém `22`.
6. **`12`**: Não repetido. Mantém `12`.
7. **`00 00 00 00 00 00 00 00 00`**: Sequência de **9** zeros (`00`). Codificamos como `FF 00 09`.

Arquivo Codificado:

```
12 22 FF 00 07 12 22 12 FF 00 09
```

Comparação de Tamanhos (em bytes/símbolos):

- **Original:** 21 símbolos (bytes)
- **Codificado:** 3 (FF 00 07) + 3 (FF 00 09) + 9 (símbolos não compactados) = **15 símbolos (bytes)**

Houve uma redução de 21 para 15 símbolos, demonstrando a eficácia da redução de redundâncias.

---

5. Descreva o processo de codificação, no método Redução de Redundâncias.

O processo de codificação (compressão) no método de Redução de Redundâncias segue os seguintes passos:

1. **Inicialização:**
    - Defina um **símbolo indicador/marcador** (no exemplo, `FF`) que será usado para sinalizar o início de uma sequência compactada. Este símbolo deve ser escolhido de forma que não cause ambiguidade com os dados originais, preferencialmente um que seja incomum ou reservado.
    - Defina um **limiar mínimo de repetições** para que a compressão seja aplicada (por exemplo, 3 ou mais repetições). Repetições abaixo desse limiar são deixadas como estão, pois codificá-las (indicador + símbolo + contador) ocuparia o mesmo espaço ou mais.
2. **Leitura Sequencial:** O algoritmo lê o arquivo de entrada símbolo por símbolo, de forma sequencial.
3. **Identificação de Repetições:**
    - Ele mantém um registro do **símbolo atual** e conta quantas vezes ele aparece **consecutivamente**.

    - Se o símbolo atual for diferente do símbolo anterior, ou se a sequência de repetições for quebrada, o contador é resetado.
4. **Decisão de Codificação:**
    - Quando uma sequência do mesmo símbolo é identificada e o **contador de repetições atinge ou excede o limiar mínimo** definido, o algoritmo decide compactar essa sequência.
    - Se o contador for menor que o limiar, os símbolos são escritos para o arquivo de saída sem modificação.
5. **Geração do Bloco Compactado:**
    - Para uma sequência que será compactada, o algoritmo escreve a seguinte sequência de símbolos no arquivo de saída:
        - O **símbolo indicador** (`FF` no exemplo).
        - O **símbolo que se repetiu** (ex: `00`).
        - O **contador de repetições** (ex: `07`, significando 7 repetições).
6. **Continuação:** O processo se repete até que todo o arquivo de entrada tenha sido lido e processado.

Esse método é eficaz para dados com longas sequências de símbolos idênticos, como imagens monocromáticas, dados de fax, ou arquivos com muitos zeros.

---

6. Descreva o processo de reversão / decodificação, no método Redução de Redundâncias.

O processo de reversão (descompressão ou decodificação) no método de Redução de Redundâncias é o inverso da codificação e é bastante direto:

1. **Inicialização:** O algoritmo de descompressão precisa saber qual é o **símbolo indicador/marcador** que foi utilizado durante a compressão (`FF` no nosso exemplo).
2. **Leitura Sequencial:** O algoritmo lê o arquivo compactado símbolo por símbolo, de forma sequencial.
3. **Verificação do Símbolo Indicador:**

- Para cada símbolo lido, o algoritmo verifica se ele é o **símbolo indicador** (`FF`).

4. **Processamento de Símbolos:**
    - **Se o símbolo lido NÃO for o símbolo indicador:** Ele é um símbolo de dado literal. O algoritmo simplesmente escreve esse símbolo diretamente no arquivo de saída (o arquivo descomprimido).
        - *Exemplo:* Se ele lê `12`, escreve `12`. Se lê `22`, escreve `22`.
    - **Se o símbolo lido FOR o símbolo indicador (`FF`):** Isso sinaliza que a próxima sequência de símbolos representa uma repetição compactada. O algoritmo então espera ler os dois símbolos seguintes:
        i. **Símbolo Repetido:** O algoritmo lê o próximo símbolo, que é o símbolo que foi repetido na sequência original (ex: `00`).
        ii. **Contador de Repetições:** O algoritmo lê o símbolo seguinte, que é o número de vezes que o símbolo anterior se repetiu (ex: `07`).
    - Uma vez que o símbolo repetido e o contador são lidos, o algoritmo **escreve o símbolo repetido no arquivo de saída o número de vezes indicado pelo contador**.
        - *Exemplo:* Se ele leu `FF 00 07`, ele escreve `00` sete vezes consecutivas no arquivo de saída.
5. **Continuação:** O processo se repete até que todo o arquivo compactado tenha sido lido, resultando na restauração do arquivo original.

Este processo garante que o arquivo descomprimido seja **idêntico bit a bit** ao arquivo original, característica da compressão *lossless*.

---

7. É possível realizar a pesquisa por um símbolo diretamente em um arquivo compactado pelo método de Redução de Redundâncias? Caso sim, descreva o funcionamento. Caso não, justifique.

Não, não é possível realizar uma pesquisa direta por um símbolo (ou uma sequência de símbolos) em um arquivo compactado pelo método de Redução de Redundâncias sem antes descompactar parcial ou totalmente o arquivo.

Justificativa:

O método de Redução de Redundâncias altera a representação dos dados. As sequências de repetição são substituídas por um formato `[Símbolo_Indicador][Símbolo_Repetido][Contador_de_Repetições]`. Isso significa que:

1. **Um símbolo original pode estar "escondido" em uma sequência compactada:** Se você procura pelo símbolo `00` no nosso exemplo, ele não aparece individualmente na sequência `FF 00 07`. Ele faz parte de uma estrutura de 3 bytes que o representa. Você teria que interpretar `FF`, depois `00` como o símbolo repetido, e `07` como o contador.
2. **Um byte pode ter múltiplos significados:** O byte `00` no arquivo compactado pode ser o símbolo `00` literal (se não foi compactado), ou pode ser o "símbolo repetido" dentro de uma tripla de compactação (`FF 00 XX`). Sem o contexto do símbolo `FF` que o precede, você não sabe o que ele representa.
3. **O "próximo" símbolo não é o próximo fisicamente:** Se você está procurando uma sequência como `00 12`, e o `00` estava compactado como `FF 00 07`, o `12` que vem *depois* dessa sequência compactada está fisicamente muito à frente no arquivo compactado, não adjacente ao `00` compactado.

Conclusão:

Para pesquisar um símbolo ou uma sequência, o algoritmo de pesquisa precisaria **emular o processo de descompressão**. Ele leria o arquivo compactado, e sempre que encontrasse o símbolo `FF`, ele "descompactaria" a sequência correspondente em sua memória temporária e então buscaria o símbolo desejado nessa representação expandida. Se o símbolo não fosse `FF`, ele simplesmente o compararia diretamente.

Este processo não é uma "pesquisa direta" no sentido tradicional de busca de strings em arquivos não compactados, pois exige a interpretação do formato de compressão. Basicamente, você está rodando parte do algoritmo de descompressão "em voo" para cada pedaço do arquivo.