Tarea - Algoritmo de Huffman

Alumno: Ángel Alonso Galarza Chávez
Profesor: Dr. Cuauhtemoc Mancillas López
Curso: Programación Avanzada

Introducción

La codificación de Huffman es un método de codificación optimo que consiste en obtener las frecuencias de los caracteres del mensaje a codificar y construir una serie de códigos para representar cada carácter, obteniendo un mensaje con un tamaño menor.

El funcionamiento del código es el siguiente:
Primero para la parte de codificación del texto:

- Obtener las frecuencias de los caracteres del mensaje e insertar los valores en una estructura.
- Ordenar de forma ascendente las frecuencias.
- Obtener los dos caracteres con la menor frecuencia y formar un nuevo elemento con la suma de las frecuencias de los dos caracteres e insertarlo en una estructura. Repetir hasta que solo quede un elemento en la estructura.
- Formar los códigos que representen cada carácter.
- Codificar el texto remplazando los caracteres con sus respectivos códigos.
- Generar el archivo comprimido.

Después la parte de decodificación del texto:

- Cargar el archivo comprimido.
- Con la estructura formada en la primera parte del codigo, leer cada digito y dependiendo si el digito es 1 o 0 recorrer la estructura a la izquierda (0) o a la derecha (1). Repetir este paso hasta llegar al ultimo elemento y obtener su caracter.
- Colocar el caracter en una cadena de caracteres, repetir este proceso hasta el ultimo digito del archivo comprimido.
- Formar el texto decodificado.
- Imprimir en pantalla el texto decodificado.

Código del algoritmo de Huffman

En esta seccion se mostrara el codigo de las funciones para realizar la codificacion de Huffman.

Para obtener las frecuencias de los caracteres del mensaje, se utilizan dos arrerglos, uno de tipo char y otro de tipo entero. Con el archivo abierto, lee caracter por caracter hasta el final del mensaje y cuentas las apariciones de cada caracter. De esta manera se obtiene dos arreglos con los caracteres en orden ascii con sus repespectivas frecuencias.e

```
void getCharactersAndFrequencies(const char *filename, char *characters,
                                 int *frequencies, int *size) {
  int freq[MAX_CHAR] = {0}; // Arreglo para contar frecuencias
  FILE *file = fopen(filename, "r");

  if (file == NULL) {
    perror("Error al abrir el archivo");
    return;
  }

```

```
  // Leer el archivo y contar las frecuencias
  char ch;
  while ((ch = fgetc(file)) != EOF) {
    freq[(unsigned char)ch]++;
  }

  fclose(file);

  // Almacenar los caracteres y sus frecuencias
  *size = 0;
  for (int i = 0; i < MAX_CHAR; i++) {
    if (freq[i] > 0) {
      characters[*size] = (char)i;
      frequencies[*size] = freq[i];
      (*size)++;
    }
  }
}
```

Listing 1: Funcion para la obtencion de las frecuencia de los caracteres

Para la construcción del árbol, se obtienen los dos primeros elementos de la lista de cola de prioridad, con los dos elementos se crea un nuevo elemento con la suma de las dos frecuencias de los elementos extraídos para asignárselo al nuevo elemento y ademas se asignan como hijos izquierdo y derecho a los elementos extraídos. Por ultimo se inserta el nuevo elemento creado a la cola de prioridad, este proceso se repetirá hasta que en la lista solo exista un elemento.

```
// Funcion para construir el arbol de Huffman
HuffmanNode *BuildHuffmanTree(char characters[], int frequencies[], int size) {
  PriorityQueue *pq = createPriorityQueue(size);

  // Insertar todos los nodos en la cola de prioridad
  for (int i = 0; i < size; i++) {
    insertPriorityQueue(pq, createNode(characters[i], frequencies[i]));
  }

  // Construir el arbol de Huffman
  while (pq->size > 1) {
    HuffmanNode *left = extractMin(pq);
    HuffmanNode *right = extractMin(pq);

    // Crear un nuevo nodo que combine los dos nodos extraidos
    HuffmanNode *newNode = createNode('\0', left->frequency + right->frequency);
    newNode->left = left;
    newNode->right = right;

    // Insertar el nuevo nodo en la cola de prioridad
    insertPriorityQueue(pq, newNode);
  }

  // El nodo restante es la raiz del arbol de Huffman
  return extractMin(pq);
}
```

Listing 2: Fucion de construccion del arbol

En la función se pasan los arreglos que contienen los caracteres del mensaje y otro arreglo con las frecuencias de los caracteres. Se crea la estructura cola de prioridad e inserta los elementos de los caracteres y sus frecuencias para almacenarlos en la lista. Una vez insertado todos los elementos en la lista y ordenarlos de forma ascendente, se extraen los dos elementos con las frecuencia menor y se crea un nuevo nodo con la sumatoria de las frecuencias de los elementos extraídos y se inserta el nuevo nodo creado a la lista.

```
// Funcion para generar los codigos de Huffman
void generateHuffmanCodes(HuffmanNode *root, char *code, int top,
                          char codes[MAX_CHAR][MAX_CODE_LENGTH]) {
  if (root->left) {
    code[top] = '0';
    generateHuffmanCodes(root->left, code, top + 1, codes);
  }

  if (root->right) {
    code[top] = '1';
```

```
    generateHuffmanCodes(root->right, code, top + 1, codes);
  }

  if (!root->left && !root->right) {
    code[top] = '\0'; // Terminar el codigo
    strcpy(codes[(unsigned char)root->character], code);
  }
}
```

Listing 3: Fucion de generacion de codigosl

Para la codificación del texto, la siguiente función, toma los parámetros de texto, los códigos generados por el árbol de Huffman y un arreglo que guarde las codificaciones del texto. Para la codificación se realiza en un ciclo donde recorre carácter por carácter y guarda el código del carácter en el arreglo encodedText.

```

// Funcion para codificar el texto
void encodeText(const char *text, char codes[MAX_CHAR][MAX_CODE_LENGTH],
                char *encodedText) {
  encodedText[0] = '\0';
  for (int i = 0; text[i] != '\0'; i++) {
    strcat(encodedText, codes[(unsigned char)text[i]]);
  }
}
```

Listing 4: Funcion para codificar el texto

Con las funciones decodeText y decodeFile podemos descifrar el archivo generado con el mensaje codificado, el proceso es similar a la codificación del texto, pero un requisito es tener el árbol de Huffman, esto para descifrar los códigos y colocar el carácter correspondiente. Esta funcion se utilizara para decodificar el archivo, en el cual la funcion decodeFile abre el archivo comprimido y obtiene el contenido del mensaje para realizar la codificacion.

```
void decodeText(Nodo *root, const char *encodedText, char **decodedText) {
  HuffmanNode *current = root;
  int index = 0;
  int decodedIndex = 0;
  int bufferSize = 2048;
  *decodedText = (char *)malloc(bufferSize * sizeof(char));

  if (*decodedText == NULL) {
    perror("Error al asignar memoria");
    return;
  }

  while (encodedText[index] != '\0') {
    if (encodedText[index] == '0') {
      current = current->left;
    } else {
      current = current->right;
    }

    if (!current->left && !current->right) {
      if (decodedIndex >= bufferSize - 1) {
        bufferSize *= 2;
        *decodedText = (char *)realloc(*decodedText, bufferSize * sizeof(char));
        if (*decodedText == NULL) {
          perror("Error al reasignar memoria");
          return;
        }
      }
      (*decodedText)[decodedIndex++] = current->character;
      current = root;
    }

    index++;
  }

  (*decodedText)[decodedIndex] = '\0';
}

// Funcion para decodificar un archivo
void decodeFile(const char *filename, Nodo *root) {
  int bitLength;
```

```
  char *encodedText =
      readCompressedFile(filename, &bitLength); // Leer el archivo comprimido
  if (encodedText == NULL) {
    return;
  }

  char *decodedText;
  decodeText(root, encodedText, &decodedText);

  printf("Texto decodificado: %s\n", decodedText);

  free(encodedText);
  free(decodedText);
}
```

Listing 5: Funcion para decodificar texto y decodificar archivo

A continuación se muestran las funciones para escritura y lectura del archivo a comprimir.
En la función writeCompressedFile se recibe como parámetro el archivo que se creara para guardar el texto codificado, para hacerlo, obtiene partes del texto codificado y lo inserta en el archivo, repite este proceso hasta que todo el texto del mensaje original haya sido codificado e insertado en el archivo generado.
Para la función readCompressedFile se recibe el archivo codificado como parámetro y se realiza el procedimiento de obtener cada dígito 1 o 0 y lo guarda en un arreglo para después con el código poder descifrar a que carácter corresponde e insertarlo en el archivo.

```
void writeCompressedFile(const char *filename, const char *encodedText) {
  FILE *file = fopen(filename, "ab");
  if (file == NULL) {
    perror("Error al abrir el archivo");
    return;
  }

  // Convertir el texto codificado a bytes
  unsigned char byte = 0;
  int bitCount = 0;

  for (int i = 0; encodedText[i] != '\0'; i++) {
    byte = (byte << 1) | (encodedText[i] - '0');
    bitCount++;

    // Si se han acumulado 8 bits, escribir el byte en el archivo
    if (bitCount == 8) {
      fwrite(&byte, sizeof(unsigned char), 1, file);
      byte = 0;
      bitCount = 0;
    }
  }

  if (bitCount > 0) {
    byte <<= (8 - bitCount);
    fwrite(&byte, sizeof(unsigned char), 1, file);
  }

  fclose(file);
}

char *readCompressedFile(const char *filename, int *bitLength) {
  FILE *file = fopen(filename, "rb");
  if (file == NULL) {
    perror("Error al abrir el archivo");
    return NULL;
  }


  fseek(file, 0, SEEK_END);
  long fileSize = ftell(file);
  fseek(file, 0, SEEK_SET);

  // Crear un buffer para almacenar los bits
  char *bitString = (char *)malloc(fileSize * 8 + 1);
  if (bitString == NULL) {
    perror("Error al asignar memoria");
```

```
    fclose(file);
    return NULL;
  }

  int index = 0;
  unsigned char byte;
  while (fread(&byte, sizeof(unsigned char), 1, file) == 1) {
    for (int i = 7; i >= 0; i--) {
      if (byte & (1 << i)) {
        bitString[index++] = '1';
      } else {
        bitString[index++] = '0';
      }
    }
  }

  bitString[index] = '\0';
  fclose(file);
  *bitLength = index;
  return bitString;
}
```

Listing 6: Fucion de escritura y lectura del archivo codificado

Por ultimo, en la función main se inicializan los arreglos que se utilizaran para obtener la frecuencias, los caracteres del mensaje a codificar y la construcción del árbol de Huffman, ademas de las funciones para codificar y decodificar el archivo.

```
int main() {
  char characters[MAX_CHAR];
  int frequencies[MAX_CHAR];
  int size;

  getCharactersAndFrequencies("texto.txt", characters, frequencies, &size);

  // Imprimir los caracteres y sus frecuencias
  for (int i = 0; i < size; i++) {
    printf("Caracter: '%c', Frecuencia: %d\n", characters[i], frequencies[i]);
  }

  Nodo *root = BuildHuffmanTree(characters, frequencies, size);

  // Generar los codigos de Huffman
  char codes[MAX_CHAR][MAX_CODE_LENGTH];
  char code[MAX_CODE_LENGTH];
  char encodedText[MAX_CHAR * MAX_CODE_LENGTH];
  char text[MAX_TEXT_LENGTH];

  generateHuffmanCodes(root, code, 0, codes);

  readFileAndEncode("texto.txt", root, codes);

  decodeFile("archivo_comprimido.bin", root);

  return 0;
}
```

Listing 7: Funcion Main

Ejecución

En la siguiente figura se mostrara la ejecución del código para codificar y decodificar un texto.

Figure 1: Ejecución del programa