% Tạo dữ liệu mẫu với tần suất xuất hiện khác nhau

originalData = ['ABBCCCDDDDEEEEEFFFFFF'];

% In ra dữ liệu gốc

disp('--- Original Data ---');

disp(['Dữ liệu gốc: ' originalData]); % In ra chuỗi ký tự gốc

disp(['Kích thước dữ liệu gốc: ' num2str(length(originalData)) ' ký tự']);

disp(['Kích thước dữ liệu gốc: ' num2str(length(originalData) \* 1) ' bytes']); % Mỗi ký tự chiếm 1 byte

%% === Run Length Encoding (RLC) ===

disp('--- Run Length Encoding (RLC) ---');

tic; % Bắt đầu đếm thời gian

compressedRLC = runLengthEncode(originalData);

timeRLC = toc \* 1000; % Chuyển đổi thời gian sang mili giây

disp(['Kích thước dữ liệu nén RLC: ' num2str(length(compressedRLC)) ' ký tự']);

disp(['Kích thước dữ liệu nén RLC: ' num2str(length(compressedRLC) \* 1) ' bytes']); % Mỗi ký tự chiếm 1 byte

disp(['Thời gian nén RLC (ms): ' num2str(timeRLC)]);

disp(['Tỷ lệ nén RLC (%): ' num2str((length(compressedRLC)/length(originalData))\*100)]);

%% === Huffman Encoding (Alternative) ===

disp('--- Huffman Encoding (Alternative) ---');

tic; % Bắt đầu đếm thời gian

[compressedHuffman, huffmanCodes] = huffmanEncodeAlternative(originalData);

timeHuffman = toc \* 1000; % Chuyển đổi thời gian sang mili giây

disp(['Kích thước dữ liệu nén Huffman: ' num2str(length(compressedHuffman)) ' bits']);

disp(['Kích thước dữ liệu nén Huffman: ' num2str(length(compressedHuffman) / 8) ' bytes']); % Chuyển đổi bits sang bytes

disp(['Thời gian nén Huffman (ms): ' num2str(timeHuffman)]);

disp(['Tỷ lệ nén Huffman (%): ' num2str((length(compressedHuffman)/ (length(originalData) \* 8)) \* 100)]);

%% === Kết quả So sánh ===

disp('--- Kết quả So sánh ---');

disp(['Kích thước dữ liệu gốc: ' num2str(length(originalData)) ' ký tự']);

disp(['Kích thước nén RLC: ' num2str(length(compressedRLC)) ' ký tự, tương ứng ' num2str(length(compressedRLC) \* 1) ' bytes']);

disp(['Kích thước nén Huffman: ' num2str(length(compressedHuffman)) ' bits, tương ứng ' num2str(length(compressedHuffman) / 8) ' bytes']);

disp(['Thời gian xử lý RLC (ms): ' num2str(timeRLC)]);

disp(['Thời gian xử lý Huffman (ms): ' num2str(timeHuffman)]);

disp(['Hiệu quả nén RLC (%): ' num2str((length(compressedRLC) / length(originalData)) \* 100)]);

disp(['Hiệu quả nén Huffman (%): ' num2str((length(compressedHuffman) / (length(originalData) \* 8)) \* 100)]);

%% === Hàm Run Length Encoding (RLC) ===

function encoded = runLengthEncode(data)

encoded = '';

count = 1;

for i = 2:length(data)

if data(i) == data(i-1)

count = count + 1;

else

encoded = [encoded, data(i-1), num2str(count)];

count = 1;

end

end

encoded = [encoded, data(end), num2str(count)];

end

%% === Hàm Huffman Encoding (Alternative) ===

function [compressedData, codes] = huffmanEncodeAlternative(data)

symbols = unique(data);

freq = zeros(size(symbols));

for i = 1:length(symbols)

freq(i) = sum(data == symbols(i));

end

huffmanTree = createHuffmanTree(symbols, freq);

codes = getHuffmanCodes(huffmanTree);

compressedData = encodeData(data, codes);

end

function tree = createHuffmanTree(symbols, freq)

nodes = arrayfun(@(sym, frq) struct('symbol', sym, 'freq', frq, 'left', [], 'right', []), symbols, freq);

while length(nodes) > 1

nodes = sortStructArray(nodes, 'freq');

newNode = struct('symbol', '', 'freq', nodes(1).freq + nodes(2).freq, 'left', nodes(1), 'right', nodes(2));

nodes = [newNode, nodes(3:end)];

end

tree = nodes;

end

function codes = getHuffmanCodes(tree, prefix)

if nargin < 2

prefix = '';

end

if isempty(tree.left) && isempty(tree.right)

codes.(tree.symbol) = prefix;

else

codes = struct();

if ~isempty(tree.left)

codesLeft = getHuffmanCodes(tree.left, [prefix '0']);

codes = mergeStructs(codes, codesLeft);

end

if ~isempty(tree.right)

codesRight = getHuffmanCodes(tree.right, [prefix '1']);

codes = mergeStructs(codes, codesRight);

end

end

end

function compressedData = encodeData(data, codes)

compressedData = '';

for i = 1:length(data)

compressedData = [compressedData codes.(data(i))];

end

end

function merged = mergeStructs(struct1, struct2)

merged = struct1;

fields = fieldnames(struct2);

for i = 1:length(fields)

merged.(fields{i}) = struct2.(fields{i});

end

end

function sorted = sortStructArray(array, fieldName)

[~, indices] = sort([array.(fieldName)]);

sorted = array(indices);

end