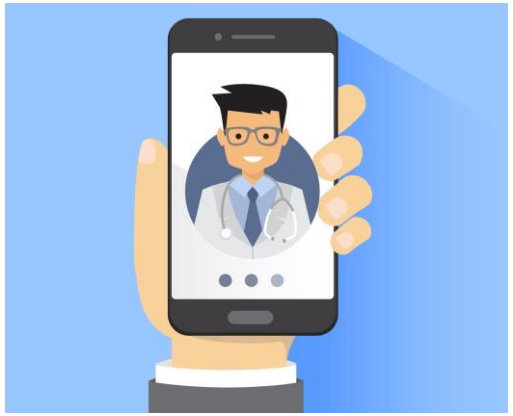
Arthur N. Guedes
Diego Vinicius da Silva
Eduardo M. de Souza
Gustavo Murayama
Hiago Lucas C. de Melo
Lucas Damazio da Costa
Lucas Tornai

TRANSMISSÃO DE ARQUIVOS MÉDICOS POR COMPRESSÃO DE HUFFMAN E RLE

TELEMEDICINA



- **Exercitar a medicina mediada através de conceitos tecnológicos**

  - Comunicação audiovisual e dados

- **PEP – Prontuário Eletrônico do Paciente**

  - Um dos formatos mais difundidos atualmente
  - Integrar históricos médicos de forma digital

- **No Brasil**

  - Implantação discutida desde 2002
  - Ritmo lento

TELEMEDICINA

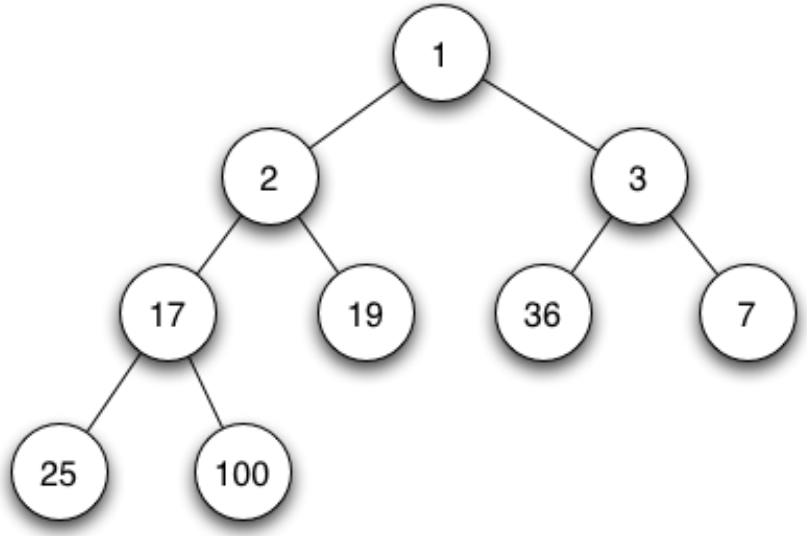
- **PEP requer transmissão de grande quantidade de dados**

  - Compressão desses dados facilita a sua transmissão e armazenamento
  - Compressão **sem perdas**
    - Pequena distorção no resultado do exame pode alterar o seu diagnóstico

- **DICOM (*Digital Imaging and Communications in Medicine*)**

  - Padronização internacional para **transmissão, armazenamento, impressão, processamento** e **apresentação** de informações visuais médicas
  - Interoperabilidade de informações e integração de equipamentos médicos de aquisição de imagem

LINGUAGEM E ESTRUTURA DE DADOS



Heap Mínimo - Pai menor ou igual a um elemento filho.

- **C# (C *Sharp*)**

  - Orientação a objetos

  - Muitas Bibliotecas

  - *Windows Forms*

- **HEAP**

  - Cria uma **fila de prioridades** (*heapify( )*)

  - Fácil ordenação e acesso direto para os valores de menor frequência (heap mínimo)

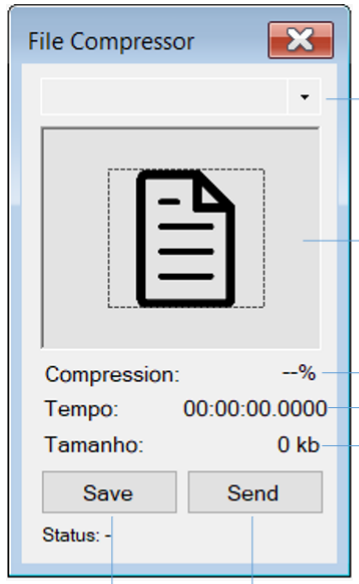
- **Árvore Binaria**

  - Árvore de prefixos de Huffman, usada na compactação e descompactação do arquivo

TRABALHOS RELACIONADOS

- **Análise e implementação de algoritmos de compressão de dados**
  - Aplicação do Algoritmo de Huffman
- **Compressão sem Perdas de sinais eletrocardiográficos**
  - Ruído do sinal captado interfere na compactação
  - Elege Huffman como o melhor método
- **Compressão de Imagens mamográficas**
  - Impacto da compactação JPEG2000 nos arquivos DICOM
  - Baixa eficiência do método RLE para tais arquivos

SOLUÇÃO



Algoritmo de compressão (Huffman ou RLE)

Arraste aqui arquivo a ser comprimido

Taxa de compressão = $\frac{\textit{Tamanho arquivo comprimido}}{\textit{Tamanho arquivo original}}$

Tempo de compressão

Tamanho do arquivo comprimido, em kilobytes

Enviar arquivo comprimido (via HTTP)

Salvar arquivo comprimido (.dat)

Interface *Windows Forms*

Implementação do algoritmo de Huffman

```
namespace Compression.Algorithms.Huffman
{
    public class HuffmanCoding : ICompressor
    {
        private int position;

        private long fileLength;

        public HuffmanCoding()
        {
            position = 0;
        }

        public decimal Percentage { get => fileLength > 0 ? (decimal)position / fileLength * 100 : 0; }

        public CompressedFile Compress(byte[] file)
        {
            position = 0;

            var frequency = file
                .GroupBy(b => b)
                .Select(g => new Node<byte> { Value = g.Key, Priority = g.Count() })
                .OrderBy(b => b.Priority)
                .ToList();

            var queue = CreatePriorityQueue(frequency);
            return Compress(file, queue);
        }

        private CompressedFile Compress(byte[] file, MinHeap<byte> queue)
        {
            fileLength = file.LongLength;

            var queueAsArray = queue.ExportQueueAsArray();
            var huffmanTree = BuildHuffmanTree(queue);
            var dictionary = BuildCodingDictionary(huffmanTree);

            var biggestCode = dictionary[queueAsArray[0].Value];
            var bits = new BitArray(biggestCode.Count * file.Length);
            int totalLength = 0;
```

Implementação do algoritmo RLE

```
namespace Compression.Algorithms.RunLengthEncoding
{
    public class RunLengthEncodingCompressor : ICompressor
    {
        private const byte MAX_PACKAGE_SIZE = 255;

        private int position;

        private long fileLength;

        public decimal Percentage { get => fileLength > 0 ? (decimal)position / fileLength * 100 : 0; }

        public CompressedFile Compress(byte [] file)
        {
            fileLength = file.LongLength;
            List<byte> newFile = new List<byte>();

            byte runValue = file[0];
            byte runCount = 1;
            for (position = 1; position < file.Length; position++)
            {
                byte currentByte = file[position];
                if(runValue == currentByte)
                {
                    if(runCount != MAX_PACKAGE_SIZE)
                        runCount++;
                    else
                    {
                        newFile.Add(runCount);
                        newFile.Add(runValue);
                        runCount = 1;
                    }
                }
                else
                {
                    newFile.Add(runCount);
                    newFile.Add(runValue);
                    runValue = currentByte;
                    runCount = 1;
                }
            }
```

SOLUÇÃO

Algoritmo de HUFFMAN:

1. Leitura do arquivo byte a byte e construção de um vetor contendo informações sobre o byte e sua frequência;

2. Construção do HEAP mínimo utilizando-se o vetor de frequências;

3. Construção da árvore de Huffman utilizando-se o HEAP mínimo;

4. Com base na árvore de Huffman obtida, foi criado um dicionário que mapeia um byte para sua nova codificação em bits.

5. Criação de vetor de bits vazio para armazenar o arquivo comprimido.

6. Leitura byte a byte do arquivo mapeando cada byte com o dicionário criado no passo 4 para seu novo prefixo e seu armazenamento no vetor criado no passo 5.

7. Transformação do vetor de bits que armazena o arquivo comprimido para bytes.

Algoritmo RLE (Run-length encoding)

1. Criação de vetor vazio para armazenar a tupla contendo a quantidade de vezes que um byte aparece e o literal do byte;

2. Leitura do arquivo byte a byte do arquivo;

3. Ao encontrar um byte diferente do lido anteriormente adição no vetor da quantidade de repetições do byte juntamente com o literal do byte.

RESULTADOS

	Tamanho	T_HUFF	Taxa_HUFF	T_RLE	Taxa_RLE
CT-MONO2-8-abdo	257KB	48ms	47,23%	2ms	70,06%
CT-MONO2-16-ankle	514KB	62ms	33,46%	9ms	199,83%
CT-MONO2-16-brain	514KB	112ms	68,51%	8ms	199,66%
CT-MONO2-16-chest	142KB	47ms	99,77%	2ms	198,32%
CT-MONO2-16-ort	514KB	104ms	63,59%	9ms	199,53%

- **RLE - Arquivo comprimido na maior parte possui quase o dobro de tamanho do original**

  - Imagens já se encontram comprimidas (JPEG Lossless)

- **Huffman**

  - Velocidade e taxa de compressão satisfatória

  - Melhor método para comprimir tomografias de padrão DICOM

Tomografias computadorizadas DICOM

CONCLUSÃO

- O algoritmo de Huffman apresentou uma média de taxa de compressão de 62,51% nos teste realizados, sendo melhor ao RLE
- A compactação JPEG2000 padrão da DICOM reduz significativamente a eficiência do método RLE
- Pelo algoritmo de Huffman, os arquivos compactados que apresentaram maior razão Bit/segundo, obtiveram maior taxa de compressão
- **Trabalhos futuros**
  - Explorar melhor os arquivos no padrão DICOM e aplicar novos métodos de compressão