Informe Cuarto Proyecto

ESTUDIANTES:

Ricardo Murillo Jiménez
2018173697

Joel Vega
2018163840

20 de junio de 2018

Curso:
Taller de Programación

Carrera Ingeniería en Computación

Escuela de Ingieniería computación

Grupo 21

INSTITUTO TECNOLÓGICO DE COSTA RICA

Costa Rica

2018

Índice

Índice de figuras

1. Introducción

La computadora y todo instrumento tecnológico como bien se sabe tienen un limite ya sea de memoria, capacidad de procesamiento entre otras cosas que a pesar de los avances que se han logrado hacer en los ultimos años, no vamos a poder tener una memoria infinita o algo que nos procese la informacion al instante. Debido a esto se ha buscado la forma de poder reducir el tamaño de los archivos que se procesan en las computadoras para que a la hora de transmitir estos archivos a otro computador no se dure tanto y el consumo de energia es menor y a la computadora receptora le es mas facil descargarlo ytenerlo guardado.

Claro, a la hora de comprimir el archivo pierde toda su estructura por lo que la computadora ya no tiene la capacidad de mostrarnos el archivo, por lo que a la hora de comprimirlo debemos guardar algunos datos que nos ayudaran a descomprimir el archivo y recomponerlo tal y como era originalmente. Estos datos son basicamente la frecuencia en que aparecen los datos en el documento que se esta comprimiendo, con los cuales se crearon los arboles para la compresión, pero esto se exlicaraá mas detalladamente mas adelante. Exiten multiples formas y algoritmos para lograr comprimr un archivo pero para este proyecto se utilizará el algoritmo de Huffman, el cual es un algoritmo para comprimir archivos sin perdida de datos.

El creador de este algoritmo fue David A. Huffman fue un profesor en E.E.U.U quien fue un personaje en el area de la computación, pero por lo que mas se conoce es por su sistema de codificación de archivos el cual se explicará con mas detalle mas adelante, pero este algoritmmo de compresión de datos es muy usado ya que se puede aplicar en muchas cosas como redes de computación y television. Basicamente lo que se hace es ver la cantidad de veces que aparece un simobolo, letra, etc, en el archivo y darle una cadena de bits unica para cada dato con los cuales se crea un arbol en donde se van uniendo los datos que aparecen menor cantidad de veces, despues se "pegan"las cadenas de bits para así poder reducir el tamaño del archivo, despues de guarda el abecedario de con las cadena de bits que representa cada dato para poder descomprimirlo sin perdida de datos.

2. Desarrollo

2.1. Descripción del problema

Debido a que algunas veces los archivos que manejan la computadoras son demasiados pesados y ocupan mucho espacio en memoria se busca la forma de reducir su tamaño, y tambien se busca esta reduccion para que a la hora de enviar archivos no se tarde tanto tiempo en este proceso ya que ocupa muchos recursos y tambien para la descarga en la computadora receptora. Así que se buscará la forma de cómprimir los archivos mediante el uso del algoritmo de Huffman o codificacion de Huffman, tambien es importante señalar que mediante este sistema no se pierden datos a la hora de descomprimir el archivo, y esta descompresion se debe de hacer por que a la hora de comprimir el archivo se pierde la forma del archivo origina, por lo que en el primer proceso se ocupa guardar algunos datos que son necesarios para no perder datos en el proceso. Esto se hará mediante la materia vista en clase y la investigación en internet de diversos temas que nos ayudaran a ver como funciona mejor el algoritmo usado.

2.2. Investigación corta

2.2.1. Biografía de David A. Huffman

David Albert Huffman fue un profesor en la Universidad de California en Santa cruz, nacido el 9 de agosto de 1925 en Ohio y fallecido el 7 de octubre de 1999 en Santa Cruz, Estados Unidos a la edad de 74 años. David fue una gra persona en el campo de las ciencias de las computación y mas que todo en la parte de codificación de datos. Por lo que es mas conocido es por el codigo de Huffman(Codificaciń Huffman) el cual es un sistemas de compresión. Este código fue el ultimo proyecto de Huffman mientras estudiaba en el Instituto Tecnológico de Massachusetts. A loa 18 años ya había conseguido su titulo de Ingeniería eléctrica. Lego a ser incluso oficial de la marina cuando estaba cumpliendo con su servicio militar. Después logro conseguir los titulos de ingeniería electrónica. En el 53 emepezó a ser docente en el MIT, después se paso a la Universidad de California, donde logró fundar el departamento de ciencia informática, después de lograr desempeñar con presidente y colaborar en el desarrollo de los programas academicos del departamento, se jubiló en el 94, aún así siguió dando clases de teoría de la información y análisis de "señales".[**?**]

Realizó importantes avances en otrasáreas como teoría de la información y la codificación, comunicaciones y diseño para circutos lógicos asíncronos. Gracias a sus estudios en ciencias de la nformatica y la información ganó diversos premios, el ultimo fue la medalla del Instituto Electrónico Richard Hamming y de los ingenieros electrónicos en el 99. Recibió premios como alumno distinguido de la Universidad de Ohio ade´as de otros premios de investigación tecnol"ogica. Su vida termino cuando después de 10 mese de lucha contra el cáncer . un aspecto importante de ver, es que Huffman nunca patento ningún trabajo ya que estaba mas centrado en la educación.

2.2.2. Compresón de archivos con pérdida y sin pérdida

En la compresión sij perdidaa el archivo comprimido es el mismo antes y después de realizar todo el proceso de compresión y descompresi"on del archivo, osea no se perdio ningun tipo de dato, claro esto lleva a un mayor tiempo de procesado y no se logra reducir tanto el archivo como en a compresi"on con perdida.

por otra parte en la cpompresi"on con perdida , el archivo se logra comprimir un tanto mas que en la anterior y el tiempo de procesado más corto, pero debido al tipo de compresi"on se pierden datos en el camino por lo que la calidad del archivo se ve reducida y es imposible volver al archivo original aunque la copia que se logra es muy proxima.

2.2.3. Descripción de la estructura abstracta Trie

"La información almacenada en un trie es un conjunto de claves, donde una clave es una secuencia de símbolos pertenecientes a un alfabeto. Las claves son almacenadas en las hojas del árbol y los nodos internos son pasarelas para guiar la búsqueda. El árbol se estructura de forma que cada letra de la clave se sitúa en un nodo de forma que los hijos de un nodo representan las distintas posibilidades de símbolos diferentes que pueden continuar al símbolo representado por el nodo padre. Por tanto la búsqueda en un trie se hace de forma similar a como se hacen las búsquedas en un diccionario:

Se empieza en la raíz del árbol. Si el símbolo que estamos buscando es A entonces la búsqueda continúa en el subárbol asociado al símbolo A que cuelga de la raíz. Se sigue de forma análoga hasta llegar al nodo hoja. Entonces se compara la cadena asociada a el nodo hoja y si coincide con la cadena de búsqueda entonces la búsqueda ha terminado en éxito, si no entonces el elemento no se encuentra en el árbol.
Por eficiencia se suelen eliminar los nodos intermedios que sólo tienen un hijo, es decir, si un nodo intermedio tiene sólo un hijo con cierto carácter entonces el nodo hijo será el nodo hoja que contiene directamente la clave completa.

Es muy útil para conseguir búsquedas eficientes en repositorios de datos muy voluminosos. La forma en la que se almacena la información permite hacer búsquedas eficientes de cadenas que comparten prefijos."("Trie", 2018)

2.2.4. Descripción de la biblioteca bitarray

"Este módulo proporciona un tipo de objeto que representa eficientemente una matriz de booleanos. Bitarrays son tipos de secuencia y se comportan de manera muy similar a las listas habituales. Ocho bits están representados por un byte en un bloque contiguo de memoria. El usuario puede seleccionar entre dos representaciones: little-endian y big-endian. Se proporcionan métodos para acceder a la representación de la máquina. Esto puede ser útil cuando se requiere acceso de nivel de bits a archivos binarios, como archivos de imagen de

mapa de bits portátiles (.pbm). Además, cuando se trata de datos comprimidos que utilizan codificación de longitud de bits variable, puede encontrar útil este módulo."("bitarray", 2018)

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

Bitarray = Crea y devuelve una objeto de bits, los cuales ocupan menos memoria y son mas faciles de manejar
tofile = Escribe datos binarios a un archivo binario.
fromfile = Lee valores(bytes) desde un archivo y los añade al bitarray, pero como si fueran valores maquina

2.3. Descripción de la solución

```
#algoritmo de ordenamiento usado para ordenar frecuencias
def insertionSort(alist):
        for index in range(1,len(alist)):
            position = index
            while position>0 and alist[position-1][1]>alist[position][1]:
                (alist[position],alist[position-1]) =
                (alist[position-1],alist[position])
                position = position-1
        return alist
```

Figura 1: Ordena las Frecuencias

Nombre: insertionSort
Modulo: frecuencias.py
Parámetros: matriz
Retorno: matriz
Resumen: Recibe una matriz que contiene las frecuencias de los caracteres, recorre dicha matriz y ve si las frecuencias estan ordenadas de mayor a menor aparicion, si no esta bien ordenada reacomoda los elementos de estos hasta que queden de mayor a menor aparición.

```
#saca las frecuencias de cada uno de los caracteres
def frecuencias(string):
        resp = []
        i = 0
        j = 0
        while(i<len(string)):

                encontrado = False
                while(j<len(resp)):
                        if string[i]==resp[j][0]:
                                resp[j][1]+=1
                                encontrado = True
                        j += 1
                if not encontrado:
                        resp.append([string[i],1])
                i+=1
                j = 0
        return insertionSort(resp)
```

Figura 2: Obtener las frecuencias

Nombre: frecuencias
Modulo: frecuencias.py
Parámetros: string
Retorno: insertionSort(resp)
Resumen: Recibe el archivo con el que se va a trabajar, lo recorre y va contando las veces que aparece determinado caracter en el archivo, para as"i saber la frecuencia que tiene y podeer crear el arbol que se ocupa, lo que devuelve es una matriz con todas las frecuencias pero aplicandole la función de ordenamiento

Nombre: crea_arbol
Modulo: frecuencias.py
Parámetros: frecuencia(la matriz de las frecuencias ordenadas)
Retorno: Retorno el arbol que se creo con las rutas de los caracteres
Resumen: Basicamente va agarrando las frecuencias que se obtuvieron con anterioridad, primero agarra las dos menos frecuentes y las une, despues vuelve a buscar las dos menos frecuentes, en caso de que 1 de las 2 que siguen ya este en el arbol y la otra en la matriz, solamente se agrega la que esta en la matriz y se junta con el arbol y así sigue hasta que solo quede un unico nodo fuente que contiene la cantidad total de frecuencias, en caso de que hayan mas de 2 o 3 con la misma frecuencia no importa, ya que va por orden de aparicion en la matriz que se creo anteriormente.
Nota. Debido al tamaño de la imagen hemos decidido no ponerla, ya que quedaría mal

```
#saca los codigos de cada caracter por separado
def caminos(subarbol,actual,diccionario):
        if type(subarbol[0]) == str:
                diccionario[subarbol[0]] = actual
        else:
                caminos(subarbol[1],actual + "0",diccionario)
                caminos(subarbol[2],actual + "1",diccionario)
```

Figura 3: Obtener caminos

Nombre: caminos
Modulo: frecuencias.py
Parámetros: subarbol=todos los subarboles dentro dela arbolpricipal/actual=el camino actual de cada caracter,ejem(11001)/diccionario= El dicccionario de las palabras y codigos
Retorno:
Resumen: recorre todos los sub arboles y va sacando las rutas de desde la raíz hasta las hojas

```
#saca el codigo de todo el string
def codigoPalabra(palabra,diccionario):
        codigo = ""
        for i in palabra:
                codigo = codigo + diccionario[i]
        return codigo
```

Figura 4: Obtiene codigo palabra

Nombre: codigoPalabra
Modulo: frecuencias.py
Parámetros: palabra= el caracter del cual se quiere obter el codigo/ Diccionario= Contiene todos los codigos con las rutas de los caracteres
Retorno: codigo, el codigo de la ruta de cada caracter en especifico
Resumen: recibe la palabra o caracter del cual se quiere obtener el codigo y lo busca en el diccionario

```
#es utilizada para la funcion crea_arbol
#suma todas las frecuencias
def suma_frecuencias(frecuencia):
        i = 0
        suma = 0
        while(i<len(frecuencia)):
                suma+=frecuencia[i][1]
                i+=1
        return suma
```

Figura 5: Suma las frecuencias

Nombre: suma_frecuencias
Modulo: frecuncias.py
Parámetros: frecuancia=las frecuencias de cada caracter, las veces que aparecen
Retorno: la suma total de todas las frecuancias
Resumen: Recorre la matriz donde se encuentran todas las frecuencias y las suma

```
#saca la altura de un arbol
def altura(arb):
        if (arb==[]):
                return 0
        return 1 + max(altura(arb[1]),altura(arb[2]))
```

Figura 6: Altura arbol

Nombre: altura
Modulo: frecuencias.py
Parámetros: arbol
Retorno: la altura del arbol
Resumen: Recorre todo el arbol de rutas y da la mayor longuitud desde la raiz hasta la hoja mas alejada de el

```
#saca la cantidad de numero de nodos por nivel
def n_nodos_level(arb,n):
        if(arb == []):
                return 0
        elif(n == 0):
                return 1
        else:
                return n_nodos_level(arb[1], n-1)+n_nodos_level(arb[2], n-1)
```

Figura 7: Nodos por nivel

Nombre: n_nodos_level
Modulo: frecuencias.py
Parámetros: abol y el nivel Recibe el arbol con las rutas y el nivel del cual se quieren obtener la cantidad de nodos del mismo
Retorno: la cantidad de nodos en determinado nivel
Resumen: Recorrre el arbol de rutas hasta el nivel que se quiere y se devuelve la cantidad de nodos de ese nivel

```
#saca la anchura del arbol
def anchura_arbol(arb):
        return ancho_arbol_aux(arb,0,[])

def ancho_arbol_aux(arb,n,acum):
        if (arb==[]):
                return 0
        elif(n>altura(arb)):
                return max(acum+[1])
        return ancho_arbol_aux(arb,n+1,acum+[n_nodos_level(arb,n+1)])
```

Figura 8: Anchura arbol

Nombre: anchura_arbol
Modulo: frecuncias.py
Parámetros: arbol
Retorno: ancho_arbol_aux, la cual es una funcion que nos ayuda a resolver la función principal
Resumen: Se llama a la función auxiliar y está nos devuelve la anchura del arbol, o sea el nivel que contiene mas nodos

```
def nodos_nivel(arb):
        i = altura(arb)-1
        lista = []
        while(i>=0):
                lista.append(["Nivel: "+ str(i),"Nodos: "+str(n_nodos_level(arb
                i-=1
        return lista
```

Figura 9: Nodo nivel

Nombre: nodos_nivel
Modulo: frecuencias.py
Parámetros: arb
Retorno: Una lista con todos los nodos de cada nivel(nivel 1 , nodos 2)

Resumen: Recorre el arbol y va metiendo en una lista los nodos por nivel indicandonos la cantidad de cada uno

```
def descomprimir(diccionario,bits):
    lista = ''
    res = ''
    letras = list(diccionario.keys())
    valores = list(diccionario.values())
    i = 0
    j = 0
    while(i<len(bits)):
        lista+=bits[i]
        while(j<len(valores) and lista != ''):
            if lista==valores[j]:
                res+=letras[j]
                lista=''
            j+=1
        j = 0
        i +=1
```

Figura 10: Descompresor

Nombre: descomprimir
Modulo: descompresor.py
Parámetros: diccionario y bits/ recibe el diccionario de las rutas para así poder "armar" de nuevo el archivo
Retorno: el archivo descomprimido
Resumen:

3. Problemáticas y limitaciones

A la hora de descomprimir el archivo e imprimirlo en pantalla hay un pequeño error y es que el programa a la hora de convetir los bytes a caracteres agrega dos 0's, emtonces esto afecta mucho ya que no se está obteniendo el archivo original.

4. Conclusiones

Con la realización de este proyecto se incentiva la investigación de los estudiantes sobre temas relacionados con el proyecto como lo es la vida de Huffman y sobre como aprender a usar su algoritmo o código de compresión de datos sin de perdida de Huffman, y asi llegar a comprender un poco mejor como funcionan y ver como manejan la información los comprimidores o descomprimidores de archivos que se ven en las computadoras alrededor de todo el mundo.
Un aspecto importante a destacar es como se mencion"o arriba, es que este código es sin perdida de datos y esto es algo muy importante ya que así la calidad de los archivos no se va a ver afectada en ningún momento por el proceso de compresión o descompresión de los datos. Un dato interesante es que a Huffman nunca le interesó el dinero, el deía que su legado eran sus estudiantes y es interesante por que muchas personas deberían aprender eso y no preocuparse sólo por el dinero.

5. Referencias

Trie. (2018). Retrieved from https://es.wikipedia.org/wiki/Trie.

bitarray. (2018). Retrieved from https://pypi.org/project/bitarray/#description

Diccionarios en Python. (2018). Recuperado de https://devcode.la/tutoriales/diccionarios-en-python/

Huffman Coding. (2018). Retrieved from https://www.youtube.com/watch?v=fPthQE7Li8M

Byte, B. (2018). Decodificando el algoritmo de Huffman. Retrieved from http://bitybyte.github.io/Descomprimi
datos-Huffman/

Byte, B. (2018). Comprimiendo datos - el algoritmo de Huffman en Python. Retrieved from http://bitybyte.github.io/Huffman-coding/