

Universidade Federal da Paraíba - UFPB

Centro de Informática - CI

Curso de Engenharia da Computação

Introdução a Teoria da Informação - ITI: Parte 1 do Trabalho Prático

Alunos: Epitácio Neto, Guilherme Moreira, Kenji Sato

Matrículas: 11506856, 20160105205, 11514918

Professor: Derzu Omaia

Introdução

Este relatório tem como proposta a implementação, na linguagem python, de um compressor e descompressor de diferentes tipos de dados utilizando o algoritmo Lempel-Ziv-Welch (LZW). O LZW é um algoritmo de compressão derivado do do algoritmo LZ78, que se baseia na localização e no registro dos padrões de uma determinada estrutura de dados. O algoritmo foi desenvolvido e patenteado em 1984 por Terry Welch.

Descrição do Problema

Parte 1:

Implemente um compressor e descompressor utilizando o PPM-C (visto em sala de aula) com o codificador aritmético, ou utilizando o algoritmo LZW. Considere que as mensagens são geradas por fontes com alfabeto $A = \{0, 1, \dots, 255\}$. Teste o compressor/descompressor com um corpus de texto em português de 16MB e com um arquivo binário de vídeo.

PPM-C + Aritmético:

O contexto deve ter tamanho máximo K (parâmetro). O modelo PPM-C alimentará um codificador aritmético. Utilize o mecanismo de exclusão quando necessário. No relatório apresente as curvas de $RC \times K$ e de Tempo de Processamento $\times K$, para $K = 0, 1, 2, 3, 4, 5, 6, 7, 8$. Não é necessário implementar o codificador aritmético, utilize algum já existente.

LZW:

O índice do dicionário deve ser testado com diferentes tamanhos K bits (parâmetro). Exemplo: $K=9$ bits tamanho do dicionário: $2^9=512$, $K=10$ bits tamanho do dicionário $2^{10}=1024$. No relatório apresente as curvas de $RC \times K$ e de Tempo de Processamento $\times K$, para $K = 9, 10, 11, 12, 13, 14, 15, 16$ bits. Indique também a quantidade total de índices presentes na mensagem final para cada K .

Observações:

- Os símbolos do arquivo de teste devem ser lidos no modo binário (números) e não no modo texto (caracteres/strings).
- O codificador deve receber como entrada um arquivo e gerar como saída o arquivo codificado.
- A execução dos experimentos é demorada, evite fazer os experimentos na véspera da entrega pois não dará tempo.

Ferramentas Utilizadas

A implementação total do algoritmo, tanto o compressor como o descompressor, foi feita em python utilizando o Visual Studio Code como ambiente de desenvolvimento. No VSCode realizamos todos os testes que serão apresentados nas próximas sessões deste relatório. Para os testes utilizamos dois arquivos de entrada diferentes disponibilizados pelo Professor Derzu, um arquivo .txt de 16MB e um vídeo .mp4 de aproximadamente 2MB.

Desenvolvimento

Bibliotecas

```
import sys
from sys import argv
import struct
from struct import *
```

In []:

```
import time
import os
from typing import Sized
import numpy as np
```

Funções

Encoder

In []:

```
def encoder(data, max_tam):
    #iniciando dicionario
    inicio = time.time()
    dicionario_tam = 256
    dicionario = {chr(i): i for i in range(dicionario_tam)}
    string = ""
    dados_comprimido = [] #armazena os dados comprimidos.

    #comeca LZW | Interando por cada indice no arquivo de entrada
    for indices in data:
        concat_indices = string + indices #concatena os indices
        if concat_indices in dicionario: #verifica se tem o indice atual no dicionario
            string = concat_indices #armazena o indice atual em string
        else:
            dados_comprimido.append(dicionario[string]) #adiciona a saida o indice atual
            if(len(dicionario) <= max_tam): #verifica se o dicionario esta cheio
                dicionario[concat_indices] = dicionario_tam #adiciona ao dicionario o novo simbolo caso c
                dicionario_tam += 1
            string = indices

    if string in dicionario:
        dados_comprimido.append(dicionario[string])

    #armazenando a string comprimida no arquivo de saida byte a byte
    out = input_file.split(".")[0]
    output_file = open(out + "_" + str(max_tam) + "_saida.lzw", "wb")
    for data in dados_comprimido:
        output_file.write(pack('>H',int(data)))

    print("RC para " + str(k) + " Bits ", len(str(data)) / len(str(output_file)))
    fim = time.time()
    print("Tempo de processamento para " + str(k) + " Bits", fim - inicio)
    output_file.close()
    file.close()
    return output_file
```

Decoder

In [2]:

```
def decoder(file, k):

    compressed_data = []
    next_code = 256
    decompressed_data = ""
    string = ""

    while True:
        rec = file.read(2)
        if len(rec) != 2:
            break
        (data, ) = unpack('>H', rec)
        compressed_data.append(data)

    dictionary_size = 256
    dictionary = dict([(x, chr(x)) for x in range(dictionary_size)])

    for code in compressed_data:
        if not (code in dictionary):
            dictionary[code] = string + (string[0])
            decompressed_data += dictionary[code]
        if not(len(string) == 0):
            dictionary[next_code] = string + (dictionary[code][0])
            next_code += 1
        string = dictionary[code]
```

```

out = input_file.split(".")[0]
output_file = open(out + "_decoded.txt", "w")
for data in decompressed_data:
    output_file.write(data)

output_file.close()
file.close()

```

Resultados

Exemplo 1: Texto

Encoder

In []:

```

input_file = argv[1]
file = open(input_file, "rb")
data = str(file.read())
tamanho_or = file.tell()

print("Quantidade de indices no arquivo de entrada: " + str(len(data)))
print("Tamanho do arquivo de entrada: " + str(tamanho_or) + " KB")
for k in range(9,16):
    max_tam = pow(2,int(k))
    encoder(data, max_tam)

```

PS C:\Users\gmcore\Desktop\ITI\projeto_final> python encoder_lzw.py corpus16MB.txt

Quantidade de indices no arquivo de entrada: 17127545

Tamanho do arquivo de entrada: 15637070 KB

k = 9

Total de indices no arquivo comprimido com 9bits :46835591

RC para 9 Bits 0.2692307692307692

Tempo de processamento para 9 Bits 10.048340797424316 segundos

Tamanho do arquivo comprimido para 9bits 19415564 KB

k = 10

Total de indices no arquivo comprimido com 10bits :37589329

RC para 10 Bits 0.2641509433962264

Tempo de processamento para 10 Bits 8.643194913864136 segundos

Tamanho do arquivo comprimido para 10bits 15280240 KB

k = 11

Total de indices no arquivo comprimido com 11bits :33698098

RC para 11 Bits 0.2641509433962264

Tempo de processamento para 11 Bits 7.7520363330841064 segundos

Tamanho do arquivo comprimido para 11bits 12677260 KB

k = 12

Total de indices no arquivo comprimido com 12bits :30554609

RC para 12 Bits 0.2641509433962264

Tempo de processamento para 12 Bits 7.402488708496094 segundos

Tamanho do arquivo comprimido para 12bits 11028876 KB

k = 13

Total de indices no arquivo comprimido com 13bits :27664108

RC para 13 Bits 0.2641509433962264

Tempo de processamento para 13 Bits 6.967217206954956 segundos

Tamanho do arquivo comprimido para 13bits 9679422 KB

k = 14

Total de indices no arquivo comprimido com 14bits :26147272

RC para 14 Bits 0.25925925925925924

Tempo de processamento para 14 Bits 6.998999357223511 segundos

Tamanho do arquivo comprimido para 14bits 8612500 KB

k = 15

Total de indices no arquivo comprimido com 15bits :24689819

RC para 15 Bits 0.25925925925925924

Tempo de processamento para 15 Bits 8.698994159698486 segundos

Tamanho do arquivo comprimido para 15bits 7709516 KB

Abaixo podemos ver o resultado do tamanho final de cada arquivo com o seu tamanho do dicionario definido anteriormente:


```
for k in range(9,16):
    max_tam = pow(2,int(k))
    encoder(data, max_tam)
```

```
PS C:\Users\lgmore\Desktop\ITI\projeto_final> python encoder_lzw.py disco.mp4
Quantidade de indices no arquivo de entrada: 6132338
Tamanho do arquivo de entrada: 2111047 KB

k = 9
Total de indices no arquivo comprimido com 9bits :17647161
RC para 9 Bits 0.19148936170212766
Tempo de processamento para 9 Bits 3.671369791030884 segundos
Tamanho do arquivo comprimido para 9bits 7661566 KB

k = 10
Total de indices no arquivo comprimido com 10bits :13809170
RC para 10 Bits 0.1875
Tempo de processamento para 10 Bits 3.397016763687134 segundos
Tamanho do arquivo comprimido para 10bits 5843632 KB

k = 11
Total de indices no arquivo comprimido com 11bits :12823246
RC para 11 Bits 0.1875
Tempo de processamento para 11 Bits 3.24899959564209 segundos
Tamanho do arquivo comprimido para 11bits 4984762 KB









k = 12
Total de indices no arquivo comprimido com 12bits :12070011
RC para 12 Bits 0.1875
Tempo de processamento para 12 Bits 2.714879274368286 segundos
Tamanho do arquivo comprimido para 12bits 4473130 KB

k = 13
Total de indices no arquivo comprimido com 13bits :10573327
RC para 13 Bits 0.1875
Tempo de processamento para 13 Bits 2.570004463195801 segundos
Tamanho do arquivo comprimido para 13bits 3806828 KB

k = 14
Total de indices no arquivo comprimido com 14bits :9785835
RC para 14 Bits 0.1836734693877551
Tempo de processamento para 14 Bits 2.3509416580200195 segundos
Tamanho do arquivo comprimido para 14bits 3268226 KB

k = 15
Total de indices no arquivo comprimido com 15bits :9260379
RC para 15 Bits 0.1836734693877551
Tempo de processamento para 15 Bits 2.4850008487701416 segundos
Tamanho do arquivo comprimido para 15bits 2913552 KB
```

Para o vídeo podemos comparar também o resultado final de cada k bits ao tamanho original na sua compressão:

Nome	Data de modificação	Tipo	Tamanho
 disco	11/05/2021 17:27	Arquivo MP4	2.062 KB
 disco_512_saida	17/05/2021 13:22	Arquivo LZW	7.482 KB
 disco_1024_saida	17/05/2021 13:22	Arquivo LZW	5.707 KB
 disco_2048_saida	17/05/2021 13:22	Arquivo LZW	4.868 KB
 disco_4096_saida	17/05/2021 13:22	Arquivo LZW	4.369 KB
 disco_8192_saida	17/05/2021 13:22	Arquivo LZW	3.718 KB
 disco_16384_saida	17/05/2021 13:22	Arquivo LZW	3.192 KB
 disco_32768_saida	17/05/2021 13:22	Arquivo LZW	2.846 KB

Decoder

```
input_file, k = argv[1:]
maximum_table_size = pow(2,int(k))
file = open(input_file, "rb")
decoder(file, maximum_table_size)
```

Podemos comparar a saída da função de descompressão com o arquivo comprimido visualizando seus arquivos:

In []:



Conclusão

Com os resultados apresentados acima é visto que para o arquivo .txt de 16MB conseguimos ter uma boa razão de compressão, principalmente quando o tamanho do dicionário é definido para 16 K bits quando temos um arquivo de saída de apenas 968 KB comparado aos 16 MB do seu arquivo original. Para o arquivo .mp4 não obtivemos resultados excelentes em sua compressão. Podemos ver que o melhor resultado, levando em conta apenas o tamanho final do arquivo de saída, foi de aproximadamente 2846 KB com um dicionário de 16 k bits, apresentando um valor ainda maior em relação ao seu arquivo de entrada que é de apenas 2062 KB.

Referências Bibliográficas

- ITECHNICA. 6. Dynamic Dictionary - LZW Encoding. Youtube, 9 de Dez. de 2017. Disponível em: https://www.youtube.com/watch?v=14wpPMN-0Fw&ab_channel=itechnica. Acesso em: 15 de Mai. de 2021.
- ITECHNICA. 7. Dynamic Dictionary - LZW Decoding. Youtube, 9 de Dez. de 2017. Disponível em: https://www.youtube.com/watch?v=pRxiCYwclqg&ab_channel=itechnica. Acesso em: 15 de Mai. de 2021.
- UNIVERSIDADE ESTADUAL DE CAMPINAS - Unicamp. Algoritmo LZW (Lempel-Ziv-Welch). Disponível em: https://www.decom.fee.unicamp.br/dspcom/EE088/Algoritmo_LZW.pdf. Acesso em: 16 de Mai. de 2021.
- SAKIA, Amartya Ranjan. LZW (Lempel-Ziv-Welch) Compression technique, 9 de Set. de 2019. Disponível em: <https://www.geeksforgeeks.org/lzw-lempe-ziv-welch-compression-technique/>. Acesso em: 16 de Mai. de 2021.
- LEMPEL-ZIV-WELCH. In: Wikipédia: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch>. Acesso em: 16 de Mai. de 2021.
- OMAIA, Derzu. Codificadores baseados em Dicionário, 13-15 de Set. de 2021. Notas de Aula.