TEC | Tecnológico de Costa Rica

TECNOLÓGICO DE COSTA RICA

Instituto Tecnológico de Costa Rica

Área Académica Ingeniería en Computación

Ingeniería en Computación

IC1802 - Taller de Programación

Grupo: 21

Compresíon con Árboles de Huffman

Fecha de entrega: 17 de junio de 2024

Profesor:

Alex Brenes Brenes

Autores:
Joseph Iván Monge Monge
Oscar Daniel Rojas Rodriguez

Lugar:

Sede Alajuela Tecnológico De Costa Rica

Índice

IC1802 - Taller de Programación
Compresíon con Árboles de Huffman

Joseph Iván Monge Monge
Oscar Daniel Rojas Rodriguez

17 de junio de 2024

1. Introducción

El proyecto denominado como 'Compresión con Árboles de Huffman'tiene como objetivo principal demostrar la capacidad de los estudiantes en el uso de la estructura de árboles, listas y recursión en particular programadas utilizando el lenguaje Python3 en el manejo y administración de archivos. El codigo del compresor funcionara cuando un archivo de tipo txt es llamado como parametro, este es comprimido en diferentes archivos los cuales son de tipo '.huff', '.table' y '.stats', estos mismos archivos son utilizados en el descompresor para convertir de esos tres archivos al original.

2. Desarrollo

2.1. Descripción del problema

decir que el problema es hacer un compresor y descompresion de archivos con arboles de huffman

2.2. Investigación corta

2.2.1. Biografiía de David A. Huffman

David Albert Huffman (9 de agosto de 1925 - 7 de octubre de 1999) fue un destacado científico estadounidense conocido por sus contribuciones fundamentales al campo de la codificación de datos. Nacido en Ohio, Huffman demostró un interés temprano por las matemáticas y la ingeniería, culminando sus estudios con una licenciatura en Ingeniería Eléctrica de la Universidad Estatal de Ohio en 1945, seguida de un doctorado obtenido en el Instituto de Tecnología de Massachusetts (MIT) en 1953.

Huffman es famoso por desarrollar el algoritmo de codificación Huffman, que revolucionó la compresión de datos al introducir un método eficiente para asignar códigos binarios a caracteres según su frecuencia de aparición [1]

Además de su trabajo pionero en codificación, Huffman realizó investigaciones significativas en teoría de la información y cibernética, estableciéndose como una figura influyente en su campo. Su legado perdura como una piedra angular en la ciencia computacional y la teoría de la información. Huffman falleció el 7 de octubre de 1999, dejando un impacto perdurable en su disciplina y más allá.

2.2.2. Compresión de archivos con pérdida y sin pérdida

La compresión de datos es fundamental en el manejo eficiente de archivos digitales, existiendo dos enfoques principales: la compresión con pérdida y la compresión sin pérdida.

La compresión con pérdida: Esta reduce el tamaño del archivo eliminando detalles redundantes o menos perceptibles, típicamente utilizada en medios multimedia como imágenes, audio y video (¿Cómo evalúa las compensaciones entre los métodos de compresión de datos con pérdida y sin pérdida?, 2023). [2] Este método sacrifica precisión en pos de la reducción de tamaño, siendo ideal para aplicaciones donde la fidelidad exacta no es crucial, pero la eficiencia de almacenamiento es prioritaria.

La compresión sin pérdida: Esta preserva todos los datos originales al reducir redundancias y patrones repetitivos, manteniendo la integridad del archivo. Este enfoque es esencial en aplicaciones donde la exactitud y la restauración perfecta de los datos son críticas, como en la compresión de archivos de texto y bases de datos [2].Aunque la compresión sin pérdida generalmente logra tasas de compresión más bajas que la compresión con pérdida, garantiza que no se pierda información importante durante el proceso..

2.2.3. Descripción de la estructura abstracta Trie

Una estructura de datos Trie es un tipo de árbol digital que se utiliza principalmente para almacenar y recuperar conjuntos de cadenas de manera eficiente [3]. También conocido como árbol de prefijos, un Trie organiza las cadenas de manera que los nodos representan caracteres individuales. Cada ruta desde la raíz hasta un nodo hoja forma una cadena completa almacenada en la estructura. Esta característica hace que los Tries sean especialmente eficientes para operaciones como la búsqueda de prefijos y la inserción de nuevas cadenas, optimizando el tiempo de acceso y consumo de memoria al agrupar y compartir prefijos comunes entre las cadenas almacenadas.

2.2.4. Descripción de la biblioteca bitarray

Esta biblioteca proporciona un tipo de objeto que representa de manera eficiente una matriz de valores booleanos. Los bitarrays son tipos de secuencia y se comportan de forma muy parecida a las listas habituales. Ocho bits están representados por un byte en un bloque de memoria contiguo. El

usuario puede seleccionar entre dos representaciones: little-endian y big-endian. Toda la funcionalidad se implementa en C. Se proporcionan métodos para acceder a la representación de la máquina, incluida la capacidad de importar y exportar buffers. Esto permite crear matrices de bits que se asignan a otros objetos, incluidos archivos asignados en memoria [4].

2.2.5. Descripción de las funciones bitarray, tofile y fromfile de bitarray

La biblioteca Bitarray en Python incluye varias funciones clave que permiten la manipulación y el manejo eficiente de datos a nivel de bits [4]. A continuación se describen brevemente tres de sus funciones principales: bitarray, tofile y fromfile.

1. bitarray: La función bitarray crea un arreglo de bits. Puede ser inicializada con una secuencia de bits o con un tamaño específico, en cuyo caso todos los bits se inicializan en 0. Esta función es fundamental para la creación y manipulación de estructuras de datos binarios compactos.

2. tofile: La función tofile permite escribir el contenido del bitarray en un archivo. Es especialmente útil para almacenar datos binarios de manera eficiente en disco, preservando la estructura compacta del bitarray

3. fromfile: La función fromfile lee los datos binarios de un archivo y los almacena en un bitarray. Esto facilita la recuperación de datos binarios previamente almacenados y su manipulación posterior.

2.3. Descripción de la solución

2.3.1. Compresor

```
import os
import sys
import heapq
class Node:
    def __init__(self, char, freq, left=None, right=None):
        self.char = char
        self.freq = freq
        self.left = left
        self.right = right

    def __lt__(self, other):
        return self.freq < other.freq

    def __str__(self):
        return f"({self.char},{self.freq})"
```

```
def tbfreq(texto):
    freq = {}
    for char in texto:
        if char in freq:
            freq[char] += 1
        else:
            freq[char] = 1
    return freq

def creaarbol(frequency_table):
    heap = [Node(char, freq) for char, freq in frequency_table.items()]
    heapq.heapify(heap)
    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        parent = Node(None, left.freq + right.freq, left, right)
        heapq.heappush(heap, parent)
    return heapq.heappop(heap)

def atravesar(node, code, algo):
    if node.char is not None:
        algo[node.char] = code
    else:
        atravesar(node.left, code + "0", algo)
        atravesar(node.right, code + "1", algo)

def codigos(a):
    tabla = {}
    atravesar(a, "", tabla)
    return tabla

def comprimir(input, output, tabla, stats):
    with open(input, 'r') as f:
        text = f.read()

    frecu = tbfreq(text)
    arbol = creaarbol(frecu)
    codi = codigos(arbol)

    data = ''.join([codi[char] for char in text])
```

```
    with open(output, 'w') as f:
        f.write(data)

    with open(tabla, 'w') as f:
        for char, code in codi.items():
            f.write(f"{char}: {code}\n")

    altu = altura(arbol)
    anch = ancho(arbol)
    nodosn = npn(arbol)

    with open(stats, 'w') as f:
        f.write(f"Altura del  rbol : {altu}\n")
        f.write(f"Anchura del  rbol : {anch}\n")
        f.write(f"Cantidad de nodos por nivel: {nodosn}\n")
        f.write("Tabla de frecuencias original:\n")
        for char, freq in frecu.items():
            if freq > 0:
                f.write(f"{char}: {freq}\n")

    tama ov = os.path.getsize(input)
    tama on = len(data)
    print(f"Compresi n exitosa. Archivo comprimido: {output}")
    print(f"Compresi n Ratio: {tama on / tama ov:.2f}")

def altura(A):
    if not A:
        return 0
    return 1 + max(altura(A.left), altura(A.right))

def ancho(A):
    if not A:
        return 0
    return 1 + ancho(A.left) + ancho(A.right)

def npn(A):
    nodes_per_level = {}
    recorrer(A, 0, nodes_per_level)
    return nodes_per_level
```

```
    def recorrer(nodo, level, A):
        if nodo is None:
            return
        if level in A:
            A[level] += 1
        else:
            A[level] = 1
        recorrer(nodo.left, level + 1, A)
        recorrer(nodo.right, level + 1, A)

    if __name__ == "__main__":
        if len(sys.argv) != 2:
            print("Se usa asi: python compresor.py <nombre_archivo>")
            sys.exit(1)

        input = sys.argv[1]
        output = os.path.splitext(input)[0] + ".huff"
        tabla = os.path.splitext(input)[0] + ".table"
        stats = os.path.splitext(input)[0] + ".stats"

        comprimir(input, output, tabla, stats)
```

Descompresor

```
import sys
import os
def leetabla(tabla):
    tb = {}
    with open(tabla, 'r') as f:
        for line in f:
            char, code = line.strip().split(": ")
            tb[code] = char
    return tb

def descomprimir(comprimido, tabla, output):
    libcodigos = leetabla(tabla)
    with open(comprimido, 'r') as f:
        infocomprimida = f.read()

    descomprimido = ""
    actual = ""
    for bit in infocomprimida:
```

```
            actual += bit
            if actual in libcodigos:
                descomprimido += libcodigos[actual]
                actual = ""

        with open(output, 'w') as f:
            f.write(descomprimido)

        print(f"Descompresi n exitosa. Archivo descomprimido: {output}")

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("se usa asi: python descompresor.py <archivo_comprimido> <archivo_
        sys.exit(1)

    comprimido = sys.argv[1]
    tabla = sys.argv[2]
    nombre = tabla
    output = os.path.splitext(nombre)[0] + ".txt"
    nombresalida = sys.argv[3]
    os.rename(output, nombresalida)

    descomprimir(comprimido, tabla, output)
```

2.3.2. FUNCIONES

1. Nombre

   tbfreq

2. Modulo:

   Compresor

3. Parametros

   Un string "texto"

4. Retorno

   Diccionario con los caracteres y las frecuencias

5. Resumen

   Una funcion que recibe un string y de este string se saca la frecuencia y los caracteres y se hace un diccionario en el cual se guardan los caracteres y sus frecuencias.

1. Nombre

   creaarbol

2. Modulo

   Compresor

3. Parametros

   una tabla de frecuencias

4. Retorno

   Retorna un arbol de huffman

5. Resumen

   La funcion crea un arbol de huffman apartir de una tabla de frecuencias que contiene los caracteres y sus frecuencias.

1. Nombre

   atravesar

2. Modulo

   Compresor

3. Parametros

   un nodo, su codigo y un diccionario

4. Retorno

   Cambia un diccionario

5. Resumen

   Funcion que se crea para recorrer el arbol y crear un diccionario de codigos en conjunto a la

funcion codigos

1. Nombre

   codigos

2. Modulo

   Compresor

3. Parametros

   un arbol

4. Retorno

   Retorna un diccionario con los caracteres y sus codigos respectivamente

5. Resumen

   La funcion recibe un arbol y en conjunto a la funcion de atravesar crean un dicionario que lleva los caracteres y sus codigos respectivamente

1. Nombre

   Comprimir

2. Modulo

   Compresor

3. Parametros

   4 archivos de texto que va cambiando

4. Retorno

   3 archivos, uno .huff, uno .table y uno .stats

5. Resumen

   La funcion comprimir utiliza las otras funciones para comprimir el archivo original que se llama ïnputz lo convierte en otros 3 archivs, un que contiene el arbol de huffman, otro que contiene las stats y otro que tiene la tabla con los codigos.

1. Nombre

   altura

2. Modulo

   Compresor

3. Parametros

   un arbol

4. Retorno

   Retorna la altura del arbol dado

5. Resumen

   La funcion recibe un arbol y despues calcula su altura

1. Nombre

   ancho

2. Modulo

   Compresor

3. Parametros

   un arbol

4. Retorno

   Retorna el ancho del arbol

5. Resumen

   La funcion recibe un arbol y despues calcula su ancho

1. Nombre

   npn

2. Modulo

Compresor

3. Parametros

   un diccionario

4. Retorno

   Retorna un diccionario con los niveles del arbol uno a uno y la cantidad de nodos que tiene cada uno de estos niveles

5. Resumen

   La funcion recibe un arbol y va de nivel en nivel recorriendo el arbol en conjunto a la funcion recorrer y devueove un diccionario que contiene los niveles y la cantidad de nodos en ellos

1. Nombre

   recorrer

2. Modulo

   Compresor

3. Parametros

   un nodo, su nivel y un diccionario

4. Retorno

   Retorna un cambio en el diccionario ingresado

5. Resumen

   La funcion recibe un nodo, su nivel y un diccionario, entonces busca el nivel en el diccionario y le agrega uno de valor por el nodo, de manera que en conjunto con npn calcula los nodos por nivel

1. Nombre

   leetabla

2. Modulo

   Descompresor

3. Parametros

   un archivo que contiene la tabla de frecuencias y los caracteres

4. Retorno

   Retorna un diccionario con los caracteres y sus codigos respectivamente

5. Resumen

   Una funcion que recibe un archivo con los caracteres y sus respectivos codigos y los reinterpreta en un diccionario

1. Nombre

   Descomprimir

2. Modulo

   Descompresor

3. Parametros

   3 archivos, el .huff, el .table y uno de output

4. Retorno

   Retorna un archivo output con los otros archivos descomprimidos

5. Resumen

   La funcion utiliza la funcion leetabla para leer la tabla y crear un diccionario de los codigos con sus caracteres y despues ir de bit en bit creando siguientes codigos que se prueban con el diccionario y si existen, se agrega el respectivo caracter.

3. Problemáticas y limitaciones

No hubo ni problematicas ni limitaciones.

4. Conclusiones

Como aprendizajes, hay que empezar por que para concretar el proyecto se tuvo que investigar acerca de las clases para utilizarlo y crear una clase para el nodo de manera

que se pudiera ver el arbol y los nodos desde otro punto de vista, de esta manera se introduce las clases y se logra un mejor resultado y mas eficiente gracias a la utilizacion de esta misma.

Despues de todo lo mencionado tambien hay que mencionar el aprendizaje sobre los diccionarios, en este proyecto se utilizo mucho los diccionarios, puesto que cada tabla y cada vez que se necesitaba recoger datos se utilizaba un diccionario de manera que fuera lo mas eficiente posible, tambien se utilizo la biblioteca heap para hacer un heap y utilizarlo eficientemente, la biblioteca sys que se utilizo para hacer la entrada del sistema y la biblioteca os que se utilizo para hacer archivos y editarlos de manera que retornara lo pedido.

5. Link de los archivos en Github

Github del proyecto

Referencias

[1] EcuRed, "David Huffman - EcuRed." Recuperado de: `https://www.ecured.cu/David_Huffman`, 2024. Fecha de consulta: 06/11/2024.

[2] LinkedIn, "¿Cómo evalúa las compensaciones entre los métodos de compresión de datos con pérdida y sin pérdida?." Recuperado de: `https://es.linkedin.com/advice/0/how-do-you-evaluate-trade-offs-between-3c?lang=es`, N/A. Fecha de consulta: 06/11/2024.

[3] OIA, "Trie – Estructura de Datos." Recuperado de: `https://oiaunlam.wordpress.com/2016/05/02/trie/`, 2016. Fecha de consulta: 06/11/2024.

[4] I. Schnell., "bitarray: efficient arrays of booleans." Recuperado de: `https://pypi.org/project/bitarray/`, N/A. Fecha de consulta: 06/11/2024.

[5] P. documentation, "heapq — Heap queue algorithm." Recuperado de: `https://docs.python.org/3/library/heapq.html`, N/A. Fecha de consulta: 06/10/2024.

[6] P. documentation, "os — Interfaces misceláneas del sistema operativo." Recuperado de: `https://docs.python.org/es/3.10/library/os.html`, N/A. Fecha de consulta: 06/10/2024.

[7] P. documentation, "sys — System-specific parameters and functions." Recuperado de: `https://docs.python.org/3/library/sys.html`, N/A. Fecha de consulta: 06/10/2024.

[8] P. Londoño, "Qué son las clases en Python, para qué sirven y cómo funcionan." Recuperado de: https://blog.hubspot.es/website/clases-python#:~:text=Una%20clase%20en%20Python%20es,en%20un%20programa%20de%20computadora, 2023. Fecha de consulta: 06/10/2024.