Atributos del curso

Instituto Tecnológico de Costa Rica

Algoritmos Y Estructuras De Datos II

Sede Central Cartago

Profesor: Jose Isaac Ramirez Herrera

Integrantes:

Sebastián Bolaños Zamora

Ciclo Lectivo: I Semestre del año 2025

Código de los algoritmos: AlgoritmosDeCompresion

Documentación del algoritmo de Huffman

1. countFrequencies(const string& text)

- **Propósito:** Recorrer la cadena de entrada y contar la frecuencia con que aparece cada carácter.
- **Retorno:** Un mapa que asocia cada carácter (`char`) con su frecuencia (`int`).
- **Notas:** Utiliza `freq[c]++` para contar caracteres, creando nuevas entradas con valor inicial cero antes del incremento.

2. printFrequencies(const unordered_map<char, int>& freq)

- **Propósito:** Mostrar en consola la tabla de frecuencias de cada carácter.

3. buildPriorityQueue(const unordered_map<char, int>& freq)

- **Propósito:** Crear e inicializar una cola de prioridad con un nodo hoja para cada carácter según su frecuencia.
- **Retorno:** Una cola de prioridad (`priority_queue`) configurada para extraer primero los nodos con menor frecuencia.
- **Detalles:** Por cada par (símbolo, frecuencia), crea un nodo con `new Node(string(1, símbolo), frecuencia)` y lo inserta en la cola.

4. buildHuffmanTree(priority_queue<Node, vector<Node>, Compare>& pq)

- **Propósito:** Construir el árbol de Huffman combinando sistemáticamente los dos nodos con menor frecuencia.
- **Retorno:** Un puntero a la raíz del árbol construido (`Node*`).

5. buildCodes(Node node, const string& prefix, unordered_map<char, string>& outCodes)

- **Propósito:** Recorrer recursivamente el árbol de Huffman para generar códigos binarios asignados a cada carácter.
- **Parámetros:**
  - `node`: Nodo actual en la recursión.
  - `prefix`: Cadena acumulada de bits desde la raíz hasta el nodo actual.
  - `outCodes`: Mapa que almacena pares `carácter→código binario`.
- **Comportamiento:**
  - Ignora si `node` es nulo.
  - Si es nodo hoja, asigna `prefix` como código al carácter (`node->symbols[0]`).
  - Si tiene hijos, llama recursivamente al hijo izquierdo con `prefix + "0"` y al derecho con `prefix + "1"`.

6. generateCodes(Node root)

- **Propósito:** Interfaz para invocar `buildCodes` y mostrar los códigos resultantes al usuario.
- **Retorno:** Un mapa `char→string` con los códigos binarios generados.

7. encodeText(const string& text, const unordered_map<char, string>& codes)

- **Propósito:** Convertir cada carácter del texto original en su código correspondiente y concatenar el resultado.

8. printStatistics(const string& text, const unordered_map<char, string>& codes)

- **Propósito:** Comparar y mostrar el tamaño original del texto frente al comprimido.

9. runHuffman(const string& text)

- **Propósito:** Gestionar y ejecutar secuencialmente todas las etapas del algoritmo de Huffman.
- **Comportamiento:**
    - Calcula y muestra frecuencias (`countFrequencies`, `printFrequencies`).
    - Construye cola de prioridad (`buildPriorityQueue`).
    - Genera el árbol de Huffman (`buildHuffmanTree`).
    - Obtiene los códigos (`generateCodes`).
    - Codifica el texto (`encodeText`).
    - Muestra estadísticas (`printStatistics`).

Documentación del algoritmo de compresión LZ77

1. getSearchBuffer(const string& text, int pos, int windowSize)

- **Propósito:** Construir la ventana de búsqueda (search buffer), tomando hasta `windowSize` caracteres anteriores a la posición actual (`pos`).
- **Parámetros:**
  - `text`: Referencia constante al texto original.
  - `pos`: Índice actual en el texto donde se inicia la búsqueda.
  - `windowSize`: Tamaño máximo permitido para la ventana de búsqueda.
- **Retorno:** Subcadena desde la posición `max(0, pos - windowSize)` hasta `pos - 1`.

2. getLookAheadBuffer(const string& text, int pos)

- **Propósito:** Construir la ventana de anticipación (look-ahead buffer), iniciando desde la posición actual (`pos`) hasta el final del texto.
- **Parámetros:**
  - `text`: Referencia constante al texto original.
  - `pos`: Índice actual donde comienza la ventana de anticipación.
- **Retorno:**

  Subcadena del texto desde la posición `pos` hasta el final.

3. struct LZ77Token { int offset; int length; char nextChar; }

- **Propósito:** Representar un token generado por la compresión LZ77.
- **Campos:**
  - `offset`: Distancia hacia atrás desde la posición actual hasta el comienzo de la coincidencia.

- `length`: Longitud de la coincidencia encontrada.
- `nextChar`: Carácter siguiente a la coincidencia (`'\0'` si no hay más caracteres).

4. findLongestMatch(const string& text, int pos, const string& searchBuffer)

- **Propósito:** Encontrar en la ventana de búsqueda (`searchBuffer`) la coincidencia más larga con la subcadena que comienza en la posición actual (`pos`).
- **Parámetros:**
  - `text`: Referencia constante al texto completo.
  - `pos`: Índice de inicio en el texto para buscar coincidencias.
  - `searchBuffer`: Subcadena previa donde se realiza la búsqueda.
- **Retorno:** Un token (`LZ77Token`) con los campos `offset`, `length` y `nextChar` definidos.
- **Detalles:**
  - Itera desde la máxima longitud posible hacia 1.
  - Usa `rfind` para hallar la última aparición del patrón en el `searchBuffer`.
  - Calcula `offset = searchBuffer.size()` - posición encontrada del patrón.
  - `nextChar` es `text[pos + length]`, o `'\0'` si se alcanza el final del texto.

5. printStep(int step, int searchStart, int pos, int n, const string& searchBuffer, const string& lookAheadBuffer, const LZ77Token& match)

Propósito: Imprimir información detallada de cada paso del proceso de compresión LZ77.

6. runLZ77(const string& text, int windowSize)

Propósito: Gestionar y ejecutar el proceso completo de compresión LZ77 sobre el texto dado.
Comportamiento:

1. Muestra información inicial (texto y tamaño de ventana).
2. Procesa el texto iterativamente hasta finalizar:
    - Construye ventanas (`searchBuffer`, `lookAheadBuffer`).
    - Llama a `findLongestMatch` para obtener el token.
    - Imprime detalles del paso usando `printStep`.
    - Avanza la posición actual por `length + 1` caracteres.

7. printCompressedTokens(const vector<LZ77Token>& tokens)

Propósito: Imprimir la lista de tokens comprimidos al final