Trabalho Prático 1 – Algoritmos 2

Paulo Palmuti Sigiani Neto

Departamento de Ciências da Computação – Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte – Brazil

1. Introdução

Essa documentação lida com o problema sugerido pela disciplina de Algoritmos II no semestre 2023/1. O problema se trata de implementar o algoritmo de compressão LZ78, usando uma Trie como estrutura de dados de suporte para o dicionário.

2. Implementação

2.1 Organização do Código

O código está organizado da seguinte maneira:

- **Main.py:** Arquivo principal do código, onde recebemos e tratamos a entrada e saída para que seja possível aplicar os algoritmos.
- **Compression.py:** Arquivo onde estão implementados os algoritmos de codificação e descodificação do LZ78
- **Trie.py:** Arquivo que implementa a estrutura de dados da Trie.

2.2 Estruturas de dados e procedimentos principais

2.2.1 Compressão

Primeiramente, lemos o arquivo de texto como binário, e convertemos para uma string composta de 0's e 1's. Então começamos o procedimento de compressão.

Para comprimir foi desenvolvida a função `encoding`, que funciona da seguinte maneira:

1 – Inicializamos uma string vazia, **P**, e lemos o próximo caractere do texto, **c,** e iniciamos o contador de prefixos com o valor 0.

2 – Procuramos por **P+c** no dicionário. Se **P+c** estiver lá definimos **P=P+c,** e memorizamos o número do prefixo à que essa string corresponde, e vamos para o próximo passo.

Se **P+c** não estiver no dicionário, então incrementamos o contador de prefixos e inserimos essa string no dicionário da trie, tendo o número do prefixo ao qual essa string corresponde como rótulo.

E colocamos no código uma tupla formada pelo número do prefixo correspondente a **P** no dicionário, e o símbolo **c**, ou seja, a tupla é **(número do prefixo, c)** . Se **P** é vazia colocamos 0 como número correspondente. Por último, resetamos **P** para a string vazia.

3- Se o texto não chegou ao final, repetimos o passo **2**. Se tiver chegado e **P** está vazia, acabamos. Se **P** tem símbolos então adicionamos uma tupla no código com o número do prefixo que **P** corresponde e o símbolo vazio.

Como só temos dois símbolos no alfabeto (0 e 1) a nossa trie é binária. Para inserir uma palavra, lemos o próximo símbolo dela, e navegamos para o índice da lista de filhos referente ao símbolo. Se estiver vazio, criamos um novo nó e rotulamos com o número do prefixo. Note que só inserimos 1 símbolo por vez, já que toda palavra que é adicionada é um prefixo já encontrado acrescido de um símbolo. Para pesquisar é o mesmo raciocínio, mas quando encontramos a string retornamos o seu rótulo, ou então, se não encontrarmos, retornamos -1.

No final teremos uma lista de tuplas do código. Depois salvamos em z78, no formato binário (mais detalhes sobre isso a frente).

2.2.2 Descompressão

Para descompressão lemos o arquivo .z78, convertemos a sequência binária para uma lista de tuplas. Então:

1 – Inicializamos uma lista onde iremos salvar os prefixos decodificados. Selecionamos a primeira tupla no código e então:

2 – Para uma tupla **(n, c),** Pegamos **n**-ésimo prefixo no dicionário. Se for 0, o prefixo é a palavra vazia. Concatenamos esse prefixo **W** no texto descomprimido, e depois concatenamos o símbolo **c**. Então adicionamos **W+c** ao dicionário, associado com o número da iteração atual (ou seja, o momento de descoberta do prefixo). Ele vai acabar sendo o mesmo da trie na compressão.

3 – Se ainda tem tuplas na lista, repetimos 2. Se não retornamos o texto descomprimido.

2.2.3 Codificação do código comprimido

Primeiro calculamos o mínimo de bits necessário para representar o código dos prefixos usando a fórmula $[log_2(número\ de\ prefixos)]$. Então converte-se o índice das tuplas nos códigos para binário.

Guardamos em 5 bits o número de bits usados para representar os inteiros. Isso permite que usemos até 31 bits para representar os códigos dos prefixos. Logo, podemos ter até 2^{31} prefixos salvos.

Como precisamos salvar a sequência binária em bytes para salvar em um arquivo binário, precisamos que o tamanho da string seja múltiplo de 8. Calculamos quantos bits faltam para que ela tenha esse tamanho. Pode ser de 0 até 7. Vamos usar 3 bits pra salvar esse número de bits faltantes para o comprimento ser múltiplo de 8. Então calculamos *(tamanho da string + 3) mod 8*. Adicionamos uma sequência de 1's desse tamanho no final da string, e depois adicionamos a representação em 3 bits do número de 1's adicionado. Então o código fica o seguinte:

- 5 primeiros bits: número de bits usado para representar os inteiros correspondentes aos prefixos.
- Códigos da tuplas do código. Representamos o número do prefixo com o número de bits mínimo calculado, e representamos o símbolo do código com 1 bit (pois é 0 ou 1)
- Sequência de 1's necessária pra a string ter tamanho múltiplo de 8.
- 3 bits representando o número de 1's utilizado

A decodificação segue a lógica inversa, com um detalhe: Se tirarmos os bits que não correspondem as tuplas, e o que sobrar não é múltiplo do número de bits usado para os inteiros + 1, quer dizer que não colocamos um símbolo na última tupla do código, logo, o símbolo dessa tupla é a palavra vazia.